Szili László és Tóth János

Matematika
és
Mathematica

ELTE Eötvös Kiadó
Budapest, 1996

Lektorálta

Pelikán József
okleveles matematikus

Ván Péter
okleveles fizikus

Nyelvi lektor

Homonyik Andrea

A fedelet tervezte

Dukay Barna

ISBN 963 463 004 9

Tartalomjegyzék

Előszó 7

1. Matematikai programcsomagok 13
1.1. Szimbolikus programcsomagok 14
1.2. Numerikus programcsomagok 17

2. Ismerkedés a *Mathematica* programmal 19
2.1. A legfontosabb tudnivalók 21
2.2. Programcsomagok 25
2.3. Az alapelvekről 29
2.3.1. Alapvető adatszerkezetek 29
2.3.2. Értékadások típusai. Mintázatok 35
2.3.3 Opciók 41
2.3.4. Attribútumok 44
2.3.5. Kiértékelés 46
2.4. Külső kapcsolatok 48
2.4.1. Be- és kimenet 49
2.4.2. A program környezete 58
2.4.3. Kapcsolat más nyelvekkel és programokkal 62
2.5. További információforrások 66
2.5.1. MathSource 67
2.5.2. Könyvek 67
2.5.3. Folyóiratok 67
2.5.4. MathGroup 68

3. Fejezetek a matematikából 69
3.1. Alapvető matematikai fogalmak 69
3.1.1. Logikai műveletek 70
3.1.2. Halmazok 77
3.1.3. Számok ábrázolása. Aritmetikai műveletek 87
3.1.4. Függvények 100
3.1.5. Függvények ábrázolása 116
3.1.6. Gyakorlatok és feladatok 133

3.2. Matematikai kifejezések 135
3.2.1. Algebrai polinomok 136
3.2.2. Racionális kifejezések 146
3.2.3. Komplex változós kifejezések 148
3.2.4. Trigonometrikus kifejezések 150
3.2.5. Egyéb matematikai kifejezések 153
3.2.6. Nevezetes összegek és szorzatok 155
3.3. Egyenletek megoldása 158
3.3.1. Az utasítások szintaxisa 158
3.3.2. Egyenletek pontos megoldása 163
3.3.3. Egyenletek közelítő megoldása 176
3.4. Analízis 181
3.4.1. Beépített speciális függvények 181
3.4.2. Számsorozatok és számsorok 183
3.4.3. Függvények határértéke 190
3.4.4. Differenciálszámítás 192
3.4.5. Integrálszámítás 201
3.4.6. Függvényközelítések 209
3.4.7. Vektoranalízis 221
3.5. Differenciálegyenletek 225
3.5.1. Iránymező két és három dimenzióban 225
3.5.2. Megoldás kvadratúrával 228
3.5.3. Első integrálok 231
3.5.4. Numerikus megoldás 231
3.5.5. Laplace-transzformáció 234
3.5.6. Megoldás hatványsorokkal 236
3.5.7. Szukcesszív approximáció 238
3.5.8. Stabilitáselmélet 240
3.5.9. Parciális differenciálegyenletek 242
3.5.10. Variációszámítás 245
3.5.11. Gyakorlatok és feladatok 246
3.6. Diszkrét matematika 248
3.6.1. Adott tulajdonságú listák 248
3.6.2. Leszámlálás 251
3.6.3. Egyszerű kombinatorikai azonosságok 253
3.6.4. Differenciaegyenletek, generátorfüggvények . . . 254
3.6.5. Gráfok és folyamok 259
3.6.6. Gyakorlatok és feladatok 263
3.7. Geometria 264
3.7.1. Geometriai alakzatok 264
3.7.2. További lehetőségek 269

3.8. Lineáris algebra 270
3.8.1. Vektorok és mátrixok megadása 270
3.8.2. Részmátrixok kezelése 276
3.8.3. Műveletek vektorokkal és mátrixokkal 278
3.8.4. Vektornormák és mátrixnormák 288
3.8.5. A Gram–Schmidt-féle ortogonalizációs eljárás . . 291
3.8.6. Lineáris egyenletrendszerek megoldása 293
3.8.7. Mátrix sajátértékei és sajátvektorai 296
3.8.8. Mátrix felbontása 299
3.8.9. Lineáris programozás 301
3.9. Számelmélet 304
3.9.1. Számrendszerek 305
3.9.2. Oszthatóság 306
3.9.3. Lánctörtek 309
3.9.4. Prímszámok. A számelmélet alaptétele 311
3.9.5. Számelméleti függvények 314
3.9.6. Kongruenciák 316
3.9.7. További fejezetek 319
3.9.8. Gyakorlatok és feladatok 320
3.10. Valószínűségszámítás 325
3.10.1. Klasszikus valószínűségi mezők 325
3.10.2. Generátorfüggvények 327
3.10.3. Tetszőleges valószínűségi változók 328
3.10.4. Karakterisztikus függvények 330
3.10.5. Határeloszlás-tételek 331
3.10.6. Véletlenszám-generálás 332
3.10.7. Markov-láncok 334
3.10.8. Folytonos idejű Markov-folyamatok 335
3.10.9. Gyakorlatok és feladatok 337
3.11. Matematikai statisztika 338
3.11.1. Az adatok előkészítése 338
3.11.2. Statisztikák 342
3.11.3. Becslések 345
3.11.4. Hipotézisvizsgálat 349
3.11.5. Korreláció- és regresszióanalízis. Szórásanalízis . . 350
3.11.6. Idősorok 352
3.11.7. Összetett feladatok, új módszerek 356
3.11.8. Gyakorlatok és feladatok 356

4. A *Mathematica* programozásáról 357
4.1. Programnyelvi elemek 358
4.2. Gyorsaság, gyorsítás 364
4.2.1. Tipikus műveletek időigénye 365
4.2.2. Gyorsítás emlékező függvényekkel 367
4.2.3. Gyorsítás a `Compile` függvénnyel 367
4.2.4. Gyorsítás listaműveletekkel 369
4.3. Tanácsok programok készítéséhez 371
4.4. Saját programcsomag készítése 372

5. Matematikán kívüli alkalmazások 373
5.1. Üzleti grafika 373
5.2. Hang 375
5.3. Idő 377
5.4. Fizika (mértékegységek) 379
5.5. Földrajz (térképek) 381
5.6. Kémia (elemek) 383

Irodalomjegyzék 385

Tárgymutató 391

Előszó

Az elmúlt tíz évben rohamosan fejlődő, szimbolikus számításokra is képes matematikai programcsomagok új távlatokat nyitnak a matematika alkalmazásai, kutatása és oktatása előtt. A matematika sok területéhez segítséget nyújtó szimbolikus programcsomagok közül az egyik legnépszerűbb és legelterjedtebb a Wolfram Research Institute által létrehozott és terjesztett *Mathematica* program (néha *nyelv*et mondunk). A program megvásárlója az összesen több mint 1400 oldalnyi [3, 89] referenciakönyveket is kézhez kapja. Angolul, németül, japánul, franciául és más nyelveken már majdnem 100 könyv jelent meg különféle aspektusairól és alkalmazásairól. Magyar nyelven azonban eddig nem állt rendelkezésre rendszeres bevezető. Ennek a hiánynak a pótlásán kívül azt is célul tűztük ki, hogy viszonylag rövid terjedelemben, lehetőség szerint teljes áttekintést adjunk a nyelv lehetőségeiről, beleértve a kiegészítő csomagokat is. Azt szeretnénk, ha a matematika valamely fejezete iránt érdeklődő Olvasót a legrövidebb időn belül eljuttathatnánk oda, hogy a lehető legmagasabb szinten kapjon segítséget a programtól saját alkalmazási, kutatási és oktatási feladatainak megoldásához.

• Előzmények, olvasók, előismeretek

A *Mathematica* programcsomagot 1992-ben kezdtük el oktatni a Gödöllői Agrártudományi Egyetemen, egy évvel később pedig az Eötvös Loránd Tudományegyetemen. Itt szerzett tapasztalataink alapján az Olvasó várhatóan rendelkezik elemi számítógépes ismeretekkel, és *érdekli a matematika alkalmazása, vagy oktatásának (számítógépek felhasználásával történő) megújítási lehetőségei.* A fizikus, geofizikus, gépészmérnök, programozó, programtervező stb. szakos egyetemistákon kívül rendszeresen találkoztunk doktoranduszokkal és fiatal egyetemi oktatókkal is a hallgatóság soraiban, köztük gyakorló biológussal, fizikussal, matematikussal, mérnökkel, vegyésszel. Az előforduló matematikai fogalmakat és tételeket csak abban az esetben ismertettük, amikor ennek szükségét éreztük. Mindezek alapján úgy gondoljuk, hogy matematikából a könyv nagy részének elolvasásához elegendő

valamilyen egy-másfél éves egyetemi kurzus elvégzése, de időnként rá fogunk mutatni azokra a lehetőségekre, amelyek a matematikust segíthetik a kutatásban.

Különleges szerepük lehet a *tanár*-olvasóknak: ők segíthetnek annak megmutatásában, hogy akár az általános iskolától kezdve lehet találni a programban felhasználásra méltó, érdekes elemeket.

• Miért a *Mathematica*?

Néhány megjegyzést teszünk a többi szimbolikus csomagról és a *Mathematica* hozzájuk való viszonyáról. Mindenekelőtt megemlítjük, hogy magyar nyelven az első ilyen programcsomagról szóló könyv a DERIVE program oktatási alkalmazásairól számolt be [79], s ez már két kiadásban is napvilágot látott. A legutóbbi időben több könyv is megjelent illetve megjelenőben van a — matematikus körökben szintén népszerű — MAPLE V programról [42, 56]. Magunk mindezt örvendetes eseménynek tekintjük, s reméljük, hogy az angol nyelvű irodalommal szemben meglévő elmaradás felszámolásához jelen munkával is hozzájárulhatunk. Azt reméljük, hogy a MAPLE V vagy a DERIVE program hűséges használója is talál ötleteket könyvünkben, ahogyan mi is vettünk más nyelvekről szóló könyvekből feladatokat, oktatási és programozási ötleteket egyaránt.

Miért éppen a *Mathematicá*val foglalkozunk?

Jelenlegi ismereteink alapján úgy látjuk, hogy a matematikai programcsomagok már ma is, és a közeli jövőben még inkább képesek lesznek arra, hogy a programozással, számítástudománnyal és numerikus matematikával nem hivatásszerűen, de kényszerűen foglalkozó fizikusok, mérnökök, orvosok, vegyészek és más szakemberek *szellemi energiájának jelentős részét felszabadítsák.* Ily módon sokkal több idejük marad a megfelelő modellek felállítására, a kapott eredmények elemzésére és értelmezésére.

Felületes szemlélő számára a legnépszerűbb szimbolikus programcsomagok — mivel nyilvánvalóak és jól definiálhatóak a felhasználói igények — hasonlóan viselkednek. Jellemző, hogy az egyik program hirdetésében szereplő feladatok a másikkal általában megoldhatók. Amiben az egyik lemarad, igyekszik utolérni a másikat. A *Mathematica* például grafikai és animációs képességeivel (ideértve a hangképzést is) tűnt ki korábban, a többiek számára ebben ez a program volt a minta.

Amiben ma is kiemelkedőnek tűnik, az a különböző *programozási stílusok*at lehetővé tévő eszköztára. A többi nyelv olyannak látszik, mint egy hagyományos procedurális programozási nyelv, azzal a többlettel, hogy szimbólumokkal és tetszőlegesen nagy, illetve tetszőlegesen pontos számokkal is lehet számolni bennük. A *Mathematicá*ban viszont mód van arra is, hogy

használjuk a szabályalapú és az objektumorientált programozás módszereit; végül — de talán ez a legfontosabb — a rendszer készítői és aktív használói a *funkcionális programozás*t különlegesen hatékony eszközzé tették. Újra (nem először, de valószínűleg nem utoljára a számítástechnika történetében) elmondható, hogy általa lehetővé válik, hogy a felhasználónak csak a feladatot kell megfogalmaznia, mégpedig a matematika nyelvén, s a megoldás teljesen a program dolga. Természetesen nem lehet eléggé hangsúlyozni az *ellenőrzés* fontosságát, hiszen bonyolult programról (programokról) van szó.

Tegyünk néhány kritikai megjegyzést is. A *Mathematica* rendkívül *hosszú azonosítók*at használ, hogy ne kelljen a felhasználónak kitalálnia az esetleges rövidítéseket. Ízlés kérdése, hogy ezt a tényt a felhasználó hogyan viseli el. Ahhoz, hogy a programmal komolyabb *szimbolikus számítások*at lehessen végezni (ami pedig a legalapvetőbb deklarált célja az ilyen programoknak), nagyon alaposan meg kell azt ismerni. Néhány esetben egészen egyszerű feladatok megoldásaként hibás eredményt kaphatunk. Ez figyelmeztet arra, hogy az eredményeket kritikusan kell fogadnunk és érdemes többféleképpen is ellenőriznünk. Időnként az a benyomásunk, hogy egyes feladatokat csak nagyon nehézkesen lehet vele megoldani. (Például amiatt, hogy a számok alapértelmezésben komplexek.) Ennek a hátránynak az értékelésénél vegyük azonban figyelembe, hogy egyetlen gyakorló matematikusnak vagy alkalmazónak sem kell a matematika *minden* területére tekintettel lennie, elég, ha egy szűk területen viszonylag következetes elnevezéseket, jelöléseket, szokásokat alakít ki. A *Mathematica* viszont, amint Wolfram könyvének alcíme mondja, olyan rendszer, amely lehetővé teszi a számítógépes matematizálást (nem pedig az algebrai, numerikus vagy statisztikai vizsgálatokat). Ez a nagyigényű célkitűzés rengeteg olyan problémát vet fel — és a megoldások nem is olyan rosszak —, amellyel gyakorló matematikus vagy alkalmazó nem is találkozik.

• Alkalmazás, oktatás, kutatás

A matematika *alkalmazás*ában (akár új területről, akár ismert eredmények oktatásáról van szó), a matematikai programcsomagok használatának jelentősége abban áll, hogy a súlypont áttevődhet a *módszerek*ről az *esettanulmányok*ra. Eddig egy-egy alkalmazási feladat megoldása rengeteg energiát emésztett föl, ma pedig ezekkel a programokkal egy-egy tanóra alatt teljes alkalmazások megoldása végigkövethető.

Az eddigiekben elsősorban matematikán kívüli alkalmazásokra gondoltunk. Reméljük azonban, hogy egyre több matematikus is hasznosnak fogja találni *kutatómunká*jához egy ilyen program megismerését, mert a segítsé-

gével fölvetődő munkahipotézisei sok esetben sokkal gyorsabban ellenőrizhetők, mint papíron, ceruzával.

Szinte nyilvánvaló, hogy a matematika *oktatásá*ban egy ilyen programcsomag jól használható. Sokkal kevésbé nyilvánvaló az, hogy hogyan. A matematika oktatásánál általános elv lehet, hogy valamely terület alapfogalmait a hagyományos módon érdemes elsajátíttatni (eközben legfeljebb a tanár használja a programcsomagot illusztrálás céljából), majd a következő területnél a már megtanult részhez tartozó számításokat a hallgató is végezheti számítógéppel. Például a lineáris algebrai alapok megfelelő elsajátítása és begyakorlása után már sokkal hatékonyabban tudunk a differenciálegyenletekre vagy a statisztikára koncentrálni, ha a sajátértékeket és sajátvektorokat nem kézzel számoljuk.

A matematika oktatásának két szélsőséges álláspontja — sarkított megfogalmazásban — az élményszerzést, illetve a pontosságot tartja a legfontosabbnak. Örvendetes, hogy a *Mathematica* mindkét álláspont képviselőinek pozitív érvekkel szolgálhat: módot ad arra, hogy akár a tiszta, akár az alkalmazott matematika területén a tanuló a megszokottnál jóval több példával ismerkedhessék meg (anélkül, hogy beleveszne az unalmas rutinmunkába), másrészt lehetővé teszi, hogy a matematikában megszokott legszigorúbb (esetenként azon túlmenő) pontosságot alkalmazzunk.

• Felépítés

A könyv *felépítése* a következő: Először általános áttekintést adunk a matematikai programcsomagokról, majd a *Mathematicá*ról. Ez utóbbi fejezet 2.3. szakasza ismerteti a *Mathematica* konstrukciós alapelveit; ehhez a nehezebb részhez a későbbiekben többször érdemes visszatérni, sőt: bocsánatos bűnnek tartjuk, ha a türelmetlen Olvasó első alkalommal nem olvassa végig ezt a szakaszt, hanem sietve rátér a 3. fejezet őt érdeklő szakaszára. A könyv gerincét ugyanis a 3. fejezet képezi, amely a matematika néhány fontos területét tárgyalja, abból a szempontból, hogy az adott terület feladatainak megoldása hogyan tehető könnyebbé a program felhasználásával. Mielőtt azonban az Olvasó rátér az analízisről vagy a számelméletről szóló szakasz olvasására, valószínűleg helyesen teszi, ha a 3.1.3–3.1.5. pontot, a 3.2. szakaszt (és esetleg a 3.3.-at is) áttanulmányozza. A programozás iránt érdeklődő Olvasó számára nyilván nem a 3. fejezet valamelyik szakasza, hanem a 4. fejezet nyújt némi (rendkívül vázlatos) tájékoztatást. Azt gondoljuk, hogy — feladattól függően — majdnem mindenki hasznosíthatja majd az 5. fejezet valamelyik részét. Végül a hivatkozások és kulcsszavak jegyzékét abban a reményben állítottuk össze, hogy ezzel minden Olvasó számára könnyebben kezelhetővé tesszük a könyvet.

Az egyes részeknél közölt gyakorlatok és feladatok nehézségi szintje és száma változó. Ezen a helyzeten talán egy később összeállítandó példatár fog majd segíteni. Úgy véljük azonban, hogy a legtöbb Olvasó rendelkezik számos saját problémával, és éppen azok megoldásához keres hatékony eszközt.

Az Olvasó munkáját megkönnyítheti az, hogy a könyv példáit elhelyeztük a MathSource-on (lásd a 2.5.1. pontot). Létezik viszont egy különleges funkciójú ingyenes kiegészítő program is ugyanott (0204–972), a MathReader, amellyel a *Mathematica* által írott állományok elolvashatók. Reméljük, hogy példáink és a MathReader együttesen teljes és gyors áttekintést adnak a *Mathematica* lehetőségeiről annak az Olvasónak, aki még nem rendelkezik a programcsomaggal.

• Jelölések

$:=$	definiáló egyenlőség
$A \subset B$	az A halmaz (nem feltétlenül valódi) részhalmaza a B halmaznak
$\mathbf{N}$	a pozitív egész számok halmaza
$\mathbf{Z}$	az egész számok halmaza
$\mathbf{Z}_p$	Adot p prímszám esetén az egész számok halmazának $\bmod p$ maradékosztályai által meghatározott test
$\mathbf{Q}$	a racionális számok halmaza
$\mathbf{R}$	a valós számok halmaza
$\mathbf{C}$	a komplex számok halmaza
$\mathbf{C}^{m\times n} \quad (m,n \in \mathbf{N})$	az $m \times n$-es komplex értékű mátrixok halmaza
$f : A \to B$	az A halmazon értelmezett B halmazbeli értékeket felvevő függvény

• Köszönetnyilvánítás

A *Mathematica* programmal való ismerkedés és a könyv megírása közben számos ötletet és hasznosítható bíráló megjegyzést, valamint biztató támogatást kaptunk kollégáinktól és tanítványainktól az Eötvös Loránd Tudományegyetemen és a Gödöllői Agrártudományi Egyetemen.

A program használata közben felmerült kérdéseinkkel elsősorban Warren Boont, a Wolfram Research Inc. (röviden: WRI) hivatásos európai tanácsadóját bombáztuk, a felfedezett vélt és valóságos hibák miatt őt vontuk állandóan felelősségre. Válaszaival nagyban hozzájárult ahhoz, hogy könyvünk hasznosabb, informatívabb legyen.

A könyv írását részben az MKM 419. és 3306. számú pályázatának keretében végeztük.

Pelikán József és Ván Péter lektorként matematikai, nyelvi, számítástechnikai és alkalmazási szempontokat egyaránt figyelembe véve, rendkívül alaposan végezte munkáját. Javaslataikat és megjegyzéseiket igyekeztünk az Olvasó javára hasznosítani.

Végül pedig megkérjük a tisztelt Olvasót, ha bármilyen hibát talál, vagy bármilyen hasznosítható megjegyzése van, juttassa el azt a szerzőkhöz vagy a kiadóhoz.

Budapest, 1996. május

Szili László

ELTE TTK
Numerikus Analízis Tanszék
Budapest, Múzeum krt 6–8.
1088
szili@ludens.elte.hu

Tóth János

GATE MGK
Számítástechnikai Tanszék
Gödöllő, Páter K. u. 1.
2103
tothj@mszi.gau.hu

1. Matematikai programcsomagok

Már az ókorban megjelentek az emberi tevékenységnek olyan területei, amelyek különösen sok számolást igényeltek. A számolások elvégzésének megkönnyítésére több módszer kínálkozott. Egyrészt *táblázatok*ban rögzítették a gyakran használt részeredményeket (Babilóniától Napierig); másrészt (cél- és univerzális) mechanikus, később elektromos, majd elektronikus *számológép*eket készítettek (az abakusztól a logarlécen át az IBM PC-ig); végül pedig — különösen az alkalmazott matematika legnagyobbjai (Arkhimédésztől Newtonon át Eulerig) — olyan *eljárás*okat dolgoztak ki és alkalmaztak, amelyek pontosabb eredményekhez kevesebb számolási munkával vezettek el.

Nem meglepő tehát, hogy a számítástechnika fejlődésének korai szakaszában (körülbelül 1950-ig) a matematikusok (gondoljunk például A. M. Turingra, Neumann Jánosra, vagy éppen Kalmár Lászlóra) jelentős szerepet játszottak. Az ötvenes évektől meginduló fejlődés azonban — a be nem váltott ígéretek, a tervezetlenség, a felhasználók (köztük a matematikusok) igényeinek figyelmen kívül hagyása és mindenekelőtt az alacsony teljesítőképesség — egyre inkább eltávolította a matematikusok táborát a számítógéptől. Kivételt talán csak a statisztikusok, a diszkrét matematikával foglalkozók egy része és a numerikus matematikusok egy része (!) jelentett. (Ennek az időszaknak a számítástechnikusai, persze, a matematikusok rovására írják a szakadék kialakulását.) Numerikus számolások végzésére ebben az időszakban is számos vegyész, fizikus vagy közgazdász alkalmazta a számítógépet.

A hatvanas-hetvenes években felvetődött, tipikus probléma volt az, hogy hogyan lehet a számítógép segítségével egy függvény deriváltját vagy valamelyik primitív függvényét a függvényt megadó képletből *szimbolikusan* kiszámítani. Erre (a lengyel forma felhasználásával) több szerző is adott megoldást. (Megjegyzendő, hogy az eredményt például az érzékenységi egyenletek felírásához, tehát *számolási* célra használták.) Ebben az időszakban kezdtek foglalkozni *tételbizonyító, tolmács- és sakkprogramok* írásával és *logikai programozási nyelvek* megalkotásával is, hogy csak néhányat említsünk a számítástechnika nem-numerikus alkalmazásai közül.

A számítógépek sebessége és az elméleti eredmények a nyolcvanas évek elejétől fogva lehetővé tették azt, hogy matematikai feladatokat megoldó *numerikus programcsomagok* mellett *szimbolikus programcsomagok* (így fogjuk nevezni a továbbiakban azokat a programrendszereket, amelyeket angolul *computer algebra systems* néven említenek, s korábban *formulamanipulációs rendszerek*nek hívtak) is létrejöjjenek. A technika rohamos fejlődésével párhuzamosan örvendetes módon a matematikai programok gomba módra szaporodnak.

Itt hívjuk fel a figyelmet arra, hogy a *Notices of the American Mathematical Society* című folyóirat 1994. decemberig rendszeresen közölt ismertetőket, illetve összehasonlításokat az ilyen jellegű programokról. A 40. kötet 6. száma (1993. július–augusztus) például mintegy 50 ilyen program néhány fontos adatát tartalmazza. Az ott ismertetett adatbázis egyébként a

```
math.berkeley.edu
```

számítógép `pub/Symbolic_Soft` könyvtárában megtalálható. Ide bejelentkezni az *anonymous* jelszóval lehet. Az 1995. februári szám pedig közli az e témában megjelent cikkek indexét.

Matematikai programcsomagokról az

```
ftp.can.nl
```

számítógépről is szerezhetünk információkat. Erre a gépre is az *anonymous* jelszóval jelentkezhetünk be.

Megjegyezzük még azt is, hogy több ilyen program kereskedelmi forgalomban kapható. Jónéhány közülük azonban ingyenes (*public domain*) program, és ezek egy része például az utóbb megemlített holland gépről is megszerezhető.

1.1. Szimbolikus programcsomagok

A matematika alkalmazásánál vagy kutatásánál, sőt az oktatása során is gyakran kell rutinjellegű szimbolikus műveleteket végezni. Gondoljunk a polinomokkal végzett különféle műveletekre (például a polinomfaktorizációra), egyenletek és egyenletrendszerek megoldására vagy függvények primitív függvényeinek meghatározására. A szimbolikus programcsomagok egyik célja éppen az, hogy az ilyen típusú feladatok megoldásához is segítségül

ajánlja a számítógépet. A lehetőségek ilyen kibővítése *minőségi változást* hozott a matematika és a számítógép kapcsolatába.

A költő Byron lánya, Lady Ada Lovelace volt az első, aki 1843-ban fölvetette, hogy szimbolikus számításokat géppel kellene végezni. Az ötlet megvalósítása azonban több mint száz évet váratott magára, mivel ennyi idő kellett ahhoz, hogy az elmélet és a technikai eszközök egyaránt megfelelő fejlettségi szintet érjenek el.

Szóljunk néhány szót arról, hogy szerintünk miért érdemes használni a matematikai programcsomagokat. Elsősorban azért, mivel ezek nagyon sok beépített függvényt tartalmaznak, amelyek összetett feladatok (például polinomok faktorizációja, függvények ábrázolása, deriváltfüggvény meghatározása, primitív függvények keresése, görbeillesztés) megoldására szolgálnak, ezért könnyen és gyorsan végezhetünk el velük rutinjellegű szimbolikus és numerikus számításokat.

E programcsomagok egyik fő erénye az, hogy *programozhatók*, azaz a beépített utasítások bizonyos nyelvtani szabályok betartásával összefűzhetők, így a lehetőségeik a felhasználó igényeinek megfelelően folyamatosan bővíthetők.

A rendszerek többsége összekapcsolható más felhasználói programokkal és programozási nyelvekkel.

Említsünk még néhány olyan tipikus helyzetet, amikor segítségünkre lehetnek a matematikai programcsomagok. Például *sejtések kialakításánál* sok esetet vizsgálhatunk meg. Bizonyításukhoz vagy cáfolatukhoz is felhasználhatjuk a meglévő eszközöket. Egy-egy *esettanulmányt* sokkal kevesebb energiával készíthetünk el, mint hagyományos módszerekkel. Az oktatásban a fentieken kívül is gyümölcsözően használhatjuk e rendszereket, például fogalmak kialakításánál és eredmények illusztrálásánál.

Igen sikeresen alkalmazták már ezeket a rendszereket olyan területeken, ahol numerikus és szimbolikus számításokra egymás mellett van szükség. Galaxistérképeket készítettek ilyen módon; de a legnagyobb valószínűség elve alapján végzett becslés — amit a 3.11. szakaszban bemutatunk — szintén ide sorolható.

A szimbolikus programcsomagok alkalmazóinak egyre növekvő száma is azt igazolja, hogy sokan megtalálták használatuk módját.

Már itt is felhívjuk a figyelmet azonban arra, hogy rengeteg lehetőséget tartalmazó, nagy rendszerekről van szó. Megismerésük és *értelmes* felhasználásuk módjának megtalálása ezért időigényes feladatot jelenthet a felhasználó számára.

Mindenesetre a szerzők véleménye az, hogy ezeknek a matematikai programoknak a megjelenése új távlatokat ad a matematika alkalmazásának, kutatásának és oktatásának.

Az első, széles körben használható változatok a 80-as évek végén jelentek meg. Azóta a cégek egymást serkentve bocsájtják ki az újabb és újabb, javított és bővített változatokat. A következő évek fejlődésének következményei szinte beláthatatlanok.

A szimbolikus programcsomagokban is alkalmazott algoritmusok matematikai megalapozásával és gyártásával a *szimbolikus számolások elmélete* (*computer algebra*) foglalkozik. Az e témakör iránt is érdeklődő Olvasó figyelmébe ajánljuk K. O. Geddes, S. R. Czapor és G. Labah kiváló könyvét (lásd [28]-at). Ebben a szerzők tematikus feldolgozásban tárgyalják a felhasználható algoritmusokat és azok elméleti alapjait. Részletesen foglalkoznak például a különböző polinomfaktorizációs eljárásokkal. Kifejtik a Gröbner-bázisok elméletét (ez az alapja az egyenletrendszer-megoldó algoritmusoknak) is. Külön fejezetet szentelnek a határozatlan integrálok kiszámítására használható Risch-algoritmusnak is.

A szimbolikus programcsomagokat a készítőik által kitűzött cél alapján szokás osztályozni.

• Általános célú programcsomagok

Ebbe a csoportba sorolhatók azok a programok, amelyek a matematika több területén felvetődő problémák megoldásához is használhatók. Ilyen programok a következők:

Axiom,	*Mathematica*,
Derive,	muMath,
Macsyma,	Reduce,
Maple,	ScratchPad.

Az általános célú matematikai programcsomagok közül hazánkban a Derive, a Maple és a *Mathematica* a legismertebb.

A Derive a programozás csúcsteljesítménye. Ilyen kis terjedelemben (alig 400 kB) ennyi képességet létrehozni — minden elismerésünk a fejlesztőké. Magyarországon, különösen általános és középiskolákban, sokáig még biztosan fontos szerepet fog játszani, részben mint olyan eszköz, amely előkészíti az érdeklődőket a professzionális programok, például a Maple vagy a *Mathematica* használatára. Lásd erről például a [79] könyvet, amely a tanároknak ad az alkalmazásához segítséget.

A Maple és a *Mathematica* méreteit és lehetőségeit tekintve lényegesen nagyobb, mint a Derive. Különböznek egymástól, de az alaplehetőségeket tekintve nagyon hasonlóak (a *Mathematica* lehetőségeit a 2.1. szakaszban ismertetjük). Mivel sok beépített eljárást tartalmaznak (a *Mathematica*-függvények száma közel 2000), ezért már igen rövid idő alatt elérhetünk

sikereket egysoros utasításaik felhasználásával is. A tapasztalatunk azonban az, hogy a lehetőségek igényeinknek megfelelő kihasználásához a programok mélyebb megismerése elengedhetetlen.

• Speciális célú programcsomagok

Ebbe a csoportba tartoznak azok a programok, amelyek segítségével a matematika egy adott részterületén felvetődő, már speciális kutatói igényeket kielégítő számolások is elvégezhetők.

Ilyen program például a CAYLEY (csoportelmélet és számelmélet), a CoCoA (kommutatív algebra), a GAP (csoportelmélet), a LiE (Lie-csoportok elmélete) vagy a Macaulay (algebrai geometria és kommutatív algebra). Kifejezések egyszerűsíthetők a FORM segítségével, a SHEEP pedig a relativitáselméletben használható.

1.2. Numerikus programcsomagok

A legnagyobb hagyománya azoknak a programcsomagoknak van, amelyek a numerikus matematika módszereivel keresik az adott probléma megoldásának egy közelítését. (Angolul az ilyeneket szokás *number cruncher*nek nevezni.)

A teljesség igénye nélkül ezek közül is fölsorolunk néhányat.

A legelterjedtebb és legteljesebb ilyen programcsomag a *Numerical Recipes* [63]. A számos kiadásban is napvilágot látott, több kötetes mű a különböző numerikus algoritmusok C, Fortran és Pascal nyelvű programjait tartalmazza.

Sokan használják a MATLAB-ot, amelyet az igényeknek megfelelően kiegészítettek szimbolikus számolásokat végző részekkel is.

Közönséges differenciálegyenletek numerikus megoldására (különösen az oktatásban) nagyon jól használható például a PHASER.

Érdemes megemlíteni a modellvizsgálat fontos speciális területéről néhány hasznos programot; ilyen a DiffEq az MLAB, a PCNonLin és a KINAL.

Különálló, fontos terület a statisztikai programcsomagoké. Néhány igen népszerű programot említünk csak: SPSS, STATGRAPHICS, SYSTAT, BMDP. (Helytelenül ide szokták sorolni a SAS-t is.)

A lineáris programozás feladatára is sok programcsomag készült. Végül megemlítjük, hogy ma már a táblázatkezelők (EXCEL, Quattro) is egyre több numerikus feladat (például egyenletek megoldása, függvények mi-

nimumának keresése) megoldására képesek; ezeket igazán akkor érdemes használni, amikor az adatok és paraméterek táblázatos ábrázolása jelentős előnyökkel jár, például a lineáris és nemlineáris programozás feladatainál.

A feladatokat nem lehet diszjunkt osztályokra bontani. A szimbolikus számolás képességének akkor is szerepe lehet, amikor eredeti célunk egy numerikus számolás elvégzése. Tekintsük például a 30×30-as Hilbert-féle mátrix inverzének numerikus kiszámolását. Ez hagyományos nyelven az elérhető gépi pontosság mellett nem végezhető el. Ha viszont szimbolikusan számoljuk ki az inverzet (ami egy racionális számokból álló mátrix lesz), majd az eredményt közelítjük, akkor megfelelő eredményt kapunk.

2. Ismerkedés a *Mathematica* programmal

A *Mathematica* egy általános célú szimbolikus programcsomag. Kifejlesztésének ötlete a részecskefizikusként, majd a sejtautomaták elméletének kutatójaként ismert Stephen Wolfram fejéből pattant ki — a legmegfelelőbb pillanatban: 1986-ban. Ekkor már az optimisták remélhették, hogy belátható időn belül a „mindenki" asztalán ott lévő személyi számítógépek képesek lesznek nem-triviális matematikai problémák megoldására.

Ma már a *Wolfram Research Inc.* több mint 100 alkalmazottal dolgozik az Egyesült Államokban, s a világ számos országában van képviselete. Négy folyóirat és mintegy 100 könyv tárgya a program. Különösebb előkészületek, illetve kiegészítések nélkül a következő lehetőségeket kínálja:

- *Szimbolikus műveletek* végzésére használható, amilyen például a matematikai kifejezések egyszerűsítése, primitív függvények keresése, algebrai és differenciálegyenletek megoldása.
- A hagyományos programnyelvekkel szemben itt akár több ezer értékes jeggyel is végezhetünk *numerikus számításokat*.
- *Grafikai lehetőségei* egészen kiválóak: alapértelmezésben is igen sok esetben megfelelő ábrát készít, szükség esetén azonban az ábrázolás folyamatát rugalmasan módosíthatjuk.
- *Magas szintű programnyelv*, amely több stílusban is lehetővé teszi a programozást: írhatunk *procedurális*, *szabályalapú*, *objektumorientált* és *funkcionális* elven készült programokat. Különösen fontos az a tény, hogy gazdag az utolsóként megemlített stílus megvalósításához használható eszközeinek tára.
- Tudományos és műszaki *tudásbázis*ok reprezentálására is alkalmas.
- Írhatunk *programok*at alá (alkalmazásokat) és fölé (azaz alprogramként használhatjuk függvényeit).
- Egyre több népszerű program (C, EXCEL, VisualBasic, Word) kapcsolható hozzá.
- Olyan *interaktív dokumentum*ok készíthetők vele, amelyekben matematikai képletek, hang, álló és mozgó ábrák (*animáció*) együtt szolgálják az oktatandó vagy kutatott anyag kifejtését.

A *Mathematica* program (Apple Macintosh gépre kidolgozott) első változatát 1988 júniusában mutatták be. A megjelenése évében a Business Week magazin minden kategóriát tekintve az év legjobb 10 új terméke közé sorolta. Ezután sorra készültek el a különféle operációs rendszerekre (például DOS, LINUX, NEXTSTEP, OS/2, UNIX és WINDOWS) adaptált változatok.

A beépített eljárások száma közel kétezer. A program beindítása után ezek közül több mint ezer azonnal használható. Ezeknek az utasításoknak a lelőhelye a

mag (*kernel*).

A fennmaradó eljárásokat a *Mathematica* programnyelvén megírt állományok, az úgynevezett

programcsomagok

(lásd a 2.2. szakaszt) tartalmazzák. A felhasználóval egy külön program tartja a kapcsolatot:

a felhasználói felület (*front end*).

Ez alapvetően kétféle típusú lehet:

szöveges vagy *ablakos*.

Általában ezeket különböző paranccsal indíthatjuk. Az ablakos változatban menük vannak, így igen kényelmesen használható, ezért a tanulás időszakában, ha van rá mód, ezt érdemes választani. A szöveges változat kevesebb memóriát igényel és gyorsabb, aminek előnyeit különösen számolásigényes feladatoknál tapasztalhatjuk.

A program a felhasználó szempontjából minden géptípuson lényegében ugyanúgy működik.

Az ablakos változatok *jegyzetfüzeteket* (*notebook*) készítenek. Ezek `.ma` kiterjesztésű állományok, amelyekkel a szokásos műveletek (például megnyitás, mentés, módosítás) végezhetők. A jegyzetfüzetek *cellákból* épülnek fel. A cellák közül *kijelölhetünk* egyet (a képernyő jobb oldalán látható szögletes zárójelre kattintva) vagy többet (az egeret egy kattintás után a zárójeleken végigvonszolva). A kijelölt cellával vagy cellákkal például a következő *műveleteket* végezhetjük a megfelelő menüpontok fölhasználásával:

törlés, áthelyezés, stílusváltoztatás, összefűzés, csoportosítás. (A csoportosítás eredménye egy olyan cella, amelyben több cella van együtt.) A több cellából álló cellákat *becsukhatjuk* a szögletes zárójelre duplán kattintva, ezáltal fokozhatjuk a jegyzetfüzet áttekinthetőségét. Amikor kíváncsiak vagyunk a becsukott cella részletes tartalmára, akkor azt (ismét dupla kattintással) újra *kinyithatjuk*. A jegyzetfüzetek, sőt az egyes cellák *stílusa* tág határok között változtatható. A bennük található anyagot igényeinknek megfelelően fejezetekbe, szakaszokba stb. szervezhetjük. Egyes részeket az áttekinthetőség érdekében időnként becsukhatunk.

A programmal megoldható feladatok mérete és a megoldáshoz szükséges idő lényegesen függ a rendelkezésre álló számítógép konfigurációjától. Magunk elsősorban a 2.2.2. és a 2.2.3. változatot használtuk WINDOWS alatt 486-os IBM PC típusú gépeken, amelyekben 8–16 Mb RAM volt. A program készítői ezekhez a változatokhoz legalább 8 Mb RAM-ot és legalább 10 Mb virtuális memóriát javasolnak. Maga a program mintegy 10 Mb helyet foglal el a merevlemezen. A helyenként megjelenő időadatok csupán tájékoztató jellegűek; értékelésüknél a fentieket figyelembe kell venni.

2.1. A legfontosabb tudnivalók

A *Mathematica értelmezőként* (*interpreter*) dolgozik, ami azt jelenti, hogy a parancsokat soronként megérti, végrehajtja, majd kiírja az eredményt. Számon tartja és kiírja azt is, hogy az adott alkalomkor (*session*) az adott utasítást hányadikként hajtotta végre. Stílusában érezhető, hogy felhasználták a C, a LISP és a UNIX nyelv tapasztalatait.

• A program működtetése

A program *indítása* rendszerfüggő, és könnyen kideríthető. Indítás után a *végrehajtani* kívánt utasítást a megfelelő szintaxissal begépeljük (a sorok bárhol megtörhetők), majd végrehajtjuk. Ez utóbbit szöveges változatnál az *Enter* billentyű lenyomásával érhetjük el. Az ablakos változatoknál általában a *Shift* és az *Enter* billentyű együttes lenyomására van szükség. A WINDOWS-os változatban alkalmazhatjuk az *Insert* gombot vagy a numerikus billentyűzet *5*-ös gombjának lenyomását is.

A program minden változatából *kilépni* a

```
Quit[ ]
```

utasítással lehet.

Felhívjuk a figyelmet arra, hogy az ablakos változatokban az egyes műveletek kiértékelése közben is begépelhetjük a következő utasításokat.

• Karakterkészlet

Az angol ábécéből és a számjegyekből indulunk ki. A nagy- és a kisbetűk között különbséget teszünk.

A *speciális karakterek*nek meghatározott jelentése van. Itt csak a leggyakrabban használtakat soroljuk fel.

Az *aritmetikai műveletek* jelei általában a szokásosak (lásd a 3.1.3. pontot); például a szorzást a `*` jelöli, de ugyanerre (kissé szokatlan módon) a szóközt is használhatjuk.

Vigyázzunk a különféle *zárójelek* gondos írására. A precedencia-szabály megtörésére használjuk a közönséges `( )` jelpárt. Függvények argumentumát szögletes zárójelek `[ ]` közé tesszük. Kapcsos zárójelekkel `{ }` listákat (lásd a 2.3.1. pontot) adhatunk meg. A lista részeire kettős szögletes zárójellel `[[ ]]` hivatkozhatunk (lásd a 2.3.1. pontot). Végül, ha programot írunk, a `(*` és a `*)` jel közé magyarázó szöveget írhatunk.

Az *egyenlőségjelek* közül az `=` jel *azonnali értékadásra*, a `:=` pedig *késleltetett értékadásra* használható. Ez utóbbi értékadás csak akkor valósul meg, amikor használni akarjuk a bal oldalán szereplő szimbólumot. A `===` és a `=!=` jelsorozattal két kifejezés karakterről karakterre való megegyezését ellenőrizhetjük, a `==` és a `!=` jelsorozattal pedig kifejezések egyenlőségét matematikai szempontból vizsgálhatjuk:

```
{alma === almas, alma =!= almas, alma == almas}
{False, True, alma == almas}

x = 3^2;
{x == 8, x == 9}
{False, True}

1 + 2x + x^2 == (1+x)^2
True
```

Itt is felhívjuk a figyelmet arra, hogy egyenleteket is a `==` jelsorozattal adhatunk meg. A ciklusváltozók (*iterátorok*) léptetésére a `+=` típusú jelek szolgálnak.

A `,` jelet felsorolásokban használhatjuk. Ha egy utasítás után a `;` jelet írjuk, akkor annak eredménye általában nem jelenik meg a képernyőn.

A legutolsó utasítás eredményére a `%` jellel hivatkozhatunk. Az utolsó előttinek a jele `%%`, míg az `n`-edik utasítás eredményének a jele `%n`. Ezek alkalmazásánál könnyű hibát elkövetni!

Az `n` szám faktoriálisa jelölhető a szokásos `n!` módon. (Szemifaktoriálisa pedig így: `n!!`.) Ha a `!` jel viszont egy logikai kifejezés előtt áll, akkor annak tagadását jelöli.

• Alapszavak

A nyelv tervezésénél (a Pascal nyelvben vagy a UNIX operációs rendszernél megszokotthoz hasonlóan) általában hosszú, értelmes, nagybetűvel kezdődő, kitalálható *azonosító*kat választottak *alapszavak*ként. Például:

```
InterpolatingPolynomial    ParametricPlot3D
Infinity                   PseudoInverse
```

Szükség esetén a meglévőkön túl magunk is bevezethetünk azonosítókat. Ilyenkor célszerű követni a fenti konvenciót. A programtól figyelmeztető üzenetet kapunk, ha az új azonosító valamelyik korábban definiálttal megegyezik, vagy egy olyanhoz hasonló.

A mintegy kétezer beépített azonosító nagy része az angol matematikai és számítástechnikai szaknyelv elemeinek ismeretében vagy kitalálható, vagy legalább annyira megközelíthető, hogy értelmes kérdést tudjunk föltenni a rendszernek. (Például: `?Plot*`.)

Az alapszavak első megközelítésben matematikai és programozási szempontból is függvények, ezért ezeket *Mathematica-függvényeknek*, *beépített függvényeknek* vagy egyszerűen *függvényeknek* fogjuk nevezni. A változatosság kedvéért viszont *eljárás* néven is fogjuk említeni ezeket, némileg eltérve a számítástechnikában meghonosodott szokástól.

A program magjában vannak a *belső függvények*, a programcsomagok tartalmazzák a *külső függvények*et. A függvények jelentése sokféle lehet:

- matematikai állandó (`E, Pi, Degree`);
- adattípus (`List, Integer`);
- matematikai függvény (`ArcTan, Cot, GCD`);
- matematikai operátor (`Sum, D, Map`);
- összetett matematikai eljárás (`NDSolve, MatrixExp, JordanDecomposition`);
- rajzoló eljárás (`Plot3D`);
- be- és kiviteli művelet (`ReadList, Input`);
- programozási eszköz (`If, Do, Nest`);
- listakezelő eljárás (`Sort, Part, Take, Reverse`);
- egyéb (`Sound, Display`).

Mivel az alapszavak jelentése igen sok esetben magától értetődő, ezért egy *Mathematicá*ban megírt program legalább olyan egyszerűen olvasható, mint egy szépen megírt Pascal-program — azzal a különbséggel, hogy működtetéséhez nem nekünk kell kibontanunk a főprogramban szereplő értelmes szavakat eljárások és függvények hosszú sorává.

A *Mathematica* felépítése a hardver- és a szoftverkörnyezettől meglehetősen független. Az aktuális kapcsolatokat a *rendszerváltozók* írják le (lásd a 2.4.2. pontot), amelyek azonosítója a `$` jellel és nagybetűvel kezdődik. Például a

```
$StringOrder
{{a, A}, {b, B}, {c, C}, {d, D}, {e, E}, {f, F},
 {g, G}, {h, H}, {i, I}, {j, J}, {k, K}, {l, L},
 {m, M}, {n, N}, {o, O}, {p, P}, {q, Q}, {r, R},
 {s, S}, {t, T}, {u, U}, {v, V}, {w, W}, {x, X},
 {y, Y}, {z, Z}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

rendszerváltozó definiálja a karakterek sorrendjét. A program ezt veszi alapul kifejezések feldolgozásánál is és megjelenítésénél is.

• Információszerzés

A beépített függvények használatához a programtól is kaphatunk segítséget. Tetszőleges azonosítóról felvilágosítást kaphatunk az alábbi beépített függvények használatával:

```
Context                 Information (??)
Contexts                NameQ
Definition              Names (?)
FileNames               Unique
FullDefinition          ValueQ
```

Ha ablakos felhasználói felülettel dolgozunk, akkor a *Help* menüponthoz is fordulhatunk.

Ha tudjuk annak a függvénynek a nevét (legyen ez például `Plot`), amelyről információt szeretnénk kapni, akkor hajtsuk végre a következő utasítást:

```
?Plot
Plot[f, {x, xmin, xmax}] generates a plot of f
     as a function of x from xmin to xmax.
Plot[{f1, f2, ...}, {x, xmin, xmax}] plots
     several functions fi.
```

Figyeljük meg a konkrét példában, hogy a `Plot` függvény különböző szintaxissal más-más feladat megoldására használható.

Részletesebb, az opciókat és attribútumokat (lásd a 2.3.3. és a 2.3.4. pontot) is megadó információt kapunk a

```
??Plot
```

utasítás eredményeként. Ha a keresett függvény nevének csak egy részét ismerjük, akkor a `*` dzsóker karaktert használhatjuk:

```
?*Pl*
```

Általában érdemes a programot is megkérdezni és a *Help* menüpontot is megnézni egy adott függvény működéséről, mert esetleg egymást kiegészítő információhoz juthatunk. Ha a fenti módon megszerzett ismeret nem elegendő, akkor a program referenciakönyveit [3, 89] használhatjuk.

Ennek a fejezetnek további részei (lásd a 2.5. szakaszt) is tartalmaznak még idevonatkozó tanácsokat.

• Üzenetek

A program működése közben üzeneteket küld a felhasználónak. *Hibaüzenetet* kapunk akkor, ha nyilvánvaló szintaktikai hibát követünk el. (Például egy kezdő zárójel párja hiányzik.) *Figyelmeztető üzenet* jelenik meg a képernyőn, ha egy függvény argumentumának a száma vagy típusa nincs összhangban annak definíciójával. A kiértékelés során *tájékoztatást* is kaphatunk például arról, hogy a *Mathematica* által alkalmazott módszer nem garantálja a teljes eredmény előállítását. Bármilyen üzenetet is kapunk (ha akarjuk, ezeket le is tilthatjuk), várjuk meg az aktuális utasítás végrehajtását, és az eredményt is figyelembe véve hasznosítsuk tartalmát.

Saját programjainkat is elláthatjuk a fenti típusú üzenetekkel.

2.2. Programcsomagok

Említettük már azt, hogy a program elindítása után több mint ezer beépített függvény közvetlen használatára van lehetőségünk. Számos feladatot már egy ilyen függvénnyel is meg tudunk oldani.

A *Mathematica* minden változata tartalmaz azonban saját programozási nyelvén megírt eljárásgyűjteményeket (ezek `.m` kiterjesztésű szöveges állományok), amelyeket a továbbiakban *(program)csomagok*nak fogunk nevezni.

A program készítői a csomagokat osztályokba sorolták. A különböző sorszámú változatoknál eltérő lehet a csomagok száma is és a csoportosítás módja is. Bizonyos operációs rendszerek (például a DOS és a UNIX) alatt működő változat esetében az osztályba sorolást „könyvtárszerkezettel" lehet megadni. Mi is az ennek megfelelő szóhasználattal élünk annak ellenére, hogy más operációs rendszerek esetében más elnevezést szokás használni.

A *Mathematica* a csomagokhoz két paramétert (ezek az azonosítókhoz hasonlóan értelmes, angol szavak) rendel: a csomag nevét és egy alkönyvtár nevét. Ennek eredménye, hogy minden operációs rendszer használata esetén ugyanazzal a szintaxissal nyithatunk meg egy adott programcsomagot. A 2.2.x változatokban az alábbi alkönyvtárnevek közül válogathatunk:

```
Algebra                 Miscellaneous
Calculus                NumberTheory
DiscreteMath            NumericalMath
Examples                ProgrammingExamples
Geometry                Statistics
Graphics                Utilities
LinearAlgebra
```

Ezek a DOS és a UNIX operációs rendszer esetén a *Mathematicát* tartalmazó könyvtár *Packages* elnevezésű alkönyvtárában találhatók meg. (A DOS-ban az alkönyvtárak és az állományok neve legfeljebb 8 karakterből áll.)

Ha használni akarjuk valamelyik programcsomagban meglévő eljárást, akkor először meg kell nyitni a megfelelő csomagot. Ennek egyik (az aktuálisan használt operációs rendszertől *független*) módja az

```
<<alkonyvtar`programcsomag`
```

szerkezetű utasítás végrehajtása. A `<<` jelsorozat (a `Get` függvény rövid alakja) utal arra, hogy meg akarunk nyitni egy állományt. Az `alkonyvtar` jelsorozat az imént felsorolt szavak valamelyikét, a `programcsomag` pedig a megnyitandó csomag (*teljes*) nevét jelöli. Figyeljük meg, hogy az utóbbit a `` ` `` karakterek (*context mark*) közé kell tennünk. Használhatjuk még a

```
Needs["alkonyvtar`programcsomag`"]
```

szerkezetű utasítást is. Az állományok megnyitásának fenti két módja között a különbség az, hogy az első esetben a *Mathematica* azonnal beolvassa a memóriába a csomag tartalmát, míg a második esetben ezt csak akkor teszi meg, ha olyan utasítást kell neki végrehajtania, amely a megjelölt

programcsomagban található. A harmadik lehetőségünk az, hogy a << jelsorozat után beírjuk (az elérési útvonal megadásával) a programcsomagot tartalmazó állomány (adott operációs rendszeren használt) nevét.

Programcsomagokra a könyvünk további részében így fogunk hivatkozni:

```
alkonyvtar`programcsomag`
```

Az elmondottak illusztrálására nézzünk egy példát. Hajtsuk végre a

```
<<Miscellaneous`Calendar`
```

utasítást, azaz olvassuk be a `Calendar` nevű programcsomagot, amely a `Miscellaneous` alkönyvtárban található. Ezután a belső függvényekkel egyenértékűen működő alábbi külső függvényeket is használhatjuk:

```
DayOfWeak           EasterSunday
DaysBetween         EasterSundayGreekOrthodox
DayPlus             JewishNewYear
CalendarChange
```

Ha kíváncsiak vagyunk például arra, hogy 1995. december 9. milyen napra esett, akkor ezt így kérdezhetjük meg a programtól:

```
DayOfWeek[{1995, 12, 9}]
Saturday
```

Az értelmes alapszavak előnyét itt is tapasztalhatjuk: a felsorolt függvények sejteni engedik azt, hogy mire lehet őket használni.

Az alkönyvtárakban lévő csomagok, valamint az ezekben definiált eljárások nevéről magától a programtól a WINDOWS-os változat esetén nem kapunk információt. Ebből a szempontból a UNIX-os változat is kevés segítséget tartalmaz. A szükséges ismereteket azért többféle módon is megszerezhetjük. Mivel a programcsomagok szöveges állományok, ezért egyrészt ezek elolvasásával is tájékozódhatunk a felhasználható függvényekről. Ha már tudjuk a bennünket érdeklő függvény nevét és beolvastuk a programcsomagot, akkor alkalmazhatjuk az információszerzés kérdőjeles módját. Sokszor ennél részletesebb információt és több mintapéldát is tartalmaz a [3] referenciakönyv. Végül könyvünk további fejezeteiben mi is utalni fogunk a programcsomagokban meglévő lehetőségekre is.

Most a programcsomagok használatával kapcsolatban egy tipikus hibára szeretnénk felhívni az Olvasó figyelmét. Előfordul, hogy tévedésből használni akarjuk valamelyik külső függvényt az őt tartalmazó programcsomag behívása előtt. Ekkor a *Mathematica* létrehoz egy új szimbólumot, amit a beadott sor változatlan formában való visszaadásával jelez. Például:

```
MultipleListPlot[{1, 2, 3, 4}, {2, 4, 6, 8}]
MultipleListPlot[{1, 2, 3, 4}, {2, 4, 6, 8}]
```

Ilyen nevű függvény a `Graphics`MultipleListPlot`` programcsomagban is megtalálható. Olvassuk be ezután ezt a programcsomagot:

```
Needs["Graphics`MultipleListPlot`"]
MultipleListPlot::shdw:
     Warning: Symbol MultipleListPlot
     appears in multiple contexts
     {Graphics`MultipleListPlot`, Global`}; definitions
     in context Graphics`MultipleListPlot`
     may shadow or be shadowed by other definitions.

MultipleListPlot[{1, 2, 3, 4}, {2, 4, 6, 8}]
MultipleListPlot[{1, 2, 3, 4}, {2, 4, 6, 8}]
```

A program tehát figyelmeztet bennünket arra, hogy több ilyen nevű utasítást is talált, és ezek közül az utolsóként létrehozott utasítást (tehát azt, amelyiket tévedésből adtunk meg) hajtotta végre. Ilyen esetekben a

`Remove`

eljárással törölhetjük a tévedésből begépelt azonosítót. A fenti példát folytatva a

```
Remove[MultipleListPlot]
MultipleListPlot[{1, 2, 3, 4}, {2, 4, 6, 8}]
```

utasítássorozat már megadja a várt eredményt.

A programcsomagokat tartalmazó alkönyvtárak mindegyikében megtalálható egy

`Master`

nevű programcsomag is. Ha ezt behívjuk, akkor ezzel az illető *alkönyvtárban* meglévő valamennyi programcsomag minden függvénye elérhetővé válik.

Sokszor szükségünk lehet arra, hogy a memóriából egy beolvasott programcsomagot eltávolítsunk. Ehhez (is) használható a `CleanSlate` programcsomag, amely a MathSource-on a 0204–310 szám alatt található.

Ha olyan feladatot akarunk megoldani a *Mathematicá*val, amelyhez nem elegendő belső vagy külső függvény közvetlen használata, akkor magunk is írhatunk új eljárásokat. Erre a könyv további fejezeteiben számos példát

fogunk mutatni. A MathSource (lásd a 2.5. szakaszt) igen sok programcsomagot is tartalmaz. Szükség esetén ott is érdemes körülnézni.

Léteznek speciális igényeket kielégítő, pénzért árult csomagok is. Ilyenek például a következők: CALCULUS&*Mathematica*, Electrical Engineering Pack, Finance Pack, MathTensor, Mechanical System Pack, Optica, Time Series Pack, TSiDynamics, TSiControls. A MathSource-on megtudhatjuk, hogy ezek hol és mennyiért szerezhetők be.

2.3. Az alapelvekről

A *Mathematica* leglényegesebb jellegzetessége az, hogy *egységes módon* kezeli a (matematikai) műveletek három fő csoportját: a grafikus, a numerikus és a szimbolikus műveleteket. A különböző típusú dolgok kidolgozott egységes kezelési módja teszi lehetővé azt, hogy a matematika és a számítástudomány aránylag kevés módszerének felhasználásával az alkalmazások igen széles köre lefedhető. Ennek az egységes szemléletnek az előnyeit a felhasználó is tapasztalja.

A program szempontjából a legalapvetőbb fogalmak a *kifejezés*, a *kiértékelés*, a *transzformációs szabály*, a *mintázat*, az *opció* és az *attribútum*. A *Mathematica működését* első közelítésben a következőképpen lehet leírni. Amikor végrehajtunk egy utasítást vagy utasítássorozatot (ezeket nevezzük a továbbiakban kifejezéseknek), akkor a program kiértékeli azt. Programozástechnikai szempontból a *Mathematica* egy *végtelen kiértékelő rendszer*. Ez a következőt jelenti. A kiértékelés bonyolult folyamatának első lépéseként a program egységes (belső) alakra hozza a beadott kifejezést. Ezután az attribútumok és az opciók figyelembevételével a *Mathematica* addig alkalmazza a készítőik vagy a felhasználó által már korábban megadott definícókat (transzformációs szabályokat), ameddig különböző alakú eredményt kap. A program egy eredményt akkor tekint végeredménynek (és általában ezt írja ki a képernyőre), ha az már változatlan alakú.

2.3.1. Alapvető adatszerkezetek

A legalapvetőbb adatszerkezet a

kifejezés.

Így fogjuk nevezni az egy vagy több (parancs)sorba begépelt jelsorozatot. A beadott parancsot a végrehajtás után a *Mathematica* kiértékeli és a végeredményt egyrészt tárolja, másrészt (számos esetben annak csupán egy részét) kiírja a képernyőre. Ezeket a jelsorozatokat is kifejezéseknek tekintjük.

A 2.1. szakaszban beszéltünk a kifejezések egyik speciális osztályáról, a

Mathematica-függvényekről.

Sokszor szükségünk lehet arra, hogy a program által megadott végeredményt vagy annak bizonyos részeit későbbi időpontban használjuk fel. Ilyen esetekben fontos ismerni azt, hogy a *Mathematica* milyen módon *tárolja* a kifejezéseket. Ennek megértéséhez a kifejezés fogalmát kell pontosítanunk.

A programozás elméletében a kifejezés fogalmát axiomatikus módon szokás definiálni. Most csak a *Mathematicá*ban használt kifejezés fogalmának egy rekurzív jellegű körülírását adjuk meg. Először az *elemi kifejezések*et (más néven *atomok*at) soroljuk fel. A *Mathematica* atomjai: a *szimbólumok* vagy azonosítók (betűknek és természetes számoknak nem természetes számmal kezdődő sorozata), az egész vagy explicit tizedesponttal megadott valós *számok* és a *füzérek* (sztringek). Minden atomot kifejezésnek tekintünk. Adott `n` természetes szám esetén az

```
f[a1, a2, ..., an]
```

jelsorozatot is kifejezésnek nevezzük, ha `f, a1, ...` és `an` is kifejezés. Ebben az esetben `f` az adott kifejezés *feje* (*head*), `a1, a2, ...` és `an` pedig a kifejezés *argumentumai* (vagy *elemei*).

A programtól a `Head` függvénnyel kérdezhetjük meg azt, hogy mit tekint egy kifejezés fejének. Például:

```
{Head[ab2D], Head[3.14], Head["valami"]}
{Symbol, Real, String}
```

A kifejezés fenti definíciójában feltételeztük azt, hogy speciális karaktersorozatok a megfelelő azonosítóval (ezek teljes listája megtalálható a [89] referenciakönyv 716–719. oldalán) vannak megadva.

A `FullForm` függvény tájékoztat bennünket arról, hogy a program milyen formában tárol egy kifejezést. Figyeljük meg először az atomok tárolási módját:

```
{FullForm[ab2D], FullForm[3.14], FullForm["valami"]}
{ab2D, 3.14, "valami"}
```

Nézzünk ezután néhány további példát:

```
{FullForm[7/5], FullForm[Sqrt[5]], FullForm[3 + 4 I]}
{Rational[7, 5], Power[5, Rational[1, 2]], Complex[3, 4]}

FullForm[y^3 + (1 + z)^2]
Plus[Power[y, 3], Power[Plus[1, z], 2]]

{FullForm[2t + 3t], FullForm[Hold[2t + 3t]]}
{Times[5, t], Hold[Plus[Times[2, t], Times[3, t]]]}
```

Adott kifejezés meghatározott részeit kettős zárójelek (`[[ ]]`, ami a `Part` függvény rövid alakja) használatával jelölhetjük ki. A

`Part[kifejezes, n]` és a `kifejezes[[n]]`

utasítás eredménye például a `kifejezes` `n`-edik argumentuma. Érdemes megjegyezni egyrészt azt, hogy a nulladik argumentum a kifejezés feje, másrészt azt is, hogy `n` negatív szám is lehet. Például:

```
kifejezes1 = f[a1, a2, a3, a4];
{Part[kifejezes1, 0], kifejezes1[[3]], kifejezes1[[-3]]}
{f, a3, a2}
```

A kiválasztás imént ismertetett *elemi módját* rekurzívan alkalmazhatjuk. Tekintsük a következő példákat. Legyen

```
kifejezes2 = h[x + bb, x*y, (u + v)^2]

h[bb + x, x y, (u + v)^2]
```

Mivel a

```
kifejezes2[[3]]

(u + v)^2
```

szintén kifejezés, ezért ennek argumentumait, valamint az argumentumok argumentumait is kiválaszthatjuk a fenti módszerrel:

```
kifejezes2[[3]][[1]]
u + v

kifejezes2[[3]][[1]][[2]]
v
```

A fenti utasítások helyett — szerencsére — az alábbiakat is használhatjuk:

```
kifejezes2[[3, 1]]
u + v
```

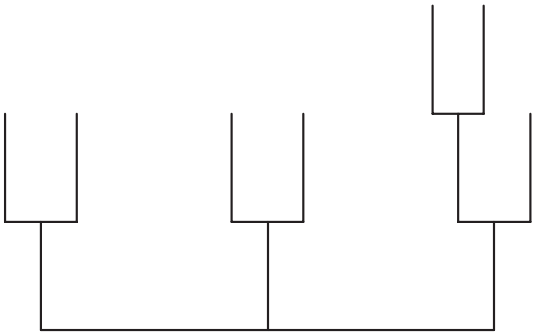
```
kifejezes2[[3, 1, 2]]
v
```

Minden kifejezéshez egyértelműen hozzárendelhető egy gyökérrel bíró, *fastruktúrájú* gráf oly módon, hogy a gráf csúcsaiba a részkifejezések fejét, levélcsúcsaiba pedig az atomokat helyezzük el. Ennek elkészítéséhez is segítséget ad a `TreeForm` függvény:

```
TreeForm[kifejezes2]
h[|          , |          , |                    ]
  Plus[bb, x]  Times[x, y]  Power[|         , 2]
                                  Plus[u, v]
```

Magát a gráfot pedig így rajzolhatjuk meg:

```
<<DiscreteMath`Tree`
ExprPlot[kifejezes2]
```



Egy részkifejezés (beleértve ezentúl az atomokat is) *szintje* (*level*) a gyökértől hozzá vezető út hossza a fenti gráfban:

```
Level[kifejezes2, 2]
{bb, x, bb + x, x, y, x y, u + v, 2, (u + v)^2}

Level[kifejezes2, {2}]
{bb, x, x, y, u + v, 2}
```

Egy részkifejezés több helyen is előfordulhat, ezen előfordulási helyek listáját is megkaphatjuk:

```
Position[kifejezes2, x, Infinity]
{{1, 2}, {2, 1}}
```

A `Depth` függvény egy kifejezés összes szintjeinek számát adja meg.

A fentiekben használt `kifejezes2[[3, 2, 1]]` formula argumentumai éppen az adott részkifejezéshez vezető utat adják meg a fenti gráfban.

Ezen kívül további műveleteket is végezhetünk a kifejezésekkel a következő függvényeket használva:

```
Append                  ReplacePart
AppendTo                Rest
Delete                  Reverse
First                   RotateLeft
Insert                  RotateRight
Join                    Select
Partition               Sort
Prepend                 Take
PrependTo
```

A kifejezések egy másik fontos osztályát alkotják a

listák.

Programozási nyelvekben a kifejezések osztályba sorolásának eszközei a

típusok.

Ezeket arra használják, hogy segítségükkel ellenőrizzék a kifejezések helyességét szemantikai szempontból: például, ha egy egész argumentumú függvény argumentumába valós típusú szám kerül, hibaüzenetet kapunk. Ez, különösen bonyolult programoknál, igen hasznos. Másrészt igen nehézkessé válhat egy olyan programozási nyelv, (például a Pascal) használata, amelyben minden objektum típusát deklarálni kell még az objektum definiálása előtt. A *Mathematica* mindkét eljárást támogatja azon a módon, hogy a kifejezések fejét úgy használja, mint más nyelvek a típusokat. Így tehát minden objektumnak van típusa, de ezt nem kell külön deklarálni, sőt használni sem, csak ha a felhasználó akarja.

A fejek jelentésének ezen kétértelműsége nagyon hasznos, mivel a program a kifejezések viselkedésének lehetséges szemantikai modelljei közül nem kényszerít rá egy kiválasztottat a felhasználóra. Például a `List` függvény egyrészt összefogja az argumentumait és nem csinál velük semmit. Másrészt viszont azt mondhatjuk, hogy argumentumaiból valami újat hoz létre, vagyis az őket tartalmazó listát. Ebben a felfogásban a `List` *függvény* voltára esik a hangsúly. Ehhez hasonló, hogy a `Sin`-ra függvényként gondolunk, de amikor egy egész számra alkalmazzuk, semmi sem történik, csak megtartja az argumentumot és létrehoz egy `Sin` típusú objektumot.

Listák *megadásával* részletesen foglalkozunk a halmazokról szóló 3.1.2. pontban és a lineáris algebráról szóló 3.8. szakaszban. Itt csak felsoroljuk az erre a célra használható függvényeket:

```
Array                    Range
List                     Table
```

A listákon végezhető átalakítások egy része az említett két részben szintén megtalálható; ezért itt csak a következőket említjük meg:

```
Distribute               MapAt
Flatten                  MapIndexed
FlattenAt                Outer
Inner                    Thread
Map                      Transpose
MapAll
```

Ezek a függvények a nagyteljesítményű egysoros eljárások, általánosabban pedig a *funkcionális programozás* alapvető eszközei. Néhány példát adunk alkalmazásukra:

```
Distribute[f[a+b]]
f[a] + f[b]

Inner[f, {a, b}, {c, d}, g]
g[f[a, c], f[b, d]]

Outer[If[#1 < #2, 0, 1]&, {1, 2, 3}, {1, 2, 3}]
{{1, 0, 0}, {1, 1, 0}, {1, 1, 1}}

Thread[Log[x == y], Equal]
Log[x] == Log[y]

MapIndexed[f, {a, b}]
{f[a, {1}], f[b, {2}]}
```

Végül a *Mathematica* egy igen érdekes, speciális lehetőségére hívjuk fel a figyelmet. Korábban említettük azt, hogy a kifejezés fogalmába „minden" beletartozik. Azt is mondhatjuk viszont, hogy minden kifejezés (általánosított értelemben) listának tekinthető. Ez, amint az alábbi példákból is kitűnik, azt jelenti, hogy tetszőleges, a hagyományos programnyelvekben listák kezelése végett bevezetett eljárás megfelelője a *Mathematicá*ban tetszőleges kifejezésre is alkalmazható. Az alkalmazás módja a következő: az

történik, mintha a kifejezés helyébe a `List` fej kerülne, s a program a keletkező listán végrehajtja az adott műveletet, majd az eredményről eltávolítja a `List` fejet és visszarakja az eredetit. Ez az eljárás általánosított listán analóg módon működik:

```
Sort[b + c + a]
Append[f[a, b, c, d], e]
a + b + c
f[a, b, c, d, e]

Position[1 + x + x^2 + y^2, x]
Reverse[%]
{{2}, {3, 1}}
{{3, 1}, {2}}
```

2.3.2. Értékadások típusai. Mintázatok

A program használata során igen gyakran szeretnénk egy kifejezést vagy annak egy részét helyettesíteni egy másik kifejezéssel. Ezt *lokális* vagy *globális értékadással* valósíthatjuk meg.

• Lokális értékadás

A lokális, azaz egyetlen kifejezésre érvényes értékadáshoz szükségünk van egy *tarnszformációs szabályra*. Ezt az *összefüggő* `->` jelsorozattal (ami a `Rule` függvény rövid alakja) adhatjuk meg. A transzformációs szabályt a `ReplaceAll` függvény (ennek rövid alakja: `/.`) felhasználásával alkalmazzuk lokális értékadásra, vagyis egyetlen kifejezés valamely részének *helyettesítésére*. Például:

```
2x + 3y /. x -> 7
14 + 3 y
```

A fenti helyettesítés az `x` értékét csak ideiglenesen változtatta meg:

```
2x + 3y
2 x + 3 y
```

Egy kifejezésben egyszerre több értékadást is végrehajthatunk, ha a transzformációs szabályokat lista elemeiként soroljuk fel:

```
x y z /. {{x -> 2, y -> 3}, {y -> 4, z -> 5}}
{6 z, 20 x}
```

```
2x + 3y /. {x -> a, 2x -> b}
b + 3 y

2x + 3y /. {2x -> y, 3x -> 3}
4 y
```

Az eddigiekben úgynevezett *azonnali* lokális értékadásról volt szó, ugyanis a transzformációs szabály jobb oldalát a program kiértékelt formában tárolja. Ezzel szemben lehetőségünk van arra is, hogy úgynevezett *késleltetett lokális értékadást* hajtsunk végre, amelynél a transzformációs szabály jobb oldala csak meghíváskor értékelődik ki. A kétféle értékadás között a szintaktikai különbség csak abban áll, hogy az utóbbinál a `:>` jelsorozatot kell használnunk a `->` helyett. A szemantikai különbséget az alábbi példán illusztráljuk:

```
Table[x, {3}]
{x, x, x}

Table[x, {3}] /. x -> Random[ ]
{0.0879328, 0.0879328, 0.0879328}

Table[x, {3}] /. x :> Random[ ]
{0.349125, 0.712814, 0.47561}
```

Ha egy kifejezést nemcsak egyszer akarunk végrehajtani, hanem ismételten, mindaddig, amíg lehetséges, akkor a `/.` helyett a `//.` eljárást (teljes nevén `ReplaceRepeated`) alkalmazzuk. Például:

```
2x + 3y /. {2x -> y, y -> 1}
3 + y

2x + 3y //. {2x -> y, y -> 1}
4
```

• Globális értékadás

Az *azonnali globális értékadás* eredményeként a program egy kifejezést vagy annak egy részét automatikusan helyettesíti egy másikkal, *valahányszor az eredeti kifejezés előfordul*. Ezt az értékadást az `=` jellel (ami a `Set` rövid alakja) fejezzük ki. Például

```
veletlen1 = Random[ ]
0.566803
```

A lokális esethez hasonlóan, ha az `=` helyett a `:=` jelsorozatot (ami a `SetDelayed` rövid alakja) használjuk, akkor *késleltetett globális értékadást* valósíthatunk meg. Például:

```
veletlen2 := Random[ ]
Table[veletlen1, {3}]
{0.566803, 0.566803, 0.566803}

Table[veletlen2, {3}]
{0.741224, 0.400726, 0.700565}
```

A függvények megadásáról szóló 3.1.3. pontban további példákat is fogunk mutatni a kétféle értékadás közötti különbség illusztrálására.

• Mintázatok

A *Mathematica* egyik leghatásosabb eszközéhez jutunk azáltal, hogy nemcsak konkrét kifejezésekkel, hanem ilyenek osztályaival, úgynevezett *mintázat*okkal is dolgozhatunk. Minden kifejezésnek meghatározott struktúrája van, s amikor kifejezések osztályaival dolgozunk, akkor tulajdonképpen a struktúrákkal végzünk műveleteket.

A mintázatokat számos helyen használjuk: függvények definiálásánál, transzformációs szabályokban (lásd alább), feltételeket fejezhetünk ki velük, amelyeket azután alkalmazhatunk listából való válogatásnál (lásd a halmazokról szóló 3.1.2. pontot).

A transzformációs szabályokat nemcsak kifejezésekre, hanem kifejezések osztályaira, azaz mintázatokra is lehet alkalmazni. Ez más szóval azt jelenti, hogy adott struktúrájú részek helyettesíthetők valamely kifejezésben.

Mintázatra tipikus példa az `f[x_]` jelsorozat. Ebben a jelsorozatban szerepel az alapvető

*aláhúzás*jel (*blank*), amely azt jelöli, hogy helyébe tetszőleges kifejezést írhatunk. Az `x` jel jelentése pedig az, hogy erre a tetszőleges kifejezésre a továbbiakban `x` néven fogunk hivatkozni:

```
f[a] + f[b] /. f[x_] -> x^2
 2    2
a  + b
```

```
Position[{f[a], g[b], f[c]}, f[_]]
{{1}, {3}}
```

Az aláhúzásjel egy kifejezésben bárhová kerülhet, így `x^n_` a rögzített `x` szám tetszőleges hatványát jelöli, ahol a kitevőre a továbbiakban `n` néven fogunk hivatkozni:

```
In[]:= {1, x, x^2, x^3} /. x^n_->r[n]
Out[]= {1, x, r[2], r[3]}
```

A példa azt is mutatja, hogy a mintázatokat *struktúrá*juk alapján azonosítja a program, tehát nem a matematikai tartalmuk szerint. (Az adott esetben ez kényelmetlenség forrása lehet, majd látni fogjuk, hogy hogyan lehet ezt kiküszöbölni.) Bizonyos egyszerű matematikai tulajdonságokat azonban, amilyen például az összeadás kommutativitása és aszociativitása, figyelembe vesz a program mintázatok *illesztés*énél:

```
In[]:= f[a+b] + f[a+c] /. f[a+x_] + f[y_+a] -> p[x, y]
Out[]= p[b, c]
```

A mintázat (majdnem teljesen) általános definíciója: olyan nem üres jelsorozat, amelynek általános alakját a programnyelvek leírásánál használt jelölésekkel így adhatjuk meg:

```
[x][_][_][_][h]([.]|[:][v])
```

ahol a szögletes zárójel azt jelenti, hogy a benne lévő jel szerepel*het* a mintázatban, a `|` jel kizáró értelemben szerepel, tehát a két oldalán előforduló jelek nem fordulhatnak elő egyszerre a mintázatban, a gömbölyű zárójel pedig itt egyszerűen csoportosításra szolgál.

Ha egy egész mintázatra akarunk az `x` névvel hivatkozni, akkor használjuk a `x:minta` alakú jelölést:

```
In[]:= f[a^b] /. f[x:_^n_] -> p[x, n]
                 b
Out[]= p[a , b]
```

Egy adott mintázathoz illeszkedő kifejezések megkereséséhez az alábbi függvényeket használhatjuk:

`Cases`	`Position`
`Count`	`Select`

Lássunk egy példát:

```
In[]:= Cases[{3, 4, x, x^2, x^4}, x^n_ -> n]
Out[]= {2, 4}
```

A mintázatok általános definíciójában szereplő `h` betű a fejre utal, hivatkozhatunk ugyanis adott típusú — általánosabban adott fejjel (példá-

ul `Complex, Integer, List, Real, Symbol`) rendelkező — mintázatokra. A `v` betű az alapértelmezés szerinti érték:

```
fakt[n_Integer] := n!
fakt[1] + fakt[2.] + fakt[x]
```

```
1 + fakt[2.] + fakt[x]
```

Adott *feltételeket* kielégítő mintázatok kijelölésére szolgál a `/;` jelsorozat:

```
Count[{1, -2, 3, -4}, x_ /; x<0]
```

```
2
```

Bizonyos feltételek teljesülése a `Q` végű függvényekkel is ellenőrizhető:

`EvenQ`	`PrimeQ`
`IntegerQ`	`PolynomialQ`
`NumberQ`	`MatrixQ`
`OddQ`	`VectorQ`

Emeljük például négyzetre a számokat egy adott listában:

```
{2, 7/8, a} /. (x_ /; NumberQ[x]) -> x^2
```

$$\{4, \frac{49}{64}, a\}$$

Mintázatok szerkezetét ellenőrzik az alábbi függvények is:

`AtomQ`	`OrderedQ`
`FreeQ`	`SameQ`
`MatchQ`	`UnsameQ`
`MemberQ`	`ValueQ`

Egy mintázatban *alternatíva*kat is megadhatunk a `|` jellel, amit kiolvasni az *akár ... akár ...* szópárral lehet:

```
h[a|b] := p
{h[a], h[b], h[c], h[d]}
```

```
{p, p, h[c], h[d]}
```

Mintázatok használata teszi lehetővé azt is, hogy olyan függvényeket definiáljunk, amelyek argumentumainak számát előre nem rögzítjük. (Ilyen függvény a beépített `Plus` függvény is.) Számítástechnikai kifejezéssel azt is mondhatjuk, hogy dinamikusan deklarált tömbökön értelmezzük ezeket a függvényeket.

Az `x__` jelsorozat egy vagy több kifejezésből álló sorozatra utal, amelynek a neve a továbbiakban `x` lesz. A háromszoros aláhúzás nulla vagy több kifejezésből álló sorozatot jelöl:

```
h[a___, x_, b___, x_, c___] := hh[x] h[a, b, c]
```

A fenti definíció a kétszer előforduló argumentumokat „kiemeli” h argumentumai közül:

```
h[2, 3, 2, 4, 5, 3]
h[4, 5] hh[2] hh[3]
```

Tanulságos egy azonosan hamis feltétel alkalmazásával megvizsgálni, hogyan próbálkozik a *Mathematica* a mintaillesztésnél:

```
f[a, b, c, d, e] /. f[x__, y__]:>q /; Print[{x}, {y}]
{a}, {b, c, d, e}
{a, b}, {c, d, e}
{a, b, c}, {d, e}
{a, b, c, d}, {e}
f[a, b, c, d, e]
```

Olyan függvényeket is definiálhatunk a mintázatok segítségével, melyeknek bizonyos argumentumai opcionálisak (tehát meghíváskor nem kell ezeknek feltétlenül értéket adnunk). Megadhatjuk az opcionális argumentum alapértelmezés szerinti értékét is:

```
j[x_, y_:1, z_:2] := jp[x, y, z]
{j[a, b], j[a]}
{jp[a, b, 2], jp[a, 1, 2]}
```

Néhány gyakran használt beépített függvény bizonyos argumentumaihoz (természetes) beépített alapértelmezés szerinti értékek tartoznak:

`x_+y_.`	`y`	alapértelmezés szerinti értéke 0
`x_*y_.`	`y`	alapértelmezés szerinti értéke 1
`x_^y_.`	`y`	alapértelmezés szerinti értéke 1

Érdemes most elővennünk egy korábbi példát:

```
{1, x, x^2, x^3} /. x^n_. -> r[n]
{1, r[1], r[2], r[3]}
```

Az általunk definiált függvények opcióinak adhatunk nevet is. Ekkor az alapértelmezés szerinti értéket így adhatjuk meg:

```
Options[j] = {opt1 -> 1, opt2 -> 2}
```

Megkérdezhetjük az `opt1` opcionális argument (röviden: *opció*) alapértelmezés szerinti értékét:

```
opt1 /. Options[j]
1
```

A mintázatok majdnem teljesen általános definíciójában szereplőkön kívül mintázat még a `kif..` és a `kif...` alakú jelsorozat is. Az első egy kifejezést jelöl, amely egyszer vagy többször ismétlődik, a második pedig olyat, amely nullaszor vagy többször ismétlődik. `f[a..]` tehát ezeket jelenti: `f[a], f[a, a], f[a, a, a],`... stb. Nézzünk meg ennek a szerkezetnek az alkalmazására is egy példát:

```
Cases[{f[a], f[a, b, a], f[a, c, a]}, f[(a|b)..]]
{f[a], f[a, b, a]}
```

A mintázatok illeszkedése (mintaillesztés) fontos szerepet játszik a *logikai programozás*nál is, amiről a 4. szakaszban még szólni fogunk.

2.3.3. Opciók

Matematikai tevékenységeink során gyakran találkozhatunk olyan problémákkal (gondoljunk például a numerikus matematikára), amelyek megoldására elvileg sem létezik *egyetlen*, jól használható módszer. Számos ilyen esetben több különböző eljárás közül válogathatunk. A matematikának ezt a jellegzetes vonását a *Mathematica* is tükrözi.

Megemlítettük már azt, hogy például a *Shift* és az *Enter* billentyű együttes lenyomása után kezdi el a program az egy vagy több parancssorba beírt feladat megoldását (a kiértékelést). A program a beépített eljárások közül meghatározott módon választ egyet azokban az esetekben, amikor többféle lehetősége is van a beadott utasítás(ok) kiértékelésére. Ez a helyzet például függvények ábrázolásánál, határozott integrálok numerikus kiszámolásánál, statisztikai próbák megválasztásánál vagy matematikai kifejezések egyszerűsítésénél.

A programnak egyik legfontosabb jellegzetessége azonban éppen az, hogy a felhasználónak igen sok esetben lehetőséget nyújt arra, hogy saját maga válassza meg a kiértékelésnél alkalmazott beépített algoritmust. Ezt a felhasználó *opciók* megadásával teheti meg.

A függvények többsége rendelkezik opciókkal. Minden opciónak meghatározott neve és több lehetséges értéke is van. Egy beépített függvény opcióinak listáját az

```
Options[beepitettfv]
```

utasítással kapjuk meg. Tekintsük például a függvények ábrázolásához használható `Plot` eljárást (lásd a 3.1.5. pontot is):

```
Options[Plot]
{AspectRatio -> 1/GoldenRatio, Axes -> Automatic,
 AxesLabel -> None, AxesOrigin -> Automatic,
 AxesStyle -> Automatic, Background -> Automatic,
 ColorOutput -> Automatic, Compiled -> True,
 DefaultColor -> Automatic, Epilog -> { },
 Frame -> False, FrameLabel -> None,
 FrameStyle -> Automatic, FrameTicks -> Automatic,
 GridLines -> None, MaxBend -> 10.,
 PlotDivision -> 20., PlotLabel -> None,
 PlotPoints -> 25, PlotRange -> Automatic,
 PlotRegion -> Automatic, PlotStyle -> Automatic,
 Prolog -> { }, RotateLabel -> True,
 Ticks -> Automatic, DefaultFont :> $DefaultFont,
 DisplayFunction :> $DisplayFunction}

Length[%]
27
```

A *Mathematica* transzformációs szabályként, tehát a következő alakban kezeli az opciókat:

opció neve −> opció értéke.

Mindegyik opcióhoz egy meghatározott alapértelmezés szerinti érték is tartozik. Ezt használja a program abban az esetben, amikor nem adjuk meg az opciót. A

`SetOptions`

függvény alkalmazásával közölhetjük a programmal azt, hogy melyik opcióértéket tekintse alapértelmezés szerinti értéknek.

Térjünk vissza a `Plot` függvény opcióihoz. Ezek közül többnek (ilyen például a `Frame`) a jelentését, valamint a lehetséges opcióértékeket (például `False` és `True`) könnyen kitalálhatjuk. Más esetben az információszerzés korábban ismertetett módját használhatjuk. (A `Plot` függvény néhány opciójának értelmét a 3.1.5. pontban is kifejtjük.)

Itt jegyezzük meg, hogy sokszor nem egyszerű feladat az opciók lehetséges értékeinek a programból történő felderítése. (Ilyenkor érdemes a `?`

súgót is megkérdezni és a *Help*-et is megnézni.) Wolfram könyve ([89]) számos esetben ezeknél több információt tartalmaz. N. Blachmané (lásd [9]-et) pedig éppen ennek a hiányosságnak a megszüntetését tűzte ki célul.

Bármely függvény argumentumában a megengedett opciók közül akárhányat bármilyen sorrendben felsorolhatunk. A különbözőket a `,` jellel kell elválasztanunk.

Befejezésül még a számos beépített függvénynél alkalmazható `Compiled` opcióról és a gyakori `Automatic` opcióértékről ejtünk néhány szót.

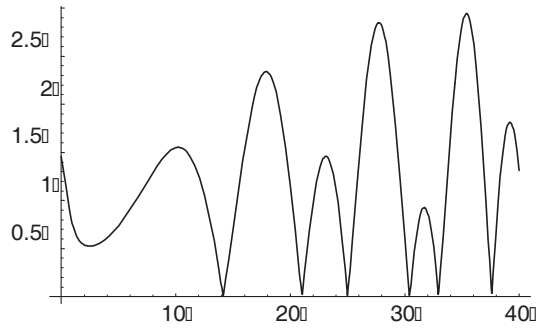
Vegyük példaként ismét a `Plot` függvényt. Mivel ez sokszor számolja ki a beadott függvény helyettesítési értékeit, fontos, hogy a program minél gyorsabban hajtsa végre ezeket a műveleteket. Ezt segíti a `Compiled` opció. Alapértelmezés szerinti értéke általában `True`, ami azt jelenti, hogy a *Mathematica* a kiértékelésnél a leggyorsabban működő algoritmusokat, tehát a gépi pontosságú számokat (lásd a 3.1.3. pontot) használja. Ha a kiértékelés során nagypontosságú számokat (lásd szintén a 3.1.3. pontot) szeretnénk alkalmazni, akkor a `Compiled -> False` opciót kell megadnunk.

Számos opció értéke lehet `Automatic`. Ilyen esetekben a program választ ki egy (a beadott feladattól — is — függő) lehetséges opcióértéket, amiről szükség esetén a

```
FullOptions
```

függvény segítségével kaphatunk információt. Például:

```
abra = Plot[Abs[Zeta[1/2 + I y]], {y, 0, 40}]
```




```
Options[abra, PlotRange]
{PlotRange -> Automatic}

FullOptions[abra, PlotRange]
{{-1., 41.}, {-0.0735488, 3.0155}}
```

2.3.4. Attribútumok

A beépített függvények többsége rendelkezik egy vagy több (*attribútum*nak nevezett) tulajdonsággal. Ezek listáját az

```
Attributes
```

függvénnyel így kaphatjuk meg:

```
Attributes[Sin]
{Listable, Protected}

Attributes[Plus]
{Flat, Listable, OneIdentity, Orderless, Protected}
```

Például a `Listable` attribútum azt jelenti, hogy ha a függvény argumentumába listát (speciálisan vektort vagy mátrixot) írunk, akkor a program a szóban forgó függvényt a lista minden elemére külön-külön alkalmazza:

```
Sin[{Pi/4, Pi/10}]
{1/Sqrt[2], Sin[Pi/10]}

Sin[{{a, b}, {c, d}}]
{{Sin[a], Sin[b]}, {Sin[c], Sin[d]}}
```

A beépített matematikai függvények mindegyikét ellátták a `Listable` attribútummal, ezért listákkal aritmetikai műveleteket is végezhetünk:

```
v = {2, 3, 4}
v^2/(v-1)
{4, 9/2, 16/5}
```

A `Flat` attribútum a függvény asszociatív, az `Orderless` pedig kommutatív tulajdonságát fejezi ki.

Függvényértékek felülírását akadályozza meg a `Protected` attribútum:

```
Sin[Pi/10] = (Sqrt[5]-1)/4
Set::write: Tag Sin in Sin[Pi/10] is Protected.
```

Az `Unprotect` függvényt felhasználva azonban beépített függvények helyettesítési értékeit is megváltoztathatjuk. Például így:

```
Unprotect[Sin]
Sin[Pi/10] = (Sqrt[5]-1)/4;
Protect[Sin];
Sin[Pi/10]
```

$$\frac{-1 + \text{Sqrt}[5]}{4}$$

Egy általunk definiált függvényhez (általában szimbólumhoz) is hozzárendelhetünk attribútumokat. A *Mathematica* 2.2.x változataiban a következők használatára van lehetőségünk:

`Constant`	`OneIdentity`
`Flat`	`Orderless`
`HoldAll`	`Protected`
`HoldFirst`	`ReadProtected`
`HoldRest`	`Temporary`
`Listable`	`Stub`
`Locked`	

Attribútumok megadásához és törléséhez pedig a következő belső függvényeket használhatjuk:

`Attributes`	`SetAttributes`
`ClearAttributes`	

Több esetben is szükségünk lehet arra, hogy csupán kijelöljünk bizonyos műveleteket. Ezt a `Hold*` attribútumok megadásával tehetjük meg. Például:

```
SetAttributes[f, HoldFirst]
f[1 + 1, 2 + 3]
f[1 + 1, 5]

SetAttributes[f, HoldAll]
f[1 + 1, 2 + 3]
f[1 + 1, 2 + 3]
```

A `Clear[f]` utasítással a korábban definiált `f` függvénynek csak a helyettesítési értékeit lehet kitörölni, attribútumai megmaradnak. A

```
ClearAll[f]
```

utasítás törli ki az `f` helyettesítési értékeit is és az attribútumokat is.

2.3.5. Kiértékelés

A *Mathematica* által végrehajtott alapvető művelet a *kiértékelés*. Valahányszor beviszünk egy kifejezést, a program kiértékeli azt, majd visszaad egy eredményt. Ehhez a beépített és a felhasználó által megadott definíciókat használja fel.

Korábban láttuk, hogy a kifejezés fogalma hogyan képes egységesíteni az előforduló objektumokat. Most azt mutatjuk meg, hogy a kiértékelés fogalma az elvégzett műveletek egységes tárgyalására alkalmas. Kiértékelés ugyanis egy számítás elvégzése, egy egyszerűsítés vagy egy eljárás végrehajtása is.

A *Mathematica* egy végtelen kiértékelő rendszer. Ez azt jelenti, hogy a bevitt kifejezéseket mindaddig alakítja, amíg további definíció nem alkalmazható az átalakítására.

Ha az `f` függvény még nincs definiálva, akkor csak a beépített függvények tulajdonságai használhatók fel:

```
f[3] + 4f[3] + 1
```

```
1 + 5 f[3]
```

Ha most definiáljuk `f`-et, akkor az eredmény tovább alakítható:

```
f[x_] = x^2
%%
```

```
46
```

A kifejezéseket kiértékelésük előtt a *Mathematica* szabványos vagy kanonikus (*standard*) alakra hozza. Eközben fölhasználja a szereplő függvények kommutatív (`Orderless`) és asszociatív (`Flat`) tulajdonságát, valamint listára való alkalmazhatóságukat (`Listable`). Az egyes részek sorbarendezésénél alkalmazza (durván szólva) az ábécérendet is. Ezért tudja például megkapni az alábbi eredményt:

```
f[a+c+b] - f[c+b+a]
```

```
0
```

A szabványos alak azonban céljainktól függően más és más lehet. Nyilvánvaló például, hogy polinomokat egyszer monomok összegeként, máskor tényezők szorzataként tudunk jobban használni. Az `Expand` függvény adja az első alakot, a másodikat pedig a `Factor` függvény.

Sok matematikai probléma nehézsége éppen a megfelelő kanonikus alak megtalálásában áll.

• Szabályos kiértékelés

A legtöbb kifejezés kiértékelésére a *Mathematica* az úgynevezett *szabályos kiértékelés*t használja. Ennek lépéseit mutatja az alábbi táblázat:

- Kiértékeli a kifejezés fejét.
- Kiértékeli egymás után az összes elemet.
- Alkalmazza a kommutativitást, asszociativitást és listára való alkalmazhatóságot.
- Alkalmazza a felhasználó által megadott definíciókat.
- Alkalmazza a beépített definíciókat.
- Kiértékeli az eredményt.

Wolfram példája alapján ismertetjük egy kifejezés kiértékelését. Tegyük fel, hogy `a=7`, és a `2a*x+a^2+1` kifejezést akarjuk kiértékelni. Ennek teljes alakja

```
Plus[Times[2, a, x], Power[a, 2], 1].
```

Először a program ezt értékeli ki: `Times[2, a, x]`, az eredmény

```
Times[2, 7, x].
```

Most fölhasználja a szorzás beépített definícióját:

```
Times[14, x].
```

A következő lépés ennek kiértékelése: `Power[a, 2]`. Az eredmény `a` kiértékelése után:

```
Power[7, 2].
```

Most fölhasználja a hatványozás beépített definícióját: `49`.
A `Plus` függvény argumentumainak kiértékelése utáni részeredmény:

```
Plus[Times[14, x], 49, 1].
```

Ezt az eredményt az összeadás beépített definíciója adja:

```
Plus[50, Times[14, x]].
```

Ezt látjuk a képernyőn:

```
50 + 14 x.
```

A lépéseket kellő gyakorlattal követhejtük a `Trace` függvény felhasználásával is.

```
a = 7;
```

```
Trace[2a x + a^2 + 1]
{{a, 7}, 2 7 x, 14 x}, {{a, 7}, 7^2, 49},
14 x + 49 + 1, 1 + 49 + 14 x, 50 + 14 x}
```

• **Különös kiértékelés**

Néhány fontos *Mathematica*-függvény nem szabályos: *különös* kiértékelési eljárás eredményeként kapja meg értékét. Ezek közül a legfontosabbak:

`Do`	`Plot`
`Function`	`Set (=)`
`If`	`Table`

Nyilvánvaló például, hogy az `x=3` (vagyis `Set[x, 3]`) kifejezés kiértékelésénél a *Mathematica* a bal oldalon megjelenő `x` értékét nem számítja ki.

A szabályos kiértékelésnél a részkifejezések egymás után értékelődnek ki. Valamely részkifejezés (az összes, az első, az összes az első kivételével) kiértékelése megakadályozható a

`HoldAll`	`HoldRest`
`HoldFirst`	

függvények közül a megfelelővel. Azt is megtehetjük, hogy kényszerítjük a programot egy egyébként ki nem értékelendő rész kiértékelésére. Erre szolgál az `Evaluate` függvény.

Felsorolunk még néhány, a témakörhöz kapcsolodó fontos függvényt:

`HeldPart`	`ReplaceHeldPart`
`Hold`	`ReleaseHold`
`HoldForm`	`Unevaluated`

2.4. Külső kapcsolatok

Áttérünk a mag és a felhasználói felület, valamint az egész program és az adatállományok kapcsolatának elemzésére. Tanulmányozzuk a memóriakezelést, továbbá a más programokkal való információcserét.

2.4.1. Be- és kimenet

Itt először a mag és a felhasználói felület közötti kapcsolattal foglalkozunk, később fogunk áttérni a program és az adatállományok kapcsolatára.

• A mag és a felhasználói felület

Számos lehetőségünk van arra, hogy az eredményeket különböző formákban kapjuk meg.

Az erre a célra használható alábbi függvények a számok alakját módosítják; hogy miként, azt a 3.1.3. pontban részletesebben is megtárgyaljuk.

```
AccountingForm          PaddedForm
BaseForm                PrecedenceForm
EngineeringForm         ScientificForm
NumberForm
```

A kifejezések különböző alakú kiíratását a következő függvények szabályozzák:

```
FullForm                PrintForm
HoldForm                TreeForm
InputForm               Shallow
OutputForm              Short
Print                   TextForm
```

A függvények másik csoportja azt teszi lehetővé, hogy az eredmények az igényeknek megfelelő alakban jelenjenek meg.

```
ColumnForm              Short
Continuation            StringBreak
Format                  StringForm
Indent                  Subscript
LineBreak               Subscripted
MatrixForm              Superscript
SequenceForm            TableForm
Shallow
```

Ha az angol nyelv helyett valamelyik nagyobb európai nyelv használatára akarunk áttérni, akkor először is be kell állítanunk a megfelelő rendszerváltozó értékét:

```
$Language="French"
French
```

Ha ezután behívjuk a `Utilities`Language`` programcsomagot, akkor érdemes megtekintenünk a `$Letters` és a `$StringOrder` rendszerváltozó

értékét. A programcsomag `CharacterTable` függvénye pedig megadja a karaktertáblázatot.

Kifejezések és füzérek alakja vizsgálható az alábbi függvényekkel:

```
DigitQ                  StringPosition
LetterQ                 SyntaxLength
LowerCaseQ              SyntaxQ
String                  UpperCaseQ
StringByteCount         $Letters
StringMatchQ
```

A függvények következő osztálya kifejezések és füzérek közötti átalakításokat végez:

```
FromCharacterCode       StringTake
StringDrop              ToCharacterCode
StringInsert            ToExpression
StringJoin              ToHeldExpression
StringLength            ToLowerCase
StringReplace           ToString
StringReverse           ToUpperCase
```

Néhány függvény arra a célra szolgál, hogy az eredményt más program részeként tudjuk felhasználni:

```
CForm                   TeXForm
FortranForm
```

Néhány függvény feladata az, hogy segítse a program és a felhasználó közötti *interaktív* kommunikációt. Egy *párbeszéd* (*dialógus*) keretében például beszélgethetünk a programmal, miközben számol. A párbeszéd végeztével a `Return` utasítással térhetünk vissza:

```
Dialog                  InputString
DialogIndent            Return
Input
```

Külső parancsok végrehajtásához használható a `!` jel és a `RunThrough` függvény.

• Állományokkal való kapcsolat

Más programokhoz hasonlóan a *Mathematicá*ban is gyakori, hogy az adatokat állományból olvassuk be, vagy az eredményeket állományokba írjuk ki. Ennek a ténynek itt azonban van néhány specialitása is. Egyrészt előfordul az is, hogy kifejezéseket írunk állományba vagy olvasunk be állo-

mányból. Másrészt amennyiben a merev lemezen rendelkezésünkre álló hely nem korlátlan, érdemes is az állományba írásra és az onnan való olvasásra berendezkednünk, mert az ilyenkor használt szöveges állományok nagyságrendileg kisebbek, mint a *Mathematica* által létrehozott, `.ma` kiterjesztésű jegyzetfüzetek. Ugyancsak az állományok használatára szoríthat bennünket az, ha kötegelt (*batch*) üzemmódban akarunk dolgozni, esetleg egy nagyobb gépen: ilyenkor ugyanis a program futása közben más feladatot is végezhetünk. (Erre még akkor is szükségünk lehet, ha gépünk operációs rendszere látszólag vagy valóságosan támogatja a párhuzamos feldolgozást.)

Állománykezelésre használható belső függvények:

`Close`	`Read`
`Find`	`ReadList`
`FindList`	`Save`
`Get`	`SetStreamPosition`
`InputStream`	`Skip`
`InString`	`Splice`
`OpenAppend`	`StreamPosition`
`OpenRead`	`Streams`
`OpenTemporary`	`StringToStream`
`OpenWrite`	`Write`
`OutputStream`	`WriteString`
`Put`	`$Output`
`PutAppend`	

Szeretnénk megtudni például azt, hogy mi van a `polyall.txt` állományban.

```
!!d:\sajatnb\maradek\polyall.txt
1-x^3
```

Ha az állomány tartalmát kifejezésként szeretnénk használni, akkor a

```
Get
```

utasítást vagy annak rövid alakját (`<<`) használhatjuk.

```
Get["d:\sajatnb\maradek\polyall.txt"]
     3
1 - x
```

```
Factor[%]
                    2
(1 - x) (1 + x + x )
```

Tényleg sikerült kifejezésként beolvasni. Az előbbivel egyenértékű az alábbi megoldás is:

```
<<d:\sajatnb\maradek\polyall.txt
```

$$1 - x^3$$

Az eredmények állományba való kiírására való a

Put

utasítás és rövid alakja (>>).

```
Factor[x^3 + 1]>>ize
!!ize
(1 + x)*(1 - x + x^2)
```

Ha az újabb kifejezést az előző állományhoz hozzá akarjuk fűzni, akkor a PutAppend utasítást vagy rövid alakját (>>>) használhatjuk:

```
PutAppend[Factor[x^2 - 1], "ize"];
Factor[x^4 - 1] >>> ize
!!ize
(1 + x)*(1 - x + x^2)
(-1 + x)*(1 + x)
(-1 + x)*(1 + x)*(1 + x^2)
```

Tegyük fel, hogy adott a matrix.txt állomány; tekintsük meg a tartalmát:

```
!!matrix.txt
10 20
30 40
50 60
```

Ha ezeket a számokat egy mátrixba szeretnénk betenni olyan módon, hogy minden sor egy listába kerüljön, akkor a következőképpen járhatunk el:

```
matrixom = ReadList["matrix.txt", {Number, Number}]
{{10, 20}, {30, 40}, {50, 60}}
```

Ha viszont három-három számból szeretnénk egy-egy listát létrehozni, akkor ezt tehetjük:

```
matrixod = ReadList["matrix.txt", {Number, Number, Number}]
{{10, 20, 30}, {40, 50, 60}}
```

És ha csak egyetlen listát akarunk?

```
matrixa = ReadList["matrix.txt", Number]
{10, 20, 30, 40, 50, 60}
```

Ha nem tudjuk, vagy nem akarjuk kihasználni, hogy egy oszlopban hány szám van, akkor a `RecordLists` opciót `True`-ra állítjuk:

```
matrixunk = ReadList["matrix.txt", Number,
       RecordLists -> True]
{{10, 20, 30}, {40, 50, 60}}
```

Bonyolultabb állományoknál megnyitjuk olvasásra az adatokat tartalmazó állományt, ezáltal létrehozunk egy befelé irányuló áramot (`InputStream`), majd soronként beolvassuk az adatokat:

```
bearam = OpenRead["matrix.txt"]
InputStream["matrix.txt", 62]
```

Ezek után soronként olvashatjuk be a számokat:

```
sor1 = Read[bearam, {Number, Number}];
sor2 = Read[bearam, {Number, Number}];
sor3 = Read[bearam, {Number, Number}];
```

Az adatok beolvasásának befejezése után lezárjuk az áramot:

```
Close[bearam]
matrix.txt
```

A sorokból összeállíthatjuk a mátrixot:

```
matrixotok = {sor1, sor2, sor3}
{{10, 20}, {30, 40}, {50, 60}}
```

Nézzünk példát egy szöveget, kifejezést és adatot egyaránt tartalmazó állomány kezelésére:

```
!!polimat.txt
Ez itt egy polinom:
1 - x^3

Ez pedig egy mátrix:
1 2
3 4
5 6
7 8
```

Nyissuk meg az állományt:

```
befele = OpenRead["polimat.txt"]
InputStream[polimat.txt, 62]
```

Beolvassuk az első feliratot — szövegként:

```
szoveg1 = Read[befele, String]
Ez itt egy polinom:
```

A következő sort kifejezésként akarjuk használni:

```
poli = Read[befele, Expression]
     3
1 - x
```

A soremelést jelző karaktertől meg kell szabadulnunk:

```
Read[befele, Character]
```

Jöhet a második szövegrészlet:

```
szoveg2 = Read[befele, String]
Ez pedig egy mátrix:
```

Mivel adatokból álló táblázat következik, használhatjuk a `ReadList` utasítást:

```
matrixuk = ReadList[befele, Number, RecordLists -> True];
```

Előfordul, hogy az állományban (amelynek neve elé szükség esetén elérési utat is írunk) az adatok *vesszővel* vannak elválasztva egymástól:

```
!!vesszos.txt
1, 2, 3
4, 5, 6
```

Ha tudjuk, hogy hány oszlopban helyezkednek el az adatok, akkor így járhatunk el:

```
Partition[Map[First, ReadList["vesszos.txt",
         {Number, Character}]], 3]
{{1, 2, 3}, {4, 5, 6}}
```

(A `ReadList` függvény ugyanis számból és vesszőből álló párok listáját állítja elő, ebből a listából `First /@` olyat készít, amely csak a párok első elemét tartalmazza, végül az így kapott eredményt a `Partition` függvény három hosszúságú részlistákra bontja.)

Ha nem ismerjük az oszlopok számát, akkor egy (lassabb, de működő) megoldásnak ez tűnhet:

```
ReadList["vesszos.txt", Word, RecordLists -> True,
        WordSeparators -> {",", " "}]
{{1, 2, 3}, {4, 5, 6}}
```

Kiderül azonban, hogy ez az eredmény nem az, aminek látszik:

```
FullForm[%]
List[List["1", "2", "3"], List["4", "5", "6"]]
```

A karaktereket tehát még számokká kell átalakítanunk. (A 2. szinten hat a `ToExpression` függvény, erre utal alább a `Map` függvény harmadik argumentuma.)

```
Map[ToExpression, ReadList["vesszos.txt", Word,
   RecordLists -> True, WordSeparators -> {",", " "}], {2}]
{{1, 2, 3}, {4, 5, 6}}
```

Tiszta függvények használatával a sebesség gyorsítható:

```
Map[ToExpression[StringJoin["{", #, "}"]]&,
   ReadList["vesszos.txt", String]]
{{1, 2, 3}, {4, 5, 6}}
```

Ha ez kicsit gyors a tisztelt Olvasónak, akkor a következőket érdemes végigcsinálnia:

```
ReadList["vesszos.txt", String]
{1, 2, 3, 4, 5, 6}

FullForm[%]
List["1, 2, 3", "4, 5, 6"]

StringJoin["{", %%, "}"]
{1, 2, 34, 5, 6}
FullForm[%]
"{1, 2, 34, 5, 6}"
```

Ezek után érthető, miért működik a fent megadott megoldás.

Egy állományba szeretnénk *kiírni* az alábbi mátrixot:

```
matrix = {{10, 20, 30}, {40, 50, 60}};
```

Nem áll rendelkezésünkre készen a `ReadList` utasításhoz hasonló kiíró utasítás erre a célra, ezért először is megnyitjuk írásra a célállományt, ezáltal létrehozunk egy kifelé irányuló áramot (`OutputStream`), majd a mátrixot egyetlen olyan füzérré alakítjuk, amelyben a számok szóközzel,

soronként pedig soremelésjellel vannak elválasztva. Lássuk ezt most a fenti egyszerű példára alkalmazva:

```
kiaram = OpenWrite["matrix.txt"]
OutputStream[matrix.txt, 49]

matrixstring = ToString[TableForm[matrix,
      TableSpacing -> {0, 1}]]
10 20 30
40 50 60

FullForm[%]
"10 20 30\n40 50 60"
```

Most ezt a füzért elküldjük a kifelé irányuló áramba:

```
WriteString[kiaram, matrixstring]
```

Ellenőrzés:

```
!!matrix.txt
10 20 30
40 50 60
```

Rendben van, akkor zárunk:

```
Close[kiaram]
matrix.txt
```

Most megadunk egy hasznos függvényt, amelynek két kötelező argumentuma egy állománynév és egy mátrix. A harmadik argumentum opcionális: az oszlopokat elválasztó karaktert — amelynek alapértelmezése a tabulátorjel (`\t`)— specifikálja:

```
writeMatrix[allomanynev_String, matrix_List,
   elvalaszto_String:"\t"] :=
   With[{belsonev = OpenWrite[allomanynev]},
      Scan[(WriteString[belsonev, First[#]];
      Scan[WriteString[belsonev, elvalaszto, #]&, Rest[#]];
   WriteString[belsonev, "\n"])&, matrix];
   Close[belsonev]]
```

Alkalmazzuk most az új függvényt:

```
writeMatrix["tabula.tor", matrix]
tabula.tor
```

```
!!tabula.tor
```

```
10 20 30
40 50 60
```

Most pedig vesszőkkel fogjuk elválasztani az elemeket:

```
writeMatrix["vesszocs.ke", matrix, ","]
```

```
vesszocs.ke
```

```
!!vesszocs.ke
```

```
10, 20, 30
40, 50, 60
```

A fenti példák némelyike egyszerűbben oldható meg a

```
Splice
```

függvény használatával.

A `Utilities`BinaryFiles`` programcsomagban olyan függvényeket találunk, amelyek a bináris állományokkal végzett munkát könnyítik meg, bár ezek tulajdonképpen csak a megfelelő belső függvények alkalmas opcióval (például `PageWidth -> Infinity`) ellátott változatai:

```
OpenAppendBinary        ReadListBinary
OpenReadBinary          ToBytes
OpenWriteBinary         WriteBinary
ReadBinary
```

Mindeddig adatok, kifejezések és füzérek állományba való írásáról és onnan való beolvasásáról volt szó. Hang és kép alkalmasan választott kimenetre küldhető a `Display` belső utasítással.

Háromdimenziós *ábrák* más programok számára használható formáját kaphatjuk meg állományban a `Graphics`ThreeScript`` programcsomag

```
ThreeScript
```

függvényének felhasználásával. A `Utilities`DXF`` programcsomag

```
WriteDXF
```

függvénye pedig az AutoCAD program számára feldolgozható formában készíti el az állományokat.

2.4.2. A program környezete

A gépen található állományok a *Mathematica* utasításaival is vizsgálhatók és módosíthatók:

```
DeleteFile                FileDate
Encode                    FileNames
EndOfFile                 FileType
FileByteCount             RenameFile
```

A könyvtárakra vonatkozó hasonló jellegű utasítások:

```
CopyDirectory             ParentDirectory
CreateDirectory           RenameDirectory
DeleteDirectory           ResetDirectory
Directory                 SetDirectory
DirectoryStack            $Path
HomeDirectory
```

Mint már említettük, a *Mathematica* interaktív program, az utasításokat soronként értelmezi. Minden alkalomkor egy külső ciklus indul el, amely várja a bemenetet, feldolgozza azt, megadja az eredményt, majd újra vár a bemenetre. Ehhez a legkülső ciklushoz tartozik néhány függvény és néhány *rendszerváltozó*, amelyekkel (legalább vázlatosan) érdemes megismerkednünk. Az itt elmondottak függhetnek a felhasználói felülettől és az operációs rendszertől.

A legtöbb rendszerváltozó két osztály valamelyikébe sorolható. Az egyik osztály tagjai egy-egy fontos *adatot* tárolnak:

```
$CommandLine              $OperatingSystem
$CreationDate             $ReleaseNumber
$DumpDates                $SessionID
$MachineID                $System
$MachineName              $Version
$MachineType              $VersionNumber
```

Ilyen például a használt *Mathematica* program változatának számát megadó paraméter, amelynek értéke az egyik általunk használt gépen a következő:

```
$Version
Windows 2.2 (May 3, 1994)
```

A másik osztály tagjai pedig arra adnak választ, hogy egy adott *szolgáltatás* rendelkezésünkre áll-e:

```
$BatchInput                $LinkSupported
$BatchOutput               $Notebooks
$DumpSupported             $PipeSupported
$Linked
```

Nézzük meg ezek közül is kettő értékét:

```
{$DumpSupported, $LinkSupported}
{False, True}
```

A kapott eredmény tükrözi az adott operációs rendszer lehetőségeit.

Az alábbi függvények az *operációs rendszerrel* való kapcsolattartás eszközei és hibakeresésre használhatók:

```
Environment                Splice
Interrupt                  Trace
Pause                      TraceDialog
Run                        TracePrint
RunThrough                 TraceScan
```

Az `Environment` függvény például az operációs rendszer valamely változójának értékét adja meg.

• Memóriakezelés

Különösen a matematikai programcsomagok igazi felhasználási területén, a szimbolikus számításoknál fordulhat elő, hogy egy számolás nagyon sok időt vesz igénybe, vagy nagyon sok memóriát használ. Az előbbin (hacsak például nem tízezer év a „nagyon sok" becsült értéke) kellő türelemmel (és a `Timing` belső függvény, valamint a `Utilities`ShowTime`` programcsomag `ShowTime` függvényének segítségével) úrrá lehetünk, az utóbbi azonban keményebb korlátnak tűnik. Ennek a korlátnak a lazításához szeretnénk most tanácsokat adni az idetartozó függvények közül a fontosabbak ismertetésével.

```
ByteCount                  MemoryConstrained
LeafCount                  MemoryInUse
MaxMemoryUsed              Share
```

A `Utilities`MemoryConserve`` programcsomagból pedig az alábbi függvényeket használhatjuk:

```
Off[MemoryConserve]        $MemoryIncrement
On[MemoryConserve]
```

Megtudhatjuk, hogy mennyi memóriát használ jelenleg a rendszer:

```
MemoryInUse[ ]
574520
```

Deklarálunk egy hatalmas mátrixot, és tapasztaljuk, hogy megnőtt a lekötött memória. Az alábbi példában kivételesen leírtuk a nem begépelendő `In[2]:=` jelsorozatot is:

```
In[2]:=
Range[10000];
MemoryInUse[ ]
771656
```

A memóriának a lista által elfoglalt része fölszabadítható, de csak úgy, ha töröljük a nagy mátrixot, amely `Out[2]`-ként szerepel a memóriában:

```
Unprotect[Out];
Out[2]=.
```

Javasoljuk az Olvasónak, hogy ellenőrizze, vajon a felhasznált memóriaterület nagysága ezután visszaállt-e az eredetihez közeli, kisebb értékre.

Megtudhatjuk azt is, hogy a jelenlegi alkalommal mennyi volt a legnagyobb lefoglalt memóriaterület:

```
MaxMemoryUsed[ ]
1000616
```

Előírhatjuk azt is, hogy egy kifejezést csak akkor értékeljen ki a program, ha ehhez legfeljebb adott mennyiséggel növekedik a lekötött memóriaterület: `MemoryConstrained[kifejezes, tobbletbyte, hibauzenet]`.

Azt, hogy egy kifejezés tárolásához hány bájt szükséges, a `ByteCount` függvény adja meg, egy kifejezés faalakjában pedig a levelek (azaz az atomok) számát a `LeafCount` függvény szolgáltatja.

Megjegyzendő, hogy a `ByteCount` a szükséges bájtok maximális számát adja meg. Ha viszont lehetséges, a *Mathematica* igyekszik a többször előforduló azonos részeket egyszer tárolni. Erre azonban a `Share[ ]` segítségével rá is kényszeríthetjük.

A `Share` belső függvény automatikus indítását — amire kis memóriájú gépnél lehet szükségünk — a `Utilities`MemoryConserve`` programcsomag segítségével végezhetjük:

```
<<Utilities`MemoryConserve`
$MemoryIncrement
100000
```

Ezután a `Share` függvény azonnal elindul, ha a memóriában elfoglalt hely 100 000 bájttal megnövekszik. Kísérletezéshez vegyünk kisebb értéket:

```
Unprotect[$MemoryIncrement];
$MemoryIncrement = 10000;
Table[ToString[0], {2^11}];
Length[%]
MemoryConserve::start:
      Running Share[ ] to conserve memory.

MemoryConserve::end:
      Finished running Share[ ]; 141064
            bytes of memory freed. MemoryConserve::start:
      Running Share[ ] to conserve memory.

MemoryConserve::end:
      Finished running Share[ ]; 41040
            bytes of memory freed.
2048
```

Mivel egy hatalmas táblázatot hoztunk létre, a `Share` függvény működésbe lép és a memória egy jelentős részét felszabadítja.

Az automatikus memóriatakarékoskodás leállítható:

```
Off[MemoryConserve]
```

és újraindítható:

```
On[MemoryConserve]
```

Memóriakezeléshez használható a `CleanSlate` programcsomag is, amelyet a 2.2. szakaszban említettünk meg.

Végül megmutatjuk, hogy a *Mathematica* használatánál is felmerülő azon igény, amely szerint szeretnénk a rendszer adott alkalomkor fennálló állapotát rögzíteni és egy későbbi időpontban felhasználni, hogyan elégíthető ki. `Dump["allomanynev"]` hatására „minden” kiíródik egy állományba. A `Dump` függvény meghívásakor nyitva lévő állományokhoz tartozó mutatók

nem mentődnek ki, ugyanis arra nincs garancia, hogy a legközelebbi felhasználáskor azok az állományok létezzenek, s így a mutatóknak értelmük legyen.

Az újrafelhasználás egy lehetséges módja:

```
Dump["allomanynev", $Line, <<init.m]
```

Mivel a `Dump` függvény egy sokmegabájtos állományt hoz létre, általában célszerűbb a

`Save`

függvény alkalmazása azokra az objektumokra, amelyekre tényleg szükségünk lehet a későbbiekben, ez ugyanis egy kicsi, szöveges állományt hoz létre:

```
f[x_] := x^3
Save["fideigl", f]
Clear[f]
<<ftmp
x^3
```

A példa mutatja, hogy a törlés ellenére az állományból visszaállítható f definíciója.

2.4.3. Kapcsolat más nyelvekkel és programokkal

Kérhetjük, hogy eredményünk olyan alakban jelenjék meg, amit azután föl tudunk használni egy C programban, egy Fortran programban, vagy pedig egy TEX állományban. Ehhez az alábbi függvények használhatók:

```
CForm                    TeXForm
FortranForm
```

A *Mathematica* nyílt rendszer. A

`MathLink`

elnevezésű program teszi lehetővé közvetlen kapcsolódását más felhasználói programokhoz és programnyelvekhez.

A `MathLink` használatát részletesen el lehet sajátítani a `MathSource`-on található oktatóprogramból (0206-693, `A MathLink Tutorial`). Itt

először megmutatjuk, hogy miként lehet a *Mathematicá*hoz C nyelven kiegészítő programot írni, illetve a *Mathematica* utasításait fölhasználni egy C nyelvű program részeként.

• Függvény írása a *Mathematicá*hoz C nyelven

A következő eljárásokat fogjuk alkalmazni:

`Install`	`Uninstall`
`LinkObject`	`Unlink`

Az alábbiakban felhasználjuk a `MathLink` példaprogramját. Az illesztéshez szükség van a `MathLink` programjaira. A példa Windows operációs rendszerre és a Borland cég C++ fordítójának 3.1. változatára készült. A futtatáshoz másoljuk be az MLINK16.DLL programot a Windows `SYSTEM` könyvtárába.

A *Mathematica* és a C nyelv között a kapcsolatot a *sablon* (*template*) hozza létre. Ez egy szöveges állomány (most: `ADDTWO.TM`), amelyet tetszőleges szövegszerkesztővel megírhatunk. Tartalma a példánkban az alábbi:

```
:Begin:
:Function:      addtwo
:Pattern:       AddTwo[i_Integer, j_Integer]
:Arguments:     {i, j}
:ArgumentTypes:{Integer, Integer}
:ReturnType:    Integer
:End:
```

Minden sor kettősponttal kezdődik. A `Begin` és az `End` alapszó jelentése nyilvánvaló, a `Function` alapszó határozza meg a függvény nevét a C programban, a `Pattern` alapszó pedig a *Mathematica*beli illeszkedést, az `Arguments` alapszó a C-beli függvény paramétereit, az `ArgumentTypes` alapszó jelenti a C függvény bemenő paramétereinek típusát *Mathematicá*ban, míg `ReturnType` a C-beli függvény visszatérő értékének típusa. Az állomány kiterjesztése: `.tm`

A C nyelvű programot a szokásos módon kell elkészíteni. A főprogram szerkezetét a használt operációs rendszer határozza meg, ez esetünkben a Windows rendszer. A főprogramot tartalmazó állomány neve: `ADDTWO.C`

```
#include "mathlink.h"

int addtwo(int i, int j)
{return i+j;}
```

```
int PASCAL WinMain(HANDLE hinstCurrent,
HANDLE hinstPrevious, LPSTR lpszCmdLine, int nCmdShow)

{char buff[512];
char FAR  *argv[ 32];
int      argc;

if(!MLInitializeIcon(hinstCurrent, nCmdShow))
return 1;
argc = MLStringToArgv(lpszCmdLine, buff, argv, 32);
return MLMain(argc, argv);}
```

A MathLink lemezén csak az MLINK16.DLL könyvtár található. Ebből létre kell hozni az MLINK16.LIB állományt a BorlandC programhoz tartozó IMPLIB.EXE segédprogrammal. Az így létrehozott MLINK16.LIB állomány már összeszerkeszthető a C nyelvű forrásprogrammal.

A létrehozott sablonból C program készíthető a MathLink-hez adott MPREP.EXE programmal. Ez a sablont paraméterként várja, az eredményt pedig a képernyőre írja, így át kell azt irányítani:

```
MPREP ADDTWO.TM >> ADDTWOTM.C
```

A BorlandC fordítójával összeszerkesztjük (linkeljük) a függvényt tartalmazó (ADDTWO.C) C programot, a sablont tartalmazó (ADDTWOTM.C) C programot és a MathLinket tartalmazó könyvtárat. A szerkesztés eredménye az ADDTWO.EXE program.

A függvényt így indítjuk el *Mathematicá*ból:

```
link1 = Install["d:\wnmath22\mathlink\samples\addtwo.exe"]
LinkObject[d:\wnmath22\mathlink\samples\addtwo.exe, 3, 2]
```

Ezután a *Mathematicá*ban megszokott módon lehet használni:

```
AddTwo[2, 3]
5
```

Végül be kell fejezni a használatot:

```
Uninstall[link1]
Unlink[LinkObject[
   d:\wnmath22\mathlink\samples\addtwo.exe, 2, 2]]
```

• **A** *Mathematica* **függvényeinek meghívása C programból**

Ha a fentiek alapján már elkészítettük az MLINK16.LIB könyvtárat, akkor már csak a C programot kell megírni. Tegyük fel, hogy az ADDINT.C forrásnyelvi állományba az alábbiakat írtuk:

```
#include <stdio.h>
#include <stdlib.h>
#include "mathlink.h"

extern void init_and_openlink P((int argc, char* argv[]));
MLEnvironment ep = (MLEnvironment) 0;
MLINK lp = (MLINK) 0;

int main(int argc, char *argv[])
{int pkt, i, j, sum;
printf("Opening Mathematica Kernel...\n\n");
init_and_openlink(argc, argv);
printf("Two integers:\n\t");
scanf("%d %d", &i, &j);
MLPutFunction(lp, "EvaluatePacket", 1L);
MLPutFunction(lp, "Plus", 2L);
MLPutInteger(lp, i);
MLPutInteger(lp, j);
MLEndPacket(lp);

while((pkt=MLNextPacket(lp), pkt) && pkt != RETURNPKT)
MLNewPacket(lp);
MLGetInteger(lp, &sum);
printf("sum=%d\n", sum);
MLPutFunction(lp, "Exit", 0);
return 0;}
void deinit (void)
   {if (ep) MLDeinitialize(ep);}
void closelink(void)
   {if (lp) MLClose(lp);}
void init_and_openlink(int argc, char *argv[])
   {ep=MLInitialize((MLParametersPointer)0);
   if(ep==(MLEnvironment)0) exit(1);
   atexit(closelink);
   lp = MLOpen(argc, argv);
```

```
if(lp==(MLINK)0) exit(1);
atexit(closelink);}
```

A programot az összeszerkesztés (ez előállítja az ADDINT.EXE programot az ADDINT.C és a MLINK16.LIB programból) után a Windowsból indíthatjuk el

```
-LINKNAME "D:\WNMATH22\MATH-MATHLINK"
```

paraméterrel. A *Mathematicá*ból indítás előtt szálljunk ki, mert ha a mag már aktív, akkor programunk nem tudja még egyszer elindítani. A futáskép:

```
Opening Mathematica Kernel...
Two integers
         2 3
sum = 5
```

A *Mathematicá*ban Fortran és Visual Basic programok hasonlóan használhatók, mint C nyelvű programok.

Mivel bizonyos felhasználói programoknak olyan programnyelvük van, amely közvetlenül képes meghívni a `MathLink` függvényeit (az Accesshez és az Excelhez a Visual Basic, a Wordhöz a WordBasic), ezért ezek különösen könnyen illeszthetők. Néhány népszerű felhasználói programra szintén készen van az illesztés; ezek például LabVIEW, Spyglass Transform, MATLAB, Xmath, IRIS Explorer, AVS.

A *Mathematicá*t az Excel táblázatkezelővel együtt a `MathLink for Excel` program segítségével tudjuk használni. Ennek az lehet az értelme, hogy a táblázatkezelővel esetleg áttekinthetőbb formában tudjuk begyűjteni az adatokat, mintha egy szöveges állományba írnánk azokat.

Végül megmelítjük, hogy az AMLÁZKA (amit látsz, azt kapod, *WYSIWYG*) típusú bevitelt és TEX, LaTEX, Lotus WordPro vagy EPS (`EncapsulatedPostScript`) alakú kivitelt segíti a `MathEdit` és a `MathPORT` program, a K-TALK Communications, Inc. terméke.

2.5. További információforrások

Az ismertetetteken kívül számos további lehetőség van információszerzésre. A teljesség igénye nélkül szeretnénk segíteni az Olvasónak abban, hogy hogyan induljon el ebbe az irányba.

2.5.1. MathSource

A *Mathematica* programmal kapcsolatos legteljesebb ingyenesen hozzáférhető elektronikus adatbázis a MathSource. Itt találhatunk többek között programcsomagokat, a *Mathematicá*ról szóló és az azt felhasználó publikációkat és ilyenek listáit. Az itt szereplő tételekhez egy-egy azonosító tartozik. Ha itt megtalálható anyagra hivatkozunk, akkor ezt az azonosítót is meg fogjuk adni, ahogy eddig is megadtuk.

A világhálózaton (*World Wide Web*) címe a következő:

```
http://www.wri.com/MathSource.html
```

Ha az ftp programmal állományokat szeretnénk onnan átmásolni a saját gépünkre, akkor a

```
mathsource.wri.com
```

gépre jelentkezünk be *anonymous* jelszóval. Ugyanezt a címet használhatjuk a gopher programmal és elektronikus levelezés céljára is.

A MathSource ennek a szakasznak további részében ismertetett dokumentumok mindegyikéről bőséges információt tartalmaz (esetenként a teljes dokumentumot).

A MathSource anyagát időről-időre CD lemezen is megjelentetik.

2.5.2. Könyvek

A program referenciakönyveit [3, 89] már eddig is többször megemlítettük. A *Mathematicá*ról számos bevezető könyv jelent meg, és ugyancsak sok (összesen mintegy 100) mű született egyes (matematikán belüli és kívüli) alkalmazásokról — elsősorban angolul, de ma már németül, franciául és japánul is. Könyvünk irodalomjegyzékében ezek közül néhányat megemlítünk, a MathSource-on pedig a teljes aktuális lista megtekinthető.

2.5.3. Folyóiratok

A *Mathematica* felhasználói közötti információcserét több folyóirat is szolgálja. A *Mathematica Journal* negyedévenként jelenik meg. Tartalmaz kutatási eredményeket, gyakorlati alkalmazásokat, ír új termékekről, könyvekről.

Lemezmellékletén a számhoz csatlakozó jegyzetfüzetek vannak.

A *Mathematica in Education and Research* szintén negyedévenként megjelenő folyóirat a *Mathematica* első változatának kibocsátásakor indult. Kezdetben csak az oktatással (matematika, természettudományok, műszaki tudományok) foglalkozott. A kutatás és ezzel együtt a Research szó újonnan került a címébe.

A *Mathematica World* hajlékony lemezen havonta megjelenő folyóirat. Jegyzetfüzeteket, interaktív tananyagokat tartalmaz a magról, a felhasználói felületről és a programcsomagokról. Az elektronikus levelezési csoportokban felmerült problémák megoldásait is közli.

A negyedévenként megjelenő *MathUser* folyóiratot a Wolfram Research Institute küldi az érdeklődő felhasználónak, ha küld egy „subscribe mathuser" tartalmú elektronikus levelet a

```
mathlist@wri.com
```

címre. Számos apró ötletet, fogást tanulhatunk belőle.

2.5.4. MathGroup

A *Mathematica* fölhasználói megalakították a (többezer tagot számláló) szabályozott (*moderated*) levelezési csoportot, a MathGroup-ot. Ennek a közösségnek a tudását többféleképpen hasznosíthatjuk.

Föltehetünk egyetlen kérdést Steve Christensennek a

```
mathgroup@christensen.cybernetics.net
```

címre, annak a megjegyzésével, hogy a levelezésben nem kívánunk részt venni, ezért a válaszokat a saját címünkre kérjük.

A levelezési csoportba úgy lehet belépni, hogy szintén Steve Christensennek írunk egy levelet a

```
mathgroup-request@christensen.cybernetics.net
```

címre. Válaszul először kapunk egy tájékoztatót, majd megindul a kérdések és válaszok tanulságos és hasznos áradata.

3. Fejezetek a matematikából

Ebben a fejezetben a matematika témakörei köré csoportosítva mutatjuk be a *Mathematica* program lehetőségeit. Fel fogjuk sorolni a felhasználható *belső függvények*et (lásd a 2.1. szakaszt) is, valamint a programcsomagokban (lásd a 2.2. szakaszt) meglévő *külső függvények*et is.

3.1. Alapvető matematikai fogalmak

Célunk ennek a szakasznak az elején az, hogy a felsőfokú matematikai tanulmányok elején szokásosan szereplő logikai és halmazelméleti tudnivalókat illusztráljuk, illetve hogy az e tárgykörben felmerülő feladatok megoldásához segítséget nyújtsunk.

A *Mathematica* alkalmazhatóságának e területen több korlátja is van. Először is: majdnem kizárólag csak *véges* halmazokkal vagy valós számhalmazokkal (elsősorban intervallumokkal) tudunk dolgozni. Másodszor: *kvantorok* alkalmazása csak kivételes esetekben sikerül. (Kivétel: a μ operátor.) Harmadszor: egyelemű halmaz és eleme, illetve a függvény és helyettesítési értéke közötti *különbségtétel* ugyan következetesebb, mint bármelyik programozási nyelvben, mégsem éri el mindenütt azt a szintet, amelyre az egyetemi oktatásban feltétlenül szükség van.

A fenti problémák ellenére meg szeretnénk győzni az olvasót arról, hogy még az itt tárgyalt területeken is segítségére lehet a *Mathematica*. Mivel maga a program az alább bemutatandóknál jóval bonyolultabb logikai műveleteket és levezetési szabályokat alkalmaz, ezért az is nyilvánvaló, hogy szorgos gyakorlással fegyvertárunk szinte korlátlanul bővíthető.

3.1.1. Logikai műveletek

A következő belső függvényeket használhatjuk:

`Alternatives (|)`	`Or (||)`
`And (&&)`	`LogicalExpand`
`Equal (==)`	`True`
`False`	`TrueQ`
`Implies`	`Unequal (!=)`
`Not (!)`	`Xor`

A *Mathematica* alkalmas logikai értelemben vett *ítélet*ek kezelésére. Ezek kiértékelésénél azt vizsgálja, hogy igazak-e, avagy hamisak. Jelöljük például p-vel a „$3 < 2$” kijelentést:

```
p = (3<2)

False
```

A p állítás tagadásának igazságértéke az ellenkezője lesz:

```
Not[p]

True
```

Ez a `Not` függvény pontosan megfelel a logikában szereplő *negáció* elnevezésű egyváltozós logikai függvénynek. Azt is mondhatjuk, hogy a `False` és `True` logikai értékek pontosan megfelelnek az *igaz* és *hamis* nullaváltozós logikai függvényeknek, azaz állandóknak. A kétváltozós matematikai logikai függvények közül a logikai konjunkció ($\wedge$) megfelelője az `And`, a diszjunkcióé ($\vee$) az `Or`, míg a kizáró vagyé a `Xor`. Az ekvivalenciának az `Equal` függvény felel meg, az implikációnak pedig az `Implies`.

Megjegyezzük, hogy a matematikai logikában néha nem, vagy nem csak logikai értékekhez logikai értékeket rendelő függvényeket szokás logikai függvényeknek nevezni, hanem olyan függvényeket, amelyek valamely más halmazon, például a természetes számok halmazán vannak definiálva, és a természetes számokhoz rendelnek logikai értékeket.

Az elveknél leírtakkal összhangban ezek a *Mathematica*-függvények általánosabbak, „többet tudnak” a megfelelő matematikai függvényeknél, az `Equal` függvény például nemcsak logikai, hanem számértékek összehasonlítására is használható.

• Érték- vagy igazságtáblázatok

Megvizsgáljuk, hogy a változók összes lehetséges értéke mellett milyen értékeket szolgáltatnak a program logikai függvényei:

```
{Not[True], Not[False]}
{False, True}
```

Jó lenne (különösen a későbbiekre is gondolva), ha a lehetséges igazságértékek:

```
igaz1 = {True, False};
```

halmazára alkalmazhatnánk a vizsgált függvényt. Ehhez hívjuk segítségül a `Map` függvényt, amelynek használata az alábbi egyszerű példából világos lesz:

```
Map[Not, {True, False}]
{False, True}
```

Talán még természetesebb az az írásmód, amelynél a függvény és a lista (vagy halmaz; a későbbiekben ezek viszonyát tisztázni fogjuk) közé mindössze a `/@` jelet írjuk, amely a `Map` függvény rövidített alakja, s amely arra figyelmeztet bennünket, hogy a `Not` függvényt közvetlenül nem alkalmazhatjuk egy listára, csak annak elemeire:

```
Not /@ {True, False}
{False, True}
```

Esetenként kényelmesebb lehet, ha úgy módosítunk egy függvényt, hogy közvetlenül alkalmazható legyen egy listára: az attribútumai között szerepeljen a `Listable` attribútum. Ennek a módja a következő:

```
Unprotect[Not];
SetAttributes[Not, Listable];
Protect[Not];
Not[{True, False}]
{False, True}
```

Térjünk át most az `And` kétváltozós függvényre. Az `And` függvény *értéktáblázat*ának vagy *igazságtáblázat*ának egy elemét például így kaphatjuk meg:

```
And[True, True]
True
```

Vezessük be az igazságértékekből álló párok halmazát:

```
igaz2 = {{True, True}, {True, False},
{False, True}, {False, False}};
```

Elegendő tehát végigmenni az `igaz2` lista elemein és mindegyiket behelyettesíteni az `And` függvénybe. Az `igaz2` lista elemei viszont maguk is listák, tehát gondoskodnunk kell arról, hogy ezek a kételemű listák szolgáltassák az `And` függvény két argumentumát („zárójelek nélkül"). A helyzet ahhoz hasonló, mint amikor matematikában egy kétdimenziós vektort beírunk egy kétváltozós függvény argumentumába. Ott el szoktuk követni azt a pontatlanságot, hogy egy zárójelpárt elhagyunk és $f(0,1)$-et írunk $f((0,1))$ — sőt $f(\binom{0}{1})$ — helyett. Itt ezt nem tehetjük meg, ezért szükségünk van az `Apply` függvényre, amely egy lista elemeiből egy többváltozós függvény argumentumait állítja elő:

```
Apply[And, igaz2[[1]] ]
True
```

(Itt `igaz2[[1]]` az `igaz2` lista első eleme, vagyis a `{True, True}` lista.) Az `Apply` függvénynek is van rövid alakja, a fentivel egyenértékű eredményt kapunk tehát így is:

```
And @@ igaz2[[1]];
```

Az értéktáblázat azon részét tehát, amely az utolsó oszlopot szokta alkotni, egy táblázat formájában állíthatjuk elő:

```
utolsooszlop = Table[Apply[And, igaz2[[i]]], {i, 1, 4}]
{True, False, False, False}
```

Jelölje most

```
f[x_] := Apply[And, x]
```

azt a függvényt, amely az `x` lista elemeit, amelyek maguk is listák, az `And` függvény argumentumaivá alakítja. A fenti eredményt ezek után úgy kaphatjuk meg, hogy kiszámítjuk az `igaz2` halmaz f melletti képét:

```
Map[f, igaz2]
{True, False, False, False}
```

Tiszta függvény (lásd alább) alkalmazásával f kiiktatható, ekkor így állíthatjuk elő `And` igazságtáblázatának utolsó oszlopát:

```
Map[Apply[And, #]&, igaz2]
{True, False, False, False}
```

Vagy rövidebben:

```
Apply[And, #]& /@ igaz2
{True, False, False, False}
```

és végül:

```
And @@ #& /@ igaz2
{True, False, False, False}
```

Lehet, hogy ez alkalommal nem túl gyorsan jutottunk el a végeredményhez, az eljárás előnyeit mégis látnunk kell, hogy a továbbiakban magunk is törekedjünk a fentihez hasonló stílusú írásmódra. A végső képletben lényegében csak a logikai művelet szerepel és annak értelmezési tartománya, valamint figyelmeztető jelek annak jelzésére, hogy különbséget kell tennünk valamely halmaz és annak elemei között. Az esztétikai szempont mellett — amelynek figyelembevétele csak némi gyakorlat után várható el a felhasználótól — igen lényeges az is, hogy a műveleteket a program sokkal gyorsabban végzi el, ha a feladatot ilyen módon adjuk meg, mintha hagyományos írásmódot alkalmaznánk. Ezt a teljesítőképesség vizsgálatánál példákon fogjuk bizonyítani.

Egy tetszőleges kétváltozós logikai művelet értéktáblázatát tehát az alábbi függvény állítja elő:

```
tabla[muvelet_] := (muvelet @@ #&) /@ igaz2
```

Használjuk most ezt a függvényt!

```
tabla[And]
tabla[Or]
tabla[Xor]
tabla[Equal]
tabla[Unequal]
tabla[Implies]
```

Az értéktáblázatot a szokásos módon is elhelyezhetjük. Ehhez egészítsük ki az értelmezési tartományt az argumentumokat jelölő két betűvel, például a-val és b-vel:

```
dom = Prepend[igaz2, {a, b}]
{{a, b}, {True, True}, {True, False},
 {False, True}, {False, False}}
```

Ezután az eredményt tartalmazó lista elejére írjuk oda az illető logikai függvény helyettesítési értékét az (a, b) helyen:

```
ran = Prepend[tabla[Or], HoldForm[a V b]]

{a V b, True, True, True, False}
```

(`Prepend[lista, elem]` a `lista` elejére odailleszti az `elem`-et. `HoldForm` pedig itt azt a célt szolgálja, hogy ne értékelődjék ki az $a \vee b$ „összeg".) Amint az a mátrixműveleteknél részletesen ki fog derülni, a `dom` 2×5-ös és a `ran` 1×5-ös mátrix így tehető egymás mellé:

```
Transpose[Join[Transpose[dom], {ran}]]

{{a, b, a V b}, {True, True, True}, {True, False, True},
{False, True, True}, {False, False, False}}
```

Hogy végül egy szép táblázatot kapjunk, nézzük meg az eredményt mátrixalakban:

```
MatrixForm[%]

  a      b     a V b
True   True   True
True   False  True
False  True   True
False  False  False
```

Haladók számára tanulságos megfontolni, hogy nem sokkal könnyítené meg a dolgunkat a `LinearAlgebra`MatrixManipulation`` csomagból vett `AppendColumns` függvény sem, mivel `ran` sorai egyetlen elemből állnak és nem listák (vagy, ha tetszik, vektorok). A `TableForm` függvény viszont rendelkezik olyan opciókkal, amelyek segítségével a feladat könnyebben megoldható lett volna; javasoljuk az olvasónak, hogy keressen meg egy ilyen megoldást.

• Logikai azonosságok

Térjünk át most a logikai azonosságok vizsgálatára. Szeretnénk egyszerű logikai azonosságokat bizonyítani a program felhasználásával.

A konjunkció kommutativitása így látható be:

```
LogicalExpand[a && b] == LogicalExpand[b && a]

True
```

Általában is igaz az, hogy bonyolultabb logikai kifejezések egységes alakra hozásában segít a `LogicalExpand` függvény: ez ugyanis előállítja egy formula *teljes diszjunktív normálformáját*. Innen látszik például a *De Morgan-féle azonosságok* érvényessége.

```
LogicalExpand[!(!a || !b)]
a && b

LogicalExpand[!(!a && !b)]
a || b
```

Abban az esetben, ha az egyszerű módszerek nem segítenek, marad az értéktáblázatok összehasonlításának módszere. Lássuk be például a korábban bevezetett `tabla` függvény alkalmazásával, hogy a diszjunkció kommutatív:

```
tabla[(#1 || #2)&] == tabla[(#2 || #1)&]
True
```

Az implikációval kapcsolatban érdemes megjegyezni, hogy sokkal többre használható, mint matematikai logikai ismereteink alapján remélnénk. `Implies[egy1,egy2]` ugyanis jelenti az `egy1` egyenlet azon részét, amely tartalmazza `egy2`-t. Ez egyenletek megoldásánál jól alkalmazható, lásd ott.

Az implikációról zárásul megemlítjük, hogy az `If` függvény *feltételes utasítások* megvalósítására szolgál, tehát a programozás egy segédeszköze. Logikai szempontból érdekes megjegyezni, hogy négy argumentummal is használható; ilyenkor a negyedik argumentumban szereplő utasítást hajtja végre a program, amennyiben nem tudja eldönteni, hogy igaz vagy hamis a feltétel értéke.

• Logikai egyenletek

Oldjuk meg az alábbi logikai egyenleteket a *Mathematica* segítségével:

```
Solve[(False || x) == False, x]
{{x -> False}}
```

Az eredmény *hozzárendelési szabály*ok egy (egyelemű) listája. Az egyetlen hozzárendelési szabály azt mondja, hogy a megoldást kapjuk meg akkor, ha x értékét `False`-sal helyettesítjük. Ehhez hasonlót jelent az alábbi utasítás és eredménye is:

```
Solve[(False || x) == True, x]
{{x -> True}}
```

Ha megoldással nem rendelkező egyenletet oldatunk meg a programmal, akkor eredményül üres listát kapunk:

```
Solve[(True || x) == False, x]
{}
```

Végül pedig, ha minden x megoldása egy egyenletnek, azaz nincs megkötés arra, hogy mit helyettesítsünk x helyébe, akkor az eredmény egyetlen szabályból álló lista, amely szabály azonban nem tartalmaz megkötést:

```
Solve[(True || x) == True, x]
{{}}
```

Oldjuk meg most a következő feladatot (lásd [83], 125. oldal). „Megyek az utcán, észreveszek a túlsó oldalon egy lányt és egy fiút. Még messze vannak, nem ismerem fel egyiket sem; de mintha Zsuzsi volna az a lány, állapítom meg magamban.
Ha Zsuzsi az, akkor vagy Feri megy vele, vagy az a mérnökhallgató, akivel mostanában gyakran látom együtt.
Közelebb jönnek. Felismertem: Zsuzsi az.
Most már a fiút is jobban látom. Nem Feri megy vele.
Mi következik ebből?"

Ha felírjuk a feltevéseket (premisszákat) rövidítve, logikai műveletekkel, akkor ezt kapjuk:

```
zsuzsi = True;
feri = False;
Solve[Implies[zsuzsi, feri || mernok] == True, mernok]
{{mernok -> True}}
```

• A μ operátor

A matematikai logikában és a rekurzív függvények elméletében szerepel a μ operátor: szokásosan $\mu_{n\in\mathbf{N}}P(n)$ jelöli azt a legkisebb természetes számot, amelyre a P állítás igaz. A `Select` függvénnyel ez (a természetes számok halmaza helyett véges halmazt szerepeltetve) leírható, sőt a `Select` függvény ennél sokkal többre is képes:

```
mu1 = Select[{a, b, c, d, 5, e, 2}, NumberQ, 1]
{5}
```

Most még nem a legkisebb, adott tulajdonságú elemet kaptuk, hanem a listában legelőször következőt. Egy rendezéssel együtt már a jó megoldás adódik:

```
mu2 = Select[Sort[{a, b, c, d, 5, e, 2}], NumberQ, 1]
{2}
```

Egy másik megoldás:

```
mu3 = Min[Select[{a, b, c, d, 5, e, 2}, NumberQ]]
2
```

Hasonlítsuk össze a két megoldás gyorsaságát nagyobb számosságú halmazokon!

3.1.2. Halmazok

Szükségünk lesz a legtöbb listakezelő függvényre is a néhány speciálisan halmazelméleti függvény mellett:

Apply	MapAll
Cases	MapAt
Complement	MapIndexed
Drop	MapThread
Flatten	MemberQ
FreeQ	Outer
Intersection	Part
Interval	Partition
List	Rest
Map	Union

A `DiscreteMath`Combinatorica`` programcsomag

CartesianProduct	RandomPartition
GrayCode	RandomSubset
KSubset	Subsets
NthSubset	

függvényei, valamint a `Calculus`DiracDelta`` programcsomag

DiracDelta	UnitStep
SimplifyUnitStep	

eljárásai is segítséget nyújthatnak.

• Alapvető fogalmak

A *Mathematica* alapvető adatszerkezete a *lista.* Halmazokat is listával reprezentálhatunk, ehhez azonban az adott listában többször előforduló azonos elemeket egyetlen listaelemmel kell helyettesítenünk. Ennek egy módjára mutatunk példát:

```
lis = {3, 1, 1, 2, 3, 1, 2};
halm = Union[lis]
```

(Eközben az elemek a `$StringOrder` rendszerváltozó aktuális értékének megfelelően rendeződnek is.)

A legcélszerűbb eljárás, ha bevezetünk egy függvényt, amely a listákat halmazzá alakítja, és halmazműveletek végzése előtt első lépésként mindig ezt alkalmazzuk:

```
halmaz[lista_] := Union[lista]
```

Az *üres halmaz* megfelelője az { } üres lista. Nem lep meg bennünket, hogy

```
{ } == {{ }}
False
```

Azt, hogy az a elem *benne van*-e az $\{a, b, c\}$ halmazban, például így vizsgálhatjuk meg:

```
MemberQ[{a,b,c},a]
True
```

Megkérdezhetjük azt is, hogy mentes-e az $\{a, b, c\}$ halmaz az a elemtől:

```
FreeQ[{a,b,c},a]
False
```

A `MemberQ` utasítás felhasználásával ellenőrizhetjük azt, hogy egy halmaz *részhalmaz*a-e a másiknak:

```
ClearAll[a,b];
a = {1,2,3};
b = {0,1,2,3,5};
Apply[And,Table[MemberQ[b,a[[i]]],{i,3}]]
True
```

Rövidebb, tömörebb és gyorsabb, ha az a halmazt nem elemenként kezeljük:

```
Apply[And, Map[MemberQ[b, #]&, a]]
```

```
True
```

Még rövidebb formulát kapunk, ha az utasítások rövid alakját használjuk:

```
resze[x_, y_] := And @@ (MemberQ[y, #]& /@ x);
resze[a, b]
```

```
True
```

*Részhalmaz*okat gyakran definiálunk úgy, hogy egy adott alaphalmaz *adott tulajdonságú* elemeit választjuk ki. Ezt többféleképpen is megtehetjük a *Mathematicá*ban:

```
halom = {a, b, c, 1, 2, 3, 4, 5};
Cases[halom, _Integer]
```

```
{1, 2, 3, 4, 5}
```

```
Cases[halom, x_ /; x<3]
```

```
{1, 2}
```

```
Select[halom, OddQ]
```

```
{1, 3, 5}
```

Kiválaszthatjuk egy halmaz (inkább: egy lista) *adott helyen lévő* elemeit is, amint azt a példák mutatják (ezek a műveletek eszközként halmazokra is jól használhatóak):

```
Take[halom, 3]
```

```
{a, b, c}
```

```
Take[halom, -3]
```

```
{3, 4, 5}
```

```
Take[halom,{3, 5}]
```

```
{c, 1, 2}
```

Hasonlóan *hagyunk el* bizonyos elemeket:

```
Drop[halom, 3]
Drop[halom, -3]
```

```
{1, 2, 3, 4, 5}
{a, b, c, 1, 2}
```

```
Drop[halom, {3}]
Drop[halom, {3,5}]
{a, b, 1, 2, 3, 4, 5}
{a, b, 3, 4, 5}
```

Adott halmaz *részhalmazát* így is megadhatjuk:

```
Rest[halom]
{b, c, 1, 2, 3, 4, 5}

Part[halom, {3, 5, 7}]
{c, 2, 4}
```

A következő utasítások eredménye pedig adott halmaz egy *eleme*:

```
First[halom]
a

Part[halom, 3]
c
```

Felhívjuk a figyelmet az elvekről szóló 2.3. szakaszra. A jelen pontban tárgyalt műveletek ugyanis általában nem csak listákra alkalmazhatók, így hatáskörük az itt ismertetettnél sokkal bővebb.

• Halmazműveletek

Természetesen nemcsak a halmaz listából való előállításának műveletét, hanem a halmazok *egyesítés*ét is a `Union` függvény végzi:

```
Union[{1, 2, a}, {2, 3, a, b}, {1, a, 2, c}]
{1, 2, 3, a, b, c}
```

Ugyanezt használhatjuk — többféleképpen is — tetszőleges (véges!) halmazrendszer egyesítésének kiszámítására is:

```
rendszer = Table[{i-1, i, i+1},{i, -1, 1}]
{{-2, -1, 0}, {-1, 0, 1}, {0, 1, 2}}

Union @@ rendszer
{-2, -1, 0, 1, 2}

Union[Flatten[rendszer]]
{-2, -1, 0, 1, 2}
```

(`Flatten` „kiszedi” a fölösleges zárójeleket, részletesen lásd az elveknél.)
Halmazok *metszet*ének kiszámítására adunk néhány példát:

```
{a, b, c}~Intersection~{b, c, d}
Intersection[{a, b, c}, {b, c, d}]
Intersection @@ rendszer
Apply[Intersection, rendszer]
```

Végtelen ($\mathbf{R}^N$-beli) halmazok például *karakterisztikus függvény*ükkel kezelhetők. A nyílt egységkörlap karakterisztikus függvényét így is megkaphatjuk:

```
ClearAll[f];
f[x_, y_] := If[1-x^2-y^2>0, 1, 0];
Timing[Plot3D[f[x, y], {x, 0, 1}, {y, 0, 1}]]
{46.192 Second, -SurfaceGraphics-}
```

Ugyanennek a függvénynek egy másik definíciója:

```
g[x_, y_] := 1 /; (x^2+y^2)<1
g[x_, y_] := 0 /; (x^2+y^2>=1)
Timing[Plot3D[g[x, y], {x, 0, 1}, {y, 0, 1}]]
{40.645 Second, -SurfaceGraphics-}
```

(Ezeket a definíciókat — és az alább következőt — ismét felfoghatjuk úgy is, hogy egy alaphalmaz, esetünkben a sík, adott tulajdonságú elemeit választjuk ki.)

A `Calculus` alkönyvtár `DiracDelta` elnevezésű programcsomagját is használhatjuk karakterisztikus függvények elkészítéséhez:

```
<<Calculus`DiracDelta`
h[x_, y_] := UnitStep[1-x^2-y^2];
```

Ha ezeket a karakterisztikus függvényeket ábrázolni akarjuk, akkor ez az első két definícióval körülbelül egyforma ideig tart, a harmadikkal körülbelül háromszor annyi ideig. A harmadikat akkor érdemes mégis választanunk, ha a karakterisztikus függvényekkel műveleteket is akarunk végezni: például a halmazműveletek megvalósítása végett. (Két halmaz metszetének karakterisztikus függvénye a halmazok karakterisztikus függvényének szorzata, egyesítésükét pedig úgy kapjuk meg, hogy a karakterisztikus függvények összegéből levonjuk azok szorzatát. Valamely halmaz komplementerének karakterisztikus függvénye az 1 függvény és a karakterisztikus függvény különbségeként kapható meg.) A bonyolult, a `UnitStep` függvényt több helyen is tartalmazó kifejezések egyszerűsítésére ugyanis az adott programcsomagban egy külön függvény, a `SimplifyUnitStep` található.

Számítsuk ki most két halmaz metszetének és egyesítésének karakterisztikus függvényét:

```
k[x_, y_] := UnitStep[x+y-1]
l[x_, y_] := SimplifyUnitStep[h[x, y] k[x, y]]
m[x_, y_] := SimplifyUnitStep[h[x, y] + k[x, y] - l[x, y]]
Plot3D[l[x, y], {x, 0, 1}, {y, 0, 1}]
```

Az a és b halmaz *Descartes-féle szorzatát* az `Outer` függvénnyel kaphatjuk meg, ez ugyanis a második és harmadik argumentumában megadott listák elemeiből képezi az összes párt, és ezeket a párokat az első argumentumként megadott függvény argumentumának adja át; végül az eredményből listát képez:

```
Flatten[Outer[List, {d, e, f}, {1, 2}], 1]
{{d, 1}, {d, 2}, {e, 1}, {e, 2}, {f, 1}, {f, 2}}
```

Érdemes tehát a következő definíciót bevezetni:

```
descartes[a_, b_] := Flatten[Outer[List, a, b], 1]
a = {d, e, f};
b = {1, 2};
descartes[a, b]
{{d, 1}, {d, 2}, {e, 1}, {e, 2}, {f, 1}, {f, 2}}
```

(A Descartes-szorzat elemei *rendezett párok*, ahogy el is várjuk.) Ugyanezt megadja a `DiscreteMath` alkönyvtár `Combinatorica` programcsomagjának `CartesianProduct` függvénye is.

Az értéktáblázatokhoz a logikai függvények értelmezési tartományának pontjait most már így is előállíthatjuk:

```
igaz = descartes[{True, False}, {True, False}];
```

Descartes-szorzat (nem csak véges halmazoké!) számos belső függvény argumentumaként előfordulhat. Például:

```
Plot3D[Sin[x^2 + y^3], {x, -Pi/2, Pi/2}, {y, 0, Pi}]
```

Ha a k szám osztója az A halmaz elemei számának, akkor

```
Partition[A,k]
```

az A halmaz egy olyan partícióját szolgáltatja, amelyben minden részhalmaznak k számú eleme van:

```
Partition[{1, 2, 3, 4, 9, 8, 7, 6}, 2]
{{1, 2}, {3, 4}, {9, 8}, {7, 6}}
```

Ne tévesszük össze a `Partition` függvényt a `DiscreteMath` alkönyvtár `Combinatorica` programcsomagjának `Partitions` függvényével, amely pozitív egész számokat bont fel pozitív egész számok összegére.

• Hatványhalmaz

Hívjuk be a `Combinatorica` programcsomagot:

```
<<DiscreteMath`Combinatorica`
```

Képezhetjük egy adott halmaz részhalmazait lexikografikus sorrendben:

```
Table[NthSubset[n, {a, b, c, d}], {n, 0, 15}]
{{ }, {a}, {b}, {a, b}, {c}, {a, c}, {b, c}, {a, b, c},
{d}, {a, d}, {b, d}, {a, b, d}, {c, d}, {a, c, d},
{b, c, d}, {a, b, c, d}}
```

Hatványhalmazt tehát így képezhetünk:

```
hatvany[a_] := Table[NthSubset[n, a], {n, 0, 2^Length[a]-1}];
hatvany[{a, b, c, d}];
```

Egy másik megoldásnál úgy soroljuk föl a részhalmazokat, hogy a sorozat minden tagja pontosan egy elemben különbözzék az előzőtől:

```
GrayCode[{1, 2, 3, 4}]
{{}, {1}, {1, 2}, {2}, {2, 3}, {1, 2, 3}, {1, 3}, {3},
{3, 4}, {1, 3, 4}, {1, 2, 3, 4}, {2, 3, 4}, {2, 4},
{1, 2, 4}, {1, 4}, {4}}
```

Vegyük észre, hogy például a 4 szám az első nyolc részhalmazban nem fordul elő, míg az utolsó nyolc mindegyikében előfordul.

Az a halmaz k elemű részhalmazait így kapjuk meg:

```
KSubsets[{a, b, c, d}, 3]
{{a, b, c}, {a, b, d}, {a, c, d}, {b, c, d}}
```

ezért hatványhalmazt így is előállíthatunk:

```
Flatten[Table[KSubsets[{a, b, c, d}, n], {n, 0, 4}], 1]
{{ }, {a}, {b}, {c}, {d}, {a, b}, {a, c}, {a, d},
{b, c}, {b, d}, {c, d}, {a, b, c}, {a, b, d}, {a, c, d},
{b, c, d}, {a, b, c, d}}
```

Végül megemlítjük, hogy ugyanebben a programcsomagban `Subsets` néven olyan függvényt is találunk, amely közvetlenül előállítja a hatványhalmazt.

Megjegyzendő, hogy gyakran van szükségünk arra, hogy előállítsuk egy halmaz kételemű részhalmazait (azaz a másodosztályú ismétlés nélküli kombinációkat). Ehhez is felhasználhatjuk a `KSubsets` függvényt:

```
parok[lista_] := KSubsets[lista, 2];
```

• Relációk

A valós számokon definiált szokásos *relációk* is használhatók: `<`, `<=`, `>`, `>=`. A matematikában megszokott összevonásokkal is élhetünk:

```
3 < 5 < 12
```

```
True
```

Nem csak számokra használható az egyenlőség és a nemegyenlőség:

```
(3<4) == True
```

```
True
```

```
{1, 2, 5, a, b} != {1, 2, 5, a, b}
```

```
False
```

```
{1, 2, 5, a, b} == {2, 1, 5, a, b}
```

```
False
```

Az `x!=y!=z` kifejezés pontosan akkor igaz, ha x, y és z páronként különbözők. (Így tehát a `Unequal` és a `Xor` háromváltozós függvény már különbözők, bár kétváltozósként, mint fönt láttuk, megegyeznek.)

A fenti relációk és logikai műveletek eredményként is igen gyakran előfordulnak:

```
Reduce[a x == b, x]
```

```
b == 0 && a == 0 || a != 0 && x == b/a
```

(A logikai diszjunkciótól megkülönböztetendő az alternatívák felsorolását jelölő | jel; ezt a mintázatokkal kapcsolatban a 2.3.2. pontban tárgyaltuk.)

Újonnan definiált relációk megadásához bizonyos szorzathalmazok részhalmazait kell kijelölni:

```
rel = {{1, 1}, {2, 2}, {3, 3}, {1, 2}, {2, 1}}
```

```
{{1, 1}, {2, 2}, {3, 3}, {1, 2}, {2, 1}}
```

Határozzuk meg a fenti reláció értelmezési tartományát:

```
reldom = Union[First[Transpose[rel]]]
{1, 2, 3}
```

Határozzuk meg értékkészletét is:

```
relran = Union[Last[Transpose[rel]]]
{1, 2, 3}
```

Vajon megegyezik-e a reláció értelmezési tartománya és értékkészlete?

```
reldom == relran
True
```

Egy reláció *reflexivitása* ellenőrizhető úgy, hogy az alaphalmaz identitásfüggvényét tartalmazza-e:

```
atlo = Thread[List[reldom, reldom]];
resze[atlo, rel]
True
```

A relációkat *gráf*oknak is tekinthetjük. Ehhez a felfogáshoz tartozó fogalmakat és eszközöket a diszkrét matematikáról szóló 3.6. szakaszban fogunk tárgyalni.

• Műveletek intervallumokkal

Az intervallumokkal mintha baj lenne. Az `Intersection` és a `Complement` függvénnyel nem működnek jól együtt; a `Union`-nal természetesen igen:

```
Intersection[Interval[{0, 1}], Interval[{0.5, 2}]]
Interval[ ]

Complement[Interval[{0, 5}], Interval[{2, 3}]]
Interval[{0, 5}]

Union[Interval[{0, 1}], Intetrval[{0.5, 2}]
Interval[{0, 2.}]
```

Mindez nincs összhangban a matematikában megszokott tulajdonságokkal. A *Mathematica* elveinek viszont megfelel, ugyanis az `Intersection` úgy működik, hogy a metszet tagjainak `Interval` fejét eltávolítja, majd megállapítja, hogy a megmaradt listáknak melyek a közös elemei, s ezekre visszarakja az `Interval` fejet. Miután ez a kellemetlenség a program használata során kiderült, speciális függvényeket vezettek be intervallumokkal végzendő halmazműveletekre:

```
IntervalUnion[Interval[{0, 1}], Interval[{0.5, 2}]]
IntervalUnion[Interval[{-1, 1}], Interval[{1, 3}]]
IntervalUnion[Interval[{0, 1}], Interval[{2, 3}]]
Interval[{0, 2}]
Interval[{-1, 3}]
Interval[{0, 1}, {2, 3}]

IntervalIntersection[Interval[{0, 1}], Interval[{0.5, 2}]]
IntervalIntersection[Interval[{0, 1}], Interval[{2, 3}]]
Interval[{0.5, 1.}]
Interval[ ]
```

Azt is megállapíthatjuk, hogy egy szám eleme-e egy intervallumnak:

```
IntervalMemberQ[Interval[{1, 2}], Sqrt[2]]
True
```

`IntervalMemberQ[i1, i2]` értéke pedig — ha i_1 és i_2 is intervallum — pontosan akkor `True`, ha $i_2 \subset i_1$:

```
IntervalMemberQ[Interval[{0, 4}], Interval[{2, 3}]]
True

IntervalMemberQ[Interval[{1, 2}], Interval[{0.5, 3}]]
False
```

Valós intervallumokon aritmetikai műveleteket is végezhetünk a szokásos módon:

```
3/7 + Interval[{0, 1}]
```

$$\text{Interval}\left[\left\{\frac{3}{7}, \frac{10}{7}\right\}\right]$$

```
3*Interval[{-1, 3}]
Interval[{-3, 9}]

Interval[{1, 2}] + Interval[{3, 4}]
Interval[{4, 6}]
```

3.1.3. Számok ábrázolása. Aritmetikai műveletek

A *Mathematica* programon belül *elvileg* akárhány jegyű számokkal végezhetünk műveleteket. A gyakorlatban az ábrázolt számjegyek maximális számát — ami minden változatnál ezrekkel mérhető — azonban az aktuálisan használt hardverkörnyezet határozza meg.

Egész számokkal és törtekkel megadott aritmetikai műveletek eredményét a program a matematika szabályainak megfelelően pontosan adja meg (*pontos aritmetika*).

A számológépekhez és a hagyományos programnyelvekhez hasonlóan itt is dolgozhatunk közelítő értékekkel (*közelítő aritmetika*). Ilyenkor igen tág határok között szabályozhatjuk a felhasznált értékes számjegyek számát. Néhány programcsomagban levő függvény a lebegőpontos aritmetika elméletének tanulmányozásához is segítséget nyújt. Ezeket ennek a pontnak a végén ismertetjük.

Már itt felhívjuk az Olvasó figyelmét a bemenő adatok szintaxisának (lásd később) gondos megválasztására. A továbbiakban részletesen ismertetjük, hogy adott kifejezés kiértékeléséhez a *Mathematica* milyen elvek alapján választ a pontos és a közelítő algoritmusok alkalmazása közül.

• Az aritmetikai műveletek szintaxisa

`Plus[x,y]`	vagy	`x+y`	összeadás
`Subtract[x,y]`	vagy	`x-y`	kivonás
`Times[x, y]` vagy `x y`	vagy	`x*y`	szorzás
`Divide[x,y]`	vagy	`x/y`	osztás
`Power[x,y]`	vagy	`x^y`	hatványozás

Az `x*y` szorzatot megadhatjuk így is: `x y`. Ebben az esetben a szóköz karakter jelöli a szorzás műveletét. Ha egy kéttényezős szorzat első tényezője (valós vagy komplex) szám, a második tényezője szimbólum (azaz nem szám), akkor a `*` is és a szóköz karakter is elhagyható. Így a `7x`, a `7*x` és a `7 x` utasítások egyenértékűek. Vigyázzunk azonban arra, hogy a fordított sorrendre ez már nem érvényes. Az `x7` egy `x`-től különböző szimbólumot jelent és nem az `x*7` szorzatot.

A *Mathematica* az aritmetikai műveleteket a matematikában megszokott precedenciaszabálynak megfelelően hajtja végre, ha a `( )` zárójelekkel ettől eltérő sorrendet nem jelölünk ki. A precedenciaszabály 3 oldalas ismertetése Wolfram könyvében ([89], 716–719. oldal) szükség esetén megtalálható. Mi-

vel elég természetes a műveletek sorrendje, ritkán kell ezzel foglalkoznunk. Ha bizonytalanok vagyunk, alkalmazzunk (gömbölyű) zárójeleket.

• **Számtípusok**

A hagyományos programnyelvekhez (Basic, C, Fortran, Pascal) hasonlóan a különböző típusú számokat a *Mathematica* is különböző módon kezeli. Az alábbi számtípusok megkülönböztetésére van mód:

Számtípus	**Definíció**	**Példák**
egész számok `(Integer)`	az egész számok halmazának (**Z**-nek) az elemei	`143` `-245`
valódi törtek `(Rational)`	a p/q alakú számok, ahol $p \in \mathbf{Z}$ és $q \in \mathbf{Z} \setminus \{0\}$	`3/17` `5/(-7)`
valós számok `(Real)`	a valós számok közelítésére szolgáló véges tizedestörtek	`31.423` `-6.792`

komplex számok
`(Complex)`
	az $a + bi$ alakban megadott komplex számok, ahol a és b az előző három számtípus valamelyike (i jelöli a képzetes egységet)	`3+4*I`
		`2/3+7.*I`
		`1.2+3.4*I`

A program szempontjából az egész számoknak és az explicit tizedesponttal megadott valós számoknak kitüntetett szerepük van (lásd a 2.3.1. pontot).

A szimbolikus programcsomagoknak, így a *Mathematica*nak is egyik fő erénye az, hogy segítségükkel alapértelmezésben dolgozhatunk akár több ezer jegyű számokkal is.

A lehetőségek ismertetéséhez szükségünk lesz a számok egy másik csoportosítására is. *Pontos numerikus értékek*nek nevezzük az egész számokat, a valódi törteket és azokat az algebrai alakban megadott komplex számokat, amelyeknek a valós és a képzetes része is az előbbiek valamelyike. Ilyen számokkal a matematikában megszokott módon *pontosan* is végezhetünk műveleteket. *Közelítő numerikus értékek*nek nevezzük az explicit tizedesponttal megadott valós vagy komplex számokat. Az ezekkel végzett műveleteknél a program a szokásos kerekítési szabályokat alkalmazza.

A *Mathematica* szimbólumoknak tekinti (és speciális módon kezeli) a *matematikai állandók*at. Sok esetben ezekkel is ugyanúgy számolhatunk, mint a pontos numerikus értékekkel. A következő táblázatban a beépített matematikai állandók közül csak néhányat sorolunk fel.

`Catalan`	a Catalan-féle állandó, azaz a $\sum_{k=0}^{\infty}(-1)^k(2k+1)^{-2}$ összeg
`Degree`	a fokot radiánba átváltó szorzó ($\pi/180^o$)
`E`	a természetes alapú logaritmus alapszáma
`EulerGamma`	a $\gamma := \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \log n\right)$ Euler-féle állandó
`GoldenRatio`	az aranymetszési állandó (az $(1+\sqrt{5})/2$ szám)
`I`	a $\sqrt{-1}$ képzetes egység
`Infinity`	a $+\infty$ szimbólum
`-Infinity`	a $-\infty$ szimbólum
`Pi`	a π szám

Komplex számok kezeléséhez az alábbi belső függvényeket használhatjuk:

```
Abs          Arg          Conjugate
Im           Re
```

Ezek jelentése a megszokott. Csupán azt emeljük ki, hogy `Arg[z]` a `z` komplex számnak a $(-\pi, \pi]$ intervallumba eső (radiánban mért) argumentumát adja meg. A következő példák ezt illusztrálják. Figyeljük meg azt is, hogy abban az esetben, amikor a bemenő adatok csak pontos numerikus értékeket tartalmaznak, akkor a program a pontos eredményt keresi:

```
{Arg[1 + I], Arg[-Cos[Pi/6] + Sin[Pi/6]*I], Arg[12 + 3*I]}
```

$$\{\frac{\texttt{Pi}}{4},\ \frac{\texttt{5 Pi}}{6},\ \texttt{ArcTan[12, 3]}\}$$

Ha az aritmetikai kifejezés közelítő numerikus értéket is tartalmaz, akkor a kért eredmény egy közelítő értékét kapjuk meg:

```
{Arg[1. + I], Arg[12 + 3.*I]}
{0.785398, 0.244979}
```

A *Mathematica* komplex számokkal végzett műveletek eredményét algebrai alakban adja meg. A *Gauss-egészek* (azok a komplex számok, ame-

lyeknek a valós és a képzetes része is egész szám) körében is dolgozhatunk azokkal a függvényekkel, amelyek rendelkeznek a `GaussianIntegers` opcióval.

A `Head` függvény ad információt arról, hogy egy adott számot a program milyen típusúnak tekint. Például:

```
{Head[312], Head[312.], Head[5/10], Head[5./10]}
{Integer, Real, Rational, Real}

{Head[12 + 0*I], Head[12. + 0.*I]}
{Integer, Complex}

{Head[Sqrt[3]], Head[Sqrt[4]], Head[Sqrt[4.]]}
{Power, Integer, Real}
```

Számos matematikai problémánál lényeges tudni azt, hogy egy adott szám hozzátartozik-e a komplex számok valamely részhalmazához. Az ilyen jellegű kérdések megválaszolásához nyújthatnak segítséget különböző logikai értékeket megadó beépített függvények. Az

```
IntegerQ[x]
```

utasítás eredménye `True`, ha `x` egész szám, és `False` az ellenkező esetben. Hasonló értelemben használhatjuk még a következő utasításokat is:

```
EvenQ[x]          OddQ[x]
Negative[x]       Positive[x]
NonNegative[x]    PrimeQ[x]
NumberQ[x]
```

• Számok felírása különböző alakban

A *Mathematica* elvileg akárhány jegyű *egész szám* ábrázolására képes. Érdemes kipróbálni például azt, hogy milyen n természetes számig adja meg 3^n minden jegyét.

Használhatunk különböző alapú számrendszereket is. (Az alapértelmezés természetesen a 10-es számrendszer.) A

```
BaseForm[esz, asz]
```

utasítás eredménye a 10-es alapú számrendszerben megadott `esz` egész szám `asz` ($\leq$ 36)-alapú számrendszerbeli alakja. A 9-nél nagyobb szám-

jegyek jelölésére rendre az `a, b, c, d, ...` karaktereket használjuk. A következő példákban figyeljük meg azt is, hogy a program alsó indexbe írja a számrendszer alapszámát:

```
{BaseForm[37, 2], BaseForm[15821961566888151, 17]}
```

$\{100101_2,\ 1a2b3c4d5e6f7g_{17}\}$

Az `asz`-alapú számrendszerbeli `esz` számot az

```
asz^^esz
```

utasítás alakítja át 10-es alapú számrendszerbeli számmá:

```
{2^^100101, 16^^ffaa00}
{37, 16755200}
```

Az `IntegerDigits` belső függvénnyel egy egész szám számjegyeit tartalmazó listát kapunk:

```
IntegerDigits[12345678]
{1, 2, 3, 4, 5, 6, 7, 8}
```

Az `IntegerDigits[esz, asz]` utasítás eredménye pedig olyan lista, amelynek elemei a 10-es számrendszerben megadott `esz` szám `asz`-alapú számrendszerbeli alakjának számjegyei.

Itt hívjuk fel a figyelmet a 6 lehetséges opciót tartalmazó `NumberForm` belső függvényre, amely (különösen sok számjegyet tartalmazó) számok áttekinthetőbb megjelenítését támogatja. Az opciókat az

```
Options[NumberForm]
```

utasítás eredménye mutatja meg. Itt példaként csupán azt említjük meg, hogy ha a 30! szám számjegyeit 3-as csoportokban szeretnénk megjeleníteni úgy, hogy az egyes csoportokat a `|` karakter válassza el egymástól, akkor ezt így tudatjuk a *Mathematicá*val:

```
NumberForm[30!, DigitBlock -> 3, NumberSeparator -> "|"]
265|252|859|812|191|058|636|308|480|000|000
```

A *Mathematica* a *valódi törteket* is pontosan kezeli. Figyeljük meg, hogy ezeket a számláló és a nevező legnagyobb közös osztójával automatikusan egyszerűsíti:

```
452/62
```

$$\frac{226}{31}$$

A BaseForm függvényt valódi törtekre is alkalmazhatjuk:

```
BaseForm[3/4, 2]
```

$$\frac{11_2}{100_2}$$

A program a *közelítő numerikus értékeket* alapértelmezésben különböző formában adja vissza:

```
{6.7^-4, 6.7^6, 6.7^8}
{0.00049625, 90458.4, 4.06068 10^6}
```

A ScientificForm belső függvénnyel a számokat a matematikában megszokott lebegőpontos alakban ábrázolhatjuk:

```
ScientificForm[%]
{4.9625 10^-4, 9.04584 10^4, 4.06068 10^6}
```

Az EngineeringForm függvénnyel mérnöki alakban, könyvelési alakban pedig az AccountingForm-mal kapjuk meg az értékeket. Adott szám mantisszájából és kitevőjéből álló listát is gyárthatunk:

```
MantissaExponent[23.42]
{0.2342, 2}
```

A RealDigits és a NumberForm eljárásokat közelítő numerikus értékek esetén is használhatjuk.

• Pontos aritmetika

Ha egy matematikai kifejezésben csak pontos numerikus értékek szerepelnek, akkor a *Mathematica* a matematika szabályainak megfelelően végzi el a műveleteket, és kerekítések alkalmazása nélkül, pontosan adja meg a végeredményt:

```
(1+7/9)^20
```

$$\frac{1208925819614629174706176}{12157665459056928801}$$

• Közelítő aritmetika

Az explicit tizedesponttal megadott számokat a program közelítő értéknek tekinti, és az ezekkel végzett műveleteknél a matematikában megszokott kerekítési szabályokat alkalmazza.

A hagyományos programnyelvek az aritmetikai műveletek elvégzéséhez alapértelmezésben felhasználják az adott számítógép processzorába beépített numerikus lehetőségeket. Ezért a matematikai műveleteket csak meghatározott számú (általában 20-nál kevesebb) értékes számjegyet tartalmazó valós számokkal tudják elvégezni, és a (sok esetben nem egyszerűen végrehajtható) hibaanalízisről a felhasználónak kell gondoskodnia.

Ezzel szemben a *Mathematicá*t használva már alapértelmezésben is akár több ezer értékes jegyet tartalmazó valós vagy komplex számokkal is tudunk dolgozni, továbbá a hibabecslésről is kapunk értékes információkat. Az ilyen nagypontosságú műveletek kiértékelésénél a program készítői nem tudják közvetlenül felhasználni a számítógép processzorának numerikus lehetőségeit. A legalapvetőbb műveletek (összeadás, kivonás, szorzás, osztás) elvégzését is a programban kell megoldani. Ez a tény természetesen a kiértékelés gyorsaságát is befolyásolja. Sok problémánál azonban a gyorsaság fontosabb szempont, mint a pontosság. A *Mathematica* (a hagyományos programnyelvekhez hasonlóan) lehetővé teszi, hogy tekintettel legyünk erre. A bemenő adatok szintaxisával a felhasználó választja meg azt, hogy a kiértékelésnél a program melyik módszerrel dolgozzék.

Az imént elmondottak miatt a *Mathematica* közelítő numerikus értékeknek az alábbi két típusát különbözteti meg: *gépi pontosságú számok* és *nagypontosságú számok*. Az előbbiek rögzített számú számjegyet tartalmaznak, és a velük végzett műveleteknél a hibabecslést a felhasználónak kell elvégeznie. A program az ilyen számokkal végzett műveleteknél használja közvetlenül az adott számítógép hardverébe beépített numerikus lehetőségeket. A nagypontosságú számok elvileg (!) akárhány számjegyet tartalmazhatnak, és a program a velük végzett műveletek során bizonyos hibabecslést is végez.

A *Mathematica* az explicit tizedesponttal megadott valós vagy komplex számot akkor tekinti gépi pontosságú számnak, ha az értékes számjegyek száma legfeljebb akkora, mint a `$MachinePrecision` rendszerváltozó értéke alapértelmezésben. Az általunk használt gépek esetében ez a szám 16:

```
$MachinePrecision
```

```
16
```

Ha bizonytalanok vagyunk, akkor a `MachineNumberQ` beépített függvénnyel kérdezhetjük meg a programtól, hogy egy adott számot gépi pontosságúnak tekint-e vagy sem:

```
{MachineNumberQ[1.2], MachineNumberQ[1.2345678901234567],
 MachineNumberQ[0.00000000001234567]}
```

```
{True, False, True}
```

Ha a tízes számrendszerben beadott szám a `$MachinePrecision` értéknél kevesebb értékes számjegyet tartalmaz, akkor a *Mathematica* a hiányzó számjegyeket 0-nak veszi, és így hajtja végre a kért műveleteket.

A `$MachinePrecision` értékénél több értékes számjegyet tartalmazó közelítő numerikus értékeket a *Mathematica* nagypontosságú számnak tekinti. Ezeknél a felhasználható értékes számjegyek maximális számát a `$MaxPrecision` rendszerváltozó tartalmazza. Alapértelmezésben

```
$MaxPrecision
```

```
50000.
```

Ebben az esetben a gépi számábrázolás határai egészen lenyűgözőek:

```
{$MinNumber, $MaxNumber}
```

$$\{1.059345952\ 10^{-323228015},\ 1.440397122\ 10^{323228010}\}$$

Arról, hogy egy számot (például nagypontosságú számokkal végzett műveletek eredményét) a program milyen pontosnak tekint, a `Precision` és az `Accuracy` belső függvény tájékoztatja a felhasználót. A

```
Precision[x]
```

utasítás értéke a `$MachinePrecision` rendszerváltozóval egyenlő, ha `x` gépi pontosságú valós szám. Ha `x` egyéb valós közelítő érték, akkor az eredmény az `x` szám tízes számrendszerbeli alakjában szereplő összes értékes számjegy száma. Ha `x` komplex szám, akkor a valós és a képzetes részre a fentiek szerint vett számok minimumát kapjuk. A `Precision[x]` szám az `x` közelítő érték egy *abszolút* hibakorlátjával hozható kapcsolatba:

```
{Precision[1.2], Precision[2.2345678901234567],
 Precision[1.2345678901234567],
 Precision[1.2+1.2345678901234567*I]}
```

```
{16, 17, 16, 16}
```

Az `x` szám tizedestört alakjában a tizedesponttól jobbra eső értékes számjegyek számát adja meg a következő utasítás:

```
Accuracy[x]
```

Ez a szám az `x` közelítő érték egy *relatív* hibakorlátjával hozható kapcsolatba:

```
{Accuracy[2.1234567890123456789*10^2],
 Accuracy[122345.], Accuracy[722345.]}
{17, 11, 10}
```

Pontos numerikus értékekhez mindkét függvény az `Infinity` szimbólumot rendeli:

```
{Precision[2/3], Accuracy[2/3]}
{Infinity, Infinity}
```

Közelítő értékekkel való számolásnál felhasználhatjuk még az alábbi belső függvényeket is:

`SetAccuracy` `SetPrecision`

Ezekről részletesebb információt a [40] dolgozatban találhatunk.

• A kiértékelés elvei

Most felsoroljuk az (aritmetikai) kifejezések kiértékelésénél a *Mathematica* által követett általános elveket.

1°) Ha egy (aritmetikai) kifejezés csak pontos numerikus értékeket tartalmaz, akkor a program szimbolikus algoritmusokat felhasználva keresi a pontos eredményt:

```
pontos = (1 + 7/9)^40
```

$$\frac{1461501637330902918203684832716283019655932542976}{147808829414345923316083210206383297601}$$

```
(1 + 2*I)^50
110422359737857437 - 276811749100242716 I
```

2°) Ha az (aritmetikai) kifejezés tartalmaz gépi pontosságú számot, akkor a program minden számot ilyen típusúra alakít, és ezekkel végzi el a műveleteket:

```
kozelit = (1. + 7/9)^40
9.88778 10^9

(1. + 2*I)^50
1.10422 10^17 - 2.76812 10^17 I
```

A képernyőn az eredményből ilyenkor csak 6 értékes számjegy jelenik meg. A `kozelit` változóban a program azonban gépi pontosságú számot tárol:

```
Precision[kozelit]
16
```

Az összes (`$MachinePrecision` számú) értékes számjegyet például az `InputForm` belső függvénnyel jeleníthetjük meg:

```
kozelit // InputForm
9.8877830446376*10^9
```

3°) Ha a gépi pontosságú számokkal végzett műveletek eredménye a gépi pontosság ábrázolási határain (ezeket a

`$MinMachineNumber` és a `$MaxMachineNumber`

rendszerváltozók tartalmazzák) kívül esik, akkor a program automatikusan a nagypontosságú algoritmusokra tér át. Ezt illusztrálja a következő példa:

```
{$MinMachineNumber, $MaxMachineNumber}
{1.11254 10^-308, 1.79769 10^308}

3.12^1000.
1.4275588424553 10^494
```

4°) A *Mathematica* a programban megírt aritmetikai műveleteket végző algoritmusokat hívja meg abban az esetben, ha a kifejezésben csak nagypontosságú számokat talál. Ilyenkor minden lépésben nyomon követi az adott művelet pontosságát (hibabecslést végez), és a beadott adatok pontosságához képest a lehető legjobb közelítést próbálja megadni. A végeredményben csak pontos számjegyeket tüntet fel. A beépített algoritmusokról részletesebb információt a [41] dolgozatban találhatunk.

5°) Nagypontosságú számokkal végzett műveletek eredményeként a program a 0.0 számot írja ki akkor, ha a végeredményben nem talál értékes számjegyet.

• Az `N` belső függvény

Az `N` beépített függvény különböző (például aritmetikai) kifejezéseknek egy közelítő értékét adja meg. Például:

```
{N[pontos], N[Sqrt[3*Pi]]}
{9.88778 10^9, 3.06998}
```

```
Sqrt[3 Pi] // N
3.06998
```

Említettük már azt, hogy közelítő numerikus értékek pontos számjegyeinek a számát a felhasználó igen tág határok között választhatja meg. Nem jelent problémát például a korábban értelmezett `pontos` szám akár több ezer jegyre kerekített értékének megadása sem. Efféle szándékunkat így közöljük a *Mathematicá*val:

```
N[pontos, 100]
9.887783044637613109188495241372711699666479884567883\
      00939082252340050944344374693500679607273009352 7 10^9
```

Most ismertetjük azt, hogy a *Mathematica* az `N` argumentumától függően hogyan választja meg az

```
N[valami, n]
```

utasítás kiértékelésénél felhasznált algoritmust, ahol `valami` egy (például aritmetikai) kifejezés és `n` pozitív egész szám.

1°) Kezdjük azzal, hogy a második argumentum elhagyható. Ennek alapértelmezésbeli értéke `$MachinePrecision`. Tehát az `N[valami]` és az `N[valami, 16]` utasítás egyenértékű.

2°) Nézzük most azt az esetet, amikor `valami` csak pontos értéket tartalmaz. Ekkor a *Mathematica* először szimbolikus algoritmusok felhasználásával próbálja pontosan meghatározni a `valami` kifejezést. Ha ez sikerül, akkor ezután veszi annak `n` jegyre kerekített értékét. Abban az esetben, amikor a `valami` kifejezést nem tudja pontosan kiértékelni, akkor a benne előforduló összes számnak veszi az `n` értékes számjegyet tartalmazó közelítő értékét, és numerikus algoritmusokat felhasználva számolja ki az eredményt. Például:

```
{kozelit4 = N[pontos, 4], kozelit23 = N[pontos, 23]}
{9.888 10^9, 9.8877830446376131091885 10^9}
```

Jegyezzük meg azonban azt, hogy az `n<$MachinePrecision` esetén kapott eredményt a további műveleteknél a *Mathematica* mindig gépi pontosságú számnak tekinti:

```
{InputForm[kozelit4], Precision[kozelit4]}
{9.88778304463762*10^9, 16}
```

```
3 kozelit4 // InputForm
2.966334913391285*10^10
```

3°) Ha a `valami` kifejezés tartalmaz gépi pontosságú számot, akkor a *Mathematica* a második argumentumtól függetlenül ilyen számok felhasználásával végzi el a kijelölt műveleteket:

```
koz = (1. + 23/7)^40;
{N[koz, 5], N[koz, 500]}
{1.9095 10^25, 1.909539243949085 10^25}
```

4°) Ha a `valami` kifejezés csak nagypontosságú számokat és pontos numerikus értékeket tartalmaz, akkor az `N[valami, n]` utasítás hatására a program `n` értékes számjegyű számokkal végzi el a műveleteket. Ilyenkor a felhalmozódó kerekítési hibák miatt a végeredmény értékes jegyeinek száma általában kisebb, mint `n`. A [41] dolgozatban olvashatunk arról bővebben, hogy a *Mathematica* ebben az esetben hogyan végzi el a hibabecslést. A `Precision` belső függvény tájékoztat arról, hogy a *Mathematica* szerint a végeredmény hány pontos számjegyet tartalmaz. Például:

```
N[Sqrt[70], 40]
Precision[%]
8.366600265340755479781720257851874893928
40

N[Sqrt[70], 40]^723
1.006795962272719463653835243187716312 9 10^667

{Precision[%], Accuracy[%%]}
{37, -630}
```

Ismételten hangsúlyozzuk, hogy az `N[valami, n]` utasítás tehát nem azt jelenti, hogy „határozzuk meg a `valami` kifejezést `n` értékes számjegyre”, hanem azt, hogy „`n` értékes számjegyet tartalmazó számokkal határozzuk meg a `valami` kifejezést”.

• Valós számok közelítése pontos értékekkel

Az `x` valós szám egész részét a `Floor[x]` utasítás eredménye adja meg. `Ceiling[x]` a legkisebb olyan egész szám, amely nagyobb vagy egyenlő, mint `x`. Az `x` szám egészekre kerekített értékét a `Round[x]` utasítással kapjuk meg. A *Mathematica* komplex számok esetén a valós és a képzetes részre alkalmazza a fentieket:

```
{Floor[-2.5], Floor[3.75], Floor[1.2-3.4 I]}
{-3, 3, 1 - 4 I}

{Ceiling[-2.5], Ceiling[3.75], Ceiling[1.2-3.4*I]}
{-2, 4, 2 - 3 I}

{Round[0.45], Round[1.5], Round[1.50001]}
{0, 1, 2}
```

• Programcsomagok

Az eddig ismertetett belső függvényeken kívül még néhány programcsomag is tartalmaz a számábrázoláshoz és az aritmetikai műveletekhez is kapcsolható függvényeket.

A részletek ismertetése nélkül csupán utalunk ezekre a lehetőségekre. Az *adott gépünk* lebegőpontos aritmetikáját tanulmányozhatjuk a

```
NumericalMath`Microscope`
```

programcsomag alábbi függvényeivel:

`MachineError`	`MicroscopicError`
`Microscope`	`Ulp`

A számítógép lebegőpontos aritmetikájának *elméletét* illusztrálhatjuk a

```
NumericalMath`ComputerArithmetic`
```

programcsomag következő függvényeivel:

`Arithmetic`	`SetArithmetic`
`ComputerNumber`	

Az elsősorban demonstrációs célokat szolgáló

```
NumericalMath`IntervalArithmetic`
```

programcsomag `Interval` függvénye az intervallumaritmetika elméletéhez nyújthat segítséget. A hibabecslésekkel kapcsolatos alkalmazásokhoz a

```
NumericalMath`IntervalAnalysis`
```

programcsomag `Interval` (amely az azonos nevű belső függvény kiterjesztése) és `SetEpsilon` függvényeit érdemes felhasználni.

3.1.4. Függvények

A *Mathematicá*val alapértelmezésben $\mathbf{C}^n \to \mathbf{C}^m$ $(n, m \in \mathbf{N})$ típusú függvényekkel dolgozhatunk. Ebben a pontban ilyen függvények megadásának különféle módjairól lesz szó. A beépített matematikai függvények igen széles köre nyújt segítséget függvények *képlettel* való értelmezéséhez. Megadhatunk függvényeket *eljárással* is, azaz definiálásukhoz a program egyéb függvényeit is felhasználhatjuk.

• Beépített matematikai függvények

A matematikában használt speciális függvények többsége beépített függvényként szerepel a *Mathematicá*ban. Ezek közül itt csak néhányat sorolunk fel. A fennmaradó függvényekről az adott témakörnél teszünk említést.

Abs	ArcTan	Plus
ArcCos	ArcTanh	Power
ArcCosh	Cos	Sec
ArcCot	Cosh	Sech
ArcCoth	Cot	Sin
ArcCsc	Coth	Sinh
ArcCsch	Csc	Sqrt
ArcSec	Csch	Subtract
ArcSech	Divide	Tan
ArcSin	Exp	Tanh
ArcSinh	Log	Times

A beépített matematikai függvényeket a program komplex változósnak tekinti, ezeket bármely komplex szám esetén kiértékeli. A helyettesítési érték kiszámolásakor az előző pontban leírt elveket követi.

Ha a függvény argumentuma pontos numerikus érték, akkor sok esetben a pontos helyettesítési értéket kapjuk meg:

```
{Sqrt[9], Sqrt[7], Sin[Pi/12], Cos[Pi/10]}
```

$$\{3,\ \text{Sqrt}[7],\ \frac{-1 + \text{Sqrt}[3]}{2\text{Sqrt}[2]},\ \text{Cos}[\frac{\text{Pi}}{10}]\}$$

```
ComplexExpand[Exp[Pi/3 I]]
```

$$\frac{1}{2} + \frac{\text{I}}{2}\ \text{Sqrt}[3]$$

Az `N` belső függvény (lásd a 3.1.3. pontot) felhasználásával a pontos helyettesítési érték akár több ezer értékes jegyét is meghatározhatjuk. Próbáljuk ki például a következő utasítást:

```
N[Cos[Pi/10], 5000]
```

Ha egy matematikai függvény argumentumába gépi pontosságú számot írunk és az `N` függvényt alkalmazzuk, akkor az eredmény az `N` második argumentumától függetlenül gépi pontosságú szám lesz:

```
N[Sin[1.2]] // InputForm
0.932039085967226

N[Sin[1.2], 5] // InputForm
0.932039085967226

N[Sin[1.2], 30]
0.932039085967226
```

Ha a `$MachinePrecision` rendszerváltozó (l. a 3.1.3. pontot) értékénél több értékes jegyre kívánjuk meghatározni egy beépített függvény helyettesítési értékét, akkor az egyik lehetőségünk az, hogy az argumentumba pontos numerikus értéket írunk:

```
N[Sin[12/10], 25]
0.9320390859672263496701344355
```

Megtehetjük azt is, hogy explicit tizedesponttal jelölt közelítő numerikus értéket használunk. Ilyenkor azonban legalább annyi számjeggyel kell megadnunk az argumentumot, amennyi értékes jegyet szeretnénk az eredményben:

```
N[Sin[1.2000000000000000000000000000000], 30]
0.93203908596722634967013443549 5
```

A beépített matematikai függvények egyik része olyan műveletekkel kapcsolatos, amelyeknek az eredménye bármely komplex argumentum esetében egyértelműen meg van határozva. Ilyen függvények például az abszolút érték függvény, a hatványfüggvény vagy a trigonometrikus függvények.

Számos beépített matematikai függvény azonban olyan művelettel kapcsolatos, amelynek az eredménye nincs egyértelműen meghatározva. Ezek közül a legegyszerűbb az `Arg` függvény. A *Mathematicá*ban `Arg[z]` a nullától különböző `z` komplex számnak a $(-\pi, \pi]$ intervallumba eső argumentumát adja meg radiánban. A `0` számhoz az `Arg` függvény a 0 számot rendeli.

Nem ennyire egyszerű a helyzet a négyzetgyök-, a logaritmus- és az arkuszfüggvények esetében. Mindenesetre ezek a beépített függvények matematikai értelemben is függvények, azaz minden komplex számhoz *egyetlen* komplex számot rendelnek. A program által adott válaszok pontos megértéséhez világosan kell látnunk azt, hogy a program készítői hogyan értelmezték ezeket a függvényeket. Azaz: a több lehetséges komplex érték közül az adott függvény melyik komplex számot szolgáltatja helyettesítési értékként.

Nézzük például a négyzetgyökfüggvényt. Ismeretes, hogy a komplex számok körében a négyzetgyökvonás művelete nem egyértelmű. Ezt a tényt kifejezhetjük úgy is, hogy a

$$\mathbf{C} \longrightarrow \mathbf{C}, \qquad z \mapsto z^2$$

függvény nem kölcsönösen egyértelmű (azaz nem *bijekció*). Ennek a függvénynek számos olyan leszűkítése van, amely már bijekció. Tekintsük például a jobb oldali komplex félsíkot:

$$D := \{\, x + iy \in \mathbf{C} \;:\; (x \in \mathbf{R}^+ \text{ és } y \in \mathbf{R}) \text{ vagy } (x = 0 \text{ és } y \in \mathbf{R}_0^+)\}.$$

Egyszerűen bebizonyítható az, hogy a

$$D \to \mathbf{C},\ z \mapsto z^2$$

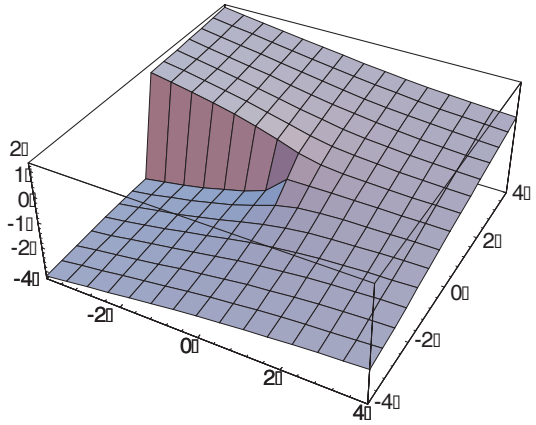
függvény már kölcsönösen egyértelmű, tehát invertálható. Az `Sqrt` belső függvény pontosan ennek a függvénynek az inverze. Ez azt jelenti, hogy valamely komplex szám két lehetséges négyzetgyöke közül az `Sqrt` függvény azt az egyértelműen meghatározott négyzetgyököt veszi, amelyik a D jobb oldali komplex félsíkba esik.

Az így értelmezett `Sqrt` függvény nem folytonos a $\mathbf{C}$ halmazon. Pontosabban fogalmazva: a komplex számsíknak csak a negatív valós féltengelytől (jelölje $T := \{x + yi \in \mathbf{C} : x \in \mathbf{R}_0^-$ és $y = 0\}$ ezt a halmazt) különböző pontjaiban folytonos. A függvénynek a T halmaz pontjaiban szakadása van (ezért ezt a halmazt az `Sqrt` függvény *szakadási ágá*nak is nevezik), ami azt jelenti, hogy a T-hez „közeli” pontokban felvett függvényértékek „távol” lehetnek egymástól:

```
{Sqrt[-1. + 0.002 I], Sqrt[-1. - 0.002 I]}
{0.001 + 1. I, 0.001 - 1. I}
```

Minderről szemléletes képet ad az alábbi ábra:

```
Plot3D[Im[Sqrt[x + y I]], {x, -4, 4}, {y, -4,4}]
```



Komplex szám négyzetgyökei közül egynek a kiválasztását tehát a szakadási ág kijelölésével tehetjük egyértelművé. (Ha a négyzetgyökfüggvényt úgy definiáltuk volna, hogy a szakadási ága a pozitív valós féltengely legyen, akkor ennek helyettesítési értékei a felső félsíkba esnének.)

A többi beépített inverz függvény szakadási ágának pontos leírása a [89] referenciakönyv 565. oldalán szükség esetén megtalálható. A választott inverz függvények éppen azok a függvények, amelyeket a komplex függvénytanban *főértékek*nek szoktak nevezni.

A beépített matematikai függvényeknek is vannak *attribútumai* (lásd a 2.3.4. pontot). Például:

```
Attributes[Tan]
{Listable, Protected}
```

A `Listable` attribútum azt jelenti, hogy ha a függvény argumentumába listát (speciálisan vektort vagy mátrixot) írunk, akkor a program a szóban forgó függvényt a lista minden elemére külön-külön alkalmazza. Függvényértékek felülírását akadályozza meg a `Protected` attribútum.

• Függvények megadása

Megemlítettük már, hogy a *Mathematicá*n belül alapértelmezésben dolgozhatunk $\mathbf{C}^n \to \mathbf{C}^m$ $(n, m \in \mathbf{N})$ típusú függvényekkel. Ezeket három különböző módon is definiálhatjuk: *késleltetett értékadással*, *azonnali értékadással* vagy *tiszta függvénnyel*.

Tekintsük először azt a legegyszerűbb esetet, amikor az egész $\mathbf{C}$ halmazon egyetlen formulával kívánunk definiálni egy függvényt. Legyen ez például a következő:

$$f(x) := x^3 \qquad (x \in \mathbf{C}).$$

Késleltetett értékadással ezt a függvényt így értelmezzük:

```
f[x_] := x^3
```

A definícióban szereplő aláhúzás karakter, a `_` jel arra utal, hogy f-et az összes lehetséges (komplex) változóra definiáljuk; az előtte álló `x` betű pedig a függvény változójára.

A későbbi hivatkozásokban `f` változóját `x` helyett más szimbólummal is helyettesíthetjük:

```
{f[u], f[valami]}
```

$$\{u^3,\ valami^3\}$$

Kihangsúlyozzuk a `_` jel szerepét. Figyeljük meg, hogy az

```
f[x] := Sin[x];
```

utasítás értékadást jelent, amely az f függvény értelmezési tartományának egyetlen rögzített elemére határozza meg f értékét, ez az elem `x`:

```
{f[x] + f[y], f[u] + f[v]}
```

$$\{y^3 + Sin[x],\ u^3 + v^3\}$$

Kiszámolhatjuk a megadott függvény bármely valós vagy komplex helyen vett helyettesítési értékét:

```
{f[3], f[2 + 3 I]}
{27, -46 + 9 I}
```

Függvényértékekből táblázatot is készíthetünk. Például a

```
g[x_] := 1/(x-3)
```

függvény helyettesítési értékeit a $[2, 3]$ intervallumban $0,5$-es lépésközt használva így kapjuk meg:

```
helyek = Table[x, {x, 2, 3, 0.5}]
{2, 2.5, 3}

fvertekek = g[helyek]
Power::infy: Infinite expression 1/0 encountered.
{-1, -2., ComplexInfinity}
```

Adjuk meg az eredményeket a szokásos táblázatformában:

```
TableForm[{helyek, fvertekek},
     TableHeadings -> {{" x", "g[x]"}, None},
     TableAlignments -> Center]
```

```
 x      2     2.5        3.
g[x]   -1    -2   ComplexInfinity
```

Egy általunk definiált függvényről ugyanúgy tájékozódhatunk, mint a beépítettekről:

```
?f
```

```
Global`f
  f[x] := Sin[x]
  f[x_] := x^3
```

Ha a továbbiakban az f szimbólumot más célra akarjuk használni, akkor a definíciót felül lehet írni, vagy pedig (ez általában célszerűbb) érvényteleníteni kell. Ehhez a következő utasítások valamelyikét használhatjuk:

```
Clear[f]
ClearAll[f]
Remove[f]
f=.
```

Függvény definíciójának „jobb oldalán” természetesen használhatjuk a beépített matematikai függvényeket, de a *Mathematica* más függvényeit is. Például:

```
kifejt[n_] := Expand[Sum[(1-x) x^i, {i, 0, n}]]
kifejt[3]
```

```
   4
1-x
```

Több utasításból álló eljárást is definiálhatunk így (figyeljük meg, hogy az utasítások sorozatát zárójelbe kell tennünk):

```
egyutthato[n_, i_] := (t=kifejt[n]; Coefficient[t, x^i])
egyutthato[3, 4]
```

```
-1
```

Az *azonnali értékadás* szintaktikailag csupán abban különbözik a késleltetett megadási módtól, hogy ennél az `=` jelet (teljes alakban: `Set`) kell használnunk a `:=` jel (ami a `SetDelayed` rövidítése) helyett. A szemantikai különbséget a következő példán érzékeltetjük. Tegyük fel, hogy az f

szimbólummal akarjuk jelölni az exp ∘ sin kompozíciófüggvény deriváltját. Késleltetett értékadással ezt így tudatjuk a *Mathematicá*val:

```
Clear[f]
f[x_] := D[Exp[Sin[x]], x]
```

Ha ezután például a függvény helyettesítési értékére vagyunk kíváncsiak, akkor a program hibaüzenetet küld:

```
f[3]
Global::ivar: 3 is not a valid variable.
D[E^Sin[3], 3]
```

Késleltetett értékadásnál a *Mathematica* nem számolja ki automatikusan a `:=` jel jobb oldalán álló kifejezést, csupán a transzformációs szabályt jegyzi meg. Az `f[3]` utasítást úgy hajtja végre, hogy a 3 számot *formálisan* behelyettesíti a definíció jobb oldalába, és ezután próbálja kiértékelni az így kapott kifejezést. Ebben az esetben ez nem működik.

A megoldás az azonnali értékadás. Ekkor a program rögtön kiértékeli az `=` jel jobb oldalán megadott kifejezést, azaz minden olyan műveletet elvégez, amelyre korábban megtanították:

```
Clear[f]
f[x_] = D[Exp[Sin[x]], x]
E^Sin[x] Cos[x]

{f[3], f[1+a]}
{E^Sin[3] Cos[3], E^Sin[1 + a] Cos[1 + a]}
```

Mivel vektorokat listával adhatunk meg (lásd a 3.8.1. pontot), ezért többváltozós függvényeket is egyszerűen definiálhatunk. Az alábbi példában egy $\mathbf{C}^2 \to \mathbf{C}^3$ típusú függvényt értelmezünk:

```
Clear[f]
f[x_, y_] := {Exp[x+y], x^2 y^3, Sin[x]/(1+y^2)}
```

A *Mathematicá*ban *tiszta függvényként* említik a matematikában megszokott $x \mapsto f(x)$ megadási módot. Az

$$x \mapsto x^2 + x + \sin(x)$$

függvényt így is definiálhatjuk:

```
h1 = Function[x, x^2 + x + Sin[x]]
Function[x, x^2 + x + Sin[x]]
```

```
h1[2.]
6.9093
```

Rövidebb alakban ezt a függvényt így is megadhatjuk:

```
h2 = (#^2 + # + Sin[#])&
  2
#1  + #1 + Sin[#1] &

h2[2.]
6.9093
```

A fenti utasításban a `#` jel utal a függvény változójára, a `&` jel pedig arra, hogy itt függvényről van szó. (Ez a fajta megadási mód ismerős lehet a LISP nyelvben járatosak számára.)

Az említett példákban nevet adtunk a függvénynek. Gyakran viszont éppen azért használjuk ezt a megadási módot, hogy ne kelljen külön elnevezni egy olyan függvényt, amely csak egy műveletben szerepel és a továbbiakban nincs szükség rá.

Tiszta függvény célszerű használatát illusztrálják a következő példák:

```
Map[(#^2)&, {1, 2, 3}]
{1, 4, 9}

Map[Function[x, x^2], a+b+c]
  2    2    2
a  + b  + c

Nest[(1/(1+#))&, x, 3]
                1
-------------------------------
                1
1 + -------------------------
                1
    1 + -----------
                1
        1 + -------
            1 + x
```

Többváltozós függvények ábrázolásánál mutatunk példát arra, hogy egy *Mathematica*-függvény opciójában tiszta függvényt mire és hogyan lehet használni.

Tiszta függvény megadásánál a `#` jel tehát az identitásfüggvény (ennek hosszabb neve: `Identity`) jele. Ne feledkezzünk meg arról, hogy a definíciót a `&` jellel kell lezárni. Többváltozós függvényeket is megadhatunk ilyen módon. Ekkor `#n` jelöli az n-edik változót:

```
l = #1^2 + #2^3&
  2    3
#1  + #2
```

```
l[x, y]
```

$$x^2 + y^3$$

Hasznos és a matematikában megszokottnál általánosabb konstrukciók is előfordulnak. ## jelöli a változók összességét, ##n pedig a változók összességét az n-edik változótól kezdve:

```
f[##, ##]&[x, y]
f[x, y, x, y]

f[##, ##]&[x, {y, x}]
f[x, {y, x}, x, {y, x}]

f[##2, #1]&[a, b, c]
f[b, c, a]
```

Végül két összetettebb példa következik:

```
Clear[f]
Apply[f[##2, #1]&, {{a, b, c}, {ap, bp}}, {1}]
{f[b, c, a], f[bp, ap]}

Map[f[##2, #1]&, {{a, b, c}, {ap, bp}}, {1}]
{f[{a, b, c}], f[{ap, bp}]}
```

Annak magyarázata, hogy látszólag nem kell előre rögzíteni, hogy egy függvénynek hány változója van, az elvekről szóló 2.3.2. pontban található.

• Feltételekkel megadott függvények

Az eddig felsorolt függvénymegadási módokat kényelmesen használhatjuk azokban az esetekben, amikor függvényünk az egész $\mathbf{C}^n$ halmazon egyetlen képlettel van definiálva.

A *Mathematica* csak egész számokra számolja ki a helyettesítési értéket akkor, ha a függvényt így definiáljuk:

```
f[x_Integer] := x^2
{f[-2], f[-2.], f[1/2], f[3], f[Sqrt[4]], f[Sqrt[7]]}
```

$$\{4,\ f[-2.],\ f[\tfrac{1}{2}],\ 9,\ 4,\ f[\sqrt{7}]\}$$

Az Integer függvény helyett a fenti mintázatba beírhatjuk például a Complex, a List, a Real vagy a Symbol függvényeket is.

A feltételekben logikai értékeket felvevő függvényeket például így használhatunk:

```
g[x_?NumberQ] := x^3
{g[2+3 I], g[2.1], g[3/2], g[y]}
```

$$\{-46 + 9\,I,\ 9.261,\ \frac{27}{8},\ g[y]\}$$

A „*mintázat*?*feltétel*” szerkezetre egy másik példa:

```
h[{x_Integer, y_Integer} ?
       (Function[v, Dot[v, v] > 4])] := er[x+y]
{h[{3, 4}], h[{1, 1}], h[{-5, 7}], h[2]}
{er[7], h[{1, 1}], er[2], h[2]}
```

(Figyeljük meg, hogy a feltételt leíró jelsorozatot gömbölyű zárójelek közé kell írnunk.)

A `Condition` (ennek rövid alakja a `/;` jelsorozat), az `If`, a `Which` vagy a `Switch` eljárást is használhatjuk:

```
p[n_Integer] := n! /; n > 0
{p[-3], p[5], p[7.5]}
{p[-3], 120, p[7.5]}
```

A `/;` jel utáni feltételt az argumentumba is beírhatjuk:

```
Clear[p]
p[n_Integer /; n>0] := n!
{p[-3], p[5], p[7.5]}
{p[-3], 120, p[7.5]}
```

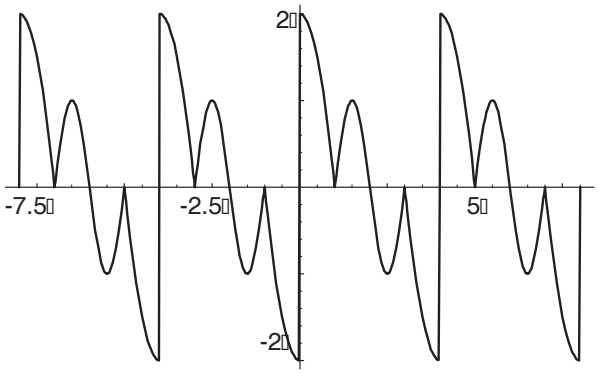
Többváltozós függvényeket is megadhatunk ilyen módon:

```
m[lista_] := lista^2 /; VectorQ[lista, IntegerQ]
{m[{1, -2}], m[{0.5, 3}], m[{{1, 2}, {3, 4}}]}
{{1, 4}, m[{0.5, 3}], m[{{1, 2}, {3, 4}}]}
```

Periodikus függvényt a `Condition` eljárással így értelmezhetünk:

```
Clear[f]
f[x_ /; 0<=x<=1] := -2 (x-1) (x+1)
f[x_ /; 1<x<=2] := -Cos[Pi (x-1/2)]
f[x_ /; -2<=x<0] := -f[-x]
f[x_ /; x<-2] := f[x+4]
f[x_ /; 2< x] := f[x-4]
```

```
Plot[f[x], {x, -8, 8}, Ticks -> {{-7.5, -2.5, 5}, {-2, 2}}]
```



Mutatunk egy példát a `Which` eljárás használatára:

```
Clear[h]
h[x_] := Which[x<0, x^2, x>5, x^3, True, 0]
{h[-4], h[7], h[2]}
```

```
{16, 343, 0}
```

Végül egy olyan függvényt definiálunk, amely egész számokhoz az a, a b vagy a c szimbólumot rendeli attól függően, hogy a szám 3-mal való osztásának maradéka 0, 1 vagy 2:

```
r[x_Integer] := Switch[Mod[x, 3], 0, a, 1, b, 2, c]
{r[3], r[3.], r[16], r[5]}
```

```
{a, r[3.], b, c}
```

• Függvényjelölési módok

A *Mathematica*ban az f függvény x helyen fölvett értékének

```
f[x]
```

jelölésén kívül használatosak még a következők is: az

```
f @ x
```

prefix alak (amelynél — a hagyományos jelöléstől eltérően — nem kell félbeszakítanunk a függvényhez tartozó jelsorozat leírását az argumentum kedvéért), az

```
x // f
```

posztfix alak (például a `% // f` jelsorozatot gépeljük be, hogy a legutolsó eredményt ne kelljen még egyszer beírnunk), végül pedig (például a halmazműveletek szemléletes megjelenítése céljából) kétváltozós függvények esetén az

```
x~f~y
```

infix alak. Például:

```
Clear[f]
{f @ x, f @ x + y, f @ (x + y)}
{f[x], f[x] + y, f[x + y]}

{x + y // f, 3^(1/4) + 1 // N}
{f[x + y], 2.31607}

{a, b, c}~Union~{a, d, e}
{a, b, c, d, e}
```

• Műveletek függvényekkel

Aritmetikai műveleteket végezhetünk $\mathbf{C} \to \mathbf{C}$ típusú függvényekkel. Képezhetjük például az összegüket:

```
h = f + g
h[2]
(f + g)[2]
```

Ahhoz viszont, hogy explicite megkapjuk az összeget, szükségünk van a `Through` függvényre:

```
Through[h[2]]
f[2] + g[2]
```

Operátorokkal is hasonlóan járhatunk el, amint az kiderül az alábbi példákból:

```
op = Identity + (D[#, x]&);
op[x^2]
```

```
(Identity + (D[#1, x] & ))[x^2]
```

```
Through[%]
```

```
2 x + x^2
```

Összetett függvény képzéséhez a `Composition` belső függvényt használhatjuk:

```
Clear[f, g, h1, h2]
f[x_] := x^2
g[x_] := Sqrt[x-1]
```

```
{h1 = Composition[f, g], h1[x]}
```

```
{Composition[f, g], -1 + x}
```

```
{h2 = Composition[g, f], h2[x]}
```

```
{Composition[g, f], Sqrt[-1 + x^2]}
```

Igen gyakran előfordul, hogy ugyanazt a függvényt akarjuk egymás után többször is alkalmazni, vagyis egy többszörösen összetett függvény értékét akarjuk kiszámítani. Ezzel a témakörrel részletesen foglalkozunk majd a programozásról szóló, 4. fejezetben, mivel az itt röviden ismertetett konstrukciók a *rekurzió*, sőt általánosabban a *funkcionális programozás* alapvető eszközei:

```
Nest[f, x, 3]
```

```
f[f[f[x]]]
```

```
NestList[f, x, 3]
```

```
{x, f[x], f[f[x]], f[f[f[x]]]}
```

A $\sqrt{2}$ szám közelítő értékeit Newton-féle iterációval például így kaphatjuk meg:

```
gyok2[x_] := N[(x+2/x)/2]
NestList[gyok2, 1.0, 5]
```

```
{1., 1.5, 1.41667, 1.41422, 1.41421, 1.41421}
```

Az iterációs lépések számát is változtathatjuk:

```
FixedPoint[gyok2, 1]
1.41421

FixedPointList[gyok2, 1]
{1, 1.5, 1.41667, 1.41422, 1.41421, 1.41421, 1.41421}

FixedPointList[gyok2, 1,
          SameTest -> (Abs[#1-#2] < 10.^-2&)]
{1, 1.5, 1.41667, 1.41422}
```

Ha pedig egy kifejezésből kiindulva egy (általában különböző) függvényekből álló lista elemeit akarjuk alkalmazni az eredményre egymás után, akkor a `ComposeList` utasítást használhatjuk:

```
Clear[f, g, h]
ComposeList[{f, g, h},x]
{x, f[x], g[f[x]], h[g[f[x]]]}
```

Itt említjük meg az egész együtthatós polinomok körében használható `Decompose` belső függvényt, amely a beadott polinomot alacsonyabb fokszámú egész együtthatójú polinomok kompozíciójára próbálja felbontani. Például:

```
q[x_] := x^6-3x^5+5x^4-5x^3-x^2+3x+2
Decompose[q[x], x]
                 2    3
{2 - 3 x + 2x  + x , (-1 + x) x}
```

A kapott eredmény azt jelenti, hogy q a

```
p1[x_] := %[[1]]
                 2    3
2 - 3 x + 2 x  + x

p2[x_] := %%[[2]]
(-1 + x) x
```

polinomoknak ebben a sorrendben vett kompozíciója. Valóban:

```
q[x] == Expand[[p1[p2[x]]]
True
```

Megjegyezzük még azt, hogy a dekompozíció művelete nem egyértelmű. Valóban, ha p_1 és p_2 polinom, akkor bármely a valós szám esetén érvényes a $p_1 \circ p_2 = \bar{p}_1 \circ \bar{p}_2$ egyenlőség, ahol

$$\bar{p}_1(x) := p_1(x-a), \qquad \bar{p}_2(x) := p_2(x) + a \qquad (x \in \mathbf{R}).$$

A *Mathematica* a végeredményt úgy adja meg, hogy a belső polinom állandó tagját 0-nak választja.

Úgy tűnik, mintha könnyen lehetne *inverz függvényt* számolni:

```
InverseFunction[Exp]
Log

{InverseFunction[f], InverseFunction[f][x]}
                 (-1)   (-1)
{f       , f       [x]}
```

A program a függvények kompozícójának inverzére vonatkozó állítást is ismeri:

```
InverseFunction[Composition[f, g]]
                          (-1)   (-1)
Composition[g       , f       ]

%[x]
 (-1)   (-1)
g      [f      [x]]
```

Ezeket következetesen alkalmazza:

```
{InverseFunction[Sin], InverseFunction[ArcSin]}
{ArcSin, Sin}

Composition[Sin, ArcSin][x]
x

Composition[ArcSin, Sin][x]
ArcSin[Sin[x]]
```

Ha viszont nem beépített, bár könnyen invertálható függvénnyel van dolgunk, akkor az `InverseFunction` eljárás nem segít:

```
InverseFunction[Function[x, 3x + 2]]
                         (-1)
Function[x, 2 + 3 x]

%[t]
                         (-1)
Function[x, 2 + 3 x]         [t]
```

Egy lehetséges — kevés esetben használható — megoldás:

```
inverz[x_] := Module[{s}, Solve[3s + 2 == x, s][[1,1,2]]]
inverz[x]
-2 + x
------
  3
```

• Rekurzív módon megadott sorozatok

Külön alpontban kívánjuk felhívni az Olvasó figyelmét arra, hogy kétféleképpen is megadhat rekurzív módon definiált sorozatokat. Ha az

```
f[n_] := f[n] = f[n-1] + f[n-2]
```

szerkezetű szintaxissal adja meg a sorozat tagjai között fennálló összefüggést, akkor a program megjegyzi (a memóriában elraktározza) az egyszer már kiszámolt függvényértéket. Ha a matematikában megszokott jelölésekhez közelebb álló

```
f[n_] := f[n-1] + f[n-2]
```

típusú szintaxist választja, akkor a *Mathematica* a szabályos kiértékelési eljárást (lásd a 2.3.5. pontot) alkalmazza. Ebben az esetben az `f[20]` helyettesítési érték meghatározásához például az `f[5]` függvényértéket a program többször is kiszámolja.

Az elmondottakból az következik, hogy az első esetben a kiértékelés gyorsabb, de több memóriát igényel, mint a második esetben.

A következő példák azt is mutatják, hogy a fenti két módszerrel történő kiértékeléshez szükséges időtartamok *lényegesen* eltérőek is lehetnek:

```
Clear[f1]
f1[0] = 1;
f1[1] = (1 - Sqrt[5])/2;
f1[n_] := f1[n] = f1[n-1] + f1[n-2]
f1[10] // Simplify
123 - 55 Sqrt[5]
----------------
       2
```

Nézzük meg a 20. tag kiszámolásához szükséges időtartamot:

```
Timing[f1[20];]
{0.988 Second, Null}
```

Most válasszuk a másik megadási módot:

```
Clear[f2]
f2[0] = 1;
f2[1] = (1 - Sqrt[5])/2;
f2[n_] := f2[n-1] + f2[n-2]
f2[10] // Simplify
```

$$\frac{123 - 55\ \mathrm{Sqrt}[5]}{2}$$

Ekkor a 20. tag kiszámolásához szükséges időtartam:

```
Timing[f2[20];]
{158.459 Second, Null}
```

A szóban forgó sorozat kezdőtagjait pontos numerikus értékekkel definiáltuk, ezért a program a pontos eredményt adta meg. Ennek a sorozatnak a tagjai nagy indexek esetén 0-hoz „közeli" értékeket vesznek fel, azaz a sorozatnak 0 a határértéke. (Hogyan lehetne ezt bebizonyítani?)

További kísérleteket is érdemes elvégezni. Hajtsuk végre a fenti utasításokat úgy, hogy a sorozatnak például az első tagját *közelítő numerikus értékekkel* adjuk meg. Ekkor a program numerikus módszereket alkalmaz a kiértékelésnél. Ha az *időeredményeket* összehasonlítjuk, akkor tapasztalhatjuk azt is, hogy a numerikus algoritmusok mennyivel gyorsabbak a pontos algoritmusoknál.

Ugyanezen a példán illusztrálhatjuk közelítő értékekkel való számolások esetében a kerekítési hibák terjedésének következményeit is. Figyeljük meg, hogy közelítő kezdőértékekkel indítva a sorozatot, már a 40-50. tag is „távol" van a 0 számtól. Magyarázzuk meg, hogy az így kapott *numerikus eredmény* miért nincsen összhangban az imént jelzett elméleti eredménnyel.

3.1.5. Függvények ábrázolása

Ebben a pontban a *Mathematicá*nak azokat a grafikai lehetőségeit mutatjuk be, amelyek explicit, implicit vagy paraméteres alakban megadott egy és többváltozós függvények megjelenítését teszik lehetővé.

A beépített eljárások a súgójukban (például `?Plot`) megadott szintaxist követve sok esetben a beadott függvény menetének megfelelő szép ábrát készítenek.

A *Mathematica* egyik legfontosabb jellegzetessége az, hogy *opciók* megadásával a felhasználó sokféleképpen befolyásolhatja az ábrázolás menetét, és készíthet az igényeinek megfelelő képet.

A különböző esetekben használható *belső* függvények nevét a szokásos módon kapjuk meg:

```
?*Plot*
 ContourPlot                ListPlot3D
 DensityPlot                ParametricPlot
 ListContourPlot            ParametricPlot3D
 ListDensityPlot            Plot3D
 ListPlot                   Plot
```

A fenti parancs a `Plot` jelsorozatot tartalmazó opcióneveket is megadja. Ezeket itt nem soroltuk fel. Több programcsomag szintén tartalmaz ábrákat rajzoló függvényeket.

• Az ábrázolás mechanizmusa

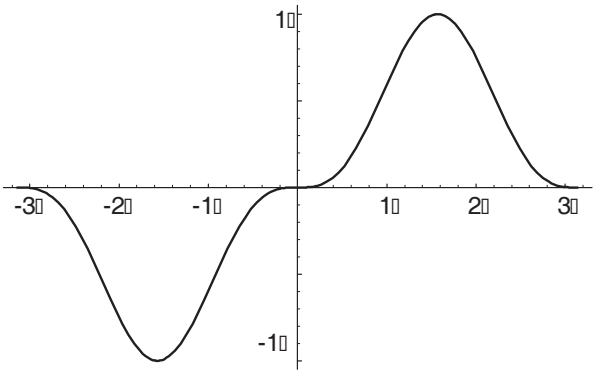
A függvényábrázolás működését az

$$f(x) := \sin^3 x \ (x \in \mathbf{R})$$

függvény példáján érzékeltetjük.

Az f függvény $[-\pi, \pi]$ intervallumra vett leszűkítésének képét így kapjuk meg:

```
Clear[f]
f[x_] := Sin[x]^3
Plot[f[x], {x, -Pi, Pi}]
```



A `Plot` függvény a belső algoritmusa által meghatározott módon választ véges sok pontot (*mintapontok*nak nevezzük ezeket) a $[-\pi, \pi]$ intervallumból. A program kiszámolja ezekben az f függvény helyettesítési értékét. Képezi, majd koordináta-rendszerben ábrázolja az $(x, f(x))$ párokat. A kapott pontokat szakaszokkal köti össze.

A kiválasztási algoritmus az adott ábrázolási intervallum n egyenlő részre osztásával kezdődik, ahol n a `Plot` függvény `PlotPoints` opciójában levő pozitív egész szám. A *Mathematica* ezután kiszámolja az első három mintapontban — x_i-vel $(i = 1, 2, 3)$ jelöljük ezeket — felvett $f(x_i)$ $(i = 1, 2, 3)$ függvényértékeket. Meghatározza az $(x_1, f(x_1))$ és az $(x_2, f(x_2))$, valamint az $(x_2, f(x_2))$ és az $(x_3, f(x_3))$ pontokat összekötő szakaszok szögét. Ha ez a szög 180^o-hoz közeli (pontosabban az eltérése 180^o-tól kisebb, mint a `Plot` függvény `MaxBend` opciójának fokban megadott értéke), akkor a program megrajzolja az első szakaszt. Az ellenkező esetben a felosztást addig finomítja, amíg 180^o-hoz közeli szöget bezáró szakaszokat talál, vagy pedig az elvégzett felosztások száma eléri a `PlotDivision` opcióban szereplő értéket. Az eljárás tehát nagyobb görbületű pontok és szingularitások környezetében sűríti a mintapontokat.

Az f függvény ábrázolásához az opciók alapértelmezésben megadott értékeit használtuk. Ezek a következők:

```
Options[Plot, {PlotPoints, MaxBend, PlotDivision}]
{PlotPoints -> 25, MaxBend -> 10., PlotDivision -> 20.}
```

• Valós–valós függvények ábrázolása

Az f függvény $[x_0, x_1]$ intervallumra vett leszűkítésének a képét a

```
Plot[f[x], {x, x0, x1}]
```

utasítás adja meg. Az f függvényt nem kell előre definiálnunk, `f[x]` helyére a gondolt függvény `x`-ben felvett helyettesítési értékét is beírhatjuk. Az `x0` és az `x1` értékek lehetnek egész számok, valódi törtek, valós számok, bizonyos beépített matematikai állandók (lásd a 3.1.3. pontot) vagy `Sqrt[5]`, `Log[Pi]` típusú szimbolikus számok. Például:

```
Plot[x, {x, 3^(1/3), 2 + Sqrt[7]}]
Plot[x, {x, Log[Pi], Sqrt[5]}]
```

Az ábrázolandó f függvény definíciójában, valamint `Plot` első argumentumában beépített matematikai függvényeket is megadhatunk. Néhány ilyen esetben azonban a `Plot` függvényt, az `Evaluate` belső függvénnyel együtt, a következő módon kell használni:

```
Plot[Evaluate[f[x]], {x, x0, x1}]
```

Ilyen esetekre hívja fel a figyelmet az alábbi példa. Tegyük fel, hogy egy függvény deriváltját szeretnénk felrajzoltatni. Az egyszerűség végett legyen ez a cos függvény, az ábrázolás intervalluma pedig a $[-\pi, \pi]$ intervallum.

Az egyik lehetséges megoldás az, hogy a függvény deriváltját szimbolikusan kiszámoló `D` belső függvényt felhasználva definiáljuk (és például az f szimbólummal jelöljük) a szóban forgó függvényt:

```
Clear[f]
f[x_] = D[Cos[x], x]
-Sin[x]
```

(Figyeljük meg, hogy *azonnali értékadással* (lásd a 3.1.4. pontot) értelmeztük f-et.) A

```
Plot[f[x], {x, -Pi, Pi}]
```

utasítás végrehajtása után a képernyőn megjelenik a várt ábra.

Próbáljuk meg most f-et *késleltetett értékadással* definiálni, majd ábrázolni:

```
Clear[f]
f[x_] := D[Cos[x], x];
Plot[f[x], {x, -Pi, Pi]}
```

A program a függvény képe helyett hibaüzeneteket küld. Ugyanezt kapjuk akkor is, ha a közvetlen utat választjuk:

```
Plot[D[Cos[x], x] {x, -Pi, Pi]}
```

A hibaüzenetek magyarázata a következő. A `Plot` függvény alapértelmezésben nem értékeli ki az első argumentumát, tehát nem végzi el szimbolikusan a kijelölt műveletet. Ehelyett az előző alpontban elmondottak szerint veszi az ábrázolási intervallum első mintapontját, majd ennek az első argumentumba való *formális* behelyettesítése után próbálja azt kiértékelni. Az adott esetben tehát a `D[Cos[-Pi], -Pi]` kifejezést kapja, amit nem tud kiértékelni. A program ezt a tényt közli a felhasználóval és leállítja az ábrázolást.

Az `Evaluate` belső függvénnyel kényszeríthetjük a *Mathematicá*t arra, hogy először szimbolikusan számolja ki a a `Plot` első argumentumát, és az így kapott eredménybe helyettesítse be a mintapontokat. Az alábbi utasítások már a várt eredményt adják:

```
Plot[Evaluate[f[x]], {x, -Pi, Pi}]

Plot[Evaluate[D[Cos[x], x]], {x, -Pi, Pi}]
```

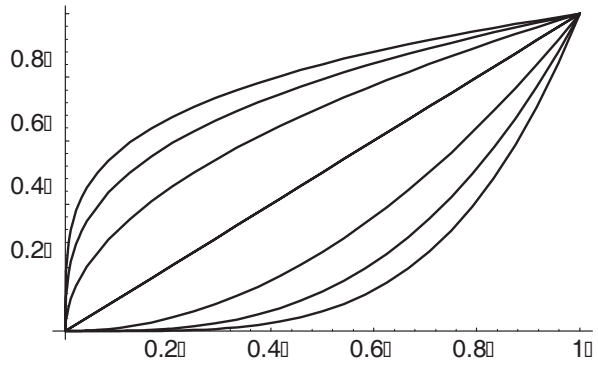
Több, azonos intervallumon definiált függvényt egy koordináta-rendszerben úgy ábrázolhatunk, hogy a `Plot` első argumentumában lista elemeiként adjuk meg a függvények általános helyettesítési értékét:

```
Plot[{x^(1/4), x^(1/3), x^(1/2), x, x^2, x^3, x^4},
     {x, 0, 1}]
```

Különböző intervallumokon értelmezett függvények ábrázolásához pedig a `Show` belső függvényt használhatjuk.

Listákat kényelmesen például a `Table` beépített függvénnyel képezhetünk (lásd a 3.8.1. pontot). Számunkra kevésbé érthető, de fontos tény az, hogy a `Plot` ebben az esetben csak az `Evaluate` függvénnyel rajzolja fel a kívánt ábrát:

```
Plot[Evaluate[
     Table[{x^(1/k), x^k}, {k, 1, 4}]], {x, 0, 1}]
```



• Opciók

Amikor a *Mathematica* felrajzolja egy függvény képét, akkor számos dolgot el kell döntenie. Például azt, hogy hova helyezze a koordinátatengelyeket, mekkora egységeket válasszon ezeken stb. A program igen sok esetben jól választ, és a függvény menetének megfelelő szép ábrát készít. Opciók megadásával azonban a felhasználónak lehetősége van arra is, hogy szükség esetén maga döntsön bizonyos hasonló jellegű kérdésekben.

A rajzolást végző függvények is rendelkeznek opciókkal. Minden opciónak meghatározott neve és egy alapértelmezés szerinti értéke van. Ezt használja a program abban az esetben, ha nem adjuk meg az opciót.

A 2.3.3. pontban már volt arról szó, hogy a `Plot` függvény 27 opcióval rendelkezik. Ezek nevét és az alapértelmezés szerinti értékét az

```
Options[Plot]
```

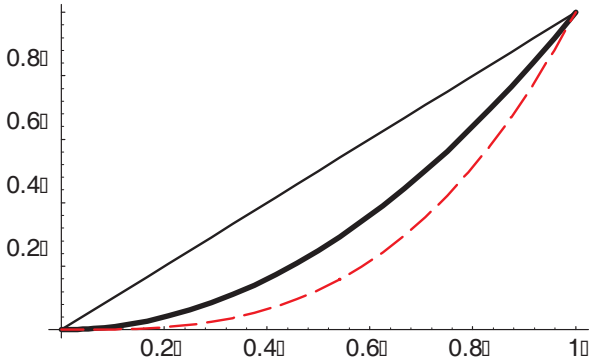
utasítással kapjuk meg. A megengedett opciók közül az

opció neve −> opció értéke

alakban akárhányat megadhatunk. A különbözőket a , jellel kell elválasztanunk.

A következő példák illusztrálják azt, hogy az opciók egyik része az ábrák *csinosítgatását* teszi lehetővé:

```
Plot[{x, x^2, x^3}, {x, 0, 1}, PlotStyle -> {
     {},
     {Thickness[0.008]},
     {Dashing[{.05, .02}], RGBColor[1, 0, 0]}}]
```



Érdemes végrehajtani a következő utasítást is:

```
Plot[x-Log[x], {x, 0.1, 3},
     AxesStyle -> Thickness[0.01],
     AxesOrigin -> {0, 0},
     AxesLabel -> {"x", "y"},
     PlotRange -> {0., 2.5},
     Ticks -> {{0, 1, 2, 3}, {1, 2}},
     Prolog -> Text["x-ln(x)", {1., 2}],
     PlotLabel -> FontForm["Csinosabb ábra",
         {"Palatino-Bold", 12}]]
```

Összetett ábrákat célszerű részenként elkészíteni, majd az eredményeket egyszerre megjeleníteni. Ehhez nyújt segítséget a

DisplayFunction

opció, amelynek lehetséges értékei `Identity` és `$DisplayFunction`. Az

```
abra = Plot[x-Log[x], {x, 0.1, 3},
     DisplayFunction -> Identity]
```

utasítás azt a szándékunkat közli a *Mathematicá*val, hogy készítse el az ábrázoláshoz szükséges számolásokat, ezt raktározza el az `abra` nevű változóban és az eredményt ne jelenítse meg a képernyőn. (Nem elég `Plot[]` után `;`-t írni, mint gondolnánk.) Egy későbbi időpontban a

Show

belső függvény a számítások ismételt elvégzése nélkül viszi képernyőre az elkészített ábrát:

```
Show[abra, AxesLabel -> {"x", "y"},
     DisplayFunction -> $DisplayFunction]
```

• Többváltozós függvények

Ebben az alpontban az explicit alakban megadott $\mathbf{R}^2 \to \mathbf{R}$ és $\mathbf{R}^3 \to \mathbf{R}$ típusú függvények szemléltetéséhez használható

```
DensityPlot                Plot3D
ContourPlot
```

belső függvényekről, a `Graphics`ContourPlot3D`` csomagban lévő

```
ContourPlot3D
```

valamint a `Graphics`Graphics`` programcsomagban elhelyezett

```
ShadowPlot3D
```

külső függvényekről lesz szó.

Az opciók a felhasználónak itt is igen sok lehetőséget kínálnak az igényeinek megfelelő ábra elkészítéséhez. A részletek ismertetésétől eltekintünk. Csupán azt említjük meg, hogy számos esetben megelégszünk azzal a képpel, amit az opciók alapértelmezés szerinti értékeivel kapunk.

Kétváltozós függvényeket térbeli derékszögű koordináta-rendszerben a

`Plot3D`

függvénnyel ábrázolhatunk. A következő utasítás eredménye például a nyeregfelület egy darabjának árnyékolt képe:

```
Plot3D[x^2 - y^2, {x, -2, 2}, {y, -2, 2}]
```

Az opciók megváltoztatásával többek között az árnyékolás módját, az ábrázolás színeit és a nézőpontot is módosítani tudjuk.

A következő példán néhány opció jelentését tanulmányozhatjuk, ha a megadott értékeket változtatgatjuk:

```
Clear[f]
f[x_, y_] := x^4/2 + y^4/2 - 4 x y + 1
Plot3D[f[x, y], {x, -3, 3}, {y, -3, 3},
      PlotRange -> {-3.2, 5},
      BoxRatios -> {1, 1, 1},
      AxesEdge -> {{-1, -1}, {-1, -1}, {-1, -1}},
      Lighting -> False,
      ClipFill -> None]
```

Kétváltozós függvényt és az általa meghatározott felület valamelyik koordinátasíkra vett árnyékolt vetületét a `Graphics`Graphics3D`` programcsomag

`ShadowPlot3D`

függvényével kaphatjuk meg. Olvassuk be ezt a programcsomagot és próbáljuk ki az alábbi utasításokat:

```
<<Graphics`Graphics3D`
ShadowPlot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}]
ShadowPlot3D[Exp[-(x^2 + y^2)], {x, -2, 2}, {y, -2, 2},
        ShadowPosition -> 1]
```

Kétváltozós függvény szintvonalas ábráját a

`ContourPlot`

belső függvénnyel készíthetjük el. A fentebb definiált f függvény szintvonalait így kapjuk meg:

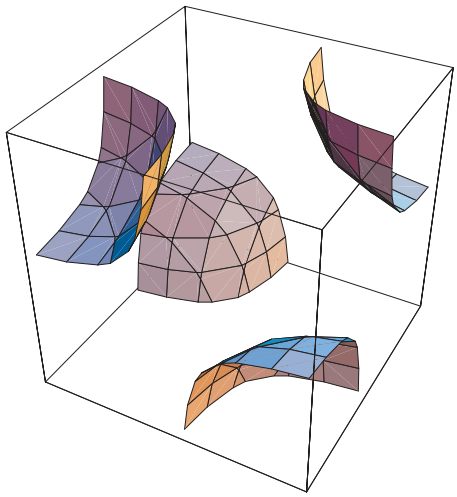
```
ContourPlot[f[x, y], {x, -3, 3}, {y, -3, 3},
       Contours -> {-3, -2, -1, 10},
       ColorOutput -> GrayLevel,
       ColorFunction -> (GrayLevel[1-#^3]&),
       Background -> GrayLevel[1]]
```

A `ContourPlot` háromdimenziós megfelelője a

`ContourPlot3D`

függvény, amely a `Graphics`ContourPlot3D`` programcsomagban található. Használata előtt be kell hívni ezt a programcsomagot:

```
<<Graphics`ContourPlot3D`
ContourPlot3D[
      x y z, {x, -1, 1}, {y, -1, 1}, {z, -1, 1},
      Contours -> {0.1}]
```



Kétváltozós valós értékű függvényt a

`DensityPlot`

belső függvénnyel is ábrázolhatunk. Az értelmezési tartomány adott pontjában a megvilágítás erőssége az ott felvett helyettesítési értéktől függ. A világosabb pontban a függvény értéke nagyobb, mint a sötétebb pontban:

```
DensityPlot[
      Sin[x] Sin[y], {x, 0, 2Pi}, {y, -Pi/2, 3Pi/2},
      PlotPoints -> 50]
Show[%, ColorFunction -> (GrayLevel[1-#^2]&)]
Show[%, ColorFunction -> (RGBColor[1-#, #, 0]&)]
```

`ContourPlot` és `DensityPlot` egyaránt a térképészek által használt ábrát szolgáltat.

• Paraméteres alakban megadott görbék és felületek

Matematikai szempontból $\mathbf{R}^n \to \mathbf{R}^m$ $(n, m \in \mathbf{N})$ típusú függvényeknek tekintjük a paraméteres alakban megadott görbéket és felületeket. Ábrázolhatunk síkbeli görbéket ($n = 1$ és $m = 2$), térgörbéket ($n = 1$ és $m = 3$), valamint kétdimenziós felületeket ($n = 2$ és $m = 3$).

Paraméteres alakban megadott síkgörbét a

`ParametricPlot`

belső függvénnyel ábrázolhatunk. Érdemes kipróbálni például a következő utasításokat:

```
ParametricPlot[{Sin[t], Cos[t]}, {t, 0, 2Pi}]
ParametricPlot[{Sin[t], Cos[2t]}, {t, 0, 2Pi}]
Show[%, Aspectratio -> Automatic]
ParametricPlot[{Cos[t]^3, Sin[t]^3}, {t, 0, 2Pi}]
```

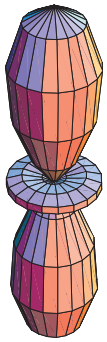
Térgörbe szemléltetéséhez a következő belső függvényt használhatjuk:

`ParametricPlot3D`

```
ParametricPlot3D[
    {Cos[t]^2, Cos[t] Sin[t], Sin[t]}, {t, 0, 2Pi}]
ParametricPlot3D[Evaluate[
    Table[{Cos[t], Sin[t], n t}, {n, 5}]], {t, 0, 2Pi},
    BoxRatios -> {1, 1, 1}]
```

Kétdimenziós felületeket ezzel a függvénnyel is ábrázolhatunk:

```
ParametricPlot3D[
  (3 Cos[t]^2-1)^2{Sin[t] Cos[p], Sin[t] Sin[p], Cos[t]},
  {t, 0, Pi}, {p, 0, 2Pi},
  Boxed -> False, Axes -> False]
```



Megjegyezzük, hogy a `Graphics`ParametricPlot3D`` programcsomag szintén tartalmaz `ParametricPlot3D` függvényt. Ennek használatával a felhasználó maga is szabályozhatja az ábrázolásnál alkalmazott felosztások számát. Érdemes összehasonlítani például a következő utasítások eredményét:

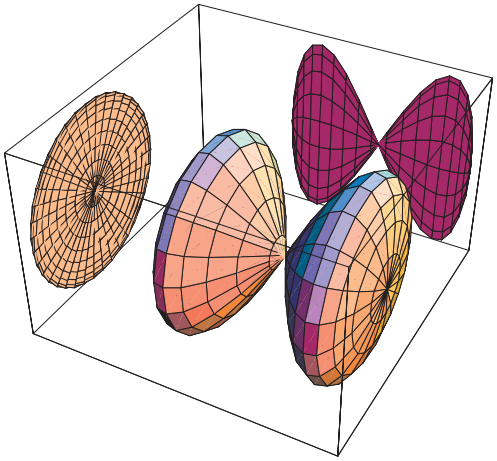
```
ParametricPlot3D[{Cos[u] Cos[v], Sin[u] Cos[v], Sin[v]},
    {u, 0, 2Pi}, {v, -Pi/2, Pi/2}]

<<Graphics`ParametricPlot3D`
ParametricPlot3D[{Cos[u] Cos[v], Sin[u] Cos[v], Sin[v]},
    {u, 0, 2Pi, Pi/20}, {v, -Pi/2, Pi/2, Pi/10}]
```

További példa egy szép felületre:

```
ParametricPlot3D[{Cosh[z] Cos[phi], Cosh[z] Sin[phi],z},
    {z, -2, 2}, {phi, 0, 2 Pi}]
```

Itt említjük meg azt, hogy a `Graphics`Graphics3D`` programcsomag `Shadow` függvényével felületek koordinátasíkokra vett árnyékolt vetületeit állíthatjuk elő:

```
<<Graphics`Graphics3D`
abra = ParametricPlot3D[
     {Sin[t], Sin[2t] Sin[u], Sin[2t] Cos[u]},
     {t, -Pi/2, Pi/2}, {u, 0, 2Pi}, Ticks -> None,
     DisplayFunction -> Identity];
Shadow[abra, DisplayFunction -> $DisplayFunction,
     ZShadow -> False]
```



• **Forgásfelületek**

A `Graphics`SurfaceOfRevolution`` programcsomagban meglevő

`SurfaceOfRevolution`

külső függvénnyel különböző alakban megadott görbék egyenes körüli megforgatásával keletkező felületeket ábrázolhatunk. Érdemes beolvasni ezt a programcsomagot is, és felrajzoltatni néhány forgásfelületet:

```
<<Graphics`SurfaceOfRevolution`

SurfaceOfRevolution[Sin[x], {x, 0, 2Pi},
     ViewVertical -> {1, 0, 0},
     Ticks -> {Automatic, Automatic, {-1., 0, 1.}}]
SurfaceOfRevolution[{1.1 Sin[u], u^2},
     {u, 0, 3Pi/2}, BoxRatios -> {1, 1, 2}]
SurfaceOfRevolution[x^2, {x, 0, 1},
     RevolutionAxis -> {1, 1, 1}]
```

• **Implicit alakban megadott görbék**

Elméleti és gyakorlati szempontból is komoly probléma az implicit alakban megadott görbék vizsgálata. Ehhez a program jelentős támogatást ad.

A `Graphics`ImplicitPlot`` programcsomagban meglevő

ImplicitPlot

függvénnyel két különböző módszerrel ábrázolhatunk implicit alakban megadott *egyváltozós* függvényeket. A módszert a felhasználó a szintaxis alkalmas megadásával választja meg. Olvassuk be ezt a programcsomagot:

```
<<Graphics`ImplicitPlot`
```

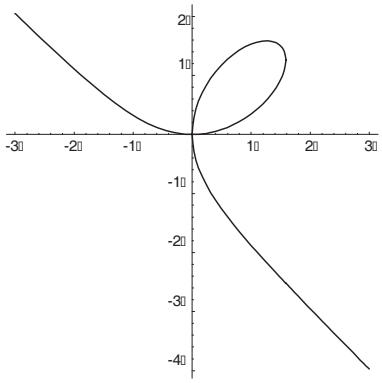
és kérjünk információt a szóban forgó függvényről:

```
?ImplicitPlot
```

A *Mathematica* által adott válaszból kitűnik: megtehetjük, hogy csak az egyik változót tartalmazó intervallumot írjuk be az `ImplicitPlot` argumentumába, de mindkét változóra is kijelölhetünk intervallumot.

Ábrázoljuk az első módszerrel például a Descartes-féle levelet (figyeljünk arra, hogy egyenletet a `==` karakterekkel adunk meg):

```
ImplicitPlot[x^3 + y^3 == 3 x y, {x, -3, 3}]
```



Az `ImplicitPlot` függvény ebben az esetben a `Solve` belső függvénybe (lásd a 3.2. pontot) beépített *szimbolikus* egyenletmegoldó algoritmusokat használja az összetartozó `x, y` számpárok meghatározásához. Pontosabban, ha $f(x, y) = 0$ jelöli az implicit egyenletet, akkor a program az

$$f(x, y) = 0,\ \partial_y f(x, y) = 0$$

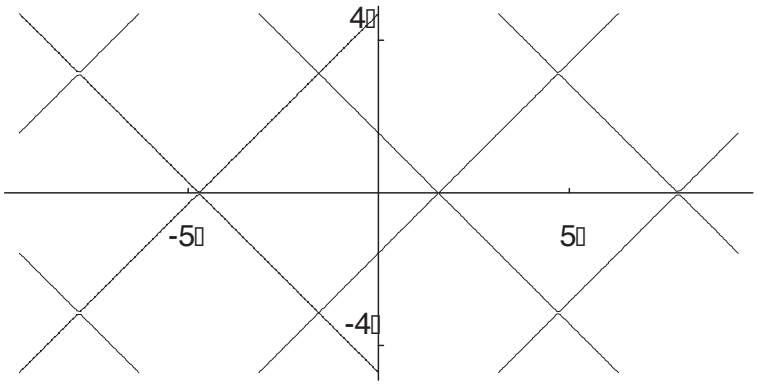
egyenletrendszer szimbolikus megoldásait keresi.

Ha a `Solve` nem találja meg a beadott egyenlet megoldását, akkor ezt közli a felhasználóval, és a program leállítja az ábrázolást. Ez történik például a következő esetben is:

```
ImplicitPlot[Sin[x] == Cos[y], {x, -7 Pi/2, 5 Pi/2}]
```

Az `ImplicitPlot` harmadik argumentumába beírhatunk egy `y` változót tartalmazó intervallumot is. Ekkor a program a `ContourPlot` belső függvénybe beépített *numerikus* algoritmusokat használja. Ezzel a módszerrel az előző példában megadott görbét is ábrázolhatjuk:

```
ImplicitPlot[
    Sin[x] == Cos[y], {x, -3Pi, 3Pi}, {y, -3Pi/2, 3Pi/2},
    AxesOrigin -> {0, 0}, PlotPoints -> 200,
    Compiled -> False, Ticks -> {{-10, -5, 5}, {-3, 3}}]
```



Az implicit módon megadott görbékről legtöbb információt nyújtó ábra elkészítéséhez a változó(ka)t tartalmazó intervallumo(ka)t alkalmas (az adott egyenletből nem leolvasható!) módon kell megválasztani. Számos esetben ehhez sok kísérletet kell elvégezni.

- **Koordináta-rendszerek**

A `Graphics`‘`Graphics`‘ programcsomagban meglevő

`PolarPlot`

függvény polárkoordinátákban megadott görbékhez használható. Ennek az eljárásnak a szintaxisa a `Plot` függvény szintaxisához hasonló. Olvassuk be ezt a programcsomagot, és próbáljuk ki a következő utasítást:

```
PolarPlot[{4/(2 + Cos[t]), 4 Cos[t] - 2}, {t, 0, 2Pi},
    PlotStyle -> {{Dashing[{0.05, 0.02}], GrayLevel[0.33]},
                  {Thickness[0.01], RGBColor[1, 0, 0]}},
    AspectRatio -> Automatic]
```

A `Graphics`ParametricPlot3D`` programcsomag

SphericalPlot3D

függvényével gömbkoordinátákban megadott felületeket ábrázolhatunk:

```
SphericalPlot3D[ 2, {theta, 0, Pi}, {phi, 0, 2Pi}]
```

Hengerkoordinátákban megadott felületek megjelenítését végezhetjük a `Graphics`ParametricPlot3D`` programcsomag

CylindricalPlot3D

függvényével. Például:

```
CylindricalPlot3D[(1+Sin[phi])*r^2, {r, 0, 1}, {phi, 0, 2Pi},
   Boxed -> False, Axes -> False, ViewPoint -> {1.5, -0.5, .2}]
```

• Diszkrét halmazon értelmezett függvények

A következő belső függvényeket használhatjuk:

ListContourPlot	ListPlot
ListDensityPlot	ListPlot3D

A `Graphics`Master`` programcsomag beolvasása után a lehetőségeket az alábbi külső függvények bővítik:

ErrorListPlot	ListSurfacePlot3D
LabeledListPlot	LogLinearListPlot
LinearLogListPlot	LogListPlot
ListAndCurvePlot	LogLogListPlot
ListContourPlot3D	MultipleListPlot
ListFilledPlot	PointParametricPlot3D
ListPlotVectorField	PolarListPlot
ListPlotVectorField3D	ScaledListPlot
ListShadowPlot3D	TextListPlot
ListSurfaceOfRevolution	

• Komplex függvények

A `Graphics`ComplexMap`` programcsomagban meglevő `PolarMap` és `CartesianMap` külső függvényeket komplex változós és komplex értékű függvények szemléltetésére használhatjuk. A

`CartesianMap`

függvény a valós és a képzetes tengellyel párhuzamos hálózat adott függvény által létesített képét rajzolja fel. A

`PolarMap`

függvény pedig az origó középpontú körök és az origóból kiinduló félegyenesek által meghatározott hálózatnak az adott komplex függvény szerinti képét ábrázolja.

A lehetőségeket a következő példán érzékeltetjük. Megmutatjuk, hogy a

$$T := \{z \in \mathbf{C} \,:\, 0 \le \operatorname{Re} z \le 1,\ 0 \le \operatorname{Im} z \le \pi/2\,\}$$

komplex számhalmazt és az exponenciális függvény által létesített képét hogyan lehet szemléltetni.

Először az említett programcsomagot kell behívni:

```
<<Graphics`ComplexMap`
```

Ezután az identitásfüggvény (a *Mathematicá*ban az `Identity` belső függvény) felhasználásával definiáljuk a fenti T tartományt:

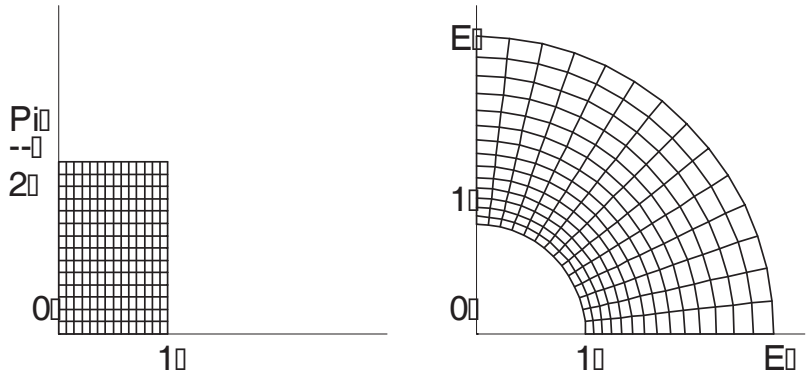
```
tartomany = CartesianMap[Identity, {0, 1}, {0, 1},
     PlotRange -> {{0, 3}, {0, 3}},
     Ticks -> {{0, 1}, {0, Pi/2}},
     DisplayFunction -> Identity];
-Graphics-
```

Figyeljük meg, hogy a `CartesianMap` függvényben az ábrázolandó függvény *nevét* kell megadni. Elkészítjük a tartomány képét:

```
kep = CartesianMap[Exp, {0, 1}, {0, Pi/2},
     PlotRange -> {{0, 3}, {0, 3}},
     Ticks -> {{0, 1, E}, {0, 1, E}},
     DisplayFunction -> Identity];
-Graphics-
```

Most megjelenítjük a tartományt és a képet:

```
Show[GraphicsArray[{tartomany, kep}],
     DisplayFunction -> $DisplayFunction]
```



• **Animáció**

Ábrák sorozatából filmet készíthetünk. Ehhez a

```
Graphics`Animation`
```

programcsomag alábbi függvényei nyújtanak segítséget:

`Animation`	`MovieDensityPlot`
`MoviePlot`	`MovieParametricPlot`
`MoviePlot3D`	`ShowAnimation`
`MovieContourPlot`	`SpinShow`

Ha ablakos felhasználói felülettel dolgozunk, akkor a következőket *is* tehetjük. Gépeljük be például ezt az utasítást:

```
Do[Plot3D[
     Cos[Sqrt[x^2+y^2]+Abs[n-2Pi]]/Sqrt[x^2+y^2+1/4],
     {x, -4Pi, 4Pi}, {y, -4Pi, 4Pi}, PlotPoints -> 26,
     Lighting -> True, PlotRange -> {-2, 2},
     BoxRatios -> {1, 1, 1}, Boxed -> False,
     Axes -> None], {n, 0, 2Pi-2Pi/16, 2Pi/16}]
```

A *Shift+Enter* billentyűpár lenyomása után a *Mathematica* elkezd dolgozni, és kirajzol egy ábrasorozatot. Jelöljük be az egymás után lejátszandó ábrákat (egy kattintással), majd vagy a *Ctrl+Y* billentyűpárral, vagy

a *Graph* menüpont *Animate Selected Graphics* alpontjával indíthatjuk a mozit. Leállítani a szóköz billentyű leütésével lehet, vagy azzal, hogy valahol belekattintunk a képbe. A *Graph* menüpont *Animation* alpontjával az animáció opciói is megváltoztathatók.

3.1.6. Gyakorlatok és feladatok

1. Az alábbi példák illusztrálják, hogy a `LogicalExpand` függvény a teljes diszjunktív normálformát állítja elő:

```
LogicalExpand[(a && b) || (c && d)]
LogicalExpand[(a || b) && (c || d)]
LogicalExpand[(a && b) || (c && !d)]
LogicalExpand[(a || !b) && (c || !d)]
```

2. Egyenletrendszereket egyenletekre így szedhetünk szét:

```
LogicalExpand[{{1, 2}, {3, 7}}.{x, y} == {4, 5}]
x+2y==4 && 3x+7y==5
```

3. Állítsuk elő a következő formulák értéktáblázatát:
a) $\neg a \wedge \neg(\neg a \vee b)$
b) $a \Rightarrow (b \Rightarrow a)$
c) $(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$

4. Igazoljuk, hogy a következő formulák a bennük szereplő változók minden értékére igazak:
a) $(a \wedge (a \Rightarrow b)) \Rightarrow b$
b) $((a \Rightarrow b) \wedge (b \Rightarrow c)) \Rightarrow (a \Rightarrow c)$
c) $((a \wedge b) \vee a) \Leftrightarrow a$
d) $((a \vee b) \wedge a) \Leftrightarrow a$

5. Állítsuk elő egy tetszőleges három- vagy többváltozós logikai függvény igazságtáblázatát a szokásos formában, fejléccel ellátva.

6. A `Union` alkalmazásakor egy opcióban megadhatjuk, hogy mikor tekintünk két elemet azonosnak. Ez némileg módosíthatja a listákból előállított halmazokat. Tanulmányozzuk ezt a jelenséget az alábbi példákon:

```
a = Table[1.0 + 2^k $MachineEpsilon, {k, 0, 8}]
Union[a]
Union[a, SameTest -> SameQ]
Union[a, SameTest -> Equal]
```

7. Írjunk függvényt a szimmetrikus differencia műveletére. Illusztráljuk példákkal, hogy ez a művelet asszociatív és kommutatív.

8. Állítsuk elő egy k-változós ($k = 0, 1, 2, 3, \ldots$) logikai függvény értelmezési tartományának elemeit lexikografikus sorrendben.

9. (A Középiskolai Matematikai Lapok 2897. gyakorlata alapján.) Az x_1, x_2 pozitív egészekből kiindulva képezzük az

$$x_3 := |x_1 - x_2|, \quad x_4 := \min\{|x_1 - x_2|, |x_1 - x_3|, |x_2 - x_3|\}, \ldots$$

sorozatot. (A feladat így szólt: bizonyítsuk be, hogy ha x_1, x_2 egyike sem nagyobb 10000-nél, akkor $x_{21} = 0$.)
Egy lehetséges megoldás:

```
<<DiscreteMathematics`Combinatorica`
parok[lista_] := KSubsets[lista, 2]
absz[par_] := Abs[par[[1]]-par[[2]]]
temp[list_] :=
  Append[#, Composition[f, parok] @@ {#}]& /@ {list}
res[x1_, x2_] :=
  Nest[Composition[Flatten, temp], {x1, x2}, 19]
```

10. Állapítsuk meg a fenti, véges halmazon definiált, homogén `rel` relációról, hogy szimmetrikus-e, és hogy tranzitív-e.

11. Hajtsuk végre az alábbi utasításokat:

```
Interval[{1, 3}, {2, 4}]
Interval[{1, 2}] + Interval[{3, 4}, {8, 10}]
Interval[{1, 4}] + Interval[{3, 6}, {8, 10}]
Interval[1.] - 1
Interval[{-2.22045 10^-16, 2.22045 10^-16}]
Interval[{-4.44089 10^-16, 4.44089 10^-16}]
1/Interval[{-2, 3}]
a = Interval[{1, 3}, 4, {7, 12}];
b = Interval[{2, 4}, {10, 20}];
IntervalIntersection[a, b]
Min[Interval[{1, Infinity}]]
Interval[{3, 6}] >= Interval[{-1, 3}]
Sin[Interval[{-1, 1}, {2, 4}]]
Abs[Interval[{-4, -3}, {2, E}]]
Max[Abs[Interval[{-2Pi, 4}]]]
```

3.2. Matematikai kifejezések

Sokszor szükségünk lehet arra, hogy egy megadott vagy a *Mathematica* által származtatott matematikai kifejezést más (például egyszerűbb) alakban írjunk fel. Ennek a témakörnek a jelentőségét mutatja az a tény is, hogy számos matematikai problémát úgy oldunk meg, hogy kifejezéseket alkalmas azonosságok felhasználásával olyan alakra hozunk, amelyről a megoldás már „egyszerűen leolvasható".

Ebben a szakaszban a programnak azokat a függvényeit ismertetjük, amelyek valamely matematikai kifejezés — így nevezzük a 3.1.4. pontban megemlített beépített matematikai függvényekből a szokásos függvényműveletek véges sokszori alkalmazásával nyert $\mathbf{C}^n \to \mathbf{C}$ ($n \in \mathbf{N}$) típusú függvény egy általános helyettesítési értékét — *szimbolikus* átalakításához nyújtanak segítséget. Foglalkozni fogunk algebrai és trigonometrikus polinomokkal, racionális, irracionális és transzcendens kifejezésekkel.

A programot igen sok azonosság (transzformációs szabály) alkalmazására megtanították. Különböző beépített függvények általában különböző típusú azonosságokat alkalmaznak. Egy adott kifejezés igényeinknek megfelelő átalakításához használható eljárás megtalálása sok esetben nem egyszerű feladat.

A *Mathematica* egy *végtelen kiértékelő rendszer*. Ez azt jelenti, hogy a beadott matematikai kifejezésre addig alkalmazza a beépített vagy a felhasználó által definiált transzformációs szabályokat, ameddig ismételten azonos alakú kifejezést nem kap.

Itt említjük meg az egyszerűsítés problémáját. Matematikai kifejezéseket sokszor a „legegyszerűbb alakra" szeretnénk hozni. Ilyenkor a

```
Simplify
```

függvényt használhatjuk. Vegyük figyelembe azonban azt, hogy nincs pontos definíció arra, hogy két (matematikai szempontból egyenlő) kifejezés közül melyik az „egyszerűbb alakú". A `Simplify` függvénybe beépített algoritmusok a megadott kifejezést transzformációs szabályok felhasználásával átalakítják. Eredményként a legkevesebb részből álló alakot (a

```
LeafCount
```

függvény adja meg a „méretet") kapjuk meg. Ez az oka annak, hogy bonyolultabb esetekben a `Simplify` függvény sokáig dolgozhat.

Figyeljük meg például azt, hogy a *Mathematica* az $(a^n - b^n)/(a-b)$ törtet különböző n természetes számok esetén hogyan egyszerűsíti:

```
kifejezes[n_] := (a^n - b^n)/(a - b)
egyszerusit[n_] := Simplify[kifejezes[n]]
```

A fenti törtet a `Simplify` függvény csak $n \leq 4$ esetén alakítja át. Például $n = 4$-re ezt kapjuk:

```
{kifejezes[4], LeafCount[%]}
```

$$\left\{\frac{a^4 - b^4}{a - b},\ 17\right\}$$

```
{egyszerusit[4], LeafCount[%]}
```

$$\{a^3 + a^2\, b + a\, b^2 + b^3,\ 17\}$$

Ha $n > 5$, akkor változatlan formában kapjuk vissza a törtet, ugyanis az $(a - b)$-vel való osztás után adódó kifejezés mérete nagyobb lenne, mint

```
LeafCount[kifejezes[n]]
17
```

3.2.1. Algebrai polinomok

A matematikának napjainkban rohamosan fejlődő egyik ága, a *szimbolikus számolások elmélete* (a *komputeralgebra*) foglalkozik a számítógépen megvalósítható szimbolikus algoritmusok elméletével. Jelenleg a matematikai kifejezések közül bizonyos értelemben a legegyszerűbbeknek, a polinomoknak az elmélete a legjobban kidolgozott terület. A *Mathematica* is egy- és többváltozós algebrai polinomokkal végzett műveletek elvégzéséhez nyújtja a legtöbb segítséget. Várható, hogy új elméleti eredmények a *Mathematica* későbbi változatainak a lehetőségeit is bővíteni fogják.

• Polinomok kifejtése

A beadott polinomot a különböző szimbólumokkal jelölt változók közötti lehetséges összeadások és kivonások automatikus elvégzése után kapjuk vissza. Szorzatok és hatványok kifejtésére az `Expand` függvény szolgál. Az

```
Expand[polinom]
```

utasítás hatására a *Mathematica* az egy- vagy többváltozós `polinom`ban elvégzi a szorzásokat és a pozitív egész kitevőjű hatványozásokat, majd a lehetséges összevonások elvégzése után adja meg az eredményt:

```
Expand[6*(x-1)^3 + 4*(x-2)^2 + 7*x + 16]
                          2    3
26 + 9 x - 14 x  + 6 x

Expand[(1 + 3x + 4y)*(1 - x + 2y)^3]
                          2    3    4
1 - 6 x  + 8 x  - 3 x  + 10 y - 6 x y -
                          2    3
   18 x  y + 14 x  y +
                          2    2    2  2    3      3      4
   36 y  - 24 x y  - 12 x  y  + 56 y  - 24 x y  + 32 y
```

Többváltozós polinom kifejtését bizonyos változókra korlátozva is elvégezhetjük:

```
Expand[(x + 1)^2*(y + 1)^2*(z + 1)^2, x]
                2          2                2          2
(1 + y)  (1 + z)  + 2 x (1 + y)  (1 + z)  +
    2          2          2
   x  (1 + y)  (1 + z)
```

Lehetőség van arra is, hogy több (de nem az összes) változó szerint fejtsünk ki egy polinomot. Ha x_1, x_2 és y jelöli a változókat, akkor az

$$(x_1 + 1)^2(x_2 + 1)^2(y + 1)^2$$

polinom x_1 és x_2 szerinti kifejtését például így kapjuk meg:

```
Expand[(x[1] + 1)^2*(x[2] + 1)^2*(y + 1)^2, x[_]]
                2                2          2
(1 + y)  + 2 (1 + y)  x[1] + (1 + y)  x[1]  +
                2          2
    2 (1 + y)  x[2] + 4 (1 + y)  x[1] x[2] +
                2  2             2          2
    2 (1 + y)  x[1]  x[2] + (1 + y)  x[2]  +
                2          2  2          2  2
    2 (1 + y)  x[1] x[2]  + (1 + y)  x[1]  x[2]
```

A következő függvények polinomok részeinek kiválasztásához nyújtanak segítséget:

`Coefficient`	`FactorTerms`
`CoefficientList`	`Length`
`Collect`	`PolynomialQ`
`Exponent`	`Variables`

Számos speciális polinom beépített függvényként szerepel a *Mathematicá*ban:

`BernoulliB`	`GegenbauerC`
`ChebyshevT`	`HermiteH`
`ChebyshevU`	`JacobiP`
`CyclotomicC`	`LaguerreL`
`EulerE`	

• Polinomok szorzattá alakítása

Polinomok különböző típusú szorzattá alakításához a

`Factor`	`FactorSquareFreeList`
`FactorList`	`FactorTerms`
`FactorSquareFree`	`FactorTermsList`

függvényeket használhatjuk.

Ezek közül csak a `Factor` függvény által nyújtott lehetőségeket ismertetjük. A

```
Factor[polinom]
```

utasítás a `polinom`ot **Z** felett irreducíbilis tényezők szorzataként írja fel abban az esetben, ha az együtthatók egész számok vagy valódi törtek:

```
Factor[2x^3 + x^2 - 4x - 12]
(-2 + x) (6 + 5 x + 2 x^2)

Factor[x^2 - (13/14)x + 3/14]
(3 - 7 x) (1 - 2 x)
----------------------
          14

Factor[4 (x*y + u*v)^2-(x^2 + y^2 - u^2 - v^2)^2]
(-u + v - x - y) (-u - v - x + y) (u + v - x + y)
    (-u + v + x + y)
```

Ha a polinom együtthatói között van explicit tizedesponttal megadott szám is, akkor a `Factor` függvény numerikus módszert használva határozza meg a polinom gyökeinek egy közelítő értékét, és ezekkel adja meg a felbontást:

```
Factor[x^2 - 4x + 2]
```

$$2 - 4\,x + x^2$$

```
Factor[x^2 - 4x + 2.]
(-3.41421 + x) (-0.585786 + x)
```

A következő példák azt is mutatják, hogy a program a bemenő adatok szintaxisától függően választja meg a kiértékelésnél felhasznált algoritmust:

```
Factor[2x^2 + 5x + 6]
```

$$6 + 5\,x + 2\,x^2$$

```
Factor[2x^2 + 5.x + 6]
```

$$2\,(3 + 2.5\,x + x^2)$$

```
Factor[2x^2 + 5x + 6.+ 0. I]
2 (1.25 - 1.19896 I + x) (1.25 + 1.19896 I + x)
```

Polinomfaktorizációt a Gauss-egészek gyűrűjében a

```
Factor[polinom, GaussianIntegers -> True]
```

utasítással kapunk. Például:

```
Factor[x^2 + 1]
```

$$1 + x^2$$

```
Factor[x^2 + 1, GaussianIntegers -> True]
(-I + x) (I + x)
```

A $\mathbf{Z}_p$ véges test feletti polinomgyűrűben a következő utasítással bonthatunk fel polinomokat irreducíbilis tényezők szorzatára:

```
Factor[polinom, Modulus -> p]
```

```
Factor[x^4 + 1, Modulus -> 2]
```

$$(1 + x)^4$$

```
Factor[x^4 + 1, Modulus -> 17]
(2 + x) (8 + x) (9 + x) (15 + x)
```

Polinomfaktorizációval nyert tényezőkből álló listát készíthetünk a következő függvényekkel:

```
FactorList                FactorTermsList
FactorSquareFreeList
```

Figyeljük meg azt is, hogy a *Mathematica* milyen formában adja meg a tényezőket:

```
FactorList[x^3 - x^2 - 5x - 3]
{{-3 + x, 1}, {1 + x, 2}}

FactorList[x^4 + 1, Modulus -> 13]
                 2          2
{{5 + x , 1}, {8 + x , 1}}

FactorList[x^2 + 1, GaussianIntegers -> True]
{{-I + x, 1}, {I + x, 1}}
```

• Műveletek polinomokkal

Képezhetjük polinomok összegét, szorzatát és kompozícióját. A

```
Decompose
```

eljárást is használhatjuk. Ennek a függvénynek a segítségével állíthatunk elő egy egész együtthatós polinomot két alacsonyabb fokszámú polinom kompozíciójaként (lásd a 3.1.4. pont Műveletek függvényekkel című alpontját).

A $\mathbf{Z}_p$ véges test feletti polinomgyűrűben is számos műveletet elvégezhetünk. Például a

```
PolynomialMod[polinom, p]
```

utasítás eredménye a `polinom`-mal mod p kongruens polinom, amelynek együtthatói a $[0, p-1]$ intervallumból valók:

```
PolynomialMod[x^5 - 17x^3 + 37x^2 + 12x + 19, 11]
                  2    3    5
8 + x + 4 x  + 5 x  + x

PolynomialMod[(2x + 3y) (x - 7)^2, 5]
                  2    3              2
3 x + 2 x  + 2 x  + 2 y + 3 x y + 3 x  y
```

Polinomok *rezultánsát* (lásd például [45]) számolja ki a `Resultant` függvény:

```
Resultant[x^2 - 6x + 2, x^2 + x + 5, x]
233

Resultant[x^2 - 4x - 5, x^2 - 7x + 10, x]
0
```

Emlékeztetünk arra, hogy két egyváltozós polinom rezultánsa az együtthatóikból képzett Sylvester-féle mátrixnak a determinánsa. A Sylvester-féle kritérium azt állítja, hogy két egész együtthatós algebrai polinomnak pontosan akkor van közös gyöke, ha a rezultánsuk 0-val egyenlő.

• Gröbner-bázisok

A szimbolikus programcsomagok polinomok kezelésénél a *Gröbner-bázisok* elméletét alkalmazzák. B. Buchberger [14] 1965-ben megírt Ph.D. disszertációjában egyenletrendszerek megoldása céljából vezette be a *Gröbner-bázis* fogalmát. (W. Gröbner volt a témavezetője.) Az ezután megindult intenzív kutatások azt mutatták, hogy ez a fogalom és a dolgozatban leírt algoritmusok más típusú feladatok (például paraméterek kiküszöbölése, kongruenciák megoldása, polinom helyettesítési értékének meghatározása adott mellékfeltételek esetén) megoldásánál is jól használhatók.

Az elmélet iránt érdeklődő Olvasó figyelmébe ajánljuk a [7] és a [28] könyvet, valamint a [35] és a [58] dolgozatot. A továbbiakban az alapprobléma és néhány eredmény megfogalmazása után csupán *érzékeltetni* szeretnénk azt, hogy a *Mathematica* hogyan használja ezt az elméletet különböző feladatok megoldásánál.

Jelölje $\mathbf{Q}[x]$ $(x = (x_1, \cdots, x_n),\ n \in \mathbf{N})$ a $\mathbf{Q}$ számtest feletti n változós polinomgyűrűt. A $\mathbf{Q}[x]$ (kommutatív) gyűrű I nemüres részhalmazát *ideál*nak nevezzük, ha egyrészt minden $p, q \in I$ esetén $p - q \in I$ teljesül, másrészt pedig minden $p \in I$ és $q \in \mathbf{Q}[x]$ mellett a pq szorzat is I-hez tartozik.

Bármely $P := \{p_1, \cdots, p_k\} \subset \mathbf{Q}[x]$ véges polinomrendszer esetén a

$$(P) := (p_1, \cdots, p_k) := \{ \sum_{i=1}^{k} a_i p_i \ :\ a_i \in \mathbf{Q}[x] \} \subset \mathbf{Q}[x]$$

halmaz nyilván $\mathbf{Q}[x]$ egy (P *által generált*) ideálja. Magát a P halmazt az ideál *bázis*ának szokás nevezni. (Ez az általánosan elfogadott szóhasználat félrevezető, ha a lineáris algebra bázisfogalmával vetjük össze.)

Hilbert bázistételének egyik következménye az, hogy $\mathbf{Q}[x]$ minden ideáljának van véges bázisa és $\mathbf{Q}[x]$ minden ideálja a fenti alakú.

Polinomokkal kapcsolatos számos problémát a következő alapfeladatra lehet redukálni:

> *Adott egy $P \subset \mathbf{Q}[x]$ véges halmaz és egy tetszőleges $q \in \mathbf{Q}[x]$ polinom. Hogyan lehet véges sok lépésben eldönteni azt, hogy a q polinom hozzátartozik-e a P által generált (P) ideálhoz?*

Egyváltozós polinomok (azaz $n = 1$) esetén bármely $P \subset \mathbf{Q}[x]$ véges halmaz által generált (P) ideál egyetlen elemmel, a P-hez tartozó polinomok legnagyobb közös osztójával (jelölje ezt h) is generálható. Így a fenti kérdésre a válasz a következő: q pontosan akkor eleme a (P) ideálnak, ha q osztható a h polinommal. A bizonyítás alapvető eszköze a jól ismert euklideszi algoritmus.

Többváltozós polinomok esetében a helyzet jóval bonyolultabb. Nem egyszerű ugyanis a (P) ideál szerkezetére jól használható jellemzést adni abban az esetben, amikor P tetszőleges polinomrendszer.

A nehézséget egyrészt az okozza, hogy az ideálok ekkor már nem generálhatók egyetlen elemmel, ezért az euklideszi algoritmust is általánosítani kell.

Másrészt egyismeretlenes polinomokra a tagok elrendezésének két természetes módját ismerjük: az ismeretlen növekvő és csökkenő hatványai szerintit. Többismeretlenes polinomok esetén azonban a

$$T := T(x_1, \cdots, x_n) := \{x_1^{\alpha_1} \cdots x_n^{\alpha_n} \ : \ \alpha_i \in \mathbf{N}\}$$

halmazon többféle módon lehet rendezési relációt értelmezni, ami egy adott többváltozós polinomban a tagok „lényegesen" különböző sorrendjét indukálja. Itt csupán a *lexikografikus rendezés*t (ezt a $\leq_L$ szimbólummal jelöljük) értelmezzük:

$$x_1^{\alpha_1} \cdots x_n^{\alpha_n} \leq_L x_1^{\beta_1} \cdots x_n^{\beta_n} \qquad (x_1^{\alpha_1} \cdots x_n^{\alpha_n},\ x_1^{\beta_1} \cdots x_n^{\beta_n} \in T)$$

pontosan akkor, ha $(\alpha_1, \cdots, \alpha_n) = (\beta_1, \cdots \beta_n)$, vagy pedig létezik olyan $i \in \{1, 2, \cdots, n\}$ természetes szám, amelyre

$$\alpha_j = \beta_j \quad (1 \leq j \leq i-1) \quad \text{és} \quad \alpha_i < \beta_i.$$

Ha $\leq$ egy alkalmas rendezési reláció a T-n, akkor egy adott polinom $\leq$ rendezésre vonatkozó *főtagjá*nak nevezzük a polinom együttható nélkül vett ($\leq$-re vonatkozó) legmagasabb fokú tagját.

Az euklideszi algoritmus többváltozós általánosítását már nem polinomosztásnak, hanem *polinomredukciós eljárás*nak nevezik. Ezzel kapcsolatban a következő alapvető állítás fogalmazható meg:

Legyen $P := \{p_1, \cdots, p_k\} \subset \mathbf{Q}[x]$ *tetszőleges halmaz és* $f \in \mathbf{Q}[x]$ *adott polinom. Ekkor léteznek olyan* $a_i \in \mathbf{Q}[x] \quad (i = 1, \cdots, k)$ *és* $r \in \mathbf{Q}[x]$ *polinomok, amelyekre az*

$$f = \sum_{i=1}^{k} a_i p_i + r$$

egyenlőség teljesül, és $a_i p_i$ $(i = 1, \cdots, k)$ *főtagja nem nagyobb, mint* f *főtagja.*

Az a_i $(i = 1, \cdots, k)$ és r polinomot a [7] monográfia 199. oldalán leírt *algoritmussal* lehet meghatározni. További problémát jelent azonban az, hogy az r „maradék"-polinom nincs egyértelműen meghatározva abban az esetben, ha *tetszőleges* P polinomrendszerből indulunk ki. B. Buchberger a már megemlített [14] dolgozatában igazolta, hogy tetszőleges véges $P \subset \mathbf{Q}[x]$ halmazhoz létezik olyan $G \subset \mathbf{Q}[x]$ véges polinomrendszer (ezt nevezte el *Gröbner-bázis*nak), amely egyrészt ugyanazt az ideált generálja, mint P, másrészt a fenti állítást a P helyett a G bázisra alkalmazva az r polinom már egyértelműen meg van határozva. Ilyen jellegű bázis *létezését* H. Hironaka [36] is bebizonyította 1964-ben. B. Buchberger azonban jól használható algoritmust is konstruált a G bázis *előállítására*.

Megjegyezzük még azt is, hogy *egyváltozós polinomok esetében ez az eljárás az euklideszi algoritmust, elsőfokú többváltozós polinomok esetében pedig a Gauss-féle eliminációs eljárást szolgáltatja.*

A Gröbner-bázis fogalmának több lehetséges ekvivalens definíciója közül a következőt emeljük ki. Jelöljön $\leq$ egy alkalmas rendezést (például a lexikografikust) a T halmazon. A $\mathbf{Q}[x]$ polinomgyűrű I ideáljának egy nemüres G véges részhalmazát akkor nevezzük az I ideál $\leq$-re vonatkozó *Gröbner-bázis*ának, ha az ideál minden polinomjának ($\leq$-re vonatkozó) főtagja osztható a G halmaz valamely polinomjának főtagjával.

Igazolható, hogy T-n adott alkalmas $\leq$ rendezés esetén a $\mathbf{Q}[x]$ polinomgyűrű bármely I ideáljához létezik Gröbner-bázis, és ez a főegyütthatóktól eltekintve egyértelműen meg van határozva (lásd [7], Theorem 5.41., Theorem 5.43.).

A *Mathematica* `GroebnerBasis` függvénye a $\mathbf{Q}[x]$ polinomgyűrűben a lexikografikus rendezést használva határozza meg adott P polinomrendszer által generált (P) ideál Gröbner-bázisát. A változók sorrendjét a függvény második argumentumában adhatjuk meg. Az eredmény ettől általában természetesen függ:

```
GroebnerBasis[{x - y^2 + 2, x^2 - y^2 -1}, {x, y}]
{3 - 5 y^2 + y^4, 2 + x - y^2}
```

```
GroebnerBasis[{x - y^2 + 2, x^2 - y^2 -1}, {y, x}]
{-3 - x + x^2, -2 - x + y^2}
```

Az `AlgebraicRules`, az `Eliminate` és a `Solve` eljárás alkalmazza a Gröbner-bázisok elméletét.

A 2.3.2. pontban a mintázatokkal kapcsolatban volt arról szó, hogy a program egy transzformációs szabály kiértékelésénél a kifejezésnek a *struktúráját*, és nem annak *matematikai jelentését* vizsgálja:

```
1 + x^2 + x^4 /. x^2 -> a
1 + a + x^4
```

A matematikában megszokott módon vezethetünk be új változókat az

`AlgebraicRules`

belső függvény segítségével:

```
ujvalt = AlgebraicRules[x^2 == a, {x, a}]
{x^2 -> a}

1 + x^2 + x^4 /. ujvalt
1 + a + a^2

1 + x^3 + x^7 /. ujvalt
1 + a x + a^3 x
```

Ezt az eljárást használhatjuk abban az esetben is, amikor egy polinomnak adott mellékfeltételek mellett szeretnénk meghatározni a helyettesítési értékét.

Tekintsük például a következő feladatot. Tegyük fel, hogy az a, b és a c valós számok kielégítik az

$$a+b+c=3, \qquad a^2+b^2+c^2=9, \qquad a^3+b^3+c^3=24$$

feltételeket. Határozzuk meg az $a^4+b^4+c^4$ összeget.

A megoldás a *Mathematicá*val ilyen egyszerű. Először a feltételeket adjuk meg:

```
feltetelek = AlgebraicRules[{a + b + c == 3,
    a^2 + b^2 + c^2 == 9, a^3 + b^3 + c^3 == 24},
    {a, b, c}]
```

```
  3          2   2                     2
{c  -> -1 + 3 c , b  -> 3 b + 3 c - b c - c ,
  a -> 3 - b - c}
```

A kérdezett összeget pedig így határozhatjuk meg:

```
a^4 + b^4 + c^4 /. feltetelek
69
```

A Gröbner-bázisok elmélete *általános módszert* szolgáltat az ilyen típusú feladatok megoldásához. Nézzük meg azt, hogy a program hogyan alkalmazta ezt a módszert a fenti feladat megoldásánál.

Az `AlgebraicRules` eljárás a

$$P := \{a + b + c - 3,\ a^2 + b^2 + c^2 - 9,\ a^3 + b^3 + c^3 - 24\} \subset \mathbf{Q}[a, b, c]$$

polinomrendszer által generált (P) ideál Gröbner-bázisát számolta ki. Valóban:

```
GroebnerBasis[{a + b + c - 3, a^2 + b^2 + c^2 -9,
    a^3 + b^3 + c^3 -24}, {a, b, c}]
         2    3              2                2
{1 - 3 c  + c , -3 b + b  - 3 c + b c +c ,
    -3 + a + b + c}
```

(Figyeljük meg, hogy az `AlgebraicRules` függvény transzformációs szabályként adja meg a Gröbner-bázist.)

Ha ezután az $a^4 + b^4 + c^4$ polinomra a korábban megemlített polinomredukciós eljárást a kiszámított Gröbner-bázissal alkalmazzuk, akkor éppen az abban szereplő, r-rel jelölt „maradék"-polinom lesz a feladat megoldása. A fenti második utasítás hatására a *Mathematica* ezt az r polinomot határozta meg.

Hasonló módon működik az

`Eliminate`

eljárás is, amelynek segítségével polinomokat tartalmazó egyenletekből paramétereket eliminálhatunk. Például:

```
Eliminate[{f == x^5 + y^5, a == x + y, b == x y}, {x, y}]
 5      3         2
a  - 5 a  b + 5 a b  == f
```

Ezt feladatot így is megoldhatjuk:

```
fel = AlgebraicRules[{x + y == a, x y == b}, {x, y}]
  2
{y  -> -b + a y, x -> a - y}
```

```
x^5 + y^5 /. fel
```

$$a^5 - 5\,a^3\,b + 5\,a\,b^2$$

Az `AlgebraicRules` most is a Gröbner-bázist használja:

```
GroebnerBasis[{x + y - a, x*y - b}, {x, y}]
```

$$\{b - a\,y + y^2,\ -a + x + y\}$$

A 3.3.2. pontban lesz szó arról, hogy a `Solve` függvény hogyan alkalmazza a Gröbner-bázisok elméletét polinomokat tartalmazó egyenletrendszerek megoldásánál.

3.2.2. Racionális kifejezések

Ebben a pontban polinomok hányadosaként megadott kifejezések átalakításához használható eljárásokról lesz szó. A program számos olyan függvényt tartalmaz, amelynek segítségével racionális kifejezéseket különböző alakban írhatunk fel. Ezek a következők:

Apart	ExpandNumerator
Cancel	Factor
Denominator	Numerator
Expand	Simplify
ExpandAll	Together
ExpandDenominator	

A *Mathematica* racionális kifejezéseknél a lehetséges összevonásokat automatikusan elvégzi. A számláló és a nevező közös tényezőivel is automatikusan egyszerűsít:

```
t = ((-3x+x^2-x)^2*(1-x)^2)/((x+1+2x)^2*(1-x))
```

$$\frac{(1 - x)\,(-4\,x + x^2)^2}{(1 + 3\,x)^2}$$

```
{Numerator[t], Denominator[t]}
```

$$\{(1 - x)\,(-4\,x + x^2)^2,\ (1 + 3\,x)^2\}$$

Szorzások és hatványozások elvégzéséhez az alábbi függvényeket használhatjuk:

```
{ExpandNumerator[t], ExpandDenominator[t]}
```

$$\left\{\frac{16\,x^2 - 24\,x^3 + 9\,x^4 - x^5}{(1 + 3\,x)^2},\ \frac{(1 - x)\,(-4\,x + x^2)^2}{1 + 6\,x + 9\,x^2}\right\}$$

```
Expand[t]
```

$$\frac{16\,x^2}{(1 + 3\,x)^2} - \frac{24\,x^3}{(1 + 3\,x)^2} + \frac{9\,x^4}{(1 + 3\,x)^2} - \frac{x^5}{(1 + 3\,x)^2}$$

```
ExpandAll[t]
```

$$\frac{16\,x^2}{1 + 6\,x + 9\,x^2} - \frac{24\,x^3}{1 + 6\,x + 9\,x^2} + \frac{9\,x^4}{1 + 6\,x + 9\,x^2} - \frac{x^5}{1 + 6\,x + 9\,x^2}$$

A racionális kifejezéseket közös nevezőre hozhatjuk és parciális törtekre is bonthatjuk:

```
u = (-4x+x^2)/(-x+x^2) + (-4+3x+x^2)/(-1+x^2)
```

$$\frac{-4\,x + x^2}{-x + x^2} + \frac{-4 + 3\,x + x^2}{-1 + x^2}$$

```
Together[u]
```

$$\frac{2\,(-4 + x^2)}{(-1 + x)\,(1 + x)}$$

```
Apart[u]
```

$$2 - \frac{3}{-1 + x} + \frac{3}{1 + x}$$

Többváltozós kifejezések parciális törtekre bontását különböző változóra is kérhetjük:

```
v = (x^2 + y^2)/(x + x y)
```

$$\frac{x^2 + y^2}{x + x\,y}$$

```
Apart[v, x]
```

$$\frac{x}{1 + y} + \frac{y^2}{x\ (1 + y)}$$

```
Apart[v, y]
```

$$-(\frac{1}{x}) + \frac{y}{x} + \frac{1 + x^2}{x\ (1 + y)}$$

A következő példákkal a `Factor` és a `Cancel` függvény működését érzékeltetjük:

```
Cancel[u]
```

$$\frac{-4 + x}{-1 + x} + \frac{4 + x}{1 + x}$$

```
Factor[%]
```

$$\frac{2\ (-2 + x)\ (2 + x)}{(-1 + x)\ (1 + x)}$$

3.2.3. Komplex változós kifejezések

Az előző pontokban ismertetett függvényekkel komplex *számokat* tartalmazó kifejezéseket is átalakíthatunk. A *Mathematica* ezekben az esetekben a változókat *formális szimbólumoknak* tekinti, és a kijelölt műveletek végrehajtása, valamint a lehetséges összevonások elvégzése után adja meg az eredményt.

Szükségünk lehet azonban arra is, hogy kijelöljük egy *matematikai kifejezés* valós, illetve képzetes részét. Ilyen feladatok megoldásához a

`ComplexExpand`

függvényt használhatjuk. Pontosan meg kell azonban mondanunk azt, hogy melyik szimbólumot tekintjük valós változónak és melyiket komplexnek. A változók specifikálásának módját a program készítői a fenti függvény második (opcionális) argumentumának megadásával oldották meg.

A `ComplexExpand[kifejezes]` utasítás kiértékelésénél a program a `kifejezes` mindegyik szimbólumát valós változónak tekinti, és a kijelölt műveletek elvégzése után (valós rész) $+$ i (képzetes rész) alakban próbálja megadni az eredményt:

```
ComplexExpand[(a + I b)/(c + I d)]
```

$$\frac{a\,c}{c^2+d^2}+\frac{b\,d}{c^2+d^2}+I\left(\frac{b\,c}{c^2+d^2}-\frac{a\,d}{c^2+d^2}\right)$$

```
ComplexExpand[Sin[x + I y]]

Cosh[y] Sin[x] + I Cos[x] Sinh[y]
```

A `ComplexExpand` függvény második argumentumában kapcsos zárójelek között sorolhatjuk fel azokat a szimbólumokat, amelyeket komplex változóknak tekintünk. A fel nem tüntetett szimbólumokat a program valós változóknak tekinti:

```
ComplexExpand[a + I b, {a}]

I (b + Im[a]) + Re[a]

ComplexExpand[Exp[I z], {z}]
```

$$\frac{\mathrm{Cos}[\mathrm{Re}[z]]}{E^{\mathrm{Im}[z]}}+\frac{I\,\mathrm{Sin}[\mathrm{Re}[z]]}{E^{\mathrm{Im}[z]}}$$

A `z` komplex változót valós paraméterekkel többféle módon is kifejezhetjük. A *Mathematica* alapértelmezésben a `Re[z] + I Im[z]` algebrai alakot használja.

A `ComplexExpand` függvény `TargetFunctions` opciójának (ennek lehetséges értékei: `Abs`, `Arg`, `Conjugate`, `Im`, `Re` és `Sign`) megváltoztatásával a `z` változót például trigonometrikus alakban így írhatjuk fel:

```
Factor[ComplexExpand[z, {z}, TargetFunctions -> {Abs, Arg}]]

Abs[z] (Cos[Arg[z]] + I Sin[Arg[z]])
```

Komplex *szám* valós, illetve képzetes részét adja meg a program magjában meglevő `Re`, illetve `Im` függvény. Komplex változós *kifejezés* valós, illetve képzetes részének előállításához az

```
Algebra`ReIm`
```

programcsomag ugyanilyen nevű függvényeit használhatjuk:

```
<<Algebra`ReIm`
Re[1/z]
```

$$\frac{\mathrm{Re}[z]}{\mathrm{Im}[z]^2+\mathrm{Re}[z]^2}$$

Figyeljük meg, hogy ez a `Re` függvény `z`-t komplex változónak tekintette. Kétféleképpen is közölhetjük azt a szándékunkat a *Mathematicá*val, hogy az `x` szimbólumot tartalmazó kifejezéseket annak feltételezésével értékelje ki, hogy az `x` *valós* változó. Erre az `UpSet` belső függvényt (ennek rövid alakja a `^=` jelsorozat) így használhatjuk:

```
Im[x] ^= 0;
```

Ugyanezt az eredményt kapjuk az

```
x /: Im[x] = 0;
```

utasítással is (a `/:` jelsorozat a `TagSet` belső függvény rövid alakja). Ha `x` valós változó, akkor

```
Re[(a + x)^2]
```

$$-\mathrm{Im}[a]^2 + (x + \mathrm{Re}[a])^2$$

Fentebb már szóltunk arról, hogy a `ComplexExpand` függvény használatakor hogyan jelezzük azt, hogy egy adott *kifejezésben* levő szimbólumok közül melyiket gondoljuk valósnak és melyiket komplexnek. A változóknak ez a specifikációja lokális jellegű. Ezzel szemben az `Im[z]^=0` globális értékadás. Az elmondottakat az *Euler-féle formula* példáján illusztráljuk:

```
eu = ComplexExpand[Exp[I t]]
Cos[t] + I Sin[t]

Re[%]
Cos[Re[t]] Cosh[Im[t]] - Cos[Re[t]] Sinh[Im[t]]
```

A `t` szimbólumot a `ComplexExpand` függvény valós változónak, a `Re` függvény pedig komplexnek tekintette:

```
Im[t] ^= 0;
Re[eu]
Cos[t]
```

3.2.4. Trigonometrikus kifejezések

A *Mathematica* előző pontokban ismertetett függvényei alapértelmezésben nem alkalmaznak trigonometrikus azonosságokat, mert a `Trig` opciójuk értéke `False`. (Érdemes végignézni, hogy a szóban forgó függvények valóban rendelkeznek a `Trig` opcióval.) Ha ezeknél a függvényeknél a

```
Trig -> True
```

opciót adjuk meg, akkor a program trigonometrikus azonosságokat is felhasználva alakítja át a beadott kifejezést.

A `Trig -> True` opció hatására a *Mathematica* a következő műveleteket végzi el. Először a trigonometrikus kifejezést komplex exponenciális alakban írja fel, ezután hajtja végre a kijelölt műveleteket. Az így kapott eredményt végül megpróbálja trigonometrikus függvényekkel kifejezni.

A `Simplify` függvény alapértelmezésben is így működik, mert a `Trig` opció értéke `True`:

```
Options[Simplify]
{Trig -> True}
```

A lehetőségeket a következő példák bemutatásával illusztráljuk:

```
{Expand[Sin[x]^2 + Sin[2x]^2], Expand[%, Trig -> True]}
```

$$\{\texttt{Sin[x]}^2 + \texttt{Sin[2 x]}^2,\ 1 - \frac{\texttt{Cos[2 x]}}{2} - \frac{\texttt{Cos[4 x]}}{2}\}$$

```
Expand[Sin[a x] Cos[b x]^2, Trig -> True]
Factor[%, Trig -> True]
```

$$\frac{\texttt{Sin[a x]}}{2} + \frac{\texttt{Sin[a x - 2 b x]}}{4} + \frac{\texttt{Sin[a x + 2 b x]}}{4}$$

$$\texttt{Cos[b x]}^2\ \texttt{Sin[a x]}$$

```
Factor[Sin[x] + Sin[y], Trig -> True]
```

$$2\ \texttt{Cos}[\frac{\texttt{x}}{2} - \frac{\texttt{y}}{2}]\ \texttt{Sin}[\frac{\texttt{x}}{2} + \frac{\texttt{y}}{2}]$$

Felhasználhatjuk még az

```
Algebra`Trigonometry`
```

programcsomagban meglevő alábbi függvényeket is:

`ComplexToTrig`	`TrigReduce`
`TrigFactor`	`TrigToComplex`

Olvassuk be ezt a programcsomagot:

```
<<Algebra`Trigonometry`
```

A `TrigReduce` függvény az addíciós tételekben kimondott azonosságok közvetlen felhasználásával alakítja át a beadott kifejezést:

```
TrigReduce[Sin[x + y]]
Cos[y] Sin[x] + Cos[x] Sin[y]
```

ugyanakkor a fentebb elmondottak miatt

```
Expand[Sin[x + y], Trig -> True]
Sin[x + y]
```

A `TrigFactor` függvény a szögek szinuszának (koszinuszának stb.) összegére vonatkozó ismert azonosságot alkalmazza, ezért ez az eljárás a

```
Factor[kifejezes, Trig -> True]
```

utasítás eredményétől eltérő alakot is adhat:

```
TrigFactor[Sin[x] - Sin[3x]]
-2 Cos[2 x] Sin[x]

Factor[Sin[x] - Sin[3x], Trig -> True]
-2 (Cos[x] - Sin[x]) Sin[x] (Cos[x] + Sin[x])
```

Trigonometrikus kifejezéseket komplex exponenciális alakban írhatunk fel a `TrigToComplex` függvény segítségével. A fordított feladat megoldásához pedig a `ComplexToTrig` eljárást használhatjuk:

```
TrigToComplex[Sin[x] + Cos[x]]
```

$$\frac{-I}{2}\left(-E^{-I\,x} + E^{I\,x}\right) + \frac{E^{-I\,x} + E^{I\,x}}{2}$$

```
TrigToComplex[Sin[x] Cos[x]] // Expand
```

$$\frac{I}{4}E^{-2\,I\,x} - \frac{I}{4}E^{2\,I\,x}$$

```
ComplexToTrig[%]
```

$$\frac{-I\,\mathrm{Cos}[2\,x] + \mathrm{Sin}[2\,x]}{4} + \frac{I\,\mathrm{Cos}[2\,x] + \mathrm{Sin}[2\,x]}{4}$$

```
Simplify[%]
```

$$\frac{\mathrm{Sin}[2x]}{2}$$

3.2.5. Egyéb matematikai kifejezések

Ebben a pontban csupán érzékeltetni szeretnénk azt, hogy a *Mathematica* hogyan kezeli az előző pontokba nem sorolható (például irracionális vagy transzcendens) kifejezéseket.

Először azt a tényt emeljük ki, hogy a program *számokat*, illetve *szimbólumokat* tartalmazó kifejezések átalakításánál különböző típusú transzformációs szabályokat (azonosságokat) alkalmaz.

Ha egy beépített matematikai függvény argumentumában csak *közelítő numerikus érték* szerepel, akkor a program a szóban forgó kifejezést a 3.1.4. pontban jelzett módon kiértékeli:

```
{Arg[-1.2 - 0.7 I], (-1.)^(1/3), Sqrt[5.]}
{-2.61352, 0.5 + 0.866025 I, 2.23607}
```

Ha egy beépített matematikai függvény argumentumába *pontos numerikus értéket* vagy beépített matematikai állandót írunk, akkor néhány esetben a helyettesítési érték egy másik (matematikai szempontból a megadottal ekvivalens) alakját kapjuk meg:

```
{Sqrt[16], Log[E^2], ArcSin[1/Sqrt[2]]}
{4, 2, Pi/4}
```

Itt is felhívjuk az Olvasó figyelmét a 3.1.4. pontban az inverz függvényekkel kapcsolatban elmondottak egy lényeges következményére. Alapértelmezésben a *Mathematica* számára például

$$\sqrt[3]{-1} = \frac{1}{2} + \frac{\sqrt{3}}{2}i$$

és nem (-1), mivel

```
ComplexExpand[(-1)^(1/3)]
1/2 + I/2 Sqrt[3]
```

Pontos numerikus értékeket tartalmazó matematikai kifejezések átalakításánál a program bizonyos azonosságokat automatikusan alkalmaz:

```
Sqrt[25/2] Sqrt[2]
5

Expand[(1 + Sqrt[3])^17]
13160704 + 7598336 Sqrt[3]
```

```
Log[E^3]
3
```

A következő kifejezést azonban nem alakítja át:

```
Log[4] - Log[3]
-Log[3] + Log[4]
```

A program szimbólumokat tartalmazó matematikai kifejezések kiértékelésénél automatikusan csak olyan azonosságokat használ fel, amelyek a szimbólumok minden komplex értéke esetén érvényesek. Ezt illusztrálják az alábbi példák:

```
{(a b)^3, E^x E^y}
  3  3   x + y
{a  b , E     }
```

```
{Sqrt[x y],  Sqrt[x] Sqrt[y]}
{Sqrt[x y],  Sqrt[x] Sqrt[y]}
```

```
{Log[a b],  Log[a] + Log[b],  Log[a^b]}
                                  b
{Log[a b],  Log[a] + Log[b], Log[a ]}
```

Az utóbbi két eredmény magyarázata az, hogy a

$$\sqrt{xy} = \sqrt{x}\sqrt{y},$$
$$\log(ab) = \log a + \log b$$

egyenlőségek komplex számok esetén általában nem érvényesek. Valóban:

```
{Sqrt[(-1) (-1)],  Sqrt[-1] Sqrt[-1]}
{1,  -1}
```

```
{Log[(-1) (-1)],  Log[-1] Log[-1]}
                          2
{0, -Pi^2}
```

Matematikai kifejezések átalakításához használhatjuk a

PowerExpand

belső függvényt, amely az $(ab)^c$ hatványt az $a^c b^c$ szorzattal helyettesíti:

```
PowerExpand[(a b)^c]
  c  c
a  b
```

```
PowerExpand[Sqrt[x y]]
Sqrt[x] Sqrt[y]
```

Vigyáznunk kell azonban arra, hogy csak olyan a, b, c paraméterek esetében alkalmazzuk ezt az eljárást, amikor az

$$(ab)^c = a^c b^c$$

egyenlőség érvényes.

Előfordulhat, hogy az eddig ismertetett függvényekkel egy adott kifejezést nem az igényeinknek megfelelő alakban kapjuk meg. Ilyenkor például transzformációs szabályokat (lásd a 2.3.2. pontot) definiálhatunk, és ezeket a

ReplaceAll

(ennek rövid alakja: `/.`), vagy pedig a

ReplaceRepeated

(ennek rövid alakja: `//.`) eljárással alkalmazhatjuk a szóban forgó kifejezésre:

```
Log[a b c d] /. Log[x_ y_] -> Log[x] + Log[y]
Log[a] + Log[b c d]

Log[a b c d] //. Log[x_ y_] -> Log[x] + Log[y]
Log[a] + Log[b] + Log[c] + Log[d]

sin2x = Sin[2 x_] -> 2 Sin[x] Cos[x];
Sin[2 (1 + t)^2] /. sin2x
                    2                   2
2 Cos[(1 + t)  ] Sin[(1 + t)  ]
```

3.2.6. Nevezetes összegek és szorzatok

Adott számú tagot (tényezőt) tartalmazó összeget (szorzatot) a

Sum (Product)

belső függvénnyel számolhatunk ki:

```
Sum[k^2, {k, 1, 1996}]
2652690986

Product[x + k, {k, 1, 4}
(1 + x) (2 + x) (3 + x) (4 + x)
```

Igen sok esetben azonban olyan összegekre (szorzatokra) is kaphatunk zárt (pontos) formulát, amelyekben a tagok (tényezők) számát szimbólummal adjuk meg. Ehhez nyújt segítséget az

```
Algebra`SymbolicSum`
```

programcsomag `Sum` (`Product`) függvénye.

Olvassuk be ezt a programcsomagot:

```
<<Algebra`SymbolicSum`
```

és először néhány jól ismert összeget határozzunk meg:

```
Sum[k, {k, 1, n}]
n (1 + n)
-----------
    2
Sum[k^2, {k, 1, n}]
n (1 + n) (1 + 2 n)
-------------------
         6
Sum[k^3, {k, 1, n}]
 2       2
n  (1 + n)
----------
    4
```

A következő példával a további lehetőségeket illusztráljuk:

```
Sum[(-3)^k Binomial[2 n, 2 k], {k, 0, n}]
    n      2 n Pi
4^n Cos[-------]
       3
```

A *Mathematica* tehát azt állítja, hogy

$$\sum_{k=0}^{n}(-3)^k\binom{2n}{2k} = 4^n\cos\left(\frac{2n\pi}{3}\right).$$

(Hogyan lehet ezt bebizonyítani?)

Ezzel a `Sum` függvénnyel számos végtelen sornak a pontos összegét is meghatározhatjuk. Például:

```
Sum[1/(5 + 14 k + 8 k^2), {k, 0, Infinity}]
```

$$\frac{2}{3} - \frac{\text{Pi}}{12} - \frac{\text{Log}[2]}{6}$$

```
Sum[1/k^2, {k, 1, Infinity}]
```

$$\frac{\text{Pi}^2}{6}$$

```
{Sum[1/k^17, {k, 1, Infinity}], N[%, 20]}
{Zeta[17], 1.6449340668482264365}
```

Trigonometrikus összegek kiszámolásához szükség esetén be kell hívni az

```
Algebra`Trigonometry`
```

programcsomagot. Például a

$$\sum_{k=1}^{n} \sin(kx)$$

összeget (a *Dirichlet-magot*) így határozhatjuk meg a *Mathematicá*val:

```
TrigReduce[
    Sum[Sin[k x], {k, 1, n}]] // Simplify
```

$$\text{Csc}[\frac{x}{2}]\ \text{Sin}[\frac{n\ x}{2}]\ \text{Sin}[\frac{(1 + n)\ x}{2}]$$

A `Sum` függvényhez hasonló lehetőségekkel rendelkezik a `Product` eljárás is. Például:

```
Product[1 - 1/(k + 5)^2, {k, 0, n}]
```

$$\frac{4\ (6 + n)}{5\ (5 + n)}$$

Ennek a függvénynek a segítségével végtelen szorzatokat is meghatározhatunk. A *Wallis-formulát* például így kaphatjuk meg:

```
Product[4 k (1 + k)/(1 + 2 k)^2, {k, 1, Infinity}]
```

$$\frac{\text{Pi}}{4}$$

A `Sum` és a `Product` külső függvénybe beépített algoritmusok (lásd [2]) elméleti alapját Gauss, Saalschutz, Kummer és Dixon klasszikus tételei képezik. Ezek az eredmények azt állítják, hogy a $\sum a_k$ (véges vagy végtelen) összeg kifejezhető a hipergeometrikus függvényekkel azokban az esetekben, amikor az a_{k+1}/a_k hányados a k index racionális törtfüggvénye.

3.3. Egyenletek megoldása

Lineáris egyenletrendszerek megoldásával a 3.8., differenciálegyenletekkel pedig a 3.5. szakaszban foglalkozunk. Ebben a szakaszban a programnak azokat a függvényeit ismertetjük, amelyeket egyéb matematikai függvényeket tartalmazó egyenletek és egyenletrendszerek megoldásánál használhatunk.

Egyenletek *pontos megoldását* a

`Reduce`	`Solve`
`Roots`	`SolveAlways`

belső függvényekkel állíthatjuk elő abban az esetben, ha a bemenő adatok a szimbolikus változókon kívül csupán pontos numerikus értékeket vagy beépített matematikai állandókat tartalmaznak.

Ha a bemenő adatok valamelyike explicit tizedesponttal megadott szám, akkor az imént felsorolt függvények numerikus módszerek alkalmazásával adják meg a pontos megoldás egy *közelítő értékét*. Szintén numerikus módszert alkalmazva határoznak meg közelítő megoldásokat az alábbi függvények:

`FindRoots`	`NSolve`
`NRoots`	

A *Mathematica* egyenletmegoldó algoritmusai a változókat és a paramétereket komplex számoknak tekintik, és alapértelmezésben a megadott egyenlet komplex megoldásait keresik.

3.3.1. Az utasítások szintaxisa

A *Mathematica* logikai állításnak tekinti az egyenleteket, ezért ezeket a matematikában megszokott `=` helyett a `==` szimbólummal jelöljük. Az

$$x^2 + x = 2$$

egyenletet a *Mathematicá*ban tehát így adjuk meg:

```
x^2 + x == 2
```

Adott `x` komplex szám esetén a fenti utasítás eredménye `True` vagy `False` attól függően, hogy `x` megoldása-e a szóban forgó egyenletnek:

```
x^2 + x == 2 /. {{x -> -2}, {x -> 4}}
```

```
{True, False}
```

• Egyismeretlenes egyenletek

Egy ismeretlent tartalmazó egyenlet *komplex* megoldásait a

```
Solve[egyenlet, ismeretlen]
```

alakban megadott utasítással kaphatjuk meg. (A második argumentum kiírása nem kötelező.) Például:

```
egyenlet1 = (1/2)x^2 - 2x + 1/2 == 0;
Solve[egyenlet1, x]
```

$$\{\{x \to \frac{4 - 2\sqrt{3}}{2}\}, \{x \to \frac{4 + 2\sqrt{3}}{2}\}\}$$

A fenti utasítássorozat első sora azt jelenti, hogy az egyenletünknek az `egyenlet1` nevet adtuk. A továbbiakban az egyenletre ezzel a változóval hivatkozhatunk. Az elnevezés természetesen nem kötelező, de sok esetben hasznos lehet.

A kapott eredményt így egyszerűsíthetjük:

```
megoldas = Simplify[%]
```

```
{{x -> 2 - Sqrt[3]}, {x -> 2 + Sqrt[3]}}
```

és visszahelyettesítéssel így ellenőrizhetjük:

```
egyenlet1 /. %;
Simplify[%]
```

```
{True, True}
```

Figyeljük meg, hogy a `Solve` függvény listák listájával és transzformációs szabályokkal adja meg a megoldásokat, amelyekre a további műveletekben kétféle módon hivatkozhatunk. Az első lehetőség az, hogy ugyanazt a változónevet használjuk egy kifejezésben, mint az egyenletben, és a megoldást transzformációs szabályként alkalmazzuk:

```
5x^4 + 3x /. megoldas
{3 (2 - Sqrt[3]) + 5 (2 - Sqrt[3])^4,
    3 (2 + Sqrt[3]) + 5 (2 + Sqrt[3])^4}

Expand[%]
{491 - 283 Sqrt[3], 491 + 283 Sqrt[3]}
```

(Mivel több gyöke van az egyenletnek, ezért az eredmény egy lista.) A másik lehetőség pedig az, hogy az eredményt kifejezésnek tekintve kiválasztjuk annak megfelelő részét a szokásos módon: az adott részhez vezető indexek egy sorozatával (lásd a 2.3.1. pontot):

```
x1 = megoldas[[1, 1, 2]]
2 - Sqrt[3]

x2 = megoldas[[2, 1, 2]]
2 + Sqrt[3]
```

A fönti példában a változón és a műveleti jeleken kívül csak pontos numerikus értékek szerepelnek. Az ilyen esetekben a `Solve` függvénybe beépített algoritmus az egyenlet pontos megoldásait keresi.

Ha az együtthatók valamelyikét közelítő numerikus értékkel (tehát a tizedespont explicit kiírásával) adjuk meg, akkor a `Solve` függvény numerikus módszer felhasználásával határozza meg a pontos megoldás egy közelítő értékét:

```
egyenlet2 = (1/2)x^2 - 2.x + 1/2 == 0;
Solve[egyenlet2, x]
{{x -> 0.267949}, {x -> 3.73205}}
```

Ugyanezt az eredményt kapjuk akkor is, ha az `N` belső függvényt alkalmazzuk a pontos megoldásra:

```
N[megoldas]
{{x -> 0.267949}, {x -> 3.73205}}
```

Többszörös gyököt a *Mathematica* annyiszor sorol fel, amennyi annak a multiplicitása:

```
Solve[x^2 - 2x + 1 == 0, x]
{{x -> 1}, {x -> 1}}
```

A program üres listával (transzformációs szabályok üres halmazával) jelzi azt a tényt, hogy a beadott egyenletnek nincs komplex megoldása:

```
Solve[1/(x (x-1)) + 1/(x (x + 1)) == 0, x]
{}

Solve[Sqrt[x-8] + Sqrt[x] == 2, x]
{}
```

Ha az egyenletnek végtelen sok megoldása van, akkor üres listát tartalmazó listát kapunk eredményül (hiszen ilyenkor megszorítást nem tartalmazó transzformációs szabály az eredmény):

```
Solve[(x + 1)^2 == x^2 + 2x + 1, x]
{{}}

Solve[1/(x-1) - 4/(2-2x) == 3/(x-1), x]
{{}}
```

A `Roots` belső függvény logikai állítás formájában adja meg egy egyenlet megoldását:

```
Roots[x^2 + 3x == 2, x]
```

$$x == \frac{-3 - \text{Sqrt}[17]}{2} \;||\; x == \frac{-3 + \text{Sqrt}[17]}{2}$$

Ebből a `ToRules` függvénnyel transzformációs szabályt leíró listákat kapunk:

```
ToRules[%]
```

$$\text{Sequence}\left[\{x \to \frac{-3 - \text{Sqrt}[17]}{2}\}, \{x \to \frac{-3 - \text{Sqrt}[17]}{2}\}\right]$$

Ha a `Solve` függvény nem találja meg a kért egyenlet pontos megoldását, akkor ezt így jelzi:

```
Solve[x^5 + 5x + 1 == 0, x]
                          5
{ToRules[Roots[5 x + x  == -1, x]]}
```

• Feltételek megadása

A `Solve` függvény első argumentumába beírhatunk egyenleteket tartalmazó logikai kifejezéseket is:

```
Solve[x^3 == x && x != 0, x]
{{x -> -1}, {x -> 1}}
```

```
Solve[x^3 == x && x != 1 || x^2 == 2, x]
{{x -> -1}, {x -> 0}, {x -> -Sqrt[2]}, {x -> Sqrt[2]}}
```

Egész együtthatós polinomegyenlet gyökeit a $\mathbf{Z}_p$ véges számtestben a

```
Modulus == p
```

feltétel megadása esetén kaphatjuk meg:

```
Solve[x^5 + 16x^4 + 7x^3 + 17x^2 + 11x + 5 == 0 &&
      Modulus == 19, x]
{{Modulus -> 19, x -> -18}, {Modulus -> 19, x -> -16},
 {Modulus -> 19, x -> -12},
 {Modulus -> 19, x -> -7}, {Modulus -> 19, x -> -1}}
```

Ha modulust nem adunk meg, akkor a `Solve` függvény harmadik argumentumában a `Mode->Modular` opciót használva a *Mathematica* olyan modulust is keres, amelyre vonatkozóan a megadott egyenleteknek van közös megoldása:

```
Solve[{x^2 + 1 == 0, x^3 + 1 == 0}, x, Mode -> Modular]
{{Modulus -> 2, x -> -1}}
```

• Paramétert tartalmazó egyenletek

Az egyenlet ismeretlenétől különböző szimbólumokat a program komplex paraméternek tekinti. A `Solve` függvény a paraméterek vizsgálatával nem foglalkozik:

```
Solve[a x + b == 0, x]
```

$$\{\{x \to -\left(\frac{b}{a}\right)\}\}$$

Számos, paraméter(eke)t is tartalmazó egyenlet matematikai szempontból korrekt megoldásait adja meg a

```
Reduce[egyenlet, ismeretlen]
```

utasítás, azaz a paraméter(ek) összes lehetséges értékét figyelembe véve („diszkusszióval együtt") kapjuk meg az egyenlet megoldását:

```
Reduce[a x + b == 0, x]
```

$$b == 0 \;\&\&\; a == 0 \;||\; a \neq 0 \;\&\&\; x == -\left(\frac{b}{a}\right)$$

Hasonlítsuk össze például a következő két utasítás eredményét:

```
Solve[a x^2 + b x + c == 0, x]
Reduce[a x^2 + b x + c == 0, x]
```

• Egyenletrendszerek

A `Solve`, a `Roots` és a `Reduce` belső függvénnyel számos *egyenletrendszer* komplex megoldásait is elő tudjuk állítani. A *Mathematica* egyik legalapvetőbb adatszerkezetének, a listának a felhasználásával ezt a szándékunkat így közölhetjük a programmal:

```
Solve[{x^2 + y^2 + z^2 == 94, x + y + z == 14,
      x y + x z == 45}, {x, y, z}]
{{y -> 2, z -> 3, x -> 9}, {y -> 3, z -> 2, x -> 9},
 {y -> (9 - Sqrt[57])/2, z -> (9 + Sqrt[57])/2, x -> 5},
 {y -> (9 + Sqrt[57])/2, z -> (9 - Sqrt[57])/2, x -> 5}}
```

Az egyenleteket logikai jelekkel is összekapcsolhatjuk:

```
Solve[x + y == 1 && x - y == 2, {x, y}]
{{x -> 3/2, y -> -1/2}}
Solve[x + y == 1 || x - y == 2, {x, y}]
Solve::svars:
   Warning: Equations may not give solutions for all
     "solve" variables.
{{x -> 1 - y}, {x -> 2 + y}}
```

3.3.2. Egyenletek pontos megoldása

Különböző típusú egyenletek pontos megoldásainak explicit előállítását számos matematikai eredmény korlátozza. Ebben a pontban ezek felidézése mellett ismertetjük a *Mathematica* program által nyújtott lehetőségeket.

Egyenletek matematikai leírásának egy lehetséges módja a következő. Legyen A és B nem üres halmaz, és tekintsük az $f : A \to B$ függvényt. Adott $b \in B$ esetén határozzuk meg a

$$M := \{x \in A : f(x) = b\} \tag{1}$$

halmazt. Erre a feladatra gyakran úgy hivatkozunk, hogy „oldjuk meg az

$$f(x) = b \qquad ?(x \in A) \tag{2}$$

egyenletet".

Ha $x \in M$, akkor x-et a (2) egyenlet *megoldásá*nak (vagy *gyök*ének), az M halmazt pedig a (2) *megoldáshalmazának* nevezzük. A (2) egyenletnek nincs megoldása, vagy az egyenlet nem oldható meg, ha M az üres halmaz. Egyértelmű a megoldás, ha az M halmaz egyelemű.

Ilyen általános feltételek mellett a megoldáshalmazról vajmi kevés mondható. Számos (több esetben mély) matematikai eredmény ismeretes különböző speciális módon megválasztott A, B halmaz és f függvény esetében.

• Polinomegyenletek

A *Mathematica* jelenlegi változatai polinomokat tartalmazó egyenletek és egyenletrendszerek megoldásainak előállításához nyújtják a legtöbb segítséget.

Tekintsük először az *egy ismeretlent* tartalmazó egyenleteket, azaz legyen $A := B := \mathbf{C}$ és

$$p(x) := \sum_{k=0}^{n} a_k x^k \qquad (x \in \mathbf{C},\ a_k \in \mathbf{C},\ n \in \mathbf{N})$$

adott algebrai polinomfüggvény.

Az „algebra alaptétele" szerint a

$$p(x) = 0 \qquad ?(x \in \mathbf{C}) \tag{3}$$

egyenletnek pontosan annyi gyöke van, amennyi a p polinom fokszáma, ha minden gyökét annyiszor számítjuk, amennyi a multiplicitása.

Ismeretes, hogy tetszőleges, legfeljebb negyedfokú polinom komplex gyökeinek explicit előállítására van (megoldó)képlet, azaz a (3) egyenlet megoldásait fel lehet írni a p polinom együtthatóival az alapműveletek és a gyökvonás véges sokszori alkalmazásával.

Érdemes kipróbálni, hogy ezeket a megoldóképleteket a *Mathematica* is ismeri, ezért segítségével bármely, legfeljebb negyedfokú polinomegyenlet komplex megoldásait elő tudjuk állítani.

Tekintsük például a következő feladatot. Jelölje x_1, x_2, x_3 az $x^3+px+q = 0$ egyenlet gyökeit. Bizonyítsuk be, hogy

$$x_1^5 + x_2^5 + x_3^5 = 5pq.$$

Egy lehetséges megoldás:

```
megoldas = Solve[x^3 + p x + q == 0, x];
x1 = megoldas[[1, 1, 2]];
x2 = megoldas[[2, 1, 2]];
x3 = megoldas[[3, 1, 2]];
x1^5 + x2^5 + x3^5;
Simplify[%];
Together[%]
```

```
5 p q
```

Mutatunk egy másik megoldást is:

```
Apply[Plus, (x^5 /. #)& /@ megoldas];
Simplify[%];
Together[%]
```

```
5 p q
```

A Galois-elmélet egyik mély eredménye azt állítja, hogy négynél magasabb fokszámú polinom gyökeinek explicit előállítására nincs általános megoldóképlet. Ez az eredmény nem zárja ki annak lehetőségét, hogy minden n-edfokú $n \geq 5$ egyenletnek legyen (esetleg egyenletről egyenletre változó) gyökképlete. Bebizonyítható azonban az is, hogy tetszőleges $n \geq 5$ természetes számhoz létezik olyan n-edfokú, egész együtthatójú polinom, amelyhez a fenti értelemben nincs megoldóképlet. Ilyen például az $x^5 - 4x + 2 = 0$ egyenlet is.

A fenti negatív jellegű eredmények ellenére a `Solve` és a `Reduce` függvény számos, négynél magasabb fokszámú polinomegyenlet pontos megoldását képes előállítani. Ilyenkor a program a matematikában megszokott módszert követi. Egyrészt az adott polinomot a `Factor` belső függvénybe beépített algoritmusok (lásd a 3.2.1. pontot) felhasználásával megpróbálja alacsonyabb fokszámú tényezők szorzatára felbontani. Másrészt alacsonyabb fokszámú polinomok kompozíciójaként igyekszik előállítani a megadott polinomot, azaz új változó bevezetésével csökkenti annak fokszámát. A `Solve` és a `Reduce` függvény ehhez a `Decompose` belső függvénybe beépített algoritmusokat (lásd a 3.1.3. pontot) hívja meg.

Próbáljuk ki például a következő utasításokat:

```
Solve[x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120 == 0, x]
Solve[x^6 + (2-a)*x^5 + (2-a)*x^4 + (2-a)*x^3
      (2-a)*x^2 + (2-a)*x + 1 - a == 0, x]
```

Az $x^5 - 4x + 2 = 0$ egyenlet megoldásaira nincs megoldóképlet. Így nem meglepő, hogy a *Mathematica* sem állítja elő a gyököket:

```
Solve[x^5 - 4x + 2 == 0, x]

{ToRules[Roots[x^5 - 4 x + 2 == 0, x]]}
```

Néhány esetben bizonyos megoldásokat megkapunk, másokat pedig nem. Például:

```
p = x^10-x^9-3x^8+3x^7+2x^4-x^3-7x^2+3x+3;
Solve[p == 0, x]

{{x -> 1}, {x -> -Sqrt[3]}, {x -> Sqrt[3]},
  ToRules[Roots[2 x + x^7 == -1, x]]}
```

Vannak azonban olyan polinomegyenletek is, amelyeknek gyökjelekkel kifejezhető megoldásait a *Mathematica* nem találja meg. A [89] referenciakönyv 609. oldalán is találhatunk egy ilyen példát:

```
egyenlet = x^6 - 9x^4 - 4x^3 + 27x^2 - 36x - 23 == 0;
Solve[egyenlet, x]

{ToRules[Roots[-36 x + 27 x^2 - 4 x^3 - 9x^4 + x^6 == 23, x]]}
```

Ennek az egyenletnek azonban $\sqrt[3]{2}+\sqrt{3}$ egyik gyöke. Valóban:

```
egyenlet /. x -> (2^(1/3) + Sqrt[3]);
Simplify[%]

True
```

Az `Algebra`CountRoots`` programcsomagban lévő

`CountRoots`

függvénnyel valós együtthatós polinom adott intervallumba eső valós gyökeinek a számát határozhatjuk meg. Ez az eljárás a Sturm-féle módszert alkalmazza:

```
CountRoots[x^2 - 1, {x, 0, 2}]

1

CountRoots[x^4 - x^3, {x, -1, 2}]

2

CountRoots[x^5 + 2x^4 + x^3 - 4x^2 - 3x -5,
           {x, 0, Infinity}]

1
```

• Egyéb egyenletek

Nincs általánosan használható *módszer* nem polinomokat tartalmazó egyenletek pontos megoldásainak az előállítására. Ennek ellenére a *Mathematica* `Solve` és `Reduce` függvénye néhány ilyen egyenlet szimbolikus megoldására is képes.

Emlékeztetünk arra, hogy az egyenletmegoldó algoritmusok általában a *komplex* megoldásokat próbálják előállítani. Felhívjuk a figyelmet arra is, hogy a program által adott eredményt célszerű ellenőrizni.

A matematikában megszokott módon járhatunk el akkor, amikor a fentebb megemlített eljárások nem tudják meghatározni a beadott egyenlet gyökeit. Nevezetesen: a program egyéb függvényeinek felhasználásával az egyenletet sokszor olyan alakra hozhatjuk, amelyről a megoldás már egyszerűen leolvasható.

Néhány példa bemutatásával illusztráljuk a *Mathematica* által nyújtott lehetőségeket. Vegyünk először egy *négyzetgyököt* tartalmazó egyenletet:

```
Solve[Sqrt[x + 1] + Sqrt[x] == 2, x]
```

$$\{\{x \to \frac{9}{16}\}\}$$

Az egyenletnek valóban 9/16 az egyetlen komplex megoldása. A következő számolások felhasználásával lehet ezt bebizonyítani:

```
Sqrt[x + 1]^2 - Expand[(2 - Sqrt[x])^2]
-3 + 4 Sqrt[x]

Solve[% == 0, x]
```

$$\{\{x \to \frac{9}{16}\}\}$$

```
Sqrt[9/16 + 1] + Sqrt[9/16]
2
```

A `Solve` függvény üres listát ad eredményül, ha az egyenletnek nincs (komplex) megoldása. Például:

```
Solve[Sqrt[x - 8] + Sqrt[x] == 2, x]
{}
```

Ezt az eredményt például így ellenőrizhetjük:

```
Sqrt[x - 8]^2 - Expand[(2 - Sqrt[x])^2]
-12 + 4 Sqrt[x]
```

```
Solve[% == 0, x]
{{x -> 9}}

Sqrt[9 - 8] + Sqrt[9]
4
```

Oldjuk most meg a következő *trigonometrikus* egyenletet:

$$\sin^4 x + \sin^4 2x + \sin^4 3x = \cos^4 x + \cos^4 2x + \cos^4 3x \qquad ?(x \in \mathbf{R}).$$

Ebben az esetben a `Solve` függvény nem ad segítséget. Rendezzük a bal oldalra az egyenletet és az így kapott kifejezést próbáljuk meg szorzattá alakítani:

```
bo = Sin[x]^4 + Sin[2x]^4 + Sin[3x]^4 -
     Cos[x]^4 - Cos[2x]^4 - Cos[3x]^4;
Factor[bo, Trig -> True]
-((1 + 2 Cos[2 x]) (Cos[2 x] - Sin[2 x])
    (Cos[2 x] + Sin[2 x]))
```

Az eredeti egyenlet M megoldáshalmaza tehát egyenlő a fenti tényezők valós zérushelyeit tartalmazó halmazok (jelöljük ezeket rendre az M_1, az M_2, illetve az M_3 szimbólummal) egyesítésével. Nézzük az első tényező zérushelyeit:

```
Solve[1 + 2 Cos[2x] == 0, x]
Solve::ifun:
   Warning: Inverse functions are being used by Solve,
     so some solutions may not be found.
{{x -> Pi/3}}
```

Figyeljük meg, hogy a `Solve` eljárás a végtelen sok megoldás közül csak egyet ad meg. A cos függvény periodicitását figyelembe véve az első tényező valós zérushelyeinek a halmaza: $M_1 = \{(3k \pm 1)\pi/3 \ : \ k \in \mathbf{Z}\}$. A második, illetve a harmadik tényező zérushelyeinek a halmaza a

```
Solve[Tan[2x] == 1, x]
Solve::ifun:
   Warning: Inverse functions are being used by Solve,
     so some solutions may not be found.
{{x -> Pi/8}}
```

```
Solve[Tan[2x] == -1, x]
Solve::ifun:
   Warning: Inverse functions are being used by Solve,
     so some solutions may not be found.
{{x -> -Pi/8}}
```

alapján

$$M_2 = \{\frac{\pi}{8} + k\frac{\pi}{2} \ : \ k \in \mathbf{Z}\}, \quad \text{illetve} \qquad M_3 = \{-\frac{\pi}{8} + k\frac{\pi}{2} \ : \ k \in \mathbf{Z}\}.$$

Az egyenlet megoldáshalmaza tehát:

$$M = M_1 \cup M_2 \cup M_3 = \{(2k+1)\frac{\pi}{8} \ : \ k \in \mathbf{Z}\} \cup \{(3k \pm 1)\frac{\pi}{3} \ : \ k \in \mathbf{Z}\}.$$

Nézzünk egy példát *logaritmust* tartalmazó egyenletre. Meghatározzuk a

$$\frac{\log_x(2t-x)}{\log_x 2} + \frac{\log_t x}{\log_t 2} = \frac{1}{\log_{t^2-1} 2} \qquad ?(x \in \mathbf{R})$$

egyenlet gyökeit, ahol t valós paraméter.

A valós logaritmusfüggvény tulajdonságaiból következik, hogy az egyenletnek csak a $t \in (1, +\infty) \setminus \{\sqrt{2}\}$ paraméterérték esetén lehet valós megoldása. Az x megoldásnak pedig ki kell elégítenie az $x > 0$, az $x \neq 1$ és a $2t - x > 0$ feltételeket.

A `Solve` és a `Reduce` függvény nem tudja megoldani ezt az egyenletet, ezért először átalakításokat végzünk:

```
kif1 = Log[x, 2t-x]/Log[x, 2]+Log[t, x]/Log[t, 2]-
       1/Log[t^2-1, 2]
-(Log[-1 + t^2]/Log[2]) + Log[2 t - x]/Log[2] + Log[x]/Log[2]
kif2 = Together[kif1]
-Log[-1 + t^2] + Log[2 t - x] + Log[x]
---------------------------------------
               Log[2]
```

A *Mathematica* logaritmusok összegét, illetve különbségét nem alakítja át automatikusan szorzat, illetve hányados logaritmusára, mert a

$$\log a + \log b = \log ab$$

egyenlőség a komplex számtestben nem érvényes (lásd a 3.2.5. pontot). A tett feltételek mellett azonban fennáll a fenti egyenlőség. Ha ezt alkalmazni szeretnénk, akkor egy transzformációs szabályt kell definiálnunk:

```
hely = {a_. Log[b_] + c_. Log[b_] -> Log[b^a d^c]}
{Log[b_] (a_.) + Log[d_] (c_.) -> Log[b^a d^c]}
```

A `Solve` eljárással így kaphatjuk meg a megoldásokat:

```
Solve[kif2 == 0, x] //. hely;
Simplify[%]
Solve::tdep:
   The equations appear to involve transcendental
     functions of the variables in an essentially
     non-algebraic way.
{{x -> -1 + t}, {x -> 1 + t}}
```

Az egyenlet megoldása tehát $x = t \pm 1$. Az $x > 0$ és a $2t - x > 0$ feltétel minden $t \in (1, +\infty) \setminus \{\sqrt{2}\}$ esetén teljesül, az $x \neq 1$ feltétel pedig csak akkor, ha $t \neq 2$. A $t = 2$ esetben az egyenletnek egyetlen megoldása: $x = 3$.

A *Mathematicá*t meg lehet tanítani bizonyos *függvényegyenletek* megoldására is. Az érdeklődő Olvasó figyelmét felhívjuk az ezzel a témával kapcsolatos [15] dolgozatra.

• Egyenletrendszerek

A `Solve` függvény a Gröbner-bázisok elméletét alkalmazza polinomokat tartalmazó egyenletrendszerek megoldásánál. A 3.2.1. pont Gröbner-bázisok című alpontjában bevezetett jelöléseket és fogalmakat ebben az alpontban további hivatkozás nélkül fogjuk használni.

Legyen n rögzített természetes szám, és keressük a $P := \{p_1, \cdots, p_k\} \subset \mathbf{Q}[x]$ polinomokkal képzett

$$p_i(x_1, x_2, \cdots, x_n) = 0 \qquad (1 \leq i \leq k) \tag{1}$$

egyenletrendszer komplex megoldásait.

Rögzítsünk egy alkalmas $\leq$ rendezési relációt a 3.2.1. pontban definiált T halmazon, és legyen $G := \{g_1, \cdots, g_l\} \subset \mathbf{Q}[x]$ a (P) ideál ($\leq$-re vonatkozó) Gröbner-bázisa. Mivel a G által generált ideál megegyezik a P halmaz által generált ideállal, ezért az (1) egyenletrendszer megoldáshalmaza egyenlő a

$$g_i(x_1, x_2, \cdots, x_n) = 0 \qquad (1 \leq i \leq l) \tag{2}$$

egyenletrendszer megoldáshalmazával.

Ha az (1) egyenletrendszer lineáris, akkor a Gauss-féle eliminációs eljárással kaphatunk egy vele ekvivalens, „háromszög alakú” (és ezért egyszerűen megoldható) egyenletrendszert.

B. Buchberger [14] ezt a módszert általánosította arra az esetre, amikor a p_i $(i = 1, 2, \cdots, k)$ függvények polinomok. Jól használható algoritmust is adott az (1) rendszerrel ekvivalens, „háromszög alakú" (2) egyenletrendszer megkonstruálására. Bebizonyította azt is, hogy minden $P \subset \mathbf{Q}[x]$ véges halmazhoz létezik ilyen G polinomrendszer (ezt nevezte el Gröbner-bázisnak), amelynek segítségével (a lineáris esethez hasonlóan) megállapítható a megoldás létezése és egyértelműsége, valamint előállítható a megoldás is.

Az elmondottakat a következő példák bemutatásával illusztráljuk.

Oldjuk meg először a $\mathbf{C}^2$ halmazon a következő egyenletrendszert:

$$\begin{aligned} xy - x - y &= 22, \\ x^2 + y^2 + 3(x + y) &= 88. \end{aligned} \tag{3}$$

Számítsuk ki először a (3) egyenletrendszernek megfelelő (P) ideál G Gröbner-bázisát:

```
GroebnerBasis[{x y - x - y - 22,
        x^2 + y^2 + 3 (x + y) - 88}, {x, y}]
{330 + 286 y - 89 y^2 + y^3 + y^4,
    176 + 23 x - 87 y + 2 y^2 + y^3}
```

Mivel $(P) = (G)$, ezért a

$$\begin{aligned} 23x - y^3 + 2y^2 - 87y &= -176, \\ y^4 + y^3 - 89y^2 + 286y &= -330 \end{aligned}$$

egyenletrendszer ekvivalens a (3) egyenletrendszerrel, és ez utóbbi már közvetlenül megoldható.

Ezzel a módszerrel oldja meg a `Solve` függvény a (3) egyenletrendszert:

```
Solve[{x y - x - y - 22 == 0,
       x^2 + y^2 + 3 (x + y) - 88 ==0}, {x, y}];
Simplify[%]
{{x -> (11 - I Sqrt[11])/2, y -> (11 + I Sqrt[11])/2},
  {x -> (11 + I Sqrt[11])/2, y -> (11 - I Sqrt[11])/2},
  {x -> -6 - Sqrt[26], y -> -6 + Sqrt[26]},
  {x -> -6 + Sqrt[26], y -> -6 - Sqrt[26]}}
```

Bebizonyítható az, hogy ha az (1) egyenletrendszernek *véges sok* megoldása van, akkor a megfelelő Gröbner-bázissal kapott (2) egyenletrendszer

„háromszög alakú". Ilyen esetekben a *Mathematica* akkor tudja meghatározni a kért egyenletrendszer pontos gyökeit, ha meg tudja oldani a megfelelő (most már egyváltozós) polinomegyenleteket.

A Gröbner-bázisok segítségével szükséges és elégséges feltétel adható az (1) egyenletrendszer megoldásainak létezésére. Az erre vonatkozó állítás (lásd például [28], Theorem 10.11) a következő: *az* (1) *egyenletrendszer akkor és csak akkor oldható meg a* $\mathbf{C}^n$ *halmazon, ha a* (P) *ideál* G *Gröbner-bázisa nem tartalmazza az* 1 *polinomot.* Ilyen esetekben a `Solve` függvény üres listát ad eredményül:

```
Solve[{x^2 y + 4 y^2 - 17 == 0,
       2 x y - 3 y^2 + 8 == 0,
       x y^2 - 5 x y + 1 == 0}, {x, y, z}]
{}
```

Számoljuk most ki a megfelelő Gröbner-bázist is:

```
GroebnerBasis[{x^2 y + 4 y^2 - 17,
      2 x y - 3 y^2 + 8, x y^2 - 5 x y + 1}, {x, y, z}]
{1}
```

A megoldások egyértelműségére vonatkozóan a következő állítás bizonyítható be (lásd például [28], Theorem 10.12). *Jelöljük a* H *szimbólummal az* 1 *rendszerhez tartozó* (P) *ideál* G *Gröbner-bázisában szereplő főtagok halmazát. Ekkor az* (1) *egyenletrendszernek pontosan akkor van véges sok megoldása, ha minden* $1 \leq i \leq n$ *indexhez létezik olyan* m *természetes szám, hogy* $(x_i)^m \in H$.

Ezt felhasználva bizonyíthajuk be például azt, hogy az

$$\begin{aligned} 2xy + yz &= 27, \\ 3yz - 2xz &= 25, \\ xz - xy &= 4 \end{aligned}$$

egyenletrendszernek véges sok megoldása van, ugyanis:

```
GroebnerBasis[{2 x y + y z - 27,
      3 y z - 2 x z - 25, x z - x y - 4}, {x, y, z}]
                 2
{-25 + z , 5 y - 3 z, 5 x - 2 z}
```

A `Solve` függvény meghatározza a gyököket:

```
Solve[{2 x y + y z - 27 == 0,
       3 y z - 2 x z - 25 == 0,
       x z - x y - 4 == 0}, {x, y, z}]
```

```
{{x -> -2, y -> -3, z -> -5},
  {x -> 2, y -> 3, z -> 5}}
```

Tekintsük most a következő egyenletrendszert:

$$\begin{aligned} zx + yx - x + z^2 &= 2, \\ xy^2 + 2zx - 3x + z + y &= 1, \\ 2z^2 + zy^2 - 3z + 2zy + y^3 - 3y &= 0. \end{aligned} \tag{4}$$

Próbáljuk megoldani ezt a `Solve` függvény segítségével. A *Mathematica* 2.2.2. változatával mi ezt az eredményt kaptuk:

```
Solve[{z x + y x - x + z^2 - 2 == 0,
       x y^2 + 2 z x - 3 x + z + y - 1 == 0,
       2 z^2 + z y^2 - 3 z + 2 z y + y^3 - 3 y == 0},
          {x, y, z}]
Solve::svars:
  Warning: Equations may not give solutions for all
    "solve" variables.
Out of memory. Exiting.
```

Ez azt jelenti, hogy a `Solve` eljárás nem tudta megoldani a (4) egyenletet. Határozzuk meg a (4) egyenletrendszerhez tartozó Gröbner-bázist:

```
GroebnerBasis[{z x + y x - x + z^2 - 2,
    x y^2 + 2 z x - 3 x + z + y - 1,
    2 z^2 + z y^2 - 3 z + 2 z y + y^3 - 3 y}, {x, y, z}]

                  2      3      4      5      6
{-10 + 8 z + 15 z  - 8 z  - 7 z  + 2 z  + z ,
                   2      3     4
  5 + y - 3 z - 5 z  + 2 z  + z ,
                  2      2     4
  -4 - 2 x + 4 z  + x z  - z }
```

A korábban kimondott állítást figyelembe véve ebből az eredményből azt kapjuk, hogy a szóban forgó egyenletrendszernek végtelen sok megoldása van. A fenti Gröbner-bázis segítségével felírhatjuk a (4) rendszerrel ekvivalens „háromszög alakú” egyenletrendszert is:

$$\begin{aligned} -2x + 4z^2 + xz^2 - z^4 &= 4, \\ y - 3z - 5z^2 + 2z^3 + z^4 &= -5, \\ 8z + 15z^2 - 8z^3 - 7z^4 + 2z^5 + z^6 &= 10, \end{aligned} \tag{5}$$

amit a `Solve` eljárás már meg tud oldani:

```
egy = {%[[1]] == 0, %[[2]] == 0, %[[3]] == 0};
Solve[egy, {x, y, z}];
```

```
Short[%, 1]
Solve::svars:
  Warning: Equations may not give solutions for all
    "solve" variables.
{{y -> 1 - Sqrt[2], z -> Sqrt[2]}, <<7>>}
```

A kapott eredményből és az (5) első egyenletéből következik, hogy az $(x, 1-\sqrt{2}, \sqrt{2})$ számhármas minden $x \in \mathbf{C}$ esetén kielégíti (4)-et, aminek tehát végtelen sok megoldása van. A program ebben az esetben csak néhány (véges sok) gyököt ad meg, és figyelmeztet bennünket arra, hogy lehetnek más megoldások is.

A `Reduce` eljárás *paramétert* tartalmazó egyenletrendszer gyökeit „diszkusszióval együtt" adja meg. Például:

```
Reduce[{a*(1 + x*y) == x - y, 2 + x + y + x y == 0},
       {x, y}];
Simplify[%]
```

$$
\begin{aligned}
&\texttt{x == -3 \&\& y == } -\frac{1}{2} \texttt{ \&\& a == -1 ||}\\
&\quad \texttt{x == } -\frac{1}{2} \texttt{ \&\& y == -3 \&\& a == 1 ||}\\
&\quad \texttt{-1 + a != 0 \&\& 1 + a != 0 \&\&}\\
&\quad \texttt{x == } \frac{\texttt{-(2 + a + Sqrt[-4 + 5 a}^{\texttt{2}}\texttt{)}}{\texttt{2 (1 + a)}} \texttt{ \&\&}\\
&\quad \texttt{y == } \frac{\texttt{2 - a + Sqrt[-4 + 5 a}^{\texttt{2}}}{\texttt{2 (-1 + a)}} \texttt{ ||}\\
&\quad \texttt{-1 + a != 0 \&\& 1 + a != 0 \&\&}\\
&\quad \texttt{x == } \frac{\texttt{-2 - a + Sqrt[-4 + 5 a}^{\texttt{2}}}{\texttt{2 (1 + a)}} \texttt{ \&\&}\\
&\quad \texttt{y == } \frac{\texttt{2 - a - Sqrt[-4 + 5 a}^{\texttt{2}}}{\texttt{2 (-1 + a)}}
\end{aligned}
$$

Paramétereket tartalmazó egyenletrendszereket a paraméterekre oldja meg a

`SolveAlways`

függvény. Pontosabban: ez az eljárás a paraméterek azon értékeit keresi, amelyekre a megadott feltételek a változók minden lehetséges komplex értékére teljesülnek.

Határozzuk meg például az a, b és $c \in \mathbf{C}$ paramétereket úgy, hogy az $ax^2 + bxy + cy^2 = 1$ egyenlőség teljesüljön minden olyan $(x, y) \in \mathbf{C}^2$ számpárra, amelyre az $x + y = 1$ is fennáll.

A feladatot a *Mathematicá*val így oldhatjuk meg:

```
SolveAlways[
  Implies[x + y == 1, a x^2 + b x y + c y^2 == 1], {x, y}
{{a -> 1, b -> 2, c -> 1}}
```

Oldjuk meg a következő feladatot is. Jelöljük a $3x^2 + ax + b = 0$ egyenlet gyökeit x_1-gyel és x_2-vel, az

$$f(x) := \frac{5x + 2}{2x - 1} \qquad (x \in \mathbf{R} \setminus \{1/2\})$$

lineáris törtfüggvénynek az x_1 és x_2 helyeken felvett értékeit y_1- és y_2-vel. Határozzuk meg az a és b értékeket, ha tudjuk, hogy y_1 és y_2 kielégíti a $7y^2 - 5y - 11 = 0$ egyenletet.

A megoldás a *Mathematicá*val a következő:

```
f[x_] = (5x + 2)/(2x - 1);
SolveAlways[
  Implies[3x^2 + a x + b == 0, 7f[x]^2 - 5f[x] - 11 == 0], {x}]
{{a -> 7, b -> 1}}
```

A `Solve` függvényhez hasonlóan a `SolveAlways` függvény is elsősorban polinomokat tartalmazó egyenletrendszerek megoldásához nyújt segítséget.

A *Mathematica* bizonyos *nem polinomokat* tartalmazó egyenletrendszerek megoldására is képes. A lehetőségek az egyismeretlenes egyenletekhez hasonlóak (lásd az Egyéb egyenletek című alpontot). Itt csupán két olyan példát mutatunk, amelyet a program meg tud oldani:

```
Solve[{x*y/(x+y) == 8/3, y*z/(y+z)== 12/5,
       z*x/(z+x) == 24/7}, {x, y, z}]
{{x -> 8, y -> 4, z -> 6}}

Solve[{Sqrt[x + y] + Sqrt[x - y] == 10,
       Sqrt[x^2 - y^2] == 9}, {x, y}]
{{x -> 41, y -> -40}, {x -> 41, y -> 40}}
```

3.3.3. Egyenletek közelítő megoldása

A *Mathematica* több lehetőséget is kínál egyenletek és egyenletrendszerek közelítő megoldásainak előállítására.

A továbbiakban jelezzük azt, hogy a program készítői milyen numerikus módszert építettek be az egyes eljárásokba és a beépített algoritmusok közül hogyan választhatjuk ki az igényeinknek megfelelőt.

A közelítő megoldásokat akár több ezer értékes jegyre is meghatározhatjuk a *Mathematicá*val. Ezt több esetben az opciók alkalmas megválasztásával tehetjük meg. A numerikus függvények számos opcióval rendelkeznek. Ezek jelentéséről és használatuk módjáról a [89] referenciakönyvből szerezhetünk információt. Ha a feladatunk megoldásához ez nem elegendő, akkor érdemes tanulmányozni a [40] és a [41] dolgozatot.

• **Az `N[Reduce[ ]]`, az `N[Roots[ ]]` és az `N[Solve[ ]]` eljárás**

Az `N` numerikus függvényt alkalmazhatjuk a pontos megoldásokat kereső `Reduce`, `Roots` és `Solve` függvényekre. A továbbiakban ezek közül csak a `Solve` függvényt tanulmányozzuk.

Ebben az esetben a *Mathematica* először az egyenlet(rendszer) *összes* pontos megoldását keresi. Ha megtalálja (elsősorban polinomot tartalmazó egyenletek és egyenletrendszerek esetében), akkor veszi azok közelítő értékét:

```
N[Solve[x^2 + 2x - 7 == 0, x], 20]

{{x -> -3.8284271247461900976},
 {x -> 1.8284271247461900976}}
```

Figyelmeztető üzenetet küld akkor, ha az egyenlet(rendszer)nek a megtalált gyök(ök)ön kívül más megoldásai is lehetnek:

```
N[Solve[Cos[x] == 1/2, x]]

Solve::ifun:
Warning: Inverse functions are being used by Solve,
         so some solutions may not be found.
{{x -> 1.0472}}
```

Ha a program szimbolikus módszerek alkalmazásával nem tudja meghatározni a pontos megoldásokat, akkor numerikus módszerek felhasználásával dolgozik tovább:

```
N[Solve[x^5 - 4x + 2 == 0, x]]
{{x -> -1.51851}, {x -> -0.116792 - 1.43845 I},
 {x -> -0.116792 + 1.43845 I}, {x -> 0.508499},
 {x -> 1.2436}}
```

• Az `NRoots` és az `NSolve` eljárás

Az `NSolve` és az `N[Solve[ ]]` eljárás között az eredményt tekintve nincs különbség. Az `NSolve` függvénybe beépített algoritmusok közvetlenül numerikus módszerek alkalmazásával keresik a megoldás(ok) egy közelítő értékét, ezért általában az `NSolve` eljárással gyorsabban kapjuk meg a választ, mint az `N[Solve[ ]]` eljárással:

```
Timing[NSolve[{x y^2 + y + x^2 - 1 == 0,
        1 + x + x y^2 == 0}, {x, y}];]
{21.366 Second, Null}

Timing[N[Solve[{x y^2 + y + x^2 - 1 == 0,
        1 + x + x y^2 == 0}, {x, y}]];]
{432.479 Second, Null}
```

A megoldásoknak akár több ezer értékes jegyét is megkaphatjuk. Ha az `NSolve` függvény harmadik argumentumába egy konkrét n természetes számot írunk, akkor a *Mathematica* n értékes jegyet tartalmazó számok (l. a 3.1.3. pontot) alkalmazásával oldja meg az egyenletet:

```
NSolve[Sqrt[x] == 1 + (1/5) x, x, 20]
{{x -> 1.90983005625052575898},
 {x -> 13.090169943749474241}}
```

• A `FindRoot` függvény

Az előzőekben ismertetett *Mathematica*-függvények több esetben nem találják meg a beadott egyenlet egyébként létező megoldását.

Tegyük fel például azt, hogy a

$$\operatorname{tg} x = 2x + \frac{1}{4} \tag{1}$$

egyenlet valós gyökeit szeretnénk meghatározni.

A pontos megoldások explicit előállításában nem reménykedhetünk, ezért először a numerikus megoldást szolgáltató `NSolve` függvénnyel próbálkozunk:

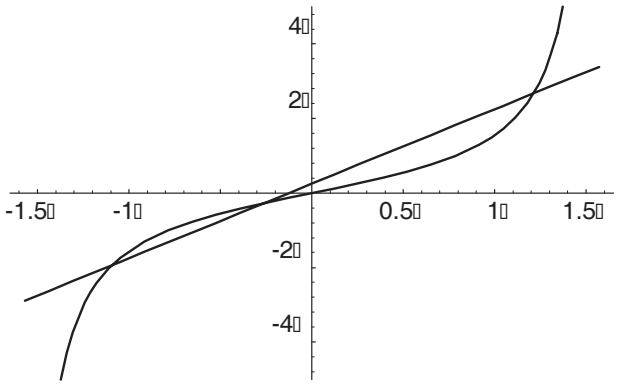
```
NSolve[Tan[x] == 2x + 1/4, x]
Solve::tdep:
     The equations appear to involve transcendental
       functions of the variables in an essentially
       non-algebraic way.
NSolve[Tan[x] == 1/4 + 2 x, x]
```

Az ehhez hasonló esetekben a `FindRoot` belső függvényt használhatjuk. A matematikában egyenlet(rendszer)ek numerikus megoldására számos iterációs eljárás ismeretes. Ezek közös vonása az, hogy egy (vagy több) kezdőértékből kiindulva képezünk olyan sorozatot, amelynek határértéke a keresett (pontos) megoldás. A `FindRoot` belső függvény is ilyen módszereket alkalmaz.

Az elmondottakból az is következik, hogy ha a `FindRoot` függvényt kívánjuk használni, akkor egyrészt kezdőértéket (azaz az iterációs sorozat első tagját) is meg kell adnunk, másrészt csupán egyetlen (a kezdőértékhez legközelebbi) megoldást fogjuk megkapni. A kezdeti közelítés „jó” megválasztásához (azaz ahhoz, hogy az „elég közel” legyen valamelyik megoldáshoz) a *Mathematica* egyéb függvényei által biztosított lehetőségeket is felhasználhatjuk.

A feladatunkhoz visszatérve tekintsük például a következő ábrát:

```
Plot[{Tan[x], 2x + 1/4}, {x, -Pi/2, Pi/2},
      PlotRange -> {-5, 5}]
```



Az ábráról leolvasható (szükség esetén be is bizonyítható), hogy az (1) egyenletnek a $(-\pi/2, \pi/2)$ intervallumban pontosan három különböző valós gyöke van. A kezdőérték megválasztásához is kapunk segítséget:

```
FindRoot[Tan[x] == 2x + 1/4, {x, 1}]
{x -> 1.21323}

FindRoot[Tan[x] == 2x + 1/4, {x, -0.1}]
{x -> -0.255724}

FindRoot[Tan[x] == 2x + 1/4, {x, -1.}]
{x -> -1.09475}
```

Másik intervallumba eső gyököt is hasonló módon határozhatunk meg.

A `FindRoot` függvénybe három különböző algoritmust építettek be: a Newton-féle érintőmódszert, a Brent-féle eljárást, valamint a szelőmódszert. Érdekes módon a felhasználó ebben az esetben nem opció megadásával, hanem az utasítás szintaxisával tudja kiválasztani a feladatának megfelelő algoritmust.

Ha az

$$f(x) = 0 \qquad ?(x \in \mathbf{C})$$

egyenlet megoldásához a

```
FindRoot[f[x] == 0, {x, x0}]
```

alakban megadott utasítást használjuk, akkor a *Mathematica* a Newton-féle módszert alkalmazza, azaz az x_0 kezdeti közelítésből kiindulva képezi az

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)} \qquad (n \in \mathbf{N})$$

sorozatot. Az f függvény deriváltját szimbolikusan határozza meg.

Ha az f függvény nem differenciálható, akkor ennek közlésével befejezi a kiértékelést:

```
FindRoot[Abs[x] == Cos[x], {x, -1}]
FindRoot::frjc:
    Could not symbolically find the
      Jacobian of {Abs[x] - Cos[x]}. Try
      giving two starting values for each variable.
FindRoot[Abs[x] == Cos[x], {x, -1}]
```

A `FindRoot` második argumentumában azonban két kezdőértéket is megadhatunk:

```
FindRoot[f[x] == 0, {x, x0, x1}]
```

A *Mathematica* ekkor az intervallumfelezésen alapuló Brent-féle eljárást használja abban az esetben, ha a két adott pontban felvett függvényérték különböző előjelű. Egyébként a program a szelőmódszert alkalmazza:

```
FindRoot[Abs[x] == Cos[x], {x, -1, 0}]
{x -> -0.739085}
```

Az $f(x) = 0$ egyenlet $[a, b]$ intervallumba eső gyökét a c kezdőértékből kiindulva a

```
FindRoot[f[x] == 0, {x, c, a, b}]
```

utasítással kapjuk meg. A program ebben az esetben a Newton-féle érintőmódszert alkalmazza. Például:

```
FindRoot[Sin[x] - x^2/4 == 0, {x, 2, 1, 3}]
{x -> 1.93375}
```

Az előzőekhez hasonlóan két kezdeti közelítést is megadhatunk:

```
FindRoot[f[x] == 0, {x, c1, c2, a, b}]
```

Ez az eljárás a Brent-féle módszerrel vagy pedig a szelőmódszerrel számítja ki a megoldást. Például:

```
FindRoot[Sin[x] - x^2/4 == 0, {x, -2, -1, -4, 1}]
{x -> 9.17319 10^-8}
```

A következő szintaxist is használhatjuk:

```
FindRoot[Sin[x] - x^2/4 == 0, {x, {-3, -2}, -4, 1}]
FindRoot::regex:
   Reached the point {1.96266} which is outside the region
    {{-4., 1.}}.
{x -> 1.96266}
```

A `FindRoot` függvényt olyan *egyenletrendszerek* megoldásához is használhatjuk, amelyekben az egyenletek száma megegyezik az ismeretlenek számával. „Jó” kezdőérték megválasztásához ebben az esetben a `Plot3D`, a `ContourPlot` vagy a `FindMinimum` függvény adhat segítséget:

```
FindRoot[{Sin[x] + y^2 + Log[z] == z,
                3x + 2y^2 - z^3 == 1,
                      x + y + z == 5}, {x, 0}, {y, 0}, {z, 2}]
{x -> 2.33687, y -> 0.740478, z -> 1.92265}
```

3.4. Analízis

Ebben a szakaszban azt szeretnénk megmutatni, hogy az analízis témaköréhez tartozó problémák megoldása során hogyan használhatjuk fel a *Mathematicá*t. A definíciókat és a tételeket kevés kivételtől eltekintve nem fogalmazzuk meg. Ezeket a [20], [44], [48], [61], valamint a [72] bevezető könyvek mindegyike tartalmazza.

Sok feladatot a matematikában megszokott módon *pontosan* (*szimbolikusan*) is megoldhatunk, például a `Limit`, a `D`, a `Sum`, az `Integrate` eljárás segítségével. A program által adott eredmények értékelésénél azonban vegyük figyelembe azt, hogy a beadott matematikai függvényeket a *Mathematica* a legtöbb esetben komplex változósaknak, a paramétereket pedig komplex értékűeknek tekinti és a kiértékelés során csak néhány azonosságot alkalmaz automatikusan. A kapott eredményt más eljárások segítségével vagy transzformációs szabályok megadásával alakíthatjuk át az igényeinknek megfelelő alakra.

Sok esetben azonban célszerűbb *numerikus módszereket* alkalmazó függvényekkel (például `NLimit`, `NSum` és `NIntegrate`) kiszámolni a pontos eredmény egy közelítő értékét. Ekkor opciók alkalmas megadásával egyrészt több numerikus módszer közül is válogathatunk, másrészt igen tág határok között szabályozhatjuk a kiértékelésnél felhasznált pontos számjegyek számát is. A numerikus függvények opcióinak „optimális” vagy az igényeinknek megfelelő (például egy meghatározott módszerrel kívánunk egy feladatot megoldani) megválasztása sokszor nem egyszerű feladat. A [89] referenciakönyvön kívül a [40] és a [41] dolgozatból ismerhetjük meg a numerikus függvények opcióinak jelentését és használatuk módját. A szerzők számos példával illusztrálják a lehetőségeket.

Az alkalmazott numerikus módszerek elméletét illetően a [47], [65], [73] és a [78] könyveket ajánljuk az Olvasó figyelmébe.

Végül megemlítjük még azt is, hogy több könyvet is szenteltek már annak a kérdésnek a megválaszolására, hogy a *Mathematicá*t miképpen lehet felhasználni az analízis oktatásában. Az Irodalomjegyzékben feltüntetett [12], [21] és [77] műveken kívül továbbiakat is találhat az érdeklődő Olvasó a MathSource-on.

3.4.1. Beépített speciális függvények

A *Mathematica* a 3.1.4. pontban felsorolt matematikai függvényeken kívül az alábbi függvényeket is tartalmazza:

AiryAi	Gamma
AiryAiPrime	GegenbauerC
AiryBi	HermiteH
AiryBiPrime	Hypergeometric0F1
BernoulliB	Hypergeometric1F1
BesselI	Hypergeometric2F1
BesselJ	HypergeometricU
BesselK	InverseJacobiSN
BesselY	JacobiAmplitude
Beta	JacobiP
ChebyshevT	JacobiSN
ChebyshevU	JacobiSymbol
CosIntegral	JacobiZeta
EllipticE	LaguerreL
EllipticExp	LegendreP
EllipticF	LerchPhi
EllipticK	LogGamma
EllipticLog	LogIntegral
EllipticPi	Pochhammer
EllipticTheta	PolyGamma
EulerE	PolyLog
EulerPhi	RiemannSiegelTheta
ExpIntegralE	RiemannSiegelZ
ExpIntegralEi	SinIntegral
FresnelC	SphericalHarmonicY
FresnelS	Zeta

A `Calculus`DiracDelta`` programcsomagban találhatók az alábbi eljárások:

DiracDelta	UnitStep

Speciális függvényeket különböző módon (például a normálás különböző megválasztásával) szokás értelmezni a matematikában. A felsorolt beépített függvények pontos definícióját a [89] referenciakönyv tartalmazza.

Néhány olyan esetben, amikor a fenti függvények argumentumába csak pontos numerikus értékeket vagy beépített matematikai állandókat (lásd a 3.1.3. pontot) írunk, a pontos helyettesítési értéket kapjuk meg:

```
{Gamma[3/2], Gamma[15/2], Zeta[6], PolyGamma[6]}
```

$$\left\{\frac{\text{Sqrt[Pi]}}{2},\ \frac{135135\ \text{Sqrt[Pi]}}{128},\ \frac{\text{Pi}^6}{945},\ \frac{137}{60}\ \text{- EulerGamma}\right\}$$

Az `N` függvényt (lásd a 3.1.3. pontot) használva azonban bármely komplex szám esetén előállíthatjuk a helyettesítési érték egy közelítését:

```
N[Gamma[3 + 4 I], 17]
0.00522553847136921 - 0.17254707929430019 I
```

Szemléltethetjük ezeket a függvényeket. Próbáljuk ki például a következő utasítást:

```
Plot[Abs[Zeta[1/2 + I*y]], {y, 0, 40}]
```

A program speciális függvények között fennálló számos összefüggést ismer, és ezeket alkalmazni is tudja. Például:

```
LegendreP[5, x]
                  3       5
15 x - 70 x  + 63 x
-------------------
         8
Integrate[LegendreP[5, x] LegendreP[6, x], {x, -1, 1}]
0
D[Gamma[x], {x, 2}]
                                2
Gamma[x] PolyGamma[0, x]  + Gamma[x] PolyGamma[1, x]
```

Itt említjük meg a

```
NumericalMath`BesselZeros`
```

programcsomagot, amelynek segítségével különböző típusú Bessel-féle függvények zérushelyeinek közelítő értékét kaphatjuk meg. A

```
<<NumericalMath`BesselZeros`
?Bessel*
```

utasítások végrehajtása után tájékozódhatunk a közvetlenül felhasználható függvények széles köréről.

3.4.2. Számsorozatok és számsorok

Mivel a *Mathematica* komplex számok kezelésére is képes, ezért komplex tagú sorozatokkal végezhetünk különféle műveleteket. Ebben a pontban a programnak azokat a lehetőségeit ismertetjük, amelyeket számsorozatok és számsorok konvergenciájának vizsgálatánál felhasználhatunk.

• Sorozatok megadása

Számsorozatot (azaz $\mathbf{N} \to \mathbf{C}$ típusú függvényt) matematikai szempontból korrekt módon például így adhatunk meg:

```
a[n_ /; (IntegerQ[n]==True && Positive[n]==True)] := n^(1/n)
```

Ezt a függvényt a program is sorozatnak tekinti:

```
N[{a[-1.7], a[0], a[1], a[2.5], a[2], a[5]}]
{a[-1.7], a[0], 1., a[2.5], 1.41421, 1.37973}
```

A sorozat határértékét azonban csak akkor számolja ki, ha a definícióban nem adunk meg feltételeket. Ezért a továbbiakban sorozatot így fogunk megadni:

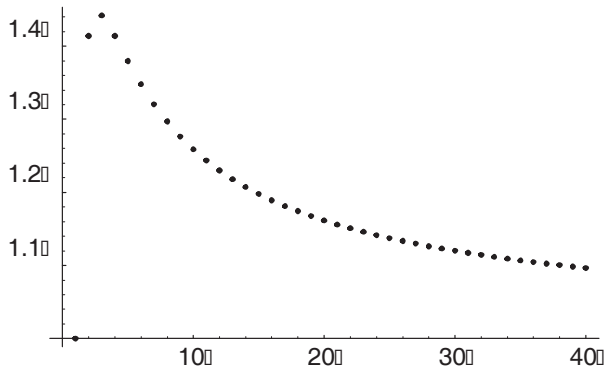
```
Clear[a]
a[n_] := n^(1/n)
b[n_] := (1 + I Pi/(2n))^n
```

Sorozat viselkedéséről gyorsan alkothatunk megbízható képet, egyrészt tetszőlegesen sok függvényérték meghatározásával. A b sorozat első 20 helyettesítési értékének egy közelítését például így kaphatjuk meg:

```
Table[N[b[n]], {n, 1, 20}]
```

Felhasználhatjuk a *Mathematica* grafikai lehetőségeit is. Például az *a valós* sorozatot a `ListPlot` függvénnyel így szemléltethetjük:

```
pontok = Table[a[k], {k, 1, 40}];
ListPlot[pontok]
```



A 3.1.4. pontban volt már arról szó, hogy *rekurzív eljárással* értelmezett sorozatokat kétféleképpen adhatunk meg. Ha a rekurzív módon megadott sorozat tagjai között lineáris vagy konvolúciós típusú összefüggés áll fenn, akkor a `DiscreteMath`RSolve`` programcsomag

```
RSolve
```

eljárása az index függvényében állítja elő a tagokat. A 3.6.4. pontban mutatunk erre vonatkozó példákat.

• Számsorozat határértéke

Az $(a_n) : \mathbf{N} \to \mathbf{C}$ sorozatot konvergencia szempontjából a

```
Limit[a[n], n -> Infinity]
```

utasítással vizsgálhatjuk. A beépített algoritmusok ebben az esetben a sorozat (tágabb értelemben vett) határértékét keresik. Például:

```
Limit[(1 - 1/(n+1))^(n+1), n -> Infinity]
```

$$\frac{1}{E}$$

```
Limit[(1 + I Pi/(2n))^n, n -> Infinity]
```

```
I
```

```
Limit[(n^2-4n)/(2n-7), n -> Infinity]
```

```
Infinity
```

A sorozat definíciója paramétert is tartalmazhat:

```
Limit[n*(a^(1/n) - 1), n -> Infinity]
```

```
Log[a]
```

Azt a tényt, hogy a beadott sorozat nem konvergens, a *Mathematica* így közli a felhasználóval:

```
Limit[(-1)^n, n -> Infinity]
```

```
Indeterminate
```

Néhány esetben változatlan formában kapjuk vissza a begépelt utasítást:

```
Limit[n^k/a^n, n -> Infinity]
```

$$\text{Limit}\left[\frac{n^k}{a^n}, \text{n -> Infinity}\right]$$

Ez azt jelenti, hogy a program nem tudta meghatározni a kérdezett határértéket. Ilyenkor a

```
Calculus`Limit`
```

programcsomag `Limit` nevű függvényét érdemes kipróbálni, amely nagyobb teljesítményű, mint belső társa:

```
<<Calculus`Limit`
Limit[n^k/a^n, n -> Infinity]
```

$$E^{\text{(-Infinity) Sign[Log[a]]}}$$

Az a és a b szám *Gauss-féle számtani-mértani közepét* az

```
ArithmeticGeometricMean[a, b]
```

utasítással kapjuk meg. Emlékeztetünk arra (lásd például [78]-at), hogy ha a és b nemnegatív valós szám, akkor az $a_0 := a$, $b_0 := b$,

$$a_{n+1} := \frac{a_n + b_n}{2}, \qquad b_{n+1} := \sqrt{a_n b_n} \qquad (n \in \mathbf{N})$$

formulával értelmezett (a_n) és (b_n) sorozat konvergens, és

$$\lim_{n\to+\infty}(a_n) = \lim_{n\to+\infty}(b_n) =: M(a,b).$$

Ezt a közös határértéket nevezzük az a és a b szám *Gauss-féle számtani-mértani közepé*nek:

```
ArithmeticGeometricMean[Sqrt[2], 1]
```

```
ArithmeticGeometricMean[1, Sqrt[2]]
```

```
N[%, 20]
```

```
1.1981402347355922074
```

Matematikatörténeti érdekesség, hogy a tinédzser Gauss már 1791-ben meghatározta a $\sqrt{2}$ és az 1 számtani-mértani közepének első 20 pontos jegyét. Megjegyezzük még azt is, hogy az $M(a,b)$ számokat elliptikus integrálokkal lehet kifejezni. Gauss 1799-ben igazolta az alábbi összefüggést:

$$\left(\int_0^1 \frac{dt}{\sqrt{1-t^4}}\right) \cdot M(\sqrt{2},1) = \frac{\pi}{2}.$$

A `Limit` belső és külső függvény *szimbolikus* (*analitikus*) *módszer* alkalmazásával számítja ki a határértéket. A `SequenceLimit` belső függvény, valamint a `NumericalMath`NLimit`` programcsomagban levő `NLimit` külső függvény a sorozat néhány tagjából *numerikus módszert* felhasználva adja meg a kérdezett határérték egy közelítését. A

`SequenceLimit`

belső függvényt olyan sorozat határértékének meghatározásához célszerű használni, amelynek az n-edik tagja

$$\sum_{k=1}^{n} P_i(n)\lambda_i^n$$

alakú, ahol P_i-k polinomok és λ_i-k különböző számok. Ebbe a függvénybe a Wynn-féle ε-algoritmust építették be. Ezt az algoritmust alkalmazza az

`NLimit`

függvény is, ha a `Method` opciójának értéke `SequenceLimit`. A határérték egy közelítő értékét az általánosított Euler-féle transzformáció módszerével kapjuk meg akkor, ha a `Method` opció értéke `EulerSum`.

A fenti függvények számos további opcióval is rendelkeznek. Ezekkel egyrészt az alkalmazott numerikus módszer paramétereit, másrészt a kiértékelés során felhasznált értékes számjegyek számát módosíthatjuk (lásd a [40] dolgozatot).

• Sorok konvergenciája

Hasznos segítséget nyújthat az `Algebra`SymbolicSum`` programcsomag

`Sum`

függvénye. Ismeretes, hogy ha az a_{k+1}/a_k ($k \in \mathbf{N}$) hányados a k racionális

függvénye, akkor a

$$\sum_{k=1}^{n} a_k, \qquad \text{valamint a} \qquad \sum_{k=1}^{\infty} a_k$$

összeg kifejezhető a hipergeometrikus függvényekkel. Ilyen eljárásokat építettek be ebbe a `Sum` függvénybe (lásd [2]-t).

Olvassuk be ezt a programcsomagot:

```
<<Algebra`SymbolisSum`
```

Ezután a $\sum a_k$ számsor konvergenciáját többféle módszerrel is vizsgálhatjuk. Szerencsés esetben a részletösszegekre zárt formulát kapunk, és ekkor a `Limit` (belső vagy külső) függvényt alkalmazhatjuk:

```
a[n_] = Sum[(-1)^k*(4k+8)/((2k+3)*(2k+5)), {k, 0, n}]
```

$$\frac{5 + 3\ (-1)^{n} + 2\ n}{3\ (5 + 2\ n)}$$

```
Limit[a[n], n -> Infinity]
```

$$\frac{1}{3}$$

A `Sum` függvény segítségével számos konvergens sor összegét a `Limit` függvény használata nélkül is meghatározhatjuk. Például:

```
Sum[(-1)^k*(4k+8)/((2k+3)*(2k+5)), {k, 0, Infinity}]
```

$$\frac{1}{3}$$

Ha a fenti módszerekkel nem kapjuk meg a kívánt eredményt, akkor a konvergencia eldöntéséhez az ismert kritériumok valamelyikét használhatjuk. Tekintsük például az

$$A := \sum n \operatorname{tg} \frac{\pi}{2^{n+1}}$$

pozitív tagú numerikus sort. Alkalmazzuk például a hányadoskritériumot:

```
Clear[a]
a[n_] := n Tan[Pi/2^(n+1)]
Limit[a[n+1]/a[n], n -> Infinity]
```

$$\frac{1}{2}$$

A szóban forgó sor tehát konvergens.

A *Mathematica* több lehetőséget is tartalmaz számsor összegének *közelítő* meghatározására. Felhasználhatjuk egyrészt az előzőekben említett `NLimit` külső függvényt, másrészt az

`NSum`

belső függvényt. Különböző numerikus módszerek természetesen különböző eredményeket adhatnak. Az `NSum` függvénynél választható eljárásokat a következő példákon mutatjuk be.

A $\sum 1/k^2$ sor összegének egy közelítő értékét az Euler–Maclaurin-féle módszerrel így számolhatjuk ki:

```
NSum[1/k^2, {k, 1, Infinity}, Method -> Integrate];
InputForm[%]
1.64493406676001
```

A kiértékeléshez a Wynn-féle ε-algoritmust így választhatjuk:

```
NSum[1/k^2, {k, 1, Infinity}, Method -> SequenceLimit];
InputForm[%]
1.643485586660751
```

Hasonlítsuk most össze a kapott eredményeket a pontos értékkel:

```
Sum[1/k^2, {k, 1, Infinity}]
```

$$\frac{\text{Pi}^2}{6}$$

```
N[%, 20]
1.6449340668482264365
```

Az előzőekben értelmezett A konvergens számsor összegének egy közelítő értékét tehát így kapjuk meg:

```
NSum[a[k], {k, 1, Infinity}]
3.40841
```

(Az `NSum` függvény `Method` opciója alapértelmezésben `Automatic`. Ez azt jelenti, hogy a program választja ki a kiértékelésnél a felhasznált numerikus módszert.)

A `NumericalMath`NLimit`` programcsomag

`EulerSum`

függvénye az Euler-féle transzformációs eljárást alkalmazza *váltakozó előjelű* sor összegének numerikus meghatározásához:

```
<<NumericalMath`NLimit`
EulerSum[(-1)^k/(2k+1), {k, 0, Infinity},
        WorkingPrecision -> 40, Terms -> 30,
        ExtraTerms -> 30]
```

```
0.7853981633974483096156608457913032254 02
```

```
% - N[Pi/4, 40]
```

$$-2.8572495647\ 10^{-29}$$

Ezzel a függvénnyel néhány esetben a pontos összeget is megkaphatjuk:

```
EulerSum[(k^2-4k+1)/2^k - (4k^2-k+2)/3^k, {k, Infinity},
     EulerRatio -> {{1/3, 3}, {1/2, 3}},
     ExtraTerms -> 6, Terms -> 0,
     WorkingPrecision -> Infinity]
```

$$-(\frac{29}{4})$$

3.4.3. Függvények határértéke

$\mathbf{C} \to \mathbf{C}$ típusú f függvény x_0 pontban vett határértékét a

```
Limit[f[x], x -> x0]
```

utasítással kaphatjuk meg. Felhasználhatjuk a belső `Limit` függvényt és a `Calculus`Limit`` programcsomag azonos nevű függvényét is, amely nagyobb teljesítményű, mint belső társa. Szükség esetén olvassuk be ezt a programcsomagot:

```
<<Calculus`Limit`
```

A *Mathematica* a fenti utasítás hatására szimbolikus algoritmusok alkalmazásával próbálja megadni a *pontos* eredményt. Kereshetjük a határértéket adott valós vagy komplex pontban:

```
Limit[1/Sin[x] - 1/Tan[x], x -> 0]
```

```
0
```

```
Limit[(z^2 + 1)/(I (z^4 - 1)), z -> I]
```

$$\frac{I}{2}$$

$+\infty$-ben, valamint $-\infty$-ben:

```
Limit[x - x^2 Log[1 + 1/x], x -> Infinity]
```

$$\frac{1}{2}$$

```
Limit[Sqrt[x^2+2x+2]-Sqrt[x^2-2x+2], x -> -Infinity]
```

```
-2
```

A függvény helyettesítési értéke paramétereket is tartalmazhat:

```
Limit[n/(1-x^n) - m/(1-x^m), x -> 1];
Together[%]
```

$$\frac{-m + n}{2}$$

Egyoldali határértéket a

Direction

opció alkalmas megválasztásával vizsgálhatunk. Ha ennek értéke `+1`, akkor a bal oldali, ha `-1`, akkor pedig a jobb oldali határértéket kapjuk meg:

```
Limit[1/(1+Exp[1/x]), x -> 0, Direction -> 1]
```

```
1
```

```
Limit[1/(1+Exp[1/x]), x -> 0, Direction -> -1]
```

```
0
```

Az opció alapértelmezés szerinti értékével (ez `Automatic`) a program néhány esetben a jobb oldali, máskor pedig az adott pontbeli határértéket adja meg:

```
Limit[1/(1+Exp[1/x]), x -> 0]
```

```
0
```

```
Limit[1/x, x -> 0]
```

```
ComplexInfinity
```

A következő példa azt mutatja, hogy nem kell feladni a reményt akkor, ha a `Limit` függvénnyel nem kapjuk meg a várt eredményt:

```
f[x_] := Sin[Sqrt[x+1]] - Sin[Sqrt[x]]
Factor[f[x], Trig -> True]
```

$$-2\,\mathrm{Cos}\left[\frac{\mathrm{Sqrt}[x]}{2} + \frac{\mathrm{Sqrt}[1 + x]}{2}\right]\,\mathrm{Sin}\left[\frac{\mathrm{Sqrt}[x]}{2} - \frac{\mathrm{Sqrt}[1 + x]}{2}\right]$$

```
Limit[%, x -> Infinity]
0
```

Könnyű ellenőrizni, hogy a kérdezett határérték valóban nullával egyenlő. Próbáljuk ki ezután a következő utasítást is:

```
Limit[f[x], x -> Infinity]
```

A `Limit` eljárás alapértelmezésben a L'Hospital-szabály alapján keresi a pontos eredményt. Ennek a függvénynek van egy `Analytic` nevű opciója is. Ha ezt az alapértelmezéssel ellentétben a `True` értékre állítjuk, akkor a program a Taylor-sorfejtést használva próbálja kiszámítani a határértéket.

A `NumericalMath`NLimit`` programcsomag

NLimit

függvénye a vizsgált pont környezetéből választott néhány pontban felvett függvényértékből *numerikus módszer* alkalmazásával adja meg a határérték egy közelítését. Tekintsük a következő példát:

```
<<NumericalMath`NLimit`
NLimit[((2^x+3^x+7^x)/3)^(1/x), x -> 0]
3.47603
```

A `Limit` függvény egyik attribútuma `Listable` és ez azt jelenti, hogy az argumentumában lista (vektor) is szerepelhet. A *Mathematica* ekkor a `Limit` függvényt a vektor minden elemére külön-külön alkalmazza. Ezért igen egyszerűen számolhatjuk ki $\mathbf{C} \to \mathbf{C}^n$ típusú függvények határértékét:

```
Clear[f]
f[x_] := {Cosh[x], Tan[x]/x, x^x}
Limit[f[x], x -> 0]
{1, 1, 1}
```

3.4.4. Differenciálszámítás

A $\mathbf{C} \to \mathbf{C}$ típusú f függvény x *pontban* vett deriváltját a

D[f[x], x]

utasítás eredménye adja meg abban az esetben, ha f-et beépített differenciálható függvényekkel definiáljuk:

```
Clear[f]
f[x_] := Sin[x^2]
D[f[x], x]
```

$$2\ x\ \mathrm{Cos}[x^2]$$

Ezt az eredményt így is megkaphatjuk:

```
f'[x]
```

$$2\ x\ \mathrm{Cos}[x^2]$$

```
D[Sin[x^2], x]
```

$$2\ x\ \mathrm{Cos}[x^2]$$

A *Mathematica* ismeri és alkalmazni is tudja a műveletek és a derivált között fennálló összefüggéseket:

```
D[f[x]^(1/g[x]), x]
```

$$\frac{f[x]^{-1+g[x]}\ f'[x]}{g[x]} - \frac{f[x]^{1/g[x]}\ \mathrm{Log}[f[x]]\ g'[x]}{g[x]^2}$$

A program a `D` függvény második argumentumától különböző szimbólumokat paraméternek tekinti. Szükség esetén a kapott eredményt más eljárások felhasználásával egyszerűsíthetjük:

```
D[b*x/a + 2 Sqrt[a^2-b^2]/a*
     ArcTan[Sqrt[(a-b)/(a+b)]*Tanh[x/2]], x];
Simplify[%];
PowerExpand[%];
% /. Sqrt[x_] Sqrt[y_] -> Sqrt[x y];
Together[%];
Simplify[%]
```

$$\frac{a + b\ \mathrm{Cosh}[x]}{b + a\ \mathrm{Cosh}[x]}$$

Szakaszonként különböző formulával értelmezett függvény deriváltját is meghatározhatjuk, ha a függvényt a `Which` eljárással értelmezzük.

```
g[x_] := Which[x>=0, x^3, x<0, -x^3]
g'[x]
```

$$\mathrm{Which}[x >= 0,\ 3\ x^2,\ x < 0,\ -(3\ x^2)]$$

Magasabb rendű deriváltat így számolhatunk ki:

```
D[Log[x]/x, {x, 5}]
```

$$\frac{274}{x^6} - \frac{120\ \text{Log}[x]}{x^6}$$

Adott függvény *deriváltfüggvényét* a

```
Derivative
```

eljárással kapjuk meg. Figyeljük meg, hogy a *Mathematica* milyen formában adja meg az eredményt (lásd a 3.1.4. pontot):

```
h[x_] := Sin[x] Exp[x]
Derivative[1][h]
```

```
E^#1 Cos[#1] + E^#1 Sin[#1] &
```

A deriváltfüggvény helyettesítési értéke:

```
%[x]
```

```
E^x Cos[x] + E^x Sin[x]
```

Magasabb rendű deriváltfüggvényeket is meghatározhatunk:

```
Derivative[2][h]
```

```
2 E^#1 Cos[#1] &
```

```
h''[x] == Derivative[2][h][x]
```

```
True
```

Adott f függvény n-edik deriváltjának x_0 pontbeli közelítő értékét az

```
ND[f[x], {x, n}, x0]
```

utasítással kapjuk meg. Ez az eljárás a `NumericalMath`NLimit`` programcsomagban található:

```
<<NumericalMath`NLimit`
ND[Exp[Sin[x]], {x, 3}, 2, Terms -> 10]-
   N[D[Exp[Sin[x]], {x, 3}] /. x -> 2]
```

```
2.23989 10^-7
```

• Többváltozós függvények deriváltjai

Parciális deriváltakat is a `D` függvénnyel számolhatunk ki. Adott $n > 1$ természetes szám esetén a $\mathbf{C}^n \to \mathbf{C}$ típusú f függvény x_j változója szerinti parciális deriváltfüggvényének $(x_1, \cdots, x_n)$ pontbeli helyettesítési értékét a

```
D[f[x1,...,xn], xj]
```

utasítással kapjuk meg. Például:

```
D[y^5 Cos[x] + x^4 Cos[y], x]
```

```
   3                5
4 x  Cos[y] - y  Sin[x]
```

```
D[y^5 Cos[x] + x^4 Cos[y], y]
```

```
   4                4
5 y  Cos[x] - x  Sin[y]
```

*Magasabb rendű parciális deriváltak*at így határozhatunk meg:

```
D[y^5 Cos[x] + x^4 Cos[y], {x, 2}, {y, 3}]
```

```
     2                2
-60 y  Cos[x] + 12 x  Sin[y]
```

A *Mathematica* feltételezi azt, hogy a beadott függvény elegendően sokszor differenciálható ahhoz, hogy a parciális deriváltak sorrendje felcserélhető legyen.

A következő példákban figyeljük meg azt, hogy a program hogyan jelöli a parciális deriváltakat:

```
D[f[x, y], x]
```

```
  (1, 0)
f      [x, y]
```

```
D[g[x, y, z], {x, 2}, {y, 3}]
```

```
  (2, 3, 0)
g         [x, y, z]
```

$\mathbf{C}^n \to \mathbf{C}$ típusú függvény *gradiens*ét a `D` eljárásnak, valamint a listakezelő függvényeknek a segítségével így értelmezhetjük:

```
gradiens[fuggveny_, valtozok_List] :=
                  D[fuggveny, #]& /@ valtozok
```

Itt felhasználtuk a *Mathematica* tisztafüggvény-fogalmát (lásd a 3.1.4. pontot) és a `Map` eljárás rövid alakját (`/@`). Figyeljük meg azt az érdekes tényt is, hogy a változók számától függetlenül értelmeztük a fenti függvényt.

A `ProgrammingExamples`VectorCalculus`` programcsomag `Grad` nevű függvénye is ezt a definíciót tartalmazza. Felhívjuk az Olvasó figyelmét arra, hogy a `Calculus`VectorAnalysis`` programcsomagban is található egy másik, az előzőtől különböző, `Grad` elnevezésű eljárás.

Próbáljuk most ki a fenti definíció működését:

```
gradiens[x^2 - 3 y^3, {x, y}]
```

$$\{2\,x,\ -9\,y^2\}$$

```
gradiens[x y + x z + y z, {x, y, z}]
```

$$\{y + z,\ x + z,\ x + y\}$$

Számítsuk most ki az $f(x,y,z) := \sin xyz$ $(x,y,z) \in \mathbf{R}^3$ függvénynek az $s = (-1,2,2) \in \mathbf{R}^3$ irány mentén vett *iránymenti deriváltj*át a $P_0(\pi, 1/2, 1/2)$ pontban:

```
f[x_, y_, z_] := Sin[x y z]
s = {-1, 2, 2};
EgysegVektor[u_List] := u/Sqrt[(Plus @@ (Abs[u]^2))]
EgysegVektor[s]
```

$$\left\{-\frac{1}{3},\ \frac{2}{3},\ \frac{2}{3}\right\}$$

```
gradiens[f[x, y, z], {x, y, z}] . EgysegVektor[s]
```

$$\frac{2\,x\,y\,\mathrm{Cos}[x\,y\,z]}{3} + \frac{2\,x\,z\,\mathrm{Cos}[x\,y\,z]}{3} - \frac{y\,z\,\mathrm{Cos}[x\,y\,z]}{3}$$

```
% /. {x -> Pi, y -> 1/2, z -> 1/2} // Simplify
```

$$\frac{-1 + 8\,\mathrm{Pi}}{12\,\mathrm{Sqrt}[2]}$$

(A `.` jel a `Dot` függvény rövid alakja. Ezzel az eljárással számolhatjuk ki két valós vektor skaláris szorzatát.)

Ha a $\mathbf{C}^n \to \mathbf{C}^m$ $(n,m \in \mathbf{N})$ típusú $f = (f_1, \cdots, f_m)$ függvény *(totálisan) differenciálható* az $a \in \mathbf{C}^n$ pontban, akkor az a-beli (totális) deriváltja a következő $m \times n$-es (Jacobi-)mátrix:

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \cdots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \cdots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix}.$$

Ezt a mátrixot adja eredményként a következő függvény:

```
SetAttributes[Rule, Listable]
DerivaltMatrix[
     fuggveny_List, valtozok_List, pont_List] :=
         Outer[D, fuggveny, valtozok] /. valtozok -> pont
```

Az első utasításban a transzformációs szabályt megadó `Rule` függvényt láttuk el a `Listable` attribútummal (lásd a 2.3.4. pontot).

Próbáljuk ki ezt a függvényt is:

```
DerivaltMatrix[{f1[x, y], f2[x, y], f3[x, y]},
        {x, y}, {a, b}] // MatrixForm
```

$$\begin{matrix} f1^{(1,0)}[a,\ b] & f1^{(0,1)}[a,\ b] \\ f2^{(1,0)}[a,\ b] & f2^{(0,1)}[a,\ b] \\ f3^{(1,0)}[a,\ b] & f3^{(0,1)}[a,\ b] \end{matrix}$$

```
DerivaltMatrix[{x^2 - y, y^2 - x, x}, {x, y}, {1, 1}];
MatrixForm[%]
```

$$\begin{matrix} 2 & -1 \\ -1 & 2 \\ 1 & 0 \end{matrix}$$

A `ProgrammingExamples`VectorCalculus`` programcsomagban lévő `JacobianMatrix` függvény is ezt a mátrixot számolja ki. Ilyen elnevezésű, de az előzőtől különböző függvény a `Calculus`VectorAnalysis`` programcsomagban is található.

• Függvények szélsőértékei

Két példán keresztül érzékeltetjük azt, hogy az analízis megfelelő eredményeinek felhasználásával hogyan használhatjuk a *Mathematicá*t bizonyos $\mathbf{R}^n \to \mathbf{R}$ típusú függvények szélsőértékeinek kiszámolásához. Lineáris függvények lineáris feltételek melletti szélsőértékhelyének meghatározására szolgál a lineáris programozás, erről a 3.8.9. pontban lesz szó.

Határozzuk meg először az

$$f(x) := x^3 e^{-x/2} \qquad (x \in \mathbf{R})$$

függvény *lokális* szélsőértékhelyeit.

A lokális szélsőérték létezésére vonatkozó elsőrendű szükséges feltétel alapján kapjuk meg a stacionárius pontokat:

```
f[x_] = x^3 Exp[-x/2];
StacPontok = Solve[f'[x] == 0, x]
{{x -> 0}, {x -> 0}, {x -> 6}}
```

Ezt a módszert azokban az esetekben használhatjuk, amikor a program (szimbolikusan vagy numerikusan) elő tudja állítani a deriváltfüggvény zérushelyeit (lásd a 3.3. szakaszt).

Alkalmazzuk most a másodrendű elégséges feltételt:

```
f''[x] /. StacPontok
{0, 0, -18/E^3}
```

Az $x_0 = 6$ pont az f függvénynek tehát lokális maximumhelye. Mivel

```
f'''[x] /. StacPontok
{6, 6, 6/E^3}
```

ezért az $x_1 = 0$ pontban f-nek nincs lokális szélsőértéke.

Számos esetben a következő függvénnyel is megkaphatjuk az $[a, b]$ intervallumon értelmezett valós értékű folytonos f függvény *abszolút maximum*ának egy közelítő értékét:

```
FvMaximum[f_, a_, b_, n_:200] :=
     Max[N[f[#]]& /@ Table[a + k (b-a)/n, {k, 0, n}]]
```

Figyeljük meg azt, hogy az első argumentumba a függvény nevét kell beírni, a negyedik (opcionális) argumentumba pedig az intervallum osztópontjainak a számát adhatjuk meg:

```
f[x_] = Sqrt[x^4 - x^2 + 1];
FvMaximum[f, -1, 2]
3.60555

Clear[f]
f[x_] = Cos[x] + 1/2 Cos[2x] + 1/3 Cos[3x];
FvMaximum[f, 0, 2Pi, 100]
1.83333
```

Abszolút minimum kiszámolásához a `Max` függvény helyett a `Min` belső függvényt használhatjuk.

Határozzuk meg most az

$$f(x, y, z) := x^3 + y^2 + z^2 + 12xy + 2z \qquad ((x, y, z) \in \mathbf{R}^3)$$

függvény *lokális* szélsőértékhelyeit.

A korábbiakban értelmezett `gradiens` függvényt használjuk a stacionárius pontok kiszámolásához:

```
Clear[f]
f[x_, y_, z_] := x^3 + y^2 + z^2 + 12 x y + 2 z
StacPontok = Solve[
      gradiens[f[x, y, z], {x, y, z}] == 0, {x, y, z}]
{{z -> -1, y -> -144, x -> 24},
  {z -> -1, y -> 0, x -> 0}}
```

A $P_1(24, -144, -1)$ és a $P_2(0, 0, -1)$ pontban lehet az f függvénynek lokális szélsőértékhelye.

A másodrendű elégséges feltétel alkalmazásához elő kell állítanunk a stacionárius pontokban az $f : \mathbf{R}^3 \to \mathbf{R}$ függvény másodrendű deriváltmátrixát (ezt *Hesse-féle mátrix*nak is szokás nevezni). Ezt számolja ki az alábbi eljárás:

```
HesseMatrix[fuggveny_, valtozok_List, pont_List]:=
     DerivaltMatrix[
          gradiens[fuggveny, valtozok], valtozok, pont]
```

A következő függvénnyel pedig négyzetes mátrix sarokaldeterminánsait határozhatjuk meg:

```
SarokAldet[A_] := Table[First[Minors[A, k][[1]]],
                  {k, 1, Length[A]}]
```

A P_1 pontban a másodrendű elégséges feltétel és a Sylvester-kritérium alapján ezt kapjuk:

```
M1 = HesseMatrix[f[x, y, z], {x, y, z}, {24, -144, -1}];
MatrixForm[M1]
```

```
144  12   0
12   2    0
0    0    2
```

```
SarokAldet[M1]
```

```
{144, 144, 288}
```

A $P_1(24, -144, -1)$ pont az f függvénynek tehát lokális minimumhelye.

A $P_2(0, 0, -1)$ pontban:

```
M2 = HesseMatrix[f[x, y, z], {x, y, z}, {0, 0, -1}];
```

```
MatrixForm[M2]
```

```
0   12  0
12  2   0
0   0   2
```

```
SarokAldet[M2]
```

```
{0, -144, -288}
```

A Sylvester-kritériumot tehát nem alkalmazhatjuk. Például a függvény definícióját felhasználva bizonyíthatjuk be azt, hogy ez a pont nem lokális szélsőértékhely.

Numerikus módszerrel keresi $\mathbf{R}^n \to \mathbf{R}$ ($n \in \mathbf{N}$) típusú függvény *egyik* lokális minimumhelyének és lokális minimumának egy közelítő értékét a

`FindMinimum`

függvény. A `FindRoot` eljáráshoz hasonlóan itt is egy vagy több kezdőérték megadása szükséges. Minimumhelyhez közeli „jó” kezdőérték(ek) megválasztásához a *Mathematica* rajzoló eljárásait használhatjuk.

A `FindMinimum` függvény a konjugált gradiens módszert alkalmazza akkor, amikor egy kezdőértéket adunk meg:

```
FindMinimum[-x^3 Exp[-x/2], {x, 1}]
```

```
{-10.754, {x -> 6.}}
```

```
InputForm[%]
```

```
{-10.75400676745861, {x -> 5.999999999977617}}
```

```
f[x_, y_, z_] := x^3 + y^2 + z^2 + 12 x y + 2 z
FindMinimum[f[x, y, z], {x, 20}, {y, -100}, {z, -2}]
```

```
{-6913., {x -> 24.001, y -> -144.012, z -> -0.999346}}
```

A `FindMinimum` függvény a deriváltat nem használó Brent-féle minimalizációs algoritmust alkalmazza (lásd [40]-et) akkor, amikor két kezdőértéket adunk meg:

```
FindMinimum[-x^3*Exp[-x/2], {x, 1, 3}]
```

```
{-10.754, {x -> 6.}}
```

```
FinndMinimum[(x+2y-3)^2-(x-1)(y-1), {x, 0, 1}, {y, 2, 1}]
```

```
                -27
6.48478 10^(-27)  , {x -> 1., y -> 1.}}
```

3.4.5. Integrálszámítás

Ebben a pontban a programnak azokat a lehetőségeit ismertetjük, amelyek $\mathbf{C} \to \mathbf{C}^n$ ($n \in \mathbf{N}$) típusú függvények primitív függvényeinek és határozott integráljának szimbolikus vagy numerikus meghatározásához, valamint többszörös integrálok kiszámolásához nyújthatnak segítséget.

• Primitív függvény keresése

$\mathbf{C} \to \mathbf{C}$ típusú függvény primitív függvényének *szimbolikus* meghatározásához az opció nélküli `Integrate` eljárást használhatjuk. A program a megadott függvény primitív függvényét a beépített matematikai függvényekkel próbálja kifejezni. Ha ez sikerül, akkor az

```
Integrate[f[x], x]
```

utasítás eredménye az f függvény *egyik* primitív függvényének az x pontban vett helyettesítési értéke. Például:

```
Integrate[1/(x^2+x-2), x];
Together[%] /. Log[x_] - Log[y_] -> Log[x/y]
```

$$\frac{\text{Log}\left[\frac{-1 + x}{2 + x}\right]}{3}$$

```
Integrate[Sqrt[1-Sin[x]^2], x]
```

$$\frac{\text{Sqrt}[1 + \text{Cos}[2\ x]]\ \text{Tan}[x]}{\text{Sqrt}[2]}$$

Adott feltételt kielégítő primitív függvényt a matematikában megszokott módon határozhatunk meg. Például így:

```
f[x_] := 3x
a = 3; b = 2;
F[x_] = Integrate[f[x], x] + c;
megoldas = Solve[F[a] == b, c];
c = megoldas[[1, 1, 2]];
F[x]
```

$$-\frac{23}{2} + \frac{3\ x^2}{2}$$

(Figyeljük meg, hogy azonnali értékadással — lásd a 3.1.4. pontot — definiáltuk az F függvényt.)

Primitív függvény meghatározását több negatív jellegű elméleti eredmény is korlátozza. Ismeretes például az, hogy bár minden folytonos függvénynek van primitív függvénye, azonban van olyan elemi függvény, amelynek a primitív függvénye nem elemi függvény. Másrészt: még elemi függvények esetében sincs olyan, véges sok lépést tartalmazó *általános módszer*, amellyel a primitív függvény előállítható lenne. Az analízisben számos, jól körülhatárolt függvényosztályra dolgoztak ki jól használható módszereket.

1969-ben R. Risch [68] publikált számítógépen is megvalósítható és elég nagy függvényosztályra eredményt adó algoritmust. Módszerének alapja az, hogy az adott elemi függvényt úgy bontja fel „egyszerű szerkezetű" függvények összegére, hogy az egyes tagok primitív függvénye már könnyen leolvasható legyen. (Az eljárás hasonló ahhoz, ahogyan racionális törtfüggvényeket parciális törtekre szoktunk bontani.) A Risch-algoritmus képezi az alapját a *Mathematica* primitív függvényt kereső eljárásának. Erről részletesebben a [69] dolgozatban olvashatunk.

A Risch-algoritmus komplex függvénytani eszközöket használ, ezért a begépelt egyváltozós függvényt $\mathbf{C} \to \mathbf{C}$ típusúnak tekinti. Ilyen függvény primitív függvényét különböző alakokban is elő lehet állítani. Például az

$$f(x) := \frac{1}{x^2+1} \qquad (x \in \mathbf{C} \setminus \{-i, i\})$$

függvény egyik primitív függvénye

$$F_1(x) := \frac{1}{2i} \ln \frac{i-x}{i+x} \qquad (x \in \mathbf{C} \setminus \{-i, i\}).$$

Komplex számok felhasználása nélkül (azaz „valós alakban") is megadhatjuk a fenti f függvény primitív függvényeit. Például

$$F_2(x) := \operatorname{arctg} x \qquad (x \in \mathbf{C} \setminus \{-i, i\})$$

is egy ilyen függvény. Az `Integrate` eljárásba beépített algoritmusok azt az elvet követik, hogy ha a beadott függvény nem tartalmaz komplex számot, akkor az eredményt is ezek használata nélkül próbálják megadni:

```
Integrate[1/(x^2+a^2), x]

  ArcTan[x/a]
-------------
      a
```

Racionális törtfüggvények primitív függvényét a program abban az esetben adja meg, ha a nevező zérushelyeit pontosan elő tudja állítani:

```
Integrate[x^4/(x^4+5x^2+4), x]
```

$$x + \frac{8\ \text{ArcTan}\left[\frac{2}{x}\right]}{3} + \frac{\text{ArcTan}[x]}{3}$$

```
Integrate[1/(x^5+3x+1), x]
```

$$\text{Integrate}\left[\frac{1}{1 + 3\,x + x^5}, x\right]$$

Megemlítettük már azt a tényt, hogy vannak olyan elemi függvények, amelyeknek a primitív függvényei nem elemi függvények. Ilyenek például a következő függvények:

$$x \mapsto \frac{\sin x}{x}, \quad x \mapsto e^{-x^2}, \quad x \mapsto \sin x^2.$$

Mivel a matematika különböző fejezeteiben gyakran használják a fenti függvények primitív függvényeit, ezért a *Mathematicá*ba beépítették ezeket a nem elemi függvényeket is. Például:

```
Integrate[Sin[x]/x, x]
SinIntegral[x]

Integrate[Exp[-x^2], x]
```

$$\frac{\text{Sqrt}[\text{Pi}]\ \text{Erf}[x]}{2}$$

```
Integrate[Sin[x^2], x]
```

$$\text{Sqrt}\left[\frac{\text{Pi}}{2}\right]\ \text{FresnelS}\left[\text{Sqrt}\left[\frac{2}{\text{Pi}}\right]\ x\right]$$

A 3.4.1. pont elején felsoroltuk a többi hasonló típusú beépített függvényt is. Elliptikus integrálokat a

```
Calculus`EllipticIntegrate`
```

programcsomag `Integrate` függvényével (ez belső társának a kiterjesztése) számolhatunk ki. Például:

```
Integrate[1/Sqrt[1-m Sin[phi]^2], phi]
EllipticF[phi, m]

N[EllipticF[0.5, 0.2], 25]
0.5040488135239274
```

• Határozott integrál

R → **C** típusú függvény határozott integráljának kiszámolásához több eljárást is használhatunk. Az

```
Integrate[f[x], {x, x1, x2}]
```

utasítás hatására a program az

$$\int_{x_1}^{x_2} f(x)dx = F(x_2) - F(x_1)$$

Newton–Leibniz-formulát alkalmazza. Először szimbolikusan meghatározza az f függvény egy F primitív függvényét, ezután veszi az $F(x_2)-F(x_1)$ különbséget. Ha az f függvény folytonos az $[x_1, x_2]$ intervallumon és a program megtalálja f egy F primitív függvényét, akkor megkapjuk a kért eredményt. Például:

```
Integrate[Sqrt[1-Cos[2 x]], {x, 0, 100 Pi}]
200 Sqrt[2]

Integrate[x^2, {a, b}]
  3     3
 b     a
 -- - --
 3     3
```

Ha a program nem tudja meghatározni a beadott függvény primitív függvényét, akkor eredményként a begépelt sort kapjuk vissza. Ilyenkor az `N` eljárást alkalmazva az integrál egy közelítő értékét kapjuk meg:

```
Integrate[x Tan[x], {x, 0, 1}]
General::intinit:
    Loading integration packages -- please wait.
Integrate[x Tan[x], {x, 0, 1}]

N[%]
0.428088
```

A szóban forgó eljárást improprius integrálok meghatározásánál is használhatjuk:

```
Integrate[1/Sqrt[x], {x, 0, 1}]
2
```

```
Integrate[1/x^(3/2), {x, -1, 1}]
Integrate::idiv: Integral does not converge.
Indeterminate

Integrate[1/x^2, {x, 1, Infinity}]
1
```

Az f függvény $[x_1, x_2]$ intervallumon vett határozott integráljának egy közelítő értékét az

```
NIntegrate[f[x], {x, x1, x2}]
```

utasítás eredménye szolgáltatja. A *Mathematica* ebben az esetben először az f függvény helyettesítési értékeit határozza meg az $[x_1, x_2]$ intervallum bizonyos pontjaiban, majd ezekből a függvényértékekből numerikus módszert alkalmazva adja meg az eredményt.

Ha az f függvény folytonos az $[x_1, x_2]$ intervallumon, akkor minden esetben megkapjuk a határozott integrál egy közelítő értékét. Például:

```
NIntegrate[E^Sqrt[x], {x, 1, 3}]
8.27544
```

Sok esetben az `N[Integrate[]]` és az `NIntegrate` eljárás ugyanazt az eredményt adja. Érdemes azonban összehasonlítani a kiértékeléshez szükséges időtartamot is:

```
Timing[N[Integrate[x Tan[x], {x, 0, 1}]]]
{29.934 Second, 0.428088}

Timing[NIntegrate[x Tan[x], {x, 0, 1}]]
{2.801 Second, 0.428088}
```

Az `NIntegrate` eljárással improprius integrálok és szakadásos függvények határozott integráljának közelítő értékét is meghatározhatjuk:

```
NIntegrate[1/Sqrt[x], {x, 0, 1}]
2.

NIntegrate[Exp[-x^3], {x, 0, Infinity}]
0.89298
```

Az integrandus szingularitását az `NIntegrate` függvény csak az intervallum végpontjaiban vizsgálja. Ha olyan függvényt adunk meg, amelynek az integrálás intervallumának belső pontjában van szakadása, akkor általában figyelmeztető üzenetek után kapunk (esetleg rossz) eredményt. Az

`NIntegrate` függvény második argumentumába a szinguláris helyeket is beírhatjuk:

```
NIntegrate[f[x], {x, x1, szh1, szh2, ..., x2}]
```

A *Mathematica* ebben az esetben különös gonddal vizsgálja az adott függvényt a jelzett pontok környezetében:

```
NIntegrate[1/Sqrt[Abs[x]], {x, -1, 0, 2}]
```

```
4.82843
```

Ugyanezt a módszert alkalmazhatjuk akkor is, ha például szakaszonként különböző formulával értelmezett függvény határozott integrálját szeretnénk kiszámítani:

```
f[x_] := Which[x<0, 0, x<1, x^2, x<2, 2x, x<3, 4, x>=3, 0]
NIntegrate[f[x], {x, -3, 0, 1, 2, 3, 6}]
```

```
7.33333
```

Az `NIntegrate` függvény 9 opciójával (ezekről a program az

```
Options[NIntegrate]
```

utasítás végrehajtása után tájékoztat bennünket) avatkozhatunk be a kiértékelés folyamatába. Megválaszthatjuk például a kiértékelés során alkalmazott értékes számjegyek számát és a felhasznált numerikus módszert is. Módosíthatjuk a numerikus algoritmusok bizonyos paramétereit is. Itt csak a `Method` opció lehetséges értékeit soroljuk fel:

`DoubleExponential`	`Multidimensional`
`GaussKronrod`	`Trapezoidal`

Ennek az opciónak az alapértelmezés szerinti értéke `Automatic`, ami azt jelenti, hogy az `NIntegrate` függvény belső algoritmusa választja ki a kiértékelés során alkalmazott numerikus módszert.

A fentebb bemutatott példák azt is mutatják, hogy az opciók alapértelmezés szerinti értékével számos esetben megfelelő eredményt kapunk. Adott határozott integrál „jó” közelítő értékének kiszámolásához az opciók alkalmas megválasztása sokszor nem egyszerű feladat. Ehhez sok segítséget nyújthat a [40] dolgozat, amelyben az opciók jelentéséről és használatáról olvashatunk.

A *Mathematica* több módszert kínál függvény határozott integráljának közelítő meghatározásához abban az esetben, amikor a függvény helyettesítési értékeit csak bizonyos diszkrét pontokban ismerjük.

A következő pontban ismertetett eljárások valamelyikével például folytonos függvényt illeszthetünk az adott pontokra, majd alkalmazhatjuk az `NIntegrate` függvényt.

A `NumericalMath`ListIntegrate`` programcsomagban meglevő

`ListIntegrate`

függvényt is használhatjuk. A *Mathematica* ebben az esetben interpolációs polinomot illeszt az adott pontrendszerre, és az így kapott függvény határozott integrálját számolja ki:

```
<<NumericalMath`ListIntegrate`
adatok = Table[n^2, {n, 0, 7}]
{0, 1, 4, 9, 16, 25, 36, 49}

ListIntegrate[adatok, 1]
343
---
3
Integrate[x^2, {x, 0, 7}]
343
---
3
```

A `NumericalMath`CauchyPrincipalValue`` programcsomag

`CauchyPrincipalValue`

függvényével improprius integrál *Cauchy-féle főértékének* egy közelítő értékét határozhatjuk meg.

Tekintsük például az

$$f(x) := \frac{1}{x^2 + x} \qquad (x \in \mathbf{R} \setminus \{-1, 0\})$$

függvényt. Mivel a

$$\lim_{\substack{\varepsilon_1 \to 0+0 \\ \varepsilon_2 \to 0+0}} \Big(\int_{-1/2}^{-\varepsilon_1} f(x)dx + \int_{\varepsilon_2}^{1} f(x)dx \Big)$$

határérték nem létezik, ezért a

$$\int_{-1/2}^{1} f(x)dx$$

improprius integrál divergens. Bebizonyítható azonban az is, hogy

$$\lim_{\varepsilon\to 0+0}\Big(\int_{-1/2}^{-\varepsilon} f(x)dx + \int_{\varepsilon}^{1} f(x)dx\Big) = -\ln 2.$$

Ez a szám a fenti improprius integrál Cauchy-féle főértéke. Ennek egy közelítő értékét így kaphatjuk meg:

```
<<NumericalMath`CauchyPrincipalValue`
CauchyPrincipalValue[1/(x^2+x), {x, -1/2, {0}, 1}]
-0.693147

InputForm[%]
-0.6931471805596515
```

A Cauchy-féle főérték pontos értékét is meghatározhatjuk. Például így:

```
Clear[f]
f[x_] := 1/(x^2+x)
jobb = Integrate[f[x], {x, c^2, 1}] //.
          Log[x_] - Log[y_] -> Log[x/y];
resz = Integrate[f[x], {x, -1/2, -c^2}] /.
          Log[x_ y_] -> Log[x] + Log[y];
bal = % /. Log[x_] - Log[y_] -> Log[x/y];
bal + jobb /. Log[x_] + Log[y_] -> Log[x y];
Limit[%, c -> 0]
    Integrate::gener: Unable to check convergence.
Log[1/2]

N[%] // InputForm
-0.6931471805599453
```

• Többszörös integrálok

Az előzőekben ismertetett eljárásokkal $\mathbf{R}^n \to \mathbf{C}$ $(n \in \mathbf{N} \setminus \{1\})$ típusú függvény adott tartományon vett határozott integrálját szimbolikusan és numerikusan is meghatározhatjuk.

A tartomány lehet például $\mathbf{R}^n$-beli *tégla*:

```
Integrate[x y Sin[x^2 + y^2],
         {x, 0, Sqrt[Pi/2]}, {y, 0, Sqrt[Pi/2]}]
1/2
```

Számítsuk most ki a következő határozott integrált:

$$\int_0^2 \int_0^{\sqrt{2y-x^2}} \sqrt{4y-x^2}dxdy = \int_0^2 \left(\int_0^{\sqrt{2y-x^2}} \sqrt{4y-x^2}dx \right) dy.$$

```
Integrate[Sqrt[4 y - x^2],
        {y, 0, 2}, {x, 0, Sqrt[2y-y^2]}]
```

$$\frac{4}{3} + \frac{\text{Pi}}{2}$$

*Normáltartomány*on vett határozott integrál kiszámolásánál vigyázzunk az integrálás sorrendjének a kijelölésére. Figyeljük meg, hogy a *Mathematica* először az utolsóként megjelölt változó szerint végzi el az integrálást.

*Improprius integrálok*at is meghatározhatunk:

```
Integrate[1/(x^2+y^2)^2,
        {x, 1, Infinity}, {y, -Infinity, Infinity}]
```

$$\frac{\text{Pi}}{4}$$

3.4.6. Függvényközelítések

Az approximációelmélet foglalkozik megadott függvény egyszerű szerkezetű függvényekkel való megközelítésével. A leggyakrabban használt egyszerű szerkezetű függvények: az algebrai és a trigonometrikus polinomok, a racionális függvények és a spline-függvények számítógép segítségével is jól kezelhetőek. A matematikai programcsomagok ezért az ilyen típusú problémák tanulmányozásához is sok segítséget adhatnak.

A legegyszerűbb esetben az approximálandó függvény az $[a,b] \subset \mathbf{R}$ intervallumon folytonos, valós értékű függvények $C([a,b])$ szimbólummal jelölt halmazához tartozik. Függvények távolságát már ebben a halmazban is többféle módon értelmezhetjük. Egyszerűen bebizonyítható például az, hogy a

$$\varrho_\infty(f,g) := \max_{x\in[a,b]} |f(x)-g(x)| \qquad (f,g \in C([a,b])) \tag{1}$$

és a

$$\varrho_2(f,g) := \int_a^b |f(x)-g(x)|^2 dx \qquad (f,g \in C([a,b])) \tag{2}$$

függvény mindegyike metrika a $C([a,b])$ halmazon.

Jelöljük G-vel az egyszerű szerkezetű függvények valamely adott halmazát. Ez lehet például legfeljebb n-edfokú (algebrai) polinomok halmaza.

Tekintsünk egy ϱ metrikát is a $C([a,b])$ halmazon. Az approximációelmélet alapvető problémái az imént jelzett (speciális) esetben a következők:

- Adott $f \in C([a,b])$ függvényhez létezik-e olyan G-beli $\varphi(f)$ elem, amelyre
$$\varrho(f,\varphi(f)) = \inf\{\varrho(f,g) \ : \ g \in G\} =: E_G(f).$$
- Ha f-nek létezik a fenti tulajdonságú *legjobb megközelítése*, akkor az vajon egyértelműen meg van-e határozva.
- Létezés és egyértelműség esetén hogyan lehet meghatározni vagy jellemezni az f függvényt legjobban megközelítő $\varphi(f)$ elemet.
- Hogyan lehet az $E_G(f)$ számra jól használható alsó és felső becslést megadni.

Alkalmas típusú G halmazok és különböző metrikák megválasztása esetén bizonyítható az, hogy minden $f \in C([a,b])$ függvényhez egyértelműen létezik a fenti tulajdonságú $\varphi(f)$ függvény. Ennek explicit előállítására nincs általános módszer. Az approximációelméletben számos olyan módszert dolgoztak ki, amely segítségével adott függvényt jól közelítő G-beli elem explicit módon előállítható.

Ebben a pontban a *Mathematicá*nak azokat az eljárásait soroljuk fel, amelyek segítségével adott függvényhez közeli, jól kezelhető függvényeket (algebrai és trigonometrikus polinomokat, racionális függvényeket, illetve spline-függvényeket) *konstruálhatunk*.

Az approximációelmélet megfelelő tételeinek és a *Mathematica* egyéb eljárásainak felhasználásával vizsgálhatjuk meg azt, hogy a kapott egyszerű szerkezetű függvények milyen közel vannak a megadott függvényhez. Ezekkel a kérdésekkel a továbbiakban nem foglalkozunk.

• Polinomapproximáció

Elegendően sokszor differenciálható $\mathbf{R} \to \mathbf{R}$ típusú f függvény a pont körüli n-edrendű *Taylor-polinom*ját a

```
Normal[Series[f[x], {x, a, n}]
```

utasítás eredménye adja meg. Például:

```
Normal[Series[x^x, {x, 1, 3}]]
```

$$(-1 + x)^2 + \frac{(-1 + x)^3}{2} + x$$

A függvénynek és különböző rendű Taylor-polinomjainak ábrázolásával gyorsan kaphatunk információt a közelítés pontosságáról. Az $f(x) := \sin x$ ($x \in \mathbf{R}$) függvény $a := 0$ pont körüli harmad- és hetedrendű Taylor-polinomjait például így szemléltethetjük:

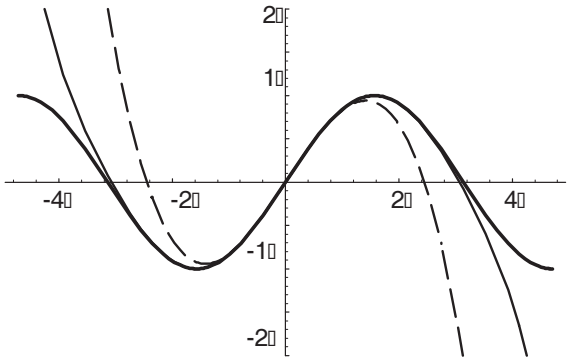
```
f[x_] = Sin[x];
x0 = -3/2 Pi; x1 = 3/2 Pi; a = 0;
t7 = Normal[Series[f[x], {x, a, 7}]]
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

```
t3 = t7[[{1, 2}]]
```

$$x - \frac{x^3}{6}$$

```
Plot[{f[x], t3, t7}, {x, x0, x1}, PlotRange -> {-2, 2},
    PlotStyle -> {{Thickness[0.006]},
                  {Dashing[{0.05, 0.02}]}, {}}]
```



Szintén a `Series` eljárással kaphatjuk meg $\mathbf{C} \to \mathbf{C}$ típusú analitikus függvény *Laurent-sor*ának szeleteit:

```
Series[1/Log[1+z], {z, 0, 4}]
```

$$\frac{1}{z} + \frac{1}{2} - \frac{z}{12} + \frac{z^2}{24} - \frac{19\,z^3}{720} + \frac{3\,z^4}{160} + O[z]^5$$

```
Series[Sin[1/z], {z, 0, 2}]
```

$$\text{Series}[\text{Sin}[\frac{1}{z}], \{z, 0, 2\}]$$

```
Series[Exp[1/z], {z, Infinity, 3}]
```

$$1 + \frac{1}{z} + \frac{1}{2\,z^2} + \frac{1}{6\,z^3} + O\left[\frac{1}{z}\right]^4$$

*Hatványsorok*kal kapcsolatos vizsgálatoknál a

```
DiscreteMath`RSolve`
```

programcsomag alábbi függvényei jelenthetnek segítséget:

`ExponentialGeneratingFunction`	`PowerSum`
`ExponentialPowerSum`	`SeriesTerm`
`GeneratingFunction`	

Tekintsük például az

$$f(x) := \frac{x+1}{(x-1)^2} \qquad (x \in \mathbf{R} \setminus \{1\})$$

függvényt. A 0 pont körüli Taylor-sorának együtthatóit a `SeriesTerm` eljárás adja meg:

```
SeriesTerm[(1+x)/(1-x)^2, {x, 0, n}];
Simplify[%]
1 + 2 n
```

Mivel

```
Limit[(2n+1)^(1/n), n -> Infinity]
1
```

ezért a Cauchy–Hadamard-tétel alapján

$$f(x) = \sum_{n=0}^{\infty} (2n+1)x^n \qquad (x \in (-1,1)).$$

A `PowerSum` eljárással számos hatványsor összegfüggvényét állíthatjuk elő. Például:

```
PowerSum[2n+1, {x, n, 0}];
Simplify[%]
   1 + x
-----------
          2
(-1 + x)
```

A `GeneratingFunction` külső függvénnyel pedig olyan hatványsor összegfüggvényét kaphatjuk meg, amelynek az együtthatói rekurzív összefüg-

géssel vannak megadva. Például:

```
GeneratingFunction[{(n+1) a[n+1]==(n+3) a[n], a[0]==1},
       a[n], n, x]
```

$$\{\{(1 - x)^{-3}\}\}$$

*Interpolációs polinomok*at az

InterpolatingPolynomial

belső függvénnyel határozhatunk meg. Lagrange-féle interpolációs polinomot például így:

```
InterpolatingPolynomial[
      {{-1, 1/2}, {1/2, 2}, {2, -1}, {3, -2}}, x];
Simplify[%]
```

$$\frac{94 - x - 61\,x^2 + 14\,x^3}{40}$$

Ellenőrizzük a kapott eredményt:

```
p1[x_] = % ;
{p1[-1], p1[1/2], p1[2], p1[3]}
```

$$\{\frac{1}{2}, 2, -1, -2\}$$

Hermite-féle interpolációs polinomokat így számolhatunk ki:

```
InterpolatingPolynomial[{{-2, -25},
      {1, {2, 3}}, {4, {1019, 1278, 1280}}}, x];
Simplify[%]
```

$$3 - 2\,x + x^5$$

Ezt az eredményt is ellenőrizzük:

```
p2[x_] = %;
p2[-2]
```

```
-25
```

```
{p2[1], p2'[1]}
```

```
{2, 3}
```

```
{p2[4], p2'[4], p2''[4]}
```

```
{1019, 1278, 1280}
```

A `NumericalMath`Approximations`` programcsomagban található `MiniMaxApproximation` eljárás a Remez-algoritmust alkalmazza adott folytonos függvényt az (1) metrikában legjobban megközelítő algebrai polinom numerikus előállítására. Ismeretes, hogy minden $f \in C([a,b])$ függvényhez egyértelműen létezik olyan $P \in \mathcal{P}_n$ polinom ($\mathcal{P}_n$ jelöli a legfeljebb n-edfokú algebrai polinomok halmazát), amelyre

$$\varrho_\infty(f,P) = \inf\{\varrho(f,p) \ : \ p \in \mathcal{P}_n\}.$$

Ennek a P polinomnak egy közelítését kaphatjuk meg a

```
MiniMaxApproximation[f[x], {x, {a, b}, n, 0}]
```

utasítással. Például:

```
<<NumericalMath`Approximations`
MiniMaxApproximation[1/(1+x), {x, {0, 2}, 1, 0}]

{{0, 1., 2.}, {0.8571428571428571 -
   0.2857142857142857 x, 0.14285714285714285}}
```

Figyeljük meg azt, hogy az eredmény két listából álló lista. Az első lista tartalmazza azokat a pontokat, amelyekben a függvénynek és a kapott polinomnak az eltérése maximális. A második lista első eleme a keresett polinom, a második pedig a függvénynek és a közelítő polinomnak a ϱ_∞ távolsága.

Ha az approximáló polinom fokszámát növeljük, akkor persze jobb közelítést kapunk:

```
MiniMaxApproximation[1/(1+x), {x, {0, 2}, 10, 0}];
Last[%[[2]]]

1.0223673529609311 10^-6
```

A *Mathematica*nak ez az eljárása adott $f \in C([a,b])$ függvényhez olyan $Q \in \mathcal{P}_n$ polinomot próbál meghatározni, amelyre

$$\max_{x\in[a,b]} \left|1 - \frac{Q(x)}{f(x)}\right| = \min_{p\in\mathcal{P}_n} \max_{x\in[a,b]} \left|1 - \frac{p(x)}{f(x)}\right|,$$

ezért azokban az esetekben, amikor az adott intervallumon az f függvény nulla értéket is felvesz, az eljárás *közvetlenül* nem használható. A [3] referenciakönyvben van példa arra, hogy az ilyen függvényeket hogyan lehet kezelni. Szükség esetén innen kaphatunk segítséget az eljárás számos opciójának használatához is.

• Racionális approximáció

*Padé-féle közelítések*hez a `Calculus`Pade`` programcsomagot használhatjuk. A

```
Pade[f[x], {x, a, m, k}]
```

utasítás eredménye olyan $r = p/q$ racionális függvény, amelynek a számlálója m-edfokú, a nevezője pedik k-adfokú polinom, és r kielégíti a következő feltételeket:

$$f^{(i)}(a) = r^{(i)}(a) \qquad (i = 0, 1, 2, \cdots, m + k).$$

A $k = 0$ esetben az f függvény a pont körüli m-edfokú Taylor-polinomját kapjuk:

```
<<Calculus`Pade`
Pade[Exp[x], {x, 0, 3, 3}];
pd1 = Simplify[%]
```

$$\frac{120 + 60\,x + 12\,x^2 + x^3}{120 - 60\,x + 12\,x^2 - x^3}$$

A Taylor-polinomokhoz hasonlóan ezek a függvények is a megadott pontnak csak valamely kis környezetében adnak jó közelítést. Az approximáció pontosságát tetszőleges *intervallumon* is növelhetjük, ha az

```
EconomizedRationalApproximation
```

függvényt (ez szintén a fenti programcsomagban található) használjuk:

```
EconomizedRationalApproximation[Exp[x],
    {x, {-1, 1}, 3, 3}];
pd2 = Simplify[%]
```

$$\frac{933217 + 466608\,x + 93120\,x^2 + 7680\,x^3}{933217 - 466608\,x + 93120\,x^2 - 7680\,x^3}$$

A fenti közelítések pontosságát szemléltethetjük a következő utasítások segítségével:

```
abra1 = Plot[pd1 - Exp[x], {x, -1, 1},
    Ticks -> {{-1, 1}, {2. 10^(-8)}},
    DisplayFunction -> Identity];
```

```
abra2 = Plot[pd2 - Exp[x], {x, -1, 1},
    Ticks -> {{-1, 1}, {4. 10^(-7)}},
    DisplayFunction -> Identity];
Show[GraphicsArray[{abra1, abra2}],
    DisplayFunction -> $DisplayFunction]
```

Felhasználhatjuk még a `NumericalMath`Approximations`` programcsomag alábbbi eljárásait is:

```
GeneralRationalInterpolation
GeneralMiniMaxApproximation
MiniMaxApproximation
RationalInterpolation
```

Interpolációs feltételeknek eleget tevő racionális függvényt így határozhatunk meg:

```
RationalInterpolation[Cos[x], {x, 2, 2},
     {-Pi/2, -Pi/4, 0, Pi/2, Pi/4}];
N[%, 6]
```

$$\frac{1. - 7.74777\ 10^{-18}\ x - 0.405285\ x^2}{1. + 4.16334\ 10^{-17}\ x + 0.0983386\ x^2}$$

Az előző alpontban ismertetett `MiniMaxApproximation` függvényt racionális függvényekre is használhatjuk:

```
MiniMaxApproximation[Cos[x] + 4, {x, {0, Pi/2}, 2, 2}];
N[%[[2, 1]], 6]
```

$$\frac{5.00022 - 0.459931\ x + 0.162379\ x^2}{1. - 0.0906869\ x + 0.126493\ x^2}$$

• Spline-approximáció

Adott síkbeli pontokon átmenő *interpolációs spline-függvény*t az

```
Interpolation
```

belső függvénnyel konstruálhatunk. Az illesztésnél alkalmazott polinomok fokszámát az `InterpolationOrder` opcióval adhatjuk meg. Például:

```
adatok = Table[{x, Cos[x]}, {x, 0, 6}];
```

```
Interpolation[adatok, InterpolationOrder -> 1]
Plot[%[x], {x, 0, 6}]
```

Ismeretes, hogy egynél nagyobb fokszámú interpolációs spline-függvények egyértelmű meghatározásához az adott pontokon kívül további mellékfeltételek megadása is szükséges. A felhasználónak nincs lehetősége az általa alkalmazni kívánt mellékfeltételek *közvetlen* megadására.

A `Graphics`Spline`` programcsomag

Spline

valamint a `NumericalMath`SplineFit`` programcsomag

SplineFit

függvényével adott síkbeli pontokhoz konstruálhatunk egyrészt harmadfokú természetes, másrészt Bezier-féle spline*görbék*et. A két eljárás lehetőségei megegyeznek, különbség csupán a görbék megjelenítésének módjában van. Például:

```
pontok = {{0,0}, {1,2}, {-1,3}, {0,1}, {3,0}};

<<Graphics`Spline`
Show[Graphics[{PointSize[0.02], Point /@ pontok}],
     Graphics[Spline[pontok, Cubic]]]

<<NumericalMath`SplineFit`
SplineFuggveny = SplineFit[pontok, Cubic]
ParametricPlot[SplineFuggveny[u], {u,0,4},
     PlotRange -> All, Compiled -> False]
```

• Trigonometrikus sorok

Az **R** halmazon periodikus, komplex értékeket felvevő függvény trigonometrikus *Fourier-sor*ának tanulmányozásánál az

`Integrate` `NIntegrate`

belső függvényeken kívül a

```
Calculus`FourierTransform`
```

programcsomag alábbi eljárásait is használhatjuk:

```
FourierCosSeriesCoefficient
FourierCosTransform
FourierExpSeries
FourierExpSeriesCoefficient
FourierSample
FourierSinSeriesCoefficient
FourierSinTransform
FourierTransform
FourierTrigSeries
InverseFourierCosTransform
InverseFourierSinTransform
InverseFourierTransform
```

A programcsomagban a fenti eljárások (amelyek a pontos eredményt próbálják meghatározni) numerikus társát is megtalálhatjuk. A felhasználható függvények nevét az

```
?NF*
```

utasítás végrehajtása után ismerhetjük meg.

Tekintsük most azt az $f : \mathbf{R} \to \mathbf{R}$ 2π szerint periodikus függvényt, amelyik a $(-\pi, 0]$ intervallumon nullával, a $(0, \pi]$ intervallumon pedig π-vel egyenlő. Meghatározzuk és ábrázoljuk az f függvény Fourier-sorának néhány részletösszegét.

A számolásokat pontosan is elvégezhetjük, ha a függvényt így értelmezzük:

```
<<Calculus`DiracDelta`
f[x_] := Pi*UnitStep[x]
```

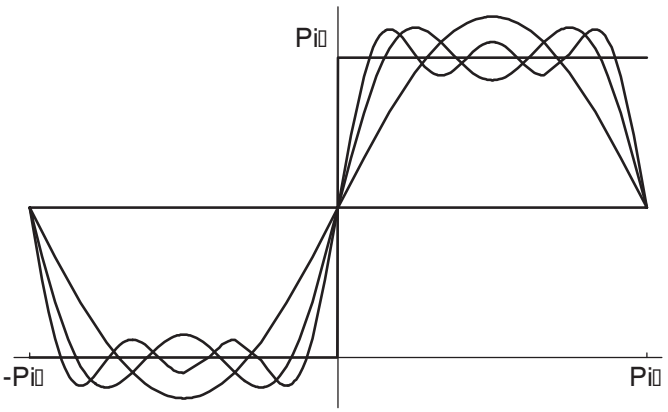
Most kiszámoljuk az f függvény Fourier-sorának egyik részletösszegét:

```
<<Calculus`FourierTransform`
FourierTrigSeries[f[x], {x, -Pi, Pi}, 5]
```

$$\frac{\texttt{Pi}}{2} + 2\,\texttt{Sin[x]} + \frac{2\,\texttt{Sin[3 x]}}{3} + \frac{2\,\texttt{Sin[5 x]}}{5}$$

A függvényt és Fourier-sorának első néhány részletösszegét így ábrázolhatjuk:

```
szeletek = Table[Sum[%[[k]], {k, 1, n}], {n, 1, 4}];
Plot[Evaluate[Flatten[{f[x], szeletek}]],
     {x, -Pi, Pi},
     Ticks -> {{-Pi, Pi}, {Pi}}]
```



A *Fourier-transzformált*at az irodalomban különböző módon szokás értelmezni. A

`FourierTransform`

külső függvény az $f : \mathbf{R} \to \mathbf{R}$ függvényhez az

$$\mathcal{F}(x) := A \int_{-\infty}^{+\infty} f(t) e^{Bixt} dt \qquad (x \in \mathbf{R})$$

Fourier-transzformáltat számolja ki, ahol az A és a B szám alapértelmezés szerinti értéke 1. Ezeket az állandókat vagy a

`$FourierOverallConstant` és a `$FourierFrequencyConstant`

rendszerváltozóval, vagy pedig a

`FourierOverallConstant` és a `FourierFrequencyConstant`

opcióval változtathatjuk meg. Például:

```
$FourierOverallConstant = 1/Sqrt[2 Pi];
$FourierFrequencyConstant = -1;
```

```
FourierTransform[Exp[-a^2 t^2], t, x]
```

$$\frac{1}{\text{Sqrt[2] a E}^{x^2/(4\,a^2)}}$$

```
InverseFourierTransform[%, x, t]
```

$$\text{E}^{-(a^2\ t^2)}$$

Megemlítjük még azt, hogy a

Fourier	InverseFourier

belső függvények a *diszkrét Fourier-transzformált*at határozzák meg.

• Négyzetes közelítések

Legyen $(x_i, y_i) \in \mathbf{R}^2$ $(i = 1, 2, \cdots, n;\ n \in \mathbf{N})$ adott pontrendszer és tegyük fel, hogy g_j $(j = 1, 2, \cdots, m;\ m \in \mathbf{N})$ legalább az x_i $(i = 1, 2, \cdots, n)$ alappontokban értelmezett $\mathbf{R} \to \mathbf{R}$ típusú függvény. A

```
Fit[{{x1, y1},...,{xn, yn}}, {g1[x],...,gm[x]}, x]
```

utasítás hatására a *Mathematica a legkisebb négyzetek módszeré*vel numerikusan határoz meg olyan

$$G(x) := \sum_{j=1}^{m} c_j g_j(x)$$

függvényt, amelyre a következő egyenlőség teljesül:

$$\sum_{i=1}^{n}(y_i - G(x_i))^2 = \inf\{\sum_{i=1}^{n}(y_i - \sum_{j=1}^{n} a_j g(x_j))^2\ :\ a_j \in \mathbf{R};\ j = 1, \cdots, m\}.$$

A lehetőségeket illusztrálja a következő utasítássorozat:

```
alappontok = Table[{x^2, x^(1/4) + x - 2}, {x, 0, 4}]
abra1 = ListPlot[alappontok,
      PlotStyle -> PointSize[0.01]]
Fit[alappontok, {1, x , x^2}, x]
abra2 = Plot[%, {x, 0, 16}]
Show[abra1, abra2]
```

Ha $g_j(x) := x^{j-1}$ $(j = 1, 2, \cdots, m)$, akkor a fenti illesztési feladat egy olyan lineáris egyenletrendszer megoldásához vezet, amelynek az együtthatómátrixa Hilbert-mátrix, és ez klasszikus példája a gyengén meghatározott mátrixoknak. A feladat numerikus megoldása során viszonylag kicsi m esetén is a kerekítésekből adódó hiba mértéktelenül megnövekszik. Ismeretes, hogy ez a numerikus instabilitás úgy javítható, hogy a g_j alapfüggvények helyett a megadott diszkrét pontrendszerre ortogonális polinomokat veszünk alapfüggvényeknek. Ezt a módszert alkalmazza a

```
NumericalMath`PolynomialFit`
```

programcsomag `PolynomialFit` függvénye is. Például:

```
<<NumericalMath`PolynomialFit`
adatok = Table[{x, Random[Real, {-1, 1}] + x^2},
               {x, -2., 2., 0.05}]
p = PolynomialFit[adatok, 2]
Plot[p[x], {x, -2, 2}]
Expand[p[x]]
```

Megemlítjük még azt is, hogy a c_j paraméterekben nemlineáris illesztési feladatok megoldásához a

```
Statistics`NonlinearFit`
```

programcsomag `NonlinearFit` függvénye adhat segítséget. Alkalmazására a 3.11. szakaszban mutatunk példát.

3.4.7. Vektoranalízis

A `Calculus`VectorAnalysis`` programcsomagban meglevő függvények a vektoranalízis alapvető fogalmainak használatakor nyújthatnak segítséget. Olvassuk be ezt a programcsomagot:

```
<<Calculus`VectorAnalysis`
```

• Koordináta-rendszerek

Számos problémánál a derékszögű koordináták helyett jóval előnyösebb más koordinátákat alkalmazni. A leggyakrabban használt

derékszögű	(`Cartesian[x, y, z]`),
henger	(`Spherical[r, theta, phi]`) és
gömbi polár	(`Cylindrical[r, phi, z]`)

koordináta-rendszerek mellett a következőkkel is dolgozhatunk:

```
Bipolar                    OblateSpheroidal
Bispherical                ParabolicCylindrical
ConfocalEllipsoidal        Paraboloidal
ConfocalParaboloidal       ProlateSpheroidal
Conical                    Toroidal
EllipticCylindrical
```

(Ezek pontos definíciója [3]-ban szükség esetén megtalálható.)

A koordináta-rendszerek jellemzésére, illetve megválasztásukra az alábbi függvények használhatók:

```
Coordinates[ ]
CoordinatesFromCartesian
CoordinateRanges[ ]
CoordinateSystem
CoordinatesToCartesian
Parameters[ ]
ParameterRanges[ ]
SetCoordinates
```

Az alapértelmezésben használt koordináta-rendszer nevéről, a koordináták elnevezéséről és lehetséges értékeikről így tájékozódhatunk:

```
CoordinateSystem
Cartesian

{Coordinates[], CoordinateRanges[]}
{{x, y, z}, {-Infinity < x < Infinity,
      -Infinity < y < Infinity, -Infinity < z < Infinity}}
```

Az alapértelmezésben használt koordináta-rendszert is módosíthatjuk:

```
SetCoordinates[Spherical]
Spherical[r, theta, phi]

CoordinateRanges[]
{0 <= r < Infinity, 0 <= theta <= Pi, -Pi < phi <= Pi}
```

További lehetőségeket illusztrálnak az alábbi példák:

```
CoordinatesToCartesian[{r, theta, z}, Cylindrical]
{r Cos[theta], r Sin[theta], z}

CoordinatesFromCartesian[{x, y, z}, Cylindrical]

                    2    2
{Sqrt[x  + y ], ArcTan[x, y], z}
```

• Műveletek vektorokkal

Vektorok skaláris, vektoriális és vegyesszorzatát különböző koordináta-rendszerekben is meghatározhatjuk:

```
DotProduct[{u1, u2, u3}, {v1, v2, v3}, Cylindrical]
u3 v3 + u1 v1 Cos[u2] Cos[v2] + u1 v1 Sin[u2] Sin[v2]

CrossProduct[{u1, u2, u3}, {v1, v2, v3}, Cartesian]
{-(u3 v2) + u2 v3, u3 v1 - u1 v3, -(u2 v1) + u1 v2}

ScalarTripleProduct[
{u1, u2, u3}, {v1, v2, v3}, {w1, w2, w3}, Cartesian]
-u3 v2 w1 + u2 v3 w1 + u3 v1 w2 - u1 v3 w2 -
u2 v1 w3 + u1 v2 w3
```

• Skalármezők és vektormezők

Skalármező (azaz $\mathbf{R}^3 \to \mathbf{R}$ típusú függvény) gradiens vektorát a `Grad` eljárással számolhatjuk ki:

```
Grad[x Exp[y] + x^2 z^2 - y^3 x, Cartesian]
    y    3        2  y        2    2
{E  - y  + 2 x z , E  x - 3 x y , 2 x  z}

Grad[U[r, theta, z], Cylindrical]

                                  (0,1,0)
{U^(1,0,0)[r, theta, z], U^(0,1,0)[r, theta, z]/r,
U^(0,0,1)[r, theta, z]}
```

Vektormező (azaz $\mathbf{R}^3 \to \mathbf{R}^3$ típusú függvény) ábrázolásáról a 3.5. szakaszban fogunk szólni. Divergenciájukat a `Div`, rotációjukat pedig a `Curl` függvénnyel kapjuk meg:

```
v[x_, y_, z_] := {x/y, y/z, x z}
Div[v[x, y, z], Cartesian]
```

$$x + \frac{1}{y} + \frac{1}{z}$$

```
% /. {x -> 1, y -> 2, z -> 3}
```

$$\frac{11}{6}$$

```
Curl[v[x, y, z], Cartesian]
```

$$\{\frac{y}{z^2}, -z, \frac{x}{y^2}\}$$

A `JacobianMatrix[pt, krsz]` megadja a `krsz` koordináta-rendszerből a Descartes-félére való áttérés transzformációjának deriváltmátrixát a `pt` pontban. A `JacobianDeterminant[pt, krsz]` utasítással pedig ennek a mátrixnak a determinánsát kaphatjuk meg.

A Laplace- és a biharmonikus operátorral is dolgozhatunk:

```
Laplacian[u[r, theta, z], Cylindrical]
```

$$r\, u^{(0,0,2)}[r, \text{theta}, z] + \frac{u^{(0,2,0)}[r, \text{theta}, z]}{r} + u^{(1,0,0)}[r, \text{theta}, z]\, r\, u^{(2,0,0)}[r, \text{theta}, z]) / r$$

```
Biharmonic[u[x, y, z]]
```

$$u^{(0,0,4)}[x, y, z] + 2\, u^{(0,2,2)}[x, y, z] + u^{(0,4,0)}[x, y, z] + 2\, u^{(2,0,2)}[x, y, z] + 2\, u^{(2,2,0)}[x, y, z] + u^{(4,0,0)}[x, y, z]$$

Azonosságokat is igazolhatunk:

```
Div[Grad[u[x, y, z], Cartesian], Cartesian] ==
    Laplacian[u[x, y, z], Cartesian]
True

Curl[Grad[u[x, y, z], Cartesian], Cartesian]
{0, 0, 0}
```

3.5. Differenciálegyenletek

Az analízis egy bonyolultabb, önmagáért is vizsgált és az alkalmazásokhoz is közelálló fejezetéhez érkeztünk. Meg fogjuk mutatni, hogy a leggyakrabban előkerülő problémák megoldásához hogyan lehet hozzáfogni, ha a *Mathematicát* is segítségül hívjuk.

Több könyvet is szenteltek már a differenciálegyenletek és a *Mathematica* kapcsolatának. Ezek közül Abell és Braselton [1] könyve arra törekszik, hogy egyszerű speciális alakú differenciálegyenletek megoldására használt módszereket reprodukáljon, a *Mathematicá*val végeztetve a részletszámításokat.

Vvedensky [86] könyve közönséges és parciális differenciálegyenletekről szól, első ránézésre a fentinél sokkal jobbnak látszik, mivel alaposan kihasználja a *Mathematica* lehetőségeit és sok feladatot tartalmaz.

Megemlítjük még Coombs és munkatársainak új könyvét [16].

Ebben a szakaszban túlnyomórészt *közönséges* differenciálegyenletekről lesz szó; ha a közönséges jelzőt elhagyjuk, ilyenekre gondolunk. Parciális differenciálegyenletek közül az elsőrendűek — egy újabb programcsomag felhasználásával — hasonlóan kezelhetők, mint ahogyan „papíron-ceruzával". Másodrendű (lineáris) egyenletekre vonatkozó feladatok közül csak igen egyszerűek oldhatók meg Laplace-transzformáció, hatványsoros vagy Fourier-módszer segítségével. Végül — ahogyan más szerzők is megteszik a differenciálegyenletek tárgyalásánál — a variációszámítás legegyszerűbb feladatairól is szót ejtünk.

Ezt a bevezető részt néhány általános tanáccsal zárjuk. Differenciálegyenletek megoldásánál igen gyakran előfordul, hogy figyelmeztető vagy hibaüzeneteket kapunk. Ezután még előfordulhat, hogy végül megkapjuk a helyes eredményt. Amiatt sem érdemes türelmetlenkednünk, hogy az eredmények esetleg igen soká születnek meg, ugyanis egy differenciálegyenlet megoldása esetenként néhányszor tíz percig vagy még tovább is eltarthat, szemben az egyszerű feladatoknál megszokott másodpercekkel.

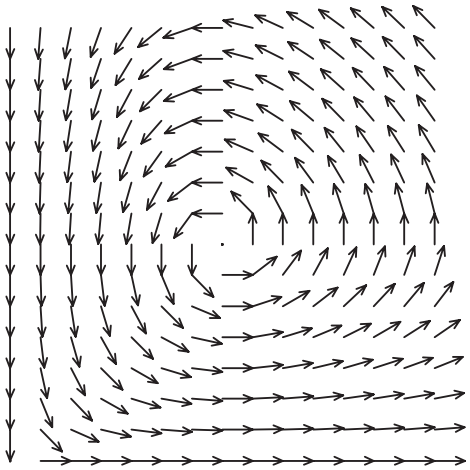
3.5.1. Iránymező két és három dimenzióban

Kezdjük azzal, amivel a tankönyvek és példatárak többsége is kezdi: ábrázoljuk néhány differenciálegyenlet iránymezőjét. Kétváltozós egyenletekhez a `Graphics`PlotField`` programcsomag függvényeit használhatjuk:

```
ListPlotVectorField       PlotPolyaField
PlotGradientField         PlotVectorField
PlotHamiltonianField
```

Első példaként nézzük meg a Lotka–Volterra-egyenletet:

```
<<Graphics`PlotField`
PlotVectorField[{x - x y, x y - y}, {x, 0, 2}, {y, 0, 2}]
```



Ha adott egy valós értékű úgynevezett *potenciálfüggvény*, amelynek gradiense a differenciálegyenlet jobb oldala, akkor *gradiens-rendszer*rel van dolgunk. Az ilyenek iránymezőjének megrajzolásához elegendő a potenciálfüggvényt megadni:

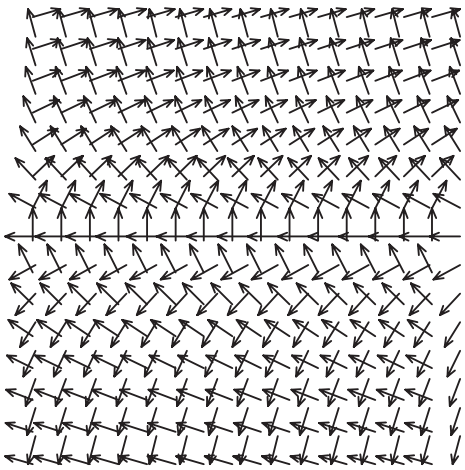
```
PlotGradientField[-x + y^2, {x, 0, 4}, {y, -2, 2},
   AspectRatio->Automatic]
```

Ugyanilyen egyszerűen lehet megrajzolni olyan egyenletek iránymezőjét is, amelyek jobb oldala egy Hamilton-féle függvényből származik deriválás útján:

```
PlotHamiltonianField[-x + y^2, {x, 0, 4}, {y, -2, 2},
   ScaleFunction->(3&), AspectRatio->Automatic]
```

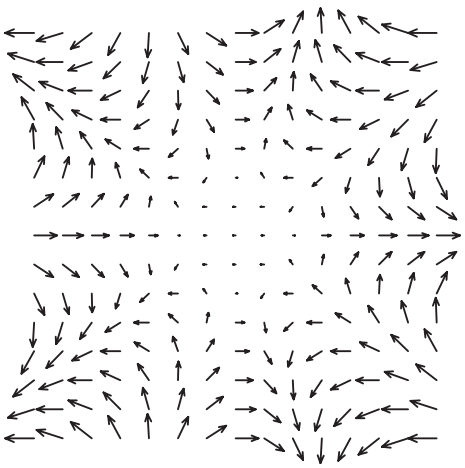
Az ugyanazon függvényből származó gradiens-rendszer és Hamilton-rendszer trajektóriái ortogonálisak egymásra, ez az iránymezőkön is jól látszik:

```
Show[%, %%]
```



Megadhatunk egy komplex függvényt is ahhoz, hogy ezt Pólya-féle reprezentációban használjuk iránymező készítésére. (Cauchy, Riemann és Erugin éppígy lehetne a névadó.) A megadott függvény valós része a jobb oldal első, képzetes része pedig második koordinátafüggvénye. Legyen $a = 1.5, 2.0, 4.0$, és ezekkel az értékekkel számítsuk ki az alábbi függvényt:

```
ppf[a_] := PlotPolyaField[(x + I y)^4 - 1,
   {x, -a, a}, {y, -a, a}]
ppf[4.0]
```



Mivel a $z \mapsto z^4 - 1$ függvénynek a negyedik egységgyökök a gyökei, ezért ezen pontok közelében kicsi a vektorok hossza.

A `ListPlotVectorField` függvény alkalmazható akkor, ha a jobb oldalt egy táblázat (például mérési adatoké) adja meg; vagy akkor, ha meghatározott pontokban adott nagyságú vektorokat akarunk csak kirajzolni. Ez utóbbira mutatunk példát:

```
lpvf[s_]:=ListPlotVectorField[Table[{{Sin[u], Cos[u]},
   {u^2 Sin[2u], u^2 Cos[2u]}}, {u, 0, 2 Pi, Pi/16}],
   ScaleFunction->(s &)]
```

Ha most $s = 0.1$ és $s = 1$ közötti értékekre elkészítjük az ábrákat és az eredményből animációs filmet készítünk (kijelöljük az ábrákat, majd rákattintunk a filmszalagot mutató ikonra vagy a megfelelő menüpontra), akkor egy lüktető (bár kissé szőrös) szív jelenik meg előttünk!

Bizonyos esetekben még három dimenzióban is áttekinthető az iránymező. Ehhez a `Graphics`PlotField3D`` programcsomag alábbi függvényei használhatóak:

```
ListPlotVectorField3D     PlotVectorField3D
PlotGradientField3D
```

Hívjuk be a programcsomagot:

```
<<Graphics`PlotField3D`
```

A válasz egy figyelmeztető üzenet, amely szerint a `ScaleFunction` és a `MaxArrowLength` azonosító több összefüggésben is előfordul, s valamelyik definíciójuk elfedheti a másikat.

Tanulmányozzuk most a Lorenz-egyenletet dinamikai szempontból érdekes paraméterértékek esetén! Ezeket az értékeket úgy választottuk meg, hogy az egyensúlyi helyzetek száma és minősége különböző legyen:

```
s = 10; b = 8/3;
lorenz[r_] := PlotVectorField3D[
   {s (y - x), r x - y - x z, x y - b z},
   {x, -1, 1}, {y, -1, 1}, {z, 1, 3}]
```

Tanulságos lehet megvizsgálni r következő értékeit: 0.9, 1.1, 20, 24.5, 25.

3.5.2. Megoldás kvadratúrával

Lineáris, valamint egyszerű szerkezetű nemlineáris egyenletek megoldását elvárjuk a matematikai programcsomagoktól. A szokásos kritérium az, hogy

Kamke [38] gyűjteményéből mennyit képesek megoldani. A *Mathematica* néhány kivételével mindet megoldja, és nemcsak explicit megoldásokat ad, hanem — amikor csak erre van mód, vagy ez az egyszerűbb — implicite-ket is. Mivel ránézésre egy egyenletről még nagy gyakorlat mellett is nehéz eldönteni, hogy megoldása mennyire könnyű vagy nem, ezért érdemes vizsgálódásunk legelején behívni a

`Calculus`DSolve`` csomag `DSolve` függvényét,

amely azonos nevű, de nagyobb teljesítőképességű, mint belső társa.

Az eredményeket érdemes mindig *ellenőrizni*. Gray [29] sok egyéb szempontból is kiváló könyvében ezt mindig megteszi; ebből kiderül, hogy maga az ellenőrzés sem mindig teljesen triviális. Az alábbiakban erre csak néhány példát mutatunk.

Először egy negyedrendű lineáris egyenletet oldunk meg:

```
egyenlet =
   (-15-16x^4) y[x] + 24x y'[x] - 24x^2 y''[x] +
   32x^3 y'''[x] + 16x^4 y''''[x] == 0;
ered = DSolve[egyenlet, y[x], x];
```

• Hivatkozási módok

Az eredmény egy olyan (esetünkben közlésre nem méltóan hosszú) lista, amelynek első és egyetlen eleme egy lista, amelynek egyetlen eleme egy — lokális értékadásban felhasználandó — *transzformációs szabály*. Mindezek alapján a megoldás x helyen vett helyettesítési értékére — amit tovább szeretnénk egyszerűsíteni — az `ered[[1, 1, 2]]` jellel hivatkozhatunk, hiszen az `ered` nevet adtuk az egész eredménynek. (A névadás igen kifizetődő befektetés, ugyanis a `%` jelnek és rokonainak használata a legkisebb javítás vagy módosítás esetén is hibákra vezet.)

A kapott eredmény áttekinthetetlen, de egyszerűbb alakra lehet hozni:

```
egyszeru = Factor[Together[PowerExpand[ered[[1, 1, 2]]]]]
((-1)^(1/8) (-((-1)^(3/4) C[1] Cos[x]) - (-1)^(1/4) C[3] Cos[x] -
   (-1)^(3/4) C[1] Cosh[x] + (-1)^(1/4) C[3] Cosh[x] +
   C[2] Sin[x] - I C[4] Sin[x] +
   C[2] Sinh[x] + I C[4] Sinh[x]))/
   (Sqrt[2Pi^3] Sqrt[x])
```

Az Olvasóra (feladatul) hagyjuk annak kiötlését, hogy az eredményt miképpen lehet úgy átalakítani, hogy a négy trigonometrikus függvény mindegyike csak egyszer szerepeljen — a megfelelő együtthatókkal.

Mint látható, a `DSolve` függvény három argumentuma közül az első az egyenlete(ke)t tartalmazza, beleértve a kezdeti és peremfeltételeket is. Szintaktikai akadálya nincs, hogy szokatlan (például nemlineáris, vagy határértékre vonatkozó) feltételekkel egészítsük ki a differenciálegyenleteket; esetenként még kaphatunk is megoldást. A második argumentum a megoldásfüggvény vagy annak helyettesítési értéke. Ha a megoldást az egyenletbe vissza szeretnénk helyettesíteni, akkor az első lehetőséget válasszuk, hogy (egyszerűen) ki tudjuk számolni a szükséges deriváltakat is. Egyébként pedig általában a második lehetőség az áttekinthetőbb. A harmadik argumentum a független változó(ka)t tartalmazza.

A három argumentum bármelyikét listaként kell megadni, ha egynél több összetevője van.

A következő feladatban az egyenletben bemenetként szerepel egy egységugrásfüggvény. Ennek az egyenletnek a megoldása közben megismerkedünk a megoldásra való hivatkozás egy másik módjával is:

```
<<Calculus`DiracDelta`
phi = DSolve[
   {y'[x] + 3 y[x] == 6 UnitStep[x + 0.2], y[0] == 1},
   y, x][[1, 1, 2]];
```

A megoldásfüggvény tehát — a fentiek értelmében — φ. Érdemes ábrázolni ezt a függvényt — a szokásos módon:

```
Plot[phi[x], {x, -0.5, 0.5}]
```

A legegyszerűbb hivatkozási mód tehát a következő:

```
psi = DSolve[{y'[x] == 1 - y[x]^2, y[0] == 2},
   y, x][[1, 1, 2]];
```

Ennél a módnál illik arra vigyázni, hogy az egyenletben szereplő ismeretlen függvény neve ne legyen azonos a konkrét megoldás nevével. Megjegyzendő, hogy (például másodfokú) algebrai egyenleteknél erre szoktak ügyelni, differenciálegyenleteknél kevésbé.

Általában viszont azt a hivatkozási módot szokták javasolni, amelynél az eredményt a továbbiakban *transzformációs szabály*ként használjuk fel:

```
DSolve[{x y''[x] + (1-x^2) y'[x] - x y[x] == 0,
   y[0.5] == 2, y'[0.5] == 1}, y[x], x];
Plot[Evaluate[y[x] /. %], {x, -0.3, 1.6}]
```

Végül egy példa arra az esetre, amikor a megoldást (hibaüzenetek sorozata után) *implicit* alakban kapjuk meg:

```
DSolve[y'[x] == a y[x]^3 + b x^(-3/2), y[x], x]
Solve[Integrate[
                    1
--------------------------------------, y[x]]
   16                                3
--------- + y[x] + 2 a x y[x]
3 Sqrt[x]
    Log[x]
  - ------ == C[1], y[x]]
      2
```

3.5.3. Első integrálok

Ha nem tudjuk megoldani kvadratúrával az egyenletet, akkor még mindig lehet, hogy meg tudjuk határozni egy vagy néhány első integrálját. Az ehhez szükséges külső függvényt a

```
Calculus`PDSolve1`
```

programcsomagban találjuk meg.

Határozzuk meg a Volterra–Lotka-rendszer egy első integrálját:

```
<<Calculus`PDSolve1`
FirstIntegrals[{x'[t]==x[t]-x[t] y[t],
   y'[t]==x[t] y[t]-y[t]}, {x[t], y[t]}, t];
Apart[%]
{-Log[x[t]] - Log[y[t]] + x[t] + y[t]}
```

3.5.4. Numerikus megoldás

Itt olyan kezdetiérték-problémákat fogunk megoldani, amelyek megoldása kvadratúrával nem határozható meg. A megoldás alapvető eszköze az

```
NDSolve
```

belső függvény. Két példát mutatunk az eljárás használatára.

Először egy neurobiológiai modellt vizsgálunk, amelynek érdekessége, hogy a felírt differenciálegyenlet-rendszer *merev*, azaz az egyes komponensfüggvények deriváltjai között több nagyságrend különbség van. Az ilyen

egyenletek megoldására Gear óta speciális módszereket alkalmaznak. A *Mathematica* automatikusan vizsgálja a merevséget és szükség esetén módszert változtat.

Vizsgáljuk először a következő leegyszerűsített formális kémiai reakciót [19]:

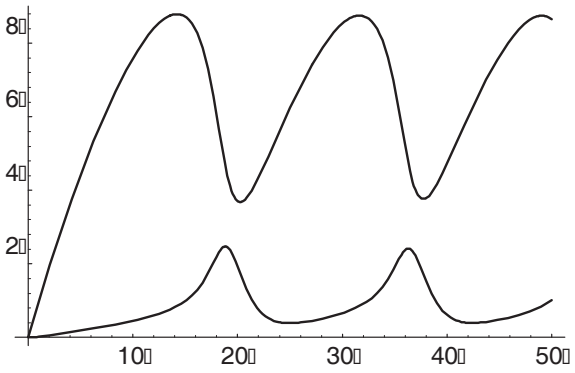
$$2\mathcal{X} + \mathcal{Y} \longrightarrow 3\mathcal{X} \quad 2\mathcal{X} \longrightarrow \mathcal{X} + \mathcal{Y} \quad \mathcal{Y} \longrightarrow \mathcal{X} \longrightarrow \mathcal{O} \longrightarrow \mathcal{Y}$$

Írjuk föl és oldjuk meg ennek a reakciónak a tömeghatás kinetikájú indukált differenciálegyenletét alkalmas sebességi állandókkal:

```
ered = Table[NDSolve[
{x'[t] == 0.1x[t]^2 y[t] - 0.1x[t]^2 + 0.05y[t] - x[t],
    y'[t] == -0.1x[t]^2 y[t] + 0.1x[t]^2 - 0.05y[t] + 1,
    x[0] == 0, y[0] == 0},
    {x[t], y[t]}, {t, 0, 50}][[1, i, 2]], {i, 1, 2}];
```

Az ábrán jól látszik, hogy egyes időszakokban nagy különbség van a két anyagfajta mennyiségének deriváltja között:

```
Plot[{ered[[1]], ered[[2]]}, {t, 0, 50}]
```



Ha az előző utasítás elejére odaírjuk még a `Parametric` szót, akkor a megoldások helyett a *trajektóriák*at (*szelektivitási görbék*et) kapjuk meg.

Második példánkban egy egyszerű *irányítás*i feladatot oldunk meg. Egy tárgy hőmérsékletét adott határok között kell tartani olyan módon, hogy ha az alsó határt eléri a test hőmérséklete, akkor bekapcsolunk valamilyen fűtést, ha pedig eléri a fölső határt, akkor kikapcsoljuk azt.

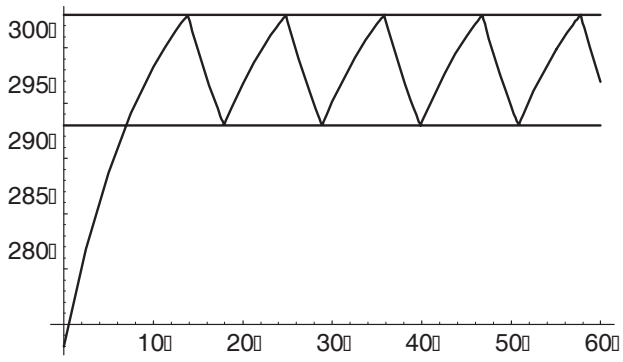
A hűlés folyamatát a Newton-féle hőátadási törvény írja le, amely szerint (önkényesen, de alkalmasan megválasztott állandóval):

$$T'(t) = -0.1(T(t) - 273).$$

A fűtés két esetben működik: ha a hőmérséklet az alsó határ alatt van, vagy pedig ha a két határ között van és már eddig is működött.

Írjuk fel a végső megoldást, s majd utólag fűzzünk hozzá megjegyzéseket:

```
NDSolve[{T'[t]==-0.1*(T[t]-273.)+Which[
   T[t]<293., h=4.,
   T[t]>303., h=0.,
   293.<T[t]<303. && h==4., 4.,
   293.<T[t]<303. && h==0., 0.],
   T[0.]==273.}, T[t], {t, 0., 60.},
   MaxSteps->Infinity, MaxStepSize->0.01]
{{T[t] -> InterpolatingFunction[{0., 60.}, <>][t]}}

Plot[Evaluate[{293., 303., T[t] /. %}], {t, 0., 60.}]
```



Tanulságos a lépések számát kisebbre vagy a maximális lépésközt nagyobbra venni és megnézni, mi történik a megoldással.

Ha a kezdeti feltételt magasabbra választjuk, akkor egy kemencéből kivett tárgyra gondolhatunk, amelynek a hőmérsékletét adott határok között kell tartani.

Ez a feladat nyilvánvalóan másutt is alkalmazható modellként. Például valamilyen gyógyszer szintjét szeretnénk adott szinten tartani a vérben. Ezt nyilván csak adott pontossággal követelhetjük meg, azaz csak azt írhatjuk elő, hogy a szint két adott határ közé essék. Ezt viszont egyszerű esetben a fenti modell jól közelíti.

Első általánosításként érdemes a Newton-egyenlet helyett *nemlineáris* egyenletet vennünk.

Értelmes általánosításhoz jutunk, ha a határokat nem állandóknak vesszük, hanem két *függvénnyel* adjuk meg. Esetleg ezeket a függvényeket valamilyen *differenciálegyenlet megoldása*ként kapjuk.

Másik, kézenfekvő általánosításhoz úgy jutunk, ha differenciálegyenlet-*rendszer*t tekintünk és a megoldástól azt követeljük meg, hogy benne maradjon egy (esetleg időtől függő) halmazban, így bizonyos típusú üldözési feladatok modelljéhez jutunk.

• **Numerikus módszerek**

Az `NDSolve` belső függvény többlépéses prediktor-korrektor módszert használ, amelynek paraméterei az opciók segítségével módosíthatók. Amennyiben a program érzékeli az egyenlet merevségét, automatikusan implicit differencia-módszerre tér át.

A Runge–Kutta-módszerrel oldhatjuk meg az egyenletet a

```
RungeKutta
```

függvény segítségével. Ezt a `ProgrammingExamples`RungeKutta`` programcsomagban találhatjuk meg. A többlépéses lineáris Runge–Kutta-típusú módszereknek a tanulmányozásához segítséget kaphatunk a

```
NumericalMath`Butcher`
```

programcsomag `RungeKutta*` függvényeitől.

3.5.5. Laplace-transzformáció

A mérnöki irodalomban a Laplace-transzformációt a leggyakrabban lineáris, állandó együtthatós közönséges differenciálegyenletekre és differenciálegyenlet-rendszerekre (vagy ilyenekre visszavezethetőkre) szokták alkalmazni, illetve olyan egyenletekre, amelyek bal oldalán egy lineáris, állandó együtthatós operátornak az ismeretlen függvényen felvett értéke, a jobb oldalán pedig egy ugyanilyen típusú operátornak egy ismert, úgynevezett *bemeneti függvény*en felvett értéke áll. Az első két típusra — mint általában, úgy itt is — fölösleges a Laplace-transzformációt alkalmazni, hiszen használata általában semmilyen előnnyel nem jár. Kivételt képeznek talán

ez alól a nem folytonos jobb oldalú differenciálegyenletek (amelyeket például bekapcsolási jelenségek modellezésére használnak). Következzék most egy ilyen példa:

```
<<Calculus`LaplaceTransform`
<<Calculus`DiracDelta`
egy = y'[x] + 4 y[x] == UnitStep[x];
LaplaceTransform[egy, x, s]
```

$$4\,\text{LaplaceTransform}[y[x], x, s] + s\,\text{LaplaceTransform}[y[x], x, s] - y[0] == \frac{1}{s}$$

```
% /. LaplaceTransform[y[x], x, s] -> Y[s]
```

$$-y[0] + 4\,Y[s] + s\,Y[s] == \frac{1}{s}$$

Kaptunk tehát egy (egyszerű, algebrai) egyenletet az Y Laplace-transzformáltra. Oldjuk meg:

```
Solve[(% /. y[0]->2), Y[s]]
```

$$\{\{Y[s] \to -\left(\frac{-1 - 2\,s}{s\,(4 + s)}\right)\}\}$$

A kapott eredményt „vissza kell transzformálni”:

```
InverseLaplaceTransform[%[[1, 1, 2]], s, x]
```

$$\frac{1}{4} + \frac{7}{4E^{4x}}$$

Integrálegyenleteknél (például konvolúciót tartalmazóknál) is előfordul, hogy a megoldás meghatározásában ténylegesen segít a Laplace-transzformáció. Most megoldunk egy ilyen integrálegyenletet:

```
integy = t==Integrate[E^(t-u) g[u], {u, 0, t}];
ltintegy = LaplaceTransform[integy, t, s]
```

$$s^{-2} == \frac{\text{LaplaceTransform}[g[t], t, s]}{-1 + s}$$

```
Solve[
(ltintegy /. LaplaceTransform[g[t], t, s] -> G[s]), G[s]]
```

$$\{\{G[s] \to \frac{-1 + s}{s^2}\}\}$$

```
InverseLaplaceTransform[%[[1, 1, 2]], s, t]
```

$$1 - t$$

Végül pedig megemlítjük, hogy a parciális differenciálegyenletekről szóló részben is alkalmazzuk majd a Laplace-transzformációt.

3.5.6. Megoldás hatványsorokkal

Ha a kvadratúrával nem tudjuk megkapni egy differenciálegyenlet megoldását, akkor megpróbálkozhatunk azzal, hogy hatványsor alakban keressük a megoldást. Ennek az eljárásnak az elméleti hátterét az az Eulertől származó eredmény képezi, amely szerint analitikus jobb oldalú egyenletnek létezik analitikus megoldása.

A *Mathematica* segítségével egy hatványsor alakú megoldás tetszőleges, de véges sok tagja meghatározható. Első példánk Graytől [29] származik.

Meg akarjuk oldani hatványsor alakban az

$$y'(x)^2 - y(x) = x$$

differenciálegyenletet. Közelítsük egy hatványsor véges szeletével az y függvényt; az ismeretlen együtthatókat jelöljük $a[i]$-vel:

```
y[x_] := SeriesData[x, 0, Table[a[i], {i, 0, 6}]]
de = D[y[x], x]^2 - y[x] == x
```

```
(-a[0] + a[1]^2 ) +
(-a[1] + 4 a[1] a[2]) x +
(-a[2] + 4 a[2]^2 + 6 a[1] a[3]) x^2 +
(-a[3] + 12 a[2] a[3] + 8 a[1] a[4]) x^3 +
(9 a[3]^2 - a[4] + 16 a[2] a[4] + 10 a[1] a[5]) x^4 +
(24 a[3] a[4] - a[5] + 20 a[2] a[5] + 12 a[1] a[6]) x^5 +
O[x]^6 == x
```

Szeretnénk az együtthatókra egy egyenletrendszert kapni:

```
ehoegy = LogicalExpand[de]
```

```
-a[0] + a[1]^2 == 0 &&
-1 - a[1] + 4 a[1] a[2] == 0 &&
-a[2] + 4 a[2]^2 + 6 a[1] a[3] == 0 &&
-a[3] + 12 a[2] a[3] + 8 a[1] a[4] == 0 &&
9 a[3]^2 - a[4] + 16 a[2] a[4] + 10 a[1] a[5] == 0 &&
24 a[3] a[4] - a[5] + 20 a[2] a[5] + 12 a[1] a[6] == 0
```

Ahhoz, hogy ezeket megoldjuk, szükségünk van még egy kezdeti feltételre:

```
ehomo = Solve[{ehoegy, a[0]==1}, Table[a[i], {i, 0, 6}]]
```

$$\{\{a[0] \to 1,\ a[6] \to 0,\ a[5] \to 0,\ a[4] \to 0,\ a[3] \to 0,\ a[2] \to 0,\ a[1] \to -1\},$$
$$\{a[0] \to 1,\ a[6] \to \frac{469}{11520},\ a[5] \to -\frac{41}{960},\ a[4] \to \frac{5}{96},\ a[3] \to -\frac{1}{12},\ a[2] \to \frac{1}{2},\ a[1] \to 1\}\}$$

Helyettesítsük vissza a két megoldást y-ba, hogy megkapjuk a hatványsor alakú közelítést:

```
sormo = y[x] /. ehomo
```

$$\{1 - x + \frac{O[x]}{7},$$
$$1 + x + \frac{x^2}{2} - \frac{x^3}{12} + \frac{5x^4}{96} - \frac{41x^5}{960} + \frac{469x^6}{11520} + O[x]^7\ \}$$

Mielőtt ábrázolhatnánk, normálalakú kifejezéssé kell alakítani ezeket:

```
Plot[Evaluate[Normal[sormo]], {x, 0, 3}]
```

Végül megmutatjuk, hogy „hatodrendben” mindkét függvény megoldás:

```
de /. ehomo
```

$$\{x + O[x]^6 == x,\ x + O[x]^6 == x\}$$

Normálformájúvá alakítva az egyenleteket, pontos egyezéseket kapunk:

```
Normal[%]
{True, True}
```

Egyes — különösen kedvező — esetekben előfordulhat, hogy meg tudunk sejteni egy, a hatványsor együtthatói között fennálló rekurzív összefüggést, s ekkor a

```
DiscreteMath`RSolve`
```

programcsomag `RSolve` függvényét használhatjuk. Triviális példát mutatunk. Legyen a megoldandó kezdetiérték-probléma:

$$y'(x) = y(x) \quad y(0) = 1.$$

```
y[x_] := SeriesData[x, 0, Table[a[i], {i, 0, 4}]]
```

```
de = D[y[x], x] - y[x] == 0
```

$$(-a[0] + a[1]) + (-a[1] + 2\,a[2])\,x + (-a[2] + 3\,a[3])\,x^2 + (-a[3] + 4\,a[4])\,x^3 + O[x]^4 == 0$$

Az együtthatókra ezt az egyenletrendszert kapjuk:

```
LogicalExpand[de]
```

```
-a[0] + a[1] == 0 && -a[1] + 2 a[2] == 0 &&
   -a[2] + 3 a[3] == 0 && -a[3] + 4 a[4] == 0
```

Használjuk föl a kezdeti feltételt, és „vegyük észre”, hogy milyen összefüggés van az együtthatók között:

```
<<DiscreteMath`RSolve`
   RSolve[{(n+1) a[n+1] - a[n] == 0, a[0] == 1}, a[n], n]
```

$$\{\{a[n] \to \frac{1}{n!}\}\}$$

Verifikáljuk, hogy amit kaptunk, az valóban megoldás:

```
f[x_] = PowerSum[(a[n] /. %), x, n][[1]];
And[f'[x] == f[x], f[0] == 1]
```

```
True
```

Végül megemlítjük, hogy Vvedensky [86] könyvében *szinguláris feladat*ok megoldására használja a hatványsoros módszert.

Hatványsor alakú megoldások keresésénél érdemes még gondolni a következő programcsomagokra:

```
Algebra`SymbolicSum`
DiscreteMath`CombinatorialSimplification`
```

3.5.7. Szukcesszív approximáció

A szukcesszív approximáció módszerét inkább elméleti vizsgálatoknál szokás alkalmazni, éppen azért, mivel a konkrét feladatok megoldásánál fellépő integrálások kézzel nehezen végezhetők el. Legyen a megoldandó egyenletek

általános alakja:

$$x'(t) = f(t, x(t)) \quad x(\tau) = \xi,$$

ahol f $\mathbf{R} \times \mathbf{R}^n \to \mathbf{R}^n$ típusú folytonos függvény, $\tau \in \mathbf{R}, \xi \in \mathbf{R}^n$. Ezek közül egy olyan esettel foglalkozunk, ahol $n = 1$; konkrétan az

$$x'(t) = x(t)^2 \quad x(0) = 1$$

egyenlet megoldását fogjuk megadni az identitásfüggvényből (*Mathematica*i megfelelője: `Function[s, s]`) mint kezdeti közelítésből kiindulva. (Az alábbiakból ki fog tűnni, hogy programozási szempontból az általános eset megoldása semmivel sem nehezebb a speciálisnál.)

A kezdeti definíciók után kiszámítjuk a pontos megoldást:

```
tau = 0;
ksi = 1;
f[u_, v_] := v^2
pontos[z_] = DSolve[
   {x'[z] == (x[z])^2, x[0] == 1}, x[z], z][[1, 1, 2]]
```

$$\frac{1}{1-z}$$

Következzék a differenciálegyenlettel egyenértékű integrálegyenlet jobb oldalát definiáló integráloperátor:

```
A[fi_] := Simplify[
  Function[t, ksi + Integrate[f[s, fi[s]], {s, tau, t}]]]
```

Ez az operátor valóban függvényekhez függvényeket rendel, ezért amikor függvényértékekre lesz szükségünk az alábbiakban, alkalmazzuk majd a `#[z]&` „függvényérték-számoló”-t.

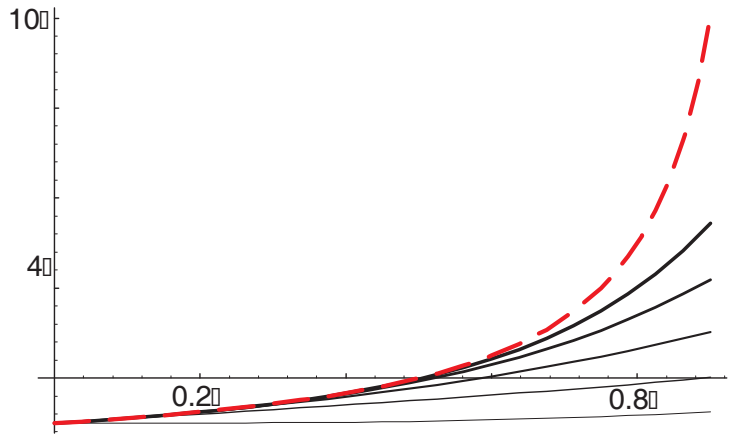
Számítsuk ki az első öt közelítést:

```
NestList[A, Function[s, s], 5];
```

Az eredményből hagyjuk el az első függvényt, és a listához vegyük hozzá a pontos megoldást az összehasonlító ábrázolás előkészítése végett, majd ábrázoljuk — áttekinthető módon — az összes függvényt:

```
Append[Map[#[z]&, Rest[%]], pontos[z]];
Plot[Evaluate[%], {z, 0, 0.9}, PlotRange -> All,
   PlotStyle -> {
   {Thickness[0.001]},
   {Thickness[0.002]},
   {Thickness[0.003]},
   {Thickness[0.004]},
```

```
{Thickness[0.005]},
{Thickness[0.006], Dashing[{.05, .03}], RGBColor[1, 0, 0]}},
Ticks -> {{0.2, 0.8}, {4, 10}}]
```



(Hogyan adhatnánk meg elegánsabban a vonalvastagságokat?)

Érdemes arra is gondolnunk, hogy a *Mathematica* megkönnyítheti az ilyenkor szükséges *becslés*ek elvégzését is.

3.5.8. Stabilitáselmélet

A Lorenz-egyenlet konkrét példáján elvégezve a lineáris stabilitásvizsgálatot megmutatjuk, hogy ez itt lényegében azonos a matematikai képletek leírásával. A szokásosan r-rel jelölt paramétert fogjuk változtatni, a többieknek konkrét értéket adunk. Első lépésben meghatározzuk a stacionárius pontokat:

```
s = 10;
b = 8/3;
f[x1_, x2_, x3_]:=
   {s (x2 - x1), r x1 - x2 - x1 x3, x1 x2 - b x3}
stac = Solve[f[x1, x2, x3] == 0, {x1, x2, x3}];
```

(Ha nem ilyen egyszerű polinommal találkozunk, akkor érdemes visszalapozni az egyenletek megoldásáról szóló 3.3. szakaszhoz.) A következő lépés a Jacobi-mátrixnak és sajátértékeinek kiszámítása a stacionárius pontokban:

```
jacobi[x1_, x2_, x3_] := Outer[D, f[x1, x2, x3],
   {x1, x2, x3}]
Map[Eigenvalues, jacobi[x1, x2, x3] /. stac];
```

A számolások eredményeiből jól látszik, hogy a sajátértékek valós részének előjele miként változik, ha ismét a fent említett $r = 0.9, 1.1, 20, 24.5, 25$ paramétersorozaton vizsgáljuk. Az első helyen például azt kapjuk, hogy az első egyensúlyi helyzet stabilis, a másik kettő pedig instabilis:

```
Chop[N[% /. r -> 0.9]] // TableForm
-2.66667 -10.9083 -0.0916731
 0.169767 -10.9736 -2.86284
 0.169767 -10.9736 -2.86284
```

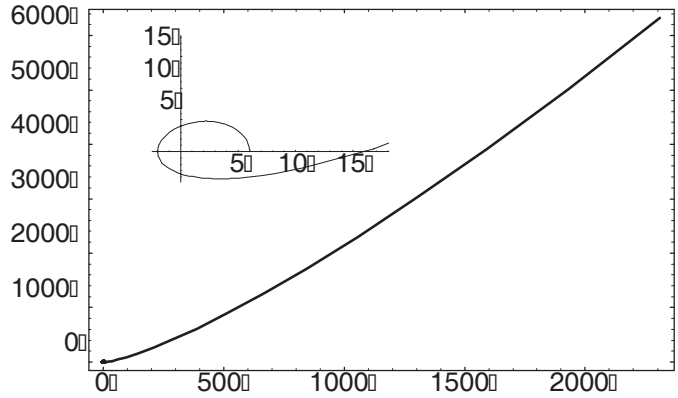
• A Mihajlov-féle kritérium

Lineáris stabilitásvizsgálatnál a végső lépésben azt kell megállapítanunk, hogy a linearizált rész karakterisztikus polinomjának sajátértékei milyen valós részűek. Ezt a legnehezebb a közismert Hurwitz-módszerrel eldönteni. (Bár ennek alkalmazását megkönnyíti a 3.4.4. pontban definiált `SarokAldet` függvény.) Egy kevés számolást igénylő eljárást ad a Mihajlov-módszer, amely szerint az n-edfokú f polinom stabilitásának szükséges és elegendő feltétele az, hogy $\omega \mapsto f(i\omega)$ ne menjen át az origón és kerülje meg azt $n\pi/2$ szögben pozitív irányban, midőn ω befutja a $[0, \infty)$ intervallumot. Illusztráljuk a kritériumot egy példán:

```
f[t_]:=t^5 + 2t^4 + 7t^3 + 8t^2 + 10t + 6
kicsi = ParametricPlot[{Re[f[I omega]], Im[f[I omega]]},
   {omega, 0, 6}]
```

A következő ábrán látszik a hosszú távú viselkedés, de elhelyeztük benne a kezdeti viselkedést mutató részt is:

```
ParametricPlot[{Re[f[I omega]], Im[f[I omega]]},
   {omega, 0, 6}, PlotRange -> All, Frame -> True,
   Prolog -> Rectangle[{50, 3000}, {1300, 6000}, kicsi]]
```



Egy nyilvánvaló potenciális alkalmazási területet a stabilitáselméleten belül csak megemlítünk: igen sok számolást igényel általában, ha adott alakú Ljapunov-függvényt keresünk egy differenciálegyenlethez. Ehhez esetleg használható lehet az első integrál, vagy azért, mert deriváltja definit, és ekkor a stacionárius pont stabilis, vagy azért, mert az egyenlet jobb oldalának egy részéhez találunk olyan első integrált, amely az egész egyenlethez megfelel Ljapunov-függvénynek.

3.5.9. Parciális differenciálegyenletek

• Elsőrendű egyenletek

Elsőrendű parciális differenciálegyenletek megoldásához általában érdemes azonnal behívni a `Calculus`PDSolve1`` csomagot:

```
<<Calculus`PDSolve1`
```

Először oldjunk meg egy *kvázilineáris* egyenletet:

```
DSolve[-(-x - y + u[x, y])^2 +
   x^2 (D[u[x, y], x] - D[u[x, y], y]) == 0,
   u[x, y], {x, y}]
```

```
{{u[x, y] ->
                                    2
   - (2 x + y - x  C[1][x+y] - x y C[1][x+y])}}
                 -1+ x C[1][x+y]
```

Míg közönséges differenciálegyenletek esetén `C[1]`, `C[2]`, ... tetszőleges állandókat jelölt, itt ezek tetszőleges (elegendően sokszor differenciálható) függvények. Az eredmény pedig az egyenlet *általános megoldás*a.

Most meghatározzuk egy lineáris, háromváltozós függvényre vonatkozó egyenlet általános megoldását:

```
DSolve[-x y z + z^2 D[w[x, y, z], z] +
   y^2 D[w[x, y, z], y] + x^2 D[w[x, y, z], x] == 0,
   w[x, y, z], {x, y, z}]
```

$$\{\{w[x, y, z] \to -\left(\frac{x\, y^2\, z\, \mathrm{Log}\left[\frac{x}{y}\right]}{-(x\, y) + y^2 + x\, z - y\, z}\right) + \frac{x\, y\, z^2\, \mathrm{Log}\left[\frac{x}{z}\right]}{-(x\, y) + x\, z + y\, z - z^2} + C[1]\left[\frac{1}{x} - \frac{1}{y}, \frac{1}{x} - \frac{1}{z}\right]\}\}$$

Az általános megoldás nem mindig adható meg. Sok esetben viszont megadható a *teljes integrál*, amelyből majdnem minden peremérték-feladat megoldása kvadratúrával előállítható:

```
CompleteIntegral[
D[u[x, y], y] == (u[x, y] + x^2 D[u[x, y], x]^2)/y,
    u[x, y], {x, y}]
```

$$\{\{u[x, y] \to \frac{-B[1]^2}{4} + y\, B[2] - \frac{B[1]\, \mathrm{Log}[x]}{2} - \frac{\mathrm{Log}[x]^2}{4}\}\}$$

Ha nincs általános megoldás, akkor önállóan a teljes megoldás meghatározásával kísérletezik a program:

```
DSolve[
-c + b D[u[x,y], y] + a y D[u[x,y], x] + D[u[x,y], x]^2
   == 0, u[x,y], {x, y}]
DSolve::nlpde:
      This is a nonlinear partial differential
            equation. General solution is not available.
            Trying to build a complete integral instead.
```

$$\{\{u[x, y] \to \frac{c\, y}{b} + x\, B[1] - \frac{a y^2 B[1]}{2b} - \frac{yB[1]^2}{b} + B[2] + y\, B[2]\}\}$$

Végül megemlítjük, hogy a megoldó algoritmusok azon alapulnak, hogy meghatározzák közönséges differenciálegyenlet-rendszerek első integráljait. Ezek közvetlenül is megkaphatók a `FirstIntegrals` függvény segítségével, amelynek használatára a közönséges differenciálegyenleteknél láttunk példákat.

• Másodrendű egyenletek

Először megmutatjuk egy olyan (vegyes) feladat megoldását, amelyhez a `Calculus`PDSolve1`` csomag nem használható. Szeretnénk megoldani a

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}$$

diffúziós vagy *hővezetési* egyenletet az

$$u(x,0) = 3\sin(2\pi x) \quad x \in [0,1]$$

kezdeti és az

$$u(0,t) = u(1,t) = 0$$

homogén peremfeltétel mellett. Alkalmazzuk a Laplace-transzformációt:

```
<<Calculus`LaplaceTransform`
LaplaceTransform[
   D[u[x, t], t] == D[u[x, t], {x, 2}], t, s]
s LaplaceTransform[u[x, t], t, s] - u[x, 0] ==
   LaplaceTransform[u[x, t], t, s]
                    (2, 0)
```

(Vegyük észre, hogy semmilyen ügyeskedésre nem volt szükség ahhoz, hogy az egyenlet két oldalát transzformáljuk.) Most a transzformált egyenletet áttekinthetőbbé tesszük:

```
% //. {LaplaceTransform[u[x_,t], t, s] :> U[x],
   LaplaceTransform[Derivative[xn_,0][u][x_,t], t, s] :>
   D[U[x], {x,xn}]}
- u[x,0] + s U[x] == U''[x]
```

Ez az egyenlet az adott kezdeti feltétellel könnyen megoldható:

```
DSolve[{% /. u[x_, 0] -> 3 Sin[2Pi x],
   U[0] == 0, U[1] == 0}, U[x], x]
                    3 Sin[2 Pi x]
{{U[x] -> ------------------------}}
                      2
          4 Pi  + s
```

A végeredményt inverz transzformációval kapjuk:

```
InverseLaplaceTransform[U[x] /. %, s, t]
```

$$\left\{\frac{3\,\text{Sin}[2\,\text{Pi}\,x]}{E^{4\text{Pi}^2 t}}\right\}$$

3.5.10. Variációszámítás

A *Mathematica* tartalmaz egy külön programcsomagot variációszámítási feladatok megoldására, ez a `Calculus`VariationalMethods``. Ez rögzített végpontú problémákat tud megoldani, amelyeknél azonban a független és a függő változó is lehet vektor.

Mindenekelőtt hívjuk be a variációszámítási programcsomagot:

```
<<Calculus`VariationalMethods`
```

A *minimális felszínű forgástest* meghatározásához határozzuk meg az

$$\int_{xmin}^{xmax} y(x)\sqrt{1+y'(x)^2}dx$$

funkcionál első variációját:

```
dminfelsz = VariationalD[y[x] Sqrt[1+y'[x]^2], y[x], x]
```

$$\frac{1 + y'[x]^2 - y[x]\ y''[x]}{(1 + y'[x]^2)^{\frac{3}{2}}}$$

Az *Euler-féle egyenlet*et — közvetlenül is — könnyen föl tudjuk írni:

```
minfelsz = EulerEquations[y[x] Sqrt[1+y'[x]^2], y[x], x]
```

$$\frac{1 + y'[x]^2 - y[x]\ y''[x]}{(1 + y'[x]^2)^{\frac{3}{2}}} == 0$$

Valós-valós függvények körében a fenti egyenlet nyilván egyenértékű azzal, amelynél a fenti bal oldal számlálóját tesszük nullával egyenlővé:

```
DSolve[Numerator[%] == 0, y[x], x]
z[x_] = %[[1, 1, 2]] /. {C[1] -> 1, C[2] -> 0}
```

$$\frac{E^x\,(1+E^{-2x})}{2}$$

Talán nem meglepő, hogy eredményül a láncgörbét kapjuk. A minimális felszínű forgástestet a következőképpen lehet megjeleníteni:

```
<<Graphics`SurfaceOfRevolution`
SurfaceOfRevolution[z[x], {x, 0, 1},
   RevolutionAxis -> 1, 0, 0]
```

• A Ritz-módszer

A variációszámítás alapvető numerikus módszere a *Ritz-módszer*. Bizonyos speciális esetekben ennek alkalmazásához a `VariationalBound` és az `NVariationalBound` függvény használható.

3.5.11. Gyakorlatok és feladatok

1. Ha egy vektormező ábrázolásánál mindkét irányban 30–30 pontot akarunk felvenni és azt akarjuk, hogy a vektorok alapértelmezés szerinti hossza 0,3-szeresére változzék, akkor ezt így tehetjük meg:

```
PlotVectorField[{Sin[x y], Cos[x y]},
   {x, 0, Pi}, {y, 0, Pi},
   PlotPoints -> 30, ScaleFunction -> (.3#&)]
```

2. Három dimenzióban is megadhatunk egy potenciálfüggvényt:

```
PlotGradientField3D[x y^2 z^3,
   {x, -1, 1}, {y, -1, 1}, {z, -1, 1}]

PlotGradientField3D[x y^2 z^3,
   {x, 0, 3}, {y, 0, 3}, {z, 0, 3}]

PlotGradientField3D[Cos[1/x y z^2],

   {x, 0.01, 0.05}, {y, 0.09, 0.1}, {z, 0.09, 0.1},
   VectorHeads -> True]
```

3. Határozzuk meg az

$$xz\frac{\partial z(x,y)}{\partial x} + yz(x,y)\frac{\partial z(x,y)}{\partial y} = -xy$$

egyenlet általános megoldását, valamint azt az integrálfelületet, amely átmegy a $z(x,x^2) = x^3$ görbén.

Útmutatás. Határozzuk meg az egyenlet általános megoldását:

```
ClearAll[z];
DSolve[
   x z[x, y] D[z[x, y], x] + y z[x, y] D[z[x, y], y] == -x y,
   z[x, y], {x, y}]
```

$$\{\{z[x, y] \to -\frac{\text{Sqrt}[y]\ \text{Sqrt}[-x^2 - 2\ C[1][\frac{y}{x}]]}{\text{Sqrt}[x]}\},$$

$$\{z[x, y] \to \frac{\text{Sqrt}[y]\ \text{Sqrt}[-x^2 - 2\ C[1][\frac{y}{x}]]}{\text{Sqrt}[x]}\}\}$$

```
Z1[x_, y_] := %[[1, 1, 2]]
Z2[x_, y_] := %[[2, 1, 2]]

x^3 == PowerExpand[Simplify[Z1[x, y] /. y -> x^2]]
```

$$x^3 == -(\text{Sqrt}[x]\ \text{Sqrt}[-x^2 - 2\ C[1][x]])$$

```
Solve[%, C[1][x]]
```

$$\{\{C[1][x] \to -\frac{x^2\ (1+x^3)}{2}\}\}$$

```
Solve[x^3 == PowerExpand[Simplify[Z2[x, y] /. y -> x^2]],
   C[1][x]]
```

$$\{\{C[1][x] \to -\frac{x^2\ (1+x^3)}{2}\}\}$$

Az Olvasóra hagyjuk az adott feltételeket kielégítő megoldás teljes meghatározását.

4. Oldjunk meg egy lineáris állandó együtthatós rendszert a `MatrixExp` függvény felhasználásával.

3.6. Diszkrét matematika

A matematikának ez a területe az utóbbi évtizedekben rendkívül gyors fejlődésen ment keresztül és számos más területtel került kapcsolatba. Néhány példán keresztül bemutatjuk, hogy feladatainak megoldásánál hogyan használható a *Mathematica* program.

Megemlítjük, hogy a számelmélettel külön is foglalkozunk a 3.9. szakaszban, a valószínűségszámítással pedig a 3.10. szakaszban.

Nem soroltuk fel itt a listakezelő függvényeket (2.3.1. pont és 3.1.2. pont) és a lineáris algebrában (3.8. szakasz) használatos függvényeket, amelyek nyilvánvalóan sok helyütt alkalmazhatók a kombinatorikában. Mindössze néhány példát adunk ezek alkalmazására.

Felhívjuk a figyelmet Skiena [76] könyvére, az általunk is bemutatandó programcsomag onnan került be a *Mathematica* programcsomagjai közé. Megadjuk továbbá a szerző elektronikus címét is:

`skiena@sbcs.sunysb.edu`

A legújabb változat anonim ftp-vel letölthető a `cs.sunysb.edu` címről.

A matematikai fogalmakat illetően a [23] és a [39] jegyzetet ajánljuk az Olvasó figyelmébe.

3.6.1. Adott tulajdonságú listák

Néhány jól használható belső függvényt sorolunk fel:

```
Flatten                 Reverse
Order                   Signature
OrderedQ                SixJSymbol
Outer                   Sort
Permutations            ThreeJSymbol
Range
```

Először a kombinatorikában előforduló, adott feltételeknek eleget tevő objektumokat (listákat vagy halmazokat) fogunk előállítani, később pedig le fogjuk számlálni ezeket. Az alkalmazott belső függények egy része szerepel a listáknál és a halmazoknál, más részük a lineáris algebránál. Itt most a kombinatorika speciális objektumainak megszerkesztésére összpontosítunk:

```
per = Permutations[{1, 2, 3}]
{{1, 2, 3}, {1, 3, 2}, {2, 1, 3},
{2, 3, 1}, {3, 1, 2}, {3, 2, 1}}
```

Meghatározhatjuk azt is, hogy az egyes permutációk párosak-e vagy páratlanok, azaz páros vagy páratlan számú inverzióval adódnak-e a nagyság szerint („lexikografikusan") sorba állított számokból:

```
Signature /@ %
{1, -1, -1, 1, 1, -1}
```

Válasszuk ki a permutációk közül a párosakat:

```
Select[per, Signature[#] == 1&]
{{1, 2, 3}, {2, 3, 1}, {3, 1, 2}}
```

A `Signature` függvény fölfogható úgy is, mint egy teljesen antiszimmetrikus tenzor, vagy mint a Levi–Cività-féle, más néven epszilon-szimbólum. Mint ilyennek, rokona a `ClebschGordan` együttható, valamint a Wignertől származó `ThreeJSymbol` és a Racah-féle `SixJSymbol`.

A permutációk a `DiscreteMath`Permutations`` programcsomaggal vizsgálhatók részletesebben. Ebben a következő függvények találhatók:

```
FromCycles                RandomPermutation
Ordering                  ToCycles
PermutationQ
```

Először állítsuk elő 10 szám véletlen permutációját:

```
<<DiscreteMath`Permutations`
RandomPermutation[10]
{4, 1, 5, 3, 2, 8, 10, 9, 6, 7}
```

Bontsuk fel a kapott permutációt ciklusokra:

```
ToCycles[%]
{{4, 3, 5, 2, 1}, {8, 9, 6}, {10, 7}}
```

A ciklusokból kiindulva viszont megszerkeszthető a megfelelő permutáció; most példaként az eredetit állítjuk vissza:

```
FromCycles[%];
```

További, a permutációk tanulmányozására használható függvények vannak a `DiscreteMath`Combinatorica`` programcsomagban:

```
InversePermutation              RandomPermutation
Inversions                      RandomPermutation1
LongestIncreasingSubsequence    RandomPermutation2
MinimumChangePermutations       RankPermutation
```

Úgy is előállíthatjuk rögzített számú elem permutációit, hogy a másodiktól kezdve mindegyik egy transzpozícióval áll elő az őt megelőzőből:

```
<<DiscreteMath`Combinatorica`
MinimumChangePermutations[{1, 2, 3}]
{{1, 2, 3}, {2, 1, 3}, {3, 1, 2},
{1, 3, 2}, {2, 3, 1}, {3, 2, 1}}
```

Ha elég sok véletlen permutációt veszünk, akkor nagy valószínűséggel az összes elő fog fordulni közöttük:

```
Length[Union[Table[RandomPermutation[3], {20}]]]
6
```

Permutációk inverze is megkapható:

```
InversePermutation /@ {{1, 5, 3, 4, 2}, {3, 5, 2, 4, 1}}
{{1, 5, 3, 4, 2}, {5, 3, 1, 4, 2}}
```

Az inverziók száma egy permutációban ugyannyi, mint az inverzében:

```
(per = RandomPermutation[50];
{Inversions[per], Inversions[InversePermutation[per]]})
{629, 629}
```

Két permutáció kompozíciójának előállítására mutatunk egy módszert:

```
per1 = RandomPermutation[5]
{4, 3, 1, 5, 2}

per2 = RandomPermutation[5]
{2, 3, 4, 5, 1}

Part[per2, Part[per1, #]]& /@ Range[5]
{5, 4, 2, 1, 3}
```

A skatulyaelv segítségével bebizonyítható [39], hogy $n^2 + 1$ különböző számból álló sorozatból kiválasztható $n + 1$ hosszúságú monoton sorozat. Ezt így illusztrálhatjuk:

```
per = RandomPermutation[17]
{12, 6, 1, 4, 3, 2, 8, 5, 13,
7, 14, 9, 15, 10, 16, 11, 17}

LongestIncreasingSubsequence[per]
{1, 2, 5, 7, 9, 10, 11, 17}
```

A leghosszabb monoton csökkenő részsorozatot pedig így választhatjuk ki:

```
ize = 18 - per
LongestIncreasingSubsequence[ize]
18 - %
{12, 6, 4, 3, 2}
```

Végül megemlítjük a `DiscreteMath`Combinatorica`` programcsomag függvényei közül a `GrayCode`, a `KSubsets` és az `NthSubset` függvényt, amelyek adott tulajdonságú listák előállítására szolgálnak, s amelyeket részben már másutt használtunk, részben pedig a szakasz végén fogunk használatukra példákat mutatni a *Gyakorlatok és feladatok* között.

3.6.2. Leszámlálás

A kombinatorika tipikus feladata a *leszámlálás*: adott feltételeknek eleget tevő, adott méretű objektumok számát határozzuk meg ilyenkor.

Egy kevéssé elegáns módszer sokszor rendelkezésünkre áll (bár gyakorlatilag esetenként kivitelezhetetlen) leszámlálási feladatok megoldására: megszerkesztjük a kívánt tulajdonságú objektumokat, ezután megszámoljuk, hogy hány van belőlük:

```
Length[Permutations[3]]
6
```

A leszámlálási feladatokhoz segédeszközként a leggyakrabban a következő belső függvényeket használjuk:

```
Factorial                PartitionsP
Factorial2               PartitionsQ
Gamma                    Pochhammer
Binomial                 Stirling1
Length                   Stirling2
Multinomial
```

n elem permutációinak száma: $n!$. A `Factorial` függvény (ennek rövid alakja `!`) nem egész számokra a `Gamma` függvény megfelelő értékét adja:

```
{30!, 3.6!}
{265252859812191058636308480000000, 13.3813}
```

Hányféleképpen választhatunk ki n elem közül kettőt? Az erre adandó $\binom{n}{2}$ választ a *Mathematicá*ban így kaphatjuk meg:

```
Binomial[n, 2]
```

$$\frac{(-1 + n)\ n}{2}$$

Hat piros és öt fehér golyót annyiféleképpen rakhatunk sorba, ahányféleképpen 11 golyó közül kiválaszthatunk ötöt (vagy hatot). Ez a szám $\frac{(6+5)!}{6!5!}$ vagy $\binom{11}{5}$, értékét így számíthajuk ki:

```
{Multinomial[6, 5], Binomial[11, 6]}
{462, 462}
```

Hányféle szó állítható össze a *Mathematica* szó betűiből?

```
Multinomial[3, 2, 2, 1, 1, 1, 1]
1663200
```

Hányféleképpen lehet 4 tárgyat úgy permutálni, hogy egyik se maradjon a helyén?

```
<<DiscreteMath`CombinatorialFunctions`
Subfactorial[4]
9
```

`Subfactorial[n]` értékét az $n!\sum_{k=0}^{n}(-1)^k$ képlet felhasználásával számítja a program.

Ugyanebben a programcsomagban megtaláljuk még a szintén leszámláláshoz (pénzváltási probléma, bolyongások száma, helyes zárójelezések száma stb.) használt `CatalanNumber` külső függvényt, amelynek az értéke az n helyen $\frac{1}{n+1}\binom{2n}{n}$, valamint két rekurzíve definiált függvényt, a `Fibonacci`- és a `Hofstadter`-függvényt is. A Fibonacci-sorozattal rokon h Hofstadter-függvény definíciója:

$$h(1) := h(2) := 1 \qquad h(n) := h(n - h(n-1)) + h(n - h(n-2))$$

Speciális leszámlálási problémák a *számpartíciós* problémák is. Hányféleképpen lehet fölbontani az 50 számot pozitív számok, illetve különböző pozitív számok összegére?

```
{PartitionsP[50], PartitionsQ[50]}
{204226, 3658}
```

Tanulmányozzuk az n szám felbontásainak, illetve az n szám különböző számok összegére való felbontásainak számát! (Érdemes a megadottaktól eltérő transzformáltakkal is kísérletezni.)

```
ListPlot[Table[(N[Log[PartitionsP[n]]])^2, {n, 100}]]

ListPlot[Table[(N[Log[PartitionsQ[n]]])^2, {n, 100}]]

ListPlot[Table[
N[Exp[Log[PartitionsQ[n]]/Log[PartitionsP[n]]]],
{n, 100, 500}], PlotRange->All]
```

3.6.3. Egyszerű kombinatorikai azonosságok

Változókat is tartalmazó kombinatorikai kifejezések egyszerűsítését külön programcsomag könnyíti meg:

```
<<DiscreteMath`CombinatorialSimplification`
(n + 5)!/n!
(1 + n) (2 + n) (3 + n) (4 + n) (5 + n)

Binomial[n, k]/Binomial[n, k-1]
1 - k + n
---------
    k
Sum[Binomial[n, k]*Binomial[n, n - k], {k, 0, n}]
   4^n Gamma[1/2 + n]
-----------------------
Sqrt[Pi] Gamma[1 + n]
```

Az alábbi összefüggést (ahol f tetszőleges, nemnegatív egészekre definiált függvény) sem kaptuk volna meg, ha csak a magot használtuk volna:

```
Product[f[k], {k, 0, n+1, 1}]/Product[f[k], {k, 0, n, 1}]
f[1 + n]
```

Megjegyzendő, hogy a megfelelő — összegekre vonatkozó — összefüggés nem kapható meg az `Algebra`SymbolicSum`` csomag felhasználásával.

A programcsomag behívásakor módosul a `Binomial, Factorial` és `Product` függvény definíciója.

3.6.4. Differenciaegyenletek, generátorfüggvények

Tekintsük a második *Bernoulli-polinom*ot az x helyen:

```
BernoulliB[2, x]
```

$$\frac{1}{6} - x + x^2$$

A $B_n(x)$ Bernoulli-polinomok *generátorfüggvény*e:

$$\sum_{n=0}^{\infty} B_n(x)\frac{t^n}{n!} = \frac{te^{xt}}{e^t - 1}.$$

```
Series[t Exp[x t]/(Exp[t]-1), {t, 0, 4}]
```

$$1 + (-(\frac{1}{2}) + x)t + (\frac{1}{12} - \frac{x}{2} + \frac{x^2}{2})t^2 + (\frac{x}{12} - \frac{x^2}{4} + \frac{x^3}{6})t^3 + (-(\frac{1}{720}) + \frac{x^2}{24} - \frac{x^3}{12} + \frac{x^4}{24})t^4 + O[t]^5$$

A Bernoulli-polinomok nulla helyen felvett értékei a $\mathcal{B}_n$ *Bernoulli-szám*ok:

```
{BernoulliB[20], NBernoulliB[20]}
```

$$\{-(\frac{174611}{330}), -529.124\}$$

Az $E_n(x)$ *Euler-polinom*ok generátorfüggvénye:

$$\sum_{n=0}^{\infty} E_n(x)\frac{t^n}{n!} = \frac{2e^{xt}}{e^t + 1}.$$

```
Series[2 Exp[x t]/(Exp[t] + 1), {t, 0, 4}]
```

$$1 + 2(-(\frac{1}{4}) + \frac{x}{2})t + 2(-\frac{x}{4} + \frac{x^2}{4})t^2 + 2(\frac{1}{48} - \frac{x^2}{8} + \frac{x^3}{12})t^3 + 2(\frac{x}{48} - \frac{x^3}{24} + \frac{x^4}{48})t^4 + O[t]^5$$

A generátorfüggvényből éppen azt kapjuk, mint a beépített `EulerE` függvény használatával:

```
EulerE[2, x]
```

$$-x + x^2$$

```
Expand[%%[[3, 3]]*2!] == EulerE[2, x]
True

2! Simplify[Coefficient[%%%, t^2]] == Factor[EulerE[2, x]]
True
```

Az $\mathcal{E}_n$ *Euler-szám*ok definíciója pedig:

$$\mathcal{E}_n := 2^n E_n(\frac{1}{2})$$

(Az `EulerE` függvény nem rendelkezik a `Listable` attribútummal.)

```
EulerE /@ {1, 2, 3, 4, 5, 6, 7}
{0, -1, 0, 5, 0, -61, 0}
```

A $\mathcal{G}_n$ *Genocchi-számok* definíciója:

$$\mathcal{G}_1 := 1, \quad \mathcal{G}_{2n} := 2(1 - 2^{2n})\mathcal{B}_{2n}, \quad \mathcal{G}_{2n+1} := 0.$$

```
genocchi[n_?EvenQ] := 2(1 - 2^n) BernoulliB[n]
genocchi[n_?OddQ] := 0
genocchi[1] = 1;
genocchi /@ {1, 2, 3, 4, 5, 6, 7, 8}
{1, -1, 0, 1, 0, -3, 0, 17}
```

Az első- ($S_n^{(m)}$) és másodfajú ($\sigma_n^{(m)}$) Stirling-számok polinomok kétféle előállítása közötti átváltáshoz használható együtthatók:

$$x(x-1)(x-2)\dots(x-n+1) = \sum_{m=0}^{n} S_n^{(m)} x^m$$

$$x^n = \sum_{m=0}^{n} \sigma_n^{(m)} x(x-1)(x-2)\dots(x-m+1)$$

A fenti összefüggéseket azonban úgy is felfoghatjuk, mint a Stirling-számok generátorfüggvényeinek definícióját. Leszámlálásnál is előfordulnak, ugyanis $(-1)^{n-m} S_n^{(m)}$ megadja n elem azon permutációinak számát, amelyek pontosan m számú ciklust tartalmaznak. $\sigma_n^{(m)}$ azt adja meg, hogy

hányféleképpen lehet egy n elemű halmazt felbontani m számú nem üres részhalmazra, tehát egy *halmazpartíciós* feladat megoldását adja:

```
Table[StirlingS1[5, i], {i, 5}]
{24, -50, 35, -10, 1}

Expand[Product[x-i, {i, 0, 4}]]
                2    3    4    5
24x - 50x  + 35x  - 10x  + x
```

Eddig jutottunk belső függvények alkalmazásával. Most külső függvényekkel először differenciaegyenleteket oldunk meg, s a végén ismét visszakanyarodunk a generátorfüggvényekhez.

A `DiscreteMath`RSolve`` programcsomag eljárásait használhatjuk differenciaegyenletek megoldására. Az itt található külső függvények a következők:

```
ExponentialGeneratingFunction
ExponentialPowerSum
GeneratingFunction
PowerSum
SeriesTerm
```

Egy összetettebb példát mutatunk:

```
<<DiscreteMath`RSolve`
RSolve[{a[n+1] - 3b[n] - 4a[n] == 1,
   a[n+1] + b[n+1] + b[n] == n,
   a[0] == b[0] == 0}, {a[n], b[n]}, n]
                                  n      n
{{a[n] -> -(4/3) - (-2)/6 + 32/2 - n,
                        n
   b[n] -> 2/3 + (-2)/3 - 2  + n}}
```

A fenti egyenlet állandó együtthatós lineáris egyenlet volt. Meg tudunk azonban oldani bizonyos változó együtthatós egyenleteket is. Egyrészt olyanokat, amelyek homogének, elsőrendűek és együtthatóik racionális törtfüggvények. (Születési-halálozási folyamatok stacionárius eloszlása például igen gyakran ilyen differenciaegyenletnek tesz eleget.) Másrészt olyanokat, amelyek ugyan változó együtthatósak, de a megoldás valamely valós C állandó mellett $C^n n!$-nál lassabban növekszik és a differenciaegyenlethez rendelt differenciálegyenletet a `DSolve` (belső vagy külső) függvény meg tudja oldani. Lássunk erre is egy példát:

```
RSolve[{(n+1) (n+2) a[n+2] - 2(n+1) a[n+1] - 3a[n] == 0,
   a[0] == a[1] == 2}, a[n], n]
```

$$\{\{a[n] \to \frac{(-1)^n}{n!} + \frac{3^n}{n!}\}\}$$

Ha egy differenciaegyenletben a nemlinearitást konvolúció okozza, akkor is reménykedhetünk:

```
RSolve[c[n+1] == Sum[c[k] c[n-k], {k, 0, n}], c[0] == 1,
   c[n], n]
```

$$\{\{c[n] \to \frac{\text{Binomial}[2\ n,\ n]}{1 + n}\}\}$$

Eredményül éppen a Catalan-féle számokat kaptuk.

Most meghatározzuk az $a_n := t^n \quad n \in \mathbf{N}_0$ sorozat generátorfüggvényét:

```
PowerSum[t^n, {z, n, 0}]
```

$$\frac{1}{1 - t\ z}$$

A Poisson-eloszlás generátorfüggvénye pedig így kapható meg:

```
PowerSum[(t^n E^(-t))/n!, {z, n, 0}]
```

$$E^{-t + t\ z}$$

Ismert generátorfüggvényből a megfelelő sorozat tagjait így állíthatjuk elő:

```
CoefficientList[Normal[Series[%, {z, 0, 5}]], z]
```

$$\{E^{-t}, \frac{t}{E^t}, \frac{t^2}{2\ E^t}, \frac{t^3}{6\ E^t}, \frac{t^4}{24\ E^t}, \frac{t^5}{120\ E^t}\}$$

Most határozzuk meg az $a_n := n^2$ sorozat

$$z \mapsto \sum_{n=0}^{\infty} n^2 z^n / n!$$

exponenciális generátorfüggvényét:

```
ExponentialPowerSum[n^2, {z, n, 0}]
```

$$E^z\ z\ (1 + z)$$

A sorozat definíciójában bizonyos *feltétel*ek is szerepelhetnek:

```
ExponentialPowerSum[
   n^2 + 4n If[Even[n], 2^n, 3^n] + 1, {z, n, 0}]
```

$$E^z + E^z\ z\ (1 + z) + 4\ \left(\frac{3z}{2E^{3z}} - \frac{z}{E^{2z}} + E^{2z}\ z + \frac{3E^{3z}\ z}{2}\right)$$

Az itt előforduló `Even` külső függvény csak abban különbözik az `EvenQ` belső függvénytől, hogy szimbolikus argumentumra nem értékelődik ki.

Megkaphatjuk egy differenciaegyenlettel adott sorozat (például a Fibonacci-sorozat) generátorfüggvényét is:

```
GeneratingFunction[{a[n] == a[n-1] + a[n-2] /; n >= 2,
   a[0] == a[1] == 1}, a[n], n, z]
```

$$\left\{\left\{\frac{1}{1 - z - z^2}\right\}\right\}$$

Hasonlóan kaphatjuk meg egy differenciaegyenlettel adott sorozat (például a Bernoulli-számok sorozata) exponenciális generátorfüggvényét:

```
ExponentialGeneratingFunction[
Sum[Binomial[n, k]B[k], {k, 0, n}] == B[n] +
   If[n == 1, 1, 0], B[n], n, z]
```

$$\left\{\left\{\frac{z}{-1 + E^z}\right\}\right\}$$

Állítsuk elő az exponenciális generátorfüggvényből a sorozat tagjait:

```
CoefficientList[Normal[Series[
   %[[1, 1]], {z, 0, 10}]], z]*Table[n!, {n, 0, 10}]
```

$$\left\{1, -\left(\frac{1}{2}\right), \frac{1}{6}, 0, -\left(\frac{1}{30}\right), 0, \frac{1}{42}, 0, -\left(\frac{1}{30}\right), 0, \frac{5}{66}\right\}$$

Ezek a számok valóban a Bernoulli-számok:

```
Table[BernulliB[n], {n, 0, 10}]
```

$$\left\{1, -\left(\frac{1}{2}\right), \frac{1}{6}, 0, -\left(\frac{1}{30}\right), 0, \frac{1}{42}, 0, -\left(\frac{1}{30}\right), 0, \frac{5}{66}\right\}$$

A `GeneratingFunction` által előállított generátorfüggvény z helyen vett helyettesítési értékére `Gf[a[n]][z]` néven, az `ExponentialGeneratingFunction` által előállított generátorfüggvény z helyen vett helyettesítési értékére pedig `EGf[a[n]][z]` néven hivatkozhatunk a későbbiekben.

Ha az együtthatók túl gyorsan nőnek, akkor esetleg saját magunknak kell alkalmas módszert választanunk:

```
Timing[RSolve[{T[0] == 5, 2 T[n] == n T[n-1] + 3 n! /;
   n>0}, T[n], n]]
```

$$\{674.699\ \text{Second}, \{\{T[n] \to (3 + 2(\frac{1}{2})^{n})\ n!\}\}\}$$

Ijesztgető hibaüzenetek sorát is megkapjuk ez alatt a hosszú idő alatt:

```
Timing[RSolve[{T[0] == 5, 2 T[n] == n T[n-1] + 3 n! /;
   n>0}, T[n], n, Methods -> MethodEGF]]
```

$$\{34.603\ \text{Second}, \{\{T[n] \to (3 + 2(\frac{1}{2})^{n})\ n!\}\}\}$$

Az `RSolve` és a `SeriesTerm` függvénynek számos opciója van, ezek használata megkönnyíti a generátorfüggvényekkel és differenciaegyenletekkel végzett munkát:

```
RSolve[{P[0] == 1, P[1] == x,
   (n+1) P[n+1] - (2n+1) x P[n] + n P[n-1] == 0 /;
   n > 0}, P[n], n]
{{P[n] -> LegendreP[n, x]}}

RSolve[{P[0] == 1, P[1] == x,
   (n+1) P[n+1] - (2n+1) x P[n] + n P[n-1] == 0 /;
   n > 0}, P[n], n, SpecialFunctions -> False]
```

$$\{\{P[n] \to \text{Sum}[((-2\ x)^{-n + 2K[1]}\ \text{Binomial}[K[1], n-K[1]]\ \text{Binomial}[2K[1], K[1]])/(-4)^{K[1]},\ \{K[1], 0, n\}]\}\}$$

3.6.5. Gráfok és folyamok

Gráfokkal kapcsolatban legelőször az merül fel, hogyan lehet egy „algebrai” alakban megadott gráf síkbeli reprezentációját (vagy térbeli reprezentációjának vetületét) megrajzolni.

Tegyük fel, hogy adott az élekkel összekötött csúcsok rendezett párjainak listája. (Megjegyezzük, hogy többszörös élek nincsenek megengedve.) Ekkor így járhatunk el (az eredményül kapott ábrát itt — és az alábbiakban igen gyakran — nem közöljük):

```
<<DiscreteMath`Combinatorica`
ShowGraph[FromOrderedPairs[
   {{1, 2}, {2, 3}, {3, 1}, {4, 4}}], Directed]
```

Egy másik megadási módnál minden egyes csúcshoz megadjuk, mely csúcsokkal van összekötve:

```
ShowGraph[FromAdjacencyLists[
   {{1, 2}, {3, 4}, {4, 1}, {2, 3}}], Directed]
```

A csúcsokat meg is címkézhetjük:

```
ShowLabeledGraph[
   MakeGraph[Range[3], Greater[#1, #2]&, {a, b, c}]]
```

*Nevezetes gráf*ok egyszerűen valamely függvény értékeiként adódnak. `K[n]` az n szögpontú teljes gráf:

```
ShowGraph[K[3]]

ToAdjacencyLists[K[4]]

Vertices[K[5]]
{{0.309017, 0.951057}, {-0.809017, 0.587785},
 {-0.809017, -0.587785}, {0.309017, -0.951057}, {1., 0}}
```

A legutóbbi példa mutatja, hogy a síkbeli reprezentációnál a program szabályos ötszöget használ.

Most hozzáveszünk egy élt egy csillaghoz (a csillag középpontja kapja a legnagyobb sorszámot):

```
ShowGraph[AddEdge[Star[10], {1, 2}]]
```

(A tízágú csillag középpontja a 11 számot kapja.)

A gráfokkal végzett többi szokásos *művelet*re is egy-egy függvény áll rendelkezésünkre (A pontos definíciókat illetően ismét Skiena könyvére [76] utalunk, de a [39] bevezető jegyzet és a hozzá tartozó [23] példatár tárgyalásmódja is teljesen hasonló az itteniekhez. Mivel a programcsomagok olvasható szöveges állományokban vannak, ezért amikor több definíció is forgalomban van, az állományból is kideríthető, hogy melyikre gondoltak a program készítői.):

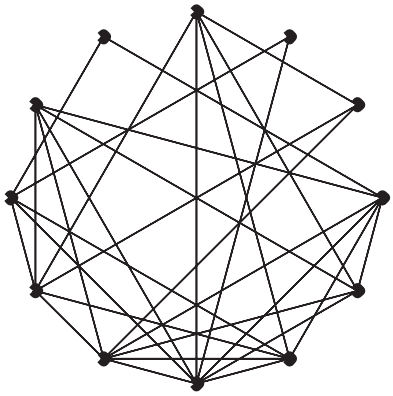
```
ShowGraph[GraphUnion[K[3], K[5, 5]]]

ShowGraph[GraphProduct[K[3], K[5]]]

ShowGraph[TransitiveClosure[FromOrderedPairs[
   {{1, 2}, {2, 3}}]], Directed]
```

Rendeljünk egy gráf éleihez csúcsokat, csatlakozó éleihez pedig az éleknek megfelelő csúcsokat összekötő éleket. A G gráfból így származtatott $L(G)$ *élgráf*ot a `LineGraph` függvény adja meg:

```
ShowGraph[LineGraph[K[5]]]
```

Most előállítunk egy véletlen gráfot, amely a teljes gráf éleinek felét tartalmazza:

```
ShowGraph[RandomGraph[12, 0.5]]
```



Mivel véletlen gráfot többféleképpen is szokás definiálni, ezért érdemes föltenni a `??RandomGraph` kérdést.

A gráfelméleti szempontból fontos *tulajdonság*ok így ellenőrizhetők:

```
ConnectedQ[DeleteEdge[Star[10], {1, 10}]]
False

SelfComplementaryQ[Cycle[5]] &&
   SelfComplementaryQ[Path[4]]
True

AcyclicQ[RandomGraph[7, 0.5, Directed], Directed]
False
```

Előállíthatjuk az összefüggő (gyengén, erősen összefüggő) komponenseket:

```
ConnectedComponents[GraphUnion[K[3], K[4]]]
{{1, 2, 3}, {4, 5, 6, 7}}
```

```
WeaklyConnectedComponents[FromAdjacencyLists[
   {{1, 2}, {3, 4}, {4, 1}, {2, 3}}], Directed]
{{1, 2, 3, 4}}
```

Fa minden éle híd:

```
Bridges[RandomTree[10]]
{{4, 5}, {4, 7}, {1, 4}, {1, 9}, {2, 8},
   {6, 8}, {3, 6}, {3, 10}, {1, 10}}
```

Fák kezelésére van egy külön programcsomag, a `DiscreteMath`Tree``, az alábbi utasításokkal:

`ExprPlot`	`TreeFind`
`MakeTree`	`TreePlot`

Meghatározhatók az Euler-féle és a Hamilton-féle körök is:

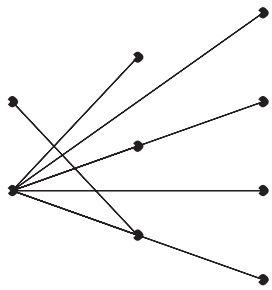
```
EulerianCycle[K[4, 4]]
{7, 2, 8, 1, 5, 4, 6, 3, 7, 4, 8, 3, 5, 2, 6, 1, 7}
```

```
HamiltonianCycle[K[3, 3]]
{1, 4, 2, 5, 3, 6, 1}
```

Hajtsuk végre a legutóbbi utasítást úgy is, hogy másodikként az `All` opcionális argumentumot is beírjuk!

Meghatározható a minimális és a maximális feszítő fa, a feszítő fák száma:

```
ShowGraph[MinimumSpanningTree[K[2, 3, 4]]]
```



```
NumberOfSpanningTrees /@ {Cycle[3], K[10], Path[6]}
{3, 100000000, 1}
```

Az 1 jelű forrástól a 4 jelű nyelőig az áram egy súlyozatlan élű irányított gráfban:

```
NetworkFlow[FromAdjacencyLists[
   {{1, 2}, {3, 4}, {1, 4}, {2, 3}}], 1, 4]
1
```

Végül fölsorolunk néhány, további búvárkodásra ösztökélő függvényt és opciót a `DiscreteMath`Combinatorica`` programcsomagból:

Automorphisms	Josephus
Backtrack	MaximumAntichain
ChromaticNumber	MaximumClique
ChromaticPolynomial	MaximumIndependentSet
Cofactor	MaximumSpanningTree
DeBruijnSequence	MinimumChainPartition
Dijkstra	MinimumSpanningTree
EdgeColoring	MinimumVertexCover
EquivalenceClasses	MultiplicationTable
EquivalenceRelationQ	PartialOrderQ
Harary	PartitionQ
HasseDiagram	Spectrum
IncidenceMatrix	TravelingSalesman
InvolutionQ	TravelingSalesmanBounds
IsomorphicQ	TriangleInequalityQ
Isomorphism	Turan

3.6.6. Gyakorlatok és feladatok

1. Vegyük 6 elem egy véletlen permutációját, majd állítsuk (véletlen) ennek megfelelő sorrendbe 3 elem összes permutációját.

2. Tanulmányozzuk az alábbi utasításokat:

```
BipartiteMatching[Cycle[6]]
MaximalMatching[K[6]]
PlanarQ[K[5]] || PlanarQ[K[3, 3]]
```

3. Hányféleképpen lehet 8 golyót fölfűzni egy nyakláncra, ha közöttük m különböző színű van?
Útmutatás. Használjuk a `Polya`-függvényt.

4. Írjuk ki a Pascal-háromszög néhány sorát, kivéve az első és utolsó elemet. Mi a sorok legnagyobb közös osztója?

3.7. Geometria

Ebben a szakaszban azt *érzékeltetjük*, hogy geometriai vizsgálatoknál hogyan lehet felhasználni a *Mathematicá*t.

A program 2.2.∗ változatai viszonylag kevés olyan belső, illetve külső függvényt tartalmaznak, amelyeket geometriai problémák megoldásánál *közvetlenül* alkalmazhatunk. A beépített grafikus, numerikus és szimbolikus eljárások összekapcsolásával írhatunk egy-egy (például koordinátageometriai) feladattípus megoldását szolgáló függvényt vagy akár programcsomagot. Ehhez sok segítséget és ötletet adhatnak a MathSource-on található programcsomagok, amelyekről a 3.7.2. pontban fogunk szólni.

3.7.1. Geometriai alakzatok

Két és háromdimenziós alakzatok ábrázolásánál a következő *grafikus elemek*et használhatjuk:

```
Circle                  Polygon
Cuboid                  Raster
Disk                    Rectangle
Line                    Text
Point
```

A fenti elemekből az alábbi eljárásokkal *grafikus objektumok*at készíthetünk:

```
ContourGraphics         Graphics3D
DensityGraphics         SurfaceGraphics
Graphics
```

amelyeket azután a `Show` függvénnyel jeleníthetünk meg a képernyőn.

Az ábrázolás módját egyrészt a grafikus opciók (például a `Graphics` függvény opciói) segítségével módosíthatjuk; másrészt az alábbi, úgynevezett *grafikus direktívák*at is használhatjuk:

```
AbsoluteDashing         GrayLevel
AbsolutePointSize       Hue
AbsoluteThickness       PointSize
CMYKColor               RGBColor
Dashing                 SurfaceColor
EdgeForm                Thickness
FaceForm
```

A grafikus direktívákat kétféleképpen alkalmazhatjuk. Ezek lehetnek a korábban tárgyalt grafikus függvények opcióinak értékei.

A grafikus objektumokat generáló függvények első argumentuma mindig egy olyan lista, amelyben grafikus direktívák (esetleg üres) sorozata és grafikus elemek felváltva követik egymást. A többi argumentumba esetenként opciókat írhatunk. Például:

```
Show[Graphics[{
          RGBColor[1, 0, 0], Thickness[0.02],
       Line[{{-5/2, -1/2}, {2, 2}}],
          RGBColor[0.5, 1, 0.25], Thickness[0.01],
       Line[{{-5/2, -3/2}, {2, 1}}],
          RGBColor[1, 0, 1], PointSize[0.01], Point[{-1, 2}]},
     PlotRange -> {{-3, 3}, {-3, 3}}, Axes -> True,
     AspectRatio -> Automatic]]
```

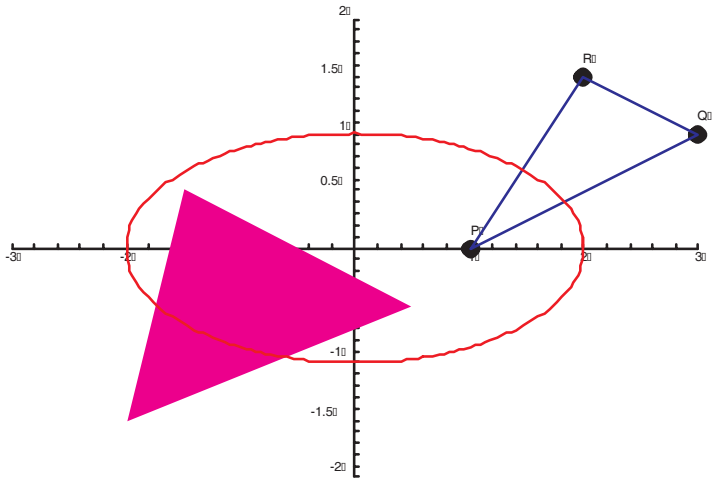
Megemlítjük még azt is, hogy az adott adatok által meghatározott koordináta-rendszertől függetlenül a képernyő pontjaira is hivatkozhatunk a `Scaled` belső függvénnyel. Ez a lehetőség jól használható például feliratok készítésénél.

A fenti függvények alkalmazásának módjára és a lehetőségek illusztrálására tekintsük a következő függvényt, amellyel kényelmesen ábrázolhatunk *síkbeli* geometriai alakzatokat:

```
Rajzol[elemek_List, {xMin_, xMax_}, {yMin_, yMax_}] :=
   Block[{l = elemek},
      l = l /. {Point[{a_, b_}], nev_} :>
         Apply[Sequence,
            {Point[{a, b}], Text[nev, {a, b}, {-1, -1}]}];
      PrependTo[l, PointSize[.02]];
      Show[Graphics[l, {AspectRatio -> Automatic,
           PlotRange -> {{xMin, xMax}, {yMin, yMax}},
           Axes -> True}]]]
```

A függvény működése kielemezhető az alábbi példából:

```
P = {1, 0}; Q = {3, 1}; R = {2, 3/2};
Rajzol[{{Point[P], "P"}, {Point[Q], "Q"}, {Point[R], "R"},
   RGBColor[1, 0, 1],
   Polygon[{{-3/2, 1/2}, {-2, -3/2}, {1/2, -1/2}}],
   RGBColor[0, 0, 1], Line[{P, Q, R, P}],
   RGBColor[1, 0, 0], Thickness[0.002], Circle[{0,0}, {2,1}]},
     {-3, 3}, {-2, 2}]
```

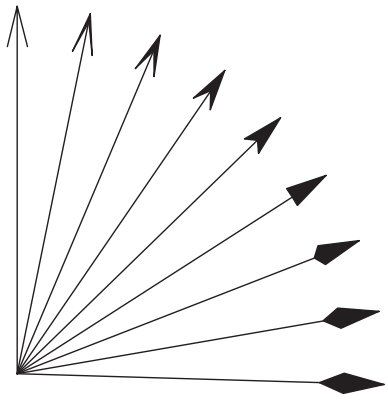


Vektorok ábrázolásánál a `Graphics`Arrow`` programcsomagban meglévő, számos opcióval rendelkező

`Arrow`

függvényt használhatjuk. Például:

```
<<Graphics`Arrow`
Show[Graphics[Table[Arrow[{0, 0}, {Sin[x], Cos[x]},
     HeadCenter -> x, HeadLength -> 0.1], {x, 0, 1.6, 0.2}]],
     PlotRange -> {{-0.1, 1}, {-0.2, 1.1}},
     AspectRatio -> Automatic]
```



Szabályos sokszögek és *szabályos testek* ábrázolásához és geometriai adataiknak meghatározásához a

```
Geometry`Polytopes`
```

programcsomag függvényei adhatnak segítséget.

A felhasználható szabályos sokszögek a következők:

```
Decagon                 Octagon
Digon                   Pentagon
Dodecagon               Square
Heptagon                Triangle
Hexagon                 Undecagon
Nonagon
```

Az ötféle szabályos konvex test elnevezése:

```
Cube                    Octahedron
Dodecahedron            Tetrahedron
Icosahedron
```

Az alábbi gegometriai adatok meghatározására van mód:

```
Area                    Faces
Circumscribed           Inscribed
Coords                  Schlafli
Dual                    Vertices
Edges                   Volume
```

Szabályos testek tanulmányozásánál felhasználhatjuk még a

```
Graphics`Polyhedra`
```

programcsomag eljárásait is. Itt az alábbi testeket nevezhetjük meg:

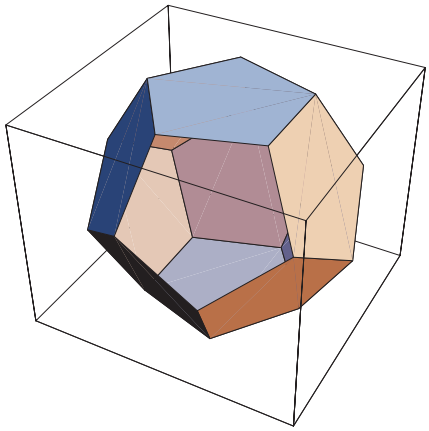
```
Cube
Dodecahedron
GreatDodecahedron
GreatStellatedDodecahedron
GreatIsocahedron
Hexahedron
Icosahedron
Octahedron
SmallStellatedDodecahedron
Tetrahedron
```

A különböző stílusú megjelenítést segítik a következő függvények:

`Geodesate`	`Stellate`
`OpenTruncate`	`Truncate`
`Polyhedron`	

Felhívjuk az Olvasó figyelmét arra, hogy ezek a függvények eleve grafikus objektumot készítenek az adott testről, a megjelenítésükhöz tehát csak a `Show` függvényt kell alkalmaznunk:

```
Show[OpenTruncate[Polyhedron[Dodecahedron], 0.4]]
```



Végül megemlítjük, hogy a

```
Geometry`Rotations`
```

programcsomagban lévő

`Rotate2D`	`RotateMatrix2D`
`Rotate3D`	`RotateMatrix3D`

függvények alakzatok elforgatásánál lehetnek hasznosak. Például:

```
<<Geometry`Rotations`
RotationMatrix2D[Pi/4] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{\text{Sqrt}[2]} & \frac{1}{\text{Sqrt}[2]} \\ -\frac{1}{\text{Sqrt}[2]} & \frac{1}{\text{Sqrt}[2]} \end{pmatrix}$$

3.7.2. További lehetőségek

Külön programcsomag áll a felhasználó rendelkezésére abból a célból, hogy geometriai problémák megoldásának hatékony algoritmusait tanulmányozhassa:

```
DiscreteMath`ComputationalGeometry`
```

Függvényei a következők:

```
ConvexHull                NearestNeighbor
DelaunayTriangulation     PlanarGraphPlot
DelaunayTriangulationQ    TriangularSurfacePlot
DiagramPlot               VoronoiDiagram
```

A továbbiakban a MathSource-ról megszerezhető programcsomagokra szeretnénk felhívni az érdeklődő Olvasó figyelmét.

Elsőként megemlítjük a 0202–509 szám alatt található

```
MathDraw
```

csomagot, amellyel kényelmesen tudjuk rajzolni az összes körzővel és vonalzóval megszerkeszthető alakzatot.

A 0205–175 szám alatt csak *demonstrációs* anyagot találunk a

```
Descartes
```

programcsomagról, amely a szokásos alapműveleteken túl képes analitikus vizsgálatok alapján eldönteni olyan tulajdonságok meglétét, mint: illeszkedés, érintkezés, párhuzamosság, koncentrikusság stb. Ezen eszközök felhasználásával elemi geometriai tételek bizonyítása (lásd [90]) és geometriai transzformációk hatása szemléltethető.

A 0205–401 szám alatti

```
MapPoint
```

program többszáz függvényt tartalmaz a klasszikus euklideszi geometria és a modern dinamikus geometria alakzatainak ábrázolására és animálására a síkon és a térben.

Négydimenziós alakzatok háromdimenziós metszeteit a 205–108 szám alatt megtalálható

```
4DKnife
```

programcsomag függvényeivel jeleníthetjük meg.

3.8. Lineáris algebra

A lineáris algebrához kapcsolható eljárások vektorok és mátrixok *szimbolikus* és *numerikus* kezelésére képesek. A felhasználó igen egyszerűen, a bemenő adatok alkalmas megadásával választja ki a kívánt kezelési módot.

Ha a vektorok vagy mátrixok elemei szimbolikus kifejezések vagy pontos numerikus értékek, akkor a program a kért műveleteket számos beépített matematikai azonosság felhasználásával *szimbolikusan* végzi el, és igyekszik megadni a pontos eredményt.

Ha a vektorok vagy mátrixok elemei valós vagy komplex számok és ezek közül legalább az egyik közelítő numerikus érték (amit a tizedespont explicit kiírásával jelzünk), akkor a program minden számot ilyenre alakít át, és a műveletek elvégzésére *numerikus módszert használva* adja meg a pontos eredmény egy közelítő értékét.

Itt is kiemeljük azt a tényt, hogy a legtöbb beépített függvény argumentumában vektor vagy mátrix (általában lista) is szerepelhet. Ebben az esetben a program a szóban forgó függvényt a vektor vagy a mátrix minden elemére külön-külön alkalmazza, ha a függvény rendelkezik a `Listable` attribútummal.

3.8.1. Vektorok és mátrixok megadása

Néhány programnyelvben a vektorok és a mátrixok különböző típusú objektumok. A *Mathematica* ezeket egységes módon, az egyik legalapvetőbb beépített adattípusával, nevezetesen *listá*val ábrázolja.

A továbbiakban a `List[x, y]` utasítással egyenértékű `{x, y}` formát fogjuk használni.

`{x, y}`	az (x, y) vektor
`{{a, b, c}, {d, e, f}}`	az $\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$ 2×3-as mátrix

Vektorok, illetve mátrixok elemei lehetnek valós vagy komplex számok, matematikai kifejezések, sőt akár ábrák is.

Mivel a „vektor” kifejezést a matematikában is különböző fogalmak jelölésére szokás használni, ezért érdemes tisztázni azt, hogy a program milyen objektumot tekint vektornak. Erről a logikai értéket adó

`VectorQ`

belső függvény tájékoztatja a felhasználót. A *Mathematicá*ban *vektor*on olyan listát értünk, amelynek egyik eleme sem lista:

```
{VectorQ[{1, Sqrt[2]}], VectorQ[{x, Sin[y]}]}
{True, True}

{VectorQ[{{1}, 2}], VectorQ[{{1}, {2}}]}
{False, False}
```

Ez a vektorfogalom a matematikában szereplő *rendezett n-es* (ilyenek például az $\mathbf{R}^n$ tér elemei) fogalmával analóg.

A program *mátrix*nak olyan listát tekint, amelynek minden eleme ugyanakkora hosszúságú vektor (ezek a mátrix sorai). Ezt az általános érvényű megállapodást a

`MatrixQ`

függvénnyel ellenőrizhetjük:

```
{MatrixQ[{{1, 2}, {3, 4}}], MatrixQ[{{x}, {y}}]}
{True, True}

{MatrixQ[{x, y}], MatrixQ[{{1}, {2, 3}}]}
{False, False}
```

Oszlopvektort, tehát például az $\mathbf{R}^{4\times 1}$ tér egy elemét így:

```
{{1}, {2}, {3}, {4}}
{{1}, {2}, {3}, {4}}
```

sorvektort, például az $\mathbf{R}^{1\times 4}$ tér egy elemét pedig következő módon adhatunk meg:

```
{{1, 2, 3, 4}}
{{1, 2, 3, 4}}
```

A korábban mondottaknak megfelelően a *Mathematica* ezeket a kifejezéseket nem vektoroknak, hanem mátrixoknak tekinti.

A `ColumnForm` függvény valamely vektor komponenseit oszlopba rendezve jeleníti meg a képernyőn:

```
ColumnForm[{x, y}]
x
y
```

Mátrixokat a szokásos formában a `MatrixForm` függvénnyel ábrázolhatunk. Például:

```
m = {{1, 2, 3, 4}, {0, 2, 3, 4}, {1, 5, 6, 7}};
MatrixForm[m]
1  2  3  4
0  2  3  4
1  5  6  7
```

A `MatrixForm` függvényt posztfix alakban (lásd a 3.1.4. pontot) is használhatjuk:

```
mforma = m // MatrixForm
```

Felhívjuk azonban az Olvasó figyelmét arra, hogy a fenti utasítás hatására a program az `mforma` változóban nem az `m` mátrixot (tehát listák listáját) tárolja, hanem a `MatrixForm[m]` kifejezést, ami az `m` mátrix „táblázatos alakja". Kihangsúlyozzuk azt is, hogy az utóbb megemlített két függvény csupán a képernyőn való *megjelenítés* eszközei.

Vektor és mátrix *dimenzióját* a

`Dimensions`

belső függvénnyel kapjuk meg. Például:

```
{Dimensions[{x, y}], Dimensions[m]}
{{2},  {3, 4}}
```

Figyeljük meg, hogy mátrix esetében az eredmény olyan lista, amelynek az első eleme a sorok, a második eleme pedig az oszlopok száma.

Adott lista elemeinek számát a

`Length`

beépített függvény adja meg. Ha `v` vektor, akkor a `Length[v]` utasítás eredménye `v` komponenseinek a száma, ha `v` mátrix, akkor `Length[v]` `v` sorainak a száma.

A `Table` és az `Array` függvények segítségével is definiálhatunk vektorokat és mátrixokat.

• **A `Table` függvény**

A `Table` függvénnyel kényelmesen adhatunk meg olyan vektorokat, illetve mátrixokat, amelyeknek a komponensei matematikai kifejezések. A

```
Table[kifejezes, {i, imin, imax, di}]
```

utasítás eredménye olyan vektor, amelynek komponensei a `kifejezes` `i` iterációs változóhoz tartozó értékei, ahol `i` `imin`-től `imax`-ig halad `di` lépésközzel.

Az `imin`, az `imax` és a `di` iterációs paraméter nem feltétlenül egész szám. A program `i`-t `imin`-től `di` lépésközzel addig növeli, amíg a következő értéke `imax`-nál nagyobb nem lesz. Az iterációs paraméterek szimbolikus kifejezések is lehetnek. Az egyetlen megkötés az, hogy `(imax-imin)/di` valós szám legyen.

A `Table` függvény bizonyos argumentumai elhagyhatók. Ilyenkor a program az alapértelmezés szerinti értékeket használja. A lépésköz (`di`) és a kezdőérték (`imin`) alapértelmezés szerinti értéke 1, ennek megfelelően a következő formákat is használhatjuk:

```
Table[kifejezes, {i, imin, imax}]
Table[kifejezes, {i, imax}]
```

Megemlítjük még, hogy a

```
Table[kifejezes, {imax}]
```

utasítás eredménye olyan `imax` dimenziójú vektor (lista), amelynek mindegyik komponense `kifejezes`.

A `Table` függvénnyel mátrixot például így adhatunk meg:

```
Table[kif, {i,imin,imax,di}, {j,jmin,jmax,dj}]
```

Az `{i,imin,imax,di}` lista a mátrix soraira, `{j,jmin,jmax,dj}` pedig a mátrix oszlopaira vonatkozó információkat tartalmazza:

```
Table[f[i, j], {i, 1, 3}, {j, 1, 5}]
{{f[1,1], f[1,2], f[1,3], f[1,4], f[1,5]},
 {f[2,1], f[2,2], f[2,3], f[2,4], f[2,5]},
 {f[3,1], f[3,2], f[3,3], f[3,4], f[3,5]}}
```

```
MatrixForm[%]
f[1,1]   f[1,2]   f[1,3]   f[1,4]   f[1,5]
f[2,1]   f[2,2]   f[2,3]   f[2,4]   f[2,5]
f[3,1]   f[3,2]   f[3,3]   f[3,4]   f[3,5]
```

Az iterációs paraméterek megadására vonatkozó fentebb ismertetett szabályok ebben a formában is érvényesek.

• Az `Array` függvény

A `Table` függvényhez hasonló a kevésbé rugalmas, de sok esetben igen kényelmesen használható `Array` függvény. Használata akkor lehet célszerű, ha az elemek egy olyan függvény értékei, amelyre csak nevével akarunk hivatkozni, szemben az előző esettel, ahol a függvényérték „általános" alakját adtuk meg.

`Array[f, n]`	az $(f[1], f[2], \ldots, f[n])$ vektor
`Array[f, {n1, n2}]`	az $\begin{pmatrix} f[1,1] & \ldots & f[1,n_2] \\ \ldots & & \\ f[n_1,1] & \ldots & f[n_1,n_2] \end{pmatrix}$ mátrix.

Az n, n_1 és n_2 paraméternek tetszőleges nemnegatív egész számnak kell lennie. Az `Array` függvény ebben a formában tehát az iterációs változót 1-től egyesével lépteti. Ha az n számú elemet a *kezd* (tetszőleges komplex) argumentumtól egyesével lépkedve akarjuk felsorolni, akkor a következő utasításokat használhatjuk:

```
Array[f, n, kezd]
Array[f, {n1, n2}, kezd]
```

Az elmondottakat az alábbi példán illusztráljuk:

```
negyzetszamok[x_] := x^2
Array[negyzetszamok, 5]

{1, 4, 9, 16, 25}
```

A következő egészen mást ad:

```
Array[negyzetszamok[x], 5]
{(x^2)[1], (x^2)[2], (x^2)[3], (x^2)[4], (x^2)[5]}
```

Mivel az `Array` argumentumában a *függvényt*, és nem annak *helyettesítési értékét* kell megadni, ezért itt felhasználhatjuk a *Mathematica* tiszta függvény-fogalmát (lásd a 3.1.4. pontot) is:

```
Array[#^2&, 5]
{1, 4, 9, 16, 25}
```

Érdemes kipróbálni a következő utasításokat is:

```
Array[negyzetszamok, 10, I]
Array[f, 3, 4]
Array[f, 3, 1, Plus]
Array[f, {3, 4}] // MatrixForm
Array[f, {2, 3}, -1]
Array[#1^#2&, {2, 2}]
MatrixForm[%]
```

• Speciális mátrixok

Számos speciális alakú mátrix közvetlen megadására van mód:

```
IdentityMatrix[3]
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

DiagonalMatrix[{a, b, c}] // MatrixForm
a  0  0
0  b  0
0  0  c
```

A `LinearAlgebra`MatrixManipulation`` programcsomag további speciális mátrixok értelmezését tartalmazza. Olvassuk be ezt a programcsomagot:

```
<<LinearAlgebra`MatrixManipulation`
```

és figyeljük meg a következő utasítások eredményét:

```
SquareMatrixQ[{{1, 2, 3}, {4, 5, 6}}]
ZeroMatrix[3] // MatrixForm
ZeroMatrix[3, 6]
UpperDiagonalMatrix[f, 3]
LowerDiagonalMatrix[#1+#2&, 3]
MatrixForm[%]
HilbertMatrix[3]
```

```
HilbertMatrix[2, 4]
HankelMatrix[4] // MatrixForm
HankelMatrix[{a, b, c, d}] // MatrixForm
```

Ezeket a mátrixokat a *Mathematica* egyéb függvényeivel is egyszerűen definiálhatjuk. A továbbiakban az eredmények közlése nélkül mutatunk néhány ilyen példát.

Alsó háromszögmátrixot például így is megadhatunk:

```
alsoharomszog[n_] := Table[
   If[i>=j, a[i, j], 0], {i, n}, {j, n}]
MatrixForm[alsoharomszog[3]]
```

Hilbert-mátrixokat a `Table` függvénnyel definiálhatunk:

```
hilbert[n_] := Table[1/(i+j-1), {i, 1, n}, {j, 1, n}]
hilbert[3] // MatrixForm
```

A következő utasítás eredménye szintén harmadrendű Hilbert-mátrix:

```
1/(Outer[Plus, Range[3], Range[3]-1])
```

Tridiagonális mátrixot a feltételeket tartalmazó kifejezések megadásánál jól használható `Switch` belső függvénnyel adhatunk meg. Például így:

```
tridiagonalis = Table[
   Switch[i-j, -1, a, 0, b, 1, c, _, 0], {i, 5}, {j, 5}]
MatrixForm[tridiagonalis]
```

A következő utasítás eredménye egy 5×5-ös Vandermonde-mátrix:

```
vandermonde = Array[x[#1]^(#2-1)&, {5, 5}]
MatrixForm[%]
```

amelyre vonatkozó ismert összefüggést így illusztrálhatjuk:

```
Factor[Det[vandermonde]]
```

3.8.2. Részmátrixok kezelése

A 2.3.1. pontban ismertetett módon hivatkozhatunk mátrix részeire (elem, sor, oszlop), illetve módosíthatjuk az egyes részeket.

`m[[i, j]]`	az m mátrix (i,j) indexű eleme
`m[[i]]`	az m mátrix i-edik sora
`Transpose[m][[j]]`	az m mátrix j-edik oszlopa

A `Transpose[m]` az m (tetszőleges téglalap-) mátrix transzponáltját adja meg. Az m mátrix j-edik oszlopát még így is megkaphatjuk:

```
Map[#[[j]]&, m]
```

Az m mátrix $i_1, \cdots, i_r$ indexű sorainak és $j_1, \cdots, j_s$ indexű oszlopainak közös részeként adódó $r \times s$ méretű részmátrixát a következő utasítással jelölhetjük ki:

```
m[[{i1,...,ir}, {j1,...,js}]]
```

Az elmondottakat az alábbi példákon mutatjuk be:

```
Clear[m]
m = {{2, 3, 5, 1}, {4, 7, 0, a}, {x, 2, 1, 7}};
MatrixForm[m]
```

```
2  3  5  1
4  7  0  a
x  2  1  7
```

```
m[[3, 2]]
```

```
2
```

```
m[[1]]
```

```
{2, 3, 5, 1}
```

Változtassuk most meg az m mátrix $(3,2)$ indexű elemét és az első sorát:

```
m[[3, 2]] = -1;   m[[1]] = {0, 3, -2, 7};
```

és ellenőrizzük az eredményt:

```
MatrixForm[m]
```

```
0   3  -2   7
4   7   0   a
x  -1   1   7
```

Az alábbi utasítások mindegyike az m mátrix harmadik oszlopát adja meg:

```
Transpose[m][[3]]
{-2, 0, 1}

Map[#[[3]]&, m]
{-2, 0, 1}
```

Az m mátrix egy részmátrixát így jelölhetjük ki:

```
m[[{1, 2}, {2, 3}]] // MatrixForm
3  -2
7   0
```

Ezt a részmátrixot még így is megkaphatjuk:

```
m[[Range[1, 2], Range[2, 3]]]
```

A `LinearAlgebra`MatrixManipulation`` programcsomag több lehetőséget is tartalmaz blokkmátrixok kényelmes kezelésére. Olvassuk be szükség esetén ezt a programcsomagot, ezután próbáljuk ki a következő utasításokat:

```
A = {{a11, a12}, {a21, a22}}; MatrixForm[A]
B = {{b11, b12}, {b21, b22}}; MatrixForm[B]
AppendColumns[A, B]; MatrixForm[%]
AppendRows[A, B] // MatrixForm
BlockMatrix[{{A, B}, {B, {{0, 0}, {0, 0}}}}]
MatrixForm[%]
mat = Array[c, {3, 4}]; MatrixForm[mat]
TakeRows[mat, -2] // MatrixForm
TakeColumns[mat, {2, 3}] // MatrixForm
TakeMatrix[mat, {2, 3}, {3, 4}]
SubMatrix[mat, {2, 3}, {2, 2}]
```

3.8.3. Műveletek vektorokkal és mátrixokkal

Ebben a pontban az aritmetikai műveletekkel, mátrixnak a determinánsával, a rangjával és az inverzével, valamint mátrixváltozós függvényekkel foglalkozunk.

• Aritmetikai műveletek

Egyenlő dimenziójú vektorok, illetve mátrixok *összegét* a matematikában megszokott módon kapjuk meg:

```
v1 = {a, b, c}; v2 = {x, y, z};
v1 + v2
{a + x, b + y, c + z}

A = {{1, a, b}, {2, 3, d}}; B = {{c, d, e}, {f, g, h}};
A + B // MatrixForm
1 + c   a + d   b + e
2 + f   3 + g   d + h
```

A `v` vektornak, illetve az `A` mátrixnak a `c` *számmal vett szorzatát* a

`c v` vagy a `c*v`

illetve

`c A` vagy a `c*A`

utasítással számolhatjuk ki.

Egyenlő dimenziójú vektorok *skaláris szorzatának* meghatározásához a `Dot` belső függvényt (ennek rövid alakja a `.` karakter) használhatjuk fel. Ha a `v1` és a `v2` vektor mindegyik komponense valós szám vagy valós szimbólum, akkor ezek skaláris szorzata:

```
v1 . v2
a x + b y + c z
```

(Komplex esetben a `Dot` függvény nem veszi a konjugáltat.)

Háromdimenziós vektorok *vektoriális szorzat*át a `Cross` függvénnyel számíthatjuk ki. Ennek használatához először be kell hívni a

```
<<LinearAlgebra`CrossProduct`
```

programcsomagot. A `v1` és `v2` vektor vektoriális szorzata:

```
Cross[v1, v2]
{-(c y) + b z, c x - a z, -(b x) + a y}
```

A `Cross` függvény ismeri és alkalmazni is tudja a skaláris és a vektoriális szorzat alapvető tulajdonságait.

Mátrixok szorzatára a `Dot` függvény ugyanazt az eredményt adja, amit a matematikában sor-oszlopszorzással szoktunk megkapni. Ez azt jelenti, hogy a *Mathematica* kiszámolja az A és a B mátrix szorzatát akkor, amikor az A mátrix oszlopainak a száma (a `Dimensions[A]` által megadott

lista utolsó eleme) megegyezik a B mátrix sorainak a számával (azaz a `Dimensions[B]` által megadott lista első elemével). Próbáljuk ki például a következő utasításokat:

```
Clear[A, B]
A = {{5, -3}, {2, 1}};  B = {{1, 4}, {2,-3}};
(A + B) . (A - B)
A . A + 2*A . B + B . B
```

A *Mathematica* sajátos módon kezeli *vektor és mátrix szorzatát*. Sok esetben a matematikában megszokott sor-oszlopszorzásnak megfelelően értelemszerűen tekint egy vektort sorvektornak vagy oszlopvektornak. Ha például

```
Clear[A, v]
A = {{a, b, c}, {d, e, f}};   v = {x, y};
```

akkor a program meghatározza a `v.A` szorzatot (ekkor v-t sorvektornak tekinti):

```
v . A
{a x + d y, b x + e y, c x + f y}
```

az `A.v` utasításra azonban hibaüzenetet küld. Másrészt

```
Clear[v]
v = {x, y, z}
A . v
{a x + b y + c z, d x + e y + f z}
```

(ebben az esetben a program `v`-t oszlopvektornak veszi), és most a `v.A` utasításra kapunk hibaüzenetet. Néhányszor azonban matematikai szempontból értelmetlen eredményt is kaphatunk. Tekintsük például a következő egyenlőséget:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \end{pmatrix} (\,5 \quad 6\,) \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 391 & 578 \\ 897 & 1326 \end{pmatrix}.$$

Nem kapjuk meg az elvárt eredményt akkor, ha a két belső tényezőt vektorként definiáljuk:

```
{{1, 2}, {3, 4}} . {5, 6} . {5, 6} . {{1, 2}, {3, 4}}
319 . {{1, 2}, {3, 4}}
```

Az ilyen esetekben célszerű megjelölni azt, hogy melyik vektort tekintjük sorvektornak és melyiket oszlopvektornak:

```
{{1, 2}, {3, 4}}.{{5}, {6}}.{{5, 6}}.{{1, 2}, {3, 4}}
{{391, 578}, {897, 1326}}
```

Ezzel a módszerrel számolhatjuk ki a *Mathematicá*val két vektor *diadikus szorzatát*, azaz egy oszlopvektornak és egy sorvektornak a mátrix-szorzatát:

```
{{x}, {y}} . {{a, b, c}} // MatrixForm
a x   b x   c x
a y   b y   c y
```

Az `Outer` belső függvénynek és a `BlockMatrix` külső függvénynek (ez a `LinearAlgebra`MatrixManipulation`` programcsomagban található) felhasználásával adhatunk meg olyan új eljárást, ami két mátrix *Kronecker-szorzatát* határozza meg:

```
<<LinearAlgebra`MatrixManipulation`
KroneckerSzorzat[a_, b_] :=
        BlockMatrix[Outer[Times, a, b]]
```

Tekintsük ezután a következő mátrixokat:

```
A = Array[a, {2, 2}]
{{a[1, 1], a[1, 2]}, {a[2, 1], a[2, 2]}}

B = Array[b, {2, 2}]
{{b[1, 1], b[1, 2]}, {b[2, 1], b[2, 2]}}
```

Az indexek ügyesebb elhelyezésével áttekinthetőbb eredményt így kaphatunk:

```
KroneckerSzorzat[A, B] /.
      {a[i_, j_] -> Subscripted[a[SequenceForm[i,j]]],
       b[i_, j_] -> Subscripted[b[SequenceForm[i,j]]]}
    // MatrixForm
a11 b11   a11 b12   a12 b11   a12 b12
a11 b21   a11 b22   a12 b21   a12 b22
a21 b11   a21 b12   a22 b11   a22 b12
a21 b21   a21 b22   a22 b21   a22 b22
```

Négyzetes mátrix pozitív egész kitevőjű *hatványát* a

`MatrixPower`

függvénnyel határozhatjuk meg. Tekintsük például a következő $A \in \mathbf{R}^{2\times 2}$ mátrixot:

```
A = {{1, 1}, {-1, 3}}
MatrixForm[A]
 1  1
-1  3
```

Ennek a négyzetét, azaz az $A^2 \in \mathbf{R}^{2\times 2}$ mátrixot így számolhatjuk ki:

```
MatrixPower[A, 2] // MatrixForm
 0  4
-4  8
```

Vigyázzunk azonban arra, hogy az `A^2` utasítást (ennek teljes alakja: `Power[A, 2]`) is használhatjuk. Mivel a `Power` függvény rendelkezik a `Listable` attribútummal, ezért a szóban forgó utasítás eredménye nem az A mátrix négyzete, hanem az elemek négyzetéből álló mátrix:

```
A^2 // MatrixForm
1  1
1  9
```

• Négyzetes mátrix determinánsa

Az `A` négyzetes mátrix determinánsát így kaphatjuk meg:

```
Det[A]
```

A $\mathbf{Z}_p$ véges számtestben is dolgozhatunk, ha a

```
Det[A, Modulus -> p]
```

utasítást használjuk. A `Modulus` opció alapértelmezés szerinti értéke 0. Ekkor a program a $\mathbf{C}$ számtestben végzi el a műveleteket.

A következő példákban a mátrix elemei pontos numerikus értékek, illetve szimbólumok, ezért a determináns *pontos* értékét kapjuk meg:

```
Det[{{1/3, -5/2, 2/5, 3/2}, {3, -12, 21/5, 15},
     {2/3, -9/2, 4/5, 5/2}, {-1/7, 2/7, -1/7, 3/7}}]
```

$$\frac{1}{35}$$

```
A = {{a, b, c, d}, {-b, a, d, -c},
   {-c, -d, a, b}, {-d, c, -b, a}};
Det[A];
```

```
Factor[%]
```

$$(a^2 + b^2 + c^2 + d^2)^2$$

Az alábbi példában a mátrix egyik eleme közelítő numerikus érték. A `Det` függvény ebben az esetben a kért eredménynek egy *közelítő értékét* határozza meg:

```
B={{Sqrt[2.], Sqrt[3], Sqrt[5], Sqrt[3]},
   {Sqrt[6], Sqrt[21], Sqrt[10], -2 Sqrt[3]},
   {Sqrt[10], 2 Sqrt[15], 5, Sqrt[6]},
   {2, 2 Sqrt[6], Sqrt[10], Sqrt[15]}};
Det[B]
9.04581
```

A program a fenti eredményt gépi pontosságú számok (lásd a 3.1.3. pontot) felhasználásával számította ki. Emlékeztetünk arra, hogy a többi értékes jegyet így jeleníthetjük meg:

```
InputForm[%]
9.04580658046875
```

Ha ennél több értékes jegyre szeretnénk a determinánst meghatározni, akkor az `N` függvényt kell alkalmaznunk.

• Mátrix rangja

Adott mátrix rangjának meghatározásához a *Mathematica* több beépített függvényét is felhasználhatjuk.

Az $A \in \mathbf{C}^{n\times m}$ $(n, m \in \mathbf{N})$ *mátrix rangja* az r $(\leq \min\{n, m\})$ természetes szám, ha A-nak van r-edrendű, nullától különböző aldeterminánsa, de minden $(r+1)$-edrendű aldeterminánsa (ha egyáltalán van ilyen) nulla.

A fenti tulajdonságokkal rendelkező r számot a `Minors` függvény segítségével kereshetünk. A

```
Minors[A, k]
```

utasítás eredménye ugyanis egy olyan mátrix, amelynek minden eleme A valamely $k \times k$-s részmátrixának determinánsa. (Ha A $n \times m$-es, akkor ez a mátrix $\binom{n}{k} \times \binom{m}{k}$ méretű.) Például:

```
Clear[A]
A = {{1, 3, 1, -1, -3}, {4, -1, 1, -1, -1},
     {-7, 5, -1, 1, -1}, {1, -5, -1, -1, 1}};
```

```
Minors[A, 4]
{{0, 0, 0, 0, 0}}

Minors[A, 3]
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
 {2, 24, 36, 6, 10, 12, -8, -12, 0, 4},
 {-4, -48, -72, -12, -20, -24, 16, 24, 0, -8},
 {-2, -24, -36, -6, -10, -12, 8, 12, 0, -4}}
```

Ennek a mátrixnak a rangja tehát 3.

Egy mátrix rangját a fentinél kevesebb számolást igénylő eljárással is meghatározhatjuk. Az ezzel kapcsolatos állítás azt mondja ki, hogy egy mátrix rangja nem változik meg akkor, ha a sorain úgynevezett elemi sorműveleteket hajtunk végre. Ilyen műveletek a következők: két sor felcserélése, egy sornak nullától különböző számmal való megszorzása, valamelyik sor tetszőleges számmal vett szorzatának egy másik sorhoz való hozzáadása. Ilyen átalakításokat alkalmaz a

`RowReduce`

belső függvény. Az eredményként adódó mátrix rangja (ami megegyezik a kiindulási mátrix rangjával) sok esetben már közvetlenül leolvasható. Például:

```
RowReduce[A] // MatrixForm
1   0   0  -4  -6
0   1   0  -3  -5
0   0   1  12  18
0   0   0   0   0
```

A *Mathematicá*nak ezzel az eljárásával illusztrálhatjuk a mátrix rangjának egy másik lehetséges (az előzővel ekvivalens) definícióját is. Nevezetesen: egy mátrix rangja egyenlő a lineárisan független sorvektorainak a maximális számával. A fentebb megadott `A` mátrixnak pontosan 3 lineárisan független sorvektora van, ezért a rangja 3.

A *Mathematica* 2.2.x változataiban nincs olyan beépített függvény, amelyik eredményként mátrix rangját adja meg. A sokat sejtető `TensorRank` belső függvényt erre nem használhatjuk. A legegyszerűbben talán a

`NullSpace`

belső függvény felhasználásával definiálhatunk mátrix rangját kiszámoló

eljárást. Ha $A \in \mathbf{C}^{n\times m}$, akkor a `NullSpace[A]` utasítás eredménye a

$$\ker(A) := \{\ x \in \mathbf{C}^m\ :\ Ax = 0\ \} \subset \mathbf{C}^m$$

altér bázisvektorainak listája:

```
NullSpace[A]
{{6, 5, -18, 0, 1}, {4, 3, -12, 1, 0}}
```

Bebizonyítható az, hogy az A mátrix rangja (a fenti jelöléseket használva) megegyezik az $m - \dim\ker(A)$ számmal. Ezért mátrix rangját megadó függvényt így definiálhatunk a *Mathematicá*ban:

```
mrang[x_] := Length[Transpose[x]] - Length[NullSpace[x]]
```

A korábban értelmezett `A` mátrixot felhasználva ellenőrizzük az új eljárás működését:

```
mrang[A]
3
```

• Négyzetes mátrix inverze

Most a négyzetes és reguláris (azaz nullától különböző determinánsú) mátrix inverzének kiszámolásához használható

```
Inverse
```

függvény lehetőségeit ismertetjük. Mátrix általánosított inverzéről a 3.8.8. pontban lesz szó.

A *Mathematica* a megfelelő műveleteket a $\mathbf{Z}_p$ véges számtestben végzi el, ha a

```
Modulus -> p
```

opciót (ennek alapértelmezés szerinti értéke 0) adjuk meg.

Az `Inverse` függvénynél is a bemenő adatok szintaxisával választhatjuk meg a kiértékelés szimbolikus (pontos) vagy numerikus (közelítő) módját.

Nézzük először azt az esetet, amikor a mátrixot csak pontos numerikus értékek, beépített matematikai állandók vagy pedig szimbólumok felhasználásával adjuk meg. A program ebben az esetben a pontos eredményt keresi. Ezt illusztrálják a következő példák:

```
Inverse[{{1, 2}, {2, 4}}]
LinearSolve::nosol:
     Linear equation encountered which has no solution.
Inverse[{{1, 2}, {2, 4}}]

Inverse[{{a, b}, {5a, 5b}}]
Inverse::sing: Matrix {{a, b}, {5 a, 5 b}} is singular.
Inverse[{{a, b}, {5 a, 5 b}}]

{{1 + Sqrt[2], 2 - Sqrt[3]}, {2 + Sqrt[3], 1 - Sqrt[2]}};
Inverse[%] // MatrixForm
-1 + Sqrt[2]   2 - Sqrt[3]
      2              2
2 + Sqrt[3   -1 - Sqrt[2]
     2             2
```

Bonyolultabb a helyzet akkor, amikor a mátrix elemei között szimbólumok is vannak. Az invertálhatóság problémájának eldöntéséhez és az inverz meghatározásához ebben az esetben különböző típusú azonosságok felhasználásával kell matematikai kifejezéseket kezelni, illetve átalakítani.

Tekintsük például a következő mátrixot:

```
A = {{1, 2 Sin[x]^2}, {1, 1 - Cos[2 x]}};
MatrixForm[%]
1    2 Sin[x]^2

1    1 - Cos[2 x]
```

Mivel $\cos 2x = 1 - 2\sin^2 x$ ($x \in \mathbf{R}$), ezért a fenti mátrix minden $x \in \mathbf{R}$ esetén szinguláris (azaz nem invertálható). Hajtsuk végre az

```
Inverse[A]
```

utasítást, és figyeljük meg, hogy a *Mathematica* által adott eredmény matematikai szempontból értelmetlen (a képernyőn megjelenő törtek nevezője ugyanis nulla). Világos, hogy az `Inverse` függvény nem alkalmazta az említett azonosságot.

Ilyen típusú problémák kezelésére szolgál a

`ZeroTest`

opció, amelyben kijelölhetjük a felhasználni kívánt azonosságok körét. Ennek az opciónak az alapértelmezés szerinti értéke:

```
Options[Inverse, ZeroTest]
{ZeroTest -> (#1 == 0 &)}
```

Ez azt jelenti, hogy az `Inverse` függvény a műveletek elvégzésénél csak az összevonások után 0-nak adódó kifejezéseket tekinti zérusnak.

Egyszerűen ellenőrizhetjük azt, hogy például a `Simplify` függvény ismeri a fentebb megemlített azonosságot. A következő módon közölhetjük a *Mathematicá*val azt, hogy az `A` mátrix inverzének meghatározásánál is vegye figyelembe az összes olyan azonosságot, amelyeket a `Simplify` függvény ismer:

```
Inverse[A, ZeroTest -> (Simplify[#] == 0 &)]
Inverse::sing:
    Matrix {{1, 2 Sin[x]^2}, {1, 1 + <<1>>}} is singular.
Inverse[{{1, 2 Sin[x]^2}, {1, 1 - Cos[2 x]}},
    ZeroTest -> (Simplify[#] == 0 &)]
```

A `ZeroTest` opciót az `Automatic` értékkel is használhatjuk.

Ha a mátrix elemei között közelítő numerikus értékek is vannak, akkor az inverzének is egy közelítő értékét kapjuk meg:

```
Inverse[{{1.2, 5.7}, {4.2, 5.6}}]
{{-0.3252, 0.3310}, {0.2439, -0.06968}}
```

Nagypontosságú számok alkalmazása esetén a *Mathematica* bizonyos hibabecslést is végez. Ezt illusztrálja a következő utasítássorozat:

```
B = N[Table[1/(i+j-1), {i, 6}, {j, 6}], 50];
(B . Inverse[B])[[1]]
{1., 0., 0., 0., 0., 0.}

Accuracy[%]
38
```

Az `Inverse` függvény ezekben az esetekben is ellenőrzi a megadott mátrix invertálhatóságát. A helyzetet itt az a tény bonyolítja, hogy invertálható és nem invertálható mátrixok közötti éles matematikai különbségtétel csak a valós számok ideális világában létezik. Mihelyt a mátrixokat kerekítési műveleteknek vetjük alá, a megkülönböztetés szükségképpen még zavarosabb lesz. Így bizonyos nem-szinguláris mátrixok szinguláris tehetők a kerekítés által keletkező perturbáció révén. Még gyakrabban, egy valóban szinguláris mátrix a kerekítés által egy közeli, nem-szinguláris mátrixba perturbálódhat.

• **Mátrixfüggvények**

Ismeretes, hogy bármely $A \in \mathbf{C}^{n\times n}$ $(n \in \mathbf{N})$ mátrix esetén a $\sum A^k/k!$ sor konvergens. Az összegének kiszámolásához, azaz az

$$e^A := \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

mátrix meghatározásához a

MatrixExp

belső függvényt használhatjuk fel. Például:

```
Clear[A]
A = {{4, 2, -5}, {6, 4, -9}, {5, 3, -7}};
MatrixExp[A] // MatrixForm
```

```
-1 + 3 E    E         1 - 3 E
 3 E        3 + E    -3 - 3 E
-1 + 3 E    1 + E    -3 E
```

A kiszámoláshoz szükséges időtartamot is érdemes megnézni:

```
Timing[MatrixExp[m];]
```

```
{3.296 Second, Null}
```

3.8.4. Vektornormák és mátrixnormák

A *Mathematica* belső és külső függvényei között nincs olyan, amellyel vektor (mátrix) valamelyik normáját közvetlenül ki tudnánk számolni. Ebben a pontban megmutatjuk, hogyan definiálhatunk ilyen függvényeket.

Ismeretes, hogy a következő leképezések mindegyike norma a $\mathbf{C}^n$ $(n \in \mathbf{N})$ lineáris téren:

$$\|x\|_\infty := \max_{1\le k\le n} \{\, |x_k| \,\} \qquad (x = (x_1, \cdots, x_n) \in \mathbf{C}^n)$$

(vektor *maximum-normája*),

$$\|x\|_p := \Big(\sum_{k=1}^{n} |x_k|^p\Big)^{1/p} \qquad (x \in \mathbf{C}^n,\ p \ge 1)$$

(vektor l_p*-normája*).

Vektor maximum-normáját kiszámoló függvényt így adhatunk meg:

```
VektorMaxNorma[x_?VectorQ] := Max[Abs[x]]
```

A `_?VectorQ` mintázattal közöltük a *Mathematicá*val azt, hogy a `:=` jelsorozat jobb oldalán lévő műveleteket csak akkor végezze el, ha a függvény argumentumába vektor kerül. Figyeljük meg azt is, hogy a dimenziószám megjelölése nélkül adtuk meg ezt a normát. Az `Abs` függvény ugyanis rendelkezik a `Listable` attribútummal:

```
Attributes[Abs]
{Listable, Protected}
```

Ez azt jelenti, hogy ha az argumentumába vektort (listát) írunk, akkor a program a lista minden elemére külön-külön alkalmazza az `Abs` függvényt.

Próbáljuk most ki ezt az új eljárást:

```
v1 = {-3, 2}; v2 = {4, 5, 6, 7, -8};
{VektorMaxNorma[v1], VektorMaxNorma[v2]}
{3, 8}

v3 = {1, {1, 3}};
VektorMaxNorma[v3]
VektorMaxNorma[{1, {1, 3}}]
```

Vektor l_p-normáját adja meg a következő függvény (a `@@` jelsorozat az `Apply` függvény rövid alakja):

```
VektorlpNorma[x_?VectorQ, p_ /; p>=1] :=
                  (Plus @@ (Abs[x]^p))^(1/p)
{VektorlpNorma[v1, 2], VektorlpNorma[v1, 1]}
{Sqrt[13], 5}
```

Tekintsük ezután a következő mátrixnormákat:

$$\|A\|_E := \Big(\sum_{i,j=1}^{n} |a_{ij}|^2\Big)^{1/2} \qquad (A \in \mathbf{C}^{n\times n})$$

(mátrix *euklideszi normája*),

$$\|A\|_S := \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}| \qquad (A \in \mathbf{C}^{n\times n})$$

(mátrix *sorösszeg-normája*),

$$\|A\|_O := \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| \qquad (A \in \mathbf{C}^{n\times n})$$

(mátrix *oszlopösszeg-normája*).

Ezek kiszámolásához használhatjuk a következő függvényeket:

```
MatrixEuNorma[x_] := (Plus @@ Flatten[Abs[x]^2])^(1/2)
MatrixSoNorma[x_] := Max[Apply[Plus, Abs[x], {1}]]
MatrixOszNorma[x_] := Max[
              Apply[Plus, Abs[Transpose[x]], {1}]]
```

Ezeknek a függvényeknek a működését is kipróbáljuk:

```
A = Array[#1^2 + #2^3&, {2, 2}];
MatrixForm[%]
```

```
2   9
5   12
```

```
{MatrixEuNorma[A], MatrixSoNorma[A], MatrixOszNorma[A]}
```

```
{Sqrt[254], 17, 21}
```

Az A négyzetes mátrix $\|\cdot\|$ mátrixnormára vonatkozó *kondíciószáma* az $\|A\| \cdot \|A^{-1}\|$ valós szám. Ezt a *Mathematicá*ban például így definiálhatjuk:

```
KondiSzam[x_, norma_:MatrixEuNorma] :=
                norma[x] norma[Inverse[x]]
```

A `KondiSzam` függvényt két argumentummal értelmeztük. Az elsőbe kell beírni a mátrixot, a másodikba pedig egy korábban már definiált mátrixnorma nevét. Ez utóbbi argumentum elhagyható. Ebben az esetben a program a `:` jel után megadott mátrixnormával számolja ki a kondíciószámot:

```
{KondiSzam[A], KondiSzam[A, MatrixEuNorma]}
```

$$\left\{\frac{254}{21}, \frac{254}{21}\right\}$$

```
{KondiSzam[A, MatrixSoNorma], KondiSzam[A, MatrixOszNorma]}
```

```
{17, 17}
```

3.8.5. A Gram–Schmidt-féle ortogonalizációs eljárás

Véges dimenziós euklideszi terekben végezhetünk bizonyos számolásokat a `LinearAlgebra`Orthogonalization`` programcsomag alábbi függvényeivel:

`GramSchmidt`	`Projection`
`Normalize`	

Az alapértelmezés az $(\mathbf{R}^n, \langle\cdot,\cdot\rangle)$ euklideszi tér, ahol n adott természetes szám és $\langle\cdot,\cdot\rangle$ a szokásos skaláris szorzat:

$$\langle x, y\rangle := \sum_{k=1}^{n} x_k y_k \qquad (x, y \in \mathbf{R}^n),$$

ami az

$$\|x\|_2 := \sqrt{\sum_{k=1}^{n} |x_k|^2} \qquad (x \in \mathbf{R}^n)$$

euklideszi normát indukálja.

Olvassuk be a szóban forgó programcsomagot:

```
<<LinearAlgebra`Orthogonalization`
```

Adott $v \in \mathbf{R}^n$ vektor esetén a `Normalize[v]` utasítás eredménye olyan $\mathbf{R}^n$-beli vektor, amelynek euklideszi normája 1:

```
v = {1, 1, 1, 1};
Normalize[v]
```

$$\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\}$$

A `Projection[v1, v2]` utasítás a $v_1 \in \mathbf{R}^n$ vektornak a $v_2 \in \mathbf{R}^n$ vektor irányába eső merőleges vetületét, azaz a

$$\frac{\langle v_1, v_2\rangle}{\langle v_2, v_2\rangle} \cdot v_2$$

vektort adja meg. Például:

```
v1 = {1, 2, 3}; v2 = {-1, 2, -6};
Projection[v1, v2]
```

$$\{\frac{15}{41}, -\frac{30}{41}, \frac{90}{41}\}$$

Az $f_1, f_2, \cdots, f_m \in \mathbf{R}^n$ vektorok által meghatározott altérben a Gram–Schmidt-féle ortogonalizációs eljárással ad meg egy ortonormált bázist a

```
GramSchmidt[{f1, f2, ..., fm}]
```

utasítás. Ugyanezt normálás nélkül így kapjuk meg:

```
f1 = {1, 2, 2, -1};
f2 = {1, 1, -5, 3};
f3 = {3, 2, 8, -7};
GramSchmidt[{f1, f2, f3}, Normalized -> False]
{{1, 2, 2, -1}, {2, 3, -3, 2}, {2, -1, -1, -2}}
```

Az előzőekben ismertetett műveleteket másik skaláris szorzattal ellátott euklideszi térben is elvégezhetjük. A skaláris szorzatot az

InnerProduct

opcióval definiálhatjuk. A felsorolt függvények mindegyike rendelkezik ezzel az opcióval, ami alapértelmezésben a pont elején megadott skaláris szorzat. Ezt a *Mathematica* így fejezi ki: `InnerProduct -> Dot`, ami ekvivalens a következővel: `InnerProduct -> (#1.#2)&`.

Egyetlen példán mutatjuk meg skaláris szorzat megadásának módját. Tekintsük a $(\mathcal{P}_n, \langle\cdot,\cdot\rangle_I)$ euklideszi teret, ahol $\mathcal{P}_n$ jelöli a legfeljebb n-edfokú valós algebrai polinomok lineáris terét, és a skaláris szorzatot így értelmezzük:

$$\langle f, g\rangle_I := \int_{-1}^{1} \frac{f(t)g(t)}{\sqrt{1-t^2}}dt \qquad (f, g \in \mathcal{P}_n).$$

Ismeretes, hogy ha például az $f_i(x) := x^i$ $(x \in \mathbf{R},\ i = 0, 1, \cdots, 4)$ polinomokra a Gram–Schmidt-féle eljárást alkalmazzuk, akkor a megfelelő indexű elsőfajú Csebisev-polinomokat kapjuk:

```
GramSchmidt[{1, x, x^2, x^3, x^4}, Normalized -> False,
        InnerProduct->
          (Integrate[(#1 #2)/Sqrt[1-x^2], {x, -1, 1}]&)]
```

$$\{1,\ x,\ -\frac{1}{2} + x^2,\ \frac{-3\,x}{4} + x^3,\ \frac{1}{8} - x^2 + x^4\}$$

Az elsőfajú Csebisev-polinomok beépített függvényként is szerepelnek a *Mathematicá*ban. Nézzük meg most ezeket is:

```
Table[ChebyshevT[k, x], {k, 0, 4}]
{1, x, -1 + 2 x^2, -3 x + 4 x^3, 1 - 8 x^2 + 8 x^4}
```

3.8.6. Lineáris egyenletrendszerek megoldása

Ebben a pontban a lineáris egyenletrendszerek megoldásához segítséget adó

`LinearSolve`	`Reduce`
`NullSpace`	`Solve`

belső függvények, valamint a `LinearAlgebra`Tridiagonal`` programcsomagban meglévő

`TridiagonalSolve`

külső függvény által nyújtott lehetőségeket ismertetjük.

Tekintsük először az $A \in \mathbf{C}^{m\times n}$ $(m,n \in \mathbf{N})$ együtthatómátrixszal és a $b \in \mathbf{C}^n$ vektorral megadott

$$Ax = b \qquad ?(x \in \mathbf{C}^n)$$

lineáris egyenletrendszert.

A `Solve` függvény (lásd a 3.3.1. pontot) lineáris egyenletrendszerek megoldásához az LU-algoritmust alkalmazza. Ha az egyenletek megadásánál pontos numerikus értékeket használunk, akkor a pontos megoldást kapjuk meg:

```
egy1 = {x+2y+3z==12, 4x+5y+6z==13, 7x+8y+11z==14};
Solve[egy1, {x, y, z}]
```

$$\{\{x \to -\frac{34}{3},\ y \to \frac{35}{3},\ z \to 0\}\}$$

Az egyenletrendszert így is megadhatjuk:

```
A1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 11}};
b1 = {12, 13, 14};
Clear[x]; x = {x1, x2, x3};
Solve[A1 . x == b1, x]
```

$$\{\{x1 \to -\frac{34}{3},\ x2 \to \frac{35}{3},\ x3 \to 0\}\}$$

A `Solve` függvény a matematikában megszokott módon oldja meg az egyenletrendszert akkor is, amikor annak végtelen sok megoldása van:

```
A2 = {{1, -2, 3, -4}, {0, 1, -1, 1},
      {1, 3, 0, -3}, {0, -7, 3, 1}};
b2 = {4, -3, 1, -3};
```

```
Clear[x];
x = {x1, x2, x3, x4};
meo = Solve[A2 . x == b2, x]
{{x1 -> -8, x3 -> 6 + 2 x4, x2 -> 3 + x4}}
```

Ennek az egyenletrendszernek az általános megoldása:

```
altmeo = {meo[[1,1,2]], meo[[1,3,2]], meo[[1,2,2]], x4}
{-8, 3 + x4, 6 + 2 x4, x4}
```

Üres listát kapunk eredményként abban az esetben, ha az egyenletrendszernek nincs megoldása:

```
A3 = {{2, -1, 3}, {3, -5, 1}, {4, -7, 1}};
b3 = {9, -4, 5};
Clear[x]; x = {x1, x2, x3};
Solve[A3 . x == b3, x]
{}
```

A `LinearSolve` belső függvény szintén az LU-algoritmust használja lineáris egyenletrendszerek megoldásához:

```
LinearSolve[A1, b1]
```

$$\{-\frac{34}{3},\ \frac{35}{3},\ 0\}$$

```
LinearSolve[A2, b2]
{-8, 3, 6, 0}

LinearSolve[A3, b3]
LinearSolve::nosol:
     Linear equation encountered which has no solution.
LinearSolve[{{2, -1, 3}, {3, -5, 1}, {4, -7, 1}},
         {9, -4, 5}]
```

A második példában figyeljük meg azt is, hogy ezzel az eljárással abban az esetben is csak egyetlen megoldást kapunk meg, amikor az egyenletrendszernek végtelen sok megoldása van.

Ismeretes, hogy az $Ax = 0$ ($\in \mathbf{C}^n$) homogén lineáris egyenletrendszer megoldásainak a halmaza a $\mathbf{C}^n$ tér egy altere. Ennek az altérnek egy bázisát kapjuk meg a `NullSpace[A]` utasítás eredményeként. Például:

```
NullSpace[A2]
{{0, 1, 2, 1}}
```

Tridiagonális mátrixszal megadott egyenletrendszert megoldhatjuk az imént megemlített függvények segítségével is, de használhatjuk a hatékonyabb algoritmust tartalmazó `TridiagonalSolve` külső függvényt is. Olvassuk be az ezt tartalmazó programcsomagot:

```
<<LinearAlgebra`Tridiagonal`
```

Tekintsük a következő tridiagonális mátrixot:

```
{a, b, c} = {{7, 1, 11}, {4, 8, 2, 12}, {5, 9, 3}};
A = Table[Switch[
    j-i, -1, a[[j]], 0, b[[j]], 1, c[[j-1]], _, 0],
    {i, 4}, {j, 4}];
MatrixForm[A]
```

$$\begin{matrix} 4 & 5 & 0 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 11 & 12 \end{matrix}$$

Oldjuk most meg az $Ax = r$ egyenletrendszert:

```
r = {2, 3, 4, 5};
TridiagonalSolve[a, b, c, r]
```

$$\{-\frac{28}{9}, \frac{26}{9}, \frac{5}{27}, \frac{20}{81}\}$$

Végül hasonlítsuk össze a különböző eljárással végzett kiértékeléshez szükséges időtartamokat:

```
Clear[x]; x = {x1, x2, x3, x4};
Timing[Solve[A . x == r, x];]
```

```
{4.723 Second, Null}
```

```
Timing[LinearSolve[A, r];]
```

```
{2.472 Second, Null}
```

```
Timing[TridiagonalSolve[a, b, c, r];]
```

```
{0.55 Second, Null}
```

Paramétereket tartalmazó egyenletrendszereket is megoldhatunk. Például a `Reduce` függvény a paraméterek összes lehetséges komplex értékét figyelembe véve adja meg a megoldást.

Oldjuk meg például a

$$\begin{aligned} tx + y + z &= 1 \\ x + ty + z &= t \qquad ?((x,y,z) \in \mathbf{C}^3) \\ x + y + tz &= t^2 \end{aligned}$$

egyenletrendszert, ahol t tetszőleges komplex paraméter:

```
Reduce[{t*x+y+z==1, x+t*y+z==t, x+y+t*z==t^2}, {x, y, z}]
```

$$\begin{aligned} &\texttt{x == 1 - y - z \&\& t == 1 ||} \\ &\texttt{-1 + t != 0 \&\& 2 + t != 0 \&\&} \\ &\texttt{x == } \frac{\texttt{-1 - t}}{\texttt{2 + t}} \texttt{ \&\& y == } \frac{\texttt{1}}{\texttt{2 + t}} \texttt{ \&\&} \\ &\texttt{z == } \frac{\texttt{1 + 2 t + t}^{\texttt{2}}}{\texttt{2 + t}} \end{aligned}$$

A kapott eredmény tehát azt jelenti, hogy ha $t \in \mathbf{C} \setminus \{1, -2\}$, akkor az egyenletrendszernek egyetlen megoldása a

$$\left(-\frac{t+1}{t+2}, \frac{1}{t+2}, \frac{(t+1)^2}{t+2}\right)$$

vektor. Ha $t = 1$, akkor az

$$(1 - y - z, y, z) \qquad (y, z \in \mathbf{C})$$

számhármasok mindegyike kielégíti a fenti egyenleteket. A $t = -2$ esetben az egyenletrendszernek nincs megoldása. Valóban:

```
t = -2;
Reduce[{t*x+y+z==1, x+t*y+z==t, x+y+t*z==t^2}, {x, y, z}]
```

```
False
```

3.8.7. Mátrix sajátértékei és sajátvektorai

A $\lambda_0 \in \mathbf{C}$ számot az $A \in \mathbf{C}^{n\times n}$ ($n \in \mathbf{N}$) mátrix *sajátértéké*nek nevezzük, ha létezik olyan nemnulla $s \in \mathbf{C}^n$ vektor (az ilyen s-re azt mondjuk, hogy az A mátrix λ_0 sajátértékéhez tartozó *sajátvektora*), amelyre az $As = \lambda_0 s$ egyenlőség teljesül. Ha a λ_0 sajátértékhez tartozó sajátvektorokhoz a nulla vektort is hozzávesszük, akkor a $\mathbf{C}^n$ tér egy alterét kapjuk. Ennek az altérnek a dimenziója a λ_0 sajátérték *geometriai multiplicitása*.

Ismeretes, hogy a λ_0 szám pontosan akkor sajátértéke az A mátrixnak, ha λ_0 megoldása a

$$\det(A - \lambda I_n) = 0 \qquad ?(\lambda \in \mathbf{C})$$

egyenletnek (I_n-nel az n-dimenziós egységmátrixot jelöltük), amelyet az A mátrix *karakterisztikus egyenleté*nek szokás nevezni. Azt mondjuk, hogy a λ_0 sajátérték *algebrai multiplicitása* l, ha λ_0 a karakterisztikus egyenletnek l-szeres gyöke. Egyszerűen bebizonyítható, hogy bármely A mátrix λ_0 sajátértékének geometriai multiplicitása nem nagyobb, mint az algebrai multiplicitása.

A *Mathematica* egyenletmegoldó algoritmusait használja az

`Eigenvalues`

függvény négyzetes mátrix sajátértékeinek meghatározásához. A 3.3. szakaszban leírtaknak megfelelően, ha a mátrixot csak pontos numerikus értékekkel vagy beépített matematikai állandókkal adjuk meg, akkor ez az eljárás a pontos sajátértékeket próbálja meghatározni. Az alábbi példákban figyeljük meg azt is, hogy az eredményként adódó listában minden sajátérték annyiszor szerepel, amennyi annak az algebrai multiplicitása:

```
A1 = {{0, 2, 1}, {-2, 0, 3}, {-1, -3, 0}};
Eigenvalues[A1]
{0, -I Sqrt[14], I Sqrt[14]}

A2 = {{3, -2, -1}, {3, -4, -3}, {2, -4, 0}};
Eigenvalues[A2]
{-5, 2, 2}

A3 = {{2, -1, -1}, {2, -1, -2}, {-1, 1, 2}};
Eigenvalues[A3]
{1, 1, 1}
```

Szimbolikus számolások eredményei esetenként nehezen áttekinthetőek. Ezt is illusztrálja a következő utasítássorozat, amelynek eredményét itt nem közöljük:

```
A4 = {{0,5,0,0,0,0}, {1,0,4,0,0,0}, {0,1,0,3,0,0},
      {0,0,1,0,2,0}, {0,0,0,1,0,1}, {0,0,0,0,1,0}};
Eigenvalues[A4]
```

Ilyen esetekben az `N` belső függvényt alkalmazva a sajátértékek egy közelítő értékét kapjuk meg:

```
N[%, 4] // Chop
{-1.889, 1.889, -0.6167, 0.6167, -3.324, 3.324}
```

A program figyelmeztető üzenetet küld akkor, ha nem találja meg a karakterisztikus egyenlet pontos megoldását:

```
A5 = {{5, 4, 3, 2, 1}, {4, 6, 0, 4, 3}, {3, 0, 7, 6, 5},
      {2, 4, 6, 8, 7}, {1, 3, 5, 7, 9}};
Eigenvalues[A5]
Eigenvalues::eival:
    Unable to find all roots of the
         characteristic polynomial.
Eigenvalues[{{5, 4, 3, 2, 1}, {4, 6, 0, 4, 3},
   {3, 0, 7, 6, 5}, {2, 4, 6, 8, 7}, {1, 3, 5, 7, 9}}]

N[%, 20]
{22.406875307580410673 1, 7.5137241542053727579,
  4.8489501203161481508, 1.32704559955676522789,
 -1.09659518165869680963}
```

Az `Eigenvalues` függvény numerikus módszert alkalmazva határozza meg a sajátértékek egy közelítő értékét abban az esetben, ha a mátrix elemei között közelítő numerikus értékek is szerepelnek:

```
Eigenvalues[{{1.2, 3.4}, {5.6, 7.8}}]
{9.97083, -0.970832}
```

Szimbólumot tartalmazó mátrix sajátértékeit is meghatározhatjuk:

```
Eigenvalues[{{0, a}, {-a, 0}}]
{-I a, I a}
```

Sajátvektorok kiszámolásánál az

`Eigenvectors`

függvény szintén a bemenő adatok szintaxisától függően választja meg a kiértékelés szimbolikus (pontos) vagy numerikus (közelítő) módját. (Az `Eigensystem` függvény a sajátértékeket és a sajátvektorokat is megadja.) Például:

```
Eigenvectors[{{1.2, 3.4}, {5.6, 7.8}}]
{{-0.361441, -0.932395},
 {-0.842853, 0.538144}}
```

```
Eigenvectors[A2]
{{1, 3, 2}, {1, 0, 1}, {2, 1, 0}}

Eigenvectors[A3]
{{1, 0, 1}, {1, 1, 0}, {0, 0, 0}}
```

Az `A3` mátrixnak egyetlen háromszoros sajátértéke van, ennek algebrai multiplicitása tehát 3. Az ehhez tartozó lineárisan független sajátvektorok száma, azaz a sajátérték geometriai multiplicitása azonban 2. A harmadik példában figyeljük meg azt, hogy a *Mathematica* ezt a tényt nulla vektor kiírásával jelzi.

3.8.8. Mátrix felbontása

Mátrix felbontásához és általánosított (vagy Moore–Penrose-féle) inverzének meghatározásához az alábbi belső függvények állnak rendelkezésünkre:

`HermiteNormalForm`	`QRDecomposition`
`JordanDecomposition`	`SchurDecomposition`
`LUDecomposition`	`SingularValues`
`PseudoInverse`	

Felhasználhatjuk még a `LinearAlgebra`Cholesky`` programcsomag

`CholeskyDecomposition`

függvényét, valamint a `LinearAlgebra`GaussianElimination`` programcsomag

`LUFactor`

függvényét is.

A fentiek közül részletesebben csak a téglalapmátrixokra alkalmazható

`SingularValues`

belső függvényt tanulmányozzuk, amely mátrix *szinguláris felbontását* szolgáltatja.

Legyen $n, m \in \mathbf{N}$ $(n \leq m)$, és tekintsünk egy $A \in \mathbf{C}^{n \times m}$ mátrixot:

```
A = {{1, 0, 1, 1}, {0, 1, -1, 0}, {1, 1, 0, 1}};
```

Jelölje A^* az A mátrix transzponáltjának a komplex konjugáltját:

```
Acsillag = Conjugate[Transpose[A]]
{{1, 0, 1}, {0, 1, 1}, {1, -1, 0}, {1, 0, 1}}
```

és r az A mátrix rangját:

```
r = Length[NullSpace[[A]]]
2
```

Ismeretes, hogy A-hoz létezik olyan

$$U \in \mathbf{C}^{r \times n}, \quad D \in \mathbf{C}^{r \times r}, \quad \text{valamint} \qquad V \in \mathbf{C}^{r \times m}$$

mátrix, amelyre a következők teljesülnek:

- a D mátrix diagonális, és főátlójában az AA^* hermitikus mátrix nullától különböző (tehát pozitív) sajátértékei állnak (Ezeknek a négyzetgyökei A *szinguláris értékei.*);
- az U és a V mátrix sor-ortonormált;
- $A = U^* D V$.

A `SingularValues[N[A]]` utasítás eredménye a fenti feltételeket kielégítő U, D és V mátrixot (ebben a sorrendben) tartalmazó lista. A D diagonális mátrixot a program csak a főátlóban levő elemek kiírásával jelzi. Felhívjuk a figyelmet arra, hogy a függvény argumentumába az A mátrix egy közelítő numerikus értékét (lásd a 3.1.3. pontot) írtuk be. Ez a függvény ugyanis csak közelítő numerikus értékeket tartalmazó mátrix felbontását adja meg.

Állítsuk most elő a fenti A mátrix egy szinguláris felbontását (A D-t a `Diag` változóval fogjuk jelölni.):

```
felbontas = SingularValues[N[A]]
{{{-0.707107, 1.51097 10^-16, -0.707107},
    {0.408248, -0.816497, -0.408248}},
   {2.23607, 1.73205},
  {{-0.632456, -0.316228, -0.316228, -0.632456},
    {-1.12989 10^-16, -0.707107, 0.707107,
      -8.09818 10^-17}}}
```

A tényezőket ebből így választhatjuk ki:

```
U = felbontas[[1]];
Diag = DiagonalMatrix[felbontas[[2]]];
V = felbontas[[3]];
```

Ellenőrizzük a kapott eredményt:

```
X = Conjugate[Transpose[U]] . Diag . V // Chop;
A == X
```

```
True
```

A `SingularValues` függvény numerikus módszerek felhasználásával határozza meg a szinguláris felbontást. A végeredmény értékes jegyeinek számát a `Tolerance` opcióval és az `N` belső függvénnyel (lásd a 3.1.3. pontot) szabályozhatjuk. Például így:

```
SingularValues[N[A, 50]]
```

Komplex elemű téglalap alakú $A \in \mathbf{C}^{n\times m}$ mátrix általánosított (vagy Moore–Penrose-féle) inverzét, azaz a (fenti jelöléseket használva) az

$$A^{+} := V^{*}D^{-1}U \in \mathbf{C}^{m\times n}$$

mátrixot így kapjuk meg a *Mathematicá*ban:

```
PseudoInverse[A]
```

Az alábbi példa mutatja, hogy itt pontos numerikus értékekkel is dolgozhatunk:

```
PseudoInverse[A]
```

$$\{\{\frac{1}{5},\ 0,\ \frac{1}{5}\},\ \{-\frac{1}{15},\ \frac{1}{3},\ \frac{4}{15}\},\ \{\frac{4}{15},\ -\frac{1}{3},\ -\frac{1}{15}\},\ \{\frac{1}{15},\ 0,\ \frac{1}{15}\}\}$$

3.8.9. Lineáris programozás

A gyakorlat számos olyan problémát vet fel, amelyben valós értékű függvény szélsőértékhelyeit (optimumhelyeit) és az ezekben felvett függvényértéket (optimumot) kell meghatározni. A matematika *általános módszerek* kidolgozásával igyekszik választ adni ilyen jellegű kérdésekre.

Ha a szóban forgó függvény például differenciálható, akkor felhasználhatjuk az *analízis* megfelelő eredményeit. Ezekkel a módszerekkel sok esetben *lokális* optimumokat tudunk egyszerűen meghatározni. A gyakorlat azonban *globális* (*abszolút*) szélsőértékek keresését is igényelheti. Az analízis módszereivel ez több esetben nem egyszerű feladat.

Az alkalmazások szempontjából fontos feladatosztályok esetében a *globális* szélsőérték-problémákat a *lineáris algebra* módszereivel is vizsgálhatjuk. Ezek közül a legegyszerűbb feladattípusnak, a lineáris programozás feladatának megoldásához nyújtanak segítséget a *Mathematica* alábbi függvényei:

```
ConstrainedMax        LinearProgramming
ConstrainedMin
```

A lineáris programozás általános feladatának lényege az, hogy az $\mathbf{R}^n$ tér bizonyos részhalmazán értelmezett valós értékű lineáris függvény abszolút szélsőértékeit keressük. A szóban forgó függvény értelmezési tartományát a változókra vonatkozó egyenlőtlenségrendszerrel adjuk meg. Pontosabban: legyen adott az n, az m, az r_1 és az r_2 ($r_1 < r_2 < m$) természetes szám, az $A = (a_{ij}) \in \mathbf{R}^{m\times n}$ mátrix, a $b \in \mathbf{R}^m$ és a $c \in \mathbf{R}^n$ vektor. Jelölje D az $\mathbf{R}^n$ tér azon $x = (x_1, \cdots, x_n)$ pontjainak halmazát, amelyekre teljesülnek a következő feltételek:

$$0 \le x_j \qquad (j = 1, 2, \cdots, n),$$

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i \qquad (i = 1, 2, \cdots r_1),$$

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (i = r_1 + 1, r_1 + 2, \cdots r_2),$$

$$\sum_{j=1}^{n} a_{ij}x_j \ge b_i \qquad (i = r_2 + 1, r_2 + 2, \cdots m).$$

Határozzuk meg az

$$L : D \longrightarrow \mathbf{R}, \qquad L(x) := \sum_{j=1}^{n} c_j x_j$$

függvény abszolút szélsőértékhelyeit és az itt felvett függvényértékeket.

A D definíciójában szereplő $0 \le x_j$ feltételeket nem kell a programmal külön közölni. Alapértelmezésben nemnegatív koordinátájú pontok körében keresi a megoldást.

Minimum keresésére vonatkozó feladatot így oldhatunk meg:

```
ConstrainedMin[x + y,
    {-2x + y <= 2, 2x + y == 9, 3x + y >= 11}, {x, y}]
```

$$\{\frac{9}{2}, \{x \to \frac{9}{2}, y \to 0\}\}$$

Maximumot pedig így határozhatunk meg:

```
ConstrainedMax[x + y,
    {-2x + y <= 2, 2x + y == 9, 3x + y >= 11}, {x, y}]
{7, {x -> 2, y -> 5}}
```

Figyelmeztető üzenetet kapunk akkor, ha a vizsgált függvény értelmezési tartományát megadó feltételek az üres halmazt határozzák meg:

```
ConstrainedMax[x, {x <= 1, x >= 2}, {x}]
ConstrainedMax::nsat:
          Specified constraints cannot be satisfied.
```

Megjegyezzük még azt is, hogy a *Mathematica* fenti függvényeinek második argumentumában a `<=` (`>=`) jel helyett a `<` (`>`) relációs jelet is használhatjuk. Érdemes megfigyelni, hogy a program hogyan kezeli az ilyen feladatokat:

```
ConstrainedMax[x, {x < 1}, {x}]
{1, {x -> 1}]
```

A `LinearProgramming` függvényt is az előzőekben ismertetett feladattípus megoldásához használhatjuk. Itt csak a problémát meghatározó mátrixot és vektorokat kell megadnunk. A

```
LinearProgramming[c, A, b]
```

utasítás eredménye az

$$L(x) := \langle c, x \rangle, \qquad x \in \{x \in \mathbf{R}^n : x \geq 0,\ A \cdot x \geq b\}$$

függvény ($c \in \mathbf{R}^n$ adott vektor) abszolút *minimumhelyét* tartalmazó lista.

```
LinearProgramming[{1, 1}, {{2, -1}, {2, 1}, {3, 1}},
        {-2, 9, 11}]
```

$$\{\frac{9}{2}, 0\}$$

3.9. Számelmélet

A számelmélet a matematikának egy olyan területe, amely régóta komoly feladatokat ad a számítástechnikának: feladatainak megoldása általában igen hosszú időt vesz igénybe. Másrészt ez a tény a hatékony algoritmusok fejlesztésére kényszerítő erővel hat.

A *Mathematicá*ban szereplő számelméleti algoritmusok több értelemben is *valószínűségi jellegűek*. Az egyik lehetőségre példa a `PrimeQ` függvény értékeit (`True`, ha az argumentum prímszám, `False` egyébként) kiszámoló algoritmus, amely kis számokra biztosan, nagyobbakra nagy valószínűséggel ad jó választ. Az egy tényezőt leválasztó `FactorIntegerECM` külső függvény (`NumberTheory`FactorIntegerECM`) viszont biztosan jó választ ad, de ezt (amint az a válasz megadásához szükséges időtartamból is kitűnik) véletlen algoritmussal teszi.

A számelmélet és az algoritmusok szempontjából is igen hasznos Wagon [87] könyve.

Végül még egy általános megjegyzés: nehezen választható szét a számelmélet a diszkrét matematikától. Ez azzal jár, hogy esetenként átfedések fordulnak elő, illetve esetenként egyes függvényeket nem azon a helyen ismertetünk, ahová azok a program készítőinek szándékai szerint kerültek.

• Számok alakjai

A számok különféle alakú reprezentálásáról már szóltunk a 3.1.3. szakaszban. A számelmélet elsősorban *egész* számokkal foglalkozik. Azt, hogy az x szám egész-e, többféleképpen dönthetjük el. Ha az, akkor `IntegerQ[x]` értéke `True`, `Head[x]` értéke pedig `Integer` lesz.

Ha azt akarjuk kikötni, hogy a továbbiakban x egész szám legyen, akkor ezt írhatjuk:

```
x/: IntegerQ[x]=True
```

Ettől még x feje nem lesz `Integer`, ha nem adunk neki értéket:

```
Head[x]

Symbol
```

Ha pedig egy valós számból egészet akarunk csinálni, akkor a `Ceiling, Floor, Round` kerekítő függvények valamelyikét használhatjuk.

3.9.1. Számrendszerek

A különböző számrendszerek közötti átváltás igen könnyen végezhető el.

```
BaseForm[113, 2]
```

megadja a tízes számrendszerbeli 113 szám kettes számrendszerbeli alakját. Az $FA0F_{16}$ (tizenhatos számrendszerbeli) számot decimálisba pedig egyszerű (az alábbi szintaxisnak megfelelő) begépelés eredményeként kapjuk:

```
16^^fa0f
64015
```

Sokszor szükségünk van az n egész szám számjegyeinek listájára. Ezt szolgáltatja az `IntegerDigits[n]` függvény, a `RealDigits[x]` pedig megadja az x valós szám számjegyeit, valamint a tizedespont előtt álló számjegyek számát. Ha ennek a két függvénynek második argumentumként a b számot is megadjuk, akkor az adott szám b alapú számrendszerbeli alakjának analóg összetevőit („jegyeit") kapjuk.

Tanulságos átváltani arab számokat római számmá. Lépésenként oldjuk meg a feladatot.

Bizonyos „osztópontok" fontos szerepet játszanak az átváltásnál; először ezek halmazát építjük fel:

```
alap = Reverse[Power[10, Range[0, 3]]];
arab = Reverse[Union[alap, 9 Rest[alap], 5 Rest[alap],
   4 Rest[alap], {0}]];
```

Most állítsuk elő az osztópontoknak megfelelő római számokat:

```
romai = ToString /@ {M, CM, D, CD, C, XC, L, XL,
   X, IX, V, IV, I, { }};
```

Függvényünk neve `toRoman` lesz. Egyszerre akarjuk megadni az összes osztópontban felvett értékét, ehhez kell, hogy listára lehessen alkalmazni:

```
SetAttributes[toRoman, Listable];
MapThread[SetDelayed, {toRoman[arab], romai}];
```

A definíció lényeges részét rekurzíve adjuk meg:

```
foresz[x_ /; 0<x<4000] := First[Select[arab, #<x&, 1]];
toRoman[x_Integer /; 0<x<4000] := SequenceForm[
   toRoman[foresz[x]], toRoman[x-foresz[x]]]
```

Alkalmazzuk függvényünket 2 néhány hatványára:

```
toRoman /@ Table[2^i, {i, 0, 12}]
{I, II, IV, VIII, XVI, XXXII, LXIV, CXXVIII,
   CCLVI, DXII, MXXIV, MMXLVIII, toRoman[4096]}
```

Ezek után az Olvasó megpróbálkozhat azzal is, hogy római számokat alakít arab számokká.

3.9.2. Oszthatóság

Az oszthatósággal kapcsolatos kérdések vizsgálatához leggyakrabban a következő fügvényeket használjuk:

Divisors	LCM
EvenQ	Mod
ExtendedGCD	OddQ
GCD	PowerMod
LatticeReduce	Quotient

Az `EvenQ[n]` kifejezés értéke `True`, ha n páros szám, az `OddQ[n]` kifejezésé pedig akkor `True`, ha x páratlan szám.

`Mod[m, n]` megadja az m szám osztási maradékát n-nel való osztás után, `Quotient[m, n]` pedig a hányados egész részét. Az

```
m == n Quotient[m, n] + Mod[m, n]
```

kifejezés értéke minden konkrét m és n egész szám mellett `True`. Ugyanez áll az

```
m n == GCD[m, n] LCM[m, n]
```

relációra, ugyanis `GCD` a legnagyobb közös osztót (*greatest common divisor*), `LCM` pedig a legkisebb közös többszöröst (*least common multiple*) adja meg (nemcsak két, hanem akárhány argumentum esetére). Így tehát m és n pontosan akkor relatív prím, ha

```
GCD[m, n]
1
```

Ha az a^b hatvány n-nel való osztási maradékát akarjuk kiszámítani, akkor ezt nemcsak `Mod[a^b, n]` segítségével tehetjük meg, hanem (a^b kiszámítása nélkül, tehát gyorsabban) így is: `PowerMod[a, b, n]`.

```
{Timing[Mod[30^100000, 7]],
   Timing[PowerMod[30, 100000, 7]}
                                  -14
{{33.285 Second, 2},{2.59237 10     Second, 2}}
```

Negatív egész b számokkal pedig *moduláris inverz* meghatározására használható a `PowerMod` függvény, ilyenkor ugyanis olyan k számot ad eredményül (ha van ilyen), amelyre `Mod[k/a^b, n]==1` teljesül:

```
PowerMod[3, -1, 7]
5
```

És valóban:

```
Mod[3 %, 7]
1
```

Az `ExtendedGCD[m, n]` kifejezés értéke egy olyan `{d, {r, s}}` lista, amelynek első eleme a d legnagyobb közös osztó, továbbá $d = rm + sn$. További általánosítást jelent a `LatticeReduce` függvény, amely (Lenstra, Lenstra és Lovász algoritmusa [50] alapján) egész koordinátájú vektorok listájához speciális tulajdonságú bázist állít elő az egész számok fölött:

```
LatticeReduce[{{1, 0, 0, 12345},
   {0, 1, 0, 12345}, {0, 1, 0, 12345}}]
{{-1, 0, 1, 9}, {9, 1, -10, 0}, {85, -143, 59, 6}}
```

A majdnem párhuzamos, hosszú vektorokból majdnem merőleges, rövid vektorokból álló bázist kaptunk. Egydimenziós vektorokhoz a függvény — nem túlságosan meglepő módon — a legnagyobb közös osztót rendeli:

```
LatticeReduce[{{48}, {120}}]
{{24}}
```

Ha az n szám pozitív osztóinak listáját akarjuk megkapni, ezt a `Divisors[n]` utasítás kiadásával érhetjük el.

Amennyiben egy „nagy” binomiális együttható adott prímszám melletti osztási maradékát akarjuk kiszámolni, akkor a természetesen adódó megoldás helyett használhatjuk a `NumberTheory`Binomial`` csomag megfelelő utasítását; ez általában (de nem mindig) gyorsabban vezet célhoz:

```
<<NumberTheory`Binomial`
{Timing[Mod[Binomial[1000,500],29]],
 Timing[BinomialMod[1000,500,29]]}
{{1.483 Second, 0}, {0.33 Second, 0}}
```

Jó tudni, hogy ugyanitt találhatunk egy `FastBinomial` és egy `FastFactorial` függvényt is.

• **Polinomok**

Polionomok tényezőre bontása, oszthatósággal kapcsolatos vizsgálata — az alábbi függvények felhasználásával — hasonlóan végezhető, mint az egész számoké:

`Factor`	`PolynomialLCM`
`FactorList`	`PolynomialMod`
`FactorSquareFree`	`PolynomialQ`
`FactorSquareFreeList`	`PolynomialQuotient`
`FactorTerms`	`PolynomialRemainder`
`FactorTermsList`	`Resultant`
`PolynomialGCD`	

A fenti függvények némelyikével már foglalkoztunk a 3.2.1. pontban, most néhány további példát mutatunk:

```
{PolynomialRemainder[x^2, x + 1, x],
   PolynomialQuotient[x^2, x + 1, x]}
```

```
{1, -1 + x}
```

Az eredmények természetesen függnek attól, hogy mit tekintünk független változónak:

```
{PolynomialRemainder[x + y, x - y, x],
   PolynomialQuotient[x + y, x - y, y]}
```

```
{2y, 2x}
```

A legnagyobb közös osztót így kaphatjuk meg:

```
PolynomialGCD[(1-x)^2 (1+x) (2+x), (1-x) (2+x) (3+x)]
```

```
(1 - x) (2 + x)
```

Határozzuk meg egy polinom együtthatóit modulo 2:

```
PolynomialMod[Expand[(1+x)^6], 2]
```

```
     2    4    6
1 + x  + x  + x
```

Bontsuk tényezőkre az eredményt az egész számok fölött:

```
Factor[%]
(1 + x^2)(1 + x^4)
```

Ha a polinomok együtthatóit modulo 2 tekintjük, akkor a felbontást tovább folytathatjuk:

```
Factor[%, Modulus -> 2]
(1 + x)^6
```

(Véletlen, hogy az eredeti polinomot kaptuk vissza?)

A `PolynomialMod` függvény csak szorzásokat és kivonásokat végez, míg a `PolynomialRemainder` osztásokat is; ez magyarázza az alábbi eredményeket:

```
{PolynomialMod[x^2, 2x + 1],
   PolynomialRemainder[x^2, 2x + 1, x]}
{x^2, 1/4}
```

3.9.3. Lánctörtek

Tetszőleges valós számot lánctörtekkel közelíthetünk a

```
NumberTheory`ContinuedFractions`
```

könyvtár függvényeivel. Az x valós szám *lánctörtbe fejtés*e:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \dots}}.$$

Az itt szereplő a_i számok a *parciális hányadosok*. Racionális számoknak véges sok parciális hányadosuk van, míg irracionális számok lánctörtbe fejtett alakja végtelen. Az $a_0, \dots a_n$ számok által meghatározott p_n/q_n racionális szám az *n-edik szelet*. A szeletek bizonyos értelemben egy adott valós szám kis nevezőjű legjobb racionális közelítését szolgáltatják:

```
<<NumberTheory`ContinuedFractions`
ContinuedFraction[N[Sqrt[499], 50], 30]
ContinuedFractionForm[{22, 2, 1, 21, 1, 2,
   44, 2, 1, 21, 1, 2, 44, 2, 1, 21, 1, 2,
   44, 2, 1, 21, 1, 2, 44, 2, 1, 21, 1, 2}]
```

Az eredmény tipikus: pozitív egész szám négyzetgyökének lánctört alakjában ismétlődő periódusok vannak. (Bizonyítsuk be!)

És hogyan írjuk föl magát a lánctörtet? Például így:

```
rec[x_, n_] := HoldForm[1/(n+x)]
lanctort[lista_] := First[lista] +
   Fold[rec, 1/Last[lista], Rest[Reverse[Rest[lista]]]]
lanctort[{a, b, c, d, e, f}]
```

$$a + \cfrac{1}{b + \cfrac{1}{c + \cfrac{1}{d + \cfrac{1}{e + \cfrac{1}{f}}}}}$$

```
lanctort[ContinuedFraction[Sqrt[2], 6][[1]]]
```

$$1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2}}}}}$$

Tipográfiai szempontból a lánctörtek legkényelmesebb jelölése:

$$a_0 + \frac{1}{a_1+}\frac{1}{a_2+}\ldots\frac{1}{+a_n}.$$

Megpróbálhatjuk ezt is előállítani programmal.

Fejtsük lánctörtbe az aranymetszést megadó arányt és számítsuk ki a közelítő törteket. Egy lehetséges megoldás:

```
ContinuedFraction[N[GoldenRatio, 40], 30]
```

```
ContinuedFractionForm[{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}]
```

Tehát az összes szereplő együttható értéke 1.

```
Table[Normal[ContinuedFractionForm[Table[1, {n}]]],
   {n, 11}]
```

$$\{1,\ 2,\ \frac{3}{2},\ \frac{5}{3},\ \frac{8}{5},\ \frac{13}{8},\ \frac{21}{13},\ \frac{34}{21},\ \frac{55}{34},\ \frac{89}{55},\ \frac{144}{89}\}$$

Az is jól látható, hogy a törtek — amelyek a szomszédos Fibonacci-számok hányadosai — tartanak az aranymetszés értékéhez:

```
% - N[GoldenRatio]
{-0.618034, 0.381966, -0.118034, 0.0486327,
   -0.018034, 0.00696601, -0.00264937, 0.00101363,
   -0.00038693, 0.000147829, -0.0000564607}
```

Racionális számmal való közelítést végez a `Rationalize` függvény, amely erre a célra a lánctörteket használja.

```
N[3/7, 30]
0.428571428571428571428571428571
```

megadja 3/7 értékét 30 jegy pontossággal. Ha ezután adjuk ki a `Rationalize[%%]` parancsot, akkor az előző érték racionális közelítéseként éppen 3/7-et kapunk.

```
Rationalize[N[E]]
2.71828
```

a szokásos 6 jegynyi pontossággal szolgáltatja az e szám racionális közelítését (és nem talál jobbat, mint e kerekített alakja).

```
Rationalize[N[E], 10^-4]
193/71
```

olyan közelítést ad, amelynek hibája nem nagyobb, mint 10^{-4}. Ha a hibahatárt 0-nak választjuk, akkor csak a bemenő érték pontossága korlátozza a racionális közelítés pontosságát.

Javasoljuk az Olvasónak, hogy határozza meg az $e-2$ szám lánctörtekkel való közelítését jónéhány (például 40) jegyre. Az eredmény azt sugallja, hogy az egymás után következő számok: $1, 2, 1,\ 1, 4, 1,\ 1, 6, 1,\ \cdots,\ 1, 2n, 1$. Ez a megdöbbentő állítás Perron tételeként ismert.

Érdemes még belenézni a `NumberTheory`Rationalize`` programcsomagba, ahol az `AffineRationalize` és a `ProjectiveRationalize` külső függvényt találjuk; továbbá kipróbálni a `NumberTheory`Recognize`` programcsomag `Recognize` függvényét is.

3.9.4. Prímszámok. A számelmélet alaptétele

A prímszámokkal kapcsolatos vizsgálatokban használt belső függvények:

`Prime, PrimePi, PrimeQ`

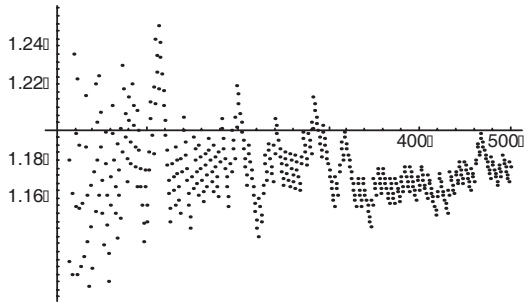
A `PrimeQ[n]` kifejezés értéke `True`, ha n prímszám, egyébként pedig `False`. (Nagy számok esetén előfordulhat, hogy ezt a kifejezést ki tudja értékelni a program, annak ellenére, hogy n-et tényezőkre bontani nem tudja.) A k-adik prímszámot adja meg `Prime[k]`, míg `PrimePi[x]` az x valós számnál nem nagyobb prímszámok számát adja meg.

Tanulmányozhatjuk a prímszámok elhelyezkedését:

```
p[eddig_] := ListPlot[Table[{k, Prime[k]}, {k, eddig}]]
```

Az alábbi definícióval megadott függvényt talán még érdekesebb:

```
f[eddig_] := ListPlot[Table[{k, Log[k] PrimePi[k]/k},
   {k, eddig}]]
f[500]
```



Jones és munkatársai [37] szerint a prímszámok halmaza megegyezik azon pozitív egészek halmazával, amelyeket az egész koordinátájú pontokban értékként felvesz az alábbi 25-ödfokú 26-változós polinom:

```
Jones[a_, b_, c_, d_, e_, f_, g_, h_, i_, j_, k_, l_,
m_, n_, o_, p_, q_, r_, s_, t_, u_, v_, w_, x_, y_, z_]:=
   (k+2)(1-(w z+h+j-q)^2-((g k+2g+k+1)(h+j)+h-z)^2
   -(2n+p+q+z-e)^2-(16(k+1)^3 (k+2)(n+1)^2+1-f^2)^2
   -(e^3 (e+2)(a+1)^2+1-o^2)^2-((a^2-1)y^2+1-x^2)^2
   -(16r^2 y^4(a^2-1)+1-u^2)^2
   -(((a+u^2(u^2-a))^2-1)(n+4d y)^2+1-(x+c u)^2)^2
   -(n+l+v-y)^2-((a^2-1)l^2+1-m^2)^2-(a i+k+1-l-i)^2
   -(p+l(a-n-1)+b(2a n+2a-n^2-2n-2)-m)^2
   -(q+y(a-p-1)+s(2a p+2a-p^2-2p-2)-x)^2
   -(z+p l(a-p)+t(2a p-p^2-1)-p m)^2)
```

Ellenőrizzük az állítást néhány helyen:

```
Do[(Print[lista = Table[Random[Integer, {-10, 10}] ,{26}]];
   Print[vajh = Jones @@ lista];
   If[vajh>0, Print[PrimeQ[vajh]]]), {20}]
```

A `NumberTheory`NumberTheoryFunctions`` programcsomag `NextPrime` függvénye az argumentumát követő legkisebb prímszámot adja meg.

Az n szám prímtényezős felbontását a `FactorInteger[n]` olyan lista formájában adja meg, amelynek elemei az n felbontásában szereplő prímekből és azok kitevőjéből álló listák. Húsznál kevesebb jegyű számoknál az utasítás biztosan eredményre vezet. Nagyobb számoknál egy alkalmas opció-beállítással (`FactorComplete->False`) elérhetjük, hogy csak egyetlen tényezőt keressen meg a program, s ezzel esetleg több lépésben elvégezhetjük a tényezőkre bontást.

A Gauss-egészek feletti (egyébként szintén egyértelmű) felbontást egy opció megfelelő beállításával kaphatjuk meg:

```
FactorInteger[n, GaussianIntegers->True]
```

(A program a nem valós tényezőknek csak a felét adja meg, a többit az előjelek megfelelő megváltoztatásával kapjuk meg, mégpedig komplex konjugáltak képzésével.)

```
FactorInteger[2, GaussianIntegers -> True]
{{-I, 1}, {1 + I, 2}}
```

Tegyük fel, hogy egy nagy számról akarjuk megállapítani, hogy milyen tényezőkből áll. Első lépésben használjuk a `PrimeQ` függvényt. Ha kiderül, hogy a szám nem prím, akkor második lépésben alkalmazhatjuk a `NumberTheory`FactorIntegerECM`` könyvtár `FactorIntegerECM` függvényét:

```
<<NumberTheory`FactorIntegerECM`
PrimeQ[2^67 + 1]
False

FactorIntegerECM[2^67 + 1]
3
```

Alkalmazzuk ugyanezt a függvényt két nagy prímszám szorzatára:

```
n = Prime[10^5] Prime[10^6]
20127115513867

PrimeQ[n]
False
```

```
FactorIntegerECM[n]
1299709
```

A prímszámokkal kapcsolatos vizsgálatokhoz a következő külső függvények találhatók a `NumberTheory`PrimeQ`` programcsomagban:

```
PrimeQCertificate
PrimeQCertificateCheck
ProvablePrime
```

a `NumberTheory`NumberTheoryFunctions`` programcsomagban pedig a `NextPrime`.

3.9.5. Számelméleti függvények

Az egészrész és a törtrész függvény a számelméletben is fontos szerepet játszik. Kiindulásként válaszoljuk meg azt a jól ismert kérdést, hogy 100! hány nullára végződik.

Könnyen írhatunk ezután függvényt annak meghatározására, hogy az $n!$ szám törzstényezős felbontásában a p prímszám milyen kitevővel szerepel:

```
f[n_, p_] := Floor[n/p]
kitevo[n_, p_] := Apply[
                    Plus, FixedPointList[f[#, p]&, Floor[n/p]]]
kitevo[100, 5]
24
```

Az n szám pozitív osztóinak listáját `Divisors[n]` állítja elő, tehát n (pozitív) osztóinak számát például így határozhatjuk meg:

```
d1[n_] := Length[Divisors[n]]
```

Ugyanerre a célra egy másik megoldás:

```
d2[n_] := Apply[Times, Transpose[FactorInteger[n]][[2]]+1]
```

A harmadik megoldást pedig

```
d3[n_] := DivisorSigma[0, n]
```

adja, ugyanis `DivisorSigma[k, n]` az n természetes szám pozitív osztói k-adik hatványainak összege. A pozitív osztók összege tehát `DivisorSigma[1, n]`.

A Moebius-féle μ függvény 0 értéket vesz fel azokon az n számokon, amelyek oszthatók egy 1-nél nagyobb szám négyzetével; azokon a számokon

pedig, amelyek négyzetmentesek, értéke $(-1)^k$, ahol k az n szám törzstényezőinek száma. (Speciálisan tehát $\mu(1) = 1$.) Egyszerűen alkalmazható ez a függvény a négyzetmentesség vizsgálatára:

```
négyzetmentes1[n_] := MoebiusMu[n] != 0
```

(A *Mathematicá*ban az ilyen függvényeket szokás *egysoros*ként említeni, bár itt a rövidség oka nem valamilyen mély *programozási* ismeret, hanem valamilyen *matematikai* állítás.) A négyzetmentesség ellenőrzésére egy természetesebbnek tűnő definíciót így kaphatunk:

```
negyzetmentes2[n_]:=
   Max[Transpose[FactorInteger[n][[2]]] < 2] /; n > 0
```

Végül megemlítjük, hogy a `NumberTheory`NumberTheoryFunctions`` csomagban készen is megtalálható a `SquareFree` függvény.

Itt jegyezzük meg, hogy az analízis Newton–Leibniz-féle alaptételének számelméleti megfelelője a *Moebius-féle megfordítási (inverziós) formula*, amely szerint, ha

$$\forall n \in \mathbf{N} \quad g(n) = \sum_{d|n} f(d),$$

akkor

$$\forall n \in \mathbf{N} \quad f(n) = \sum_{d|n} \mu(d) g(n/d).$$

(Ekkor azt mondjuk, hogy a g függvény f *összegzési függvénye*.)

Az *Euler-féle φ függvény* megadja a $1, 2, \ldots, n$ sorozat n-hez relatív prím elemeinek számát minden pozitív egész n szám esetére. Értékeit az `EulerPhi` belső függvény számolja ki.

Számos további hasznos számelméleti függvény található a

```
NumberTheory`NumberTheoryFunctions`
```

programcsomagban, például:

```
ChineseRemainderTheorem    PrimitiveRoots
ClassList                  SqrtMod
ClassNumber                SquareFreeQ
QuadraticRepresentation
```

Ezek némelyikével még fogunk találkozni.

3.9.6. Kongruenciák

• Elsőfokú kongruenciák

Az $ax \equiv 1 \pmod{m}$ alakú kongruenciákat már megoldottuk, a megoldást `PowerMod[a, -1, m]` adja. Az $ax + by = c$ lineáris kétváltozós diophantoszi egyenlet egy megoldását `{c/d r, c/d s}` adja, ha

```
ExtendedGCD[a, b]
```

```
{d, {r, s}}
```

Mivel az $ax \equiv b \pmod{m}$ elsőfokú egyismeretlenes kongruencia azt jelenti, hogy létezik olyan $y \in \mathbf{Z}$, amellyel $ax - b = my$, vagyis $ax - my = b$, ezért az ilyen kongruenciák megoldását visszavezettük az előző feladatra.

• Elsőfokú kongruenciarendszerek (szimultán kongruenciák)

A `NumberTheory`NumberTheoryFunctions`` könyvtárban található az a külső függvény (`ChineseRemainderTheorem`), amely megadja azt a legkisebb számot, amelynek egy adott lista elemeivel vett osztási maradéka megegyezik egy másik adott lista elemeivel. Ha az első lista elemei relatív prímek, akkor ilyen szám biztosan létezik. Használatát a következő közismert feladaton mutatjuk be.

Karcsi Mikulás-zacskókat töltött meg dióval. Ha mindegyik zacskóba 9 darabot tett, akkor az utolsóba csak 8 darab került, ha mindegyikbe 8 szemet tett, akkor az utolsóba 7 szem jutott, ha hetet tett egy-egy zacskóba, akkor az utolsóba 6 darab került, amikor hatosával osztotta szét a diókat, akkor az utolsóba 5 maradt. Hány darab diója volt Karcsinak, ha 600-nál kevesebb diót szánt az ajándékozásra? (Ez a feladat el szokott hangzani sorbaállított ólomkatonákkal és különféleképpen ültetett gyerekekkel is.)

```
<<NumberTheory`NumberTheoryFunctions`
ChineseRemainderTheorem[{6, 7, 8}, {7, 8, 9}]
```

```
503
```

Mivel

```
Mod[503, {6, 7, 8, 9}]
```

```
{5, 6, 7, 8}
```

ezért 503 tényleg jó megoldás.

• Magasabb fokú kongruenciák, binom kongruenciák

Ezek megoldhatók egy véges algoritmussal: be kell helyettesítenünk egy mod m teljes maradékrendszer elemeit a kongruencia bal oldalán álló polinomba. Vannak egyszerűsítési módszerek, például *binom kongruenciák* megoldására. (Binom kongruenciának az $ax^k \equiv b \pmod{m}$ alakú kongruenciát nevezzük. — Amint már említettük, a $b = 1$ speciális eset megoldását `PowerMod` megadja.) Az általános eset látszólag nagyon egyszerűen kezelhető: a `Solve` függvénynek megadhatjuk egyenletként a modulus értékét, vagy a `Mode->Modular` opció-beállítással elérhetjük, hogy a program maga keressen alkalmas modulust, amely mellett van az egyenletnek megoldása:

```
Solve[{5+11x+17x^2+7x^3+16x^4+x^5==0, Modulus==19}, x]
{{Modulus -> 19, x -> -18}, {Modulus -> 19, x -> -16},
   {Modulus -> 19, x -> -12}, {Modulus -> 19, x -> -7},
   {Modulus -> 19, x -> -1}}

Solve[x^2 + 1 == 0, x^3 + 1 == 0, x, Mode -> Modular]
{{Modulus -> 2, x -> -1}}
```

Lássunk egy példát ellenőrzéssel együtt:

```
f[x_, y_] = x^3 + 4x + 17;
Solve[f[x, y] == D[f[x, y], x] == 0, x, Mode -> Modular]
{{Modulus -> 8059, x -> 1001}}

{f[x, y], D[f[x, y], x]} /. %
{{1003007022, 3006007}}

Mod[%, 8059]
{{0, 0}}
```

• Kvadratikus reciprocitás

A másodfokú kongruenciák vizsgálatához használatos *Jacobi-féle* $(\frac{m}{n})$ *szimbólum*ot `JacobiSymbol[m,n]` számolja ki. Amikor n páratlan prímszám, akkor értéke a *Legendre-féle szimbólum*ra specializálódik.

A kvadratikus reciprocitási tétel szerint, ha p és q páratlan prímszámok, akkor

$$(\frac{p}{q}) = (\frac{q}{p})(-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Először oldjunk meg kétféleképpen egy kvadratikus kongruenciát:

```
Solve[{x^2 == 3, Modulus == 11}, x]
{{Modulus -> 11, x -> -6}, {Modulus -> 11, x -> -5}}
```

Valóban:

```
Mod[6^2, 11]
3

Mod[5^2, 11]
3
```

A másik megoldási mód:

```
<<NumberTheory`NumberTheoryFunctions`
   SqrtMod[3, 11]
5
```

Ha a modulus nem prímszám, akkor a `Solve` függvény nem találja meg a megoldást:

```
Solve[{x^2 == 3, Modulus == 11^3}, x]
Roots::modp: Value of option Modulus -> 1331
   should be a prime number.
{ToRules[Modulus == 1331 && Roots[x^2 == 3,
   x, Modulus -> 1331]]}
```

A külső `SqrtMod` függvény ilyenkor is beválik:

```
SqrtMod[3, 11^3]
578
```

Most tanulmányozzuk a Legendre-szimbólumot:

```
Mod[(Range[5]-1)^2, 5]
{0, 1, 4, 4, 1}

JacobiSymbol[Range[5]-1, 5]
{0, 1, -1, -1, 1}
```

3.9.7. További fejezetek

Végezetül néhány további számelméleti érdekességre hívjuk fel a figyelmet.

• A Collatz-függvény

A Collatz-féle függvénnyel — amelynek értéke az 1 helyen 1, páros helyen az argumentum fele, páratlan helyen pedig az argumentum háromszorosa plusz 1 — is kísérletezhetünk, ugyanis több helyen is megtalálhatjuk. Az `Examples`Collatz`` programcsomagban találjuk a `Collatz` valamint a `TotalStoppingTime` függvényt, a `ProgrammingExamples`Collatz`` csomagban pedig a `Collatz, FindMaxima` és a `StoppingTime` függvényt.

• Ramanujan

A `NumberTheory`Ramanujan`` programcsomagot teljes egészében Ramanujan munkássága indukálta. Ebben a következő függvényeket találjuk:

```
RamanujanTau                        RamanujanTauTheta
RamanujanTauDirichletSeries         RamanujanTauZ
RamanujanTauGeneratingFunction
```

• Számpartíciók

A számelmélet és a diszkrét matematika nehezen szétválasztható területek, ezért nem meglepő, hogy az előbbi iránt érdeklődők számos hasznos és érdekes dolgot fognak találni a `DiscreteMath` programcsomag különböző könyvtáraiban, például a `'Combinatorica'` elnevezésűben:

```
Compositions                        PartitionQ
FerrersDiagram                      Partitions
LongestIncreasingSubsequence        RandomTableaux
NumberOfTableaux                    Tableaux
```

3.9.8. Gyakorlatok és feladatok

1. Határozzuk meg az Avogadro-féle állandó karakterisztikáját és mantisszáját:

```
<<Miscellaneous`PhysicalUnits`;
  MantissaExponent[AvogadroConstant Mole]
{0.6022136699999999, 24}
```

2. Határozzuk meg az n-edik számjegyét annak a számnak, amelyet úgy kapunk, hogy egymás mellé leírjuk a természetes számokat:

```
f[1] = {1};
f[n_] := Flatten[Append[f[n-1], IntegerDigits[n]]]
h[n_] := f[n][[n]]
Timing[h[50]]
{1.483 Second, 3}

k[1] = {1};
k[n_] := k[n]=Flatten[Append[k[n-1], IntegerDigits[n]]];
l[n_] := l[n]=k[n][[n]]
Timing[l[50]]
{0.824 Second, 3}
```

(A második függvény nagyobb számokra gyorsabban számol.)

3. Írjunk olyan `alak` nevű függvényt, amely a `FactorInteger` utasítás eredményét a szokásos formában adja meg, tehát például

```
alak[FactorInteger[24]]
 3
2  · 3
```

4. Hasonlítsuk össze az osztók számát számoló három függvény gyorsaságát.

5. Határozzuk meg az x pozitív valós számhoz azokat a legkisebb p és q pozitív egész számokat, amelyekre a p/q tört ε-nál kisebb hibával közelíti az x számot:

```
Rationalize[N[Pi], 10^-#]& /@ Range[0, 8]
{3, 22/7, 22/7, 355/113, 355/113, 355/113, 355/113, 104348/33215, 104348/33215}
```

6. Hány számjegyet írunk le, ha a pozitív egészeket 1-től 1996-ig bezárólag felírjuk 10-es számrendszerben?
Útmutatás. Nyilván általánosabb feladatot old meg az alábbi függvény.

```
g[k_, b_] := Sum[Length[IntegerDigits[i, b]], i, k]
g[#, 10]& /@ {9, 99, 1996}
```

```
{9, 189, 6877}
```

7. Tegyük fel, hogy adott egy szám b-alapú számjegyeinek listája. Állítsuk elő a szám tízes számrendszerbeli alakját!
Útmutatás. J. Adams egysorosa:

```
digitsToNumber[lista_, b_] := Fold[#1*b+#2&, 0, lista];
digitsToNumber[{1, 3, 5, 2}, 7]
```

```
527
```

$2 \leq b \leq 10$ esetén értelmes bemenő adatokra működik. Milyen ellenőrzésre lenne szükség? Hogyan lehetne kiterjeszteni $10 < b \leq 36$ esetére?

8. Írjunk olyan függvényt, amely előállítja az $x \in [0,1)$ tízes számrendszerbeli szám faktoriálisos számrendszerbeli alakjának első néhány jegyét. Az x szám előállítása faktoriálisos számrendszerben ([81]):

$$x = \sum_{n=2}^{\infty} \frac{\alpha_n}{(n!)^2} \quad 0 \leq \alpha_n \leq n^2 - 1.$$

9. Írjunk olyan függvényt, amely előállítja az $x \in [0,1)$ tízes számrendszerbeli szám Cantor-féle felbontását. (Tekintsük pozitív egészek egy $q_n \geq 2$ $n \in \{1, 2, \dots\}$ sorozatát. Az x szám Cantor-féle felbontása [66]:

$$x = \sum_{n=1}^{\infty} \alpha_n/(q_1 q_2 \dots q_n) \quad 0 \leq \alpha_n \leq q_n - 1.)$$

10. Írjunk külön függvényt a héttel való oszthatóság ellenőrzésére:

```
multipleOfSevenQ[n_Integer] := Mod[n, 7]==0
```

11. Állapítsuk meg egy pozitív egészekből álló lista elemeiről, hogy *páronként* relatív prímek-e.
Egy lehetséges megoldás:

```
<<DiscreteMathematics`Combinatorica`
parok[lista_] := KSubset[lista, 2]
lis = {4, 6, 9};
Max @@ (GCD @@ parok[lis]) > 1
```

```
True
```

Tanulságos a feladatot megoldani hagyományos programozási stílusban is, feltételekkel és ciklusokkal, majd az eredmény gyorsaságát nagy listákra összevetni az itt adott megoldással.

12. Hasonlítsuk össze a `GCD` belső függvény gyorsaságát az alábbiakban definiált, az euklideszi algoritmusra épülő egysoroséval (J. Adams):

```
lnko[a_, b_] := If[b==0, a, lnko[b, Mod[a, b]]];
lnko[12, 42]
```

```
6
```

(Lényegében ez a megoldás szerepel a programozási tankönyvekben is a *rekurzív algoritmus* mintapéldájaként.)

13. Alkalmazzuk az összes `Factor`ral kezdődő és nem `Integer`rel folytatódó nevű függvényt a

```
t=Expand[2 (1+x)^2 (2+x) (3+x)]
```

polinomra!

14. Legyen az f függvény összegzési függvénye g. Ellenőrizzük az alábbi függvénypárokra az inverziós formulát n néhány értékére.

a) Ha $\forall n \quad f(n) = 1$, akkor $\forall n \quad g(n) = d(n)$.
b) Ha $\forall n \quad f(n) = n$, akkor $\forall n \quad g(n) = \sigma(n)$.
c) Ha $\forall n \quad f(n) = \mu(n)$, akkor $\forall n \quad g(n) = \delta_{1,n}$, ahol $\delta_{n,m}$ a Kronecker-féle szimbólumot jelöli.

15. Írjunk olyan függvényt, amely a `GCD` függvény felhasználásával számítja ki az Euler-féle φ függvény értékét.

16. Ellenőrizzük az Euler–Fermat-féle tételt, amely szerint

```
Mod[a^EulerPhi[n], n]
1
```

minden olyan a számra, amely relatív prím n-hez.

17. Oldjuk meg a $176x \equiv 118 \pmod{206}$ kongruenciát.

18. Hányféleképpen állítható elő az n pozitív egész szám két négyzetszám összegeként, ha az előjelekre és a sorrendre is tekintettel vagyunk?
Útmutatás. J. Adams egysoros megoldása:

```
sumOfSquares[n_] := If[EvenQ[n], sumOfSquares[n/2],
   If[I^n==-I, 0, 4Plus @@ Im[I^Divisors[n]]]];
sumOfSquares /@ {1, 2, 3, 4, 5}
```

19. Hányféleképpen állítható elő az 50 pozitív egész szám két négyzetszám összegeként, ha az előjelekre és a sorrendre nem vagyunk tekintettel?
Útmutatás. Lépésenként fölépítünk egy egysorost (A *MathUser* alapján.):

```
Divisors[50, GaussianIntegers->True]
{1, 1 + I, 1 + 2 I, 1 + 3 I, 1 + 7 I, 2, 2 + I, 2 + 4 I,
   3 + I, 3 + 4 I, 4 + 2 I, 4 + 3 I, 5, 5 + 5 I,
   5 + 10 I, 5 + 15 I, 6 + 8 I, 7 + I, 8 + 6 I,
   10, 10 + 5 I, 10 + 20 I, 15 + 5 I, 20 + 10 I,
   25, 25 + 25 I, 50}

Select[Divisors[50, GaussianIntegers->True],
   Abs[#]^2==50&]
{1 + 7 I, 5 + 5 I, 7 + I}

Map[{Re[#], Im[#]}&,
   Select[Divisors[50, GaussianIntegers->True],
   Abs[#]^2==50&]]
{{1, 7}, {5, 5}, {7, 1}}


Union[Map[Sort[{Re[#], Im[#]}]&,
   Select[Divisors[50, GaussianIntegers->True],
   Abs[#]^2==50&]]]
{{1, 7}, {5, 5}}
```

```
SumOfTwoSquares[n_] := Union[Map[Sort[{Re[#], Im[#]}]&,
   Select[Divisors[n, GaussianIntegers->True],
   Abs[#]^2==n&]]];

TableForm[Table[If[SumOfTwoSquares[n]=!={ },
   {n, SumOfTwoSquares[n]}], {n, 100, 150}]]
```

20. Határozzuk meg azon tagok szorzatának a maximumát, amelyeket egy adott n pozitív egész szám pozitív egészek összegére való felbontásakor kapunk.
Útmutatás. (Nem a felbontást adjuk meg.)

```
<<DiscreteMath`Combinatorica`
   tjp1[n_] := Max[Map[Apply[Times, #]&, Partitions[n]]]
   tjp1 /@ {2, 4, 8, 16}
{2, 4, 18, 324}
```

Tömörebben:

```
tjp2[n_] := Max[(Times @@ #&) /@ Partitions[n]]
   tjp2 /@ {2, 4, 8, 16}
{2, 4, 18, 324}
```

21. (Armstrong-féle számok.) Határozzuk meg azokat a háromjegyű számokat, amelyeknek a jegyeit köbre emelve és összeadva eredményül magát a számot kapjuk.
Útmutatás. Procedurális nyelvekhez illő megoldást adunk:

```
Do[Do[Do[
   If[100x+10y+z==x^3+y^3+z^3, Print[x, y, z]],
   {z, 0, 9}], {y, 0, 9}], {x, 0, 9}]
000
001
153
370
371
407
```

Oldjuk meg a feladatot négyjegyű számokkal és negyedik hatványokkal és így tovább. Ezután általánosítsuk a feladatot arra az esetre, amikor tetszőleges alapú számrendszerben felírt számokról van szó.

3.10. Valószínűségszámítás

A valószínűségszámítás Kolmogorov-féle felépítése halmazelméleti eszközöket használ. Ezekkel kapcsolatos további példákat a 3.1.2. szakaszban megadottakon túl nem adunk. A diszkrét valószínűségi mezőknél fellépő problémák leszámlálással, kombinatorikai ismeretekkel oldhatók meg; erre mutatunk majd néhány példát. A *valószínűségszámítás és az analízis határterületei*n (a valószínűségi változók jellemzői, valószínűségi változók függvényeinek jellemzői, peremeloszlások kiszámítása, generátorfüggvények, karakterisztikus függvények stb.) lehet talán a legjobban alkalmazni a *Mathematicá*t; ezekre fogunk elsősorban koncentrálni. Így tehát itt fogjuk tárgyalni több szempontból is a

```
Statistics`ContinuousDistributions` és a
Statistics`DiscreteDistributions`
```

programcsomag függvényeit is.

Véletlentől függő jelenségek és időbeli folyamatok *szimulálása*: ez olyan terület, amely számítógép nélkül nem is tanulmányozható. Meg fogjuk mutatni, hogyan generálhatók adott (ismert vagy feltételekkel meghatározott) eloszlású véletlen számok (ha szükséges — mint például új statisztikai módszerek tanulmányozásánál —, *reprodukálható* módon). Egyszerűbb, diszkrét és folytonos idejű Markov-folyamatokat is fogunk tanulmányozni, illetve megmutatjuk azt is, hogy lineáris algebrai és gráfelméleti módszereket hogyan lehet ezek leírásánál igénybe venni.

3.10.1. Klasszikus valószínűségi mezők

A leszámlálási feladatokhoz felhasználható függvényeket a diszkrét matematikáról szóló 3.6. szakaszban már fölsoroltuk. Ezek a magon kívül elsősorban az alábbi programcsomagokban találhatók:

```
DiscreteMath`CombinatorialFunctions`
DiscreteMath`CombinatorialSimplification`
DiscreteMath`Combinatorica`
DiscreteMath`Permutations`
```

Lássunk néhány példát. Mi annak a valószínűsége, hogy egy, az $1, 2, 3, 4$ számokból alkotott négyjegyű szám egyetlen jegye sincs a helyén?

```
<<DiscreteMath`CombinatorialFunctions`
Subfactorial[4]/4!
```

```
3/8
```

Kilenc vendég rendel egy-egy tételt, összesen kettő üveg sört, négy süteményt és három kávét. Mi annak a valószínűsége, hogy a feledékeny pincér helyesen osztja szét, amit hozott?

```
1/Multinomial[2, 3, 4] // N[#, 2]&
```

```
0.00079
```

Mennyi annak a valószínűsége, hogy két, egymástól függetlenül kitöltött lottószelvény közül legalább az egyik négytalálatos?

```
p = Binomial[5, 4] Binomial[85, 1]/Binomial[90, 5]
N[2 p - p^2, 2]
```

```
0.000019
```

1000 villanykörte közül egyenlő valószínűséggel lehet 0, 1,..., 5 darab selejtes. Ha már 100 körtét megvizsgáltunk az 1000-ből, és jónak találtuk mindet, mi annak a valószínűsége, hogy mind az 1000 jó?

Alkalmazzuk Bayes-tételét a következő szereposztással. Legyen

- A az az esemény, hogy 100 körtét megvizsgáltunk, és mind jó volt;
- C_i az az esemény, hogy az 1000 között i darab hibás van.

Feltevéseink szerint $P(C_i) = p \ \ (i = 0, 1, \ldots, 5)$, és a $P(C_0|A)$ valószínűséget kell kiszámolnunk a

$$P(C_0|A) = \frac{P(A|C_0)P(C_0)}{\sum_{i=0}^{5} P(A|C_i)P(C_i)}$$

képlet alapján. Legyen $q_i := P(A|C_i)$.

```
q[i_] := Binomial[1000 - i, 100]/Binomial[1000, 100]
ered = 1.0*p/(Sum[p*q[i], {i, 0, 5}])
```

```
0.213485
```

A klasszikus valószínűségeloszlások legtöbbje megtalálható a

```
Statistics`DiscreteDistributions`
```

programcsomag függvényei között:

```
BernoulliDistribution
BinomialDistribution
DiscreteUniformDistribution
GeometricDistribution
HypergeometricDistribution
NegativeBinomialDistribution
PoissonDistribution
```

Hívjuk be ezt a programcsomagot és válasszunk egy binomiális eloszlást:

```
<<Statistics`DiscreteDistributions`
belo = BinomialDistribution[34, 0.3];
Mean[belo]
10.2
```

A mediánt kétféleképpen is előállíthatjuk:

```
{Quantile[belo, 0.5], Median[belo]}
{10.2, 10.2}
```

Ezek az eloszlások szimbolikusan is használhatók:

```
{Mean[#], PDF[#, k]}&[BinomialDistribution[n, p]]
{n p, If[0 <= k <= n,
   If[IntegerQ[k], Binomial[n, k] p^k (1-p)^(n-k), 0], 0]}
```

3.10.2. Generátorfüggvények

Diszkrét eloszlások számos jellemzője kiszámítható generátorfüggvényükből vagy momentumgeneráló (esetleg: kumulánsgeneráló) függvényükből. Ezek kezeléséhez elsősorban a

```
DiscreteMath`RSolve`
```

programcsomag függvényeit alkalmazhatjuk, amelyeket már a 3.4.2. és a 3.6.2. szakaszban is használtunk.

Egy binomiális eloszláscsalád generátorfüggvénye így kapható meg:

```
Factor[PowerSum[PDF[BinomialDistribution[17, p], k],
   {z, k}]]
(1 - p + p z)^17
```

Határozzuk meg másféle módon a Poisson-eloszlás generátorfüggvényét:

```
poissongen[z_, lambda_] = Simplify[
   PowerSum[lambda^n Exp[-lambda]/n!, {z, n, 0}]
E^(lambda (-1 + z))
```

Ismeretes, hogy *elágazó folyamatok* esetén, amennyiben G a generátorfüggvénye az utódok eloszlásának, akkor az n-edik nemzedék eloszlásának a generátorfüggvényét G önmagával vett n-szeres kompozíciója adja. Ha tehát az utódok eloszlása Poisson-eloszlás, akkor az ötödik nemzedék eloszlásának generátorfüggvényét például így kaphatjuk meg:

```
Nest[poissongen, poissongen, 4];
```

A folyamat kihalásának valószínűségét pedig a $G(z) = z$ egyenlet kisebbik gyöke adja ($z_1 := 1$ ugyanis mindig gyök). Legyen a fenti folyamatban a Poisson-eloszlás paramétere (vagyis az utódok várható száma) 2, ekkor ezt kapjuk:

```
FindRoot[poissongen[z, 2] == z, {z, 0}]
{z -> 0.203188}
```

Megemlítjük még, hogy egy eloszlás *kumulánsgenerátor-függvénye* éppen az eloszlásnak mint sorozatnak az exponenciális generátorfüggvénye.

3.10.3. Tetszőleges valószínűségi változók

Felsoroljuk a szakasz több pontjában és a matematikai statisztikáról szóló 3.11. szakaszban is használt, eloszlásokat előállító függvényeket:

```
BetaDistribution            LaplaceDistribution
CauchyDistribution          LogNormalDistribution
ChiSquareDistribution       LogisticDistribution
ExponentialDistribution     RayleighDistribution
ExtremeValueDistribution    StudentTDistribution
FRatioDistribution          UniformDistribution
GammaDistribution           WeibullDistribution
NormalDistribution
```

Az eloszlások jellemző paramétereit megadó függvényeket a matematikai statisztikáról szóló, következő szakaszban soroljuk fel.

A Γ-eloszlást igen sokszor használják — különösen a matematikai statisztikában —, mert két paraméterének változtatásával szinte bármilyen —

például adatokból becsült — eloszlás jól közelíthető. Megjegyzendő, hogy speciális eseteként adódik az exponenciális eloszlás:

```
<<Statistics`ContinuousDistributions`
{PDF[GammaDistribution[1, 1/lambda], x],
PDF[ExponentialDistribution[lambda], x]}
```

$$\{\frac{\text{lambda}}{E^{\text{lambda x}}}, \frac{\text{lambda}}{E^{\text{lambda x}}}\}$$

```
{Mean[GammaDistribution[1, 1/lambda]],
Variance[GammaDistribution[1, 1/lambda]]}
```

$$\{\frac{1}{\text{lambda}}, \text{lambda}^{-2}\}$$

A Weibull-eloszlás alkatrészek *élettartam*ának eloszlására használatos, mivel az exponenciális eloszlás olyan általánosítása, amelynek két paramétere van, s ezáltal finomabb illesztést tesz lehetővé:

```
CDF[WeibullDistribution[c, alfa, x]]
```

$$1 - \text{Exp}[-c\ \frac{x}{\text{alfa}}]$$

A lognormális eloszlás gyakran fordul elő *aprítási feladatok*nál.

A leggyakrabban előforduló abszolút folytonos eloszlások a Pearson-féle eloszláscsaládba tartoznak, azaz logaritmikus deriváltjuk racionális függvény. Ezt most csak a normális eloszlásra mutatjuk meg, a többi eloszlás (egyszerre végrehajtandó!) vizsgálatát az Olvasóra hagyjuk:

```
normalis[x_] = PDF[NormalDistribution[m, sigma], x];
Simplify[normalis'[x]/normalis[x]]
```

$$\frac{\text{m - x}}{\text{sigma}^2}$$

Állapítsuk meg, hogy az A, a, b paraméterek milyen értékeinél lesz sűrűségfüggvény az

$$x \mapsto \frac{A}{1 + a(x-b)^2} \quad (x \in \mathbf{R})$$

függvény:

```
Solve[Integrate[aa/(1 + a (x-b)^2),
   {x, -Infinity, Infinity}] == 1, aa]
```

$$\{\{\text{aa} \to \frac{\text{Sqrt[a]}}{\text{Pi}}\}\}$$

A program csak a számolásban segített, a diszkusszióban természetesen nem. A pontos válasz: b tetszőleges valós szám, a pozitív szám, $A = \sqrt{a}/\pi$.

Következő kérdésünk hasonló: eloszlásfüggvény-e az

$$x \mapsto \exp(-\exp(-x)) \quad x \in \mathbf{R}$$

függvény? Meg kell vizsgálnunk, hogy

- monoton növő-e,
- balról folytonos-e,
- határértéke $-\infty$-ben nulla-e és $+\infty$-ben 1-e?

Az igenlő választ alátámasztják a következő számolások:

```
fg[x_] = Exp[-Exp[-x]];
{Limit[fg[x], x -> -Infinity],
 Limit[fg[x], x -> Infinity]}
```

```
{0, 1}
```

```
fg'[x]
```

```
   -x
 -E   - x
E
```

Ismeretes, hogy ha a ξ valószínűségi változó sűrűségfüggvénye f, U pedig szigorúan monoton függvény, akkor az $U \circ \xi$ valószínűségi változó g sűrűségfüggvénye $g = (f/U') \circ U^{-1}$. Legyen például ξ adott `param` paraméterű exponenciális eloszlású valószínűségi változó, és határozzuk meg négyzetgyökének sűrűségfüggvényét:

```
gyoksfv[y_] = PowerExpand[
   PDF[ExponentialDistribution[param], y^2] *
   (D[Sqrt[x], x] /. x -> y^2)]
```

$$\frac{2\,\mathtt{param}\,y}{E^{\mathtt{param}\,y^2}}$$

3.10.4. Karakterisztikus függvények

Könnyen kiszámíthatjuk egy eloszlás karakterisztikus függvényét:

```
CharacteristicFunction[CauchyDistribution[0, 1], t]
```

```
 -(t Sign[t])
E
```

Az itt használt `CharacteristicFunction` függvény szintén a

```
Statistics`ContinuousDistributions`
```

programcsomagban található.

A centrális momentumok és a karakterisztikus függvény deriváltjai közötti néhány összefüggést így illusztrálhatunk a standard normális eloszlás példáján:

```
dist = NormalDistribution[0, 1]
cm[n_] := D[CharacteristicFunction[dist, t], {t, n}] /.
   t->0
Table[cm[n], {n, 1, 4}]
{Mean[dist], Variance[dist], Skewness[dist],
   Kurtosis[dist]}
{0, 1, 0, 3}
{0, 1, 0, 3}
```

3.10.5. Határeloszlás-tételek

A Moivre–Laplace-tétel szerint a binomiális eloszlás jól közelíthető vele azonos várható értékű és szórású normális eloszlással. Illusztráljuk ezt egy példán. Először ábrázoljuk a $(10, 0.5)$ paraméterű binomiális eloszlást:

```
bino = BinomialDistribution[10, 0.5];
binom[x_] := PDF[bino, x]
bin = ListPlot[Transpose[{Range[0, 10],
Table[binom[x], {x, 0, 10}]}],
   PlotStyle -> AbsolutePointSize[2]]
```

Most áttérünk a közelítő normális eloszlás ábrázolására:

```
{atlag, szoras} = {Mean[bino], Sqrt[Variance[bino]]};
nor = NormalDistribution[atlag, szoras];
kozel[x_] := PDF[nor, x]
koz = Plot[kozel[x], {x, 0, 10},
   PlotStyle -> {Thickness[0.003]}]
```

Tekintsük meg együtt a két ábrát:

```
Show[bin, koz]
```

3.10.6. Véletlenszám-generálás

Ennél a témakörnél elsősorban két belső függvényre van szükségünk, ezek: `Random` és `SeedRandom`.

Kezdjük egy megjegyzéssel. Azt gondolnánk, hogy az $x - x$ kifejezés értéke mindig nulla. Ha azonban x egy valószínűségi változó értéke, akkor ez nem igaz:

```
x := Random[ ]
x-x
-0.319965
```

Ha viszont azonnali értékadással számítjuk ki x értékét, akkor nem számolódik újra minden használatnál:

```
y = Random[ ]
0.799367

y - y
0.
```

A $[0, 1]$ intervallumban *egyenletes eloszlású* véletlen számokból álló 3 elemű listát így kaphatunk:

```
Table[Random[ ], {3}]
{0.5886454, 0.567327, 0.970526}
```

Ha a generált valós típusú $[0, 1]$-beli véletlen számokat 25 jegy pontossággal akarjuk megkapni, akkor ezt írhatjuk:

```
Table[Random[Real, {0, 1}, 25], {3}];
```

A véletlenszám-generátor természetesen itt is determinisztikus algoritmust használ. Az eredmények *reprodukálhatók* lesznek (erre szükségünk lehet például akkor, ha egy statisztikai módszert akarunk ellenőrizni), ha generálás előtt meghívjuk a `SeedRandom` függvényt azonos (egész) argumentummal. Ezt most a komplex sík $-1 - i$ és $1 + i$ pontok mint átellenes csúcsok által meghatározott négyzetéből egyenletes eloszlás szerint vett mintán mutatjuk meg:

```
SeedRandom[142857]
Table[Random[Complex, {-1-I, 1+I}, 3], {2}]
{-11. + 1. I, -6.37 + 0.795 I}
```

```
SeedRandom[142857]
Table[Random[Complex, {-1-I, 1+I}, 3], {2}]
```

```
{-11. + 1. I, -6.37 + 0.795 I}
```

Szabályos pénzérmével végzett pénzfeldobások egy 17 elemű sorozatát például az alábbi módon szimulálhatjuk:

```
Table[If[Random[Integer]==0,"F","I"], {17}]
```

```
{F, F, I, F, I, F, F, F, F, F, F, I, F, I, I, F, F}
```

Egyenletes eloszlású véletlen számokat arra is szokás használni, hogy segítségükkel valamely sokdimenziós tér egységkockáján definiált függvény *határozott integrál*ját számítjuk ki olyan módon, hogy megszámoljuk, az egységkockából véletlenül választott pontok hányad része esik a megadott függvény grafikonja alá. Először kiszámítjuk az integrál pontos értékét:

```
4 Integrate[1, {x, 0, 1}, {y, 0, Sqrt[1 - x^2]}]
```

```
Pi
```

Egy procedurális stílusú sorsoló modult készítünk:

```
g[n_] := Module[{k}, jo = 0;
   Do[If[x^2 + y^2 < 1, jo++, jo], k, n];
   Print[N[4 jo/n]]]
g[1000]
```

```
3.196
```

Elegánsabb (és gyorsabb) megoldás ugyanerre a feladatra:

```
ggg[n_] := N[Length[Select[4 Count[Table[
   {Random[ ], Random[ ]}, {n}], #1^2 + #2^2 < 1&]] / n
Table[ggg[n], {n, 1, 1001, 100}]
```

```
4., 3.08911, 3.28358, 3.20266, 3.19202, 2.99401, 3.09484
   3.17832, 3.14107, 3.10766, 3.0969
```

Ez a példa természetesen csak egyszerű illusztráció; ahhoz, hogy például parciális differenciálegyenleteket tudjunk megoldani Monte–Carlo-módszerrel, jóval bonyolultabb algoritmusra van szükségünk, lásd ehhez a megfelelő szakirodalmat. Az ilyen egyszerű példákat, ahol az integrál értéke más módszerrel egyszerűen kiszámítható, éppen a véletlenszám-generátor ellenőrzésére szokás felhasználni.

Felmerül az a kérdés, hogy a generált számok *független*eknek, illetve *egyenletes eloszlású*aknak tekinthetők-e. Ezekre a kérdésekre a matematikai statisztika módszereivel kaphatunk választ.

Tetszőleges eloszlású véletlenszámokat kétféle módon generálhatunk. Az egyik lehetőség: a `Random[elo]` olyan véletlenszámot ad, amelynek eloszlása a megadott `elo` eloszlás. A másik (gyakran alkalmazott) lehetőség: egyenletes eloszlású véletlenszámokra alkalmazzuk `elo` eloszlásfüggvényének inverzét. Ez utóbbi eljáráshoz rendelkezésünkre állnak a

```
Statistics`InverseStatisticalFunctions`
```

programcsomag függvényei:

```
InverseBetaRegularized    InverseErfc
InverseErf                InverseGammaRegularized
```

Standard normális eloszlású véletlenszámokat tehát így generálhatunk:

```
standard = NormalDistribution[0, 1]
Table[Random[standard], {5}];
```

Exponenciális eloszlásúak generálására is alkalmazhatjuk a fenti módszert:

```
expo1 = ExponentialDistribution[1]
Table[Random[expo1], {5}];
```

Támaszkodhatunk arra a tényre is, hogy az exponenciális eloszlás a Γ-eloszlás speciális esete:

```
expo2 = GammaDistribution[1, 1]
Table[Random[expo2], {5}];
```

Most alkalmazzuk a másodikként leírt módszert:

```
<<Statistics`InverseStatisticalFunctions`
tabla = Table[Random[ ], {5}];
Map[InverseBetaRegularized[0, #, 1, 1]&, tabla]
{0.618788, 0.797721, 0.44583, 0.482795, 0.483111}
```

3.10.7. Markov-láncok

Diszkrét idejű, diszkrét állapotterű Markov-láncok kezelésére a lineáris algebra (3.9. szakasz) és a gráfelmélet (3.6. szakasz) eszközeit szokás használni. Egy egyszerű példát mutatunk.

Legyen egy Markov-lánc átmenetvalószínűségi mátrixa

$$P_1 = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/3 & 0 & 2/3 \\ 1 & 0 & 0 \end{pmatrix}$$

és rajzoljuk meg a lánc gráfját:

```
p1 = {{1/2, 1/2, 0}, {1/3, 0, 2/3}, {1, 0, 0}}
Select[Flatten[Table[{i, j}, {i, 3}, {j, 3}], 1],
   ((p1[[#1, #2]] > 0&) @@ #&)]
```

```
{{1, 1}, {1, 2}, {2, 1}, {2, 3}, {3, 1}}
```

```
ShowGraph[FromOrderedPairs[%], Directed]
```

Határozzuk meg az egyes állapotok valószínűségét az ötödik időpontban, ha a rendszert az első állapotból indítottuk:

```
MatrixPower[p1, 5].{1, 0, 0}
```

$$\left\{\frac{149}{288}, \frac{259}{432}, \frac{79}{144}\right\}$$

Számítsuk ki a lánc határeloszlását:

```
Limit[MatrixPower[p1, n], n->Infinity]
```

$$\left\{\left\{\frac{6}{11}, \frac{2}{11}, \frac{3}{11}\right\}, \left\{\frac{6}{11}, \frac{2}{11}, \frac{3}{11}\right\}, \left\{\frac{6}{11}, \frac{2}{11}, \frac{3}{11}\right\}\right\}$$

(Ismeretes, hogy azonos, a stacionárius eloszlásból álló sorokat kapunk.)

3.10.8. Folytonos idejű Markov-folyamatok

A (skaláris) tiszta ugró Markov-folyamatok közül a születési-halálozási típusúak stacionárius eloszlása igen gyakran explicite megoldható differenciaegyenletnek tesz eleget. Tekintsük például azt a folyamatot, amelynek infinitézimális átmenetvalószínűségei:

$$\begin{aligned} n &\to n+1 \quad k_1 n h + o(h) \\ n &\to n-1 \quad k_2 n(n-1) h + o(h). \end{aligned}$$

Ekkor a (p_n) stacionárius eloszlásra vonatkozó

$$0 = k_1(n-1)p_{n-1} + k_2(n+1)np_{n+1} - (k_1 n + k_2 n(n-1))p_n$$

egyenlet (amint ez teljes indukcióval látható) egyenértékű a következővel:

$$k_1 p_n = k_2(n+1)p_{n+1}.$$

Oldjuk meg ezt az egyenletet:

```
<<DiscreteMath`RSolve`
RSolve[k1 p[n] == k2 (n+1) p[n+1], p[n], n]
```

$$\{\{p[n] \to \frac{(\frac{k1}{k2})^{n}\ p[0]}{n!}\}\}$$

Látható, hogy ez, a feltételek figyelembevételével, éppen Poisson-eloszlás:

```
<<Algebra`SymbolicSum`
Solve[Sum[%[[1, 1, 2]], {n, 0, Infinity}] == 1, p[0]];
%% /. %
```

$$\{\{p[n] \to \frac{(\frac{k1}{k2})^{n}}{E^{\frac{k1}{k2}}\ n!}\}\}$$

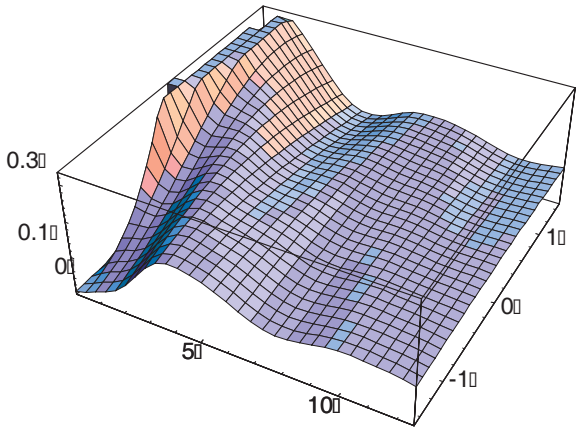
Folytonos idejű, folytonos állapotterű folyamatok közül tekintsünk most egy olyat, amelynek az abszolút eloszlásfüggvénye

$$f(x,t) = \frac{1}{\sqrt{2\pi}\sigma(t)} \exp\left(-\frac{(x-m(t))^2}{2\sigma(t)^2}\right).$$

Az m és a σ függvény alkalmas megválasztásával tanulmányozhatjuk a valószínűségi sűrűségfüggvény szétkenődését. Ebből kiderül, hogy a folyamat a legnagyobb valószínűséggel a várható értékhez közel tartózkodik, de az is jól látható, hogy az idő múltával egyre pontatlanabb jóslásokat tehetünk csak az értékére, mivel szórása végtelenhez tart. Ez a viselkedés tipikusnak tekinthető:

```
m[t_] : = Sin[t]
sigma[t_] := Sqrt[t]
f[t_, x_] := Exp[-(x - m[t])^2/(2 (sigma[t])^2)]
   /(Sqrt[2 Pi] sigma[t])
```

```
Plot3D[f[t, x], {t, 0.001, 4Pi + 0.001}, {x, -1.5, 1.5}
       PlotPoints -> 30]
```



3.10.9. Gyakorlatok és feladatok

1. Határozzuk meg az m várható értékű σ szórású normális eloszlás sűrűségfüggvényének inflexiós pontjait. Mi annak a valószínűsége, hogy az adott sűrűségfüggvényű valószínűségi változó értéke a két inflexiós pont abszcisszája közé esik?

2. Milyen m értékekre lesz az

$$f(x) := \frac{1}{2\sqrt{2\pi}} \left(e^{-\frac{(x-m)^2}{2}} + e^{-\frac{(x+m)^2}{2}} \right) \qquad (x \in \mathbf{R})$$

sűrűségfüggvénynek két maximuma?

3. Szimuláljuk a Galton-deszka működését.

3.11. Matematikai statisztika

A statisztikai programcsomagok általában jó numerikus eljárásokat tartalmaznak, de nem jól programozhatók, nem interaktívak, grafikai képességeik gyengék, szimbolikus számításokat nem lehet velük végezni és az új módszereket nehéz beléjük illeszteni. A *Mathematica* tehát akkor lehet igazán segítségünkre, ha a problémák megoldása során az említett akadályokba ütközünk. Az alábbiakban amellett, hogy bemutatjuk a szokásos módszerek alkalmazásának módját, elsősorban arra összpontosítunk, hogy megmutassuk a *Mathematica* előnyeit a fentiekben vázolt körülmények között.

Itt is felhívjuk a figyelmet arra, hogy a *MathLink* segítségével C nyelven megírt függvényt fölhasználhatunk a *Mathematicá*ban. A *MathLink for Excel* pedig arra ad módot, hogy az *Excel* táblázatkezelővel begyűjtött adatokra az *Excel*en belül használhassuk a *Mathematica* függvényeit. (Az *Excel* által elkészített adatállományokat minden előkészület nélkül be tudjuk olvasni.)

3.11.1. Az adatok előkészítése

A tulajdonképpeni statisztikai elemzés elkezdése előtt általában többféle előkészületre van szükség.

• Adatbevitel

A statisztikai adatfeldolgozásnál többnyire sok adattal van dolgunk, amelyeket külső állományokban tárolunk, ahová azok esetleg más programokból vagy mérőberendezésből kerültek. Állománykezelésről már másutt (2.4.1. szakasz) esett szó, itt most csak a hiányzó adatok statisztikai szempontból érdekes esetének kezelését mutatjuk meg.

Az adatokat igen gyakran szöveges állományba gyűjtjük. Egy sorba kerül egy egyed összes adata, ezt nevezik általában *rekord*nak; egymás alatt helyezkednek el az azonos jelentésű (különböző egyedekre vonatkozó) *mező*k.

Ha a `bemeno` állományban *hiányzó adatok* vannak, amit megtekintéskor is láthatunk:

```
!!bemeno
2.1 77 51
2.2    49
1.9 80
```

akkor a beolvasáskor a következőképpen járhatunk el. Először megnyitunk írás céljából egy *bemenő áram*ot:

```
befele = OpenRead["bemeno"];
```

Ezután rekordonként beolvassuk a táblázatot, figyelve a kihagyott értékeket (Ez azért fog sikerrel járni, mert az adatokat *tabulátor* karakter választja el egymástól.):

```
ReadList[befele, Word, RecordLists -> True,
   NullWords -> True] // ToExpression
{{2.1, 77, 51}, {2.2, Null, 49}, {1.9, 80, Null}}
```

Végül pedig lezárjuk a bemenő áramot:

```
Close[bemeno];
```

• Az adatok rendezgetése

Előfordulhat, hogy az adatokból bizonyosakat akarunk csak használni, illetve hogy már beolvasás után azonnal el akarunk készíteni néhány elemi statisztikát. Ehhez nyújtanak segítséget a (lineáris algebra feladataihoz is hasznos)

```
Statistics`DataManipulation`
```

programcsomag függvényei:

```
BooleanSelect              DropNonNumericColumn
Column                     Frequencies
ColumnDrop                 LengthWhile
ColumnJoin                 QuantileForm
ColumnTake                 RowJoin
CumulativeSums             TakeWhile
DropNonNumeric
```

Ezek némelyikére — mint azt a nevek is elárulják — azért van szükség, mert a mátrixokat a program olyan listaként kezeli, amelynek elemei a *sor*okat tartalmazó listák. A statisztikában (és az adatbázis-kezelésben) viszont többnyire egy mátrix sorai jellemeznek egy egyedet, amint föntebb említettük, a vizsgálandó valószínűségi vektorváltozó egy koordinátájának értékei tehát egy *oszlop*ban helyezkednek el.

Néhány egyszerű példát mutatunk a fentiek közül az adatmanipulációs függvények alkalmazására:

```
<<Statistics`DataManipulation`
adat = {{a, 3}, {b, 6}, {c, 4}, {d, i}, {e, 5}, {f, 4}};
oszlop2 = Column[adat, 2]
{3, 6, 4, i, 5, 4}

ujadat = DropNonNumeric[oszlop2]
{3, 6, 4, 5, 4}

TakeWhile[oszlop2, NumberQ]
{3, 6, 4}
```

• Ábrázolás

Statisztikai célú számítások megkezdése előtt szinte mindig ajánlatos az adatokat különféle módokon ábrázolni, hogy valamilyen előzetes elképzelésünk legyen arról, milyen számítást érdemes végeznünk. Ilyenkor a `*List*` függvények valamelyike lehet a segítségünkre:

```
adat = {{0.1, 0.4}, {0.3, 0.9}, {0.4, 1.0}, {0.7, 1.3}
   {1.0, 1.3}, {1.2, 1.1}, {1.3, 0.8}, {1.3, 0.3}};
ListPlot[adat, PlotStyle -> AbsolutePointSize[2]]
elso = ListPlot[adat,
   PlotRange -> {0, 1.5},
   PlotJoined -> True,
   AxesOrigin -> {.8, .4},
   AxesLabel -> {"szele", "hossza"}]
```

Most az `adat`hoz egy kis zajt adunk. Úgy képzeljük, hogy az első koordináta pontosan beállított, a második pedig a számunkra érdekes, mért mennyiség:

```
veladat = Map[
   {#[[1]], #[[2]] + Random[Real, {-0.5, 0.5}]}&, adat];
masodik = ListPlot[veladat, PlotJoined -> True,
   PlotStyle -> Dashing[{0.02, 0.02}]]
```

Javasoljuk az Olvasónak, hogy nézze meg együtt a két ábrát:

```
Show[elso, masodik, AxesLabel -> {"ido", "ertek"},
   AxesOrigin -> {0, 1}, PlotRange -> {{0, 1.8}, {0, 1.8}},
   PlotLabel -> "Osszehasonlitas"]
```

Vektorváltozók alkalmazására is lássunk példákat. Kezdjük úgy a vizsgálatot, hogy a koordináták összefüggéseit páronként ábrázoljuk:

```
vektoradat = Table[{x, Random[ ], -x^2, x^3},
   {x, 0, 1, 0.05}];
```

Milyen viszonyban van egymással az első két koordináta?

```
ListPlot[(Take[#, 2]&) /@ vektoradat, Frame -> True,
   PlotStyle -> {AbsolutePointSize[2], RGBColor[0, 1, 0]}]
```

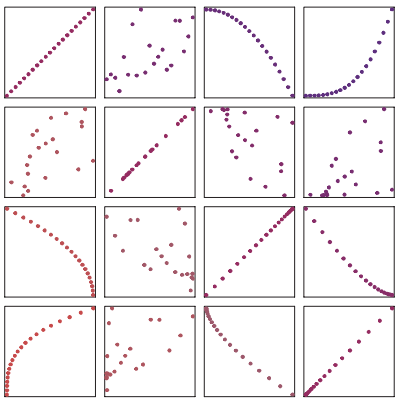
Benyomásunk valószínűleg az lesz, hogy a második független az elsőtől, ezt *függetlenségvizsgálat*tal akár ellenőrizhetnénk is.

Most elkészítjük a hasonló ábrákat az összes koordinátapárra, de az ábrát nem jelenítjük meg azonnal, amit a `DisplayFunction` opció értékének alkalmas megválasztásával érünk el:

```
sokrajz =
   Table[ListPlot[Map[{#[[i]], #[[j]]}&, vektoradat],
   Frame -> True, FrameTicks -> None, AspectRatio -> 1,
   PlotStyle -> {AbsolutPointSize[2],
   RGBColor[i/(i+j), 0, j/(i+j)]},
   DisplayFunction -> Identity], {i, 4}, {j, 4}]
```

Megnézzük egyszerre az ábrákat:

```
Show[GraphicsArray[sokrajz],
   DisplayFunction :> $DisplayFunction]
```



Az átlós elemek természetesen érdektelenek.

3.11.2. Statisztikák

A leggyakrabban használt statisztikai függvények vagy *statisztikák* kiszámításához használhatunk néhány belső függvényt:

```
Complement              Select
Count                   Sort
Intersection            Take
Max                     Union
Min
```

További gazdag választékot kínál a

```
Statistics`DescriptiveStatistics`
```

programcsomag, amelyben az alábbi függvények találhatók:

```
CentralMoment           PearsonSkewness2
DispersionReport        Quantile
GeometricMean           QuartileDeviation
HarmonicMean            Quartiles
InterpolatedQuantile    QuartileSkewness
InterquartileRange      RootMeanSquare
Kurtosis                SampleRange
KurtosisExcess          ShapeReport
LocationReport          Skewness
Mean                    StandardDeviation
MeanDeviation           TrimmedMean
Median                  Variance
MedianDeviation         VarianceMLE
Mode                    VarianceOfSampleMean
PearsonSkewness1
```

A `Statistics`DataManipulation`` programcsomagban is találhatunk még néhány hasznos függvényt:

```
CumulativeSums          QuantileForm
Frequencies
```

Megjegyzendő, hogy a különböző programcsomagokban azonos néven szereplő függvények nem teljesen azonos módon működnek.

Lássunk néhány példát. Egy minta átlagát a `Mean` függvény számolja ki:

```
adat = {1.3, 2.4, 5.2, 3.6, 2.4, 0.4}
Needs["Statistics`DescriptiveStatistics`"]
Mean[adat]
2.55
```

Vektorváltozókra ez csak értelemszerű módosítással alkalmazható. A fenti `vektoradat` sorai egy valószínűségi vektorváltozóra vonatkozó minta egyes elemei:

```
Map[Mean, Transpose[vektoradat]]
0.5, 0.50373, -0.341667, 0.2625
```

A *kiugró adatok*tól így szabadulhatunk meg:

```
kiugro =
   {1.7, 1.8, 1.5, 1.6, 1.9, 23.9, 1.6, 1.6, 1.7, 1.2}
TrimmedMean[kiugro, 0.3]
1.65
```

(A terjedelem 0,3 részét mind a két oldalon levágtuk a mintából.) Gyakran használt mintajellemzők a *centrális momentum*ok: ha értékük közel van a megfelelő elméleti értékhez, akkor érdemes alaposabban megvizsgálni az adott minta eloszlását:

```
CentralMoment[
   {1.7, 1.8, 1.5, 1.6, 1.9, 1.6, 1.6, 1.7, 1.2}, 3]
-0.00545953
```

Az eloszlás alakjának számos jellemzőjét egyszerre megkaphatjuk így:

```
{DispersionReport[#], LocationReport[#],
   ShapeReport[#]}&[kiugro]
{Variance -> 49.665, StandardDeviation -> 7.04734,
   SampleRange -> 22.7, MeanDeviation -> 4.01,
   MedianDeviation -> 0.1, QuartileDeviation -> 0.1},
{Mean -> 3.85, HarmonicMean -> 1.76206, Median -> 1.65},
{Skewness -> 2.27381, QuartileSkewness -> 0.5,
   KurtosisExcess -> 3.56113}}
```

Mivel az eredmény transzformációs szabályok összessége, ezért egyes részeire a szokásos módon hivatkozhatunk:

```
ferde = Skewness /. %
2.27381
```

A fenti eredményül kapott transzformációs szabályok bal oldalán álló azonosítók önállóan is használhatók függvényekként:

```
Skewness[kiugro]
```
```
2.27381
```

A következőkben *sűrűséghisztogram*ot készítünk. Behívjuk az adatgeneráláshoz és az ábrázoláshoz szükséges csomagokat:

```
Map[Needs,
     {"Statistics`ContinuousDistributions`",
      "Statistics`DataManipulation`",
      "Graphics`Graphics`"}]
```

Adatokat generálunk:

```
adat = Union[
   Table[Random[NormalDistribution[-1, 1]], {20}],
   Table[Random[NormalDistribution[1, 1]], {20}]];
```

Elkészítjük a rendezett mintát, majd ezek felhasználásával meghatározzuk, hogy a terjedelem egyes részintervallumaiba az adatok közül mennyi esik. Megjegyzendő, hogy a `Range` függvény nem állítja elő a minta terjedelmét, erre a `SampleRange` való:

```
rendezett = Sort[adat];
   osztalyokszama = 8;
   i = First[rendezett];
   a = Last[rendezett];
   h = SampleRange[adat]/osztalyokszama;
   gyak = BinCounts[adat, {i, a, h}]
```
```
{3, 3, 4, 11, 7, 2, 5, 4}
```

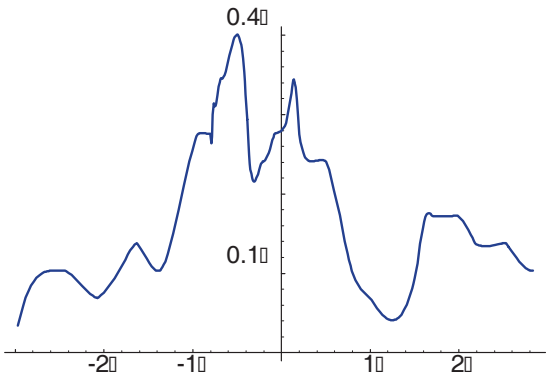
(A rendezett minta első elemét a `BinCounts` függvény figyelmen kívül hagyta.) Ez a részeredmény természetesen esetenként ettől eltérhet az aktuálisan előforduló véletlenszámok miatt:

```
BarChart[Transpose[{gyak, Range[i + h/2, a, h]}]]
```

(Figyeljük meg, hogyan változik az ábra, ha nyolc osztópont helyett többet vagy kevesebbet veszünk.) „Összekötjük” a pontokat interpolációval:

```
halmaz = Union[adat];
surfv = Interpolation[Transpose[{halmaz, Map[
   Count[adat, x_ /;
   (#-h/2) < x < (#+h/2)]/(h*Length[adat])&, halmaz]}]];
```

```
{tol, ig} = surfv[[1]];
Plot[surfv[x], {x, tol, ig}, Ticks -> {{-2,-1,1,2}, {.1, .4}}]
```



3.11.3. Becslések

Megmutatjuk — elsősorban folytonos eloszlásokon —, hogy hogyan lehet paramétereikre *pontbecslés*eket konstruálni a leggyakoribb módszerek alapján. Ezután áttérünk az *intervallumbecslés*ek, vagy másképpen *megbízhatósági (konfidencia-) intervallum*ok megszerkesztésére.

• A legnagyobb valószínűség (maximum likelihood) elve

Mutatunk egy példát arra, hogy hogyan szerkeszthetünk becslést a legnagyobb valószínűség elve alapján.

Független, nulla várható értékű normális eloszlású koordinátákkal bíró pont távolsága az origótól *Rayleigh-eloszlás*ú. Ha (egyetlen pozitív) paraméterét σ-val jelöljük, akkor sűrűségfüggvényének értéke az $x \in \mathbf{R}^+$ helyen

$$\frac{x}{e^{x^2/2\sigma^2}\sigma^2}.$$

Reprodukálható módon — erről gondoskodik a `SeedRandom` függvény — sorsolunk ilyen pontokat:

```
SeedRandom[100];
adatok = Table[Random[RayleighDistribution[2.0]], {20}]
```

Meg szeretnénk szerkeszteni a log-likelihood függvényt, ehhez szükségünk

van arra, hogy a Log függvény rendelkezzék a valós számoknál megszokott tulajdonságokkal:

```
Unprotect[Log];
Log[a_ b_] := Log[a] + Log[b];
Log[a_^b_]: = b Log[a];
Protect[Log];
maxval[sigma_] = Apply[Plus, Map[
Log[PDF[RayleighDistribution[sigma], #]]&, adatok]]
```

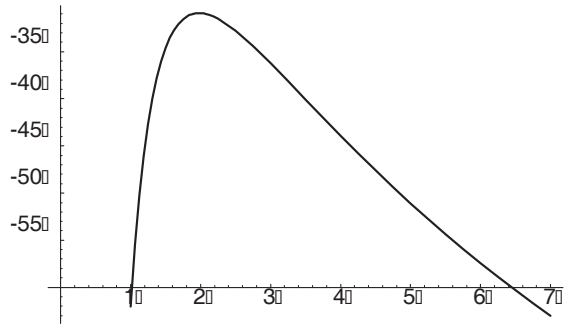
$$16.4297 - \frac{78.44}{\text{sigma}^2} - 40\ \text{Log[sigma]}$$

A *Mathematicá*ban haladottabbak inkább így számolnak:

```
maxval[sigma_] = Plus @@ ((
Log[PDF[RayleighDistribution[sigma], #]]&) /@ adatok)
```

Durván szólva azt a σ paraméterértéket szeretnénk elfogadni, amelynél az adott minta bekövetkezésének a valószínűsége maximális. A számolás előtt ábrázoljuk a likelihood-függvényt:

```
Plot[maxval[sigma], {sigma, 1.0, 7.0}]
```



A deriváltat nullával tesszük egyenlővé, ehhez a deriváltat szimbolikusan számoljuk. A kapott egyenleteket már numerikusan oldjuk meg:

```
NSolve[D[maxval[sigma], sigma] == 0, sigma]
{{sigma -> -1.78056}, {sigma -> 1.78056}}
```

Természetesen a pozitív értéket választjuk. Vajon a kapott pontok közül az értelmes másodikban a második derivált negatív-e, vagyis tényleg maximumhoz jutottunk?

```
D[maxval[sigma], {sigma, 2}] /. %[[2]]
-25.23359660282691
```

Az utolsó előtti lépésben használhatjuk a `FindMinimum` függvényt is; erre rá is kényszerülünk, amikor a deriváltat nem tudjuk szimbolikusan meghatározni. Másrészt ehhez a számoláshoz föl tudunk használni egy, az ábráról leolvasható, jó kezdeti becslést:

```
FindMinimum[-maxval[sigma], {sigma, 2.0}]
{30.038, {sigma -> 1.78056}}
```

Tanulságos megvizsgálni, hogy a becslés a minta elemszámának növelésével hogyan javul.

• Becslés a momentumok módszere alapján

A momentumok módszere alapján úgy becsülhetünk paramétereket, hogy egyenlővé tesszük az elméleti és az adatokból becsült momentumokat, majd ezeket az egyenlőségeket megoldjuk az eloszlás ismeretlen paramétereire. Alkalmazzuk a módszert a fenti feladat megoldására:

```
Solve[Mean[RayleighDistribution[sigma]] == Mean[adatok],
   sigma]
{{sigma -> 1.767684646314304}}
```

Látható, hogy a kapott becslés elég jól megegyezik az előzővel. Ezzel a módszerrel gyorsabban jutunk eredményhez, míg a legnagyobb valószínűség elve alapján meghatározott paramétereknek statisztikai szempontból bizonyítottan jó tulajdonságaik vannak.

• A legkisebb négyzetek módszere

Ennek alapján a paramétereket úgy választjuk meg, hogy a belőlük számolt, valamint a mért függvényértékek négyzetes eltérése a lehető legkisebb legyen. Erről a módszerről a regresszióról szóló résznél fogunk részletesebben szólni. (Itt is megpróbálkozhatnánk azzal, hogy a — valamilyen önkényes felosztással számolt — empirikus sűrűségfüggvény és az elméleti sűrűségfüggvény négyzetes eltérését minimalizáljuk, de a sűrűségfüggvények becslésére ennél sokkal jobb módszerek vannak.)

• Megbízhatósági intervallumok

Sok eszköz áll rendelkezésünkre megbízhatósági intervallumok szerkesztésére:

```
<<Statistics`ConfidenceIntervals`
adat1 = {2.1, 1.2, 3.5, 4.3, 4.1, 2.8, 1.9, 2.5, 3.8}
```

A Student-féle t-eloszlás alapján 95%-os szintű megbízhatósági intervallumot kaphatunk a várható értékre:

```
MeanCI[adat1]
{1.72263, 2.91737}
```

A további feltételek és változtatási lehetőségek innen láthatók, éppúgy, ahogyan az is, miként kell normális eloszlást feltételezve elkészíteni a megbízhatósági intervallumot:

```
Options[MeanCI]
{ConfidenceLevel -> 0.95, KnownStandardDeviation -> None,
   KnownVariance -> None}
```

Természetesen így is ugyanazt kapjuk:

```
StudentTCI[Mean[adat1], StandardErrorOfSampleMean[adat1],
   Length[adat1] - 1]
{1.72263, 2.91737}
```

Az utolsó paraméter a *szabadságfok.*

Két populáció várható értékének a különbségére is kaphatunk megbízhatósági intervallumot

- ismert szórásokat használva a normális eloszlás,
- egyenlő, de ismeretlen szórásokat feltételezve a t-eloszlás,
- becsült szórásokat feltételezve pedig a Welch-féle eloszlás

alapján:

```
adat2 = {1.1, 4.4, 2.2, 3.3, 4.4, 2.2, 1.1, 2.2, 3.3}
MeanDifferenceCI[adat1, adat2, EqualVariances -> True]
{-1.3841, 0.64632}
```

Ha nem feltételezhetjük, hogy azonosak a szórások, akkor természetesen nagyobb intervallumot kapunk:

```
MeanDifferenceCI[adat1, adat2]
{-1.42443, 0.686655}
```

Kiderül, hogy a megbízhatósági szint is változtatható:

```
Options[MeanDifferenceCI]
```

Egy populáció szórásnégyzetére a χ^2-eloszlás alapján kaphatunk megbízhatósági intervallumot:

```
VarianceCI[adat1]
{0.32992, 2.32411}

VarianceCI[adat2]
{0.705401, 5.6745}
```

Két szórásnégyzet hányadosára 90%-os megbízhatósági intervallumot így kaphatunk:

```
VarianceRatioCI[adat1, adat2, ConfidenceLevel -> 0.9]
{0.113119, 1.45662}
```

Ha olyan intervallumot keresünk, amelyre nagyobb annak a valószínűsége, hogy tartalmazza a szórásnégyzetek hányadosát, akkor az természetesen nagyobb lesz:

```
VarianceRatioCI[adat1, adat2, ConfidenceLevel -> 0.95]
{0.103512, 1.85008}
```

Ha valamilyen ismert eloszlású statisztikára akarunk megbízhatósági intervallumot szerkeszteni, akkor előbb alkalmazzuk az eredeti adatokra valamelyik leíró statisztikai függvényt, majd az alábbi függvények közül válogathatunk: `NormalCI, StudentTCI, ChiSquareCI, FRatioCI`.

3.11.4. Hipotézisvizsgálat

A matematikai statisztika tipikus feladata az, amelynél azt akarjuk eldönteni, hogy valamely feltevés (*hipotézis*) a mérési adatoknak ellentmond-e vagy nem. Általában azt vizsgáljuk tehát, hogy ha sokszor végeznénk el a méréseinket, az esetek hányad részében kapnánk az adott esethez hasonló jellegű eredményeket a tett hipotézis mellett. Ha az esetek nagy részében ilyen típusú eredményt kapunk, akkor azt mondjuk, hogy adataink nem mondanak ellent a tett hipotézisnek, míg ha valószínűtlen az eredmény, akkor a hipotézist elvetjük.

Külön programcsomag áll rendelkezésünkre hipotézisvizsgálat céljára, ennek neve:

```
Statistics`HypothesisTest`
```

Fontosabb függvényei:

```
ChiSquarePValue          NormalPValue
FRatioPValue             StudentTPValue
MeanDifferenceTest       VarianceRatioTest
MeanTest                 VarianceTest
```

Kétoldali ellenhipotézissel szemben ellenőrizni szeretnénk, hogy két minta azonos szórásúnak tekinthető-e:

```
<<Statistics`HypothesisTests`
VarianceRatioTest[adat1, adat2, 1,
   TwoSided -> True, FullReport -> True]
{FullReport ->
   Ratio     TestStat  NumDF  DenDF,
   0.451024  0.451024  9      8
   FRatioDistribution, TwoSidedPValue -> 0.257104}
```

Megkaptuk tehát a szórásnégyzetek arányát, a próba alapjául szolgáló statisztikát (ez éppen az arány), a számláló és a nevező szabadsági fokát, azt, hogy milyen eloszlást használ a program a próbához, végül azt a valószínűséget, amilyen szinten a két szórásnégyzet megegyezésének hipotézisét, a *nullhipotézis*t, el kell vetnünk. Vagyis az esetek mintegy 25,7 %-ában számíthatunk arra, hogy ilyen szórásnégyzetarányt kapunk, annak ellenére, hogy a két szórásnégyzet megegyezik.

Természetesen ugyanarra az eredményre jutunk, ha a szórásnégyzeteket, valamint a számláló és a nevező szabadságfokát magunk szolgáltatjuk:

```
FRatioPValue[Variance[adat1]/Variance[adat2],
   Length[adat1] - 1, Length[adat2] - 1,
   TwoSided -> True]
TwoSidedPValue -> 0.257104
```

3.11.5. Korreláció- és regresszióanalízis. Szórásanalízis.

Valószínűségi változók közötti összefüggések elemzésére szolgál a címben említett terület.

• Lineáris regresszió

Ha csak *lineáris paraméterbecslés*t akarunk végezni a legkisebb négyzetek módszere alapján (azaz feltételezzük, hogy az illesztendő függvény *paramétereiben* lineáris, és nem akarjuk az eredményeket különösebb statisztikai elemzésnek alávetni), akkor elegendő ennyit írnunk:

```
adat = {{0.211, 150}, {0.248, 189}, {0.351, 253},
   {0.182, 142}, {0.091, 97}, {0.167, 124},
   {0.232, 172}, {0.055, 90}, {0.284, 209},
   {0.138, 107}};
Fit[adat, {1, x, x^2}, x]
70.9638 + 196.053 x + 965.336 x^2
```

Ez a feladat tehát a `Fit` belső függvénnyel megoldható.

• Szórásanalízis

Mód van azonban arra is, hogy részletesebb elemzést: *szórásanalízist* végezzünk. Ehhez (valamint az ezzel rokon kísérlettervezéshez) a

```
Statistics`LinearRegression`
```

programcsomag alábbi függvényei használhatók:

```
DesignMatrix                Regress
DesignedRegress
```

Vegyük elő még egyszer a fenti példát:

```
<<Statistics`LinearRegression`
Regress[adat, 1, x, x^2, x]
ParameterTable ->
      Estimate SE       TStat   PValue,
  1   70.9638  10.7948  6.5739  0.000311792

  x   196.053  115.029  1.70437 0.132087

  x^2 965.336  280.95   3.43597 0.0108966
 RSquared -> 0.985311, AdjustedRSqured -> 0.98111114,
 EstimatedVariance -> 52.429,
```

```
ANOVATable ->

       DoF SoS      MeanSS   FRatio   PValue
Model  2   24617.1  12308.5  234.766  0

Error  7   367.003  52.429

Total  9   24984.1
```

A táblázat néhány elemének jelentése nyilvánvaló, a többiek jelentését illetően a matematikai statisztika tankönyveire utalunk. A `Regress` függvény számos opciója közül kiemeljük a `Weights` nevűt, amellyel a súlyozást állíthatjuk be statisztikai modellünknek megfelelően.

A `DesignedRegress` és a `DesignMatrix` függvény kísérlettervezéshez használható.

Amennyiben az illesztendő függvény még paramétereiben sem lineáris, a

```
Statistics`NonlinearFit`
```

programcsomag `NonlinearFit` függvénye lehet segítségünkre. Illesszünk $x \mapsto a+bx+e^{cx}$ alakú függvényt előző adatainkra (A futási eredményeknek csak az utolsó lépését közöljük.):

```
NonlinearFit[adat, a + b x + Exp[c x], x,
   {{a, 70}, {b, 100}, {c, 3}},
   ShowProgress -> True, AccuracyGoal -> 5,
   PrecisionGoal -> 5]
...
Iteration:13 ChiSquared:577.2309320244032 Parameters:
   {53.395, 442.14, 11.0429}
{a -> 53.395, b -> 442.14, c -> 11.0429}
```

3.11.6. Idősorok

Időben változó véletlen folyamatok közül a legegyszerűbb, legáttekinthetőbb szerkezetűek talán a diszkrét idejű, stacionárius *idősor*ok. Elméletüket illetően a [82] cikkgyűjteményre utalunk, alkalmazásukra Csáki P. példáját [82, 79–83. oldal] mutatjuk meg. A *Drosophila melanogaster* törzs albumin szekrécióját 60 generáción keresztül mérték. Ez a sorozat az öröklődési szabályok miatt egy elsőrendű autoregresszív folyamat realizációjának tekinthető 1/2 autoregressziós együtthatóval és 5 várható értékkel:

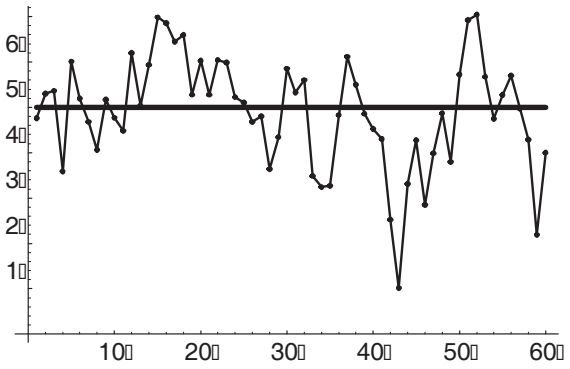
```
drosophila = {
   4.761, 5.302, 5.361, 3.582, 6.008,
   5.196, 4.677, 4.063, 5.171, 4.767,
   4.484, 6.196, 5.042, 5.940, 6.984,
   6.859, 6.446, 6.597, 5.276, 6.027,
   5.279, 6.043, 5.987, 5.227, 5.102,
   4.679, 4.801, 3.636, 4.341, 5.850,
   5.327, 5.601, 3.488, 3.245, 3.271,
   4.832, 6.116, 5.500, 4.857, 4.525,
   4.297, 2.529, 1.016, 3.316, 4.276,
   2.851, 3.989, 4.871, 3.794, 5.719,
   6.922, 7.046, 5.678, 4.747, 5.273,
   5.704, 4.975, 4.291, 2.188, 4.004};
```

Levonjuk a várható értéket, hogy egy nulla várható értékű folyamatot kapjunk:

```
drophil = drosophila - 5;
```

Ábrázoljuk a mért értékeket:

```
ab1 = ListPlot[drosophila, PlotJoined -> True]
ab2 = ListPlot[drosophila,
   PlotStyle -> AbsolutePointSize[2]]
ab3 = Plot[5, {gen, 1, 60},
   PlotStyle -> Thickness[0.01]]
Show[ab1, ab2, ab3]
```



```
t = Length[drosophila]
60
```

Meghatároztuk a nemzedékek számát. Szeretnénk meghatározni a legfontosabb jellemzőket:

```
<<Statistics`DescriptiveStatistics`
#[drosophila]& /@ {Mean, Variance, VarianceMLE}
{4.89887, 1.48025, 1.45558}
```

Az utóbbi a *korrigált tapasztalati szórásnégyzet*. Most kiszámítjuk az empirikus autokovariancia-függvényt:

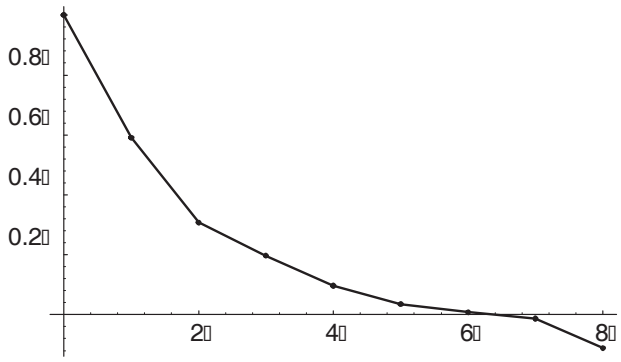
```
c = Table[1/t Sum[
   (drophil[[i]] - Mean[drophil])*
   (drophil[[i+k]] - Mean[drophil]), {i, 1, t - k}], {k, 0, 8}]
{1.45558, 0.859109, 0.44711, 0.28646, 0.140774,
   0.0508499, 0.011371, -0.0191493, -0.163013}
```

Ebből az empirikus autokorrelációs függvény egyszerűen adódik:

```
r = c/c[[1]];
```

Ábrázoljuk is ezt a függvényt.

```
auto1 = ListPlot[Transpose[Range[0, 8], r],
   PlotJoined -> True, PlotStyle -> Thickness[0.008]]
auto2 = ListPlot[Transpose[Range[0, 8], r],
   PlotStyle -> AbsolutePointSize[3]]
Show[auto1, auto2]
```



Következzék az autokorrelációs mátrix:

```
autokorrmatrix = Table[
   Table[RotateRight[Take[r, k], j], {j, 0, k-1}],
   {k, 1, 8}]
```

Most számítsuk ki a parciális autokorrelációs függvényt:

```
parautokor = Table[
   Det[Append[Drop[autokorrmatrix[[k]], -1],
   Rest[Take[r, k + 1]]]]/
   Det[autokorrmatrix[[k]]], {k, 1, 8}]
{0.590216, -0.0632035, 0.047947, -0.0349174,
   -0.00855083, -0.0023358, -0.0101465, -0.0922125}
```

A mintából becsült első parciális autokorreláció értéke 0,59, a nagyobb indexűeké beleesik a nulla körüli megbízhatósági intervallumba, azaz nem tér el szignifikánsan a nullától.

A Yule–Walker-egyenletek esetünkben így írhatók fel:

```
egy[p_] := Append[Table[
   c[[k+1]] == Sum[a[i] c[[k+1-i]], {i, 1, k}],
   {k, 1, p}],
   c[[1]] == Sum[a[i] c[[i+1]], {i, 1, p}] + var]
   egy[1]
{0.859109 == 1.45558 a[1],
   1.45558 == var + 0.859109 a[1]}
```

Most pedig néhány p értékre megoldjuk az egyenleteket:

```
Solve[egy[1], {a[1], var}]
{a[1] -> 0.590216, var -> 0.948524}

Solve[egy[2], {a[1], a[2], var}]
{a[1] -> 0.590216, a[2] -> -0.0411863,
   var -> 0.966939}

Solve[egy[3], {a[1], a[2], a[3], var}]
{a[1] -> 0.590216, a[2] -> -0.0411863,
a[3] -> 0.039814, var -> 0.955533}
```

A példa fő tanulsága, hogy a matematikai képleteket szinte változtatás nélkül beírhattuk a programba.

A legkézenfekvőbb, amit egy véletlen hibával terhelt mérési sorozattal tehetünk, hogy *simítjuk*. Ez a `Statistics`MovingAverage`` programcsomag `MovingAverage` függvényével végezhető, akár szimbolikusan is:

```
adat = {a, b, c, d , e, f};
MovingAverage[adat, 2]
```

$$\left\{\frac{a+b+c}{3}, \frac{b+c+d}{3}, \frac{c+d+e}{3}, \frac{d+e+f}{3}\right\}$$

Vektorértékű sorozatokra a `MovingAverage` függvény ugyanígy használható. Az adatokat úgy kell elhelyeznünk, hogy az egy időponthoz tartozó értékek az adatmátrix oszlopait alkossák.

3.11.7. Összetett feladatok, új módszerek

Néhány olyan összetett feladatot sorolunk fel, amelyeknek a megoldása megtalálható valamelyik programcsomagban vagy idézett cikkben. Tukey eredményei alapján *nemparaméteres simítás*ra vonatkozó eljárásokat tartalmazó programcsomagot ír le [88, 331. oldal]. Ugyanott megad egy *genetikai algoritmus*t a tőzsdén szerezhető haszon maximalizálására.

Martin fejlettebb módszereket leíró cikkében [55] bemutatja a sűsűségfüggvények *Edgeworth-féle sorfejtés*ét, ahol jól használható a `Series` függvény a szükséges szimbolikus számolások elvégzéséhez. Megmutatja, hogy hogyan lehet finomítani a *nemlineáris paraméterbecslés* eredményeként kapott paraméteregyüttest, és hogy miként kell értékelni az eredményeket. Ismertet egy programot a görbületen alapuló diagnosztikára is, amely Bates és Watts egy 1980-as cikkén alapul. Ezután tárgyal robosztus becsléseket, a medián fogalmának (nemtriviális) kétdimenziós általánosítását, valamint egy módszert, amely adatok térbeli elhelyezkedését vizsgálja statisztikai szempontból, így akár a térinformatikához történő statisztikai hozzájárulásnak is tekinthető.

3.11.8. Gyakorlatok és feladatok

1. Ábrázoljuk az u-próba erőfüggvényét a paraméter és a mintaelemszám függvényében.

2. Írjunk függvényt, amellyel a Sarkadi-próba [85, 137. oldal] alapján adott mintáról eldönthető, hogy valamely minta normális eloszlásból származónak tekinthető-e, ha sem a szórást, sem a várható értéket nem ismerjük.

3. Döntsük el véletlenített (randomized) próba alapján, hogy egy k és egy l elemű minta közül melyik a "nagyobb".

4. A *Mathematica* programozásáról

Megemlítettük már azt, hogy a *Mathematica* magasszintű programozási nyelvnek is tekinthető. Az előző fejezetben számos példát láttunk arra, hogy a belső és a külső függvények összekapcsolásával hogyan definiálhatunk új eljárást, azaz hogyan írhatunk programot. Ebben a fejezetben azt szeretnénk kihangsúlyozni, hogy a *Mathematica* használható különböző stílusú programok készítésére.

- Írhatunk programokat a hagyományos *procedurális* (vagy imperatív) programozási nyelvek (Basic, C, FORTRAN, Pascal) stílusában, szekvenciákat, eljárásokat, ciklusokat, blokkokat és feltételes utasításokat használva.
- Utánozhatjuk a logikai vagy szabályalapú nyelvek (PROLOG) logikáját, transzformációs szabályok megadásával és mintázatok illesztésével.
- *Funkcionális* programozási nyelvek (APL, LISP, LOGO, Miranda) stílusában is írhatunk programokat függvényekkel, operátorokkal, rekurzióval és a programszerkezetet befolyásoló műveletekkel. (A funkcionálissal majdnem azonos jelentésű a *listakezelő*.)
- *Objektum-orientált* programozási nyelvek (C++, SIMULA, SMALLTALK, Actor, a Turbo Pascal 5.5 utáni változatai) eszközeit (osztályok, polimorfizmus, öröklődés) is alkalmazhatjuk.

Az első három programozási stílust beépített belső és külső függvények támogatják. Ha objektum-orientált stílusban szeretnénk programjainkat megírni, akkor ehhez a MathSource-on a 0200–293 és a 0205–186 szám alatt található kiegészítéseket használhatjuk, vagy a [29] könyv 9. fejezetére támaszkodhatunk.

A példaként említett programnyelvek általában nem képviselik tisztán valamelyik programozási stílust, hanem egyik vagy másik stílus jellegzetességei dominálnak bennük. Bár a *Mathematicá*ról azt mondtuk, hogy ebben mindegyik stílus teljes egészében megvalósítható, ez lényegét tekintve (több szerző szerint) *funkcionális programozási nyelv*. Ezt a módszert éppen matematikai feladatok hatékony megoldására fejlesztették ki [8], hiszen

> fölösleges részeikre szedni a matematikai objektumokat, csak azért, hogy azután újra összerakjuk őket.

(Lásd [29], 8. fejezet.)

Különböző stílusban megírt eljárások kiértékeléséhez szükséges időtartamok lényegesen különbözők lehetnek, adott feladat megoldásához a leghatékonyabb stíluselemek kiválasztása sokszor nem egyszerű feladat. Ehhez a programcsomagok szöveges állományainak tanulmányozása adhat segítséget. Az `Examples` és a `ProgrammingExamples` alkönyvtárakban található rövid programcsomagokból sokat meríthet az érdeklődő Olvasó. R. Maeder [51] könyvéből lépésről lépésre tanulhatjuk meg nagyobb programok írásának fortélyait. J. W. Gray [29] könyve pedig különböző stílusú programok írására mutat be számos példát. A MathSource-ról megszerezhető dokumentumok (például a 0203–904, 0203–926, 0205–748 szám alattiak) is számos hasznos információt tartalmaznak a *Mathematica* programozásával kapcsolatosan.

A programozásnál is nagy jelentősége van annak, hogy szimbólumokat is tudunk kezelni, nem csak számokat.

A megírt programok többféleképpen futtathatók: begépelés után azonnal (hiszen a *Mathematica* alapvetően értelmező módban, másképpen interaktívan működik), állományból kötegelt üzemmódban, valamint a megírt programrészeket elraktározhatjuk bármikor felhasználható programcsomagokban is. Mód van arra is, hogy a *Mathematica* nyelvén és más nyelven megírt programjainkat összeépítsük a MathLink program felhasználásával.

Oktatási szempontból különös jelentősége van annak, hogy a *Mathematicá*n belül *mindegyik* programozási stílus teljes egészében bemutatható.

4.1. Programnyelvi elemek

Az alábbiakban felsoroljuk a *Mathematicá*nak mint programozási nyelvnek az elemeit:

- karakterkészlet,
- alapszavak,
- adattípusok (füzérek, számok, szimbólumok; adatellenőrzés),
- adatszerkezetek (kifejezés, lista, függvény, tömb),
- transzformációs szabályok, mintázatok,
- adatszerkezetek átalakításai (értékadás, listaműveletek),

- programszerkezetek (szekvencia, elágazás, ciklus; blokk, modul, programcsomag),
- nyomkövetés (`FullForm`, `On`, `Off`, `Short`, `Trace`, `TraceForm`),
- üzenetek (`Check`, `Message`).

Ezek egy részével a 2. fejezetben már foglalkoztunk. A továbbiakban csak az eddig nem tárgyalt elemekről szólunk.

• Szekvencia

Például a Pascal nyelvben szereplő összetett utasításnak felel meg a *Mathematicá*ban a *szekvencia* fogalma. Ha arra van szükségünk, hogy egymás után végrehajtandó utasítássorozatot *egyetlen* utasításnak tekintsünk, akkor az utasításokat egymástól pontosvesszővel választjuk el, és gömbölyű zárójelek közé tesszük:

```
(ut1; ut2; ut3)
```

Ez rövid alakja a

```
Sequence[ut1, ut2, ut3]
```

kifejezésnek. (A `;` egyébként a `CompoundExpression` függvény rövid, infix alakja.)

• Elágazás

Számos eszköz áll rendelkezésünkre alternatívák kezelésére. Előfordul, hogy *egy logikai kifejezés* értékétől szeretnénk függővé tenni, hogy három kifejezés közül melyiket értékelje ki a program. Erre szolgál a legegyszerűbb a feltételes utasítás:

```
If[feltetel, haigaz, hahamis, egyiksem]
```

Ha a `feltetel` értéke igaz (`True`), akkor a `haigaz` utasítás, ha a `feltetel` értéke hamis (`False`), akkor a `hahamis` utasítás, ha pedig a `feltetel` értéke nem számítható ki (például azért, mert a benne szereplő változók nem kaptak értéket), akkor az `egyiksem` utasítás hajtódik végre. A függvény utolsó vagy utolsó két argumentuma (értelemszerű hatásokkal) elhagyható:

```
If[x < y, Quit[ ], 4, 8]
8
```

```
x = 5; y = 6;
If[x < y, 3, 4, 8]
3
```

Ha többirányú elágazással van dolgunk, akkor is használhatjuk az `If` függvényt, de ebben az esetben bonyolult kifejezésekhez jutunk. Ez elkerülhető, ha a

```
Which[teszt1, ut1, teszt2, ut2, ...]
```

konstrukciót alkalmazzuk. Ekkor az első olyan utasítás hajtódik végre, amelyik előtt igaz értéket felvevő logikai kifejezés áll. Például:

```
f[a_] := Which[a == 1, a, EvenQ[a], a/2, True, 3a + 1]
f /@ {5, 1, 24}
{16, 1, 12}
```

`EvenQ` értéke pontosan akkor `True`, amikor argumentuma páros. — A fenti `f` függvény a *Collatz-féle probléma* (vagy $3a+1$ probléma) vizsgálatánál használható; lásd még az `Examples` és a `ProgrammingExamples` (egymástól különböző!) `Collatz` nevű prgoramcsomagját.

Ha egy (nem feltétlenül logikai) kifejezés értékétől akarunk függővé tenni egy kifejezést, akkor használjuk a `Switch` függvényt:

```
Switch[kif, ertek1, ut1, ertek2, ut2, ...]
```

Oldjuk meg az előbbi feladatot ezzel a konstrukcióval is:

```
ff[a_] := Switch[a,
                 1, 1,
                 2 Quotient[a, 2], Quotient[a, 2],
                 _, 3a + 1]
ff /@ {5, 1, 24}
{16, 1, 12}
```

Látható, hogy az `_` jelet itt is tetszőleges kifejezés jelölésére használtuk.

Előfordul az is, hogy egy *általánosított* listából szeretnénk kiválasztani az első néhány olyat, amely kielégít egy bizonyos kritériumot (logikai értékeket felvevő függvényt). Ilyenkor ezt írjuk:

```
Select[altlista, krit, elsonehany]
```

Például:

```
Select[{3, 4, 5, 2, x, x^2, x^3}, NumberQ, 2]
{3, 4}
```

Ha viszont egy (általánosított) lista adott mintázatnak megfelelő elemeit akarjuk kiválasztani, akkor a

```
Cases[altlista, mintazat]
```

utasításra van szükségünk:

```
Cases[{3, -4, 5, -2}, x_ /; x<0]
{-4, -2}

Cases[3 + (-4)*Sin[5 x]^x^(-2), x_ /; x<0]
{-4, -2}
```

Az adott mintázatnak megfelelő listaelemek számának meghatározására két példa:

```
Count[{3, 4, x, x^2, x^3}, x^_]
2

Count[{3, 4, x, x^2, x^3}, x^_.]
3
```

Itt a

```
Count[altlista, mintazat]
```

szerkezetet használtuk, és a második esetben utaltunk a kitevő alapértelmezésbeli értékére.

• Ciklus

A szokásos ciklusutasítások mindegyike használható a *Mathematicá*ban. A legegyszerűbb közülük talán a `Do`. Lássunk egy példát:

```
L[a_] := N[Log[a]]
Do[Print["a=", a, " L[a]=", L[a]], {a, 1, 3}]
a=1 L[a]=0
a=2 L[a]=0.693147
a=3 L[a]=1.09861
```

Látható, hogy a ciklus törzsét jelentő utasítás áll elöl, másodikként pedig a ciklusváltozót, kezdő- és záróértékét tartalmazó listát kell írnunk.

A `For` és a `While` egyaránt *elöltesztelő*, általános alakjuk:

```
For[start, teszt, novekmeny, torzs]
While[teszt, torzs]
```

Próbáljuk ki például a következő két parancsot:

```
For[i = 1; t = x, i^2<10, i++, t = t^2 + i; Print[t]]
```

$$1 + x^2$$

$$2 + (1 + x^2)^2$$

$$3 + 2 + (1 + x^{2^2})^2$$

Ügyeljünk a különféle írásjelekre!

```
n = 17;
While[(n = Floor[n/2]) != 0, Print[n]]
```

```
8
4
2
1
```

Bár az értékadás és a feltételvizsgálat összekapcsolása tömör írásmódot tesz lehetővé, mégis ezek a módszerek a *Mathematicá*ban általában kevésbé hatékonyan működnek, mint a funkcionális stílust támogató

`FixedPoint`	`Nest`
`FixedPointList`	`NestList`
`Fold`	`Table`
`FoldList`	

függvények. Ezt illusztrálják a következő példák. Elsőként meghatározzuk a `Cos` függvény fixpontját:

```
cos[x_] := N[Cos[x]]
FixedPoint[cos, 1]
```

```
0.739085
```

Másodikra szimulálunk egy *szimmetrikus véletlen bolyongás*t:

```
bolyongas[n_] :=
   FoldList[Plus, 0, Table[Random[ ]-1/2, {n}]]
ListPlot[bolyongas[30], PlotJoined -> True]
```

Harmadikként bemutatjuk Novak megoldását a Gram–Schmidt-féle ortogonalizálásra, amely a `LinearAlgebra`Orthogonalization`` csomagjában is szerepel. A példa részletes elemzését az Olvasóra hagyjuk.

```
vetites[v1_, v2_] := (v1 . v2) v2/(v2 .v2)
listaravetites[v1_, vektorlista_] :=
   Plus @@ (vetites[v1, #]& /@ vektorlista)
GramSchmidt[matrix_] :=
   Fold[Join[#1, {#2 - listaravetites[#2, #1]}]&,
   { }, matrix]
```

• Lokális változók használata

A `With`, a `Module` és a `Block` belső függvények lehetővé teszik lokális állandók és lokális nevű, illetve lokális értékű változók használatát. Ezek a konstrukciók a strukturált nyelvekben megszokott *blokk*oknak felelnek meg. Lokális változókat használhatunk a `Block` és a `Module` szerkezetben.

A modul használatát az indokolhatja, hogy

- ne ütközzenek a *változók*;
- egyszer kiszámolt értéket többször akarunk használni;
- növelni akarjuk a program áttekinthetőségét.

A kettő közül általában a `Module` részesítendő előnyben. Ha viszont interaktív számolásokra készülünk, akkor ajánlatosabb a `Block` használata. Ezzel elkerülhetjük a változó*nevek* ütközését.

Tekintsük az alábbi utasítássorozatot:

```
m = i^2
```

$$i^2$$

```
Block[{i = a}, i + m]
```

$$a + a^2$$

```
Module[{i = a}, i + m]
```

$$a + i^2$$

Látható, hogy az első esetben a blokk `i` lokális változója az `i + m` kifejezés kiértékelésében végig szerepet játszik. A modul kiértékelésénél a lokális változóra vonatkozó értékadás csak a kiértékelendő `i + m` kifejezésben explicite szereplő helyre vonatkozik. Ezért mondjuk azt, hogy az első szerkezetben lokális *értékű*, míg a másodikban lokális *nevű* változó szerepel.

A `Module` függvénynek két argumentuma van. Az első a lokális változók listája, amelyben esetleg kezdeti értékadás is szerepel. A második argumentum (általában összetett) kifejezés. Az eredmény az utolsó utasítás végeredménye. Figyeljünk arra, hogy a `;` jel erősebb, mint a `,` jel, eltérően a legtöbb más nyelvben megszokottaktól.

A következő példában lokális *állandó* szerepel, azaz olyan azonosító, amelyhez egyetlen egyszer rendelünk hozzá valamilyen értéket (Ezért azt mondhatjuk, hogy a `With` konstrukció a `Module` konstrukciónak speciális esete: mintha csak a kezdeti értékadás hajtódnék végre.):

```
t = 17
17

w[x_] := With[{t = x + 1}, t + t^3]
w[a]
                    3
1 + a + (1 + a)

t
17
```

Tehát a program különbséget tett `t` két különböző előfordulása között, a globális `t` értékét megtartotta.

Programjainkban használhatjuk a más nyelvekben megszokott megszakításokat és visszatéréseket is:

```
Break                    Label
Catch                    Return
Check                    Throw
Continue
Goto
```

4.2. Gyorsaság, gyorsítás

Ebben a szakaszban azzal fogunk foglalkozni, hogy milyen gyorsan számol ki valamit a *Mathematica*, és hogy ez a számolási idő milyen módszerekkel rövidíthető.

4.2.1. Tipikus műveletek időigénye

Mielőtt rátérnénk a programok gyorsításának módszereire, érdemes tájékozódásul néhány időadatot megnéznünk. Ezek értékelésénél figyelembe kell venni, hogy nemcsak a *Mathematica* aktuálisan használt változatának számától és az adott gép konfigurációjától függenek, hanem még attól is, hogy az adott alkalmon belül az adott művelet mikor, milyen előzmények után került sorra.

A táblázat bemutatása előtt, annak megértéséhez és reprodukálásához némi segítséget adunk.

Ha meg akarjuk tudni, hogy mennyi ideig dolgozik a központi egység az $(a+bx)^{700}$ polinom kifejtésén, akkor a következőt kell beírnunk:

```
Timing[Expand[(a + b x)^700]]
```

Ennek hatására visszakapjuk a művelet elvégzéséhez szükséges időtartamot másodpercben és az eredményt. Az időtartam nullának mutatkozik, ha kisebb egy minimális értéknél, ami rendszerint a másodperc hatvanad- vagy századrésze. Amikor a gép teljesítőképességét vizsgáljuk, előfordulhat, hogy magára a számítási eredményre nincs is szükségünk. Ekkor az eredmény képernyőre való kiírását az utasítás után tett pontosvesszővel akadályozzuk meg:

```
Timing[Expand[(a + b x)^700];]
```

Mátrixok invertálásának időigényét a különösen rosszul kondicionált Hilbert-féle mátrixok (amelyek i-edik sorában és j-edik oszlopában az $1/(i+j)$ szám áll) példáján tanulmányoztuk.

A táblázatban a „$(3x+4y+5z)^{20}$ szorzattá” kifejezés annak a rövidítése, hogy előbb kifejtjük az adott polinomot, majd megvizsgáljuk, hogy mennyi idő alatt alakítja az összeget szorzattá a program. Ekkor tehát két egymás utáni művelet együttes elvégzéséhez szükséges idő az, amit mérünk. Ehhez a következőket írjuk be:

```
Timing[Factor[Expand[(3x + 4y + 5z)^20]];]
```

Fibonacci$_{20}$ a rekurzióval a következő pontban definiált Fibonacci-sorozat 20-adik tagját jelenti.

A megadott faktorizálandó számok pedig olyan nagy számok, amelyeknek legkisebb prímtényezőjük is nagy, ezeket szokás gépek gyorsaságának ellenőrzésére használni.

A méréseket IBM PC típusú gépeken végeztük; egy-egy adatot közlünk:

Feladat	**Időtartam** (sec) Gép I. 486 50 MHz 8 MB	Gép II. 386SX 20 MHz 10 MB
11^{11111}	0.879	13.566
$(a+bx)^{100}$	0.164	2.472
$(a+bx)^{500}$	0.934	13.787
$(a+bx)^{700}$	1.373	18.894
$(a+bx)^{1000}$	2.032	25.43
$(a+bx)^{2000}$	4.944	59.594
$(1/(i+j-1))^{-1}_{5\times5}$	0.604	2.087
$(1/(i+j-1))^{-1}_{10\times10}$	0.659	9.777
$(1/(i+j-1))^{-1}_{20\times20}$	8.128	143.19
$(1/(i+j-1))^{-1}_{30\times30}$	54.156	856.171
$(3x+4y+5z)^{10}$ szorzattá	2.636	19.553
$(3x+4y+5z)^{20}$ szorzattá	7.58	114.904
$(3x+4y+5z)^{30}$ szorzattá	27.792	396.119
Fibonacci_{10}	0.055	0.494
Fibonacci_{20}	0.055	0.824
Fibonacci_{25}	-1.1279910^{-13}	0.22
Fibonacci_{50}	0.109	1.098
FactorInteger$[2^{67}-1]$	10.82	140.333
FactorInteger$[2^{201}-1]$	Nem bontja föl	Nem bontja föl

Ha azt gyanítjuk, hogy valaminek a kiszámolása sok időt fog igénybe venni, akkor érdemes a következőképpen eljárni. Készítünk egy olyan függvényt, amelyben a feladat mérete az egyetlen változó, és különböző méretek esetén a `Timing` függvény felhasználásával megmérjük, hogy a feladat végrehajtásának időtartama hogyan függ a mérettől. Sajnos, bár ez nem meglepő, az érdekes esetekben azt fogjuk tapasztalni, hogy a végrehajtási idő a méretnek exponenciális függvénye. Az előzetes kísérletezésbe fektetett munka ekkor sem volt fölösleges, mert extrapolációval megbecsülhetjük, hogy az igazi méret mellett feladatunk elfogadható időn belül megoldható-e.

Különösen sokáig tart az *ábrák* és *hangok* előállítása; ilyen példát nem szerepeltettünk a táblázatban.

4.2.2. Gyorsítás emlékező függvényekkel

Ha a szokásos módon, a `SetDelayed` (röviden: `:=`) utasítás felhasználásával adunk meg egy függvényt, akkor annak értékét minden meghívás alkalmával kiszámolja a program. Előfordul viszont, hogy egy függvénynek egy adott helyen fölvett értékét sokszor akarjuk fölhasználni. Ekkor a *Mathematica* rávehető arra, hogy megjegyezze a már egyszer kiszámolt értéket. Ez a technika különösen jól használható rekurzív módon definiált függvényeknél:

```
fibgyors[x_] :=
   fibgyors[x] = fibgyors[x - 1] + fibgyors[x - 2]
fibgyors[0] = fibgyors[1] = 1;
Timing[fibgyors[12]]
```

```
{0.329 Second, 233}
```

Ha most kiadjuk a `?fibgyors` kérdést, akkor a válasz mutatja, hogy valóban, az argumentum 12 értékéig a program megjegyezte a kiszámított függvényértékeket. Ha még egyszer megkérdezzük a 12. tagot, sokkal gyorsabban kapunk választ:

```
Timing[fibgyors[12]]
```

```
{0. Second, 233}
```

A fenti megoldást Gray igen találóan *dinamikus programozás*nak nevezi.

4.2.3. Gyorsítás a `Compile` függvénnyel

Tetszőleges argumentumra definiáljuk a `gg` függvény-t az alábbi módon:

```
gg[x_] := x Sin[x]
```

vagyis argumentuma lehet valós vagy komplex szám, lista, kifejezés stb. Ha nem kell az összes lehetőségre fölkészülni, hanem feltehető, hogy az argumentum csak gépi pontosságú szám (vagy logikai érték) lehet, akkor a kiértékelés jelentősen meggyorsítható. A gyorsítás különösen akkor jelentős, ha a függvény értékét számoló képletben sok egyszerű (például aritmetikai) művelet van:

```
g = Function[x, x Sin[x]]
```

```
Function[x, x Sin[x]]
```

```
gC = Compile [x, x Sin[x]]
CompiledFunction[{x}, x Sin[x], -CompiledCode-]
```

Készítsünk most egy-egy táblázatot mindkét függvényből:

```
Timing[Table[g[x], {x, 0., 1., 0.01}];]
{1.044 Second, Null}

Timing[Table[gC[x], {x, 0., 1., 0.01}];]
{0.714 Second, Null}
```

Nézzünk meg egy másik példát is:

```
lPC = Compile[x, Evaluate[LegendreP[10, x]]]
CompiledFunction[{x}, (-63 + 3465 x^2 - 30030 x^4 +
   90090 x^6 - 109395x^8 + 46189x^10 )/256, -CompiledCode-]

Timing[Table[LegendreP[10, x], {x, 0., 1., 0.01}];]
{4.285 Second, Null}

Timing[Table[lPC[10, x], {x, 0., 1., 0.01}];]
{0.934 Second, Null}
```

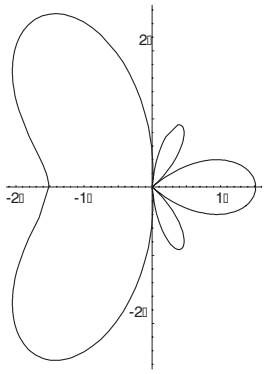
Ezzel szemben, ha olyan függvénnyel van dolgunk, amilyen például a `BesselK` vagy az `Eigenvalues`, akkor nem tapasztalunk jelentős gyorsítást, mivel a számolási idő nagy része a *Mathematica* belső algoritmusainak végrehajtásával telik, amire a *fordítás* (fordítsuk így a *compilation* szót) nincs hatással.

Most arra mutatunk egy példát, hogy hogyan gyorsítható meg az *ábrázolás* a `Compile` utasítás felhasználásával. Az alábbi függvényt szeretnénk ábrázolni:

```
matyo[t_] = Abs[1 +
   E^(-I (7 Pi/12 + Pi Cos[t])) +
   E^(-I (7 Pi/6 + 2 Pi Cos[t]))];
Timing[PolarPlot[matyo[t], {t, -Pi, Pi},
   Plotrange -> All]]
{583.743 Second, Graphics}
```

Ez igen sokáig tartott. A definiált függvény túl sokat tud: argumentuma lehet valós szám, egész szám, szimbolikus kifejezés stb. Nekünk viszont a rajzoláshoz elegendő egy olyan függvény, amely csak valósakra van értelmezve (A `Compile` függvény rendelkezik a `HoldAll` attribútummal, ezért kell alkalmaznunk az `Evaluate` függvényt.):

```
matyoC = Compile[{{t, _Real}}, Evaluate[matyo[t]]];
Timing[PolarPlot[matyoC[phi], {phi, -Pi, Pi},
   PlotRange -> All]]
{4.449 Second, Graphics}
```



Most rajzoljunk Mandelbrot-halmazt:

```
mandelbrotC = Compile[{x, y, lim}, Module[{z, ct = 0},
   z = x + I y; While[Abs[z] < 2.0 && ct <=lim,
   z = z^2 + (x + I y); ++ct]; ct]];
DensityPlot[-mandelbrotC[x,y,50], {x,-2.,1.}, {y,-1.5,1.5},
   PlotPoints -> 50, Mesh -> False]
```

Végül megemlítjük, hogy számos beépített függvény rendelkezik a `Compiled` opcióval, amelynek ilyenkor az alapértelmezés szerinti értéke `True`. Egyébként amikor csak lehetséges, beépített függvényt érdemes használnunk, mert a legtöbb esetben így kaphatjuk meg a leggyorsabb megoldást.

4.2.4. Gyorsítás listaműveletekkel

Az alábbiakban egy lista minden eleméhez többféle módon hozzá fogjuk adni az 1 számot. Kiderül majd, hogy a művelet időigénye jelentősen függ attól, hogy mennyire használjuk ki a *Mathematica* lehetőségeit:

```
lista = Table[Random[Integer, {1, 10}], {1000}];
Timing[ujlista1 = {};
   For[i = 1, i < Length[lista], i++,
   AppendTo[ujlista1, lista[[i]] + 1]]]
```

```
{70.304 Second, Null}
```

Második megoldásként a `Table` utasítással hozunk létre egy új listát:

```
Timing[ujlista2 = Table[lista[[i]] + 1,
   {i, 1, Length[lista]}];]
{6.92 Second, Null}
```

Még gyorsabban jutunk célhoz, ha nem hivatkozunk az egyes elemekre, hanem alkalmazzuk a `Map` függvényt:

```
Timing[f[x_] = x + 1; ujlista3 = Map[f, lista];]
{4.394 Second, Null}
```

Mivel a `Plus` függvény listára alkalmazható (rendelkezik a `Listable` attribútummal), ezért a legegyszerűbb (és leggyorsabb) megoldást így kapjuk:

```
Timing[ujlista4 = lista + 1;]
{2.252 Second, Null}
```

Természetesen mindig ugyanazt az eredményt kaptuk:

```
ujlista1 == ujlista2 == ujlista3 == ujlista4
True
```

Végül még egy (kezdetben ijesztőnek ható) példát említünk. A *Mathematica* bizonyos beépített függvényei között is vannak „egyenlőbbek” (Az alábbi számolásokat nem ugyanolyan gépen végeztük, mint a föntebbieket, ezért csak az egymás közötti arányokra érdemes figyelni, illetve azokra az időadatokra, amelyeket az Olvasó saját gépén kapott.):

```
Timing[Table[i^2, {i, 1000}];]
{0.15 Second, Null}

Timing[Range[1000]^2;]
{0.0833333 Second, Null}

Timing[#^2& /@ Range[1000];]
{0.06666667 Second, Null}
```

Ezek szerint tehát az előző példában a `lista` előállításának célszerű módja a következő:

```
lista = Random[Integer, {1, 10}]& /@ Range[1000]
```

Vizsgáljuk meg, hogy a mért időadatok alátámasztják-e a fenti feltételezést!

4.3. Tanácsok programok készítéséhez

Bár a *Mathematica* magja és programcsomagjai összesen több, mint kétezer belső és külső függvényt tartalmaznak, a felhasználó gyakran találkozik olyan feladattal, amelyet nem tud egyetlen függvény segítségével megoldani. A továbbiakban az ilyen esetekhez szeretnénk tanácsokat adni.

Egy *procedurális* nyelven megírt program minden nehézség nélkül átírható a *Mathematica* programnyelvére. J. W. Gray [29] könyvében a 8.4. szakaszban mutat példákat C és Pascal nyelvű programok közvetlen átírására. A szerző ugyanitt azt is megmutatja, hogy a *Mathematica* funkcionális stíluselemeit felhasználva hogyan lehet a feladatot jóval rövidebben megoldani.

Itt ismét kiemeljük azt, hogy a feladatokat általában különböző stíluselemeket használva is megoldhatjuk. A kiértékeléshez szükséges időtartamok azonban lényegesen különbözők lehetnek; tapasztalatunk az, hogy általában a funkcionális stílusú írásmódot támogatja a legjobban a program.

Amikor gépidőigényes feladatunk van, akkor érdemes szem előtt tartani a gyorsításáról fentebb mondottakat.

Adunk négy különböző megoldást Novak [60] alapján a faktoriális kiszámítására:

```
proceduralis[x_] := Module[{gyujto, index},
   If[Not[IntegerQ[x] && Positive[x]],
      Return["Hibas bemenet"]];
   gyujto = 1; index = 1;
   While[index <= x, gyujto = gyujto*index;
      index = index + 1;]; Return[gyujto]]
proba = {1, 2, 3, 4, -5, 9.2};
proceduralis /@ proba
```

```
{1, 2, 6, 24, Hibas bemenet, Hibas bemenet}
```

```
funkcionalis[x_] := If[x <= 1, Return[1],
   Return[x funkcionalis[x - 1]]]
funkcionalis /@ proba
```

```
{1, 2, 6, 24, 1, 621344.}
```

```
szabalyos[1] := 1;
szabalyos[x_Integer?Positive] := x szabalyos[x - 1]
szabalyos /@ proba
```

```
{1, 2, 6, 24, szabalyos[-5], szabalyos[9.2]}
```

```
fofunkci[x_Integer?Positive] := Times @@ Range[x]
fofunkci /@ proba
{1, 2, 6, 24, fofunkci[-5], fofunkci[9.2]}
```

Hasonlítsuk össze az időeredményeket!

```
Timing[proceduralis /@ Range[100];]
{17.852 Second, Null}

Timing[funkcionalis /@ Range[100];]
{17.026 Second, Null}

Timing[szabalyos /@ Range[100];]
{15.434 Second, Null}

Timing[fofunkci /@ Range[100];]
{1.428 Second, Null}
```

Ez elég meggyőző lehet! Fontoljuk meg azonban azt is, hogy milyen választ várunk, ha a bemenet nem pozitív egész szám. Ügyeljünk arra is, hogy a csábítóan tömör és elegáns funkcionális stílusú programokat is el kell látnunk ellenőrzésekkel és védelemmel.

4.4. Saját programcsomagok készítése

Ha hosszabb ideig és gyakran használjuk a *Mathematicá*t, akkor felmerülhet bennünk az igény saját program*csomag*ok készítésére. Ennek egyik célja az lehet, hogy a saját magunk által definiált gyakran használt függvényeket ugyanúgy szeretnénk használni, mint a *Mathematica* külső függvényeit, másik célja pedig az lehet, hogy alkotásainkat közkinccsé szeretnénk tenni például a MathSource-on, vagy valamelyik folyóiraton, például a *Mathematica Journal*on keresztül. Ebben az esetben a fent leírtakból levonható tanulságokon kívül a leghasznosabb, ha valamelyik programcsomagot tanulmányozzuk. Külön felhívjuk a figyelmet az

Examples	ProgrammingExamples

könyvtárakban található csomagokra. A másik jól használható mankó pedig az említett újság szerzői utasításai, ahol további formai és tartalmi tanácsok találhatók.

5. Matematikán kívüli alkalmazások

Már az eddigiekből is kitűnhetett, hogy a *Mathematica* program a matematikán kívül is számos területen alkalmazható. Ebben a fejezetben azt kívánjuk áttekinteni és néhány példával illusztrálni, hogy melyek azok a belső és külső függvények, amelyek ilyen célokra közvetlenül alkalmazhatók. A legtipikusabb és leggyakoribb (nem túl mélyen szántó) felhasználási lehetőség a fizika, földrajz és kémia területén úgy adódik, hogy kézikönyvek és táblázatok helyett a megfelelő programcsomagokból olvassuk ki a szükséges adatokat. Igazi, komplex alkalmazásokat itt nem adunk, ezekről egy gondolatébresztő bevezető könyv: [18].

Megemlítjük, hogy a *Mathematica* (kézenfekvő) természet- és műszaki tudományos alkalmazásain kívül *közgazdasági* alkalmazásairól [84] vagy döntéstámogató rendszerekkel kapcsolatos felhasználásairól [43] szóló könyv is jelent már meg.

5.1. Üzleti grafika

A táblázatkezelőkhöz vagy a hagyományos programozási nyelvek újabb, mindennel felszerelt változatához szokott felhasználó elvárja, hogy rendelkezésére álljanak azok a grafikai eszközök, amelyekkel az üzleti és a politikai életben megszokott ábrák elkészíthetők. A *Mathematica* készítői a

```
Graphics`Graphics`
```

programcsomag jelentős részét ennek a feladatnak a megoldására szentelték. Az alábbi függvényekre gondolunk:

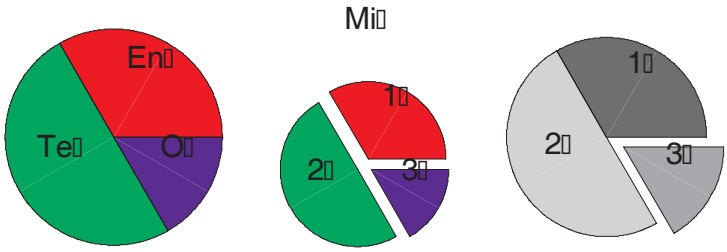
```
BarChart                    PieChart
GeneralizedBarChart         StackedBarChart
PercentileBarChart
```

A `Graphics`Graphics3D`` programcsomag tartalmazza a következő, hasonló céllal készített függvényeket:

`BarChart3D` `StackGraphics`

A felsorolt függvények számos opcióval rendelkeznek. A lehetőségek illusztrálására lássunk néhány példát:

```
<<Graphics`Graphics`
DisplayTogetherArray[
   PieChart[{0.2, 0.3, 0.1}, PieLabels -> {"En", "Te", "O"}],
   PieChart[{0.2, 0.3, 0.1}, PieExploded -> All,
      PlotLabel -> "Mi"],
   PieChart[{0.2, 0.3, 0.1}, PieExploded -> {{3, 0.2}},
      PieStyle ->
     {GrayLevel[0.3], GrayLevel[0.8], GrayLevel[0.6]}]]
```

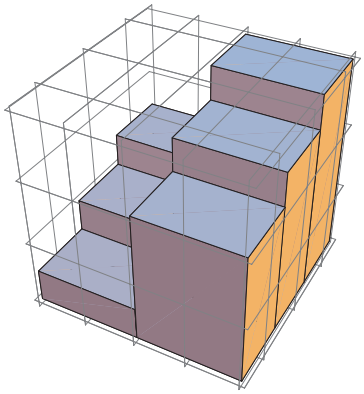


Tortadiagramokat tehát akár a *Mathematicá*val is elkészíthetünk. Kétdimenziós oszlopdiagramot szolgáltatnak az alábbi utasítások:

```
BarChart[{5, 3, 2, -2, 2, 6}, BarStyle ->
    (Which[#>4, RGBColor[0, 1, 0], #<0, RGBColor[1, 0, 0],
           True, RGBColor[1, 1, 0]]&), GridLines -> Automatic]

PercentileBarChart[{1, 3, -4, 5, 3.5, 3}, {-3, 2, 5, 3},
    BarStyle-> {RGBColor[0, 1, 0], RBGColor[1, 1, 0]}
    BarOrientation -> Horizontal, Axes -> False, Frame -> True]
```

Háromdimenziós oszlopdiagramot így készíthetünk:

```
<<Graphics`Graphics3D`
BarChart3D[{{1, 2, 3}, {4, 5, 6}}, Boxed -> False,
    Axes -> False, FaceGrids -> All, PlotRange -> All]
```



A következő utasítás eredménye pedig azt mutatja, hogy függvények egy diszkrét paramétertől függő sorozata hogyan ábrázolható célszerűen:

```
hozam = Table[
   Plot[1000/kamat (1 + kamat/100)^ido, {ido, 1, 10},
   DisplayFunction -> Identity], {kamat, 10, 35, 5}]
Show[StackGraphics[hozam]]
```

5.2. Hang

A matematikai programcsomagok között sem gyakori, hogy egy programban hanggal kapcsolatos függvények is vannak (procedurális nyelvekben — Pascal, BASIC — inkább). A magban a következő függvényeket találjuk:

`Play`	`SampledSoundList`
`SampledSoundFunction`	`Sound`

A `Miscellaneous`Audio`` programcsomag függvényei:

`AmplitudeModulation`	`ReadSoundfile`
`FrequencyModulation`	`Waveform,`
`ListWaveform`	

továbbá a `Miscellaneous`Music`` csomag függvényei:

```
CentsToHertz                Scale
HertzToCents
```

a hang finomabb módosítását és zenei hatások elérését segítik. (A hangsebességgel — `SpeedOfSound` — mint fizikai állandóval másutt — a

```
Miscellaneous`PhysicalConstants`
```

programcsomagban — találkozunk.)

Ahhoz, hogy a különböző hangokat lejátszhassuk, kell, hogy a számítógépben alkalmas *hangkártya* legyen.

Első példaként gépeljük be a következőket:

```
Play[Sin[300 t + 2 Sin[400 t]], {t, 0, 2}]
```

Kettős eredményt kapunk: megkapjuk a lejátszandó függvény képét, valamint azt a hangjelenséget, amelynek amplitúdóját a `Play` függvény első argumentuma írja le. Következzék egy félelmetesebb, összetett hanghatást mutató példa:

```
Play[Sin[(Sin[t^4] Sin[Sqrt[t + 10] t]) t] *
   Sin[Sqrt[t + 6]^4] Sin[2000 t], {t, 0, 20}]
```

Ha újra meg szeretnénk hallgatni, akkor kattintsunk kétszer a képet tartalmazó cellának arra a részére, ahol a kicsi hangszóró látható.

Kétszólamú zene előállításához kételemű listát kell megadnunk a `Play` függvény argumentumául.

Adott alakú hanghullámokat így állíthatunk elő:

```
<<Miscellaneous`Audio`
sqr440 = Waveform[Square, 440, 0.2, Overtones -> 8]
{-Sound-}
```

(A négyzetalakon kívül még néhány további is választható: `Triangle, Sinusoid, Sawtooth`.) Amit kaptunk, az egy hangobjektum (hasonló, mint az ábrázolásnál szokásosan előálló `Graphics` objektum), megszólaltatni is hasonlóan lehet, mint a grafikus objektumokat megtekinteni:

```
Show[sqr440]
{-Sound-}
```

Saját magunk is alkothatunk egy „hangszert”:

```
hangszer = {{1, 1}, {1.1, 0.1}, {1.2, 0.9},
   {1.3, 0.2}, {1.4, 0.8}, {1.5, 0.3}, {1.6, 0.7},
   {1.7, 0.4}, {1.8, 0.6}, {1.9, 0.5}}
```

Hozzunk létre egy hat hangból álló kromatikus hangsort a fenti alakú hangokból a 440-es rezgéssszámú hangból kiindulva:

```
sequence = Table[ListWaveform[hangszer, 440 2^(x/12), 0.2]]
   // Show
{-Sound-}
```

Amplitudó- és frekvenciamodulációt is alkalmazhatunk:

```
{s1, s2, s3} = {Waveform[Sawtooth, 880, 2],
AmplitudeModulation[440, 5300, 1, 5],
FrequencyModulation[660, {{300, 400}, {600, 200}}, 3,
   Type -> Parallel]} // Show
Graphics
```

Következzék egy dúr skála:

```
<<Miscellaneous`Music`
Scale[JustMajor, 440, 3] // Show
-Sound-
```

Választhatunk pitagoraszi, temperált stb. skálát is.

Most egy tiszta kvintet fogunk hallani:

```
Play[Sin[2 Pi Aflat4 t] + Sin[2 Pi Eflat5 t], {t, 0, 0.2}]
-Sound-
```

Természetesen a *Mathematica* lehetőségeinek fölhasználásával — nem kevés munkával — zenét is szerezhetünk, akár megadott stílusban.

5.3. Idő

Az alábbi belső függvények a dátum és az időpont megadásához és különböző átszámításokhoz használhatók:

```
AbsoluteTime            TimeZone
Date                    ToDate
FromDate
```

A `Miscellaneous`Calendar`` programcsomag függvényei a naptárakkal kapcsolatos számolásokhoz használhatók:

```
CalendarChange          EasterSunday
DayOfWeek               EasterSundayGreekOrthodox
DaysBetween             JewishNewYear
DaysPlus
```

A (mindenféle) számítások elvégzéséhez szükséges időtartammal kapcsolatosak a

```
Pause                   $TimeUnit
SessionTime             TimeUsed
TimeConstrained         Timing
```

belső függvények és a 2.4.2. pontban már tárgyalt `Utilities`ShowTime`` programcsomag.

A mai dátumot — ahogyan azt a számítógép operációs rendszere megadja — így kaphatjuk meg:

```
Date[ ]

{1995, 12, 10, 19, 32, 33}
```

Az időzónát a `TimeZone` függvény szolgáltatja. A Japánban használt idő a +9-es zónának felel meg, ezért az ottani idő:

```
Date[9]

{1995, 12, 11, 1, 34, 42}
```

Szeretnénk megtudni, hogy hány másodperc telt el 1900. január elseje óta, a *Mathematica* mostani indítása óta, és hogy a mostani alkalommal mennyi időt dolgozott a központi egység (CPU):

```
{AbsoluteTime[ ], SessionTime[ ], TimeUsed[ ]}

{3124160301, 456.78, 23.5}
```

Kérdezzük meg ugyanezt 10 másodperc várakozás után, vagyis `Pause[10]` kiadásával, és figyeljük meg, hogy a központi egység a szünet közben nem dolgozott.

Az abszolút időből szokásos alakú dátumot a `ToDate` függvény ad, az ellenkező irányú átalakítás a `FromDate` függvény feladata.

A `Miscellaneous`Units`` programcsomagban az idő legkülönbözőbb mértékegységei találhatók meg. Néhány példa:

```
Fortnight                SiderealSecond
Millenium                TropicalYear
```

S ha nem lenne elég, akkor újat is definiálhatunk:

```
millecentennium := 1.1 Millenium
```

5.4. Fizika (mértékegységek)

A fizikában előforduló mértékegységek közül nehéz olyat megnevezni, amelyik ne lenne használható, ezért megelégszünk csupán a mértékegységek *csoport*jainak felsorolásával:

- az erő egységei
- a távolság reciprokának egységei
- térfogategységek
- viszkozitásegységek
- fényenergia- és intenzitásegységek
- a sugárzás egységei
- szögek
- a teljesítmény egységei
- területegységek
- anyagmennyiség
- gravitációs gyorsulás
- mágneses egységek
- egységszorzók
- nyomásegységek
- energiaegységek
- frekvenciaegység
- sebességegység

Mindezek — az átalakításaikhoz használható

```
Convert                  MKS
ConvertTemperature       SI
CGS
```

függvényekkel együtt — a

```
Miscellaneous`Units`
```

programcsomagban találhatók.

Lássunk példát arra, hogyan lehet az egyes mértékegységeket a különböző rendszerekben megkapni:

```
<<Miscellaneous`Units`
{CGS[#], MKS[#], SI[#]}&[Mina]
{429.234 Gram, 0.429234 Kilogram, 0.429234 Kilogram}

{CGS[#], MKS[#], SI[#]}&/@{Pound, Yard, Inch}
{{453.592 Gram, 0.45359237 Kilogram, 0.45359237 Kilogram},
   {91.44 Centimeter, 0.9144 Meter, 0.9144 Meter},
   {2.54 Centimeter, 0.0254 Meter, 0.0254 Meter}

Convert[Mina, Pound]
0.9463 Pound
```

Külön függvény kell a hőmérséklet egységeinek átszámításához, mert ott a kapcsolatok általában nem *homogén* lineárisak:

```
ConvertTemperature[100, Fahrenheit, #]& /@
   {Celsius, Rankine, Reaumur}
{37.7778, 559.67, ConvertTemperature[310.928, Kelvin, Reaumur]}
```

A Réaumur-féle skálát tehát nem ismeri a program.

Tartalma miatt nem meglepő, hogy néhány kémiaiként besorolt függvényt, amelyek a `Miscellaneous`ChemicalElements`` csomagba tartoznak, is itt aduk meg:

`Density`	`MeltingPoint`
`HeatOfFusion`	`StableIsotopes`
`HeatOfVaporisation`	`ThermalConductivity`

Tekintsük a következő példákat:

```
HeatOfFusion[Nitrogen]
0.72 Joule Kilo
     Mole
Density[Nitrogen]
Density::temp:
     Returned density is for Nitrogen at 21 Kelvin.

1026. Kilogram
          3
    Meter
```

A `Miscellaneous`PhysicalConstants`` programcsomag sokféle fizikai (és néhány később megemlítendő kémiai) állandót tartalmaz:

```
AccelerationDueToGravity     IcePoint
AgeOfUniverse                MagneticFluxQuantum
AvogadroConstant             MolarGasConstant
BohrRadius                   MolarVolume
BoltzmannConstant            PlanckConstant
ClassicalElectronRadius      PlanckConstantReduced
EarthMass                    PlanckMass
EarthRadius                  ProtonMass
ElectronCharge               RydbergConstant
ElectronComptonWavelength    SolarConstant
ElectronGFactor              SolarRadius
ElectronMagneticMoment       SpeedOfLight
ElectronMass                 SpeedOfSound
FaradayConstant              StefanConstant
FineStructureConstant        ThomsonCrossSection
GravitationalConstant        WeakMixingAngle
HubbleConstant
```

Fejezzük ki a Föld tömegét, a Planck-féle tömeget és a proton tömegét az elektron tömegével:

```
<<Miscellaneous`PhysicalConstants`
{EarthMass, PlanckMass, ProtonMass}/ElectronMass
```

$$\{6.56026\ 10^{54},\ 2.38952\ 10^{22},\ 1836.15\}$$

Az utolsó szám feltehetőleg mindenkinek ismerős.

5.5. Földrajz (térképek)

A geográfia változni nem szűnő tudásanyagának egy részét is megtaláljuk a *Mathematicá*ban — az éppen aktuális állapotnak megfelelően. A városokra vonatkozó adatokat a `Miscellaneous`CityData`` programcsomag alábbi fügvényeivel állíthajuk elő:

```
CityData                 CityPopulation
CityDistance             $CityFields
```

Különböző módokon megadott térképeket találunk a

```
Miscellaneous`WorldPlot`
```

programcsomagban, amelynek függvényeit az alábbiakban adjuk meg:

```
ToMinutes                    WorldGraphics
WorldData                    WorldPlot
```

Az országok között megtalálhatjuk (ilyen csoportosításban) Észak-Amerika, Európa, Dél-Amerika, Óceánia (ebbe beleértik Ausztráliát), Ázsia, Közel-Kelet és Afrika országait. Elsősorban ide tartoznak a `Miscellaneous`PhysicalConstants`` csomag bizonyos állandói:

```
AccelerationDueToGravity     EarthMass
AgeOfUniverse
```

Végül megemlítjük még a `Miscellaneous`Geodesy`` programcsomagot, mint rokon vonatkozásút.

Határozzuk meg Budapest északi szélességét és keleti hosszúságát:

```
<<Miscellaneous`CityData`
CityData["Budapest"]
{{CityPosition, {{47, 30}, {19, 5}}}}
```

Mennyi Budapest és Boston távolsága?

```
CityDistance["Budapest", "Boston"]
6707.98
```

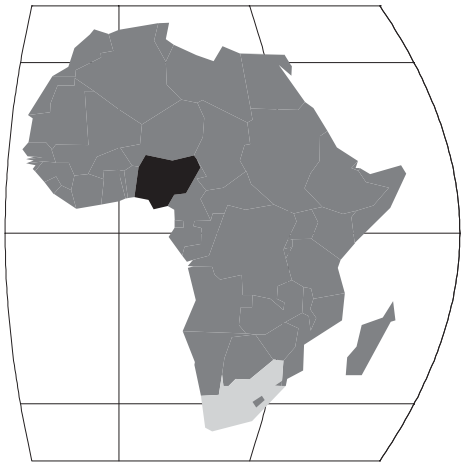
A mértékegységről nem esik szó. Az alábbiak szerint valószínű, hogy mérföld:

```
CityDistance["Budapest", #]& /@
   {"Vienna", "Prague", "Warsaw"}
{217.773, 445.438, 545.69}
```

Ábrázolás céljából kiválaszthatunk földrészeket és országokat is. A sokféle lehetőség illusztrálására itt csak az alábbi példákat közöljük:

```
<<Miscellaneous`WorldPlot`
arnyek[orszag_] := Switch[orszag, "Nigeria", GrayLevel[0.],
         "South Africa", GrayLevel[0.9], _, GrayLevel[0.4]]
```

```
WorldPlot[{Africa, arnyek}, WorldProjection -> Sinusoidal]
```



Különböző stílusú világtérképeket így készíthetünk:

```
WorldPlot[{World, RandomGrays}]

WorldPlot[{World, RandomGrays}, WorldRotation -> {90, 0, 0},
    WorldRange -> {{0, 90}, {-180, 180}},
    WorldProjection -> LamberAzimuthal]
```

5.6. Kémia (elemek)

Jól tanulmányozhatók a kémiai elemek a

```
Miscellaneous`ChemicalElements`
```

programcsomag függvényeivel:

```
Abbreviation                BoilingPoint
AtomicNumber                Density
AtomicWeight                ElectronConfiguration
```

```
ElectronConfigurationFormat   MeltingPoint
Elements                      ThermalConductivity
HeatOfFusion                  StableIsotopes
HeatOfVaporisation
```

Szeretnénk megtudni néhány fontos adatot a hidrogénről, a káliumról és a nátriumról:

```
<<Miscellaneous`ChemicalElements`
{AtomicWeight[#], HeatOfFusion[#], Density[#],
   ThermalConductivity[#], ElectronConfigurationFormat[#]}& /@
   {Hydrogen, Potassium, Sodium}
```

```
Density::temp:
      Returned density is for Hydrogen at 11 Kelvin.
ThermalConductivity::form:
      Returned thermal conductivity is for the gaseous
            form of Hydrogen.
```

$$\{\{1.00794,\ \frac{0.12\ \text{Joule Kilo}}{\text{Mole}},\ \frac{76.\ \text{Kilogram}}{\text{Meter}^3},\ \frac{0.1815\ \text{Watt}}{\text{Kelvin Meter}},\ 1s^1\},$$

$$\{\{39.0983,\ \frac{2.4\ \text{Joule Kilo}}{\text{Mole}},\ \frac{862.\ \text{Kilogram}}{\text{Meter}^3},\ \frac{102.4\ \text{Watt}}{\text{Kelvin Meter}},\ 1s^2\ 2s^2\ 2p^6\ 3s^2\ 3p^6\ 4s^1\},$$

$$\{\{22.98977,\ \frac{2.64\ \text{Joule Kilo}}{\text{Mole}},\ \frac{971.\ \text{Kilogram}}{\text{Meter}^3},\ \frac{141.\ \text{Watt}}{\text{Kelvin Meter}},\ 1s^2\ 2s^2\ 2p^6\ 3s^1\}\}$$

Irodalomjegyzék

[1] Abell, M. L. and Braselton, J. P., *Differential Equations with Mathematica*, Academic Press, New York, 1993.

[2] Adamchik, V., *Finite and Infinite Series*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0204–006).

[3] Adamchik, V. et al., *Guide to Standard Mathematica Packages (Version 2.2.)*, Technical Report, Wolfram Research, Inc., 1993.

[4] Andrásfai B., *Ismerkedés a gráfelmélettel*, Tankönyvkiadó, Budapest, 1971.

[5] Andrásfai B., *Gráfelmélet, folyamok, mátrixok*, Akadémiai Kiadó, Budapest, 1983.

[6] Arnold, V. I., *Közönséges differenciálegyenletek*, Műszaki Könyvkiadó, Budapest, 1987.

[7] Becker, T. and Weispfenning, V. (In Cooperation with H. Kredel), *Gröbner Bases (A Computational Approach to Commutative Algebra)*, Springer-Verlag, 1993.

[8] Bird, R. and Wadler, P., *Introduction to Functional Programming*, Prentice Hall, New York, 1988.

[9] Blachman, N., *Mathematica: Quick Reference, Version 2*, Variable Symbols, Inc., Oakland, CA, 1992.

[10] Bocharov, A. V., *Solving Equations Symbolically with Mathematica*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0204–017).

[11] Bocharov, A.V., *Solving Nonlinear Differential Equations with DSolve*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–960).

[12] Braden, B., Krug, D.K., McCartney, P.W. and Wilkinson S., *Discovering Calculus with Mathematica*, John Wiley & Sons, New York etc. 1992.

[13] Bressoud, D., *Factorization and Primality Testing*, Springer-Verlag, 1989.

[14] Buchberger, B., *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*, Ph.D. Thesis. Math. Inst. Univ. of Innsbruck, Austria, 1965.

[15] Castillo, E., *Solving a Functional Equation*, The *Mathematica* Journal, Volume 5, Issue 1 (1995) 82–86.

[16] Coombs, K. R., *Differential equations with Mathematica*, John Wiley and Sons, New York etc., 1995.

[17] Coxeter, H. S. M., *A geometriák alapjai*, Műszaki Könyvkiadó, Budapest, 1973.

[18] Crandall, R. E., *Mathematica for the Sciences*, Addison-Wesley, New York, 1991.

[19] Császár, A., Jicsinszky, L. and Turányi, T., *Generation of Model Reactions Leading to Limit Cicle Behavior*, React. Kinet. Catal. Let. **18(1–2)**, (1981), 65–71.

[20] Császár Á., *Valós analízis I.-II.*, Tankönyvkiadó, Budapest, 1983-84.

[21] Davies, B., Porta, H. and Uhl, J., *Calculus&Mathematica*, Addison Wesley, New York, 1994.

[22] Deák I., *Véletlenszám-generátorok és alkalmazásuk*, Akadémiai Kiadó, Budapest, 1986.

[23] Elekes Gy., *Kombinatorika feladatok*, (Egyetemi jegyzet), ELTE Budapest, 1992.

[24] Ellis, W. Jr. and Lodi, E., *A Tutorial Introduction to Mathematica*, Brooks/Cole, 1991.

[25] Feagin, J. M., *Quantum Methods with Mathematica*, Springer-Verlag, 1994.

[26] Gaylord, R. J., Kamin, S. N., and Wellin, P. R., *Introduction to Programming with Mathematica*, Springer-Verlag, 1993.

[27] Gaylord, R. J. and Wellin, P. R., *Computer Simulations with Mathematica*, Springer-Verlag, 1994.

[28] Geddes, K.O., Czapor, S.R. and Labah, G., *Algorithms for Computer Algebra*, Kluwer Academic Publishers, Boston, 1992.

[29] Gray, J.W., *Mastering Mathematica, (Programming Methods and Applications)*, Academic Press, New York Professional, Boston, 1994.

[30] Gray, T.W. and Glynn, J., *Exploring Mathematics with Mathematica*, Addison-Wesley, New York, 1991.

[31] Hajós Gy., *Bevezetés a geometriába*, Tankönyvkiadó, Budapest, 1964.

[32] Halmos, P. R. – Sigler, L.E., *Elemi halmazelmélet – Halmazelméleti feladatok*, Műszaki Könyvkiadó, Budapest, 1981.

[33] Harper, D., Waoff, Ch., Hodgkinson, D., *A Guide to Computer Algebra Systems*, J. Wiley & Sons, Chichester, New York, Brisbane, Toronto, 1991.

[34] Heck, A., *Introduction to Maple*, Springer-Verlag, 1993.

[35] Helzer, G., *Gröbner Bases*, The *Mathematica* Journal, Volume 5, Issue 1 (1995) 67–73.

[36] Hironaka, H., *Resolution of Singularities of an Algebraic Variety over a Field of Characteristic Zero*, Ann. Math. **79** (1964) 109–326.

[37] Jones, J.P., Sato, D., Wada, H. and Wiens, D., *Diophantine Representation of the Set of Prime Numbers*, Amer. Math. Monthly 83 (**6**) (1976), 449–464.

[38] Kamke, E., *Differentialgleichungen (Lösungsmethoden und Lösungen)*, Leipzig, 1959.

[39] Katona Gy. Y. és Recski A., *Bevezetés a véges matematikába*, (Egyetemi jegyzet), ELTE Budapest, 1992.

[40] Keiper, J., *Mathematica Numerics: Controlling the Effects of Numerical Errors in Computation*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–937).

[41] Keiper, J., *The N Functions of Mathematica*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–948).

[42] Klincsik M. és Maróti Gy., *Maple 8 tételben*, Novadat, 1995.

[43] Korsan, R. J., *Decision Support Systems in Mathematica*, Springer-Verlag, 1994.

[44] Kósa A., *Ismerkedés a matematikai analízissel*, Műszaki Könyvkiadó, Budapest, 1981.

[45] Kuros, A. G., *Felsőbb algebra*, Tankönyvkiadó, Budapest, 1967.

[46] Lavrov, I. A., Makszimova, L. L., *Halmazelméleti, matematikai logikai és algoritmuselméleti feladatok*, Műszaki Könyvkiadó, Budapest, 1987.

[47] Ledneczkiné Várhelyi Á. és Száva G., *Numerikus analízis példatár személyi számítógépekhez*, (Egyetemi jegyzet), ELTE Budapest, 1990.

[48] Leindler L. és Schipp F., *Analízis I.*, (Egyetemi jegyzet), ELTE Budapest, 1994.

[49] Lenstra, H.W., *Factoring Integers with Elliptic Curves*, Annals of Mathematics, 126 (1987).

[50] Lenstra, H.W.Jr., Lenstra, A.K., Lovász, L., *Factoring polynomials with rational coefficients*, Math. An. **261** (1982), 515–534.

[51] Maeder, R., *Programming in Mathematica*, Second Edition, Addison-Wesley, New York, 1991.

[52] Maeder, R., *Programming in Mathematica*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–904).

[53] Maeder, R., *The Mathematica Programmer*, Academic Press, New York Professional, Cambridge, 1994.

[54] Martin, E., *Introductory Statistics*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–915).

[55] Martin, E., *Advanced Statistics*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–916).

[56] Molnárka Gy., Gergó L., Wettl F., Horváth A. és Kallós G., *A MapleV és alkalmazásai*, Springer Hungarica Kiadó Kft., Budapest, 1996.

[57] Móri T. F. és Székely J. G. (szerk.), *Többváltozós statisztikai analízis*, Műszaki Kiadó, Budapest, 1986.

[58] Nakos, G. and Glinos, N., *Computing Gröbner Bases over the Integers*, The *Mathematica* Journal, Volume 4, Issue 3 (1994) 70–75.

[59] Niven, I., Zuckerman, H. S., *Bevezetés a számelméletbe*, Műszaki Könyvkiadó, Budapest, 1978.

[60] Novak, J.M., *Mathematica Programming Style*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–926).

[61] Pál J., Simon P. és Schipp F., *Analízis II.*, (Egyetemi jegyzet), ELTE Budapest, 1994.

[62] Pontrjagin, L. Sz., *Közönséges differenciálegyenletek*, Akadémiai Kiadó, Budapest, 1972.

[63] Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T., *Numerical Recipes in Pascal*, Cambridge University Press, Cambridge, 1992.

[64] Prékopa A., *Valószínűségelmélet műszaki alkalmazásokkal*, Műszaki Könyvkiadó, Budapest, 1974.

[65] Ralston, A., *Bevezetés a numerikus analízisbe*, Műszaki Könyvkiadó, Budapest, 1972.

[66] Rényi A., *A számjegyek eloszlása valós számok Cantor-féle előállításaiban*, Mat. Lapok **7** (1956) 77–100.

[67] Rényi A., *Valószínűségszámítás*, Tankönyvkiadó, Budapest, 1966.

[68] Risch, R., *The Problem of Integration in Finite Terms*, Trans. Amer. Math. Soc. **139** (1965) 167–189.

[69] Roach, K., *Indefinite and Definite Integration*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0203–993).

[70] Ross, C. C., *Introductory Ordinary Differential Equations with Mathematica*, Springer-Verlag, 1994.

[71] Rózsa P., *Lineáris algebra és alkalmazásai*, Műszaki Könyvkiadó, Budapest, 1974.

[72] Rudin, W., *A matematikai analízis alapjai*, Műszaki Könyvkiadó, Budapest, 1978.

[73] Simon P., *Ismerkedés a numerikus analízissel*, ELTE TTK Továbbképzési Csoportjának kiadványa, Budapest, 1990.

[74] Simonovits M., *Számítástechnika*, Tankönyvkiadó, Budapest, 1983.

[75] Skeel, R. D. and Keiper, J. B., *Elementary Numerical Computing with Mathematica*, McGraw-Hill, 1993.

[76] Skiena, S., *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley, New York, 1991.

[77] Stroyan, K. D., *Calculus Using Mathematica (Preliminary Edition)*, Academic Press, New York, 1992.

[78] Todd, J., *Basic Numerical Mathematics, Vol. 1, Vol. 2*, Academic Press, New York, 1980.

[79] Török T., *Számítógép a matematikaórán*, TYPOTEX, Budapest, 1994.

[80] Trott, M., *The Mathematica Guidebook*, Springer-Verlag, 1994.

[81] Turán P., *„Faktoriálisos" számrendszerbeli „számjegyek" eloszlásáról*, Mat. Lapok **7** (1956) 71–76.

[82] Tusnády G. és Ziermann M. (szerk.):, *Idősorok analízise*, Műszaki Kiadó, Budapest, 1986.

[83] Varga T., *Matematikai logika 1.-2.*, Tankönyvkiadó, Budapest, 1966.

[84] Varian, H. (et. al), *Economic and Financial Modeling with Mathematica*, Springer-Verlag, 1993.

[85] Vincze I., *Matematikai statisztika ipari alkalmazásokkal*, Műszaki Könyvkiadó, Budapest, 1968.

[86] Vvedensky, D., *Partial Differential Equations with Mathematica*, Addison-Wesley, New York, 1993.

[87] Wagon, S., *Mathematica in Action*, W. H. Freeman, New York, 1991.

[88] Withoff, D., *Statistics Examples*, Reprint from The *Mathematica* Conference, Boston, 1992 (MathSource 0201–508).

[89] Wolfram, S., *Mathematica: A System for Doing Mathematics by Computer*, Second edition, Addison-Wesley, New York, 1993.

[90] Wu, Wen–Tsun, *Basic Principles of Mechanical Theorem Proving in Elementary Geometries*, J. Automated Reasoning **2** (1986) 221–252.

Tárgymutató

; (CompoundExpression) 22
% (Out[]) 23
%n (Out[n]) 23
? (Names) 24
?? (Information) 25
(*...*) (kommentár) 22
&& (And) 70
| (Alternatives) 70
|| (Or) 70
! (Not) 70
+ (Plus) 87
- (Subtract) 87
* (Times) 22, 87
/ (Divide) 87
^ (Power) 87
. (Dot) 279
’ (Derivative) 194
! (Factorial) 23, 252
!! (Factorial2) 23, 252
^^ 91
= (Set) 22, 36
:= (SetDelayed) 22, 37
== (Equal) 22
=== (SameQ) 22
!= (Unequal) 22
=!= (UnsameQ) 22
< (Less) 84
<= (LessEqual) 84
> (Greater) 84
>= (GreaterEqual) 84
{...} (List) 22
[...] 22
[[...]] (Part) 22, 31
& (Function) 107
# (Slot) 107
(SlotSequence) 108
_ (Blank) 37
__ (BlankSequence) 39
___ (BlankNullSequence) 39
? (PatternTest) 109
-> (Rule) 35
:> (RuleDelayed) 36
@ (függvényalkalmazás) 110
@@ (Apply) 72, 77, 289
/@ (Map) 34, 77
//@ (MapAll) 34, 77
/. (ReplaceAll) 35, 155
//. (ReplaceRepeated) 36, 155
/; (Condition) 39, 109

Abs 89, 100
AccountingForm 49, 92
Accuracy 94
alapszavak 23
AlgebraicRules 144
Alternatives 70
And 70
Apart 146
Apply 72, 77, 289
approximáció
    Padé 215
    polinom 210
    racionális 215
    spline 216
Arg 89, 101
ArithmeticGeometricMean 186
Array 34, 274
atom 30
Attributes 44
Automatic 43

BaseForm 90
beépített függvény 23
belső függvény 23
BesselZeros 183
Binomial 251

Cancel 146
CartesianMap 131
Cases 361

`Catalan` 89
Cauchy-féle főérték 207
`CauchyPrincipalValue` 207
`Ceiling` 98
`CForm` 50
`Circle` 264
`Clear` 45
`ClearAll` 45
`ClearAttributes` 45
`Close` 51
`Coefficient` 137
`CoefficientList` 137
`Collect` 137
`ColumnForm` 272
`Complement` 77
`Complex` 88
`ComplexExpand` 148
`ComplexMap` 131
`ComposeList` 113
`Composition` 112
computer algebra systems 14
`Condition` 109
`Conjugate` 89
`ConstrainedMax` 302
`ConstrainedMin` 302
`ContourPlot` 123
`CopyDirectory` 58
`Count` 38, 361
`CountRoots` 166
`Cuboid` 264
`CylindricalPlot3D` 130

`D` 192
`Decompose` 113, 140
`Denominator` 146
`DensityPlot` 124
`Derivative` 194
Descartes-szorzat 82
`Det` 282
`DiagonalMatrix` 275
`Dimensions` 272
`DiracDelta` 77
Dirichlet-mag 157
`$DisplayFunction` 122
`DisplayFunction` 122
`Do` 361
`Dot` 279
`Drop` 77
`DSolve` 229

`E` 22, 89
`Eigenvalues` 297
`Eigenvectors` 298
`Eliminate` 145
eljárás 23
`EllipticIntegrate` 203
`EngineeringForm` 92
`Equal` (`==`) 22, 70
`EulerGamma` 89
Euler–Maclaurin-féle módszer 189
`EulerSum` 189
`Evaluate` 48
`EvenQ` 90
`Expand` 136
`ExpandAll` 146
`ExpandDenominator` 146
`ExpandNumerator` 146
`ExpPlot` 32
értékadás
  azonnali 36
  globális 36
  késleltetett 36
  lokális 35

`Factor` 138, 308
`Factorial` (`!`) 23, 251
`Factorial2` (`!!`) 23, 251
`FactorInteger` 313
`FactorList` 140, 308
`FactorSquareFree` 138, 308
`FactorSquareFreeList` 140, 308
`FactorTerms` 137, 308
`FactorTermsList` 140, 308
felhasználói felület 20
`FindMinimum` 200
`FindRoot` 179
`First` 33
`Fit` 220
`FixedPoint` 113
`FixedPointList` 113
`Flat` 44
`Flatten` 34
`FlattenAt` 34
`Floor` 98
`Fold` 310
`For` 362
`FortranForm` 50
`Fourier` 220
Fourier-transzformált 219

`FreeQ` 77
`FullForm` 30
`FullOptions` 43
`Function` 106
függvények
    beépített 23
    belső 23
    külső 23
    matematikai 100
    Mathematica 30
    megadása 103
    primitív 201
    speciális 181
    szélsőértékei 197
    tiszta 106
függvényjelölési módok 110

`GaussianIntegers` 90
`GCD` 306
`Get` 51
gépi pontosságú számok 93
`GoldenRatio` 89
grafikus
    direktívák 264
    elemek 264
    objektumok 264
`Graphics` 264
`GraphicsArray` 132
`GroebnerBasis` 143, 171
Gröbner-bázisok 141, 170

határérték
    sorozaté 185
    függvényé 190
hatványhalmaz 83
`Head` 30
Hesse-féle mátrix 199
`Hold` 48
`HoldAll` 48
`HoldFirst` 48
`HoldForm` 49
`HoldRest` 48

`I` 89
`Identity` 107
`IdentityMatrix` 275
`If` 359
igazságtáblázat 71
`Im` 89, 149
`ImplicitPlot` 128
`Implies` 70
`Indeterminate` 185
információszerzés 24
`Infinity` 89
`Inner` 34
`InString` 51
`Integer` 88
`IntegerDigits` 91
`IntegerQ` 90
`Integrate` 201, 204
`InterpolatingPolynomial` 213
`Interpolation` 216
`Interval` 85
`IntervalIntersection` 86
`IntervalMemberQ` 86
`IntervalUnion` 86
`Inverse` 285
`InverseFourier` 220
`InverseFunction` 114

Jacobi-mátrix 196
jegyzetfüzet (notebook) 20
`Join` 33
`JordanDecomposition` 299

karakterkészlet 22
kernel (mag) 20
kiértékelés
    elvei 95
    szabályos 47
    különös 48
kifejezés
    matematikai 135
    Mathematica 29
közelítő aritmetika 92
közelítő numerikus érték 88
külső függvény 23
kvadratikus reciprocitás 317

`$Language` 49
`LatticeReduce` 307
`LCM` 306
`LeafCount` 59, 135
legkisebb négyzetek módszere 220
`Length` 272
`Less` (`<`) 84
`LessEqual` (`<=`) 84
`Level` 32

`Limit` 185, 190
lineáris regresszió 351
`LinearProgramming` 303
`LinearSolve` 293
`List` 33
lista 33, 78
    megadása 34, 78, 270
    -műveletek 34, 369
`Listable` 44
`ListContourPlot` 130
`ListDensityPlot` 130
`ListIntegrate` 207
`ListPlot` 130
`ListPlot3D` 130
`LogicalExpand` 70
logikai azonosságok 74
`LUDecomposition` 299

`$MachineID` 58
`$MachineName` 58
`MachineNumberQ` 93
`$MachinePrecision` 93
`$MachineType` 58
mag (kernel) 20
`MantissaExponent` 92
`Map` 77
`MapAll` 77
`MapAt` 77
`MapIndexed` 77
`MapThread` 77
`MathLink` 62
`MatrixExp` 288
`MatrixForm` 272
`MatrixPower` 281
`MatrixQ` 271
mátrix
    determinánsa 282
    felbontása 299
    inverze 285
    megadása 270
    műveletek 278
    rangja 283
    speciális 275
`$MaxMachineNumber` 96
`MaxMemoryUsed` 59
`$MaxNumber` 94
`$MaxPrecision` 94
`MemberQ` 78
memóriakezelés 59
`MemoryConstrained` 59
`MemoryInUse` 59
Mihajlov-kritérium 241
`$MinMachineNumber` 96
`$MinNumber` 94
`Minors` 283
mintázat 37
`Mod` 306
`Module` 333
`Modulus` 139, 162
`Multinomial` 252

`N` 96
nagypontosságú számok 93
`NDSolve` 231
`Needs` 26
`Negative` 90
`Nest` 362
`NestList` 362
`NIntegrate` 205
`NLimit` 187, 192
`NonlinearFit` 221
`NonNegative` 90
`Not` 70
`NRoots` 177
`NSolve` 177
`NSum` 189
`Numerator` 146

`OddQ` 90
opciók 41
`$OperatingSystem` 58
`Options` 41
`Or` 70
`Orderless` 44
`Outer` 34, 82
`$Output` 51
`OutputForm` 49

`Pade` 215
`ParametricPlot` 125
`ParametricPlot3D` 125
`Part` 31
`Partition` 77, 82
`Permutations` 248
`Pi` 89
`Plot` 118
`Plot3D` 123
`Plus` 87

`Point` 264
`PolarMap` 131
`PolarPlot` 129
polinom
    interpolációs 213
    kifejtése 136
    szorzattá alakítása 138
polinomegyenletek 164
`Polygon` 264
`PolynomialFit` 221
`PolynomialGCD` 308
`PolynomialLCM` 308
`PolynomialMod` 308
`PolynomialQ` 308
`PolynomialQuotient` 308
`PolynomialRemainder` 308
pontos aritmetika 92
pontos numerikus érték 88
`Positive` 90
`Power` 87
`PowerExpand` 154
`PowerMod` 307
`Precision` 94
`Prime` 311
`PrimePi` 311
`PrimeQ` 90, 311
`Print` 49
`PrintForm` 49
`Product` 155
programcsomag
    matematikai 13
    Mathematica 20, 25
    numerikus 17
    szimbolikus 14
programozási stílusok 357
`Protect` 44
`Protected` 45
`PseudoInverse` 301
`Put` 52
`PutAppend` 52

`QRDecomposition` 299
`Quit` 21
`Quotient` 306

`Random` 332
`Range` 34
`Rational` 88
`Rationalize` 311
`Re` 89, 149
`Read` 54
`ReadList` 54
`ReadProtected` 45
`Real` 88
`RecordLists` 53
`Rectangle` 264
`Reduce` 158, 162
`ReIm` 149
relációk 84
Remez-algoritmus 214
`Remove` 28
rendszerváltozó 24
`ReplaceAll` 35, 155
`Resultant` 141
`Reverse` 33
Risch-algoritmus 202
`Roots` 158
`Round` 98
`RSolve` 185, 237, 256
`Rule` 35
Runge–Kutta-módszer 234

`SameQ` (`===`) 22
`Save` 62
`SchurDecomposition` 299
`ScientificForm` 92
`SeedRandom` 332
`Select` 360
`Series` 211
`Set` 36
`SetAccuracy` 95
`SetAttributes` 45
`SetDelayed` 37
`Short` 49
`Show` 122
`Simplify` 135
`SingularValues` 299
`Slot` (`#`) 107
`Solve` 158
`SolveAlways` 158, 174
sor
    konvergenciája 187
    Laurent 211
    Taylor 210
    trigonometrikus 217
sorozat
    megadása 184
    határértéke 185

`Sound` 375
`Splice` 57
`Sqrt` 100
`Subtract` 87
`Sum` 155, 187
`Switch` 360
`Symbol` 30
`$System` 58

szabályos
    sokszögek 267
    testek 267
számok alakjai 304
számrendszerek 305
számtani-mértani közép 186
számtípusok 88
szórásanalízis 351

`Table` 273
`Take` 33
`Thread` 34
`Through` 111
`Times` 87
`Timing` 59
`ToCharacterCode` 50
`ToExpression` 50
`Together` 146
`Trace` 47
`Transpose` 277
transzformációs szabály 35
`TreeForm` 32
`TridiagonalSolve` 293
`Trig` 150
`Trigonometry` 151

`Union` 78
`UnitStep` 77

`VectorQ` 271
`VectorAnalysis` 221
`VectorCalculus` 196
vektor
    ábrázolása 266
    megadása 271
    műveletek 278
`$Version` 58
`$VersionNumber` 58

Wallis-formula 157
`Which` 360
`While` 362
`With` 364
`Write` 51
`WriteString` 51
`Xor` 70