Fülep Dávid, Pusztai Pál, Szörényi Miklós
SZE-MTK, Matematika és Számítástudomány Tanszék
Szerkesztette: Kallós Gábor

Informatikai eszközök alkalmazása mérnöki számításokhoz

2013

Műszaki és természettudományos alapismeretek
tananyagainak fejlesztése a mérnökképzésben
Pályázati azonosító: TÁMOP-4.1.2.A/1-11/1-2011-0054

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

IMPRESSZUM

Szerkesztette: Kallós Gábor
Széchenyi István Egyetem, Műszaki Tudományi Kar, Matematika és Számítástudomány Tanszék
Lektor: Dr. Füvesi István, Szegedi Tudományegyetem, Informatikai tanszékcsoport

ISBN 978-963-7175-87-9

Kiadó: Széchenyi István Egyetem, Műszaki Tudományi Kar

Támogatás:
Készült a TÁMOP-4.1.2.A/1-11/1-2011-0054 számú, "Műszaki és természettudományos alapismeretek tananyagainak fejlesztése a mérnökképzésben" című projekt keretében.

Kulcsszavak: *Visual Basic, Excel programozás, Matlab, Solver, lineáris programozás, lineáris egyenletrendszer, statisztikai alkalmazások, relációs adatmodell, SQL nyelv, lekérdezések*

Tartalmi összefoglaló: Az Informatika 2. tananyag négy modulra tagolódik, ezekben rendre az Excel programozását, a Matlab rendszer használatát, az Excellel történő mérnöki/számítási feladatok megoldását és az adatbázis-kezelést tekintjük át. A 2. és a 3. modul szorosan kapcsolódik egymáshoz, több esetben a megoldott problémák is ugyanazok. Az első modulhoz speciálisan programozási bevezető is tartozik, az önálló munkát sok (részben vagy teljesen) kidolgozott mintafeladat is támogatja.

Technikai megjegyzések a jegyzet használatához.

Ez a tananyag egy *elektronikus jegyzet*.

2013-ban, a megjelenés évében annyira elterjedtek az elektronikus tartalomfogyasztásra alkalmas eszközök, hogy bátran feltételezhetjük: az egyetemisták túlnyomó többsége rendelkezik saját számítógéppel, tablet-géppel vagy elektronikus könyvolvasóval. A tananyag elektronikus formája sok előnnyel rendelkezik a nyomtatotthoz képest:

- Aktív tartalmak: az elektronikus változatban belső kereszthivatkozások, külső linkek, mozgóképek, stb. helyezhetők el. A tartalomjegyzék fejezetszámai, az egyenlet- és ábrasorszámok automatikusan belső linket jelentenek, így biztosítják a kényelmes és gyors belső hivatkozást, de a Szerző tetszőleges helyre tud akár a dokumentum belsejébe, akár egy külső webhelyre mutató linket elhelyezni, ami a szokásos klikkentéssel aktivizálható.
- Rugalmasság: a nyomtatott könyv statikus, míg az elektronikus jegyzet esetében könnyű hibajavításokat, frissítéseket alkalmazni.
- Erőforrás-takarékosság, környezetvédelem: az elektronikus formában való terjesztés sokkal kisebb terhelést jelent a környezetre, mint a nyomtatott. Különösen igaz ez, ha a tananyagban sok a színes ábra.

A használt fájlformátum: *PDF*.

A Portable Document Format az Adobe által kifejlesztett formátum, mely igen széles körben elterjedt. Sok helyről szerezhetünk be programot, mely a PDF fájok olvasására alkalmas. Ezek egy része azonban nem tartalmazza a teljes szabvány minden elemét, ezért speciális tartalmak nem, vagy nem pontosan jelenhetnek meg, ha nem az Adobe olvasóját, az AdobeReader-t használjuk. (Letölthető innen.)

A legtöbb megjelenítőprogram jól fogja kezelni az alapszöveget, ábrákat és linkeket, de gondok lehetnek a speciálisabb funkciókkal, pl. a beágyazott dokumentumok kezelésével, az aktív tesztek, kérdőívek használatával.

A jegyzet *képernyőn való megjelenítésre* lett optimalizálva.

A jelenlegi általánosan elérhető könyvolvasó hardverek mérete és felbontása kisebb, mint a nyomtatott könyveké és a számítógépek monitorai általában fektetett helyzetűek. Ehhez igazítottuk a formátumot arra optimalizálva, hogy fektetett kijelzőn teljes képernyős üzemmódban lehessen olvasni. Ehhez állítottuk be a karaktertípust és -méretet valamint azt is, hogy csak kis margót hagyunk, minél több pixelt biztosítva ezzel a tartalomnak. Azért, hogy teljes képernyős üzemmódban is lehessen navigálni, a margón kis navigáló-ikonokat helyeztünk el, melyek a megszokott módon kezelhetők:

- Lapozás előre és hátra: a függőleges oldalak közepén elhelyezett, nyújtott nyilakkal.
- Címoldalra ugrás: kis házikó szimbólum a bal felső sarokban.
- Vissza és előreugrás a dokumentumban: két kicsi szimbólum a bal felső részen. Ezek nem azonosak a lapozással, hanem a web-böngészők vissza- és előrelépéséhez hasonlóan a hiperlinkeken való navigálást szolgálják.

A jegyzet *segítséget nyújt a tanulás ütemezésében*.

A megtanulandó tanagyag a szokásos fejezet-alfejezet felosztáson túl leckékre való bontást is tartalmaz. A leckék különböző számú alfejezetből állhatnak, de közös bennük, hogy a Szerző megítélése szerint egy lecke „együltő helyben” megtanulható, azaz várhatóan 1–1,5 óra alatt feldolgozható.

A leckék elején rövid leírás található a tárgyalt témakörökről, a szükséges előismeretekről, a végén pedig önellenőrző kérdések, melyek sok esetben a PDF fájlban (AdobeReader-rel) aktív tartalomként jelennek meg feleletkiválasztós teszt, számszerű vagy képletszerű kérdés formájában. Érdemes tehát leckénként haladni a tanulásban, mert ez segít az ütemezés tervezésében illetve a leckevégi ellenőrzések segítenek annak eldöntésében, tovább szabad-e haladni vagy inkább ezt vagy az előző leckéket kell újra elővenni.

Ha a tananyag indokolja, nagyobb egységeket „modulokba” szervezünk és a modulok végén a leckevégi önellenőrzéshez képest komolyabb feladatblokkot találhatunk.

Tartalom

I. MODUL | Excel programozás modul

2. Matematikai számítások Matlabbal

7. lecke

- 2.1. Alapvető információk a Matlabról
  - 2.1.1. A Matlab indítása, ablakok, kilépés
  - 2.1.2. Help rendszer
  - 2.1.3. A Matlab elemei
  - 2.1.4. Műveleti jelek, kifejezések, függvények
  - 2.1.5. Függvénycsaládok
  - 2.1.6. Elemi matematikai függvények
  - 2.1.7. Adatok mentése, visszatöltése

8. lecke

- 2.2. Mátrixok kezelése
  - 2.2.1. Mátrix és elemhivatkozások
  - 2.2.2. Speciális mátrixok
- 2.3. A Matlab programozása
  - 2.3.1. Szelekciós szerkezetek
  - 2.3.2. Iterációk
  - 2.3.3. Felhasználói függvények definiálása
  - 2.3.4. Adatbekérés, adatkiírás

III. MODUL | Matematikai számítások - Excel

18. lecke

Előszó

Ebben a jegyzetben elsősorban azokat az ismereteket tárgyaljuk, amelyek a Széchenyi István Egyetemen a mérnökhallgatók számítástechnikai/informatikai oktatásában az alkalmazási részeket tartalmazzák. Ezek a részek a mérnökképzés speciális szaktárgyaiban szemelvényszerűen megemlítődnek, de egy egységes ívű, az ismereteket folyamatosan bővítő tárgyalásmódra ott nincs lehetőség. A *Mérnöki számítások* című tantárgyat a Műszaki Tudományi Kar hallgatói a két féléves informatika oktatás második félévében tanulják heti 2 óra előadásban és 2 óra gyakorlatban. A tárgy tananyagának egyharmadát (4 hét) Excel programozási alapismeretek, a másik kétharmadát (8 hét) MatLab és Excel alkalmazási ismeretek teszik ki. Technikai okok miatt jegyzetünk végére került a közgazdász hallgatók informatikai alapképzésének részeként az Adatbázis-kezelés modul.

Az önálló tanulást segítendő, modulokra, azon belül leckékre bontottuk a jegyzetben található ismeretanyagot. Az egyes leckék, illetve modulok végén ellenőrző kérdések és feladatok találhatók. A feladatok között a programozási feladatok megoldása jelenti a legnagyobb kihívást, ezért egyrészt igyekeztünk sokféle feladatot kitűzni (reméljük így mindenki talál majd kedvére valót, tudásához illeszkedőt, amelyen szívesen töri a fejét, próbálja ki tudását), másrészt a megoldást egy-egy tippel (egy lehetséges megoldási elgondolással) segítjük.

A kialakításnál azt az elvet követtük, hogy minden hallgató – függetlenül attól, hogy konkrétan milyen szakon tanul, és mennyire mélyen kell/szeretne megismerkedni(e) az adott részterülettel – haszonnal tudja forgatni a jegyzetet, és megtalálja benne azokat az ismereteket is, amelyek – a rendelkezésre álló korlátozott időkeret miatt – az előadásokon és gyakorlatokon csak rövidebben kerülhetnek terítékre.

Az egyes részek és szerzőik:

1. Excel programozás (Pusztai Pál)
2. Matematikai számítások – MatLab (Szörényi Miklós)
3. Matematikai számítások – Excel (Szörényi Miklós)
4. Adatbázis-kezelés (Fülep Dávid)

A Matematikai számítások moduljai között a feldolgozott témakörök részben átfedik egymást. Ez a tárgyalási mód a matematikai problémák alaposabb megismerését, elemzését és megoldását teszi lehetővé. Ennek a két résznek sarkalatos pontja a szükséges matematikai előismeretek birtoklása!

Az anyag elsajátítása akkor tekinthető sikeresnek, ha a hallgató képessé válik a jegyzetünkben kitűzött feladatok megoldására is. Ez természetesen függ a korábbi egyéni felkészültségtől, a tanulási sebességtől, de több-kevesebb idő ráfordításával mindenki eredményes lehet.

Reméljük ugyanakkor, hogy a jegyzetet a zárthelyikre és a vizsgára való felkészülésen túl is eredményesen használják majd a hallgatóink.

A szerzők köszönetet mondanak dr. Füvesi Istvánnak türelméért és lelkiismeretes lektori munkájáért, amellyel nagy mértékben hozzájárult a kézirat javításához, és Fehérvári Arnoldnak az ellenőrzésben végzett áldozatos tevékenységéért.

Győr, 2013. január

A Szerkesztő és a Szerzők

I. MODUL

Excel programozás modul

1. LECKE

A Visual Basic programozási nyelv
Egyszerű adattípusok
Adatok kezelése

1. Excel programozás

A számítógépes szoftverfejlesztés az ember-gép kommunikáció legmélyebb, legnehezebben elsajátítható, de leginkább intellektuális szintjét képviseli. Az ehhez szükséges programozási ismeretek elsajátítása sok tanulást, és még több önálló gyakorlást igényel. Ez az időmennyiség általában több féléves képzést jelent, amelynek tükrében bizony igen szerény időkeretnek tűnik az Excel programozásra szánt 4 hét. Ennyi idő alatt természetesen nem lehet szoftverfejlesztő szakembereket képezni, de megpróbálunk olyan alapozást adni, amelyre a későbbi tanulmányok, illetve a mérnöki munka során már építkezni lehet.

Miért pont Excel programozás?! Joggal kérdezheti ezt egy mérnökhallgató, akinek ezt a kötelező tárgyat *(Mérnöki számítások)* teljesítenie kell. Választásunkat az alábbi főbb szempontok indokolták:

- Az MS Excel nemcsak a mérnöki munkában, de az élet szinte minden területén használatos. Elterjedt, könnyen hozzáférhető szoftver.
- Olyan programfejlesztő környezetet tartalmaz, amely alkalmas a programozási alapismeretek oktatására.
- Könnyen tanulható, magas szintű programozási nyelvet (Visual Basic) használ.
- Vizuális tervezést, eseményvezérelt programozást biztosít.
- Az automatikusan ismétlődő feladatok programozásával segíthető, megkönnyíthető a táblázatkezelővel végzett munka.

Ha egy feladat megoldására számítógépes programot készítünk, akkor azt általában nem úgy tesszük, hogy elkezdjük begépelni a program utasításait. Minél nagyobb, összetettebb a megoldandó feladat, annál inkább szükséges a következő megoldási lépések végrehajtása:

1. A feladat megfogalmazása, pontosítás, általánosítás.
2. Matematikai (vagy egyéb) modell kiválasztása, megadása (ha szükséges, illetve lehetséges).
3. Az adatszerkezet definiálása, az input-output specifikálása.

4. A megoldást megvalósító algoritmus megtervezése, elkészítése.

5. Programírás, kódolás (az adatszerkezet és az algoritmus alapján).

6. Tesztelés, hibakeresés.

7. Dokumentálás (felhasználóknak, fejlesztőknek).

Természetesen az adott feladat (vagy munka) jellegéből adódóan bizonyos lépések el is maradhatnak (pl. 2., 7.), illetve javítás, módosítás esetén szükség lehet egy korábbi szintre való visszalépésre is. A program tényleges megírása (5.) csak a már elkészített megoldó algoritmus (4.) ismeretében lehetséges, ami viszont nem készülhet el anélkül, hogy tisztáznánk, milyen bemenő és milyen eredmény adataink lesznek (3.).

Az Excel programozásról szóló modulunkat a programíráshoz (5.) szükséges (rövidített) nyelvi ismertetővel kezdjük (lásd 1.1. fejezet), amit a programok kipróbálásához (6.) szükséges ismeretek követnek (lásd 1.2. fejezet). A harmadik részben (lásd 1.3. fejezet) az Excel objektumairól, azok programból történő használatáról lesz szó. A terjedelmi korlátok miatt a feldolgozott témakörnek csak egy kisebb (de reményeink szerint a lényeges dolgokat tartalmazó) szeletét mutatjuk be.

A szakirodalom szerencsére bőségesen ellát minket programozásról szóló szakkönyvekkel. Az Excel programozás után érdeklődő olvasóknak az [1], a Visual Basic nyelvvel kapcsolatosan a [2], az algoritmusok tervezését illetően kezdőszinten a [3], haladóbb szinten a [4] szakirodalmat ajánljuk.

1.1. A Visual Basic programozási nyelv

A BASIC (Beginner's All-purpose Symbolic Instruction Code) programnyelvet oktatási célokra hozták létre 1964-ben (Dartmouth College USA, Kemény János és Thomas Kurtz). Az általános célú felhasználhatóság és a könnyű tanulhatóság elsődleges szempont volt, ez a mozaikszó szavaiból is kiderül.

A BASIC programozási nyelv magas szintű, az emberi gondolkodáshoz, jelölésrendszerhez közel álló programozási nyelv. A BASIC nyelven megírt programokat (az ún. forrásprogramokat) a számítógép nem tudja egyből végrehajtani. Ezek végrehajtásához fordítás (compile) vagy értelmezés (interpretation) szükséges.

A fordítóprogramok a teljes forrásprogramot lefordítják a számítógép által már végrehajtható utasításokká, míg az értelmezők utasításonként értelmezik és hajtják végre a forrásprogramot.

A nyelvnek többféle változata létezett, kezdve az iskola-számítógépek beépített BASIC értelmezőjétől, a 80-as években elterjedt személyi számítógépek Qbasic-jén keresztül, az 1991-ben megjelent Microsoft Visual Basic (röviden VB) nyelvig. A Visual Basic for Applications (röviden VBA) az MS Office szoftverek makrónyelve, a Visual Basic Script a Windows operációs rendszer scriptnyelve, a 2002-ben megjelent Visual Basic .NET pedig a .NET keretrendszer programozási nyelve.

A Visual Basic nyelv után ejtsünk pár szót a nyelvet használó szoftverfejlesztő rendszerekről is.

Az MS Visual Studio egy általános célú szoftverfejlesztő keretrendszer, amelyben többféle (pl. Visual Basic, C#) programnyelven is fejleszthetünk.

Az MS Visual Basic for Applications (pl. az általunk használt Excel VBA) főbb tulajdonságai:

- Csak Visual Basic nyelven programozhatunk, (a Visual Studio-hoz képest) korlátozott fejlesztési eszköztár mellett.
- Csak az adott szoftverrel (pl. Excel) együtt használható.
- Önállóan futtatható (*.exe) fájlok nem készíthetők.
- Az adott szoftverhez igazodó, „készen kapott” objektumrendszert tartalmaz.

Az egyes utasításoknál megadjuk az utasítások (esetleg egyszerűsített) szintaktikáját. Az egyszerűsítéssel a lényeges dolgok kiemelése a célunk, az utasítások teljes szintaktikája a súgóban megtalálható.

Megjegyzés

- Jegyzetünkben az MS Excel 2010 VBA segítségével szemléltetünk, a leírások is ehhez igazodnak, de a 2003-as Excel VBA környezete ugyanúgy alkalmas a tanulásra, a feladatok megoldására.
- Az utasítások szintaktikájában a szögletes zárójelben lévő részek elhagyhatók, a kapcsos zárójelek között, függőleges vonallal elválasztva választási lehetőségek felsorolása található, a három pont a tetszőleges számú ismétlés jelölésére szolgál.

1.1.1. Egyszerű adattípusok

Minden programozási nyelv (így a VB is) meghatározza az adatok azon körét, amelyet kezelni tud. Azt, hogy milyen fajta adatokat használhatunk, ezekkel milyen műveleteket végezhetünk, ezek hogyan tárolódnak, az *adattípusok* definiálják. Attól függően, hogy az adattípus egy vagy több logikailag összetartozó adat használatát engedi meg, megkülönböztetünk *egyszerű* és *összetett* adattípusokat. Az egyszerű adattípusokról ebben a fejezetben, az összetett adattípusokról az 1.1.5. fejezetben lesz szó.

1.1.1.1. Egész adattípusok

Az egész számok használatát többféle egész típus biztosítja, amelyek az adatok tárolására felhasznált memóriaterület méretében, az előjel kezelésében, így az egyes típusokhoz tartozó egész számok tartományában különböznek.

1.1. táblázat. Az egész adattípusok

Adattípus	Tárolási méret	Tartomány
Byte	1 bájt	0 .. 255
Integer	2 bájt	-32768 .. 32767
Long	4 bájt	-2147483648 .. 2147483647

Míg a Byte típus 1 bájtja csupán 2^8=256 db, addig a Long típus 4 bájtja már 2^{32} (kb. 4 milliárd) különböző érték (egész szám) tárolását biztosítja.

Az egész típusú adatokkal a matematikában szokásos műveletek végezhetők el, amelyet az 1.2. és 1.3. táblázatok szemléltetnek. Az aritmetikai műveletek táblázatbeli sorrendje a műveletek erősorrendjét (prioritás, precedencia) tükrözi, ahol legelöl a legerősebb (legmagasabb prioritású) hatványozás található.

A hasonlítások között nincs erősorrendbeli különbség (azonos prioritásúak), de prioritásuk gyengébb, mint az aritmetikai műveleteké. A hasonlítási műveletek nemcsak egész számokra, de más adatokra (pl. valós számok, sztringek, stb.) is értelmezettek, eredményük logikai típusú. A logikai adattípusról az 1.1.1.3. fejezetben, míg a kifejezések kiértékelésének szabályairól az 1.1.2.2. fejezetben lesz szó.

1.2. táblázat. Az egész adattípusok aritmetikai műveletei

Aritmetikai műveletek
Hatványozás (^)
Negáció (–)
Szorzás (*), osztás (/)
Egész osztás hányadosa (\)
Egész osztás maradéka (Mod)
Összeadás (+), kivonás (–)

1.3. táblázat. A hasonlítási műveletek

Hasonlítások
Egyenlő (=)
Nem egyenlő (<>)
Kisebb (<)
Nagyobb (>)
Kisebb vagy egyenlő (<=)
Nagyobb vagy egyenlő (>=)

Pl. -3^2 → –9 3/2 → 1.5 5\3 → 1 5 Mod 3 = 2 → True

Megjegyzés

• A kisebb vagy egyenlő (<=), illetve a nagyobb vagy egyenlő (>=) műveletek (ahogyan azt a nevük is sugallja) csak akkor igazak, ha a kisebb (<) vagy egyenlő (=), illetve a nagyobb (>) vagy egyenlő (=) műveletek legalább egyike igaz.

• A műveleteket operátoroknak, a műveletekben résztvevő adatokat operandusoknak, a hasonlításokat pedig relációs műveleteknek (vagy egyszerűen csak relációknak) is nevezik.

1.1.1.2. Valós adattípusok

A valós számok esetén kétféle típust használhatunk, amelyek jellemzőit az 1.4. táblázat szemlélteti. Noha a használható számok nagyságrendje kb. 10^{38}, a tárolásra használt 4, illetve 8 bájt csak kb. 7-8, illetve 15-16 értékes (decimális) számjegyet biztosít (a többi számjegy a kerekítés miatt 0 lesz).

1.4. táblázat. A valós adattípusok

Adattípus	Tárolási méret	Értékes jegyek
Single	4 bájt	7-8
Double	8 bájt	15-16

A valós adattípusok műveletei az egész osztás (\, Mod) műveletek kivételével megegyeznek az egész adattípusok műveleteivel.

Pl. $4^{\wedge} - 0.5 \rightarrow 0.5$

1.1.1.3. Logikai adattípus

A logikai (Boolean) adattípusban kétféle érték létezik, az igaz (True) és a hamis (False). A három legfontosabb logikai művelet a *tagadás* (Not), az *és* (And), és a *vagy* (Or), amelyek igazságtáblázatát az 1.5. táblázat szemlélteti. A logikai értékekre értelmezettek a hasonlítás műveletek (lásd 1.3. táblázat) is.

1.5. táblázat. Logikai műveletek

A	B	Not A	A And B	A Or B
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

A *tagadás* egyoperandusú művelet az ellenkezőjére változtatja a logikai értéket (igazból hamis lesz és fordítva), az *és* kétoperandusú művelettel összekapcsolt logikai kifejezés csak akkor lesz igaz, ha mindkét operandus igaz, míg a *vagy* kétoperandusú művelettel összekapcsolt logikai kifejezés csak akkor lesz hamis, ha mindkét operandus hamis.

Megjegyzés

- Logikai művelet még a *kizáró vagy* (Xor), az *ekvivalencia* (Eqv), és az *implikáció* (Imp) is.
- A logikai értékek között is értelmezett a sorrendiség (bár ezt ritkán használjuk), nevezetesen a False érték megelőzi a True értéket, azaz False < True → True, így a logikai értékekre definiált az összes hasonlítási művelet.

1.1.1.4. Szöveges adattípus

A szöveges (String) adattípus szöveges adatok (karaktersorozatok) használatát biztosítja. A String kulcsszó változó hosszú sztringeket deklarál, amelyek a maximális adathosszig (2^{31} darab karakter) tetszőleges hosszúak lehetnek. A sztringeket macskakörmök közé kell tenni. Az üres sztringnek ("") nincs egyetlen karaktere sem.

A sztringekre az *összefűzés* művelet (más néven *konkatenáció*) (+, &), és a hasonlítások (lásd 1.2. táblázat) értelmezettek. A + művelet csak szövegeket fűz össze, az & számokat is képes összefűzni, amelyeket a művelet elvégzése előtt sztringgé alakít (konvertál).

Pl. "alma" + "fa" → "almafa"

$3 \,\&\, 3 * 5 \rightarrow$ "315"

A hasonlítási műveletek kiértékelése a sztringek karakterei alapján történik. Két sztring egyenlő (=), ha egyforma hosszúak és karaktereik rendre megegyeznek, egyébként nem egyenlők (<>). A kisebb (<), illetve nagyobb (>) hasonlítási műveleteknél a sztringek első különböző karakterpárja határozza meg az eredményt (lásd megjegyzés). Ha nincs ilyen karakter (az egyik sztring kezdőszelete a másiknak), akkor a rövidebb sztring lesz a kisebb.

Pl. "Alma" < "alma" → True

"Kovács" < "Kovácsné" → True

"Kovacs" < "Kovács" → True

Megjegyzés

- A sztringek összehasonlítására az Option Compare (modulszintű) utasítás is hatással van.
  - Az Option Compare Binary (alapértelmezett) esetben a karakterek (Windows kódlapbeli) kódjai alapján történik a hasonlítás (A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø).
  - Az Option Compare Text utasítás olyan összehasonlítást eredményez, amely nem különbözteti meg a kis- és nagybetűket (case insensitive) ((A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)).
- Lehetőség van fix hosszú sztringek használatára is. Ekkor a String kulcsszó után (egy csillag karakterrel elválasztva) megadandó a sztringek hossza is (pl. String * 30). Egy ilyen típusú változóban tárolt sztring hossza fixen a típusban rögzített hossz lesz (a példában 30), a hosszabb sztringek jobbról csonkulnak, a rövidebbek pedig szóközökkel egészülnek ki. (A változókról az 1.1.2.1. fejezetben lesz szó, a fix hosszú sztringek használatáról egy példát az 1.1.5.2. és 1.1.5.3. fejezetekben láthatunk.) Bizonyos esetekben (pl. véletlenelérésű adatfájloknál) csak ez a sztringtípus használható. A maximális adathossz 2^{16} darab karakter.
- A sztringek minta alapján történő hasonlítására a Like művelet használható.

1.1.1.5. Egyéb adattípusok

A dátum/idő (Date) adattípus segítségével dátum és/vagy időadatokat tudunk kezelni. Egy (8 bájton tárolt) valós szám egész része a dátum, tört része pedig az időpont megadására használatos. Az időkezelés megegyezik az Excel időkezelésével, a dátumkezelés kicsit eltérő. Az Excel csak pozitív számokat tud dátumként értelmezni, a VB negatív számokat is kezel, amellyel a dátum 100.01.01-től 9999.12.31-ig terjedő érték lehet.

Pl. Excel: 1.25 ~ 1900.01.01 6:00:00, VB: 1.25 ~ 1899.12.31 6:00:00, –1.5 ~ 1899.12.29 12:00:00

A Currency adattípus a pénzügyi számításokhoz ajánlott. Egy ilyen típusú adat fixen négy tizedes jegyet tartalmaz, mert az adat tízezerszerese tárolódik egész számként (8 bájton, ami 19-20 értékes decimális jegy pontosságot jelent).

A Variant adattípust akkor használjuk, ha egy adatnak nem tudjuk előre a típusát, vagy egy változóban különböző típusú adatokat (pl. szám, szöveg) szeretnénk tárolni. A változókról az 1.1.2.1. fejezetben lesz szó. A kényelmes használat ára a nagyobb memóriaterület (szám esetén 16 bájt, szöveg esetén 22 + a szöveg karaktereinek számával megegyező bájt).

Megjegyzés: A Variant típusú változók speciális értékeket (Empty, Error, Nothing, Null) is tartalmazhatnak.

A VBA előszeretettel használja a felsorolt (Enum) típust akkor, amikor az adatoknak csak néhány lehetséges értéke van. Az Enum típusú adatok mindegyikéhez egy azonosító és egy egész szám rendelhető. A forrásprogramokban ugyan mindkettő használható, de célszerű az azonosítók használata, így a forráskód kifejezőbb, érthetőbb lesz.

Szintaktika:

[Private|Public] Enum *name*

membername[=*constantexpression*]

membername[=*constantexpression*]

...

End Enum

Private	A típus csak abban a modulban hivatkozható, amelyikben deklaráltuk.
Public	A típus minden modulból hivatkozható (ez az alapértelmezés).
name	A típus azonosítója.
membername	A típushoz tartozó elem azonosítója.
constantexpression	A típushoz tartozó elem értéke.

Az egyes elemek számértékeit megadó kifejezések (*constantexpression*) konstansokból álló, egész értékű kifejezések lehetnek. Ezek a kifejezések elhagyhatók, ekkor az első elem esetén 0, a többinél az előző értéknél eggyel nagyobb érték definiálódik.

Pl.

```
Enum SecurityLevel
    IllegalEntry = -1
    SecurityLevel1 = 0
    SecurityLevel2 = 1
End Enum
```

Megjegyzés

- Az Enum és End Enum utasítások közötti rész (példában szereplő) beljebb tagolása csak az áttekinthetőséget segíti.
- Az Enum utasítás modulszintű utasítás, azaz a modulok elején helyezendő el. A modulok felépítéséről az 1.2.2.5. fejezetben lesz szó.
- Az egyes utasítások szintaktikáját a VBA fejlesztőkörnyezet súgója alapján adjuk meg.

1.1.2. Adatok kezelése

Ahhoz, hogy adatainkat a számítógép kezelni tudja, tárolnia is kell. A tárolás mikéntje, konkrét megvalósítása egyrészt az adatok típusától, másrészt az alkalmazott fejlesztőkörnyezettől, és az operációs rendszertől is függ.

Adatainkat alapvetően a számítógép memóriájában tároljuk, de szükség esetén külső adathordozón (adatfájlokban) is eltárolhatjuk. A fájlokban történő adattárolásra akkor van szükség, ha adatainkat két programfutás között is meg szeretnénk őrizni. A memóriában történő adattárolásról ebben a fejezetben, az Excel-fájlokban (munkafüzetekben) történő adattárolásról az 1.3.1. fejezetben lesz szó.

1.1.2.1. Változó

Változón olyan azonosítóval ellátott memóriaterületet értünk, ahol a változó típusának megfelelő értéket (pl. adatot, részeredményt) tárolhatunk. Egy változóban tárolt érték a program végrehajtása során megváltozhat – innen ered az elnevezése –, ilyenkor a változóba kerülő új érték felülírja a régit.

A változók használatát általában megelőzi azok deklarálása, amikor is megadjuk a változó típusát. A legtöbb programozási nyelvben (pl. C, Pascal) kötelező a változók deklarálása, de a Visual Basic megengedi a változók deklaráció nélküli használatát is. A deklarálatlan változók típusa Variant lesz.

Mindazonáltal a deklarálatlan változók (a „kényelmes” használat mellett) lehetséges hibaforrások is egyben (pl. egy változó azonosítójának elgépeléséből adódó hiba csak futásidőben derül ki), ezért a VB külön utasítást biztosít arra, hogy a változókat deklarálni kelljen. Az Option Explicit (modulszintű) utasítás kikényszeríti a változók deklarálását azáltal, hogy szintaktikai (formai) hibát kapunk egy nem deklarált változó használatakor (lásd 1.2.2.6. fejezet).

A változók deklarálásának (egyszerűsített) szintaktikája:

Dim *varname* [As *type*] [,...]

varname	A változó (betűvel kezdődő) azonosítója (neve).
type	A változó típusa (ha hiányzik, a változó Variant típusú lesz).

Pl.

```
Dim i As Integer          'Egy Integer típusú változó deklarálása
Dim v                     'Egy Variant típusú változó
Dim a,b As Single         'Egy Variant és egy Single típusú változó
```

Megjegyzés

• A példában szereplő sorok végén magyarázó megjegyzések találhatók, amelyeket a Visual Basic Editor (lásd 1.2.2.4. fejezet) alapértelmezetten zöld színnel emel ki.

• Egy változó azonosítójának (mint minden más programbeli azonosítónak) be kell tartaniuk a VB névmegadási szabályait (naming rules). Ezek többek között előírják, hogy betűvel kell kezdődnie, nem haladhatja meg a 255 karaktert, nem tartalmazhat speciális karaktereket (pl. ., !, @, &, $, #), stb. Célszerű olyan beszédes (azaz a változó szerepére, a benne tárolt adatra/adatokra utaló), rövid, alfanumerikus karaktersorozatot használni,

amelyeknek nincs más jelentésük a VB-ben.

• Az azonosítókban magyar ékezetes betű, és az alulvonás karakter (_) is használható (pl. Év_Hó_Nap), de a kis és nagybetűk között nincs különbség. Egy változó azonosítója a deklaráláskor megadott formában jelenik meg hivatkozáskor (pl. ha i-t deklaráltunk, akkor az I-vel való hivatkozás i-re cserélődik a kódszerkesztő ablakban).

• Noha a deklarált változóknak a VB ad kezdőértéket (a numerikus változók 0, a logikai változók False, míg a String típusú változók üres sztring kezdőértéket kapnak), lehetőleg csak olyan változók értékeit használjuk fel, amelyeknek korábban már definiáltuk az értékét!

Típusdeklarációs karakterek

A VB nyelv (hasonlóan, mint a korábbi BASIC nyelvek) megengedi azt, hogy a nem deklarált változók típusát típusdeklarációs karakterrel jelezzük. Ezek használatának jelentősége csökkent, hiszen a VB-ben kikényszeríthetjük a változók deklarálását, amikor is explicit módon meg kell adnunk a változó típusát.

A használható karakterek és a hozzájuk tartozó adattípusok:

% Integer, ! Single, # Double, $ String, @ Currency

Megjegyzés

• Az első értékadásnál megadott típusdeklarációs karakter később el is hagyható.

• A változók típusa a Deftype utasításokkal (pl. DefInt, DefStr, ...) is szabályozható.

Pl.

```
i% = 2.8                    'Az i változóba 3 kerül (kerekítés)
c@ = 123456789012#          'A c változóba egy Double konstanst teszünk
st$ = 2                     'Az st változóba "2" kerül
st = st + st                'Az st változóba "22" kerül
```

A fenti példák az értékadó utasítást használják, amiről részletesen az 1.1.2.4. fejezetben lesz szó.

1.1.2.2. Kifejezés

Kifejezésen olyan számítási műveletsort értünk, amellyel megmondjuk, hogy milyen adatokkal, milyen műveleteket, milyen sorrendben kívánunk elvégezni. A kifejezés kiértékelésekor egy új érték – a kifejezés értéke – keletkezik.

A kifejezésben szerepelhetnek:

- Konstansok, változók, függvényhívások
- Műveletek
- Zárójelek

Pl. (-b+Sqr(b*b-4*a*c))/(2*a)

A példában a 4 és 2 konstansok, az a, b, c változók, az Sqr a négyzetgyök függvény.

A műveletek prioritása csökkenő erősorrendben:

- Aritmetikai
  - Hatványozás (^)
  - Negáció (–)
  - Szorzás, osztás (*, /)
  - Egész osztás hányadosa, maradéka (\, Mod)
  - Összeadás, kivonás (+, –)
- Szöveg összefűzés (&, +)
- Hasonlítások (=, <>, <, >, <=, >=, Like, Is)
- Logikai (Not, And, Or, Xor, Eqv, Imp)

Az egyes hasonlítások azonos prioritásúak, az egész osztás műveleteit és a logikai műveleteket csökkenő prioritás szerinti sorrendben adtuk meg.

A kifejezések kiértékelésének szabályai:

- A zárójelbe tett kifejezések és függvényhívások operandus szintre emelkednek.
- A magasabb prioritású műveletek végrehajtása megelőzi az alacsonyabb prioritású műveletek végrehajtását.
- Az azonos prioritású műveleteknél a balról-jobbra szabály érvényes, ami azt jelenti, hogy ezen műveletek végrehajtása balról jobbra haladva történik.

Megjegyzés

- Ha szükséges, akkor a típuskonverziók automatikusan végrehajtódnak.
- Az Is művelettel objektumhivatkozások egyezése vizsgálható.

Pl.

3 * 4 <= 12 And "a" & 12 = "a12" → True

3.8 \ 2 * 3 → 0

(3.8 \ 2) * 3 → 6

Az első példában a szorzás, majd az összefűzés (a 12 szöveggé konvertálásával), utána a hasonlítások, végül az And művelet hajtódik végre, amivel True értéket kapunk. A második és harmadik példában szereplő 3.8 valós szám egy egész osztásban szerepel, ezért értéke egészre konvertálódik (4-re kerekítődik). A második példában először a szorzás (eredménye 6), majd az egész osztás (4\6) hajtódik végre, így az eredmény 0 lesz. A harmadik példa zárójelezése megváltoztatja a prioritásból adódó sorrendet, így először az egész osztás (4\2) hajtódik végre, majd a szorzás, így az eredmény 6 lesz.

1.1.2.3. Függvények

A programozási nyelvek beépített függvényekkel segítik a számolást, adatfeldolgozást. Ezeket csak használnunk kell, azaz a megfelelő paraméterekkel meg kell hívnunk őket. Az alábbiakban (csoportokba foglalva) felsorolunk

néhány gyakran használatos függvényt (de hangsúlyozzuk, hogy ez a felsorolás korántsem teljes, a VB-ben sokkal több függvényt használhatunk).

Matematikai függvények

Abs(X)	X abszolút értéke.
Exp(X)	Az exponenciális függvény (e^x) értéke az X helyen.
Log(X)	A természetes alapú logaritmus függvény értéke az X helyen.
Sin(X)	X szinusza (X radiánban adott).
Cos(X)	X koszinusza (X radiánban adott).
Sqr(X)	X négyzetgyöke.
Int(X)	A legnagyobb egész szám, amely még nem nagyobb, mint X.
Rnd()	Egy véletlen szám a [0, 1) intervallumból.

Pl.

Int(3.8) $\rightarrow$ 3 Int(-3.8) $\rightarrow$ -4

Egy véletlen egész szám az [a,b] intervallumból: Int((b-a+1)*Rnd)+a

Megjegyzés: A paraméterek nélküli függvényeknél az üres zárójelpár elhagyható (mint a példában az Rnd esetén).

Konverziós függvények

Asc(X)	Az X karakter ASCII kódja.
Chr(X)	Az X ASCII kódú karakter.
Str(X)	Az X numerikus adat szövegként.
Val(X)	Az X számot tartalmazó szöveg numerikus értéke.

Pl.

Asc("A")→65 Chr(65)→ "A" Str(2.3)→" 2.3" Val("2.3")→ 2.3

Adott típusú értékké konvertáló függvények

CStr(X) X értékét String értékké.

CInt(X) X értékét Integer értékké.

CSng(X) X értékét Single értékké.

CDate(X) Az X érvényes dátumkifejezést Date értékké.

Pl.

CStr(2.6) → "2,6" CInt(2.6) → 3 CSng("2,6") → 2.6

Megjegyzés

• Ezekből a konvertáló függvényekből csak néhányat ragadtunk ki, de minden egyszerű adattípushoz létezik ilyen függvény.

• A kékkel kiemelt függvénynevek (csakúgy, mint a többi, kék színnel kiemelt szó) kulcsszavak, nem használhatók másra (pl. egy változó azonosítójának).

• Az átalakítandó X érték tetszőleges típusú érték lehet (pl. szám, szöveg, logikai érték).

• A CStr függvény az operációs rendszerbeli tizedesjelet használja (a példa azt az esetet szemlélteti, amikor a beállított tizedesjel a vessző).

• Ha számot tartalmazó szöveges adatot konvertálunk numerikus adattá, akkor tizedesjelként a vessző és az operációs rendszerben beállított tizedesjel használható. Ha pl. vessző az operációs rendszerben beállított tizedesjel, akkor a pont használata esetén típuskeveredési hibát kapunk (pl. CInt("2.6")).

• Ha az X érték nem konvertálható az eredménytípus értékkészletébe, akkor hibát kapunk (pl. CByte(-1)).

Szövegkezelő függvények

Len(X) Az X sztring hossza (karaktereinek száma).

Left(X,Y)	Az X sztring elejéről Y darab karakter.
Right(X,Y)	Az X sztring végéről Y darab karakter.
Mid(X,Y[,Z])	Az X sztring Y-adik karakterétől Z darab karakter.
Trim(X)	Az X sztring vezető és záró szóközeinek levágása.
InStr([X,]Y,Z)	A Z sztring megkeresése az Y sztringben (az X-edik karaktertől kezdődően).

Pl.

Len("Alma")→ 4	Left("Alma",2)→ "Al"	Right("Alma",2)→ "ma"
Mid("Alma",3,1)→ "m"	Mid("Alma",3)→ "ma"	Trim(" a b ")→ "a b"
InStr("alma","a")→l	InStr(2,"alma","a")→ 4	InStr("alma","A")→ 0

Megjegyzés

- Vezető szóközök levágása: LTrim, záró szóközök levágása: RTrim.
- Ha a Mid függvény Z paraméterét nem adjuk meg, vagy megadjuk, de nincs annyi darab karakter az Y-adik karaktertől kezdődően, akkor az eredmény az X sztring utolsó karakteréig tart.
- Az InStr függvény X paramétere elhagyható, ekkor a keresés az Y sztring első karakterétől indul.
- Létezik Mid utasítás is, amellyel egy sztring karakterei más karakterekre cserélhetők.

Dátum- és időkezelő függvények

Date	Az aktuális (operációs rendszerbeli) dátum.
Time	Az aktuális (operációs rendszerbeli) idő.

Megjegyzés: A dátumok hasonlóan kezelhetők, mint az Excel-ben (pl. a Date-1 kifejezés a tegnapi dátumot adja).

Adatformázó függvény

Format(X[,Y]) Az X kifejezés sztringgé alakítása az Y formázó sztring alapján.

Pl.

Format(100/3,"0.00") → "33,33"

Format(CDate("89.12.05"), "Long Date") → "1989. december 5."

1.1.2.4. Az értékadó utasítás

Az értékadó utasítással egy változónak adhatunk értéket. Az utasítás szintaktikája:

[Let] *varname* = *expression*

varname A változó azonosítója.

expression A tárolandó értéket meghatározó kifejezés.

Az utasítás végrehajtásakor először kiértékelődik az értékadás jobb oldalán álló kifejezés (*expression*), azaz kiszámítódik a kifejezés értéke, majd ha a kapott érték tárolható az adott változóban (*varname*), akkor megtörténik a tárolás, különben futási (run-time) hiba lép fel.

Megjegyzés

- Az utasítás kulcsszava (Let) elhagyható, ezért többnyire el is hagyjuk.
- A számolás közben, illetve a tárolás előtt a VB implicit (általunk nem definiált) típuskonverziókat hajthat végre. A megfelelő konverziós függvények használatával azonban explicit (általunk megadott) módon szabályozhatjuk a szükséges típuskonverziókat, így elkerülhetjük az implicit típuskonverziókat és az esetlegesen ebből eredő programhibákat.

Pl.

```
Dim i As Byte                'Az i változó deklarálása
i = 255                      'A legnagyobb Byte érték
i = i + 1                    'Túlcsordulási hiba (Overflow)!
```

```
v = "szöveg"          'Variant típusú változóba szöveget
v = 3.14              'Variant típusú változóba számot
i = "a"               'Típuskeveredési hiba (Type mismatch)
i = "2" + "2"         'Az i változóba 22 kerül
i = "2" + "2" * "2"   'Az i változóba 6 kerül
```

A példában egy Byte típusúra deklarált i és egy deklarálatlan (ezért Variant típusú) v változónak adunk különböző értékeket.

Az első értékadás a legnagyobb Byte típusú adatot teszi az i változóba. Ezzel nincs semmi baj, de ekkor a második értékadás túlcsordulási (overflow) futási hibát eredményez, hiszen a kifejezés értéke 256 lesz, ami már nem tárolható az i változóban. Ha az i változó aktuális értéke (tartalma) nem 255, akkor az utasítás rendben végrehajtódik, azaz az i változóban eggyel nagyobb szám lesz, mint az értékadás előtt.

A harmadik és negyedik értékadás azt szemlélteti, hogy egy Variant típusú változóba különböző típusú adatokat is tehetünk.

Az ötödik értékadás egy típuskeveredési (type mismatch) futási hibát ad, hiszen egy (számmá nem alakítható) szöveget szeretnénk tárolni egy numerikus változóban.

Az utolsó két értékadás az automatikus típuskonverziót szemlélteti. Az egyik értékadás kifejezésének eredménye 22, hiszen két db sztringet fűzünk össze, de az i változóba ennek a sztringnek az egész számmá alakított értéke kerül. A másik értékadásban a szorzás művelet hajtódik végre először (a nagyobb prioritás miatt), ezért a szorzás operandusai számmá konvertálódnak, majd összeszorzódnak (4). Mivel a + művelet egyik operandusa numerikus, ezért ez a művelet az összeadás művelet lesz (és nem a sztringek összefűzése), ami szintén numerikussá alakítja a bal oldali operandusát ("2"). Eredményül tehát a 6 érték számolódik ki, és tárolódik az i változóban.

1.1.2.5. Adatok bekérése

Egy változónak nemcsak az előbb ismertetett értékadó utasítással adhatunk értéket. Lehetőség van a változók értékeinek a program futása (végrehajtása) során történő megadására is. Például egy másodfokú egyenletet

megoldó program esetén célszerű az egyenletet meghatározó együtthatókat bekérni, ahelyett, hogy azok konkrét értékeit a programba beírnánk, hiszen a bekérő utasítás használatával a program módosítása nélkül tudunk különböző másodfokú egyenleteket megoldani. Elegendő csak újra futtatnunk a megoldó programot és más bemenő adatokat adni.

Az alábbi VB függvény egy párbeszédablak segítségével egy szöveges adat bekérését, megadását biztosítja. Az utasítás (egyszerűsített) szintaktikája:

InputBox(*prompt*[,*title*][,*default*])

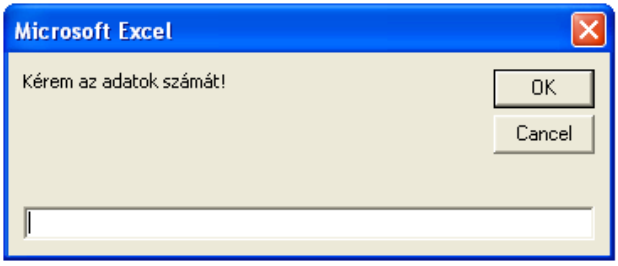
prompt	Tájékoztató üzenet.
title	Az ablak fejlécében megjelenő szöveg.
default	Az adat alapértelmezett értéke.

Pl.

```
Dim n As Integer

n = InputBox("Kérem az adatok számát!")
```

A példa egy egész szám bekérését szemlélteti, ahol csak a kötelező (*prompt*) paramétert adtuk meg. A bekért szöveges adatot egy numerikus változóba (n) tesszük (implicit típuskonverzióval), ezért rossz adat (pl. nem számmá alakítható szöveg) megadása esetén hibaüzenetet kapunk.

A függvény végrehajtásakor egy párbeszédablak jelenik meg (lásd 1.1. ábra). Az adatot a beviteli mezőben kell megadni. A függvény eredménye az OK gomb megnyomása (vagy az Enter billentyű leütése) esetén a megadott szöveg lesz, egyébként pedig (Cancel gomb, Esc billentyű, ablak bezárás) az üres sztring.



1.1. ábra. A példában szereplő InputBox függvény párbeszédablaka

Megjegyzés

- A függvénynek egyéb opcionális paraméterei is vannak.
- Valós számok bekérése esetén ügyeljünk arra, hogy az implicit típuskonverzió tizedesjelként csak a vesszőt és az operációs rendszerben beállított tizedesjelet fogadja el. Ha pl. vessző az operációs rendszerben beállított tizedesjel, akkor a pont használata esetén típuskeveredési hibát kapunk.

1.1.2.6. Adatok kiírása

Többnyire még a legegyszerűbb programoknak is vannak bemenő (input) és eredmény (output) adatai. A bemenő adatok egyik megadási lehetőségét az előzőekben ismertettük, az eredmények megjelenítési lehetőségeiről most lesz szó.

Az alábbi VB függvény egy párbeszédablak segítségével egy szöveges adatot jelenít meg. Az utasítás (egyszerűsített) szintaktikája:

MsgBox(*prompt*[,*buttons*][,*title*])

prompt	A kiírandó adat.
buttons	Az ablak nyomógombjait definiáló érték.
title	Az ablak fejlécében megjelenő szöveg.

Pl.

MsgBox("2*3=" & 2*3)

MsgBox "2*3=" & 2*3

MsgBox 3^2, ,"Három négyzete"

Az első két esetben a kiírandó adatot két adat (egy szöveg és egy szám) összefűzésével állítottuk elő, a harmadik esetben nem adtuk meg a *buttons* paramétert. A megfelelő párbeszédablakok az 1.2. ábrán láthatók (az első két MsgBox hívás ugyanazt eredményezi).

1.2. ábra. A példában szereplő MsgBox függvények párbeszédablakai

Megjegyzés

• A kiírandó adat tetszőleges típusú kifejezéssel megadható, ekkor a kifejezés értéke (implicit típuskonverzióval) sztringgé konvertálódik. Valós számok tizedesjele az operációs rendszerben beállított tizedesjel lesz.

• Ha a *buttons* paramétert elhagyjuk, akkor csak az OK gomb jelenik meg. A paraméterrel nemcsak a megjelenő nyomógombok, de a párbeszédablak egyéb tulajdonságai is definiálhatók.

• A függvénynek egyéb opcionális paraméterei is vannak.

• Noha a MsgBox függvény, a függvény visszatérési értékét csak akkor használjuk, ha be kell azonosítani, hogy melyik gombbal zárják be az ablakot.

• A példában szereplő MsgBox függvényt eljárásként hívtuk meg. A szubrutinokról és azok hívási szabályairól (pl. zárójel kirakás/elhagyás) az 1.1.4.1. fejezetben lesz szó.

Ha sok adatot kell megjelenítenünk, akkor az egyenkénti megjelenítés esetén minden adat után be kell zárnunk a megjelenő párbeszédablakot. Alkalmazva az előző példában szereplő gondolatot, célszerű az összes megjelenítendő adatot egy sztringgé összefűzni, így elegendő egyetlen MsgBox hívás az adatok megjelenítésére. Ennél a megoldásnál az adatokat a Chr(13) (carriage return), illetve Chr(10) (linefeed) karakterek segítségével többsoros szöveggé is alakíthatjuk. Mivel a MsgBox által kiírt szöveg max. 1024 db karakterből állhat, ezért nagy mennyiségű adat kiírására inkább az alábiakban ismertetésre kerülő megoldást használjuk.

Az adatmegjelenítés történhet a Debug objektum Print metódusának segítségével is, amely a Visual Basic Editor (lásd 1.2.2. fejezet) Immediate ablakába (lásd 1.2.2.1. fejezet) ír ki. Az objektumokról az 1.1.6. fejezetben, az Excel objektumairól az 1.3.1. fejezetben lesz szó.

Szintaktika:

Debug.Print [*outputlist*]

outputlist	A kiírandó kifejezést vagy kifejezéseket megadó lista.

A lista egy elemének (kifejezésének) megadási szintaktikája:

[{Spc(*n*)|Tab(*n*)}] *expression charpos*

Spc(*n*)	*n* db szóköz kiírása (opcionális).
Tab(*n*)	A kiírás az *n*-edik oszlopban kezdődjön (opcionális).
expression	A kiírandó kifejezés (opcionális).
charpos	A következő kiírandó adat kezdőpozíciójának megadása (opcionális).

Pl.

```
Debug.Print ("2*3=" & 2*3)
Debug.Print "2*3="; 2*3; Tab(10); "3*4=”; 3*4
```

Megjegyzés

- A metódushívásokra a szubrutinok hívási szintaktikája érvényes (lásd 1.1.4.1. fejezet).

• Ha a *charpos* pontosvessző, akkor az aktuális sorban folytatódik a kiírás, egyébként meg a következő sor elején, így egy Debug.Print hívás utolsó adata után kirakott vagy elhagyott pontosvesszővel szabályozható, hogy a következő Debug.Print hol kezdje a kiírást.

• Ha a kiírás már meghaladta a Tab(*n*) által megadott oszlopot, akkor a következő sor *n*-edik oszlopában folytatódik a kiírás.

• A kiírandó adatok vesszővel is elválaszthatók.

• A Print metódus az operációs rendszerbeli tizedesjelet használja.

• Az Immediate ablak (lásd 1.2.2.1. fejezet) tartalma szerkeszthető. Szerkesztéskor ügyeljünk arra, hogy a kiírás majd attól a helytől kezdődik el, ahol a kurzor állt, így könnyen összekeveredhetnek az egyes futások által kiírt adatok. Célszerű tehát az ablakban lévő szöveg végére állni (pl. Ctrl+End), vagy törölni az ablak teljes tartalmát (pl. Ctrl+A, Del), mielőtt elindítunk egy programot, amely az Immediate ablakba ír.

• Az ismertetett utasításokon kívül más eszközök is rendelkezésünkre állnak arra, hogy a programunk adatokat kapjon, illetve adatokat jelenítsen meg. Az 1.2.2.7. fejezetben vizuális vezérlők segítségével, míg az 1.3.1.3. fejezetben Excel munkafüzet segítségével valósul meg a felhasználói input/output.

• A VB általános fájlkezeléséről nem lesz szó, de azt megemlítjük, hogy a fájlkezelő utasítások segítségével adatainkat fájlba menthetjük, illetve onnan visszatölthetjük, így két különböző futás között is megőrizhetjük adatainkat.

Önellenőrzés

A kérdések után feladatokat talál. Ezek megoldásához szükséges plusz ismeretanyag:

- Makrók (1.2.1. fejezet).
- A Visual Basic Editor használata (1.2.2. fejezet).
- Az Immediate ablak (1.2.2.1. fejezet).

A feladatokat az Immediate ablak segítségével, az utasítások közvetlen végrehajtásával oldjuk meg!

1. Az alábbi állítások közül melyek igazak az Excel VBA fejlesztőkörnyezetre vonatkozóan?
   - ☐ Csak VB nyelven programozhatunk.
   - ☐ Az Excel nélkül is használható.
   - ☐ Önállóan futtatható (*.exe) fájlok készíthetők.
   - ☐ A forrásprogramokat C# nyelven is megírhatjuk.
   - ☐ Az Excel objektumok kezelését objektumrendszer támogatja.
   - ☐ Eseményvezérelt programozást biztosít.
   - ☐ Nem támogatja a vizuális tervezést.
   - ☐ Magas szintű programozási nyelven programozható.
2. Az alábbi állítások közül melyek igazak az egyszerű adattípusokra vonatkozóan?
   - ☐ A Byte típussal előjeles egész számok is kezelhetők.
   - ☐ Az Integer típussal előjeles egész számok is kezelhetők.
   - ☐ A Long típus 2 bájtot használ egy egész számérték tárolására.
   - ☐ A Double típus 11-12 értékes (decimális) számjegyet biztosít.

- [ ] A VB Date adattípusával időszámítás előtti dátumokat is kezelhetünk.
- [ ] A VB Date adattípusával honfoglaláskori dátumokat is kezelhetünk.

3. Az alábbi állítások közül melyek igazak az egyszerű adattípusok műveleteire vonatkozóan?

- [ ] A / művelet értelmezett két egész számra.
- [ ] A \ művelet értelmezett két valós számra.
- [ ] Az előjel művelet erősebb prioritású, mint a hatványozás művelet.
- [ ] Az And művelettel összekapcsolt logikai kifejezés igaz, ha valamelyik operandus igaz.
- [ ] Az Or művelettel összekapcsolt logikai kifejezés hamis, ha mindkét operandus hamis.
- [ ] A + sztringösszefűző művelet csak sztringeket tud összefűzni.
- [ ] Az & sztringösszefűző művelet sztringgé konvertálja az operandusait, ha szükséges.

4. Az alábbi állítások közül melyek igazak a változók deklarálására vonatkozóan?

- [ ] Egy változó azonosítója az alulvonás karakterrel is kezdődhet.
- [ ] Egy változó azonosítójában kis és nagybetűk, valamint magyar ékezetes betűk is szerepelhetnek.
- [ ] Egy változó azonosítójában a szóköz is szerepelhet.
- [ ] Ha egy változónak deklaráláskor nem adjuk meg a típusát, akkor a változó Variant típusú lesz.
- [ ] A VB megengedi a deklarálatlan változók használatát is.
- [ ] A VB-ben típusdeklarációs karaktereket is használhatunk.
- [ ] A változók kötelező deklarálása kikényszeríthető.

5. Az alábbi állítások közül melyek igazak a kifejezések kiértékelésével kapcsolatosan?

- [ ] A numerikus (aritmetikai) műveletek prioritása erősebb, mint a szöveges műveleteké.
- [ ] A logikai műveletek erősebb prioritásúak, mint a hasonlítás műveletek.
- [ ] A szorzás és osztás műveletek erősebb prioritásúak, mint az egész osztás műveletek.
- [ ] A balról jobbra szabály az azonos prioritású műveletekre vonatkozik.
- [ ] Egy kifejezés kiértékelésekor implicit típuskonverziók is végrehajtódhatnak.

- [ ] A / és a \ azonos prioritású művelet.
- [ ] Az And művelet a legerősebb prioritású logikai művelet.

6. Az alábbi állítások közül melyek igazak a VB függvényeivel kapcsolatosan?
   - [ ] Rnd függvény egy véletlen egész számot ad eredményül.
   - [ ] Az Int függvény kerekít.
   - [ ] A Left függvény második paramétere elhagyható.
   - [ ] A Mid függvény harmadik paramétere elhagyható.
   - [ ] Az InStr függvénynek van opcionális paramétere.
   - [ ] A Format függvény eredménye sztring típusú.
7. Kifejezések kiértékelése
   • A kiíró utasítás (Print, ?) segítségével értékeljünk ki kifejezéseket azért, hogy az egyes műveleteket, azok prioritását, a függvények használatát megismerjük!
   • Kipróbálandók az egyszerű adattípusok műveletei (kb. 20 db), és a kifejezéseknél felsorolt függvények (kb. 20 db).
   Pl.

```
Print 3.8\2*3; (3.8\2)*3; mid("alma",3,1)
? -2^2; (-2)^4; int(-3.2)
```

8. Értékadó utasítások végrehajtása
   • Az értékadó és kiíró utasítások segítségével nézzük meg az egyes adattípusok értékkészletét, ábrázolási pontosságát, az értékadó utasítás esetleges automatikus típuskonverzióit!
   • Kipróbálandó az összes egyszerű adattípus (kb. 10 db).
   Pl.

```
i=2
j=10
? i; j; i^j; j^i
```

2. LECKE

Vezérlőszerkezetek, szubrutinok

1.1.3. Vezérlőszerkezetek

Az eddigiekben szó volt a változók deklarálásáról, az értékadó utasításról, az adatbekérő és adatkiíró lehetőségekről, így ezekből már összeállíthatunk egy olyan programot, amely bizonyos bemenő adatokból valamilyen eredményadatokat számol, és ezeket meg is jeleníti. A megfelelő utasításokat ilyenkor egyszerűen sorban egymás után (szekvenciálisan) megadjuk, illetve végrehajtatjuk.

De mi van akkor, ha egy utasítást csak egy adott feltétel teljesülése esetén kell végrehajtani, vagy ha egy utasítást többször is végre szeretnénk hajtani? Szükségünk van olyan eszközökre, amelyekkel az egyes tevékenységek végrehajtási sorrendje előírható. Ezeket az eszközöket *vezérlőszerkezeteknek* nevezzük.

A strukturált programozás három vezérlőszerkezet használatát engedi meg, ezek a *szekvencia*, a *szelekció* és az *iteráció*. Bizonyítottan minden algoritmus megadható ezekkel a vezérlőszerkezetekkel, ezért a feladatok megoldásánál mi is csak ezeket fogjuk használni.

Szekvencia:	Az utasítások egymás után, a megadásuk sorrendjében hajtódnak végre.
Szelekció:	A végrehajtandó utasítás(ok) feltétel(ek)től függően kerül(nek) kiválasztásra.
Iteráció:	Egy vagy több utasítás ismételt végrehajtása.

Megjegyzés

- A strukturáltság és a forrásprogramok ehhez igazodó tagolása áttekinthetőbb, ezáltal könnyebben karbantartható programokat eredményez.
- A VB megenged nem strukturált programokat eredményező, speciális vezérlésátadó utasításokat (pl. GoTo, Exit) is, de ezekkel nem foglalkozunk.

1.1.3.1. Szekvencia

A szekvencia vezérlőszerkezetben a program utasításai (algoritmus esetén ez egyes lépések, tevékenységek) egymást követően, az utasítások megadási sorrendjében hajtódnak végre. A VB-ben az egyes utasításokat általában külön sorokba írjuk, de egy sorba több utasítást is írhatunk, ekkor az utasítások közé kettőspontot kell tenni.

Pl.

```
' Utasítások megadása egy sorban (végrehajtás: balról jobbra haladva)
Dim i As Integer: i = 3 * 5: MsgBox (i)
' Utasítások megadása egymást követő sorokban (végrehajtás: fentről lefelé)
Dim i As Integer
i = 3 * 5
MsgBox (i) Megjegyzés: A kétféle megadás tetszés szerint, vegyesen is alkalmazható.
```

1.1.3.2. Szelekció

A szelekció vezérlőszerkezet segítségével a végrehajtandó utasítások között szelektálhatunk, azaz megadhatjuk, hogy mikor melyik utasítás, illetve utasítások hajtódjanak végre. Az alábbiakban azt a szelekciós utasítást ismertetjük, amelyben a végrehajtandó utasítások (*statements*) logikai kifejezésekkel (*condition*) választhatók ki.

Az If utasítás szintaktikája:

```
If condition Then [statements] [Else elsestatements]
```

vagy

```
If condition Then
        [statements]
[ElseIf condition-n Then
        [elseifstatements]]
...
```

[Else

    [*elsestatements*]]

End If

Az első esetben az egész utasítás egy sorba kerül (ekkor nem kell End If az utasítás végére), míg a második esetben több sorba (ahol az egyes ágakban lévő utasítások beljebb tagolása nem kötelező, de javítja az áttekinthetőséget).

Végrehajtás:

• Az első teljesülő feltételhez tartozó utasítások kerülnek végrehajtásra.

• Ha egyik feltétel sem teljesül és az Else ág definiált, akkor ezek az utasítások (*elsestatements*) kerülnek végrehajtásra.

Pl.

```
If d = 0 Then
    MsgBox ("Nulla")
ElseIf d > 0 Then
    MsgBox ("Pozitív")
Else
    MsgBox ("Negatív")
End If
```

Megjegyzés

• Az egyes ágakban újabb If utasítások is lehetnek, azaz az If utasítások egymásba ágyazhatók.

• A VB másik szelekciós utasítása a Select Case utasítás, amely egy (általában nem logikai, hanem pl. egész, szöveg) kifejezés értéke alapján választja szét az egyes ágakat. Az utasítást nem részletezzük, de megjegyezzük, hogy minden Select Case utasítás felírható If utasítás segítségével is.

1.1.3.3. Iteráció

Az iteráció vezérlőszerkezet segítségével egy vagy több utasítás ismételt végrehajtása valósítható meg. Azokat az utasításokat, amelyeket az iteráció (vagy más néven ciklus) ismételten végrehajt, ciklusmagnak nevezzük. Három alapvető ciklusfajta létezik: az *elöltesztelő,* a *hátultesztelő* és a *növekményes* ciklus.

Az elöltesztelő ciklus a ciklust vezérlő feltételt a ciklusmag végrehajtása előtt vizsgálja, a hátultesztelő ciklus pedig a ciklusmag végrehajtása után. A növekményes ciklus egy speciális elöltesztelő ciklus, amely a ciklusmagot egy változó adott értékei mellett hajtja végre.

Mivel a ciklusszervezés a programozás egyik alapvető eleme, ezért a magas szintű programozási nyelvek (így a VB is) mind a három ciklusfajtát támogatják. Ez lehetővé teszi, hogy az adott feladathoz legjobban illeszkedő ciklusfajtát válasszuk, amivel a megoldásunk (programunk) érthetőbb és áttekinthetőbb lesz, de megjegyezzük, hogy az elöltesztelő ciklussal a másik két ciklusfajta is megvalósítható.

A VB ciklusszervező utasításait ismertetjük a továbbiakban.

A While ciklus szintaktikája:

While *condition*

    [*statements*]

Wend

Végrehajtás:

- Amíg a ciklust vezérlő feltétel (*condition*) igaz, addig végrehajtódnak a ciklusmag utasításai (*statements*).
- A feltételt mindig a ciklusmag végrehajtása előtt vizsgálja (elöltesztelő ciklus), ezért lehet, hogy a ciklusmag egyszer sem kerül végrehajtásra (ekkor *üres ciklusról* beszélünk).

Megjegyzés: A ciklusmag utasításainak hatással kell lennie a feltétel igazságértékére (ez a Do ciklusra is érvényes), különben a ciklus üres ciklus, vagy végtelen ciklus (ekkor a Ctrl+Break vagy az Esc billentyűkkel leállítható a programfutás).

Pl.

```
i = 1
While i <= 3
    MsgBox i: i = i + 1
Wend
```

A For ciklus szintaktikája:

```
For counter=start To end [Step step]
    [statements]
Next [counter]
```

Végrehajtás:

- A ciklusmag (*statements*) végrehajtási feltétele:

  counter<=*end*, ha *step*>=0

  counter>=*end*, ha *step*<0

- A feltételt mindig a ciklusmag végrehajtása előtt vizsgálja (elöltesztelő ciklus), ezért lehet, hogy a ciklusmag egyszer sem kerül végrehajtásra.

- A ciklusváltozó (*counter*) a ciklusmag minden egyes végrehajtásakor a *step* értékével megváltozik. A *step* megadása nem kötelező, ha elhagyjuk, akkor az alapértelmezett értéke 1.

Pl.

```
For i = 1 To 5 Step 2
    MsgBox i
Next
```

Megjegyzés: A VB-ben létezik egy olyan speciális ciklusutasítás is (For Each), amelyben a ciklusváltozó egy gyűjtemény objektumain „lépdel végig” (lásd 1.3.2.4. fejezet).

A Do ciklus szintaktikája:

Do [{While|Until} *condition*]

    [*statements*]

Loop

vagy

Do

    [*statements*]

Loop [{While|Until} *condition*]

Végrehajtás:

• A ciklust vezérlő feltétel (*condition*) vizsgálata az első esetben a ciklusmag (*statements*) végrehajtása előtt (előltesztelő ciklus), a második esetben pedig utána (hátultesztelő ciklus) történik meg.

• While kulcsszó esetén, ha a feltétel igaz, akkor a ciklusmag (ismételt) végrehajtása következik, egyébként a ciklusból való kilépés.

• Until kulcsszó esetén, ha a feltétel igaz, akkor a ciklusból való kilépés történik, egyébként a ciklusmag (ismételt) végrehajtása.

A Do ciklusutasítás tehát egy olyan általános ciklusszervező utasítás, amellyel mind az elöltesztelő, mind a hátultesztelő ciklusfajta megvalósítható, ráadásul úgy, hogy a feltétellel megadhatjuk az ismétlés, illetve a kilépés feltételét is.

Megjegyzés

• A ciklusmag utasításainak beljebb tagolása nem kötelező, de az áttekinthetőség érdekében célszerű.

• Az If utasításhoz hasonlóan a ciklusok is egymásba ágyazhatók.

• Vegyük észre a For ciklus *step*=0 esetét, és a Do ciklust vezérlő feltétel elhagyhatóságát! Ekkor a ciklusokból való kilépés az Exit For, Exit Do utasítások segítségével történhet, amelyeket a strukturáltság megőrzése érdekében nem említettünk (de a VB megengedi a használatukat).

1.1.4. Szubrutinok

Szubrutinon egy jól meghatározott, önálló tevékenységsort megvalósító, formailag is elkülönülő programrészt értünk. A szubrutinok tényleges végrehajtásához általában egy másik, a szubrutint aktivizáló, azt *meghívó* programrész is szükséges. Egy szubrutin többször is meghívható, így annak szolgáltatásai a program megfelelő helyein egy-egy hívással igénybe vehetők.

A szubrutin a programtervezés, illetve programozás alapszintű, de egyben alapvető fontosságú eszköze. Használatukkal érvényesíthető az *egységbezárási* elv, miszerint egy adott feladat megoldásához szükséges adatoknak és a „rajtuk dolgozó” algoritmusnak egy olyan egysége valósítható meg, amely csak a szükséges mértékben kommunikál a „külvilággal”, a szubrutint hívó programrészekkel, egyébként zárt, „dolgait” saját maga „intézi”, azokba „nem enged bepillantást”.

Megkülönböztetjük a szubrutin *deklarálását,* ahol definiáljuk a szubrutin által végrehajtandó tevékenységeket, a külvilággal való kapcsolattartás módját és szabályait, valamint a szubrutin *hívását,* amikor felhasználjuk a szubrutin szolgáltatásait, vagyis definiálva a kapcsolattartás eszközeit, kiváltjuk a tevékenységsor egy végrehajtását.

A kapcsolattartás legfőbb, de nem kizárólagos (hiszen pl. „mindenhonnan látható” (public) változókon keresztül is történhet a kommunikáció) eszközei a *paraméterek*.

A helyes programozási stílus a *teljes paraméterezésre* való törekvés. Ez azt jelenti, hogy egy szubrutin a működéséhez szükséges adatokat a paraméterein keresztül kapja, és az eredményeket (a függvényérték kivételével) ezeken keresztül adja vissza. Ezzel ugyanis elkerülhető egy esetleges módosítás olyan „mellékhatása”, amely a program egy „távoli részének” végrehajtásakor, előre nem látott, nem tervezett változást, s ezzel többnyire nehezen felderíthető hibát okoz.

Megjegyzés

• A VBA-ban megírt összes forrásprogram modulokba, azon belül szubrutinokba szervezett. Egy VBA projekt felépítéséről az 1.2.2.2. fejezetben lesz szó.

• Az objektumorientált programozás objektumtípusai (osztályai) az adatoknak és a rajtuk dolgozó algoritmusoknak egy még általánosabb egységét valósítják meg azáltal, hogy az adatokhoz nemcsak egy, de tetszőleges számú algoritmus rendelhető. Az algoritmusokat megvalósító szubrutinok (más terminológiával metódusok, tagfüggvények) egyrészt „látják" az objektum adatait, másrészt ők definiálják, írják le, modellezik az objektum viselkedését. Osztályokat nem fogunk definiálni, de az Excel objektumait (lásd 1.3.1. fejezet) használni fogjuk.

1.1.4.1. Deklaráció és hívás

Kétféle szubrutint különböztetünk meg, az *eljárást* és a *függvényt*. Az eljárást általában akkor használjuk, ha valamilyen tevékenységet kell végrehajtanunk, a függvényt pedig akkor, ha valamilyen eredményértéket kell kiszámolnunk.

Mivel mind az eljárás, mind a függvény képes adatokat kapni és eredményadatokat visszaadni, így egy adott szubrutin eljárás, illetve függvény alakban is programozható. Az, hogy egy szubrutint eljárás vagy függvény formájában írunk meg, nem elvi kérdés, inkább programozói stílus kérdése. A két forma egymással ekvivalens módon mindig helyettesíthető. Az eljárások és függvények deklarálása és hívása, ha kicsit is, de eltérő.

Eljárás deklarálásának (egyszerűsített) szintaktikája:

[Private|Public] Sub *name*[(*arglist*)]

[*statements*]

End Sub

Függvény deklarálásának (egyszerűsített) szintaktikája:

[Private|Public] Function *name*[(*arglist*)] [As *type*]

[*statements*]

[*name*=*expression*]

End Function

Private	A szubrutin csak abból a modulból hívható, amelyikben deklaráltuk.
Public	A szubrutin minden modulból meghívható (ez az alapértelmezés).
name	A szubrutin azonosítója.
arglist	A szubrutin formális paraméterei (opcionális).
type	A függvény eredményértékének a típusa.

Vegyük észre, hogy a függvény utasításai között szerepel egy olyan értékadó utasítás is, amivel a függvény nevének (*name*) mint változónak adunk értéket. Ez az utasítás szolgál a függvény eredményének a beállítására. Több ilyen utasítás is elhelyezhető a függvény utasításai között, de célszerű egyet elhelyezni, azt is a függvény végén, azaz az End Function utasítás előtt.

A formális paraméterek megadásának szintaktikája:

[Optional][ByVal|ByRef][ParamArray]*varname*[()][As *type*][=*defaultvalue*]

Optional	A paraméter elhagyható.
ByVal	A paraméter átadása érték szerinti.
ByRef	A paraméter átadása cím szerinti (ez az alapértelmezés).
ParamArray	Tetszőleges számú Variant típusú paraméter átadásához.
varname	A paraméter azonosítója.
type	A paraméter típusa.
defaultvalue	Az opcionális paraméter alapértelmezett értéke.

A paraméterek neve előtt álló kulcsszavakkal a paraméterek elhagyhatósága, valamint a paraméterek átadási módja definiálható. Az érték szerinti (ByVal) paramétereken keresztül a szubrutin csak kaphat adatokat, míg a cím szerinti átadású (ByRef) paramétereken keresztül kaphat is, és vissza is adhat eredményeket. Ezért a ByVal paramétereket a szubrutin bemenő (input) adatainak megadásához, a ByRef paramétereket pedig a szubrutin bemenő és/vagy kimenő (output) adatainak megadásához használhatjuk.

Megjegyzés

- Ha egy szubrutinnak több paramétere van, akkor a paramétereket vesszővel kell elválasztani.
- Egy Optional paramétert csak Optional paraméterek követhetnek, azaz egy elhagyható paramétert nem követhet kötelezően megadandó paraméter.
- A ParamArray paraméter csak a paraméterlista utolsó eleme lehet. Ilyen paraméter esetén nem használhatunk Optional paramétereket.
- Ha egy paraméternek nem adjuk meg a típusát, akkor a paraméter Variant típusú lesz.
- A paraméter neve (*varname*) mögötti üres zárójelpár azt jelzi, hogy az átadott paraméter egy tömb (lásd 1.1.5.1. fejezet). A tömböket csak cím szerinti paraméterátadással lehet átadni.

Eljárás hívása

[Call] *name*[*argumentlist*]

name	A meghívott eljárás azonosítója.
argumentlist	Az aktuális paraméterek.

Az eljárás hívásának kulcsszava (Call) elhagyható. Megadásakor az argumentumlistát zárójelbe kell tenni, ha elhagyjuk, akkor a zárójelet is el kell hagyni.

Függvény hívása

A függvények kifejezések részeként hívhatók (hasonlóan, mint a VB függvényei), azaz a függvényhívás bárhol állhat, ahol egy, a függvény eredményének megfelelő érték állhat (pl. egy értékadó utasítás jobb oldalán). Az aktuális paraméterek zárójelei nem hagyhatók el.

Egy szubrutin hívásakor megtörténik az aktuális és formális paraméterek megfeleltetése (a paraméterek átadása), majd a végrehajtás a hívott szubrutin utasításaival folytatódik. A hívott szubrutin befejeztével a végrehajtás a hívást követő utasítással folytatódik.

Hívás megnevezett paraméterekkel

Ez a fajta szubrutinhívás kényelmes hívást biztosít olyan szubrutinok esetén, amelyeknek sok opcionális paramétere van. Elegendő csak azokat a paramétereket és aktuális értékeiket felsorolni, amelyeket át akarunk adni, a többinek meghagyjuk az alapértelmezett értékét.

Pl.

```
ActiveWorkbook.SaveAs FileName:="Mentes.xls"
```

A példában egy objektum egy metódusát hívtuk meg, ami az aktív munkafüzetet menti el a megadott névvel. Az objektumokról és a metódusokról az 1.1.6., illetve 1.3.1. fejezetekben lesz szó.

Megjegyzés

• Általában a programnyelvekben kötött a szubrutinok deklarálásának és hívásának sorrendje, nevezetesen a deklarálásnak meg kell előznie (forráskód szinten) a hívást, de a VB megengedi, hogy egy szubrutint a hívó szubrutin után deklaráljunk.

• Egy függvény eljárásként is meghívható (pl. ha a függvény eredményére nincs szükségünk). Ekkor az eljárás hívási szabályai érvényesek. A példáinkban szereplő MsgBox függvényt is eljárásként hívtuk meg a Call szó elhagyásával. Az egyetlen paramétert néhol zárójelbe raktuk, de ekkor a zárójelek nem a MsgBox függvény zárójelei, hanem a paraméter egy felesleges zárójelezése.

• Ha egy szubrutint eljárásként hívunk meg egyetlen paraméterrel és elhagyjuk a Call szót, akkor a paraméter esetleges (felesleges) zárójelezésével a paraméter átadása érték szerinti lesz.

• A metódushívásokra a szubrutinok hívási szabályai érvényesek.

• A megnevezett paraméterek tetszőleges sorrendben megadhatók. A paraméterek neve és értéke közé a legyen egyenlő (:=) jelet, míg a paraméterek közé vesszőt kell tenni.

1.1.4.2. Mintafeladat

A szubrutinok deklarálását és hívását az alábbi feladat segítségével szemléltetjük.

Feladat: Fordítsunk meg egy adott sztringet!

Pl. "Réti pipitér" → "rétipip itéR"

Az alábbiakban mind függvénnyel, mind eljárással megoldjuk a feladatot, és mindkét esetben egy hívó, tesztelő szubrutint is készítünk. Erre azért van szükség, mert a paraméterekkel rendelkező szubrutinok nem futtathatók közvetlenül (lásd 1.2.2.6. fejezet), márpedig a megoldó függvénynek, illetve eljárásnak lesz paramétere. A paramétert a megfordítandó sztring átadására, az eljárással történő megoldás esetén még az eredmény visszaadására is használjuk.

Az első megoldásban (Fordit függvény) a kezdetben üres eredménysztringhez (er) jobbról hozzáfűzzük a sztring karaktereit fordított sorrendben, azaz először a legutolsót, majd az utolsó előttit, legvégül az elsőt. Amikor az eredménysztring előállt, akkor a megfelelő értékadással (amikor is a függvény neve értéket kap) gondoskodunk az eredmény beállításáról.

A második megoldásban (MegFordit eljárás) a kezdetben üres eredménysztringhez (er) balról hozzáfűzzük a sztring karaktereit eredeti sorrendben, aminek eredményeként a sztring első karaktere kerül az eredménysztring legvégére, a második karaktere lesz az utolsó előtti, végül az utolsó karaktere kerül az eredménysztring legelejére.

Természetesen a megoldások lényegét adó ciklusok lehetnének egyformák is, de a kétfajta megközelítéssel azt szerettük volna érzékeltetni, hogy még egy ilyen egyszerű feladatnak is létezhet többféle megoldása.

Megjegyzés: Olyan megoldás is elképzelhető, amelyben a sztring megfelelő karakterpárjait cserélgetjük fel. Az első karaktert az utolsóval kell kicserélni, a másodikat az utolsó előttivel, és így tovább.

```
'Sztring megfordítása függvénnyel
Function Fordit(st As String) As String
Dim i As Integer
Dim er As String
```

```
er = ""
For i = Len(st) To 1 Step -1
        er = er + Mid(st, i, 1)
Next
Fordit = er
End Function
'A Fordit tesztjéhez ez indítandó
Sub Teszt1()
'Híváskor a cím szerinti (ByRef) paraméter (st) helyén állhat érték is
MsgBox Fordit("Kitűnő vőt rokonok orrtövön ütik")
End Sub
'Sztring megfordítása eljárással
Sub MegFordit(st As String)
Dim i As Integer, er As String
er = ""
For i = 1 To Len(st)
        er = Mid(st, i, 1) + er
Next
'A cím szerinti (ByRef) paraméterben (st) adjuk vissza az eredményt
st = er
End Sub
'A MegFordit tesztjéhez ez indítandó
```

```
Sub Teszt2()
Dim s As String
'Call esetén kell a zárójel
s = "Indul a görög aludni": Call MegFordit(s): MsgBox s
'Call nélkül nem kell a zárójel
s = "Géza kék az ég": MegFordit s: MsgBox s
End Sub
```

A fejezet lezárásaként megjegyezzük, hogy a VB nem engedi meg a szubrutinok egymáson belüli deklarálását, de a rekurziót (amikor is egy szubrutin önmagát hívja meg, vagy több szubrutin hívja egymást kölcsönösen) igen. A rekurzív algoritmusok iránt érdeklődőknek a szakirodalmat (pl. [3], [4]) ajánljuk.

Önellenőrzés

1. Az alábbi állítások közül melyek igazak a vezérlőszerkezetekkel kapcsolatosan?
   - ☐ Egy If utasításban lehet olyan eset is, amikor egyik feltétel sem teljesül.
   - ☐ Az elöltesztelő ciklusban a ciklusmag legalább egyszer végrehajtódik.
   - ☐ A While ciklus hátultesztelő ciklus.
   - ☐ A For ciklus elöltesztelő ciklus.
   - ☐ A Do utasítással mind elöl-, mind hátultesztelő ciklus programozható.
2. Az alábbi állítások közül melyek igazak a szubrutinokkal kapcsolatosan?
   - ☐ Egy szubrutin pontosan annyi paraméterrel hívható meg, ahány paraméterét deklaráltuk.
   - ☐ Egy ByVal kulcsszóval deklarált paraméterben a szubrutin eredményt tud visszaadni.
   - ☐ Ha egy szubrutinnak egynél több eredménye van, akkor csak eljárásként programozható.
   - ☐ Egy függvénynek nem lehet elhagyható (opcionális) paramétere.
   - ☐ Egy függvény meghívható eljárásként is.
   - ☐ A megnevezett paraméterek tetszőleges sorrendben megadhatók.

A feladatok megoldásához szükséges plusz ismeretanyag:
- A kódszerkesztő ablak (1.2.2.4. fejezet).
- Futtatás, fordítás (1.2.2.6. fejezet).

Hozzunk létre egy új modult, majd készítsünk egy vagy több szubrutint (ebbe a modulba) az alábbi témakörökben (eljárás formájában), és futtassuk őket! Próbáljuk ki a projekt fordítását is!

3. Az adattípusok és az értékadó utasítás vizsgálata
   Teszteljük az egyszerű adattípusok műveleteit értékadó és kiíró utasításítások segítségével!
   Pl.

```
Sub Valami()
Dim i As Byte
i = 2 ^ 5          '5 helyett pl. 8 kitevővel (Overflow) futási hiba
MsgBox i
End Sub
```

4. Az iterációs utasítások gyakorlása
Írjuk ki az egész számokat 1-től 10-ig (a Debug.Print segítségével) az összes különböző ciklusszervezéssel (6 db: While, For és négyféle Do)!
Pl.

```
Sub Ciklus()
Dim i As Integer
i = 1
Do
    Debug.Print i
    i = i + 1
Loop Until i > 10
End Sub
```

5. Programozási feladatok
Készítsünk Visual Basic szubrutinokat az alábbi feladatokra! A bemenő adatokat az InputBox függvénnyel kérjük be, az eredményeket a MsgBox függvénnyel írjuk ki! Feltehető, hogy a felhasználó jó adatot ad meg!
Pl.

```
Sub Negyzet()
Dim i As Integer
i = InputBox("Adj egy egész számot!")
MsgBox "A szám négyzete: " & i ^ 2
End Sub
```

6. Kérjük be egy kör sugarát, és írjuk ki a kör területét és kerületét! (*Tipp:* A bekérést, a számolást, és a kiírást egymás után (szekvencia) hajtsuk végre!)

7. Kérjünk be két egész számot, és írjuk ki a nagyobbik szám értékét! (*Tipp:* A bekérés után a megfelelő eredmény kiírása szelekció segítségével történhet.)
8. Adott három szám esetén mondjuk meg azt, hogy lehet-e a három szám egy síkbeli háromszög három oldalának a hossza vagy sem! (*Tipp:* Ha bármelyik két oldal hosszának összege nagyobb, mint a harmadik oldal hossza, akkor „lehet”, egyébként „nem”.)
9. Adott két kör a síkon a középpontjának koordinátáival és a sugarával. Mondjuk meg azt, hogy van-e a köröknek közös pontja vagy sem! (*Tipp:* Ha a középpontok (Pitagorasz-tétellel kiszámolt) távolsága nagyobb, mint a két sugár összege, akkor „nincs”, egyébként „van”.)
10. Oldjuk meg az $ax^2 + bx + c = 0$ másodfokú egyenletet a valós számok körében, ahol az *a, b,* és *c* együtthatók értékei valós számok! (*Tipp:* A feladat teljes megoldása kezeli az elsőfokú és a másodfokú eseteket (0, 1, illetve 2 db valós gyök) is!)
11. Egy sztringben egy olyan név szerepel, amely két részből áll (családnév, keresztnév), a két részt pontosan egy szóköz választja el. Cseréljük fel a név két részét! (*Tipp:* Használjuk az InStr, Left és Right függvényeket!)
Pl. ”Makk Marci” → ”Marci Makk”
12. állítsunk elő adott karakterből álló, adott hosszúságú sztringet! (*Tipp:* Egy For ciklus segítségével egy kezdetben üres sztringhez fűzzük hozzá a megadott karaktert!)
Pl. ”*”, 3 → ”***”
13. Soroljunk be egy adott, legalább két karakterből álló sztringet az alábbi 4 osztály valamelyikébe! Egy sztring

    - *növekvő,* ha jelei (szigorúan) növekvő sorrendben vannak;
    - *csökkenő,* ha jelei (szigorúan) csökkenő sorrendben vannak;
    - *konstans,* ha jelei azonosak;
    - *egyéb*, minden más esetben.

    (*Tipp:* Az első három esethez vegyünk fel egy-egy logikai változót, és állítsuk őket igaz (True) kezdőértékre, majd egy ciklussal vizsgáljuk meg a sztring szomszédos karaktereit (először az elsőt és a másodikat, azután a másodikat és a harmadikat, ..., legvégül az utolsó előttit és az utolsót)! A vizsgálat állítsa hamisra (False) azokat a logikai változókat, amelyek nem teljesülnek (pl. ha a két karakter különböző, akkor a sztring jelei

már nem lehetnek azonosak)! A ciklus befejeztével írjuk ki a besorolás eredményét a logikai változók értéke alapján (pl. ha a három logikai változó egyike sem igaz, akkor a sztring az „egyéb” kategóriába tartozik)!

14. Adott *n* pozitív egész számra írjuk ki a *Fibonacci-sorozat* első *n* elemét! A képzés módszere: az első két elem értéke 1, ezután minden következő az előző kettő összege (Pl. *n* = 6: 1, 1, 2, 3, 5, 8). (*Tipp:* Az *n*<=2 esetek külön kezelendők, egyébként meg használjunk három változót és egy ciklust a feladat megoldására!)

15. Határozzuk meg két pozitív egész szám legkisebb közös többszörösét! (*Tipp:* Jelölje a kisebbik számot *a,* a másikat *b*. Vizsgáljuk *a* többszöröseit (*a,* 2**a,* 3**a,* ...) egészen addig, amíg *b* egy többszörösét nem kapjuk. Ilyen szám előbb utóbb lesz, ha más nem, akkor *a***b*. Egy *c* szám *b* többszöröse, ha *b* megvan benne maradék nélkül, azaz *c* Mod *b*=0.)

16. Számítsuk ki két adott, egy napon belüli időpont között eltelt időt! Az időpont óra, perc, másodperc értékekkel adottak, az eredményt is így adjuk meg! (*Tipp:* Alakítsuk „napon belüli” másodperccé a két időpontot, vegyük ezek különbségét, majd alakítsuk óra, perc, másodperc alakra!)

17. Döntsük el három egész számról azt, hogy Pitagoraszi számhármast alkotnak-e vagy sem! Ha a három szám éppen egy síkbeli, derékszögű háromszög három oldalának a hossza, akkor ilyen számokról van szó, egyébként nem. (*Tipp:* Ha valamelyik két oldal négyzetének összege éppen a harmadik oldal négyzetét adja, akkor „igen”, egyébként „nem”.)

18. Adott két zárt intervallum a valós számegyenesen. Mondjuk meg, hogy van-e közös pontjuk vagy sem! (*Tipp:* Ha valamelyik intervallum bal széle bent van a másik intervallumban, akkor „van”, egyébként „nincs”.)

19. Határozzuk meg az *X* és *Y* koordinátáját egy polárkoordinátákkal adott *P*(*r*, φ) síkbeli pontnak! A φ szög fokban adott. (*Tipp:* A Sin és Cos függvények használatához előbb számoljuk át radiánba a fokban adott szöget!)

20. Adott *n* és *k* pozitív egész számokra határozzuk meg a binomiális együttható értékét! (*Tipp:* A képletet (*n*!/(*k*!*(*n* - *k*)!)) ne a számláló és a nevező kiszámolása utáni osztással számoljuk ki (a nagy számok miatti pontatlanság/túlcsordulás miatt), hanem előbb egyszerűsítsük a képletet *k*!-sal, majd a kapott törtet bontsuk *n* - *k* db tört szorzatára, és ezt számoljuk ki!)

3. LECKE

Összetett adattípusok

1.1.5. Összetett adattípusok

Az eddig megismert adattípusokkal ellentétben, ahol egy változóban csak egy adatot tárolhatunk egy időben, az összetett adattípusok segítségével több adat együttes kezelésére is lehetőségünk nyílik.

1.1.5.1. Tömbök

A tömb egy általánosan és gyakran használt eszköz a szoftverfejlesztők, programozók körében, mivel a tömbökkel kényelmesen kezelhetünk több, azonos típusú adatot. Az adatok elérésére, vagyis a tömbök elemeinek kiválasztására sorszámokat, más néven indexeket használunk.

Ha az elemeket egy sorszámmal azonosítjuk, akkor egydimenziós tömbről, ha két sorszámmal, akkor kétdimenziós tömbről, ha *n* db sorszámmal azonosítjuk, akkor *n* dimenziós tömbről beszélünk. Egy tömb egy elemének kiválasztásához tehát pontosan annyi index szükséges, ahány dimenziós az adott tömb.

Ahogy az egyszerű adattípusok esetén is megkülönböztettük az adatot (pl. 3) az őt tároló változótól (pl. i) és annak típusától (pl. Integer), itt is kitérünk ezek különbözőségére. A tömb elnevezés ugyanis a rövidsége miatt mind a három esetben használatos, így a tömb lehet:

- *Tömbadat:* amely egydimenziós tömb esetén egy adatsornak, vagy matematikai fogalommal egy vektornak, kétdimenziós tömb esetén egy adattáblázatnak, vagy mátrixnak felel meg.
- *Tömb adattípus:* amely definiálja a tömb dimenzióit és a tömb elemeinek típusát.
- *Tömbváltozó:* amelyben az adatok tárolódnak. Egy elem kiválasztásával, azaz a tömbváltozó indexelésével egy, a tömb elemtípusával megegyező típusú változót hivatkozunk.

Egy tömböt a használata előtt deklarálni kell, azaz definiálnunk kell a tömb elemeinek típusát és a tömb dimenzióit.

A deklarálás (egyszerűsített) szintaktikája:

Dim *varname*[([*subscripts*])] [As *type*] [,...]

Az indexek (*subscripts*) megadásának szintaktikája:

[*lower* To] *upper* [,[*lower* To] *upper*] ,...

Pl.

```
Dim a(10) As Integer                'Egydimenziós tömb
Dim b(1 To 5, -3 To 2) As Double    'Kétdimenziós tömb
```

Vegyük észre, hogy az egyes dimenziók alsó indexhatára elhagyható. Az explicit módon nem definiált alsó indexhatárok értékeit az Option Base (modulszintű) utasítás határozza meg.

Az utasítás szintaktikája:

Option Base {0|1}

Értelemszerűen a megadott érték (0 vagy 1) lesz a tömbök alsó indexhatára. Alapértelmezésben az Option Base 0 van érvényben, azaz a tömbelemek alsó indexhatára 0.

Megjegyzés: Noha a VB-ben egy tömb dimenzióinak maximális száma 60 lehet, azaz használhatunk három, négy, ..., 60 dimenziós tömböket, a gyakorlatban az egy- és kétdimenziós tömbök a leggyakoribbak.

Az előzőekben ismertetett deklarálás ún. *statikus* tömböket deklarál, amelyek mérete fordítási időben meghatározódik, és futási időben már nem változtatható. A VB azonban megengedi az ún. *dinamikus* tömbök használatát is, amelyek mérete a program futása során megváltoztatható. Ezeket a tömböket is a Dim utasítással deklaráljuk, de az első deklarálásnál csak a tömbelemek típusát adjuk meg, a dimenziókat nem. A tömb átméretezése ezután a ReDim utasítással történik, amelyben előírhatjuk, hogy a már definiált értékű tömbelemek értékeit megőrizzük-e (Preserve) vagy sem.

Ha egy tömb méretét kisebbre állítjuk, akkor az „eltűnt” tömbelemek értékeit elveszítjük, míg egy tömb méretének növelésekor „keletkező” új elemek (a deklarált változókhoz hasonlóan) numerikus elemtípusú tömb esetén 0, a logikai elemek esetén False, míg String típus esetén az üres sztring kezdőértéket kapják. Ilyen értékeket kapnak az új tömb elemei akkor is, ha a Preserve kulcsszót elhagyjuk.

Szintaktika:

ReDim [Preserve] *varname*(*subscripts*) [As *type*] [,...]

Pl.

```
Dim a() As Integer          'Dinamikus tömb egészekből
ReDim a(5)                  'A tömb legyen 5 elemű
For i = 1 To 5              'A tömb használata
    a(i) = i
Next
ReDim Preserve a(10)        'A tömb legyen 10 elemű
                            'Az első 5 elem értéke megmarad, a többi 0 lesz
```

Megjegyzés

- A példában feltettük, hogy a modulban az Option Base 1 utasítás van érvényben.
- Egy dinamikus tömb többször is átméretezhető, de az elemek típusa nem változtatható meg.
- A dinamikus tömbök a ReDim utasítással közvetlenül (a Dim utasítás nélkül) is deklarálhatók, ekkor az elemek típusát is itt adjuk meg (pl. ReDim a(5) As Integer).

Az, hogy statikus vagy dinamikus tömböket használunk-e egy feladat megoldása során, az attól függ, hogy ismerjük-e előre az adatok maximális számát vagy sem. Ha ismerjük, akkor statikus (az adott maximumra deklarált) tömböt használunk, és az adatok aktuális számát egy külön változóban tároljuk. Ebben az esetben csak akkor lesz helypazarló a megoldásunk, ha az adatok maximális száma (azaz a tömb által lefoglalt memória mérete) jóval nagyobb, mint az adatok aktuális száma.

Ha az adatok maximális számát nem tudjuk előre meghatározni, akkor a megoldást dinamikus tömbökkel végezzük. Ebben az esetben a tömböt mindig akkorára deklaráljuk, hogy az adataink éppen elférjenek benne, így a megoldás sose foglal le több memóriát, mint amennyi szükséges.

A VB az UBound (felső határ) és LBound (alsó határ) függvényekkel biztosítja a tömbök deklarált indexhatárainak lekérdezhetőségét.

Szintaktika:

UBound(*arrayname*[,*dimension*])

LBound(*arrayname*[,*dimension*])

arrayname A tömb azonosítója.

dimension A lekérdezendő dimenzió sorszáma (alapértelmezett értéke 1).

Pl.

Dim a(10) As Integer

Dim b(1 To 5, -3 To 2) As Double

UBound(a) → 10	UBound(b) → 5	UBound(b,2) → 2
LBound(a) → 0	LBound(b) → 1	LBound(b,2) → -3

Megjegyzés: A példában feltettük, hogy a modulban az Option Base 0 utasítás érvényes.

1.1.5.2. Rekordok

A tömbök (legyenek azok egy-, vagy többdimenziósak) *azonos típusú* adatok egy összességének használatát biztosították. Ha azonban *különböző típusú* adatok egy összességét szeretnénk együtt kezelni, akkor egy újabb típust, a rekord adattípust kell használnunk. A rekord adattípussal ugyanis több, tetszőlegeses számú és típusú adatot foglalhatunk egy logikai egységbe.

Ezt az adattípust külön kell definiálnunk, ugyanis meg kell adnunk a rekord felépítését, azaz a rekordot alkotó *mezők* azonosítóját és típusát.

A rekordtípus megadásának (egyszerűsített) szintaktikája:

Type *name*

elementname[([*subscripts*])] As *type*

elementname[([*subscripts*])] As *type*

...

End Type

name	A rekordtípus azonosítója.
elementname	Az adott mező azonosítója.
subscripts	Ha a mező tömb, akkor annak indexhatárai.
type	Az adott mező típusa.

Pl.

```
Type Adat
    Nev As String * 20      'Fix (20) hosszú sztring
    Jegyek(1 To 2) As Byte
End Type
```

A példa egy olyan rekordtípust (Adat) definiál, amelynek két mezője van. Az első mező (Nev) egy fix hosszú sztring, a második mező (Jegyek) egy 2 elemű, egydimenziós, Byte elemtípusú tömb.

Megjegyzés

- Az egyes mezők beljebb tagolása nem kötelező, csak az áttekinthetőséget segíti.
- A példában fix hosszú sztringtípust (String * 20) használtunk, de a változó hosszú String típust is használhattuk volna.
- A Type utasítás modulszintű utasítás.

Egy definiált rekordtípus ugyanúgy használható változók deklarálására, mint a VB többi adattípusa.

Pl.

```
Dim h As Adat, a(1 To 3) As Adat
```

A példa két változót deklarál. A h változóban egy rekord adatai tárolhatók, míg az a változó három rekord tárolására alkalmas.

A rekord mezőire való hivatkozás:

rekordváltozó.mezőnév

A rekordok egyes mezőire úgy hivatkozhatunk, hogy a rekordváltozó és a mezőnév közé egy pontot teszünk. Ezzel a hivatkozással egy, a mező típusával megegyező típusú változót hivatkozunk, amit ennek megfelelően használhatunk. A rekordokra nem definiáltak műveletek, de az azonos típusú rekordváltozók között megengedett az értékadás (lásd 1.1.5.3. fejezet).

Pl.

```
h.Nev = "Halász Helga"
a(1).Jegyek(1) = 5
```

A mezőkre való hivatkozás a With utasítás segítségével rövidíthető. Ekkor a rekordváltozót elegendő egyszer megadni (a With kulcsszó után), a mezőnevek előtt elhagyhatjuk, de a pontot ki kell tennünk.

Pl.

```
With a(2)
    .Nev = "Vadász Viktória": .Jegyek(1) = 3: .Jegyek(2) = 4
End With
```

A példa a korábban deklarált a tömb második elemében tárolt rekord egyes mezőinek ad értéket.

Megjegyzés

- A With utasítás objektumokra is használható. Az objektumokról az 1.1.6., illetve 1.3.1. fejezetekben lesz szó.
- A With blokkban szereplő utasítások beljebb tagolása nem kötelező, csak az áttekinthetőséget segíti.

• A With utasítások egymásba ágyazhatók.

• A rekord szó a szakirodalomban általánosan használt (ezért a jegyzetben is ezt használtuk), de a VB súgója a rekord adattípusra a *felhasználó által definiált adattípus* (user-defined data type) elnevezést használja.

• Az összetett adattípusú (tömb, rekord) deklarált változók is kapnak kezdőértéket, amit a tömbelemek, illetve az egyes mezők típusa határoz meg (lásd 1.1.2.1. fejezet).

1.1.5.3. Mintafeladat

A tömbök és rekordok használatát az alábbi feladat segítségével szemléltetjük.

Feladat: Egy hallgatói nyilvántartásban ismerjük a hallgatók nevét és adott számú érdemjegyét. Minden hallgatónak ugyanannyi érdemjegye van. Az adatokat egy rekordokból álló egydimenziós tömbben tároljuk. Készítsünk névsor szerint rendezett listát, amelyen a hallgatók érdemjegyei is szerepelnek!

Az alábbi megoldásban először egy konstanst deklarálunk, amely definiálja a hallgatók érdemjegyeinek számát. Az olyan értékeket, amelyek a programban több helyen is szerepelnek, célszerű névvel ellátni, azaz külön deklarálni, mert akkor az érték esetleges módosításához elegendő egy helyen (a deklarációban) módosítanunk.

A konstansok deklarálásának szintaktikája:

[Public|Private] Const *constname* [As *type*] = *expression*

Public	A konstans hivatkozható lesz minden modulból.
Private	A konstans csak abban a modulban hivatkozható, ahol deklarálták (ez az alapértelmezés).
constname	A konstans azonosítója.
type	A konstans típusa (opcionális).
expression	A konstans értékét megadó kifejezés.

Pl.

```
Const db = 2
```

Megjegyzés

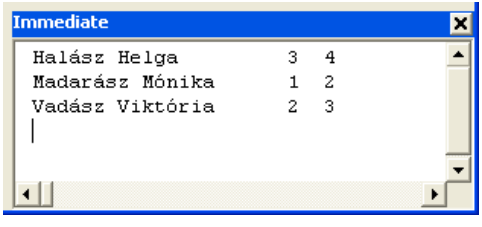
• Az utasítás modul szinten is, és szubrutinon belül is használható. Az utóbbi esetben a konstans csak az adott szubrutinon belül hivatkozható, és a láthatóságát (hivatkozhatóságát) nem lehet a Public és Private kulcsszavakkal módosítani.

• A konstans értékét megadó kifejezésben konstans értékek, deklarált (névvel azonosított) konstansok, valamint műveletek állhatnak.

• A típust (*type*) általában elhagyjuk (a példában is így tettünk), ekkor a konstans típusa a kifejezés értékének legmegfelelőbb típus lesz (esetünkben Byte).

A konstans definiálása modul szinten történt, hiszen az egész modulban szeretnénk használni. A rekordtípus (amely leírja egy hallgató adatait) deklarációja viszont csak modul szinten (azaz a modul elején) történhet. A nevekhez fix hosszú sztring típust használtunk, amely nem engedi meg, hogy a nevek a megadott hossznál (a példában 20) hosszabbak legyenek.

Az adatok kiírását egy szubrutin (Kiiras) végzi, amely egy Adat rekordokból álló tömböt kap paraméterként (a), amit a paraméter után szereplő (üres) zárójelpár jelez. A kapott tömböt egydimenziós tömbként kezeljük, és feltesszük, hogy a tömb minden eleme definiált, amit a Kiiras szubrutin hívója (RendezesTeszt) biztosít, ezért a tömbelemeken végiglépdelő ciklus szervezését a tömb lekérdezett (LBound, UBound) indexhatárai alapján végezhetjük. Az adatkiírások után akkor emelünk sort (egy paraméter nélküli Debug.Print hívással), ha az adott hallgató összes adatát (név, érdemjegyek) kiírtuk, így minden hallgató külön sorba kerül. Az eredménylista az 1.3. ábrán látható. Vegyük észre a nevek jobbról szóközökkel való feltöltöttségét (ez a fix hosszú sztringek értékadásakor automatikusan megtörténik).



1.3. ábra. A mintafeladat eredménylistája

Az adatok rendezését a Rendezes szubrutin végzi, amely (a Kiiras szubrutinhoz hasonlóan) szintén megkapja a rendezendő adatok tömbjét. Az adatkiírás nem változtatott a paraméterként kapott adatokon, de a rendező szubrutinnak éppen ez a dolga, nevezetesen hogy névsor szerinti sorrendbe tegye a tömbben lévő adatokat. Mivel a tömb átadására deklarált (a) paraméter cím szerinti átadású (ByRef az alapértelmezés, és tömböket nem is adhatunk át érték szerint, lásd 1.1.4.1. fejezet), ezért a tömb elemein végzett változtatásokat a hívó is „érzékelni fogja".

A rendezést egy, a minimum kiválasztás elvén alapuló rendező algoritmus segítségével valósítottuk meg. A külső ciklus minden egyes lépésében megkeressük (a belső ciklussal) a még rendezetlen tömbrész legkisebb elemét (annak indexét tárolja a k változó), és ha ez az elem nincs a helyén (az i-edik helyen), akkor odatesszük. A rekordok között értelmezett az értékadás, ezért két elem (i-edik és k-adik) felcserélése elvégezhető három értékadással (nevezetesen megjegyezzük az egyiket, oda áttesszük a másikat, majd a másikba betesszük a megjegyzett egyiket). Vegyük észre, hogy a cseréhez használt változó (cs) a tömbelemekkel megegyező (Adat) típusú.

A minimumkeresés úgy történik, hogy megjegyezzük az adott tömbrész első elemének indexét, és megvizsgáljuk a többi elemet. Ha kisebbet találunk (a rekordok Nev mezője szerinti összehasonlítás szerint, hiszen névsor szerinti sorrendet szeretnénk előállítani), akkor annak indexét jegyezzük meg (hiszen most már ő az aktuálisan legkisebb). A rendezetlen tömbrész kezdetben (i=1 esetén) az egész tömb, utána (i=2 esetén) csak a második elemtől az utolsó elemig (hiszen az első, vagyis a legkisebb elem már a helyére, a tömb elejére került), és így tovább, legvégül (i=UBound(a)-1 esetén) már csak a tömb két utolsó elemét tartalmazza.

A Rendezes szubrutint a RendezesTeszt szubrutin hívja meg azután, hogy kezdőértékekkel látta el az adattárolásra használt tömbváltozót. A tömb azonosítására mindkét helyen ugyanazt az azonosítót (a) használtuk, de használhattunk volna különböző azonosítókat is. A harmadik személy (akinek adatait a tömb második elemében tároljuk, hiszen a modulban alapértelmezésben az Option Base 0 érvényes) adatainak megadásakor a rekordmezőkre a With utasítás segítségével hivatkoztunk, hogy ezt is bemutassuk.

Megjegyzés

• A megoldás csak a VB nyelv lehetőségeit használta, így szükség volt egy rendező algoritmus programozására. Ha azonban kihasználtuk volna az Excel VBA lehetőségeit, akkor a hallgatók adatait egy Excel munkalapon

tárolva egyetlen metódushívással „elintézhető” lett volna az adatok rendezése. Az Excel objektumairól és használatukról az 1.3.1. fejezetben lesz szó.

• Igen sokféle rendező algoritmus ismert, amelyekből ez egyik legegyszerűbbet választottuk ki. A témakör iránt érdeklődőknek a szakirodalmat (pl. [3], [4]) ajánljuk.

```
'Egy hallgató érdemjegyeinek száma
Const db = 2
'Egy hallgató adatait leíró rekordtípus
Type Adat
        Nev As String * 20 'Fix (20) hosszú sztring
        Jegyek(1 To db) As Byte
End Type
'Adat típusú elemekből álló tömb kiírása
Sub Kiiras(a() As Adat)
Dim i As Integer, j As Integer
For i = LBound(a) To UBound(a)
        Debug.Print a(i).Nev;
        For j = 1 To db
                Debug.Print a(i).Jegyek(j);
        Next
        Debug.Print
Next
End Sub
```

```
'Adat típusú elemekből álló tömb rendezése név szerint
Sub Rendezes(a() As Adat)
Dim cs As Adat, i As Integer, j As Integer, k As Integer
For i = LBound(a) To UBound(a) - 1
        k = i
        For j = i + 1 To UBound(a)
                If a(j).Nev < a(k).Nev Then
                        k = j
                End If
        Next
        If k > i Then
                cs = a(i): a(i) = a(k): a(k) = cs
        End If
Next
End Sub
'A rendezés tesztjéhez ez indítandó
Sub RendezesTeszt()
Dim a(2) As Adat
a(0).Nev = "Madarász Mónika": a(0).Jegyek(1) = 1: a(0).Jegyek(2) = 2
a(1).Nev = "Vadász Viktória": a(1).Jegyek(1) = 2: a(1).Jegyek(2) = 3
With a(2)
        .Nev = "Halász Helga": .Jegyek(1) = 3: .Jegyek(2) = 4
```

```
End With
Call Rendezes(a)
Call Kiiras(a)
End Sub
```

1.1.6. Objektumok, objektumtípusok

Az objektum szót már használtuk (pl. a Debug objektum kapcsán), de még nem beszéltünk arról, hogy pontosan mit jelent, mit takar az objektum elnevezés az informatikában.

Objektumon adatok és a rajtuk értelmezett tevékenységek egy logikailag összefüggő egységét értjük. Az objektum (hasonlóan, mint a tömb) egy gyakran és általánosan használt szó, ami jelenthet adatot (egy konkrét objektumpéldányt), változót (ami tárolja és elérhetővé teszi az objektum adatait és tevékenységeit), valamint típust is (az objektum típusát, amely definiálja az objektum adatainak típusát és az objektum tevékenységeit).

Az objektumok tevékenységeit definiáló szubrutinokat *metódusoknak* (methods), az objektumok típusát *osztályoknak* (class) nevezzük. Az objektumok adataihoz általában *tulajdonságok* (properties) segítségével férhetünk hozzá, amelyek védik az adatok helyességét (lásd 1.2.2.3. fejezet).

Az objektumok azon metódusait, amelyek valamilyen *esemény* bekövetkezésekor (pl. egy nyomógomb esetén a nyomógomb feletti kattintás) aktivizálódnak, *eseménykezelőknek* nevezzük. Az események általában valamilyen felhasználói interakció eredményeként következnek be, de más futó folyamatok is kiválthatják az egyes események bekövetkezését (pl. meghívunk egy eseménykezelőt).

Megjegyzés: A VBA megengedi az osztályok definiálását, de mi csak a „készen kapott” osztályokat fogjuk használni.

Önellenőrzés

1. Az alábbi állítások közül melyek igazak az összetett adattípusokkal kapcsolatosan?
   - ☐ A tömbök indexeinek alsó határa 0 vagy 1.
   - ☐ Egy tömb egy elemének kiválasztásához pontosan annyi darab index szükséges, mint ahány dimenziós az adott tömb.
   - ☐ A dinamikus tömbök elemtípusa a ReDim utasítással megváltoztatható.
   - ☐ Egy rekord mezői lehetnek tömbök is és rekordok is.
   - ☐ Egy rekordváltozó deklarálása a Type kulcsszóval kezdődik.
   - ☐ Azonos típusú rekordváltozók között megengedett az értékadás.
   - ☐ A With utasítással a rekordváltozóra való hivatkozás rövidíthető.
2. Az alábbi állítások közül melyek igazak az objektumokkal kapcsolatosan?
   - ☐ Objektumon adatok egy halmazát értjük.
   - ☐ A szubrutinok egyben metódusok is.
   - ☐ A metódusok egyben szubrutinok is.
   - ☐ Az eseménykezelők egyben metódusok is.
   - ☐ A metódusok egyben eseménykezelők is.
   - ☐ Az objektumpéldányokat osztályoknak nevezzük.
   - ☐ Az osztály egyben típus is.
3. Programozási feladatok
   Készítsünk Visual Basic szubrutinokat a fejezetben található feladatokra! A megoldások elkészítésére az alábbi lehetőségek közül választhatunk, célszerű mindkét megoldást kipróbálni, elvégezni!
   A megoldás megtekintéséhez kattintson ide!

4. **Darabszám:** Határozzuk meg egy adott [*a, b*] intervallumba ($a < b$ pozitív egészek) eső, adott *c* számmal osztható számok darabszámát! (*Tipp:* Egy darabszám változót állítsunk 0-ra, majd egy ciklussal vizsgáljuk meg az [*a, b*] intervallumba eső egész számokat! Ha a vizsgált szám osztható *c*-vel, akkor növeljünk meg eggyel a darabszám változó értékét!)
5. **Összeg:** Oldjuk meg az előző feladatot úgy, hogy a feltételt teljesítő számok összegét is meghatározzuk! (*Tipp:* Az összeget tartalmazó változót is állítsuk 0-ra, majd amikor a darabszámot eggyel növeljük, akkor az összeget gyűjtő változót növeljük meg a vizsgált számmal!)
6. **Minimum:** Adott egy nem üres sztring. Határozzuk meg a sztring legkisebb ASCII kódú karakterét! (*Tipp:* Jegyezzük meg a sztring első karakterét mint eredményt, majd vizsgáljuk meg a sztring többi karakterét! Ha valamelyik vizsgált karakter ASCII kódja kisebb, mint az addigi eredmény ASCII kódja, akkor módosítsuk az eredményt, azaz jegyezzük meg a vizsgált karaktert eredményként!)
7. **Maximum:** Adott egy nem üres sztring. Határozzuk meg a sztring legnagyobb ASCII kódú karakterét! (*Tipp:* Az előző minimumkereséshez hasonlóan végezzük, csak most akkor módosítsuk az eredményt, ha nagyobbat (nagyobb ASCII kódú karaktert) találunk!)
8. **Átlag:** Adott *N* db szám, mint input adat. Kérjük be őket, és írjuk ki az átlagnál nagyobb adatokat! Az adatok száma is input adat, értéke legfeljebb 10. (*Tipp:* Az adatok tárolására használjunk tömböt! Előbb határozzuk meg az adatok összegét (lásd Összeg), azután egy *N*-nel való osztással az átlagot, majd egy ciklussal írjuk ki az átlagnál nagyobb elemeket!)
9. Döntsük el egy pozitív egész *n* számról, hogy prímszám-e vagy sem! Egy szám prím, ha 1-en és önmagán kívül nincs más osztója. (*Tipp:* Az $n<=2$ esetek külön kezelendők, (1 nem prím, 2 prím) egyébként meg, ha a számnak nincs osztója a [2, Int(Sqr(*n*))] intervallumban, akkor a szám „prím”, egyébként „nem”! A feladat megoldását a 4. lecke tartalmazza.)
10. Az előző feladat segítségével írjuk ki egy adott [*a, b*] intervallumba ($a < b$ pozitív egészek) eső összes prímszámot! (*Tipp:* Egy For ciklus segítségével lépkedjünk végig az [*a, b*] intervallumba eső egész számokon, az előző algoritmussal döntsük el azt, hogy az adott szám prímszám-e vagy sem, és ha az, akkor írjuk ki az adott prímszámot!)
11. Adott egy sztring, amelyben egy legalább kétjegyű, pozitív egész szám van. A szám nagy is lehet, akár 100 számjegyből álló is (ezért is van sztringben megadva). Mondjuk meg, hogy a szám osztható-e 2-vel,

3-mal, 4-gyel, 5-tel! (*Tipp:* A szám (a nagysága miatt) számként nem használható (azaz nem végezhetünk vele numerikus műveletet), így a megoldáshoz a szám megfelelő számjegyeit kell megvizsgálni! A 2-vel való oszthatósághoz az kell, hogy az utolsó számjegye páros legyen, a 3-mal való oszthatósághoz az, hogy a szám számjegyeinek összege osztható legyen 3-mal, a 4-gyel való oszthatósághoz az, hogy a két utolsó számjegyből álló szám osztható legyen 4-gyel, az 5-tel való oszthatósághoz az utolsó számjegy 0 vagy 5 volta szükséges. Egy *c* számjegykarakter számértékét az Asc(*c*)-48 kifejezés adja.)

12. Generáljunk egy szabályos 5-ös lottószelvényt, azaz 5 db különböző egész számot az [1, 90] intervallumból! (*Tipp:* A lottószámokat egy (5 elemű) tömbbe tegyük! Annak eldöntésére, hogy egy számot ismételten generáltuk-e vagy sem, használjunk keresőciklust (lásd 1.2.2.8. fejezet)!)

13. Adott egy pozitív egész szám, mint egy euróban kifizetendő pénzösszeg. Fizessük ki a legkevesebb címlet felhasználásával! A felhasználható címletek: 500, 200, 100, 50, 20, 10, 5, 2, 1.
Pl. 1243 → 500: 2db, 200: 1db, 20: 2 db, 2: 1 db, 1: 1 db
(*Tipp:* Az egyes címleteket tegyük egy (9 elemű) tömbbe nagyság szerint csökkenő sorrendben, és az összeg kifizetésekor ilyen sorrendben vegyük az egyes címleteket, azaz először a legnagyobb címlettel (500) kezdjünk, és a legkisebbel (1) fejezzük be! Egy adott címlettel való kifizetésnél egész osztást végezzünk, a hányados az adott címlet darabszámát adja, míg a maradék a további címletekkel kifizetendő összeget!)

14. Egy kártyajátékot 52 lapos kártyával játszanak. Készítsünk véletlen leosztást *N* (*N*<=4, input adat) játékosnak úgy, hogy minden játékos *M* (*M*<=13, input adat) db lapot kap! A lapokat a sorszámukkal azonosítsuk, egy lapot csak egyszer osszunk ki, az eredmény egy *N*×*M*-es mátrix legyen! (*Tipp:* Használjunk egy 52 elemű logikai tömböt az egyes lapok kiosztottságának adminisztrálására! Kezdetben minden elem legyen hamis (False), hiszen még nem osztottunk ki egyetlen lapot sem! A véletlenszerű osztást játékosonként, azon belül laponként haladva végezzük két, egymásba ágyazott For ciklussal! Egy adott játékos adott lapjának generálását egy hátultesztelő ciklussal végezzük, mert csak olyan lap generálható, amelyik még nincs kiosztva! Ha egy lapot kiosztunk (betesszük a lap sorszámát az eredménymátrixba), akkor a lap kiosztottságát állítsuk igazra (True)!)

4. LECKE

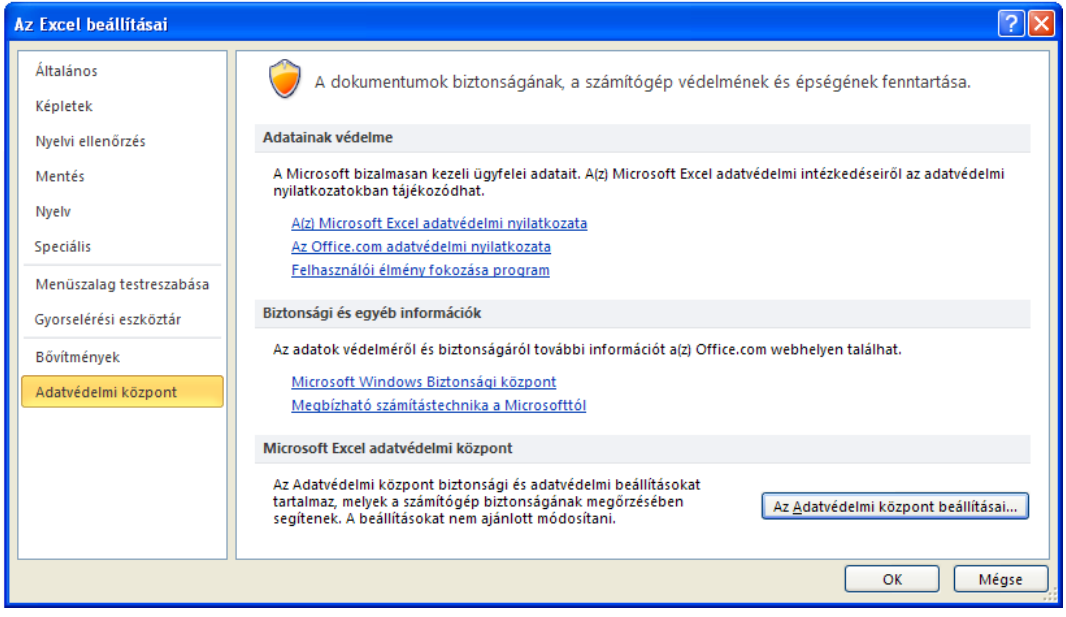
Az Excel VBA használata

1.2. Az Excel VBA használata

Ebben a fejezetben az Excel VBA fejlesztőkörnyezetről lesz szó. Elsősorban a forrásprogramok megadásához, azok futtatásához, a hibakereséshez kapcsolódó funkciókat részletezzük, de itt kapott helyet a vizuális formtervezés is.

1.2.1. Makrók

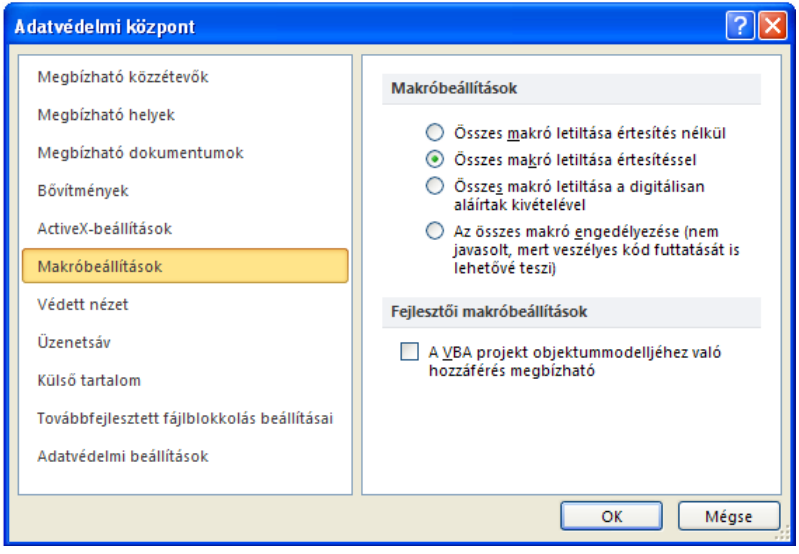
„A makró parancsok sorozata, amely ismétlődő feladatok végrehajtásának automatizálására használható.” – idézet az Excel súgójából.



1.4. ábra. Az adatvédelmi központ

A makró szó az adott MS Office (esetünkben az MS Excel) alkalmazás programfejlesztő környezetében létrehozott objektumok és VBA forráskódok általános, összefoglaló elnevezése. Az Excel 2010 a makrókat tartalmazó dokumentumokra a „Makróbarát Excel munkafüzet” elnevezést használja. A makrókat tartalmazó dokumentumokat ilyen fájlként mentsük (ezek kiterjesztése xlsm).

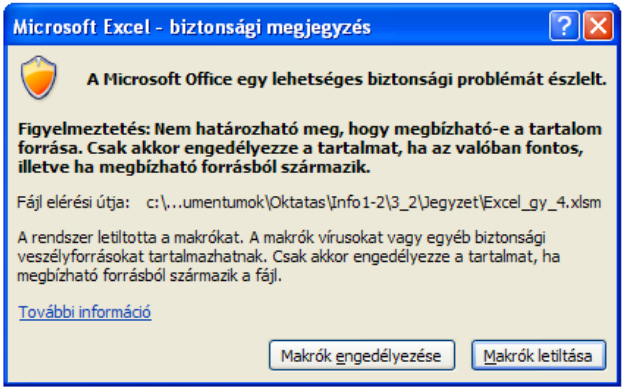
A makróvírusok elterjedésének megakadályozása érdekében az Excel bizonyos beállításai a makrókat tartalmazó dokumentumok kezelésére vonatkoznak. A Fájl, Beállítások, Adatvédelmi központ (lásd 1.4. ábra) ablakban a jobb alsó sarokban található „Az Adatvédelmi központ beállításai...” nyomógomb segítségével (többek között) megadhatók a makrókra vonatkozó beállítások is (lásd 1.5. ábra).



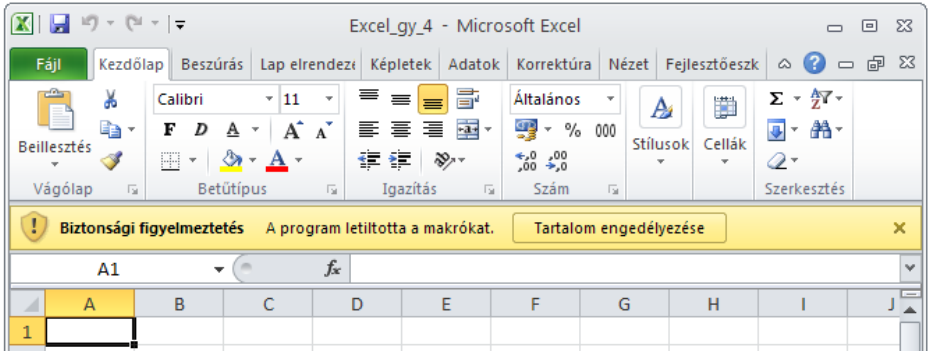
1.5. ábra. Az adatvédelmi központ makróbeállításai

Ahhoz, hogy a makrókat tartalmazó dokumentumok megnyitásakor eldönthessük, hogy engedélyezzük vagy letiltjuk az adott dokumentumban lévő makrókat, az „Összes makró letiltása értesítéssel” választógombot kell

bekapcsolni (lásd 1.5. ábra). Ebben az esetben egy makrókat tartalmazó dokumentum megnyitásakor üzenetet kapunk. Ha a Visual Basic Editor meg van nyitva, akkor az 1.6. ábrán szereplő párbeszédablak jelenik meg, egyébként meg az 1.7. ábrán látható üzenetsáv. Mindkét esetben engedélyezhetjük a dokumentumban lévő makrókat.



1.6. ábra. Makrók engedélyezése, illetve letiltása párbeszédablak



1.7. ábra. Makrók engedélyezése, illetve letiltása üzenetsáv

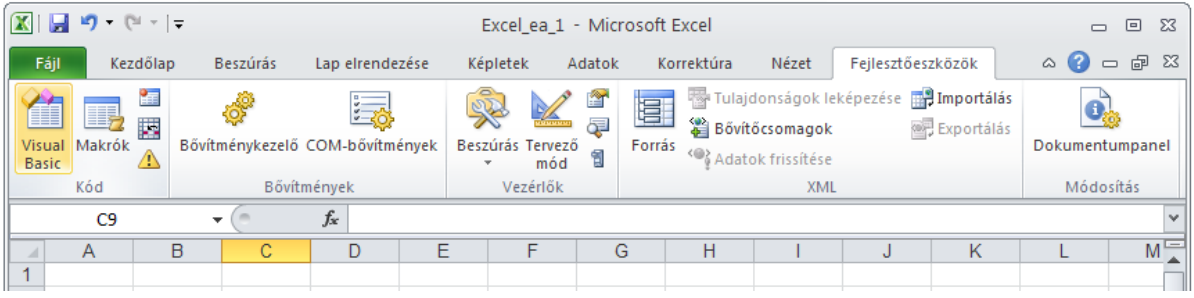
Az engedélyezés, illetve letiltás csak a makrók futtatására vonatkozik, így letiltott esetben is megnézhetjük, sőt akár módosíthatjuk is a makrókat. Az engedélyezés, illetve tiltás (adott dokumentumra vonatkozó) megváltoztatásához zárjuk be és nyissuk meg újra (a kívánt módon) az adott dokumentumot.

Megjegyzés

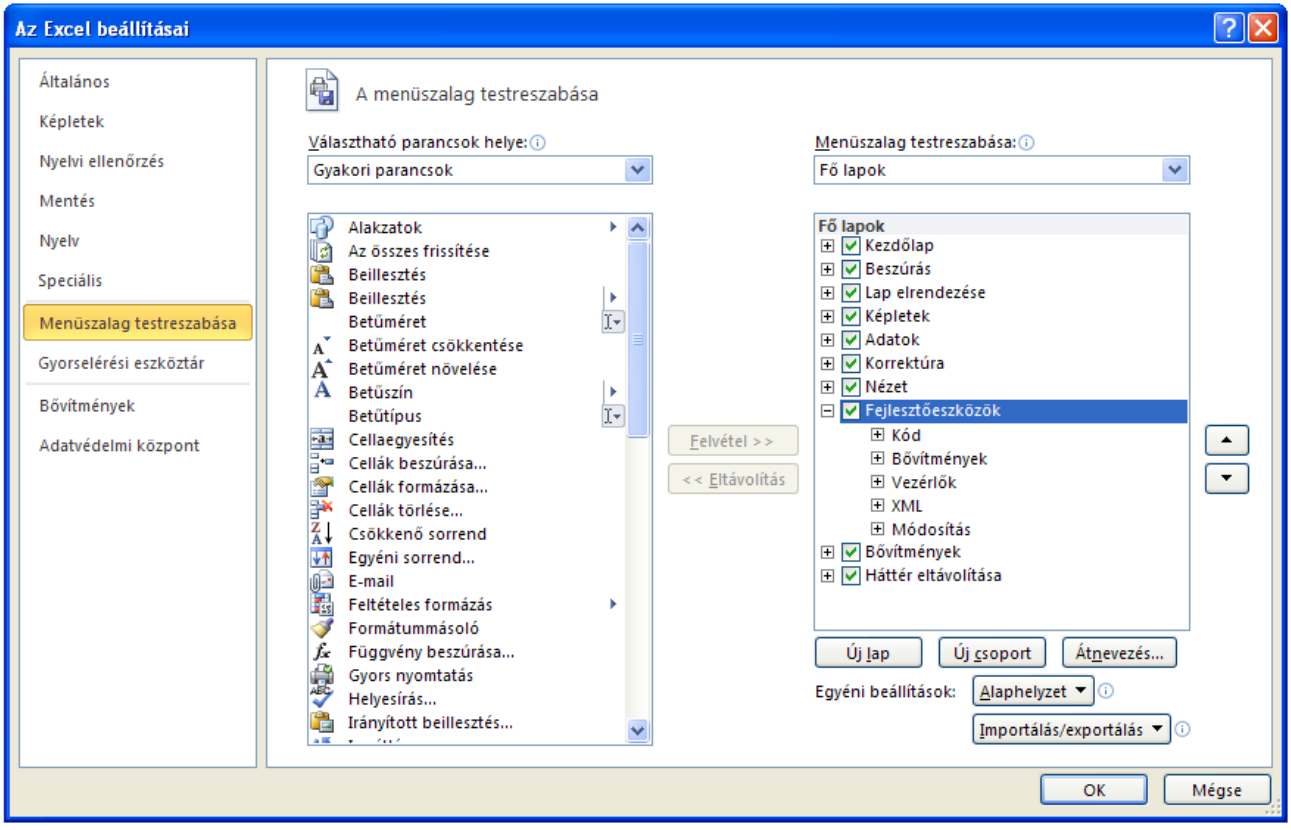
- Ha az 1.7. ábrán látható sárga színű üzenetsávban engedélyezzük a makrókat („Tartalom engedélyezése” gomb), akkor az adott számítógépen a dokumentum megbízható dokumentumként kezelődik a továbbiakban, ezért újabb megnyitásakor már nem kell külön engedélyezni a makrókat, egyébként (az üzenetsáv bezárásával) meghagyjuk a makrók letiltását.
- Egy újonnan létrehozott munkafüzetben készített makrók a beállításoktól függetlenül mindig futtathatók.

1.2.2. A Visual Basic Editor

Alapértelmezésben az Excel menüszalagján nem látható a Fejlesztőeszközök lap (lásd 1.8. ábra), amelyen a Visual Basic fejlesztőrendszer, a Visual Basic Editor elérhető, de a gyorsbillentyűjével (Alt+F11) ekkor is megnyitható. A Fejlesztőeszközök lap megjelenítéséhez a Fájl, Beállítások, Menüszalag testreszabása párbeszédablakban be kell kapcsolni a lap megjelenítését szabályozó jelölőnégyzetet (lásd 1.9. ábra).

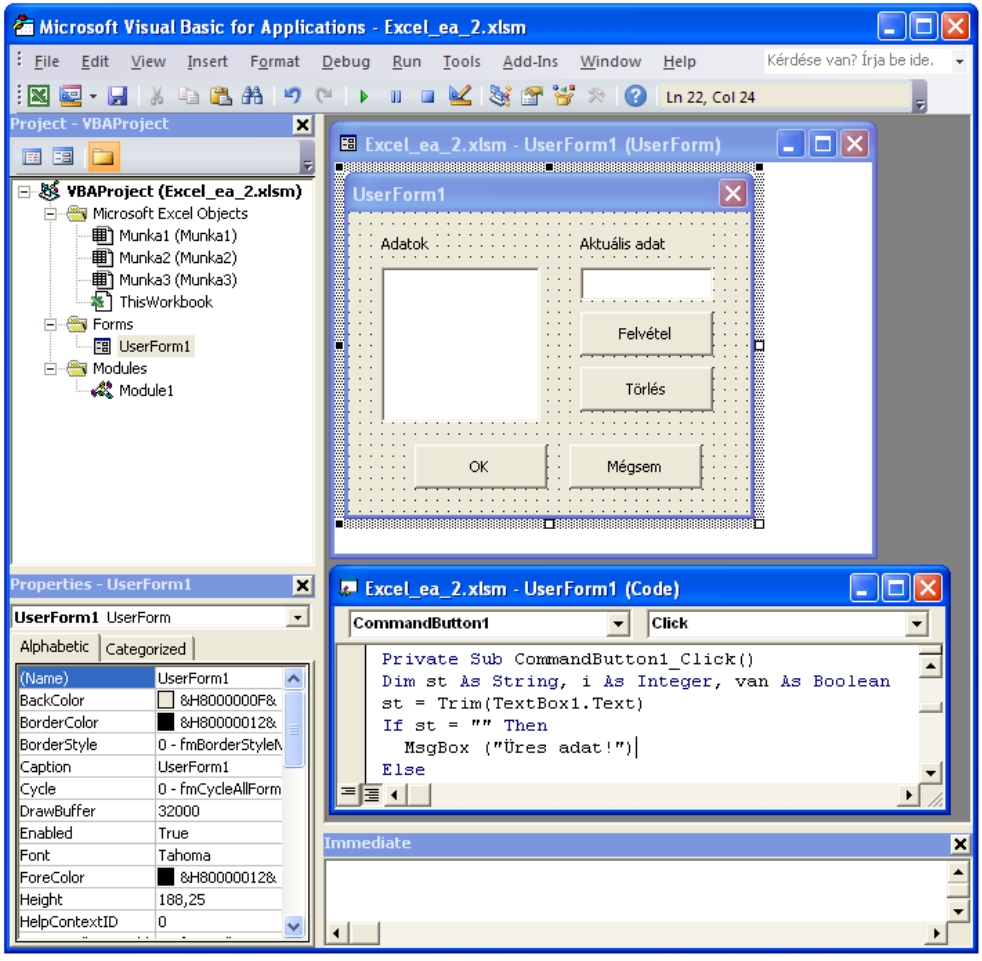


1.8. ábra. A menüszalag Fejlesztőeszközök lapja



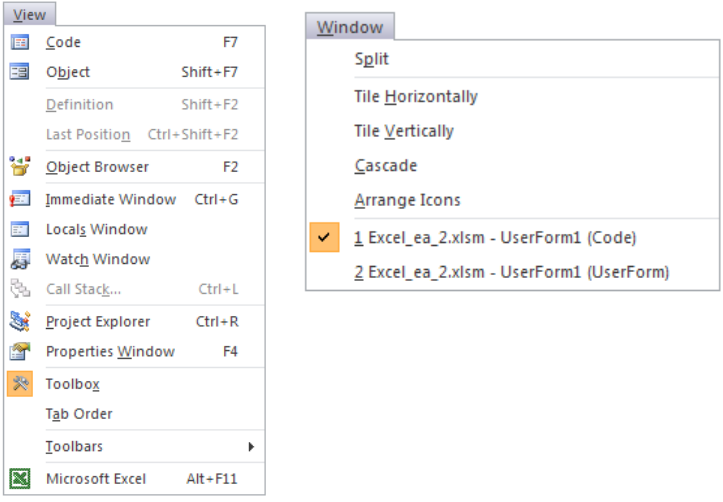
1.9. ábra. A menüszalag testreszabása

A Visual Basic Editor egy önálló ablakban jelenik meg, de az Excel bezárásával ez az ablak is bezáródik. A fejlesztőkörnyezet angol nyelvű (menü, súgó, hibaüzenetek, stb.), hiába magyar nyelvű az Excel, amihez tartozik.



1.10. ábra. A Visual Basic Editor

Bár az ablak elrendezése beállítható, egy tipikus elrendezés az 1.10. ábrán látható. Felül a menüsor, alatta az eszköztársor, balra fent a Project ablak, alatta a Properties ablak, a jobb oldali, szürke hátterű munkaterületen most két ablak látható. Az egyik a UserForm1 objektum tervezőablaka, a másik a UserForm1 objektumhoz tartozó forráskód szerkesztőablaka. A munkaterület alatt az Immediate ablak látható.



1.11. ábra. A Visual Basic Editor View és Window menüje

Valamennyi ablak szabadon megjeleníthető, méretezhető, más helyre tehető, vagy akár be is zárható. A View menü segítségével (lásd 1.11. ábra) bármikor újra megjeleníthető egy bezárt ablak. A gyakran használatos funkciók a helyi menü segítségével, vagy a hozzájuk tartozó gyorsbillentyűkkel is aktivizálhatók (pl. F7, Shift+F7). A helyi menü (a Windows-ban megszokott módon) az adott helyen, az egér jobb gombjával hozható be.

Megjegyzés

• A Visual Basic Editor-ban létrehozott makrókat nem kell külön menteni, ezek a munkafüzet mentésekor elmentődnek. A Visual Basic Editor mentés (Save) funkciója az egész munkafüzetet menti (csakúgy, mint az Excel-beli mentés funkció).

• Bizonyos ablakok (pl. Project, Properties, Immediate) rögzíthetők (dockable) a munkaterületen, sőt akár a Visual Basic Editor ablakán kívülre is mozgathatók. Az ablakelrendezés (mely ablakok hol és mekkora méretben jelennek meg) megőrződik, minden belépéskor a legutóbbi ablakelrendezésben jelenik meg a Visual Basic Editor.

• A nem rögzíthető ablakok a nyitáson és záráson kívül még minimalizálhatók (amikor is ikonként a munkaterület aljára kerülnek) és maximalizálhatók (ilyenkor a teljes munkaterületet kitöltik eltakarva egymást). A nyitott ablakok közötti váltás a kívánt ablakon való kattintás mellett, a Window menü segítségével is elvégezhető, amely a Windows alkalmazásokban szokásos funkciókkal (ablakok különféle elrendezése, a nyitott ablakok listája, stb.) rendelkezik (lásd 1.11. ábra).

A fontosabb ablakok szerepéről a következő alfejezetekben lesz szó.

1.2.2.1. Az Immediate ablak

Az Immediate ablak amellett, hogy a programok egyik output megjelenítő eszköze (lásd 1.1.2.6. fejezet), utasítások közvetlen végrehajtására is használható. A végrehajtani kívánt utasításokat egy sorba kell írni, több utasítás esetén az utasítások közé kettőspontot kell tenni (lásd 1.1.3.1. fejezet). A végrehajtást az Enter billentyű leütésével kérhetjük.

Az Immediate ablak tartalma szabadon szerkeszthető, így pl. a korábban végrehajtott sorok (esetleges módosítással) újra végrehajthatók, a felesleges sorok törölhetők, stb.

Pl.

```
i=2:j=3
Print i;"*";j;"=";i*j;tab(1);i;"^";j;"=";i^j
?"1-10-ig az egész számok":for i=1 to 10:?i;:next:?
```

Megjegyzés

- A Print és a ? a Debug objektum Print metódusát (Debug.Print) hivatkozza röviden.
- Nem minden VB utasítás hajtható végre az Immediate ablakban (pl. Dim utasítás).
- Természetesen itt is csak szintaktikailag helyes utasítások hajthatók végre. Az esetleges hibaüzenetet az Enter billentyű leütése után kapjuk meg (szemben a kódszerkesztő ablakban beírt utasításokkal, ahol már a szintaktikailag hibás sor elhagyásakor megkapjuk a megfelelő hibaüzenetet, ha az erre vonatkozó kapcsoló (Auto Syntax Check, lásd 1.2.2.4. fejezet 1.23. ábra) bekapcsolt állapotú.
- Az Immediate ablakban csak akkor lehet utasításokat végrehajtani, ha a munkafüzetre vonatkozó biztonsági beállítások engedélyezik a makrók futtatását (lásd 1.2.1. fejezet).

1.2.2.2. A Project ablak

Az MS Office VBA fejlesztőkörnyezetben a projektek foglalják egységbe az adott dokumentumot a hozzá tartozó makrókkal. A projekt az adott dokumentumfájlban tárolt objektumok áttekinthetőségét, kezelését hivatott szolgálni. A Project ablak a Visual Basic Editor View menüjének Project Explorer funkciójával (lásd 1.11. ábra) jeleníthető meg (lásd 1.10. ábra).

Egy Excel projektben az alábbi objektumok szerepelnek, illetve szerepelhetnek:

- Excel objektumok (Microsoft Excel Objects)
  - Az egyes munkalapok (Munka1, ...)
  - A munkafüzet (ThisWorkbook)
- Formok (Forms)
- Modulok (Modules)
- Osztálymodulok (Class Modules)

Az Excel objektumok csoport elemei az adott munkafüzethez igazodnak (pl. annyi munkalap objektum szerepel a projektben, ahány munkalapja van az adott munkafüzetnek). A munkalap objektumokkal az

egyes munkalapok, a ThisWorkbook objektummal a (makrókat tartalmazó) munkafüzet hivatkozható. Ezek az objektumok nem hozhatók létre, illetve nem törölhetők a Visual Basic Editor-ban, de a hozzájuk tartozó esetleges eseménykezelők (pl. a munkafüzet megnyitása, bezárása, egy munkalap aktivizálása, stb.) az objektumhoz tartozó modulban definiálhatók (lásd 1.3.2.4. fejezet).

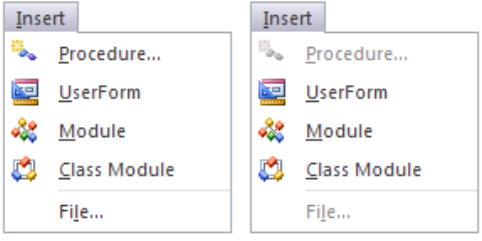
A többi csoport (formok, modulok, osztálymodulok) elemeit szabadabban kezelhetjük. Létrehozhatunk (Insert), törölhetünk (Remove), behozhatunk (Import), kivihetünk (Export) elemeket (lásd 1.13. ábra).

A formokról, a vizuális formtervezésről az 1.2.2.7. fejezetben lesz szó.

A modulok csoport moduljai a forráskódok megadására használatosak. A logikailag összetartozó forráskódokat célszerű egy modulba tenni. Az eseménykezelőket a megfelelő objektum moduljában (azaz nem a modulok csoportban) kell elhelyezni. A modulok felépítéséről az 1.2.2.5. fejezetben lesz szó.

Az osztálymodulokban osztályokat (objektumtípusokat) definiálhatunk.

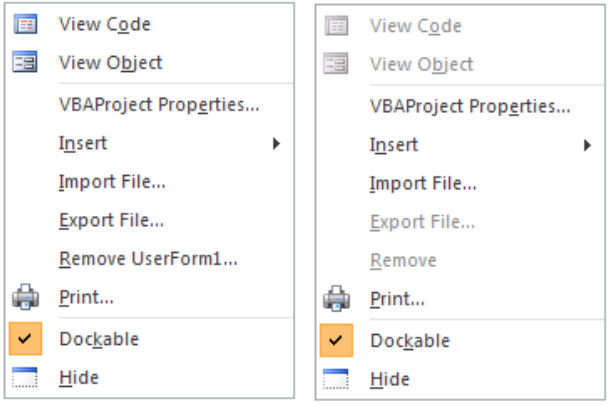
Az objektumokhoz tartozó modulok megnyitása (a Project ablak megfelelő elemének kiválasztása után) a View menü Code funkciójával (lásd 1.11. ábra), vagy a Project ablak helyi menüjének View Code funkciójával (lásd 1.13. ábra) történhet. A modulok csoport moduljait még (a megfelelő modulon való) dupla kattintással is megnyithatjuk.



1.12. ábra. A Visual Basic Editor Insert menüje

Új objektum létrehozása a Visual Basic Editor Insert menüjével (lásd 1.12. ábra), vagy a Project ablak helyi menüjével (lásd 1.13. ábra) történhet. A menükben található menüpontok választhatósága az aktuális állapottól

(kódszerkesztésben voltunk-e éppen vagy sem), illetve a Project ablak aktuális elemétől függ.



1.13. ábra. A Project ablak helyi menüje

A Project ablak elemeit a helyi menü Remove funkciójával törölhetjük, ahol a menüpont felirata a kiválasztott objektum nevét is tartalmazza (lásd 1.13. ábra bal oldali képét, ahol a UserForm1 objektum helyi menüje látható). A törlés előtt lehetőségünk van a törölt elem exportálására (azaz önálló fájlként való kimentésére), ami egyébként a helyi menü Export File... funkciójával is megtehető.

Az Import File... funkcióval betölthetünk és az aktuális projekthez adhatunk önálló fájlokban lévő formokat (*.frm), modulokat (*.bas), és osztálymodulokat (*.cls). Az export és import funkciókkal tehát könnyen tudunk objektumokat átvinni projektek között.

Megjegyzés

• A *.frm, *.bas, *.cls fájltípusokat nemcsak az Excel, hanem más MS Office alkalmazások, valamint az MS Visual Studio fejlesztőrendszer szoftverei is egységesen kezelik.

• A formok exportálásakor a *.frm fájlon kívül egy *.frx fájl is létrejön, amely bináris információkat tárol a formról.

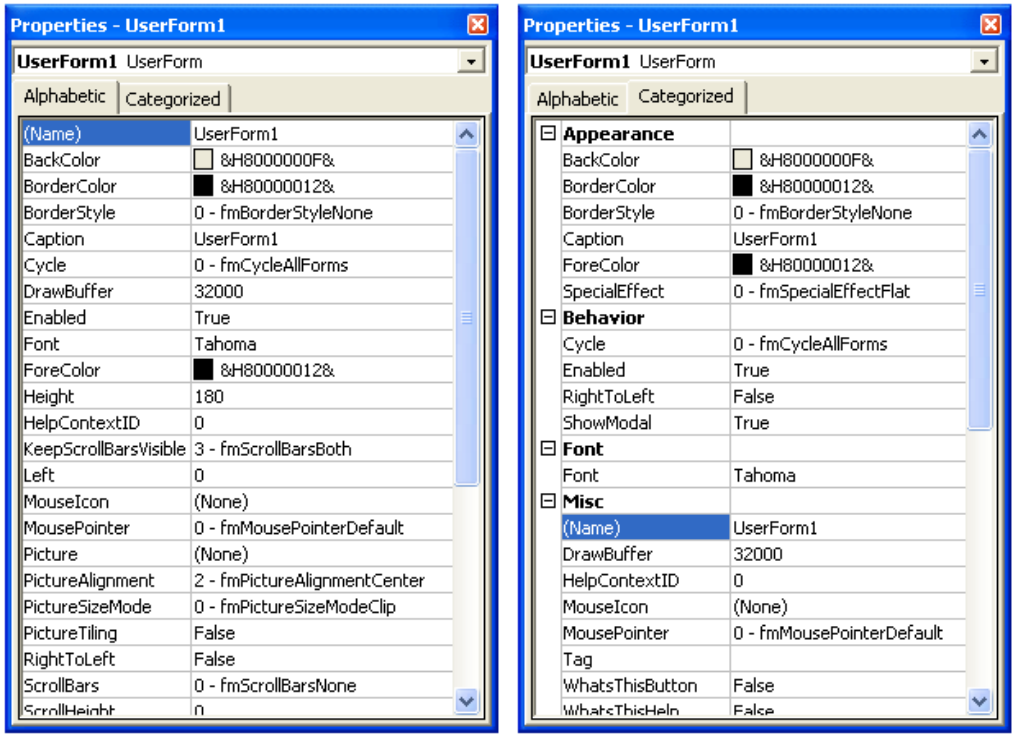
• Az osztálymodulokkal nem foglalkozunk (mert ez a témakör „túlnő” a programozási alapismereteken).

1.2.2.3. A Properties ablak

A Properties (tulajdonságok) ablak a projekthez tartozó objektumok tulajdonságainak megjelenítésére és azok tervezéskori megadására, módosítására használatos. A vizuális formtervezés (lásd 1.2.2.7. fejezet) során itt adjuk meg az adott form, illetve a formon lévő vezérlők tulajdonságait (lásd 1.10., 1.14. ábra).

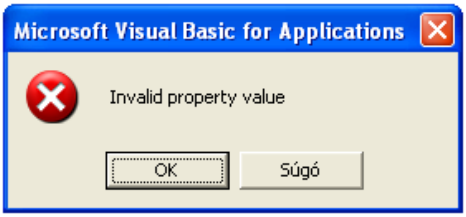
Az ablak tetején lévő legördülő listában kiválasztható az a vezérlő (a formot is beleértve), amelynek tulajdonságait látni, illetve megadni szeretnénk. A tulajdonságokat kétféle sorrendben tudjuk megjeleníteni (a megfelelő fül kiválasztásával): névsor szerint (Alphabetic fül), vagy a tulajdonságokat kategóriák szerint csoportosítva (Categorized fül). Ha már tudjuk az adott tulajdonság nevét, akkor a névsor szerinti fülön gyorsan megtaláljuk, különben célszerű a kategorizált fülön keresgélni, hiszen ott az összes tulajdonság helyett elegendő azon csoport végignézése, amelyikhez a keresett tulajdonság tartozik.

A Properties ablakban (mindkét fül esetén) két oszlop látható, a bal oldali a tulajdonságok neveit, a jobb oldali azok aktuális értékeit tartalmazza. Az adatok megadása a jobb oldali oszlopban történik, ahol is az adott tulajdonságtól függően vagy begépeljük a tulajdonság értékét (pl. Name, Caption, Left), vagy egy készletből kiválasztjuk (pl. Enabled, MousePointer, PictureAlignment), vagy egy párbeszédablakkal adjuk meg (pl. Font, Picture).



1.14. ábra. A Properties ablak

A begépeléssel megadott adatok értéke ellenőrződik, érvénytelen adat esetén (pl. szám helyett szöveget adunk meg) hibaüzenetet kapunk (lásd 1.15. ábra).



1.15. ábra. Érvénytelen tulajdonság értékről tájékoztató hibaüzenet

Megjegyzés

• Az Alphabetic fülön a név (Name) tulajdonság nem a névsor szerinti helyén jelenik meg, hanem a lista elején.

• A Categorized fülön az egyes tulajdonság csoportok (mint pl. megjelenés (Appearance), viselkedés (Behavior)) az előttük megjelenő (mínusz, plusz) ikonokkal becsukhatók (egysorossá), illetve kinyithatók.

• A tulajdonságok nevéből általában már kiderül a szerepük, de a tulajdonság ablakban is „él” a környezetfüggő súgó, így az aktuálisan kiválasztott tulajdonság súgójához elegendő leütni az F1 billentyűt.

• Bizonyos tulajdonságok (pl. BackColor, ForeColor) értéke többféle (pl. begépelés, választás) módon is megadható.

• Csoportos kijelölés esetén (lásd 1.2.2.7. fejezet) a Properties ablakban a kijelölt vezérlők közös tulajdonságai jelennek meg (pl. a Name tulajdonság ekkor nem látható), és ezek egyszerre módosíthatók.

• Az ablak tetején lévő legördülő listában olyan vezérlő is kiválasztható, amely nem jelenik meg a formon tervezéskor, így nem tudunk rákattintani (pl. ha véletlenül olyan pozíciót (lásd Left, Top tulajdonságok) adtunk meg, amivel a vezérlő „lekerült” a formról).

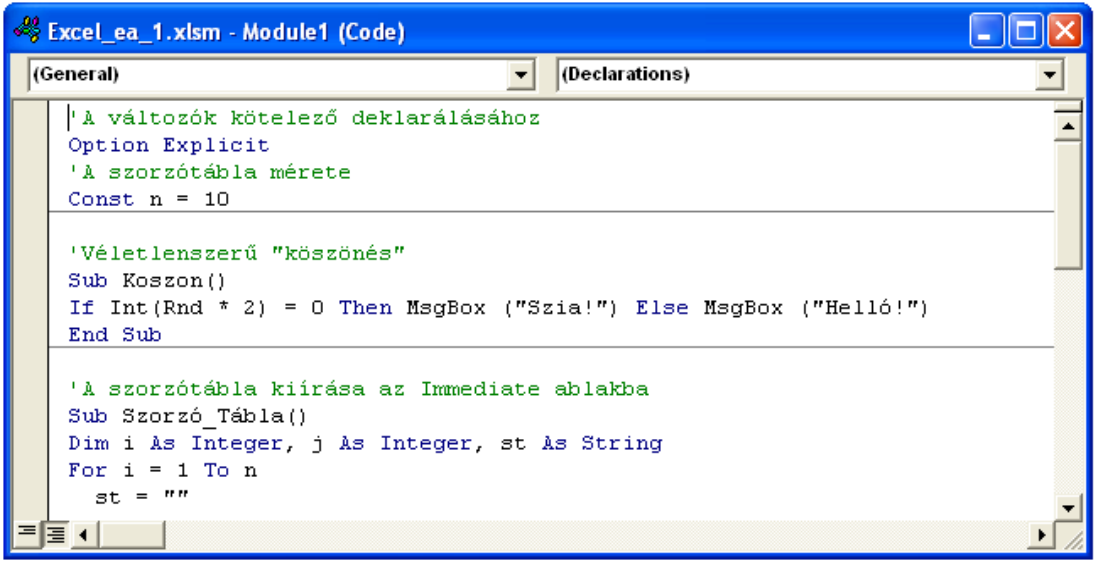
• Vannak olyan tulajdonságok, amelyek csak futási időben érhetők el, és vannak olyanok is, amelyek csak olvashatók (read-only) (pl. ListBox.ListCount).

• A fontosabb tulajdonságokról a vizuális formtervezésben (lásd 1.2.2.7. fejezet) lesz szó.

1.2.2.4. A kódszerkesztő ablak

A kódszerkesztő ablak a VBA forráskódok megírására, azok módosítására szolgál. A forráskódok modulokba, azon belül pedig szubrutinokba szervezettek. A kódszerkesztő ablak tetején két legördülő lista található, a bal oldaliban az objektumok, a jobb oldaliban a szubrutinok nevei jelennek meg, illetve választhatók ki.

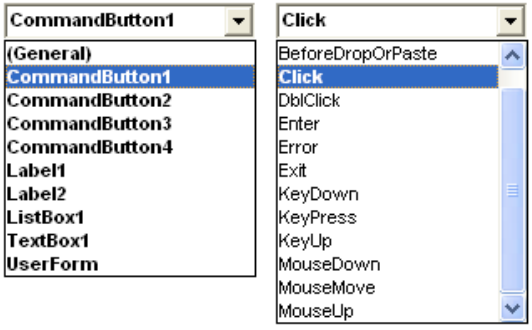
A listák tartalma egyfelől igazodik a kurzor aktuális (forráskódbeli) helyéhez, másfelől segítségükkel gyorsan rápozícionálhatunk a modul kívánt részére. Választáskor általában először a bal oldali listából, utána a jobb oldaliból választunk, mert a jobb oldali lista tartalma a bal oldali lista aktuális eleméhez igazodik.



1.16. ábra. A kódszerkesztő ablak

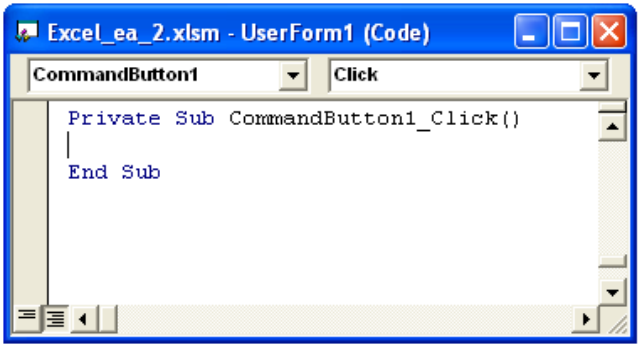
A bal oldali listában az első elem mindig egy (General) nevű elem. Ha ez az elem van kiválasztva, akkor a jobb oldali listában az első elem egy (Declarations) nevű elem (lásd 1.16. ábra), a többi elem pedig a modulban

található általános (nem eseménykezelő) szubrutinok nevei. A (Declarations) elem kiválasztása a modul elejére pozícionál (azaz a deklarációs részre, ahol a modulszintű utasítások találhatók, lásd 1.2.2.5. fejezet), egyébként meg a kiválasztott szubrutin elejére.



1.17. ábra. A legördülő listák tipikus tartalma egy formhoz tartozó modul esetén

A bal oldali listában (a (General) elemen kívül) az adott modul objektumai (azok nevei) jelennek meg (pl. egy form moduljában a form és a rajta található vezérlők (lásd 1.17. ábra), a munkafüzethez tartozó modulban a munkafüzet objektum). Ha egy objektum van kiválasztva a listában (és nem a (General) elem), akkor a kiválasztott objektumhoz tartozó eseménykezelők (mint szubrutinok) nevei jelennek meg a jobb oldali listában (lásd 1.17. ábra). A már definiált eseménykezelők nevei félkövéren jelennek meg. Egy még nem definiált eseménykezelő kiválasztása létrehozza az adott eseménykezelőt üres tartalommal, amit szerkeszthetünk (lásd 1.18. ábra), egyébként meg a kiválasztott eseménykezelő elejére ugrik a kurzor a kódszerkesztő ablakban.



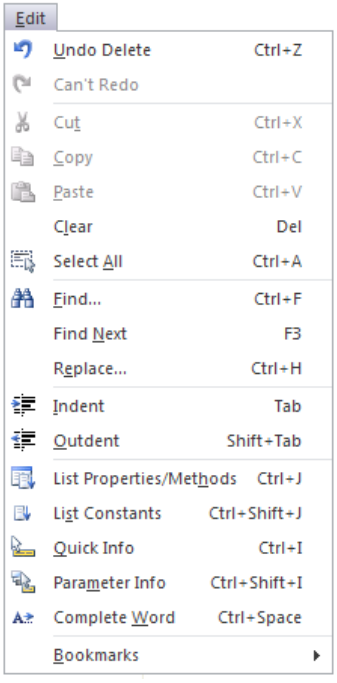
1.18. ábra. Egy létrehozott új eseménykezelő

Megjegyzés

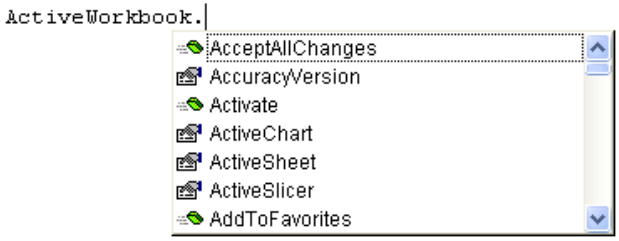
• Ha a jobb oldali legördülő listában egy eseménykezelő van kiválasztva, és a bal oldali listában egy másik objektumot választunk ki, akkor, ha van az adott objektumnak olyan eseménykezelője, akkor arra pozícionálunk, ha nincs, akkor létrejön az objektum ezen eseménykezelője. Az így létrejött (esetleg felesleges) üres eseménykezelők (amik a futást nem befolyásolják) természetesen törölhetők.

• A kódszerkesztő ablakok két részre oszthatók a Window menü Split funkciójával (lásd 1.11. ábra), így a forráskód két tetszőleges részét láthatjuk, illetve szerkeszthetjük.

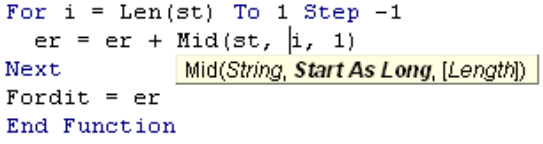
A kódszerkesztő „viselkedése” a hozzá tartozó beállításoktól (lásd 1.23. ábra) is függ. Ezektől a beállításoktól függetlenül a kódszerkesztőben lehetőség van (többek között, lásd 1.19. ábra) egy objektum tulajdonságainak és metódusainak a megjelenítésére (List Properties/Methods, lásd 1.20. ábra), a szubrutinok paraméterezésének megjelenítésére (Parameter Info, lásd 1.21. ábra), valamint a szókiegészítésre (Complete Word, lásd 1.22. ábra) is.



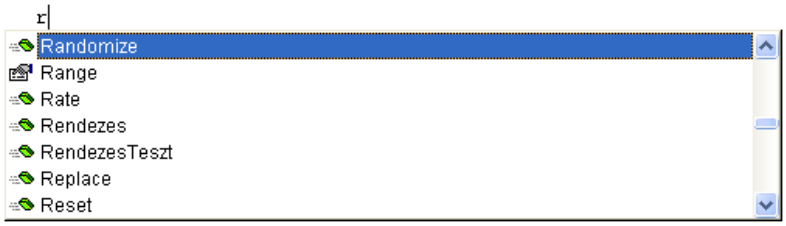
1.19. ábra. A Visual Basic Editor Edit menüje



1.20. ábra. A List Properties/Methods funkció

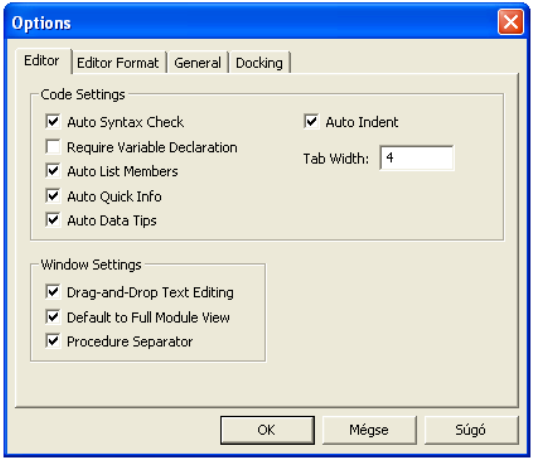


1.21. ábra. A Parameter Info funkció



1.22. ábra. A Complete Word funkció

A Visual Basic Editor „testreszabása” a Tools menüben található Options... funkcióval tehető meg. A funkció egy párbeszédablakot jelenít meg (lásd 1.23. ábra), ahol egyrészt megtekinthetők, másrészt módosíthatók az egyes beállítások. Az első két fül (Editor, Editor Format) a kódszerkesztő beállításait tartalmazza, a General fülön általános beállításokat végezhetünk, míg a Docking fülön az ablakok rögzíthetőségét szabályozhatjuk.



1.23. ábra. A Visual Basic Editor beállításainak párbeszédablaka

Az ablak négy fülén beállítható értékekből csak néhány, a kódszerkesztő működésére vonatkozó beállítást ragadunk ki. Az Auto Syntax Check kapcsoló a kódszerkesztő ablakban megadott forráskód sorainak automatikus szintaktikai ellenőrzését szabályozza. Bekapcsolt állapotban egy szintaktikailag helytelen sor elhagyásakor hibaüzenetet kapunk, és a sor a megfelelő színnel (az Editor Format fülön megadott Syntax Error Text alapján, ami alapértelmezésben piros betűszínű) kiemelődik.

Az Auto List Members kapcsoló az automatikus kódkiegészítést szabályozza. Bekapcsolt állapotban a forráskód begépelésekor megjelenik egy olyan lista a képernyőn, amely logikailag kiegészítheti az adott utasítást (pl. egy objektum után leütött pont hatására kiajánlódnak az objektum megfelelő tulajdonságai, metódusai).

Az Auto Quick Info kapcsoló bekapcsolt állapotában a szubrutinok paraméterezéséről kapunk egy felbukkanó súgót kódszerkesztés közben, ahol az éppen megadandó paraméter félkövéren jelenik meg.

Megjegyzés

- Célszerű megtartani az alapértelmezett beállításokat (amelyben pl. a fent említett kapcsolók mindegyike bekapcsolt állapotú).
- A forráskód szerkesztésekor a szintaktikailag helyes sor néha „átalakul”, amint a kurzor elhagyja az adott sort. A kulcsszavak kiemelődnek (alapértelmezésben kék színnel), a felismert (azaz nem elgépelt) azonosítók a deklaráláskor megadott alakban (kis- és nagybetű) jelennek meg, esetlegesen szóközök szúródnak be, illetve felesleges szóközök törlődnek. Ez az „egységesített küllem” javítja az áttekinthetőséget, a forráskód olvashatóságát.
- Az objektumok tulajdonságait és metódusait célszerű a kódkiegészítéssel kiajánlott listából kiválasztani, mert gyorsabb, és így biztosan nem gépeljük el.
- Az egyes szubrutinokat (valamint a deklarációs részt) alapértelmezésben egy vízszintes vonal választja el egymástól (aminek megjelenését a Procedure Separator kapcsoló szabályozza (lásd 1.23. ábra)).

1.2.2.5. Modulok felépítése

Minden modul, legyen az objektumhoz tartozó vagy önálló modul (lásd 1.2.2.2. fejezet), ugyanazt az egyszerű felépítést követi. A modul elején a modulszintű utasításokat kell megadni, amelyeket a modul szubrutinjai követnek.

Modulszintű utasítások

- Deftype utasítások (pl. DefInt)
- Opciókat megadó utasítások (pl. Option Explicit)
- Típusdeklarációk (pl. Type, Enum)
- Modulszintű konstansok (Const), illetve változók (Public, Private, Dim) deklarálása

Megjegyzés: Az opciókat megadó utasításokkal a Visual Basic fordító/értelmező működését szabályozhatjuk. Ezek, illetve a Deftype utasítások érvényessége (scope) csak az adott modulra terjed ki. A modulszintű típus, konstans és változó deklaráció azonban deklarálhat más modulból is hivatkozható (publikus) elemeket is.

Pl.

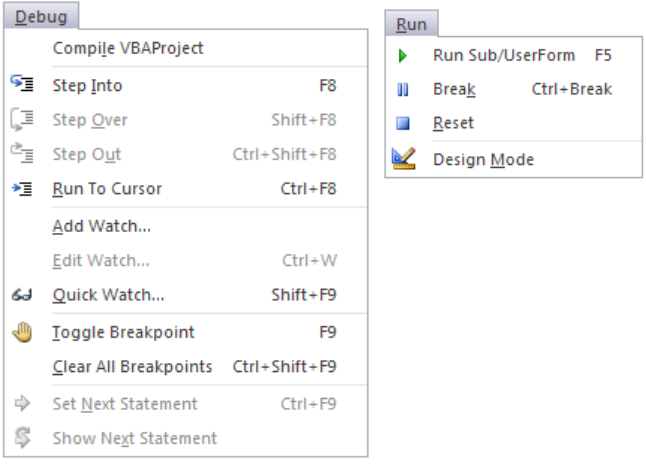
```
'A változók kötelező deklarálásához
Option Explicit
'A szorzótábla mérete
Const n = 10
'Véletlenszerű "köszönés"
Sub Koszon()
If Int(Rnd * 2) = 0 Then MsgBox "Szia!" Else MsgBox "Helló!"
End Sub
'A szorzótábla kiírása az Immediate ablakba
```

```
Sub Szorzo_Tabla1()
Dim i As Integer, j As Integer, st As String
For i = 1 To n
        st = ""
        For j = 1 To n
                st = st + " " & i & "*" & j & "=" & i * j
        Next
        Debug.Print st
Next
End Sub
'A szorzótábla egy másfajta kiírása az Immediate ablakba
Sub Szorzo_Tabla2()
'Egy adat mezőszélessége
Const msz = 5
Dim i As Integer, j As Integer
For i = 1 To n
        Debug.Print Tab(i * msz + 1); i;
Next
Debug.Print
For i = 1 To n
        Debug.Print i;
        For j = 1 To n
```

```
            Debug.Print Tab(j * msz + 1); i * j;
        Next
        Debug.Print
    Next
End Sub
```

1.2.2.6. Fordítás, futtatás, hibakeresés

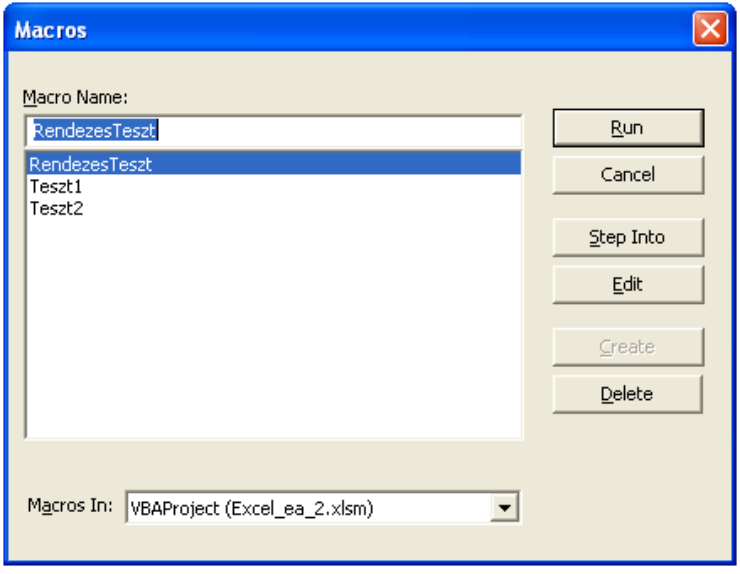
A forrásprogramok írásakor a kódszerkesztő már ellenőrizni tudja az adott sor szintaktikáját (lásd 1.2.2.4. fejezet). A Visual Basic Editor Debug menüjének Compile VBAProject funkciójával (lásd 1.24. ábra) az adott projekthez tartozó összes forrásprogram lefordítható. Ezzel kiszűrhetők a forrásprogramok szintaktikai (formai) hibái, ugyanis az első megtalált hibáról üzenetet kapunk, a kódszerkesztő ablakban ez a kódrészlet jelenik meg, és a hibás sor kiemelődik.



1.24. ábra. A Visual Basic Editor Debug és Run menüje

A forráskódok természetesen nemcsak fordíthatók, de futtathatók is. A futtatás funkció a Run menü Run Sub/UserForm funkciójával (lásd 1.24. ábra) végezhető. A funkció neve is utal arra, hogy egy szubrutint, vagy egy felhasználó által készített formot (UserForm) lehet futtatni, és a kettő között különbség van, ugyanis, ha egy formhoz tartozó modulban kérjük ezt a funkciót (pl. az F5 gyorsbillentyűvel), akkor függetlenül attól, hogy a modul mely részén áll a kurzor, a form futtatását kérjük. Az ilyen modulban lévő szubrutinok tehát nem futtathatók külön-külön. A futtatás funkció nem fordítja le az eseménykezelő szubrutinokat.

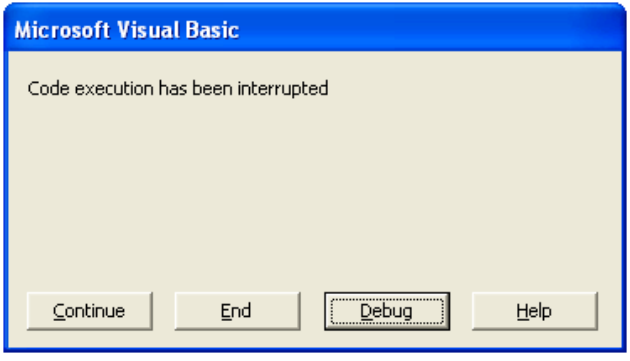
Más a helyzet a többi (nem formhoz tartozó) modullal. Itt ugyanis minden olyan szubrutin külön futtatható, amelynek nincsen paramétere. álljunk a kurzorral a futtatni kívánt szubrutin egy sorára és kérjük a futtatás funkciót. Ekkor a szubrutin fordítása is megtörténik. A paraméterrel rendelkező szubrutinokat nem lehet közvetlenül futtatni, ezek csak közvetve, valamilyen őket meghívó szubrutin segítségével futtathatók.



1.25. ábra. A futtatható szubrutinok párbeszédablaka

Ha az aktuális sor nem tartozik egyik szubrutinhoz sem (pl. két szubrutin közötti üres soron vagy a modul deklarációs részében áll a kurzor), akkor a futtatás funkció egy párbeszédablakot jelenít meg (ugyanezt teszi a Tools menü Macros... funkciója is), amelyben a modulban található futtatható (paraméter nélküli) szubrutinok listája jelenik meg (lásd 1.25. ábra). Az ablakban kiválasztható a futtatandó (Run), lépésenként futtatandó (Step Into), szerkesztendő (Edit), vagy törlendő (Delete) szubrutin, amely akár egy másik projektben is lehet (mivel a Macros In legördülő lista segítségével akár az összes nyitott projekt futtatható szubrutinjának nevét is megjeleníthetjük a makrókat tartalmazó listában).

Egy program futásának megszakítása a Run menü Break funkciójával kérhető, amelynek gyorsbillentyűje Ctrl+Break vagy Esc (pl. egy végtelen ciklusba esett program futását csak ezekkel a billentyűkkel lehet megszakítani). A megszakított programfutás esetén a futás folytatható (Continue), leállítható (End), illetve felfüggeszthető (Debug) (lásd 1.26. ábra).



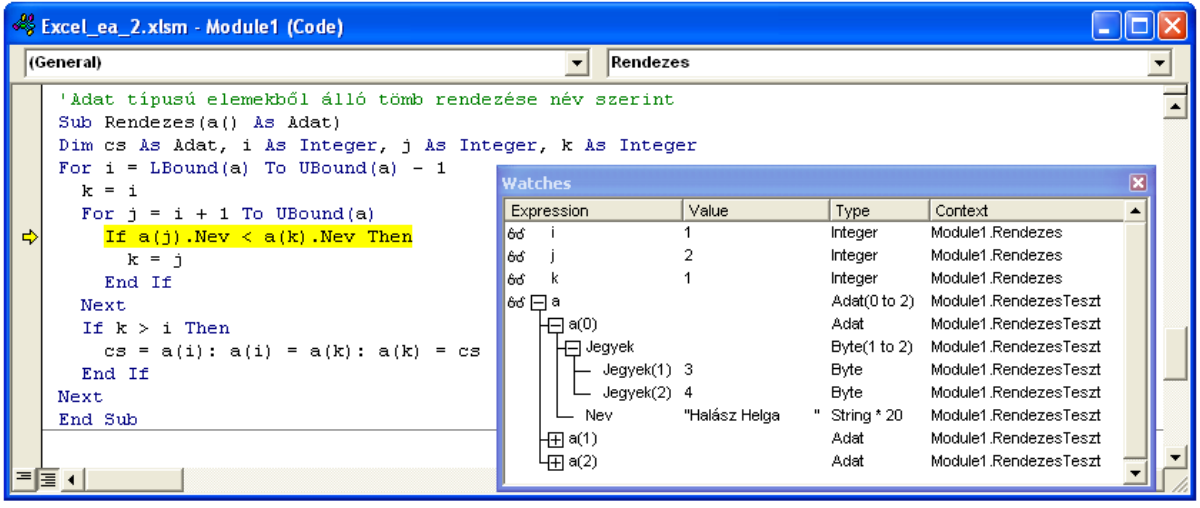
1.26. ábra. A megszakított programfutás párbeszédablaka

Egy futó (vagy futásában felfüggesztett) program futásának a befejezése a Run menü Reset funkciójával kérhető. A Run menü funkciói az eszköztársor segítségével is aktivizálhatók.

Egy program helyes működésének tesztelése a programfejlesztői munka része. Sajnos gyakran előfordul, hogy a programunk nem úgy működik, ahogy szeretnénk. A programok szemantikai (tartalmi, jelentésbeli,

logikai) hibáinak feltárása már nehezebb feladat (szemben azzal, hogy a szintaktikai hibákat egyetlen fordítással kiszűrhetjük).

A programfejlesztő rendszerek (így a VBA is) speciális funkciókat biztosítanak a programok szemantikai hibáinak felderítésére. Ezek a funkciók alapvetően két fontos szolgáltatást nyújtanak: megtudhatjuk, hogy merre halad a vezérlés (milyen utasítások kerülnek végrehajtásra), és hogy az egyes változókban milyen értékek vannak.

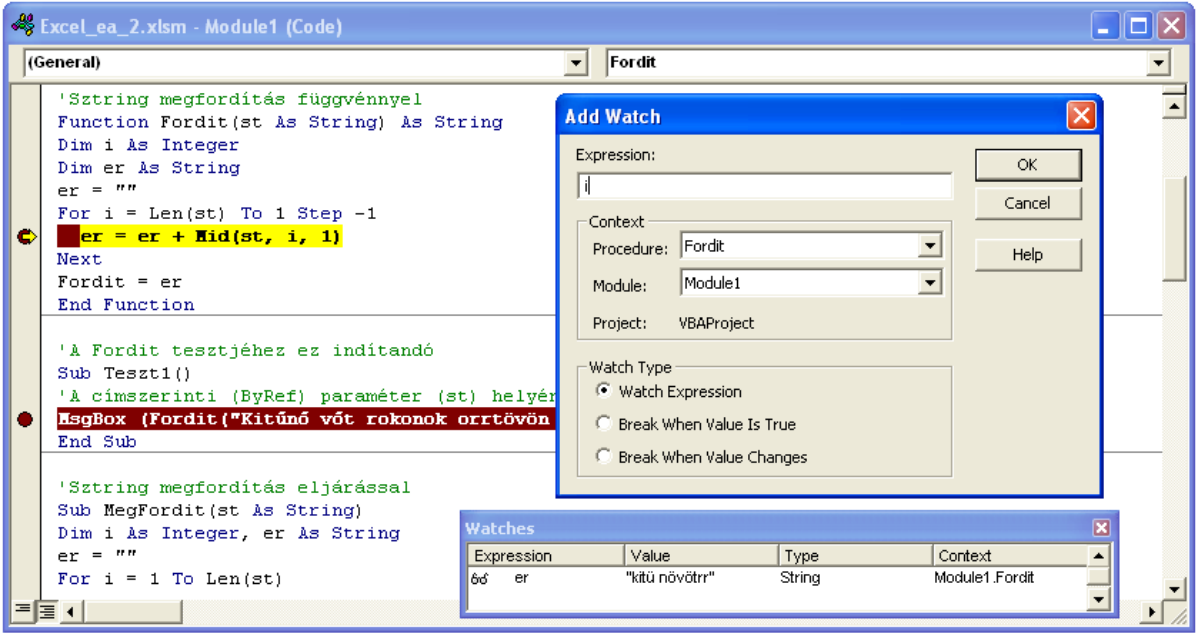


1.27. ábra. Lépésenkénti futtatás és a Watches ablak

A programok lépésenkénti futtatása a Debug menü Step Into funkciójával végezhető. Ez azt jelenti, hogy a végrehajtás utasításonként (de mivel általában egy sorba egy utasítást írunk, ezért gyakorlatilag soronként) történik, és minden egyes utasítás végrehajtása előtt a program futása felfüggesztődik. Ekkor lehetőségünk van a változók tartalmának megtekintésére (esetleges módosítására), majd a futás folytatására, illetve a

futás leállítására (ha pl. már rájöttünk az esetleges hibára). Az éppen végrehajtásra kerülő utasítás (alapértelmezésben) sárga háttérszínnel (és a margó sávon egy sárga nyíllal) emelődik ki (lásd 1.27. ábra).

Egy hosszabb forrásprogram esetén hasznos lehet, ha csak ott kezdjük el a lépésenkénti futtatást, ahol a hibát sejtjük. Ez megtehető a forráskódban elhelyezhető töréspontok (Breakpoints) segítségével. A töréspontok adott sorra való elhelyezése, illetve levétele a Debug menü Toggle Breakpoint funkciójával végezhető. A programfutás egy töréspontra érve felfüggesztődik (azaz lehetőségünk van a lépésenkénti futtatásra, stb.). A töréspontok sorát a margón egy kör jelzi, míg a sor (alapértelmezésben) bordó háttérszínnel emelődik ki (lásd 1.28. ábra).



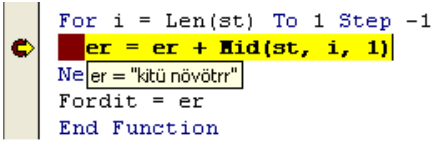
1.28. ábra. Egy töréspontón felfüggesztett programfutás a Watches és az Add Watch ablakokkal

A forráskód egy adott soráig való futtatás (Debug menü Run To Cursor funkció) is azt a célt szolgálja, hogy ott függesszük fel a program futását, ahol a hibát sejtjük.

A Debug menü Step Over funkcióját akkor használjuk, ha lépésenkénti futtatáskor egy olyan utasításra lépünk, amelyik szubrutinhívást tartalmaz (pl. egy saját eljárást vagy függvényt hívunk meg), és szeretnénk ezt az utasítást egy lépésben végrehajtani (mert pl. tudjuk, hogy a hiba nem abban a szubrutinban van). A Step Into funkció ugyanis ekkor belép az adott szubrutinba, és ott folytatódik a lépésenkénti futtatás. Ha beléptünk egy szubrutinba, akkor használható a Step Out funkció is, ami lefuttatja az adott szubrutint, majd a lépésenkénti futtatás a hívó szubrutinban a hívást követő utasítással folytatódik.

A változók értékeinek megfigyelési (Watches) ablakát (lásd 1.27. ábra) a View menü Watch Window funkciójával jeleníthetjük meg. Az ablak helyi menüjével új változókat vehetünk fel (Add Watch...) (lásd 1.28. ábra), vagy egy meglévőt módosíthatunk (Edit Watch...), illetve törölhetünk (Delete Watch...).

A tömb- és rekordváltozók az előttük megjelenő (mínusz, plusz) ikonokkal becsukhatók, illetve kinyithatók így az egyes tömbelemek, illetve rekordmezők is megtekinthetők (lásd 1.27. ábra).



1.29. ábra. Egy változó értékének megjelenítése az egérkurzor segítségével

Megjegyzés

• Felfüggesztett programfutás esetén az egyszerű (nem összetett adattípusú) változók tartalma úgy is megtekinthető, hogy az egérrel az adott változó fölé pozícionálunk (lásd 1.29. ábra). Ehhez az Auto Data Tips kapcsolónak (lásd Tools menü Options... funkció, Editor fül) bekapcsolt állapotban kell lennie (lásd 1.23. ábra).

• A Watches ablakban nemcsak változók értékeit figyelhetjük meg, de kifejezéseket is kiértékelhetünk. A változók tartalmát az értékük (Value) módosításával meg is változtathatjuk.

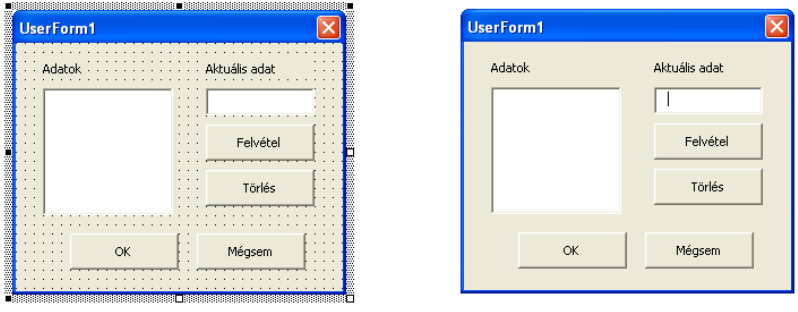
• Töréspontokat nem helyezhetünk el akárhol (pl. üres vagy egy Dim utasítást tartalmazó soron nem lehet töréspont). A töréspontokat a kódszerkesztő ablak bal szélén (a margó sávon) való egérkattintással is elhelyezhetünk (illetve levehetünk).

1.2.2.7. Vizuális formtervezés

A Windows alkalmazásokban a felhasználóval történő kommunikáció egyik alapvető eszköze az ablak (más néven form, űrlap). Természetesen az alkalmazás helyes működése a legfontosabb, de a küllem, az áttekinthetőség, a könnyű kezelés, vagyis a felhasználóbarát viselkedés is lényeges szempont. Éppen ezért a Windows platformú szoftverfejlesztő rendszerek (így az MS Excel VBA is) támogatják a formok vizuális, interaktív tervezését.

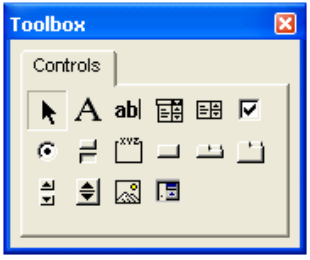
A tervezés (manapság általánosan elterjedt) jellemzője a WYSIWYG (What You See Is What You Get) tervezési mód, ami azt jelenti, hogy a tervezett ablak pontosan úgy fog megjelenni futáskor, mint ahogyan azt a tervezéskor látjuk (lásd 1.30. ábra).

Egy új form létrehozása a Visual Basic Editor Insert menüjének UserForm funkciójával (vagy a Project ablak helyi menüjének segítségével) hozható létre. A form (pl. a jobb szélén és alján lévő fehér keretező négyzetek segítségével) tetszőlegesen átméretezhető.



1.30. ábra. Egy form tervezéskor és futáskor

A formra tehető, használható vezérlőket (Controls) a Toolbox ablak jeleníti meg (lásd 1.31. ábra). Egy form tervezésekor (azaz ha a UserForm objektum ablaka aktív), a Toolbox ablak automatikusan megjelenik (és ha az ablakot esetleg bezárnánk, akkor a View menü Toolbox funkciójával, vagy az eszköztársor megfelelő ikonjával újra megnyithatjuk).



1.31. ábra. A Toolbox ablak

A Toolbox ablak a fontosabb Windows vezérlőket tartalmazza, mint pl. címke (Label), beviteli mező (TextBox), legördülő lista (ComboBox), lista (ListBox), jelölőnégyzet (CheckBox), választógomb (OptionButton), nyomógomb (CommandButton).

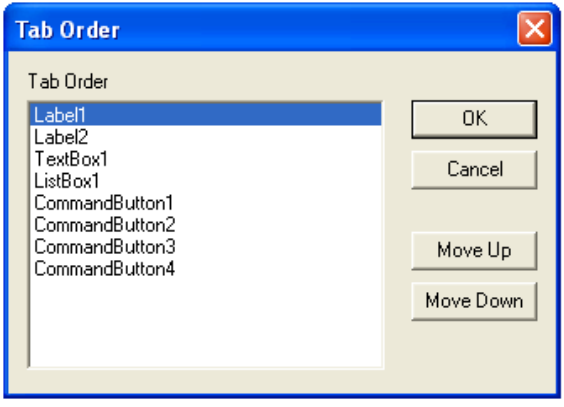
Egy vezérlő formra helyezése az egérrel történhet. A kívánt vezérlő a Toolbox ablakból a formra vihető a „fogd és vidd” egérművelettel, de a kívánt vezérlőn, majd a formon való kattintás is ugyanezt teszi, azaz elhelyezi a formon az adott vezérlő egy példányát (alapértelmezett méretben). Ha a formon kattintás helyett egy téglalap alakú területet jelölünk ki, akkor a vezérlő a megadott méretben kerül a formra.

A form, illetve a formon lévő vezérlők egérkattintással kijelölhetők, de a Windows-ban szokásos kijelölési technikákkal (lenyomott Ctrl vagy a Shift billentyű és egérkattintás, vagy jelölőkeret rajzolásával) csoportos elemkijelölés is végezhető. Csoportos kijelölést akkor használunk, ha a vezérlők egy csoportjára szeretnénk valamilyen műveletet (pl. mozgatás, törlés, adatmegadás) elvégezni.

A kijelölt vezérlők az egér „fogd és vidd” művelettel szabadon mozgathatók (áthelyezhetők), és méretezhetők (ehhez valamelyik keretező négyzetet kell mozgatnunk). Ha több vezérlő van kijelölve, akkor a Properties ablakban csak a közös tulajdonságok jelennek meg.

A kijelölt vezérlő (vagy vezérlők) kezelése az Edit menü (vagy a helyi menü) segítségével is történhet. A törlés a Delete billentyűvel (illetve a Delete funkcióval) végezhető, de a vágóasztal többi művelete (pl. vágólapra másolás, beillesztés) is használható.

Futáskor alapesetben az egyes vezérlők a formra való felkerülésük sorrendjében aktivizálhatók. Az első vezérlő lesz fókuszban, és a felhasználó a Tab (illetve Shift+Tab) billentyűkkel tud ezen sorrend szerint „lépkedni” az egyes vezérlőkön. Ez azonban nem feltétlenül igazodik a vezérlők formon való elrendezéséhez. A form helyi menüjének Tab Order funkciójával módosítható a vezérlők futáskori sorrendje (lásd 1.32. ábra). Az aktuálisan kiválasztott vezérlő felfelé, illetve lefelé mozgatható a megfelelő nyomógombokkal (Move Up, Move Down), így a megfelelő sorrend kialakítható.



1.32. ábra. A Tab Order ablak

A formra tett vezérlők a vezérlő típusából és egy sorszámból álló azonosítót (Name) kapnak (pl. CommandButton1). A sorszám egytől kezdődően kerül kiosztásra. A kifejezőbb, olvashatóbb programkód érdekében a vezérlőket át is nevezhetjük, ezt célszerű közvetlenül a formra való elhelyezés után megtenni, még mielőtt eseménykezelőket definiálnánk hozzá (ugyanis a megváltozott neveket az eseménykezelők nevei nem követik).

Egy vezérlő egy eseménykezelőjének létrehozása (illetve már meglévő eseménykezelő esetén az arra való rápozícionálás) általánosan a kódszerkesztő ablak tetején lévő legördülő listákból való választással történhet. Azonban a vezérlők alapértelmezett eseménykezelője (ami pl. nyomógomb esetén a kattintásnak megfelelő Click, beviteli mező esetén a tartalom megváltozásának megfelelő Change) a vezérlőn való dupla kattintással is megnyitható.

Az eseménykezelők azonosítója az adott objektum azonosítójából (Name), az alulvonás karakterből, és az esemény nevéből áll, a paraméterezésük pedig rögzített (pl. CommandButton1_Click(), ListBox2_DblClick(ByVal Cancel As MSForms.ReturnBoolean)).

Megjegyzés

- Egy vezérlő a Properties ablak legördülő listájával is kijelölhető.
- Az objektumoknak lehetnek olyan tulajdonságai is, amelyek csak futási időben érhetők el, ezek meg sem jelennek a Properties ablakban (pl. ListBox.ListIndex).
- Néhány fontosabb tulajdonság: név (Name), felirat (Caption), elhelyezkedés (Top, Left), méret (Width, Height), választhatóság (Enabled), láthatóság (Visible).

1.2.2.8. Mintafeladat

Ebben a fejezetben egy nem túl bonyolult, de azért már kellően összetett feladat megoldását mutatjuk be. Az egyszerűség és a könnyebb követhetőség érdekében a teljes feladatot több részfeladatra bontottuk.

1. Feladat: Készítsünk adatbeviteli formot több adat megadására!

Megoldás: Ahhoz, hogy több adat megadható legyen, biztosítanunk kell egy olyan vezérlőt, amelyben egy adat megadható, és egy olyat, amelyben az eddig megadott adatok megtekinthetők. Egy beviteli mezőt és egy lista vezérlőt fogunk használni. A beviteli mezőben megadott adat felvételét, elfogadását egy (Felvétel feliratú) nyomógombbal kérhetjük. A két vezérlőt címkékkel is ellátjuk (lásd 1.30. ábra).

Elvégzendő tevékenységek:

- A form létrehozása (Insert menü, UserForm funkció).

• A vezérlők formon való elhelyezése (kattintás a megfelelő vezérlőn, majd a formon, esetleges mozgatás, méretezés).

• Feliratok megadása (Caption tulajdonságok a Properties ablakban).

• A Felvétel gomb kattintás eseménykezelőjének megírása (pl. duplakattintás a Felvétel nyomógombon létrehozza az üres eseménykezelőt). Az eseménykezelőben a lista elemeit kell bővíteni a beviteli mezőben megadott adattal.

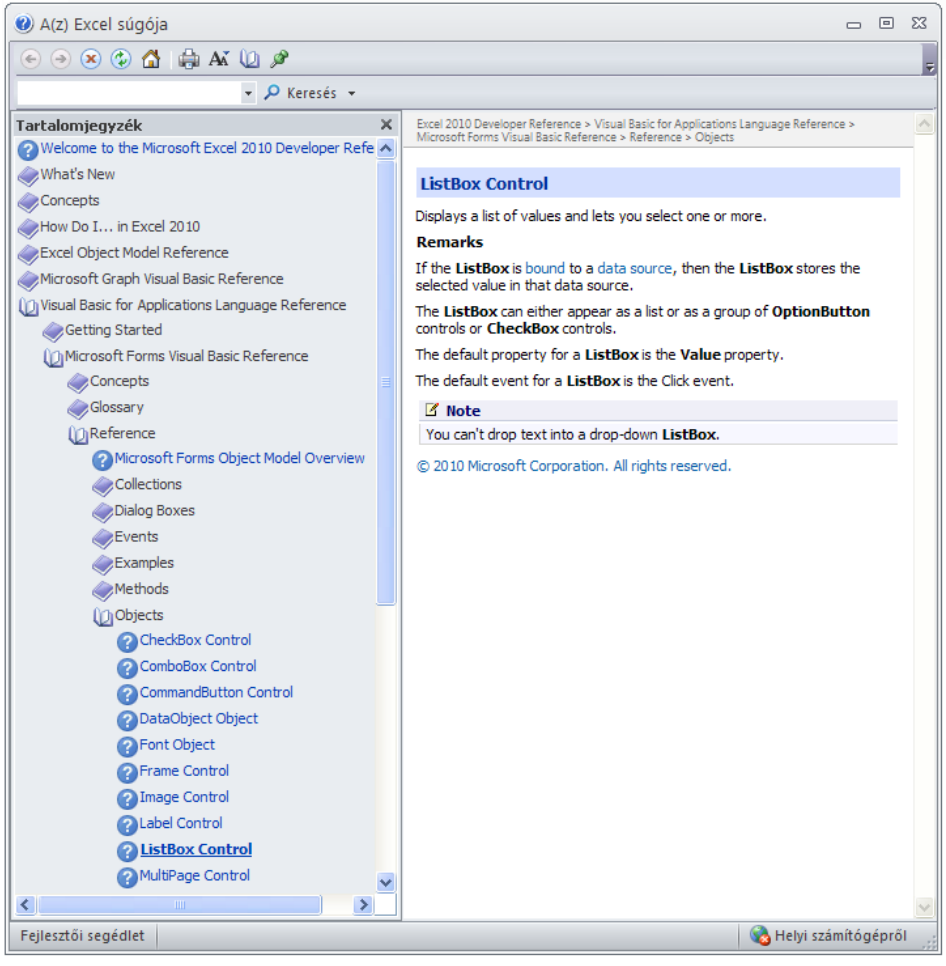
Az eseménykezelő forráskódja:

```
'Felvétel (minden adatot felvesz)
Private Sub CommandButton1_Click()
ListBox1.AddItem TextBox1.Text
End Sub
```

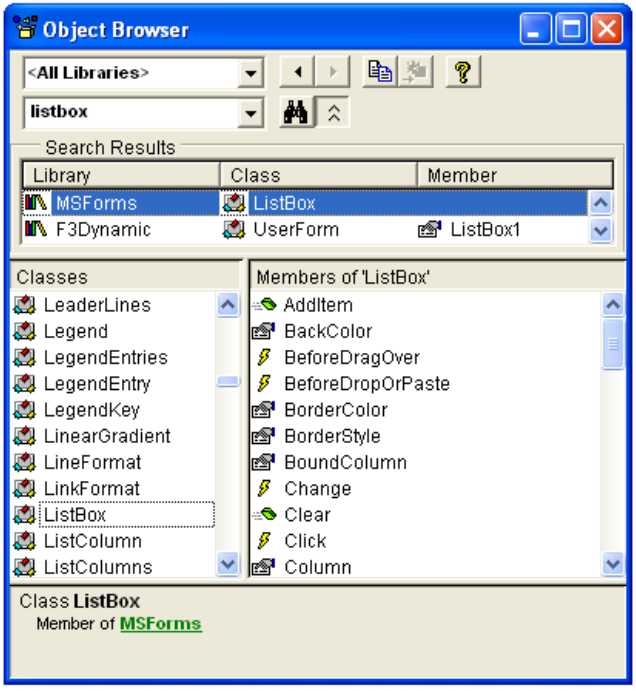
A feladat megoldásához szükséges információkhoz (nevezetesen, hogy hogyan hivatkozzuk a beviteli mezőben megadott adatot, és hogyan lehet egy lista elemeit bővíteni) a súgó, illetve az objektumtallózó (Object Browser) segítségével juthatunk.

Ha pl. tervezéskor kijelöljük a lista vezérlőt és lenyomjuk az F1 billentyűt, akkor az 1.33. ábrán látható súgótartalom jelenik meg. A súgó a Windows-ban megszokott módon használható (pl. kereshetünk benne, a megnézett oldalakra visszamehetünk, stb.)

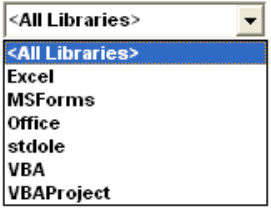
Az objektumtallózó a View menü Object Browser funkciójával jeleníthető meg. Itt (lásd 1.34. ábra) egyben láthatjuk az osztályok (így az egyes vezérlők) tulajdonságait (pl. BackColor), metódusait (pl. AddItem), és eseménykezelőit (pl. BeforeDragOver). Az ablakban keresési és súgó funkció is használható. A súgó itt is környezetfüggő (azaz a kérdőjel ikonnal (vagy az F1 leütésével) éppen az 1.33. ábrán látható súgótartalom jeleníthető meg, de ha a jobb oldali listában választunk ki egy elemet (pl. AddItem), akkor az ahhoz tartozó súgóoldal jelenik meg).



1.33. ábra. A Visual Basic Editor súgója



1.34. ábra. Az Object Browser



1.35. ábra. Az Object Browser ablak Project/Library legördülő listájának tartalma

Az Object Browser ablak (bal felső sarkában található) Project/Library legördülő listájával kiválasztható az a projekt, illetve rendszerkönyvtár (library), amelynek tartalmát megjeleníteni szeretnénk (lásd 1.35. ábra). Az 1.34. ábrán az All Libraries elem van kiválasztva, így minden elérhető osztály megjelenik a Classes listában.

2. Feladat: Ne fogadjunk el üres adatot!

Megoldás: A Felvétel gomb kattintás eseménykezelőjének módosítása. Üres adatnak vesszük azokat az adatokat is, amelyekben csak szóközök szerepelnek. Az adat kezdő és záró szóközeinek eltávolítására a Trim függvényt használtuk.

Az eseménykezelő forráskódja:

```
'Felvétel (üres adatot nem vesz fel)
Private Sub CommandButton1_Click()
Dim st As String
st = Trim(TextBox1.Text)
If st = "" Then
        MsgBox "Üres adat!"
Else
        ListBox1.AddItem st
End If
End Sub
```

3. Feladat: Egy korábban megadott adatot lehessen törölni!

Megoldás: Egy új nyomógomb (Törlés) formra helyezése, felirat megadása, és a gomb kattintás eseménykezelőjének megírása. Egy listában kiválasztott elem indexét a ListIndex tulajdonság mutatja (-1 esetén nincs a listában kiválasztott elem). Az első elem indexe 0, a másodiké 1, ..., az utolsóé ListCount-1 (ahol a ListCount tulajdonság a lista elemeinek számát adja). A törlést a lista RemoveItem metódusával végezzük.

Megjegyzés: A megoldásban kihasználtuk, hogy a listánkban (alapértelmezésben) egyszerre legfeljebb egy elem lehet kiválasztva. A többszörös elemkiválasztást megengedő listák esetén a Selected tulajdonságot kell használni az elemek kiválasztottságának lekérdezéséhez. Azt, hogy egy lista egyszeres vagy többszörös elemkijelölést enged-e meg, a MultiSelect tulajdonság szabályozza.

Az eseménykezelő forráskódja:

```
'A listában kiválasztott adat törlése
Private Sub CommandButton2_Click()
If ListBox1.ListIndex = -1 Then
    MsgBox "Nincs kiválasztott adat!"
Else
    ListBox1.RemoveItem ListBox1.ListIndex
End If
End Sub
```

4. Feladat: Legyen a formon egy OK és egy Mégsem feliratú nyomógomb! Az OK megnyomására írjuk ki a megadott elemek számát, a Mégsem megnyomására zárjuk be a formot!

Megoldás: Két új nyomógomb (OK, Mégsem) formra helyezése, feliratok megadása. A listában lévő elemek száma a lista ListCount tulajdonságával kapható meg. A form bezárására az End utasítást használjuk, ami befejezi a program futását.

Az eseménykezelők forráskódja:

```
'OK gomb
Private Sub CommandButton3_Click()
MsgBox "A megadott adatok száma:" & ListBox1.ListCount
End Sub
```

```
'Mégsem gomb
Private Sub CommandButton4_Click()
End
End Sub
```

5. Feladat: Ne lehessen két egyforma adatot megadni!

Megoldás: Egy keresőciklus segítségével megvizsgáljuk, hogy szerepel-e már a felvett adatok (azaz a lista elemei) között a beviteli mezőben megadott (kezdő és záró szóközöket már nem tartalmazó) adat, ha igen, akkor üzenetet adunk, egyébként meg felvesszük a lista elemei közé.

A keresésre (egy elöltesztelő) While ciklust használunk. A ciklus sorban megvizsgálja a lista elemeit az első elemtől kezdve, legrosszabb esetben egészen az utolsóig. Ha megtaláljuk a keresett adatot, akkor kilépünk a ciklusból.

```
'Felvétel (nem vesz fel üres vagy már létező adatot)
Private Sub CommandButton1_Click()
Dim st As String, i As Integer, van As Boolean
st = Trim(TextBox1.Text)
If st = "" Then
        MsgBox "Üres adat!"
Else
        'Szerepel-e már a kijelölt elem a ListBox1-ben?
        van = False: i = 1
        While Not van And (i <= ListBox1.ListCount)
                If st = ListBox1.List(i - 1) Then van = True Else i = i + 1
        Wend
```

```
        If van Then
                MsgBox "Van már ilyen adat!"
        Else
                'Nem szerepel még, felvesszük
                ListBox1.AddItem st
        End If
End If
End Sub
```

Megjegyzés

• A keresőciklusban az i változó értékét 1-től indítottuk ListCount-ig, és a lista elemeire való hivatkozásnál korrigáltunk (hogy jó legyen az elemekre történő hivatkozás). Természetesen 0-tól ListCount-1-ig is lépkedhettünk volna, ekkor az i-edik (és nem az i-1-edik) elemre kellene a ciklusban hivatkozni.

• A keresésre azért használtunk elöltesztelős ciklust, mert a lista lehet üres is. Ekkor ugyanis a ciklusmagban a listaelemre történő hivatkozás (futási) hibát eredményezne, azaz a ciklusmagot üres lista esetén nem szabad végrehajtani.

• Az alkalmazott kereső algoritmust soros (lineáris, szekvenciális) keresésnek nevezik, mert sorban vizsgálja meg azokat az adatokat, amelyek között keresünk. Ha esetleg rendezett adatok között kellene keresni, akkor egy hatékonyabb, ún. bináris keresés (lásd szakirodalom, pl. [3]) is használható lenne. Ez a keresés ugyanis egy sikertelen hasonlítás után felére csökkenti azon elemek számát, amelyek még szóbajöhetnek az elem megtalálását illetően. A soros keresés csupán egyetlen (az éppen megvizsgált és nem egyező) elemmel csökkenti a találatra esélyes elemek halmazát.

Önellenőrzés

1. Az alábbi állítások közül melyek igazak a makrókkal kapcsolatosan?
   - [ ] A VBA forráskódok egyben makrók is.
   - [ ] Egy makrókat tartalmazó dokumentum megnyitásakor meg kell adni, hogy engedélyezzük a dokumentumban lévő makrókat vagy sem.
   - [ ] A nem engedélyezett makrók nem láthatók és nem módosíthatók.
   - [ ] A makrók engedélyezése a makrók futtatására vonatkozik.
   - [ ] Egy új dokumentumban létrehozott makrók futtathatók.
   - [ ] A makrók is mentődnek a dokumentum mentésekor, ha annak formátuma megfelelő.
2. Az alábbi állítások közül melyek igazak a Visual Basic Editor-ral kapcsolatosan?
   - [ ] A VBE az Alt+F11 gyorsbillentyűvel akkor is elindítható, ha az Excel menüjében nem látható az indítást végző ikon.
   - [ ] Ha a makrók nem engedélyezettek egy dokumentumban, akkor az Immediate ablakban sem hajthatunk végre utasításokat.
   - [ ] A Project ablak egy objektumának törlésekor lehetőség van az adott objektum önálló fájlba való exportálására.
   - [ ] A Properties ablakban egy adatmegadással több objektum adata is megadható.
   - [ ] A kódszerkesztő ablak legördülő listáiból való választás egy új eseménykezelőt nyit.
   - [ ] A kódszerkesztő a beállításoktól függetlenül képes a List Properties/Methods funkcióra. (igaz)
   - [ ] A Toolbox ablakból egy egér „fogd és vidd” művelettel több vezérlő is a formra tehető.
   - [ ] A Tab Order funkció a vezérlők formon való elrendezésére szolgál.
   - [ ] Az Object Browser funkcióval a saját projektjeink objektumai is megjeleníthetők.

3. Az alábbi állítások közül melyek igazak a fordítás, futtatás, hibakereséssel kapcsolatosan?

- ☐ A szubrutinokat egyenként is lefordíthatjuk.
- ☐ Egy végtelen ciklusba esett program futása megszakítható.
- ☐ Egy töréspontonon a program futása befejeződik.
- ☐ A Watches ablakban összetett típusú változók tartalma is megjeleníthető.
- ☐ Töréspont egy forrásprogram tetszőleges során elhelyezhető.
- ☐ Az Auto Data Tips funkció alkalmas az összetett típusú változók tartalmának megjelenítésére.

4. Hibakeresés, nyomkövetés

Hozzunk létre egy új modult, és azokat a szubrutinokat, amelyeket közvetlenül el szeretnénk indítani, tegyük ebbe a modulba!

Pl.

```
Sub Ciklus()
Dim i As Integer
i = 1
While i < 10
      Debug.Print i
      i = i + 2
Wend
End Sub
```

- Helyezzünk el töréspontokat a tesztelendő szubrutinban, majd futtassunk törésponttól töréspontig, illetve lépésenként!
- Vegyünk fel változókat a Watch ablakba, és nézzük meg az aktuális értékük változását!
- Állítsunk le egy felfüggesztett futást!

5. Vizuális formtervezés

- Hozzunk létre egy formot, és jelenítsük meg a Properties, illetve Toolbox ablakot (ha esetleg nem látszódnának)!
- Tegyünk fel többféle vezérlőt a formra!
- Méretezzünk, helyezzünk más helyre vezérlőket a formon!

- Próbáljuk ki a vágóasztal műveleteit az egyes vezérlők másolására, törlésére!
- Módosítsunk az egyes vezérlők tulajdonságain a Properties ablakban!
- Próbáljuk ki a vezérlők adatainak csoportos módosítását!

6. A mintafeladat elkészítése
   - Készítsük el a vizuális tervezés mintafeladatát (lásd 1.2.2.8. fejezet)! A form több adat megadását biztosítja egy beviteli mező (TextBox), egy lista (ListBox), és négy darab nyomógomb (CommandButton) (Felvétel, Törlés, OK, Mégsem) segítségével. Készítsük el a formot és a megfelelő eseménykezelőket, majd teszteljük a form működését!

7. Programozási feladatok
   Oldjuk meg az előző leckék programozási feladatait úgy, hogy a felhasználóval való kommunikációt form segítségével végezzük! Célszerű feladatonként egy-egy formot létrehozni! Az adatok bekérését beviteli mezők (TextBox), az egyes funkciók végrehajtását nyomógombok (CommandButton) segítségével végezzük! Pl.

```
Function Haromszog(a As Single, b As Single, c As Single) As Boolean
Haromszog = a + b > c And b + c > a And a + c > b
End Function
Private Sub CommandButton1_Click()
'A típuskonverzió (szöveget számmá) automatikusan végrehajtódik
If Haromszog(TextBox1.Text, TextBox2.Text, TextBox3.Text) Then
        MsgBox "A három szám lehet egy háromszög három oldala!"
Else
        MsgBox "A három szám nem lehet egy háromszög három oldala!"
End If
End Sub
```

5. LECKE

Az Excel programozása

1.3. Az Excel programozása

Az eddigi fejezetek általános ismereteket adtak a Visual Basic nyelvről, a Visual Basic Editor fejlesztőkörnyezetről, a vizuális tervezésről, és általában a fejlesztés mikéntjéről, de nem használták az Excel objektumait, a VBA-ban „készen kapott” objektumhierarchia lehetőségeit. Ebben a fejezetben a fontosabb Excel objektumokról, és azok forráskódból történő használatáról lesz szó.

1.3.1. A használható objektumok, objektumtípusok

Az objektumokhoz kötődő alapfogalmakról már olvashattunk (lásd 1.1.6. fejezet), most az Excel VBA környezetben használható objektumokról és objektumtípusokról lesz szó.

A fejlesztőkörnyezetben egységesen használható osztályokat (illetve modulokat, szubrutinokat, típusokat) az őket tároló fájlok (illetve rendszerfájlok) csoportosítják (lásd 1.35. ábra):

- Excel osztályok (pl. Workbooks, Range, Chart, Dialogs)
- MS Forms osztályok (pl. TextBox, ListBox, UserForm)
- MS Office osztályok (pl. CommandBar, CommandBarControl)
- VBA osztályok (pl. Collection)
- VBA Project osztályok (az adott projektben definiált osztályok)

Megjegyzés

- Az általunk használt objektumok (pl. Application, Debug) az Excel futásakor már létező (azaz hivatkozható) objektumok.
- Futásidőben is létrehozhatók új objektumok (pl. Worksheets.Add), illetve már létező objektumok megszüntethetők (pl. Charts(1).Delete).
- Az objektumtípusokat hasonlóan kezelhetjük, mint a VBA többi típusát (pl. Dim r As Range).

1.3.1.1. Az Application objektum

Az Application objektum az Excel alkalmazás objektuma. Segítségével az alkalmazáshoz tartozó összes objektum (pl. a nyitott munkafüzetek, projekt objektumok) elérhető. Az Application objektum feladata az alkalmazásszintű beállítások, tevékenységek végrehajtása is (pl. DisplayAlerts, ReferenceStyle).

Az Application objektum néhány fontosabb tulajdonsága: Workbooks, Worksheets, Charts, Sheets, ActiveWorkbook, ActiveSheet, ActiveCell, ActiveChart, Range, Cells, Rows, Columns, Selection.

Megjegyzés

- Az Application objektum tulajdonságaira, illetve metódusaira való hivatkozásból az Application szó elhagyható (pl. Application.ActiveCell helyett elegendő az ActiveCell hivatkozás), ezért általában el is hagyjuk. Az objektum nélküli hivatkozásokban (pl. Range) az Application objektum tulajdonságait, illetve metódusait hivatkozzuk.
- A Selection olyan speciális tulajdonság, amely az aktuálisan kijelölt objektumot adja eredményül, így az eredmény típusa attól függ, hogy éppen mi van kijelölve (pl. cellatartomány esetén egy Range objektum, diagram esetén egy Chart objektum lesz az eredmény).

1.3.1.2. Gyűjtemények

A gyűjtemények azonos típusú objektumok egy összessége. Egy gyűjtemény elemeire indexekkel (1-től induló sorszámokkal) hivatkozhatunk (pl. Workbooks(1).Worksheets(1).Cells(1,1)). Ha egy gyűjtemény elemei rendelkeznek név (Name) tulajdonsággal, akkor az elemekre a nevükkel is hivatkozhatunk (pl. Workbooks(”Munkafüzet1.xlsm”).Worksheets(”Munka1”).Cells(1,1)).

A gyűjtemények rendelkeznek Count tulajdonsággal, amivel a gyűjtemény elemeinek száma kapható meg, és többnyire Add metódussal is, amivel a gyűjtemény egy újabb elemmel bővíthető. A további tulajdonságok és metódusok (pl. az elemek törlésére szolgáló Delete metódus) létezése az egyes gyűjteményektől függ (pl. a Charts gyűjteménynek van ilyen metódusa, a Workbooks gyűjteménynek nincs).

A fontosabb gyűjtemények:

- Munkafüzetek: Workbooks gyűjtemény (munkafüzet (Workbook) objektumokból).

- Munkalapok: Worksheets gyűjtemény (munkalap (Worksheet) objektumokból).
- Diagramok: Charts gyűjtemény (diagram (Chart) objektumokból).
- Lapok (munkalapok, önálló lapon lévő diagramok, stb.): Sheets gyűjtemény.
- Párbeszédablakok: Dialogs gyűjtemény (beépített párbeszédablak (Dialog) objektumokból).
- Menük: CommandBars gyűjtemény (menüsor (CommandBar) objektumokból).

Megjegyzés

- A gyűjteményeket könnyű felismerni a nevük végén lévő „s” betűről (angol többes szám).
- A Sheets gyűjtemény elemeinek (lapjainak) típusa a Type tulajdonsággal kapható meg (Pl. Sheets(1).Type).
- Az önálló lapon lévő diagramok a Charts gyűjteménnyel, míg az egyes munkalapokon lévő diagramok az adott munkalap ChartObjects gyűjteményével (pl. Worksheets(1).ChartObjects) kezelhetők.

1.3.1.3. A Range objektum

A Range objektum cellatartomány kezelésére használatos. Az alábbiakban felsoroljuk a Range objektum néhány fontosabb tulajdonságát és metódusát, majd mintapéldákkal szemléltetjük ezek használatát.

Tulajdonságok: Address, Areas, Cells, Column, Columns, ColumnWidth, Count, CurrentRegion, Font, Formula, FormulaLocal, FormulaR1C1, FormulaR1C1Local, Height, Name, NumberFormat, NumberFormatLocal, Offset, Range, Resize, Row, RowHeight, Rows, Value, Width.

Metódusok: AutoFill, Clear, ClearFormats, Copy, Delete, Find, PasteSpecial, Select, Sort.

Megjegyzés: A cellatartományra a blokk szó is használatos (mert rövidebb), de a két dolog nem pontosan ugyanazt jelenti. Blokkon ugyanis egy összefüggő, téglalap alakú cellatartományt értünk. Egy cellatartomány állhat több blokkból is, de ez fordítva nem igaz.

Hivatkozás

- Az *A1 stílusú* hivatkozás: a cellatartományt egy sztringben megadott kifejezéssel hivatkozzuk.

Pl. Range(”A1”), Range(”c2”), Range(”A2:B3”), Range(”A:A”), Range(”1:1”), Range(”A1:A5,C1:C5”), Range(”A:A,C:D”).

- Az *R1C1 stílusú* hivatkozás: a cellatartományt a sor- és oszlopindexek segítségével hivatkozzuk.

  Pl. Cells(1,1), Cells(2,3), Range(Cells(2,1), Cells(3,2)), Columns(1), Rows(1).

Kijelölés: Select metódus

Pl.

```
’Az első munkafüzet első munkalapján az A1:B3 blokk kijelölése
Application.Workbooks(1).Worksheets(1).Range("A1:B3").Select
’Az aktuális munkalap A1:B3 blokkjának kijelölése
Range("b3:a1").Select
```

Adatmegadás: Value tulajdonság

Pl.

```
’Az aktuális munkalap celláiba teszünk adatokat
Range("A1").Value = "Maci Laci"          ’Egy szöveg
Range("A2").Value = 2*3                  ’Egy szám
Cells(3,1).Value = Date                  ’Az aktuális dátum
Cells(4,1).Value = CDate("2012.12.24")   ’Egy adott dátum
Cells(4,1).Value = ""                    ’Adat törlése
```

Megjegyzés

- A Value tulajdonság el is hagyható (pl. Range("A2") = 3.14)
- Egy cella üressége az IsEmpty függvénnyel kérdezhető le.

Tartalom törlése: ClearContents metódus

Pl.

```
'Az aktuális munkalap egy blokkjának tartalmát töröljük
Range("A1:A4").ClearContents
```

Formátum törlése: ClearFormats metódus

Pl.

```
'Az aktuális munkalap egy blokkjának formátumát töröljük
Range("A1:A4").ClearFormats
```

A tartalom és a formátum törlése: Clear metódus

Pl.

```
'Az aktuális munkalap egy blokkjának formátumát és tartalmát töröljük
Range("A1:A4").Clear
```

A blokk celláinak törlése: Delete metódus

Pl.

```
'Az aktuális munkalap egy blokkjának celláit töröljük
Range("A1:A4").Delete        'A jobbra lévő cellák balra lépnek
Range("A1:D1").Delete        'A lenti cellák feljebb lépnek
```

Megjegyzés: A Delete metódusnak van egy opcionális paramétere, amellyel megadható, hogy a törölt cellák helyére hogyan lépjenek be (jobbról vagy lentről) a szomszédos cellák. Ha a paramétert nem adjuk meg (mint a példában), akkor ezt a törölt objektum alakja alapján dönti el az Excel.

Formátum beállítása: NumberFormat, NumberFormatLocal tulajdonság

Pl.

```
'Az aktuális munkalap A oszlopára dátum formátumot
Range("A:A").NumberFormat = "yyyy.mm.dd"
'Ez ugyanazt csinálja, mint az előző sor
Columns(1).NumberFormatLocal = "éééé.hh.nn"
'Az aktuális munkalap C és D oszlopára pénznem formátumot
Range("C:D").NumberFormat = "#,##0 $"
```

Az aktuális cellát tartalmazó összefüggő blokk: CurrentRegion tulajdonság

Pl.

```
'Az aktuális munkalap A1-es celláját tartalmazó blokk méretei
s = Range("A1").CurrentRegion.Rows.Count          'Sorok száma
o = Range("A1").CurrentRegion.Columns.Count       'Oszlopok száma
```

Amint azt láttuk, az Excel munkalapokon tárolt adatok a cellák Value tulajdonságával elérhetők, azaz adatokat tudunk a cellákba tenni, és azokat fel tudjuk használni. Ha az adatokból valamilyen eredményadatokat szeretnénk meghatározni, akkor ezt általában nem programozzuk (pl. egy összeg esetén egy összegző algoritmussal), hanem az Excel-t „kérjük meg" az eredmények kiszámítására (pl. használjuk a SZUM függvényt). Ennek két oka is van. Egyrészt az Excel-ben nagyon sok beépített függvény használható, másrészt ez a megoldás követi az adatok esetleges megváltozását (ha a képletek automatikus számítása (Fájl, Beállítások, Képletek, Számítási beállítások, Munkafüzet kiszámítása, Automatikus választógomb) be van kapcsolva, de ez általában bekapcsolt állapotú az Excel-ben). A programmal kiszámolt eredmények „frissüléséhez" ugyanis az eredményt számító forráskódot újra kell futtatni.

Ahhoz, hogy forráskódból olyan Excel képleteket tudjunk az egyes cellákba tenni, amelyek a kívánt feladatot elvégzik (kiszámolják a megfelelő eredményeket a megfelelően hivatkozott cellatartományok adataiból), szükségünk van az Excel cellahivatkozásainak ismeretére.

A cellahivatkozások stílusa az Excel-ben beállítható (lásd Fájl, Beállítások, Képletek, S1O1 hivatkozási stílus jelölőnégyzet). általánosan elterjedt az *A1 stílusú* cellahivatkozás, amely az oszlopokat egy vagy több betűvel, míg a sorokat sorszámokkal jelöli. Beállítható azonban az ún. *S1O1 stílusú* cellahivatkozás is, ahol az oszlopok is sorszámokkal azonosíthatók.

Megjegyzés: Az S1O1 stílusú hivatkozásban az S betű a sort, az O betű az oszlopot jelenti. Az S1O1 cellahivatkozási stílus angol elnevezésében (*R1C1 style*) R jelenti a sort (row), C pedig az oszlopot (column).

A VBA környezetben az Excel cellahivatkozási stílusát az Application objektum ReferenceStyle tulajdonsága tárolja, így a cellahivatkozási stílus lekérdezése, illetve beállítása ezzel a tulajdonsággal történhet.

Pl.

```
'Lekérdezés
If Application.ReferenceStyle = xlR1C1 Then
    MsgBox ("Az Excel S1O1 stílusú cellahivatkozást használ")
Else
    MsgBox ("Az Excel A1 stílusú cellahivatkozást használ")
End If
'Beállítás
Application.ReferenceStyle = xlA1
```

Az S1O1 cellahivatkozási stílusban a cellákra a sorok és oszlopok sorszámával hivatkozunk. A cellahivatkozások ugyanúgy lehetnek abszolút, relatív és vegyes hivatkozások, mint az A1 stílus esetén. A relatív hivatkozásnál a megadott sorszámokat szögletes zárójelbe kell tenni, az abszolút hivatkozásnál nem. A relatív hivatkozás mindig a hivatkozást tartalmazó cellához képest relatív, ahhoz viszonyítva értendő. Ha a hivatkozást tartalmazó cella sorát, illetve oszlopát szeretnénk hivatkozni, akkor nem kell sorszámot megadnunk.

Az alábbiakban az S1O1 cellahivatkozási stílusokra adunk példákat (ahol a ~ karakter után megadjuk az S1O1 stílusú cellahivatkozás A1 stílusú megfelelőjét).

• Abszolút hivatkozás (pl. S1O1 ~ $A$1; S12O3 ~ $C$12).

• Relatív hivatkozás (pl. SO[1] ~ ugyanaz a sor, az oszlop az eggyel jobbra lévő; S[-1]O ~ a sor az eggyel feljebb lévő, az oszlop ugyanaz).

• Vegyes hivatkozás (pl. S2O[1] ~ abszolút sor (a második), relatív oszlop (az eggyel jobbra lévő); SO1 ~ relatív sor (ugyanaz a sor), abszolút oszlop (az első)).

Egy cellatartomány hivatkozásának lekérdezése: Address tulajdonság

Az Address tulajdonság használatának (egyszerűsített) szintaktikája:

expression.Address(*RowAbsolute*, *ColumnAbsolute*, *ReferenceStyle*, *RelativeTo*)

RowAbsolute	Az eredményhivatkozás sor abszolút legyen-e (alapértelmezésben True).
ColumnAbsolute	Az eredményhivatkozás oszlop abszolút legyen-e (alapértelmezésben True).
ReferenceStyle	Az eredményhivatkozás stílusa (xlA1 vagy xlR1C1, alapértelmezésben xlA1).
RelativeTo	Az R1C1 típusú eredményhivatkozás viszonyítási Range objektuma.

Pl.

```
'Az aktuális munkalap egy cellájának hivatkozása A1 stílusban
MsgBox Cells(2,3).Address                    '$C$2
MsgBox Cells(2,3).Address(True, False)       'C$2
MsgBox Cells(2,3).Address(False, True)       '$C2
MsgBox Cells(2,3).Address(False, False)      'C2
'Az aktuális munkalap egy cellájának címe R1C1 stílusban
MsgBox Cells(2,3).Address(ReferenceStyle:=xlR1C1)     'R2C3
MsgBox Cells(2,3).Address(ReferenceStyle:=xlR1C1, _
        RowAbsolute:=False, ColumnAbsolute:=False, _
```

```
RelativeTo:=Cells(1,1))                    'R[1]C[2]
```

Megjegyzés: Ha egy utasítást több sorba írunk, akkor az alulvonás karaktert (_) kell azon sorok végére tenni, amelyek még folytatódnak (lásd a fenti példában).

Képletek használata: Formula, FormulaLocal tulajdonságok

Mindkét tulajdonság képlet megadására használatos. A képletet egy sztringkifejezéssel definiálhatjuk, lényegében ugyanúgy, mint ahogy azt az Excel-ben is megadnánk.

A Formula tulajdonságban az Excel függvényeire az angol nevükkel kell hivatkozni (az Excel súgója tartalmazza a függvények angol megnevezését és szintaxisát is), a FormulaLocal tulajdonság esetén pedig azon a nyelven, amit az Excel-ben is használunk.

A képletek könnyen megadhatók abban az esetben, ha tudjuk, hogy melyik cellába akarjuk a képletet elhelyezni, és ha az alkalmazni kívánt képletben ismerjük az esetlegesen hivatkozott cellatartományt. Nehezebb a dolgunk, ha nem tudjuk a képletet tartalmazó cella címét akkor, amikor a forráskódot írjuk (mert pl. ez függ a munkalapon lévő adatsorok, adatoszlopok számától).

A következőkben mindkét esetre mutatunk példát. Az első két utasítás ismert célcellába ismert cellatartományra hivatkozó képletet tesz. A másik két képlet elhelyezésekor nem tudjuk az adatokat tartalmazó blokk méretét, csak azt, hogy az A1-es cellától kezdődik az az összefüggő blokk, amelyre ki akarunk számolni valamit (példánkban átlagot és összeget számolunk). A CurrentRegion tulajdonság segítségével megtudható az adatokat tartalmazó blokk mérete. Az Address tulajdonság segítségével megkaphatók azok a (relatív) cellacímek, amelyek a képletekhez (az első oszlop átlagához, és az első sor összegéhez) kellenek.

Pl.

```
'Az aktuális munkalapon fix képletet fix helyre
Range("C5").Formula = "=SUM(C2:C4)"
Range("D5").FormulaLocal = "=SZUM(D2:D4)"
'Az A1-es cellát tartalmazó blokk alá az első oszlopba
s = Range("A1").CurrentRegion.Rows.Count
```

```
st = Cells(s, 1).Address(False, False)
Cells(s + 1, 1).FormulaLocal = "=ÁTLAG(A1:" + st + ")"
'Az A1-es cellát tartalmazó blokk mellé az első sorba
o = Range("A1").CurrentRegion.Columns.Count
st = Cells(1, o).Address(False, False)
Cells(1, o + 1).FormulaLocal = "=SZUM(A1:" + st + ")"
```

Képletek használata: FormulaR1C1, FormulaR1C1Local tulajdonságok

Ez a két tulajdonság olyan képletek megadására használható, amelyekben a cellatartományokra az R1C1 cellahivatkozási stílussal hivatkozunk. A következő példák mindegyike a C2-es cellába helyez el egy képletet, amely egy cella tartalmának kétszeresét számolja ki. A cella, amire hivatkozunk, a hivatkozástól függően változik. A C2-es cellába kerülő A1 hivatkozási stílusú képletet a sorok végén található megjegyzésekben láthatjuk.

Pl.

```
'Egy egyszerű képletet a C2-es cellába
Range("C2").FormulaR1C1 = "=R1C1*2"              ' "=$A$1*2"
Range("C2").FormulaR1C1 = "=RC1*2"               ' "=$A2*2"
Range("C2").FormulaR1C1 = "=R1C*2"               ' "=C$1*2"
Range("C2").FormulaR1C1 = "=R[1]C[-1]*2"         ' "=B3*2"
'A FormulaR1C1Local tulajdonságban az S és O betűk szerepelnek
Range("C2").FormulaR1C1Local = "=S1O1*2"         ' "=$A$1*2"
Range("C2").FormulaR1C1Local = "=S[1]O[-1]*2"    ' "=B3*2"
```

Megjegyzés

• A példákban most nem használtuk az Address tulajdonságot, de a hivatkozott cella R1C1 stílusú címe ezzel is előállítható lett volna.

• Az Excel-ben a függvénynevek magyarul (helyi (local) nyelven) jelennek meg (akkor is, ha a Formula, illetve FormulaR1C1 tulajdonságokkal definiáljuk őket).

• A képleteket kezelő tulajdonságok (Formula, FormulaLocal, FormulaR1C1, FormulaR1C1Local) egyikének megadásával mindegyik definiálódik.

Pl.

```
Range("C2").FormulaLocal = "=ÁTLAG(A2:B2)"
MsgBox Range("C2").FormulaR1C1 ' =AVERAGE(RC[-2]:RC[-1]
```

Képletek másolása: Copy, AutoFill, PasteSpecial metódusok

A képleteket tartalmazó cellákat gyakran másoljuk, hogy ne kelljen azokat többször megadni. A másolás az Excel-ben többféle módon is elvégezhető (pl. a vágóasztal segítségével, a másolandó cella jobb alsó sarkánál lévő kitöltőjel segítségével), így ezt forráskódból is többféleképpen végezhetjük.

Pl.

```
'Az s+1. sor 1. oszlopában lévő képlet másolása az s+1. sor
'oszlopaiba a 2. oszloptól az o. oszlopig
Cells(s + 1, 1).Copy Destination:= _
    Range(Cells(s + 1, 2), Cells(s + 1, o))
'Az 1. sor o+1. oszlopában lévő képlet másolása az o+1. oszlop
'soraiba a 2. sortól az s. sorig
'AutoFill esetén a céltartománynak a forrást is tartalmaznia kell
Cells(1, o + 1).AutoFill Destination:= _
```

```
        Range(Cells(1, o + 1), Cells(s, o + 1))
'Mint az előző, csak másolással és beillesztéssel
Cells(1, o + 1).Copy
'Beillesztés
Range(Cells(2, o + 1), Cells(s, o + 1)).PasteSpecial
'A forrástartományt jelölő (villogó) szegély levétele
Application.CutCopyMode = False
```

Egyéb lehetőségek: Rows, Columns, Cells, Range, Offset, Resize

A Range objektum fenti tulajdonságai Range objektumot adnak eredményül. Az első négy tulajdonság jelentése és használata értelemszerű; az Offset tulajdonsággal egy, az adott Range objektumhoz képest relatív elmozdulással nyert Range objektum hivatkozható; a Resize tulajdonsággal pedig átméretezhető egy adott blokk. A példákban az egyes tulajdonságok eredményeként kapott Range objektumoknak a Select metódusát hívjuk meg (amellyel látható lesz az eredmény). Az utolsó példa az A1-es cella egy feleslegesen cifra hivatkozását szemlélteti.

Pl.

```
'Egy Range objektum egy sora, illetve egy oszlopa
Range("C3:D6").Rows(2).Select
Range("C3:D6").Columns(2).Select
'Egy Range objektum egy cellája
Range("C3:D6").Cells(1, 2).Select
'Ugyanazt jelöli ki, mint az előző
Range(Cells(3, 3), Cells(6, 4)).Range("B1").Select
'Egy Range objektum egy Range objektuma
```

```
Range("C3:D6").Range("A2:B2").Select
'Egy Range objektumból relatív elmozdulással nyert Range objektum
Range("C3:D6").Offset(-1, 1).Select
'Egy Range objektum átméretezése
Cells(2,2).Resize(3,2).Select
'Range objektumok egyesítése
Application.Union(Columns(1), Columns(4)).Select
'Az A1-es cella egy feleslegesen cifra hivatkozása
Range("A1").Cells(1,1).Offset(0,0).Resize(1,1).Cells(1).Select
```

1.3.1.4. A WorksheetFunction objektum

Az előző fejezetben láttuk, hogyan helyezhetünk el képleteket az egyes cellákba. Ezzel a megoldással az Excel végzi a számolásokat, és jeleníti meg az eredményeket a képleteket tartalmazó cellákban. Ennél a megoldásnál tehát bizonyos cellák tartalma módosul. Előfordulhat azonban olyan eset, amikor ez a megoldás nehezebben realizálható (pl. ha az adatokat tartalmazó munkalap védett, akkor egy másik (nem védett, akár egy másik munkafüzethez tartozó) munkalap szükséges a megoldáshoz).

Az Excel munkalapfüggvényei azonban nemcsak a cellákban elhelyezett képletekben használhatók. A forráskódból meghívható munkalapfüggvényeket a WorkSheetFunction tároló (container) objektum foglalja egy logikai egységbe. Az objektum függvénymetódusai az egyes munkalapfüggvények, amelyek paraméterei egyaránt lehetnek munkalapok Range objektumai, és memóriaváltozók (pl. egy tömbváltozó).

A függvénymetódusok hasonlóan hívhatók, mint a VB függvényei (a hívás bárhol állhat, ahol az eredmény típusának megfelelő érték állhat), így az eredmény megkapható a cellák módosítása nélkül is (igaz, az adatok módosulása esetén a számolást végző forráskódot újra kell futtatni).

A forráskódban az objektumtípusok hasonlóan használhatók, mint a VB többi adattípusa. Az objektumok értékadó utasítása a Set (nem opcionális) kulcsszóval kezdődik (szemben az eddig használt értékadás Let kulcsszavával, ami elhagyható volt).

Az objektumok értékadó utasításának (egyszerűsített) szintaktikája:

Set *objectvar* = *objectexpression*

Az értéket kapó objektumváltozónak (*objectvar*) ugyanolyan típusúnak kell lennie, mint az objektumkifejezés (*objectexpression*) eredményeként előálló objektumnak.

Az alábbi példa a WorksheetFunction objektum használata mellett az objektumok értékadását is bemutatja.

Pl.

```
'Munkalapfüggvény használata cellatartományra
Sub MunkalapFgv1()
Dim r As Range, min As Variant
Set r = Application.Worksheets("Munka1").Range("A1:C5")
min = Application.WorkSheetFunction.Min(r)
MsgBox min
End Sub
'Munkalapfüggvény használata változókra
Sub MunkalapFgv2()
Dim a(1 To 10) As Integer, i As Integer
For i = 1 To 10: a(i) = i: Next
MsgBox WorkSheetFunction.Sum(a)        '55
End Sub
```

Megjegyzés

• Az egyes munkalapokra kérhető olyan beállítás is, amely a képleteket tartalmazó cellákban magát a képletet és nem az eredmény értékét jeleníti meg (lásd Fájl, Beállítások, Speciális, Beállítások megjelenítése ehhez a munkalaphoz, Számított eredmények helyett képletek megjelenítése a cellákban jelölőnégyzet).

• Általában akkor használunk külön objektumváltozót (a példában r), ha az adott objektummal több dolgot is el szeretnénk végezni. A példában csak a Min munkalapfüggvényt hívjuk meg egy adott blokkra, ami az objektumváltozó nélkül is megtehető lett volna.

• A példában szereplő min változó Variant típusú, mert az eredmény típusa a megfelelő cellákban (A1:A5) található adatok típusától függ.

Önellenőrzés

1. Az alábbi állítások közül melyek igazak az Excel VBA környezetben használható objektumokkal és objektumtípusokkal kapcsolatosan?
   - ☐ Az osztályok egységesen használhatók, függetlenül attól, hogy melyik rendszerfájlban találhatók.
   - ☐ A gyűjtemények azonos típusú objektumok összessége.
   - ☐ Az olyan gyűjteményekben, amelyben az elemeknek van Name tulajdonsága, az elemek a nevükkel is kiválaszthatók.
   - ☐ A WorkSheets gyűjtemény magában foglalja a Sheets gyűjteményt.
   - ☐ A Sheets gyűjtemény magában foglalja a Charts gyűjteményt.
   - ☐ A Charts gyűjtemény magában foglalja a ChartObjects gyűjteményt.
   - ☐ A Range szó osztályt is és tulajdonságot is jelöl.
2. Az alábbi állítások közül melyek igazak a Range objektummal kapcsolatosan?
   - ☐ Az A1 stílusú hivatkozásban a cellatartományt a sor- és oszlopindexekkel hivatkozzuk.
   - ☐ A Range objektumnak létezik Range tulajdonsága, amely egy Range objektumot ad eredményül.
   - ☐ A Cells gyűjtemény egy eleme egy index segítségével is hivatkozható.
   - ☐ A Cells gyűjtemény egy eleme Range típusú objektum.
   - ☐ A Formula tulajdonság megadásakor az Excel függvények angol neveit kell használnunk.
3. A súgó használata
   Indítsuk el a Visual Basic Editor-ban található súgót, majd az Excel Object Model Reference témakörben nézzük meg következő objektumokhoz tartozó oldalakat: Application, Workbook, Worksheet, Range, Chart. Az objektumok tulajdonságaihoz (Properties) és metódusaihoz (Methods) tartozó oldalakba is nézzünk bele! A következő gyűjtemények súgóoldalait is nézzük meg: Workbooks, Worksheets, Sheets, Charts!

4. Az Application objektum
Az Application objektum segítségével jelenítsük meg a következő adatokat: a nyitott munkafüzetek száma, az aktuális munkafüzet munkalapjainak száma, az első munkalap neve, az aktuális munkafüzet neve; az aktuális munkalap neve. Használjuk az Immediate ablakot, illetve készítsünk szubrutint a megoldásra!

5. A Range objektum
Range objektumok segítségével végezzük el az alábbi feladatokat! Az aktuális munkafüzet aktuális munkalapján dolgozzunk! Hasonlóan az előző feladathoz, próbáljuk ki az utasítások közvetlen végrehajtásával (Immediate ablak), illetve a szubrutinnal történő megoldást is! Az adatok megjelenítéséhez a MsgBox függvényt vagy a Debug.Print metódust használjuk!

- Jelöljük ki a következő cellatartományokat mind a két hivatkozási stílus (A1, illetve R1C1) használatával! A C5-ös cella; az 5. sor 3. oszlopbeli cella; a C3:D5 blokk; a 2. sor 3. oszlop bal felső sarkú és az 5. sor 6. oszlop jobb alsó sarkú blokk; az első sor; a 35. oszlop; a 3. sor; az A és a D oszlop.
- Tegyünk a cellákba: számot; szöveget; dátumot; az aktuális dátumot; különböző kifejezések eredményét, majd jelenítsük meg a cellák értékét!
- Állítsunk be: egy tetszés szerinti dátumformát; pénznem formátumot; százalékos megjelenítést; általános formátumot, illetve jelenítsünk meg cellaformátumokat!
- Töröljünk adatot, formátumot, illetve mindkettőt megadott cellatartományokban! Próbáljuk ki a Range objektum Delete metódusát is!
- Jelenítsük meg különböző cellatartományok Address tulajdonságát! Nézzük meg az A1 stílusú hivatkozás mind a négy lehetséges esetét!
- Helyezzünk el képleteket egy-egy cellába az alábbiak szerint! Jelenítsük meg egy képletet tartalmazó cella képletét (az összes képlet tulajdonság kipróbálásával), illetve a cellában lévő eredményértéket!
  - Egy adott cellába egy olyan képletet, amely: nem használ cellahivatkozást; használ cellahivatkozást; munkalapfüggvényt használ.
  - Az A1-es cellát tartalmazó összefüggő blokk első sorának legkisebb elemét a blokk utáni oszlop első sorába.

– Az A1-es cellát tartalmazó összefüggő blokk első oszlopának legnagyobb elemét a blokk utáni sor első oszlopába.

• Az előző feladatban (dinamikusan) elhelyezett képleteket másoljuk végig a képlet sorában, illetve oszlopában úgy, hogy az A1-es cellát tartalmazó összefüggő blokk összes sorára megkapjuk a sorminimumokat, illetve az összes oszlopára megkapjuk az oszlopmaximokat!

6. Munkalapfüggvények

Számoljuk ki az Application objektum WorksheetFunction tulajdonságának segítségével egy adott cellatartomány adataira a következőket, és az eredményeket jelenítsük meg!

• Elemek átlaga; elemek összege; legkisebb elem; legnagyobb elem.

6. LECKE

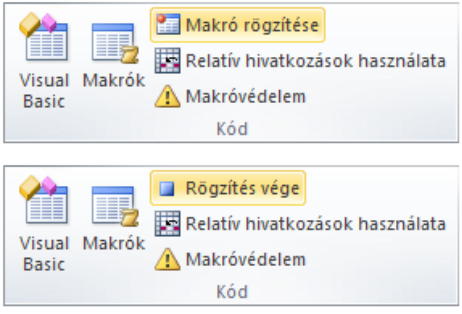
Egyéb VBA lehetőségek

1.3.2. Egyéb VBA lehetőségek

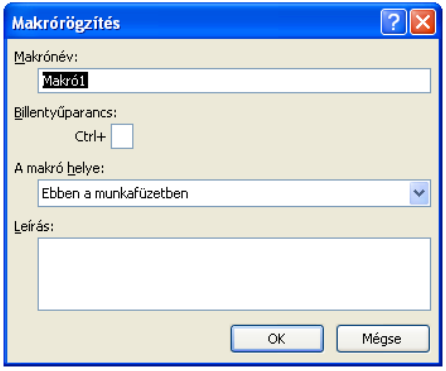
Ebben a fejezetben olyan témákkal foglalkozunk, amelyek talán nem alapvető fontosságúak az Excel programozását illetően, de elég hasznosak ahhoz, hogy ha röviden is, de érintsük őket.

1.3.2.1. Makrók rögzítése

Az Excel képes arra, hogy a felhasználó által elvégzett tevékenységeket rögzítse, azaz forráskód szinten megadja azokat a VBA utasításokat, amelyekkel az elvégzett tevékenységek végrehajthatók. A Makró rögzítése funkció az Excel menüszalagján a Fejlesztő eszközök lap Kód csoportjában található (ott, ahol a Visual Basic Editor program ikonja is szerepel, lásd 1.36. ábra).



1.36. ábra. A Makró rögzítése és a Rögzítés vége funkciók a menüszalag Fejlesztőeszközök lapján



1.37. ábra. A rögzítendő makró adatainak megadására szolgáló párbeszédablak

A Makró rögzítése funkció egy párbeszédablakot jelenít meg (lásd 1.37. ábra), ahol megadhatók a rögzítendő makró adatai (név, gyorsbillentyű, hely, leírás).

A rögzített makrók egy új modulba kerülnek, a forráskódok tetszőlegesen felhasználhatók (szerkeszthetők, futtathatók, stb.). Az alábbi példa egy olyan rögzített makró forráskódját tartalmazza, amelyben egy cella tartalmát töröltük.

Pl.

```
Sub Makró1()
'
' Makró1 Makró
'
    Range("B4").Select
    Selection.ClearContents
End Sub
```

1.3.2.2. Diagramok kezelése

Az Excel-ben könnyen készíthetünk adatainkból szemléletes diagramokat. A diagramok kezelése (létrehozás, módosítás, törlés, stb.) VBA utasításokkal is elvégezhető.

A diagramkészítés tipikus lépéseit egy példa segítségével szemléltetjük. A diagramot az első munkalap A1-es celláját tartalmazó összefüggő blokk adataiból készítjük, és az első munkalapon helyezzük el. Először a Charts gyűjtemény Add metódusával létrehozunk egy új (még üres) diagramot, majd megadjuk a diagram legfontosabb adatait (típus, forrásadatok, célhely), végezetül pedig (hogy ne a diagram legyen az aktuálisan kijelölt objektum), az első munkalap A1-es celláját aktivizáljuk.

Pl.

```
'Diagramkészítés
Sub Diagram()
Dim r As Range
'Az A1-es cellát tartalmazó összefüggő blokk adataiból...
Set r = Worksheets(1).Range("A1").CurrentRegion
'Egy új, üres diagram létrehozása
Charts.Add
'A diagram testreszabása
With ActiveChart
    'A diagram típusa
    .ChartType = xlColumnClustered
    'A diagram forrásadatai és az adatsorozatok képzése (sorokból)
    .SetSourceData Source:=r, PlotBy:=xlRows
    'A diagram elhelyezése: objektumként az első munkalapra
```

```
        .Location Where:=xlLocationAsObject, Name:=Worksheets(1).Name
End With
Worksheets(1).Activate          'Az első munkalap aktivizálása
Range("A1").Select              'Az A1-es cella kijelölése
End Sub
```

A következő példa törli az önálló lapon, valamint az első munkalapon lévő diagramokat. Az önálló lapon lévő diagramok törlésekor (mivel teljes lapok törlődnek), az Excel egy figyelmeztető üzenetet ad, és a törlés csak jóváhagyás után történik meg. Ennek a figyelmeztető üzenetnek a megjelenítését kapcsoljuk ki az Application objektum DisplayAlerts tulajdonságával.

Pl.

```
'Diagramok törlése
Sub DiagramTorles()
'Az önálló lapon lévőket
If Charts.Count > 0 Then
        Application.DisplayAlerts = False    'Ne legyen figyelmeztető üzenet
        Charts.Delete
        Application.DisplayAlerts = True     'A figyelmeztetés visszakapcsolása
End If
'Az első munkalapon lévőket
If Worksheets(1).ChartObjects.Count > 0 Then
        Worksheets(1).ChartObjects.Delete
End If
End Sub
```

Megjegyzés

• A diagramokat tartalmazó gyűjtemények (Charts, ChartObjects) Delete metódusa futási hibát ad akkor, ha a gyűjteménynek nincs egyetlen eleme sem, ezért a törlést csak akkor végezzük el, ha van mit törölni.

• Az Application objektum DisplayAlerts tulajdonsága az összes figyelmeztető üzenet megjelenítését szabályozza (pl. egy nem mentett munkafüzet bezárásakor megjelenő üzenet sem jelenik meg, ha ez a kapcsoló ki van kapcsolva (False értékű), ezért a törlés után visszakapcsoljuk (True értékűre)).

1.3.2.3. A beépített párbeszédablakok használata

Az Excel sokféle párbeszédablakot használ, amelyek a VBA környezetből is elérhetők. Az Application objektum Dialogs gyűjteménye ezeket a párbeszédablakokat (Dialog objektumokat) foglalja egy logikai egységbe.

A forráskódból használni kívánt párbeszédablak a Dialogs gyűjtemény egy elemének a kiválasztásával hivatkozható. A kiválasztás egy XlBuiltInDialog felsorolt típus egy elemének a megadásával történhet (a példában a fájlmegnyitó párbeszédablakot hivatkozzuk az xlDialogOpen elemmel).

A párbeszédablakok Show metódusa megjeleníti a párbeszédablakot, ahol a felhasználó megadhatja a megfelelő adatokat. A metódus eredményül egy logikai értéket ad attól függően, hogy a felhasználó melyik nyomógombbal zárta be az ablakot. Az OK gomb esetén True az eredmény, a Mégse (Cancel) gomb (illetve az ablak bezárása) esetén False.

Az alábbi példa a fájlmegnyitó párbeszédablak segítségével megadott (Excel-ben megnyitható típusú) fájl teljes elérési útját írja ki a képernyőre. Az OK gomb választása esetén a Show metódus meg is nyitja a kiválasztott fájlt, így ez lesz az aktuális munkafüzet. A nyitott munkafüzetek (így az aktuális munkafüzet) teljes elérési útja a FullName tulajdonsággal érhető el. Ennek megjegyzése után bezárjuk a megnyitott fájlt, így újra a makrókat tartalmazó munkafüzet lesz az aktuális (aktív) munkafüzet.

Pl.

```
'Fájlválasztás párbeszédablakkal
Sub FajlValasztas()
Dim s As String
```

```
s = ""
If Application.Dialogs(xlDialogOpen).Show Then
    'A Megnyitás gombot nyomták meg, megjegyezzük a fájl specifikációját
    s = ActiveWorkbook.FullName
    'Lezárjuk a megnyitott munkafüzetet
    ActiveWorkbook.Close
End If
If s = "" Then
    MsgBox "Nem választottak fájlt!"
Else
    MsgBox "A kiválasztott fájl:" + vbCrLf + s
End If
End Sub
```

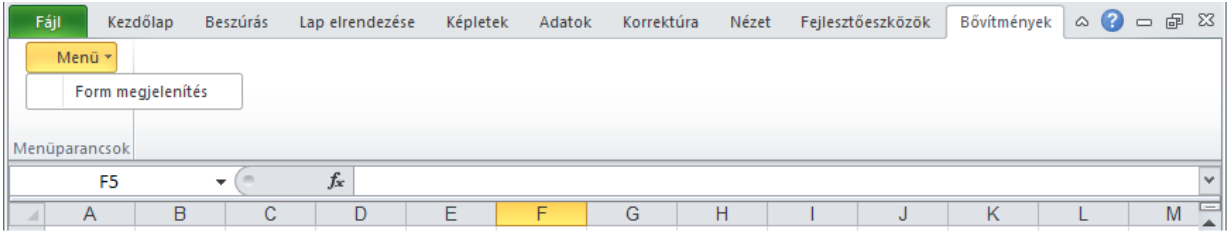
Megjegyzés

- A Dialogs gyűjtemény csak olvasható (read-only) (így pl. nincs Add metódusa sem).
- Ha a példában kiválasztunk egy fájlt, akkor egy kétsoros üzenetet kapunk, ahol a kiválasztott fájl teljes specifikációja (meghajtó, elérési út, fájlazonosító) a második sorba kerül. A sortörést a vbCrLf konstanssal végeztük (Chr(13)+Chr(10)).

1.3.2.4. Saját menü készítése

Ha az Excel funkcióit saját fejlesztésű makrókkal bővítjük, akkor a makróink elérhetőségét, futtathatóságát egyszerű, a felhasználók által könnyen kezelhető módon kell biztosítanunk. Eddig csak olyan futtatási lehetőségekről volt szó, amelyhez a Visual Basic Editor-t is használni kellett. Ezt azonban egy átlagos Excel felhasználótól nem várhatjuk el.

Szerencsére megvannak azok az eszközök, amellyel ez a probléma megoldható. Egyfelől az Excel menüje VBA eszközökkel kezelhető, így pl. forráskódból bővíteni tudjuk az Excel menüjét a makróink futtatását végző funkciókkal. Másfelől a munkafüzetekhez is tartoznak eseménykezelők, amelyek automatikusan lefutnak a munkafüzethez kapcsolódó események (pl. megnyitás, mentés, bezárás) bekövetkezésekor.



1.38. ábra. A saját menü megjelenése a menüszalag Bővítmények lapján

A makróinkat tartalmazó munkafüzet megnyitásakor automatikusan (külön indítás nélkül) lefuttathatjuk az Excel menüjét bővítő makrónkat, így a többi makró (az általunk készített funkciók) már az Excel menüjéből indítható. A makróinkat tartalmazó dokumentum bezárásakor pedig visszaállíthatjuk a menü eredeti állapotát.

Az alábbi példa ezt szemlélteti. A MenuKirak szubrutin bővíti az Excel menüjét egy új (Menü feliratú) menüponttal. Ez a menüpont az Excel menüszalagján a Bővítmények lapon fog megjelenni (lásd 1.38. ábra). A menün belül egyetlen almenüpontot definiálunk (Form megjelenítés felirattal), ami egy formot (UserForm1) jelenít meg. A MenuLevesz szubrutin törli a MenuKirak által (akár több példányban) létrehozott Menü feliratú menüpontot.

Pl.

```
'Saját menü létrehozás
Sub MenuKirak()
Dim fomenu As CommandBar
Dim fomenupont As CommandBarControl, almenupont As CommandBarControl
```

```
Set fomenu = Application.CommandBars.ActiveMenuBar
'Egy új (felbukkanó) menüpont létrehozása
Set fomenupont = fomenu.Controls.Add(Type:=msoControlPopup)
fomenupont.Caption = "Menü"
'Az új főmenüponthoz egy új almenüpontot
Set almenupont = fomenupont.CommandBar.Controls.Add(Type:=msoControlButton)
almenupont.Caption = "Form megjelenítés"
'A menüpont aktivizálásakor lefutó szubrutin
almenupont.OnAction = "FormKirak"
End Sub
Sub FormKirak()
UserForm1.Show
End Sub
'Saját menü levétele
Sub MenuLevesz()
Dim menupont As CommandBarControl
For Each menupont In Application.CommandBars.ActiveMenuBar.Controls
        If menupont.Caption = "Menü" Then menupont.Delete
        End If
Next
End Sub
'Az eredeti rendszermenü visszaállítása (ha elrontanánk a menüt)
```

```
Sub MenuAlaphelyzet()
Application.CommandBars("Worksheet Menu Bar").Reset
End Sub
```

A fenti szubrutinokat (menükezelés, a menüpontok aktivizálásakor végrehajtandó szubrutinok) egy modulban helyezendők el (így elérhetők, futtathatók lesznek), míg az alábbi eseménykezelő szubrutinokat a ThisWorkbook objektum moduljában kell elhelyezni.

Pl.

```
'A munkafüzet megnyitásakor aktivizálódik
Private Sub Workbook_Open()
'A saját menü kirakása
Call MenuKirak
End Sub
'A munkafüzet lezárásakor aktivizálódik
Private Sub Workbook_BeforeClose(Cancel As Boolean)
'A saját menü levétele
Call MenuLevesz
End Sub
```

Megjegyzés

• A MenuAlaphelyzet szubrutint azért készítettük, hogy ha elrontanánk az Excel menüjét, akkor gyorsan visszaállítható legyen a menü alapállapota. A menüszalag alaphelyzetbe állítása ugyan az Excel beállításai között is megtalálható (lásd Fájl, Beállítások, Menüszalag testreszabása, Alaphelyzet), de ez nem érinti a forráskódból megtett változtatásokat.

• A példában csak kétféle típusú vezérlőt használtunk (msoControlPopup, msoControlButton), de másféle (pl. legördülő listát megvalósító) vezérlő is elhelyezhető a menüben.

1.3.3. Mintafeladat

A fejezetet (s egyben az egész témakört) a következő, összetettebb feladat megoldásával zárjuk. Igaz, hogy az Excel-t rendszerint adatfeldolgozási feladatokra használjuk, így a programozási feladatok zöme is ilyen jellegű, most mégis (kicsit kedvcsináló célzattal) egy másfajta feladatot választottunk. Bizonyára mindenki ismeri a Torpedó nevű játékot, amelyben két személy próbálja meg mielőbb „kilőni" az ellenfél által elrejtett alakzatokat.

A játéktér egy-egy (pl. 10×10-es) négyzetrács (egy az elrejtett, egy pedig a kilövendő alakzatokhoz), amelyben az oszlopokat angol betűkkel (pl. A-tól J-ig), a sorokat sorszámokkal (pl. 1-től 10-ig) azonosítjuk. Az egyes négyzeteket (ahová elrejteni, illetve lőni lehet) az oszlop- és sorazonosítók határozzák meg (pl. A1, C4). Az előre rögzített számú és formájú alakzat elrejtése után a két játékos felváltva lő, és a játékot az nyeri, aki előbb lövi ki (süllyeszti el) az ellenfél alakzatait.

A játék eredeti formában történő megvalósítása (két játékos, több négyzetből álló alakzatok) „túl nagy falat" lenne, ezért egy egyszerűsített változatot definiálunk és oldunk meg. Ez az egyszerűbb feladat is alkalmas lesz az eddig tanult ismeretek összefoglalására, és talán tartogat még annyi kihívást, illetve játékélményt, hogy megérje akár önállóan megoldani, akár „összerakni" és kipróbálni az általunk adott megoldást.

Feladat: Egy párbeszédablak segítségével valósítsuk meg az alábbi *Torpedó* játékot!

• A játék pályaméretét egy *Pályaméret* feliratú legördülő listával lehessen megadni, ami 2-től 6-ig egy egész szám, kezdőértéke legyen 4! A legördülő lista feltöltését ciklussal végezzük!

• A *pálya* az első munkalap A1-es cellájától kezdődő *pályaméret*pályaméret* méretű blokkja, míg az elrejtett alakzatok tárolására a második munkalap hasonló blokkját használjuk *(háttér)!* A játék során az első munkalapot lássuk!

• Egy *Elrejt* feliratú nyomógomb megnyomására helyezzünk el a *háttéren* véletlenszerűen annyi alakzatot, amennyi a *pályaméret!* Az alakzatok egy cellából állnak, jelzésükre az „O" betűt használjuk!

• Egy *Játék indít* feliratú nyomógomb megnyomására (ha már rejtettünk el alakzatokat) kezdjük el a játékot!

A gomb felirata legyen *Játék leállít,* és amíg a játéknak nincs vége, addig ne lehessen pályaméretet választani és alakzatokat elrejteni, viszont lehessen az elrejtett alakzatokra lőni! Egy elkezdett játék a *Játék leállít* gomb megnyomásáig, vagy az összes elrejtett alakzat kilövéséig tart.

• A lövés célcellájának címe egy beviteli mezőben legyen megadható! A lövést egy *Lövés* feliratú (az Enter billentyűre aktív) nyomógombbal lehessen leadni! Csak pályára eső célhelyre lehessen lőni!

• Értékeljük ki a lövést úgy, hogy ha olyan helyre lőttünk, ahová már lőttünk, akkor erről tájékoztassunk, ha eltaláltunk egy alakzatot (a másik munkalapon elrejtett alakzatok közül), akkor azt jelezzük a *pálya* megfelelő cellájában elhelyezett „X” (talált) vagy „*” (nem talált) karakterrel!

• Egy *Súgó* feliratú nyomógomb megnyomására adjunk „súgót” az elrejtett alakzatokról, azaz jelenítsük meg a második munkalapot, hogy az elrejtett alakzatok helyét megnézhessük! Amíg a „súgót” látjuk, addig ne lehessen lövést leadni, valamint a súgó gomb felirata legyen *Bezár!* A *Bezár* gomb megnyomására állítsuk vissza a játék állapotot (azaz a *pályát* lássuk, *Súgó* gombfelirat, lövés engedélyezése)!

• Az ablakban jelenjen meg a kilőtt alakzatok száma! Ha az összes alakzatot kilőttük, akkor a játéknak vége, erről tájékoztassunk, majd kezdhessünk új játékot!

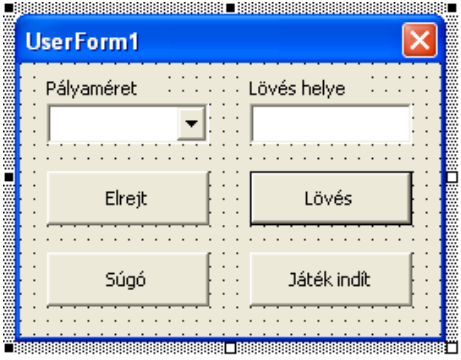
Megoldás: A feladat megoldását az egyes részfeladatokhoz illeszkedően, több lépésben végezzük.

1. lépés: A játék párbeszédablakának (formjának) elkészítése.

A form létrehozása után elhelyezzük a megfelelő vezérlőket a formon: egy legördülő listát (ComboBox), egy beviteli mezőt (TextBox), két címkét (Label), és négy darab nyomógombot (CommandButton). A két címkevezérlő a legördülő lista és a beviteli mező feladatáról tájékoztat, ezért feliratukat (Caption) *Pályaméret,* illetve *Lövés helye* értékekre állítjuk.

A forráskód könnyebb olvashatósága érdekében a vezérlőknek új nevet (Name) adunk. A legördülő lista nevét *cbPalya,* a beviteli mező nevét *tbLoves,* a négy nyomógomb nevét pedig *btnElrejt, btnLoves, btnSugo, btnJatek* értékekre állítjuk. A nyomógombok feliratait is megadjuk: *Elrejt, Lövés, Súgó, Játék indít*.

Ellenőrizzük a vezérlők (Tab Order) sorrendjét, és ha nem megfelelő, beállítjuk az elrendezéshez igazodóan.



1.39. ábra. A játék formja tervezéskor

Megjegyzés

• A vezérlők neveit a típusukra utaló rövidítésből és a funkciójukra utaló névből állítottuk össze (ami egyfajta programozási konvenció).

• A két címkevezérlőnek és magának a formnak nem adtunk külön nevet, mert nem hivatkozunk ezekre a forráskódból. Ha egy több formos alkalmazást készítünk, akkor a formnak is célszerű beszédesebb nevet adni.

• A pályaméret legördülő lista tartalmát (2-től 6-ig, egész számok) tervezéskor nem tudjuk beállítani, ezért a lista feltöltését futáskor (a kezdőtevékenységek között) kell majd elvégeznünk.

• A legördülő lista egy tulajdonsága (Style) azt szabályozza, hogy meg lehet-e adni új adatot (fmStyleDropDownCombo), vagy csak a legördülő listában szereplő elemekből lehet választani (fmStyleDropDownList). A feladathoz az utóbbi illeszkedik, ezért ezt fogjuk használni (amit szintén futási időben állítunk majd be).

• A párbeszédablakokban (a gyorsabb kezelés érdekében) gyakran használunk az Enter, illetve az Esc billentyű leütésére aktivizálódó nyomógombokat. Ehhez a megfelelő nyomógombok Default, illetve Cancel tulajdonságát kell igazra (True) állítani. A példánkban a Lövés feliratú gomb Default tulajdonságát állítottuk igazra, így a

lövés célcellájának megadása után elegendő leütnünk az Enter billentyűt a lövés végrehajtásához (amit a Lövés feliratú nyomógomb végez). A Default nyomógomb szélesebb kerettel emelődik ki mind tervezéskor (lásd 1.39. ábra), mind futáskor (lásd 1.40. ábra).

2. lépés: Modulszintű utasítások megadása.

Az alábbi modulszintű utasítások a form moduljának elején helyezendők el. Használjuk a változók kötelező deklarálását kikényszerítő (Option Explicit) utasítást, majd a feladathoz illeszkedő konstansokat deklaráljuk (pl. a pályaméret határait), illetve azokat a modulszintű változókat, amelyeket több szubrutinból is hivatkozni szeretnénk (pl. az aktuális pályaméret).

```
'A változók kötelező deklarálásához
Option Explicit
'A pályaméret határai
Const Tol = 2
Const Ig = 6
'A lövések jelzésére
Const Talalt = "X"
Const Melle = "*"
Const Alakzat = "O"
'A form fejlécéhez
Const Fejlec = "Torpedó"
'Az aktuális pályaméret
Dim n As Integer
'A kilőtt alakzatok száma
Dim Kilott As Integer
```

3. lépés: Kezdőtevékenységek.

A form megjelenése előtti kezdőtevékenységeket a form Initialize eseménykezelőjében kell programozni. Feltöltjük a legördülő lista tartalmát (az AddItem metódus segítségével, a feladatkiírás szerint ciklussal), majd beállítjuk a legördülő lista kezdőértékét (a Value tulajdonsággal).

```
'A form megjelenése előtt fut le
Private Sub UserForm_Initialize()
Dim i As Integer
'A legördülő lista feltöltése
For i = Tol To Ig
        cbPalya.AddItem i
Next
'A legördülő lista kezdőértéke és stílusa
cbPalya.Value = 4: cbPalya.Style = fmStyleDropDownList
'Az ablak fejlécének beállítása
Caption = Fejlec
'A vezérlők választhatóságának beállítása
btnSugo.Enabled = False: btnJatek.Enabled = False
btnLoves.Enabled = False: tbLoves.Enabled = False
'Az első munkalapot lássuk
Worksheets(1).Activate
'Pálya törlése
Call Torles
```

```
'A véletlenszám-generátor inicializálása
Randomize
End Sub
```

1.40. ábra. A játék formja kezdetben, és a pályaméret választás

Megjegyzés

• A legördülő listák (és a listák (ListBox)) elemei sztring típusúak. Az AddItem metódust most egész számokkal hívjuk meg, ezért egy implicit típuskonverzió történik (a szám sztringgé alakul), és ezután kerül az új elem a legördülő lista elemei közé. Az új elemek alapértelmezetten a lista végére kerülnek (ahogy most is), de az AddItem meghívható adott helyre történő beszúrásra is.

• A pálya törlését elvégző segédszubrutint (Torles) is meghívjuk (amelynek kifejtése a 4. lépésnél található).

• A Randomize utasítás a véletlenszám-generátor inicializálását végzi el. Ekkor a véletlen számok előállítására használt Rnd függvény (lásd 5. lépés) futásonként eltérő véletlen számokat fog előállítani.

4. lépés: A pályaméret változása, a játéktér törlése, a kilőtt alakzatok számának megjelenítése.

Ebben a lépésben a pályaméret megváltozását lekezelő eseménykezelő, és két segédszubrutin kapott helyet. A pályaméret megváltozásakor (cbPalya_Change) az aktuális méretet megjegyezzük (az n változóban), a Torles szubrutin a maximális területű pályát törli (hiszen nagyobb pálya után egy kisebb pályán játszva, ha csak az aktuális méretű pályát törölnénk, akkor esetleg ott maradna egy-egy zavaró karakter az előző játékból), a harmadik pedig a form fejlécét (Caption) aktualizálja.

```
'A legördülő lista értékének megváltozásakor fut le
Private Sub cbPalya_Change()
n = cbPalya.Value
End Sub
'A játéktér (pálya, háttér) törlése az aktuális munkalapon
Sub Torles()
Range(Cells(1, 1), Cells(Ig, Ig)).Clear
End Sub
'A kilőtt alakzatok számát a form fejlécében jelenítjük meg
Sub FejlecKirak()
```

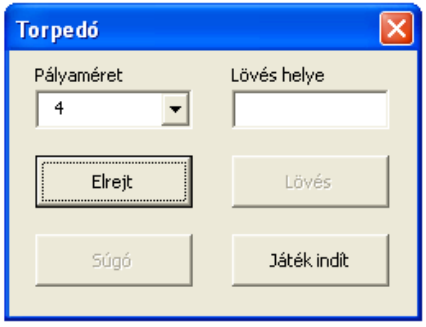
```
Caption = Fejlec + " (kilőve:" & Kilott & "db)"
End Sub
```

Megjegyzés: Az aktuális pályaméret beállításakor egy implicit típuskonverzió történik, mivel a legördülő listában lévő elemek sztringek (amelyek most egész számokat tartalmaznak (lásd 3. lépés), így a konverzió végrehajtható), az n változó pedig egész (Integer) típusú.

5. lépés: Alakzatok elrejtése.

Az alakzatok elrejtésénél n db alakzatot kell elhelyezni egy n×n-es táblázatban úgy, hogy nem teszünk ugyanarra a helyre több alakzatot. Az elrejtett alakzatokat a második munkalapra kell elhelyezni, ezért először aktivizáljuk a második munkalapot. Ezután töröljük az esetleg kint lévő, korábban elrejtett alakzatokat (Torles). A külső ciklus (For) biztosítja az n db alakzat elrejtését, a belső ciklus (Do) pedig azt, hogy egy véletlenszerű, még üres cellába (IsEmpty) kerüljön az éppen elrejtendő alakzat. Végezetül az első munkalapot aktivizáljuk, és választhatóvá tesszük a játék indítására szolgáló nyomógombot.

```
'Alakzatok (n db) véletlenszerű elrejtése
Private Sub btnElrejt_Click()
Dim i As Integer, s As Integer, o As Integer
Worksheets(2).Activate
Call Torles
For i = 1 To n
        Do
                s = Int(Rnd * n) + 1
                o = Int(Rnd * n) + 1
        Loop Until IsEmpty(Cells(s, o))
        Cells(s, o).Value = Alakzat
Next
Worksheets(1).Activate
'A játékot elkezdhessük
btnJatek.Enabled = True
End Sub
```



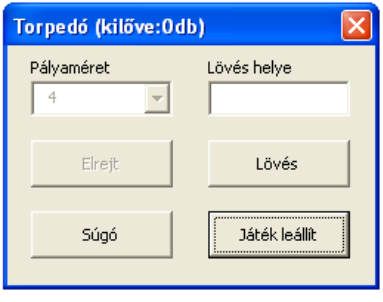
1.41. ábra. Elrejtés után már indítható a játék

6. lépés: Játék indítása, leállítása.

Itt lényegében csak a megfelelő vezérlők választhatóságát állítjuk be attól függően, hogy a játékot éppen indítjuk vagy leállítjuk. Azt, hogy éppen mit kell tenni, azt a nyomógomb aktuális feliratából deríthetjük ki. A játék leállításakor az elrejtésre szolgáló nyomógombot (btnElrejt) hozzuk fókuszba.

```
'Játék indítása, leállítása
Private Sub btnJatek_Click()
If btnJatek.Caption = "Játék indít" Then
        'A pálya törlése
        Call Torles
        btnJatek.Caption = "Játék leállít"
```

```
        btnLoves.Enabled = True: tbLoves.Enabled = True
        btnSugo.Enabled = True
        cbPalya.Enabled = False: btnElrejt.Enabled = False
        Kilott = 0
        Call FejlecKirak
Else
        btnJatek.Caption = "Játék indít"
        btnJatek.Enabled = False: btnLoves.Enabled = False
        tbLoves.Enabled = False: btnSugo.Enabled = False
        cbPalya.Enabled = True: btnElrejt.Enabled = True
        'Az Elrejt gomb legyen kiválasztva
        btnElrejt.SetFocus
        Caption = Fejlec
End If
End Sub
```

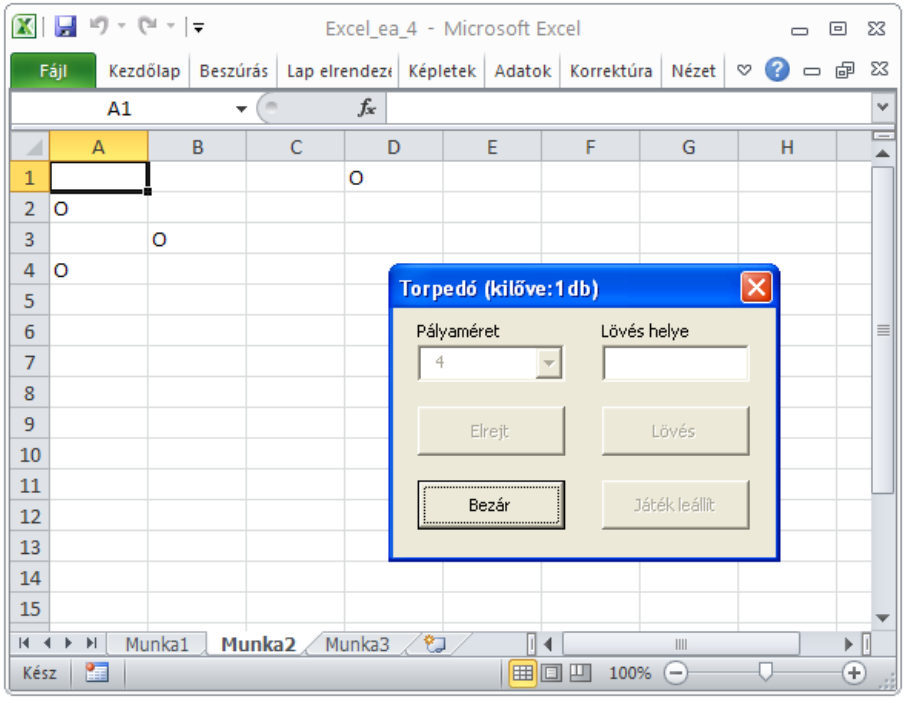


1.42. ábra. Egy már elindított játék

7. lépés: Súgó megvalósítása.

A súgó amellett, hogy aktivizálja a megfelelő munkalapot, a megfelelő vezérlők választhatóságát is beállítja. Hasonlóan a játék indítás, leállítás funkcióhoz (lásd 6. lépés), itt is a nyomógomb aktuális feliratából tudjuk meg, hogy éppen megjeleníteni, vagy bezárni kell-e a súgót.

```
'A súgó gomb tevékenységei
Private Sub btnSugo_Click()
If btnSugo.Caption = "Súgó" Then
        Worksheets(2).Activate
        btnSugo.Caption = "Bezár"
        btnLoves.Enabled = False
        tbLoves.Enabled = False
        btnJatek.Enabled = False
Else
        Worksheets(1).Activate
        btnSugo.Caption = "Súgó"
        btnLoves.Enabled = True
        tbLoves.Enabled = True
        btnJatek.Enabled = True
End If
End Sub
```



1.43. ábra. A játék súgója

8. lépés: A lövés célcellájának ellenőrzése.

A függvény a paraméterben megadott célcella címének helyességét ellenőrzi. Az adatot nagybetűsre alakítja (UCase) (így kis- és nagybetűvel is megadható a célcella oszlopának betűjele), majd ellenőrzi a hosszt, az első, illetve a második karaktert (ahol is az aktuális pályamérettel (n) dolgozunk). Hiba esetén a megfelelő üzenettel tájékoztatunk. A függvény eredménye a kapott célhely (st) helyessége (mint logikai érték).

```
'A célcella helyességének ellenőrzése
```

```
Function Ellenoriz(st As String) As Boolean
Dim kar As String, ok As Boolean
ok = False
st = UCase(st) 'Nagybetűsre alakítás
If Len(st) <> 2 Then
        MsgBox ("Nem megfelelő hossz!")
Else
        kar = Chr(Asc("A") + n - 1)
        If Left(st, 1) < "A" Or Left(st, 1) > kar Then
                MsgBox ("Nem megfelelő oszlop!")
        Else
                kar = Chr(Asc("1") + n - 1)
                If Right(st, 1) < "1" Or Right(st, 1) > kar Then
                        MsgBox ("Nem megfelelő sor!")
                Else
                        ok = True
                End If
        End If
End If
Ellenoriz = ok
End Function
```

9. lépés: A lövés végrehajtása.

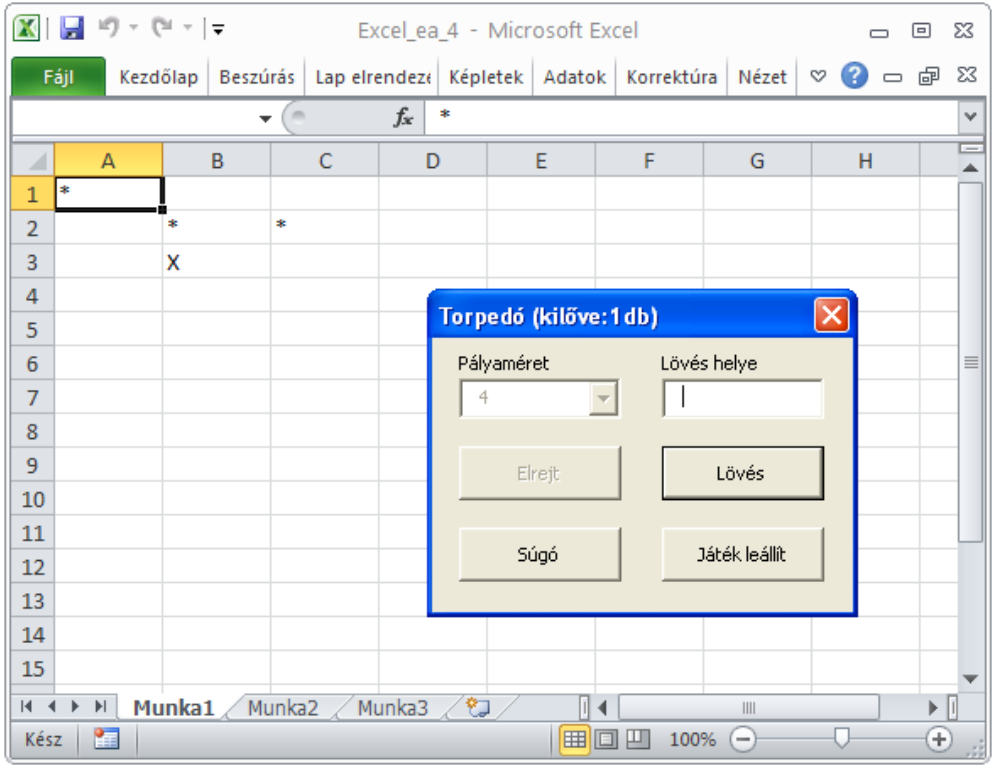
A lövés végrehajtását a beviteli mezőben megadott célhely helyességének vizsgálatával kezdjük. Ha formailag rossz a célhely, akkor az ellenőrző függvény (Ellenoriz) a hibáról is tájékoztat. Ha a célhely formailag helyes (azaz a pályára esik), akkor megvizsgáljuk, hogy ide lőttünk-e már korábban. Ha igen, akkor erről tájékoztatunk, egyébként meg kiértékeljük a lövést. Ha eltaláltunk egy alakzatot (a második munkalapon elrejtett alakzatok közül), akkor adminisztráljuk a találatot, frissítjük a fejlécet (ahol a kilőtt alakzatok száma megjelenik), és ha vége a játéknak (ha már kilőttük az összes alakzatot), akkor erről tájékoztatunk, és új játékot kezdünk. A találatot a Talalt, a mellé lövést a Melle adattal (lásd 2. lépés) jelezzük a célcellában.

```
'A lövés végrehajtása
Private Sub btnLoves_Click()
Dim st As String
st = tbLoves.Text
If Ellenoriz(st) Then
    'Jó a célcella címe
    If Not IsEmpty(Worksheets(1).Range(st)) Then
            MsgBox ("Ide már lőtt!")
    Else
            'Kiértékelés
            If Worksheets(2).Range(st) = Alakzat Then
                    Range(st) = Talalt
                    Kilott = Kilott + 1
                    Call FejlecKirak
                    If Kilott = n Then
                            MsgBox ("A játéknak vége!")
```

```
                    'A játék leállítása
                    Call btnJatek_Click
                End If
            Else
                Range(st) = Melle
            End If
        End If
End If
'Lövés törlése a beviteli mezőben
tbLoves.Text = ""
'Ha még nincs vége a játéknak, akkor a beviteli mezőre állunk
if Kilott <> n Then
    tbLoves.SetFocus
End If
End Sub
```

Végezetül néhány ötlet (feladat) a játék továbbfejlesztési lehetőségeit illetően:

- A játéktér (pálya, háttér) kiemelése (pl. színekkel, keretezéssel), négyzetrácsos megjelenítés.
- Értékelés bevezetése (pl. számoljuk az összes lövés darabszámát, és a játék végén a talált/összes lövés arányában minősítjük a játékos teljesítményét).
- Súgó kezelése (pl. a súgót csak korlátozott számban engedjük megnézni, akkor is csak korlátozott ideig (lásd Timer függvény)).
- Hangjelzések (lásd Beep utasítás, Speech objektum, pl. Application.Speech.Speak "Great").
- Összetett, több cellából álló alakzatok kezelése (elrejtés változása, alakzat „elsüllyedése”).
- Gép elleni játék (pl. a gép is lő az általunk elrejtett alakzatokra).



1.44. ábra. A játék közben a pályát látjuk

Önellenőrzés

1. Az alábbi állítások közül melyek igazak?
   - ☐ A makrórögzítéssel megtudhatók azok az utasítások, amellyel egy „kézzel elvégzett” tevékenység végrehajtható.
   - ☐ Egy metódushívással törölhető egy munkafüzet összes diagramja.
   - ☐ A menükezeléssel elérhető az, hogy egy felhasználó a Visual Basic Editor használata nélkül futtasson makrókat.
   - ☐ A Dialogs gyűjtemény elemei bővíthetők és módosíthatók.
2. Makrórögzítés
   Rögzítsük egy tetszőleges, az Excel-ben „kézzel” elvégzett tevékenység (pl. egy diagramkészítés) makróját, nézzük meg a rögzített makrót, esetleg módosítsuk, illetve futtassuk tetszőlegesen!
3. Vizuális formtervezés
   • Készítsük el a jegyzetben szereplő *Torpedó* játékot (lásd 1.3.3. fejezet)! (*Tipp:* Először a párbeszédablakot készítsük el a megfelelő vezérlők (egy legördülő lista, egy beviteli mező és négy darab nyomógomb) segítségével. Fontos, hogy a vezérlők neve (*Name*) rendre *cbPalya, tbLoves, btnElrejt, btnJatek, btnSugo, btnLoves,* a nyomógombok felirata (*Caption*) *Elrejt, Játék indít, Súgó, Lövés* legyen! A játékhoz tartozó modulszintű utasítások, segédszubrutinok, valamint a vezérlőkhöz tartozó eseménykezelők forráskódja a jegyzetből a vágólap segítségével áthozható. Minden forráskódot másoljunk át, a modulszintű utasításokat a form moduljának elejére tegyük!)
   • Fejlesszük tovább a *Torpedó* játékot a jegyzetben szereplő szempontok szerint (pl. a játékteret színekkel emeljük ki, használjunk négyzet alakú cellákat, a játék végén értékeljük a játékos teljesítményét, stb.)!
4. Programozási feladatok
   A megoldásokhoz az aktuális munkalapot használjuk! A megoldásokra lehetőleg teljes körűen paraméterezett szubrutinokat készítsünk!

• Készítsünk egy *nxn*-es szorzótáblát az A1-es cellától kezdődően! Az első sorban, illetve oszlopban az egész számok legyenek 1-től *n*-ig, a táblázatban pedig, a megfelelő elemek szorzata! (*Tipp:* Először az első sor, illetve első oszlop tartalmát állítsuk elő (egy-egy, vagy akár egyetlen For ciklussal), majd a táblázat „belsejét" két, egymásba ágyazott For ciklussal!)

• Generáljunk véletlenszerűen ötöslottó számokat (5 darab egész számot az [1, 90] intervallumból) az A1:E1 blokkba!-(*Tipp:* Használjunk (egy) For ciklust és az Rnd függvényt!)

• Generáljunk véletlenszerűen egy szabályos ötöslottó szelvényt (5 darab, különböző egész számot az [1, 90] intervallumból) az A1:E1 blokkba! (*Tipp:* A megoldáshoz egy For ciklusba ágyazott hátultesztelő ciklus szükséges. A For ciklus egyszerűen 1-től 5-ig menjen egyesével, a belső ciklus pedig mindaddig, amíg egy olyan számot sikerül generálnia, amely még nem szerepel az eddig generáltak között! Az ellenőrzés megoldható egy ellenőrző While ciklus segítségével (lásd 1.2.2.8. fejezet), de egy 90 elemű logikai tömb segítségével is. Az utóbbi esetben a tömb elemei adminisztrálják egy adott szám generáltságát (előfordulását, szereplését). Az elején minden érték legyen hamis (False), hiszen még egyik számot se generáltuk. Egy generált szám jó, ha a hozzá tartozó tömbelem értéke hamis, ekkor állítsuk igazra (True) a tömbelemet, és tegyük a számot a megfelelő cellába!)

• Végezzük el az előző feladatot úgy, hogy a számok egy megadott sor első öt oszlopába kerüljenek (pl. 3 esetén a 3. sorba)! (*Tipp:* A paraméter értékét használva a munkalap ilyen sorszámú (indexű) sorába tegyük a generált számokat!)

• Az előző feladat megoldását felhasználva állítsunk elő adott számú szabályos ötöslottó szelvényt! A szelvényeket az első sortól kezdve, folyamatosan tegyük az aktuális munkalapra (pl. 10 szelvény esetén az első 10 sorba)! (*Tipp:* Egy For ciklusból egyszerűen hívjuk meg az előző feladat megoldását elvégző szubrutint!)

• Készítsünk egy *nxn*-es sakktáblát! A sakktábla bal felső sarka az A1-es cella legyen, minden cellája legyen négyzet alakú, és legyen egy adott színnel sakktáblaszerűen kiszínezve! A sötét és a világos mezők (cellák) soronként és oszloponként váltakozzanak! A sötét mezők színezésére az adott színt használjuk, a világos mezőket hagyjuk fehéren! (*Tipp:* Használjuk a sorok magasságának beállítására a Range objektum RowHeight tulajdonságát, az oszlopok szélességének beállítására a Range objektum ColumnWidth tulajdonságát! A színezést két, egymásba ágyazott For ciklussal végezzük, amelyben a soron következő cella háttérszínét (Interior tulajdonság) váltogassuk, azaz minden másodikat színezzük csak ki!)

• Tegyük fel, hogy az aktuális munkalap bal felső részén szabályos ötöslottó szelvények helyezkednek el. Minden sorban egy szelvény számai találhatók (az első 5 oszlopban). Tekintsük az első sor számait a nyerőszámoknak, majd értékeljük ki a többi sor szelvényeit, mint megjátszott számokat! Egy adott szelvény találatainak számát az adott sor 7. oszlopába tegyük! (*Tipp:* Használjunk három egymásba ágyazott For ciklust! A legkülső ciklusban a kiértékelendő szelvényeket vegyük sorra, és minden szelvényre végezzük el a megfelelő találatszámolást két egymásba ágyazott ciklus segítségével! Az egyik ciklusban a nyerőszámokon, a másikban az éppen kiértékelendő szelvény számain lépdeljünk!)
• Az előző feladat értékelését végezzük el úgy, hogy az eltalált számokat (azok betűszínét) egy adott színnel ki is emeljük! (*Tipp:* Egy eltalált szám betűszínét a Font.Color tulajdonság segítségével változtassuk meg!)
• Az értékelést végezzük el úgy is, hogy nem tudjuk, hogy mekkora a feldolgozandó adatblokk, azaz nem tudjuk, hogy hány szelvényünk van! (*Tipp:* Használjuk a Range objektum CurrentRegion tulajdonságát!)
• Határozzuk meg az A1-es cellát tartalmazó összefüggő blokk méreteit a CurrentRegion tulajdonság használata nélkül! Feltehető, hogy az A1-es cellát tartalmazó összefüggő blokk minden cellájában van adat. (*Tipp:* Használjunk egy-egy While ciklust és az IsEmpty függvényt!)

Modulzáró feladatok

5. Programozási mintafeladatok

A programozási mintafeladatokhoz az alábbi segédszubrutinokat fogjuk használni (illetve a programozási feladatoknál is ezeket kell majd használni).

```
'Adott számú (Mennyit), véletlen, a [Tol, Ig] intervallumba eső
'egész szám generálása egy tömbbe (Mibe), adott Seed érték mellett
Sub Szamokat_General(Mibe() As Integer, Mennyit As Integer, _
        Tol As Integer, Ig As Integer, Seed As Integer)
Dim i As Integer, db As Integer
Rnd (-1)
Randomize (Seed)
db = Ig - Tol + 1
For i = 1 To Mennyit
        Mibe(i) = Int(Rnd * db) + Tol
Next
End Sub
'Adott hosszú, véletlenszerű szöveges adat (sztring) generálása
'betűkből, számjegyekből és egyéb karakterekből
Function Szoveget_General(Hossz As Integer, Seed As Integer) As String
Const Kisbetuk = "aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz"
Const Szamjegyek = "0123456789"
Const Egyeb = "'!%$/=<>()\/[]#&@{}<,;:.?+-_ "
Dim i As Integer, db As Integer, Karakterek As String, st As String
Rnd (-1)
Randomize (Seed)
Karakterek = Kisbetuk + UCase(Kisbetuk) + Szamjegyek + Egyeb
db = Len(Karakterek)
st = ""
For i = 1 To Hossz
```

```
        st = st + Mid(Karakterek, Int(Rnd * db) + 1, 1)
Next
Szoveget_General = st
End Function
'Prímszám vizsgálat
Function Prim(a As Long) As Boolean
Dim o As Long, b As Long, van As Boolean
If a <= 1 Then
        Prim = False
Else
        o = 2: van = False: b = Int(Sqr(a))
        While o <= b And Not van
                If a Mod o = 0 Then
                        van = True
                Else
                        o = o + 1
                End If
        Wend
        Prim = Not van
End If
End Function
'Relatív prím vizsgálat
Function RelPrim(a As Long, b As Long) As Boolean
Dim o As Long, c As Long, van As Boolean
If a <= b Then c = a Else c = b
o = 2: van = False
While o <= c And Not van
        If a Mod o = 0 And b Mod o = 0 Then
                van = True
```

```
        Else
                o = o + 1
        End If
Wend
RelPrim = Not van
End Function
```

- mintafeladat: A Szamokat_General szubrutin segítségével generáljunk 5 darab egész számot az [1, 10] intervallumban 1-es Seed értékkel, majd határozzuk meg a generált adatok minimumát, maximumát, összegét és átlagát!
  A megoldás megtekintéséhez kattintson ide!
- mintafeladat: A Szoveget_General szubrutin segítségével generáljunk 10 karakter hosszú szöveges adatot 2-es Seed értékkel, majd határozzuk meg a kapott adatban található angol betűk darabszámát!
  A megoldás megtekintéséhez kattintson ide!
- mintafeladat: A Prim függvény segítségével határozzuk meg az [1, 10] intervallumba eső prímek darabszámát!
  A megoldás megtekintéséhez kattintson ide!

6. Programozási feladatok
A feladatokhoz most nem adunk megoldási tippeket, mert bízunk benne, hogy a mintafeladatok és az eddig megszerzett tudás alapján, tippek nélkül is sikerül majd megoldani őket.

- A Szamokat_General szubrutin segítségével generáljunk 10 darab egész számot az [1, 5] intervallumban 3-as Seed értékkel, majd határozzuk meg a generált adatokra az alábbiakat:
  - a legkisebb adat,
  - a legkisebb adat darabszáma,
  - a második legnagyobb adat darabszáma,
  - a legfeljebb 2 értékű adatok összege,
  - a legalább 3 értékű adatok átlaga,

  - az adatok átlagánál nagyobb adatok darabszáma.

- A Szoveget_General szubrutin segítségével generáljunk egy 10 karakter hosszú szöveges adatot 2-es Seed értékkel, majd határozzuk meg a kapott adatra (a sztringben található karakterekre) az alábbiakat:
  - a legkisebb ASC kódú karakter darabszáma,
  - a számjegy karakterek ASC kódjainak összege,
  - az angol kisbetűk ASC kódjainak átlaga,
  - az angol nagybetűk és kisbetűk darabszámának eltérése.
- A Prim függvény segítségével határozzuk meg az alábbiakat:
  - egy adott a számnál nagyobb n-edik ($n > 0$ egész) prímszám,
  - egy adott $[a, b]$ intervallumba ($a < b$ pozitív egészek) eső prímszámok összege, átlaga.
- A RelPrim függvény segítségével határozzuk meg:
  - egy adott $[a, b]$ intervallumba ($a < b$ pozitív egészek) eső relatív prím számpárok darabszámát. Egy számpárt csak egyszer vegyünk figyelembe (pl. a 3, 8 számpárt 8, 3 párként már ne vegyük figyelembe)!
  - egy adott $[a, b]$ intervallumba ($a < b$ pozitív egészek) eső azon egész számok darabszámát, amelyek relatív prímek egy adott n ($n > 0$ egész) számmal.

II. MODUL

Matematikai számítások - Matlab

7. LECKE

A Matlab alapjai

2. Matematikai számítások Matlabbal

2.1. Alapvető információk a Matlabról

A Matlab szoftver (Matrix Laboratory) a mérnöki számítások nagyon fontos segítőeszköze. Ez a mérnöki és matematikai számításokat támogató programcsomag több komponensből áll:

1. Fejlesztői és futtatói környezet (adat és szövegszerkesztő, parancsablak, hibakereső eszközök, előzmények ablaka, munkakönyvtár katalógus stb.);
2. Programozási nyelv;
3. Függvénykönyvtárak;
4. ToolBox-ok (speciális alkalmazási bővítmények);
5. Help.

Jegyzetünkben áttekintjük a Matlab alapvető parancsait és a leggyakrabban használt függvényeket. A saját függvények megírása mellett programozási ismereteket is szerzünk, amelyek segítségével összetett feladatok megoldására is képesek leszünk. A kiszámított eredmények grafikus megjelenítésének legfontosabb lépéseivel is foglalkozni fogunk. A jegyzet keretei között azonban csak egy néhány hetes modult lefedő tárgyalásra van lehetőség. A Matlab iránt mélyebben érdeklődőknek érdemes további szakkönyveket is tanulmányozni. Különösen ajánljuk a [9] művet.

A Matlab elsajátítása a könyvek átolvasásán túlmenően is nagyon sok gyakorlást és kísérletezést igényel. Az itt közölt minták csak az elindulást segítik. Az önálló munka legfontosabb része a probléma korrekt megfogalmazása, majd annak a Matlab eszköznek a beazonosítása, amelynek segítségével a feladatunk megoldhatóvá válik. A legnagyobb akadályt a hiányos matematikai ismeretek jelentik. Éppen ezért csak akkor érdemes a Matlab segítségét kérni, ha a feladatunk matematikai hátterével tisztában vagyunk. Ehhez nem csak a megfelelő egyetemi matematikai jegyzetek (például: [8]), hanem a források között megjelölt internetes leírások is segítséget nyújtanak. Mindezek mellett a Matlab beépített helpje és dokumentációja is fontos szerepet kaphat.

A Matlab legfontosabb adattípusa a mátrix, de a skalárokkal is tud számolni, mint 1x1-es mátrixokkal. A mátrixokon értelmezett műveletek és függvények segítségével nagyon összetett feladatok megoldását lehet röviden és tömören megfogalmazni.

A Matlab-ban 10-nél több alapvető adattípus található, amelyek mindegyikéből lehet mátrixokat képezni.

A mátrixok megadására az alábbi lehetőségeket használhatjuk:

1. Elemek közvetlen felsorolása;
2. Kisebb mátrixokból való összerakás;
3. Beépített utasításokkal, függvényekkel, műveletekkel, kifejezésekkel történő előállítás;
4. Külső programok felhasználása;
5. Adatállományból való betöltés.

Az, hogy a Matlab alapvetően mátrixokkal számol, csak első látásra szokatlan. Más elemző rendszerek is használnak mátrixmanipulációs eszközöket. Ilyen például az SPSS statisztikai elemző rendszer is, mely a MATRIX. és END MATRIX. szekcióba zárt parancsaival és függvényeivel szintén tömör mátrixmanipulációs lehetőségeket biztosít. (pl. EIGEN() – sajátértékszámítás, SVD() – Singular Value Decomposition, T() – Transpose, MDIAG() – oszlopvektor diagonálmátrixba alakítása, stb.)

A Matlab valójában egy interpreter, amely interaktív módban dolgozik, azaz a feltett kérdésekre (parancs, kifejezés, ...) azonnal válaszol. Ezt a működési módot csak egyszerűbb feladatok megoldására használjuk. Sokkal gyakoribb az ún. M-fájlok használata, amikor a felhasználó egy szövegszerkesztő segítségével a Matlab programnyelvi elemeit használva megírja a saját programját, és ezt később az interpreter segítségével lefuttatja.

Az M-fájlok .m kiterjesztésű szöveges fájlok, amelyek Matlab programkódot tartalmaznak.

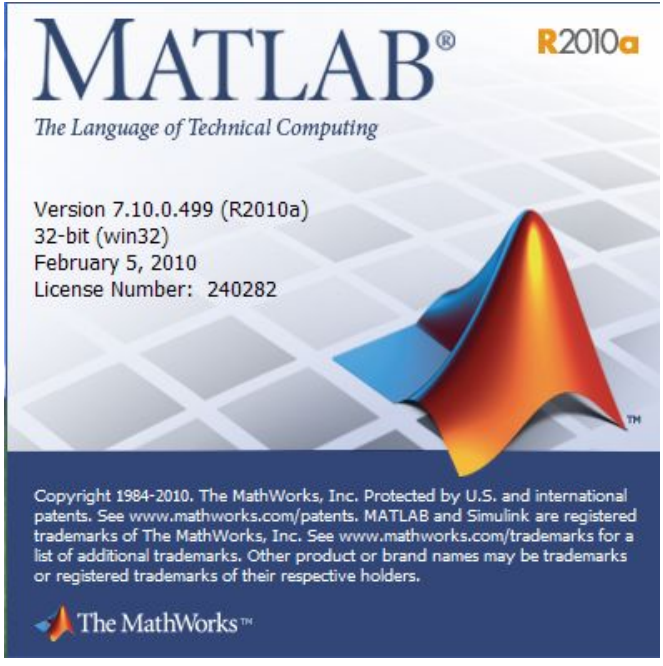
A következőkben olyan szinten ismerkedünk meg a Matlab-bal, hogy a mérnöki munkákban előforduló egyszerűbb számítási feladatokhoz segédeszközként használni tudjuk. Nem célunk a Matlab mélyebb megismerése. Nem foglalkozunk olyan típusokkal mint a struktúra, függvény-referenciák, felhasználói osztályok, Java osztályok, és nem foglalkozunk az objektum-orientált programozással sem.

Nem használjuk a speciális ToolBoxokat (bővítmények) sem, bár néhányról meg kell emlékeznünk. A Simulink bővítmény a mérnöki tervezés igen fontos része, amelyet a szaktantárgyak keretében fognak megismerni és felhasználni. A Symbolic Math Toolbox segítségével a Maple-rendszerhez hasonlóan szimbolikus számításokat végezhetünk.

2.1.1. A Matlab indítása, ablakok, kilépés

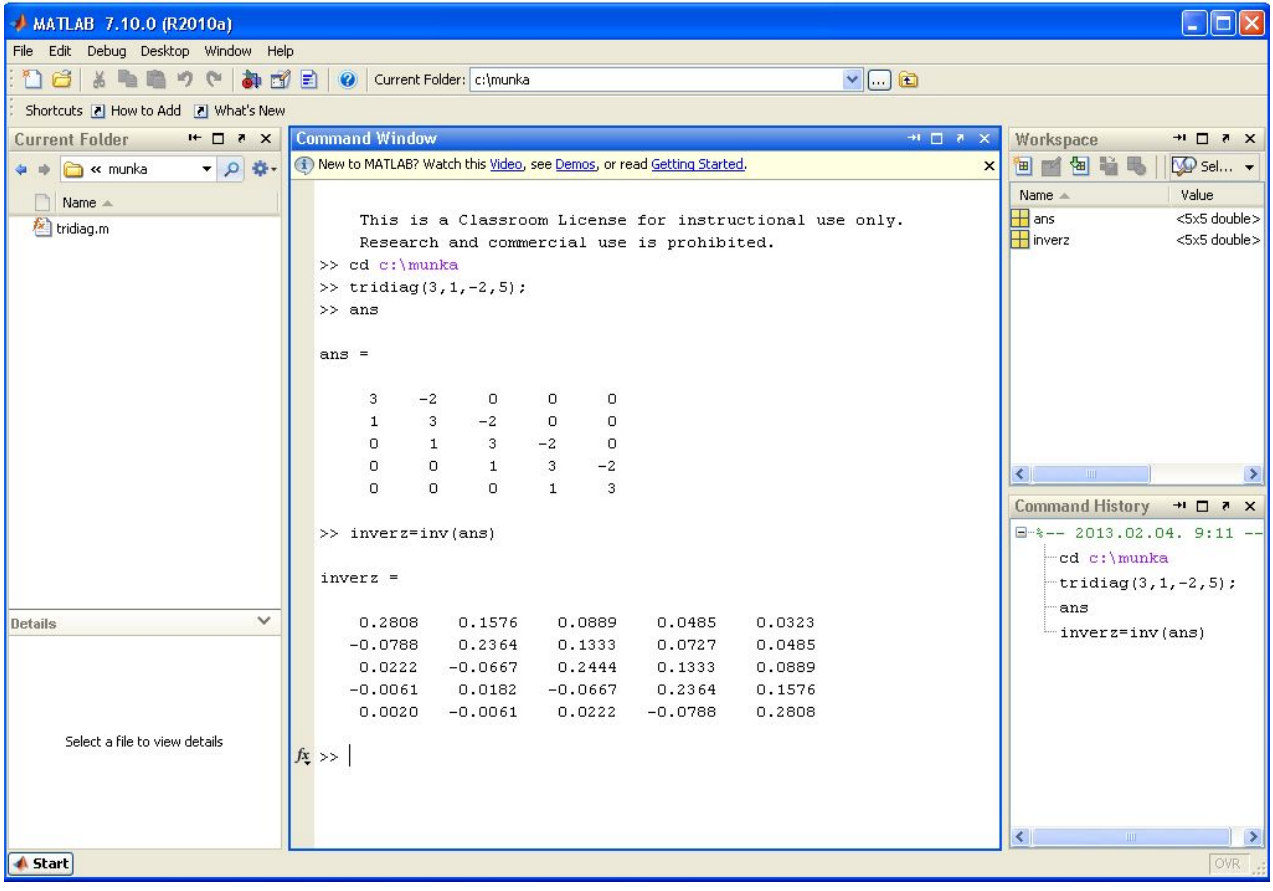
A Matlab indítása asztalon lévő parancsikonnal, fájlkezelőből, vagy Start menü programok almenüjéből történhet. A Matlab a hozzá társított speciális kiterjesztésű fájlokra duplán kattintva is elindul, ilyenek (.m, .fig, .mat, . . .).

Indítás után rögtön feltűnik a névjegye:



2.1. ábra: A Matlab névjegye

Az indítás és néhány parancs begépelése után az alábbi fejlesztői felületet látjuk:



2.2. ábra: A Matlab szokásos ablakai

A szokásos ablakok (alapbeállítás esetén megnyílók, balról jobbra):

1. Current Folder – aktuális könyvtár és fájljai;
2. Details – aktuális könyvtárbeli kijelölt fájl információi;
3. Command Window – parancsablak, parancsok begépelésének és a rendszerválaszok helye;
4. Workspace – munkaterületi változók;
5. Command History – parancselőzmények.

Speciális ablakok (ezeket esetenként nyitjuk meg; csak a fontosabbak, névsorban):

1. Editor;
2. Figure Window/Editor;
3. Help;
4. Variable Editor.

A parancsablakban (**Command Window**) azonnal megjelenik a >> parancs prompt, itt kezdhetjük el a munkát. A parancsablakban Student verziók esetében figyelmeztetést is kapunk, hogy csak oktatási célra használható!

Az első parancsként célszerű olyan aktuális könyvtárat (**Current Folder**) beállítani, amelyhez kellő írási és olvasási jogunk van.

```
>> cd c:\textbackslash munka
```

A begépelt és helyes parancsot, vagy kifejezést az Enter lenyomása után a Matlab azonnal kiértékeli, illetve végrehajtja, és ha kell, akkor válaszol rá, majd a végrehajtott parancs megjelenik az előzmények (**Command History**) ablakban is. A Command History ablakbeli parancsainkat vágólapon keresztül szöveges fájlokba másolhatjuk későbbi ismételt felhasználás céljára. A Command History tartalma tetszés szerint törölhető is.

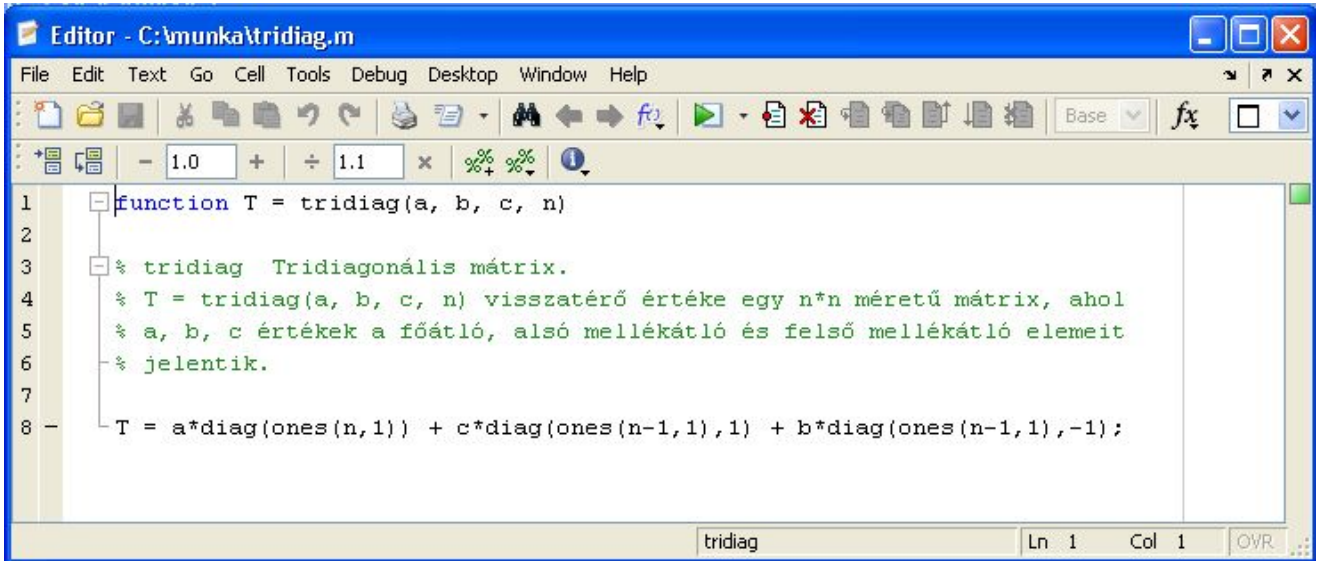
Ha a parancs, vagy kifejezés megjeleníthető eredményét nem rendeljük változóhoz, akkor az automatikusan az **ans** nevű belső változóhoz rendelődik (lásd **tridiag**(3, 1, -2, 5) függvényhívás), és annak korábbi értékét felülírja.

Ha a parancsot a (;) pontosvessző jel zárja, akkor a végrehajtás utáni eredmény nem kerül kijelzésre (lásd tridiag(3, 1, -2, 5); függvényhívás).

Amíg nem lépünk ki a Matlab-ból, az aktív parancs promptnál a kurzor fel és le billentyűk segítségével a korábbi parancsaink között tallózhatunk, és az előhívott parancsot közvetlenül, vagy átszerkesztve megismételhetjük.

A parancsablakban vagy direkt parancsokat, kifejezéseket közlünk, vagy az aktuális könyvtárban, vagy más elérhető könyvtárban található parancsköteg fájlra (ún. M-fájlra) hivatkozhatunk. Az M-fájlok a direkt parancsokon kívül programozási elemeket (iteráció, szelekció, stb.) is tartalmazhatnak. Az M-fájlokat az **Editor** ablakban nyithatjuk meg, amely egy szövegszerkesztő segítségével támogatja a programírást.

Az Editor ablakbeli szövegszerkesztő a programozási elemeket színezéssel emeli ki a könnyebb áttekinthetőség érdekében. Az aktuális könyvtárbeli **tridiag.m** fájl segítségével egy sávmátrixot létrehozó függvényt adtunk meg a beépített **ones**() és **diag**() függvények segítségével (ezeket később tárgyaljuk):



2.3. ábra: Szerkesztő ablak

Az olyan függvényekre, amelyek definíciós M-fájlja a Matlab számára elérhető könyvtárban van, a munkánk során mindig hivatkozhatunk. A következő

```
>> ans
```

hivatkozás egy legális kifejezés, hiszen egy most már létező változóra hivatkozik, és mivel nincs mögötte (;) jel, ezért a kiértékelés eredménye azonnal megjelenik. Az ezt követő

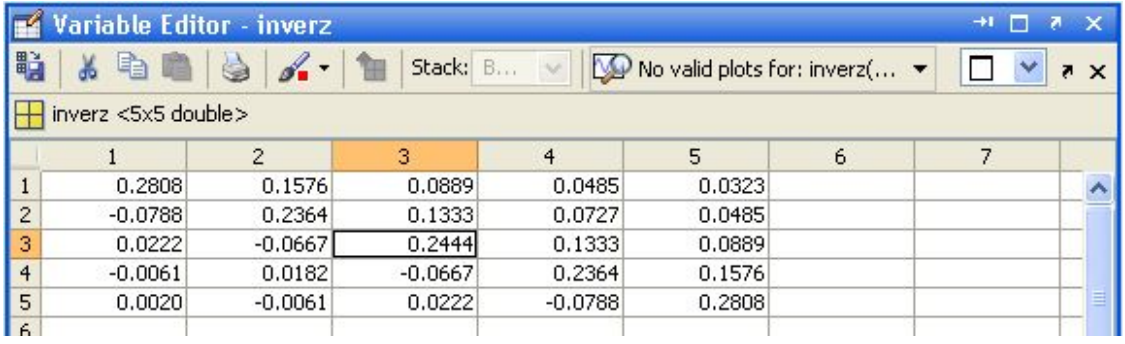
```
>> inverz=inv(ans)
```

értékadó kifejezés a korábban létrehozott sávmátrix inverzét állítja elő és jeleníti meg. A megjelenítés alapértelmezés szerinti **short** formátumú, azaz 4 értékes jegyet látunk a tizedespont mögött.

A munkaterület ablakban (**Workspace**) láthatjuk a már létrehozott változóinkat és szerkeszthetjük a tartalmukat. A munkaterületen lévő változók parancsok segítségével

1. felülírhatók (értékadó kifejezés),
2. fájlba menthetők és visszatölthetők (save, load, stb. parancsok és függvények),
3. részben vagy teljesen törölhetők (clear parancs).

Az adatszerkesztő (**Variable Editor**) ablak akkor nyílik meg, ha a munkaterületi változóink valamelyikére duplán rákattintunk. Itt akár újabb sorokkal és oszlopokkal is bővíthetjük a mátrixainkat:



	1	2	3	4	5	6	7
1	0.2808	0.1576	0.0889	0.0485	0.0323		
2	-0.0788	0.2364	0.1333	0.0727	0.0485		
3	0.0222	-0.0667	0.2444	0.1333	0.0889		
4	-0.0061	0.0182	-0.0667	0.2364	0.1576		
5	0.0020	-0.0061	0.0222	-0.0788	0.2808		
6							

2.4. ábra: Adatszerkesztő ablak

Az aktuális könyvtár ablakban (**Current Folder**) az ott elérhető fájlok listáját látjuk, míg a részletek ablakocskában (**Details**) az aktuális könyvtár kijelölt fájljára vonatkozó információkat találjuk.

A Matlab-ból való kilépést a **quit**, vagy **exit** parancsokkal, illetve a File menüből is kezdeményezhetjük, de a **CTRL+Q** forróbillentyűvel is végrehajthatjuk.

Ha az otthoni gépünkön nincs jogunk a Matlab használatához, akkor a hasonló felépítésű és szerepű, de egyszerűbb Octave ingyenes szoftvert használhatjuk, melyet a következő linkről tölthetünk le:

http://www.gnu.org/software/octave/download.html

2.1.2. Help rendszer

A Matlab-bal való alaposabb megismerkedést a Help rendszerrel érdemes kezdeni. Ennek komponensei:

1. Help parancs/függvény;
2. Help browser, doc parancs/függvény;
3. Lookfor parancs.

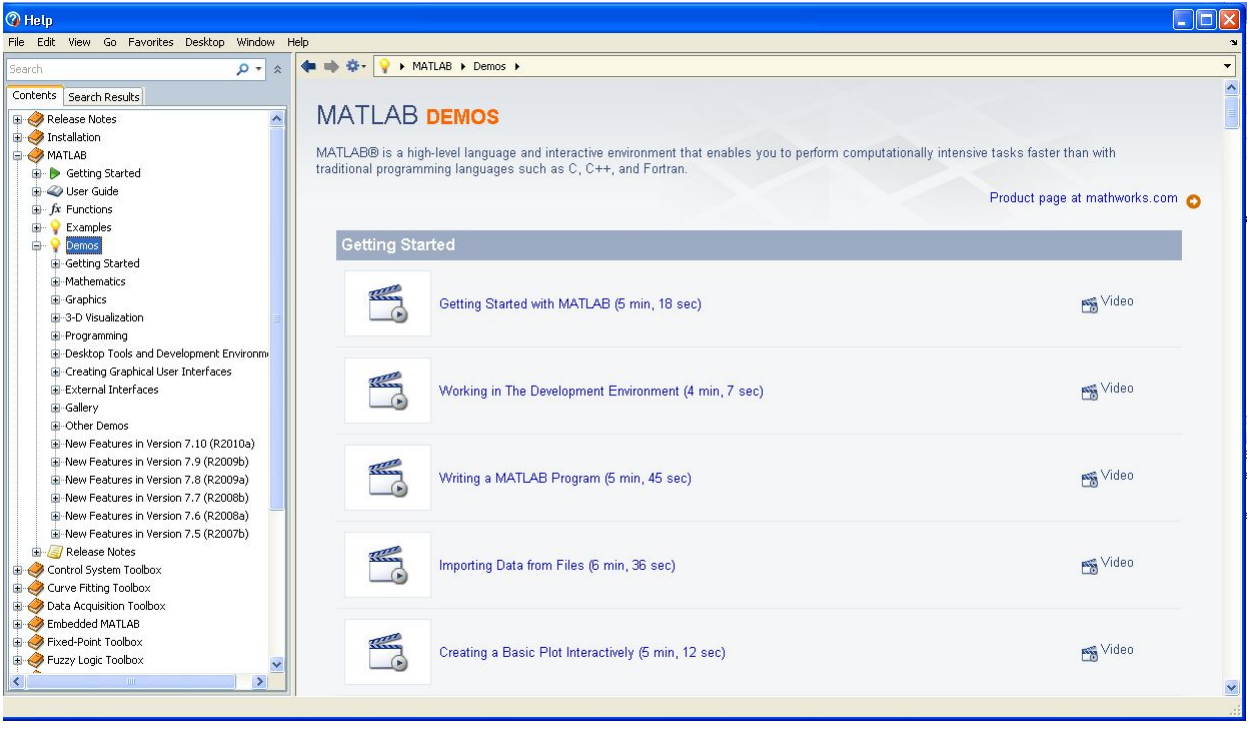
A parancsablakban kiadott help inv parancs az inv() függvényről szolgáltatja az alapvető tudnivalókat:

```
>> help inv
 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or nearly singular.
    See also slash, pinv, cond, condest, lsqnonneg, lscov.
    Overloaded methods:
       lti/inv
    Reference page in Help browser
       doc inv
```

A Help browser, amely önálló Help ablakot nyit a menüből a

```
>> doc
```

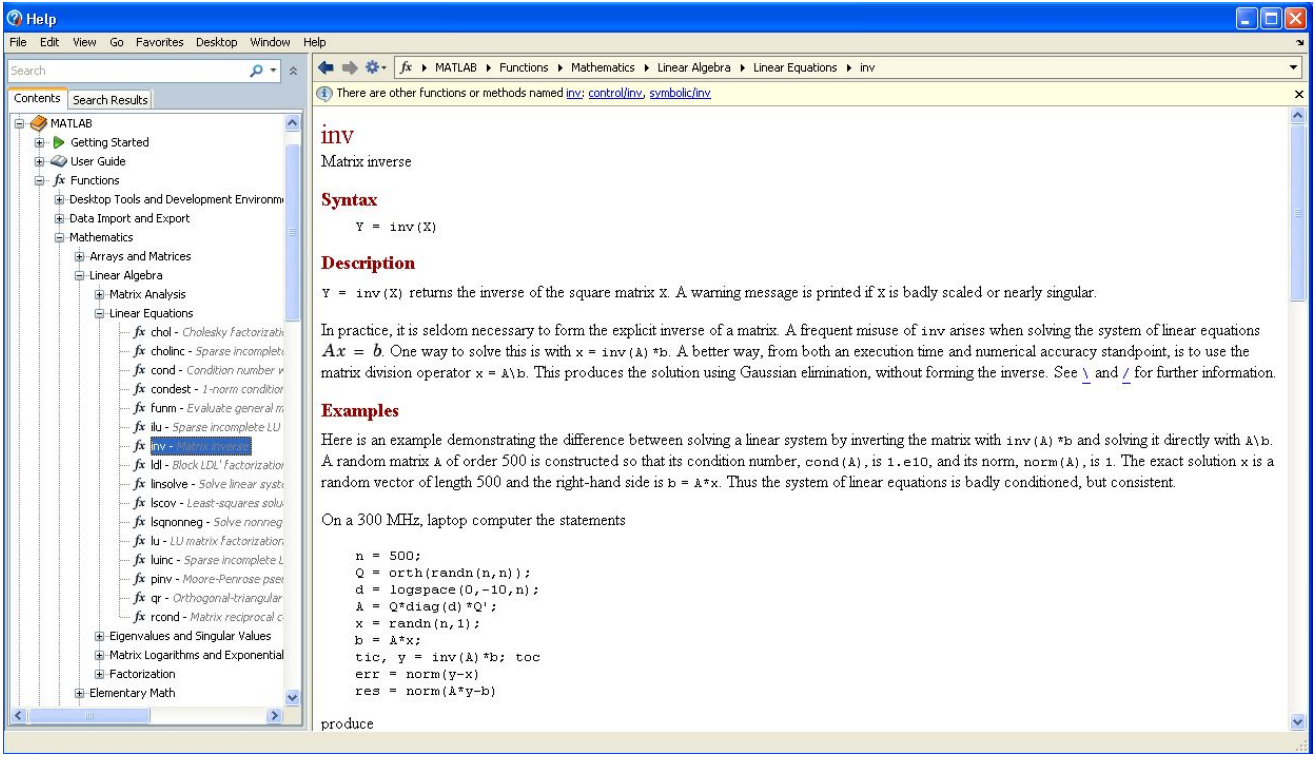
paranccsal nyitható meg. A Help rendszer fa-struktúrájú katalógusában a minket érdeklő témát könnyen megtaláljuk. Érdemes az ismerkedésünk elején a beépített demo videókat megnézni!



2.5. ábra: Demo fájlok a Helpben

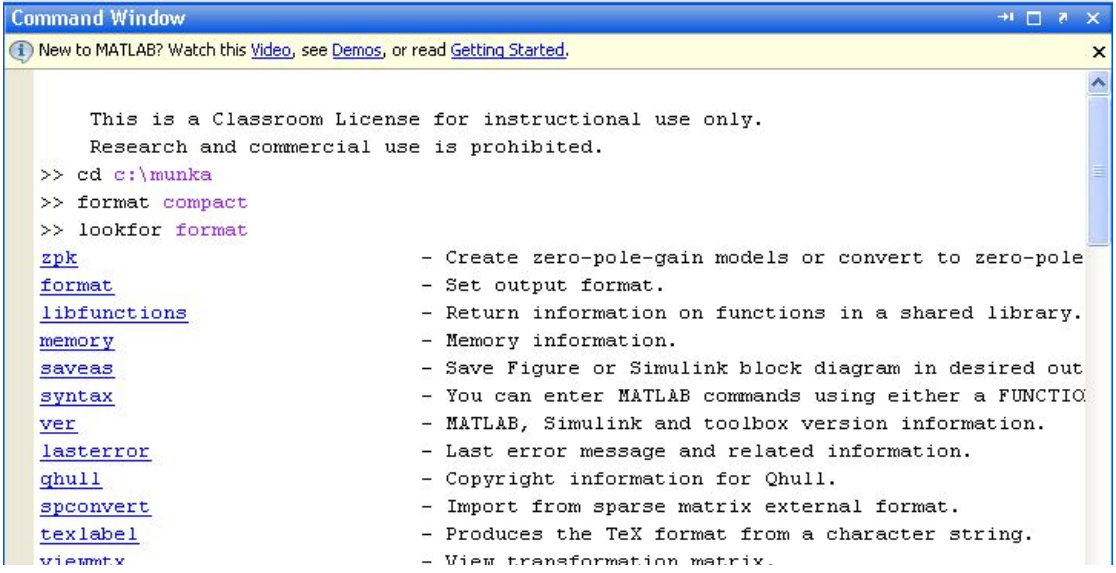
A doc parancs mögé egy másik Matlab parancs, vagy függvény nevét írva a Help-nek erre a parancsra, függvényre fókuszált része, mely példákat is tartalmaz, kinyílik.

```
>> doc inv
```



2.6. ábra: Az inv parancs Helpje

A **lookfor** parancs segítségével egy fogalomra kerestethetünk a help rendszerben. A válaszok közül bármelyiket választva, kinyílik annak a helpje. Például a format parancsra kerestetve több mint száz választ kapunk:



2.7. ábra: A lookfor format keresés eredményei (részlet)

2.1.3. A Matlab elemei

Jelkészlet

A Matlab-ban a következő írásjeleket használhatjuk:

1. Az angol abc kis és nagybetűi, melyek között a Matlab különbséget tesz;
2. Számjegyek;
3. Space;
4. Speciális jelek: _ . , : ; < > / * ^ ~ = ( ) [ ] { } ’ ! @ & |% melyek közül a % jel után kommentárt írhatunk.

A (;) pontosvessző parancszáró írásjel, ha a parancs után kitesszük, akkor a parancs eredménye nem kerül a képernyőre; egyébként pedig az eredmény, amely automatikusan az **ans** (answer) változóba kerül, rögtön ki is jelződik.

Numerikus adatok tárolása

A numerikus adatok tárolása automatikusan az ún. **double** lebegőpontos típusban (lásd IEEE754 szabvány) történik, de a Matlab további típusokat is használ:

1. Előjeles egészek: int8, int16, int32, int64;
2. Előjel nélküli egészek: uint8, uint16, uint32, uint64;
3. Egyszeres pontosságú lebegőpontos: single.

Ha egy értéket nem a double típusban akarunk tároltatni, akkor konvertáltatni kell:

```
>> z = int16(2013); uint32max = uint32(4294967295);
```

Az egyes tárolási típusok határait a 2.1. táblázat értékei tartalmazzák.

2.1. táblázat. Numerikus típusok tárolási határai

típus	**intmin('típus')**	**intmax('típus')**	**bájt**
int8	-128	127	1
uint8	0	255	1
int16	-32768	32767	2
uint16	0	65535	2
int32	-2147483648	2147483647	4
uint32	0	4294967295	4
int64	-9223372036854775808	9223372036854775807	8
uint64	0	18446744073709551615	8
	realmin('típus')	**realmax('típus')**	
single	1.175494e-038	3.402823e+038	4
double	2.2250738585072e-308	1.79769313486232e+308	8

Ha egy típusban a kiértékelés közben túl- vagy alulcsordulás történik, akkor vagy valótlan érték képződik (az ebben a típusban tárolható értékek határa), vagy egy speciális jelentésű kód, az Inf. (A példákban – itt és a későbbiekben is – többször is helytakarékos módon (üres sorokat elhagyva, sorokat összevonva) közöljük a rendszer válaszait.)

```
>> x=int16(30000), y=x+x, z=1e160, w=z*z, s=uint8(6), w=uint8(8), s-w

 x = 30000

 y = 32767

 z = 1.0000e+160

 w = Inf

 s = 6

 w = 8

 ans = 0
```

A **végtelen** speciális érték (Infinity) mellett a Matlab a **nem szám** (Not a Number) fogalmat is használja, például ez jön létre a 0 és a végtelen szorzataként.

```
>> Inf*0

ans =

   NaN
```

Megjegyzések:

1. Az egész típusok számjelzője a tároláshoz használt bitek számát jelenti, például az int32 típusú adat 32 biten tárolódik, és a vezető bit az előjelbit.

2. Az int64 és uint64 típusok adminisztrációs célokat szolgálnak, az ilyen típusú adatokkal matematikai művelet nem végezhető!

Szöveges adatok tárolása

Egy szöveg az írásjeleinek ASCII kódjaival sorvektorban tárolódik:

```
>> szoveg = 'Matlab', int8(szoveg)

szoveg = Matlab

ans =
   77   97  116  108   97   98
```

Láthatjuk, hogy az int8(szoveg) kifejezés eredménye az ans mátrixváltozóba került, és sorvektorként ki is íródott. Mivel egy szöveg az írásjelkódok sorvektoraként tárolódik, vele aritmetikai és mátrixművelet is elvégezhető.

Hasonló, viszont double tárolású sorvektor az eredményünk, ha a double('Matlab') függvényhívást adjuk ki:

```
>> double('Matlab')

ans =

    77    97   116   108    97    98
```

Azt, hogy egy szöveg valójában írásjelenkénti kódokkal tárolódik, egyszerűen ellenőrizhetjük:

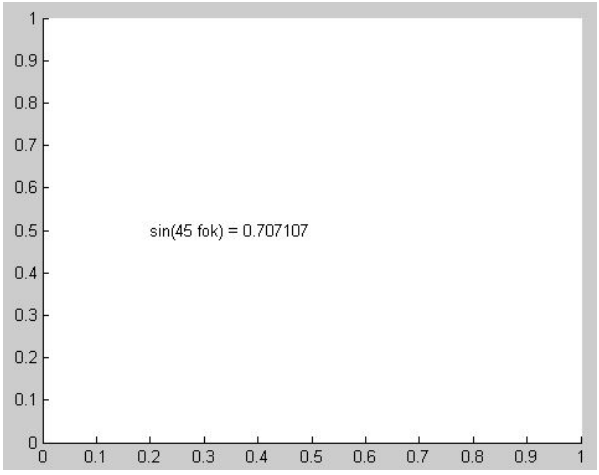
```
>> szoveg + 1  % a sorvektor minden elemét 1-gyel növeljük
```

```
ans =

    78    98   117   109    98    99
```

Szöveges adatot többféle célra használ a Matlab. Az egyik és nyilvánvaló célja a kiírandó szöveg előállítása, tárolása, kiírása. A másik cél a begépelt képletek szövegszerű letárolása, majd elemzése, illetve végrehajtása. A szövegként tárolt képlet az **eval**() függvény segítségével értékelhető ki, illetve hajtatható végre. A következő példánk mindezt tömören illusztrálja.

```
>> x=45; z='sin(x*pi/180)';

s=sprintf('sin(%d fok) = %g', x, eval(z));

text(0.2,0.5,s)
```

Az első sorban a z változóban elhelyeztük szövegesen a kiértékelendő kifejezést, a második sorban az **eval**() függvényhívás előállítja a kiszámított értéket, majd az **sprintf**() függvény a C nyelv szabályainak megfelelő kiíratható karakterláncot rendel az s változóhoz. Ezt a szöveget pedig a **text**() függvény egy grafikus ablakban pozícionáltan írja ki. Ezekről a függvényekről később még lesz szó.



2.8. ábra: Karakterlánc kiírása

Komplex számok

A Matlab a komplex számokat is kezelni tudja, ha azt normál alakban adták meg: 4 + 3i. Az i szimbólum mellett a j szimbólum is (ezt a villamosmérnökök szeretik) az imaginárius egységet jelöli.

```
>> z=4+3i, abs(z), z*z'

z = 4.0000 + 3.0000i

ans = 5

ans = 25
```

Az első sorban egy (,) vessző jelekkel elválasztott kifejezéslistát közöltünk, melynek elemeit a Matlab egymás után kiértékelte. Az első értékadó kifejezés a *z* változóhoz rendelt egy komplex értéket, az abs(z) függvényhívás előállította a komplex szám abszolút értékét, és a harmadik művelet pedig ennek a négyzetét. Itt találkoztunk először a változó mögé írt nagyon fontos (') aposztróf műveleti jellel, mely művelet egy mátrix konjugált transzponáltját állítja elő. Itt viszont az 1x1 méretű mátrix (skalár) esetében persze ez csak a konjugálást jelenti.

Az i szimbólum nem keveredik össze az *i* változóval, a használat módja dönti el az értelmezést. Ha az i azonosítót még nem használtuk változó tárolására, akkor a következő írásmódok használhatók:

```
>> z = 4 + 3i, w = 4 + 3*i  % mögé írva és szorzásjellel is jó

z = 4.0000 + 3.0000i

w = 4.0000 + 3.0000i
```

Ha viszont az i már értéket kapott, akkor a szorzásjel már a változóra vonatkozik.

```
>> i=10, z = 4 + 3i, w = 4 + 3*i

i = 10

z = 4.0000 + 3.0000i

w = 34
```

Az exponenciális függvény segítségével exponenciális alakban is megadhatjuk a komplex számot, amit a Matlab normál alakra hoz, és ezzel az alakkal számol tovább. Az átalakítás szabálya:

$$R*\exp(i*\varphi) = R*\cos(\varphi) + i*(R*\sin(\varphi))$$

ahol R a komplex szám abszolút értéke és φ a radiánban mért szöge.

Az exponenciális alakot az argumentumbeli i-vel szorzás miatt csak akkor használhatjuk, ha i-nek nincs változó szerepe. Éppen ezért nem célszerű az i szimbólum változókénti használata. Ha mégis használjuk, akkor a komplex használat előtt mentjük az értéket, majd felszabadítjuk az *i* változót, és a komplex használat után visszatöltjük.

```
>> i_save = i, clear i; z=2*exp(i*pi/4), i=i_save, w=2*exp(i*pi/4)

i_save = 10

z = 1.4142 + 1.4142i

i = 10

w = 5.1519e+003
```

A **clear** paranccsal a munkaterületi változóinkat egyesével, csoportosan, vagy mindet törölhetjük:

```
clear i,   % csak az i változót törli

clear z*, % csak a z-vel kezdödö azonosítójú változókat törli

clear  % az összes munkaterületi változót törli
```

Természetesen a Workspace ablakbeli kiválasztott változókat az ablakbeli delete művelettel is törölhetjük. (Próbáljuk meg a clear = pi; értékadás után a clear változót másképp törölni!)

A komplex számokkal való munkát további függvények is támogatják. Ilyenek például az **exp**(), **isreal**(), **real**(), **imag**(), **conj**(), **angle**().

```
>> fi=60, r=10, z=r*exp(i*fi*pi/180), valos_e=isreal(z),
a=real(z), b=imag(z), z_konjug=conj(z), abs_z=abs(z)
alfa_rad=angle(z), alfa_fok=alfa_rad/pi*180
fi = 60
r = 10
z = 5.0000 + 8.6603i
valos_e = 0
a = 5.0000
b = 8.6603
z_konjug = 5.0000 - 8.6603i
abs_z = 10
alfa_rad = 1.0472
alfa_fok = 60
```

Logikai értékek

A logikai **igaz** és **hamis** értékek tárolására a szokásos 1 és 0 numerikus értékeket használja a Matlab.

```
>> 'Matlab' == 'Matek ' % karakterkódok páronkénti hasonlítása

ans =

     1     1     1     0     0     0
```

Itt a == hasonlítási műveleti jellel két azonos méretű sorvektort karakterenként hasonlítottunk össze, az eredmény egy ugyanilyen méretű sorvektorba került. Az 1 és 0 logikai ill. numerikus értékek a **true** és **false** alapszavakkal is megadhatók és hivatkozhatók:

```
>> x=true, 3+x

x = 1

ans = 4
```

Szövegek azonosságát viszont egyetlen válaszként az **strcmp**() függvénnyel kaphatjuk meg:

```
>> strcmp('Matlab','Matek')

ans = 0
```

Numerikus értékek, sőt mátrixok is kiértékelhetők logikailag. Skalár esetében minden nem 0 érték igaznak számít, és csak a 0 számít hamisnak.

```
>> a=5, if a fprintf('igaz\n'); else fprintf('hamis\n'); end;

a = 5

igaz
```

Mátrix esetében csak akkor true a logikai értékelés, ha minden eleme nem 0, egyébként false a logikai érték.

```
>> load A.dat; A,  if A fprintf('igaz\n'); else fprintf('hamis\n'); end;

A =

     1     3     6

     2     0    -1

     6     1     2

hamis

>> A(2,2)=1,  if A fprintf('igaz\n'); else fprintf('hamis\n'); end;
```

```
A =

     1     3     6

     2     1    -1

     6     1     2

igaz
```

Dátum-idő értékek

A dátum-idő tárolására a Matlab double típusban tárolt valós számot használ: az 1 jelenti a 0000 január 1-et, és törtszámokkal adhatjuk meg a napon belüli időpontot pl. 0.625 a délután 3 órának felel meg. A dátum-idő megjelenítése az Excel-hez hasonlóan viszont sokféle lehet. A **now**() függvény a pillanatnyi dátum-időt adja vissza, amit a **datevec**() és **datestr**() függvényekkel átalakíthatunk. A datevec() függvény év, hó, nap, óra, perc, másodperc elemű double tárolású elemeket tartalmazó sorvektort állít elő:

```
>> most=now(), dt_str=datestr(most), dt_vec=int16(datevec(most))

most = 7.352695198623727e+005

dt_str = 04-Feb-2013 12:28:36

dt_vec =

   2013      2      4     12     28     36
```

A datevec() double megjelenítését 2 bájtos tárolású egészre konvertálással kerültük ki. A dt_vec-hez hasonló sorvektort eredményez a **clock** beépített változó is:

```
>> int16(clock)

ans =

     2013      2      4     12     31     15
```

Az eltelt idő mérését belső változók, műveletek (**tic**, **toc**, **cputime**) támogatják:

```
>> x=sqrt(3); tic, for i=1:10^8 x=x+1.0000001; end, toc;

x=sqrt(3); tic, for i=1:10^8 x=x*1.0000001; end, toc;

x=sqrt(3); tic, for i=1:10^8 x=x/1.0000001; end, toc;

Elapsed time is 1.528049 seconds.

Elapsed time is 1.521122 seconds.

Elapsed time is 4.698885 seconds.
```

Látható, hogy az összeadás és szorzás műveleti időigénye közel azonos, az osztásé lényegesen több.

Változók, védett alapszavak

A Matlab változói betűvel kezdődő betűszámsorok lehetnek, viszont a kulcsszavak védettek (break, case, colon, continue, else, elseif, end, for, if, otherwise, switch, while). A további gyakori parancsszavakat, függvényeket, belső változókat sem célszerű adatazonosítóként felhasználni (pl. clear, dir, exit, format, help, load, save, eps, realmin, realmax, intmin, intmax, pi, inf, nan, ...).

Ha ez utóbbiak közül valamelyiket pl. a dir-t változóként felhasználtuk, akkor az eredeti funkcióját nem tudjuk mindaddig használni, amíg a változóként nem töröltük (a clear dir parancsot kiadjuk, vagy a workspace ablakban töröljük).

Fontos, hogy a változóinkat célszerű nevekkel azonosítsuk, mert nagyobb munkák esetén ezeket már nehéz áttekinteni.

A munkaterületi változók nevét a **who** paranccsal kérdezhetjük le:

```
>> who

Your variables are:

ans      dt_vec   inverz   most     szoveg   z

dt_str   fi       ludolf   r        x        uint32max
```

A **whos** parancs már részletekkel is szolgál:

```
>> whos
```

Name	Size	Bytes	Class	Attributes
ans	1x1	8	double	
dt_str	1x20	40	char	
dt_vec	1x6	12	int16	
fi	1x1	8	double	
inverz	5x5	200	double	
ludolf	1x1	8	double	
most	1x1	8	double	
r	1x1	8	double	
szoveg	1x6	12	char	
uint32max	1x1	4	uint32	
x	1x1	1	logical	
z	1x1	16	double	complex

A Matlab a C-nyelvekhez hasonlóan különbséget tesz a kisbetűk és a nagybetűk között, ez pedig a kezdő alkalmazóknál sokszor okoz megfejthetetlennek tűnő hibaüzeneteket.

```
>> Z

??? Undefined function or variable 'Z'.
```

Értékek megjelenítési formátuma

Az egész értékek (a double típusban tárolt is) megjelenítése 9 jegyű számokig pontos, ennél több jegy esetén az ún. tudományos formátumot használják.

```
>> 123456789

ans = 123456789

>> 1234567890

ans = 1.2346e+009
```

A megjelenítés jegyeinek számát a format paranccsal szabályozhatjuk. A format long parancs a tizedes pont mögött 14 vagy 15 értékes jegyű kiírást biztosít. A format short parancs visszakapcsol a tizedespont mögötti 4 értékes jegyű kijelzésre.

```
>> format long

>> 1234567890, pi, 1000*pi
```

```
ans = 1.234567890000000e+009

ans = 3.141592653589793

ans = 3.141592653589793e+003
```

Az exponens rész csak akkor kerül kiírásra, ha nem 0. Az exponens mindenkori kiírását az e paraméter biztosítja:

```
>> format long e; pi

ans = 3.141592653589793e+000
```

Ha a g paramétert használjuk, akkor az exponens csak akkor íródik ki, ha szükséges.

```
>>  format long; 100*pi, format long g; ans

ans = 3.141592653589793e+002

ans = 314.159265358979
```

Néhány további érdekesebb formátum:

1. format compact, elnyomja a felesleges és extra soremeléseket;
2. format loose, visszakapcsolja az extra soremeléseket;
3. format hex, a tárolt érték hexadecimális megjelenítése:

```
>> format hex; % extra értékek tárolt kódja hexadecimálisan kijelezve
```

```
uintmax=uint16(65535), uint32max, vegtelen=inf, nem_szam=nan, ludolf=pi, 0

uintmax = ffff

uint32max = ffffffff

vegtelen = 7ff0000000000000

nem_szam = fff8000000000000

ludolf = 400921fb54442d18

ans = 0000000000000000
```

Láthatjuk, hogy a 16-bites tárolás 4 félbájton történik, a double tárolású értékek bitjei pedig az IEEE szabvány szerintiek.

Itt kell megemlíteni, hogy egy valós érték tört illetve lánctört alakú közelítését is elő lehet állítani egy karakterláncban tárolt képletként a **rat**() és **rats**() függvények segítségével. Ez hasonlít az Excel törtalakú megjelenítési formátumához.

```
>> format long g, pi_lanctort=rat(pi), eval(pi_lanctort)

pi_lanctort = 3 + 1/(7 + 1/(16))

ans = 3.14159292035398

>> pi_tort=rats(pi), eval(pi_tort)
```

```
pi_tort = 355/113

ans = 3.14159292035398
```

Itt az **eval()** függvény egy karakterláncban megadott legális, azaz kiértékelhető kifejezés kiértékelését végzi el.

Megjegyzés: ha a pi pontosabb törtalakú közelítését szeretnénk előállítani, akkor a rat() parancsban második paraméterként a kért pontosságot kell megadni. Ekkor az eredmény nem a lánctört karaktersorozata lesz, hanem egy kételemű és egész számokat tartalmazó sorvektor.

```
>> [szamlalo nevezo] = rat(pi, 1e-12), pi_kozelit = szamlalo/nevezo,

hiba=pi_kozelit-pi

szamlalo = 5419351

nevezo = 1725033

pi_kozelit = 3.141592653589815

hiba = 2.220446049250313e-014
```

Változók, tömbök, cellatömbök

Láttuk, hogy egy mátrixváltozó egy értékadó kifejezés(utasítás) hatására jön létre (íródik felül) a munkaterületen. Az értékadó kifejezés jobb oldalán pedig (mátrix)értékkel bíró kifejezés lehet. A kifejezés konstansokból, változókból, függvényekből és zárójelezett kifejezésekből műveleti jelek segítségével épül fel. Az utasításokat és az önálló kifejezéseket vessző, pontosvessző, újsor karakter (Enter) zárja le. A sorbeli utasítások, kifejezések végrehajtását, kiértékelését az Enter lenyomása indítja.

Egy hosszú parancssort **Shift+Enter** kombinációval a parancshatárnál több sorra tördelhetünk a parancsablakban. Hosszú kifejezés tördelési lehetőségét a sorvégi . . . jelsorozat biztosítja.

Először nézzük meg egy mátrixváltozó tömbkonstanssal való feltöltését.

```
>> A=[4 5 6; 5 6 8; 1 7 7], b=[23;30;11]

A =

     4     5     6

     5     6     8

     1     7     7
```

```
b =

    23

    30

    11
```

A tömb elemeit szögletes zárójelbe zárva sorfolytonosan adjuk meg. A sor elemeit (,) vessző jel, vagy space (egy ill. több) választja el egymástól. Az utolsó sor kivételével egy sor végét a (;) pontosvessző jelzi. A példánkban az **A** mátrix 3x3 elemű, és a **b** mátrix pedig valójában egy 3 elemű oszlopvektor.

Korábban már volt arról szó, hogy egy szöveget a karakterei ASC kódjait tartalmazó sorvektorban tárolja a Matlab. A következő példánk is ezt illusztrálja. Az s sorvektorban egyesével illetve karakterláncként adtuk meg az elemeket.

```
>> s = ['E' num2str(4) ' = eye(4)'], s_hossz=length(s), eval(s)

s = E4 = eye(4)

s_hossz = 11
```

```
E4 =

     1     0     0     0

     0     1     0     0

     0     0     1     0

     0     0     0     1
```

A **cella-tömb** elemei nem azonos típusúak, sőt mátrixok, vagy akár cellatömbök is lehetnek. Feltöltéskor a szögletes zárójel helyett kapcsosat kell használni:

```
>> tabla = {'Time', 'Temp'; 12 20; 13 22; 14 25; 15 24}

tabla =

    'Time'    'Temp'

    [  12]    [  20]

    [  13]    [  22]

    [  14]    [  25]

    [  15]    [  24]
```

2.1.4. Műveleti jelek, kifejezések, függvények

Logikai műveletek, függvények

A logikai műveletek, függvények azonos méretű mátrixokra, elempáronként hajtódnak végre. Bemutatásukhoz tekintsük a következő két mátrixot:

```
>> A= [5 -2; 4 3], B=[2 4; 4 -2]

A =

     5    -2

     4     3

B =

     2     4

     4    -2
```

2.2. táblázat. Logikai mátrixműveletek és -függvények

Logikai operátor	Függvénnyel	Jelentés	Eredmény
==	eq(A,B)	egyenlő	0 0 1 0
<	lt(A,B)	kisebb	0 1 0 0
>	gt(A,B)	nagyobb	1 0 0 1
<=	le(A,B)	kisebb, vagy egyenlő	0 1 1 0
>=	ge(A,B)	nagyobb, vagy egyenlő	1 0 1 1
~=	ne(A,B)	nem egyenlő	1 1 0 1
&	and(A,B)	elemenkénti logikai ÉS	1 1 1 1
|	or(A,B)	elemenkénti logikai VAGY	1 1 1 1
~	not(A)	elemenkénti logikai tagadás	0 0 0 0
	xor(A,B)	logikai kizáró VAGY	0 0 0 0
	any(A)	vektorra: van-e benne nem 0? mátrixra: oszlopokon értékeli ugyanezt, sorvektort ad	1 1
	all(A)	hasonlóan működik, mint az any() értékelés művelete: mind nem 0?	1 1

Nézzünk egy példát az any() és all() függvényekre:

```
>> A=[5 2 1; 2 6 0; 1 0 4], al=all(A), alal=all(all(A)), an=any(A),
anan=any(any(A))
A =
     5     2     1
     2     6     0
     1     0     4
al =
     1     0     0
alal = 0
an =
     1     1     1
anan = 1
```

Skalárokra is használhatók logikai műveletek, de a paletta itt „szegényesebb”.

2.3. táblázat. Skalár-logikai operátorok

Logikai operátor	Jelentés
~	numerikus érték logikai tagadása
&&	logikai ÉS (skalárokra)
||	logikai VAGY (skalárokra)

```
>> ~2013, ~0 % nem 0 tagadása 0, 0 tagadása 1

 ans = 0

 ans = 1
```

A logikai műveletek kiértékelésekor az ún. rövidzárt használja a Matlab, azaz ha egy kifejezés eredményét az első operandus meghatározza, akkor a többi operandus már nem értékelődik ki:

```
 >> p=[], ures=isempty(p)

 p = []

 ures = 1

 >> ures ||  p<2

 ans = 1

 >> p<2 ||  ures
```

```
??? Operands to the || and && operators must be convertible to logical scalar
values.
```

Aritmetikai műveletek

A Matlab-nak két különböző típusú aritmetikai operátora van:

1. Mátrix-aritmetikai operátorok (lineáris algebrában tanult műveleteket hajtanak végre);
2. Tömb-aritmetikai operátorok (elemenként, elempáronként hajtódnak végre).

A mátrix-aritmetikai műveleti jel elé írt (.) pont karakter különbözteti meg a tömb-aritmetikai operátorokat a mátrix operátoroktól. A mátrix és a tömb aritmetika az összeadás és a kivonás esetén nem különbözik egymástól, ezért a .+ és a .- tömb-aritmetikai műveletek nem léteznek.

Az aritmetikai műveletek bemutatásához a fentiekhez hasonlóan két mátrixot használunk. A mátrixok méreteit úgy választjuk meg, hogy a műveletek elvégezhetők legyenek.

```
>> A= [5 -2; 4 3], B=[2 4; 1 -2]

A =

     5    -2

     4     3
```

```
B =

     2     4

     1    -2
```

2.4. táblázat. Mátrix-aritmetikai operátorok

Mátrix	Jelentés	Eredmény
A + B	összeadás	7 2 5 1
A - B	kivonás	3 -6 3 5
A*B	(mátrix)szorzás	8 24 11 10
A/B	osztás jobbról A*inv(B)	1.0000 3.0000 1.3750 1.2500
A\B	osztás balról inv(A)*B	0.3478 0.3478 -0.1304 -1.1304
A ^ 2	hatványozás A*A	17 -16 32 1
A’	konjugált transzponálás	5 4 -2 3

Megjegyzések:

1. B értéke skalár is lehet, ilyenkor a művelet az A mátrix minden elemén végrehajtódik.
2. Ha B invertálható, akkor elméletileg A/B == A*inv(B).

3. Ha A invertálható, akkor elméletileg A\B == inv(A)*B.
4. Ha az A vagy B mátrixok inverze nem létezik, akkor az A\B illetve A/B műveletek a *Warning: Matrix is singular to working precision* hibaüzenetet eredményezik.

A fenti 2. és 3. megjegyzés a számítási-kerekítési hibák miatt nem mindig teljesül:

```
>> A^B, inv(A)*B, A^B == inv(A)*B

ans =

    0.3478    0.3478

   -0.1304   -1.1304

ans =

    0.3478    0.3478

   -0.1304   -1.1304

ans =

     1     0

     1     1
```

2.5. táblázat. Tömb-aritmetikai operátorok

Tömb	Jelentés	Eredmény
A.*B	elempáronkénti szorzás	10 -8 4 -6
A./B	elempáronkénti osztás jobbról	2.5000 -0.5000 4.0000 -1.5000
A.\B	elempáronkénti osztás balról	0.4000 -2.0000 0.2500 -0.6667
A.^B	elempáronkénti hatványozás	25.0000 16.0000 4.0000 0.1111
A.^2	elemenkénti hatványozás	25 4 16 9
2.^A	elemenkénti hatványozás	32.0000 0.2500 16.0000 8.0000
A.'	csak transzponálás	5 4 -2 3

Hasonlítsuk össze a csak transzponálás műveletét a konjugált transzponálás műveletével komplex elemű mátrixon!

```
>> A = [exp(i*pi/3), exp(i*pi/4); exp(i*pi/6), exp(i*pi/2)]

A =

   0.5000 + 0.8660i   0.7071 + 0.7071i

   0.8660 + 0.5000i   0.0000 + 1.0000i
```

```
>> A'

ans =

   0.5000 - 0.8660i   0.8660 - 0.5000i

   0.7071 - 0.7071i   0.0000 - 1.0000i

>> A.'

ans =

   0.5000 + 0.8660i   0.8660 + 0.5000i

   0.7071 + 0.7071i   0.0000 + 1.0000i
```

Az aritmetikai operátorok közé soroljuk még a fentieken túl a sorozatképző operátorokat (colon) is.

2.6. táblázat. Sorozatképző operátorok

Operátor	Jelentés	Példa
:	sorozatképzés sorvektorként, lépésköz 1	» 4:6 ans = 4 5 6
: :	sorozatképzés sorvektorként, lépésköz adott	» 4:2:11 ans = 4 6 8 10

A műveletek rangsora a Matlabban a következő:

1. Hatványozás;
2. Előjel: +, −;
3. Szorzás, osztás: *, .*, /, ./, \;
4. Összeadás, kivonás: +, −;
5. Sorozatképzés.

Ha ettől és az ún. balról-jobbra szabály szerinti végrehajtástól el akarunk térni, akkor kellőképp zárójelezni kell! Jegyezzük meg, hogy a felesleges, de jól elhelyezett zárójelpárok a számítási eredményt nem változtatják meg.

Vizsgáljuk meg a következő példát!

```
>> c = 5, x = 2 + c:10, y = 2 + (c:10)

c = 5

x =

     7     8     9    10

y =

     7     8     9    10    11    12
```

Mivel az összeadás nagyobb precedenciájú művelet, mint a sorozatképzés, ezért az x sorozat képzése 5+2-ről indul és 10-ig tart. Az y sorozatnál először az [5 6 7 8 9 10] sorozat készül el, majd ennek minden eleme 2-vel növelődik.

A sorozatképzés üres sorozatot is generálhat:

```
>> x=10:0, urese=isempty(x), merete=size(x)

x =

   Empty matrix: 1-by-0

urese = 1

merete =

     1     0
```

A mátrixműveleteket elsősorban a lineáris algebrai feladatnál tudjuk jól felhasználni. Hozzuk létre a következő **A** és **b** mátrixokat:

```
>> A=[4 5 6; 5 6 8; 1 7 7], b=[23;30;11]

A =

     4     5     6

     5     6     8

     1     7     7
```

```
b =

    23

    30

    11
```

Tekintsük azt a lineáris egyenletrendszert, amely tömören az $\mathbf{Ax} = \mathbf{b}$ alakban írható fel. Ennek megoldása formálisan $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

Matlab-ban a megoldást a fenti 3. megjegyzésünk (a 2.4. táblázathoz kapcsolódóan) szerint az A\b művelettel állíthatjuk elő:

```
>> x = A\b

x =

    4.000000000000000e+000

   -1.000000000000000e+000

    2.000000000000000e+000
```

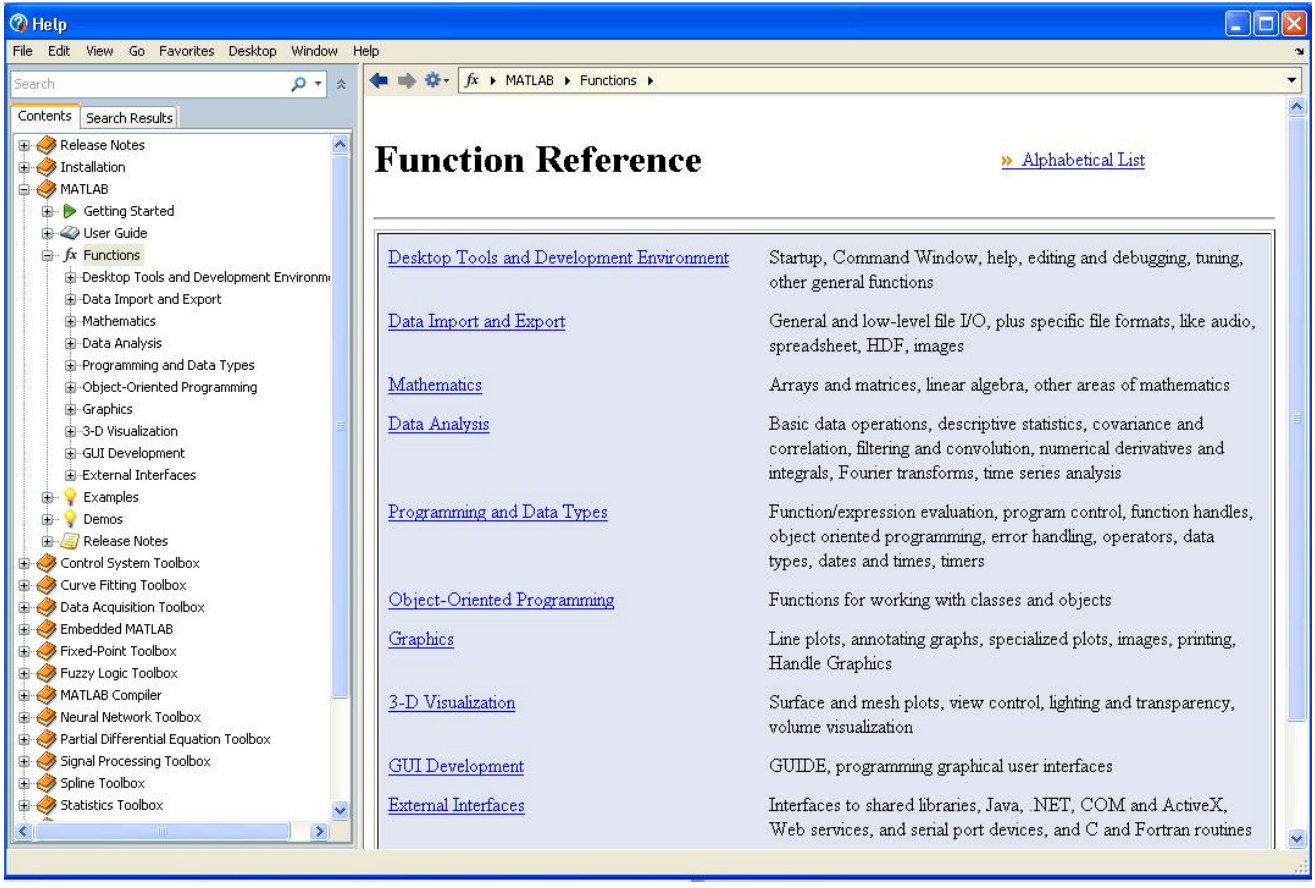
Végezzük el az ellenőrzést is!

```
>> A*x, ans == b

ans =

    2.300000000000000e+001

    3.000000000000000e+001

    1.100000000000000e+001

ans =

     1

     1

     0
```

Annak ellenére, hogy a megoldás jónak tűnik, az egyenletrendszer harmadik egyenletének teljesülését egy pici kerekítési hiba megzavarta. Ebből azt a következtetést vonhatjuk le, hogy az == egyenlőségi relációt a közel azonos értékű double típusú változók esetében nagyfokú óvatossággal kell használni.

2.1.5. Függvénycsaládok

A Matlab igen sok beépített függvénnyel rendelkezik, és ahogy a bevezetőben láttuk, a függvénykészletet M-fájlok segítségével mi magunk is tetszés szerint bővíthetjük. A Matlab függvényei több családba sorolhatók. Ennek megfelelően a Help rendszerben is könnyebben tájékozódhatunk.



2.9. ábra: A Matlab függvénycsaládjai

A függvények közül a mi alapozó kurzusunkban csak a legfontosabbakkal foglalkozunk, azaz a

1. Matematikai;
2. Adatelemzést végző;
3. Grafikai;
4. Input, output

függvények egy részével fogunk dolgozni. Bármelyik függvényt is szeretnénk használni, előtte érdemes a

```
>> help fnev  % a függvény neve
```

vagy a

```
>> doc fnev
```

parancsot kiadni. Sok jó ötletet kaphatunk.

A következőkben a kijelölt parancsokkal és függvényekkel elsősorban konkrét feladatcsoportok kapcsán ismerkedünk meg részletesebben.

2.1.6. Elemi matematikai függvények

A Matlab a programozási nyelvekhez hasonlóan beépített rutinként tartalmazza az elemi matematikai függvényeket. Ezek listáját a **help elfun** parancs előhozza (a trigonometrikus, exponenciális és komplex listának helytakarékossági okból csak az elejét közöljük).

```
>> help elfun

  Elementary math functions.

  Trigonometric.

    sin         - Sine.

    sind        - Sine of argument in degrees.

    sinh        - Hyperbolic sine.

    asin        - Inverse sine.

    asind       - Inverse sine, result in degrees.

    asinh       - Inverse hyperbolic sine.

    cos         - Cosine.

    cosd        - Cosine of argument in degrees.

    cosh        - Hyperbolic cosine.

    acos        - Inverse cosine.

    acosd       - Inverse cosine, result in degrees.
```

```
  acosh       - Inverse hyperbolic cosine.
  tan         - Tangent.
  ...           ...
Exponential.
  exp         - Exponential.
  expm1       - Compute exp(x)-1 accurately.
  log         - Natural logarithm.
  log1p       - Compute log(1+x) accurately.
  log10       - Common (base 10) logarithm.
  log2        - Base 2 logarithm and dissect floating point number.
  pow2        - Base 2 power and scale floating point number.
  realpow     - Power that will error out on complex result.
  reallog     - Natural logarithm of real number.
  ...           ...
```

```
Complex.

  abs          - Absolute value.

  angle        - Phase angle.

  complex      - Construct complex data from real and imaginary parts.

  conj         - Complex conjugate.

  imag         - Complex imaginary part.

  real         - Complex real part.

  ...            ...

Rounding and remainder.

  fix          - Round towards zero.

  floor        - Round towards minus infinity.

  ceil         - Round towards plus infinity.

  round        - Round towards nearest integer.

  mod          - Modulus (signed remainder after division).
```

```
rem          - Remainder after division.

sign         - Signum.
```

Itt nem részletezzük ezeket, hanem csak akkor, ha ez szükségessé válik. Annyit azonban megjegyezhetünk, hogy skalár (1x1-es mátrix) argumentumra a szokásos módon működnek. Vektor vagy mátrix argumentummal a számítás többnyire elemenként történik, azaz az eredmény mérete az argumentum méretével egyezik. Amennyiben problémánk van a működést illetően, akkor nézzünk utána a help-ben!

2.1.7. Adatok mentése, visszatöltése

A munkaterületi adatainknak fáljba mentése vagy a save paranccsal, vagy más alkalmazásba való exporttal pl. **xlswrite**() függvénnyel történhet.

```
>> clear, A=[4 5 6; 5, 6, 8; 1 7 7], b=[23;30;11], save

A =

     4     5     6

     5     6     8

     1     7     7
```

```
b =

    23

    30

    11

Saving to: matlab.mat
```

Itt a **clear** parancs a munkaterület korábbi változóit törölte. Majd az **A** és **b** mátrixokat létrehoztuk, és ezután a teljes munkaterület tartalmát a **save** parancs egy speciális **matlab.mat** nevű és az adatainkat bináris formában tartalmazó (információvesztés nélküli) fájlba mentette.

Ellenőrizzük **load** visszatöltéssel a mentésünket:

```
>> clear, load

Loading from: matlab.mat
```

Itt a clear, save, load parancsokat a legegyszerűbb formában használtuk. A save paranccsal nem csak a teljes munkaterület tartalmát lehet menteni, hanem a célfájl nevét, típusát, a mentendő adatainkat és a mentett adatok tárolási formátumát is megadhatjuk. Ha a típust nem adjuk meg, akkor automatikusan egy .mat típusú fájlba menti a binárisan tárolt adatokat. Egy .mat fájl a mentett adatokon kívül azok azonosítóját is tartalmazza, ezért a visszatöltéskor a változó neve automatikusan használódik fel.

```
>> save egyenlet.mat A b
```

Most nézzünk egy példát text fájlba mentésre!

```
>> P=A+10*eps, q=b+10*eps, save adatok.txt P q -ascii -double -tabs

P =

    4.0000    5.0000    6.0000

    5.0000    6.0000    8.0000

    1.0000    7.0000    7.0000

q =

   23.0000

   30.0000

   11.0000
```

A parancsunk a következő tartalmú **adatok.txt** fájlt hozta létre:

```
4.0000000000000036e+000 5.0000000000000036e+000 6.0000000000000036e+000

5.0000000000000036e+000 6.0000000000000036e+000 8.0000000000000036e+000

1.0000000000000044e+000 7.0000000000000036e+000 7.0000000000000036e+000

2.3000000000000007e+001
```

```
3.0000000000000007e+001

1.1000000000000004e+001
```

A parancs mindkét, **P** és **q** mátrixunkat mentette, az adatok 16 értékes jegyet tartalmazó formában mentődtek, a soron belüli adat-elválasztójel pedig a tabulátor. Azt is látjuk, hogy az eps igen picit megnövelte az eredetileg egész értékeket.

Az **eps** belső változó a double típus relatív számítási pontosságát jelzi, definíciója: az a legkisebb pozitív double típusban tárolt szám, amit 1-hez adva már 1-nél nagyobb értéket kapunk.

```
>> eps

ans = 2.220446049250313e-016
```

Az adatok.txt fájllal egyetlen baj van, nem lehet jól visszatölteni az adatainkat. A load parancs szöveges fájlból csak egyetlen téglalap alakú táblázatot képes beolvasni!

```
>> load adatok.txt -ascii

??? Error using ==> load

Number of columns on line 3 of ASCII file c:\munka\adatok.txt

must be the same as previous lines.
```

A probléma megoldása egyszerű, a mátrixainkat külön-külön mentjük:

```
>> save PP.dat P -ascii, save qq.dat q -ascii
```

Töltsük vissza a mentett P és q adatainkat.

```
>> load PP.dat -ascii, load qq.dat -ascii
```

A betöltött adatok neve a fájlok nevéből adódik! Hasonlítsuk össze a mentett P és q adatokat a visszatöltött PP és qq adatokkal!

```
>> PP == P, qq == q

ans =

     0     0     0

     0     0     0

     0     0     0

ans =

     0

     0

     0
```

Azt látjuk, hogy elemenként nem azonosak! Hol a hiba? A hiba a mentés opcióiban van. Ha azt akarjuk, hogy egy text fájlba mentett adatunk visszatöltéskor pontosan ugyanazt az értéket kapja vissza, akkor a -double opcióról nem szabad megfeledkeznünk! Ez biztosítja ugyanis a 16 értékes jegyre történő (és pontos) megjelenítést.

```
>> save PP.dat P -ascii -double, save qq.dat q -ascii -double

>> load PP.dat -ascii, load qq.dat -ascii, PP == P, qq == q

ans =

     1     1     1

     1     1     1

     1     1     1

ans =

     1

     1

     1
```

Ennyi kísérletezés után törölhetjük a P, PP, q, qq változóinkat!

```
>> clear P* q* ans
```

A változókat mentő, visszatöltő és törlő parancsokban lehet a (*) csillag írásjelet mint joker-szimbólumot használni. Jelentése: tetszőleges folytatás.

Ezek után felmerül a kérdés, milyen fájlba mentsük a munkaterületi változóinkat? Ha az adatainkon továbbra is a Matlab segítségével óhajtunk tovább dolgozni, akkor egyértelműen a .mat fájlokat célszerű használni. A text fájlok használata más feldolgozó rendszerekkel kapcsolatban javasolt, mert ezek legtöbbje képes a jól tagolt

szöveges fájlok beolvasására és létrehozására. Itt például a Labview rendszer által kibocsátott eredményfájlok feldolgozására is gondolhatunk. A Matlab az Excel rendszerrel kapcsolatban saját export/import függvényekkel rendelkezik.

Példaként a munkaterületen megmaradt **A** és **b** mátrixainkat fogjuk menteni Excel táblázatba, és onnét vissza is töltjük:

```
>> xlswrite('egyenlet.xls', [A b], 'Ab')
```

Az első paraméter a fájl neve, a második paraméter a mentendő mátrix, ami kifejezéssel is megadható, a harmadik és nem kötelező paraméter a füzetlap azonosítója. A második paraméterben megadott [A b] mátrixkifejezés az **A** és **b** mátrixok egyesítését jelenti, azaz a 3x3-as **A** mátrix mögé a 3x1-es **b** oszlopvektort illesztettük. Így egy 3x4-es mátrixot kaptunk és ezt mentettük el.

Visszatöltéskor a füzetlap más-más tartományából külön-külön nyerjük vissza az adatainkat.

```
>> clear

>> A = xlsread('egyenlet.xls', 'Ab', 'A1:C3')

A =

     4     5     6

     5     6     8

     1     7     7

>> b = xlsread('egyenlet.xls', 'Ab', 'D1:D3')
```

```
b =

    23

    30

    11
```

A **dlmread**() és **dlmwrite**() parancsokkal delimiterrel tagolt ASCII fájlokba menthetjük a mátrixainkat, és tölthetjük onnét vissza. A különféle fájltípusok mentéséhez és betöltéséhez lásd a

```
>>doc fileformats
```

segítséget.

Önellenőrzés

1. Mennyi decimális számjegyet jelezhetünk ki a törtrészben a **format long** parancs kiadása után?
   A megoldás megtekintéséhez kattintson ide!
2. Melyik egész típusú adatokkal nem lehet aritmetikai műveleteket elvégezni?
   A megoldás megtekintéséhez kattintson ide!
3. Milyen nagyságrendű a **realmax**('double') értéke?
   A megoldás megtekintéséhez kattintson ide!
4. Lehet-e a pi azonosító változó? Ha igen, akkor írjuk fel azt az értékadást, amely után a helyes érték tárolódik!
   A megoldás megtekintéséhez kattintson ide!
5. Igaz-e, hogy egyetlen numerikus értéket a Matlab skaláris típusú változóban tárol?
   A megoldás megtekintéséhez kattintson ide!
6. Ha z 1x1-es mátrix, akkor mennyit ér az angle(z) + angle(z') kifejezés értéke?
   A megoldás megtekintéséhez kattintson ide!
7. Legyen z = 3 + 4i. Mennyit ér a z* z' ?
   A megoldás megtekintéséhez kattintson ide!
8. Lehet-e a clear=2013; értékadás után a clear változót a **clear** paranccsal törölni?
   A megoldás megtekintéséhez kattintson ide!
9. Adjunk úgy értéket az x változónak, hogy az **isempty**(x) értéke igaz legyen!
   A megoldás megtekintéséhez kattintson ide!
10. Mit jelent logikai műveleteknél a rövidzár?
    A megoldás megtekintéséhez kattintson ide!

11. Mikor lesz egy A mátrixra az **any**(**any**(A)) kifejezés hamis?
A megoldás megtekintéséhez kattintson ide!

12. Hogyan lehetséges az, hogy egy numerikus x értékre a 2*x/2 kifejezés nem lesz egyenlő x-szel?
A megoldás megtekintéséhez kattintson ide!

13. Ha az s változó értékét s='1+2+3' karakterlánccal megadjuk, mi lesz az eval(s)/2 értéke?
A megoldás megtekintéséhez kattintson ide!

14. Mit eredményez az 1 + eps/2 - 1 kifejezés?
A megoldás megtekintéséhez kattintson ide!

8. LECKE

Mátrixok, programozás
Felhasználói függvények

2.2. Mátrixok kezelése

2.2.1. Mátrix és elemhivatkozások

Az eddigiekben már láttuk, hogy egy mátrixot az elemeivel adunk meg és így hozzuk létre. Az elemek megadása direkt felsorolással, vagy mátrixértékű kifejezéssel történhet, sőt részmátrixokból is összerakhatjuk:

```
>> A=[1 2; 3 4], B=[10 20; 30 40], D=-[1 2; 3 4], E=2*B, X=[A B; D E]

A =

     1     2

     3     4

B =

    10    20

    30    40

D =

    -1    -2

    -3    -4
```

```
E =

    20    40

    60    80

X =

     1     2    10    20

     3     4    30    40

    -1    -2    20    40

    -3    -4    60    80
```

Például az a' oszlopvektor és b sorvektor mátrixszorzata (diadikus szorzat):

```
>> a = [1:5], b=[10:5:30], AB = a'*b

a =

     1     2     3     4     5

b =

    10    15    20    25    30
```

```
AB =

    10    15    20    25    30

    20    30    40    50    60

    30    45    60    75    90

    40    60    80   100   120

    50    75   100   125   150
```

Egy mátrix valamelyik elemére ún. indexelt módon hivatkozunk, és ennek az elemnek értéket is adhatunk:

```
>> AB(1, 5)

ans = 30

>> AB(7, 2)

??? Index exceeds matrix dimensions.

>> AB(7, 2)= -1
```

```
AB =

    10    15    20    25    30

    20    30    40    50    60

    30    45    60    75    90

    40    60    80   100   120

    50    75   100   125   150

     0     0     0     0     0

     0    -1     0     0     0
```

Az indexelt elemhivatkozás indexeit kerek zárójelpárba írjuk és vesszővel választjuk el. Nem létező elemre hivatkozás hibajelzést okoz, viszont egy nem létező elemre történő értékadás a mátrix bővülését eredményezi. A bővülés során nem definiált elemek 0-val töltődnek fel.

Hivatkozni nem csak egy mátrixelemre, hanem részmátrixra is lehet. Például a sorozatképző operátor segítségével egy teljes sorra, vagy teljes oszlopra is hivatkozhatunk:

```
>> AB(1,:), AB(:,2)

ans =

    10    15    20    25    30
```

```
ans =

    15

    30

    45

    60

    75

     0

    -1
```

Vigyázat! Egy vektor összes elemére hivatkozás mindig oszlopvektort eredményez:

```
>> b(:)'

ans =

    10    15    20    25    30
```

Tetszőleges sor illetve oszlop kiválasztásával részmátrixra hivatkozhatunk:

```
>> AB( [5:7], : )

ans =
```

```
    50    75   100   125   150

     0     0     0     0     0

     0    -1     0     0     0

>> AB( [5:7], [2 4] )

ans =

    75   125

     0     0

    -1     0
```

Azt, hogy egy mátrix vagy vektor milyen méretű a **size**() vagy **length**() függvénnyel kérdezhetjük le:

```
>> meret=size(AB), sorok=length(AB(:,1)), oszlopok=length(AB(1,:))

meret =

     7     5

sorok = 7

oszlopok = 5
```

Részmátrixhoz (sorokhoz illetve oszlopokhoz) üres mátrix rendelése ezek törlését eredményezi. Töröljük az AB mátrix utolsó két sorát!

```
>> AB( 6:7, : )=[]

AB =

    10    15    20    25    30

    20    30    40    50    60

    30    45    60    75    90

    40    60    80   100   120

    50    75   100   125   150
```

Ha mátrixváltozóhoz az üres mátrixot rendeljük, akkor az 0 sorral és 0 oszloppal fog rendelkezni. Ha viszont egy üres sorozatot rendelünk hozzá, akkor annak 1 sora és 0 oszlopa lesz.

```
>> P=[], P_size = size(P), Q = [1:0], R = [1:0]', S = [Q; Q]

P = []

P_size =

     0     0
```

```
Q =

   Empty matrix: 1-by-0

R =

   Empty matrix: 0-by-1

S =

   Empty matrix: 2-by-0
```

A mátrix speciális részeire függvényekkel hivatkozhatunk (**diag**, **tril**, **triu** függvények). A diag függvény a mátrix főátlójára, vagy vele párhuzamos valamelyik mellékátlójára hivatkozik, és annak elemeit helyezi el egy oszlopvektorban. A diag függvény második paramétere az átlószám, azt adja meg előjelesen, hogy milyen messze vagyunk a főátlótól.

```
>> diag(AB), diag(AB,1)', diag(AB,-2)'

ans =

    10

    30

    60

   100
```

```
    150

ans =

     15     40     75    120

ans =

     30     60    100
```

A diag függvény oszlopvektorra is alkalmazható. Ekkor egy megfelelő méretű négyzetes mátrix megjelölt átlójában helyezi el az oszlopvektor elemeit:

```
>> diag(diag(AB)) + diag(diag(AB,1),1) + diag(diag(AB,-2),-2)

ans =

     10     15      0      0      0
      0     30     40      0      0
     30      0     60     75      0
      0     60      0    100    120
      0      0    100      0    150
```

A tril és triu (l – lower, u – upper) függvények a mátrix alsó vagy felső háromszög alakú részét emelik ki, beleértve a főátlót is. Ha második paraméterként átlószám is szerepel, akkor a kiemelés a megjelölt mellékátlóig tart:

```
>> triu(AB), tril(AB)

ans =

    10    15    20    25    30
     0    30    40    50    60
     0     0    60    75    90
     0     0     0   100   120
     0     0     0     0   150
```

```
ans =

    10     0     0     0     0

    20    30     0     0     0

    30    45    60     0     0

    40    60    80   100     0

    50    75   100   125   150
```

A következő kifejezések az AB mátrixot három diszjunkt részre szeletelik, amelyek összegeként visszakapjuk az eredeti AB mátrixot.

```
>> fent=triu(AB,1), lent=tril(AB,-1), atlo=diag(diag(AB)), fent+lent+atlo

fent =

     0    15    20    25    30

     0     0    40    50    60

     0     0     0    75    90

     0     0     0     0   120

     0     0     0     0     0
```

```
lent =

     0     0     0     0     0

    20     0     0     0     0

    30    45     0     0     0

    40    60    80     0     0

    50    75   100   125     0

atlo =

    10     0     0     0     0

     0    30     0     0     0

     0     0    60     0     0

     0     0     0   100     0

     0     0     0     0   150
```

```
ans =

    10    15    20    25    30

    20    30    40    50    60

    30    45    60    75    90

    40    60    80   100   120
```

2.2.2. Speciális mátrixok

A Matlab számos speciális mátrixot függvényként szolgáltat, vagy a gallery mátrixgyűjteményből állít elő (lásd **help gallery**). A fontosabbakat a következőkben bemutatjuk.

Az egységmátrix főátlójában minden elem 1, azon kívül minden elem 0. Ezt az **eye**(méret), **eye**(sorméret, oszlopméret), **eye**(méretadatok, 'tárolási típus') formában adhatunk meg:

```
>> eye(4), eye(3,4),eye(3,'int8'),whos ans

ans =

     1     0     0     0

     0     1     0     0

     0     0     1     0

     0     0     0     1
```

```
ans =

     1     0     0     0

     0     1     0     0

     0     0     1     0

ans =

     1     0     0

     0     1     0

     0     0     1

  Name        Size                Bytes  Class     Attributes

  ans         3x3                     9  int8
```

A(z) **ones**() függvény csupa 1-el feltöltött mátrixot ad, paraméterezése hasonló az eye() függvényéhez, de itt 2-nél több dimenziós mátrix (pl. tégla) is előállítható:

```
>> egy2x3x4=ones(2,3,4), egy_byte=ones(3, 'int8'), whos egy_byte
```

```
egy2x3x4(:,:,1) =

     1     1     1

     1     1     1

egy2x3x4(:,:,2) =

     1     1     1

     1     1     1

egy2x3x4(:,:,3) =

     1     1     1

     1     1     1


egy2x3x4(:,:,4) =

     1     1     1

     1     1     1
```

```
egy_byte =

     1     1     1

     1     1     1

     1     1     1

  Name           Size              Bytes  Class     Attributes

  egy_byte       3x3                   9  int8
```

A **zeros**() függvénnyel csupa 0 elemet tartalmazó mátrix állítható elő, amelynek paraméterezése az ones() függvénnyel azonos.

A **linspace()** parancs egyenlőközű sorvektor előállítására szolgál.

```
K = linspace(10,1000), L = linspace(10, 15, 11)
```

A kétparaméteres linspace(tól, ig) verzió mindig egy 100-elemű sorvektort generál a megadott intervallumban. A második verzió – megadható – harmadik paramétere a generált elemek száma:

```
L =

  Columns 1 through 8

 10.0000   10.5000   11.0000   11.5000   12.0000   12.5000   13.0000   13.5000
```

```
  Columns 9 through 11

   14.0000   14.5000   15.0000
```

A Hilbert mátrix – amelyben a természetes számok reciprokai szerepelnek – a Matlabban a **hilb**(méret) függvénnyel állítható elő. Ezeket a törteket mi is megadhatjuk, persze bonyolultabban:

```
>> H4 = [1./[1:4];1./[2:5];1./[3:6];1./[4:7]]

H4 =

    1.0000    0.5000    0.3333    0.2500

    0.5000    0.3333    0.2500    0.2000

    0.3333    0.2500    0.2000    0.1667

    0.2500    0.2000    0.1667    0.1429

>> hilb(4)
```

```
ans =

    1.0000    0.5000    0.3333    0.2500

    0.5000    0.3333    0.2500    0.2000

    0.3333    0.2500    0.2000    0.1667

    0.2500    0.2000    0.1667    0.1429

>> H4 == hilb(4)

ans =

     1     1     1     1

     1     1     1     1

     1     1     1     1

     1     1     1     1
```

A Pascal háromszög a binomiális együtthatók sorozatát tartalmazó mátrix. A Matlabban a **pascal()** parancs állítja elő, a háromszög csúcsa az (1, 1) pozícióban van:

```
>> pascal(5)

ans =

     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70
```

A Vandermonde mátrix oszlopaiban egy oszlopvektor csökkenő kitevőjű hatványai állnak. Előállítására a **vander()** parancs szolgál:

```
>> x= [1,2,3,4]', v4=vander(x), [x.^3, x.^2, x.^1, x.^0]

x =

     1
     2
     3
     4
```

```
v4 =

     1     1     1     1

     8     4     2     1

    27     9     3     1

    64    16     4     1

ans =

     1     1     1     1

     8     4     2     1

    27     9     3     1

    64    16     4     1
```

A Matlabban a véletlenszám-generálás is tevékenységei is a mátrixfüggvények közé sorolódnak. A végrehajtás többnyire a [0; 1) intervallumban egyenletes eloszlású adatokat generáló **rand**() és standard normális eloszlású adatokat generáló **randn**() függvényekkel történik. Az eredmény oszlopvektorba (oszlopvektorokba) kerül. Példaként generáljunk egy kétoszlopos és oszloponként 1000 elemű standard normális eloszlású mintát, majd kérdezzük le a minta átlagát és tapasztalati szórását!

```
>> x=randn(1000, 2); atlag=mean(x), szoras=std(x)
```

```
atlag =

    0.0306   -0.0590

szoras =

    0.9762    1.0089
```

Látható, hogy a várható érték becslése 0 körüli és a szórás becslése 1 körüli eredményt produkált.

Adatoszlopokban elhelyezett adatminták feldolgozása

Az adatoszlopok feldolgozását támogató függvényeket a 2.7. táblázatban mutatjuk be.

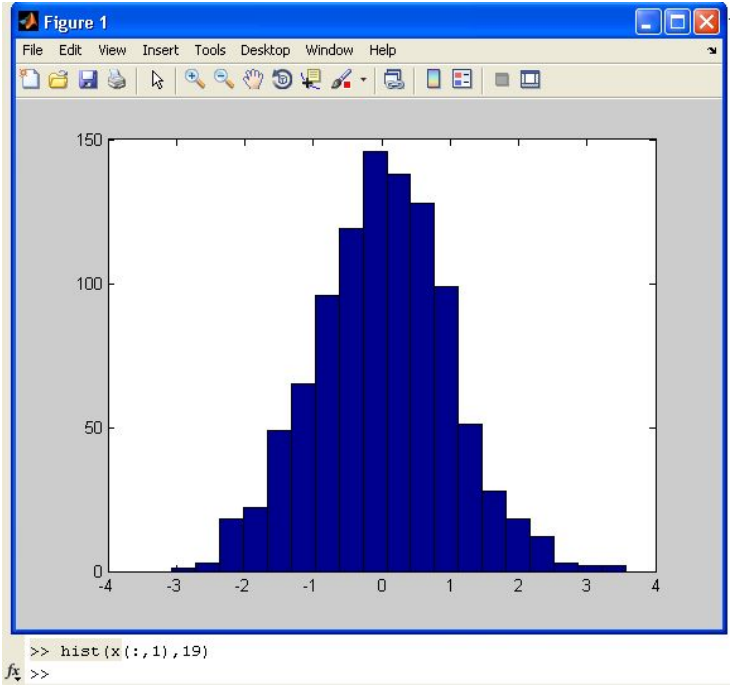
2.7. táblázat. Adatoszlopokban elhelyezett minták feldolgozására szolgáló függvények

Függvény	**Jelentés**
max	Maximális elem
min	Minimális elem
mean	Átlag
median	Rendezett minta közepe
std	Tapasztalati szórás
sum	Összeg
prod	Elemek szorzata
cumsum	Kumulatív részösszegek
cumprod	Kumulatív részszorzatok
sort	Növekvő sorrendbe rendezés
diff	Elemek differenciái
hist	Hisztogram
corrcoef	Korrelációs mátrix
cov	Kovariancia mátrix

Például a **corrcoef**(mátrix) függvény az adatoszlopok elemeiből, mint a valószínűségi változókra vonatkozó megfigyelt értékekből a korrelációs mátrix becslését szolgáltatja:

```
>> korrelacio = corrcoef(x)

    1.0000    0.0006

    0.0006    1.0000
```

Láthatjuk, hogy az **x** adatmátrix két adatoszlopa gyakorlatilag függetlennek tekinthető, mert a korrelációs együttható becsült értéke 0-hoz közeli.



2.10. ábra: Hisztogram

Itt találkozunk először olyan függvénnyel, amely grafikus ábrát szolgáltat. Rajzoltassuk fel az **x** adatmátrixunk első oszlopának gyakoriság-diagramját 19 egyenlőközű részintervallum (bin) segítségével (2.10. ábra)! Ehhez a **hist**() függvényt fogjuk használni, majd elmentjük az x adatmintánk első oszlopát a későbbi felhasználás céljára.

```
>> hist(x(:,1),19);

x(:,2)=[]; % töröljük a 2. oszlopot

save x_normal.dat x -ascii
```

Az adatfeldolgozó függvények többnyire vektorra (sor illetve oszlopvektor) is működnek. Eredményük értelemszerűen skalár, vagy vektor(ok). Például a **sort**() függvény sorvektorra alkalmazva két eredményvektort állít elő. Az első vektor a rendezett adatsort, a második vektor a rendezett elemek eredeti helyének sorszámát, azaz indexét adja:

```
>> p=[2  0  -1  5  7  3], [q ind]=sort(p), p(ind)

p =

     2     0    -1     5     7     3

q =

    -1     0     2     3     5     7

ind =

     3     2     1     6     4     5
```

```
ans =

    -1     0     2     3     5     7
```

Láthatjuk, hogy az eredeti adatok indexsorrendbeli listázása, a p(ind) hivatkozás is a rendezett adatsort adja.

2.3. A Matlab programozása

A Matlab programozása elsősorban M-fájlok segítségével történik. Az elérhető M-fájl nevének parancsként begépelésével az M-fájlbeli tevékenységsor végrehajtódik. Hasonlóképp, ha az M-fájl függvénydefiníciót tartalmaz, akkor az ott definiált és elérhető függvényre már hivatkozhatunk (lásd tridiag() függvény).

Az M-fájlbeli parancsok, kifejezések a begépelés sorrendjében hajtódnak végre. Ettől eltérni a szelekciós valamint iteratív szerkezetek és speciális vezérlésátadó utasítások segítségével lehet.

2.3.1. Szelekciós szerkezetek

Egyágú szelekció

Szerkezete: **if** ***feltétel*** tevékenység(ek) **end**

A szerkezet végén vagy a (;) pontosvessző, vagy a (,) vessző jelet kell elhelyezni.

A feltétel skalárértékű vagy tömbértékű kifejezés lehet. A feltétel akkor minősül hamisnak, ha van benne 0 érték. Ez a C-nyelv szabályaira emlékeztet. Minden egyéb esetben a feltétel igaznak minősül. Tevékenység alatt kifejezést, vagy utasítást értünk.

```
>> S=ones(3); if S S=true; end; S

S =  1

>> S=eye(3); if S S=true; end, S

S =

     1     0     0

     0     1     0

     0     0     1
```

Ha két azonos dimenziójú A és B mátrixot hasonlítunk össze, akkor az A == B hasonlítás többnyire nem megfelelő, mert a Matlab az összehasonlítást elemenként végzi el. Persze az is lehetséges, hogy pont ezt akartuk. Ha a két mátrix teljes egyezését akarjuk ellenőrizni, akkor a megoldás az isequal(A, B) használata, amely skalárokra is jó. Mátrixokra vonatkozó feltételek leírásához lásd még az **isequal**, **isempty**, **all** és az **any** függvényeket.

Kétágú szelekció

if ***feltétel***

tevékenység(ek)1;

else

tevékenység(ek)2;

end

```
>> t=rand(); if round(t) s=' nem kisebb 0.5-nél'; else s=' kisebb, mint 0.5'; end;
 strcat(mat2str(t,5), s)

ans =

0.38836 kisebb, mint 0.5

>> t=rand(); if round(t) s=' nem kisebb 0.5-nél'; else s=' kisebb, mint 0.5'; end;
 strcat(mat2str(t,5), s)

ans =

0.81769 nem kisebb 0.5-nél
```

A példában alkalmazott még nem használt függvények:

1. **mat2str**(érték, tizedesek): karakterlánccá alakítja a numerikus értéket
2. **round**(): szokásos kerekítés
3. **strcat**(karakterláncok): egyberagasztja, összefűzi a karakterláncokat

Többágú szelekció

Megvalósítható if szerkezettel, a következő módon:

if ***feltétel***

tevékenység(ek)1;

elseif ***feltétel2***

tevékenység(ek)2;

else

tevékenység(ek)3;

end

A **switch** többirányú elágazást tesz lehetővé. Az elágazást irányító kifejezés(változó) skalár vagy string típusú lehet. A **case** és az **otherwise** kulcsszavak mögé írjuk az egyes ágakat, és a switch szerkezetet az **end** kulcsszó zárja le.

A Matlab a C nyelvvel szemben nem igényli az egyes ágak lezárását a **break** utasítással, és a **case** kulcsszavak után nem csak egész értékeket írhatunk, hanem stringkonstansokat, sőt változóneveket is.

```
>> n= uint16(input('Kérek egy pozitív egészet (max. 65535): '));

switch (mod(n,4) == 0) + (mod(n,2) == 0)

    case 0

        disp(strcat(num2str(n), ' páratlan'));

    case 1
```

```
        disp(strcat(num2str(n),' nem osztható 4-gyel'));
    case 2
        disp(strcat(num2str(n), ' osztható 4-gyel'));
    otherwise
        disp('Nem lehetséges!');
end
```

```
Kérek egy pozitív egészet (max. 65535): 16

16 osztható 4-gyel
```

Megjegyzés:

A **disp**(X) függvény az X mátrix esetén a mátrix elemeit a mátrix neve nélkül jeleníti meg, ha pedig X egy sztring, akkor magát a sztringet jeleníti meg.

2.3.2. Iterációk

A Matlab kétféle iterációt használ:

1. érték felsorolásos, for iteráció
2. elől tesztelő, while iteráció.

A for iteráció

Szerkezete:

for változó = értéksorozat tevékenység(ek) **end**

A következő példánkban az m sorvektort folyamatosan bővítjük a ciklusváltozó értékeivel:

```
>> m=[]; for i=1:2:7 m=[m i]; end; m

m =

     1     3     5     7
```

A for iterációk egymásba is ágyazhatók. Állítsuk elő programmal a korábbi H4 Hilbert mátrixunkat:

```
>> H4P=[];
for i=1:4
  for j=1:4
    H4P(i,j)=1/(i+j-1);
  end
end
H4P
```

```
H4P =

    1.0000    0.5000    0.3333    0.2500

    0.5000    0.3333    0.2500    0.2000

    0.3333    0.2500    0.2000    0.1667

    0.2500    0.2000    0.1667    0.1429
```

A ciklusváltozó tetszőleges, sorvektorbeli értékeket, akár oszlopvektorokat is felvehet:

```
>> for i=[pi, inf, nan] i, end

i =   3.1416

i =   Inf

i =   NaN

>> A = rand(2,4),  for x=A   x, end;

A =

    0.1818    0.1455    0.8693    0.5499

    0.2638    0.1361    0.5797    0.1450
```

```
x =

    0.1818

    0.2638

x =

    0.1455

    0.1361

x =

    0.8693

    0.5797

x =

    0.5499

    0.1450
```

A while iteráció

Az **elől tesztelő iteráció** lépésszáma nem ismert, addig működik, amíg a feltétel igaznak bizonyul.
while ***feltétel*** tevékenység(ek) **end**
Példaként állítsuk elő a 30, 45, 60 fok érték szinuszát a szinusz függvény MacLaurin-sorával eps pontossággal!

```
>> format long;

fok=[30 45 60]; x=fok*pi/180; s=zeros(1, length(x)); t=x; n=3;

while max(abs(t))>eps s=s+t; t=-t.*x.^2/(n*(n-1)); n=n+2; end;

fok, s, szinusz=sin(x)

fok =

    30    45    60

s =

   0.500000000000000   0.707106781186547   0.866025403784438

szinusz =

   0.500000000000000   0.707106781186547   0.866025403784439
```

Az s részletösszegeket sorozatban előállító while iterációban azt használtuk ki, hogy a szinusz függvény MacLaurin sorának következő t tagja az előző t taghoz képest előjelváltó, x^2-tel szorzódik, és két egymást követő természetes szám szorzatával osztódik.

Az iterációs szerkezetekből a **break** utasítással lehet korábban kitörni:

```
>> for i=1:10 i, if i==3 break; end; end;

i =      1

i =      2

i =      3
```

Itt kell megemlíteni még a **continue** parancsot is, amelyet a for vagy while ciklusok belsejében helyezhetünk el. Hatására a ciklus belsejének continue utáni része nem hajtódik végre, hanem a következő iterációs lépés indul el.

2.3.3. Felhasználói függvények definiálása

Egyszerű (nem rekurzív) függvények

A függvények definícióit M-fájlokban helyezzük el a következő formában:

function visszatérési paraméter(ek) = **függvénynév**(bemenő paraméterek);

utasítások;

A visszatérési paraméterek/értékek listáját, ha az több elemet tartalmaz, akkor szögletes zárójelbe kell tenni. A lista elemei közé space-t kell írni. Célszerű, ha a függvény neve megegyezik annak az M-fájlnak a nevével, amelyben van. A formális paraméterek listájának elemeit (,) vesszővel választjuk el.

Tehát definiálunk egy **függvénynév.m** szöveges fájlt, amelyben leírjuk mit és hogyan kell számolni a bemenő értékekből kiindulva, és a visszatérő formális paraméterek is itt kapnak értéket.

Az M-fájlok kommentárokat is tartalmazhatnak. A kommentár sorok % jellel kezdődnek.

Példaként egy négyzetes mátrix maximális sajátértékét és az ehhez tartozó sajátvektort meghatározó függvény definíciója, azaz a **hatvany_iteracio.m** fájl tartalma a következő lehet:

```
% abszolút értelemben maximális sajátérték és a hozzá tartozó

% sajátvektor kiszámítása a Mises féle hatványiteráció módszerével

% lambda - maximális sajátérték, u -- az ehhez tartozó sajátvektor

% iteracio - a végrehajtott iterációs lépések száma

% lásd [6], 11.3.4. tétel

% domináns főátlójú szimmetrikus mátrixra kiválóan müködik

% ilyenek például a korrelációs mátrixok

function [u,lambda,iteracio]=hatvany_iteracio(A,max_iteracio,pontossag);

    n=max(size(A));

    u=(ones(n,1));

    u=u/norm(u);
```

```
    lambda=u'*A*u;

    hiba=1;

    iteracio=0;

    while hiba>pontossag && iteracio<max_iteracio

        iteracio=iteracio+1;

        u=A*u;

        u=u/norm(u);

        lambda_regi=lambda;

        lambda=u'*A*u;

        hiba=abs(lambda-lambda_regi);

    end
```

Tesztelésekor a formális kimenő paraméterek helyére konkrét változók, a bemenő paraméterek helyére pedig konkrét értékek, vagy értékkel bíró kifejezések írandók:

```
>> u=3; A=[5 2 1; 2 6 0; 1 0 4], [sv,se,it]=hatvany_iteracio(A,100,eps), u
```

```
A =

     5     2     1

     2     6     0

     1     0     4

sv =

    0.6312

    0.7563

    0.1720

se = 7.6691

it =    34

u =     3
```

Látható, hogy az u azonosítójú és az M-fájlban használt változó csak a függvényben élő lokális változó, mert a kívül definiált u változó értékét nem változtatta meg a definícióban lévő értékadás.

Ellenőrzésként meghívjuk a beépített **eig**() függvényt, amely a teljes sajátérték-feladatot oldja meg:

```
>> [U Lambda]=eig(A)
```

```
U =

  -0.679313061986337   0.374361954783071   0.631178968776483

   0.431981482758553  -0.491296263511568   0.756320024865991

   0.593233311917385   0.786435698751378   0.172026536792908

Lambda =

   2.854897308799578                   0                   0

                   0   4.476023602918134                   0

                   0                   0   7.669079088282288
```

Rekurzív függvények

Egy függvény definíciója rekurzív, azaz saját magát meghívó is lehet. A legegyszerűbb rekurzív definíció az $n!$ kiszámítása $n! = n*(n - 1)!$ módon. Természetesen a terminálódásról is gondoskodni kell, azaz $n = 0$ esetén érvényesíteni kell a $0! = 1$ definíciót.

Példaként nézzük meg egy pozitív egész szám prímtényezős felbontását elvégző primbont() függvény definícióját. A primbont() függvény a meghívandó bont() függvény segítségével 2-vel kezdi a felbontási kísérleteket.

```
function []=primbont(n);

   if isempty(n) || n~{}=uint32(n) fprintf('\nHibás felbontandó!\n'); return; end;

   fprintf('\n%lu = ', n); bont(n,2);
```

A bont() függvény nem ad vissza értéket, viszont mellékhatásként a soron következő prímtényezőt a képernyőre kiírja. A kiírást a C-nyelv szabályainak megfelelően felírt **fprintf**() függvény segítségével hajtjuk végre (lásd **help fprintf**)

```
% A p szám oszthatóságának vizsgálata q-val.

%  Ha p eleve nem osztható a túl nagy q-val (q*q > p esete),
% akkor p kiírása és kész.

% Itt a q*q > p vizsgálat helyett a túlcsordulást nem okozó q > p/q relációt
% ellenörizzük.

% Ha p nem osztható q-val, akkor a következö q+1 osztóval kísérletezünk.

% Ha p osztható q-val, akkor kiírjuk a q tényezöt, majd a p/q szám q-val való
% továbboszthatóságát nézzük meg.

function []=bont(p, q);

  if isempty(p) || p<2 fprintf('\nHibás felbontandó!\n'); return; end;

  if q>p/q fprintf('%lu\n',p); return; end;
```

```
    if mod(p,q) bont(p, q+1);

    else fprintf('%lu * ',q); bont(p/q, q);

    end;
```

A primbont és bont függvények tesztelése 4 bájton tárolt nem negatív egész adattípuson (uint32):

```
 >> n = uint32(input('Kérek egy pozitív egészet (max. 4294967295): '));

primbont(n);
```

Futtatási eredmények:

```
 Kérek egy pozitív egészet (max. 4294967295): 123458200

 123458200 = 2 * 2 * 2 * 5 * 5 * 19 * 53 * 613

 >> primbont(8888888889); % 10 számjegy, nem lehet uint32 típusú

 Hibás felbontandó!

 >> primbont(888888888); % ez elférne az uint32 típusban

 888888888 = 2 * 2 * 2 * 3 * 3 * 37 * ??? Maximum recursion limit of 500 reached.
 Use set(0,'RecursionLimit',N)to change the limit.

Be aware that exceeding your available stack space can crash MATLAB and/or your
computer.
```

```
Error in ==> bont
```

Ha a rekurzió mélysége túl nagy, vagy megtelik a rendelkezésre álló veremtár, hibaüzenet generálódik. Most tudunk segíteni. Megnöveljük a beállított rekurziószám korlátot:

```
>> set(0,'RecursionLimit',1000)

>> primbont(888888888);

888888888 = 2 * 2 * 2 * 3 * 3 * 37 * 333667
```

Célszerű a rekurziószám korlátjának növelését beépíteni a primbont.m fájl 3. sora elé a következőképp:

```
if get(0,'RecursionLimit')$<$1000 set(0,'RecursionLimit',1000) end
```

Megjegyezzük, hogy a prímfelbontást a Matlab a **factor**() függvény segítségével el tudja végezni, és a prímtényezőket egy sorvektorban helyezi el. A factor() függvény csak 2^{52}-nél kisebb számok esetén működik numerikus értékekre, tehát a fenti 888888888 értékre már nem működik. Túl nagy (2^{52}-nél nagyobb) számok prímfelbontási feladatára a Matlab a szimbolikus számítási moduljának használatát javasolja. A szimbolikus számításokkal később foglalkozni fogunk.

```
>> factor(sym('12345678901234567890'))

ans =

   2*3^2*5*101*3541*3607*3803*27961
```

Rekurzív függvényre egy másik jellemző példa a négyzetes mátrix determinánsának kiszámítása a kifejtési tétel alkalmazásával. A kifejtést az első sor szerint végezzük. Az **mdeterm.m** fájl tartalma:

```
% Az mdeterm függvény egy négyzetes mátrix determinánsát számolja az 1. sor
% szerinti kifejtéssel

% rekurzív definíció

function f = mdeterm(A);

if max(size(A))==1 f=A; return; end;    % ha már csak 1 elemü a mátrix

f=0;     % a gyüjtö nullázása

sgn=1;

for k=1:max(size(A))

  if A(1,k)  % ha a sor aktuális eleme 0, akkor nincs új tag

    B=A;     % segédmátrix, az A másolata

    B(:,k)=[] ; % töröljük belöle a k-ik oszlopot

    B(1,:)=[];       % töröljük belöle az elsö sort

    f=f+sgn*A(1,k)*mdeterm(B); % kifejtési tétel szerinti gyüjtés

  end;

  sgn= -sgn;     % sakktábla szabály
```

```
end;
```

Ellenőrzés:

```
>> load A.dat, A, det_A=det(A), mdeterm_A=mdeterm(A)

A =

     4     5     6

     5     6     8

     1     7     7

det_A =

 -16.999999999999996

mdeterm_A = -17
```

Jegyezzük meg, hogy a fenti és a kifejtési tételen alapuló determinánsszámításnak a műveleti igénye a méretével rohamosan növekszik, $n!$ nagyságrendű. Éppen ezért csak kicsi méretű determináns kiszámítására alkalmas. Miért lett a det_A érték nem pontos? A válasz erre a kérdésre az, hogy a Matlab a determinánst a Gauss eliminációval végrehajtott LU felbontással hajtja végre, amelyben már osztási műveletek is szerepelnek, ezért pici kerekítési hibák is terhelik:

```
[L,U] = lu(A)

s =  det(L)          % Ez mindig +1 vagy -1

det(A) = s*prod(diag(U))
```

A kapott L mátrix egy permutált alsó háromszög-mátrix, az U mátrix pedig valódi felső háromszög-mátrix.

Egy mátrix szingularitásának ellenőrzésére a determináns használata helyett a kondíciószám (lásd később) segítségével történő vizsgálatot javasolja a Matlab.

Rekurzív függvény iterációvá alakítása

Ha egy rekurzív függvény definíciójában csak egyetlen paraméter van, és a rekurzív hívások külön ágakon szerepelnek, akkor nagyon egyszerű while-os iteratív szerkezetté alakítani. A terminálódó rész return-je helyett a break utasítást kell használni, a rekurzív hívás helyett pedig a continue utasítást. A pbont() függvény rekurzív hívások nélkül végzi el a prímfelbontást.

```
function []=pbont(n);

  if isempty(n) \textbar \textbar  n\~{}=uint32(n)
  fprintf('\nHibás felbontandó!\n'); return; end;

  fprintf('\n%lu = ', n);

  p=n; q=2;

  while 1
```

```
    if q$>$p/q fprintf('%lu\n',p); break; end;

    if mod(p,q) q=q+1; continue; end;

    fprintf('%lu * ',q); p=p/q;

  end;
```

Tesztelés:

```
>> pbont(4111111111)

4111111111 = 19 * 647 * 334427
```

Hasonlítsuk össze a pbont() és a factor() függvények futási idejét!

```
>> tic; for n=100:20000 pbont(n); end; toc

Elapsed time is 9.739860 seconds.

>>tic; for n=100:20000 factor(n), end; toc

Elapsed time is 5.440056 seconds.
```

Látható hogy a Matlab beépített függvénye gyorsabban dolgozik, tehát csak a legritkább esetben kell egy beépített függvény helyett saját függvényt használni.

Globális változók

Egy Matlab függvény alapértelmezés szerint lokális változókat használ. Ez azt jelenti, hogy ha a függvény egy változójával azonos nevű változó van a hívás szintjén, akkor a hívás szintű változó a függvényben nem érhető el, és a függvényen belüli értékadás a külső változó értékét nem változtatja meg.

Viszont adódhat olyan feladat, hogy a hívás szintű változót a függvényben is szeretnénk használni. Ennek megoldására szolgál a globális változó fogalma. Egy m változó akkor lesz globális, ha a hívás szintjén is és a hívott függvényben is globálisnak nyilvánítjuk a

```
>> global m
```

paranccsal. Illusztrálásként az mdeterm hívásunk rekurzív lépéseinek számát jelezzük ki! Az mdeterm függvény így módosul:

```
% Az mdeterm függvény egy négyzetes mátrix determinánsát
% számolja az 1. sor szerinti kifejtéssel

% rekurzív definició

function f = mdeterm(A);

  global m; % globális változó a hívások számlálásához (kommentározható
  ha nem kell)

  if max(size(A))==1 f=A; return; end;    % ha már csak 1 elemü a mátrix

  f=0;  % a gyüjtö nullázása

sgn=1;

for k=1:max(size(A))
```

```
  if A(1,k)  % ha a sor aktuális eleme 0, akkor nincs új tag
    B=A;     % segédmátrix, az A másolata
    B(:,k)=[] ; % töröljük belöle a k-adik oszlopot
    B(1,:)=[];       % töröljük belöle az elsö sort
    m=m+1; % rekurzív hívás jön (ez is kommentározható)
    f=f+sgn*A(1,k)*mdeterm(B); % kifejtési tétel szerinti gyüjtés
  end;
  sgn= -sgn;     % sakktábla szabály
end;
```

A híváskor pedig így járunk el:

```
>> global m; m=0; A=rand(6,6), determinans=mdeterm(A), lepesszam=m
A =
    0.3500    0.8308    0.7537    0.7792    0.3371    0.6020
    0.1966    0.5853    0.3804    0.9340    0.1622    0.2630
```

```
    0.2511    0.5497    0.5678    0.1299    0.7943    0.6541

    0.6160    0.9172    0.0759    0.5688    0.3112    0.6892

    0.4733    0.2858    0.0540    0.4694    0.5285    0.7482

    0.3517    0.7572    0.5308    0.0119    0.1656    0.4505

determinans =    0.0033

lepesszam =        1236
```

Ha a hívó szinten az m változó nem globális, akkor a függvénybeli módosítások nem jutnak el a hívó szintre.

```
>> clear m; m=0, mdeterm(A), m

m =     0

ans =   0.0033

m =     0
```

2.3.4. Adatbekérés, adatkiírás

A switch szerkezet kapcsán mintát láthattunk az adatbekérésre és eredménykiírásra.

Az **fprintf**() függvény szöveges adatfolyamot állít elő, ami ha nem fájlba irányul, akkor a Command Window-ba kerül. Ehhez hasonló az **sprintf**(), ami szintén szöveget állít elő, de az az ans változóba kerül. Az fprintf és sprintf függvények paraméterezése az Ansi C szabványnak megfelelő, ezért a C-programozásban jártasak számára igen könnyű.

A **disp**() parancs egy változó értékét írja ki a változó neve nélkül.

A változó = **input**() parancs az argumentumként megadott üzenet kiadása után a felhasználói begépelést eltárolja a megjelölt változóban.

A grafikai részben példát fogunk látni arra, hogyan helyezzünk el szöveget az ábrán a **text**() függvénnyel.

Az input-output további műveleteinek részletesebb ismertetése már nem része egy alapozó Matlab-kurzusnak.

Önellenőrzés

1. Állítsuk elő azt a 13 elemű sorozatot a 0..360 tartományban, amelynek növekménye 30! A sorozatképzést a sorozatképzés operátorával is és a linspace() függvénnyel is adjuk meg!
   A megoldás megtekintéséhez kattintson ide!
2. Ha az előző sorozat értékeit fokoknak tekintjük, akkor állítsuk elő a radiánba átszámított értéksorozatot is!
   A megoldás megtekintéséhez kattintson ide!
3. Állítsuk elő mindhárom sorozatképző módon a következő oszlopvektort!

$$\begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

   A megoldás megtekintéséhez kattintson ide!
4. Legyen D = rand(5)
   Mit eredményez a triu(D, 1) + diag(diag(D)) + tril(D, -1) kifejezés?
   A megoldás megtekintéséhez kattintson ide!
5. Milyen sávmátrixot eredményeznek a következő kifejezések?
   a./ diag(diag(D, -1), -1) + diag(diag(D, 1), 1) + diag(diag(D))
   b./ diag(diag(D, -1), 1) + diag(diag(D, 1), -1) + diag(diag(D))
   A megoldás megtekintéséhez kattintson ide!
6. Mit eredményez az eye(size(D)).*D kifejezés?
   A megoldás megtekintéséhez kattintson ide!

7. Állítsuk elő diadikus szorzatként (egy oszlopvektor és egy sorvektor mátrixszorzataként) a

```
10    20    30    40

20    40    60    80

30    60    90   120

40    80   120   160
```

mátrixot!
A megoldás megtekintéséhez kattintson ide!

8. Állítsuk elő egyetlen for iterációval a hilb(5) 5x5 méretű Hilbert mátrixot!
A megoldás megtekintéséhez kattintson ide!

9. Legyen b = rand(1, 5)
Mivel egyenlő a sum(b' == b(:)) kifejezés, és miért?
A megoldás megtekintéséhez kattintson ide!

10. Legyen A=rand(5) - 0.5
A max(sum(abs(A))) kifejezés a mátrixelemek abszolút-értékeinek oszlopösszegeit képzi, és ennek maximumát adja. Ezt szolgáltatja a norm(A, 1) kifejezés is.
Adjuk meg azt a kifejezést, amelyik a mátrixelemek abszolút-értékeinek sorösszegeit képzi, és ennek maximumát adja. A kapott eredményt hasonlítsuk össze a norm(A, Inf) értékével!
A megoldás megtekintéséhez kattintson ide!

9. LECKE

Lineáris algebrai feladatok, a grafika alapjai

2.4. Mátrixértékű függvények és műveletek használata lineáris algebrai alapfeladatokra

2.4.1. Speciális szorzatok

A Matlabba – elvárásainknak megfelelően – beépítettek megfelelő függvényeket a speciális szorzatok meghatározására.

dot(x,y) - euklideszi skalárszorzat (szorzatösszeg), ahol x és y azonos méretű vektorok.

cross(a,b) – vektorszorzat, ahol a és b 3-elemű oszlopvektorok.

Például ha x, y és z 3-elemű vektorok, akkor a dot(cross(x,y),z) kifejezés a három vektor vegyesszorzatát jelenti, és a három vektor által kifeszített paralelepipedon térfogatát adja. Ha ez 0-nak adódik, akkor a 3 vektor lineárisan nem független, azaz egy síkban fekszenek.

subspace(x,y) = acos(dot(x,y)/(norm(x)*norm(y))) a két vektor szögét adja radiánban, ahol a norm(x) = sqrt(dot(x,x)) kifejezés az euklideszi norma, azaz az x vektor általánosított hossza. A subspace(A, B) függvény az A és B mátrixok oszlopvektorai által meghatározott alterek szögét számolja ki. Ennek akkor van értelme, ha az A, B mátrixok oszlopainak száma kisebb, mint a sorok száma, azaz valódi alterekről van szó.

```
A = [1 2; 2 -1; 4 3], B=A+1, x=A(:,1); y=A(:,2); z=B(:,1); vektorok=[x y z]

A =

     1     2

     2    -1

     4     3
```

```
B =

     2     3

     3     0

     5     4

vektorok =

     1     2     2

     2    -1     3

     4     3     5

>> terfogat = det(dot(cross(x,y),z))

terfogat =    10

>> x_y_szoge = [subspace(x,y), acos(dot(x,y)/(norm(x)*norm(y)))]

x_y_szoge =

    0.7956    0.7956
```

```
>> A_B_szoge = subspace(A,B)

A_B_szoge =    0.1643
```

2.4.2. Egyenletrendszerek megoldása

Már korábban láttuk, hogy egy **Ax** = **b** lineáris egyenletrendszer **A** együtthatómátrixának és az inhomogén jobboldali **b** oszlopvektor ismeretében az **x** megoldásvektor egyszerűen számítható. A megoldhatóság feltétele az, hogy az **A** mátrix ne legyen szinguláris, azaz a determinánsa ne legyen 0.

```
>> load A.dat, load b.dat,

determ=det(A)

if det(A) x=A\b; else x=[]; end; megoldas=x

A(3,2:3)=[1 2]

% az A mátrix harmadik sora a második és elsö sor különbsége!

determ=det(A)

if det(A) x=A\b; else x=[]; end; megoldas=x

determ =

  -17.0000
```

```
megoldas =

    4.0000

   -1.0000

    2.0000

A =

     4     5     6

     5     6     8

     1     1     2

determ =

  2.2204e-015

Warning: Matrix is close to singular or badly scaled.

         Results may be inaccurate. RCOND = 1.982541e-017.
```

```
megoldas =

  1.0e+015 *

    7.2058

   -3.6029

   -1.8014
```

Itt találkozunk először egy mátrix kondíciószámának fogalmával. Egy mátrix kondíciószámát a **cond**() függvény adja. Egy mátrix akkor gyengén kondicionált, ha egy elemének igen pici megváltoztatásának hatására a vele számított eredmények (pl. egyenletrendszer megoldása) óriási mértékben megváltoznak. Az üzenetben látott RCOND érték a kondíciószám reciproka.

Az egyenletrendszer megoldhatóságának ellenőrzése a determináns kiszámításával nem mindig szerencsés, mivel a számítási pontatlanságok miatt az elvileg 0 determináns sokszor nem annak mutatkozik. Ehelyett célszerűbb az együtthatómátrix rangjának (lineárisan független sorvektorok, illetve oszlopvektorok száma) kiszámítása:

```
>> load A.dat, load b.dat,

rang=rank(A)

if rank(A) == max(size(A)) x=A\b;  else x=[]; end; megoldas=x

A(3,2:3)=[1 2]

% a harmadik sor a második és elsö sor különbsége!
```

```
rang=rank(A)

if rank(A) == max(size(A)) x=A\b;  else x=[]; end; megoldas=x

rang =     3

megoldas =

    4.0000

   -1.0000

    2.0000

A =

     4     5     6

     5     6     8

     1     1     2

rang =     2

megoldas =     []
```

2.4.3. Szinguláris, ellentmondó és túlhatározott rendszerek, pszeodoinverz

Az előző példánkban az **A** mátrix rangja már csak 2, és a **b**-vel kibővített mátrix rangja 3, ezért így ez az egyenletrendszerünk ellentmondó.

```
>> rank(A), rank([A b])

ans =      2

ans =      3
```

Ha a b(3) értékét 7-re változtatjuk, akkor az egyenletrendszerünknek csak két független egyenlete lesz, és ekkor végtelen sok megoldása lehet.

```
>> b(3)=7, rank(A), rank([A b])

b =

    23

    30

     7

ans =      2

ans =      2
```

Mindkét esetben az egyenletrendszernek valamilyen értelemben optimális megoldását a **pinv**() Moore-Penrose-féle pszeudoinverz segítségével lehet felírni.

Tekintsük az első, ellentmondó esetet!

```
>> load A.dat, load b.dat,

A(3,2:3) = [1 2]

% az A mátrix harmadik sora a második és az elsö sor különbsége!

rang = rank(A), rang_bov = rank([A b])

pinv_A = pinv(A)

x = pinv(A)*b

b, Ax = A*x, elteres = A*x - b


A =

     4     5     6

     5     6     8

     1     1     2

rang =     2

rang_bov =     3
```

```
pinv_A =

    0.2063   -0.0635   -0.2698

    0.7302   -0.3016   -1.0317

   -0.6349    0.3492    0.9841

x =

   -0.1270

   -3.6032

    6.6984

b =

    23

    30

    11
```

```
Ax =

   21.6667

   31.3333

    9.6667

elteres =

   -1.3333

    1.3333

   -1.3333
```

A most kapott közelítő megoldás optimális abban az értelemben, hogy normában minimalizálja az **Ax** - **b** eltérést. A túlhatározott egyenletrendszer esetében is pontosan ezt az utat kell követni. (A mátrix illetve vektornormákat a következő alfejezetben tárgyaljuk.)

Szinguláris esetben kevesebb független egyenletünk van, mint ismeretlen. Ekkor is a pszeudoinverz segítségével adunk optimális megoldást. Az **x** optimális megoldás normája (hossza) minimális a lehetséges megoldások halmazában.

```
>> load A.dat, load b.dat,

A(3,2:3) = [1 2], b(3) = 7

% a harmadik egyenlet a második és az elsö egyenlet különbsége lett!
```

```
rang = rank(A), rang_bov = rank([A b])

pinv_A = pinv(A)

x = pinv(A)*b

Ax = A*x

format long e, elteres = A*x - b

 A =

     4     5     6

     5     6     8

     1     1     2

b =

    23

    30

     7

rang =     2
```

```
rang_bov =     2

pinv_A =

    0.2063   -0.0635   -0.2698

    0.7302   -0.3016   -1.0317

   -0.6349    0.3492    0.9841

x =

    0.9524

    0.5238

    2.7619

Ax =

   23.0000

   30.0000

    7.0000
```

```
elteres =

    3.552713678800501e-015

    1.065814103640150e-014

    3.552713678800501e-015
```

Ha egy **At** mátrixnak több sora van, mint oszlopa, és rangja az oszlopok száma, akkor a pinv() pszeudoinverz az (At'*At)\At' módon számolódik:

```
>> load At.dat, At, rang=rank(At), pinv_At=pinv(At), (At'*At)\At'

At =

     1     5    13

     2     3    12

    -4     7     3

     3     1     4

    -2     4     1

rang =     3
```

```
pinv_At =

   -0.0618   -0.0690   -0.0331    0.4118    0.0841

   -0.0434   -0.0807    0.0932    0.2802    0.1315

    0.0614    0.0735   -0.0235   -0.1382   -0.0562

ans =

   -0.0618   -0.0690   -0.0331    0.4118    0.0841

   -0.0434   -0.0807    0.0932    0.2802    0.1315

    0.0614    0.0735   -0.0235   -0.1382   -0.0562
```

A pszeudoinverz ekkor At\eye(sorok száma) módon is számolható!

```
>> At\eye(5)

ans =

   -0.0618   -0.0690   -0.0331    0.4118    0.0841

   -0.0434   -0.0807    0.0932    0.2802    0.1315

    0.0614    0.0735   -0.0235   -0.1382   -0.0562
```

Ha egy **As** mátrixnak kevesebb sora van, mint oszlopa, és rangja a sorok száma, akkor a pinv() pszeudoinverz az As'/(As*As') módon számolódik:

```
>> load As.dat, As, rang=rank(As), pinv_As=pinv(As), As'/(As*As')

As =

     1     5    13     1     4

     2     3    12    -2    -3

    -4     7     3    -5     1

     3     1     4     0     2

rang =     4

pinv_As =

   -0.1169    0.0248   -0.0024    0.3234

   -0.0035   -0.0214    0.0796    0.0539

    0.0581    0.0469   -0.0301   -0.0691

    0.1356   -0.0458   -0.0999   -0.1995

    0.0608   -0.1202    0.0241    0.1261
```

```
ans =

   -0.1169    0.0248   -0.0024    0.3234

   -0.0035   -0.0214    0.0796    0.0539

    0.0581    0.0469   -0.0301   -0.0691

    0.1356   -0.0458   -0.0999   -0.1995

    0.0608   -0.1202    0.0241    0.1261
```

A pszeudoinverz ekkor eye(oszlopok száma)/As módon is számítható!

```
>> eye(5)/As

ans =

   -0.1169    0.0248   -0.0024    0.3234

   -0.0035   -0.0214    0.0796    0.0539

    0.0581    0.0469   -0.0301   -0.0691

    0.1356   -0.0458   -0.0999   -0.1995

    0.0608   -0.1202    0.0241    0.1261
```

Összefoglalva, akármelyik esetről is van szó, az $\mathbf{Ax} = \mathbf{b}$ egyenletrendszer „megoldása”

```
x = pinv(A)*b
```

módon mindig előállítható, legfeljebb időigényesebb lesz a megoldás.

2.4.4. Vektornormák, mátrixnormák

Egy vektor hosszának több dimenzióban való általánosítása a vektor normája. A vektor normájának tulajdonságai:

1. Nem negatív, és csak akkor 0, ha a vektor maga a 0-vektor.
2. A vektor megnyújtottjának normája a vektor normája szorozva a nyújtási tényező abszolút értékével.
3. Érvényes a háromszög-egyenlőtlenség.

Ezt a definíciót a szám n-esek vektorterében többféle módon lehet teljesíteni. Éppen ezért a matematikában többféle normát definiáltak. Ezek konvergens vektorsorozatokra ekvivalensek, ami azt jelenti, hogy ha egy x_k vektorsorozat eltérése egy x_0 vektortól valamelyik normában mérve 0-hoz konvergál, akkor a többi normában mérve is.

A Matlab segítségével a következő normákat használhatjuk:

```
>> help norm

 NORM   Matrix or vector norm.

    For matrices...

      NORM(X) is the 2-norm of X.
```

```
    NORM(X,2) is the same as NORM(X).

    NORM(X,1) is the 1-norm of X.

    NORM(X,inf) is the infinity norm of X.

    NORM(X,'fro') is the Frobenius norm of X.

    NORM(X,P) is available for matrix X only if P is 1, 2, inf or 'fro'.

  For vectors...

    NORM(V,P) = sum(abs(V).^P)^(1/P).

    NORM(V) = norm(V,2).

    NORM(V,inf) = max(abs(V)).

    NORM(V,-inf) = min(abs(V)).
```

A norm(X) 2-es norma az X'*X mátrix abszolútértékben legnagyobb sajátértékéből vont gyök (a sajátérték fogalmát lásd a következő pontban):

```
>> A, norm(A), sqrt(max(abs(eig(A'*A))))

A =

     1     3     6

     2     0    -1

     6     1     2

ans =    8.0548

ans =    8.0548
```

A normák jobb megismeréséhez természetesen ajánljuk az egyetemeken folyó lineáris algebra kurzusok lelkiismeretes látogatását, illetve a megfelelő jegyzetek (például: [8]) tanulmányozását.

2.4.5. Kondíciószám

Egy **X** négyzetes és invertálható mátrix kondíciószáma azt mutatja meg, hogy egy mátrixalgebrai feladat megoldása mennyire érzékeny az **X** mátrix pici megváltoztatására.

$$cond(X,p) = \|X\|_p \left\|X^{-1}\right\|_p \quad ahol\ p = 1,\ 2,\ \inf,\ 'fro'$$

Ez a definíció a 2-es normában a legnagyobb és legkisebb szinguláris érték hányadosa (X mátrix szinguláris értékeinek az X'*X mátrix, ami már szimmetrikus mátrix lesz, sajátértékeit nevezzük, lásd **help svd**)

```
>> cond(A), sqrt(max(abs(eig(A'*A))))/sqrt(min(abs(eig(A'*A)))),
max(svd(A))/min(svd(A))
```

```
ans =   19.8143

ans =   19.8143

ans =   19.8143
```

Matlab megfogalmazással a többi normában a kondíciószám norm(X,p)*norm(inv(X,p)) módon számítódik ki. A nagy kondíciószám közel szinguláris mátrixot jelent. A Matlab-ban az **rcond**(X) reciprok kondíciószámot is lehet használni, ami az 1-es normával számolt kondíciószám reciproka. Jól kondicionált mátrixokra az rcond értéke 1-hez közeli, és a rosszul kondicionált mátrix esetén az rcond értéke eps nagyságrendű.

```
>> A=[4 5 6; 5 6 8; 1 7 7], norm(A)*norm(inv(A)), cond(A),

1/(norm(A,1)*norm(inv(A),1)), rcond(A)

A =

     4     5     6

     5     6     8

     1     7     7

ans =  53.289730170756549

ans =  53.289730170756457

ans =   0.011564625850340
```

```
ans =    0.011564625850340
```

A Hilbert mátrixok gyengén kondicionált mátrixok.

```
>> cond(hilb(10)), rcond(hilb(10))

ans =    1.602466539329798e+013

ans =    2.828556315915063e-014
```

2.4.6. Sajátérték, sajátvektor

A műszaki matematika, statisztika, stb. nagyon sok problémájának megoldása az ún. sajátérték-feladat, vagy az általánosított sajátérték-feladat megoldására támaszkodik. Ez a feladat adódik például egy híd terheléspróbájából megnyúlások alapján a híd rezonancia-frekvenciáinak kiszámítására. A sajátérték-feladat legegyszerűbb megfogalmazása:

$$Au = \lambda u$$

Itt azt kell tudni, hogy ha az A lineáris transzformációt az **A** mátrix reprezentálja, azaz A(**x**) = **A***x, akkor melyek azok az irányok (**u** sajátvektorok), amelyeket az A transzformáció csak megnyújt? Hasonlóképpen milyen nyújtási tényezők (sajátértékek) adódnak? Ilyen feladat lehet például egy merev test esetében annak a forgástengelynek (szabad tengely) megállapítása, amely körüli forgásnál eltűnnek a deviációs momentumok.

Az **A** négyzetes mátrix sajátértékeit a karakterisztikus polinomjának zérus helyei adják. A karakterisztikus polinomot az **I** - λ**A** mátrix determinánsának kifejtésével tudjuk felírni. Ezt persze nem kell megtennünk, mert a **poly**(A) függvényhívás egy sorvektort eredményez, amelynek elemei a karakterisztikus polinom együtthatói a hatványkitevő csökkenő sorrendjében. Ennek gyökeit a **roots**() függvényhívás eredménye szolgáltatja, a sajátértékek oszlopvektorára alkalmazott poly(lambda) függvényhívás előállítja ismét a polinomot.

```
>> A=[4 5 6; 5 6 8; 1 7 7], a = poly(A), lambda=roots(a), poly(lambda)

A =

     4     5     6

     5     6     8

     1     7     7

a =

    1.0000  -17.0000    7.0000   17.0000

lambda =

   16.5138

    1.2864

   -0.8002

ans =

    1.0000  -17.0000    7.0000   17.0000
```

A sajátérték feladatot az [U Lambda]= eig(A) függvényhívással oldjuk meg.

```
>> [U Lambda]= eig(A)
```

```
U =

   -0.5226   -0.6218   -0.0704

   -0.6596   -0.4400   -0.7379

   -0.5402    0.6479    0.6712

Lambda =

   16.5138         0         0

         0    1.2864         0

         0         0   -0.8002
```

Az U mátrix oszlopvektorai az 1-re normált sajátvektorokat adják, a Lambda mátrix főátlóbeli elemei pedig a sajátértékeket.

```
>> A*U, U*Lambda
```

```
ans =

   -8.6297   -0.8000    0.0563

  -10.8923   -0.5660    0.5905

   -8.9214    0.8335   -0.5371

ans =

   -8.6297   -0.8000    0.0563

  -10.8923   -0.5660    0.5905

   -8.9214    0.8335   -0.5371
```

Nem szinguláris szimmetrikus mátrix esetében ezeknek a mátrixoknak segítségével előállíthatjuk az eredeti mátrixot (mátrixfelbontás):

```
>> A=[6 1 2; 1 5 3; 2 3 7], [U Lambda]= eig(A), U*Lambda*U'

A =

     6     1     2

     1     5     3

     2     3     7
```

```
U =

    0.1444    0.8771    0.4580

    0.7725   -0.3892    0.5018

   -0.6184   -0.2813    0.7338

Lambda =

    2.7854         0         0

         0    4.9148         0

         0         0   10.2998

ans =

    6.0000    1.0000    2.0000

    1.0000    5.0000    3.0000

    2.0000    3.0000    7.0000
```

2.5. A grafika alapjai

A Matlab grafikai képessége igen fejlett. Az egyszerűbb kétdimenziós ábrák készítésén és hangolásán kívül igen sok szolgáltatással rendelkezik. Az elkészült ábrák különféle formátumú és felbontású fájlokba menthetők, sőt akár animáció készítésére is lehetőséget nyújtanak.

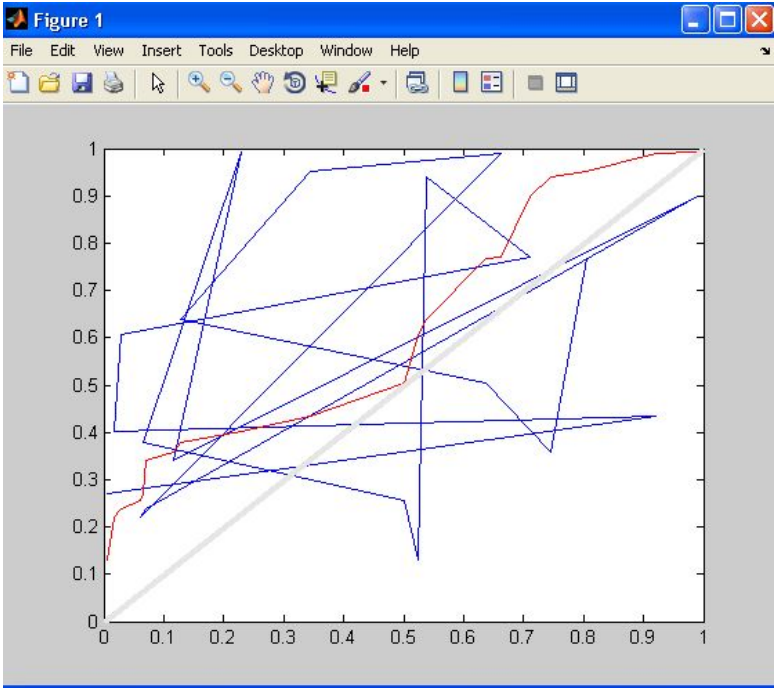
A 2.8. táblázat névsorba szedve egy kis ízelítőt mutat be a grafikai rendszer függvényeiből és parancsaiból.

2.8. táblázat. A Matlab grafikai függvényei és parancsai

axes	tengelyek kirajzolása	mesh	3D metszet felületek kirajzolása
axis	tengelyek skálázása és kirajzolása	newplot	új tengelyek és grafikus képernyő
bar	oszlopgrafikon	plot	töröttvonalas rajzolás
caxis	tengelyek skálázása álszínnel	plot3	3D rajzolás
cla	tengelyek törlése	polar	rajzolás poláris koordinátákban
clf	rajz törlése	print -djpeg90	aktív ábra jpeg fájlba mentése
close	rajz bezárás	refresh	grafikus kép frissítése
colormap	színtérkép	semilogx	x-tengelyen logaritmikus rajzolás
contour	2D kontúr rajzolás	semilogy	y-tengelyen logaritmikus rajzolás
contour3	3D kontúr rajzolás	stairs	lépcsőzetes ábrázolás
contourc	kontúr rajz számolás	surf	3D árnyékolt területek kirajzolása
figure	grafikus ablak létrehozása	text	grafikon szövegezés
fplot	függvény kirajzolása	title	grafikon címe
grid	rácsozat	xlabel	az x-tengely jelölése
gtext	szöveg elhelyezése egérrel	ylabel	az y-tegely jelölése
hold on	ábrák egymásra rétegzése	zlabel	z-tengely jelölése
loglog	logaritmusos skálájú rajzolás	zoom	nagyítás / kicsinyítés

2.5.1. A plot utasítás

A **plot** paranccsal síkbeli pontokat az őket összekötő vonalakkal (szakaszokkal) együtt rajzolhatjuk fel. Külön vektorban adjuk meg az egyes pontok x- illetve y-koordinátáit. A plot parancs a megadott koordinátapárok alapján egy Figure ablakban beméretezi pontsorozatot, és a sorozat szomszédos pontjait egyenes szakaszokkal összeköti. Ha a pontsorozat x koordinátái nem rendezett növekvően, sőt egyáltalán nem rendezettek, akkor az ábránk kaotikus lehet. A plot parancs az ábraelemek stílusára vonatkozó információkat is tartalmazhat.



2.11. ábra: Vonalas plot (kód lent)

```
>> x=rand(1,20); y=rand(1,20);

plot(x,y); % cikk-cakk rajzolás automatikus színezéssel

hold on; % a további grafikák ugyanebbe az ablakba kerülnek

x=sort(x); y=sort(y);

plot(x, y, 'r') % növekvö törött vonal piros színnel

plot([0,1],[0,1],'LineWidth',3, 'Color',[0.9 0.9 0.9]);

% vastag átlós vonal és szürke, az RGB komponensek nem teljes intenzitásúak

hold off; % kikapcsoljuk a további ábrák idekerülését
```

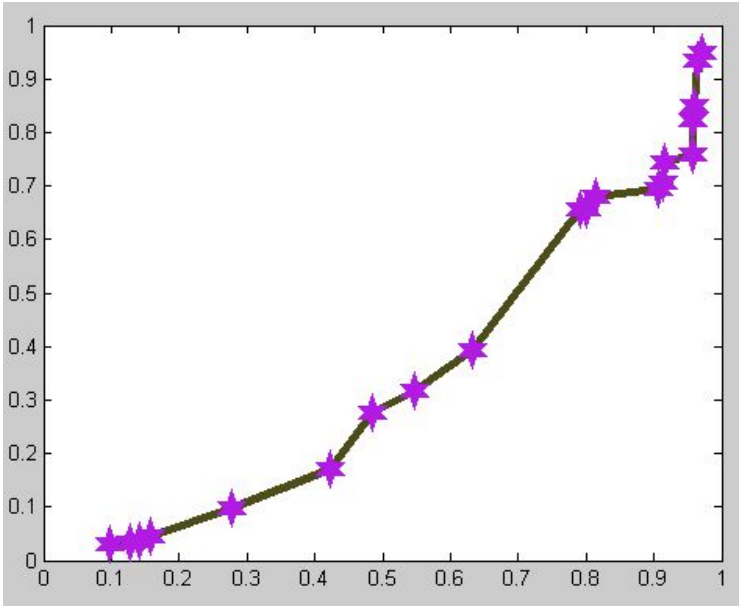
A plot parancs a koordináták után egy sztringet is tartalmazhat, amelynek karaktereivel a grafikus elemek megjelenését szabályozhatjuk a 2.9. táblázat szerint:

2.9. táblázat. Ábraelemek megjelenésére vonatkozó beállítások (plot parancs)

Pont	Vonal	Szín
. pont	- folytonos	y sárga
* csillag	- - szaggatott	r piros
x x betű	: pontozott	g zöld
o kör	-. folytonos és pontozott	w fehér
+ plusz jel		m magenta
s négyzet		c cián
d rombusz		b kék
<,>,v,^ háromszögek		k fekete
p ötszög		
h hexagon		

Az x, y, sztring adathármasok után a finomabb grafikus tulajdonság neve, értéke párokkal további szabályozást végezhetünk. Egy példa:

```
>>plot(x,y,'LineWidth',4,'Marker','h','MarkerSize',8,'Color',[0.3 0.3 0.1],
'MarkerEdgeColor',[0.7 0.1 0.9])
```

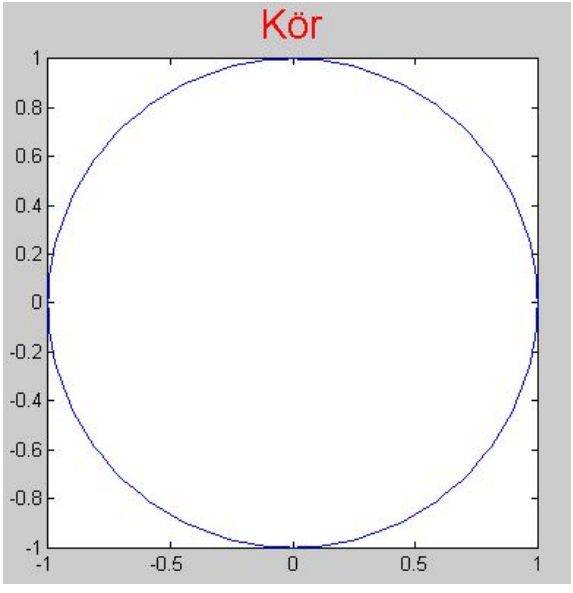


2.12. ábra: Töröttvonal jelölővel

A koordinátapárokat komplex számsorral is megadhatjuk:

```
>> z=exp(i*linspace(0,2*pi,181)); plot(z);
axis square; % az ábra méretezését négyzetesre alakítjuk
h=title('Kör', 'Color', 'R' ); % feliratot teszünk az aktív rajzra
% az objektum mutatót feljegyezzük
```

```
get(h); % megjelenítjük az objektum hangolható tulajdonságait

set(h, 'FontSize',20) % beállítjuk a betüméretet
```



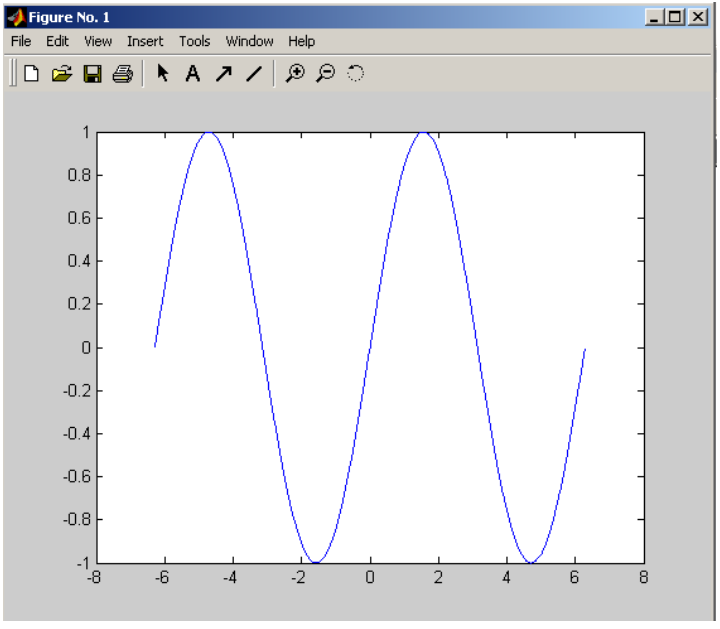
2.13. ábra: Kör

A hangolható tulajdonságok értékeit a teljes lista nélkül is lekérdezhetjük és megváltoztathatjuk:

```
get(h,'XGrid')
```

```
set(h,'XGrid','on','YGrid','on')
```

Rajzoljuk ki a sin(x) függvény grafikonjának pontjait a [-2π, 2π] intervallumban 1001 pont segítségével! (lásd linspace() függvény).

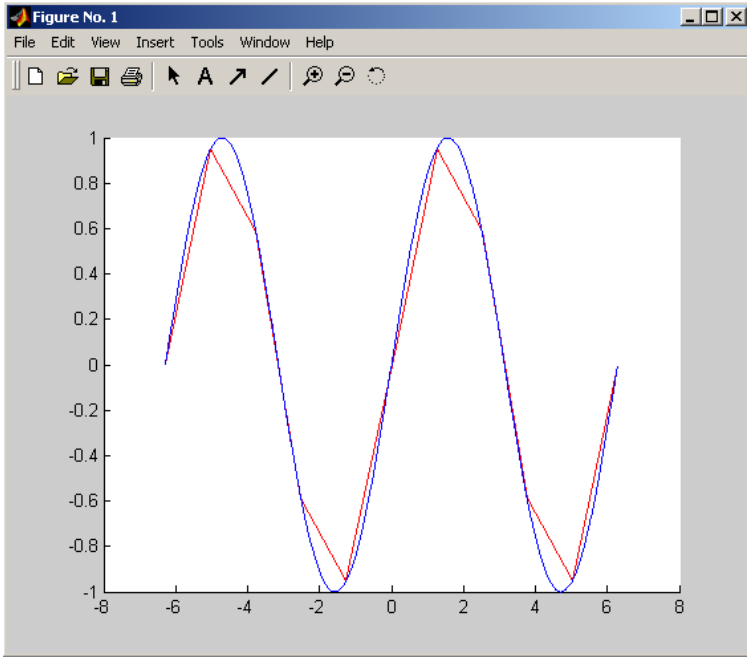


2.14. ábra: Vonalas ábra sűrű pontokból

```
>>x=linspace(-2*pi,2*pi,1001);  plot(x,sin(x))
```

Nézzük meg, hogy mi történik, ha az x sorozat csak 11 elemű! Ehhez ismételjük meg az előző grafikon kirajzolását is, de úgy, hogy a függvények közös ábrán szerepeljenek! A vonalak színe legyen különböző!

```
>>hold on;

x2=linspace(-2*pi,2*pi,1001);

plot(x2,sin(x2),'LineWidth',1,'Color',[0 0 1]);

x1=linspace(-2*pi,2*pi,11);

plot(x1,sin(x1),'LineWidth',1,'Color',[1 0 0]);

hold off
```



2.15. ábra: Vonalas ábra ritka és sűrű pontokból

Megjegyzés: A hold utasítás nélkül is lehet több grafikont egy ábrára tenni:

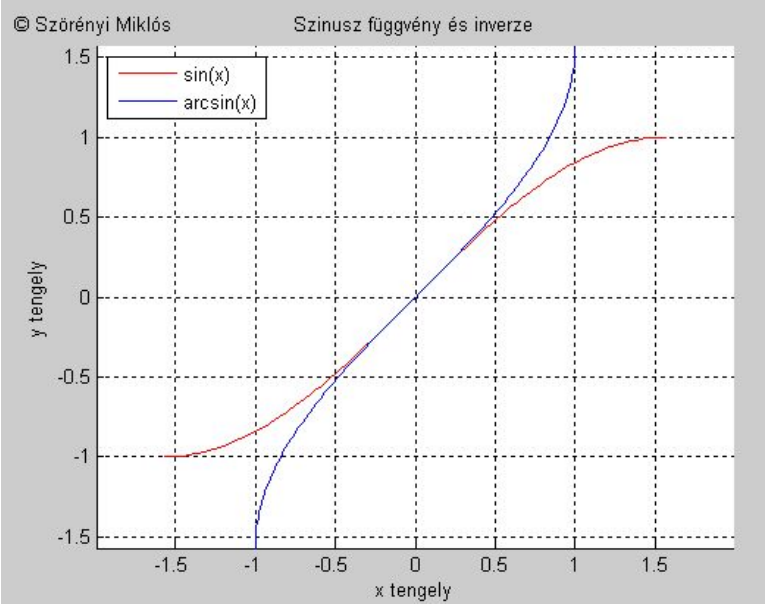
```
plot(x1,y1,string1,x2,y2,string2, ... )
```

ahol a string1 pl. 'r' lehet. Próbáljuk ki!

A vízszintes és függőleges tengelyekre az **xlabel** és az **ylabel** parancsokkal tudunk szöveget kiíratni. A **text** parancs segítségével pedig a rajz bármelyik, koordinátájával megadott pontjára feliratot, szöveget helyezhetünk. A **grid** paranccsal egy olyan rácsot illeszthetünk a koordináta-rendszerre, amely illeszkedik a tengelyek

beosztására. A **legend** paranccsal a plot parancsok után jelmagyarázatot tehetünk az ábrára. A jelmagyarázatok tartalmának és sorrendjének illeszkedni kell a plot-ok jelentéséhez és sorrendjéhez, egyébként keveredés lesz. A legend tartalmazhat sarokpozicionáló kódokat is. 1 = jobbra fent, 2 = balra fent, 3 = balra lent, 4 = jobbra lent a rajzterületen belül.

Ha az x és y koordinátákat felcseréljük, akkor az inverz függvény grafikonját tudjuk kirajzoltatni. Rajzoltassuk ki a sin(x) függvényt és inverzét az x: [-1, 1] intervallumban! Adjunk feliratot és jelmagyarázatot is!



2.16. ábra: Vonalas ábra feliratokkal (kód lent)

```
>> x=linspace(-pi/2,pi/2,101); y=sin(x);
```

```
hold on;

plot(x,y, 'r');

plot(y,x, 'b');

xlabel('x tengely')

ylabel('y tengely')

title('Szinusz függvény és inverze')

text(-2.5,1.7, 'copyright  Szörényi Miklós')

legend('sin(x)', 'arcsin(x)', 2);

grid on

axis equal

hold off;
```

Természetesen közvetlenül az **asin**() függvényt is használhattuk volna.

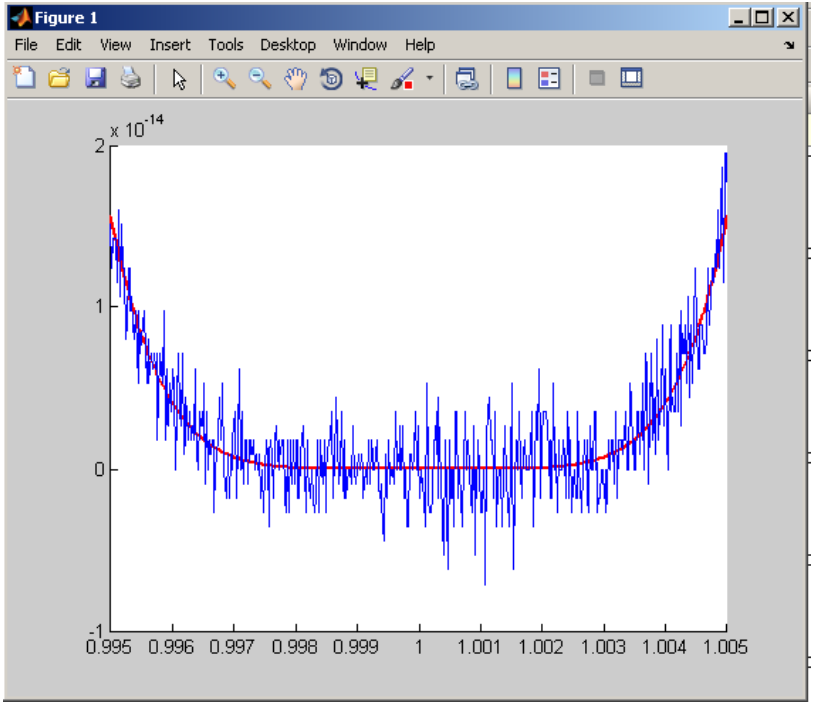
2.5.2. Egy érdekes és tanulságos ábra a numerikus számítások hibájáról

Nem mindegy, hogy egy képletet hogyan számoltatunk ki. Például az $(1 - x)^6$ kifejezés 6. hatványozással, vagy a két tag hatványának felbontásával adódó kifejezéssel is kiszámíttatható.

$(1 - x)^6 = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$ (ld. binomiális együtthatók).

```
x = linspace(0.995, 1.005, 501);

y1 = (1 - x).^6;

y2 = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 + 15*x.^2 - 6*x + 1

hold on

plot(x, y1, 'LineWidth', 2, 'Color', [1 0 0]);

plot(x, y2, 'LineWidth', 1, 'Color', [0 0 1]);

hold off
```

A probléma ott van, hogy az y2 első hat tagja összességében nagyon -1 közeli értéket ad, és két 1-hez közeli érték különbsége nagyon nagy relatív pontatlansággal bírhat. A probléma leírása Csendes Tibor egyetemi jegyzetében ([5]) található.



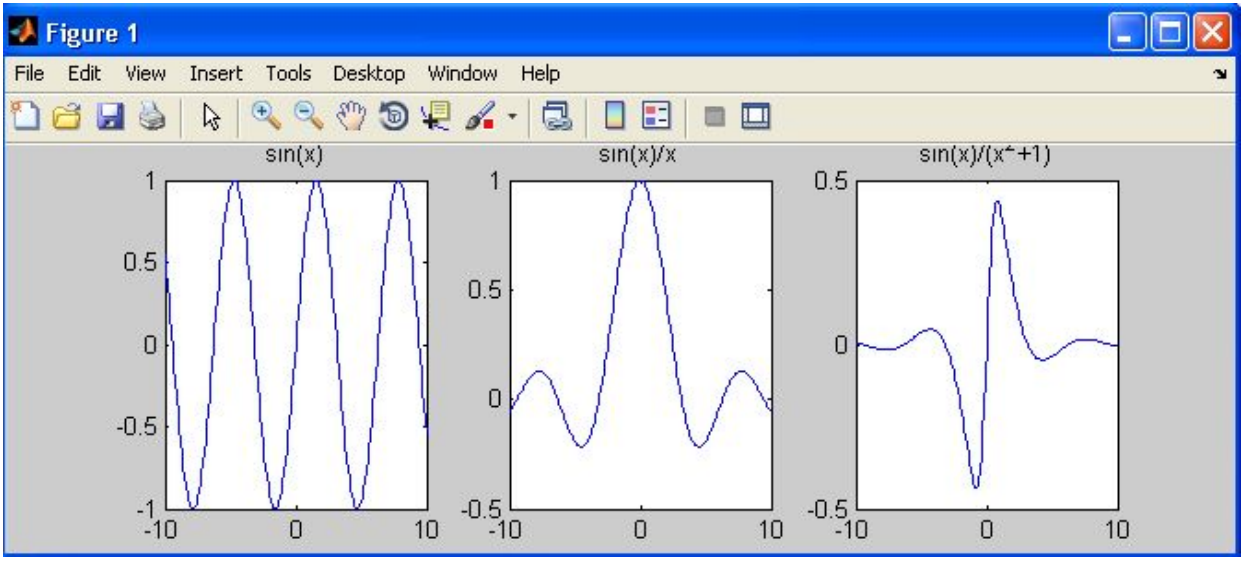
2.17. ábra: Numerikus hibaterjedés

2.5.3. Subplot utasítás

A **subplot** utasítással egy képtáblázat megfelelő cellájába illesztjük be a következő plot utasítással elkészülő képet.

A subplot(n, m, p) utasítás az n sorú és m oszlopú képtáblázat p-edik, sorfolytonosan számozott celláját teszi aktívvá. A plot utasítás itt helyezi el a következő képet. Most egy 1*3 méretű képtáblázatba három függvénydiagramot helyezünk el:

```
>>x=-10:0.1:10;

y=sin(x);

subplot(1,3,1);

plot(x,y); title('sin(x)');

y=sin(x)./x;

subplot(1,3,2);

plot(x,y); title('sin(x)/x');

y=sin(x)./(x.^2+1);

subplot(1,3,3);

plot(x,y); title('sin(x)/(x^2+1)');
```



2.18. ábra: Subplot

2.5.4. Az fplot utasítás

Az **fplot** utasítás/függvény segítségével egyváltozós függvények grafikonját rajzoltathatjuk ki. A plot utasítással szemben az az előnye az, hogy az adott matematikai függvény automatikus változó osztással, a gyorsabb függvényértékváltozásoknál sűrűbben kerül kiértékelésre. Ezáltal sokkal kifejezőbb grafikus képet kapunk.

Az fplot függvény meghívása előtt az adott matematikai függvényt egy M-file-ban definiálhatjuk, vagy a képletét az fplot függvény első paramétereként aposztrófok között adhatjuk meg. Ez utóbbi esetben a független változón kívül más paraméter nem szerepelhet benne, csak konstansok. A második paraméter az intervallum megadása [tól ig] formában, a harmadik paraméter pedig a vonalspecifikáció.

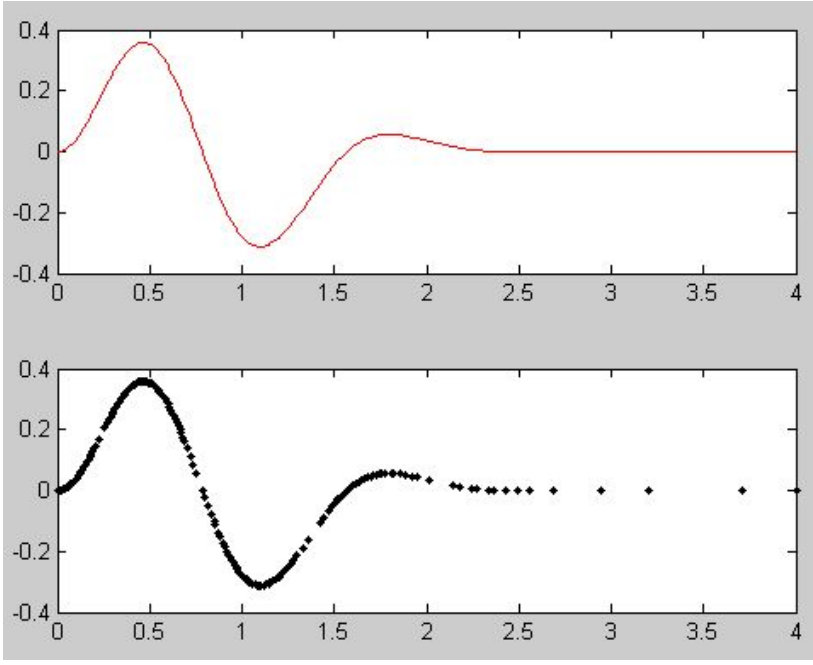
Az fplot illusztálásakor példaként használt fv függvény definíciója a következő:

```
function y = fv(t);

  y = t*exp(-t^2)*sin(4*t);
```

Ha az fplot hívás [x y] visszaadott értékeit eltároljuk, akkor a grafikon nem jelenik meg. Ezt felhasználva előállítjuk az fv függvény sima vonalas rajzát fplot-tal, majd az fplothoz szükséges x, y koordinátapárokat feljegyezve egy plot hívással csak az x, y koordinátapárokkal megadott pontsorozatot rajzoltatjuk ki.

```
subplot(2,1,1);

fplot('fv',[0,4], 'r');  % sima vonalas függvényrajz

% feljegyezzük az fplothoz generált koordinátákat
[x,y] = fplot('t*exp(-t^2)*sin(4*t)',[0,4]);

subplot(2,1,2);

plot(x,y,'k.');
```

A 2.19. ábrán látható, hogy az fplot-hoz szükséges intervallumfelosztás lépésköze nem lett egyenletes!



2.19. ábra: Fplot és az osztáspontjai

Az fv függvény definíciójában nem használtuk a (.) pont műveletminősítő előtagot, és a függvényünk képe mégis jó lett. Jegyezzük meg, az a biztos, ha a mátrixműveletek helyett az elempáronkénti műveleti jeleket használjuk:

```
function y = fv(t);

  y = t.*exp(-t.^2).*sin(4*t);
```

Önellenőrzés

1. A háromdimenziós térben két nem párhuzamos vektor meghatároz egy síkot (és annak a normálvektorát). Az A = [1 2 -1; 3 1 4]', B = [3 0 2; -1 2 -2]' mátrixok így meghatároznak egy-egy síkot (2 dimenziós alteret). Ellenőrizzük azt, hogy a normálvektoraik szöge megegyezik az alterek szögével!
   A megoldás megtekintéséhez kattintson ide!
2. Az **Ax** = **b** egyenletrendszer együtthatóit az M = [A b] mátrix tartalmazza. Az M mátrix elemei a következő módon adottak:

   ```
   M=[2 1 -5 3 6; 3 -1 4 2 25; -1 -2 3 -2 1; 1 2 1 2 5]
   ```

   Ha egy mátrix minden eleme egész, akkor a determinánsa is az. Mennyi az **A** mátrix determinánsa pontosan? Oldjuk meg az egyenletrendszert! A kapott **x** vektor elemeit a rats() függvény segítségével írassuk ki tört alakban!
   A megoldás megtekintéséhez kattintson ide!
3. A Hilbert mátrixok gyengén kondícionált mátrixok. Tekintsük a következő egyenletrendszert: **Hx** = **b**, ahol

   ```
   H = hilb(4), b = [13/6, 7/6, 49/60, 19/30]'
   ```

   Ennek elméletileg pontos megoldása, amiről behelyettesítéssel meg is győződhetünk:

   ```
   x = [2, -1, 2, 0]'
   ```

   Módosítsuk a **H** mátrixot úgy, hogy a második sor első elemét 1 ezrelékkel megnöveljük. Mennyi lesz a megoldásvektor elemeinek abszolút-értékbeli változása? Az egyenletrendszerek megoldását a H\b formula segítségével állítsuk elő!
   A megoldás megtekintéséhez kattintson ide!

4. Bármely invertálható mátrix kondíciószáma nem lehet kisebb 1-nél. Ellenőrizzük mind a négy norma felhasználásával hogy az egységmátrixok (pl. eye(5)) kondíciószáma teljesíti ezt! Miért nem szeretjük használni mátrixokra a Frobenius normát?
   A megoldás megtekintéséhez kattintson ide!
5. Határozzuk meg az A = [5 1 -2 1; 1 6 -1 1; -2 -1 4 2; 1 1 2 8] szimmetrikus mátrix sajátvektorait és sajátértékeit! Ellenőrizzük a sajátértékegyenlet teljesülését is!
   A megoldás megtekintéséhez kattintson ide!
6. Ábrázoljuk az $r = R * (1 + cos(\varphi)),\ R = 2,\ 0 \leq \varphi \leq 2\pi$ polárkoordinátás egyenletű kardiodidot!
   A megoldás megtekintéséhez kattintson ide!
7. Ábrázoljuk az $m = 10$ várható értékű és a $d = 2.5$ szórású normális eloszlású valószínűségi változó sűrűségfüggvényét a [0 20] intervallumban!

$$f(x) = \frac{1}{d\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2d^2}}$$

   A megoldás megtekintéséhez kattintson ide!
8. Ábrázoljuk az f: t 1 t*exp(-t^2) függvényt a [0, 3] intervallumban! Tegyünk fel a rajzra tengelyfeliratokat és a címet is!
   A megoldás megtekintéséhez kattintson ide!

10. LECKE

Haladó grafika

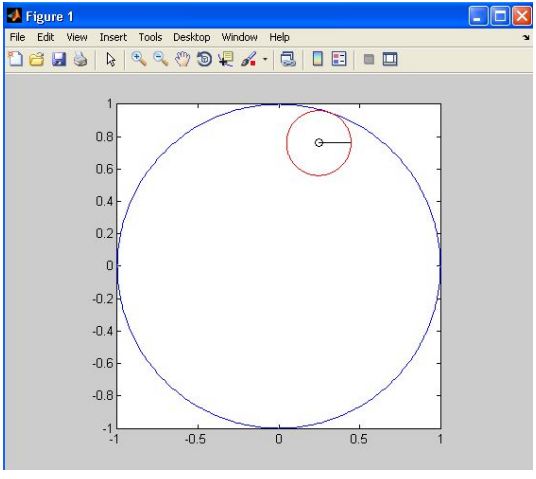
2.5.5. Grafikák fájlba mentése, animáció készítés

Ha valamilyen iterációban ugyanarra az ábrára gyakori időközönként egy pici változtatást ráhelyezünk, akkor egy egyszerű animációt tudunk végrehajtani. Egy aktuális ábrát a **print** utasítással menthetjük fájlba a képtípus és fájlnév megadásával.

A következő példában egy nagyobb kör belső kerületén végiggördülő kisebb kört látunk, a végső képet fájlba mentjük. Az ábra köreinek rajzolása a komplex számok exponenciális alakjának felhasználásával a legegyszerűbb. A küllőt a két végpontjának helyzetével szintén komplex számként adjuk meg. A küllő irányszögét ívhossz-egyeztetéssel számoltuk ki.

```
>>R=1; r=0.2;

z=R*exp(i*linspace(0,2*pi,361)); % nagy kör fokonként megadott pontokkal

w=r*exp(i*linspace(0,2*pi,361)); % kicsi kör

axis square; % az ábra méretezését négyzetesre váltjuk

for k=1:length(z)

   plot(z, 'b' ); % nagy kör kékkel

   hold on  % közös ábrára

   plot((R-r)*z(k)+w, 'r'); % a kicsi kör pirossal

   plot((R-r)*z(k), 'ok'); % a kicsi kör közepe fekete

   kullo=[(R-r)*z(k)  (R-r)*z(k)+r*exp(-i*(R-r)/r*angle(z(k)))];
```

```
    plot(kullo, 'k'); % küllö feketével
    hold off  % hogy a következö rajz üres vászonra menjen
    axis square
    pause(0.05) % késleltetés
end;
>>print --djpeg korok; % a végsö rajz korok.jpg fájlba mentése
```



2.20. ábra: Gördülő kör animációja

Az elkészült animáció avi fájlba is elmenthető, ehhez jelenetenként kell a frame-ket a fájlhoz adni:

```
mov = avifile('koranim.avi'); % új avi-fájl létrehozás

R=1; r=0.2;

z=R*exp(i*linspace(0,2*pi,361));

w=r*exp(i*linspace(0,2*pi,361));

axis square;

for k=1:length(z)

   plot(z, 'b');

   hold on;

   plot((R-r)*z(k)+w, 'r');

   plot((R-r)*z(k), 'ok');

   kullo=[(R-r)*z(k) (R-r)*z(k)+r*exp(-i*(R-r)/r*angle(z(k)))];

   plot(kullo, 'k');

   hold off;
```

```
    axis square;

    F = getframe(gca); % jelenet pillanatfelvétele

    mov = addframe(mov,F); % pillanatfelvétel filmhez adása

end;

mov=close(mov); % film lezárása
```

A küllővégpont pályájának kirajzolásával kibővített avi fájl: koranim.avi

A film elkészülte után a következő üzenetet kapjuk:

```
Adjustable parameters:

               Fps: 15.0000

       Compression: 'Indeo5'

           Quality: 75

    KeyFramePerSec: 2.1429

         VideoName: 'koranim.avi'

Automatically updated parameters:

        Filename: 'koranim.avi'
```

```
      TotalFrames: 361
            Width: 344
           Height: 344
           Length: 24.0667
        ImageType: 'Truecolor'
     CurrentState: 'Closed'
```

2.5.6. Interaktív grafika

Egy grafikus ábrán több lehetőségünk van a beavatkozásra, vagy a pozíciók leolvasására. Ennek legfontosabb eszköze a **ginput**() függvény, amely aktív grafikus ablak mellett működik:

```
[x_koordináták, y_koordináták, billentyükódok]=ginput(n)
```

ahol n a kért kattintások száma, az output elemek pedig oszlopvektorok. A beolvasott koordináták a grafika kirajzolt intervallumába esnek. A kapott billentyűkódok:

1 – bal egérgombot nyomtak

2 – középső egérgombot nyomtak

3 – jobb egérgombot nyomtak

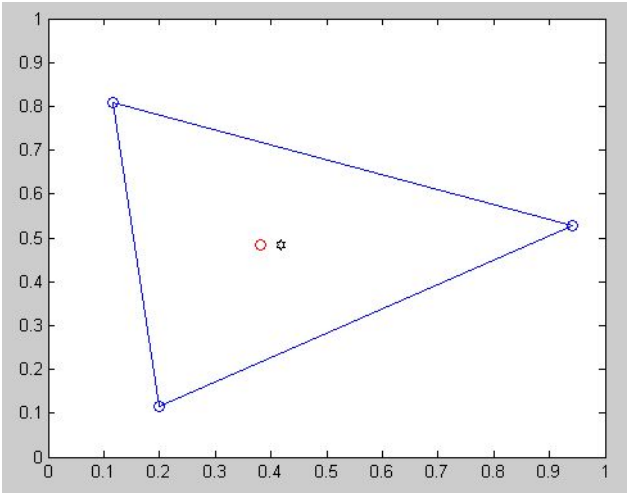
egyéb – a lenyomott billentyű ASC kódja

A következő kis példánkban az üres ábra felületén ki kell jelölni egérrel 3 pontot, ekkor még a vászon méretezése az első síknegyed egységnégyzete. A felrajzolt háromszög súlypontjára kell a lehető legjobban tippelve kattintani. A súlypont megcélzása után felrajzoljuk a tippelés helyét, a tényleges súlypontot és minősítést is adunk.

```
m=figure; % új üres ábra, az m mutató segít majd a törléskor

t=0;

while sum(t)\~{}=3  % mindaddig, amíg mindhárom kattintás nem a balegér

  disp('Bökj a bal egérrel három különböző pontra!');

  [x y t]=ginput(3); % oszlopvektorokba az adatpárok

end;

axis([0 1 0 1]);

s=[mean(x), mean(y)];   % súlypont

x=[x' x(1)]; y=[y' y(1)];  % sorvektorba a koordináták, és az elsővel lezárva

plot(x,y, 'ob-');  % háromszög rajza

axis([0 1 0 1]);

disp('Bökj a háromszög feltételezett súlypontjára bal egérrel! Majd Enter!');
```

```
[p q]=ginput(1); pont=[p q]; % tippelés

hold on;

plot(p,q, 'or'); % tipp kirajzolás

tavolsag=norm(s-pont)   % euklideszi távolság
```

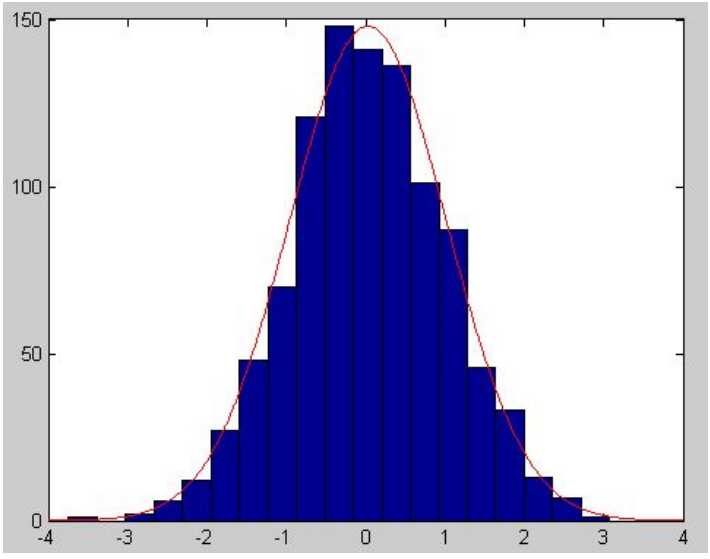
```
if tavolsag$<$0.0001 minosit='találat!';
 elseif tavolsag<0.01 minosit='kiváló!';
 elseif tavolsag<0.05 minosit='jó!';
 elseif tavolsag<0.1 minosit='közeli!';
 else minosit='pancser!';
 end
 disp(minosit)
 % a tipp minösítése
 waitforbuttonpress;
 plot(s(1),s(2), 'hk');
 % súlypont kirajzolás
 pause(5);
 delete(m);
 % ábra letakarítás
```



2.21. ábra: Interaktív grafika, súlypontkeresés

A második példánkban a korábban elmentett normális eloszlású adatmintánkat visszatöltjük, és a hisztogramjára felrajzoltatjuk a legjobban illeszkedő Gauss-görbét.

```
>> load x_normal.dat; x=x_normal; atlag=mean(x); szoras=std(x);

z=linspace(-4,4,801);

y=exp(-(z-atlag).^2/(2*szoras))/(sqrt(2*pi)*szoras);

% az aktuális normális eloszlás sürüségfüggvénye, a Gauss-görbe

hist(x,19) % hisztogram 19 bin-nel

[p q]=ginput(1);

% egérrel leolvassuk a maximális gyakoriságot egy bin-ben

hold on; plot(z,q/max(y)*y,'r'); hold off;

% a jól illeszkedö sürüségfüggvényt felrakjuk arányos nagyítással az ábrára
```



2.22. ábra: Hisztogram Gauss-görbével

2.5.7. Kétváltozós függvények megjelenítése

Kétváltozós függvény megjelenítéséhez az *xy* síkon egy rácshálót kell létrehozni, amelynek csomópontjai felett lehet megjeleníteni az $f(x, y)$ kétváltozós függvény értékei által meghatározott pontokat. A felület megjelenítését sokféle módon teszi lehetővé a Matlab.

Először nézzük a rácsháló(rácsozat) előállítását! Az $f(x, y)$ függvény értékeit az x sorozat és az y sorozat minden lehetséges párjára elő kell állítani. Éppen ezért két, a rácsháló méretével azonos méretű mátrixot kell generálni. Ezek speciális tartalmúak. Az X mátrix minden egyes sora az x sorvektort tartalmazza, az Y mátrix minden egyes oszlopa az y vektort tartalmazza. Ez fogja lehetővé tenni, hogy minden lehetséges koordinátapárra a Z = f(X, Y) értékeket, a függvényértékek mátrixát pontonként előállítsuk. Itt nagyon kell vigyázni, hogy a mátrixműveletek

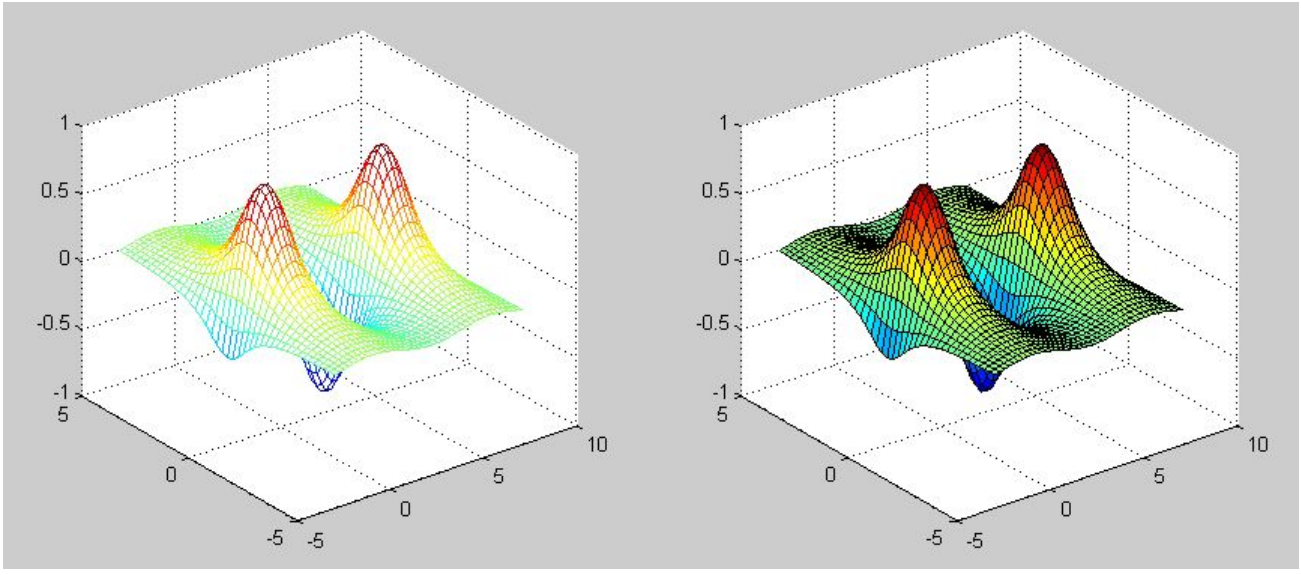
helyett elempáronkénti kiértékelést előíró műveleti jeleket, azaz a pont előtagú műveleti jeleket használjuk! A rácsozat két mátrixát a **meshgrid**() függvény állítja elő.

```
>> x=linspace(1,3,3), y=linspace(10,40,4)

[X Y] = meshgrid(x, y)

x =

     1     2     3

y =

    10    20    30    40

X =

     1     2     3

     1     2     3

     1     2     3

     1     2     3
```

```
Y =

    10    10    10

    20    20    20

    30    30    30

    40    40    40
```

Ha ezzel elkészültünk, akkor már csak a felület felrajzolása van hátra, ennek legegyszerűbb parancsai: **mesh**(X, Y, Z) és **surf**(X, Y, Z). Nézzünk egy példát!

```
>>x=-3:0.25:7; y=-5:0.25:5;

[X Y]=meshgrid(x,y);

Z=cos(1+X).*sin(1./(1+Y.^2));

subplot(1,2,1);

mesh(X,Y,Z);

subplot(1,2,2);

surf(X,Y,Z);
```



2.23. ábra: Kétváltozós függvény ábrázolása a ráccsal

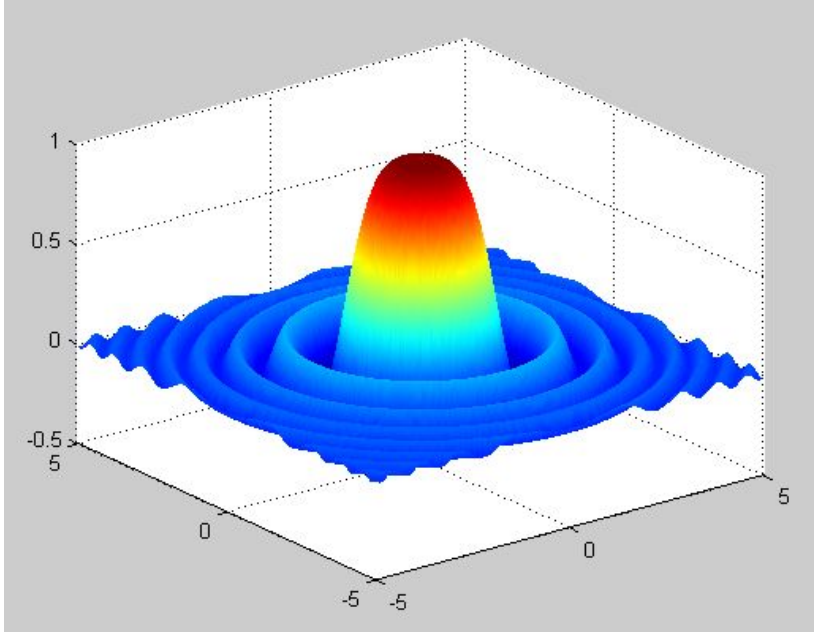
A színek vezérlését is sok parancs támogatja. Mi ezek közül a szép színátmenetet produkáló

```
shading interp;
```

parancsot választottuk ki. Illusztrálásként egy példa:
Rajzoljuk ki az $f(x, y) = \sin(x^2 + y^2)/(x^2 + y^2)$ függvény grafikonját az [-5, 5]x[-5, 5] tartomány felett!

```
>> x=-5:0.05:5;

y=x;
```

```
[X,Y]=meshgrid(x,y);

Z=sin(X.*X + Y.*Y)./(X.*X + Y.*Y);

surf(x,y,Z);

shading interp;
```



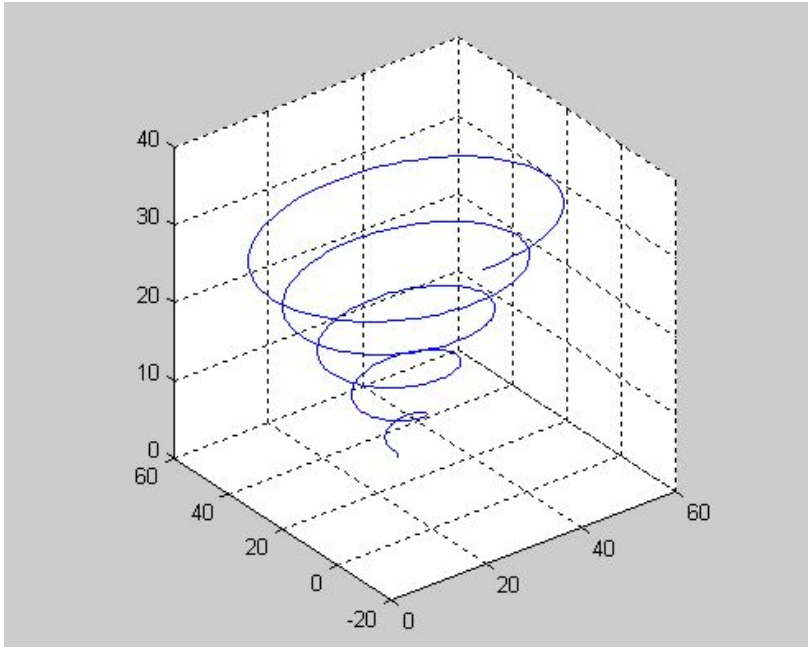
2.24. ábra: Kétváltozós függvény színátmenettel

2.5.8. Térgörbék

A **plot3**(x, y, z) utasítással a térgörbe pontjainak x, y, z koordinátáit tartalmazó vektorok segítségével kérhetjük a rajzolást.

A következő példánk egy szűkülő kúpra rátekert fonalat ábrázol.

```
t = 0:pi/50:10*pi;

plot3(30-t.*sin(t),30-t.*cos(t),t);

axis square; grid on
```



2.25. ábra: Térgörbe

Önellenőrzés

1. Az előző lecke 8. feladatának grafikonján olvastassuk be egérrel a grafikon móduszának („púpjának”) koordinátáit közelítőleg. A kapott két értéket írassuk ki a grafikonra text-ként!
   A megoldás megtekintéséhez kattintson ide!
2. Rajzoljuk ki az

$$f(x, y) = x^3 \cos(y)$$

   függvény grafikonját az x: $[-2\pi, 2\pi]$ és y: $[-2\pi, 2\pi]$ tartomány felett. A rács először ne legyen túl sűrű, csak 50x50-es.
   A megoldás megtekintéséhez kattintson ide!
3. Rajzoltassuk ki egy hengerre felcsavarodó fonál 5 menetének a képét! A térgörbe paraméteres egyenlete: $x = \cos(t)$; $y = \sin(t)$; $z = t$.
   A megoldás megtekintéséhez kattintson ide!

11. LECKE

Gyakori alkalmazások

2.6. A Matlab alkalmazása gyakori matematikai feladatokra

2.6.1. Algebrai alapfeladatok, szimbolikus számítások

A Matlab alapvető algebrai feladatok megoldását is támogatja. Korábban már láttuk, hogy a **factor**() függvény a tényezőkre bontás elvégzésére használható. Ott azt is megemlítettük, hogy a Matlabnak van formális számítások modulja. Nézzük meg közelebbről, hogyan működik!

A legnagyobb közös osztót (lnko) az ún. euklideszi algoritmussal határozhatjuk meg hatékonyan. Ha p és q egészek, akkor az $r = \mathrm{mod}(p, q)$; $p = q$; $q = r$; műveletsort mindaddig kell ismételni, amíg q értéke 0 nem lesz. A **mod**() függvény az osztás maradékát állítja elő.

```
>> p=460071, q=142569

p =       460071

q =       142569

>> while q>0 r=mod(p,q); p=q; q=r; end; lnko=p

lnko =    279
```

Ezt pedig a factor() függvénnyel, vagy a mi primbont() függvényünkkel tényezőkre bonthatjuk.

```
>> primbont(lnko)

279 = 3 * 3 * 31
```

A Matlabnak a **gcd** (Greatest common divisor) függvénye az euklideszi algoritmus kibővített változatát használja, azaz a legnagyobb közös osztó mellett a

$$px + qy \equiv \text{lnko}(x, y)$$

ún. diofantikus egyenlet egy megoldását is szolgáltatja.

```
>> [lnko,x,y]=gcd(460071,142569), 460071*x + 142569*y

lnko =   279

x =   163

y =  -526

ans =   279
```

A legkisebb közös többszöröst az **lcm** (Least common multiple) függvény szolgáltatja:

```
lkkt=lcm(460071,142569), primbont(lkkt)

>> lkkt=lcm(460071,142569), primbont(lkkt)

lkkt =   235096281

235096281 = 3 * 3 * 7 * 17 * 31 * 73 * 97
```

Ha a számok túl nagyok (2^{52}-nél nagyobbak), akkor ezek a függvények nem működnek. Ilyenkor jön segítségül a Matlab Symbolic Toolbox bővítménye.

A szimbolikus számításokhoz szimbolikus objektumokat kell először létrehozni a **sym**() függvénnyel, vagy a **syms** utasítással, majd a szimbolikus objektumokra végrehajtatjuk a kért műveletet.

Határozzuk meg két polinom, az $x^3 - 3x^2 + 3x - 1$ és $x^2 - 5x + 4$ legnagyobb közös osztóját, majd a legkisebb közös többszörösét, és végezetül ennek a tényezőkre bontását!

```
>> syms x

gcd(x^3 - 3*x^2 + 3*x - 1,  x^2 - 5*x + 4)

lcm(x^3 - 3*x^2 + 3*x - 1,  x^2 - 5*x + 4)

factor(ans)

ans =

x - 1

ans =

x^4 - 7*x^3 + 15*x^2 - 13*x + 4

ans =

(x - 4)*(x - 1)^3
```

Természetesen a felbontást egy tetszőleges polinomra is elvégeztethetjük.

```
>>factor(x^9-1)

ans =

(x - 1)*(x^2 + x + 1)*(x^6 + x^3 + 1)
```

Most nézzük meg ezt a túl nagy számokra is!

```
p=sym('8888888888'), q=sym('4444488888')

lnko=gcd(p,q), lkkt=lcm(p,q), factor(lkkt)

p = 8888888888

q = 4444488888

lkkt = 444453333288888

ans =

2^3*3*7*11*41*271*2381*9091
```

Az alap- és a kibővített euklideszi algoritmus iránt részletesebben érdeklődő olvasónak a [7] irodalmat ajánljuk.

2.6.2. Egyváltozós függvény gyökeinek keresése

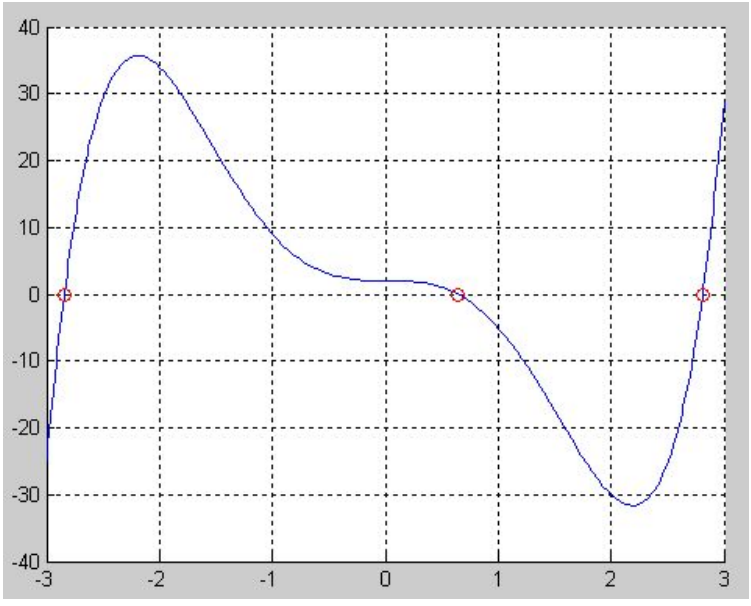
Az **fzero**(függvény, hol) parancs iterációs módszerrel keresi meg a megadott függvény egy zérushelyének jó közelítését. Az első paraméter a függvény képlete, vagy a neve aposztrófok között, a második paraméter az intervallum (2 elemű vektor), ahol a gyököt keressük, vagy az iterációs módszer kezdeti értéke.

Példák:

```
>> fzero('cos',[1,2])

>> fzero('cos',[-1,1])

>> fzero('sin',3)
```

Legyen az f1.m fájl tartalma:

```
function y=f1(x)

y=x.^5-8*x.^3+2;

% A gyököket külön-külön megkeressük

x = [fzero('f1',0) fzero('f1',-4) fzero('f1',4)]

z = [0 0 0];

% kirajzoljuk a függvényt és berajzoljuk a zérushelyeket

hold on; fplot('f1',[-3,3]); plot(x,z,'or'); grid on; hold off;
```



2.26. ábra: Egyváltozós függvény és zérushelyei

```
x =

    0.6411   -2.8438    2.8125
```

2.6.3. Határozott integrál közelítése

Ha a trapéz szabállyal szeretnénk dolgozni, akkor az intervallumnak vesszük egy x beosztását, és a beosztás pontjaiban előállítjuk a függvényértékeket az y vektorban. A két vektort kell átadni a **trapz** parancsnak.

```
>> x=0:0.1:pi; y=sin(x); trapz(x,y)

ans =

   1.997468926590933

>> x=0:0.001:pi; y=sin(x); trapz(x,y)

ans =

   1.999999657714212
```

A határozott integrált a Matlabban az adaptív Simpson-szabállyal is közelíthetjük. Ekkor a **quad** parancsot használjuk. Az első paraméter a függvény képlete illetve neve, a második és harmadik paraméter az intervallum kezdete és vége. Megadható egy negyedik paraméterben, hogy milyen pontosságú legyen a közelítő érték.

```
>> quad('sin',0,pi)

ans =

   1.999999996398431

>> quad('sin',0,pi,eps)

ans =

   2.000000000000000

>> quad('f1',0,1,eps)
```

```
ans =

    0.166666666666667
```

A kétváltozós függvények határozott integráljának közelítő meghatározására szolgáló **dblquad** parancs meghívása hasonló a quad függvényéhez. Legyen az f2.m fájl tartalma:

```
function z=f2(x,y)

z=x*y-3*x-y^3;
```

Számítsuk az első síknegyedbeli egységnégyzet feletti integrálját!

```
>> dblquad('f2',0,1,0,1,eps)

ans =

  -1.500000000000000
```

2.6.4. Függvények minimum- és maximumhelye

Egyváltozós függvények minimumhelyét az **fminbnd** függvény segítségével határozhatjuk meg. Először meg kell adni a vizsgált függvényt, a második és a harmadik paraméter pedig az intervallum alsó és felső határa:

```
>> x=fminbnd('f1',0,3)

x =   2.190895443959338
```

A negyedik paraméter az opciók beállítására szolgál, és az fminbnd valójában három értéket képes visszaadni:

```
>> [x fx kod]=fminbnd('f1',0,3, optimset('TolX',eps)), f1(x)

x =

   2.190890240855162

fx =

 -31.652073933117400

kod =     1

ans =  -31.652073933117400
```

A kód értékei:

1 fminbnd konvergált az érvényes opciók mellett (minden OK)

0 elértük a lépésszámkorlátozást

-1 a függvény miatt terminálódás (pl. negatívból gyökvonás)

-2 inkonzisztens határok

Az **fminsearch** függvény többváltozós függvény minimumhelyének megkeresésére szolgál, és direkt kereséssel dolgozik. A beépített algoritmus a Nelder-Mead szimplex eljárás. A módszer legegyszerűbb meghívása: első paraméter a függvény neve, a második paraméter az iterációs módszer kezdeti értéke.

Az f3.m fájl definiálja az f3() függvényt:

```
function z=f3(x)

z=(x(1)-1)^2+3*(x(2)-x(1))^2;
```

Indítsuk a keresést az origóból!

```
>> x=fminsearch('f3',[0,0], optimset('TolX',eps))

x =

   1.000000000000000   1.000000000000000
```

A beállítható opciók:

Display, TolX, TolFun, MaxFunEvals, MaxIter, FunValCheck, PlotFcns, OutputFcn

Ezek hasonlók az fminsearch opcióihoz.

Kezdő felhasználók számára sokszor meglepő az a tény, hogy a Matlabban a maximumhely keresésére nincs külön beépített függvény. Ehhez a feladathoz is a minimumhely keresésre szolgáló függvények használatosak, az $f(x)$ maximumhelye = $-f(x)$ minimumhelye megfeleltetéssel!

2.6.5. Függvényvizsgálat az eddig tanultak alapján

Határozzuk meg az alábbi függvény zérus helyeit és szélsőérték helyeit a [2; 8] intervallumban! Készítsünk címmel, jelmagyarázattal és rendezővonalakkal ellátott grafikont, amelyen a meghatározott pontokat is megjelöltük!

$$g1(x) = 1{,}4e^{-0{,}3x} + 0{,}4sin(x) - 0{,}1$$
$$g1\_deriv(x) = -0{,}42e^{-0{,}3x} + 0{,}4cos(x)$$

(Itt $g1$_deriv(x) a $g1(x)$ függvény deriváltja.)

Első lépésként a függvények definíciós M-fájljait helyezzük el a munkakönyvtárunkban.

```
function f=g1(x)

f=1.4*exp(-0.3*x)+0.4*sin(x)-0.1;

function f=g1_(x)

f=-0.42*exp(-0.3*x)+0.4*cos(x);
```

Ezt követően közös ábrán elhelyezzük a függvényeket:

```
hold on;

fplot('g1', [2,8],'b');

fplot('g1_deriv',[2,8],'r');

grid on;

title('g1(x)= 1.4*exp(-0.3*x)+0.4*sin(x)-0.1 függvény és deriváltja');
```

Ezután megkeressük a zérushelyeket. Az első zérushely 4 körül van:

```
xz=fzero('g1',4);
```

A második 6 környékén:

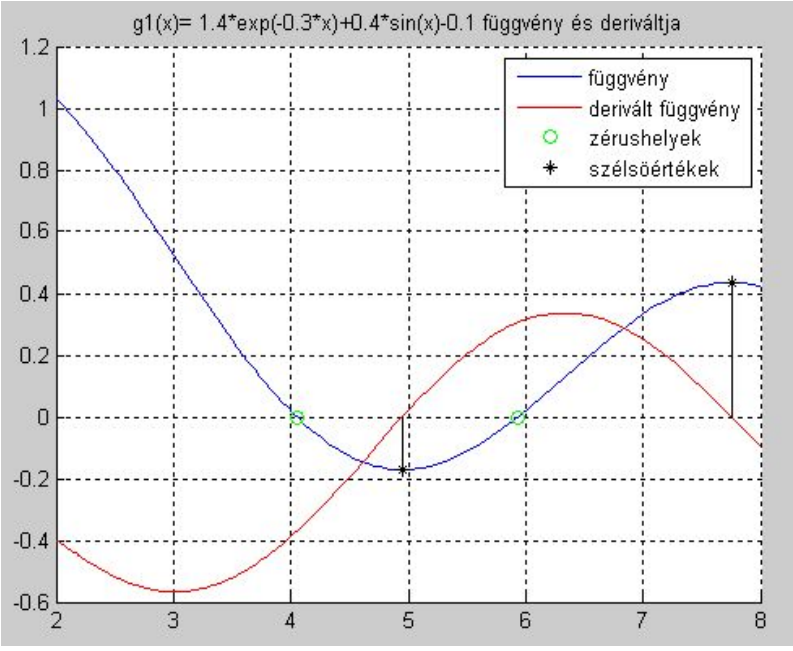
```
xz=[xz fzero('g1',6)];
```

Itt felbővítettük az x vektort a következő zérushellyel, majd az y koordinátákat is megadjuk:

```
yz=[0 0];
```

és kirajzoljuk:

```
plot(xz,yz,'og');
```



2.27. ábra: Függvényvizsgálat

A derivált zérushelyeinek keresése és a szélsőértékpontok kirajzolása:

```
xm=fzero('g1_deriv',5);

xm=[xm fzero('g1_deriv',7.8)];

ym=g1(xm);
```

```
plot(xm,ym,'*k');

% rendezövonalak a szélsöértékpontokhoz

plot([xm(1) xm(1)],[0 ym(1)],'k');

plot([xm(2) xm(2)],[0 ym(2)],'k');

% jelmagyarázatok

legend('függvény','derivált függvény','zérushelyek','szélsöértékek');

hold off
```

Az ábra fájlba mentése (abra1.jpg):

```
print -djpeg90 -r300 abra1
```

2.6.6. Differenciálegyenletek (kezdetiérték-feladat) megoldása

Az

$$x' = f(t,x), \quad x(t_0) = z, \quad x : R \to R'', \quad f : R \times R'' \to R''$$

alakú differenciálegyenlet-rendszerek közelítő megoldását például az **ode45**() függvénnyel számolhatjuk ki.

Ennek meghívása:

1. első paraméter: az egyenlet jobb oldalát definiáló f függvény neve;
2. második paraméter: azon t értékek vektora, ahol a numerikus közelítést ki szeretnénk számolni (a vektor első eleme a kezdeti időpont);
3. harmadik paraméter a z kezdeti érték.

Hasonló, de más numerikus módszert használó differenciálegyenlet megoldó parancsok:

ode23, ode113, ode15s, stb.

Legyen a megoldandó feladatunk a következő alakú:

$$x_1{}' = x_2$$

$$x_2{}' = -x_1 - 2x_2 + cos(3t)$$

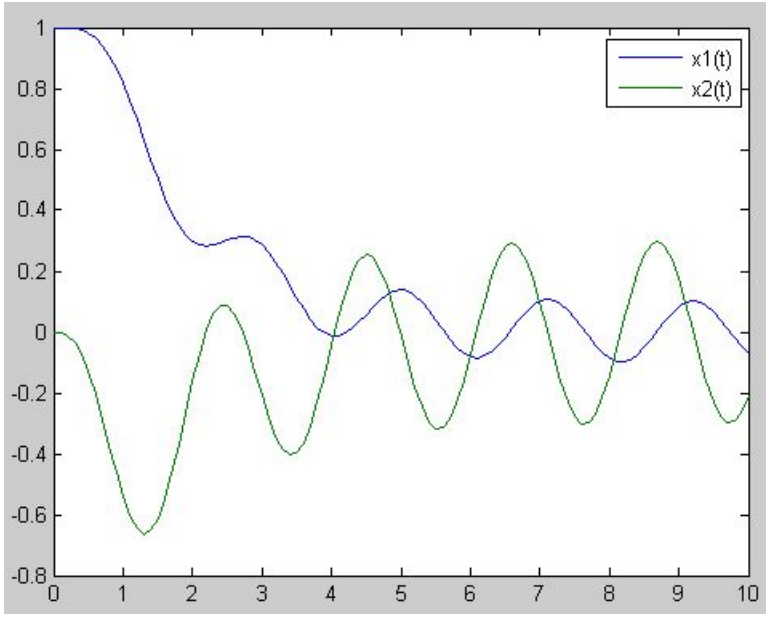
$$x_1(0) = 1, \quad x_2(0) = 0, \quad t \in [0,10]$$

Az egyenlet jobb oldalát definiáló de2.m M-fájl:

```
function y=de2(t,x)

y=[x(2);-x(1)-2*x(2)+cos(3*t)];
```

A megoldás ode45-tel:

```
>>tt=0:0.1:10; [t y]=ode45('de2',tt,[1 0]); plot(t,y);

legend('x1(t)', 'x2(t)', 1);
```



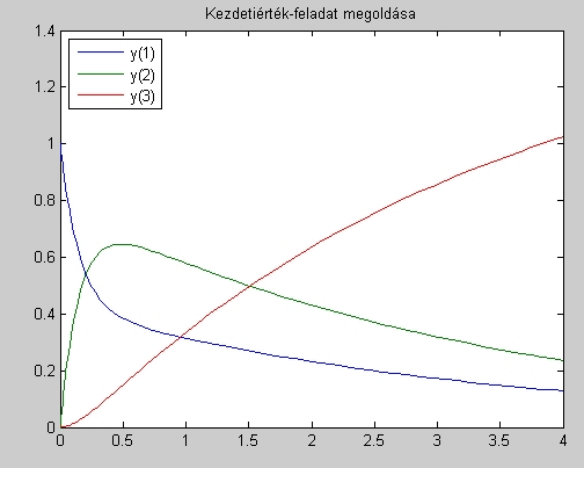
2.28. ábra: Kezdetiérték-feladat (de2.m) megoldása

A kezdeti-érték feladat jobboldalát leíró függvényt ún. inline módon is megadhatjuk (ld. **doc inline**). Oldjuk meg a következő kezdeti érték feladatot!

$$y' = f(t,y) = \begin{cases} -4y_1(t) + 2y_2(t) & y_1(0) = 1 \\ 5y_1(t) - 3y_2(t) & y_2(0) = 0 \\ 0{,}6y_2(t) & y_3(0) = 0 \end{cases} \qquad t \in [0, 4]$$

```
>> f = inline('[(-4*y(1) + 2*y(2)); (5*y(1) - 3*y(2)); 0.6*y(2)]','t','y')
```

```
[t,y] = ode45(f,[0 4],[1 0 0]);

plot(t,y), legend('y(1)','y(2)','y(3)',2),

title('Kezdetiérték-feladat megoldása')

f =

     Inline function:

     f(t,y) = [(-4*y(1) + 2*y(2)); (5*y(1) - 3*y(2)); 0.6*y(2)]
```



2.29. ábra: Kezdetiérték-feladat (inline) megoldása

2.6.7. Spline interpoláció

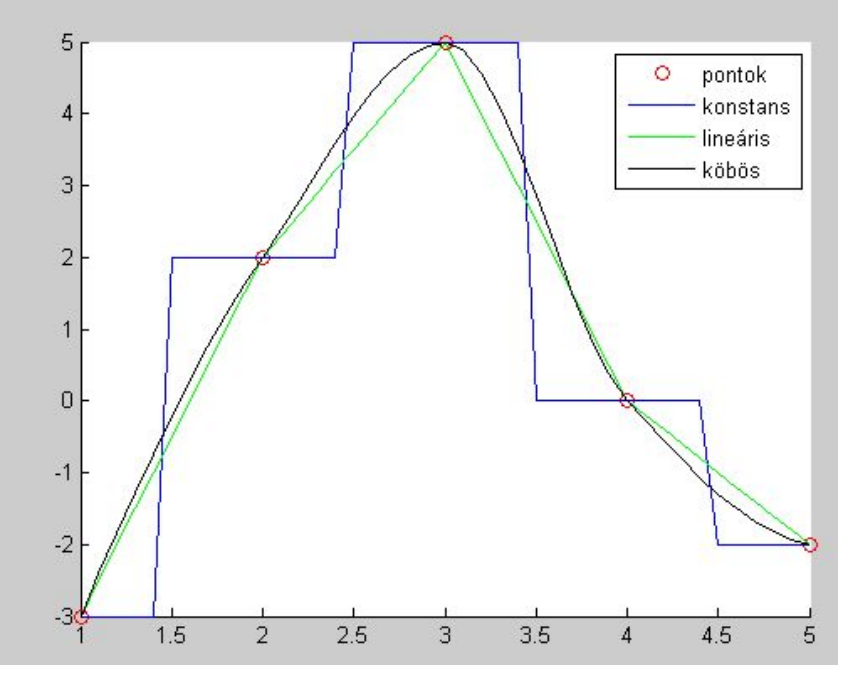
Az interpoláció feladatában hatványfüggvényekkel kötünk össze megadott pontokat.

Az **interp1**() függvény

1. Első két paraméterében az input pontok x illetve y koordinátáit adjuk meg;
2. A harmadik paraméterben azon x értékek koordinátáit adjuk meg, ahol az interpoláló függvényt ki szeretnénk értékelni, a függvény outputja az interpolációs értékeket tartalmazó vektor.
3. A negyedik paraméterben megadhatjuk a 'nearest', 'linear' (default), 'cubic' opciókat, amely az interpolációs módszert határozza meg:
   - 'nearest' szakaszonként konstans;
   - 'linear' szakaszonként lineáris;
   - 'cubic' pedig szakaszonként harmadfokú spline-polinomokkal köti össze a megadott pontokat. A harmadfokú függvények a megadott pontokban teljesen simán csatlakoznak, azaz az első és második deriváltjuk itt megegyezik.

```
>> x =[1, 2, 3, 4, 5]; y=[-3, 2,5,0,-2]; t=1:0.1:5;

hold on;

plot(x,y,'or');

z1=interp1(x,y,t,'nearest'); plot(t,z1,'b');

z2=interp1(x,y,t,'linear'); plot(t,z2,'g');
```

```
z3=interp1(x,y,t,'cubic'); plot(t,z3,'k');

legend('pontok','konstans', 'lineáris', 'köbös', 1);

hold off;
```

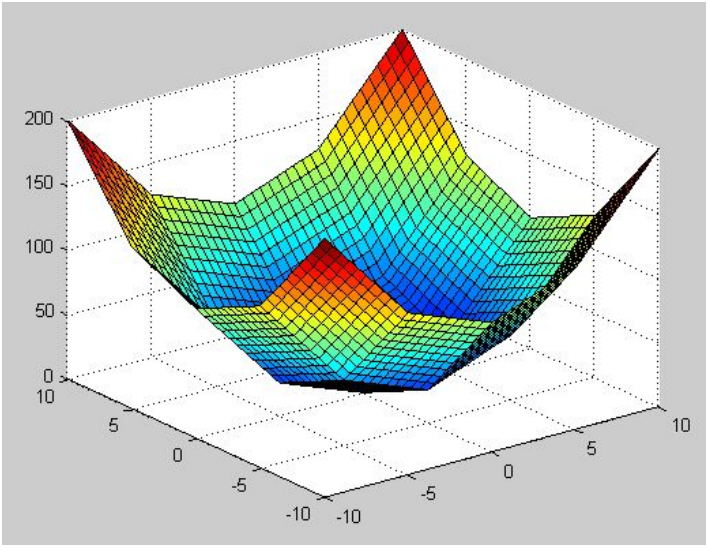


2.30. ábra: Spline interpoláció

2.6.8. Kétváltozós függvények interpolációja

Az **interp2**() függvény a kétváltozós függvények értékeit interpolálja illeszkedő síkidomokkal.

```
>> % kétdimenziós interpoláció

% egy durva rács a [-5,5]x[-5,5] négyzeten

[x,y]=meshgrid(-10:5:10,-10:5:10);

% függvényértékek a rácspontokban

z=x.^2+y.^2;

% egy finomabb rács a rajzoláshoz

[xi,yi]=meshgrid(-10:0.5:10,-10:0.5:10);

% a megadott értékek interpolációja

zi=interp2(x,y,z,xi,yi);

% rajzolás

surf(xi,yi,zi)
```



2.31. ábra: Kétváltozós függvény interpolációja

2.7. Polinomiális regresszió, az eddigi tudásunk alkalmazása

Gyakran van arra szükség, hogy az (x, y) koordinátapárú pontok közé jól illeszkedő (minimális négyzetes hibájú) polinomot (egyenest, parabolát, köbös polinomot, stb. . .) illesszünk. A pontok száma legyen legalább eggyel több, mint az alkalmazni kívánt polinom fokszáma.

Adjunk meg pár pontot, majd vegyünk fel egy kellően finom t sorozatot a minimális és a maximális x értékek között!

```
x = [1, 3 , 6, 10, 12, 16, 20, 24];

y = [ 6, 3, 1, 3,  5,  10, 12, 11];

t = min(x) : (max(x)-min(x))/500 : max(x);
```

Az adott pontok közé illesszünk egyenest, parabolát, köbös polinomot! Az ábrára tegyük fel a determinációs együtthatók (R^2) értékeit is!

Ehhez meg kell oldani az **Va** = **y** alakú túlhatározott lineáris egyenletrendszereket, ahol a **V** együtthatómátrixok rendre a fordított oszlopsorrendű Vandermonde mátrix egyre bővülő $n = 2, 3, 4$ oszlopából állnak.

A kapott $a_{n,k}$ együtthatók annak a $P_n(x)$ polinomnak az együtthatói, amely a

$||P_n(x) - y|| \rightarrow \min$

feladat megoldását jelenti.

```
% egyenesnél:

V =[ones(size(x)) x];

% majd a parabolánál az elözö V mátrix új oszloppal bövítettje:

V = [V x.^2];

% és a köbös polinomnál:

V = [V x.^3];
```

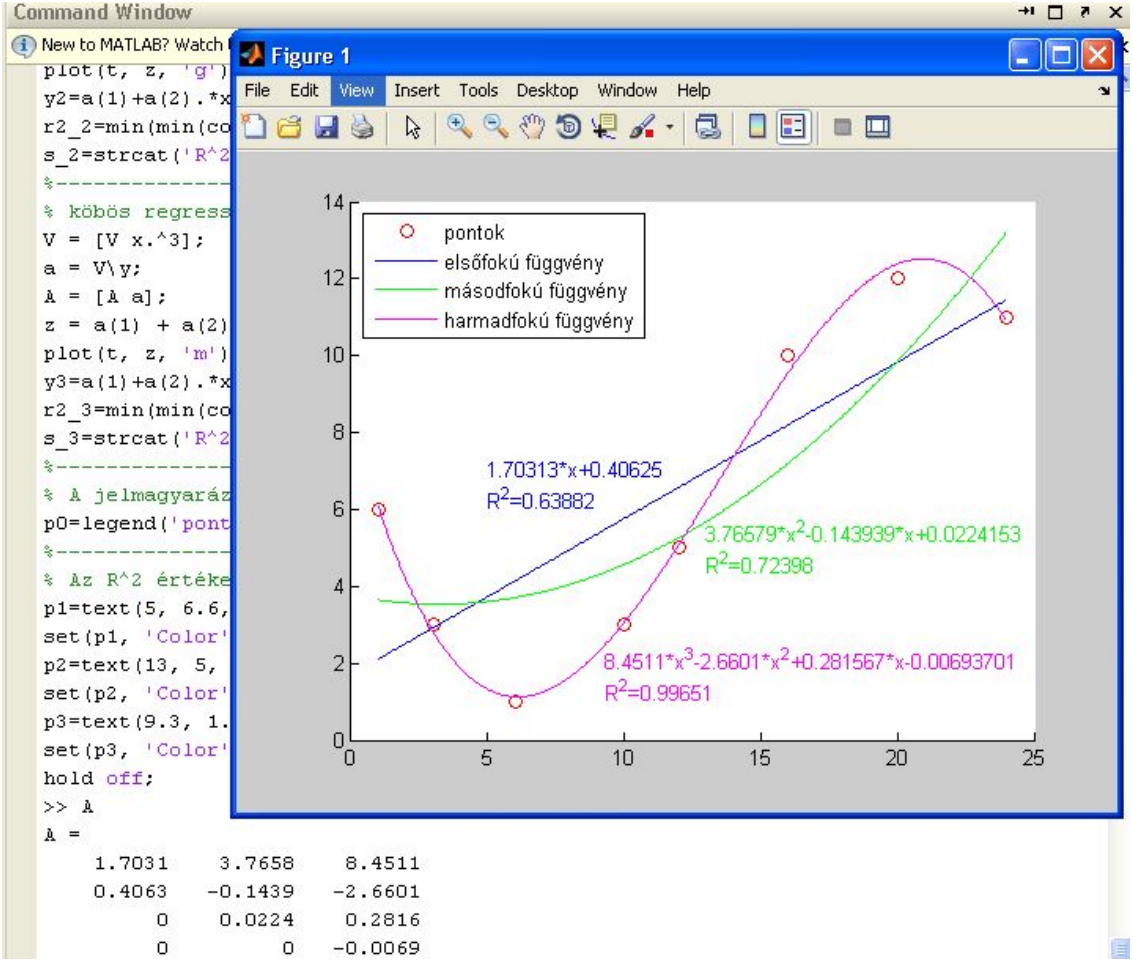
Ha felhasználjuk a már előállított x, y, t vektorokat, akkor a megoldás teljes menete:

```
x = [1, 3 , 6, 10, 12, 16, 20, 24];

y = [ 6, 3, 1, 3,  5,  10, 12, 11];

t = min(x) : (max(x)-min(x))/500 : max(x);

hold off; % elözö képet lezárjuk

hold on;

x = x'; % oszlopvektorrá alakítjuk

y = y';

%-------------------------------------------------------------------------

% lineáris regresszió

V =[ones(size(x)) x]; % együttható mátrix

a = V\y;  %  megoldás a minimális hibájú lesz (ld. pszeudoinverz)

A = [a; 0; 0;];  % a 2 elemüt 4-elemü oszlopvektorrá egészítjük ki

z = a(1) + a(2).*t; % t-pontokbeli lineáris interpoláció
```

```
plot(x, y, 'or', t, z, 'b'); % kék színü egyenes a pontok közé

y1=a(1)+a(2).*x;  % a regressziós egyenesre esö pontok

r2_1=min(min(corrcoef([y y1]).^2)); % R^2 értéke

s_1=strcat('R^2=',num2str(r2_1)); % szöveggé alakítás

%-------------------------------------------------------------------

% parabolikus regresszió

V = [V x.^2];  % az együtthatómátrix már 3 oszlopos lesz

a = V\y;

A = [A [a; 0]]; % A második oszlopába a 4-elemüre kiegészített megoldás kerül

z = a(1) + a(2).*t + a(3).*t.^2;

plot(t, z, 'g'); % zöld színü parabola

y2=a(1)+a(2).*x+ a(3).*x.^2; % a parabolára esö pontok

r2_2=min(min(corrcoef([y y2]).^2)); % R^2 értéke

s_2=strcat('R^2=',num2str(r2_2)); % szöveggé alakítás
```

```
%-------------------------------------------------------------------------

% köbös regresszió

V = [V x.^3]; a = V\y;

A = [A a];

z = a(1) + a(2).*t + a(3).*t.^2 + a(4).*t.^3;

plot(t, z, 'm'); % magentaszínü harmadfokú függvény

y3=a(1)+a(2).*x+ a(3).*x.^2+ a(4).*x.^3; % a harmadfokú függvényre esö pontok

r2_3=min(min(corrcoef([y y3]).^2)); % R^2 értéke

s_3=strcat('R^2=',num2str(r2_3)); % szöveggé alakítás

%-------------------------------------------------------------------------

% A jelmagyarázatot az automatikus jobb felsö sarokból a bal felsöbe kérjük

p0=legend('pontok', 'elsöfokú függvény', 'másodfokú függvény', 'harmadfokú
függvény', 2);

%-------------------------------------------------------------------------

% A polinomokat és R^2 értékeket pedig saját színben jelenítjük meg
```

```
% A kétsoros text kiírást cellatömb segítségével oldjuk meg

p1=text(5, 6.6, [{polytext(A,1)}; {s_1}]);

set(p1, 'Color', 'Blue');

p2=text(13, 5, [{polytext(A,2)}; {s_2}]);

set(p2, 'Color', 'Green');

p3=text(9.3, 1.7, [{polytext(A,3)}; {s_3}]);

set(p3, 'Color', 'Magenta');

hold off;
```



2.32. ábra: Polinomiális regresszió

Megjegyzések

1. A corrcoef([y y1]) parancs 2x2-es korrelációs mátrixot eredményez, amelynek főátlójában 1 van. A belső min parancs sorvektort eredményez a külső pedig csak egy számot.
2. A text()-ek koordinátáit most fixen adtuk meg. A text()-ek leendő pozícióit az addig elkészült ábráról leolvashatjuk, vagy mint azt a korábbi példáinkban is láttuk, a ginput(3) parancs segítségével feljegyzett pozíciók felhasználásával állíthatjuk be.
3. Szövegkezelő függvények (konverzió, konkatenáció, stb.) segítségével a polinomok konkrét alakját is előállítottuk, és text parancsok segítségével az R^2-tet és a polinomalakot tartalmazó cellatömböt az ábrára felraktuk. Az Excel egy aktív pontdiagramra a trend szolgáltatással pontosan ilyen ábrát állít elő.
4. Az **A** mátrix oszlopaiban a regressziós polinomok együtthatóit látjuk:

$P_1(x)$	$P_2(x)$	$P_3(x)$	együttható
1.7031	3.7658	8.4511	konstans
0.4063	-0.1439	-2.6601	elsőfokú
	0.0224	0.2816	másodfokú
		-0.0069	harmadfokú

A polinomok kiírási szövegét a polytext.m függvény állítja elő:

```
% A regressziós polinomok együtthatóit az A mátrix oszlopvektorai

% tartalmazzák, a második paraméter a fokszám, azaz oszlopindex

function s = polytext(A,n);

if n$>$1 s=sprintf('%g*x^%d',A(1,n),n); % elsö tag hatványkitevös
```

```
else s=sprintf('%g*x',A(1,n)); % ha elsöfokú, nincs hatványkitevö
end;
for k=n-1:-1:0 % az egyre csökkenö fokszámú tagok jönnek
  switch k
   case 0
      s=strcat(s,sprintf('%+g',A(n+1-k,n)));
   case 1
      s=strcat(s,sprintf('%+g*x',A(n+1-k,n)));
   otherwise
      s=strcat(s,sprintf('%+g*x^%d',A(n+1-k,n),k));
  end
end
```

Az elkészült ábrát a következő paranccsal menthetjük el:

```
>>print -djpeg90 -r300 pic1
```

A fenti megoldás azt illusztrálja, hogy egy összetett feladatot mi is képesek vagyunk alaplépéseken keresztül megoldani. Természetesen a polinomiális regresszió feladatára beépített függvény is használható.

```
>> p = polyfit(x,y,3)

p =

   -0.0069    0.2816   -2.6601    8.4511
```

Látható, hogy a harmadfokú regressziós polinom együtthatóit egyetlen paranccsal megkaptuk.

Önellenőrzés

1. Ábrázoljuk az

```
function y=f4(x)

y=x.^4-3*x.^3+2*x.^2-0.07;
```

definíciójú függvényt a [-1 2,5] intervallumban!
A megoldás megtekintéséhez kattintson ide!

2. Állapítsuk meg mind a négy zérushelyét növekvő sorrendben!
A megoldás megtekintéséhez kattintson ide!

3. Határozzuk meg az [1,5 2] intervallumbeli minimumhelyét és az itt felvett függvényértéket!
A megoldás megtekintéséhez kattintson ide!

4. Határozzuk meg a második és harmadik zérushely között a határozott integrál értékét eps pontossággal!
A megoldás megtekintéséhez kattintson ide!

5. Határozzuk meg ode45 módszerrel a következő kezdetiérték feladat megoldását a [0 30] intervallumban! Ábrázoljuk is!

```
y' = 10 + 4*sin(x) - 2*sqrt(y); y(0) = 1
```

A megoldás megtekintéséhez kattintson ide!

Modulzáró feladatok

6. Az fplot(szinusz, [0, 2*pi]) parancs a következő hibajelzést adja:

```
??? Error using ==> fcnchk at 108

FUN must be a function, a valid string expression, or an inline function
object.
```

A parancs megváltoztatása nélkül oldjuk meg a problémát! (Segítség: az azonosító legyen valid string expression)

A megoldás megtekintéséhez kattintson ide!

7. Egy időfüggvény $f(x) = amp*\cos(om1*x)*\sin(om2*x) + konst$ alakú, ahol az *amp*, *om*1, *om*2, *konst* globális paraméterek adottak: $amp = 12$, $om1 = 0.07$, $om2 = 1.2$, $konst = 1$.

   1. Helyezzük el a függvény M-fájlját a c:\munka könyvtárban!
   2. Ábrázoljuk a függvényt a [20, 26.6] intervallumban!
   3. Jelöljük meg a zérushelyeket, lokális szélsőértékeket!
   4. A függvénygrafikon 661 egyenlőközűen felvett pontjához illesszünk lineáris trendvonalat!
   5. Adjunk jelmagyarázatot is!

A megoldás megtekintéséhez kattintson ide!

8. Oldjuk meg a következő lineáris egyenletrendszert!
$6x_1 - 6x_2 - 2x_3 + 5x_4 = 20$
$-x_1 + 8x_2 + 5x_3 + 5x_4 = -15$
$2x_1 - 9x_2 - 9x_3 - 3x_4 = 22$
$4x_1 - x_2 + 3x_3 + 10x_4 = 6$
Javasolt megoldási lépések:

   1. Az aktuális könyvtárat irányítsuk a c:\munka könyvtárra!
   2. Hozzuk itt létre a 4x4 méretű A.dat együtthatómátrix szöveges adatfájlt és olvassuk be!
   3. Hozzuk itt létre a négyelemű b.dat oszlopvektor szöveges adatfájlt és olvassuk be!
   4. Számítsuk ki az A mátrix inverzét (az i_A változóba)! Jelenítsük meg tört alakban is!
   5. Határozzuk meg az x megoldásvektort!
   6. Ezt az eredeti egyenletrendszerbe helyettesítve végezzünk ellenőrzést! (ell=Ax és elteres=ell-b.)

A megoldás megtekintéséhez kattintson ide!

9. Az utolsó leckezáró 1–4. feladat számolási részét oldjuk meg a p = [1, -3, 2, 0, -0.07] polinomra alkalmazott

   1. roots() – polinom gyökkeresése;
   2. polyval() – helyettesítési érték;
   3. polyder() – derivált polinom együtthatóinak előállítása;
   4. sign() – előjel, a második derivált előjelének vizsgálatához;
   5. polyint() – primitív függvény előállítása a Newton-Leibniz szabály alkalmazásához;
   6. sort() – értékek növekvő sorrendbe rendezése

műveletek felhasználásával!
A megoldás megtekintéséhez kattintson ide!

10. Egy valós, szimmetrikus, nem szinguláris mátrix minden sajátértéke valós. Az

```
A = rand(5); A = 0.5*(A + A') + eye(5)
```

módon előállított mátrix is ilyen. A poly(A) parancs az A mátrix karakterisztikus polinomját állítja elő. Írassuk ki csökkenő sorrendben a sajátértékeket sorvektorként
a./ sajátérték-feladat megoldásával!
b./ karakterisztikus polinom gyökeiként!
A megoldás megtekintéséhez kattintson ide!

III. MODUL

Matematikai számítások - Excel

12. LECKE

Pontosság, nevesítések
Blokkműveletek és -függvények

3. Matematikai számítások Excellel

3.1. Pontosság, nevesítések, blokkműveletek és -függvények

3.1.1. Pontosság

Amikor az Excel segítségével matematikai, műszaki, gazdasági feladatokat oldunk meg, nem árt azt tudnunk, hogy minden numerikus értéket az ún. double lebegőpontos formában (ld. IEEE-754 szabvány) tárol, és ezekkel számol. Ennek a számábrázolási formának a pontossága nem haladja meg a 16 decimális számjegyet. Tehát ha egy értéket olyan cellaformátumban akarunk megjeleníteni, amely 16-nál több értékes jegy kiírását kéri, akkor az utolsó jegyek teljesen használhatatlanok lesznek. Ez a pontosság, illetve még inkább pontatlanság olyan eredményekhez is vezethet, amelyek megjelenítése zavarokat okozhat. Tegyük fel, hogy egy általános formátumú cellában a számítási eredményünk elméletileg 0 lenne. Ehelyett ott a számítási pontatlanságok miatt például a 2.62E-16 kijelzést látjuk. Ekkor érdemes a következő egyéni és egyben feltételes cellaformázást alkalmazni, amely a 0-hoz nagyon közeli értékek helyett magát a 0-t jeleníti meg:

[<-1E-14]-Normál;[>1E-14]Normál;0

Gépi epszilonnak nevezik azt a double típusban tárolható legkisebb pozitív számot, amelyet 1-hez adva az eredmény értéke még 1-nél nagyobbnak adódik. Mivel a 64 biten tárolt double típus 52 bitet használ a bináris törtrész megjelenítésére, ezért a gépi epszilon értéke:

$$2^{-52} = 2.2204460492503125 * 10^{-16}$$

Az ábrán azt mutatjuk be, hogy az Excel milyen pontosan számol, illetve a decimális kijelzési forma előállításakor mennyi jegyig korrekt. A **B** oszlopban 1-ből indulva folyamatos felezgetéssel jutunk a megfelelő hatványig. Az értékek megjelenítését tudományos formában kértük 16 tizedes jeggyel. Az 56-os sorban közölt érték az utolsó és még pontosan közölt 52-dik sorbeli érték sorozatos 2-vel osztásakor kézzel kiszámolt és szövegként visszaírt érték!

	A	B	C	D	E
1	**n**	**2^n**	**1+2^n**	**1 = (1+2^n)**	
2	0	1,0000000000000000E+00	2,0000000000000000E+00	HAMIS	
3	-1	5,0000000000000000E-01	1,5000000000000000E+00	HAMIS	
4	-2	2,5000000000000000E-01	1,2500000000000000E+00	HAMIS	
5	-3	1,2500000000000000E-01	1,1250000000000000E+00	HAMIS	
6	-4	6,2500000000000000E-02	1,0625000000000000E+00	HAMIS	
48	-46	1,4210854715202000E-14	1,0000000000000100E+00	HAMIS	
49	-47	7,1054273576010000E-15	1,0000000000000100E+00	HAMIS	
50	-48	3,5527136788005000E-15	1,0000000000000000E+00	IGAZ	
51	-49	1,7763568394002500E-15	1,0000000000000000E+00	IGAZ	
52	-50	8,8817841970012500E-16	1,0000000000000000E+00	IGAZ	
53	-51	4,4408920985006300E-16	1,0000000000000000E+00	IGAZ	
54	-52	2,2204460492503100E-16	1,0000000000000000E+00	IGAZ	
55					
56		2.2204460492503125E-16	gépi epszilon		
57					

3.1. ábra: Gépi epszilon pontatlan kijelzése az Excelben

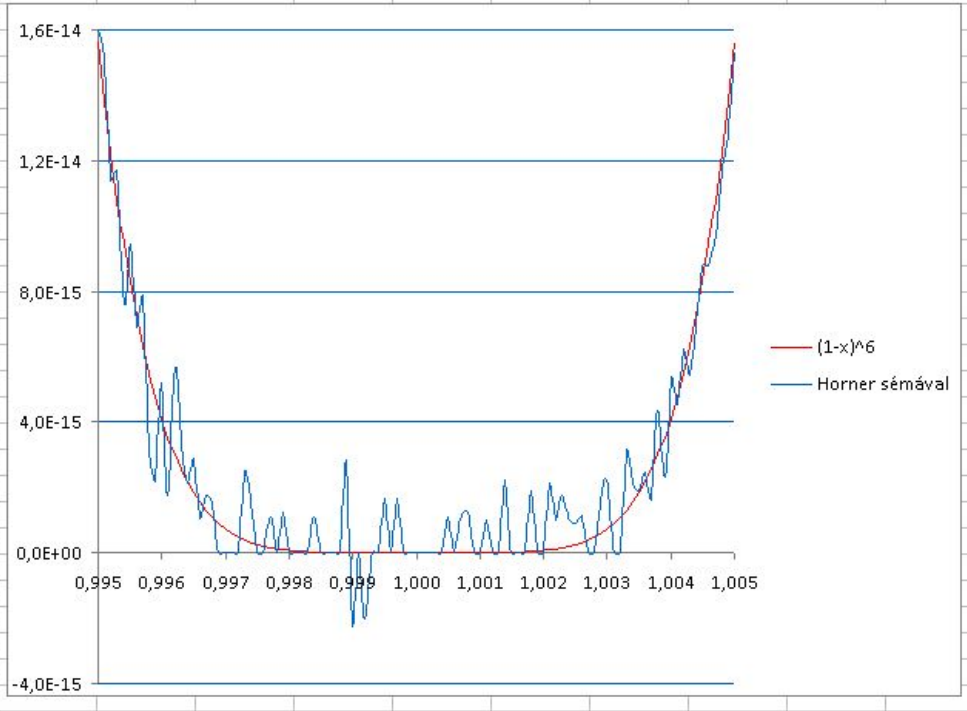
Látjuk, hogy az Excel csak a 14 tizedes jegyig (15 értékes jegy) jelzi ki pontosan az értéket, és a további jegyek helyett 0-t jelez ki, és ezekkel már nem számol.

A számítási lépéseink közben a pontatlan értékeken végzett műveletek tovább növelik az eredmény pontatlanságát (relatív hibáját). Nem mindegy, hogy egy értéket a matematikailag azonos eredményt szolgáltató lépéssorok közül melyikkel számítunk ki. A következő példánk is ezt illusztrálja:

$$(1-x)^6 = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1 = (((((x-6)x+15)x-20)x+15)x-6)x+1$$

Az első és az utolsó ún. Horner-zárójelezéses alak segítségével kiszámíttattuk a hatodfokú függvény értékeit a [0,995; 1,005] intervallumban és vonalas grafikonon ábrázoltuk.

	A	B	C
1	x	(1-x)^6	Horner sémával
2	0,9950	1,5625E-14	1,5987E-14
3	0,9951	1,3841E-14	1,5210E-14
4	0,9952	1,2231E-14	1,1435E-14
5	0,9953	1,0779E-14	1,1657E-14
6	0,9954	9,4743E-15	7,6605E-15
7	0,9955	8,3038E-15	9,4369E-15
8	0,9956	7,2563E-15	6,8834E-15
9	0,9957	6,3214E-15	7,7716E-15
10	0,9958	5,4890E-15	3,1086E-15
11	0,9959	4,7501E-15	2,2204E-15
12	0,9960	4,0960E-15	5,2180E-15
13	0,9961	3,5187E-15	1,7764E-15
14	0,9962	3,0109E-15	5,6621E-15
15	0,9963	2,5657E-15	3,3307E-15
16	0,9964	2,1768E-15	2,1094E-15
17	0,9965	1,8383E-15	2,8866E-15
18	0,9966	1,5448E-15	1,1102E-15
19	0,9967	1,2915E-15	1,7764E-15
20	0,9968	1,0737E-15	1,5543E-15
21	0,9969	8,8750E-16	0,0000E+00
22	0,9970	7,2900E-16	0,0000E+00
23	0,9971	5,9482E-16	0,0000E+00
24	0,9972	4,8189E-16	0,0000E+00
25	0,9973	3,8742E-16	2,4425E-15
26	0,9974	3,0892E-16	1,6653E-15
27	0,9975	2,4414E-16	0,0000E+00
28	0,9976	1,9110E-16	0,0000E+00
29	0,9977	1,4804E-16	1,1102E-15
30	0,9978	1,1338E-16	0,0000E+00
31	0,9979	8,5766E-17	1,2212E-15
32	0,9980	6,4000E-17	0,0000E+00
33	0,9981	4,7046E-17	0,0000E+00
34	0,9982	3,4012E-17	0,0000E+00



3.2. ábra: Halmozódó számítási hibák (kiegyszerűsödés) hatása

3.1.2. Cellák, tartományok nevesítése

Már korábban foglalkoztunk a nevek használatával (ld. Excel alapok). Most ezt kiegészítjük néhány fontos információval.

A cellák és tartományok (blokkok) nevesítésének alapvető célja az, hogy a rögzített cellákra és tartományokra való abszolút hivatkozás egyszerűbb legyen.

Sokkal áttekinthetőbb képleteket írhatunk egy paraméteresen megadott függvény vizsgálatakor és ábrázoltatásakor, ha a paramétereket egy külön táblarészben nevesítve adjuk meg.

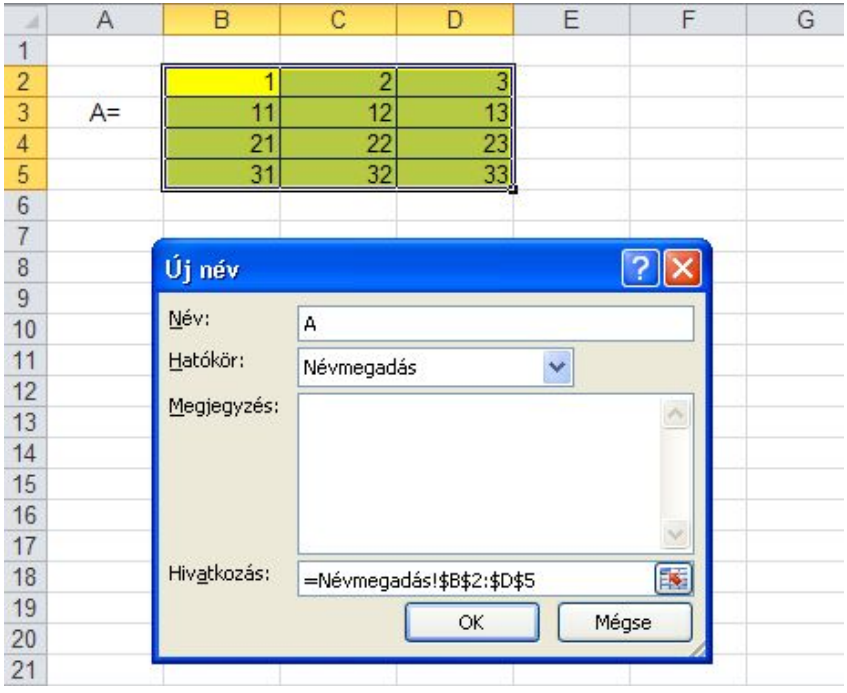
Hasonlóképpen a mátrixokat használó feladatok esetén a mátrixblokkot névvel hivatkozva egyrészt kevesebbet kell begépelni, másrészt áttekinthetőbb és beszédesebb hivatkozásokhoz jutunk. Például egy lineáris egyenletrendszer együtthatómátrixát **A**-nak, az inhomogén tagok oszlopvektorát **b**-nek, és az ismeretlenek oszlopvektorát **x**-nek nevezhetjük el.

A nevek megadásakor egyaránt használhatunk kis- és nagybetűket, nincs különbség közöttük.

Különbség van viszont a nevek használatakor az egyes Excel verziók között. Amíg az Excel 2003 csak a füzetszintű nevet ismeri, addig az Excel 2010-ben egy név hatóköre a teljes munkafüzet, vagy csak egy füzetlap lehet. Ez utóbbi esetben ugyanazt a nevet más-más füzetlapon más-más célra is használhatjuk.

A keveredések elkerülése végett az aktuális lapon szerepelő lapszintű név elsőbbséget élvez a füzetszintű névhez képest. A másik füzetlapon lévő azonos névre minősített névvel lehet hivatkozni. Például a **Munka1** lapon lévő lapszintű **z** névre az ugyancsak **z** nevet tartalmazó **Munka2** lapon **Munka1!z** minősített módon hivatkozhatunk.

A blokk nevének megadását célszerű a blokk kijelölése után a másodlagos egérgombbal kezdeményezhető helyzet-érzékeny menü segítségével kezdeményezni. Ugyanis ekkor lehetőséget kapunk a név lapszintű megadására. A hagyományos névmegadási mezőben viszont csak füzetszintű név adható meg! Példánkban a Névmegadás füzetlapon adunk nevet.



3.3. ábra: Lapszintű név adása

A nevek első jele betű, vagy \, vagy _ jel lehet. Egy név nem tartalmazhat space-t, műveleti jeleket és egyéb extra jeleket. Egy név nem ütközhet beépített névvel, vagy más objektum nevével. Így nem használhatjuk a Sor, Oszlop fogalmak első **S** ill **O** betűjét, mert ezek védettek. Az Excel 2010-ben az **R**, **C** nevek is védettek (**r**ow, **c**olumn). Érdekes dolog történik, ha egy **r**, **c** neveket tartalmazó Excel 2003 munkafüzetet megnyitunk Excel 2010-ben. Ekkor a védett nevek automatikusan kiegészülnek a _ vezető jellel, és **_r** ill **_c** néven használódnak tovább. Ez az átalakítás minden **r** és **c** nevet tartalmazó hivatkozásban is végrehajtódik.

3.1.3. Blokkműveletek (tartományműveletek)

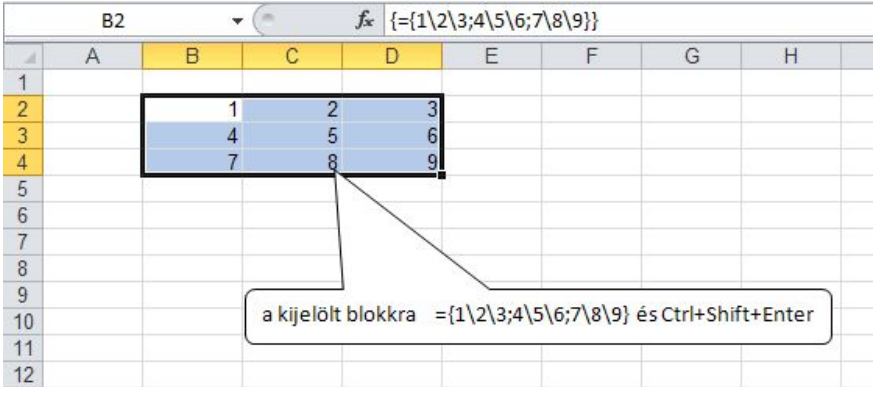
Az adatblokkokkal és blokkokon műveletek végezhetők el (matematikai, függvény). Ezek eredménye többnyire egy újabb tartományba, blokkba kerül. Amikor egy képlet argumentumai blokkok és az eredmény is egy blokk, akkor az eredményt egy ún. blokkművelettel helyezzük el a célterületen. Ennek lépései a következők:

1. Kijelöljük a megfelelő méretű célterületet (itt lehet a legtöbbet hibázni!);
2. Begépeljük a megfelelő képletet a forrástartományokra hivatkozva;
3. Ctrl + Shift + Enter billentyűkombinációval végrehajtatjuk a műveletet.

Az, hogy egy cellában blokkművelettel származtatott eredmény van, onnan látszik, hogy a szerkesztősávbeli képlet automatikusan { } zárójelpárba kerül. Ezek az automatikus kapcsos zárójelek nem szerkeszthetőek.

Milyen képleteket és műveleteket használva tölthetünk fel értékekkel egy új adatblokkot? A fontosabb szabályok a következők:

A./ Blokkot fel lehet tölteni tömbállandóval is.

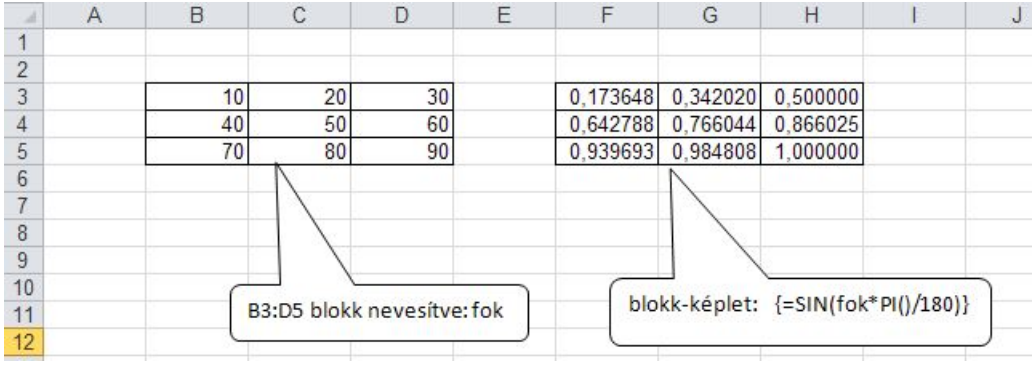


3.4. ábra: Tömbállandó

Tömbállandó: {} zárójelpárba helyezett értéklista. Az Excel 2003 esetén a sorok adatait pont választja el, Excel 2010 esetén a \jel, a sorokat pedig mindkét esetben a ; jel. (Megjegyzés: a tömbállandókat ritkán használjuk.)

B./ Ha egy cellába kerülő értéket legális kifejezéssel más cellák tartalmára hivatkozva származtatunk, akkor ugyanez a kifejezés cellák helyett blokkokra is alkalmazható, csak minden cellahivatkozás helyett azonos méretű blokkhivatkozás kell használnunk.

Legyenek például szögértékek fokokban megadva egy blokkban. Helyezzük el egy ugyanilyen méretű blokkban a szögértékek szinuszát!



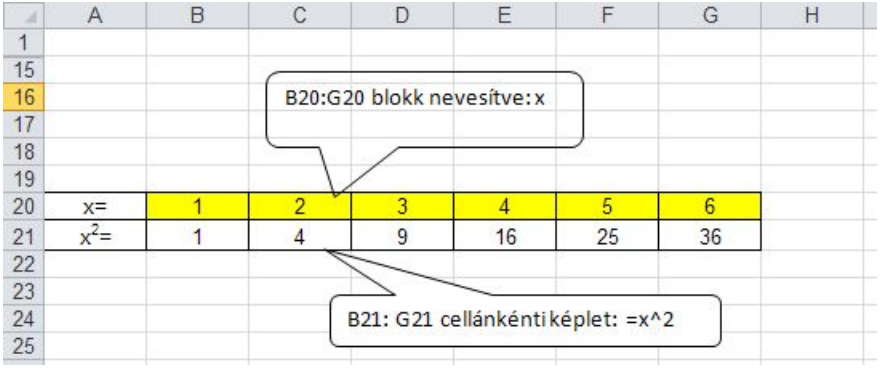
3.5. ábra: Blokk-képlet

Ez a példa azt is mutatja, hogy az egyváltozós függvényeket lehet tartományokra alkalmazni, és az eredmény ugyanilyen méretű tartományba kerül (blokkművelettel).

Hasonlóképpen igen egyszerűen számítható a két azonos méretű blokkon végzett aritmetikai műveletek (összeadás, kivonás, szorzás, osztás) eredménye, amely cellapáronként készül el.

C./ Ha a forrástartomány nevesített adatsor, vagy adatoszlop, és a leképezés ugyanilyen méretű parallel elhelyezkedésű tartományt (adatoszlop esetén a sorindexek azonosak, illetve adatsor esetén az oszlopindexek azonosak) eredményez, akkor nem szükséges a blokkművelet használata.

Legyen például a 20. sorban egy x-szel nevesített értéksorozat, és a 21. sorban szeretnénk ezen értékek négyzetét parallel társítva elhelyezni. Ekkor elegendő bármelyik eredménycellában az =x^2 képlethivatkozást megadni ahhoz, hogy minden érintett eredménycellába a vele egy oszlopban lévő x érték négyzete kerüljön.



	A	B	C	D	E	F	G	H
1								
15								
16								
17								
18								
19								
20	x=	1	2	3	4	5	6	
21	x^2=	1	4	9	16	25	36	
22								
23								
24								
25								

3.6. ábra: Nevesített blokkra hivatkozó cellaképlet

Ugyanezt az eredményt érjük el természetesen, ha mindezt blokkművelettel származtatjuk.

3.1.4. Blokkfüggvények

Az Excel néhány speciális és a mátrixszámításban alkalmazható blokkfüggvényt is használ, amelyek a lineáris algebrai feladatok megoldásakor nyújtanak segítséget. Ezek a következők:

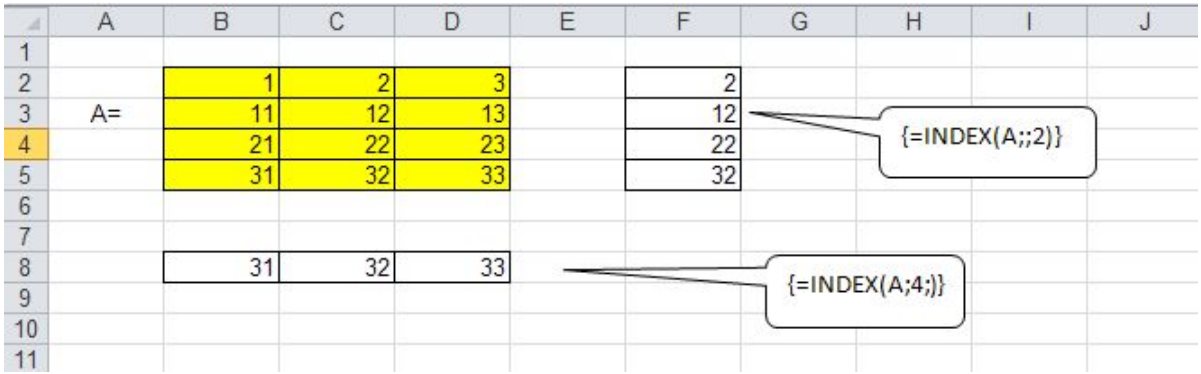
Blokkot (mátrixot, vektort) eredményeznek:

1. Index();
2. Transzponálás();
3. Mszorzat();
4. Inverz.mátrix().

Értéket eredményez: Mdeterm().

További, blokkot eredményező függvények (nem teljes felsorolás): Lin.ill(), Log.ill(), Növ(), Trend(), stb.

Az **Index()** blokkfüggvény a mátrix egy elemét a blokkon belüli relatív sor és oszlopindexen keresztül éri el. Azért tekinthetjük blokkfüggvénynek, mert ha csak a sorindexet adjuk meg, akkor sorvektort eredményez, ha pedig csak az oszlopindexet adjuk meg, akkor oszlopvektort eredményez.



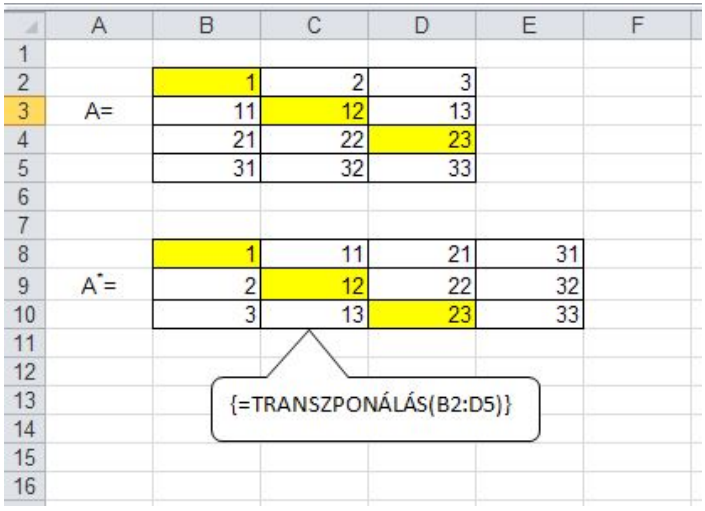
3.7. ábra: Az index() függvény használata

A **Transzponálás()** blokkfüggvény egy mátrix elemeit a főátlóra (azonos sor és oszlopindexű elemek) tükrözi. A matematikában a mátrix(vektor) neve után felső indexbe helyezett * jellel jelölik: $\mathbf{A}^*$. Sorvektor transzponáltja oszlopvektor és fordítva, oszlopvektor transzponáltja sorvektor.

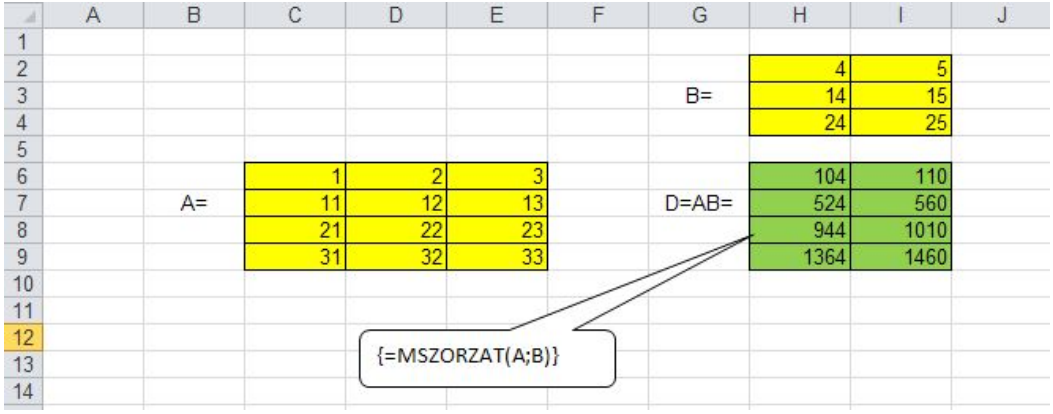
A lenti 3.8. ábra egy nem nevesített blokk transzponálását illusztrálja. Sorvektor transzponáltja oszlopvektor lesz, és egy oszlopvektor transzponáltja sorvektor lesz.

Az **Mszorzat(tömb1; tömb2)** blokkfüggvény blokkművelettel előállítja két összeszorozható mátrix mátrixszorzatát. Két mátrixot akkor nevezünk összeszorozhatónak, ha az első tényező oszlopainak száma megegyezik a második tényező sorainak számával. Az eredménymátrix egy-egy eleme az első mátrix megfelelő

sorvektora és második mátrix megfelelő oszlopvektora elempárjainak szorzatösszegével (skaláris szorzat) számítódik ki.



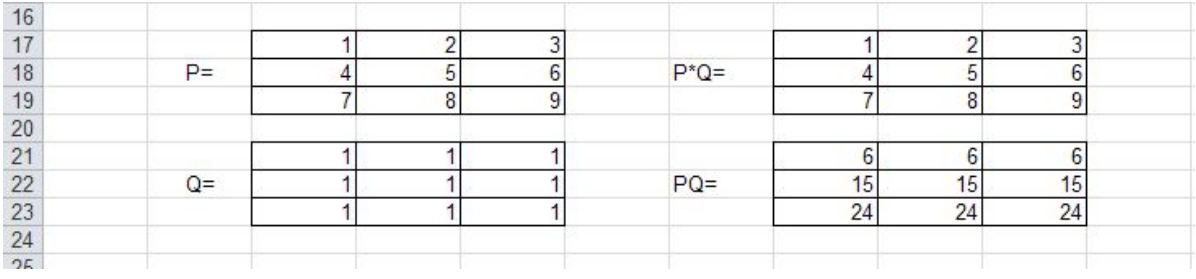
3.8. ábra: A transzponálás() függvény



3.9. ábra: Az Mszorzat() függvény

Az elmondottak szerint a **D** mátrix bal felső eleme az 1*4 + 2*14 + 3*24 művelettel számítódott ki.

A lineáris algebrában gyakorlatilag nem használják az azonos méretű P és Q blokkok P*Q elempáronkénti szorzatát, ami természetesen egészen mást eredményez, mint az Mszorzat(P; Q) függvénnyel kapott PQ mátrix, ha az utóbbi egyáltalán kiszámítható!

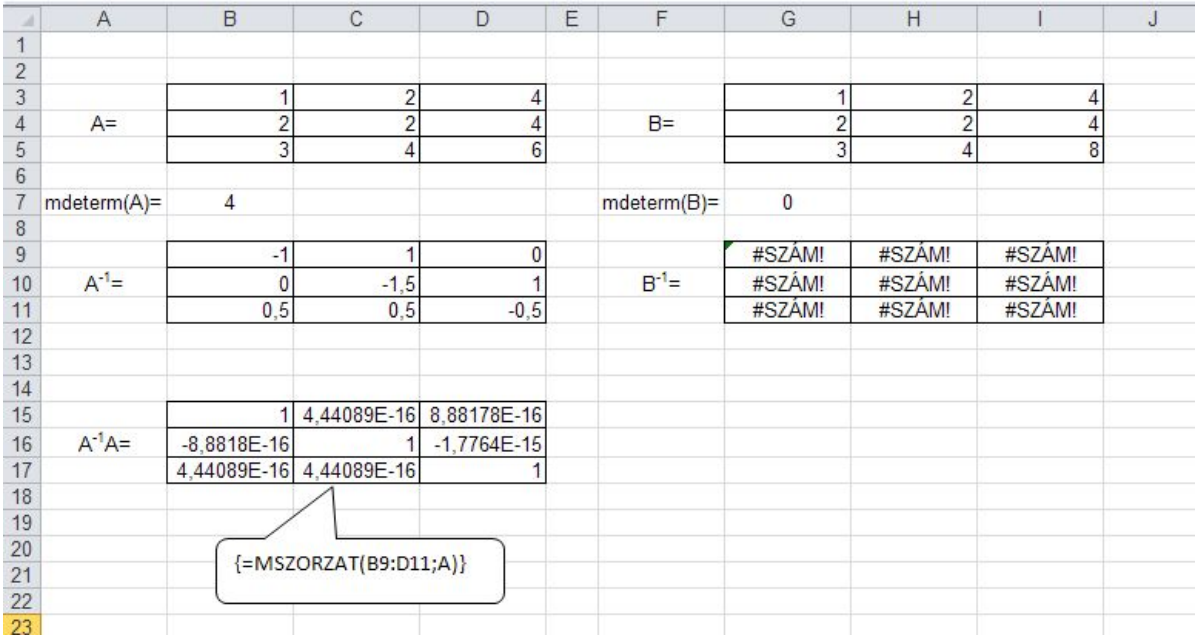


3.10. ábra: Elempáronkénti szorzat és az mszorzat() különbözősége

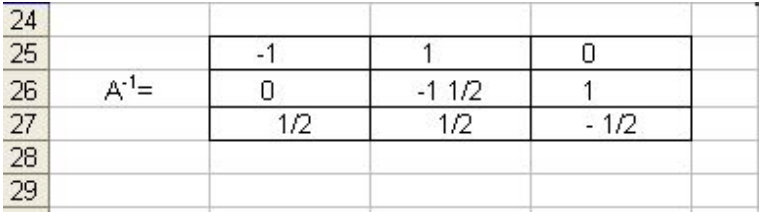
Az **Inverz.mátrix(tömb)** blokkfüggvény egy reguláris négyzetes méretű mátrix inverzét számítja ki, amely pontosan ugyanilyen méretű négyzetes mátrix lesz. A matematikában az **A** mátrix inverzét $\mathbf{A}^{-1}$-gyel jelölik. Ha egy négyzetes mátrixnak létezik az inverze, akkor a mátrix és az ő inverz mátrixának mátrixszorzata az egységmátrix lesz, amely a főátlójában csupa 1-est tartalmaz, azon kívül csak 0-kat.

Ahhoz, hogy egy négyzetes méretű mátrix invertálható legyen, az szükséges, hogy bármelyik oszlopvektora lineárisan független legyen a többitől. Egy mátrix oszlopvektorai lineárisan összefüggők, ha bármelyikük is kifejezhető a többi vektorból lineáris műveletekkel (nyújtás, összeadás). Hasonlót mondhatunk a sorvektorokra is. A négyzetes méretű mátrix oszlopvektorai lineáris függetlenségének ellenőrzésére az **Mdeterm(mátrix)** függvényt használhatjuk, amely egyetlen értéket eredményez. Ha ez 0, akkor a mátrix nem invertálható, és az inverz helyén hibajelzést kapunk. Az ábráról az is leolvasható, hogy a szorzással kapott egységmátrix 0 elemei nem látszanak annak. Itt lehetne alkalmazni a **pontosság** részben ismertetett egyéni cellaformátumot.

Jegyezzük meg, ha egy négyzetes mátrix elemei egész számok, akkor a kifejtési tétel miatt, amelyben az additív műveleteken kívül csak szorzás van, a determináns értéke egész szám lesz. Az inverz mátrix előállítása során (ld. Lineáris Algebra kurzusok) az **A** mátrix adjungáltját a mátrix determinánsával kell osztani. Így ekkor az inverz mátrix minden eleme racionális tört lesz, azaz maximum 3-jegyű determináns esetén az Excel tört cellaformátuma segítségével ezt a tört alakot is megjeleníthetjük.



3.11. ábra: Az inverz.mátrix() függvény



3.12. ábra: Egészelemű mátrix inverze törtekkel

Önellenőrzés

1. Állapítsuk meg milyen legkisebb n-re minősíti az Excel a 2^n és $2^n - 1$ értékét azonosnak!
   Útmutatás: készítsük el az n; 2^n; 2^n–1 fejléces táblázatot, ahol az n oszlopbeli értékek 1-esével növekednek, a 2^n oszlopbeli értékek pedig a felettük lévő érték duplája.
   A megoldás megtekintéséhez kattintson ide!
2. Illusztráljuk, hogy az $a_n = 2^n - 1$ sorozat az $a_0 = 0; a_n = 2a_{n-1} + 1$ rekurzív sorozattal is előállítható!
3. Az Excel 2010-ben a p, q, r, s egybetűs nevek közül melyek védettek?
   ☐ p ☐ q ☐ r ☐ s
4. Adjuk meg 2010-es Excelben tömbállandóval azt a 3x3-as mátrixot, amelynek elemei a sorindex és oszlopindex összege!
   A megoldás megtekintéséhez kattintson ide!
5. Egy füzetlapon a füzetszintű és a lapszintű x név közül melyik az érvényes?
   ☐ füzetszintű ☐ lapszintű
6. Döntsük el az alábbi mdeterm függvénnyel kapcsolatos kijelentésekről, hogy igazak vagy hamisak!
   a) Az mdeterm tetszőleges méretű blokk determinánsát számítja ki.
   ☐ igaz ☐ hamis
   b) Az mdeterm függvény eredménye egyetlen szám, vagy hibajelzés.
   ☐ hamis ☐ igaz
7. Mutassuk meg konkrét (3x3-as) példán, hogy a szorzat transzponáltja a transzponáltak fordított sorrendű mátrixszorzata!

8. Mutassuk meg konkrét (3x3-as) példán, hogy a szorzat inverze az inverzek fordított sorrendű mátrixszorzata!
9. Mutassuk meg konkrét (3x3-as) példán, hogy a mátrix szorzatának determinánsa a determinánsok szorzata!
10. Számítsuk ki A= {1\5\-2\2; 4\12\1\-1; -2\4\1\-4; 2\5\0\-1} blokk-konstanssal megadott mátrix inverzét! Adjuk meg az inverz második sorának első elemét valódi tört alakban.
    A megoldás megtekintéséhez kattintson ide!

13. LECKE

Blokkműveletek alkalmazása

3.2. Transzformációk a lineáris térben, speciális mátrixok

A megismert blokkfüggvényekkel igen sok matematikai és elsősorban lineáris algebrai feladat Excellel is kezelhető. A direkt számítási lépéseken alapuló megoldások a lineáris algebra fogalmaival írhatók le (lásd [12]). A precíz definíciók megismétlése helyett csak a szemléletes bemutatásra szorítkozunk.

Mindenek előtt azt kell felidéznünk, hogy egy lineáris térbeli **x** vektor T lineáris transzformációjának **y** eredményvektorát a transzformáció **T** mátrixának és az **x** vektornak ún. mátrixszorzatával számítjuk.

$$\mathbf{y} = T(\mathbf{x}) = \mathbf{T}\mathbf{x}$$

A T transzformáció mátrixának oszlopvektorait a lineáris tér $\mathbf{e}_1$, $\mathbf{e}_2$,..., $\mathbf{e}_n$ triviális bázisvektorainak (a 3-dimenziós vektorok vektoralgebrájában ezek az **i**, **j**, **k** egységvektorok) transzformáltjai adják. Azaz az első oszlopba $T(\mathbf{e}_1)$ koordinátái kerülnek, és így tovább.

3.2.1. Egységmátrix, vektor általánosított hossza

Az $E(\mathbf{x}) = \mathbf{x}$ egységtranszformáció (helybenhagyás) mátrixa az **egységmátrix**, amelynek oszlopaiban az $\mathbf{e}_1$, $\mathbf{e}_2$,..., $\mathbf{e}_n$ vektorok szerepelnek:

$$\mathbf{E} = \begin{bmatrix} 1 & 0 & . & 0 \\ 0 & 1 & . & 0 \\ 0 & 0 & . & 0 \\ 0 & 0 & . & 1 \end{bmatrix}$$

Az egységmátrix főátlójában minden elem 1, azon kívül pedig 0. Az egységmátrix vektorai páronként merőlegesek, azaz ortogonálisak (páronként skaláris szorzatuk 0) és a hosszuk, pontosabban mondva az euklideszi normájuk (Pitagorasz-tétel általánosítása) 1. Egy vektorrendszert **ortonormáltnak** mondunk, ha vektorai páronként ortogonálisak és mindegyikük normája 1.

Egy T transzformáció lineáris, ha a lineáris műveletekkel sorrendben felcserélhető, azaz teljesen mindegy, hogy egy vektort előbb megnyújtunk, és azután transzformáljuk, vagy előbb transzformáljuk, és ennek eredményét nyújtjuk meg. Hasonlót mondhatunk az összeadás és a transzformáció sorrendjéről is.

Egy euklideszi lineáris térbeli **x** oszlopvektor (olyan n dimenziós vektor, amelynek n darab koordinátája van) normájának négyzetét, azaz általánosított hosszának négyzetét többféleképp is kiszámíthatjuk. Nevesítsük x-szel a kérdéses oszlopvektort, ekkor a következő két lehetőség adódik az **x** vektor önmagával vett skaláris szorzatának kiszámítására:

=Négyzetösszeg(x)

=Mszorzat(Transzponálás(x); x)

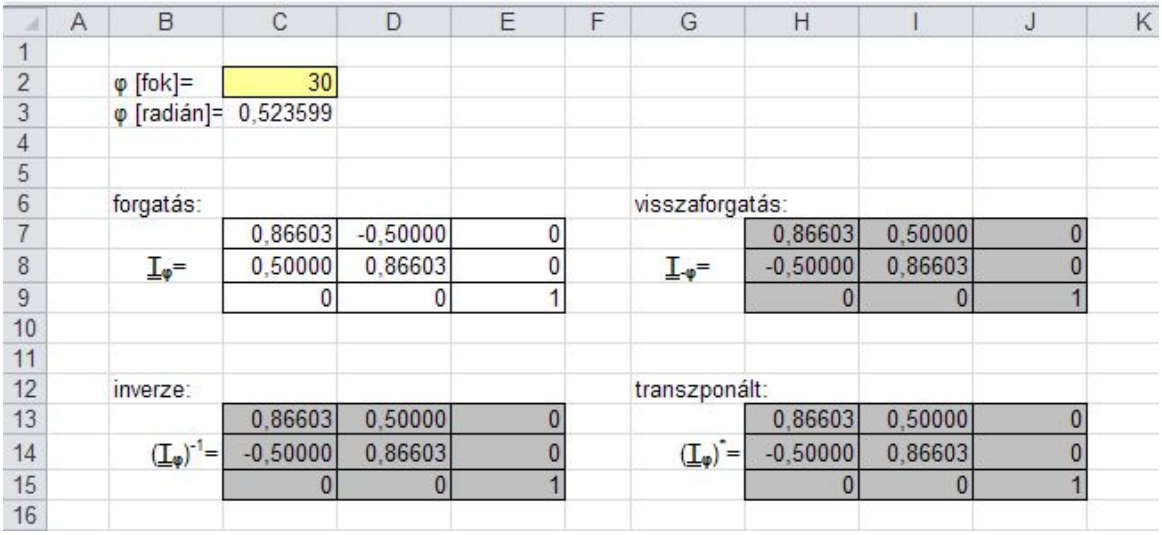
Az utóbbi metodika szerint egy sorvektort szorzunk a mátrixszorzás szabályai szerint egy oszlopvektorral, ami így egyetlen számot eredményez.

3.2.2. Koordinátatengely körüli forgatás

Példaként írjuk fel a 3-dimenziós térben a z tengely körüli φ szögű forgatás mátrixát! Az első oszlop kiszámításánál nem kell mást tenni, mint az egységnyi hosszú $\mathbf{e}_1$ vektor elforgatottjának új koordinátáit megállapítani, amelyek az x és y tengelyre való vetületek segítségével könnyen megkaphatók. Hasonlóan járunk el a másik két bázisvektornál is:

$$T(e_1) = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} \quad T(e_2) = \begin{bmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{bmatrix} \quad T(e_3) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Látható (3.13. ábra), hogy a $\mathbf{T}_\varphi$ forgatómátrix transzponáltja és az inverze is megegyezik, sőt ezek a -φ szögű forgatás mátrixával is egyeznek.



	A	B	C	D	E	F	G	H	I	J	K
1											
2		φ [fok]=	30								
3		φ [radián]=	0,523599								
4											
5											
6		forgatás:					visszaforgatás:				
7			0,86603	-0,50000	0			0,86603	0,50000	0	
8		$\underline{I}_{\varphi}$=	0,50000	0,86603	0		$\underline{I}_{-\varphi}$=	-0,50000	0,86603	0	
9			0	0	1			0	0	1	
10											
11											
12		inverze:					transzponált:				
13			0,86603	0,50000	0			0,86603	0,50000	0	
14		$(\underline{I}_{\varphi})^{-1}$=	-0,50000	0,86603	0		$(\underline{I}_{\varphi})^{*}$=	-0,50000	0,86603	0	
15			0	0	1			0	0	1	
16											

3.13. ábra: z tengely körüli forgatás

3.2.3. Tetszőleges tengely körüli forgatás

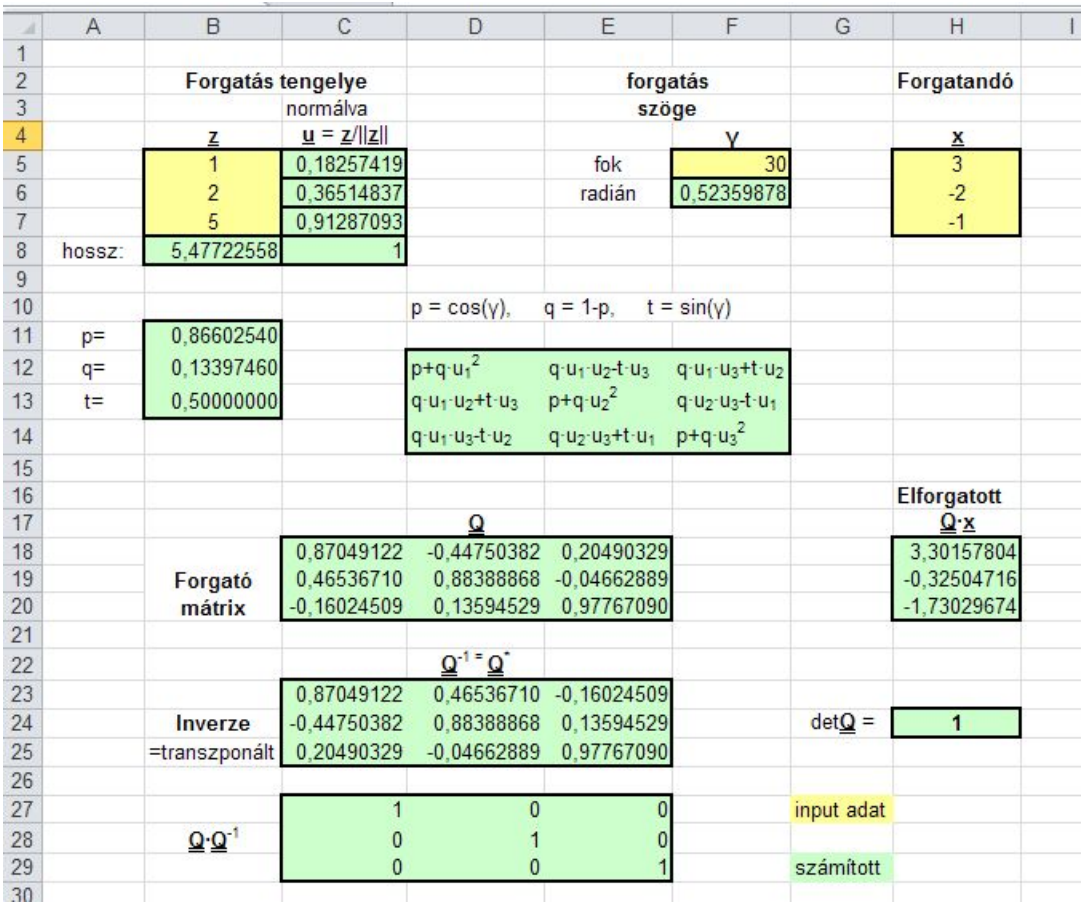
Egy **c** egységvektor körüli γ szögű forgatás **Q(c,** γ) mátrixát a Rodrigues-formulákkal adhatjuk meg. A forgatás mátrixát egymás utáni elemi forgatási transzformációk mátrixának szorzataként fogjuk majd származtatni (lásd hasonlósági transzformációk).

A Rodrigues-formulák szerint (lásd [13], 14.10-5 és 14.10-6 képletek):

$$Q(c,\gamma) = \begin{bmatrix} p+qc_1^2 & qc_1c_2 - rc_3 & qc_1c_3 + rc_2 \\ qc_1c_2 + rc_3 & p+qc_2^2 & qc_2c_3 - rc_1 \\ qc_1c_3 - rc_2 & qc_2c_3 + rc_1 & p+qc_3^2 \end{bmatrix} \quad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

ahol $p = \cos\gamma$, $q = 1 - p$ és $r = \sin\gamma$.

Egy forgatómátrix csak akkor lehet szimmetrikus, ha az $r = \sin\gamma$ érték 0, azaz pl. 180 fokos forgatásról van szó.



	A	B	C	D	E	F	G	H	I
1									
2		**Forgatás tengelye**			**forgatás**			**Forgatandó**	
3			normálva		**szöge**				
4		**z**	**u = z/||z||**			**γ**		**x**	
5		1	0,18257419		fok	30		3	
6		2	0,36514837		radián	0,52359878		-2	
7		5	0,91287093					-1	
8	hossz:	5,47722558	1						
9									
10				p = cos(γ), q = 1-p, t = sin(γ)					
11	p=	0,86602540							
12	q=	0,13397460		$p+q\cdot u_1^2$	$q\cdot u_1\cdot u_2-t\cdot u_3$	$q\cdot u_1\cdot u_3+t\cdot u_2$			
13	t=	0,50000000		$q\cdot u_1\cdot u_2+t\cdot u_3$	$p+q\cdot u_2^2$	$q\cdot u_2\cdot u_3-t\cdot u_1$			
14				$q\cdot u_1\cdot u_3-t\cdot u_2$	$q\cdot u_2\cdot u_3+t\cdot u_1$	$p+q\cdot u_3^2$			
15									
16								**Elforgatott**	
17				**Q**				**Q·x**	
18			0,87049122	-0,44750382	0,20490329			3,30157804	
19		**Forgató**	0,46536710	0,88388868	-0,04662889			-0,32504716	
20		**mátrix**	-0,16024509	0,13594529	0,97767090			-1,73029674	
21									
22				$\underline{Q}^{-1} = \underline{Q}^{*}$					
23			0,87049122	0,46536710	-0,16024509				
24		**Inverze**	-0,44750382	0,88388868	0,13594529		detQ =	1	
25		=transzponált	0,20490329	-0,04662889	0,97767090				
26									
27			1	0	0		input adat		
28		$\underline{Q}\cdot\underline{Q}^{-1}$	0	1	0				
29			0	0	1		számított		
30									

3.14. ábra: Forgatás adott tengely körül

A sárga hátterű cellák tartalma változtatható, a többi automatikusan számolódik.

A képletek beírása közben a következő lapszinten nevesített cellákra hivatkoztunk: p, q, t, u_1, u_2, u_3. Itt a Rodrigues-formulák r értékét t-vel nevesítettük (lásd védett név).

Általánosan az is igaz, hogy egy forgatómátrix determinánsa 1.

Ha adott egy általános forgató mátrix, akkor meghatározható a **c** egységvektor és a γ forgatási szög. Legyen például **Q** az általános forgató transzformációs mátrix, és az egyes elemeit pedig Q_{ij} jelölje. Ha a mátrix főátlójában elhelyezkedő elemek összegét (a mátrix **nyomát**, angol elnevezéssel trace) a szokásos módon Tr(**Q**) jelöli, akkor ha kiszámoljuk a Rodrigues-mátrix nyomát és végrehajtunk egyszerűsítéseket, a következőket kapjuk:

$$\mathrm{Tr}(\mathbf{Q}) = 1 + 2\cos\gamma\ .$$

Ebből pedig a Rodrigues-transzformáció forgatási szöge a következő:

$$\gamma = \arccos\left(\frac{\mathrm{Tr}(Q) - 1}{2}\right)$$

Ha már ismerjük a forgatási szöget, akkor az a **c** egységvektor, ami körül forgatunk, szintén meghatározható. Tekintsük ugyanis a főátlóra szimmetrikusan elhelyezkedő elemek különbségét és fejezzük ki a **c** egységvektor koordinátáit:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1}{2\sin\gamma} \begin{bmatrix} Q_{32} - Q_{23} \\ Q_{13} - Q_{31} \\ Q_{21} - Q_{12} \end{bmatrix}.$$

Fontos megjegyezni, hogy **c** csak akkor meghatározott, ha $0 < \gamma\ < \pi$.

3.2.4. Magasabb rendű forgatás

Ha egy **Q** forgatómátrix valamely n-edik hatványai egységmátrixok, akkor az 1-nél nagyobb, de egyébként minimális *n* értéket a forgatás rendjének nevezzük. Például a 120^0-os forgatás harmadrendű. A következő **Q**

forgatómátrix pedig ötödrendű (72^0-os) forgatást reprezentál:

$$Q = \frac{1}{2}\begin{bmatrix} \frac{1}{h} & -h & 1 \\ h & 1 & \frac{1}{h} \\ -1 & \frac{1}{h} & h \end{bmatrix} \quad \text{ahol} \quad h = \frac{\sqrt{5}+1}{2}$$

$h = \frac{1+\sqrt{5}}{2}$ = 1,61803399 értékkel

		1/h	-h	1		0,30901699	-0,80901699	0,50000000
	Q=0,5*	h	1	1/h	=	0,80901699	0,50000000	0,30901699
		-1	1/h	h		-0,50000000	0,30901699	0,80901699
						-0,80901699	-0,50000000	0,30901699
				Q^2	=	0,50000000	-0,30901699	0,80901699
						-0,30901699	0,80901699	0,50000000
						-0,80901699	0,50000000	-0,30901699
				Q^3	=	-0,50000000	-0,30901699	0,80901699
						0,30901699	0,80901699	0,50000000
						0,30901699	0,80901699	-0,50000000
				Q^4	=	-0,80901699	0,50000000	0,30901699
						0,50000000	0,30901699	0,80901699
						1	-2,7756E-17	0
				Q^5	=	1,6653E-16	1	-2,776E-17
						-1,1102E-16	-8,3267E-17	1

3.15. ábra: 72^0-os forgatás

Meghatározzuk a forgatás szögét és azt az egységvektort is, ami körül forgatunk. Ehhez az előző pont képleteit használjuk:

	A	B	C	D	E	F	G
35							
36		cos(γ)=	(tr(**Q**)-1)/2	=	0,30901699		
37							
38		γ=	1,256637	radián	sin(γ)=	0,951057	
39			72	fok	1/(2sin(γ))=	0,525731	
40							
41							
42		u_1		0,000000000			
43		u_2	=	0,525731112			
44		u_3		0,850650808			
45							
46		$\|\|\mathbf{u}\|\|^2$	=	1			
47							

3.16. ábra: Forgatás szöge a forgatómátrixból

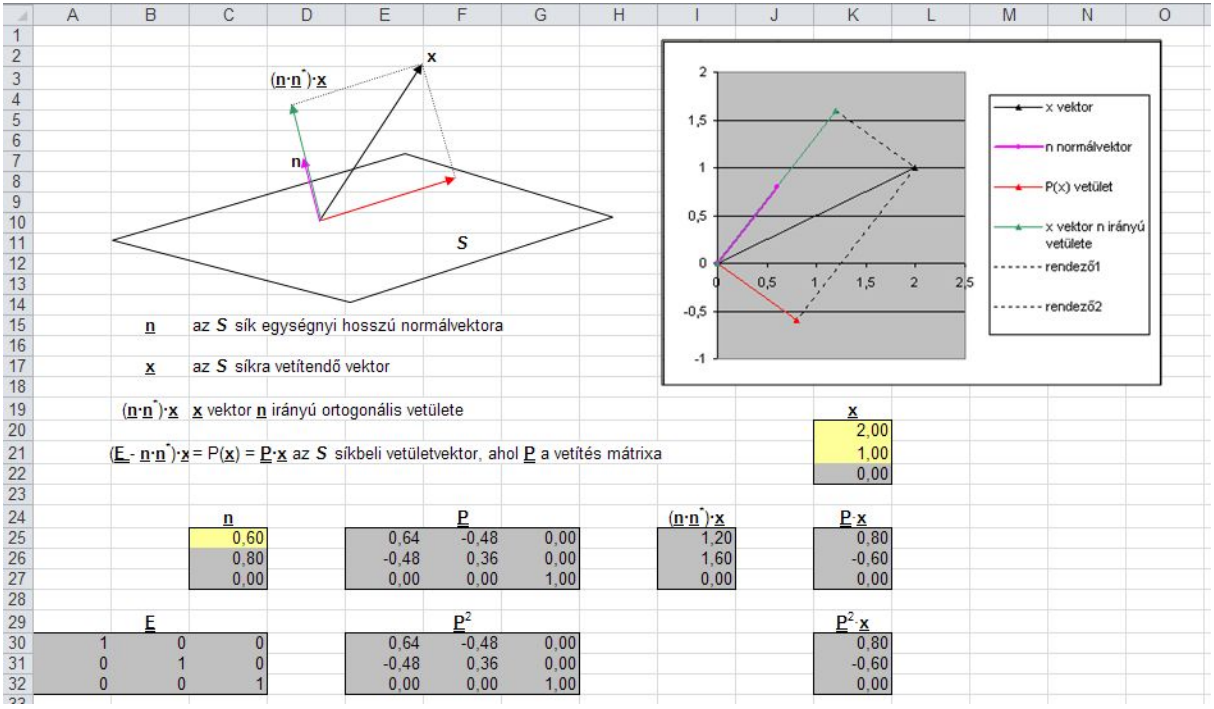
3.2.5. Síkra (altérre) vetítés

Lineáris algebrából tudjuk (lásd [12], 650–651. old.), hogy ha ismert az S sík $\mathbf{n}$ normálvektora, akkor egy tetszőleges $\mathbf{x}$ vektor S síkra való vetületét a

$$\mathbf{P} = (\mathbf{E} - \mathbf{n}\cdot\mathbf{n}^*)$$

projektormátrix segítségével szorzással $\mathbf{Px}$ módon számítjuk ki.

Egy $\mathbf{P}$ projektormátrix önmagával vett szorzata megegyezik az eredeti $\mathbf{P}$ mátrixszal, és a determinánsa 0, mivel a síkra való vetítés nem invertálható transzformáció.



n az **S** sík egységnyi hosszú normálvektora

x az **S** síkra vetítendő vektor

$(\underline{n}\cdot\underline{n}^{\circ})\cdot\underline{x}$ **x** vektor **n** irányú ortogonális vetülete

$(\underline{E} - \underline{n}\cdot\underline{n}^{\circ})\cdot\underline{x} = P(\underline{x}) = \underline{P}\cdot\underline{x}$ az **S** síkbeli vetületvektor, ahol **P** a vetítés mátrixa

x
2,00
1,00
0,00

n
0,60
0,80
0,00

P		
0,64	-0,48	0,00
-0,48	0,36	0,00
0,00	0,00	1,00

$(\underline{n}\cdot\underline{n}^{\circ})\cdot\underline{x}$
1,20
1,60
0,00

P·**x**
0,80
-0,60
0,00

E		
1	0	0
0	1	0
0	0	1

$\underline{P}^2$		
0,64	-0,48	0,00
-0,48	0,36	0,00
0,00	0,00	1,00

$\underline{P}^2\cdot\underline{x}$
0,80
-0,60
0,00

3.17. ábra: Vetítés síkra

3.2.6. Síkra tükrözés

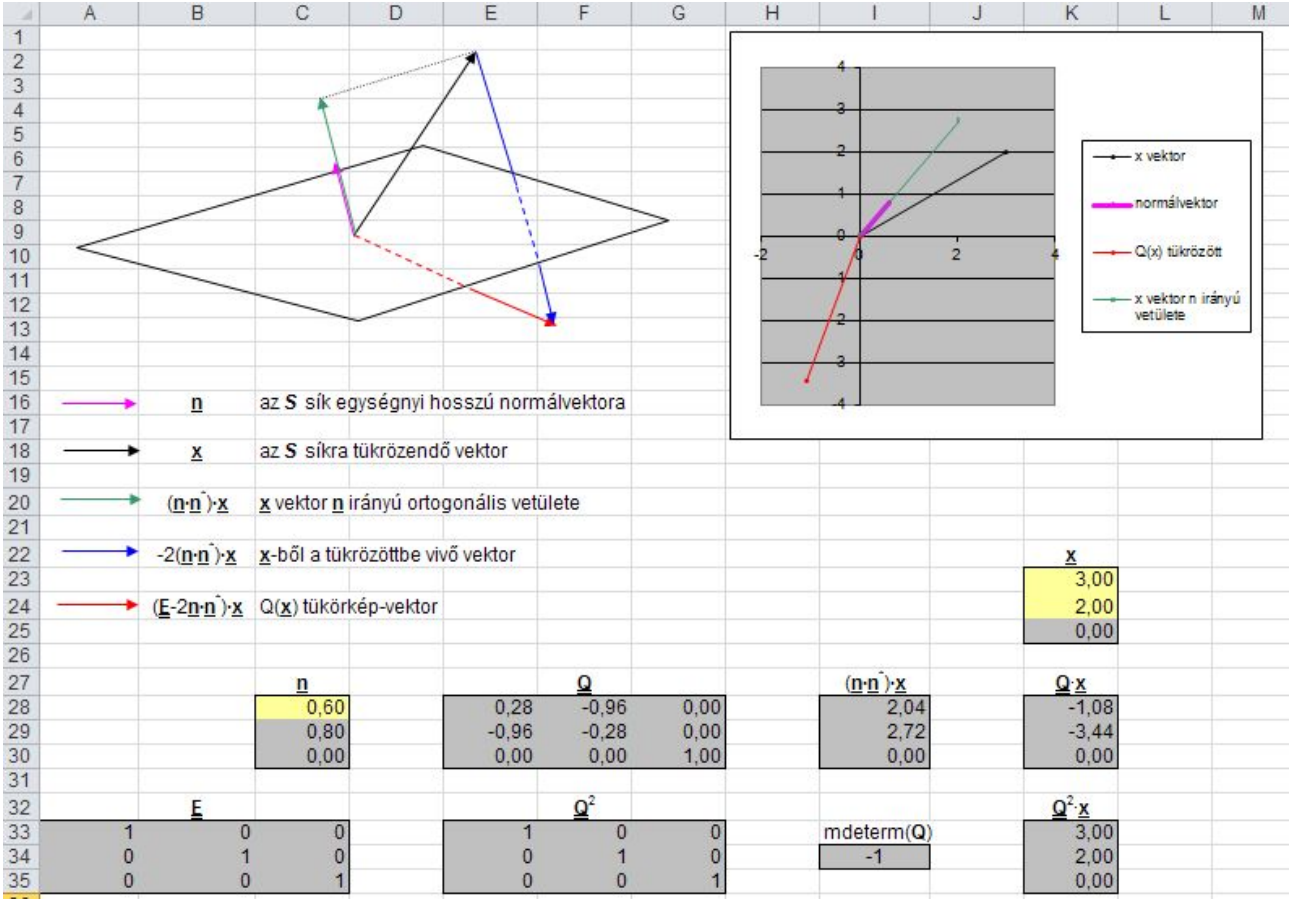
Lineáris algebrából azt is tudjuk ([12], 650–651. old.), hogy ha ismert az S sík $\mathbf{n}$ normálvektora, akkor egy tetszőleges $\mathbf{x}$ vektor S síkra való tükrözöttjét a

$$\mathbf{Q} = (\mathbf{E} - 2\mathbf{n} \cdot \mathbf{n}^*)$$

tükröző mátrix segítségével szorzással $\mathbf{Qx}$ módon számítjuk ki. A tükröző mátrixok ortonormáltak, azaz

$$\mathbf{Q}^*\mathbf{Q} = \mathbf{QQ}^* = \mathbf{E}$$

A tükröző mátrixok determinánsa -1.



3.18. ábra: Tükrözés síkra

3.2.7. Tükrözés altérre

Ha egy 4-dimenziós lineáris térben egy 3-dimenziós altérre tükrözünk, akkor meg kell adnunk azt a **z** irányt, amelyik ortogonális a 3-dimenziós altérre (azaz annak mindhárom bázisvektorára). Ennek normálásával (normával való osztás) kapjuk az **n** vektort, amelynek segítségével felírhatjuk a **Q** tükröző mátrixot

$$\mathbf{Q} = (\mathbf{E} - 2\mathbf{n}\cdot\mathbf{n}^*)$$

Mivel $(\mathbf{n}\cdot\mathbf{n}^*)^* = \mathbf{n}^{**}\cdot\mathbf{n}^* = \mathbf{n}\cdot\mathbf{n}^*$, ezért a tükröző mátrixok szimmetrikusak.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3		irány		normálva			tükröző mátrix							
4		z		n = z/||z||			Q = E - 2nn*				x		Qx	
5		3		0,54772		0,40000	-0,20000	-0,40000	0,80000		2		6,00000	
6		1		0,18257		-0,20000	0,93333	-0,13333	0,26667		-4		-2,66667	
7		2		0,36515		-0,40000	-0,13333	0,73333	0,53333		1		3,66667	
8		-4		-0,73030		0,80000	0,26667	0,53333	-0,06667		6		0,66667	
9														
10														
11		egységmátrix					tükröző mátrix négyzete							
12		E					Q²						Q(Qx)	
13	1	0	0	0		1	0	0	0				2	
14	0	1	0	0		0	1	0	0	detQ=	-1		-4	
15	0	0	1	0		0	0	1	0				1	
16	0	0	0	1		0	0	0	1				6	
17														

3.19. ábra: Tükrözés altérre

3.2.8. Hasonlósági transzformáció

Az *n* dimenziós lineáris tér egy új bázisának vektorait foglaljuk össze a **T** mátrixban (amely így nyilván reguláris mátrix, azaz a determinánsa nem 0). Az eredeti bázisban felírt **x** vektor új bázisbeli koordinátáit

$$\mathbf{x'} = \mathbf{T}^{-1}\mathbf{x}$$

módon kaphatjuk meg (azaz $\mathbf{x} = \mathbf{Tx'}$). Most vizsgáljuk meg azt az A transzformációt, amelyik az eredeti bázisban az $\mathbf{x}$ vektort $\mathbf{y}$-ba viszi:

$$\mathbf{y} = \mathbf{Ax}$$

Az $\mathbf{y}$ vektor új bázisbeli koordinátáit

$$\mathbf{y'} = \mathbf{T}^{-1}\mathbf{y}$$

szolgáltatja. Emiatt

$$\mathbf{y'} = \mathbf{T}^{-1}\mathbf{y} = \mathbf{T}^{-1}\mathbf{Ax} = \mathbf{T}^{-1}\mathbf{ATx'} = \mathbf{A'x'}$$

tehát ugyanazt az A transzformációt, amelynek az eredeti bázisbeli mátrixa $\mathbf{A}$, a $\{\mathbf{t}_1, \mathbf{t}_2, \ldots \mathbf{t}_n\}$ bázisban már a

$$\mathbf{A'} = \mathbf{T}^{-1}\mathbf{AT}$$

mátrix írja le. Emiatt a $\mathbf{T}$ reguláris mátrix segítségével felírt és az $\mathbf{A}$ mátrixon végrehajtott

$$\mathbf{T}^{-1}\mathbf{AT}$$

transzformációt **hasonlósági transzformáció**-nak nevezzük.

A hasonlósági transzformációk között fontos szerepet kapnak azok, amelyeknél a $\mathbf{T}$ mátrix speciális tulajdonságú (háromszög-alakú, ortogonális, stb.).

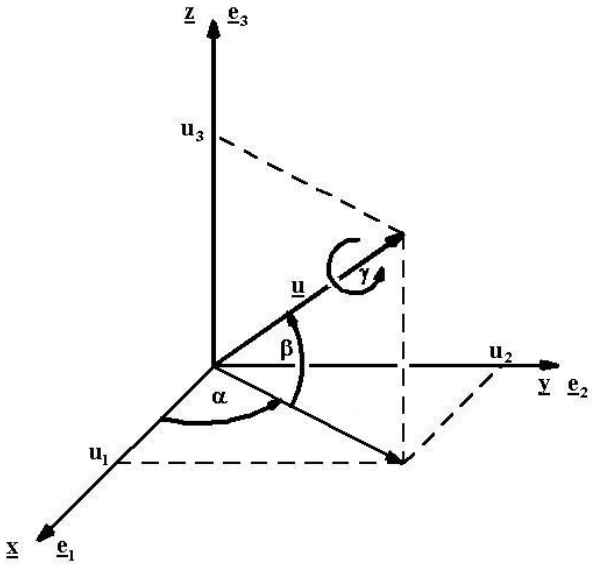
Amikor a **T** mátrix ortogonális, akkor az új bázis is ortonormált. Az ortogonális mátrixokat többnyire $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .. \mathbf{u}_n]$ -nel ill. $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, .. \mathbf{q}_n]$-nel jelölik, amelynek vektorai ortonormáltak.

Az alkalmazásokban az ortogonális hasonlósági transzformációk következő tulajdonságait használják ki:

Ortogonális mátrixok szorzata is ortogonális, azaz ha egymás után ortogonális mátrixok segítségével hajtunk végre hasonlósági transzformációt, akkor az egész transzformációs sorozat egyetlen ortogonális mátrixszal végrehajtott hasonlósági transzformációval helyettesíthető. Tehát

$$\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_m \text{ esetén } \mathbf{Q}_m{}^*\mathbf{Q}_{m-1}{}^* \ldots \mathbf{Q}_1{}^*\mathbf{A}\,\mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_m = \mathbf{Q}^*\mathbf{A}\mathbf{Q}$$

Speciális esetben, ha maga az **A** mátrix is ortogonális, akkor az ortogonális mátrixok segítségével felírt és az **A**-hoz hasonló **A'**$= \mathbf{Q}^*\mathbf{A}\mathbf{Q}$ mátrixok is ortogonálisok lesznek. Ezt használjuk ki a geometriai térben az **u** egységvektor körüli γ szögű forgatást leíró Rodrigues képletek (ld. speciális – ortogonális mátrixok) származtatásánál.



3.20. ábra: Rodrigues formulák származtatása

Az ábra és az **u** egységvektor tulajdonsága alapján

$$\sin\alpha = \frac{u_2}{\cos\beta}, \quad \cos\alpha = \frac{u_1}{\cos\beta}, \quad \sin\beta = u_3, \quad \cos\beta = \sqrt{u_1^2 + u_2^2}$$

Jelölje **Q(r,** φ) az **r** egységvektor vektor körüli φ szögű forgatás mátrixát az {$\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$} bázisban. Hajtsuk végre a bázisunk α szögű elforgatását a **z** tengely körül. A bázistranszformáció átmeneti mátrixa **Q**($\mathbf{e}_3$, α) lesz. Ebben az új bázisban a vizsgált forgatásunk mátrixa:

$$\mathbf{Q}^*(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{u}, \gamma)\mathbf{Q}(\mathbf{e}_3, \alpha)$$

Forgassuk most tovább a bázisunkat az új **y** tengely körül β szöggel. Az újabb bázis-transzformáció átmeneti mátrixa $\mathbf{Q}(\mathbf{e}_2, \beta)$ lesz. Ebben az újabb bázisban a vizsgált forgatásunk mátrixa:

$$\mathbf{Q}^*(\mathbf{e}_2, \beta)\mathbf{Q}^*(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{u}, \gamma)\mathbf{Q}(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{e}_2, \beta)$$

Viszont ebben a legújabb bázisban a forgatásunk nem lesz más, mint az **x** tengely körüli γ szögű elforgatás, azaz a mátrixa:

$$\mathbf{Q}(\mathbf{e}_1, \gamma) = \mathbf{Q}^*(\mathbf{e}_2, \beta)\mathbf{Q}^*(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{u}, \gamma)\mathbf{Q}(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{e}_2, \beta)$$

Ebből pedig – az ortogonális mátrixok tulajdonságai alapján – adódik:

$$\begin{aligned}\mathbf{Q}(\mathbf{u}, \gamma) &= \mathbf{Q}(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{e}_2, \beta)\mathbf{Q}(\mathbf{e}_1, \gamma)\mathbf{Q}^*(\mathbf{e}_2, \beta)\mathbf{Q}^*(\mathbf{e}_3, \alpha) = \\ &= \mathbf{Q}(\mathbf{e}_3, \alpha)\mathbf{Q}(\mathbf{e}_2, \beta)\mathbf{Q}(\mathbf{e}_1, \gamma)\mathbf{Q}(\mathbf{e}_2, -\beta)\mathbf{Q}(\mathbf{e}_3, -\alpha)\end{aligned}$$

Tehát a Rodrigues-képleteket öt elemi forgatási transzformációs mátrix szorzataként kapjuk.

Itt az elemi forgatási mátrixok:

$$Q(e_1,\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \quad Q(e_2,\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \quad Q(e_3,\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A negatív szögű forgatások mátrixa pedig ezekből triviálisan felírható. Ha az öt mátrixot összeszorozzuk és a behelyettesítéseket, majd az egyszerűsítéseket elvégezzük, akkor a már korábban közölt Rodrigues-képletekhez jutunk. A konkrét számítási eredményünk pedig a direkt képletekkel felírt korábbi eredmény lesz.

A **Q** forgatómátrixot előállító Excel blokk-kifejezés:

{=MSZORZAT(F86:H88;MSZORZAT(F92:H94;MSZORZAT(F98:H100; MSZORZAT(F104:H106;F110:H112))))}

Elemi forgatási mátrixok:

1. $Q(\underline{e}_3, \alpha)$ a z tengely körüli α szögű forgatás

$$\underline{Q}(\underline{e}_3, \alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0{,}44721360 & -0{,}89442719 & 0 \\ 0{,}89442719 & 0{,}44721360 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. $Q(\underline{e}_2, \beta)$ az y tengely körüli β szögű forgatás

$$\underline{Q}(\underline{e}_2, \beta) = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} = \begin{bmatrix} 0{,}40824829 & 0 & -0{,}91287093 \\ 0 & 1 & 0 \\ 0{,}91287093 & 0 & 0{,}40824829 \end{bmatrix}$$

3. $Q(\underline{e}_1, \gamma)$ az x tengely körüli γ szögű forgatás

$$\underline{Q}(\underline{e}_1, \gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0{,}86602540 & -0{,}50000000 \\ 0 & 0{,}50000000 & 0{,}86602540 \end{bmatrix}$$

4. $Q(\underline{e}_2, -\beta)$ az y tengely körüli -β szögű forgatás

$$\underline{Q}(\underline{e}_2, -\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} = \begin{bmatrix} 0{,}40824829 & 0 & 0{,}91287093 \\ 0 & 1 & 0 \\ -0{,}91287093 & 0 & 0{,}40824829 \end{bmatrix}$$

5. $Q(\underline{e}_3, \alpha)$ a z tengely körüli -α szögű forgatás

$$\underline{Q}(\underline{e}_3, -\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0{,}44721360 & 0{,}89442719 & 0 \\ -0{,}89442719 & 0{,}44721360 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A fenti 5 mátrix szorzataként kapjuk az $\underline{u}$ körüli γ szögű forgatás mátrixát:

$$\underline{Q}(\underline{u}, \gamma) = \underline{Q}(\underline{e}_3, \alpha)\underline{Q}(\underline{e}_2, \beta)\underline{Q}(\underline{e}_1, \gamma)\underline{Q}(\underline{e}_2, -\beta)\underline{Q}(\underline{e}_3, -\alpha) = \begin{bmatrix} 0{,}87049122 & -0{,}44750382 & 0{,}20490329 \\ 0{,}46536710 & 0{,}88388868 & -0{,}04662889 \\ -0{,}16024509 & 0{,}13594529 & 0{,}97767090 \end{bmatrix}$$

3.21. ábra: Rodrigues képletek ellenőrzése

Önellenőrzés

1. Táblázatosan adott a következő A mátrix:

- 1/5	- 2/5	- 4/5	2/5
- 2/5	13/15	- 4/15	2/15
- 4/5	- 4/15	7/15	4/15
2/5	2/15	4/15	13/15

Jelöljük meg a helyes választ az alábbi kérdéseknél:
Szimmetrikus?

☐ hamis ☐ igaz

Determináns?

☐ 1 ☐ -1 ☐ más

Négyzete?

☐ A ☐ E ☐ más

Tükröző mátrix?

☐ igen ☐ nem

2. Táblázatosan adott a következő B mátrix:

0	1	0
1	0	0
0	0	-1

Jelöljük meg a helyes választ az alábbi kérdéseknél:
Szimmetrikus?

☐ hamis ☐ igaz

Determináns?

☐ 1 ☐ -1 ☐ más

Négyzete?

☐ B ☐ E ☐ más

Forgató mátrix?

☐ igen ☐ nem

Forgatás rendje?

☐ első ☐ másod ☐ más

14. LECKE

Lineáris egyenletrendszerek

3.3. Lineáris egyenletrendszerek megoldása Excel segítségével

3.3.1. Lineáris egyenletrendszer megoldása inverz mátrix segítségével

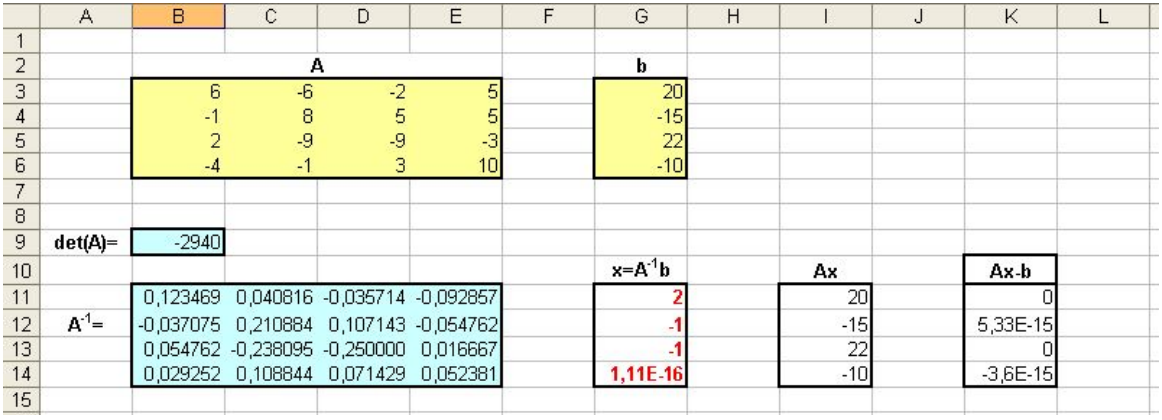
Az Excel beépített blokkfüggvényei segítségével az egyértelműen megoldható lineáris egyenletrendszerek egyszerűen kezelhetők. Nem kell mást tudnunk csak azt, hogy ha a lineáris egyenletrendszer **A** együtthatómátrixa invertálható, akkor a megoldás az

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

képlet segítségével számítható. Tekintsük a következő lineáris egyenletrendszert:

$$\begin{aligned} 6x_1 - 6x_2 - 2x_3 + 5x_4 &= 20 \\ -x_1 + 8x_2 + 5x_3 + 5x_4 &= -15 \\ 2x_1 - 9x_2 - 9x_3 - 3x_4 &= 22 \\ -4x_1 - x_2 + 3x_3 + 10x_4 &= -10 \end{aligned}$$

A megoldás és ellenőrzés lépéseit a 3.22. ábra illusztrálja.



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2				A			b					
3		6	-6	-2	5		20					
4		-1	8	5	5		-15					
5		2	-9	-9	-3		22					
6		-4	-1	3	10		-10					
7												
8												
9	det(A)=	-2940										
10							x=A^{-1}b		Ax		Ax-b	
11		0,123469	0,040816	-0,035714	-0,092857		2		20		0	
12	A^{-1}=	-0,037075	0,210884	0,107143	-0,054762		-1		-15		5,33E-15	
13		0,054762	-0,238095	-0,250000	0,016667		-1		22		0	
14		0,029252	0,108844	0,071429	0,052381		1,11E-16		-10		-3,6E-15	
15												

3.22. ábra: Lineáris egyenletrendszer megoldása inverz mátrix segítségével

Ha az **A** és **b** blokkokat nevesítettük, akkor az **x** vektor értéke az {=Mszorzat(Inverz.Mátrix(**A**); **b**)} blokkművelettel számítható.

Itt nem érdemes az inverz mátrixot törtalakban megjeleníteni, mert a determináns már négyjegyű szám, és ilyenkor a megjelenített törtalakok a törtek szintjén kerekített, közelítő értékek lesznek.

3.3.2. Összefüggő egyenletrendszer

Egy lineáris egyenletrendszer akkor összefüggő, ha bármelyik egyenlete a többi lineáris kombinációjaként előállítható, azaz a többi egyenlet valamilyen konstans szorosai összegeként is felírható. Mi most csak azzal az esettel foglalkozunk, amikor a független egyenletek száma csak eggyel kevesebb, mint az ismeretlenek *n* száma. Ezt az esetet az jellemzi, hogy az

1. Az **A** együtthatómátrix determinánsa 0;
2. Az **A** első oszlopvektorának elhagyásával és a **b** vektor hozzá vételével kapott mátrix determinánsa is 0;

3. Az utolsó egyenlet elhagyásával kapott egyenletrendszer egyenletei viszont már lineárisan függetlenek lesznek.

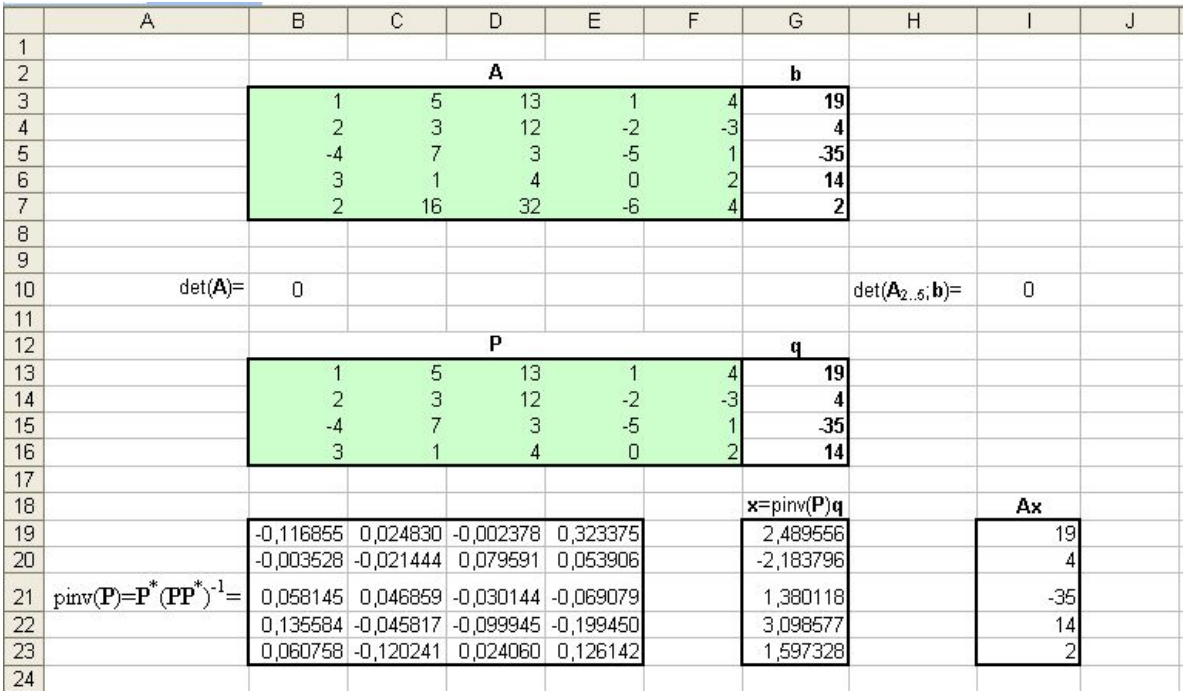
Ekkor az egyenletrendszernek végtelen sok megoldása van, ezek közül azt az egyet szokták kitüntetni, amelynek a normája minimális, azaz a megoldásvektor koordinátáinak négyzetösszege minimális. Ez a megoldás mátrixszorzással is számítható, de most a **b** vektort balról nem az **A** mátrix inverzével szorozzuk, ami nem is létezik, hanem az ún. Moore–Penrose-féle pszeudoinverzével. A pszeudoinverz függvény nincs az Excelbe építve.

Jelölje **P** az utolsó egyenlet elhagyásával kapott, $m = n - 1$ független egyenletből álló n ismeretlenes egyenletrendszer együtthatómátrixát, **q** a jobboldali és **b**-nél eggyel kevesebb elemű inhomogén vektort. Ennek a P mátrixnak a pszeudoinverze (részletesebben lásd [14], 267–272. old.):

$$\text{pinv}(\mathbf{P}) = \mathbf{P}^*(\mathbf{P}\mathbf{P}^*)^{-1}$$

Ennek mérete *n***m*, azaz $\mathbf{P}^*$ méretével egyezik meg. A pszeudoinverz a következő blokkművelettel számítható ki (3.23. ábra):

{=Mszorzat(Transzponálás(**P**); Inverz.Mátrix(Mszorzat(**P**; Transzponálás(**P**))))}



	A	B	C	D	E	F	G	H	I	J
1										
2				**A**			**b**			
3		1	5	13	1	4	**19**			
4		2	3	12	-2	-3	**4**			
5		-4	7	3	-5	1	**-35**			
6		3	1	4	0	2	**14**			
7		2	16	32	-6	4	**2**			
8										
9										
10	det(**A**)=	0						det($\mathbf{A}_{2..5}$;**b**)=	0	
11										
12				**P**			**q**			
13		1	5	13	1	4	**19**			
14		2	3	12	-2	-3	**4**			
15		-4	7	3	-5	1	**-35**			
16		3	1	4	0	2	**14**			
17										
18							**x**=pinv(**P**)**q**		**Ax**	
19		-0,116855	0,024830	-0,002378	0,323375		2,489556		19	
20		-0,003528	-0,021444	0,079591	0,053906		-2,183796		4	
21	pinv(**P**)=$\mathbf{P}^*(\mathbf{PP}^*)^{-1}$=	0,058145	0,046859	-0,030144	-0,069079		1,380118		-35	
22		0,135584	-0,045817	-0,099945	-0,199450		3,098577		14	
23		0,060758	-0,120241	0,024060	0,126142		1,597328		2	
24										

3.23. ábra: Összefüggő egyenletrendszer

3.3.3. Ellentmondó egyenletrendszer

Ellentmondó egyenletrendszerrel akkor találkozunk, ha az **A** mátrix rangja kisebb az ismeretlenek számánál, de a **b**-vel kibővített mátrix rangja az **A** mátrix rangjánál nagyobb.

Ebben az esetben az egyenletrendszernek nincs megoldása.

A rangokat a kibővített [**A** **b**] mátrix négyzetes minormátrixai (részmátrixai) determinánsainak megvizsgálásával állapíthatjuk meg.

	A	B	C	D	E	F	G	H
1								
2				**A**			**b**	
3		1	5	13	1	4	**19**	
4		2	3	12	-2	-3	**4**	
5		-4	7	3	-5	1	**-35**	
6		3	1	4	0	2	**14**	
7		2	16	32	-6	4	**5**	
8								
9								
10	Mdeterm(**A**)=	0						
11								
12	Mdeterm($\mathbf{A}_{2..5}$;**b**)=	-1974						
13								

3.24. ábra: Ellentmondó egyenletrendszer

Ellentmondó egyenletrendszer például akkor adódhat, ha a mérési adatokon alapuló egyenletrendszerbe pontatlan mérési eredmények kerülnek (térképészeti háromszögelési mérések stb.). Ilyenkor a hibakiegyenlítés módszerei jöhetnek szóba (lásd később a Solveres elemzést).

3.3.4. Túlhatározott egyenletrendszer

Ha az együtthatómátrix oszlopvektorai függetlenek, de az egyenletek száma az ismeretlenek számánál több, akkor túlhatározott egyenletrendszerrel van dolgunk. Ekkor az **A** mátrixnak több sora van, mint oszlopa. Ilyenkor az **Ax** - **b** eltérést minimalizáljuk normában. Ez is az **A** mátrix Moore-Penrose féle pszeudoinverzének a fogalmához vezet, csak ekkor ezt a

$$\text{pinv}(\mathbf{A}) = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*$$

módon számítjuk. Sajnos a pinv() függvény nincs az Excelbe beépítve, így csak a képlet direkt beírásával számíthatjuk ki (3.25. ábra).

	A	B	C	D	E	F	G	H
1								
2			**A**		**b**			
3		1	5	13	**6**			
4		2	3	12	**5**			
5		-4	7	3	**3**			
6		3	1	4	**-1**			
7		-2	4	1	**2**			
8								
9								
10		-0,061826	-0,069017	-0,033149	0,411821	0,084101		
11	pinv(**A**) = $(\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*$ =	-0,043377	-0,080691	0,093232	0,280244	0,131522		
12		0,061365	0,073468	-0,023481	-0,138187	-0,056169		
13								
14				**x=pinv(A)b**	**Ax**		**Ax-b**	
15				-1,0591073	5,916963		-0,083037	
16				-0,4012212	4,969361		-0,030639	
17				0,6909366	3,500691		0,500691	
18					-0,814797		0,185203	
19					1,204266		-0,795734	
20								

3.25. ábra: Túlhatározott egyenletrendszer

A pszeudoinverz mérete **A*** méretével egyezik meg és az

{=Mszorzat(Inverz.Mátrix(Mszorzat(Transzponálás(**A**);**A**));Transzponálás(**A**))}

blokkművelettel számíthatjuk.

3.4. Lineáris egyenletrendszerek megoldása Solverrel

3.4.1. A Solver használata

A Solver eszköz az Excel egyik legfontosabb bővítménye és része. Ahhoz hogy használni lehessen egy olyan Excelben, amely eddig még nem használta ezt a bővítményt, aktiválni kell. Egy bővítmény (Solver, Analysis ToolPak, stb.) aktiválásának lépései az Excel 2010 helpje szerint:

Excel-bővítmény aktiválása:

1. Kattintson a **Fájl** lap **Beállítások** parancsára, majd a **Bővítmények** kategóriára.
2. Válassza a **Kezelés** lista **Excel-bővítmények** elemét, és kattintson az **Ugrás** gombra. Ekkor megjelenik a **Bővítmények** párbeszédpanel.
3. Jelölje be a **Létező bővítmények** mezőben az aktiválandó bővítmény melletti jelölőnégyzetet, és kattintson az **OK** gombra.

Eztán az aktivált bővítmény a megfelelő menüben már látható lesz.

A Solver bővítménynek több verziója van, de bármelyik verzióban a használata hasonlóképpen történik. Mi most a 2010-es verzió használatát fogjuk csak ismertetni. Ha valaki mélyebben akar a Solver matematikai hátterével foglalkozni, akkor látogassa meg a `http://www.solver.com` weboldalt.

A Solver használatát konkrét problémák megoldásán keresztül sajátíthatjuk el.

3.4.2. Lineáris egyenletrendszer megoldása

Bár az inverz mátrix felhasználásával egy lineáris egyenletrendszer egyszerűen megoldható, nem árt, ha a Solver segítségével történő megoldást is megismerjük.

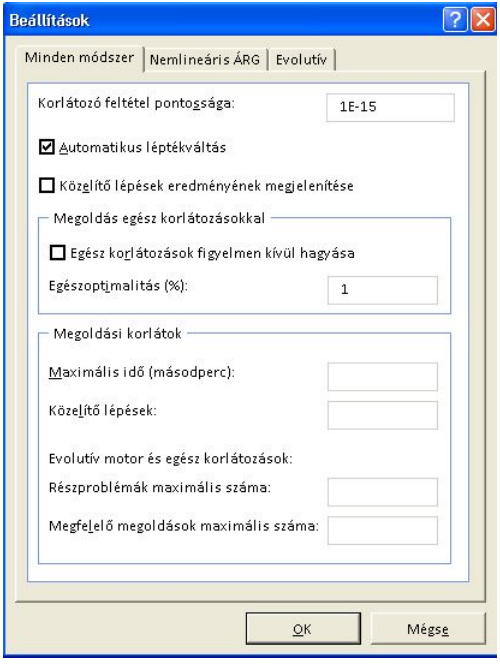
Ugyanazt az egyenletrendszert fogjuk megoldani, amit korábban az inverz mátrix segítségével oldottunk meg.

Az Adatok főmenüből tudjuk indítani a Solvert. (A 2003-as Excelben az Eszközök főmenü alatt találjuk).

A **Célérték** egy olyan cella lehet, amelyik a **Változó cellák** tartalmára közvetlenül vagy közvetve hivatkozik. A **Változó cellák** nem tartalmazhatnak hivatkozást, csak konkrét értékeket. A Solver a célcella értékét a kért **Cél** (**Min**, **Max**, megadott **Érték**) szerint próbálja a változó cellák tartalmának módosítása révén beállítani.

A feladat korlátozásokat is tartalmazhat, amelyeket a Hozzáadás nyomógomb megnyomása után adhatunk meg.

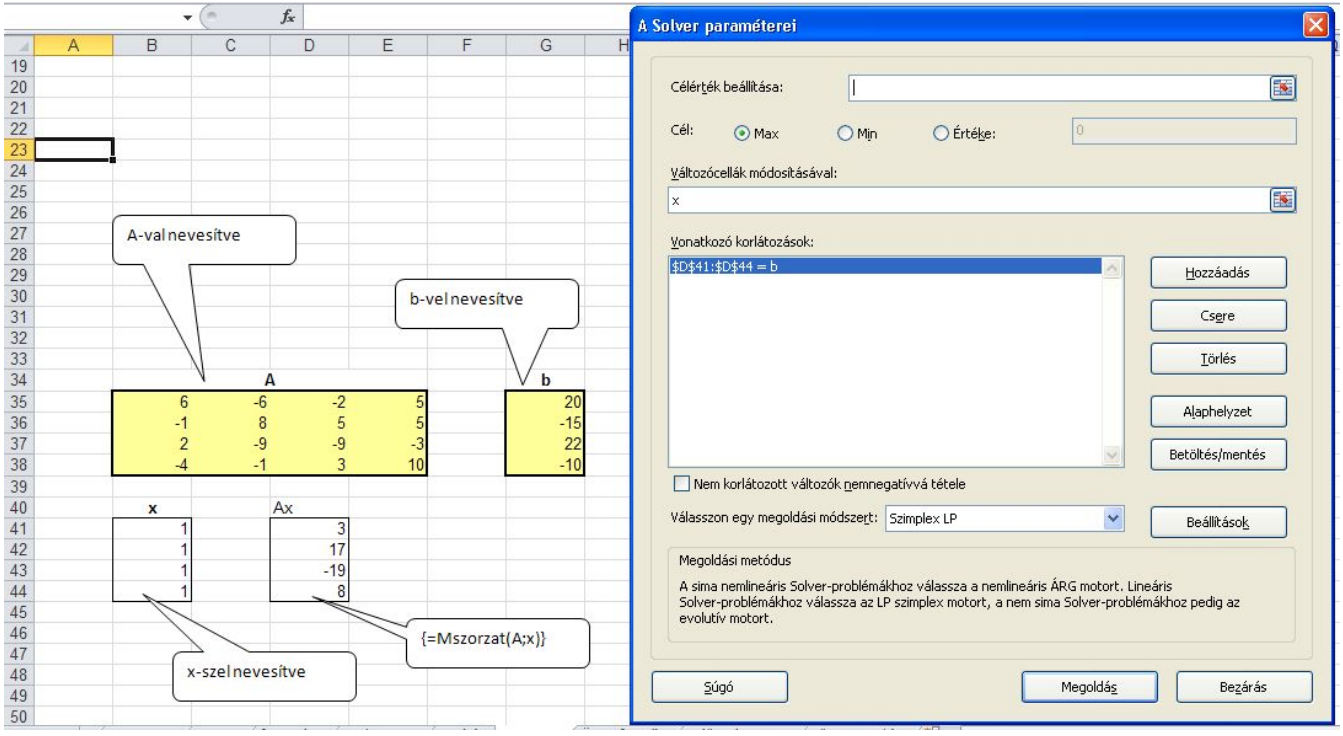
A **Nem korlátozott változók nem negatívvá tétele** jelölőnégyzetet csak akkor kell bejelölni, ha a változó cellák értékei a feladat jellege miatt nem lehet negatív. Például egy termelési feladatnál a gyártandó termékmennyiségek értelemszerűen nem lehetnek negatívok.



3.26. ábra: Solver pontosságának beállítása

A megoldási módszerek közül a **Nemlineáris ÁRG** a legtöbb problémához használható, a **Szimplex LP** lineáris feladatokhoz alkalmazható, az **Evolutív** pedig genetikus algoritmus segítségével keres megoldást.

A **Beállítások** nyomógombbal olyan párbeszédablakba lépünk (3.26. ábra), amelyen a megoldó algoritmusok működését szabályozhatjuk. Itt most csak a korlátozó feltétel pontossága cellára hívjuk fel a figyelmet. Célszerűen érdemes a pontossági értéket kicsire választani. Mostani feladatunkban ezt 1E-15-re fogjuk átállítani.

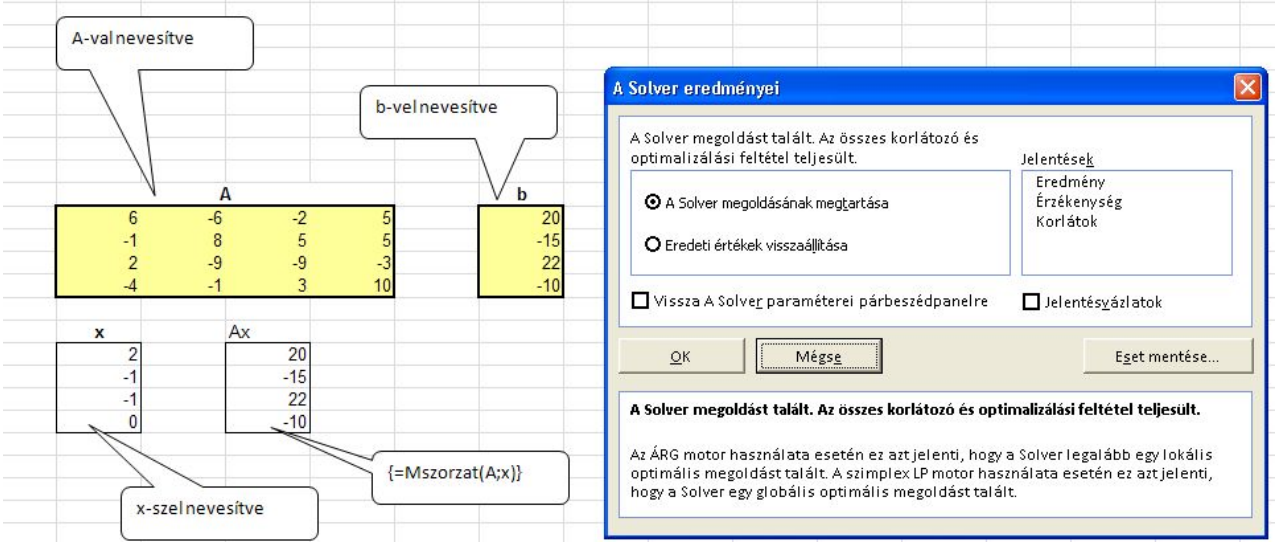


3.27. ábra: A Solver paraméterezése lineáris egyenletrendszer megoldásakor

A lineáris egyenletrendszer megoldásakor, ha az együtthatómátrix determinánsa nem nulla, nem kell célcellát és célt választani! Elegendő az egyenletrendszer teljesülését a korlátozó feltételek között megkövetelni (3.27. ábra).

A változó cellákra nevesítve is hivatkozhatunk, és nevesített blokkokat is megadhatunk a korlátozó feltételek között.

A változó cellák értékeit valamilyen (nem teljesen lehetetlen) kezdőértékekre állítjuk. Ez most a keresett **x** vektor kezdeti koordinátáit jelenti, amelyeket mind 1-nek választunk. Azt kell tudnunk, hogy több megoldással rendelkező feladatnál azt a megoldást fogja a Solver megtalálni, amelynek közeléből indultunk.

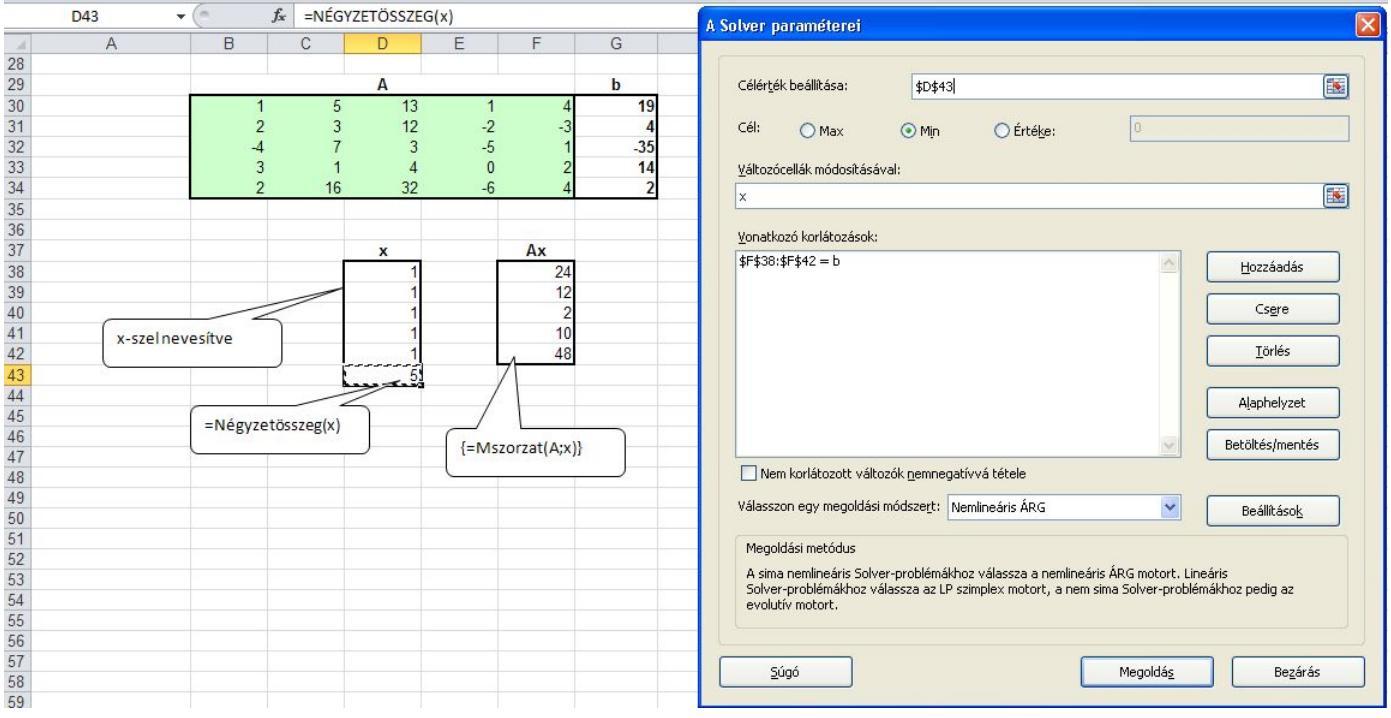


3.28. ábra: A Solver eredményének elfogadása

Látható, hogy amíg a direkt megoldásban (ld. előző lecke) az $x_4 = 0$ helyett a gépi epszilon környéki érték jelentkezett, addig a Solveres megoldás a teljesen pontos értékeket szolgáltatta.

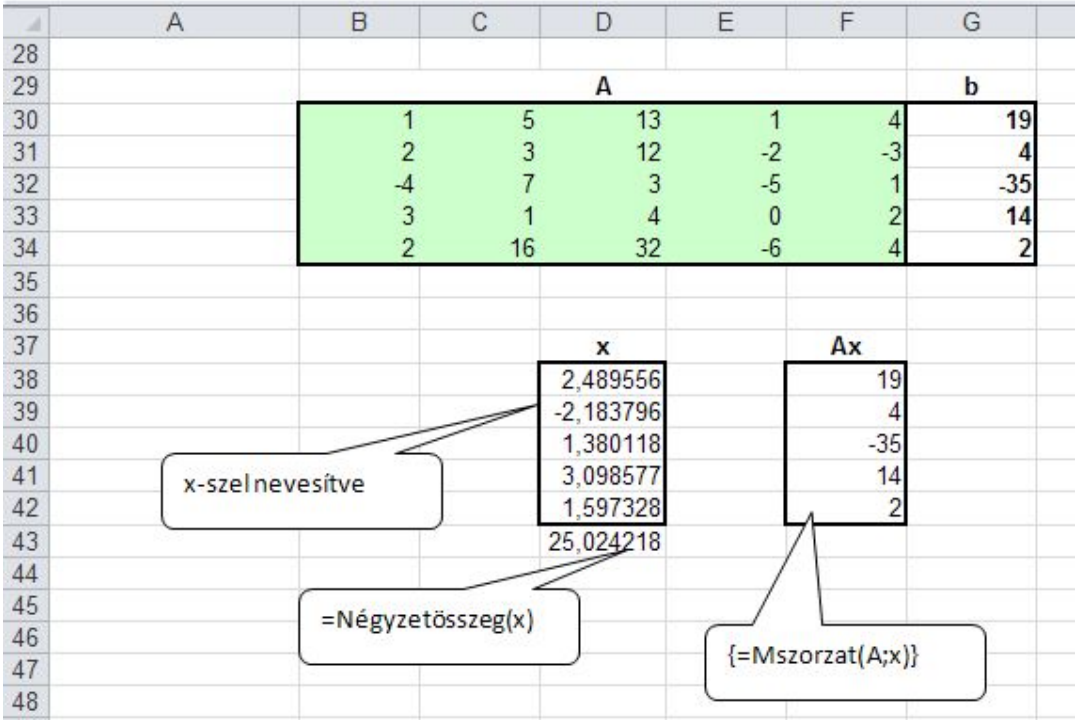
3.4.3. Összefüggő egyenletrendszer

Ezt a feladatot a pszeudoinverz segítségével már megoldottuk. Most ugyanezt a problémát a Solver segítségével fogjuk megoldani. A feladat egy minimumfeladat, az **x** megoldásvektor koordinátáinak négyzetösszegét kell minimalizálni.



3.29. ábra: Összefüggő egyenletrendszer paraméterezése Solverrel

A beállítások panelen a pontossági értéket 1E-12-re állítottuk be. A megoldás teljes mértékben megegyezik a pszeudoinverz módszerrel kapottal.

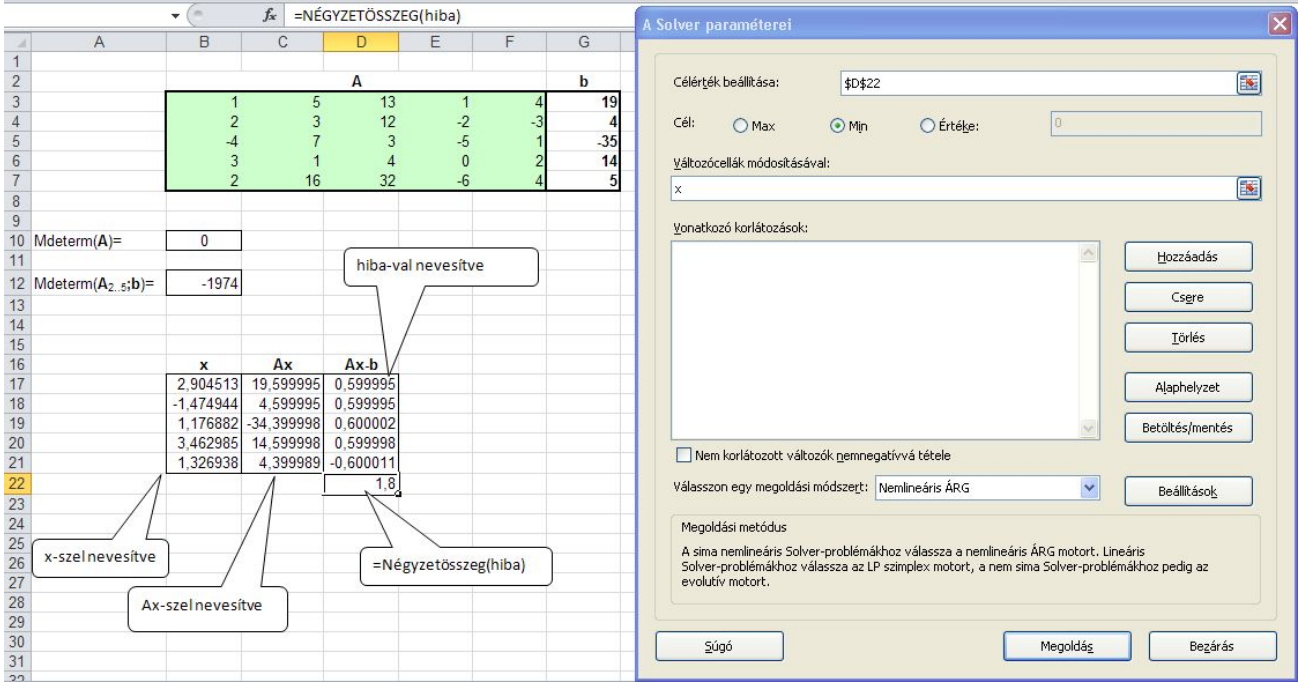


3.30. ábra: Összefüggő egyenletrendszer minimális normájú megoldása Solverrel

3.4.4. Ellentmondó egyenletrendszer

Ellentmondó egyenletrendszernek nincs megoldása. Mégis megpróbáljuk az **x** vektort úgy meghatározni, hogy az **Ax** vektor minél közelebb kerüljön a **b** vektorhoz, azaz az **Ax - b** eltérésvektor koordinátáinak négyzetösszegét fogjuk minimalizálni.

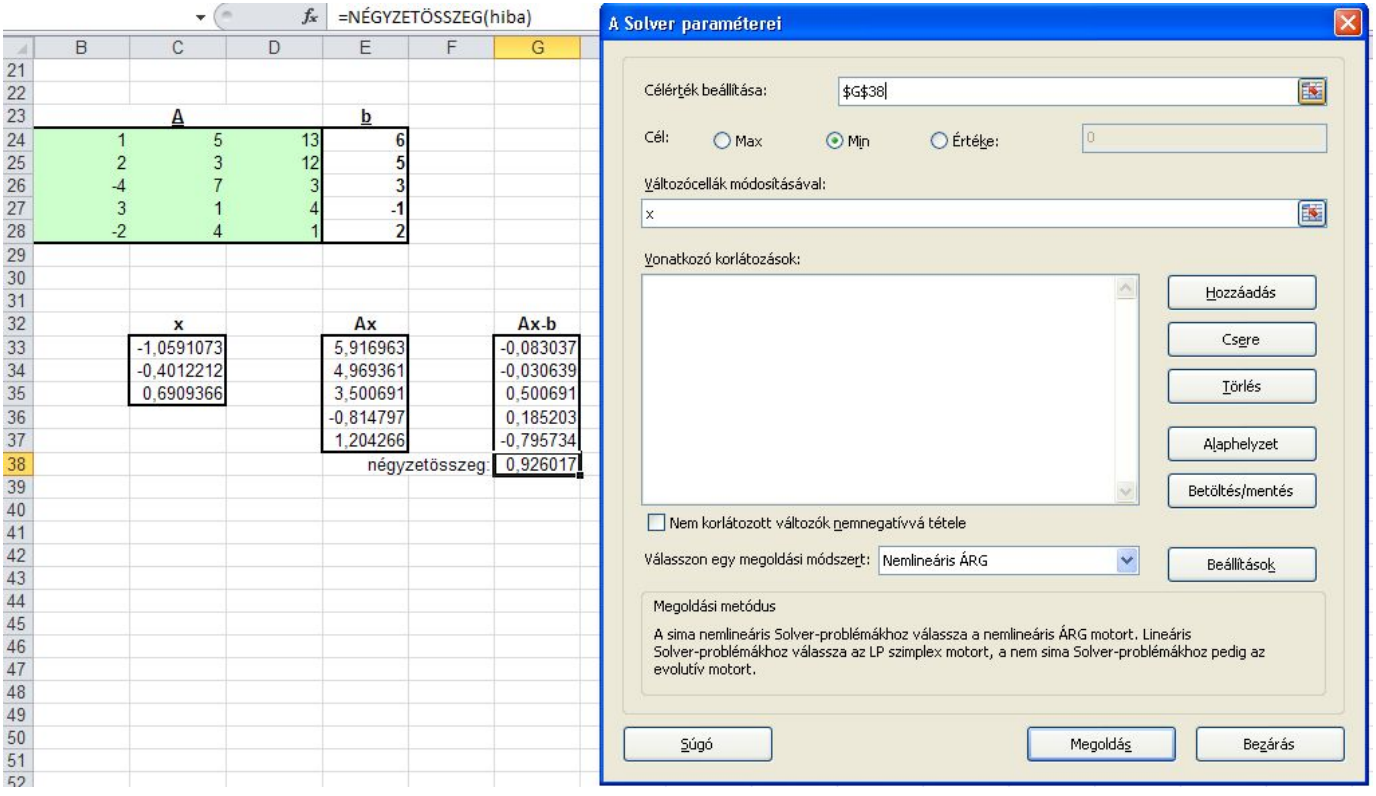
Minden adatblokkot füzetszinten nevesítettünk. Az x vektor koordinátáit induláskor mind 1-re állítottuk be. A pontosság 1E-12. A12. A 3.31. ábra a kapott megoldáson kívül a Solver paraméterezését is mutatja.



3.31. ábra: Ellentmondó egyenletrendszer paraméterezése és megoldása Solverrel

3.4.5. Túlhatározott egyenletrendszer

A túlhatározott egyenletrendszerek esetében hasonlóan járunk el, mint az ellentmondó egyenletrendszereknél, azaz az **Ax** - **b** hibavektor koordinátáinak négyzetösszegét minimalizáljuk (3.32. ábra).



3.32. ábra: Túlhatározott egyenletrendszer paraméterezése és megoldása Solverrel

Önellenőrzés

(A feladatokat először inverz mátrix segítségével oldjuk meg.)

1. Oldjuk meg a következő táblázattal adott **Ax** = **b** lineáris egyenletrendszert!

3	-1	5	4		35
5	10	-1	2		-7
2	-2	4	6		32
-1	0	2	8		13

Számítsuk ki az együtthatómátrix determinánsát, inverzét is!
Végezzük el a kapott megoldás segítségével az **Ax** - **b** = 0 ellenőrzést is!
A megoldás megtekintéséhez kattintson ide!

2. Oldjuk meg a következő táblázattal adott **Hx** = **b** lineáris egyenletrendszert!

1	1/2	1/3	1/4		13/6
1/2	1/3	1/4	1/5		7/6
1/3	1/4	1/5	1/6		49/60
1/4	1/5	1/6	1/7		19/30

(Az adott speciális **H** mátrixot Hilbert-mátrixnak nevezik)
Számítsuk ki a **H** együtthatómátrix determinánsát, inverzét is!
Végezzük el a kapott megoldás segítségével az **Hx** - **b** = 0 ellenőrzést is!
A megoldás megtekintéséhez kattintson ide!

3. Az előző feladatban a 2. sor 1. együtthatóját növeljük meg 1 ezrelékkel! Mi történik a megoldással?
A megoldás megtekintéséhez kattintson ide!

4. Vizsgáljuk meg a következő táblázattal adott lineáris egyenletrendszert!

2	-2	0	1		-14
-3	2	-2	8		-6
4	1	2	-4		8
19	-3	8	-18		2

Ellenőrizzük, hogy a baloldal (4x4-es méretű) mátrixának determinánsa 0!
Ellenőrizzük, hogy az első egyenlet kétszeresének és a harmadik egyenlet háromszorosának összegéből a második egyenletet levonva éppen a negyedik egyenletet kapjuk!
Nevesítsük az első három egyenlet **(**ezek már lineárisan függetlenek**)** baloldalának mátrixát **A**-val (3x4 méretű) és az első három egyenlet jobboldalát **b**-vel (3x1 méretű)!
A végtelen sok megoldásvektor közül határozzuk meg a „legrövidebbet”, amelynek képlete:
$\mathbf{x} = \mathbf{A}^*(\mathbf{A}\mathbf{A}^*)^{-1}\mathbf{b}$
(Itt * a transzponálás jele és $^{-1}$ az inverz képzés jele!)
A részszámítási lépések sorozata:
$\mathbf{A}^*$ (4*3 méretű); $\mathbf{A}\mathbf{A}^*$ (3x3 méretű); $(\mathbf{A}\mathbf{A}^*)^{-1}$ (3x3 méretű); $\mathbf{A}^*(\mathbf{A}\mathbf{A}^*)^{-1}$ (4*3 méretű); $\mathbf{x}$ (4x1 méretű)
Az eredeti (4x4) együtthatómátrix és az **x** vektor összeszorzásával ellenőrizzük, hogy a **b** vektort kaptuk-e!
A megoldás megtekintéséhez kattintson ide!

5. Oldjuk meg Solver segítségével is az előző feladatokat!
A megoldás megtekintéséhez kattintson ide!

15. LECKE

Lineáris programozás, nemlineáris feladatok

3.5. Lineáris és nemlineáris feladatok megoldása Solverrel

3.5.1. Lineáris programozási feladatok

A lineáris programozási feladatoknál egy többváltozós lineáris célfüggvény optimumát keressük, miközben a változókra többféle lineáris korlátozást kell betartani. Ezek a következők lehetnek:

1. Bizonyos lineáris kombinációik korlátosak;
2. A változók nem negatívok;
3. A változók egészek;
4. A változók binárisak (csak 0 vagy 1 értéket vehetnek fel).

Amikor egy szöveges feladatot megoldunk, akkor a következő fogalmakkal kell tisztában lennünk. Milyen ráfordítási értékek és korlátozások vannak a feladatban? Hogyan írjuk fel a célfüggvényt?

Tekintsük a következő példákat ([15])!

1. Példa

Egy kis bútoripari szövetkezet irodák számára gyárt asztalokat, székeket és könyvespolcokat. A faanyagot saját gépeiken szabják le.

1. A gépek napi kapacitása 100 munkaóra. Egy asztal, szék és könyvespolc anyagának leszabása 0,8, 0,4, illetve 0,5 órát igényel.
2. A szövetkezet festésre és fényezésre 650 munkaórát fordíthat naponta. Az egyes termékek festésének és fényezésének munkaidő-szükséglete 5, 3, illetve 2 munkaóra.

3. A kész bútorokat egy 150 m^2-es raktárban tárolják. Raktáron kívül bútort tárolni nem szabad. Így a szövetkezet naponta csak annyi bútort gyárthat, amennyi elfér a 150 m^2-es raktárban. A termékek fajlagos helyigénye a raktárban rendre: 1, 0,5 és 1,125 m^2.
4. A jelenlegi piaci helyzetben az asztalok 300, a székek 160, a polcok 250 Ft nyereséget hoznak darabonként.

Mennyit gyártson a bútoripari szövetkezet az egyes irodabútorokból, hogy nyeresége maximális legyen? (A nem teljesen készre gyártott bútornak is van értelme, mivel a termelés a következő nap is folytatódik.)

A feladatot a következő egyenlőtlenségrendszer és célfüggvény jellemzi:

$$\begin{aligned} x_1, \quad & x_2, \quad & x_3 \quad & \geq 0 \\ 0{,}8x_1+ \quad & 0{,}4x_2+ \quad & 0{,}5x_3 \quad & \geq 100 \\ 5x_1+ \quad & 3x_2+ \quad & 2x_3 \quad & \geq 650 \\ 1x_1+ \quad & 0{,}5x_2+ \quad & 1{,}125x_3 \quad & \geq 150 \\ 300x_1+ \quad & 160x_2+ \quad & 250x_3 \quad & \rightarrow max \end{aligned}$$

A feladatot jellemző értékeket a következő módon célszerű elhelyezni egy táblázatban:

	A	B	C	D	E	F	G	H
1	**Feladat:**							
2								
3			asztal	szék	polc	korlátozás	tény	
4		szabás (óra/db)	0,8	0,4	0,5	100	1,7	
5		festés (óra/db)	5	3	2	650	10	
6		raktár (m^2/db)	1	0,5	1,125	150	2,625	
7		haszon (Ft/db)	300	160	250	célfv.:	710	
8		gyártás (db)	1	1	1			
9								

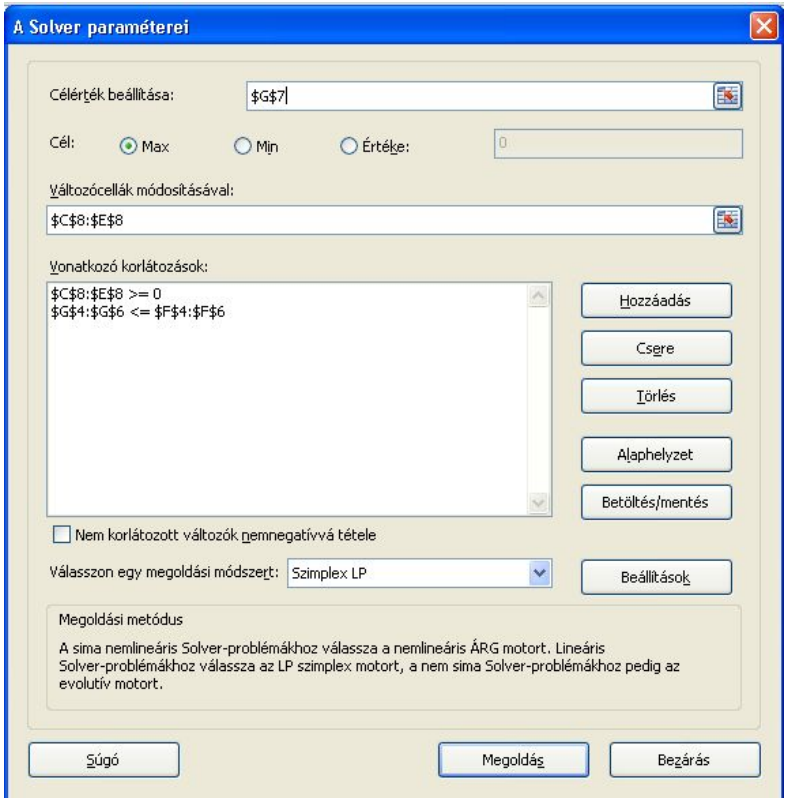
3.33. ábra: Az 1. lineáris programozási feladat adatelrendezése

A gyártás sorban elhelyeztük az x_1, x_2, x_3 termelés hipotetikus értékeit és a tény oszlopban pedig mátrixszorzás segítségével kiszámítottuk a jelenlegi felhasználás és a célfüggvény értékeit:

{=MSZORZAT(C4:E7; TRANSZPONÁLÁS(C8:E8))}

A Solver feladata lesz a gyártási értékek olyan változtatása, hogy a célfüggvény maximális legyen, miközben a tényértékek nem haladhatják meg a korlátozásokat.

A Solver paraméterezése nem nehéz, hiszen a célcella, a feladat és változó cellák megadása egyértelmű, a korlátozó feltételek megadása pedig a Hozzáadás nyomógombbal kezdeményezhető. A Solver beállítását és a kapott eredményt a következő ábrákon (3.34. és 3.35.) láthatjuk:



3.34. ábra: A Solver beállításai az 1. lineáris programozási feladathoz

	A	B	C	D	E	F	G
1	**Megoldás:**						
2							
3			asztal	szék	polc	korlátozás	tény
4		szabás (óra/db)	0,8	0,4	0,5	100	100
5		festés (óra/db)	5	3	2	650	650
6		raktár (m^2/db)	1	0,5	1,125	150	150
7		haszon (Ft/db)	300	160	250	célfv.:	42250
8		gyártás (db)	12,5	162,5	50		

3.35. ábra: Az 1. lineáris programozási feladat megoldása

2. Példa

Egy vegyészeti termékeket gyártó vállalatnál növényvédő szereket is készítenek, amelyek poralakban kerülnek forgalomba. A vállalat ötfajta növényvédő szert állít elő. Ezek a következők: BCM, Fundasol SOWP, Chinofurgin, Fundasol 25EC és Furoxon. A termékek előállítása (az alap- és segédanyagokból) ugyanazon a keverőgépen történik. A fajlagos időnorma termékenként a keverőgépen 2,5; 1,5; 3; 4; 4 óra/tonna. A keverőgép kapacitása 1000 óra. A termékek előállításához tízféle ható- és segédanyag szükséges. Ezekből négy anyag (A, B, C, D) felhasználása korlátozott. A növényvédő szerek fajlagos igénye ezekből az anyagokból (kg/tonnában) valamint a rendelkezésre álló mennyiségek (tonnában) a 2.10. táblázatban találhatók.

2.10. táblázat. Adatok a 2. lineáris programozási feladathoz

Anyagok	Növényvédő szerek					Felhasználható /korlátozás
	BCM	FSOWP	CHF	F25EC	FX	
A	500	0	0	0	0	65000
B	0	0	50	500	500	60000
C	50	25	0	50	50	12000
D	0	25	5	50	0	6000

Az egyes növényvédő szerek tonnánkénti nyeresége rendre: 6000, 2000, 2500, 4000, illetve 3500 Ft. Hány tonnát állítson elő az egyes növényvédő szerekből a vegyészeti termékeket gyártó vállalat, ha a maximális nyereség a cél?

A feladat matematikai modellje:

$$
\begin{aligned}
500x_1 & & & & & \leq 65000 \\
& & 50x_3+ & 500x_4+ & 500x_5 & \leq 60000 \\
50x_1+ & 25x_2+ & & 50x_4+ & 50x_5 & \leq 12000 \\
& 25x_2+ & 5x_3+ & 50x_4 & & \leq 6000 \\
2{,}5x_1+ & 1{,}5x_2+ & 3x_3+ & 4x_4+ & 4x_5 & \leq 1000 \\
6000x_1+ & 2000x_2+ & 2500x_3+ & 4000x_4+ & 3500x_5 & \rightarrow max \\
x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0
\end{aligned}
$$

Az időnorma és nyereség sorokat értelemszerűen kell a megadott adatokkal kitölteni. A **Gyártás** sorba kerülnek a megoldás (x_1, x_2, x_3, x_4, x_5) értékei. Ezt a sort töltsük ki valamilyen induló adatokkal, pl. csupa 1-gyel. A **Korlátozás** oszlopot a fenti táblázat szerint töltsük ki és írjuk ide még az időkorlátot is (1000 óra).

A **Tényleges** oszlopba az A, B, C, D, időnorma, nyereség sorok 6x5-os mátrixának és az *x*-eket tartalmazó sorvektor transzponáltjának mátrixszorzata kerül! (**Kulcslépés**!!!)

Helyezzük el az együttható-táblázatot és a célfüggvény együtthatóit a C5:G10 cellatartományban. Módszernek a Szimplex LP-t válasszuk.

Most már nekifoghatunk a Solverrel történő megoldásnak.

1. A célfüggvény értékét a maximalizálandó $I$10 célcella tartalmazza.
2. Módosuló cellák a tényleges termelést leíró **x** vektor elemei, azaz a $C$11:$G$11 cellák.
3. Korlátozás: **x** vektor minden eleme nem negatív: $C$11:$G$11 ≥ 0.
4. Korlátozás: a tényleges adatok zöld hátterű tartalmai nem haladhatják meg a korlátozás oszlop lila hátterű tartalmait: $I$5:$I$9 $\leq$ $H$5:$H$9.

A modell és a megoldás a 3.36. ábrán látható.

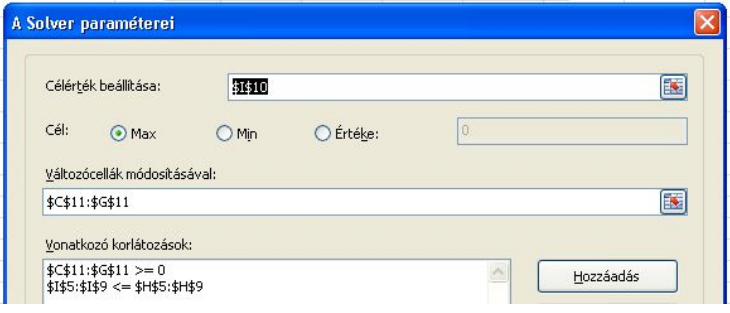
Anyagok	Növényvédő szerek					Felhasználható/korlátozás	Felhasznált /tényleges
	BCM	FSOWP	CHF	F25EC	FX		
A	500	0	0	0	0	65000	65000
B	0	0	50	500	500	60000	6451,612903
C	50	25	0	50	50	12000	12000
D	0	25	5	50	0	6000	6000
időnorma	2,5	1,5	3	4	4	1000	1000
nyereség	6000	2000	2500	4000	3500	Gépkapacitás	1505564,516 max.
Gyártás	130	217,0968	114,5161	0	1,451613		Célfüggvény:

A gyártás sorába a termelési adatok x oszlopvektoronának transzponáltja került

A feladat matematikai modellje:

	$500x_1$					≤ 65000
			$50x_3+$	$500x_4+$	$500x_5$	≤ 60000
	$50\ x_1+$	$25x_2+$		$50x_4+$	$50x_5$	≤ 12000
		$25x_2+$	$5x_3+$	$50x_4$		≤ 6000
	$2{,}5x_1+$	$1{,}5x_2+$	$3x_3+$	$4x_4+$	$4x_5$	≤ 1000
	$6000x_1+$	$2000x_2+$	$2500x_3+$	$4000x_4+$	$3500x_5$	$\rightarrow$max
	$x_1,$	$x_2,$	$x_3,$	$x_4,$	x_5	≥ 0

A tényleges oszlop mind 6 elemébe blokkművelettel =MSZORZAT(C5:G10;TRANSZPONÁLÁS(C11:G11)) kerül



3.36. ábra: A 2. lineáris programozási feladat modellje és megoldása

3. Példa

1000 darab 7 méteres oszlopból 1,5 és 2,5 méteres oszlopokat vágunk. Legalább négyszer annyi 1,5 méteres oszlopra van szükségünk, mint 2,5 méteresre. Hogyan vágjuk fel az 1000 darab 7 méteres oszlopot, hogy a hulladékképződés minimális legyen?

(Nyilvánvaló, hogy egy 7 méteres oszlop feldarabolásakor legfeljebb 1 méter hosszú hulladék keletkezhet.)

A megoldáshoz azt kell végiggondolni, hogy a felvágás az alábbi módszerekkel történhet (3.1. táblázat):

3.1. táblázat. Adatok a 3. lineáris programozási feladathoz

	Kész oszlop hossza		hulladék
	2,5 m	1,5 m	
1. módszer	2	1	0,5
2. módszer	1	3	0
3. módszer	0	4	1

Az egyes módszerek alkalmazásainak darabszámát x_1, x_2, x_3 jelölje. Ekkor $(2x_1 + x_2 + 0x_3)$ darab 2,5 méteres és $(x_1 + 3x_2 + 4x_3)$ darab 1,5 méteres oszlop lesz. A hosszabb oszlopok darabszámának 4-szerese nem haladhatja meg a rövidebb oszlopokét.

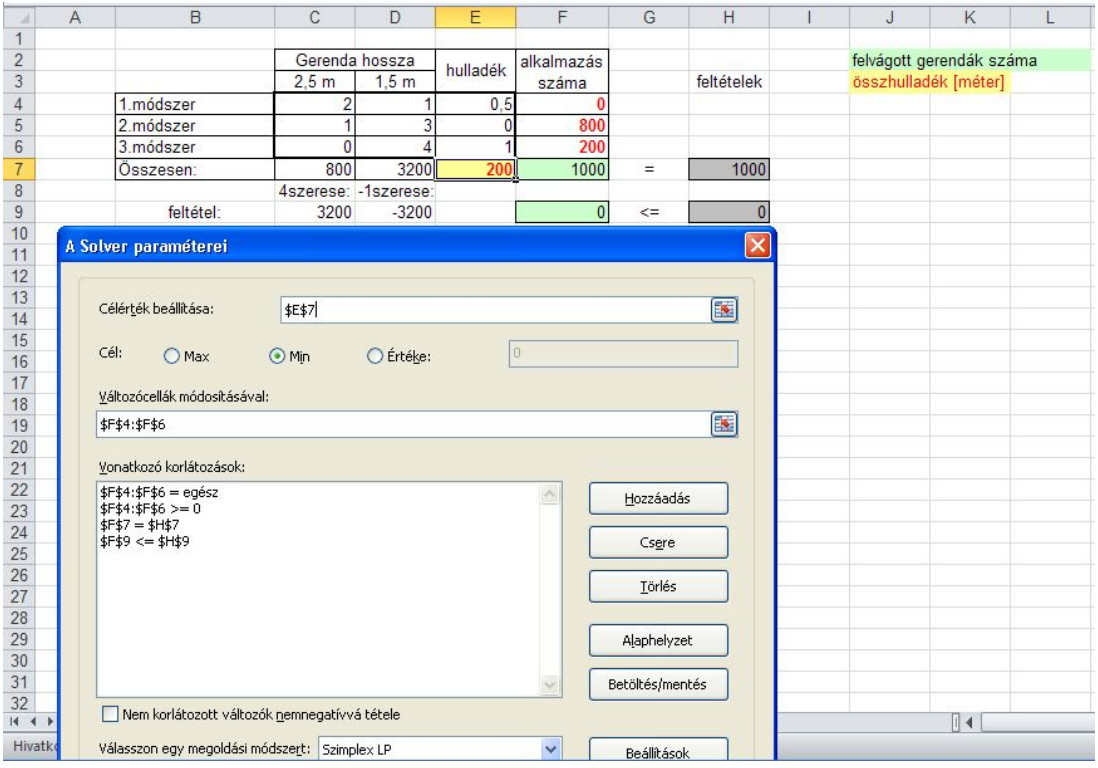
Ezzel a feladat matematikai modellje:

$$x_1 + x_2 + x_3 = 1000$$
$$4(2x_1 + x_2 + 0x_3) - (x_1 + 3x_2 + 4x_3) \leq 0$$
$$0{,}5x_1 + 0x_2 + x_3 \to min$$

Az ismeretlenek értékeinek helyét most célszerű a sorok végén kijelölni, így ha a fenti táblázatot jobbról kiegészítjük az **x** vektorral és alulról a keletkezett darabok számaival (2,5 méteres, 1,5 méteres, hulladék), akkor a Solver könnyedén paraméterezhető lesz:

1. Az **Összesen** sorba (C7:E7) a felette lévő és aktuális oszlopbeli értékek, valamint az F oszlopbeli értékek (4–6 sorbeli) szorzatösszege kerül.
2. Az F7 cellába a felette álló értékek összege kerül.
3. A C9 cellában a C7 négyszeresére hivatkozunk és a C10 cellában a D7 -1-szeresére hivatkozunk.
4. Az induló alkalmazás értékeket állítsuk 1-re!

Ekkor a feltételek könnyen felírhatók és hivatkozhatók lesznek.



3.37. ábra: A 3. lineáris programozási feladat paraméterezése és megoldása

3.5.2. Nemlineáris feladatok

A Solver segítségével nemlineáris egyenleteket és egyenletrendszereket is megoldhatunk.

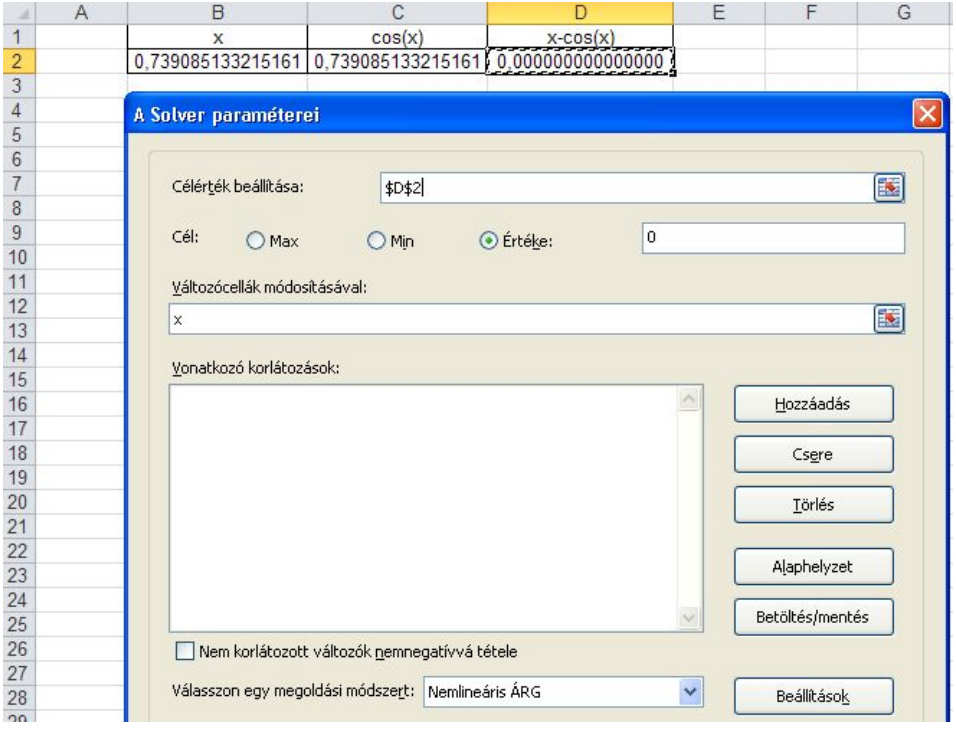
1. Példa

Tekintsük a következő nemlineáris egyenletet: $x = \cos(x)$.

Ennek az ún. fixponti egyenletrendszernek létezik a megoldása, mert az $x \to \cos(x)$ leképezés a $[\pi/6; \pi/4]$ intervallumban kontraktív (összehúzó), hiszen a deriváltjának abszolút értéke kisebb 1-nél és ezen kívül az

n	Pi()/n	cos(Pi()/n)
4	0,523599	0,866025
6	0,785398	0,707107

értéktáblázat, és a folytonos függvényekre vonatkozó Weierstrass-tétel alapján a megoldásnak léteznie kell.



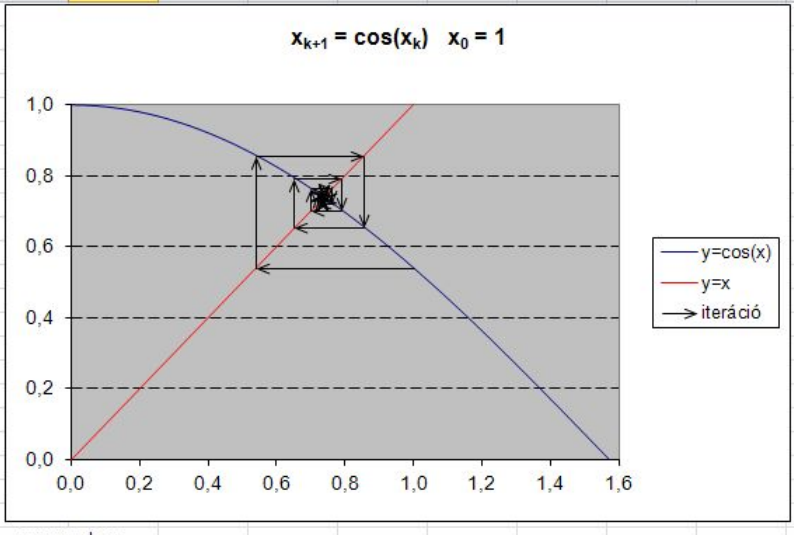
3.38. ábra: Az $x = \cos(x)$ feladat megoldása

Ezek szerint a megoldást akár fixponti iterációval (körkörös hivatkozás) is megkereshetnénk, de bízzuk a megoldás keresését a Solverre.

A megoldáshoz készítsük el a x, cos(x), x-cos(x) fejlécű táblázatot, amelyben a **Szám** cellaformátum 15 tizedes jegyű. Az induló 0,5 értéket tartalmazó cellát nevesítsük x-szel, majd a Solverrel kerestessük meg azt az x értéket, amelynél az x - cos(x) értéket tartalmazó cella 1E-15 pontossággal a 0 értéket veszi fel (lásd 3.38. ábra).

A fixponti iteráció lépéssorozatát ábrázolja a 3.39. ábra.

	x	y=cos(x)	spirálishoz	
2	0,000	1,0000	~~1,0000~~	0,5403
3	0,025	0,9997	~~0,5403~~	0,5403
4	0,050	0,9988	~~0,5403~~	0,8576
5	0,075	0,9972	~~0,8576~~	0,8576
6	0,100	0,9950	0,8576	0,6543
7	0,125	0,9922	0,6543	0,6543
8	0,150	0,9888	0,6543	0,7935
9	0,175	0,9847	0,7935	0,7935
10	0,200	0,9801	0,7935	0,7014
11	0,225	0,9748	0,7014	0,7014
12	0,250	0,9689	0,7014	0,7640
13	0,275	0,9624	0,7640	0,7640
14	0,300	0,9553	0,7640	0,7221
15	0,325	0,9477	0,7221	0,7221
16	0,350	0,9394	0,7221	0,7504
17	0,375	0,9305	0,7504	0,7504
18	0,400	0,9211	0,7504	0,7314
19	0,425	0,9110	0,7314	0,7314
20	0,450	0,9004	0,7314	0,7442
21	0,475	0,8893	0,7442	0,7442
22	0,500	0,8776	0,7442	0,7356
23	0,525	0,8653	0,7356	0,7356
24	0,550	0,8525	0,7356	0,7414
25	0,575	0,8392	0,7414	0,7414
26	0,600	0,8253	0,7414	0,7375
27	0,625	0,8110	0,7375	0,7375
28	0,650	0,7961	0,7375	0,7401
29	0,675	0,7807	0,7401	0,7401



egyeneshez	
x	y=x
0	0
1	1

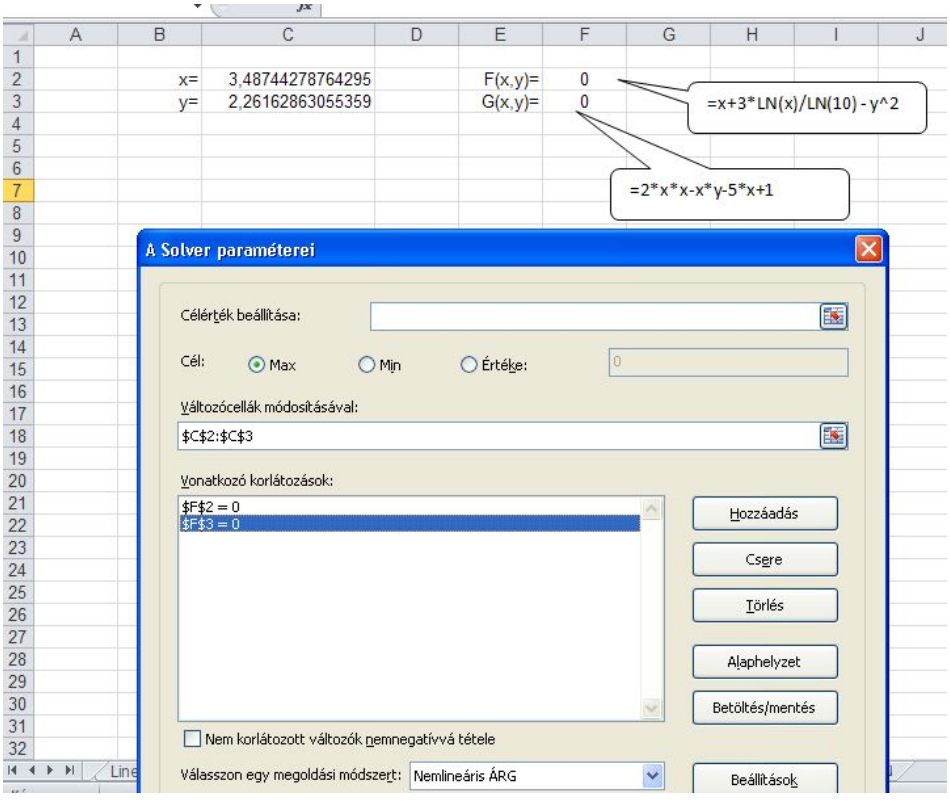
3.39. ábra: Az $x = cos(x)$ feladat iteratív megoldása

2. Példa

Határozzuk meg a következő

$$F(x, y) = x + 3\lg(x) - y^2 = 0$$
$$G(x, y) = 2x^2 - xy - 5x + 1 = 0$$

kétismeretlenes nemlineáris egyenletrendszer olyan megoldását, ahol x az $[1; 5]$ intervallumba esik!



3.40. ábra: Kétismeretlenes nemlineáris feladat megoldása

Ilyen egyenletrendszereknél csak akkor van esélyünk egy megoldás (több is lehet) megtalálására, ha a Solveres keresést a megoldás közeléből indítjuk. Erre vonatkozóan a feladat fizikai modelljéből, vagy esetleg az implicit egyenletek explicit alakjának (ha ez egyáltalán felírható) grafikus vizsgálata alapján indulhatunk ki.

A mostani feladatunknak az adott intervallumban van megoldása. Legyen x induló értéke 3, ennek és a második

egyenletnek megfelelően *y* induló értéke 2*3 - 5 + 1/3 = 4/3 lesz.

Állítsuk be a pontosságot 1E-15-re és kerestessük meg az itteni megoldást a Solverrel, amelyhez nem kell célérték, csak a korlátozó feltételeket fogjuk megadni (lásd 3.40. ábra).

3. Példa

Legyen adott egy ellipszis, amelynek centruma az origó, x tengely irányú féltengelyének hossza 2, az y tengely irányú féltengelyének a hossza pedig 1. Legyen adott továbbá egy kör, amelynek a középpontja a (0; 1) pont, sugara 2. Tekintsük azt a tartományt, amelynek pontjai az ellipszislap és a körlap közös pontjai. Határozzuk meg a tartomány azon pontját, amely a (2; 1) ponthoz legközelebb esik!

A feladat matematikai megfogalmazása:

Az ellipszis egyenlete:

$$\frac{x^2}{2^2} + \frac{y^2}{1^2} = 1$$

A kör egyenlete:

$$x^2 + (y-1)^2 = 2^2$$

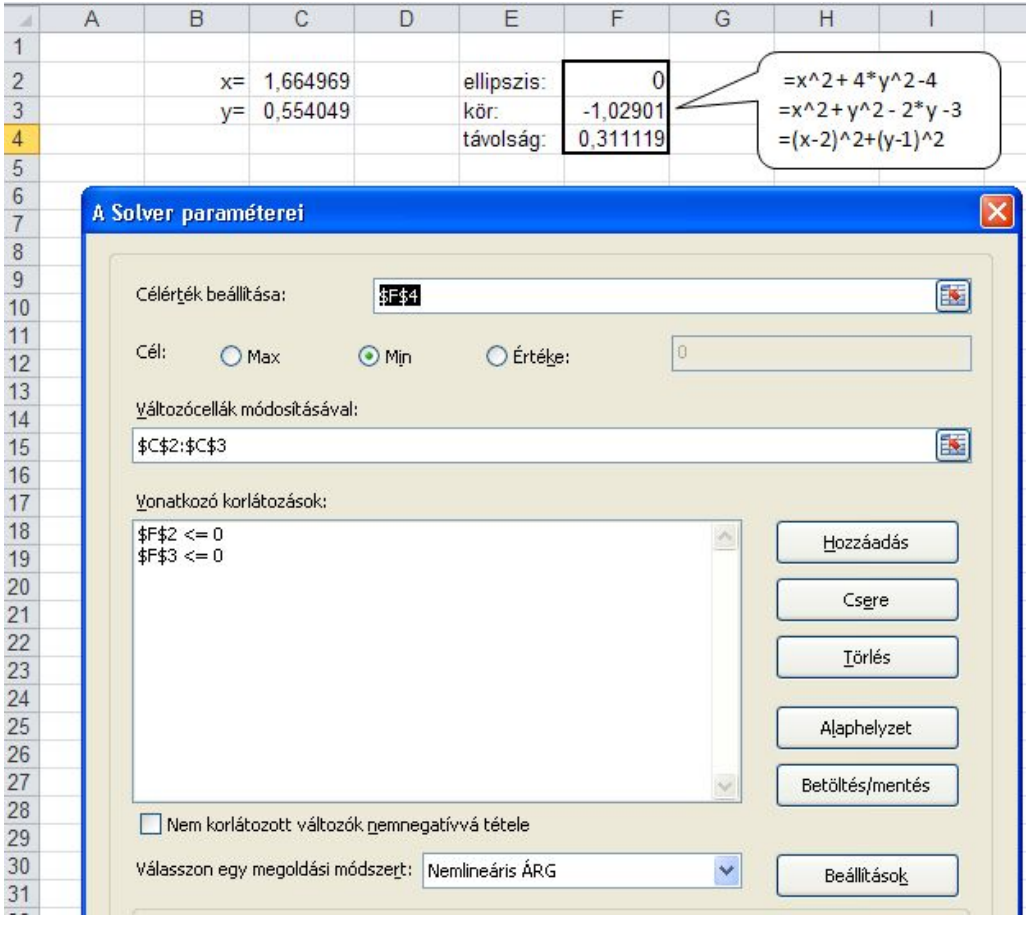
Ennek megfelelően a nemlineáris optimalizálási feladat a következő:

A távolság minimális: $(x - 2)^2 + (y - 1)^2 \rightarrow \min$

Nem vagyunk az ellipszisen kívül: $x^2 + 4y^2 - 4 \leq 0$

Nem vagyunk a körön kívül: $x^2 + y^2 - 2y - 3 \leq 0$

Az *x*, *y* értékeket az ellipszis (0; 1) pontjából indíthatjuk, az F2, F3 cellákba a korlátozó feltételekhez szükséges kifejezéseket írjuk, a célfüggvény kifejezését pedig az F4 cellában helyezzük el. A pontosság legyen 1E-14.



3.41. ábra: Nemlineáris programozási feladat adatelrendezése és megoldása

Önellenőrzés

1. Oldjuk meg a következő lineáris programozási feladatot!
Egy vállalat kétféle termék gyártását akarja bevezetni. A két termék gyártása három gépen történik (*munkafázisok*).
Az első termék egy darabjának megmunkálásához szükséges gépidők rendre 1, 1, 1 gépóra; a második termék egységének gépidő-szükséglete pedig rendre 2, 3, 1 gépóra. Az egyes gépeknek egy adott időszakban a rendelkezésre álló kapacitása 25, 33, 20 gépóra. Az egyes termékek várható eladási egységára rendre 3 és 5 pénzegység.
A vállalat milyen termékösszetételben gyártson, ha maximális árbevételre törekszik úgy, hogy a gyártás során a gépek kapacitását nem lépheti túl? Megjegyzés: félkész termék nem gyártható!
Segítségként megadjuk a feladat matematikai leírását is:
Jelölje x_1 és x_2 a gyártandó mennyiségeket! Ezzel

$$1x_1 + 2x_2 \leq 25$$
$$1x_1 + 3x_2 \leq 33$$
$$1x_1 + 1x_2 \leq 20$$
$$x_1; x_2 \geq 0$$
$x_1; x_2$ egész
$3x_1 + 5x_2 \rightarrow \max!$

16. LECKE

Függvényábrázolás

3.6. Függvényábrázolás és -vizsgálat

3.6.1. Egyváltozós függvények

Ábrázoljuk a következő függvényeket a [-4; 4] intervallumban!

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \qquad \phi(x) = \int_{-\infty}^{x} \varphi(t)dt$$

Ezek a standard normális eloszlást jellemző függvények. (Az Excel-be beépítettek!)

Az Excel csak úgy tud függvényt ábrázolni, ha egy olyan értéktáblázattal rendelkezünk, amelynek egyik oszlopában (sorában) az értelmezési tartomány egy növekvő értéksorozata van, a másik oszlopában (sorában) az ehhez tartozó függvényértékek szerepelnek.

1. lépés: Az adattábla létrehozása

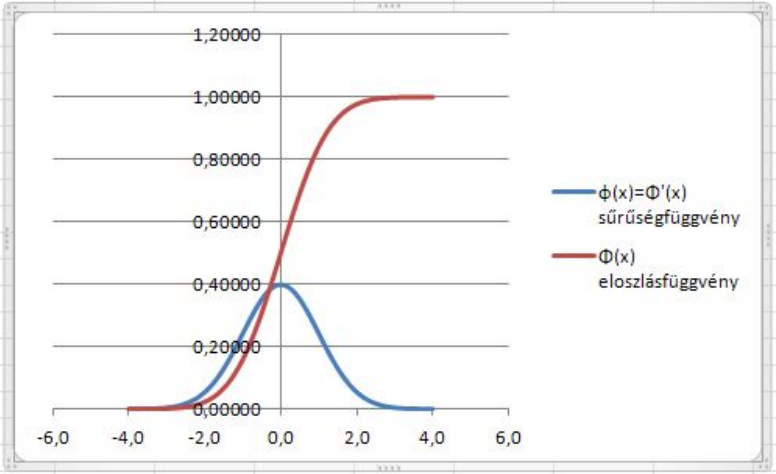
1. Fejlécek begépelése (görög betűk és egyéb jelek esetén: Beszúrás/Szimbólum).
2. Az x oszlopbeli adatok kitöltése a [-4; 4] intervallumban 0,1-es lépésközzel (sorozat, húzással).
3. Első függvényértékek bevitele a B és C oszlopokba:
   =NORM.ELOSZL(A2;0;1;0) =NORM.ELOSZL(A2;0;1;1)

Ezt másoljuk az oszlop többi cellájába. A NORM.ELOSZL() függvény 2. és 3. adata egy normális eloszlás várható-érték és szórás paramétereit közli, ami standardizált eloszlások esetén mindig 0 és 1. Az utolsó 4. adat egy flag, ami azt jelzi, hogy a valószínűségsűrűség-függvényre, vagy a kumulatív eloszlásfüggvényre van-e szükségünk.

2. lépés: A diagram elkészítése

Kijelöljük a forrásadat-tartományt – az x, φ(x), Φ(x) oszlopokat – fejléccel együtt, majd a diagramot létrehozzuk a **Beszúrás/Diagram/Pontdiagramok/Pont görbített vonalakkal jelölők nélkül** menüsoron keresztül.

	A	B	C
1	x	φ(x)=Φ'(x) sűrűségfüggvény	Φ(x) eloszlásfüggvény
2	-4,0	0,00013	0,00003
3	-3,9	0,00020	0,00005
4	-3,8	0,00029	0,00007
5	-3,7	0,00042	0,00011
6	-3,6	0,00061	0,00016
7	-3,5	0,00087	0,00023
8	-3,4	0,00123	0,00034
9	-3,3	0,00172	0,00048
10	-3,2	0,00238	0,00069
11	-3,1	0,00327	0,00097
12	-3,0	0,00443	0,00135
13	-2,9	0,00595	0,00187
14	-2,8	0,00792	0,00256
15	-2,7	0,01042	0,00347
16	-2,6	0,01358	0,00466
17	-2,5	0,01753	0,00621
18	-2,4	0,02239	0,00820
19	-2,3	0,02833	0,01072
20	-2,2	0,03547	0,01390
21	-2,1	0,04398	0,01786
22	-2,0	0,05399	0,02275
23	-1,9	0,06562	0,02872
24	-1,8	0,07895	0,03593



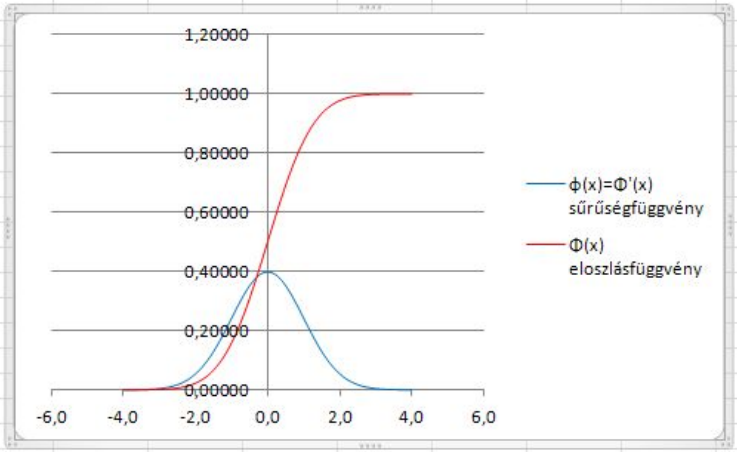
3.42. ábra: Fejléces értéktáblázatból jelölő nélküli görbített vonalas diagram

3. lépés: A grafikon hangolása

Ezt legegyszerűbben a **Diagramok/Elrendezés** menüben kezdeményezhetjük. A **Diagramterület** legördülő menüben kiválasztjuk azt a grafikus objektumot, amit hangolni akarunk, majd **Kijelölés formázása** indításával elvégezzük a hangolást.

A függvénygörbéink színét és vonalvastagságát állítjuk be először (3.43. ábra).

	A	B	C
1	x	φ(x)=Φ'(x) sűrűségfüggvény	Φ(x) eloszlásfüggvény
2	-4,0	0,00013	0,00003
3	-3,9	0,00020	0,00005
4	-3,8	0,00029	0,00007
5	-3,7	0,00042	0,00011
6	-3,6	0,00061	0,00016
7	-3,5	0,00087	0,00023
8	-3,4	0,00123	0,00034
9	-3,3	0,00172	0,00048
10	-3,2	0,00238	0,00069
11	-3,1	0,00327	0,00097
12	-3,0	0,00443	0,00135
13	-2,9	0,00595	0,00187
14	-2,8	0,00792	0,00256
15	-2,7	0,01042	0,00347
16	-2,6	0,01358	0,00466
17	-2,5	0,01753	0,00621
18	-2,4	0,02239	0,00820
19	-2,3	0,02833	0,01072
20	-2,2	0,03547	0,01390

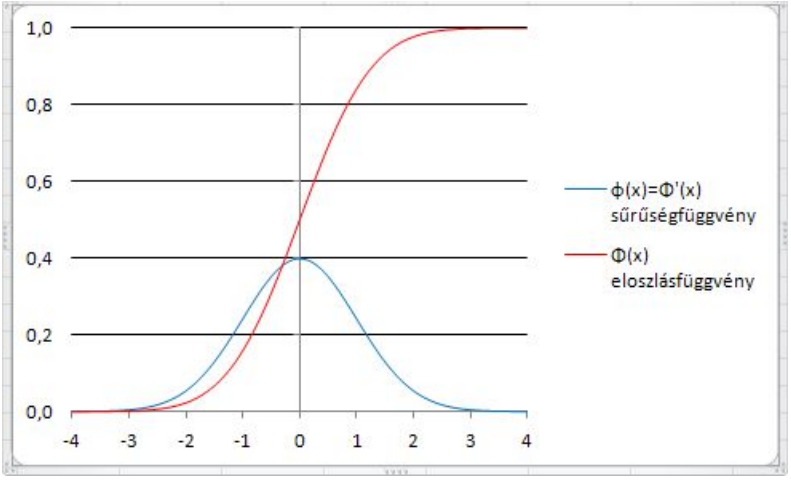


3.43. ábra: Jelölő nélküli görbített vonalas diagram 0,5 pontos vonallal

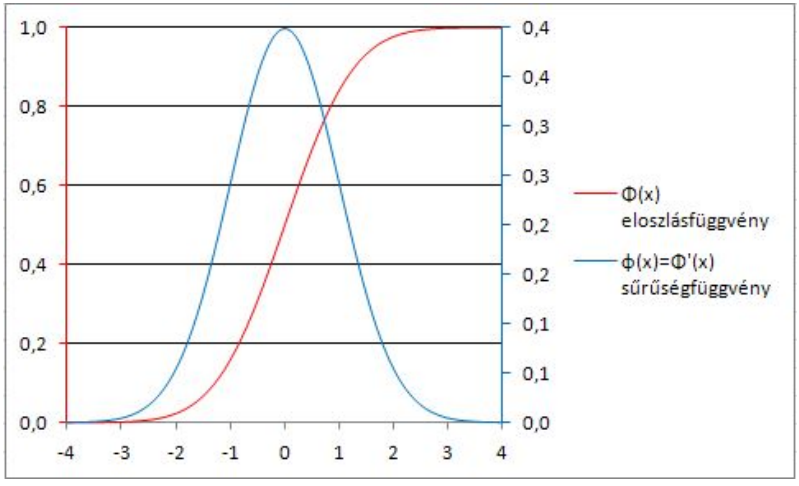
Ezután a tengelyek rendbetétele következik (3.44. ábra):

1. Min. és max. értékek korrigálása (a tényleges intervallumra korlátozás);
2. Fő lépték beállítása;
3. Tengelyek metszéspontja;
4. Számformátum – tizedes jegyek;
5. Tengelyfeliratok helye – pl. alul.

Az alacsonyabbik vonalat másodlagos tengellyel is elláthatjuk az **Adatsor formázása**kor, amit természetesen ismét hangolhatunk. A 3.45. ábrán a felvett másodlagos tengely skálázásának számformátuma még nem lett hangolva! Láthatóan még ezt is be kell állítani!



3.44. ábra: Adott intervallumra korlátozott görbített vonalas diagram

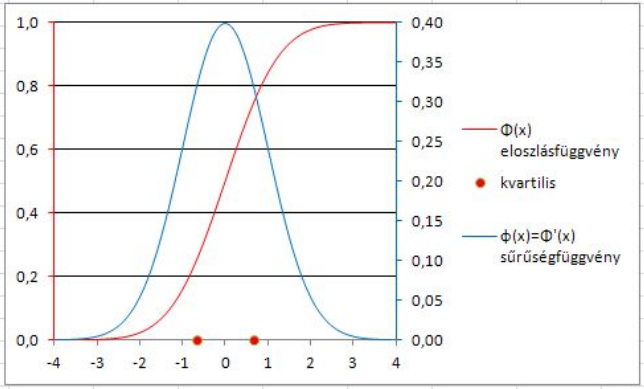


3.45. ábra: Másodlagos értéktengellyel ellátott görbített vonalas diagram

4. lépés: Nevezetes pontok keresése, ábrázolása

A függvénygörbénk nevezetes pontjait a Solverrel kerestethetjük meg. Példaképpen kerestessük meg az eloszlásfüggvényen azokat a helyeket, ahol az a 0,25 illetve a 0,75 értékeket veszi fel (kvartilisek). Ehhez az értéktáblázatból kimásolunk egy szeletet (két sort) és ezeken végezzük el a Solveres keresést, majd az aktív diagramon a másodlagos egérrel kezdeményezett menüben az **Adatok kijelölése** pontot választjuk, és a kis értéktáblázatunk x, y koordinátájú pontjait felvesszük a grafikonra. A másodlagos tengely számformátumát is javítottuk!

	A	B	C
1	x	φ(x)=Φ'(x) sűrűségfüggvény	Φ(x) eloszlásfüggvény
2	-4,0	0,00013	0,00003
3	-3,9	0,00020	0,00005
4	-3,8	0,00029	0,00007
5	-3,7	0,00042	0,00011
6	-3,6	0,00061	0,00016
7	-3,5	0,00087	0,00023
8	-3,4	0,00123	0,00034
9	-3,3	0,00172	0,00048
10	-3,2	0,00238	0,00069
11	-3,1	0,00327	0,00097
12	-3,0	0,00443	0,00135
13	-2,9	0,00595	0,00187
14	-2,8	0,00792	0,00256
15	-2,7	0,01042	0,00347
16	-2,6	0,01358	0,00466
17	-2,5	0,01753	0,00621
18	-2,4	0,02239	0,00820
19	-2,3	0,02833	0,01072
20	-2,2	0,03547	0,01390
21	-2,1	0,04398	0,01786
22	-2,0	0,05399	0,02275
23	-1,9	0,06562	0,02872
24	-1,8	0,07895	0,03593
25	-1,7	0,09405	0,04457
26	-1,6	0,11092	0,05480
27	-1,5	0,12952	0,06681
28	-1,4	0,14973	0,08076



x	y	
-0,67449	0,00000	0,25000
0,67449	0,00000	0,75000

=NORM.ELOSZL(F22;0;1;1)

3.46. ábra: Nevezetes pontok a görbített vonalas diagramon

Hasonlóan kerestethetjük meg és ábrázolhatjuk a függvényeink zérus helyeit, szélsőérték-helyeit, stb.

3.6.2. Paraméteresen adott függvények

A műszaki életben sok síkgörbe csak paraméteresen adható meg, a pontjainak x és y koordinátái közötti kapcsolatot direkt képlettel nem lehet leírni. Sok esetben a paraméteres megadást a polárkoordinátás leírás jelenti. Más esetekben a paraméterek jelentése már összetettebb lehet.

Ilyen, paraméteresen adott görbék például a kardiodid, epiciklois, lemniszkáta, Descartes-levél, Archimédeszi spirális stb. Ezeket a görbéket az Excel segítségével nem túl bonyolult feladat ábrázolni.

Vegyük például a körevolvenst és az epicikloist. Ezek metszik egymást. A feladatunk az lesz, hogy a metszéspontjaikat.

Egyenleteik paraméteresen adottak (lásd [17]):

$$\text{Körevolvens:} \quad x = p(\cos(\phi) + \phi \sin(\phi)) \quad y = p(\sin(\phi) - \phi \cos(\phi))$$

$$\text{Epiciklois:} \quad x = (q+w)\cos(\alpha) - w\cos(\tfrac{q+w}{w}\alpha) \quad y = (q+w)\sin(\alpha) - w\sin(\tfrac{q+w}{w}\alpha)$$

Legyenek a síkgörbéink paraméterei:

$$p = 3, q = 9, w = 3$$

Első lépésként elkészítjük az értéktáblázataikat, a körevolvensét 2 fokonként, az epicikloisét fokonként 0-tól 360 fokig. A paraméterértékekre nevesítve hivatkozunk (3.47. ábra).

Ezután következik a diagram létrehozása. Először az xK, yK értékpárokból készítjük el a folytonos vonalú pontdiagramot, majd aktív diagramterületnél az **Adatok kijelölése** menüben **Hozzáadjuk** az xE, yE értékpárokat. A vonalak színét, vastagságát, a tengelyeket meghangoljuk (3.48. ábra).

Három metszéspontot találunk. A metszéspontok közelítő koordinátapárjait megkeressük mindkét síkgörbe táblázatában, és ezeket a szögadatokkal együtt külön táblázatrészbe másoljuk azért, hogy a Solverrel a metszéspontokat pontosítsuk.

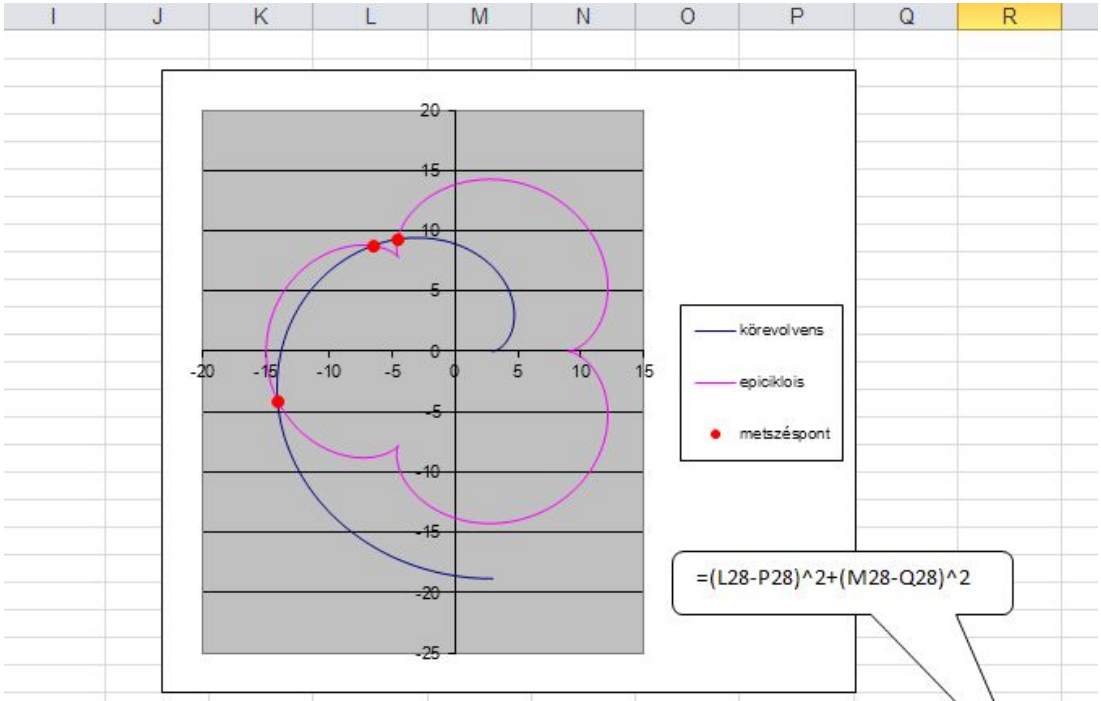
	A	B	C	D	E	F	G	H
1	p=	3			q=	9	w=	3
2	körevolvens				epiciklois			
3	ϕ [fok]	ϕ	xK	yK	α [fok]	α	xE	yE
4	0	0,000000	3,000000	0,000000	0	0,000000	9,000000	0,000000
5	2	0,034907	3,001827	0,000043	1	0,017453	9,005480	0,000159
6	4	0,069813	3,007302	0,000340	2	0,034907	9,021886	0,001275
7	6	0,104720	3,016404	0,001147	3	0,052360	9,049112	0,004296
8	8	0,139626	3,029101	0,002717	4	0,069813	9,086984	0,010166
9	10	0,174533	3,045345	0,005300	5	0,087266	9,135259	0,019808
10	12	0,209440	3,065078	0,009147	6	0,104720	9,193626	0,034132
11	14	0,244346	3,088225	0,014502	7	0,122173	9,261711	0,054017
12	16	0,279253	3,114702	0,021607	8	0,139626	9,339073	0,080319
13	18	0,314159	3,144411	0,030701	9	0,157080	9,425209	0,113858
14	20	0,349066	3,177241	0,042017	10	0,174533	9,519560	0,155415
15	22	0,383972	3,213067	0,055781	11	0,191986	9,621507	0,205733
16	24	0,418879	3,251757	0,072215	12	0,209440	9,730379	0,265506
17	26	0,453786	3,293162	0,091534	13	0,226893	9,845456	0,335380
18	28	0,488692	3,337124	0,113946	14	0,244346	9,965970	0,415950
19	30	0,523599	3,383474	0,139650	15	0,261799	10,091110	0,507752
20	32	0,558505	3,432033	0,168840	16	0,279253	10,220027	0,611266
21	34	0,593412	3,482608	0,201696	17	0,296706	10,351837	0,726909
22	36	0,628319	3,535000	0,238395	18	0,314159	10,485627	0,855034
23	38	0,663225	3,588999	0,279099	19	0,331613	10,620457	0,995931
24	40	0,698132	3,644385	0,323963	20	0,349066	10,755367	1,149818
25	42	0,733038	3,700930	0,373131	21	0,366519	10,889380	1,316850
26	44	0,767945	3,758397	0,426735	22	0,383972	11,021508	1,497107
27	46	0,802851	3,816544	0,484897	23	0,401426	11,150757	1,690601

3.47. ábra: Körevolvens és epiciklois fejléces értéktáblázata

A Solvernek azt a feladatot adjuk, hogy egy metszéspont megkeresése érdekében az

$$(xK - xE)^2 + (yK - yE)^2$$

értékét minimalizálja a φ és az α értékek módosítása révén. Az egyszerre módosítandó cellák az első metszéspont keresésekor a 3.48. ábrán látható táblázat J28; N28 cellái lesznek.



körevolvens				epiciklois				
ϕ [fok]	ϕ	xK	yK	α [fok]	α	xE	yE	
274,2222	4,786080	-14,098400	-4,048969	190,1009	3,317887	-14,098496	-4,049093	0,0000
200,7735	3,504159	-6,533485	8,765031	140,7605	2,456735	-6,533454	8,764979	0,0000
189,2451	3,302950	-4,552968	9,298161	103,1725	1,800700	-4,552984	9,298159	0,0000

3.48. ábra: Körevolvens és epiciklois metszéspontjai, és a Solveres kereséshez szükséges táblázat

Mindhárom metszéspont meghatározása után az **Adatok kijelölése** menüben a zölddel jelölt területről **Hozzáadjuk** a metszéspontok koordinátáit csak jelölővel, de vonal nélkül.

3.6.3. Felületábrázolás

Folytonos kétváltozós függvénnyel megadott felületet is gyorsan lehet ábrázolni az Excellel. Ehhez az kell, hogy az alapsíkon egy rácsháló szélein az x ill. y értékeit felsoroljuk és a belső rácspontokban felvett függvényértékeket a belső cellákban elhelyezzük.

Példaként tekintsük a következő hiperbolikus paraboloidot:

$$z = \frac{x^2}{a^2} - \frac{y^2}{b^2}$$

Legyen az a paraméter értéke 2, a b paraméteré pedig 3. Az előkészített rácsháló a 3.49. ábrán látható.

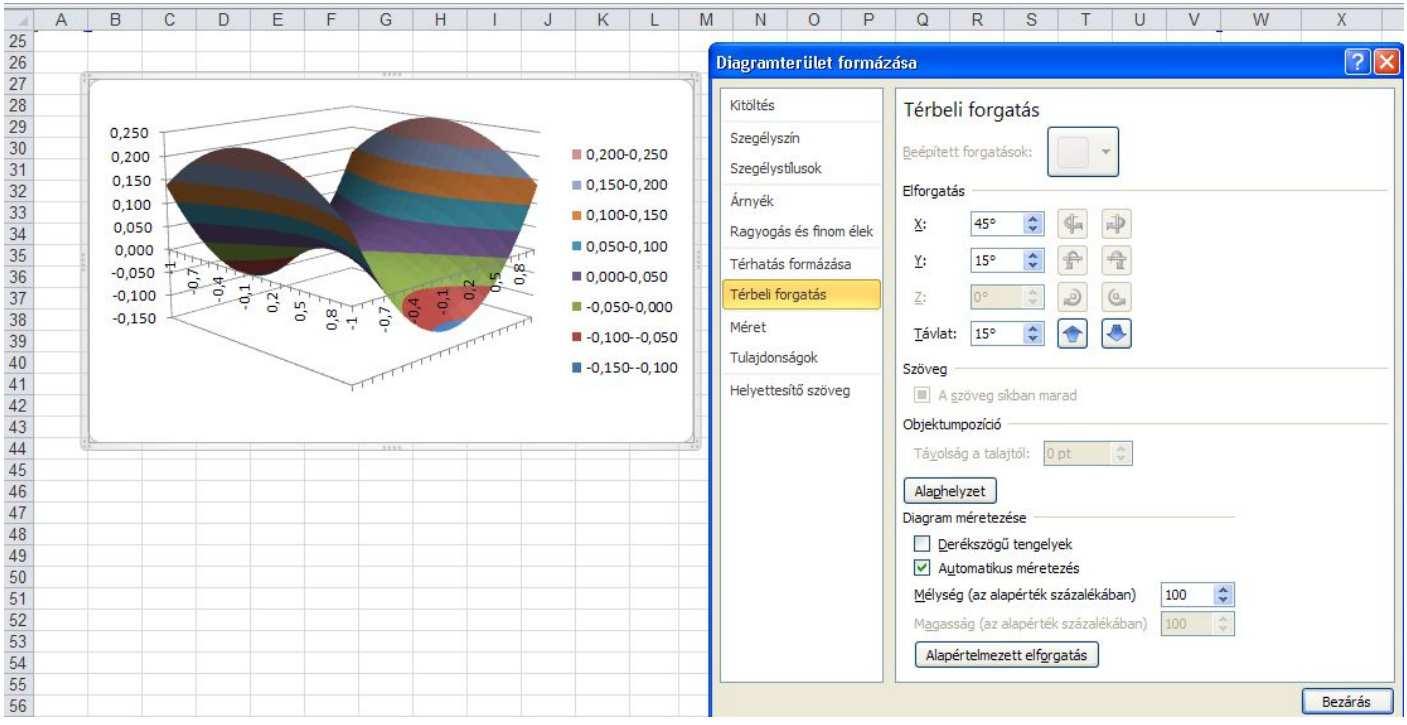
D6 =x^2/a^2-y^2/b^2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1		a=	2		b=	3																
2																						
3		-1	-0,9	-0,8	-0,7	-0,6	-0,5	-0,4	-0,3	-0,2	-0,1	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
4	-1	0,139	0,160	0,179	0,196	0,210	0,222	0,232	0,240	0,246	0,249	0,250	0,249	0,246	0,240	0,232	0,222	0,210	0,196	0,179	0,160	0,139
5	-0,9	0,091	0,113	0,131	0,148	0,163	0,175	0,185	0,193	0,198	0,201	0,203	0,201	0,198	0,193	0,185	0,175	0,163	0,148	0,131	0,113	0,091
6	-0,8	0,049	0,070	0,089	0,106	0,120	0,132	0,142	0,150	0,156	0,159	0,160	0,159	0,156	0,150	0,142	0,132	0,120	0,106	0,089	0,070	0,049
7	-0,7	0,011	0,033	0,051	0,068	0,083	0,095	0,105	0,113	0,118	0,121	0,123	0,121	0,118	0,113	0,105	0,095	0,083	0,068	0,051	0,033	0,011
8	-0,6	-0,021	0,000	0,019	0,036	0,050	0,062	0,072	0,080	0,086	0,089	0,090	0,089	0,086	0,080	0,072	0,062	0,050	0,036	0,019	0,000	-0,021
9	-0,5	-0,049	-0,028	-0,009	0,008	0,023	0,035	0,045	0,053	0,058	0,061	0,063	0,061	0,058	0,053	0,045	0,035	0,023	0,008	-0,009	-0,028	-0,049
10	-0,4	-0,071	-0,050	-0,031	-0,014	0,000	0,012	0,022	0,030	0,036	0,039	0,040	0,039	0,036	0,030	0,022	0,012	0,000	-0,014	-0,031	-0,050	-0,071
11	-0,3	-0,089	-0,068	-0,049	-0,032	-0,018	-0,005	0,005	0,013	0,018	0,021	0,023	0,021	0,018	0,013	0,005	-0,005	-0,018	-0,032	-0,049	-0,068	-0,089
12	-0,2	-0,101	-0,080	-0,061	-0,044	-0,030	-0,018	-0,008	0,000	0,006	0,009	0,010	0,009	0,006	0,000	-0,008	-0,018	-0,030	-0,044	-0,061	-0,080	-0,101
13	-0,1	-0,109	-0,088	-0,069	-0,052	-0,038	-0,025	-0,015	-0,008	-0,002	0,001	0,003	0,001	-0,002	-0,008	-0,015	-0,025	-0,038	-0,052	-0,069	-0,088	-0,109
14	0	-0,111	-0,090	-0,071	-0,054	-0,040	-0,028	-0,018	-0,010	-0,004	-0,001	0,000	-0,001	-0,004	-0,010	-0,018	-0,028	-0,040	-0,054	-0,071	-0,090	-0,111
15	0,1	-0,109	-0,088	-0,069	-0,052	-0,038	-0,025	-0,015	-0,008	-0,002	0,001	0,003	0,001	-0,002	-0,008	-0,015	-0,025	-0,038	-0,052	-0,069	-0,088	-0,109
16	0,2	-0,101	-0,080	-0,061	-0,044	-0,030	-0,018	-0,008	0,000	0,006	0,009	0,010	0,009	0,006	0,000	-0,008	-0,018	-0,030	-0,044	-0,061	-0,080	-0,101
17	0,3	-0,089	-0,068	-0,049	-0,032	-0,018	-0,005	0,005	0,013	0,018	0,021	0,023	0,021	0,018	0,013	0,005	-0,005	-0,018	-0,032	-0,049	-0,068	-0,089
18	0,4	-0,071	-0,050	-0,031	-0,014	0,000	0,012	0,022	0,030	0,036	0,039	0,040	0,039	0,036	0,030	0,022	0,012	0,000	-0,014	-0,031	-0,050	-0,071
19	0,5	-0,049	-0,028	-0,009	0,008	0,023	0,035	0,045	0,053	0,058	0,061	0,063	0,061	0,058	0,053	0,045	0,035	0,023	0,008	-0,009	-0,028	-0,049
20	0,6	-0,021	0,000	0,019	0,036	0,050	0,062	0,072	0,080	0,086	0,089	0,090	0,089	0,086	0,080	0,072	0,062	0,050	0,036	0,019	0,000	-0,021
21	0,7	0,011	0,033	0,051	0,068	0,083	0,095	0,105	0,113	0,118	0,121	0,123	0,121	0,118	0,113	0,105	0,095	0,083	0,068	0,051	0,033	0,011
22	0,8	0,049	0,070	0,089	0,106	0,120	0,132	0,142	0,150	0,156	0,159	0,160	0,159	0,156	0,150	0,142	0,132	0,120	0,106	0,089	0,070	0,049
23	0,9	0,091	0,113	0,131	0,148	0,163	0,175	0,185	0,193	0,198	0,201	0,203	0,201	0,198	0,193	0,185	0,175	0,163	0,148	0,131	0,113	0,091
24	1	0,139	0,160	0,179	0,196	0,210	0,222	0,232	0,240	0,246	0,249	0,250	0,249	0,246	0,240	0,232	0,222	0,210	0,196	0,179	0,160	0,139

3.49. ábra: Kétváltozós függvény megjelenítéséhez szükséges értéktáblázat

Az A4:A24 blokkot x-szel, a B3:V3 blokkot y-nal nevesítettük, így mindegyik belső cellába ugyanaz a képlet kerül, mint amit a D6 cellában látunk.

Jelöljük ki az x és y értékeket is tartalmazó A3:V24 blokkot és a **Beszúrás/Egyéb diagram**-nál a **Felületet** válasszuk. Az aktív diagram vásznán másodlagos egérgombbal válasszuk a **Forgatás** menüpontot, és a forgatási szöget ízlésünk szerint beállíthatjuk.

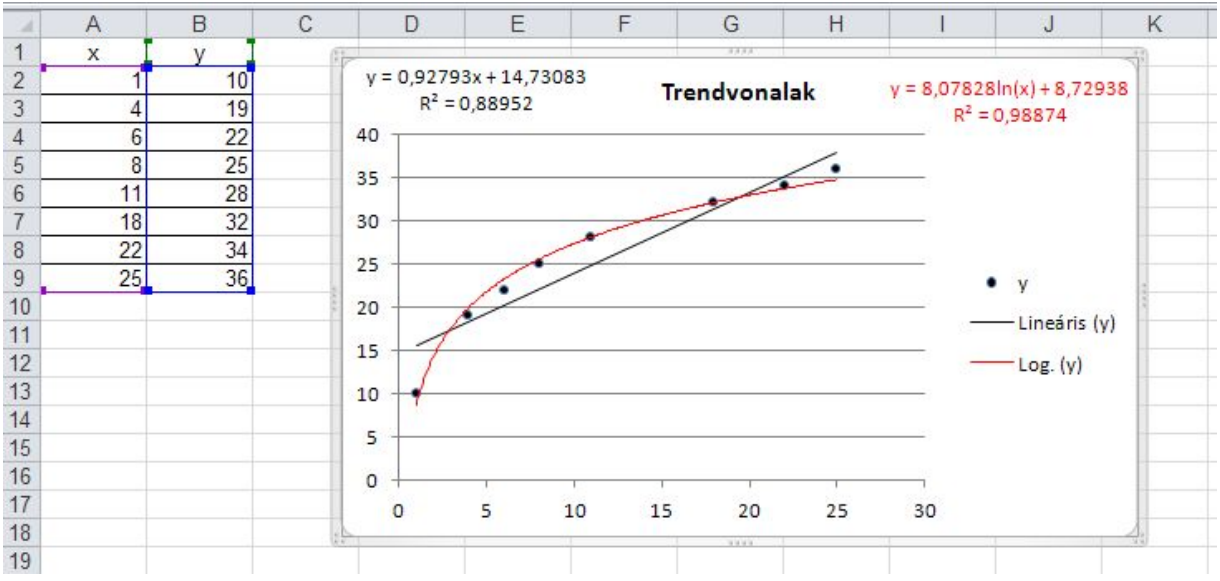


3.50. ábra: Kétváltozós függvény megjelenítése forgatással

3.6.4. Trendvonalak

Gyakori feladat az, hogy a mérési adatsorainkhoz a feladat fizikai, gazdasági törvényszerűségeinek megfelelő függvényt illesszünk. Az illesztett függvényről többnyire nem követeljük meg azt, hogy minden mérési pontra illeszkedjék (interpoláció esete), hanem csak azt, hogy összességében a lehető legkisebb hibával közelítse a mérési adatpontjainkat. Az Excel néhány egyszerűbb típusfüggvényre ezt az illesztést azonnali szolgáltatásként nyújtja, amit az Excel **trendvonalnak** nevez.

A trendvonal szolgáltatáshoz először el kell készíteni az $(x; y)$ pontpárjaink pontdiagramját. Aktív diagram mellett kérhető a trendvonal szolgáltatás a **Diagram eszközök/Elrendezés/Trendvonal** menüben. Itt mindig célszerű a **További trendvonal beállításokat** választani. A kiválasztott típushoz további szolgáltatásokat is kérhetünk, mint például a képlet, vagy R-négyzet megjelenítését, vagy a megadott x intervallumon túli folytatást is.

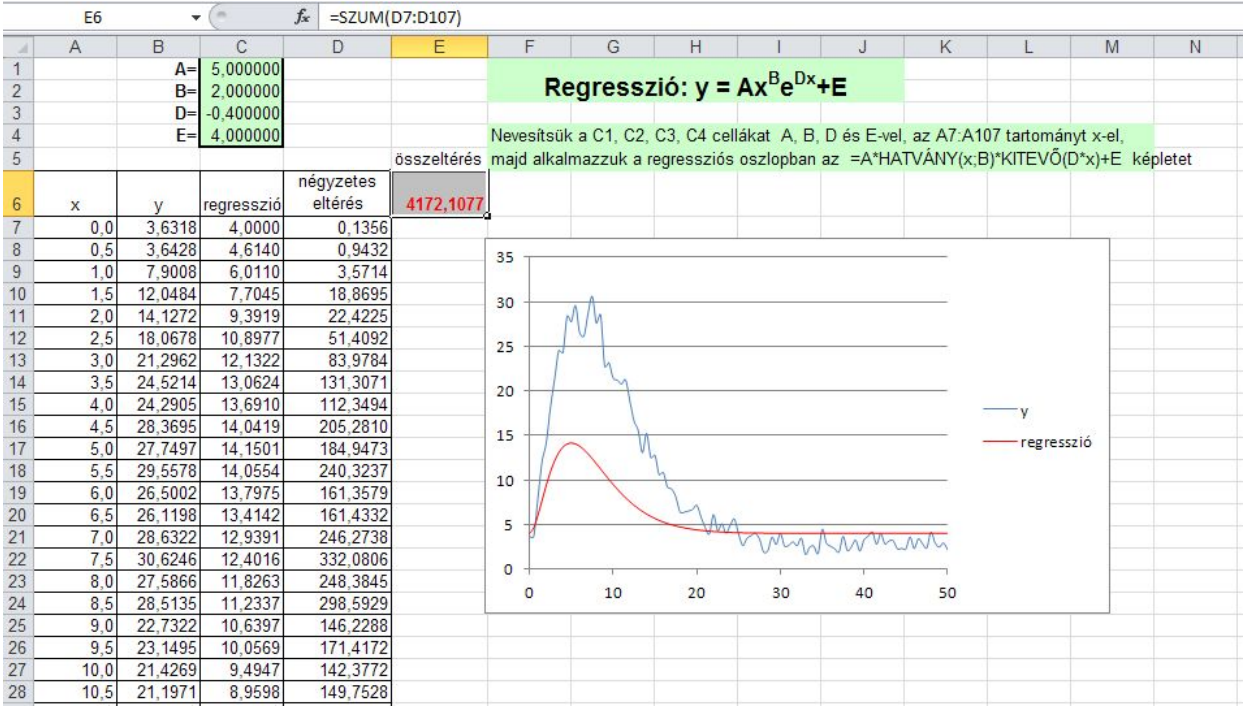


3.51. ábra: Trendvonalak a pontdiagramon

Az R-négyzet, az ún. determinációs együttható a közelítés jóságát jellemzi. Ez valójában az eredeti és a becsült y értékpárok korrelációs együtthatójának a négyzete. Minél közelebb van 1-hez, annál jobb a közelítés. A grafikonra kikért információk helye, színe, betűtípusa, a számok tizedes jegyeinek száma, stb. utólag is hangolható.

3.6.5. Nemlineáris regresszió, paraméterbecslés

Az Excel trendvonal szolgáltatásai csak az egyszerűbb függvényekkel való közelítésre alkalmasak. Összetettebb, több paraméterrel rendelkező függvénnyel való közelítésre a legkisebb négyzetek módszerét alkalmazva a Solver ad lehetőséget.



x	y	regresszió	négyzetes eltérés
0,0	3,6318	4,0000	0,1356
0,5	3,6428	4,6140	0,9432
1,0	7,9008	6,0110	3,5714
1,5	12,0484	7,7045	18,8695
2,0	14,1272	9,3919	22,4225
2,5	18,0678	10,8977	51,4092
3,0	21,2962	12,1322	83,9784
3,5	24,5214	13,0624	131,3071
4,0	24,2905	13,6910	112,3494
4,5	28,3695	14,0419	205,2810
5,0	27,7497	14,1501	184,9473
5,5	29,5578	14,0554	240,3237
6,0	26,5002	13,7975	161,3579
6,5	26,1198	13,4142	161,4332
7,0	28,6322	12,9391	246,2738
7,5	30,6246	12,4016	332,0806
8,0	27,5866	11,8263	248,3845
8,5	28,5135	11,2337	298,5929
9,0	22,7322	10,6397	146,2288
9,5	23,1495	10,0569	171,4172
10,0	21,4269	9,4947	142,3772
10,5	21,1971	8,9598	149,7528

3.52. ábra: Nemlineáris többparaméteres regresszió előkészítése

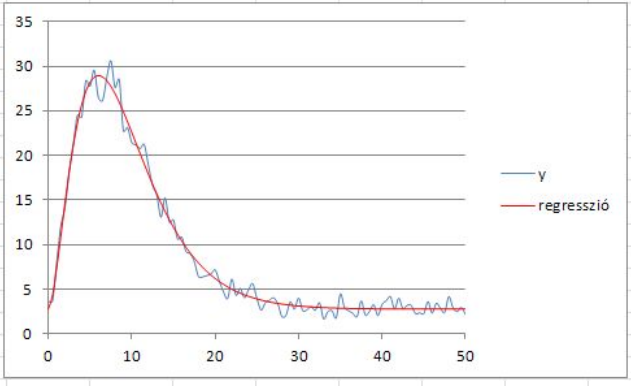
Ekkor a következőképp járunk el. Legyen adott az *x* és *y* összetartozó értékpárok sorozata. A probléma jellegéből adódóan tudjuk, hogy milyen típusú függvényt kell illesztenünk és a paraméterek szóba jöhető értékintervallumát is ismerjük. Ez utóbbiból választunk kezdő paraméter értékeket, majd ezek és a képlet segítségével becslést adunk az *y* értékekre. A tényleges és becsült *y* értékek persze jócskán eltérhetnek.

A 3.52. ábrán egy speciálisan gerjesztett, de időben csillapodó rendszer időbeli viselkedésének paramétereit becsüljük a mért adatokból. Az A, B paraméterek elsősorban a gerjesztést jellemzik, a D csillapítási tényező a rendszertől függ és az E paraméter a végső és stabil egyensúlyi állapotot jellemzi.

	A	B	C	D	E
1		A=	6,182339		
2		B=	1,814638		
3		D=	-0,301223		
4		E=	2,751117		
5					összeltérés
6	x	y	regresszió	négyzetes eltérés	92,8546
7	0,0	3,6318	2,7511	0,7756	
8	0,5	3,6428	4,2629	0,3844	
9	1,0	7,9008	7,3255	0,3309	
10	1,5	12,0484	10,9634	1,1771	
11	2,0	14,1272	14,6574	0,2811	
12	2,5	18,0678	18,1051	0,0014	
13	3,0	21,2962	21,1375	0,0252	
14	3,5	24,5214	23,6716	0,7221	
15	4,0	24,2905	25,6807	1,9326	
16	4,5	28,3695	27,1747	1,4276	
17	5,0	27,7497	28,1862	0,1906	
18	5,5	29,5578	28,7609	0,6350	
19	6,0	26,5002	28,9510	6,0064	
20	6,5	26,1198	28,8109	7,2419	
21	7,0	28,6322	28,3939	0,0568	
22	7,5	30,6246	27,7504	8,2614	
23	8,0	27,5866	26,9268	0,4352	
24	8,5	28,5135	25,9650	6,4950	
25	9,0	22,7322	24,9016	4,7065	
26	9,5	23,1495	23,7687	0,3834	
27	10,0	21,4269	22,5937	1,3614	
28	10,5	21,1971	21,3994	0,0409	

Regresszió: $y = Ax^B e^{Dx} + E$

Nevesítsük a C1, C2, C3, C4 cellákat A, B, D és E-vel, az A7:A107 tartományt x-el, majd alkalmazzuk a regressziós oszlopban az =A*HATVÁNY(x;B)*KITEVŐ(D*x)+E képletet



3.53. ábra: Nemlineáris többparaméteres regresszió a paraméterek meghatározása után

Elkészítjük és szebbé formáljuk az (x, y) ill. (x, regresszió) diagramot. Kiszámítjuk az eltérések négyzetösszegét, és ezt az egyetlen értéket minimalizáljuk a Solver segítségével oly módon, hogy a paraméterek értékeit változtatjuk meg. Ezek lesznek a változó cellákban (3.52. ábra).

A Solveres pontosítás után a regressziós függvényünk igen jól fog illeszkedni (3.53. ábra).

Önellenőrzés

1. Ábrázoljuk a következő függvény grafikonját!

$$f(x) = a(x-b)e^{-d(x-b)^2} \quad a,b,d \text{ pozitív számok}$$

Helyezzünk fel a grafikonra nevezetes pontokat!
Lépések:
Az a, b, d paramétertáblát helyezzük el a G1:G3 cellatartományban és nevet is rendeljünk hozzá! (Induló értékek: $a = 10$; $b = 1$; $d = 0{,}5$.)
Az A oszlopban x fejléccel helyezzük el a [-5; 5] intervallumbeli $\Delta x = 0{,}1$ lépésközű értéksorozatot.
A B oszlopban f(x) fejléccel pedig az x értékekhez tartozó függvényértékeket helyezzük el! (számformátum 10 tizedes jeggyel)
Az elkészített grafikonon az x tartomány skálázását korrigáljuk [-5; 5]-re!
Helyezzük el a C oszlopban f'(x) fejléccel az $f(x)$ függvény deriváltjának értékeit és bővítsük ki a grafikonunkat az új forrásadatokkal!

$$f'(x) = a\left[1 - 2d(x-b)^2\right]\ e^{-d(x-b)^2}$$

A diagram fejléce f(x)=a*(x-b)*exp(-d*(x-b)^2) legyen!
Változtassuk meg a paramétertábla értékeit, és figyeljük meg a grafikon változásait!
A görbéink nevezetes pontjainak (zérus-helyek, lokális szélsőérték pontok, inflexiós pontok) adatait a Solver segédeszközzel állapítjuk meg.

Lépések:

1. Minden nevezetes pontnál értéktáblázat készítése x, f(x) és f'(x) értékekhez az x adatot tartalmazó cellahivatkozással.
2. A grafikonon tapogatva leolvassuk a megfelelő x értéket. és beírjuk a megfelelő x cellába.
3. A Solver segítségével (Adatok/Elemzés menüben) a kiválasztott az f(x) ill. f'(x) cella értékét pontosítjuk (0, min, max, stb.) az x értékének módosíttatásával. (A Solver beállításaira ügyeljünk!)
4. A Tervezés/Adatok kijelölése menüben a kívánt pontot, vagy vonalat meghatározó pontpár adatait hozzáadjuk, majd ezek stílusát hangoljuk.

Tegyük fel a grafikonra az $f(x)$ maximumpontját és a függőleges rendezővonalat az x = 1,95 hely környékén!
A megoldás megtekintéséhez kattintson ide!

2. Ábrázoljuk a következő függvények grafikonját!

$$f(x) = \sin(Ax + B)\cos(Dx + E)$$
$$f'(x) = A\cos(Ax + B)\cos(Dx + E) - D\sin(Ax + B)\sin(Dx + E)$$

Ahol

A	B	D	E
1	3	1	6

Lépések:

1. Az A1:D2 cellákba gépeljük be a fenti táblázatot és a 2. sor számértékeit nevesítsük A, B, D, E-vel!
2. Készítsük el az A, B, C oszlopokban az x, f(x), f'(x) értékek fejléces értéktáblázatát a [2; 5,5] intervallumban 0,1-es lépésközzel. (Az f(x) és f'(x) értékek formátuma: 5 tizedes-jegy, az x értékeké 1 tizedesjegy.)

3. Ez alapján ábrázoljuk az f(x), f'(x) függvények grafikonját folytonos görbe vonallal. f(x) – sötétkék, f'(x) – lila vékony folytonos görbe vonal.
4. Határozzuk meg Solver-rel az $f(x)$ függvény [2; 5,5] intervallumbeli zérushelyeit, szélsőértékpontjait (ez utóbbit a deriváltfüggvény zérushelyeinek segítségével)!
5. 5-pontos piros teli pöttyként a zérushelyeket és barna teli pöttyként a szélsőértékpontokat tegyük fel az f(x) grafikonra!
6. A grafikon tengelyeinek formájára (hely, megjelenés, stb.) ügyeljünk!

A megoldás megtekintéséhez kattintson ide!

3. A következő (x; y) táblázattal adott pontokat ábrázoljuk összekötő nélküli pontdiagramon, majd a grafikonra tegyük fel a lineáris és másodfokú trendvonalakat képletekkel és a determinációs együttható értékével együtt!

3.2. táblázat. Pontsorozat trendvonalak készítéséhez

x	y
1	-7
3	-2
6	3
7	4
12	10
16	14

Lépések:
A kijelölt (x; y) táblázat mellett a **Beszúrás/Diagramok/Pont/Pont csak jelölőkkel** menüpontokkal létrehozzuk a pontdiagramot.

Aktív pontdiagram esetén a **Diagrameszközök/Elrendezés/Trendvonal/További trendvonal beállítások** menüben beállítjuk a típust, a feliratot, az előrejelzés (előre, vissza) értékeit, és bejelöljük az egyenlet, valamint az R-négyzet látszik jelölőnégyzetet.
Megadhatjuk a trendvonal színét és vonalstílusát is. Végezetül az egyenlet szövegdobozát a diagramvászon semleges területére húzzuk, majd kellő színűre állítjuk.
A megoldás megtekintéséhez kattintson ide!

17. LECKE

Statisztikai alkalmazások
Speciális matematikai problémák

3.7. Az Excel statisztikai eszközei

Az alap Excel számos matematikai statisztikában használt függvényt tud kezelni. A korábbi verziókhoz képest konzisztensebb függvényneveket vezettek be. Egyes statisztikai függvények neve módosult, így a tudományosan elfogadott meghatározással és az Excel többi függvényével is konzisztens lett. Az új függvénynevek pontosabban jellemzik a funkciót is. A régebbi verziókban létrehozott munkafüzetek a névváltozások ellenére is működnek, mivel az eredeti függvények megmaradnak a Kompatibilitás kategóriában. A nagyon hasonló nevű függvények a korábbi verziók miatt szerepelnek az új függvénynév mellett – például gammaln() és gammaln.pontos().

Mi most csak az Excel 2010 statisztikai függvényeit soroljuk fel (helytakarékossági okból a lista első felét közöljük).

3.3. táblázat. Az Excel 2010 statisztikai függvényei

ÁTL.ELTÉRÉS	Az adatpontoknak átlaguktól való átlagos abszolút eltérését számítja ki.
ÁTLAG	Argumentumai átlagát számítja ki.
ÁTLAGA	Argumentumai átlagát számítja ki (beleértve a számokat, szöveget és logikai értékeket).
ÁTLAGHA	A megadott feltételnek eleget tévő tartomány celláinak átlagát (számtani közepét) adja eredményül.
ÁTLAGHATÖBB	A megadott feltételeknek eleget tévő cellák átlagát (számtani közepét) adja eredményül.
BÉTA.ELOSZL	A béta-eloszlást számítja ki.
BÉTA.INVERZ	Adott béta-eloszláshoz kiszámítja a béta eloszlásfüggvény inverzét.
BINOM.ELOSZL	A diszkrét binomiális eloszlás valószínűségértékét számítja ki.
BINOM.INVERZ	Azt a legkisebb számot adja eredményül, amelyre a binomiális eloszlásfüggvény értéke nem kisebb egy adott határértéknél.

CSÚCSOSSÁG	Egy adathalmaz csúcsosságát számítja ki.
DARAB	Megszámolja, hogy argumentumlistájában hány szám található.
DARAB2	Megszámolja, hogy argumentumlistájában hány érték található.
DARABHATÖBB	Egy tartományban összeszámolja azokat a cellákat, amelyek eleget tesznek több feltételnek.
DARABTELI	Egy tartományban összeszámolja azokat a cellákat, amelyek eleget tesznek a megadott feltételnek.
DARABÜRES	Egy tartományban összeszámolja az üres cellákat.
ELŐREJELZÉS	Az ismert értékek alapján lineáris regresszióval becsült értéket ad eredményül.
EXP.ELOSZL	Az exponenciális eloszlás értékét számítja ki.
F.ELOSZL	Az F-eloszlás értékét számítja ki.
F.ELOSZLÁS.JOBB	Az F-eloszlás értékét számítja ki.
F.INVERZ	Az F-eloszlás inverzének értékét számítja ki.
F.INVERZ.JOBB	Az F-eloszlás inverzének értékét számítja ki.
F.PRÓB	Az F-próba értékét adja eredményül.
FERDESÉG	Egy eloszlás ferdeségét határozza meg.
FISHER	Fisher-transzformációt hajt végre.
GAMMA.ELOSZL	A gamma-eloszlás értékét számítja ki.
GAMMA.INVERZ	A gamma-eloszlás eloszlásfüggvénye inverzének értékét számítja ki.
GAMMALN	A gamma-függvény természetes logaritmusát számítja ki.
GAMMALN.PONTOS	A gamma-függvény természetes logaritmusát számítja ki.
GYAKORISÁG	A gyakorisági vagy empirikus eloszlás értékét függőleges tömbként adja eredményül.
HARM.KÖZÉP	Argumentumai harmonikus átlagát számítja ki.
HIPGEOM.ELOSZLÁS	A hipergeometriai eloszlás értékét számítja ki.

INVERZ.FISHER	A Fisher-transzformáció inverzét hajtja végre.
KHINÉGYZET.ELOSZLÁS	A bétaeloszlás sűrűségfüggvényének értékét számítja ki.
KHINÉGYZET.ELOSZLÁS.JOBB	A khi-négyzet-eloszlás egyszélű valószínűségértékét számítja ki.
KHINÉGYZET.INVERZ	A bétaeloszlás sűrűségfüggvényének értékét számítja ki.
KHINÉGYZET.INVERZ.JOBB	A khi-négyzet-eloszlás egyszélű valószínűségértékének inverzét számítja ki.
KHINÉGYZET.PRÓBA	Függetlenségvizsgálatot hajt végre.
KICSI	Egy adathalmaz k-adik legkisebb elemét adja meg.
KORREL	Két adathalmaz korrelációs együtthatóját számítja ki.
KOVARIANCIA.M	A statisztikai minta kovarianciáját adja eredményül, amely az egyes adatpont-párok átlagtól való eltérése szorzatának várható értéke.
KOVARIANCIA.S	A kovarianciát, azaz a páronkénti eltérések szorzatának átlagát számítja ki.
KVARTILIS.KIZÁR	Az adathalmaz kvartilisét számítja ki a 0..1 tartományba eső percentilis értékek alapján (a végpontok nélkül).
...	...

Forrás: Microsoft Excel súgója (F1), keresőszó: statisztikai függvények

A fenti függvények közül több felhasználható általános célra is. A gazdaságelemzési és statisztikai szakmai kurzusok szívesen használják ezeket, a használatukat ott fogják ismertetni.

3.7.1. Az Analysis ToolPak

Mi most az Excel az **Analysis ToolPak** bővítménye segítségével elérhető összetett statisztikai szolgáltatásaiból tekintünk át néhányat. Az elemzéshez szükséges adatok és paraméterek megadása után az Analysis ToolPak eszköz a megfelelő statisztikai vagy mérnöki makró-függvényekkel végrehajtja a számításokat, majd megjeleníti az eredményeket egy eredménytáblában. Néhány eljárás az eredménytábla mellett diagramot is készít. Az elemző eszközök megjelenítéséhez az **Adatok** lap **Elemzés** csoportjában az **Adatelemzés** parancsot kell kiválasztani. Ha az **Adatelemzés** parancs nem látható a menüben, akkor be kell tölteni az Analysis ToolPak

bővítményt.

Az **Adatelemzés** szolgáltatásai névsorban:

1. Exponenciális simítás;
2. Fourier-analízis;
3. Hisztogram;
4. Kétmintás f-próba a szórásnégyzetre;
5. Kétmintás z-próba a várható értékre;
6. Korrelációanalízis;
7. Kovariancia-analízis;
8. Leíró statisztika;
9. Mintavétel;
10. Mozgóátlag;
11. Rangsor és százalékos rangsor;
12. Regresszió-analízis;
13. T-próba;
    Kétmintás, párosított t-próba a várható értékre;
    Kétmintás t-próba egyenlő szórásnégyzeteknél;
    Kétmintás t-próba nem egyenlő szórásnégyzeteknél;

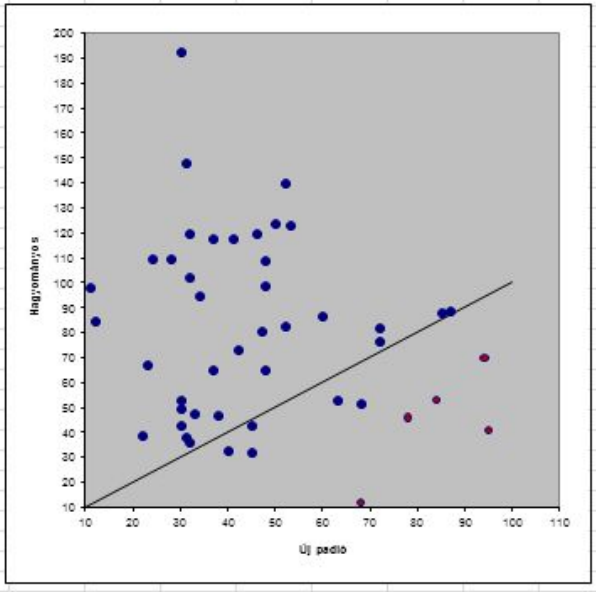
14. Varianciaanalízis;
    Egytényezős varianciaanalízis;
    Kéttényezős, ismétléses varianciaanalízis;
    Kéttényezős varianciaanalízis ismétlések nélkül;
15. Véletlenszám-generálás.

Statisztikai elemzés

Mindezek után nézzünk egy példát a statisztikai elemzésre ([18])!

Egy kórházban új antibakteriális padlót építettek be bizonyos helyekre. Az ikerfolyóson teljesen azonos elrendezésben hagyományos műpadló van. Azonos időben és helyeken ugyanazon működési körülmények között a padlókról mintákat vettek és a laboratóriumban a kórokozók számát azonos egységre vetítve megállapították. Az alábbi táblázat mutatja a felmért adatok egy részét és a pontdiagram illusztrálja az összetartozó adatpárokat:

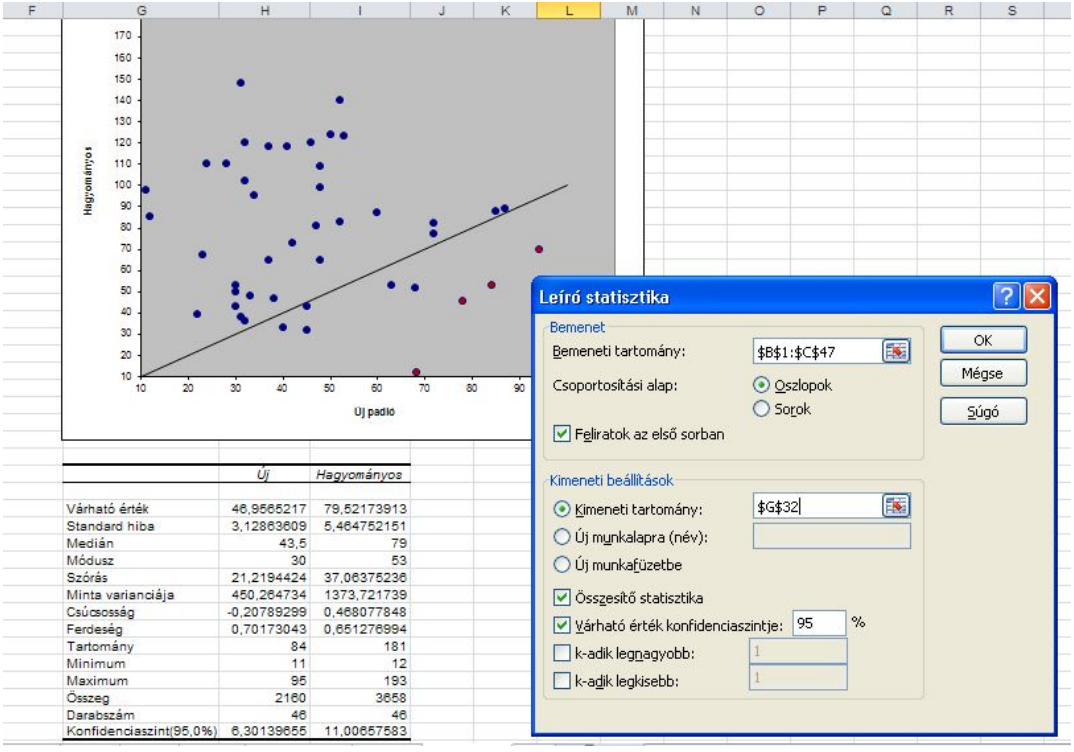
Eset	Új padló	Hagyományos padló	Hely	Idő
1	30	50	Bejárat	2011.09.30
2	30	43	Bejárat	2011.09.30
3	45	32	Bejárat	2011.09.30
4	32	38	Bejárat	2011.09.30
5	24	110	Bejárat	2011.10.07
6	31	38	Bejárat	2011.10.07
7	32	102	Bejárat	2011.10.07
8	30	53	Bejárat	2011.10.07
9	68	52	Bejárat	2011.12.02
10	95	41	Bejárat	2011.12.02
11	37	118	Mosdó előtt	2011.09.30
12	48	109	Mosdó előtt	2011.09.30
13	63	53	Mosdó előtt	2011.09.30
14	60	87	Mosdó előtt	2011.09.30
15	22	39	Mosdó előtt	2011.10.07
16	45	43	Mosdó előtt	2011.10.07
17	38	47	Mosdó előtt	2011.10.07
18	37	65	Mosdó előtt	2011.10.07
19	72	82	Mosdó előtt	2011.12.02
20	84	53	Mosdó előtt	2011.12.02
21	48	99	Növérpult/ szekrény	2011.09.30
22	50	124	Növérpult/ szekrény	2011.09.30
23	52	140	Növérpult/ szekrény	2011.09.30
24	31	148	Növérpult/ szekrény	2011.09.30
25	23	67	Növérpult/ szekrény	2011.10.07
26	33	48	Növérpult/ szekrény	2011.10.07
27	68	12	Növérpult/ szekrény	2011.12.02
28	72	77	Növérpult/ szekrény	2011.12.02
29	87	89	Betegágy mellett	2011.09.30
30	85	88	Betegágy mellett	2011.09.30
31	78	46	Betegágy mellett	2011.09.30
32	94	70	Betegágy mellett	2011.09.30
33	53	123	Betegágy mellett	2011.10.07
34	30	193	Betegágy mellett	2011.10.07
35	32	120	Ablak előtt világos rész	2011.09.30
36	41	118	Ablak előtt világos rész	2011.09.30
37	28	110	Ablak előtt világos rész	2011.09.30
38	34	95	Ablak előtt világos rész	2011.09.30
39	12	85	Ablak előtt világos rész	2011.10.07
40	11	98	Ablak előtt világos rész	2011.10.07
41	40	33	Ablak előtt világos rész	2011.12.02
42	46	120	Ablak előtt világos rész	2011.12.02
43	48	65	Betegágyak között	2011.09.30



3.54. ábra: Két valószínűségi változó megfigyelt értékeinek pontdiagramja

A grafikonon elhelyezett és a (10; 10) illetve (100; 100) pontokat összekötő vonal felett azokat az eseteket látjuk, amikor a hagyományos padlón több volt a kórokozó. Ezen esetek száma láthatóan több. Kérdés: van-e statisztikailag lényeges különbség a két adatoszlop között?

Az **Adatelemzés/Leíró statisztika** szolgáltatással nem kell a statisztikai függvények értékeit egyesével meghatározni, hanem táblázatos formában egyszerre megkapjuk ezeket. A 3.55. ábrán a szolgáltatás paraméterezését és a kapott eredményeket látjuk. Az eredménytáblázatot az aktuális füzetlapra kértük, és ennek a bal felső sarkának a pozícióját kellett csak megadni.



	Új	Hagyományos
Várható érték	46,9565217	79,52173913
Standard hiba	3,12863609	5,464752151
Medián	43,5	79
Módusz	30	53
Szórás	21,2194424	37,06375236
Minta varianciája	450,264734	1373,721739
Csúcsosság	-0,20789299	0,468077848
Ferdeség	0,70173043	0,651276994
Tartomány	84	181
Minimum	11	12
Maximum	95	193
Összeg	2160	3658
Darabszám	46	46
Konfidenciaszint(95,0%)	6,30139655	11,00657583

3.55. ábra: Két valószínűségi változó megfigyelt értékeinek leíró statisztikája

Mindkét adatoszlopra az jellemző, hogy

1. A várható érték és a medián (50%-os vágási hely) becsült értéke közel azonos;
2. A csúcsosság és ferdeség becsült értékei 0-hoz közeli számok;
3. A várható érték körüli szórásnyi intervallumban az adatok kb. 68%-a van.

Ezek alapján az adataink normális eloszlásból való származását feltételezhetjük. Ezt természetesen speciális statisztikai alkalmazásokkal (pl. SPSS) ellenőriztethetjük. A normalitás feltételezése után a **Kétmintás párosított t-próbával** eldönthetjük a várható értékek egyezését, illetve a lényeges különbözőségét. Ennek a paraméterezését látjuk a 3.56. ábrán.

Kétmintás párosított t-próba a várható értékre

	Új	*Hagyományos*
Várható érték	46,95652	79,52173913
Variancia	450,2647	1373,721739
Megfigyelések	46	46
Pearson-féle korreláció	-0,18493	
Feltételezett átlagos eltérés	0	
df	45	
t érték	-4,80276	
P(T<=t) egyszélű	8,84E-06	
t kritikus egyszélű	1,679427	
P(T<=t) kétszélű	1,77E-05	
t kritikus kétszélű	2,014103	

3.56. ábra: Kétmintás párosított t-próba és paraméterezése

Konklúziók:

A két minta között lényeges eltérés van. Ez több dologból is látszik. Egyrészt az átlagok (a várható értékek becslései) lényegesen távol vannak egymástól, hiszen a 95%-os megbízhatósági szintű konfidencia intervallumaik (46,96 $\pm$ 6,30 és 79,52 $\pm$ 11,01) nem fednek egymásba. Másrészt a páros t-próba alapján a várható értékek egyezését 95%-os megbízhatósági szinten el kell vetni, hiszen a próba t értékénél extrémebb esetek valószínűsége 0,05-nél kisebb, gyakorlatilag 0 (lásd sárga hátterű cellák).

Többváltozós lineáris regresszió

A következő példánkban a **hisztogram**, a **korreláció**- és a többváltozós lineáris **regresszió-elemzést** mutatjuk be. Az autok.xlsx munkafüzet több autótípus műszaki paramétereit és eladási árát tartalmazza a 3.4. táblázat szerint.

3.4. táblázat. Autók adatai többváltozós lineáris regressziós feladathoz

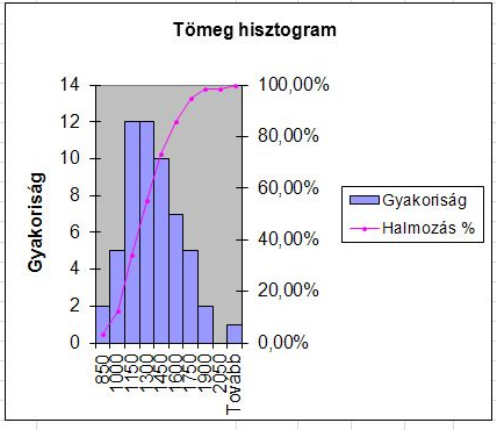
Változó	**Megnevezés**
kobcm	a gépkocsi hengerűrtartalma (cm^3)
telj	teljesítmény (LE)
nyom	maximális nyomaték (Nm)
gyors	gyorsulás (m/sec^2)
vmax	maximális sebesség (km/h)
tomeg	tömeg (kg)
hossz	hosszúság (mm)
szeles	szélesség (mm)
csomag	a csomagtartó térfogata (liter)
fogy	fogyasztás (liter/100 km)
ar	az új autó alapára (ezer Ft)

Kérdéseink a következők:

1. Hogyan ábrázolható az autók tömegeloszlása?
2. Milyen erős kapcsolatban vannak az egyes műszaki paraméterek az árral?
3. Milyen lineáris regressziós képlettel lehet az árat jól megbecsülni?
4. Mely autótípusok a túl-, illetve alulárazottak?

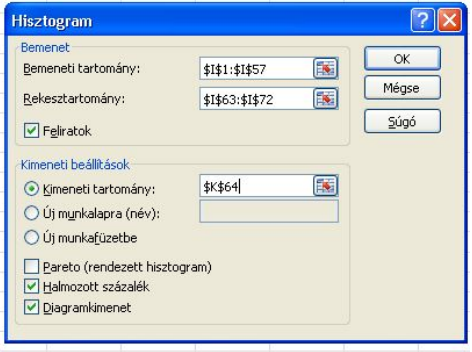
Első lépésként az alapadat táblázatunk minden numerikus oszlopát kiegészítjük az átlag, szórás, min. és max. adatokkal. A két utóbbi érték a hisztogram készítéséhez elengedhetetlenül szükséges. Mivel a tömeg adatok között 820 a minimális és 2075 a maximális, ezért a mintaterjedelmet 850-től kezdve 2050-ig 150 távolságonként elhelyezett osztópontok segítségével részintervallumokra bontjuk (fejléces **rekesztartomány,** az ábrán sárga hátterű). Ezután az **Adatelemzés / Hisztogram** szolgáltatását kérhetjük, amelynek paraméterezését és az elkészült eredményt a **Hisztogram** objektumainak kellő hangolása után a 3.57. ábra mutatja.

	A	B	D	E	F	G	H	I	J	K	L	M	N
1	sorsz.	Tipus	Kobcm	Telj	Nyom	Gyors	Vmax	Tomeg	Hossz	Szeles	Csomag	Fogy	Ar
54	53	VW Polo Tdi	1422	75	195	2,153	170	1034	3743	1632	245	4,9	3546
55	54	Fiat Multipla JTD	1910	105	200	2,240	170	1300	3995	1870	430	7,2	5395
56	55	Renault Scenic dTi	1870	100	200	2,187	174	1290	4169	1698	410	6,5	4640
57	56	Rover 75 V6	2497	175	240	3,157	220	1370	4745	1780	430	12,1	10875
58		átlag	1930,18	119,07	204,57	2,47	187,61	1300,09	4344,00	1723,52	400,89	8,28	6116,61
59		szórás	486,77	42,24	74,24	0,60	27,76	274,89	381,29	58,19	180,54	1,89	3176,19
60		min	1191	60	110	1,362	143	820	3500	1620	115	3,1	2150
61		max	2995	250	400	4,085	250	2075	5040	1870	1290	12,8	13890



Tömeg
850
1000
1150
1300
1450
1600
1750
1900
2050

Tömeg	Gyakoriság	Halmozás %
850	2	3,57%
1000	5	12,50%
1150	12	33,93%
1300	12	55,36%
1450	10	73,21%
1600	7	85,71%
1750	5	94,64%
1900	2	98,21%
2050	0	98,21%
Tovább	1	100,00%
	56	



3.57. ábra: A tömegeloszlás hisztogramja előkészítéssel és a paraméterezése

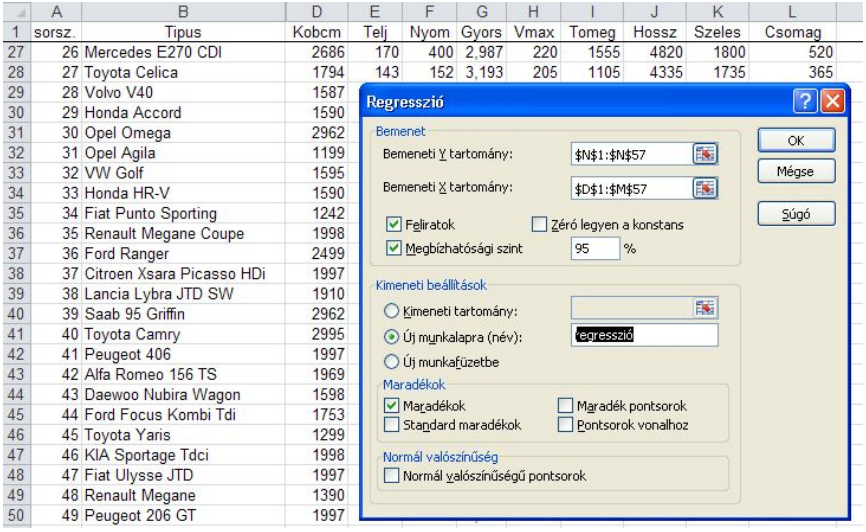
Itt a **Halmozás** görbe az adatsor ún. tapasztalati-eloszlásfüggvényét ábrázolja.

A változók páronkénti lineáris kapcsolatának erősségét az ún. korrelációs mátrix mutatja, amely szimmetrikus. Éppen ezért az Excel csak az alsó háromszög alakú részt jeleníteni meg. A **Korrelációanalízis** eredményét új füzetlapra kértük (3.58. ábra).

	A	B	C	D	E	F	G	H	I	J	K
1		*Kobcm*	*Telj*	*Nyom*	*Gyors*	*Vmax*	*Tomeg*	*Hossz*	*Szeles*	*Csomag*	*Fogy*
2	Kobcm	1									
3	Telj	0,750436	1								
4	Nyom	0,8569	0,712138	1							
5	Gyors	0,323246	0,79807	0,357642	1						
6	Vmax	0,427314	0,836641	0,454459	0,924247	1					
7	Tomeg	0,810403	0,498836	0,767362	-0,0613	0,074022	1				
8	Hossz	0,77785	0,568661	0,681689	0,11042	0,284192	0,813907	1			
9	Szeles	0,69409	0,524396	0,6329	0,108673	0,223071	0,788961	0,685854	1		
10	Csomag	0,382567	0,307708	0,363363	0,076291	0,181909	0,483343	0,482642	0,583473	1	
11	Fogy	0,587514	0,628174	0,384842	0,353058	0,30603	0,499987	0,581196	0,49378	0,265225	1
12	Ar	0,848481	0,893494	0,850877	0,558585	0,660903	0,691331	0,658447	0,674627	0,414921	0,587681

3.58. ábra: A műszaki adatok és az ár korrelációs mátrixa

Az ár és a műszaki adatok kapcsolatát jellemző értékeket az utolsó sorban találjuk. Itt pirossal az igen szoros kapcsolatokat, feketével a leggyengébb kapcsolatokat jeleztük.



	A	B	D	E	F	G	H	I	J	K	L
1	sorsz.	Tipus	Kobcm	Telj	Nyom	Gyors	Vmax	Tomeg	Hossz	Szeles	Csomag
27	26	Mercedes E270 CDI	2686	170	400	2,987	220	1555	4820	1800	520
28	27	Toyota Celica	1794	143	152	3,193	205	1105	4335	1735	365
29	28	Volvo V40	1587								
30	29	Honda Accord	1590								
31	30	Opel Omega	2962								
32	31	Opel Agila	1199								
33	32	VW Golf	1595								
34	33	Honda HR-V	1590								
35	34	Fiat Punto Sporting	1242								
36	35	Renault Megane Coupe	1998								
37	36	Ford Ranger	2499								
38	37	Citroen Xsara Picasso HDi	1997								
39	38	Lancia Lybra JTD SW	1910								
40	39	Saab 95 Griffin	2962								
41	40	Toyota Camry	2995								
42	41	Peugeot 406	1997								
43	42	Alfa Romeo 156 TS	1969								
44	43	Daewoo Nubira Wagon	1598								
45	44	Ford Focus Kombi Tdi	1753								
46	45	Toyota Yaris	1299								
47	46	KIA Sportage Tdci	1998								
48	47	Fiat Ulysse JTD	1997								
49	48	Renault Megane	1390								
50	49	Peugeot 206 GT	1997								

3.59. ábra: Sokváltozós lineáris regresszió paraméterezése

Következő lépésben a regressziós elemzést végezzük el, amelynek paraméterezése a 3.59. ábrán látható.

Az új lapon kapott összesítő eredmények táblázatából leolvashatjuk a regresszió jóságát jellemző R-négyzet (determinációs együttható) értékét, a változónkénti együtthatók értékét (koefficiensek) és azt, hogy ez az együttható tekinthető-e 0-nak, azaz elhanyagolható összetevője a regressziónak (sárga hátterű változók). Ha egy változó jelentéktelen szerepű, akkor az együtthatójának konfidencia-intervalluma lefedi a 0 értéket.

	A	B	C	D	E	F	G	H	I
1		ÖSSZESÍTŐ TÁBLA							
2									
3		*Regressziós statisztika*							
4		r értéke	0,963173588						
5		r-négyzet	0,927703361						
6		Korrigált r-négyzet	0,911637441						
7		Standard hiba	944,148757						
8		Megfigyelések	56						
9									
10		VARIANCIAANALÍZIS							
11			*df*	*SS*	*MS*	*F*	*F szignifikanciája*		
12		Regresszió	10	514735812	51473581,2	57,74355704	2,60244E-22		
13		Maradék	45	40113759,39	891416,8754				
14		Összesen	55	554849571,4					
15									
16			*Koefficiensek*	*Standard hiba*	*t érték*	*p-érték*	*Alsó 95%*	*Felső 95%*	
17		Tengelymetszet	-8079,680348	6570,2199	-1,229742759	0,225184273	-21312,78232	5153,421623	
18		Kobcm	0,376792561	0,700532579	0,537865865	0,593319981	-1,034152459	1,787737581	
19		Telj	45,99004068	15,07197838	3,051360579	0,003813189	15,6335184	76,34656297	
20		Nyom	16,36006135	4,29339736	3,81051647	0,000418331	7,712715307	25,0074074	
21		Gyors	-2535,401272	904,6872066	-2,802516996	0,007453181	-4357,534814	-713,2677302	
22		Vmax	44,01114671	17,64006808	2,494953336	0,01633422	8,482226326	79,5400671	
23		Tomeg	0,043408311	1,726245185	0,025146087	0,980049622	-3,433427915	3,520244537	
24		Hossz	-1,850757651	0,736474328	-2,512996829	0,015621487	-3,334093069	-0,367422233	
25		Szeles	4,793737804	3,996650032	1,199438972	0,236634789	-3,255928451	12,84340406	
26		Csomag	0,843722032	0,903349239	0,933993185	0,35529229	-0,975716704	2,663160768	
27		Fogy	244,5262661	125,5382459	1,947822867	0,057690688	-8,320736708	497,373269	

3.60. ábra: Sokváltozós lineáris regresszió összesítő táblázata

Az összesítő táblázatok alján a **Maradék táblát** látjuk (3.61. ábra).

	A	B	C	D
30				
31	MARADÉK TÁBLA			
32				
33	*Megfigyelés*	*Becsült Ar*	*Maradékok*	
34	1	2930,796431	569,2035691	
35	2	9160,28331	202,7166903	
36	3	3062,377348	-563,3773475	
37	4	6975,375686	-775,3756856	
38	5	12124,71627	225,2837301	
39	6	3728,194223	-388,194223	
40	7	3388,769368	410,2306319	
41	8	9323,842828	433,1571723	
42	9	8075,876048	2594,123952	

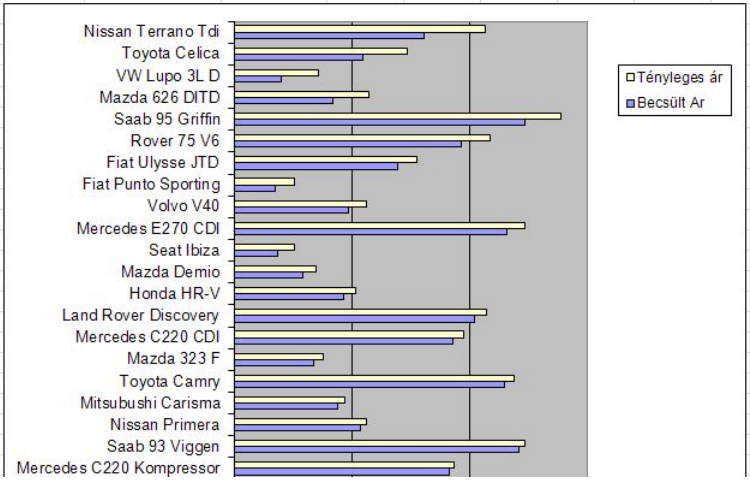
3.61. ábra: Sokváltozós lineáris regresszió maradék táblázata

Itt érdemes

1. A megfigyelési sorszámokat az alapadat-táblából idemásolt márkanevekre cserélni;
2. Kiegészíteni a táblázatot a Becsült Ár + Maradékok összegével, ami a tényleges árat eredményezi;
3. Egy megfelelően tájolt oszlopdiagramon a tényleges árat a becsült árral együtt szerepeltetni az eltérések csökkenő sorrendjében.

Ekkor a táblázat elején az alulárazott, az oszlopdiagram elején pedig a túlárazott gépkocsi márkák szerepelnek, és az itteni eredményekből azt a következtetést vonhatjuk le, hogy a műszaki paramétereken túl a márkanévnek és a divatos típusnak is nagy szerepe van az árképzésben (3.62. ábra).

	A	B	C	D	E
28					
29					
30					
31		Maradék szerint rendezve			
32					
33		*Megfigyelés*	*Becsült Ar*	*Maradékok*	*Tényleges ár*
34	35	Renault Megane Coupe	6768,189826	-1568,189826	5200
35	36	Ford Ranger	6334,104921	-1419,104921	4915
36	50	Mazda B2500 TD	5964,877264	-1226,877264	4738
37	25	Volvo V70 T5	14170,98366	-1180,983656	12990
38	41	Peugeot 406	7166,126573	-1046,126573	6120
39	37	Citroen Xsara Picasso HDi	5526,348425	-996,3484245	4530
40	18	Renault Clio	3437,503789	-907,503789	2530
41	54	Fiat Multipla JTD	6293,88935	-898,8893503	5395
42	4	Opel Astra Coupe	6975,375686	-775,3756856	6200
43	10	Lada 111	2882,866774	-732,8667741	2150
44	21	VW Bora	7812,941955	-725,9419547	7087
45	49	Peugeot 206 GT	5685,035593	-695,0355928	4990
46	44	Ford Focus Kombi Tdi	4571,465826	-692,4658264	3879
47	30	Opel Omega	11777,47183	-592,4718256	11185
48	12	Renault Laguna Dci	6387,266633	-587,2666327	5800
49	3	Seat Cordoba Sportline	3062,377348	-563,3773475	2499
50	32	VW Golf	4395,277779	-499,2777791	3896
51	13	Skoda Octavia TDI4x4	5041,498505	-493,4985049	4548



3.62. ábra: Sokváltozós lineáris regresszió kibővített maradék táblázata és oszlopdiagramja

3.8. Az Excel alkalmazása speciális matematikai problémák megoldására

3.8.1. Fourier-analízis

Lineáris rendszerek és periodikus jelekből vett mintavétel (idősor) elemzésére használható, az adatokat a gyors Fourier-transzformáció (Fast Fourier Transform - FFT) módszerével a komplex frekvenciatartományba transzformálja. Itt pl. sávszűréssel kizárjuk a nem kívánt frekvenciájú komponenseket. Ezután az inverz transzformáció is elvégezhető, amelynek során a transzformált és szűrt adatokból visszakapjuk az eredeti jel megszűrt összetevőjét.

Az időfüggvényből 2 hatvány (pl. $n = 512$) elemszámú mintát veszünk egyenlő lépésközzel. A mintát τ időközönként vesszük, és ez $1/\tau$ mintavételi frekvenciát jelent. A frekvenciaspektrum $F = 1/\tau$ frekvenciánként ismétlődik. Egy periódusa n elemű minta esetén $\Delta f = F/n$ frekvencia-felbontású spektrumot jelent. Az időablak

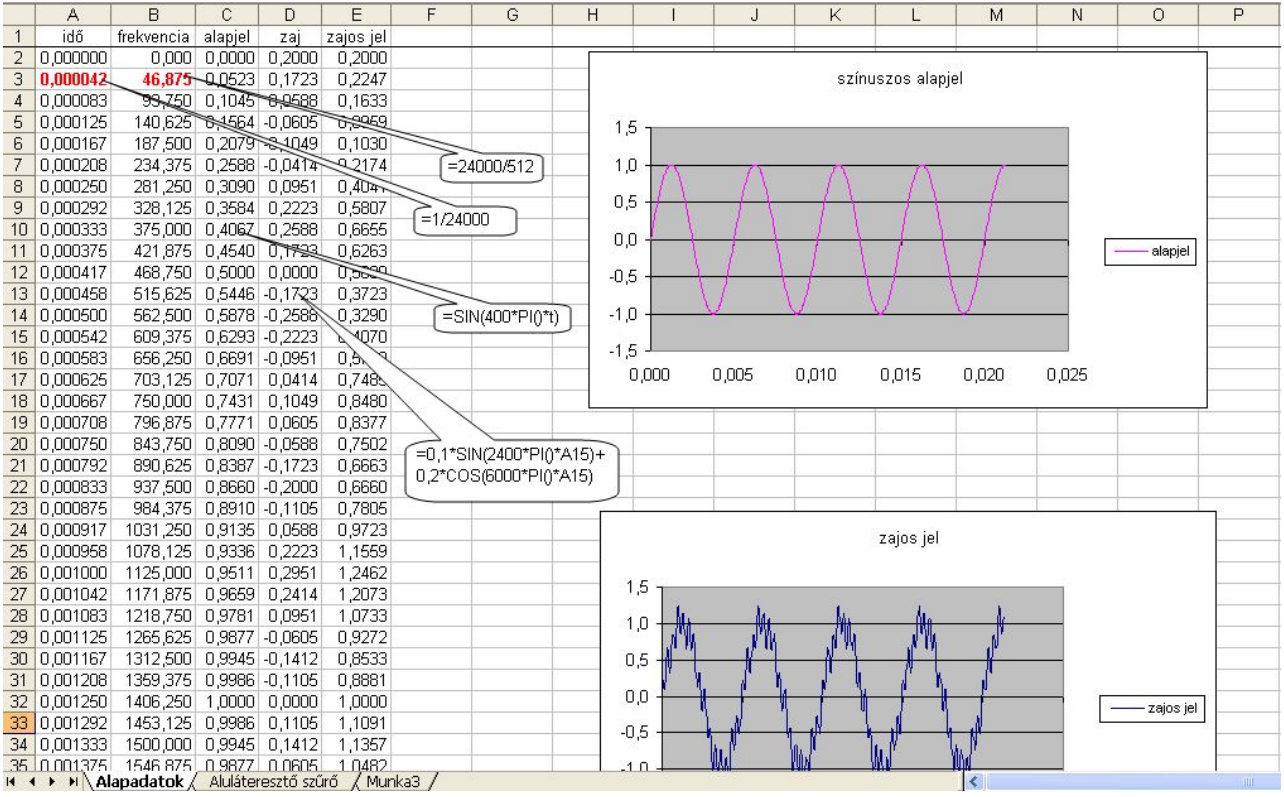
$\Delta T = n\tau$ (feltételezzük, hogy a jel eszerint periodikus). Legyen most $F = 24000$.

Ekkor $\Delta f = 24000/512 = 46{,}874$ és $\tau = 1/24000 = 0{,}000042$ és $\Delta T = 0{,}021333$.

Mintánkban egy 200 Hz-es szinuszos időfüggvényre két magasabb frekvenciájú „zavart” ültetünk:

1. Egy 0,1 amplitúdójú 1200 Hz-es szinuszos;
2. Egy 0,2 amplitúdójú 3000 Hz-es koszinuszos

jelet adunk hozzá.



3.63. ábra: FFT vizsgálat időfüggvényének előállítása

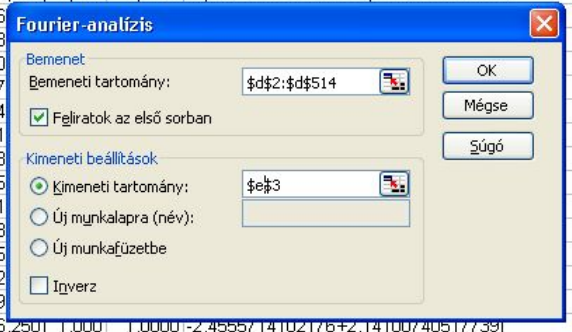
Lépések

1. Előállítjuk az alapadatok táblázatát és a grafikonokat. Az idő oszlop adatait lapszinten t-vel nevesítjük (3.63. ábra).

2. A forrásadatok másolatából előállítjuk a zajos jel Fourier transzformáltját. Az Adatelemzés menü

Fourier-analízis menüpontját választjuk, a forrástartomány a feliratos zajos jel, a céltartomány a következő oszlop (3.64. ábra).

2	idő	frekvencia	alapjel	zajos jel	Fourier transzformált
3	0,000000	0,000	0,000	0,2000	21,193332635297
4	0,000042	46,875	0,052	0,2247	22,4203457057057-4,70410218442341i
5	0,000083	93,750	0,105	0,1633	27,137245785203-11,3978515559969i
6	0,000125	140,625	0,156	0,0959	41,8277395675799-26,3942711444623i
7	0,000167	187,500	0,208	0,1030	174,316175373105-147,023944858422i
8	0,000208	234,375	0,259	0,2174	-56,3894801590073+59,6611026893834i
9	0,000250	281,250	0,309	0,4041	-21,4477786899284+27,3646501468683i
10	0,000292	328,125	0,358	0,5807	-12,3251130756423+18,4696478926645i
11	0,000333	375,000	0,407	0,6655	-8,22485882675817+14,2141358869327i
12	0,000375	421,875	0,454	0,6263	-5,93583332658481+11,6814087417673i
13	0,000417	468,750	0,500	0,5000	-4,49282521913283+9,98441787155533i
14	0,000458	515,625	0,545	0,3723	-3,50700666610274+8,76014033525911i
15	0,000500	562,500	0,588	0,3290	-2,79220747696376+7,83164161169834i
16	0,000542	609,375	0,629	0,4070	-2,24816886287905+7,10207478030478i
17	0,000583	656,250	0,669	0,5740	-1,8158325143611+6,5139106546421i
18	0,000625	703,125	0,707	0,7485	-1,45758128747585+6,03092038447151i
19	0,000667	750,000	0,743	0,8480	-1,14742876212312+5,62939095687129i
20	0,000708	796			
21	0,000750	843			
22	0,000792	890			
23	0,000833	937			
24	0,000875	984			
25	0,000917	1031			
26	0,000958	1078			
27	0,001000	1125			
28	0,001042	1171			
29	0,001083	1218			
30	0,001125	1265			
31	0,001167	1312			
32	0,001208	1359			
33	0,001250	1406,250	1,000	1,0000	-2,45557141021776+2,14100740517739i
34	0,001292	1453,125	0,999	1,1091	-2,11953028395234+2,14916663767879i
35	0,001333	1500,000	0,995	1,1357	-1,88301263060939+2,13189783357972i



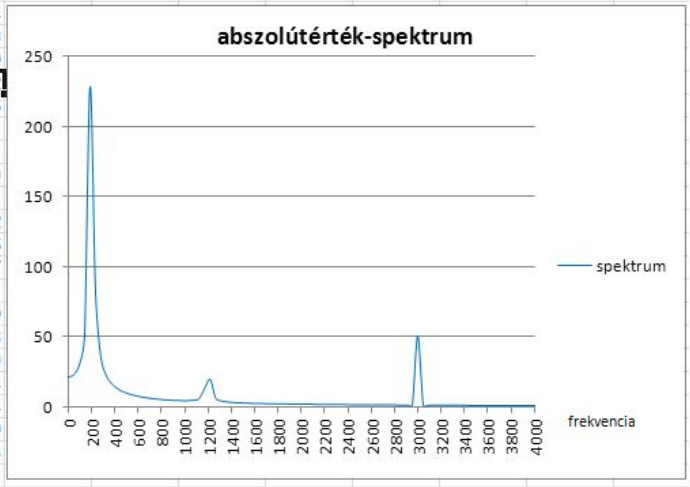
3.64. ábra: FFT vizsgálat paraméterezése

3. Elkészítjük frekvenciatartományban a jel spektrumát, ehhez kiszámítjuk a transzformált értékek abszolút

értékét és ábrázoljuk (3.65. ábra).

G6 =KÉPZ.ABSZ(F6)

B	E	F	G
frekvencia	zajos jel	Fourier transzformált	spektrum
0,000	0,2000	21,193332635297	21,193333
46,875	0,2247	22,4203457057057-4,70410218442341i	22,908524
93,750	0,1633	27,137245785203-11,3978515559969i	29,433673
140,625	0,0959	41,8277395675799-26,3942711444623i	49,459249
187,500	0,1030	174,316175373105-147,023944858422i	228,039842
234,375	0,2174	-56,3894801590073+59,6611026893834i	82,092756
281,250	0,4041	-21,4477786899284+27,3646501468683i	34,768251
328,125	0,5807	-12,3251130756423+18,4696478926645i	22,204421
375,000	0,6655	-8,22485882675817+14,2141358869327i	16,422240
421,875	0,6263	-5,93583332658481+11,6814087417673i	13,103031
468,750	0,5000	-4,49282521913283+9,98441787155533i	10,948702
515,625	0,3723	-3,50700666610274+8,76014033525911i	9,436056
562,500	0,3290	-2,79220747696376+7,83164161169834i	8,314507
609,375	0,4070	-2,24816886287905+7,10207478030478i	7,449411
656,250	0,5740	-1,8158325143611+6,51391065464211i	6,762269
703,125	0,7485	-1,45758128747585+6,03092038447151i	6,204558
750,000	0,8480	-1,14742876212312+5,62939095687713i	5,745140
796,875	0,8377	-0,865514917574138+5,29356370197631i	5,363854
843,750	0,7502	-0,594404360923289+5,01324919534951i	5,048364
890,625	0,6663	-0,315708291339672+4,78279017161709i	4,793199
937,500	0,6660	-5,48050707743419E-003+4,60123156990586i	4,601235
984,375	0,7805	0,374712644037373+4,47436803202757i	4,490031
1031,250	0,9723	0,900320219560874+4,42149611403213i	4,512228
1078,125	1,1559	1,75721956200312+4,49888005222353i	4,829880
1125,000	1,2462	3,58928473949398+4,91241265631l4i	6,083976
1171,875	1,2073	11,3196458049213+7,25902427594441i	13,447223
1218,750	1,0733	-19,3449398322671-2,85524129130809i	19,554516
1265,625	0,9272	-6,14234586230082+1,29637002439035i	6,277658
1312,500	0,8533	-3,91253109742516+1,89374458811038i	4,346742
1359,375	0,8881	-2,97705449860722+2,08030835675685i	3,631878
1406,250	1,0000	-2,45557141021759+2,14100740517738i	3,257874



3.65. ábra: FFT vizsgálat a frekvenciatartományban

Látható, hogy az időtartománybeli zajos jelünk 300, 1200, 3000 Hz körüli komponenseket tartalmaz.

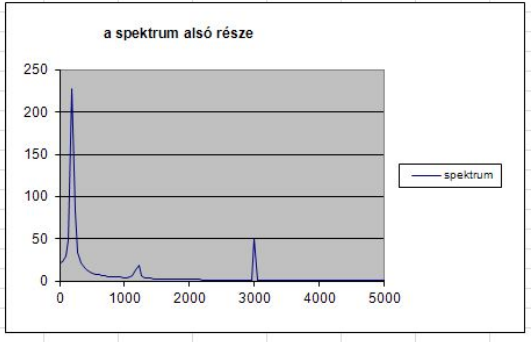
4. Az $f_0 = 1000$ Hz feletti részt levágjuk a frekvenciatartományban, azaz kinullázzuk. (A szimmetria miatt f_0 és $F - f_0$ közötti részt kell 0-vá tenni, mert az N darab spektrumvonalból csak $N/2$ független!)

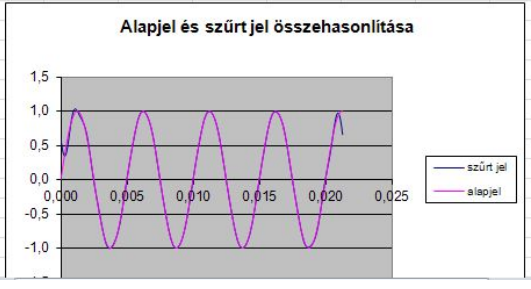
5. Visszatranszformálunk az időtartományba. Itt is a Fourier-analízis elemző eszközt használjuk, csak be kell jelölni az inverz-transzformáció jelölőnégyzetét.

6. Felrajzoltatjuk a szűrt jelet és összehasonlítjuk az alapjellel.

G22 | =HA(ÉS(B22>=vágás;B22<=24000-vágás);0;E22)

	A	B	C	F	G	H	I
1				1000	=vágás	szűrt jel	
2	idő	frekvencia	alapjel	spektrum	szűrt transzformált	Inverz FFT-je	szűrt jel
3	0,000000	0,000	0,000	21,193333	21,193332635297	0,598076806442988	0,59808
4	0,000042	46,875	0,052	22,908524	22,4203457057057-4,70410218442341i	0,540648211334684	0,54065
5	0,000083	93,750	0,105	29,433673	27,137245785203-11,3978515559969i	0,48742767102434	0,48743
6	0,000125	140,625	0,156	49,459249	41,8277395675799-26,3942711444623i	0,440407704295271	0,44041
7	0,000167	187,500	0,208	228,039842	174,316175373105-147,023944858422i	0,401349424441071	0,40135
8	0,000208	234,375	0,259	82,092756	-56,3894801590073+59,6611026893834i	0,371708862725204	0,37171
9	0,000250	281,250	0,309	34,768251	-21,4477786899284+27,3646501468683i	0,352577000018646	0,35258
10	0,000292	328,125	0,358	22,204421	-12,3251130756423+18,4696478926645i	0,344636454879229	0,34464
11	0,000333	375,000	0,407	16,422240	-8,22485882675817+14,2141358869327i	0,348136917125303	0,34814
12	0,000375	421,875	0,454	13,103031	-5,93583332658481+11,6814087417673i	0,362890445247013	0,36289
13	0,000417	468,750	0,500	10,948702	-4,49282521913283+9,98441787155533i	0,388286717464018	0,38829
14	0,000458	515,625	0,545	9,436056	-3,50700666610274+8,76014033525911i	0,423327296757406	0,42333
15	0,000500	562,500	0,588	8,314507	-2,79220747696376+7,83164161169834i	0,46667699658462	0,46668
16	0,000542	609,375	0,629	7,449411	-2,24816886287905+7,10207478030478i	0,516729569701928	0,51673
17	0,000583	656,250	0,669	6,762269	-1,8158325143611+6,5139106546421i	0,57168423464007	0,57168
18	0,000625	703,125	0,707	6,204558	-1,45758128747585+6,03092038447151i	0,629629040892045	0,62963
19	0,000667	750,000	0,743	5,745140	-1,14742876212312+5,62939095687129i	0,688626781537088	0,68863
20	0,000708	796,875	0,777	5,363854	-0,865514917574137+5,29356370197631i	0,746799104896945	0,74680
21	0,000750	843,750	0,809	5,048364	-0,594404360923289+5,01324919534951i	0,802404655521584	0,80240
22	0,000792	890,625	0,839	4,793199	-0,315708291339671+4,78279017161709i	0,853907476582575	0,85391
23	0,000833	937,500	0,866	4,601235	-5,4805070774338E-003+4,60123156990586i	0,900032505336134	0,90003
24	0,000875	984,375	0,891	4,490031	0,374712644037374+4,47436803202757i	0,939805754469203	0,93981
25	0,000917	1031,250	0,914	4,512228	0	0,972577649868453	0,97258
26	0,000958	1078,125	0,934	4,829880	0	0,99802893859826	0,99803
27	0,001000	1125,000	0,951	6,083976	0	1,01615953554804	1,01616
28	0,001042	1171,875	0,966	13,447223	0	1,02726158935851	1,02726
29	0,001083	1218,750	0,978	19,554516	0	1,03187886717975	1,03188
30	0,001125	1265,625	0,988	6,277658	0	1,03075523905055	1,03076
31	0,001167	1312,500	0,995	4,346742	0	1,02477555040348	1,02478
32	0,001208	1359,375	0,999	3,631878	0	1,01490248023513	1,01490



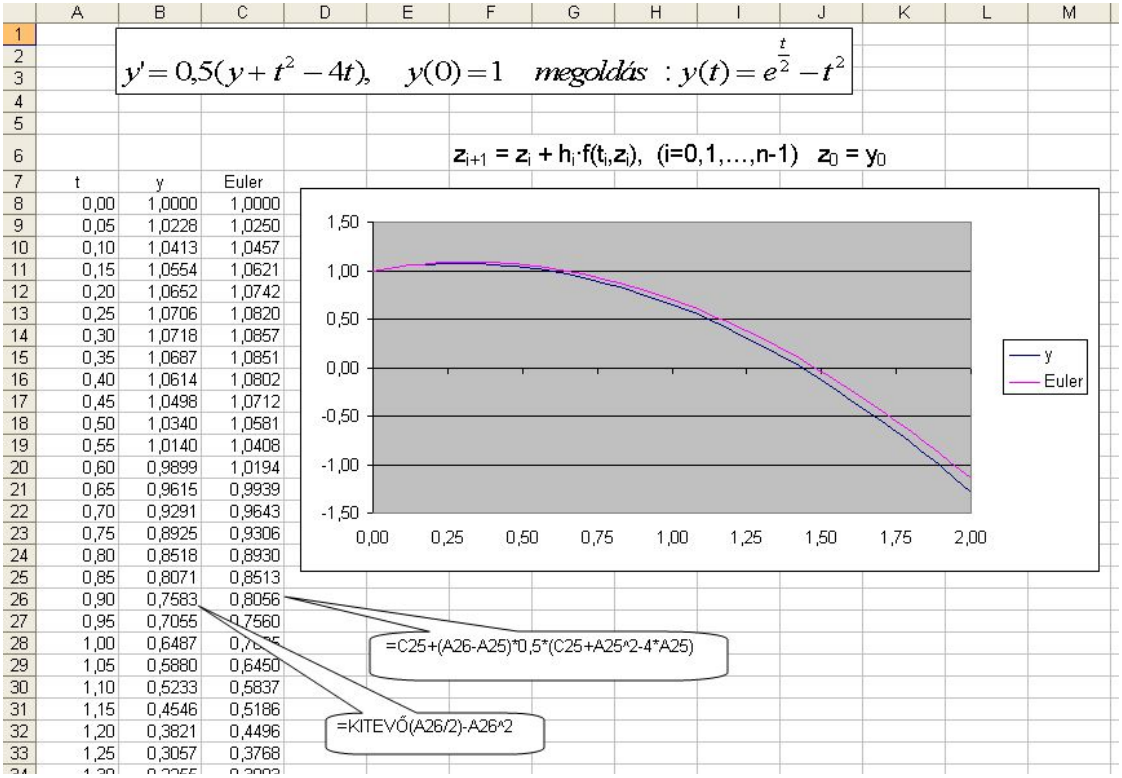


3.66. ábra: Sávszűrés a frekvencia-tartományban és inverz FFT

3.8.2. Közönséges differenciálegyenletek Excellel

Az egyváltozós kezdeti-érték feladatok jól kezelhetők az Excellel. A kezdeti-érték feladat a következő alakban adott:

$$y'(t) = f(t,y) \quad y(t_0) = y_0$$



3.67. ábra: Kezdeti-érték feladat Euler módszerrel

Az Exceles közelítés használatát olyan problémákon keresztül mutatjuk be, amelyek elméleti megoldása is ismert, ezért a közelítő megoldás jóságának ellenőrzése szemléletesen elvégezhető. A példáinkat a diffegy.xls munkafüzet tartalmazza.

Euler módszer

Az Euler módszer a kezdőpontból indulva kicsi lépésközönként a derivált becslésének segítségével előre haladva állítja elő a közelítő megoldást.

$$y_{i+1} = y_i + h_i f(t_i, y_i) \quad t_{i+1} = t_i + h_i \quad i = 0, 1, \ldots$$

Legyen a konkrét példánk ismert megoldású kezdeti érték feladat:

$$y' = 0.5(y + t^2 - 4t), \quad y(0) = 1, \quad \text{megoldása:} \quad y(t) = e^{\frac{t}{2}} - t^2$$

A közelítés lépései a 3.67. ábráról leolvashatók.

Runge-Kutta módszerek

A Runge-Kutta módszerek közül a klasszikus negyedrendű séma hosszú intervallumon is jól közelít, sőt a szakadási helyeken is sokszor jól át tud lépni. Ezt illusztrálja a következő kezdeti-érték feladat megoldása:

$$y' = F(x,y) = \frac{1 - y - \cos(x)}{\sin(x)}, \quad x_0 = 1, \quad y_0 = 1$$

A probléma megoldása ismert:

$$y(t) = \left[\frac{y_0}{tg\left(\frac{x_0}{2}\right)} + x - x_0 \right] tg\left(\frac{x}{2}\right)$$

A RK4 képletek pedig:

$$h_2 = h/2; \qquad k_1 = F(x_i; y_i); \qquad k_2 = F(x_i + h_2; y_i + h_2 * k_1); \qquad k_3 = F(x_i + h_2; y_i + h_2 * k_2);$$
$$k_4 = F(x_i + h; y_i + h * k_3); \qquad y_{i+1} = y_i + (k_1 + 2 * k_2 + 2 * k_3 + k_4) * h/6;$$

Annyi extra teendőnk van csak, hogy az $F(x, y)$ függvényérték kiszámítását elvégző Visual-Basic modult elkészítsük, a többi számítási lépés a fenti képletek alapján történik:

	A	B	C	D	E	F	G
1	x_0	1					Kezdeti érték
2	y_0	1					
3	h	0,0250					Lépésköz
4	h_2=h/2	0,0125					
5							
6	k_1,k_2,k_3,k_4						
7	segédoszlopok						
8							
9	x	y(RK-4)	k1	k2	k3	k4	y
10	1,0000	1,00000	1,73470	1,75904	1,75939	1,78409	1,00000
11	1,0250	1,04398	1,78409	1,80913	1,80950	1,83492	1,04398
12	1,0500	1,08922	1,83491	1,86070	1,86107	1,88725	1,08922
13	1,0750	1,13574	1,88725	1,91382	1,91420	1,94119	1,13574
14	1,1000	1,18359	1,94118	1,96858	1,96896	1,99679	1,18359
15	1,1250	1,23281	1,99679	2,02505	2,02544	2,05416	1,23281
16	1,1500	1,28345	2,05416	2,08333	2,08373	2,11338	1,28345
17	1,1750	1,33554	2,11338	2,14351	2,14392	2,17456	1,33554
18	1,2000	1,38913	2,17456	2,20571	2,20612	2,23780	1,38913
19	1,2250	1,44428	2,23780	2,27002	2,27044	2,30322	1,44428
20	1,2500	1,50104	2,30322	2,33656	2,33700	2,37093	1,50104
21	1,2750	1,55946	2,37093	2,40546	2,40591	2,44107	1,55946
22	1,3000	1,61961	2,44106	2,47685	2,47731	2,51376	1,61961
23	1,3250	1,68154	2,51376	2,55088	2,55135	2,58917	1,68154
24	1,3500	1,74532	2,58917	2,62769	2,62818	2,66744	1,74532
25	1,3750	1,81102	2,66744	2,70744	2,70795	2,74874	1,81102
26	1,4000	1,87871	2,74874	2,79032	2,79085	2,83326	1,87871
27	1,4250	1,94848	2,83326	2,87651	2,87706	2,92119	1,94848
28	1,4500	2,02041	2,92119	2,96621	2,96678	3,01274	2,02041
29	1,4750	2,09457	3,01274	3,05964	3,06023	3,10813	2,09457
30	1,5000	2,17107	3,10812	3,15703	3,15764	3,20760	2,17107
31	1,5250	2,25001	3,20759	3,25862	3,25926	3,31141	2,25001
32	1,5500	2,33149	3,31141	3,36470	3,36537	3,41986	2,33149
33	1,5750	2,41562	3,41985	3,47556	3,47625	3,53323	2,41562
34	1,6000	2,50252	3,53323	3,59150	3,59223	3,65187	2,50252
35	1,6250	2,59233	3,65187	3,71289	3,71365	3,77613	2,59233
36	1,6500	2,68516	3,77612	3,84008	3,84088	3,90639	2,68516
37	1,6750	2,78118	3,90639	3,97348	3,97432	4,04308	2,78118
38	1,7000	2,88054	4,04308	4,11354	4,11443	4,18667	2,88054
39	1,7250	2,98339	4,18667	4,26073	4,26166	4,33764	2,98339
40	1,7500	3,08993	4,33764	4,41557	4,41656	4,49656	3,08993

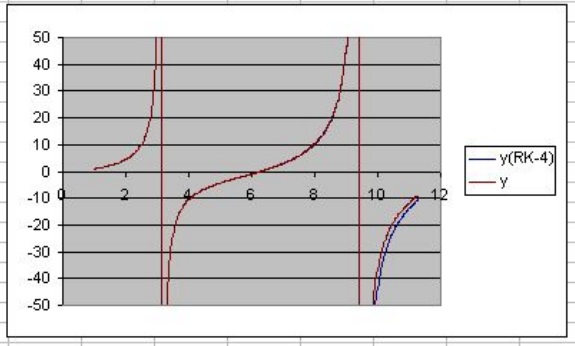
$y' = F(x, y) = (1 + y - \cos(x)) / \sin(x)$ $x_0 = 1$; $y_0 = 1$ kezdetiérték feladat megoldásának numerikus közelítése 4-edredű Runge-Kutta módszerrel

Képletek:

$k_1 = F(x_i; y_i)$ $k_2 = F(x_i + h_2; y_i + h_2*k_1)$; $k_3 = F(x_i + h_2; y_i + h_2*k_2)$;

$k_4 = F(x_i + h; y_i + h*k_3)$; $y_{i+1} = y_i + (k_1 + 2*k_2 + 2*k_3 + k_4)*h/6$;

Elméleti megoldás: $y = [y_0*\tg(x_0/2) + x - x_0]*\tg(x/2)$



Visual Basic modul:

```
Function F(x, y)
  Dim r
  If Sin(x) = 0 Then
    r = "Hiba!"
  Else
    r = (1 + y - Cos(x)) / Sin(x)
  End If
  F = r
End Function
```

3.68. ábra: Kezdeti-érték feladat negyedrendű Runge-Kutta módszerrel

Önellenőrzés

1. Generáljunk véletlen egész számokat pl. 0 és 100 között (1000 db). Készítsünk rekesztartományt tíz egységes felosztással. Adatelemzéssel kérjük le az adatok hisztogramját és eloszlásfüggvényét! Határozzuk meg a szórást! Értelmezzük az eredményt!
   A megoldás megtekintéséhez kattintson ide!
2. Végezzük el az előző feladatot egy hallgatói csoport magasságadataival is (centiméterben adottak)! Segítség: Vegyünk fel 20-30 értelmes adatot. A minimum és a maximum értékek megfelelő megválasztása mellett ügyeljünk az „életszerűségre” is (átlagos/tipikus férfi és női magasság). Milyen különbség adódik az előző feladathoz képest?
   A megoldás megtekintéséhez kattintson ide!
3. Készítsük el a lecke mindkét kezdeti-érték feladatának (3.8.2. alfejezet) közelítő megoldását az ismertetett másik módszerrel!
   A megoldás megtekintéséhez kattintson ide!
4. Készítsük el az $y' = x^2 + y^2$; $y(0) = 0$; $0 \leq x \leq 2$ kezdeti-érték feladat mindkét módszerrel való közelítő megoldását a következő fejlécű táblázatban! A h osztásköz legyen 0,05!

x	y_euler	k1	k2	k3	k4	y_rk4

   Készítsük el a két közelítő megoldás közös grafikonját is (görbe vonallal összekötött pontdiagram)!
   A megoldás megtekintéséhez kattintson ide!

Modulzáró feladatok

5. Állítsuk elő π kellő pontosságú racionális közelítését, azaz adjuk meg két pozitív egész szám hányadosaként a közelítő értéket! A megoldáshoz a lánctörtbe fejtést használjuk, majd a lánctört közös nevezőre hozásával kapjuk meg a kívánt tört alakot. Készítsük el a 4, majd az 5-lépéses közelítést is!
Egy 1-nél nagyobb x szám lánctörtbe fejtése

$$b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{\ddots\, b_{n-1} + \cfrac{1}{b_n + x_{n+1}}}}}$$

alakú, ahol a b_k érték egy 1-nél nem kisebb szám egész része és a mellette álló tört ennek a számnak a tört része, melynek a nevezőjében így 1-nél nagyobb érték van. Irracionális x szám esetén a lánctört végtelen hosszú. Ekkor, ha az x_{n+1} törtrészt az n-edik lépés után nem bontjuk tovább, hanem 0-val (tört alakban 0/1-gyel) helyettesítjük, akkor az x szám egy jó közelítését kapjuk. Ezután a lánctört sorozatos közös nevezőre hozásával az x szám racionális közelítéséhez jutunk.
A megoldás megtekintéséhez kattintson ide!

6. Ábrázoljuk a következő függvényt és deriváltját, és Solver segítségével határozzuk meg a függvény nevezetes helyeit. Tegyük fel ezeket a grafikonokra.

$$f(x) = (Ax^2 + Bx + D)\sin(x + E)$$

$$f'(x) = (2Ax + B)\sin(x + E) + (Ax^2 + Bx + D)\cos(x + E)$$

Ahol

A	B	D	E
1	-16	9	6

A megoldás megtekintéséhez kattintson ide!

7. Ábrázoljuk az $r = r_0$*abs(sin(2*φ)) paraméteres polárkoordinátás egyenlettel adott „négylevelű lóherét”! A φ paraméter 360 lépésben fusson végig a $[0; 2\pi]$ intervallumon, és az $r_0 = 12$ értékkel dolgozzunk!

Határozzuk meg az ábra összes olyan pontját, ahol az x illetve y értékek maximálisak illetve minimálisak. Ezeket a pontokat új forrásadatként tegyük fel az ábrára, és a szomszédos pontokat (8 darab van) egyenes, piros színű szakaszokkal kössük össze!

A megoldás megtekintéséhez kattintson ide!

8. Egy $\mathbf{Ax} = \mathbf{b}$ lineáris egyenletrendszer együtthatóinak táblázatával adott (az utolsó oszlop a $\mathbf{b}$ vektor).

3.5. táblázat. Lineáris egyenletrendszer együtthatói

5	-2	3	1	4	19
-3	6	2	4	0	8
2	3	-1	-3	2	-16
-1	4	1	2	-3	10
9	10	2	-3	-1	9

1. Mutassuk meg, hogy az egyenletrendszer összefüggő!
2. Az 5. egyenlet az első négy egyenlet lineáris kombinációjaként állítható elő. Határozzuk meg a lineáris kombináció együtthatóit!
3. Ellenőrizzük, hogy az $\mathbf{x} = [2; -1; 4; 3; -2]$ vektor az egyik megoldása az egyenletrendszernek!
4. Határozzuk meg azt az $\mathbf{x}$ megoldásvektort, amelynek az euklideszi normája minimális!

A megoldás megtekintéséhez kattintson ide!

9. Határozzuk meg az

$$y' + y\cos(x) = 0{,}5\sin(2x) \qquad y(0) = 1$$

differenciálegyenlet közelítő megoldását a [0; 25] intervallumban 0,1 lépésközzel az ismertetett negyedrendű Runge-Kutta módszer segítségével! Hasonlítsuk össze a kapott eredményt a probléma behelyettesítéssel ellenőrizhető pontos megoldásával, amely

$$y = 2e^{-\sin(x)} + \sin(x) - 1.$$

A megoldás megtekintéséhez kattintson ide!

10. Ebben a feladatban egy típusú ömlesztett árut kell elszállítani három telephelyről öt területi raktárba. Az áruk bármely telephelyről bármelyik raktárba elszállíthatók, de természetesen a távolabbi raktárba való szállítás többe kerül. A feladat annak meghatározása, hogy mennyi árut kell elszállítani az egyes telephelyekről az egyes raktárakba úgy, hogy a szállítási költség minimális legyen, a szükségleteket kielégítsük, de ne akarjunk sehonnan se az ott lévő készletnél többet elvinni. Az alábbi költségtáblázat egyetlen egység szállítási költségét tartalmazza.

3.6. táblázat. Adatok a raktáros feladathoz

		Az x telephelyről az y területi raktárba való szállítás költsége:				
Telephely:	*Készlet (egység)*	*Sopron*	*Pápa*	*Kaposvár*	*Veszprém*	*Budapest*
Kecskemét	310	10 000	8 000	6 000	5 000	4 000
Pécs	260	6 000	5 000	4 000	3 000	6 000
Szombathely	280	3 000	4 000	5 000	5 000	9 000

Forrás: C:\Program Files\Microsoft Office\Office10\Samples\solvsamp.xls

A megoldás megtekintéséhez kattintson ide!

IV. MODUL

Adatbázis-kezelés

Kulcsszavak: adatbázis, relációs adatbázis, SQL, adatbázis-kezelő

Ebben a modulban azzal ismerkedünk meg, hogy mi az egyik legáltalánosabb, leginkább szabványos módja az adataink számítógéppel történő kezelésének. Megismerkedünk az adatbázis és adatbázis-kezelő fogalmával, majd a relációs adatbázis-kezelők kezelőnyelvével.

18. LECKE

Bevezetés az adatbázis-kezelésbe

4. Adatbázis-kezelés

Hétköznapi életünkben is észrevehetjük, hogy egyre több adat, információ vesz körbe bennünket. Időjárás, árfolyam, benzinár. A következő vizsga kódja, dátuma, időpontja. A bankkártyánk PIN kódja. Jelszavaink. A focibajnokság eredményei. Adószám, TAJ-szám, neptunkód. Ahogy az utóbbi évtizedekben minden munkahelyre és háztartásba belopózott a számítógép, úgy lett egyre természetesebb, hogy adatainkat számítógépen tároljuk. A számítógép segít az adatok tárolásában, de az adatok tárolásának miként jét nekünk kell meghatározni.

4.1. Adatkezelési alapfogalmak

Érdemes tisztáznunk az *adat* szó jelentését. Adaton valamely tény reprezentálását értjük. Ezek a tények mindenki számára egyértelmű, objektív események, történések, állapotok. Úgy is mondhatjuk, az adat valamely dolog valamely tulajdonságának az értéke. Az adatokat természetesen valamely adathordozó tárolja, de ez az adathordozó független az adattól. Adatok tárolhatók sokféleképpen, mi természetesen számítógépen tárolt adatokkal foglalkozunk a későbbiekben. Az információt ezzel szemben egy hasznos közlésnek, egy jelentéssel bíró adatnak tekinthetjük. Nap mint nap sok-sok információhoz jutunk. Információ például egy hír a zárthelyi dolgozat várható feladattípusairól. Az információhoz szorosan kapcsolódik a

- hordozó közeg (papír, hanghullám, ...)
- jelentés
- a feldolgozó (én magam, akit érdekel, mi lesz a zárthelyin).

A fentiek miatt az információ mindig szubjektív fogalom, mert függ a feldolgozótól. Aki nem vette fel az adott tárgyat, annak számára a zárthelyi tartalma vajmi kevés információval bír, máshogy értékeli a hallottakat az is, aki már harmadik alkalommal igyekszik aláírást szerezni. A szubjektivitás mellett vegyük észre, hogy a hétköznapi életben is azokat a dolgokat találjuk érdekesnek, amelyeknek bekövetkeztére kevésbé számítunk. Mindenki hallotta már, hogy nem az a hír, ha a kutya megharapja a postást, hanem az, hogyha ez fordítva történik – mert ilyet ritkábban hallunk. A zárthelyi dolgozat kérdései is azok számára a legizgalmasabb

hírek, akik nem jártak órára, ezért alig tudják, mire kell a számonkérésnél számítani. Mindezt úgy is megfogalmazhatjuk, hogy egy adat információtartalma fordítottan arányos a valószínűséggel.

Az adat és információ kapcsolata ezek alapján már egészen világos: azok a számok vagy egyéb tényszerű dolgok, amiket a számítógépen tárolunk, az adatok. Ezen adatokból mindenki a számára fontosakat használja fel a saját szubjektív szempontjai alapján, így nyerünk az adatfeldolgozás során információt.

4.2. Adattárolás számítógépen

A számítógépen történő adattárolás sokféleképpen megvalósítható. Tárolhatjuk adatainkat szövegek vagy fotók formájában. A fájlokba történő rendezés és a mapparendszer teljesen hétköznapi megoldás. Mégis érezzük, hogy nagyon sokféle adatkezeléshez nem ez a megfelelő megoldás. Furcsállnánk, ha a tranzakcióinkat úgy tartaná nyilván a bank, hogy mondjuk egy alkalmazott egy szövegfájlba minden alkalommal beírja az aktuális pénzmozgás adatait, ami aztán formázva hónap végén bankszámla-kivonatként nyomtatható is lenne. Azt is furcsállnánk, ha az aláírásmintánk mapparendszerben, képfájlokban lenne tárolva.

A számítógépes adatkezelés a számítógépekkel egyidős, és rég felmerült az igény egy egységes, strukturált adatkezelési módszerre.

4.3. Adatkezelési elvárások

Vegyük sorra, milyen követelményeket fogalmazhatunk meg egy általános célra megfelelő adatkezelő rendszerrel kapcsolatban:

Nagy mennyiségű adatok hatékony kezelése

Elsőként fogalmazzuk meg, hogy az adatkezelő rendszerektől gyakorlatilag „korlátlan” mennyiségű adat megfelelő kezelését várjuk el. A korlátlanságot természetesen a gyakorlati, ésszerű határokon belül értjük, ott viszont komolyan is vesszük: senki nem fogadná el, ha egyik évben nem lehetne újabb hallgatókat felvenni az egyetemre, mert „betelt” a Neptun rendszer és „nem fér bele” újabb hallgató. Holott, biztosan van fizikai határa a lehetséges hallgatói létszámnak, csak ez olyan nagy, hogy a gyakorlatban soha nem fogjuk elérni. A

Google keresőjétől pedig azt vesszük természetesnek, hogy *nem lassabb*, mint 10 éve, pedig ennyi idő alatt a fellelhető webes tartalom a sokszorosára duzzadt.

A nagy adatmennyiség mellett is vannak elvárásaink a programok válaszidejére. Az elvárt válaszidő függ az adott alkalmazástól. Legtöbb esetben, ha egy gomb lenyomása után 1-2 másodpercig nem kapunk választ vagy eredményt, akkor a programot már rossznak ítéljük – ilyenkor a tapasztalatlanok elkezdenek türelmetlenül tovább kattintgatni, mert úgy látják, „lefagyott" a program... Az interaktív programokat jellemzően úgy kell elkészíteni, hogy másodpercen belül reagáljanak a legtöbb akcióra. Egyes esetekben, ha a felhasználót egyértelműen tájékoztatjuk a munka készültségi fokáról (százalékos kijelzés, vagy a „progress bar"), elfogadható egy maximum 20-30 másodperces várakozás is. Egy gyár termelésütemező rendszerében azonban, amikor a következő havi termelés paramétereit számoltatják ki a szoftverrel, valószínűleg teljesen elfogadható egy 8 órás futási idő.

A nagy adatmennyiség velejárója a nagy helyszükséglet. Ezzel kapcsolatban vegyük észre (a későbbiekben részletesen foglalkozni fogunk a témával), hogy a különböző funkciók gyorsításához sok esetben további, redundáns adatok tárolására van szükség. Azt is könnyű belátni, hogy tömörített tárolással helyet takaríthatunk meg, azonban a rendszer használata lassabbá válik. Így egyensúlyt kell teremteni az adatkezelő rendszer által felhasznált tárterület és a felhasználó számára elfogadható válaszidők között is. Az autóba szerelt GPS navigáció használatakor 10 másodperc útvonal-tervezési időt még elfogadhatónak tartunk, miközben a teljes Európa-térkép elfér a 4 GB-os belső memórián. Biztosan nem aratna nagy sikert egy olyan, nehezebb és sokkal drágább berendezés, ami ezt az időt 5 másodpercre szorítja olyan áron, hogy egy 500 GB méretű háttértárat kellene csatlakoztatni, amin minden település-pár közötti útvonal tárolva lenne, amit így csak be kell tölteni.

Konkurens hozzáférés támogatása

Az adatkezelő rendszerek jellemzően nem személyes használatra készülnek. Természetes igény, hogy minden felhasználó elvárja, hogy a rendszert bárki más munkájától függetlenül tudja használni. Az egyidőben ugyanazokon az adatokon dolgozó programokat, illetve felhasználókat a „konkurens" szóval jellemezzük.

Az egyes felhasználók műveleteinek – az ésszerű határokon belül – egymástól függetleneknek kell maradniuk. Az olvasási műveletekkel nincs különösebb gond, legfeljebb jelentéktelen mértékű lassulást tapasztalhatunk. Az írási műveleteknél a konkurens hozzáférések biztosítása, illetve működése nem triviális. Azt kell mindenképpen

rendszerszinten biztosítani, hogy az egyidejű írási műveletek ne vezethessenek hibákhoz. Ennek egyik tipikus példája az ún. *lost update* jelenség: képzeljünk egy egy vállalatot, ahol a dolgozók 10% béremelést kapnak, az egyik kiváló dolgozó pedig emellett 50 ezer forint egyszeri jutalomban is részesül. A két feladatot két ügyintéző hajtja végre. Nézzük, milyen feladatokat lát el az első ügyintéző (természetesen a béreket nyilvántartó programja segítségével):

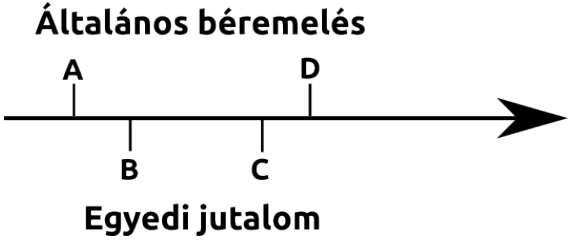
1. A dolgozó bérének kiolvasása
2. E havi bér := alapbér + 10%
3. A dolgozó e havi bérének letárolása

Ezt a műveletet természetesen a vállalat összes dolgozója esetében el kell végezni. Nézzük, mit csinál eközben a másik ügyintéző a megjutalmazott kolléga bérével:

1. A dolgozó bérének kiolvasása
2. E havi bér := alapbér + 50000
3. A dolgozó e havi bérének letárolása

Tekintsük a 4.1. ábrát:

Az első ügyintéző programja az 1-es (A) időpontban olvassa ki a béremelést kapó kolléga adatait, kiszámolja a fizetésemelés alapján az új bért és igen kis, de mégis valamennyi idő múlva (D) időpontban visszaírja az adatot. Ha a másik ügyintéző éppen ezalatt (B időpontban) olvassa ki az adott dolgozó fizetését, még természetesen az eredeti értéket látja. Valami oknál fogva hamarabb végez a számítással és C időpontban be is tudja írni az ötvenezer forinttal megnövelt összeget. Ahogy az ábráról leolvasható, ez az összeg azonban pillanatokkal később (D időpontban) felülíródik a másik értékkel. Így, bár mindkét ügyintéző jól dolgozott, egy kolléga mégsem kapta meg az ötvenezer forintos jutalmat, az egyik módosítás a konkurens módosítások nem megfelelő kezelése miatt elveszett. Még csak hibaüzenet sem érkezett róla (hiszen hiba végül is nem történt), így talán az illető dolgozó sosem tudja meg, hogy főnöke meg akarta jutalmazni... Ez a szituáció elkerülhető úgy, hogy az adatkezelő rendszer B időpontban nem engedi kiolvasni az adatot, hiszen tudja, hogy azt egy másik folyamat



4.1. ábra. *A lost update jelenség*

A időpontban már kiolvasta (és zárolta). D időpontban megtörténik az új érték beírása és a zárolás feloldódik. Ekkor már a második folyamat is megkaphatja a kért adatot és dolgozhat rajta. A második folyamatnak ez egy észrevehetetlenül kicsi, $D-B$ időveszteséget okoz, cserébe a program a módosítások az elvárt módon történnek meg.

Integritásőrzés

Egy jó adatkezelő rendszer nagyon sok rutinmunkát és felelősséget levesz a használója válláról, hogy előre megadott szabályok alapján folyamatosan ellenőrizni tudja az adatokat és nem engedi, hogy a feltételeknek ellentmondó adatok a rendszerbe kerüljenek. A feltételek sokrétűek lehetnek és különböző komplexitásban vizsgálhatók. A legegyszerűbb ellenőrzések közé tartozik, hogy a raktárkészlet nemnegatív egész szám, az adószám megfelel az adóhatóság által megadott formátumnak, személyeknél a név mező pedig nem lehet üres. Bonyolultabb, több adat összekapcsolásával is megfogalmazhatók integritási kritériumok: ha egy személy beltag egy betéti társaságban, akkor másik Bt-ben már nem tölthet be ugyanilyen tisztséget. Azzal is sok további kényelmetlenség elkerülhető, ezért jogos elvárás az adatkezelő rendszertől, ha egy olyan autó tulajdonosi viszonyait nem lehet átírni az önkormányzatnál, ami szerepel a lopott gépjárművek rendőrségi nyilvántartásában. Az integritási feltételekről részletesebben a 4.4.9. fejezetben lesz szó.

Adatvédelem (Safety + Security)

A *biztonság* szónak angolul két, eltérő jelentésű megfelelője van: a *safety* és a *security*.

Safety: adatvesztés lehetőségének minimalizálása, kizárása.
A safety olyan fajta biztonságot takar, amikor az adatvesztés lehetőségét próbáljuk kizárni, vagy ha ez nem sikerül, legalább minimális, elfogadható szinten tartjuk. Ennek különböző megoldásai lehetségesek, egy megbízható hardveren futó megbízható szoftver adatait is érdemes rendszeresen menteni és tervezett ideig megőrizni. A fizikailag elkülönített tárolásról sose feledkezzünk meg: tűz esetén mit sem ér, ha a gondosan, napi rendszerességgel elkészített biztonsági másolatokat a szerver tetején tároljuk egy dobozban. A biztonsági mentések készítésének módja természetesen az adott alkalmazástól függ: egy bank saját ingatlanjainak nyilvántartását lehet, hogy elég naponta vagy hetente menteni, de a pénzügyi tranzakciók esetében az utolsó 5 perc átutalásainak elvesztése is katasztrofális lenne.

Security: jogosulatlan hozzáférések megakadályozása.
A biztonság másik vonatkozását az angol *security* szó írja le a legmegfelelőbben. Az információs társadalomban egyre jobban megtanuljuk, hogy az információ érték. Adatainkat mindig védeni kell az illetéktelen hozzáféréstől, legyen az akár olvasási vagy írási jellegű. Alapvető védelmi megoldás a felhasználók névvel és jelszóval történő azonosítása. Az adatkezelő rendszereknél, mint azt említettük már, alapvető feltétel a többfelhasználós működés, ami szinte azonnal feltételezi azt is, hogy a rendszer hálózaton elérhető. Ez pedig biztonsági kérdéseket is azonnal felvet, hiszen egy hálózatba kötött gépet komoly szakértelemmel *kell* megvédeni a lehetséges támadásoktól, ami sokkal komolyabb munka, mint a fizikai védelem, amit sok esetben egyetlen zárt ajtó is megfelelően biztosít. A jelszavas azonosítás mellett jellemző védelmi megoldás az is, ha az egyes felhasználók a saját belépési azonosítójuk birtokában is csak kijelölt helyekről férhetnek hozzá a rendszerhez. Például, az igazgató jogosultságával csak az igazgatói irodából lehessen a rendszerhez férni, a közeli nyilvános könyvtárból ne.

Szabványosság

Egyre természetesebb elvárás, hogy az egyébként egyre bonyolultabbá váló eszközeinket mégis egyre egységesebben, egyszerűbben tudjuk kezelni. Mobiltelefon vagy televízió vásárlása után azonnal használni szeretnénk az eszközöket. Valószínűleg akkor is haza tudjuk vinni az új autónkat, ha a kereskedés előtt nem olvassuk végig alaposan az egyébként sok hasznos részletet leíró vaskos kézikönyvet. Ugyanezt várjuk el szoftvereinkkel kapcsolatban: aki bérszámfejtő munkakörben jól megfelelt az egyik cégnél, nem akar munkahely-változtatás után újabb hónapokat tölteni egy hasonló de merőben más logikával működő másik rendszer megismerésével. Ez a szoftvertervezés terén egyrészt a kvázi szabványos, ergonomikus felhasználói felületeknek köszönhető, másrészt annak, hogy a mai szoftvermérnökök a fejlesztett rendszereik „belsejében” is szabványos megoldásokra törekszenek. Az adatkezelő rendszereknél ez különösen fontos elvárás, hiszen sokszor kell különböző adatkezelő rendszerek közötti adatcserét megvalósítani.

Szabványos adatkezelő rendszereken tehát a fejlesztés, illetve a felhasználók betanítása, majd a későbbi munkájuk sokkal gyorsabb (azaz olcsóbb), kényelmesebb.

Kényelem

Ez az elvárás valójában nem új fogalmat takar, inkább az előzőek összevonásának tekinthető. Az adatainkat kényelmesen, bármikor, bárhonnan el akarjuk érni úgy, hogy a rugalmas rendszer egyébként minden felhasználói hibát akadályozzon meg és az adataink csak a jogosultaknak legyenek hozzáférhetők. Mindezt persze gyorsan, olcsón és minimális erőforrásigénnyel...

Túlzóan optimistának tűnik ez az elvárás, mégis erre törekszünk és elmondhatjuk, hogy a mai modern adatkezelő rendszerek egészen jól teljesítik is ezeket a követelményeket. A továbbiakban vizsgáljuk meg részleteiben, hogyan működnek a legelterjedtebb rendszerek.

4.4. Relációs adatmodell

4.1. definíció: Modell: A valóság olyan matematikai vagy tárgyi leképezése, ami a modellalkotó számára fontos tulajdonságokban egyezést mutat a valósággal.

A *modell* fogalmat (elsősorban nem a tárgyi modellre gondolunk) a tudományos életben, kutatásban igen gyakran használják, hiszen a jól megválasztott, matematikailag kidolgozott fogalmak, eszközök segítségével áttekinthetően és rugalmasan leírható a valóság vizsgált részlete. Mi az adatkezelés módját is egy modellel, az ún. *relációs adatmodellel* fogjuk leírni. Ehhez a modellhez természetesen sok-sok kísérletezés útján sikerült eljutni, a korábbi és későbbi, kevésbé elterjedt adatmodellekkel pedig nem foglalkozunk. A relációs adatmodell alapjait az IBM-nél dolgozó Codd fektette le 1970 nyarán egy cikkében („A Relational Model of Data for Large Shared Data Banks"). A cikk első gondolata az volt, hogy az adatbankok használatánál a belső, fizikai tárolással ne kelljen törődni, így felvázolt egy logikai tárolási, adatelérési módot. Az egész átlátható, és egyszerű volt, így gyorsan népszerűvé vált. Az elméleti megalapozottság, a precíz matematikai háttér pedig a kutatók szimpátiáját is kiváltotta. Mindennek következtében az adatmodellt hamar több különböző implementációban is megvalósították, és hamar jelentősen háttérbe szorított minden más hasonló próbálkozást.

A relációs adatmodell egyrészt leírja az adatok tárolási struktúráját, másrészt a ezeken a struktúrákon végzendő műveleteket. Emellett van egy harmadik fontos komponense is, az integritási feltételek. Ez nem része sem a strukturális, sem a műveleti komponensnek, hanem magukat az adatértékeket, illetve az értékek közötti kapcsolatokat, szabályszerűségeket szabályozza.

A relációs adatmodell tehát az alábbi három komponensből tevődik össze:

1. adatstruktúra
2. műveletek
3. integritási feltételek

A relációs adatmodellben az adott feladathoz tartozó összes adatot tekintjük egy adatbázisnak. (Így beszélhetünk a kórház betegnyilvántartó adatbázisáról, vagy az iskolai könyvtár adatbázisáról.)

4.4.1. Reláció, mező, rekord

4.2. definíció: Reláció = azonos típusú rekord-előfordulások összessége

Az azonos típusú adatokat relációkba szervezzük. A reláció legegyszerűbb esetben egy táblázat, vagy ahogy inkább nevezni szokták, tábla. Az iskolai könyvtárban egyik táblában tárolhatjuk a könyvek adatait, hiszen minden könyvnek hasonló típusú adatai vannak (cím, szerző, oldalszám) egy másikban pedig a kölcsönző személyek adatait.

A relációk (táblák) meghatározott tulajdonságokat tartalmazhatnak, ezeket mezőknek hívjuk. Mint említettük, mező lehet a könyv címe, szerzője és oldalszáma.

Amikor a tábla mezőit meghatároztuk (ezek vizuálisan egy tényleges táblázat fejlécei lehetnek), alá soronként egy-egy konkrét adat-előfordulást írhatunk:

Cím	Szerző	Megj_év	Oldalszám
Versek	Petőfi Sándor	1848	245
Hitel	Széchenyi István	1830	159

4.2. ábra. *Példa a relációs adatmodell táblájára*

Egy ilyen sort rekordnak is szokás nevezni, a 4.2. ábra táblája tehát 2 rekordot tartalmaz.

A reláció nem csak tábla lehet, hanem több táblából is összeállhat. Egyszerű (és ritkább) esetben két teljesen megegyező felépítésű tábla összeillesztéséből is kaphatunk egy ugyanolyan felépítésű, de természetesen nagyobb rekordszámú relációt.

Descartes szorzat, halmazok

Gyakran szükséges, hogy két logikailag összefüggő, de különböző felépítésű relációból (táblából) állítsunk elő egy újabb relációt. Ehhez meg kell ismerkednünk egy matematikai, azon belül pedig halmazelméleti fogalommal, a Descartes-szorzattal.

4.3. definíció: Az A és B halmaz Descartes-szorzatán azt a halmazt értjük, melynek azon rendezett párok az elemei, amiknek első eleme A-beli, második eleme pedig B-beli és a szorzat minden lehetséges párt tartalmaz.

Tegyünk még hozzá annyit, hogy a definíció értelemszerűen kibővíthető kettőnél több halmazra is. Továbbá, az esetek döntő többségében a Descartes-szorzatnak nem használjuk minden párosítását (az eredmény minden sorát), hanem csak azokat, amelyek valamely összetartozási feltételnek megfelelnek. Ezeket a megfelelő sorokat (a személy azonosítója = autó *tulajdonos* mezője) a 4.3. ábrán sárga színnel jelöltük.

Reláció tehát nemcsak tábla, hanem táblák Descartes-szorzata is lehet.

Nagyon fontos megérteni, hogy két nagy tábla Descartes-szorzata hatalmas adatmennyiséget eredményez, ami az esetek jelentős részében valószínűleg szükségtelen is. A 4.3. ábrán is csak a sárga színnel kiemelt, összetartozó rekordokra van szükség. Az ilyen párosításokat az adatbázis-kezelők hatékonyabban is elő tudják állítani, nincs szükség a teljes Descartes-szorzat kifejtésére, sőt, valójában általában el kell kerülnünk, hogy nagy táblák esetén a Descartes-szorzat előállításra kerüljön. Ebben segítséget nyújthat az adatbázis-kezelő optimalizálója, de a lekérdezések készítőjének mindenképpen kell erre gondolkodnia.

Rendszám	Márka	Típus	Tulajdonos
EFF-362	Opel	Astra 1.4	1
JAU-725	Ford	Focus 2.0	2
LOL-962	BMW	X5	1

Azonosító	Név	Lakcím	Adószám	Magasság
1	Ádám	...	...	...
2	Éva	...	...	...

Rendszám	Márka	Típus	Tulajdonos	Azonosító	Név	Lakcím	Adószám	Magasság
...	Opel	...	**1**	**1**	Ádám	...	...	...
...	Opel	...	1	2	Éva	...	...	...
...	Ford	...	2	1	Ádám	...	...	...
...	Ford	...	**2**	**2**	Éva	...	...	...
...	BMW	...	**1**	**1**	Ádám	...	...	...
...	BMW	...	1	2	Éva	...	...	...

4.3. ábra. *Descartes-szorzat*

Aktivitás: Próbálja meg értelmezni a 4.3. ábra adatait: hány autója lehet Ádámnak?

A Descartes-szorzat kapcsán említettük a halmazokat. A relációs modell valójában mindig, mindenhol halmazokban gondolkodik. A halmazelméleti megközelítés mélyen benne van a modellben, és erősíti az elméleti gondolkodásmódot. A fizikai tárolásból azonban lista következik. Felhasználói igény is van a listák használatára, hiszen nagyon sok esetben számít a sorrend. A kinyert adatokat például eleve nem lehet halmazként, sorrendiség nélkül kinyomtatni, és legtöbbször nem „összevissza” sorrendben kapjuk a megfelelő eredményt, hanem meghatározunk egy bizonyos sorrendet (pl. a nevek ábécérendje szerint). Ezzel a lépéssel pedig a halmazból egy listát csináltunk. Ennek ellenére értenünk kell, hogy a relációs adatbázis logikai szinten halmazokkal, halmazműveletekkel dolgozik.

Vegyük észre, hogy az adatmodell logikai szinten gondolkodik és nem foglalkozik a fizikai tárolás problémáival.

A relációs adatmodellt megvalósító adatbázisokat RDBMS-nek (Relational DataBase Management System) nevezik.

4.4.2. Kulcs

A relációs modell szerint az adataink a táblák soraiban (rekordjaiban) vannak tárolva. Nagyon fontos, hogy két rekordot mindig meg tudjuk egymástól különböztetni.

4.4. definíció: Azon mezőt vagy mezőcsoportot, amik egyértelműen azonosítják egy reláció egyetlen sorát, kulcsnak nevezzük.

A kulcs tehát minden rekordra (sorra) egyedi és állhat egyetlen tulajdonságból, ilyenkor egyszerű kulcsról beszélünk (pl. személyi szám) de lehet összetett is, például gyártó és cikkszám. A felesleges redundancia elkerülése miatt fontos, hogy a jó kulcsnak nincs olyan részhalmaza, ami kulcsként használható lenne. Ez azt jelenti, hogy ha összetett kulcsból akármit elveszünk, már nem lehet kulcs. Példánkban például a cikkszám nem elegendő kulcsnak, mert nem lehetünk biztosak abban, hogy a 4B0905594H cikkszámot az Audin kívül más gyártó is használja-e, a gyártó megadásával azonban már biztos, hogy egyedi, ezért azonosításra alkalmas.

Másik példaként említsük meg a Matematika és Számítástudomány Tanszéket, ami szintén nem lehet kulcs, mert bármely más egyetemen is lehet azonos nevű tanszék. Így a tanszékek nyilvántartásában a tanszék neve + az egyetem neve lehet (összetett) kulcs.

Fontos, hogy a kulcs amellett, hogy minimális elemszámú, minimális méretű is legyen: az előző példában a tanszék és egyetem neve együtt bár lehet kulcs, de nem praktikus, mert igen hosszú és sok helyet foglal. Jobb megoldás például beszámozni az egyetemeket és a tanszékeket, és ezeket az egyedi azonosító számokat használni kulcsnak.

A kulcsérték egyediségének betartását a jól tervezett adatbázis is megköveteli. Az egyediségből következik, hogy a kulcsértéket minden esetben ki kell tölteni.

Ha minden rekordot tudunk azonosítani annak kulcsértékével, a kulcsok segítségével könnyűszerrel hivatkozhatunk egy tábla egyik rekordjára egy másik tábla valamely rekordjából.

4.5. definíció: Az idegen kulcs olyan mező, – amiben táblák összekapcsolása céljából – egy másik tábla kulcsértéke szerepel.

Az idegen v. kapcsoló kulcsnak (angolul foreign key) a következő feltételeket kell teljesítenie:

- értéke vagy üres (NULL),
- vagy pedig a hivatkozott tábla kulcsmezőjében előfordul.

Ezt a szabályt hivatkozási integritási szabálynak nevezzük.

A táblák közötti kapcsolatokat, az idegen kulcsok meglétét jelezni is lehet az adatbázisnak. A kulcs ugyanis felfogható egyfajta integritási feltételnek is (lásd 4.4.9. fejezet). Ha nem tesszük meg, akkor is működőképes lehet a rendszer, de nem várhatunk el integritásőrző (lásd ugyanott) és kényelmi automatizmusokat.

4.4.3. Kapcsolatok

A relációs adatmodell szerint a táblák közötti kapcsolatok adatértékeken keresztül valósulnak meg. Ez azt jelenti, hogy valamely tábla megfelelő mezőjébe egy másik tábla kulcsát írjuk, ami a kapcsolatot így egyértelműen leírja. (Erre vezettük be a 4.4.2 fejezetben az idegen kulcs fogalmát.)

A kapcsolatok leírásánál, tárolásánál nagyon fontos tudni az összekapcsolandó elemek számosságát. Általában nem is a pontos szám a fontos, hanem annak eldöntése, hogy pontosan 1 (vagy más nagyon kicsi egész számú) kapcsolódó elem lehetséges-e, vagy akár több is. Egy autónak például egyetlen alvázszáma lehet. Ez fordítva is igaz, egy alvázszám egyetlen autóhoz tartozhat. Ez a kapcsolat olyan közvetlen, hogy az alvázszám mező betehető az autó táblába az autó egyéb tulajdonságai közé (valójában csak ilyenek tehetők oda), és az ilyen típusú összerendeléseket, bár nevezhetnénk pl. 1:1 megfeleltetésnek, általában nem is szoktuk a kapcsolatok között tárgyalni. Ha mégis, akkor az 1:1 kapcsolat is kezelhető a 4.4.4. fejezetben leírt egy-több kapcsolathoz hasonló módon is. A továbbiakban megnézzük, hogy a különböző számosságú, bonyolultabb kapcsolatok hogyan kezelhetők.

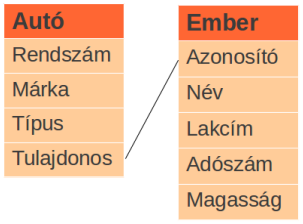
4.4.4. Egy-több kapcsolat

A személyek tábla megtervezésekor, ha a szülő-gyerek kapcsolatokat is tárolni szeretnénk, tudhatjuk, hogy mindenkinek pontosan egy anyja van, de a gyerekek számát nem tudjuk előre, a rendszert pedig e változó számú kapcsolat kezelésére kell felkészítenünk. Fontos észrevenni, hogy a kapcsolatban csak az egyik oldal (a gyerekek) száma változhat, a másik oldal (az anya) egyetlen elemből állhat. Az ilyen kapcsolatot egy-több vagy 1:N kapcsolatnak nevezzük.

Figyeljük meg a 4.4. ábra példáján, hogy a kapcsolatok adatértékek által valósulnak meg. Ebben a kapcsolattípusban abba a táblába, amelyből több elem is kapcsolódhat a másikba, egyszerűen beírjuk a másik tábla megfelelő kulcsát.

4.4.5. Több-több kapcsolat

Gyakori az olyan kapcsolat is, amikor az összekapcsolt elemek számosságáról azt mondhatjuk, hogy bármilyen kapcsolat lehetséges. Ilyen a vállalatok és személyek kapcsolata: egy cégnek akárhány alkalmazottja lehet, egy



Rendszám	Márka	Típus	Tulajdonos
EFF-362	Opel	Astra 1.4	1
JAU-725	Ford	Focus 2.0	2
LOL-962	BMW	X5	1

Azonosító	Név	Lakcím	Adószám	Magasság
1	Ádám	...	...	...
2	Éva	...	...	...

4.4. ábra. *Egy-több kapcsolat*

ember pedig több cégnél is dolgozhat. Az ilyen kapcsolatoknál nem tudjuk a kapcsolatokat közvetlenül a két adattáblában tárolni, egy segédtábla szükséges. Ebben a táblában a kapcsolatban álló előfordulások (rekordok) kulcsmezőit tároljuk.

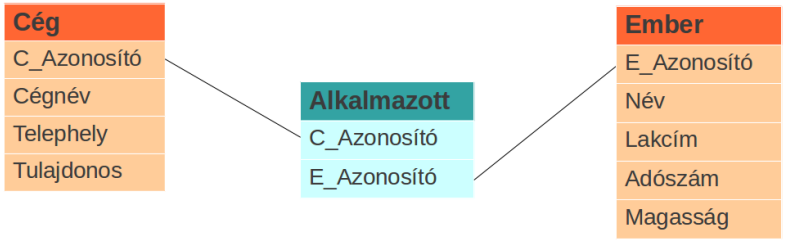
Aktivitás: Olvassuk le a 4.5. ábráról, hogy mely cégeknek mely személyek az alkalmazottai!

Aktivitás: Gondolkozzunk el azon, hogy a 4.5. ábra példáján mi lehet az Alkalmazott tábla kulcsa, illetve miért nincs külön létrehozott kulcsmező.

Ahogy a 4.4.2. fejezetben már említettük, e példa kapcsán is érdemes megfigyelni, milyen fontos, hogy a kulcsok lehetőleg kis méretűek legyenek. A kapcsolótábla méretét (és ezáltal használatának sebességét) a kulcsok nagymértékben befolyásolják.

4.4.6. Index

Ha egy fájlban tárolt adatot meg akarunk találni, annak alapvető módja, hogy a fájlt végigolvassuk, így előbb-utóbb eljutunk a keresett adatig. Ennek hétköznapi analógiája, hogyha egy tartalomjegyzék és tárgymutató nélküli könyvben akarunk valamit megtalálni: végig kell olvasnunk mindent, a keresett információ



C_Azonosító	Cégnév	Telephely	Tulajdonos
1	Béli Kft.	...	...
2	Béni Kft.	...	...
3	Béci Kft.	...	...

C_Azonosító	E_Azonosító
1	1
1	2
3	2

E_Azonosító	Név	Lakcím	Adószám	Magasság
1	Ádám	...	...	...
2	Éva	...	...	...

4.5. ábra. *Több-több kapcsolat*

kinyerési ideje gyakorlatilag csak a szerencsénktől függ: attól, hogy a könyv mely részén helyezkedett el. Nem nehéz belátni, hogy a keresési idő egyenesen arányos a könyv méretével: kétszer akkora könyvben átlagosan dupla annyi ideig tartana így valamit megtalálni. Ha különböző keresési lehetőségeket készítettek a könyvbe (tartalomjegyzék, névmutató, tárgymutató, ábrajegyzék, hivatkozások jegyzéke, táblázatok jegyzéke), akkor sokkal gyorsabban megtaláljuk, amit keresünk.

4.6. definíció: Az index az egyes táblákhoz kapcsolódó, a gyors keresést lehetővé tevő, az adatbázis-kezelő rendszer által automatikusan használt segédtáblázat.

Az indexelés épp ezt valósítja meg az adatbázisban. Az egyébként (hogyan máshogy?) sorosan letárolt adatok mellé általunk meghatározott meta-adatok gyors keresést tesznek lehetővé az adatokon. A hatékonyság miatt az indexek létrehozása nem automatikus, nekünk kell megmondani, milyen szempontok szerint fogunk keresni az adatbázisban. Egy történelmi könyvben bizonyára van értelme az évszámokhoz kapcsolódó mutatóknak, egy gyógyszerészeti kézikönyvben kevésbé. Ott inkább a hatóanyagokat érdemes kigyűjteni, hogy melyikről hol olvashatunk. Ezek a listák jellemzően nem az eredeti előfordulás sorrendjében, hanem rendezetten, ábécésorrendben vagy szám szerint növekvően tárolják a kigyűjtött adatokat.

Említettük, hogy az index létrehozása nem automatikus. Azonban ha egyszer definiáltuk, utána már az: az adatbázis-kezelő minden szükséges esetben aktualizálni fogja a benne lévő adatokat és a keresési műveleteknél szintén automatikusan használni is fogja azokat. Ez azt jelenti, hogy egy indexet egyszer létre kell hoznunk, onnan kezdve pedig gyakorlatilag nem kell törődnünk vele: szükség esetén segíteni fogja a munkánkat, észrevétlenül.

A könyves analógiát annyiban kell módosítanunk, hogy amíg ott például egy tárgymutatóba a szerkesztő egyedi döntése, hogy mely szavak kerüljenek bele, ha itt egy vagy több mezőre létrehozunk egy indexet, akkor a tábla minden egyes rekordjának megfelelő adata automatikusan bele fog kerülni.

Az index létrehozásával gyakorlatilag azt jelezzük az adatbázis-kezelő rendszer számára, hogy később milyen tipikus keresésekkel fogunk az adatbázishoz fordulni, így a szoftvernek lehetősége van ezekre a tipikus keresésekre a megfelelő listák előállításával felkészülni.

Nem szabad megfeledkeznünk arról, hogy az indexeknek – az elemszámtól és a kulcstól függően – jelentős helyfoglalása lehet és ebből adódóan plusz terhelést jelent a használata (nem csupán kereséskor, hanem beszúráskor, módosításkor, törléskor is). Az indexeket akkor érdemes használni, amikor ezek a hátrányok ellensúlyozni tudják a használatából adódó előnyöket.

Később látni fogjuk, hogy előny nem csak a sebességnövekedés lehet, hanem indexek segítségével integritási feltételeket is megfogalmazhatunk. Egy személyi nyilvántartásban sok indexet érdemes létrehozni, köztük valószínűleg érdemes lesz külön leindexelni a személyi számot. Mivel ez minden személy esetében egyedi, és ezt jelezzük is az index létrehozásakor (UNIQUE kulcsszóval), akkor ezt a szabályt az adatbázis-kezelő a későbbiekben minden esetben be fogja tartatni, természetesen automatikusan, és semmilyen körülmények között nem fogunk tudni tévedésből két embert felvinni ugyanazzal a személyi számmal. Hasonlóan azt is

jelezhetjük az index létrehozásánál, ha egy adott mezőt mindenképpen ki kell tölteni (NOT NULL).

Az indexek létrehozását a 4.10.1. fejezetben tárgyaljuk.

4.4.7. Adattípusok

Attól függően, hogy az adatbázisunk tábláinak mezőiben miféle adatot szeretnénk tárolni, meg kell adnunk az egyes mezők típusát. Az alaptípusok eldöntése az első lépés, de ezen gondolkodni sem igen kell: számot szám típusban, szöveges adatot szöveges típusban, stb. tárolunk. Ezeknek a csoportoknak is sok-sok precízebben megfogalmazott altípusa van: ezek célja a minél hatékonyabb adattárolás. Fontos tehát jól választani, ehhez pedig ismerni kell a választékot. Egy *igen-nem* értéknek elegendő 1 bit, vagy ha egy számot karakterláncként, "2,53625715621" formában tárolunk, akkor nehéz lesz vele matematikai műveleteket végezni.

Adatbázis-kezelőtől függően más és más adattípusok vannak, de ezek között természetesen sok az azonosság és nagy a hasonlóság. Konkrétumokért minden esetben az egyes adatbázis-kezelők dokumentációját kell átolvasnunk. Tipikus példák:

- Szöveges típusok
  - CHAR [(n)]
    Pontosan n hosszúságú karakterlánc, ha n nincs megadva, n=255.
  - VARCHAR [(n)]
    VARiable CHARacter
    Legfeljebb n hosszúságú karakterlánc. Csak a karakterláncnak megfelelő hosszúságú területet foglalja el. Maximum n, illetve 255 karakterig.
    Figyeljük meg, hogy ebben az esetben a maximális hossz megadása nem más, mint egy beépített ellenőrzés: ha tudjuk, hogy a szöveges adatunk maximálisan n karakterből állhat, akkor n megadásával biztosíthatjuk, hogy hosszabb, téves adat ne kerülhessen be az adatbázisba (bővebben lásd: 4.4.9. fejezet).
  - CLOB (m)
    Character Large OBject

Nagyméretű szöveg, az informatika szokásos mértékegységeivel: K (kilo-), M (mega-), G (gigabájt)
Például CLOB(5M) bőven elég egy Biblia méretű könyv teljes szövegének tárolására.

- Szám típusok
  - INTEGER, TINYINT, SMALLINT, BIGINT
    Egész értékek, különböző értéktartománnyal (8,16,32,64 bit)
  - NUMBER, NUMERIC, DECIMAL [(szélesség [, tizedes)]]
    Pontos értékű tizedestörtek tárolására. A szélesség mellett a tizedespont utáni jegyek száma is megadható.
  - REAL, FLOAT, DOUBLE
    Különböző mérettartományú lebegőpontos szám tárolására.
- Dátum/idő típusok
  - DATE, TIME, DATETIME/TIMESTAMP
    Dátum és/vagy időpont tárolására, akár időzónával együtt.
    DATE: '2012-05-01'
    TIMESTAMP '2012-05-01 16:10:05+1:00'
  - DATETIME
    Dátum és időpont tárolására
- Logikai típus
  - BOOLEAN
    TRUE, FALSE, UNKNOWN (NULL) érték.
- Bináris adattípusok
  - RAW/BINARY (n)
    fix méretű bináris adat

- BLOB (m)
  BINARY LARGE OBJECT (m) nagyméretű adat: az adatok értelmezésére nincs semmilyen megkötés, így segítségükkel tetszőleges adatokat, például fotókat tárolhatunk. A méret megadása mértékegységgel: K, M, G pl: BLOB(2G)
- RAW/BINARY
  tetszőleges bináris adatok, például képek tárolására

4.4.8. A NULL érték

Nagyon gyakori, hogy megengedhető, illetve nem elkerülhető, hogy egy mező kitöltetlen maradjon. (Például, ilyen lehet a férfiak esetében a leánykori név, vagy egy régi irat esetében a szerző ismeretlen volta.) Ilyen esetben a mező NULL értéket vesz fel. Ez egy olyan különleges érték, ami nincsen benne semmilyen értéktartományban. Nem azonos a 0-val és nem azonos a "NULL" karakterlánccal sem. A NULL értékkel bármely műveletet vagy összehasonlítást akarunk végezni az eredmény NULL lesz. Ezek alapján a NULL érték egy ún. „harmadik igazságértéknek” tekinthető, hiszen logikai adataink a NULL révén az IGAZ és HAMIS mellett egy „nem definiált”, vagy más szóval „ismeretlen” értéket is felvehetnek. Ezt nevezzük háromértékű logikának (3 Value Logic, 3VL). A szokásos Boole-algebrai műveleteket tehát ennek fényében kell értelmeznünk, bár nagyon egyszerű dolgunk van, hiszen bármely esetben, ahol legalább az egyik operandus NULL, az eredmény is az lesz.

Érdekesség: A fentiek alapján egyszerű összehasonlítással (pl. IF mezo=NULL) nem lehet eldönteni egy adatról, hogy NULL értékű-e, hiszen annak a műveletnek az eredménye is mindig NULL érték lenne. A probléma megoldását a későbbiekben, az adatbázis-kezelő nyelv tárgyalásakor meg fogjuk mutatni.

4.4.9. Integritási feltételek

Az adatbázis működéséhez az adatoknak sokféle feltételnek kell megfelelnie, ahogyan azt a 4.3. fejezetben megfogalmaztuk az elvárásaink között. Az integritási szabályok célja az adat-előfordulások megengedett körének a behatárolása. Ezeket a feltételeket különböző szinteken lehet megfogalmazni:

- mező szint,
- rekord szint,
- tábla szint,
- adatbázis szint.

Mezőszinten bármi mástól függetlenül az adott mező értékét szabályozhatjuk. Például, az életkor $>= 0$, az adószám pedig pontosan 12 számjegyből áll. Ilyenkor a mező tartalmára állított megkötéseket nagyobb összefüggéseiben nem vizsgáljuk.

Rekordszinten egy rekord elfogadhatóságáról döntünk. Személyi adatoknál rekordszintű feltétel lehet, hogy bizonyos életkorok alatt a különböző iskolai végzettségeket nem engedjük beállítani, hiszen egy 5 éves gyereknél az egyetemi végzettség nyilvánvaló adatbeviteli tévedésre utal, holott az egyes mezők egyenként megfelelőnek látszó értékeket tartalmazhatnak. A házastárs rovat pedig ne tartalmazhasson adatot, ha a családi állapot mező „nőtlen/hajadon” tartalmú.

A táblaszintű ellenőrzéshez a tábla összes rekordját figyelembe véve kell a táblát megvizsgálni. Alapvető feltétel lehet, hogy egy mezőnek egyedinek, tehát az adott rekordnak megkülönböztethetőnek kell lennie – a kulcsmező eleve ilyen, és sok példát tudunk rá mondani: egy autónyilvántartásban a rendszámnak, alvázszámnak és ugyanazon banknál a biztosítás kötvényszámának biztosan egyedinek kell lennie.

Táblaszinten más jellegű feltételek is megfogalmazhatók, például egy cég szabályzata alapján megfogalmazható, hogy az autói átlagéletkora a vásárlásuk pillanatában nem lehet több 6 évnél.

Az adatbázisszintű megkötések esetében az integritási feltételt több táblában elhelyezkedő mezőkkel fogalmazzuk meg. Itt fogalmazhatók meg a táblák közötti kapcsolatok, függőségek. Tipikus példa, hogy egy autónyilvántartásba csak ismert, létező típuskódú autó kerülhet be, azaz a kérdéses kódnak a típusokat tartalmazó táblában szerepelnie kell.

Az integritási feltételek adatbázisban történő beállítása „nem kötelező”. Ezek a feltételek a kliens programokba is beépíthetők, például az adatfelviteli és adatmódosító űrlapok mögé. Ott azonban sokkal több munka lenne megvalósítani ezeket, és mivel ugyanazokon az adatokon többféle kliens is dolgozhat, sosem tudhatjuk, hogy pontosan ugyanazokat a feltételeket valósítja-e meg minden kliens – összességében egy kevésbé

megbízható eredményre jutunk, sokkal több munkával. Ezért az integritási feltétek adatbázis-szinten történő megfogalmazása mindenképpen ajánlatos.

Az integritási feltételek megadásának módját az adatbázis-kezelő nyelvének ismeretében a 4.10.1. fejezetben fogjuk tárgyalni.

4.5. Redundancia

4.7. definíció: A redundancia (terjengősség) a felesleges (és/vagy kiszámítható) adatok mértéke.

A *redundancia*, bár sokaknak talán idegenül hangzik, egészen hétköznapi fogalom. Azt fejezzük ki vele, hogy valamely információ mennyire szűkszavúan, vagy épp ellenkezőleg, mennyire terjengősen van megfogalmazva. Mindannyiunknak van tapasztalata abban, hogy ugyanazt a mondanivalót egyszer hosszan elnyújtva kell kifejtenünk, például azért, hogy a minimális oldalszámot elérjük, máskor pedig tömören kell fogalmaznunk (például, mikor SMS-t írunk).

Az emberi nyelvek általában kimondottan terjengősek, ebben jelentős különbség van a nyelvek között, a magyar nyelv egyike a legkifinomultabbaknak, így szójátékok végtelen sorát kínálja, többek között a redundancia megfigyelésére. Ismerős gyerekkori játék csupán magánhangzókkal beszélni és kitalálni, mit mondott a beszélő. A dolog fordítva is meglepően jól működik: a magyar nyelvben a magánhangzók alig hordoznak jelentést:

Érdekesség: Akar falyamatasan as lahat magyaral baszalna, i misik migyir migis migirti, higy mit is ikirink möndönö ö mösök mögyörnök. *(Kiss Dénes nyomán)*.

Tekintsük a következő mondatot: *Szóljatok már nekik, hogy majd hívjanak meg engemet.* Próbáljuk meg lefordítani az általunk ismert nyelvekre ! Utána pedig fogalmazzuk meg ezt kicsit tömörebben, magyarul: *meghívattathatnátok.*

Próbáljuk meg megérteni, milyen előnyökkel és hátrányokkal jár a terjengősség:

4.6. ábra. *Két különböző módon tárolt számsor, sérülés után. A redundancia néha hasznos.*

Az biztos, hogy ha a fenti mondatot egy zajos telefonvonalon mondjuk ki, a beszélgető partnerünk sokkal könnyebben megérti, és kisebb eséllyel érti félre, mintha az azt helyettesítő egyetlen szót mondanánk ki. Az egyetlen szóból álló változatból ha csak egy töredéknyi rész is elvész, a hallgató már biztosan nem érti meg a kérésünket. Ha jól hall minket, akkor is bizonyára gyorsabban, biztosabban felfogja a kívánságunkat, mintha az egyébként frappánsan megfogalmazott egyetlen szót mondtuk volna ki. Az tény azonban, hogy az üzenet hosszában jelentős különbség van.

Tekintsük a 4.6. ábrát, amin a jövő heti lottószámokat súgta meg nekünk valaki. Kár, hogy a papír úgy elmaszatolódott, hogy alig olvasható. Szerencsénkre azonban ugyanaz az adat kétféleképpen is a papíron volt: számjegyekkel és szövegesen. A két sor információtartama pontosan megegyezett, de jól látszik, hogy a hosszabb, redundánsabb adattárolási mód nagy segítségünkre van abban, hogy a sérült adatokat mégis értelmezni tudjuk.

Az előző példákból talán úgy tűnik, hogy a redundancia egy előnyös dolog. Ez az esetek egy jelentős részében így is van. Ha valamely értékes adatunk elvész, például az iskolában felejtjük a jegyzetünket, felbecsülhetetlen érték, ha volt belőle másolat (fénymásolat). Sok esetben azonban hátrányt jelenthet: senki nem szereti a sokat fecsegő embereket.

Az adatbázisunkban tárolt adatok redundanciájára kissé másféleképpen kell tekintenünk. Az adatbázis néhány fájlban tárolja összes adatát, és olyan típusú adatvesztés, mint ami egy elmosódó kézírásnál vagy zajos telefonbeszélgetésnél megtörténhet, a számítógépes környezetben nem fordulhat elő. Ezért nem is ilyen típusú redundanciában kell gondolkodnunk, ha az adataink visszanyerhetőségén gondolkodunk. Arra az lesz a jó megoldás, hogy például mentést, másolatot készítünk a teljes adatbázisról, így hiba esetén sem veszhetnek el

Azonosító	AzonIgTip	Név	Születés	Életkor	Lakhely	Foglalkozás
419953LA	személyi	Okoska	1963	49	Aprajafalva	Rajzfilmhős
623712LC	személyi	Törpojáca	1962	50	Aprajafalba	Rajzfilmhős
A674621	útlevél	Zokogi	1965	47	Aprajafalva	Képregényhős
B284743	útlevél	Ügyi	1965	47	Aprajafalva	Rajzfilmhős

4.7. ábra. *Egy rosszul tervezett tábla*

az adataink. A napi szinten használt adatbázisunkban a redundancia azonban sok esetben csak nehézségeket okoz, ezért tanuljuk meg a redundanciát jól kézben tartani és számunkra hasznosítani.

Aktivitás: Tekintsük a 4.7. ábrát, amin egy több szempontból rosszul tervezett táblát mutatunk be. Fogalmazzunk meg minél több konkrét problémát a táblaszerkezettel kapcsolatosan!

A 4.7. ábrán például a következő problémákat találhatjuk:

- Életkor: felesleges, hiszen a születési évből kiszámítható. Egy év múlva ráadásul már hibás adatokat fog tartalmazni.
- Lakóhely: Az, hogy minden rekordba begépelésre került egy városnév, fölösleges gépelést, helyfoglalást és tévesztési lehetőséget jelent
- Ha az igazolvány típusánál a „személyi” szót a hivatalosabb „személyi igazolvány”-ra cserélnénk, ezt több helyen kell megtenni, ami feleslegesen sok munka és tévesztési lehetőség (kifelejtünk valamit, illetve nem vesszük észre, hogy máshol „szem.ig”-nek hívták ugyanazt az igazolványt)
- Listázzuk ki a rajzfilmhősöket! Zokogi biztos kimarad, mert az ő foglalkozása másképpen van megfogalmazva
- Aprajafalva lakóit keresve Törpojáca fog kimaradni egy elgépelés miatt.

A továbbiakban azzal foglalkozunk, milyen szisztematikus módszerrel lehet elkerülni, illetve javítani az itt bemutatott adatbázis-tervezési hibákat.

4.6. Normalizálás

Ugyanarra a feladatra több, egymástól kisebb vagy nagyobb mértékben különböző adatmodell tervezhető. Ezek a modellek nem egyformán „jók”: különböző hatékonyságúak és tartalmazhatnak bizonyos logikátlanságokat, (szépség)hibákat, amelyek a működés hatékonyságát csökkenthetik. A relációs modell megtervezésekor a fizikai megvalósítás hatékonyságára illetve a kezelés hibalehetőségeire, hatékonyságára is gondolni kell.

4.8. definíció: A normalizálás az az adatbázis-tervezési folyamat, ami során a relációs adatmodellnek *jól* megfelelő, optimálisnak tekinthető táblaszerkezetet hozunk létre.

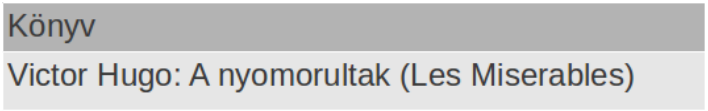
A normalizálás célját úgy is megfogalmazhatjuk, hogy olyan táblákat hozzunk létre, amikben csak a kulcsra vonatkozó tényeket tárolunk.

A normalizálás eredménye: áttekinthető, praktikusan használható, redundancia-mentes táblák. A redundancia megszüntetését általában táblák (több táblára történő) felbontásával érjük el (dekompozíció). A normalizálás többlépcsős, egymásra épülő folyamat, és fontos észrevenni, hogy az ajánlásokat, bár az esetek nagy többségében javítják az adatbázis minőségét, „nem kötelező” minden esetben maradéktalanul elfogadni: lehetnek speciális esetek, amikor *tudatosan* úgy döntünk, hogy mégis egy redundáns, valamely normálformának ellentmondó megoldás mellett maradunk. Emiatt is, és a normalizálási lépések maradéktalan betartása esetén is több különböző, de egyformán jó megoldás is születhet, nincs „legjobb” megoldás.

Több normálforma ismert, a tananyagban a három legegyszerűbbel foglalkozunk.

Első normálforma

Tekintsük a 4.8. ábrát. Könyvek szerzőjét és címét tároljuk a táblában.



4.8. ábra. *Egy javítandó tábla*

Aktivitás: Mielőtt tovább olvasnánk, próbáljuk magunk megfogalmazni, mi a baj a táblaszerkezettel!

A tábla, ha így próbálnánk meg használni, sokféle problémát felvet: semmiképp nem praktikus, hogy több különböző (összesen háromféle) adatelem egyetlen mezőben szerepel – Például, hogyan fogunk tudni a szerző nevére keresni? Egyszerű szóelőfordulás-kereséssel megtalálnánk a címben a szerző nevét tartalmazó, épp róla szóló könyveket is. Hasonlóan az eredeti és magyarra fordított cím is összekeveredhet – a használt kettőspont és a zárójelezés sem megoldás, mert ezek bármelyike szerepelhet magában a címben is. Ezekre a problémákra javasol triviális megoldást az első normálformának nevezett szabály: ne használjunk olyan mezőt, amiben több érték szerepel egymás mellett.

4.9. definíció: Első normálformában van a tábla, ha teljesül, hogy minden mezője elemi értéket hordoz.

Ezen egyszerű szabály alapján a 4.9. ábrán látható táblát alakíthatjuk ki: új mezők létrehozásával elemi értékek tárolására való mezőket hoztunk létre. Figyeljük meg, hogy az elemiség megítélése szubjektív és feladatfüggő: más jó megoldás is lehetett volna, mi most ezt választottuk.

Aktivitás: Mielőtt tovább olvasna, találjon ki másik megoldást is és fogalmazza meg, annak milyen előnye van a bemutatottal szemben!

Másik módja lehet az elemi mezők kialakításának, ha a szerzők neveivel sokat foglalkozunk a feladatunkban, valószínűleg érdemes a vezetéknevet és keresztnevet külön mezőbe rendezni. Így sokkal könnyebb lesz a névre vonatkozó kereséseket, vagy a szerzők vezetékneve szerinti ábécésorrendben történő listázást megoldani.

Író	Cím_eredeti	Cím_magyar
Victor Hugo	Les Miserables	A nyomorultak

4.9. ábra. *Első normálformának megfelelően átalakított tábla*

Gondolnunk kell azonban mindenre: ha külön tárolunk kereszt- és vezetéknevet, meg kell terveznünk, mit csinálunk a kettőnél több szóból álló szerzőnevekkel, és fontos, hogy a nevek eredeti írási sorrendjét is tároljuk (mint köztudott, különböző nemzetek különböző sorrendben használják családi és utóneveiket).

A következőkben az egyetemi teremfoglalás (teremórarend) rendszeréhez tervezzük meg az adatmodellt. Tegyük fel, korábban egy Excel táblázatban tárolt lista jelentette a nyilvántartást, tekintsük most ezt kiindulópontnak (4.10. ábra).

Terem	Előadás
D1, földszint (600 fő)	Üzleti informatika (NGB_SZ_003_9)

4.10. ábra. *Teremfoglalás nyilvántartása*

Az előzőek alapján az egy mezőben tárolt, de önmagában is jelentéssel bíró adatokat külön mezőkbe rendezzük, ahogyan az a 4.11. ábrán látható. Ezzel teljesítettük is az első normálformát.

Terem	Kapacitás	Emelet	Tárgy	Tárgykód
D1	600	0	Üzleti informatika	NGB_SZ_003_9

4.11. ábra. *Első normálformának megfelelően átalakított tábla*

Második normálforma

A normalizálás következő lépésében már az egyes mezők kulcs-szerepét is figyelembe kell vennünk. Határozzuk meg tehát, mi lehet kulcs ebben a táblában: teremfoglalásról lévén szó, kulcs a terem és az időpont azonosítója együttesen. (Az előző ábrákon az időpont jelölése még nem is szerepelt, ezentúl jelöljük, a példán látható K5 pedig kedd 5. órát jelenti.) A kulcsmezőket a továbbiakban vastagított mezőnévvel jelöljük. A táblánk, kicsit átalakítva jelenleg tehát a 4.12. ábra szerint néz ki:

Előadóterem	Méret	**Időpont**	Tárgy	Tárgykód
D1	600	K5	Üzleti informatika	NGB_SZ003_9

4.12. ábra. *A kulcsokat jelölő és időpontot is tartalmazó tábla*

4.10. definíció: Második normálformában van a tábla, ha az első normálforma teljesül és minden nem-kulcs mező függ a teljes kulcstól.

A mezők tartalma már elemi, de a nem-kulcs mezők több esetben is nem a teljes kulcstól függenek. Például, a teremméret, mint nem-kulcs mező, nem a teljes kulcstól (Előadóterem + Időpont) függ, hanem a kulcsnak csak egy részétől – ezeket az állapotokat kell most megszüntetnünk. Erre a kézenfekvő megoldás a több táblára bontás (idegen szóval dekompozíció), amivel könnyen elérhetjük, hogy minden tábla minden nem-kulcs mezője valóban csak a kulcsmezőtől (vagy kulcsmezőktől) függjön. Ezt láthatjuk a 4.13. ábrán.

Harmadik normálforma

4.11. definíció: Harmadik normálformában van a tábla, ha a második normálforma teljesül és nincs függőség a nem-kulcs mezők között.

Előadóterem	Méret
D1	600

Előadóterem	Időpont	Tárgy	Tárgykód
D1	K5	Üzleti informatika	NGB_SZ003_9

4.13. ábra. *Második normálformának megfelelően felbontott táblák*

A 4.13. ábra második táblázatán épp ezt látjuk! A tantárgyak neve és kódja között természetesen van összefüggés, a tárgy neve függ a tárgykódtól. A kézenfekvő megoldás ismét a több táblára bontás, mégpedig úgy, hogy az eredeti táblában idegen kulcsként az új tábla kulcsát helyezzük el (4.14. ábra).

Előadóterem	Méret
D1	600

Előadóterem	Időpont	Tárgykód
D1	K5	NGB_SZ003_9

Tárgykód	Tárgy
NGB_SZ003_9	Üzleti informatika

4.14. ábra. *Harmadik normálformának megfelelően felbontott táblák*

További bonyolultabb esetekkel és az azok problémáit kiküszöbölő normálformákkal nem foglalkozunk. Jegyezzük meg, hogy az első három normálformát 1NF, 2NF és 3NF formában szokták rövidíteni.

Önellenőrzés

1. Tervezzünk adatbázis-táblákat az otthoni könyvtárunk adatainak kezelésére.
2. A megtervezett könyvtári táblák mellé – ha eddig nem tettük volna meg – jelöljük ki a kulcsmezőket és tervezzük meg, mely mezőkre lesz érdemes indexelést használni.
3. Mutassuk be, hogy miként teljesítik tábláink a második normálformát.
4. Gondolkozzunk el, ha az volna a feladat, hogy a régi kartoték-alapú rendszert vezessük be, számítógépes megoldás helyett, milyen adatokat írnánk az egyes kartonlapokra.

19. LECKE

Adatbázis-kezelés a gyakorlatban

4.7. Ismert adatbázis-kezelők

Az adatkezeléssel kapcsolatban eddig megfogalmazott elveket, elvárásokat sok-sok szoftver teljesíti. Ha csak a relációs adatmodellt használó adatkezelő programokat vesszük számításba, akkor is sokféle szoftver (RDBMS, Relational DataBase Management System) közül válogathatunk. Ezek puszta felsorolása is nehézségbe ütközne és valójában nem is válna hasznunkra, azonban a relációs adatbázis-kezelő piac legjelentősebb képviselőit név szerint is megemlítjük.

Az egyik legnagyobb név az adatbázis-kezelők között az Oracle. Az 1977-es alapítású cég sokféle üzleti szoftvermegoldást kínál, ezek természetesen mind a legalapvetőbb termékükre, az adatbázis-kezelőre épül, amit önállóan is árusítanak. Elsősorban a nagyvállalati környezetre koncentrálnak. Felmérések szerint a világ összes adatbázis-kezelőjének a kb. fele (!) Oracle termék. Az Oracle Database etalonnak számít elképesztő funkcionalitásával. Minden komoly szerver-operációs rendszeren fut. Szokták mondani, hogy amit ez a rendszer nem tud, arra biztos hogy nincs szükség...

4.15. ábra. *Ismert adatbázis-kezelők logói*

A Microsoft SQL Server nevű terméke az 1980-as évek végére nyúlik vissza, amikor a Sybase felvásárlásával szert tettek egy komoly adatbázis-kezelőre, amit utána az új néven saját maguk fejlesztettek tovább. Elterjedtségének legfőbb oka a Microsoft egyéb termékeihez való kötődés. A Microsoftra jellemzően csak Windows környezetben fut, amivel sok-sok felhasználásból eleve kizárta magát. Windowsos irodai környezetben azonban sokszor találkozhatunk vele.

A PostgreSQL talán a legfejlettebb adatbázis-kezelő a szabad szoftverek piacán. Szintén a 80-as években indult a fejlesztése, Ingres néven, Nagyjából mindent tud, amit fizetős társai, és természetesen sokféle platformon elérhető.

Végül megemlítjük a MySQL rendszert, amit az 1990-es évek közepe óta fejlesztenek és a legelterjedtebb

adatbázis-kezelők között van. Többféle licenszeléssel, a legtöbb esetben ingyen elérhető. Ingyenessége úgy is megmaradt, hogy felvásárolta a Sun, a Sun-t pedig bekebelezte az Oracle – azaz a MySQL ma gyakorlatilag Oracle termék, a „saját" adatbázis-szerverük kistestvére. Olyan rendszerek futnak ezen az adatbázison, mint például a Wikipedia, Facebook vagy Twitter. (A Google is több szolgáltatáshoz használja, de tudni lehet, hogy épp a kereséseket egy saját fejlesztésű, és *nem* relációs adatbázis-kezelő rendszer szolgálja ki.) Kb. 30 operációs rendszeren elérhető. Nagy port kavart, mikor jópár éve a NASA a korábbi (Oracle) adatbázisait MySQL szerverekre cserélte.

4.7.1. Desktop adatbázis-kezelők

Folyamatosan megismerjük, milyen felépítésű és működési logikájú egy adatbázis-szerver. A szerverekhez pedig az egyedi feladatokat megvalósító kliensprogramok kapcsolódnak. Ezeket a programokat futtatják például a gyárak adminisztrátorai, vagy a bankok és hivatalok ügyfélszolgálatosai.

Az ilyen szoftvermegoldásokat jellemzően profi informatikus és szoftverfejlesztő szakemberek tervezik meg és készítik el, nekünk ebben a tárgyban nem ilyen jellegű tudás elsajátítása a célunk. Az egyetem elvégzése után ha egy termelő nagyvállalathoz, adóhivatalhoz vagy pénzügyi szolgáltató céghez megyünk dolgozni, minden bizonnyal a cég legfőbb tevékenységeire a már mások által megtervezett, elkészített vagy megvásárolt adatbázis-kliensprogramokat kell majd használnunk, nem kell az adatbázissal magával foglalkoznunk.

Azonban az élet sok területén képzelhető el, hogy mégis olyan jellegű adatkezelésre kényszerülünk saját magunk, ami már túlmutat az egy-két Excel fájlban tárolható adatok bonyolultsági szintjén. Ilyenkor nagy segítség, hogy a kurrens irodai programcsomagok (pl. Microsoft Office, LibreOffice) tartalmaznak saját adatbázis-kezelő szoftvert, ezek használata pedig sokkal kevesebb szakmai jártasságot igényel „komolyabb" társaiknál.

Az adatbázis-kezelők használatát a tárgyhoz kapcsolódó gyakorlatokon tehát ún. desktop típusú adatbázis-kezelővel fogjuk gyakorolni. Ez az az adatbázis-kezelő típus, ami a nem-informatikusok számára is a legegyszerűbben telepíthető, kezelhető, ezért ők egyedi adatbázis-kezelési feladatok megoldására valószínűleg a későbbiekben is ilyen eszközhöz nyúlnak.

A desktop típusú adatbázis-kezelők is a megismert elvek szerint működnek, a nagy különbség alapvetően

4.16. ábra. *Két ismert desktop adatbázis-kezelő (Microsoft Access, Libreoffice Base)*

a felhasználási területből adódó egyszerűsítési lehetőségekben keresendő. A saját irodai gépünkön sokkal kisebb adatmennyiséggel, kevésbé komplex adatokkal, kevesebb jogosultsági kérdéssel kell foglalkoznunk, így a sok-sok apróságban egyszerűsített rendszer használata a kezdő számára áttekinthetőbb.

Sok szoftvert készítettek erre a felhasználási területre, elsőként említést érdemel a DBase rendszer. Ennek megjelenése szintén az 1980-as évek elejére tehető, kimondottan asztali számítógépekre, kisebb feladatokra tervezték. Valójában annyira egyszerű lett a rendszer, hogy definíció szerint nem is lenne adatbázis-kezelőnek tekinthető, mert sok alapvető funkciót eleve nem valósítottak meg benne, másokat pedig csak „kézzel" lehet elvégezni benne, amit egy valódi RDBMS automatikusan megold. Inkább nevezzük csak adatkezelő szoftvernek.

Fájlformátuma elterjedtsége okán kvázi-szabvány lett, a mai napig sok program felismeri, egyszerűbb adatátvitelekhez használható és természetesen visszaszorulóban van.

Amíg a DBase kizárólag parancssori, karakteres felülettel rendelkezett (és egyedi programokat kellett írni a használatához), addig a FoxPro előbb szöveges interfésszel (valójában karakteres, de karaktergrafikai elemeket, ablakokat, egeret is használó felület) és kényelmes, saját fejlesztőrendszerrel is rendelkezett. Korábban egy különálló cég fejlesztette, később felvásárolta a Microsoft, majd az 1990-es évek közepén felhagyott a fejlesztésével.

Szintén az 1990-es évek technológiájának jelentős képviselője a Borland Database Engine, amit Windows alkalmazások adattárolásának támogatására fejlesztettek ki.

Az említett rendszerek a szépen lassan eltűntek, jelenleg a legismertebb hasonló szoftver a Microsoft Office programcsomag Access nevű adatbázis-kezelője, az ingyenes konkurens LibreOffice programcsomagban pedig a Base néven találunk otthoni használatra szánt adatbázis-kezelőt. Ezek a szoftverek saját, „beépített" adatbázis-kezelővel is rendelkeznek, de természetesen kliensként is használhatók: a rendelkezésre álló kezelőfelületen keresztül, vagy akár saját fejlesztésű kliensalkalmazással is használhatunk másik, külső, komolyabb adatbázis-kezelő rendszereket is a segítségükkel.

A kettő közül egyértelműen az Access a komolyabb tudású program, de épp egyszerűsége miatt a továbbiakban példáinkban a LibreOffice Base-t használjuk. Többféle operációs rendszeren elérhető és ingyenes, így nyugodt szívvel kérhetünk telepítést és gyakorlást a hallgatóinktól, a kedves olvasótól. A két szoftver működési logikája, képernyőképe egyébként sok esetben szinte megtévesztésig hasonló, az egyik szoftverrel szerzett tapasztalatainkat könnyűszerrel hasznosíthatjuk a másik termék használata esetén is.

Külső adatbázishoz való kapcsolódásnak kétféle módja lehetséges:

1. A legfontosabb adatbázisokhoz saját interfészt biztosít
2. A többi esetben ODBC (Open DataBase Connection), JDBC (Java DataBase Connectivity): szabványos programozói interfész (API) biztosításával gyakorlatilag bármely adatbázishoz való kapcsolódást lehetővé tesz.

A desktop adatbázis-kezelők tehát három terméket tartalmaznak egybeépítve:

1. adatbázis-szervert,
2. adatbázis-klienst,
3. fejlesztőeszközt egyszerűbb adatbázis-kezelő alkalmazások készítésére.

4.8. SQL nyelv

A 4.3. fejezetben megfogalmazott elvárásokat akkor tudja egy adatkezelő rendszer legjobban teljesíteni, ha fejlesztője nem az egyes feladatokra készít egyéni megoldásokat, hanem nagyobb munkával bár, de elkészít egy

általános célra használható, átgondolt rendszert, aminek segítségével az egyéni feladatok már hatékonyabban, gyorsabban megoldhatók. A relációs adatbázisok egyik alapötlete is éppen ez az egységesítési törekvés volt. Természetes, hogy az elméleti modell mellett gyakorlati, valódi nyelvrendszert is ki kellett fejleszteni – a történelmi kitérőkkel nem törődve mondjuk azt, hogy a relációs adatbázis-kezelők szabványos kezelőnyelve a Structured Query Language, rövidítve az SQL lett. A kifejezés jelentése strukturált lekérdezőnyelv, a strukturáltság az egységességet, a tiszta logikát jelzi, a *lekérdezés* kifejezés pedig onnan jön, hogy az adatbázis általános célja, hogy benne tárolt adatokat *kérdezzünk le* belőle – természetesen, nem csak a tényleges lekérdezések részei az SQL-nek, hanem minden, az adatbázis-kezelő számára küldött parancs (például adatbevitel) is.

4.12. definíció: Az SQL a relációs adatbázis-kezelő rendszer (RDBMS) szabványos kezelőnyelve.

A különböző cégek munkája, tapasztalata tehát szabványban egyesült – a fejlődés miatt azonban maga a szabvány (ANSI és ISO szabványokról beszélünk) sokszor módosult, több lépcsőben fejlődött. A szabványok kidolgozói azonban gondosan ügyeltek arra, hogy az új nyelvi elemek a korábbiakkal összeegyeztethetők legyenek – az SQL nyelvvel kezdő szinten dolgozók számára nem is nagyon fontos, hogy pontosan melyik szabvánnyal dolgoznak, számukra elég **az** SQL nyelv ismerete.

Az egyes szoftvergyártók, bár alapvetően elfogadták az SQL szabványt, különböző SQL szabványt vehetnek alapul és nem biztos, 100%-osan meg is valósítják annak minden részletét. Ráadásul megtörténhet, hogy apró egyéni ötleteikkel itt-ott bővítenek a szabványokon – így aki sokat dolgozik egy bizonyos termékkel, szépen lassan annak a nyelvét ismeri meg. Nem kell nagy különbségekre gondolni, az SQL változatok igen egységesek, az egyik rendszer ismeretével a másik gond nélkül használható, legfeljebb belefutunk egy-egy szintaktikai hibát okozó apróságba, amit könnyűszerrel javíthatunk. Ilyen apróság például, hogy az SQL nyelvben és az egyszeres (') vagy dupla idézőjel (") jelzi a karakterláncok elejét és végét, de van olyan adatbázis-kezelő, ami erre a célra csak az egyszeres idézőjelet (') fogadja el.

Ezen tananyag hátralévő részében is lehetnek jelentéktelen eltérések a tananyag példái és az általunk épp használt adatbázis-kezelő között, a példákat értelemszerűen használjuk.

Az SQL tehát a relációs adatbázis kezelőnyelve, és mivel csak relációs típusú adatbázisokkal foglalkozunk,

ezért sokszor el is hagyjuk e jelzőt. Az SQL nem önálló szoftver, hanem mindig az adott adatbázis-kezelőhöz kapcsolódik, annak integrált része, illetve magának a nyelvnek a neve.

Az SQL nyelv a relációs kalkulus logikájára épül: nem azt kell leírni, hogy hogyan kapjuk meg a kívánt adatokat, hanem azt, mely adatokat szeretnénk látni. A kért adatok konkrét előkeresésének módját az adatbázis-kezelőre bízzuk. A nyelv úgy lett kialakítva, hogy a parancsok formalizált angol mondatok – minimális angol nyelvtudással az egyszerűbb SQL parancsokat nagyon könnyű értelmezni:

SELECT name FROM students WHERE neptunid=”ABC123”

A fenti mondat egy SQL parancs, és az angolul tudók azonnal értik, hogy azon hallgató nevét keressük ki a hallgatók táblából, akinek a neptunkódja ABC123. Az SQL-t egyszerű szintaktika jellemzi és a parancsokban a kis- és nagybetűket nem különbözteti meg (a könnyű olvashatóság érdekében a példákban az SQL nyelvi elemeket nagybetűvel fogjuk írni). Egy-egy SQL parancs nagyon hosszú lehet, több tucat sorból is állhat, ezért a parancsok végét érdemes jelölni, a szabvány szerint ez az interaktív karakteres felületeken a pontosvessző (;). Ebben az anyagban is fogunk pontosvesszőt használni a többsoros parancsoknál, ez egyértelművé teszi, hogy a sorok egyetlen parancsként értendőek.

Figyeljük meg, hogy a fenti parancs nem mondta meg, hogy pontosan milyen algoritmus mentén lehet a kérdéses hallgató nevét megtalálni, csupán egy „kérdést” tett fel, és annak megválaszolási módját az adatbázisra bízta. Maga az SQL nem is tartalmaz algoritmikus elemeket, például ciklusokat, vagy feltételes elágazást: ezekre ha szükség van, akkor az SQL parancsokat valamely más környezetbe, például programnyelvbe ágyazva kell kiadnunk.

4.9. Kommunikáció az adatbázis-kezelő rendszerrel

Tekintsük át, milyen módokon adhatunk SQL parancsokat az adatbázis-kezelő számára:

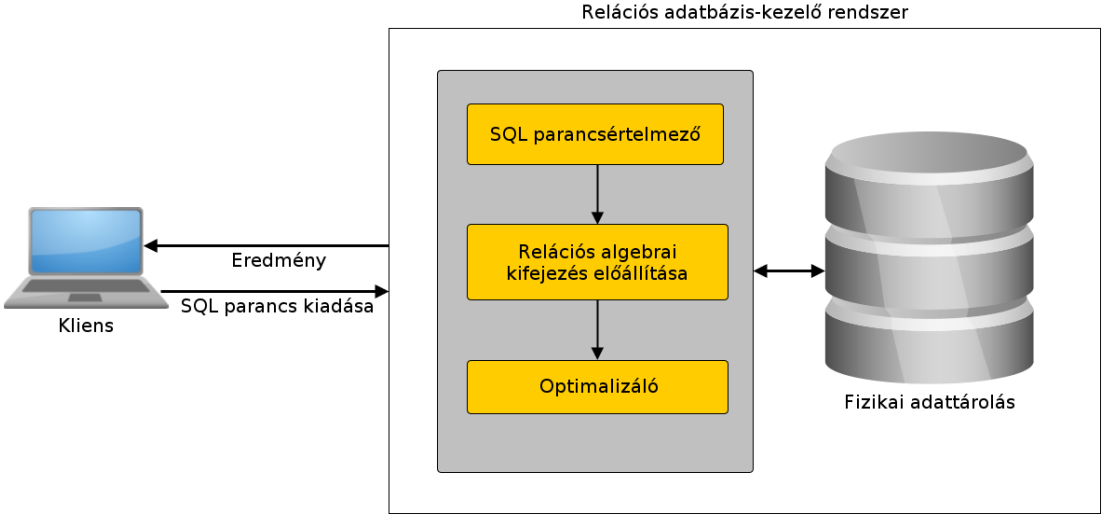
1. Fejlesztőeszközök, programnyelvek
   SQL parancsok segítségével kommunikálhatunk az adatbázissal fejlesztőeszközökbe, procedurális programnyelvekbe (C, Java, PHP) és ezáltal felhasználói programokba építve (például űrlapok, jelentések, listák készítésénél). A fejlesztőeszközök sok esetben legalább részlegesen elrejtik az SQL parancsokat

a felhasználó elől, ezért bizonyos esetekben SQL ismeretek nélkül is használhatók, de a grafikus felhasználói felület mögött a háttérben ugyanazok az SQL parancsok futnak le, és bonyolultabb esetekben a felhasználónak is lehetősége van a fejlesztőeszköz által generált SQL parancsokon módosítani. Programnyelvekbe ágyazás esetén egyértelmű, hogy a programozónak kell az SQL parancsokat megterveznie.

2. Interaktív , közvetlen gépelés
   Minden adatbázis-kezelőnek van egy alapvetően karakteres kliensfelülete, ahol egyszerű begépeléssel közvetlenül kiadhatók az SQL parancsok. Nagyon fontos lehetőség, de rendszergazdáknak, hozzáértőknek való egyedi feladatok elvégzésére.

3. SQL scriptek
   Az előző pontban említett közvetlen parancskiadás sokszor ismétlődő lépéseket jelent, ezért lehetőség van az SQL parancsokat ún. scriptekbe is szokás. Ezen scriptek néha az adatbázis-kezelő saját programnyelvét jelentik, máskor pedig valamely külső, általános nyelvet. Ilyen módon tipikusan az ismétlődő rendszergazdai feladatokra, például adatbázis-mentések vagy adat importálási/exportálási feladatok automatizálására használják.

Az előző módok bármelyikét is használjuk tehát, az adatbázis felé mindenképp egy szöveges formájú, SQL nyelven megfogalmazott parancs megy – akár mi magunk fogalmaztuk meg a parancsot, akár az általunk használt program adta ki (4.17. ábra). Az adatbázis-kezelő rendszeren belül az SQL parancsunkat az ún. SQL parancsértelmező kapja meg, ami, amint a neve is mutatja, értelmezi az általunk megfogalmazott parancsot. Ez a szoftverkomponens jelezné azonnal, ha például hibás, tehát végrehajthatatlan parancsot adtunk ki. Ha a parancsot helyesnek találja, elő kell állítani az adatbázis-kezelő belső felépítésének megfelelő algebrai kifejezést – ennek mikéntje már az adatbázis-kezelő rendszer belső ügye. Ezután fontos momentum az optimalizálás: ugyanis, mint korábban már említettük, az SQL nem procedurális nyelv: nem mondjuk meg, hogy pontosan mit, hogyan csináljon az adatbázis-kezelő, hanem azt mondjuk meg, hogy milyen eredményt szeretnénk látni. A megvalósítás megtervezése az adatbázis-kezelő rendszer feladata, és itt az előforduló hatalmas adatmennyiségek miatt hatalmas szerep jut a jó optimalizálásnak. Emlékezzünk arra, hogy két, egyenként 10 millió rekordot tartalmazó tábla Descartes-szorzata 10^{14} rekordot eredményezne, ha az optimalizáló nem figyelné, hogy egy ilyen műveletet pontosan hogyan, milyen sorrendben *kell* végrehajtani. Amikor a belső



4.17. ábra. *A kommunikáció folyamata*

adatmodellnek megfelelően, elkészült egy optimálisnak tekintett, az eredeti parancsunknak megfelelő kifejezés, megtörténhet a tényleges adathozzáférés, a kért adatok kinyerése, vagy változtatások végrehajtása. Az eredmények pedig visszakerülnek a klienshez (eredményekről alapvetően a lekérdezéseknél beszélhetünk, de valójában mindig van, egy törlési műveletnél például visszaadott eredmény az az üzenet, hogy „48500 rekord törölve”.

4.10. RDBMS Utasítások és csoportosításuk

A továbbiakban áttekintjük az SQL nyelv legfontosabb elemeit. Nem célunk teljes, mindenre kiterjedő leírást tenni – messze meghaladná ezen tárgy kereteit. Ebben a tananyagban csupán a megértéshez és a legalapvetőbb

feladatok elvégzéséhez szükséges nyelvi elemek egyszerűbb alapváltozataival foglalkozunk. Az egyes parancsok általában sokkal részletesebb funkcionalitással is bírnak, mint ahogy itt tárgyaljuk – mi az egyszerűbb, alapvető eseteket és használati módokat tekintjük át. A teljes, és sokkal részletesebb parancsokért a használt adatbázis-kezelő dokumentációját kell tanulmányozni, és persze a használat során gyakorlatot, tapasztalatot szerezni.

A relációs adatbázis-kezelők szerteágazó parancsait a következőképpen szokás csoportosítani:

- DDL „adat definiálás”
  Objektum létrehozás (CREATE)
  Objektum módosítás (ALTER)
  Objektum törlés (DROP)
- DML „adat módosítás”
  Rekord felvitel (INSERT)
  Rekord módosítás (UPDATE)
  Rekord törlés (DELETE)
- DQL „adat lekérdezés”
  Adatlekérdezés (SELECT)
- DCL „adat felügyelet”
  Hozzáférés-szabályozás (GRANT, REVOKE)
  Tranzakció-kezelés (COMMIT)

A zárójelbe írt szavak az SQL parancsok kulcsszavai. Angolul értőknek nagyon könnyű lesz memorizálni őket, mivel mint említettük, az SQL nyelv parancsai angolul értelmes mondatok.

4.10.1. DDL

A DDL (Data Definition Language – adatdefiníciós nyelv) csoportba tartoznak a táblák és különböző egyéb komponensek létrehozását, szükség szerinti módosítását és törlését végző parancsok.

Ezen parancsok dolgoznak a háttérben akkor is, amikor a 4.18. ábrán láthatóhoz hasonló felületeken hozzuk létre vagy módosítjuk egy tábla szerkezetét és paramétereit.

Létrehozás – CREATE

A létrehozás kulcsszava a CREATE, a parancs általános formája:

CREATE objektumtípus paraméterek

Az egyik legegyszerűbb eset, a tábla létrehozása:

CREATE TABLE táblanév (mező1 típus1 megszorítás1, . . .);

A táblanév és a mezők nevei az angol ábécé betűiből, aláhúzásjelből (_) és számokból állhatnak. A mezőnevet követi az adattípus megadása (lásd 4.4.7. fejezet). Ha több mezőből áll a tábla, a szükséges adatokat vesszővel (,) elválasztva egymás után soroljuk fel.

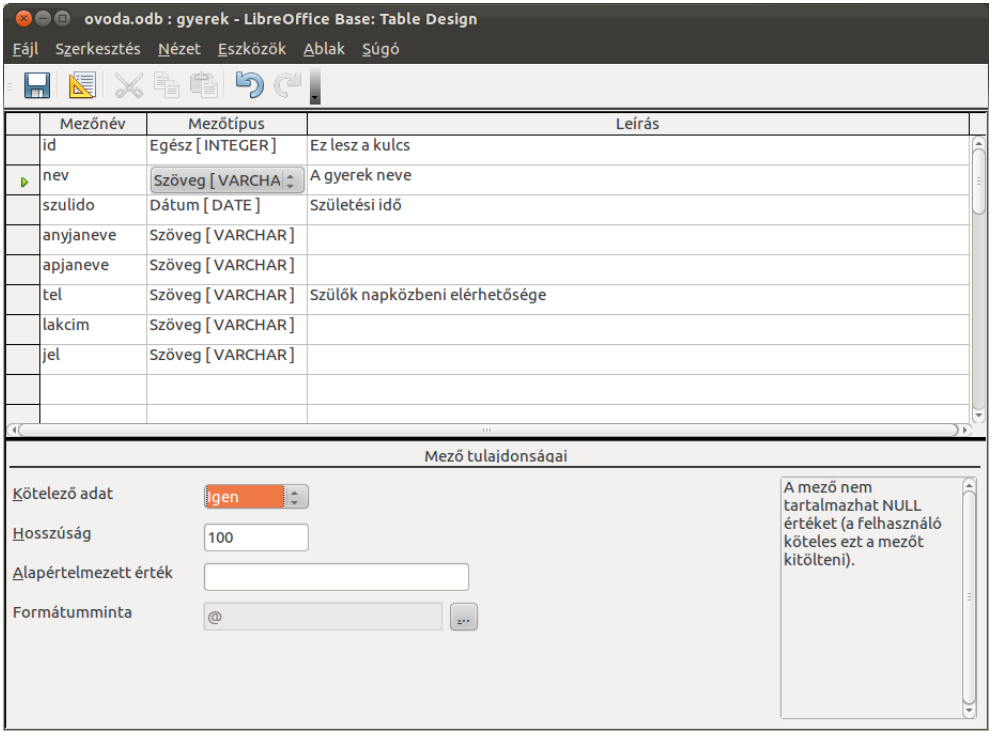
A megszorításokról, más szóval integritási feltételekről olvashattunk a 4.3. fejezetben, a konkrét megvalósítást pedig a 4.10.1 fejezetben tárgyaljuk.

Példa egy tábla létrehozására:

CREATE TABLE dolgozo (id INT PRIMARY KEY, nev VARCHAR(30), munkakor CHAR(1));

A relációs adatbázis-kezelők alapötlete, hogy nemcsak a konkrét tárolandó adatokat, hanem minden egyéb szükséges információt is olyan módon tároljon az adatbázis „saját” magában, hogy ezek kezelése hasonlítson a normál táblaadatokéra. Így indexet is a CREATE paranccsal készíthetünk:

CREATE INDEX munkakor ON dolgozo (munkakor NOT NULL);



4.18. ábra. *Tábla definíció grafikus felületen*

Ez a parancs egy *munkakor* nevű indexet hozott létre, ami a *dolgozo* táblához kapcsolódóan a *munkakor* mezőt indexeli. Az indexnév a későbbi hivatkozáshoz (például az index törléséhez) kell szokásos módon ugyanazt a nevet adtuk neki, mint az indexelt mezőnek. Figyeljük meg, hogy az SQL a nevek egyediségét csak azonos típusú objektumok esetén követeli meg. Az esetleges megszorításokat az indexelendő mező után írjuk. Ha több mezőt akarnánk bevonni az indexelésbe, akkor ezeket vesszővel (,) elválasztva egymás után sorolhatjuk fel.

A következő parancs egy adatbázis-felhasználót hoz létre, és azonnal jelszót is beállít hozzá:

CREATE USER név IDENTIFIED BY jelszó;

Módosítás – ALTER

A CREATE paranccsal elkészített objektumok szükség esetén az ALTER paranccsal módosíthatók:

ALTER objektumtípus paraméterek

Például módosítsuk a *dolgozo* tábla *nev* mezőjének típusát:

ALTER TABLE dolgozo ALTER/MODIFY nev VARCHAR(40) NOT NULL;

Módosítás az is, ha például egy új mezőt veszünk be a táblastruktúrába:

ALTER TABLE dolgozo ADD lakcim VARCHAR(40);

Törlés – DROP

A korábban létrehozott, de szükségtelenné vált objektumok a DROP (eldobás) paranccsal törölhetők:

DROP objektumtípus paraméterek

Például, töröljük ki a *munkakor* indexet, majd utána a *dolgozo* táblát (Az index törlését csak a példa kedvéért írtuk le: a *dolgozo* tábla törlése kapcsán automatikusan törlődött volna minden hozzá kapcsolódó adat, így például a tábla indexei is.)

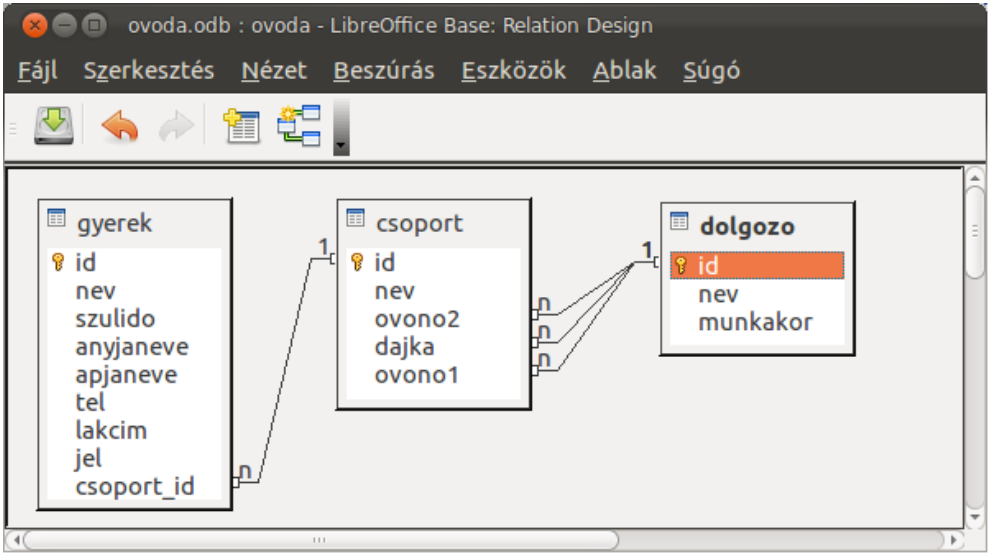
DROP INDEX munkakor;

DROP TABLE dolgozo;

Integritási feltételek megadása

Az integritási szabályok értelméről, fajtáiról a 4.4.9. fejezetben olvashattunk. A konkrét SQL megvalósítások a következők:

- NULL
  Az adat megadása nem kötelező (alapértelmezett, ezért nem szokás kiírni).
- NOT NULL
  Az adat megadása kötelező, mert tudjuk, hogy minden adatrekordhoz kötelezően tartozik ilyen adat, és erre számítunk is az adatbázisban. (Például, egy könyvtári nyilvántartásban előírjuk, hogy minden könyvnek kötelezően van címe és oldalszáma)
- PRIMARY KEY
  Az adott mező a tábla elsődleges kulcsa. Az elsődleges kulcsnak jelölt mezőt az adatbázis-kezelő automatikusan indexeli, PRIMARY KEY kiegészítéssel.
- UNIQUE
  Csak egyedi értékek vihetők fel az így jelölt mezőbe (pl. adószám, rendszám).
  Vegyük észre, hogy a PRIMARY KEY kulcsszavakkal jelölt mező nagyjából azonosan működik, mintha arra egy UNIQUE indexet hoztunk volna létre – egy fontos különbség azonban van. A PRIMARY KEY esetén egynél több mező nem maradhat üresen (NOT NULL), de a UNIQUE indexelés ezt megengedi.
- CHECK (feltétel)
  Csak a feltételnek megfelelő adatok kerülhetnek az oszlopba, például Győr város díszpolgára csak 50 év feletti helyi lakos lehet:
  CREATE TABLE díszpolgár ... CHECK (lakcim_varos=”Győr” AND kor>50)
- FOREIGN KEY (mező) REFERENCES tábla(mező) [ON DELETE ...] [ON UPDATE ...]
  Idegen kulcs jelölése. Idegen kulcsokkal már korábban foglalkoztunk (a 4.4.2. fejezetben), most csak az SQL leírási módot mutatjuk meg.
  A grafikus felület egyik előnye, hogy a táblák közötti kapcsolatok jól áttekinthetően ábrázolhatók (4.19.

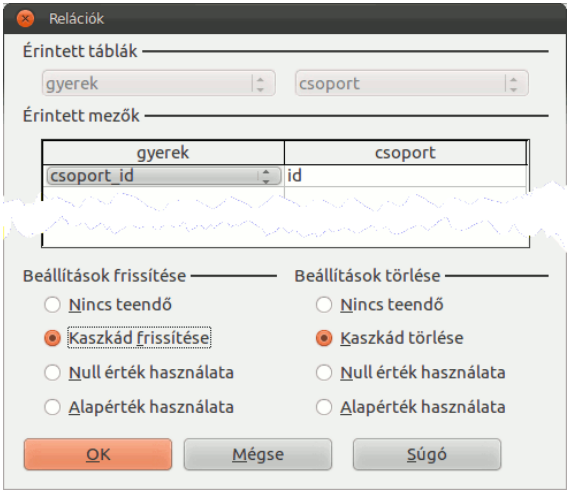


4.19. ábra. *Táblák közötti kapcsolatok megjelenítése*

ábra). A 4.20. ábrán pedig beállítható, mi történjen a kapcsolókulcs változása vagy törlése esetén. Az ábrákon látható *gyerek* és *csoport* táblák között kapcsolat megvalósítása az SQL nyelvben a következő:

CREATE TABLE gyerek (*meződefiníciós lista* ... FOREIGN KEY (csoport_id) REFERENCES csoport(id) ON DELETE CASCADE ON UPDATE CASCADE);

- Frissítés esetén (ON UPDATE):
  - CASCADE: A külső kulcsok is módosulnak (a 4.20. ábrán ez van beállítva).
  - NULL: A külső kulcs értéke NULL értéket vesz fel
  - SET DEFAULT: A külső kulcs értéke az alapértelmezettre áll



4.20. ábra. *Kapcsolati beállítások*

– Törlés esetén (ON DELETE):
  * CASCADE: A kapcsolódó rekordok törlődnek! (A 4.20. ábrán ez van beállítva: ha kitöröljük a csoportot, a csoporthoz tartozó összes gyerek adata is automatikusan törlődik. Sokszor hasznos, de veszélyes opció!)
  * NULL: A külső kulcs értéke NULL értéket vesz fel
  * SET DEFAULT: A külső kulcs értéke az alapértelmezettre áll

A 4.20. ábrán látható két „Nincs teendő” állapotot SQL-ben legegyszerűbben úgy érjük el, hogy nem írjuk ki az ON UPDATE és ON DELETE módosítókat.

4.10.2. DML

A DML (Data Manipulation Language – adatmódosító nyelv) csoportba tartoznak azok a parancsok, amik a táblákban tárolt adatok bevitelét, módosítását és törlését végzik.

Beszúrás – INSERT

A legegyszerűbb eset, amikor a felsorolt konstans értékeket akarjuk egy táblába beszúrni:

INSERT INTO "dolgozo" VALUES (NULL,'Sári néni','o', 56)

Módosítás – UPDATE

Aktivitás: Fogalmazzuk meg, pontosan mit csinál az alábbi parancs:
UPDATE "dolgozo" SET "nev"='Sári néne';

Ha az előző kérdésre válaszoltunk, akkor válaszolhatunk erre is:

Aktivitás: Mit csinál a következő SQL parancs?
UPDATE "dolgozo" SET "nev"='Sári néne' WHERE "id"=32;

A két kérdés közül az elsőre az a helyes válasz, hogy az összes rekord név mezője a megadottra módosul, a második esetben csak az *id*-nek megfelelő rekord. Mindez teljesen logikus, de figyeljünk rá: ha nem adunk meg (szelekciós, szűkítő) feltételt, a parancs minden elérhető rekordra vonatkozik. Így van ez a lekérdezéseknél, de sokkal veszélyesebb, hogy ugyanez történik módosításnál és törlésnél is.

Törlés – DELETE

Egy példa a DELETE parancs használatára:

DELETE FROM dolgozo;

Az előző példák alapján már tudnunk kell, hogy ez a parancs a *dolgozo* tábla összes rekordját kitörölte. Általában nem ezt akarjuk, hanem valamely szelekciós feltétel alapján törlünk:

DELETE FROM dolgozo WHERE dolgozo_id=8392;

Törölni mindig csak rekord egységenként lehet.

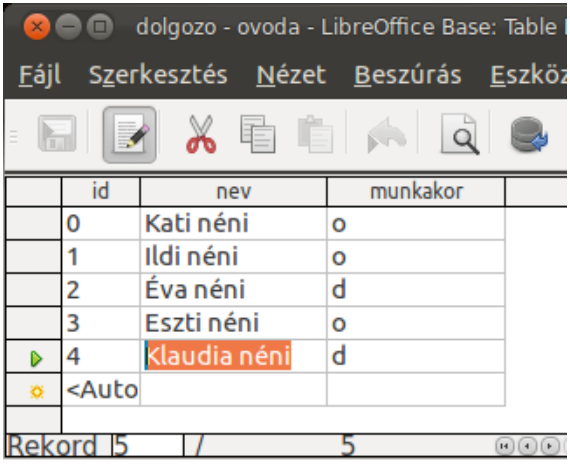
Aktivitás: Ha csak egy egész rekord törölhető, mit tehetünk, ha egy rekordunk egyik mezőjének tartalmára már nincs szükségünk? És ha úgy döntünk, hogy az adott mezőre a későbbiekben egyáltalán nincs szükség?

Ha valamilyen grafikus felületen érjük el az adatbázist, amint az például a 4.21. ábrán látható, akkor is az itt megismert DML parancsok dolgoznak a háttérben.

Talán a DML parancsok kapcsán érdemes megjegyezni, de természetesen minden SQL parancsra igaz, hogy a parancsok eredménye azonnal, visszavonhatatlanul tárolódik az adatbázisban: nincs visszalépés (undo) lehetőség, és nem kérdezi meg a későbbiekben, hogy akarjuk-e a változtatásokat menteni. Már megtette. Ezen szabály alól a desktop adatbázis-kezelők és egyéb grafikus alkalmazások kiskaput jelenthetnek, ahol a gyakorlatlanabb felhasználók kedvéért sok esetben fenntartanak egy ilyen lehetőséget, de még ezeken a rendszereken sem lehet bármilyen műveletet visszavonni.

4.10.3. DCL

A DCL a Data Control Language (adatvezérlő nyelv) rövidítése. A magyar fordítás nem túl szerencsés, hiszen nem a klasszikus értelemben vett vezérlő utasítások tartoznak ide! A programozási nyelvekben vezérlő utasítások közé a ciklusszervező utasítások, feltételes elágazások (for és while ciklus, if és switch szerkezetek) tartoznak: az SQL *nem* programozási nyelv és semmi hasonlót nem is tartalmaz! Említettük már, de e gondolat



4.21. ábra. *Adatkezelés (bevitel, módosítás, törlés) grafikus felhasználói felületen*

kapcsán hasznos megismételni, hogy míg a hagyományos, procedurális programnyelvekben a programozó a végrehajtandó lépéseket fogalmazza meg, addig az SQL nyelvben csak az elvárt működésre, eredményre utalunk, nem írjuk le a feladat végrehajtásának algoritmusát. (Létezhetnek, léteznek procedurális kiegészítők az SQL nyelvhez, például az Oracle PL/SQL programnyelve, de ezek már túlmutatnak az általunk most tanulmányozott alap SQL nyelven és ezen a tananyagon.)

Ezek alapján a DCL magyar megfelelőjének talán az adatfelügyelő nyelv megnevezés lenne a szerencsés, a mindenki által használt adatvezérlővel szemben.

A DCL csoportba az adatbázis-kezelő rendszer felhasználóival, a hozzájuk és az adatbázis-objektumokhoz kapcsolódó jogosultságokkal foglalkozó parancsok tartoznak.

Nézzünk néhány példát az ide tartozó parancsokra. Először is, (egy már ismert DDL parancs segítségével) hozzunk létre egy felhasználót:

CREATE USER diak1

Bár megtehettük volna, de nem adtunk neki jelszót. Tegyük meg utólag:

ALTER USER diak1 SET PASSWORD 'Frédi'

Hozzunk létre egy másik felhasználót is az adatbázisban:

CREATE USER diak2 PASSWORD 'Béni'

A felvett felhasználóink számára létrehozunk egy *hallgato* nevű jogosultságot, és ezt a – egyelőre nem részletezett – jogot a két diákunknak adományozzuk:

CREATE ROLE hallgato

GRANT hallgato TO diak1

GRANT hallgato TO diak2

E két utóbbi műveletet akár úgy is megfogalmazhatnánk, hogy a két megadott felhasználót hozzáadtuk a *hallgato* csoporthoz. Már csak azt kell beállítanunk, hogy a *hallgato* csoportnak mihez legyen joga az adatbázisunkban:

GRANT SELECT, UPDATE ON TABLE táblanév TO hallgato

A *hallgato* csoport, tehát a mindenkor hozzátartozó felhasználók jogot kaptak a *táblanév* nevű tábla lekérdezésére és adatainak módosítására, de ez alapján nincsen joguk a táblába új rekordot felvinni, vagy onnan törölni.

Ennél sokkal finomabban is beállíthatjuk az egyes felhasználók jogosultságait. Beállíthatjuk többek között, hogy honnan jelentkezhetnek be a hálózaton keresztül az adatbázisba, hogy például egy óra alatt hány lekérdezést vagy más műveletet végezhetnek (hogy ne terheljék túl a rendszert, vagy hogy elejét vegyük az adataink jogosulatlan böngészésének), előírhatjuk, hogy egy táblának csak bizonyos oszlopait kérdezhetik le, vagy csak azokat módosíthatják, szabályozhatjuk, hogy a kapott jogokat tovább adhatják-e más felhasználóknak (GRANT OPTION), stb.

Az átadott jogokat a REVOKE paranccsal vehetjük vissza. Egy felhasználótól a következő paranccsal minden jogát elvesszük:

REVOKE ALL PRIVILEGES, GRANT OPTION FROM diak1

Az ALL PRIVILEGES az összes jogot egyben jelenti, így biztosan nem felejtettünk ki semmit felsorolni. Így a hallgatónk semmit nem tud többé kezdeni az adatbázisban, de maga a felhasználó még létezik. Ha nincs rá szükség, csak egy mozdulat végleg kidobni:

DROP user diak1

4.10.4. DQL

A DQL a Data Query Language (adatlekérdező nyelv) rövidítése. Gyakorlatilag egyetlen parancs, a SELECT tartozik ide, aminek célja az adatbázisban tárolt adatok kinyerése.

A SELECT parancs kiemelt szerepet kap a többi parancs között: amellett, hogy a legtöbb alkalmazásban ezt használják a legtöbbet, messze a legösszetettebb is. Ezért külön, a 4.11. fejezetben foglalkozunk részletesen vele.

Grafikus felületeken is összeállíthatók egyszerűbb lekérdezések (4.22. ábra), az ezeken elérhető lehetőségek azonban csak töredékei a mögöttes parancsok teljes funkcionalitásának.

A lekérdezések más parancsokba ágyazhatók (részletesen lásd később a 4.12. fejezetben):

- Táblakészítés
- Hozzáfűzés
- Frissítés
- Törlés

A SELECT legalapvetőbb formája a következő:

SELECT mit FROM honnan WHERE feltételek

Az ennél bonyolultabb eseteket a 4.11. fejezetben tárgyaljuk.

Mező	nev	jel	szulido
Alias	Gyermek neve	Gyermek jele	
Tábla	gyerek	gyerek	gyerek
Rendezés	növekvő		
Látható	☑	☑	☐
Függvény			
Feltétel			> #2005-12-31#

4.22. ábra. *Adatlekérdezés grafikus felhasználói felületen*

Önellenőrzés

1. Milyen feladatokra alkalmas egy desktop adatbáziskezelő?
2. Milyen lehetséges módjai vannak az adatbázis-szerverrel történő kommunikációnak? Mindegyikre mondjon 1-1 tipikus felhasználási területet!
3. Mi a különbség a DROP és DELETE utasítások között? Melyik mit töröl?
4. A 4.22. ábra alapján mit mondhatunk a harmadik oszlop rendezettségéről? Milyen sorrendben fognak megjelenni az adatok?

20. LECKE

Lekérdezések

4.11. A SELECT parancs

A SELECT utasítás az egyetlen parancs, amivel adatokat nyerhetünk ki az adatbázisból. Ez már önmagában sejteti, hogy egy szerteágazóan használható paranccsal van dolgunk: a SELECT a legösszetettebb az SQL parancsok között. Ebben a leírásban megmutatjuk a használatát a legfontosabb esetekben, de több részletre egyáltalán nem térünk ki, ezért ez a bemutatás nem teljes. Elegendő azonban ahhoz, hogy jól átlássuk a működési logikáját és dokumentáció alapján tetszőlegesen bonyolult lekérdezéseket is összeállíthassunk.

Az ismerkedés elején érdemes azonban azt is megérteni, hogy konkrét adatbázis-szerkezet és feladat esetén a modellből adódóan kitalálhatók olyan lekérdezések, amit nem tudunk egyetlen SELECT paranccsal megvalósítani – tehát nem biztos, hogy a mi hibánk, ha egy bonyolult esetre nem sikerül az elvárt végeredményt biztosító egyetlen parancsot megalkotni – ilyen esetekben kerülőutakon, több lekérdezés egymás után történő végrehajtásával kaphatjuk meg a várt eredményt. Az ilyen helyzetek azonban ritkák, mint látni fogjuk, egészen bonyolult kérdésekre is választ kaphatunk egyetlen jól összerakott SELECT paranccsal.

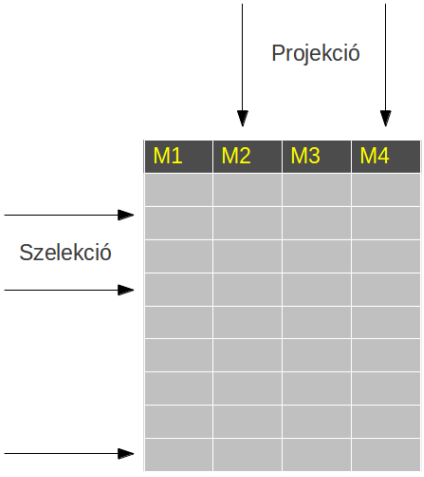
Tekintsük a 4.23. ábra táblázatát, ahol az M2 és M4 mezőkre (oszlopokra) van szükségünk, de csak a szelekciós feltétel által meghatározott rekordok (sorok) esetében. Ezt a lekérdezést a következőképpen fogalmazhatjuk meg:

SELECT M2, M4 FROM táblanév WHERE szelekciós feltétel

4.13. definíció: A projekció az a művelet, amivel egy reláció (tábla) egyes mezőit (oszlopait) válogatjuk ki (szűkítjük le) egy lekérdezésben.

4.14. definíció: A szelekció az a művelet, amivel egy reláció (tábla) egyes rekordjait (sorait) válogatjuk ki (szűkítjük le) egy lekérdezésben.

A szelekció meghatározása a WHERE kulcsszó után történik. A feltételek megadási módjával később, a 4.11.1. fejezetben foglalkozunk.



4.23. ábra. *A SELECT művelet két legfontosabb eleme*

A projekció célja, hogy az eredményben csak a számunkra fontos adatok (pontosabban mezők) szerepeljenek. A projekció meghatározása úgy történik, hogy a SELECT és FROM kulcsszavak között tételesen felsoroljuk (vesszővel elválasztva, nekünk tetsző sorrendben) a szükséges mezőket. Ha minden mezőre szükségünk van, akkor a teljes lista begépelését megspórolhatjuk egyetlen * jellel, ebben az esetben a mezők megjelenési sorrendjére nincs befolyásunk, tábladefiníciókor használt sorrendben fognak megjelenni. Ha több táblát használunk, akkor természetesen a táblák mezői pedig eszerint követik egymást.

Azt a táblát, amire a keresésünk vonatkozik, a FROM után írjuk. Ha nem egy tábla, hanem több tábla Descartes-szorzata (lásd 4.4.1. fejezet) alapján keresünk, akkor itt lehet a táblákat felsorolni, vesszővel elválasztva. Ebben az esetben szinte biztos, hogy nem az összes párosításra van szükségünk, hanem a WHERE után egy olyan feltételt is megfogalmazunk, ami előírja a több tábla összekapcsolásának módját. Nézzünk erre egy példát:

SELECT emberek.*, szemelygepjarmuvek.rendszam FROM szemelygepjarmuvek, emberek WHERE

szemelygepjarmuvek.tulajdonos_id=emberek.id

Érdekesség: Ez a lekérdezés nagyobb táblaméreteknél erősen leterhelheti az adatbázis-kezelő rendszert, hiszen Descartes-szorzat előállítását kértük, ami nagy tábláknál igen nagy méretű lehet! Jobb megoldás ezt a helyzetet mindenképpen elkerülni és más módon a JOIN művelettel kérni az összetartozó párok előállítását:
SELECT emberek.*, szemelygepjarmuvek.rendszam FROM szemelygepjarmuvek
JOIN emberek ON szemelygepjarmuvek.tulajdonos_id=emberek.id
Ez a lekérdezés jellemzően sokkal gyorsabban lefut, ugyanazt a végeredményt produkálva.

A * használatára is láthatunk példát, az *emberek* táblából minden mezőt, a *szemelygepjarmuvek* táblából csak a rendszámot íratjuk ki. A két tábla közötti kapcsolatot az adja, hogy járművek táblájában van egy *tulajdonos_id* mező, ahova beírjuk egy személy kulcsmezőjének értékét.

A példa kapcsán észrevehető, hogy ugyanazt a táblanevet többször is le kell írni, ami zavaró lehet, különösen, ha a táblanév hosszú. Ezért van lehetőség – egyetlen lekérdezés erejéig – a tábláinknak egy rövid nevet adni, használjuk most az *a* (autó) és *e* (ember) betűket táblanévként, így sokkal olvashatóbb a parancs:

SELECT e.*, a.rendszam FROM szemelygepjarmuvek a, emberek e WHERE a.tulajdonos_id=e.id

Több tábla összekötésének ennél finomabban szabályozható módja is van (az érdeklődők a JOIN műveletet keressék), de az nem képezi a tananyag részét.

Sok implementáció-függő speciális eset is van, amivel most szintén nem foglalkozunk. A következő számolás például működik, és a várt eredményt adja:

SELECT 5*5;

4.11.1. Szelekciós feltételek

A szelekciós feltételeket egyetlen logikai kifejezésként a WHERE kulcsszó után adjuk meg. Több feltétel esetén a feltételeket AND, OR, NOT logikai kifejezéssel (ha kell, akár zárójelezéssel) kapcsoljuk össze. Például

SELECT * FROM m WHERE m1<5 AND (m2=3 OR m2>8)

A szelekciós feltételekben használható nyelvi elemek

- Összehasonlító operátorok: = < > <= >= <> !=
- Halmazoperátor: IN ('a','b','c')
  SELECT * FROM gyumolcs WHERE nev IN ('Alma','Körte')
  Az IN utáni zárójelbe akár egy másik SELECT utasítás is tehető, aminek természetesen egyetlen mezőt szabad tartalmaznia a projekcióban (SELECT nev FROM ...). Ezekben az esetekben a hatékonysággal probléma lehet, úgyhogy csak megfelelő körültekintéssel ágyazzunk egymásba parancsokat. Lásd még: 4.12. fejezet.
- Tartományba esés ellenőrzése: BETWEEN a AND b
- Mintaillesztés: LIKE (az angol szó egyik jelentése: „olyan, mint")
  A használatához a LIKE kulcsszó után idézőjelben egy mintát kell megadnunk, ami a tényleges keresett szöveg mellett a következőket tartalmazhatja:
  - _ egyetlen, bármilyen karakter
  - % akárhány karakter

  Például:

LIKE 'a_'	a betűvel kezdődő kétbetűs
LIKE 'a%'	a betűvel kezdődő
LIKE '%a%'	a betűt tartalmazó
NOT LIKE ...	a mintával nem egyező
CLIKE ...	a mintaillesztésnél nem különbözteti meg a kis- és nagybetűket (A LIKE megkülönbözteti!)

- Kitöltetlenség ellenőrzése: IS NULL
  Nézzük meg, mely gyerekek esetén hiányos a nyilvántartásunk:
  SELECT * FROM gyerek WHERE anyja_neve IS NULL

Érdekesség: A 4.4.8. fejezetben a NULL érték kapcsán láttuk, hogy egyszerű összehasonlítással (pl. IF mezo=NULL) nem lehet eldönteni egy adatról, hogy NULL értékű-e, hiszen annak a műveletnek az eredménye is mindig NULL érték lenne. E probléma elkerülésére való az SQL speciális, függvényhez hasonló IS NULL megoldása.

A SELECT utasításból eddig tehát a bemeneti táblák megadását, a szelekciót és projekciót ismerjük, de van még jópár további lehetőség is. Nézzük a SELECT utasítás teljes alakját (lásd a 4.24. ábrát is)

SELECT [DISTINCT] oszloplista	Projekció
FROM táblanévlista [alias]	Descartes-szorzat
[WHERE] feltételek	Szelekció
[GROUP BY oszloplista]	Csoportosítás
[HAVING feltételek]	Csoport szelekció
[ORDER BY oszloplista]	Rendezés

4.11.2. Csoportosítás

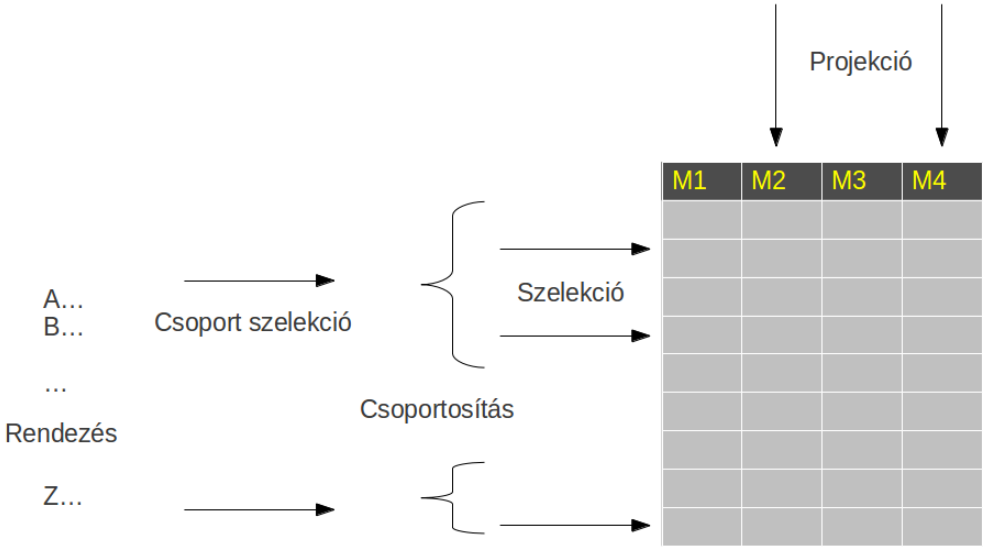
Sokszor szükségünk van arra, hogy az eddig tárgyalt projekció és szelekciós feltétel után egy újabb válogatást végezzünk valamely oszlopértékek azonossága alapján. Például, egy iskolai tanuló-nyilvántartásból kilistázzuk a gyerekek nevét, tanulmányi átlagát és évfolyamát:

SELECT nev, atlag, evfolyam FROM tanulo

Nem az egyes gyerekek tanulmányi átlagára, hanem az évfolyamok átlagára van szükségünk, ezért a fenti lekérdezéssel kapott rekordokat csoportosítjuk az évfolyam azonossága alapján:

SELECT . . . FROM tanulo GROUP BY evfolyam

A projekciós részt most azért nem írtuk, mert módosítanunk kell rajta. Az egyes gyerekek nevére eleve nincs szükségünk, a tanulmányi átlagukat pedig évfolyamonként (csoportonként) átlagolnunk kell. Hasznos lehet még látni, hogy az egyes évfolyamokon hány gyerek van. Két függvénnyel kell megismerkednünk: az AVG



4.24. ábra. *A kibővített SELECT művelet*

(average – átlag) és a COUNT (számol) aggregációs függvényekkel, amikre további példákat majd a 4.11.7. fejezetben láthatunk. A lekérdezésünk tehát így alakul:

SELECT evfolyam, COUNT(*) AS Letszam, AVG(atlag) AS Evfolyamatlag FROM tanulo GROUP BY evfolyam

A szelekcióval kiválasztott sorok csoportosításának szempontját (egy, vesszővel (,) elválasztva több mezőt) a GROUP BY kulcsszó után írjuk. Ha az így előállt listát szeretnénk tovább szűkíteni (csoportszelekció), például csak 4-es átlagot elérő évfolyamokra vagyunk kíváncsiak, azt a HAVING kulcsszó után tehetjük meg. Ez előző lekérdezést folytatva:

... HAVING AVG(atlag) >= 4

4.11.3. Végrehajtási sorrend

A SELECT parancsok esetében komoly jelentősége van annak, hogy az egyes parancs-részeket milyen sorrendben értelmezzük mi, illetve a parancsértelmező. Ez az ún. végrehajtási sorrend logikus, de nem magától értetődő, hiszen eltér a szokásos balról-jobbra olvasási rendtől. Nézzük tehát az egyes részlépések precedenciáját:

1. FROM ...
   Descartes-szorzat, itt állítjuk elő a „bemeneti” táblákból a Descartes-szorzatnak megfelelő kimeneti tábla kezdeti formáját
   JOIN ...
   Táblák összekapcsolása
2. WHERE ...
   Szelekció, tehát a reláció sorai közül válogatunk.
3. GROUP BY ...
   Csoportosítás, amivel a reláció egyes sorait összevonhatjuk
4. HAVING ...
   Csoport szelekció, tehát a csoportosítás végrehajtása után újra válogatunk a reláció sorai között.
5. SELECT ...
   Projekció, tehát a reláció oszlopai közül válogatunk.
6. ORDER BY ...
   Rendezés, itt már csak a rekordok átadásának sorrendjét állítjuk be

Természetes, hogy egyes részlépések kimaradhatnak, hiszen nem kötelező mindet, egy időben használni.

4.11.4. Halmazműveletek

A relációs adatmodell megismerésekor a 4.4.1. fejezetben említettük már, hogy a modell a tárolt adatokat halmazként definiálja, és halmazként is kezeli.

4.25. ábra. *Halmazműveletek*

Van néhány olyan művelet, ahol ez a halmazos megközelítés nem csak a modell működésének megértéséhez fontos, hanem mi magunk akarunk konkrét halmazműveleteket végezni: ilyenek az unió, metszet, különbség képzések (4.25. ábra). Ezek a halmazműveletek természetesen csak megfelelő (kompatibilis) táblaszerkezet esetén működnek.

- Unió
  SELECT * FROM T1 UNION SELECT * FROM T2
  Az óvoda teljes névsora, beleértve a gyerekeket és az ott dolgozókat is:
  SELECT nev FROM gyerek UNION SELECT nev FROM dolgozo
- Metszet
  SELECT * FROM T1 INTERSECT SELECT * FROM T2
- Különbség
  SELECT * FROM T1 EXCEPT SELECT * FROM T2

Fontos látni, hogy a halmazműveleteknél nincsen semmiféle sorrendiség a halmaz elemei között. Az SQL nyelv használata során észrevehetünk azonban két esetet, amikor a halmazelemek sorrendjével is kell törődnünk. Az egyik a projekció, ahol természetesen csak valamely sorrendben tudjuk felsorolni a szükséges mezőket, a másik pedig a lekérdezés eredményének rekordonkénti sorba állítása, erre való az ORDER BY módosító. Ha nem adnánk meg, akkor is kialakulna valamilyen sorrend, természetesen, ahogyan az adatbázis-kezelő sorba veszi a megtalált adatokat. Ez a két eset annyira természetes, hogy nem zavar minket, hogy valójában listát képeztünk a halmazokból.

4.11.5. Rendezés

Ahogy az imént említettük, a lekérdezések eredménye elméletileg egy halmaz, amit a gyakorlatban mindig valamely listába rendezve kapunk meg az adatbázis-kezelőtől. Sok esetben meg szeretnénk határozni, hogy milyen sorrendben kérjük a kinyert adatokat. Az ORDER BY kulcsszó használatával egy vagy akár több oszlopot is felsorolhatunk (természetesen csak a projekcióban szereplő mezők közül):

ORDER BY oszlopnév [DESC], . . . , oszlopnév [DESC]

A rendezés alapértelmezésben növekvő, ha fordítva szeretnénk, akkor a DESC (descending – csökkenő) szót kell a mezőnév után írni. Ezt akár mezőnként is külön-külön meghatározhatjuk. A normál, növekvő rendezés kulcsszava az ASC (ascending – növekvő), ezt azonban, felesleges kiírni, mert ez az alapértelmezett.

Az ORDER BY mindig a legutolsó művelet, mivel a relációs modell halmazokkal dolgozik, a rendezésnek korábban semmilyen értelme nem lenne, ezért nem is megengedett.

4.11.6. Függvények

Az SQL parancsokban sok-sok, a programozási nyelvekből, vagy matematikából ismerős függvény használható. Nagyon sok beépített függvény létezik, a pontos függvénylista implementációfüggő, ezért a részletekért mindig az adott adatbázis saját dokumentációját érdemes megnézni.

Az attribútumok általában mezőnevek, de a függvények egymásba ágyazhatók, így attribútumok helyén más függvények is lehetnek.

A függvények a paraméterben szereplő értékre, vagy akár a táblára vagy a teljes adatbázisra vonatkoznak, értelemszerűen. A teljesség igénye nélkül, nézzünk néhány példát, ami alapján sejtésünk lehet a használható függvények köréről:

- LENGTH() hosszúság
- ROUND() kerekítés
- LOWER() kisbetűsre konvertál

- UPPER() nagybetűsre konvertál
- ROWNUM() sorok száma
- TO_CHAR() számot vagy dátumot karakterlánccá konvertál
- HOUR() óraérték 0-23
- WEEK() a hét száma 1-54
- SIN() szinusz
- PI() pi
- REPLACE() karakterlánc-csere
- ISNULL() null-érték
- DATABASE_VERSION() adatbázis-verziószám lekérdezése
- CURRENT_USER() az aktuális felhasználó azonosítója

Saját függvények is definiálhatók (CREATE FUNCTION ...), ez minden implementációban teljesen különböző, és ebben a tananyagban nem foglalkozunk vele.

4.11.7. Aggregációs függvények

A függvények között külön helyet foglalnak el az ún. *aggregációs* függvények, amelyek nem egyetlen rekord mezőértékén, hanem az összes kiválasztott rekordon hajtódnak végre. E csoport legjellemzőbb képviselői a következők:

- MIN() legkisebb érték
- MAX() legnagyobb érték
- SUM() összeg

- COUNT() elem darabszám
- AVG() átlag

Saját aggregációs függvények is definiálhatók (CREATE AGGREGATE FUNCTION), ennek implementáció-függő részleteire nem térünk ki.

Lássunk néhány példát. Legidősebb gyerek születésnapja:

SELECT MIN(szulido) FROM GYEREK

A leghosszabb név (ez hasznos lehet akkor, amikor egy pl. egy fix hosszúságú név mezőt tervezünk, vagy képernyőképek tervezésénél):

SELECT MAX(LENGTH(nev)) FROM gyerek

A következőben pedig példát láthatunk arra, hogyan tudunk egy aggregációs függvényérték mellé valamely konkrét mezőértéket tenni. A leghosszabb nevű gyerek:

SELECT nev FROM gyerek gy1

WHERE LENGTH(gy1.nev) = (SELECT MAX(LENGTH(gy2.nev)) FROM gyerek gy2);

4.12. Beágyazott SELECT parancs

A legutóbbi példában egy korábban nem látott trükkel éltünk: egy SELECT parancsba alkalmas helyen beágyaztunk egy másik SELECT parancsot. Ezt az egymásba ágyazást az SQL nyelv megengedi, így nagyon kibővülnek a lehetőségeink.

A SELECT parancs ráadásul nem csak SELECT parancsba, hanem más SQL parancsokba is ágyazható. Erre leginkább olyan esetekben van szükség, amikor a lekérdezés eredményét nem szükséges tárolni, hanem csak tovább akarunk dolgozni a kapott (rész)eredményhalmazzal.

Figyeljük meg, hogy a 4.11.4. fejezetben tárgyalt halmazműveletek is valójában két SELECT parancs egybeágyazását jelentik.

A következő tipikus példák kapcsán nézzük meg, a valódi lekérdezéseken kívül mi mindenre lehet használható a beágyazott SELECT parancs:

- Táblakészítés
  CREATE TABLE maci AS (SELECT * FROM gyerek WHERE csoport=0);
  Bármilyen táblastruktúra létrehozható, ilyenkor természetesen a beágyazott SELECT parancs projekciós részének megfelelő alakúnak kell lennie.
- Hozzáfűzés
  INSERT INTO gyerek SELECT * FROM masikovi;
  Ezt a feladatot egyébként az INTO nyelvi elemmel beágyazás nélkül is meg tudjuk oldani:
  SELECT * FROM masikovi INTO gyerek;
  Természetesen ezek a parancsok azonos táblastruktúrát feltételeznek.
- Módosítás
  Aki máshova jár, ide nem járhat:
  UPDATE gyerek
  SET csoport = NULL
  WHERE EXISTS (SELECT * FROM masikovi WHERE gyerek.id = masikovi.id);
- Törlés
  Töröljük ki azon gyerekeket, akik másik óvodai nyilvántartásban is szerepelnek:
  DELETE FROM gyerek WHERE id IN (SELECT id FROM masikovi WHERE gyerek.id=masikovi.id);

Önellenőrzés

1. Soroljon fel néhány példát, az Ön adatai milyen adatbázisokban szerepelhetnek!
2. Mi a különbség az ALTER és UPDATE *módosító* SQL parancsok között?
3. Az általunk választott adatbázis-kezelőben hozzuk létre a korábbi feladatok kapcsán megtervezett adatbázis-táblákat és indexeket.
4. Fogalmazzunk meg egy SQL parancsot (parancsokat), ami egy könyv adatait beteszi a megfelelő táblába vagy táblákba.
5. Miért van az, hogy a SELECT utasításban a rendezésnek csak az összes művelet után van értelme?
6. Fogalmazzunk meg egy SQL parancsot, amivel lekérdezhető, van-e Andrew S. Tanenbaum által írt könyv az adatbázisban.
7. Hány oszlopból fog a hosszu tabla állni?

   ```
   CREATE TABLE hosszu AS SELECT * FROM vonat WHERE hossz>1024;
   ```

   - ☐ Annyiból, ahány oszlopa van a vonat táblának.
   - ☐ Annyiból, ahány sora van a vonat táblának.
   - ☐ 1024 vagy még több, az adatok függvényében.
   - ☐ A fenti parancs hibás, SELECT paranccsal nem lehet táblát létrehozni.
8. Mit csinál a következő parancs?

   ```
   DELETE FROM diak;
   ```

   - ☐ A diak tábla összes rekordját törli.
   - ☐ A diak táblát törli
   - ☐ A parancs hiányos, ezért nem fut le.

Megoldások

I. modul 3. lecke - 1. kérdés:

1.1. Megoldás egy szubrutinnal

A megoldást egy szubrutin végzi, amely az adatok bekérését és az eredmények kiírását is elvégzi.

Pl.

```
Sub Kor()
Dim r As Single, t As Single, k As Single
r = InputBox("Kérem a kör sugarát!")
t = r * r * 3.14
k = 2 * r * 3.14
MsgBox "A kör területe:" & Format(t, "0.00") & Chr(13) & _
    "A kör kerülete:" & Format(k, "0.00")
End Sub
```

1.2. Megoldás egy megoldó és egy tesztelő szubrutinnal

A megoldást egy (lehetőleg teljes körűen) paraméterezett szubrutin végzi, amely csak a feladat megoldását tartalmazza, a működéséhez szükséges adatokat a paraméterein keresztül kapja, az eredményeket pedig visszaadja. Az adatok bekérését és az eredmények kiírását egy (paraméterek nélküli) tesztelő szubrutinba tegyük, amely kérjen be adatokat (InputBox), hívja meg a megoldó szubrutint (a megadott adatokkal), majd írja ki (MsgBox, Debug.Print) az eredményül kapott adatokat!

Pl.

```
Function Haromszog(a As Single, b As Single, c As Single) As Boolean
Haromszog = a + b > c And b + c > a And a + c > b
```

```
End Function
Sub Haromszog_Teszt()
Dim a As Single, b As Single, c As Single
a = InputBox("Kérem az első számot!")
b = InputBox("Kérem a második számot!")
c = InputBox("Kérem a harmadik számot!")
If Haromszog(a, b, c) Then
    MsgBox "A három szám lehet egy háromszög három oldala!"
Else
    MsgBox "A három szám nem lehet egy háromszög három oldala!"
End If
End Sub
```

Vissza a kérdésekhez!

I. modul záró feladatok - 1. mintafeladat:

```
Sub Mintafeladat1()
'Az adatok száma
Const n = 5
'Az adatokat tároló tömb
Dim a(1 To n) As Integer
'Az eredményeket tároló változók
Dim Min As Integer, Max As Integer, Osszeg As Long, Atlag As Single
'Segédváltozó
Dim i As Integer
'Adatok generálása
Call Szamokat_General(a, n, 1, 10, 1)
'Adatok kiírása az Immediate ablakba (egy sorba)
Debug.Print "Az adatok:";
For i = 1 To n
        Debug.Print a(i);
Next
Debug.Print
'Az adatok feldolgozása
'Legkisebb, legnagyobb elem
'Megjegyezzük az első elemet
Min = a(1): Max = a(1)
```

```
'Megvizsgáljuk a többit, van-e kisebb, van-e nagyobb?
For i = 2 To n
    If a(i) < Min Then
        'Van egy kisebb, módosul az eredmény
        Min = a(i)
    End If
    If a(i) > Max Then
        'Van egy nagyobb, módosul az eredmény
        Max = a(i)
    End If
Next
'Kiírjuk az eredményt
Debug.Print "Legkisebb:"; Min
Debug.Print "Legnagyobb:"; Max
'Megoldás munkalapfüggvénnyel
Debug.Print "Legkisebb:"; WorksheetFunction.Min(a)
Debug.Print "Legkisebb:"; WorksheetFunction.Small(a, 1)
Debug.Print "Legnagyobb:"; WorksheetFunction.Max(a)
Debug.Print "Legnagyobb:"; WorksheetFunction.Large(a, 1)
'Összeg
'Kezdőérték
Osszeg = 0
```

```
'Minden elemet hozzáadunk
For i = 1 To n
        Osszeg = Osszeg + a(i)
Next
'Kiírjuk az eredményt
Debug.Print "Összeg:"; Osszeg
'átlag
Atlag = Osszeg / n
'Kiírás 1 tizedesjeggyel
Debug.Print "átlag:" + Format(Atlag, "0.0")
'Megoldás munkalapfüggvénnyel
Debug.Print "átlag:" + Format(WorksheetFunction.Average(a), "0.0")
MsgBox "Az adatok és az eredmények az Immediate ablakban láthatók!"
End Sub
```

Vissza a kérdésekhez!

I. modul záró feladatok - 2. mintafeladat:

```
Sub Mintafeladat2()
'A szöveg hossza
Const n = 10
'A feldolgozandó szöveg
Dim st As String
'Az eredményt tároló változó
Dim db As Integer
'Segédváltozók
Dim i As Integer, kar As String
'Adatgenerálás
st = Szoveget_General(n, 2)
'Adat kiírása az Immediate ablakba
Debug.Print "Az adat:"; st
'Az adat feldolgozása
'Darabszám (a szövegben található angol betűk darabszáma)
'Kezdőérték
db = 0
'A szöveg minden karakterét megvizsgáljuk
For i = 1 To Len(st)
        'A kar változóba a szöveg i. karakterét
        kar = Mid(st, i, 1)
```

```
        'Az adott karakter (kar) vizsgálata (angol betű-e?)
        If kar >= "A" And kar <= "Z" Or kar >= "a" And kar <= "z" Then
                'Angol betű, növeljük a darabszámot
                db = db + 1
        End If
Next
'Kiírjuk az eredményt
Debug.Print "Angol betűk száma:"; db
MsgBox "Az adat és az eredmény az Immediate ablakban látható!"
End Sub
```

Vissza a kérdésekhez!

I. modul záró feladatok - 3. mintafeladat:

```
Sub Mintafeladat3()
Const Mettol = 1
Const Meddig = 10
Dim i As Long, db As Long, st As String
'Kezdőérték
db = 0
'A prímek kiírásához
st = ""
For i = Mettol To Meddig
        If Prim(i) Then
                db = db + 1
                'A prímek kiírásához
                st = st + " " & i
        End If
Next
'A prímek kiírása
Debug.Print "Prímek:"; st
'Az eredmény kiírása
Debug.Print "Darabszám:"; db
```

```
MsgBox "Az eredmény az Immediate ablakban látható!"
End Sub
```

Vissza a kérdésekhez!

II. modul 1. lecke - 1. kérdés:

Maximum 16 tizedesjegyet a törtrészben:

```
>> 1234.5678901234567

ans = 1.234567890123457e+003

>> 1234.56789012345678

ans = 1.234567890123457e+003

>> 0.001234567890123456

ans = 0.001234567890123
```

Vissza a kérdésekhez!

II. modul 1. lecke - 2. kérdés:

Az int64 és uint64 típusú adatokkal nem lehet aritmetikai műveleteket elvégezni:

```
>> t=tic, t+1

t = 307996174668312

??? Undefined function or method 'plus' for input arguments of type 'uint64'.
```

Vissza a kérdésekhez!

II. modul 1. lecke - 3. kérdés:

1e+308

Vissza a kérdésekhez!

II. modul 1. lecke - 4. kérdés:

Igen:

```
>> type('pi'), pi=pi, whos('pi'), pi=pi+1, clear pi, type('pi'), pi

'pi' is a built-in function.

pi = 3.141592653589793

  Name      Size            Bytes  Class     Attributes

  pi        1x1                 8  double

pi = 4.141592653589793

'pi' is a built-in function.

ans = 3.141592653589793
```

Vissza a kérdésekhez!

II. modul 1. lecke - 5. kérdés:

Nem! Skaláris típust nem használ a Matlab. Egyetlen numerikus értéket 1x1 méretű tömbben tárol.

Vissza a kérdésekhez!

II. modul 1. lecke - 6. kérdés:

Az eredmény attól függetlenül 0, hogy a z valós, vagy komplex adatot tárol (lásd komplex aritmetika):

```
>> z=1+i, angle(z), angle(z)+angle(z')

z = 1.000000000000000 + 1.000000000000000i

ans = 0.785398163397448

ans = 0
```

Vissza a kérdésekhez!

II. modul 1. lecke - 7. kérdés:

z abszolútértékének négyzetét:

```
>> z=3+4i, z*z', abs(z)^2

z = 3.000000000000000 + 4.000000000000000i

ans = 25

ans = 25
```

Vissza a kérdésekhez!

II. modul 1. lecke - 8. kérdés:

Nem. A törléshez a Workspace ablak Delete vezérlése használható.

```
>> clear=0,  clear clear

clear = 0

??? Error: "clear" was previously used as a variable,

 conflicting with its use here as the name of a function or command.

 See MATLAB Programming, "How MATLAB Recognizes Function Calls That Use Command

 Syntax" for details.
```

Vissza a kérdésekhez!

II. modul 1. lecke - 9. kérdés:

```
>> x=[], isempty(x)

x = []

ans = 1
```

Vissza a kérdésekhez!

II. modul 1. lecke - 10. kérdés:

Ha egy kétoperandusú logikai művelet eredményét az első operandus meghatározza, akkor a második operandus nem értékelődik ki. Például ha **y** 1x1-es numerikus adat, akkor a **false & y** értéke az y értékétől függetlenül 0 lesz.

Vissza a kérdésekhez!

II. modul 1. lecke - 11. kérdés:

Ha az A mátrix egy zeros() mátrix.

```
>> A=[0 1; 0 0], any(A), any(any(A)), A(1,2)=0,  any(A), any(any(A))


A =

     0     1

     0     0

ans =

     0     1

ans = 1

A =

     0     0

     0     0
```

```
ans =

     0     0

ans = 0
```

Vissza a kérdésekhez!

II. modul 1. lecke - 12. kérdés:

Ha 2*x értéke túlcsordulást okoz, akkor a 2*x/2 kifejezés értéke nem lesz x-el egyenlő.

```
>> x=int16(20000), 2*x/2

x = 20000

ans = 16384
```

Vissza a kérdésekhez!

II. modul 1. lecke - 13. kérdés:

A megjelenő érték 3.

Vissza a kérdésekhez!

II. modul 1. lecke - 14. kérdés:

0, mert 1 + eps/2 értéke 1 marad (lásd eps definíciója).

Vissza a kérdésekhez!

II. modul 2. lecke - 1. kérdés:

```
>> format short, 0:30:360, linspace(0,360,13)

ans =

     0    30    60    90   120   150   180   210   240   270   300   330   360

ans =

     0    30    60    90   120   150   180   210   240   270   300   330   360
```

Vissza a kérdésekhez!

II. modul 2. lecke - 2. kérdés:

```
>> linspace(0,2*pi,13)

 ans =

   Columns 1 through 13

         0    0.5236    1.0472    1.5708    2.0944    2.6180    3.1416    3.6652
    4.1888    4.7124    5.2360    5.7596    6.2832
```

Vissza a kérdésekhez!

II. modul 2. lecke - 3. kérdés:

```
>> [101:104]', [101:1:104]', linspace(101,104,4)'
```

Vissza a kérdésekhez!

II. modul 2. lecke - 4. kérdés:

Magát a D mátrixot, mert triu(D, 1) mátrixban csak a főátló alatti elemek nem nullák, diag(diag(D)) mátrixban csak diagonális nem 0 és a tril(D, -1) mátrixban csak a főátló feletti elemek nem nullák.

Vissza a kérdésekhez!

II. modul 2. lecke - 5. kérdés:

A diag(diag(D, -1), -1) + diag(diag(D, 1), 1) + diag(diag(D)) kifejezés első tagja a főátló alatti mellékátlót, a második tag a főátló feletti mellékátlót, a harmadik tag magát a főátlót eredményezi egy D-vel azonos méretű mátrixban, tehát az eredmény 1-1 mellékátlóval rendelkező sávmátrix lesz.

Vissza a kérdésekhez!

II. modul 2. lecke - 6. kérdés:

D-vel azonos méretű mátrixot, amelyben a főátlón kívüli elemek zérusok.

Vissza a kérdésekhez!

II. modul 2. lecke - 7. kérdés:

```
>> [1:4]'*[10:10:40]
```

Vissza a kérdésekhez!

II. modul 2. lecke - 8. kérdés:

```
>> A~=~[]; n~=~5; for k~=~1:n A=[A; 1./[k:k+n-1]]; end; A
```

Vissza a kérdésekhez!

II. modul 2. lecke - 9. kérdés:

Az eredmény 5, mert egy sorvektor minden elemére hivatkozás a sorvektort oszlopvektorrá transzformálja, így a b' oszlopvektor saját magával hasonlítódik össze. Ennek eredménye ones(5, 1) lesz, mely elemeinek összege adja az eredményt.

Vissza a kérdésekhez!

II. modul 2. lecke - 10. kérdés:

Az eredmények megegyeznek.

```
>> max(sum(abs(A')))== norm(A,Inf)

ans = 1
```

Vissza a kérdésekhez!

II. modul 3. lecke - 1. kérdés:

```
>> A = [1 2 -1; 3 1 4]', B=[3 0 2; -1 2 -2]',

 nA = cross(A(:,1),A(:,2)), nB = cross(B(:,1),B(:,2)), subspace(nA,nB),
 subspace(A,B)

 A =

      1      3

      2      1

     -1      4

 B =

      3     -1

      0      2

      2     -2
```

```
nA =

     9

    -7

    -5

nB =

    -4

     4

     6

ans = 0.4138

ans = 0.4138
```

Vissza a kérdésekhez!

II. modul 3. lecke - 2. kérdés:

```
>> M=[2 1 -5 3 6; 3 -1 4 2 25; -1 -2 3 -2 1; 1 2 1 2 5], A = M(:,1:4), b= M(:,5),

det(A), round(det(A)), x=A\b, rats(x)

M =

     2     1    -5     3     6

     3    -1     4     2    25

    -1    -2     3    -2     1

     1     2     1     2     5

A =

     2     1    -5     3

     3    -1     4     2

    -1    -2     3    -2

     1     2     1     2
```

```
b =

     6

    25

     1

     5

ans = 20.000000000000004

ans = 20

x =

   2.500000000000000

  -3.500000000000000

   1.500000000000000

   4.000000000000001
```

```
ans =

     5/2

    -7/2

     3/2

       4
```

Vissza a kérdésekhez!

II. modul 3. lecke - 3. kérdés:

```
>> H = hilb(4), b = [13/6, 7/6, 49/60, 19/30]', x = H\b, (2,1)=H(2,1)+0.001,
 xm = H\b, hiba=abs(xm-x)
 H =
    1.000000000000000   0.500000000000000   0.333333333333333   0.250000000000000
    0.500000000000000   0.333333333333333   0.250000000000000   0.200000000000000
    0.333333333333333   0.250000000000000   0.200000000000000   0.166666666666667
    0.250000000000000   0.200000000000000   0.166666666666667   0.142857142857143
 b =
    2.166666666666667
    1.166666666666667
    0.816666666666667
    0.633333333333333
```

```
x =

   2.000000000000007

  -1.000000000000101

   2.000000000000260

  -0.000000000000175

H =

   1.000000000000000   0.500000000000000   0.333333333333333   0.250000000000000

   0.501000000000000   0.333333333333333   0.250000000000000   0.200000000000000

   0.333333333333333   0.250000000000000   0.200000000000000   0.166666666666667

   0.250000000000000   0.200000000000000   0.166666666666667   0.142857142857143
```

```
xm =

    2.272727272727259

   -3.727272727272573

    8.136363636363267

   -3.818181818181579

hiba =

    0.272727272727252

    2.727272727272472

    6.136363636363008

    3.818181818181405
```

Vissza a kérdésekhez!

II. modul 3. lecke - 4. kérdés:

Ha x vektor normája 1 és az A mátrix egységmátrix, akkor a Frobenius norma kivételével teljesül az A*x szorzatra a szorzat normája egyenlő a normák szorzatával triviális összefüggés.

```
>> A=eye(3), x=[0.6 0.8 0]', norm(x), norm(x,'fro')

A =

     1     0     0

     0     1     0

     0     0     1

x =

    0.6000

    0.8000

         0

ans = 1

ans = 1

>> norm(A*x)==norm(A)*norm(x), norm(A*x,'fro')==norm(A,'fro')*norm(x,'fro')
```

```
ans = 1

ans = 0
```

Vissza a kérdésekhez!

II. modul 3. lecke - 5. kérdés:

```
>> A = [5 1 -2 1; 1 6 -1 1; -2 -1 4 2; 1 1 2 8], [U L]=eig(A), ellenorzes=A*U - U*L

 A =

      5     1    -2     1

      1     6    -1     1

     -2    -1     4     2

      1     1     2     8

 U =

   -0.5156    0.6092    0.5748   -0.1805

   -0.1366   -0.7493    0.5826   -0.2837

   -0.7730   -0.2010   -0.5541   -0.2348

    0.3435    0.1643   -0.1523   -0.9121
```

```
L =

    1.6005         0         0         0

         0    4.6995         0         0

         0         0    7.6763         0

         0         0         0    9.0237

ellenorzes =

  1.0e-014 *

         0         0         0    0.0666

   -0.0222    0.1332         0         0

   -0.0888    0.0222   -0.0888    0.0888

    0.0666   -0.0222         0    0.1776
```

Látjuk, hogy a sajátértékegyenletek 1e-14-nél kisebb hibával teljesülnek!

Vissza a kérdésekhez!

II. modul 3. lecke - 6. kérdés:

```
>> fi=linspace(0, 2*pi, 361); z=2*(1+cos(fi)).*exp(i*fi); plot(z); axis square,...
hold on, grid on, xlabel('x'), ylabel('y'), title('kardiodid'), hold off
```

Vissza a kérdésekhez!

II. modul 3. lecke - 7. kérdés:

```
>> x = 0:0.2:20; mu=10; sigma=2.5;

y = exp(-(x-mu).^2/(2*sigma))/(sqrt(2*pi)*sigma); plot(x,y)
```

A Matlab a normális eloszlás sűrűségfüggvényét is tudja (normpdf, itt a „pdf” a Probability Density Function rövidítése).

```
>> x = 0:0.2:20;  y = normpdf(x, 10, 2.5);  plot(x,y)
```

Vissza a kérdésekhez!

II. modul 3. lecke - 8. kérdés:

```
>> fplot('t*exp(-t^2)',[0 3]), xlabel('t'), ylabel('y'),
title('t*exp(-t^2) függvény')
```

Vissza a kérdésekhez!

II. modul 4. lecke - 1. kérdés:

```
>> fplot('t*exp(-t^2)',[0 3]), xlabel('t'), ylabel('y'), title('t*exp(-t^2)
függvény')

 [xmax ymax]=ginput(1); hold on, text(1.5, 0.4, ['csúcspont: (' num2str(xmax) ', '
 num2str(ymax) ')'])
```

Vissza a kérdésekhez!

II. modul 4. lecke - 2. kérdés:

```
x = linspace(-2*pi,2*pi,50); y = x; [X Y] = meshgrid(x,y); z = X.^3.*cos(Y);

 mesh(x,y,z); title('Az x^3*cos(y) grafikonja')
```

Vissza a kérdésekhez!

II. modul 4. lecke - 3. kérdés:

```
t = linspace(0,10*pi, 901); plot3(cos(t), sin(t), t), axis square, grid on
```

Vissza a kérdésekhez!

II. modul 5. lecke - 1–4. kérdés:

Ha a függvény definíciója nem érhető el M-fájlban, akkor a következőképp kaphatjuk meg az eredményeket:

```
>> % karakterláncként adjuk meg a függvényt és erre hivatkozunk

fv = 'x.^4 - 3*x.^3 + 2*x.^2 - 0.07',

zhx = [fzero(fv, -0.2) fzero(fv, 0.2) fzero(fv, 0.8) fzero(fv, 2)],
zhy = [0 0 0 0];

% minimumhelyek:

minx = [fminbnd(fv, 1.5, 2) fminbnd(fv, -0.2, 0.2)],

% x változó megkapja a minimumhelyeket, és a függvényt kiértékeljük

x = minx; miny = eval(fv);

% maximumhely a függvény -1-szeresének (ezt is karakterláncként adjuk meg)
minimumhelye

mfv = ['-(' fv ')'], maxx = fminbnd(mfv,0.5,1),

% x változó megkapja a maximumhelyet, és a függvényt kiértékeljük

x = maxx, maxy = eval(fv)

% rajzolások
```

```
fplot(fv, [-1 2.5]), grid on, hold on,

plot(zhx, zhy, 'ko', minx, miny, 'ro', maxx, maxy, 'go'), legend('függvény',
'zérushely', 'minimum', 'maximum')

title([fv ' függvény és nevezetes pontjai'])

% függvény alatti terület a 2. és 3. zérushely között

ter = quad(fv, zhx(2), zhx(3), eps)
```

Vissza a kérdésekhez!

II. modul 5. lecke - 5. kérdés:

Tegyük elérhetővé a következő de5.m függvénydefiníciót!

```
function [ z ] = de5( x, y )

%UNTITLED2 Summary of this function goes here

%   Detailed explanation goes here

z = 10 + 4*sin(x) - 2*sqrt(y);

end
```

Ekkor a megoldás:

```
x = 0:0.1:30; [x y] = ode45('de5',x, 1); plot(x,y); legend('y(x)', 4);
```

Vissza a kérdésekhez!

II. modulzáró feladatok - 1. kérdés:

```
>> szinusz = 'sin'; fplot(szinusz, [0, 2*pi])
```

Vissza a kérdésekhez!

II. modulzáró feladatok - 2. kérdés:

Az időfüggvény M-fájlja:

```
function [ y ] = ft(x, amp, om1, om2, konst)

% Az idöfüggvény definíciója

global amp om1 om2 konst

y = amp*cos(om1*x).*sin(om2*x) + konst;

end
```

A végrehajtott parancsok:

```
>> cd c:\textbackslash munka

global amp om1 om2 konst

fplot('ft', [20, 26.6]), grid on, hold on

% zérushelyek keresése és felrajzolása (2 darab az ábra alapján)

zerushx = [fzero('ft', 20.5) fzero('ft', 26.5)], zerushy = ft(zerushx),

plot(zerushx, zerushy, 'ro')

% lokális minimumpont keresés és felrajzolás
```

```
minx = fminbnd('ft', 22.5, 23.5), miny = ft(minx), plot(minx, miny, 'go')

% lokális maximumpontok keresése és felrajzolás (2 darab)

maxx = [fminbnd('-ft(x)', 21, 22) fminbnd('-ft(x)', 25, 26)], maxy = ft(maxx)

plot(maxx, maxy, 'ko')

% trendvonal alappontjai és a függvényértékek

x = linspace(20, 26.6, 661);  y = ft(x);

% az elsöfokú regressziós polinom együtthatói a p kételemü sorvektorba kerülnek

p = polyfit(x, y, 1)

% regressziós egyenes berajzolása a két végpont összekötésével

xx = [20 26.6]; yy = polyval(p, xx); plot(xx, yy, 'k')

% jelmagyarázat jelei egymás alá (vonalhúzás 2 ponttal)

tx = 22.5;

plot(tx, -0.2, 'ro', tx, -0.7, 'go', tx, -1.2, 'ko', [tx tx+0.3], [-1.7 -1.7],
'k-', [tx tx+0.3], [-2.2 -2.2], 'b-')
```

```
% jelmagyarázat szövegei egymás alá

tx = 23.1;

text(tx, -0.2, 'zérushely'); text(tx, -0.7, 'minimum'); text(tx, -1.2, 'maximum');

text(tx, -1.7, ['elsöfokú trendvonal: ' num2str(p(1)) 'x ' num2str(p(2))]);

text(tx, -2.2, 'f(x) függvény')
```

Vissza a kérdésekhez!

II. modul záró feladatok - 3. kérdés:

```
% begépeljük az A.dat és b.dat text fájlokba az együtthatókat

 % betöltjük az együtthatókat

 load A.dat, load b.dat

 % determináns

 detA = det(A)

 % rangok

 rang_A = rank(A), rang_bovitett = rank([A b])

 % inverz

 i_A = inv(A)

 % inverz tört alakban

 tortinvA = rats(i_A)
```

```
% megoldás balosztóval, ellenörzés, hibaszámítás

x = A\b, ell = A*x, hiba = ell-b
```

Vissza a kérdésekhez!

II. modul záró feladatok - 4. kérdés:

```
% polinomokkal

p = [1, -3,  2, 0, -0.07];

% zérushelyek növekvöen

zhx = sort(roots(p)), zhy = [0, 0, 0, 0];

% elsö és második derivált együtthatói

pd1 = polyder(p), pd2 = polyder(pd1)

% elsö derivált zérushelyei növekvö sorrendben

zhd1 = sort(roots(pd1))

% ezeken a helyeken a második derivált értékeinek elöjele, ettöl függ, hogy
minimum, vagy maximumhely

sgnd2 = sign(polyval(pd2, zhd1));

% a derivált zérushelyeit egy iterációban két csokorba szedjük

minx = []; maxx = [];

for k = 1:length(zhd1)
```

```
  if sgnd2(k)==1 minx = [minx zhd1(k)];  % eggyel több minimumhely
  elseif sgnd2(k)==-1 maxx = [maxx zhd1(k)];  % eggyel több maximumhely
  end
end
% kiíratjuk a minimum és maximumhelyeket
minx, maxx
% ahol szélsöérték van, ott kiszámítjuk a függvényértéket is
miny = polyval(p, minx), maxy = polyval(p, maxx),
% terület a Newton-Leibniz szabállyal
ter = polyval(polyint(p),zhx(3)) - polyval(polyint(p), zhx(2))
```

Vissza a kérdésekhez!

II. modul záró feladatok - 5. kérdés:

```
% a szimmetrikus mátrix

A = rand(5); A = 0.5*(A + A') + eye(5),

% a sajátérték-feladat megoldása

[U L] = eig(A); sajatertekek = sort(diag(L)', 'descend')

% a karakterisztikus polinom gyökei

kar_pol_gyok = roots(poly(A))'
```

Vissza a kérdésekhez!

III. modul 1. lecke - 1. kérdés:

$n = 50$

Vissza a kérdésekhez!

III. modul 1. lecke - 4. kérdés:

={2\3\4; 3\4\5; 4\5\6}

Vissza a kérdésekhez!

III. modul 1. lecke - 10. kérdés:

Az inverz második sorának első eleme: 4/47

Vissza a kérdésekhez!

III. modul 3. lecke - 1. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 3. lecke - 2. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 3. lecke - 3. kérdés:

Tapasztalat: pici változtatás is nagy változást eredményez a megoldásban (gyengén kondicionált mátrix).

Vissza a kérdésekhez!

III. modul 3. lecke - 4. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 3. lecke - 5. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 5. lecke - 1. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 5. lecke - 2. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 5. lecke - 3. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 6. lecke - 1. kérdés:

A leckében bemutatott példák alapján járjunk el.

Az 1. feladat esetében a sűrűségfüggvény egyenletes eloszlást mutat.

Vissza a kérdésekhez!

III. modul 6. lecke - 2. kérdés:

A leckében bemutatott példák alapján járjunk el.

A 2. feladatnál jól megválasztott adatok esetén normális eloszlást kapunk, a sűrűségfüggvény így a Gauss-görbét adja (lásd 5. lecke, 3.6.1. alfejezet).

Vissza a kérdésekhez!

III. modul 6. lecke - 3. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul 6. lecke - 4. kérdés:

A leckében bemutatott példák alapján járjunk el.

Vissza a kérdésekhez!

III. modul záró feladatok - 1. kérdés:

Először az algoritmust adjuk meg.

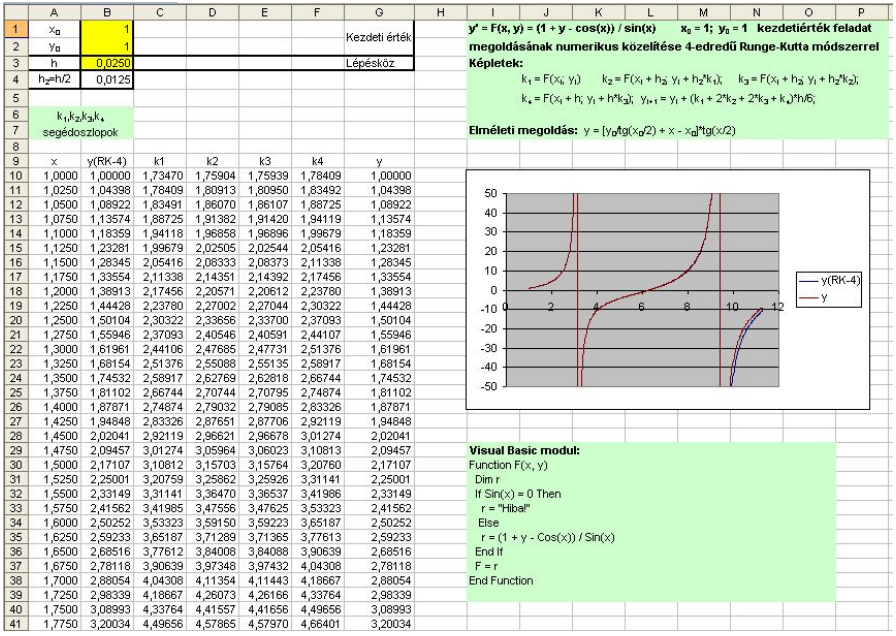
Lánctörtbe alakítás:

1. Legyen $b_0 = [x]$ (egészrész), $x_1 = \{x\}$ (törtrész), ahol $x = b_0 + x_1$.
2. Ha $x_1 \neq 0$, akkor legyen $b_1 = [1/x_1]$, $x_2 = \{1/x_1\}$, ahol $x = b_0 + x_1 = b_0 + 1/(b_1 + x_2)$.
3. Ha $x_2 \neq 0$, akkor legyen $b_2 = [1/x_2]$, $x_3 = \{1/x_2\}$.
4. ...
5. Ha $x_n \neq 0$, akkor legyen $b_n = [1/x_n]$, és $x_{n+1} = \{1/x_n\}$ helyett legyen 0/1.

A lánctört egyszerű törtté visszaalakítása:

1. Írjuk az x_k törtrészeket a visszaalakítási sorozatban $x_k = p_k/q_k$ alakba!
2. A $p_{n+1} = 0$ és $q_{n+1} = 1$ értékekből kiindulva a közös nevezőre hozás a $p_k = q_{k+1}$, $q_k = b_k q_{k+1} + p_{k+1}$, $k = n, n - 1, \ldots, 0$ sorozatot eredményezi, amelynek a végén a q_0/p_0 hányados adja x tört alakú közelítését.
3. Az utolsó lépésnél azért nem a p_0/q_0 hányadossal számolunk, mert ekkor nem kell reciprokot képezni.

Ez az algoritmus az Excel segítségével egy képleteket ismétlő táblázatot eredményez.



	A	B	C	D	E	F	G
1	x_0	1					Kezdeti érték
2	y_0	1					
3	h	0,0250					Lépésköz
4	h_2=h/2	0,0125					
5							
6	k_1,k_2,k_3,k_4						
7	segédoszlopok						
8							
9	x	y(RK-4)	k1	k2	k3	k4	y
10	1,0000	1,00000	1,73470	1,75904	1,75939	1,78409	1,00000
11	1,0250	1,04398	1,78409	1,80913	1,80950	1,83492	1,04398
12	1,0500	1,08922	1,83491	1,86070	1,86107	1,88725	1,08922
13	1,0750	1,13574	1,88725	1,91382	1,91420	1,94119	1,13574
14	1,1000	1,18359	1,94118	1,96858	1,96896	1,99679	1,18359
15	1,1250	1,23281	1,99679	2,02505	2,02544	2,05416	1,23281
16	1,1500	1,28345	2,05416	2,08333	2,08373	2,11338	1,28345
17	1,1750	1,33554	2,11338	2,14351	2,14392	2,17456	1,33554
18	1,2000	1,38913	2,17456	2,20571	2,20612	2,23780	1,38913
19	1,2250	1,44428	2,23780	2,27002	2,27044	2,30322	1,44428
20	1,2500	1,50104	2,30322	2,33656	2,33700	2,37093	1,50104
21	1,2750	1,55946	2,37093	2,40546	2,40591	2,44107	1,55946
22	1,3000	1,61961	2,44106	2,47685	2,47731	2,51376	1,61961
23	1,3250	1,68154	2,51376	2,55088	2,55135	2,58917	1,68154
24	1,3500	1,74532	2,58917	2,62769	2,62818	2,66744	1,74532
25	1,3750	1,81102	2,66744	2,70744	2,70795	2,74874	1,81102
26	1,4000	1,87871	2,74874	2,79032	2,79085	2,83326	1,87871
27	1,4250	1,94848	2,83326	2,87651	2,87706	2,92119	1,94848
28	1,4500	2,02041	2,92119	2,96621	2,96678	3,01274	2,02041
29	1,4750	2,09457	3,01274	3,05964	3,06023	3,10813	2,09457
30	1,5000	2,17107	3,10812	3,15703	3,15764	3,20760	2,17107
31	1,5250	2,25001	3,20759	3,25862	3,25926	3,31141	2,25001
32	1,5500	2,33149	3,31141	3,36470	3,36537	3,41986	2,33149
33	1,5750	2,41562	3,41985	3,47556	3,47625	3,53323	2,41562
34	1,6000	2,50252	3,53323	3,59150	3,59223	3,65187	2,50252
35	1,6250	2,59233	3,65187	3,71289	3,71365	3,77613	2,59233
36	1,6500	2,68516	3,77612	3,84008	3,84088	3,90639	2,68516
37	1,6750	2,78118	3,90639	3,97348	3,97432	4,04308	2,78118
38	1,7000	2,88054	4,04308	4,11354	4,11443	4,18667	2,88054
39	1,7250	2,98339	4,18667	4,26073	4,26166	4,33764	2,98339
40	1,7500	3,08993	4,33764	4,41557	4,41656	4,49656	3,08993
41	1,7750	3,20034	4,49656	4,57865	4,57970	4,66401	3,20034

y' = F(x, y) = (1 + y - cos(x)) / sin(x) x_0 = 1; y_0 = 1 kezdetiérték feladat megoldásának numerikus közelítése 4-edredű Runge-Kutta módszerrel

Képletek:

$k_1 = F(x_i; y_i)$ $k_2 = F(x_i + h_2; y_i + h_2*k_1)$; $k_3 = F(x_i + h_2; y_i + h_2*k_2)$;

$k_4 = F(x_i + h; y_i + h*k_3)$; $y_{i+1} = y_i + (k_1 + 2*k_2 + 2*k_3 + k_4)*h/6$;

Elméleti megoldás: $y = [y_0/tg(x_0/2) + x - x_0]*tg(x/2)$

Visual Basic modul:

```
Function F(x, y)
  Dim r
  If Sin(x) = 0 Then
    r = "Hiba!"
  Else
    r = (1 + y - Cos(x)) / Sin(x)
  End If
  F = r
End Function
```

3.69. ábra: A π racionális közelítése

A négylépéses eljárás a közismert 355/113 közelítést eredményezi. Ezt kapjuk a =Pi() tartalmú cella háromjegyű törtformátumú kijelzésekor is. Az ötlépéses eljárás már 10 decimális értékes jegyre pontos közelítést eredményez!

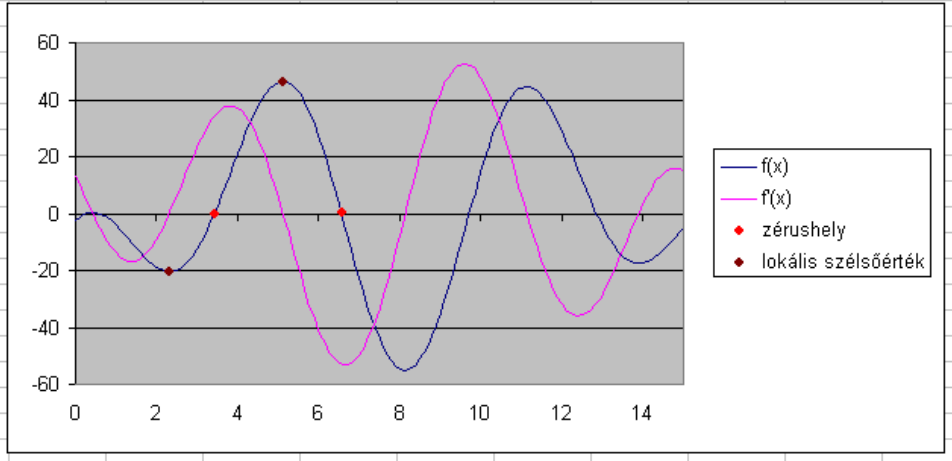
Vissza a kérdésekhez!

III. modul záró feladatok - 2. kérdés:

A	B	D	E
1	-16	9	6

$$f(x) = (Ax^2 + Bx + C)\sin(x + E)$$
$$f'(x) = (2Ax + B)\sin(x + E) + (Ax^2 + Bx + C)\cos(x + E)$$

	-2	
x	f(x)	f'(x)
0,0	-2,51474	13,11218
0,1	-1,34982	10,16419
0,2	-0,48524	7,11600
0,3	0,07213	4,03046
0,4	0,32168	0,96964
0,5	0,26890	-2,00607
0,6	-0,07477	-4,83887
0,7	-0,69229	-7,47440
0,8	-1,56140	-9,86253
0,9	-2,65504	-11,95802
1,0	-3,94192	-13,72123
1,1	-5,38708	-15,11857
1,2	-6,95253	-16,12304
1,3	-8,59791	-16,71449
1,4	-10,28122	-16,87993
1,5	-11,95950	-16,61360
1,6	-13,58959	-15,91706
1,7	-15,12886	-14,79907
1,8	-16,53588	-13,27544
1,9	-17,77117	-11,36871
2,0	-18,79781	-9,10780
2,1	-19,58208	-6,52754
2,2	-20,09400	-3,66813
2,3	-20,30789	-0,57447
2,4	-20,20272	2,70448
2,5	-19,76256	6,11644
2,6	-18,97682	9,60664
2,7	-17,84050	13,11861
2,8	-16,35428	16,59502
2,9	-14,52459	19,97855



x	f(x)	f'(x)	
3,42478	0,00000	34,06734	zérushely
6,56637	0,00000	-52,94471	zérushely
2,31789	-20,31304	0,00000	minimum
5,11837	46,34459	0,00000	maximum

3.70. ábra: Függvény és deriváltja, nevezetes pontok

Javasolt lépések:

1. Az A1:D2 cellákba gépeljük be a konstansok értékeit tartalmazó táblázatot és a 2. sor számértékeit nevesítsük A, B, D, E-vel!
2. Készítsük el az A, B, C oszlopokban az x, f(x), f'(x) értékek fejléces értéktáblázatát a [2; 7] intervallumban 0,1-es lépésközzel. (Az f(x), f'(x) értékek formátuma 5 tizedesjegy legyen.)
3. Ez alapján ábrázoljuk az f(x) és f'(x) függvények grafikonját folytonos görbe vonallal. A példában alkalmazott színek: f(x) – sötétkék, f'(x) – lila. A vízszintes tengelyre az x értékek kerülnek!
4. Határozzuk meg külön táblázatban Solverrel az f(x) függvény [2; 7] intervallumbeli zérushelyeit, szélsőérték-pontjait (ez utóbbiak pontosabb lesznek, ha az f'(x) zérushelyeinek segítségével állapítjuk meg őket)!
5. Ötpontos piros teli pöttyként a zérushelyeket és barna teli pöttyként a szélsőérték-pontokat (csúcspont, gödörpont) tegyük fel az y(x) grafikonra!
6. Ügyeljünk a grafikon tengelyeinek formájára (hely, megjelenés stb.)!

A kész megoldást a 3.70. ábra mutatja.

Vissza a kérdésekhez!

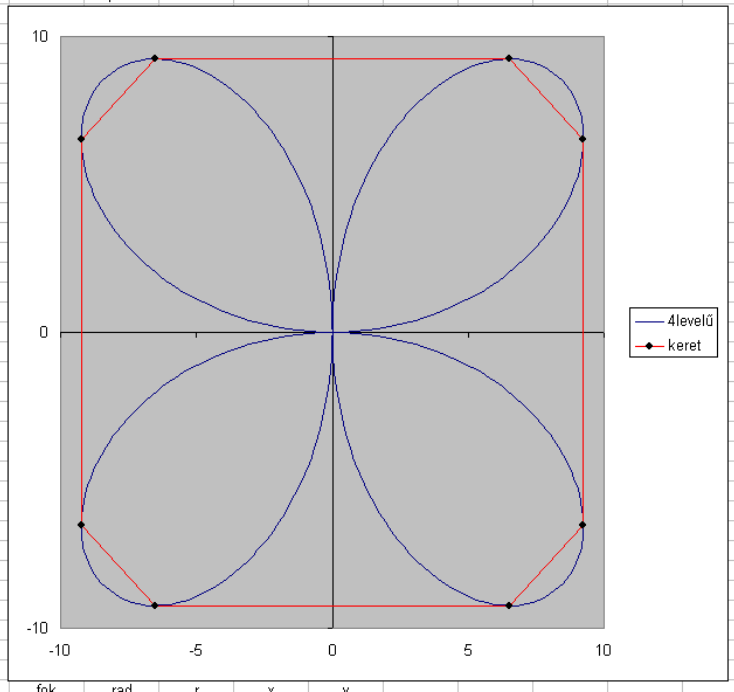
III. modul záró feladatok - 3. kérdés:

A megoldáshoz a fok, rad, r, x, y fejlécű táblázatot használjuk, ahol a fok érték 0-tól 360-ig 1-esével növekedve fut. A rad oszlop a radiánba átszámított fok értékeket tartalmazza. Az r oszlop az r0*abs(sin(2*φ)) értékeket tartalmazza, ennek x és y irányú vetületei a következő két oszlopba kerülnek.

A 8 darab nevezetes pont közül csak az elsőt kell Solverrel meghatározni, a többi a szimmetria miatt már adódik. A pontok zárt összekötéséhez az első pontot kilencedikként is hozzá kell venni. Az összekötő tulajdonságainál a görbített vonal jelölőnégyzet bejelölését törölni kell!

A kész megoldást a 3.71. ábrán láthatjuk.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	fok	rad	r	x	y		r0=	12,00000								
2	0	0,00000	0,00000	0,00000	0,00000											
3	1	0,01745	0,41879	0,41873	0,00731											
4	2	0,03491	0,83708	0,83657	0,02921											
5	3	0,05236	1,25434	1,25262	0,06565											
6	4	0,06981	1,67008	1,66601	0,11650											
7	5	0,08727	2,08378	2,07585	0,18161											
8	6	0,10472	2,49494	2,48127	0,26079											
9	7	0,12217	2,90306	2,88142	0,35379											
10	8	0,13963	3,30765	3,27546	0,46034											
11	9	0,15708	3,70820	3,66255	0,58009											
12	10	0,17453	4,10424	4,04189	0,71269											
13	11	0,19199	4,49528	4,41269	0,85774											
14	12	0,20944	4,88084	4,77418	1,01478											
15	13	0,22689	5,26045	5,12563	1,18334											
16	14	0,24435	5,63366	5,46632	1,36291											
17	15	0,26180	6,00000	5,79555	1,55291											
18	16	0,27925	6,35903	6,11269	1,75279											
19	17	0,29671	6,71031	6,41711	1,96191											
20	18	0,31416	7,05342	6,70820	2,17963											
21	19	0,33161	7,38794	6,98543	2,40528											
22	20	0,34907	7,71345	7,24827	2,63816											
23	21	0,36652	8,02957	7,49625	2,87754											
24	22	0,38397	8,33590	7,72891	3,12268											
25	23	0,40143	8,63208	7,94587	3,37282											
26	24	0,41888	8,91774	8,14676	3,62717											
27	25	0,43633	9,19253	8,33126	3,88493											
28	26	0,45379	9,45613	8,49911	4,14529											
29	27	0,47124	9,70820	8,65007	4,40743											
30	28	0,48869	9,94845	8,78396	4,67051											
31	29	0,50615	10,17658	8,90063	4,93370											
32	30	0,52360	10,39230	9,00000	5,19615											
33	31	0,54105	10,59537	9,08201	5,45702											
34	32	0,55851	10,78553	9,14665	5,71546											
35	33	0,57596	10,96255	9,19396	5,97063											
36	34	0,59341	11,12621	9,22404	6,22170		fok	rad	r	x	y					
37	35	0,61087	11,27631	9,23701	6,46783		35,26439	0,61548	11,31371	9,23760	6,53197					
38	36	0,62832	11,41268	9,23305	6,70820		54,73561	0,95532	11,31371	6,53197	9,23760					
39	37	0,64577	11,53514	9,21237	6,94202		125,2644	2,18628	11,31371	-6,53197	9,23760					
40	38	0,66323	11,64355	9,17524	7,16848		144,7356	2,52611	11,31371	-9,23760	6,53197					
41	39	0,68068	11,73777	9,12196	7,38682		215,2644	3,75707	11,31371	-9,23760	-6,53197					
42	40	0,69813	11,81769	9,05288	7,59627		234,7356	4,09691	11,31371	-6,53197	-9,23760					
43	41	0,71558	11,88322	8,96838	7,79609		305,2644	5,32787	11,31371	6,53197	-9,23760					
44	42	0,73304	11,93426	8,86889	7,98558		324,7356	5,66771	11,31371	9,23760	-6,53197					
45	43	0,75049	11,97077	8,75487	8,16404		35,26439	0,61548	11,31371	9,23760	6,53197					
46	44	0,76794	11,99269	8,62682	8,33082											



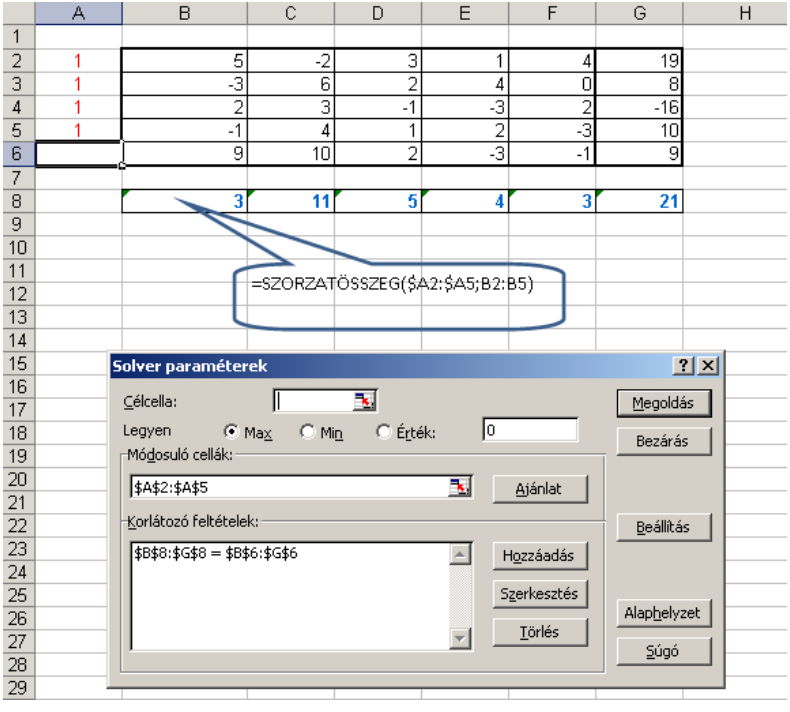
3.71. ábra: Négylevelű lóhere

Vissza a kérdésekhez!

III. modul záró feladatok - 4. kérdés:

Az egyenletrendszer összefüggő, mert az [A b] 5x6-os mátrix minden ötödrendű mátrixának a determinánsa 0, de az utolsó sor elhagyásával kapott mátrix minden 4x4 méretű részének a determinánsa már nem 0.

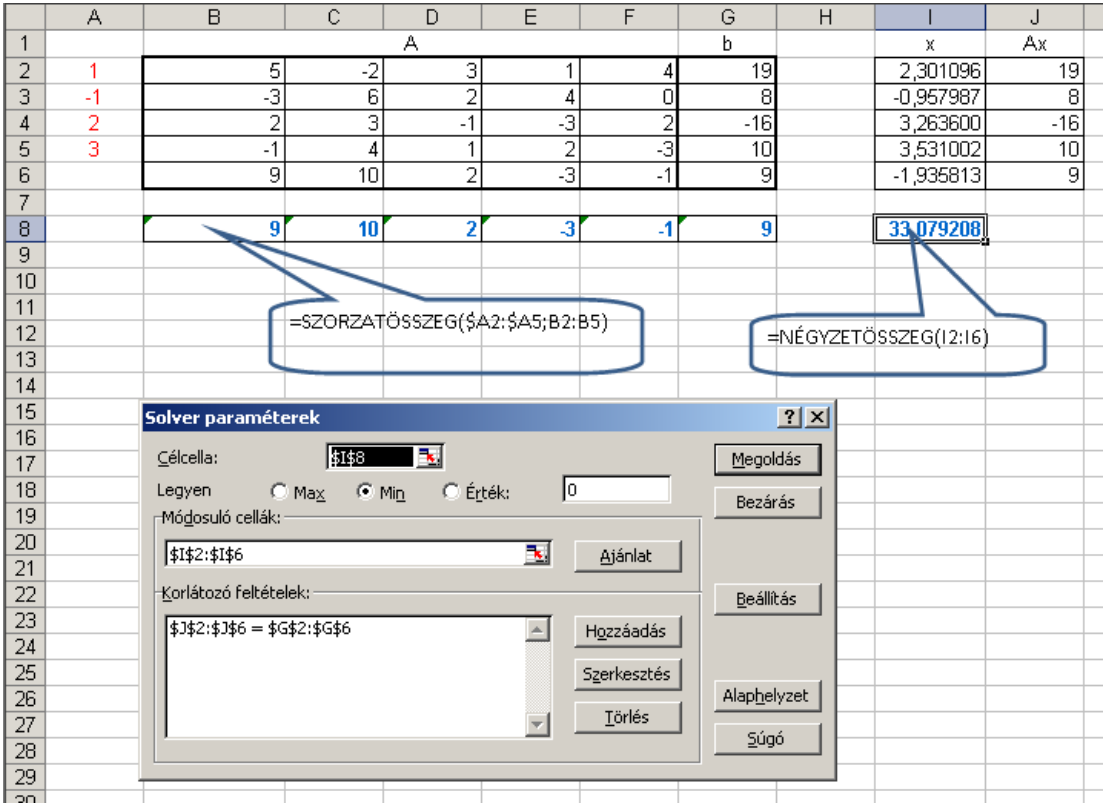
A lineáris kombinációs feladatot a Solverrel oldjuk meg. Beállítjuk, hogy a nyolcadik sorbeli szorzatösszegeknek fel kell venniük az ötödik egyenletbeli együtthatók értékeit (3.72. ábra). A megoldás [1; -1; 2; 3] (az ábrán még a kezdőértéknek megadott 1-es koordinátákat láthatjuk).



3.72. ábra: A Solver beállításai a lineáris kombinációs feladatnál

Annak ellenőrzése, hogy az **x** = [2; -1; 4; 3; -2] vektor valóban megoldása az egyenletrendszernek, az Mszorzat függvénnyel végezhető el.

A megadott megoldásvektor normájának négyzete 34, a Solverrel ezen próbálunk meg javítani (3.73. ábra).



3.73. ábra: Minimális hosszú megoldásvektor előállítása a Solverrel

Vissza a kérdésekhez!

III. modul záró feladatok - 5. kérdés:

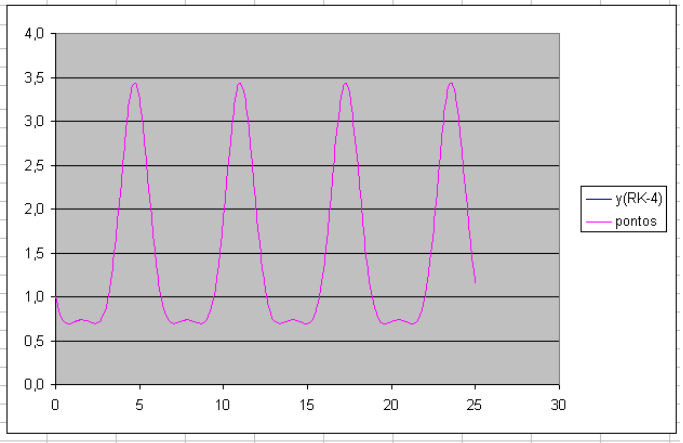
Csak annyi a dolgunk, hogy az ismertetett RK-4 módszer sémájában az intervallumot, annak lépésközét, valamint a kétváltozós függvényt számoló Visual Basic modult és a pontos megoldás tartalmazó oszlopot lecseréljük.

Az új VB modulunk:

```
Function F(x,y)

    F = 0.5*Sin(2*x)-y*cos(x)

End Function
```

A megoldás a 3.74. ábrán látható.

	A	B	C	D	E	F	G	H
1	x_0	0					Kezdeti érték	
2	y_0	1						
3	h	0,1000					Lépésköz	
4	h_2=h/2	0,0500						
5								
6	k_1,k_2,k_3,k_4							
7	segédoszlopok							
8								
9	x	y(RK-4)	k1	k2	k3	k4	y	eltérés
10	0,0	1,00000	-1,00000	-0,89890	-0,90394	-0,80573	1,00000	0,00000000
11	0,1	0,90981	-0,80593	-0,71199	-0,71663	-0,62673	0,90981	0,00000012
12	0,2	0,83831	-0,62689	-0,54217	-0,54627	-0,46636	0,83831	0,00000020
13	0,3	0,78381	-0,46648	-0,39227	-0,39576	-0,32681	0,78381	0,00000026
14	0,4	0,74432	-0,32689	-0,26384	-0,26668	-0,20906	0,74432	0,00000029
15	0,5	0,71770	-0,20911	-0,15734	-0,15955	-0,11316	0,71770	0,00000031
16	0,6	0,70177	-0,11318	-0,07238	-0,07401	-0,03836	0,70177	0,00000032
17	0,7	0,69436	-0,03835	-0,00791	-0,00902	0,01665	0,69436	0,00000033
18	0,8	0,69344	0,01666	0,03762	0,03693	0,05358	0,69344	0,00000034
19	0,9	0,69709	0,05360	0,06610	0,06574	0,07446	0,69709	0,00000036
20	1,0	0,70362	0,07448	0,07965	0,07952	0,08148	0,70362	0,00000038
21	1,1	0,71153	0,08150	0,08054	0,08056	0,07698	0,71153	0,00000041
22	1,2	0,71954	0,07700	0,07114	0,07123	0,06337	0,71954	0,00000044
23	1,3	0,72662	0,06338	0,05386	0,05396	0,04307	0,72662	0,00000047
24	1,4	0,73199	0,04308	0,03116	0,03123	0,01856	0,73199	0,00000049
25	1,5	0,73510	0,01856	0,00548	0,00550	-0,00771	0,73510	0,00000051
26	1,6	0,73565	-0,00771	-0,02070	-0,02075	-0,03325	0,73565	0,00000051
27	1,7	0,73358	-0,03325	-0,04493	-0,04503	-0,05561	0,73358	0,00000050
28	1,8	0,72910	-0,05561	-0,06475	-0,06488	-0,07232	0,72910	0,00000048
29	1,9	0,72265	-0,07230	-0,07771	-0,07781	-0,08091	0,72265	0,00000046
30	2,0	0,71491	-0,08089	-0,08138	-0,08139	-0,07898	0,71491	0,00000043
31	2,1	0,70682	-0,07895	-0,07336	-0,07321	-0,06414	0,70682	0,00000040
32	2,2	0,69955	-0,06411	-0,05134	-0,05094	-0,03414	0,69955	0,00000037
33	2,3	0,69450	-0,03411	-0,01312	-0,01239	0,01313	0,69450	0,00000035
34	2,4	0,69330	0,01316	0,04329	0,04445	0,07954	0,69330	0,00000034
35	2,5	0,69777	0,07956	0,11958	0,12125	0,16658	0,69777	0,00000033



3.74. ábra: Differenciálegyenlet közelítő megoldása

Vissza a kérdésekhez!

III. modul záró feladatok - 6. kérdés:

A 3.75. ábrán bemutatjuk a megoldáshoz szükséges táblázatokat és a Solver paraméterezését.

	A	B	C	D	E	F	G	H	I	J	K	L
1			*Az x telephelyről az y területi raktárba szállítandó termékek száma*									
2	*Telephely:*	*Összesen(sorösszeg)*	*Sopron*	*Pápa*	*Kaposvár*	*Veszprém*	*Budapest*					
3	Kecskemét	5	1	1	1	1	1			Színkódok		
4	Pécs	5	1	1	1	1	1					
5	Szombathely	5	1	1	1	1	1				Célcella	
6												
7	Összesen(oszlopösszeg):		3	3	3	3	3				Módosuló cellák	
8												
9		*Nagykereskedelmi igény -->*	180	80	200	160	220				Korlátozó feltételek	
10	*Telephely:*	*Készlet*	*Az x telephelyről az y területi raktárba való szállítás költsége:*									
11	Kecskemét	310	10 000	8 000	6 000	5 000	4 000					
12	Pécs	260	6 000	5 000	4 000	3 000	6 000					
13	Szombathely	280	3 000	4 000	5 000	5 000	9 000					
14												
15	*Szállítási költség:*	**83 000 Ft**	19 000 Ft	17 000 Ft	15 000 Ft	13 000 Ft	19 000 Ft					
16												
17												
18	**A Solver paraméterezése:**											
19												
20	Célcella	B15	Cél a teljes szállítási költség minimalizálása.									
21												
22	Módosuló cellák	C3:G5	Az egyes telephelyekről az egyes raktárakba szállítandó mennyiségek.									
23												
24	Korlátozó feltételek	B3:B5 <= B11:B13	Az összesen elszállított mennyiség nem haladhatja meg a telephely raktárkészletét.									
25												
26		C7:G7 >= C9:G9	Az egyes raktárakba beszállított mennyiségnek legalább a raktárban jelentkező igényt fedeznie kell.									
27												
28		C3:G5 >= 0	Csak nem-negatív mennyiség szállítható.									
29												
30	Modell:	lineáris										
31												
32	A C15 cellába a Sopronba érkező mennyiségek összköltsége kerül =SZORZATÖSSZEG(C3:C5;C11:C13)											
33	A D15 cellába a Pápára érkező mennyiségek összköltsége kerül, stb.											
34												

3.75. ábra: A raktáros feladat megoldása

Vissza a kérdésekhez!

Irodalomjegyzék

[1] Kovalcsik Géza: *Az Excel programozása.* Computerbooks, Budapest, 2005.

[2] Kuzmina Jekatyerina, Tamás Péter, Tóth Bertalan: *Programozzunk Visual Basic rendszerben!* Computerbooks, Budapest, 2006.

[3] Pusztai Pál: *Algoritmusok és adatstruktúrák,* UNIVERSITAS-GYŐR Nonprofit Kft., 2008.

[4] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.: *Új algoritmusok.* Scolar kiadó, 2003.

[5] Csendes Tibor: *Közelítő és szimbolikus számítások*, Polygon Kiadó, Szeged, 2007.

[6] Rózsa Pál: *Lineáris algebra és alkalmazásai*, Műszaki Könyvkiadó, Budapest, 1974.

[7] Iványi Antal szerk.: *Informatikai algoritmusok I.*, ELTE Eötvös kiadó, Budapest, 2004.

[8] Kiss Béla, Krebsz Anna: *Lineáris algebra, többváltozós függvények, valószínűségszámítás*, Universitas-Győr Kht., 2005.

[9] Stoyan Gisbert: *Matlab – frissített kiadás*, Typotex, 2005.

[10] http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

[11] http://nyelvek.inf.elte.hu/leirasok/Matlab/index.php?chapter=1

[12] G. A. Korn, T. M. Korn: *Matematikai kézikönyv műszakiaknak*, Műszaki Könyvkiadó, Budapest, 1975.

[13] Máté László: *Funkcionálanalízis műszakiaknak*, Műszaki Könyvkiadó, Budapest, 1976.

[14] Csikós Miklósné: *LP feladatok megoldása Excellel* (*oktatási segédanyag*), SZIE GEK, Gödöllő, 2006.

[15] Ferenczi Zoltán: *Operációkutatás*, Novadat, Győr, 2000.

[16] I. N. Bronstejn, K. A. Szemengyajev: *Matematikai zsebkönyv mérnökök és mérnökhallgatók számára*, 6. átdolgozott kiadás, Műszaki Könyvkiadó, Budapest, 1987.

[17] Szörényi Miklós: *Előzetes elemzés a Silver-Knight és a hagyományos padlóborítás összehasonlításáról (kézirat)*, 2012.

[18] Szörényi Miklós: *Szemléletes lineáris algebra – összefoglaló (kézirat)*, http://www.sze.hu/˜szorenyi/linal1.pdf

[19] http://office.microsoft.com/hu-hu/excel-help/az-excel-2010-ujdonsagai-HA010369709.aspx

[21] http://office.microsoft.com/hu-hu/excel-help/problema-meghatarozasa-es-megoldasa-a-solverrel-HP0103424

[22] Michael J. Hernandez - John L. Viescas: *SQL-lekérdezések földi halandóknak*, Kiskapu Kiadó, Budapest, 2009.

[23] Buza Antal: *Az adatbáziskezelés alapjai*. Dunaújváros, 2011.

[24] Reizinger Zoltán: *Bevezetés az adatbázis használatába*, Sun Report, 2009.