Számítógépes alapismeretek
– gyakorlat jegyzet –

Stéger József és Fekete Attila

2013.03.07

Tartalomjegyzék

Bevezetés

Ez a jegyzet az ELTE TTK első éves fizika BSc. hallgatók *Számítógépes alapismeretek* gyakorlatához készült. Mivel a számítógépes alapismeretek önmagában is rendkívül tág fogalom, a rendelkezésre álló gyakorlati órákban reménytelen vállalkozás lenne minden területet érinteni, az operációs rendszerektől a programnyelvekig, az irodai programoktól a matematikai algebrarendszerekig. Nem is lenne sok értelme, mivel minden egyes téma egy önálló gyakorlat témája lehetne. Másrészt az „alapismeretek" fogalma is meglehetősen viszonylagos. Hiszen az azonos érdeklődési körű fizika hallgatók között is rendkívül eltérőek azok az ismeretek amikkel az egyes hallgatók a gyakorlat kezdetekor rendelkeznek. Van aki „csupán" az Internetet böngészte eddig, más pedig már önálló programokat is írt.

Először is határozzuk meg tehát, hogy mi is a fizika hallgatók számára „testreszabott" gyakorlati számítógépes alapismeret. Tanulmányai során minden hallgató számtalanszor találkozik majd „tudományos beszámolók" készítésével: a mérési jegyzőkönyvektől a TDK dolgozatig, szakdolgozattól egészen a referált folyóiratcikkekig. Ezért azt a célt tűztük ki, hogy egy „tudományos beszámoló projekten" keresztül mutatjuk be a számítógépes alapismereteket. A beszámoló elkészítéséhez a következő eszközöket fogjuk használni:

- *vim* szövegszerkesztő,
- LaTeX, egyenletek készítéséhez alkalmas betűszedő (typesetting) rendszer,
- *gnuplot* ábrakészítő program,
- *gawk* adatfeldolgozó program, és
- *bash* shell adatfájlok kezelésére.

Hogyan használjuk a jegyzetet?

A számítógép-használat elsajátítása és a nyelvtanulás számos tekintetben nagyon hasonló. Kezdő szinten mindkét esetben első ránézésre nagyon nehéznek és érthetetlennek

tűnhetnek a dolgok. Később aztán fokozatosan érthetővé válnak a szavak (parancsok) és a nyelvtan (szintaktika). Ahogy az idegen nyelveknél is a leggyakoribb szavakat és nyelvtani szerkezeteket tanuljuk először, úgy a számítógépes ismereteket is célszerű az egyszerűbb, gyakran használt, elemekkel kezdeni és fokozatosan mélyíteni az ismereteket, a korábban tanultak folyamatos ismétlése mellett.

A gyakorlati órákon több témakörből is lesznek újdonságok, általában egyre komplexebb és komplexebb formában. Ehhez nyújt a jegyzet támogatást. A jegyzet egymás utáni fejezetei szorosan egymásra épülnek, és az egyes fejezetek feldolgozásához szükség van a korábbi fejezetek anyagának ismeretére. Az egy-egy témakört átölelő illetve bővítő fejezetek gyakorlati elsajátítását példák segítik a fejezetek végén, melyeknek kidolgozott megoldása a jegyzet végén található meg. A példák után feladatok következnek. A feladatokat úgy állítottuk össze, hogy legtöbbjét a példák megfelelő átalakításával lehet megoldani. A feladatsorok végén néhány nehezebb probléma is található, melyek megoldása mélyebb ismeretet feltételez, természetes intuíciót és külső információ forrásokban keresgélést.

Van, akinek könnyebben megy a nyelvtanulás, van akinek nehezebben, de a legfontosabb talán, hogy *senkinek sem megy gyakorlás nélkül.* Ugyanez a helyzet a számítógépes rendszerek használatával is. Csupán a rendelkezésre álló kötelező órai gyakorlatok alatt nem lehetséges megfelelő gyakorlatot szerezni. A jegyzetben található feladatok otthoni megoldása nélkülözhetetlen.

Hogyan tudunk gyakorolni?

Az órai gyakorlatok során Linux operációs rendszert használunk a számítógép laboratóriumban. Nyilván sokakban felmerül a kérdés, hogy miért éppen Linuxot használunk, amikor a számítógépek többségén a Windows operációs rendszer valamilyen változata van? Nos, ez való igaz, de az is tény, hogy az egyetemeken és a kutatóhelyeken a számítógépes modellezéshez használt számítógépek döntő többségén valamilyen Linux, a Unix operációs rendszer ingyenes, nyílt forráskódú, szabad „klónja" fut.

Semmiképpen nem kívánjuk eldönteni, hogy melyik operációs rendszer a jobb, mivel mindegyiknek megvannak az előnyei és a hátrányai. A Unixot évtizedekkel ezelőtt a nagy számítógépes központok (mainframe-ek) szervereire fejlesztették ki, ezért a legfontosabb szempontok a stabilitás és a biztonság volt a többfelhasználós működés mellett. A Windows ellenben a személyi számítógépekkel együtt terjedt el, ahol tipikusan egy felhasználót kellett kiszolgálni, ezért a hangsúlyt a kényelemre helyezték.

Habár a korai Linux változatok valóban nem voltak kényelmesek, az utóbbi időben sokat javult a helyzet. Linuxon is használhatunk már grafikus ablakkezelőt, és már néhány kattintással kényelmesen változtathatunk a beállításokon. Mindamellett azonban megmaradtak azok a Unixból örökölt tulajdonságok is, amelyek rendkívül hatékonnyá teszik a *tudományos* munkát.

Habár a gyakorlatokon Linuxos számítógépeket használunk, az otthoni gyakorlás könnyen megoldható Window-os gépeken is. A következő megoldások valamelyikét javasoljuk.

Gyakorlás Windowson

Egy önálló windows-os számítógépen a következő csomagok telepítésére van szükség:

- A *vim* telepítéséhez menjünk a szövegszerkesztő honlapjára, és keressük ki a „PC: MS-DOS and MS-Windows" szekcióból a **Self-installing executable** részt. Töltsük le az aktuális verziót a böngészővel, és indítsuk el a telepítő fájlt. A telepítés után a program a Start menüből a Start → Programok → Vim 7.2 → gvim menüponttal indítható.

- A LaTeX programcsomag windows-os változatát MiKTeXnéven tölthetjük le a projekt honlapjáról. A rendszerünknek megfelelő verziót válasszuk ki. Az alaprendszer csaknem 100MB, ezért telepítés előtt győződjünk meg arról, hogy rendelkezésre áll elegendő szabad tárterület. A telepítés után az egyes pontok a Start menüből elérhetőek.

- A *gnuplot* grafikonkészítő program forrásállományát a sourceforge-ról tölthetjük le. Sajnos a korábbi megszokásokkal ellentétben a legfrissebb verziójú Windowsos futtatható csomagot itt nem érjük el. A jegyzet írásakor a legfrissebb telepítő csomag érhető el. Csomagoljuk ki a tömörített állományt egy alkalmas könyvtárba (pl. `Desktop`, `C:\Program Files`, stb.). A programot a `gp470-20120916-win32-mingw.exe` fájllal indíthatjuk. Ha jobb egérgombbal kattintunk erre a fájlra, akkor a Küldés... Asztalra (Send to... Desktop) menüponttal egy könnyen elérhető hivatkozást hozhatunk létre az Asztalon.

- A *gawk* telepítőjét is megtaláljuk az Interneten. Ehhez keressük meg a Downloads cím alatt a „Complete package, except sources" listaelem melletti linket. A telepítő futtatása után be kell állítani a futtatható állományokhoz az elérési útvonalat. A Start menüben kattintsunk jobb gombbal a My Computer-re, majd a Tulajdonságok (Properies) menüpontra. A felugró ablakban keressük ki a Haladó (Advanced) fület, és válasszuk ki. Itt kattintsunk a Környezeti változók (Environmental variables) gombra. A felugró ablakban keressük ki a PATH változót a rendszerváltozók között, majd kattintsunk a Módosítás (Edit) gombra. Végül adjuk meg a változófelsorolás végén, pontosvesszővel elválasztva, a *gawk* bináris fájljainak elérési útvonalát (`C:\Program Files\GnuWin32\bin`).

- A *bash* parancssor értelmező is a sorceforge-ról tölthető le Windowsra. A jelenlegi legfrissebb fordítás 2011. március dátummal érhető el.

Gyakorlás Cygwinen

A Cygwin egy Linux-szerű környezet Windowshoz. A gyakorlat során használt alkalmazásokon túl, számos alapvető Linux eszköz megtalálható a Cygwinben. Azonban a Cygwin nem teszi lehetővé, hogy eredeti Linux programokat futtassunk Windowson vagy, hogy az eredeti Windowsos programok felismerjék a Linux funkcióit. A Cygwin környezet letölthető a projekt honlapján található „Install or update now!" linkről.

A telepítőprogram segítségével válasszuk ki a gyakorlathoz szükséges programokat, majd indítsuk el a Cygwint a Start menüből.

Gyakorlás virtuális gépen

Az egyik legrugalmasabb megoldás, ha egy virtuális gépet telepítünk a Windows alá. Ilyen megoldások például a Virtualbox, vagy a VMware. A Virtualbox szabad hozzáférésű, egyszerűen letölthető, míg a VMware letöltéséhez regisztráció szükséges.

A virtuális gépek telepítése után töltsünk le egy Linux disztribúciót (pl. a Mint vagy Ubuntu), majd telepítsük a virtuális gépen. Ezt a megoldást akkor válasszuk, ha a számítógépünk viszonylag sok memóriával rendelkezik (min. 2GB), és van elegendő szabad tárterület (min. 5GB).

Gyakorlás önálló Linuxon

A legradikálisabb megoldás, ha nem csak egy virtuális gépre, hanem magára a számítógépre telepítünk Linuxot. A Linux nagyon kis hardverigényű, ezért régebbi számítógépek is alkalmasak ehhez a megoldáshoz. Ha van elegendő tárterület, akkor lehetőség van az eredeti Windows területét lecsökkenteni, és a Linuxot a Windows mellé telepíteni. Ezután vagy az egyik, vagy a másik rendszert lehet elindítani.

Példák és feladatok

A gyakorló példák és a feladatok legtöbbjéhez mintafájlokat készítettünk és tettünk közzé, melyeket bármely böngészővel letölthetünk a gyakorlat honlapjáról, vagy egy Linux terminálba beírt

```
user@host:~$ wget http://complex.elte.hu/szamitogepesalapismeretek/lesson-1/
    Gy1.1
```

paranccsal, ahol az URL végén a gyakorló feladat azonosítója szerepel (pl. Gy1.1).

A feladatok letöltése után Windows esetén indítsuk el a *vim*et a Start → Programok → Vim7.2 → gVim alól. A grafikus ablak Fájl → Megnyitás menüpontjával olvassuk be a letöltött fájlt.

Linux esetén nyissunk egy terminált a Alkalmazások → Kellékek → Terminál menüpont segítségével és a terminálba írjuk be egyszerűen, hogy

```
user@host:~$ vim Gy1.1
```

ahol `Gy1.1` az imént letöltött gyakorló feladathoz tartozó fájl.

1. fejezet

Szövegszerkesztés

Nyilván sokan használták már a Word, vagy az LibreOffice[1] szövegszerkesztőket. Az irodai munkában ugyan hasznosak ezek a WYSIWYG[2] szövegszerkesztők, ahol a felhasználó gyakorlatilag a végleges szöveget látja a képernyőn, de programok, szkriptek írására nem igazán jók. Az első gyakorlatokon megismerkedünk a *vim* karakteres szövegszerkesztővel, amely a maga nemében egyedülálló.

A *vim* az egyik legjobb és leghatékonyabb szövegszerkesztő. A *vim*ben minden szerkesztési művelet néhány billentyűleütéssel elvégezhető, így nem kell folyamatosan a legördülő menükre kattintgatni az egérrel. A hatékonyságnak és funkcionalitásnak azonban ára van: kezdetben nagyon nehéz a felhasználók tanulási folyamata. A kezdeti tanulásba befektetett idő és energia azonban később sokszorosan megtérül.

Ez a szövegszerkesztő számos operációs rendszerhez letölthető a forráskódot is nyilvánossá tévő *vim* projekt címről. A *vim* nagyon jól dokumentált. A beépített segítséggel minden parancsról részletes leírást kaphatunk, kereshetünk témák szerint, és rendelkezésre áll egy gyakorlóprogram is `vimtutor` néven. Ezen kívül ingyenesen letölthető egy 572 oldalas kézikönyv hozzá.

1.1. A házitanító `vimtutor`

A *vim*mel legkönnyebben a `vimtutor` program segítségével ismerkedhetünk meg. Mielőtt elkezdenénk a példákkal foglalkozni, mindenképpen végezzük el a `vimtutor` gyakorló feladatait! A feladatok megoldásához 25-30 perc szükséges, attól függően, hogy mennyit kísérletezünk.

Ha Linux grafikus felületet használunk, akkor először nyissunk meg egy parancsértelmező terminál ablakot. Számos ablakkezelőben gyorsbillentyű segíti a terminálnyitást, próbálkozzunk a `<Ctrl-Alt-t>` gombkombinációval. Lehetőségünk van menüből egér

[1] Szabad hozzáférésű irodai software-csomag, elődei: OpenOffice.org, StarOffice

[2] Mozaikszó, az angol *What You See Is What You Get* kifejezésre

segítségével is terminálnyitásra a Alkalmazások → Kellékek → Terminál[3] menüpontból. A terminál megnyitása után rögtön egy parancssort kapunk:

```
user@host:~$ ■
```

A parancssor elején látjuk a *prompt*-ot, a végén pedig egy kurzor villog. Írjuk a kurzor helyére a `vimtutor` parancsot[4]:

```
user@host:~$ vimtutor
```

Windowsban a Start → Programok → Vim 7.2 → Vim tutor menüpontból érhető el a gyakorlóprogram.

1.2. *vim* üzemmódok

A kezdő *vim* felhasználóknak az okozza a leggyakoribb problémát, hogy a *vim* többféle üzemmódban lehet. Mielőtt rátérnénk az alapvető parancsok ismertetésére röviden összefoglaljuk a legfontosabb *vim* üzemmódokat:

Normal mód A *vim* Normal módban indul, és ebbe a módba kerülünk az `<Esc>` billentyű megfelelően sokszori megnyomásával. A normál módban a bevitt karakterek nem a szövegbe kerülnek, hanem valamilyen parancsként funkcionálnak.

Visual mód Olyan, mint a Normal mód, csak a kurzor mozgatásával a szöveg egy részét kijelölhetjük. A nem kurzormozgatására szolgáló parancsokat a szövegszerkesztő a kijelölt szövegrészre alkalmazza.

Háromféle vizuális mód van:

- `v` billentyűvel a karakterenkénti,
- `V` billentyűvel a soronkénti, és a
- `<Ctrl-V>` billentyűvel a blokkonkénti

vizuális mód érhető el.

Select mód Hasonló, a MS Windows kijelölés módjához. Az egérrel kijelölt rész törlődik egy nyomtatható karakter leütésekor, és belép a beszúrás módba.

Insert mód Ebben a módban a leütött nyomtatható karakterek bekerülnek a szerkesztett szövegbe. A beszúrás módba többféleképpen kerülhetünk, pl. a Normál módból az `a` vagy az `i` billentyűk lenyomásával.

[3] Angol nyelvű környezetben Applications → Accessories → Terminal

[4] A továbbiakban a kurzort nem jelöljük.

Command-line mód A parancssor módban a *vim* legalsó sorbába egy teljes sornyi szöveget gépelhetünk. Itt adhatunk meg Ex parancsokat a „:” billentyűvel (például kilépés (`:q`), dokumentum mentés (`:w`, segítség kérés `:h`), minta keresést a `?` és a `/` után, illetve szűrő parancsokat a `!` jel után.

Ex mód Hasonló, mint a Parancssor mód azzal a különbséggel, hogy a parancs megadása után Ex módban maradunk kevés szerkesztési lehetőséggel.

1.3. A legfontosabb *vim* parancsok

A fejezet végén található 1.1–1.7 táblázatokban röviden összefoglaljuk a gyakorlatok megoldásához használandó legfontosabb *vim* parancsokat. A táblázatokban szereplő N egy pozitív egész számot jelöl, ennyiszer hajtódik végre az adott parancs. Számos parancsot könnyen megjegyezhetünk a parancshoz kapcsolódó angol szavakból: pl. append, insert, delete, replace, change, yank, put, Join, word, end, backward, for, till.

A {motion} kurzormozgatási parancsokat jelent (ld. az 1.2. táblázat). A {char} egy tetszőleges karaktert, az {a-z} és {A-Z} pedig egy, a megadott tartományba eső karaktert jelöl. A {visual} azt jelenti, hogy a parancs kiadásakor Visual módban van a szövegszerkesztő.

Az Ex parancsokat mindig kettősponttal (`:`) kell kezdeni. A kettőspont után le lehet rövidíteni a parancsokat. Az Ex parancsoknak azt a részét, amelyet nem kötelező kiírni, szögletes [] zárójellel jelöljük (ld. 1.1. táblázat). Az Ex módban a `<Tab>` billentyűvel kiegészíttethetjük a parancsokat vagy az argumentumokat (pl. fájlneveket). Nagyon fontos a `:h[elp]` parancs, mivel ezzel tudunk bővebb információt kérni az egyes parancsokról.

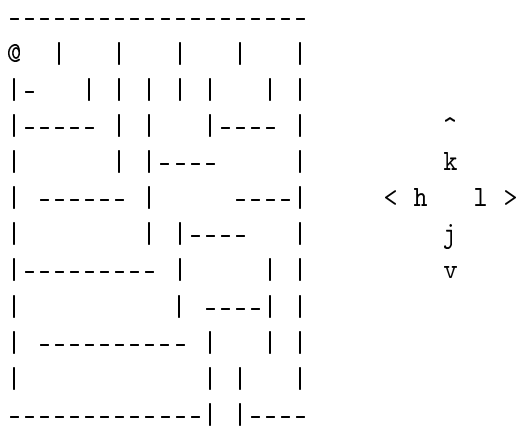
1.4. Példák és feladatok

Gyakorló példák

Gy1.1. Vigyük a kurzort az 1.1. ábrán látható labirintuson a @ jelhez, és haladjunk végig a kurzorral a labirintuson a `hjkl` gombok segítségével! A jobb mutató-, középső-, és gyűrűsujjunk alaphelyzetben legyen sorban a `jkl` billentyűk fölött.[5]

Gy1.2. Hősünk, akit ismét a @ jelez, át szeretne kelni az 1.2. ábrán látható patakon. A vizet `o` jelzi, a köveket üres helyek. Vezessük el a kurzort a `hjkl` valamint a `tTo123456789` (ld. 1.2. tábla) billentyűk segítségével a `@`-tól a túlparton látható `X` jelig úgy, hogy közben nem léptetjük a kurzort a „vízbe”. A szigetek között legfeljebb 9 távolságra ugorhatunk egyszerre a {1-9}`hjkl` billentyűpárokkal. Például `5k` paranccsal öt hellyel ugorhatunk felfelé.

[5] Ez azért lényeges, mert ez az alap ujjkiosztás a gépírásnál.



1.1. ábra. Labirintus a gyakorló feladathoz.

Gy1.3. Írjuk fel a hét napjait egymás alá. Rendezzük őket ABC sorrendbe Normal módban kiadott parancsokkal (ld. 1.4. tábla)!

Gy1.4. Gépeljük be, majd húzzuk alá a

```
Hacking Vim: A Cookbook to get the Most out of the Latest Vim Editor
```

címet „-” jelekkel kizárólag Normal módban kiadott parancsokkal! Az aláhúzáshoz használjuk a vizuális módot.

Gy1.5. Mozgassuk át az alábbi mondókában a hiányzó szavakat a megfelelő helyre a Normal módban kiadott parancsokkal (ld. 1.4. és 1.5. táblákat)!

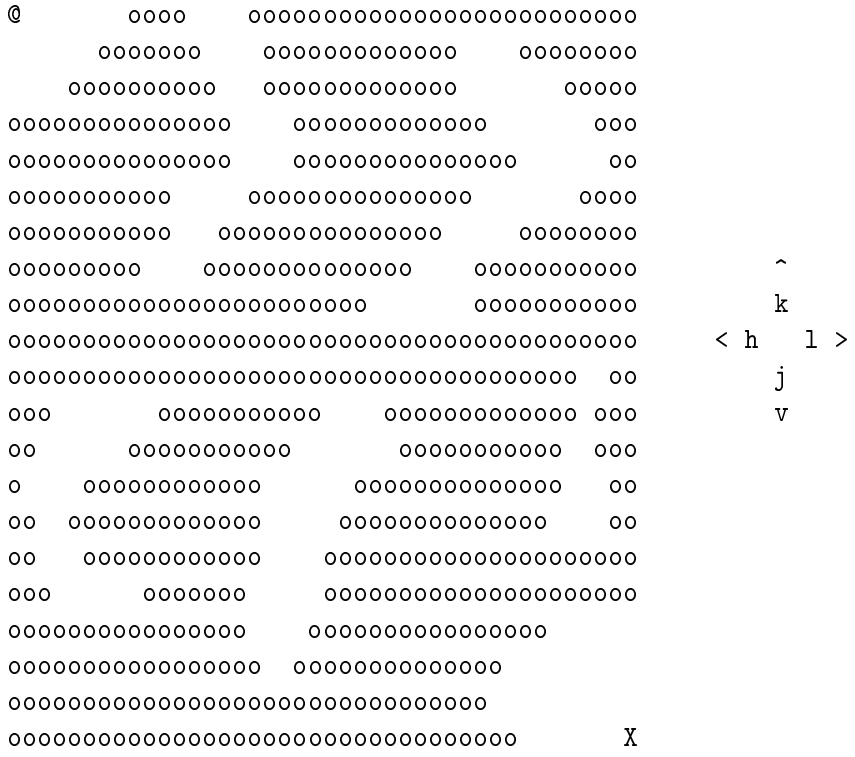
```
boci tarka füle farka lakni tejet

Boci, ____, _____,
Se ____, se _____.
Oda megyünk _____,
Ahol _____ kapni!
```

Gy1.6. A makrók olyan egy „felvett” utasítássorozat, amelyet később néhány billentyű leütésével visszajátszhatunk (ld. 1.6. tábla). Írjunk egy makrót, ami az adott sor első karakterét a sor legvégére viszi!

Feladatok

F1.1. Ismételjük meg a Gy1.1 feladatot az 1.3. ábrán látható labirintussal az alábbi szabályokkal!



1.2. ábra. „Szigetek” a gyakorló feladathoz.

1. Ha kevesebb, mint harmadszor megyünk neki a falnak, akkor visszaléphetünk az útra, és folytathatjuk a játékot.
2. Ha áthaladtunk a falon, vagy harmadszor is nekimentünk a falnak, akkor kezdjük elölről a játékot.

F1.2. Ismételjük meg a Gy1.2 feladatot az 1.4. ábrán látható folyóval!

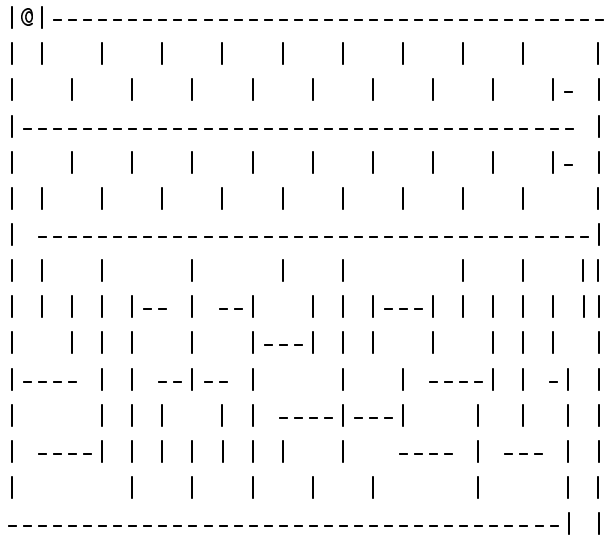
F1.3. Írjuk fel a hónapok nevét sorban egymás alá. Rendezzük őket ABC sorrendbe Normal módban kiadott parancsokkal!

F1.4. Az alábbi szövegből töröljük ki a csupa nagybetűvel beírt mondatot, és cseréljük meg a két paragrafust.

> A Turing-gép úgynevezett absztrakt automata: a valóságos digitális számítógépek nagyon leegyszerűsített modellje (részletesebben ld. következő fejezet). EZ A HOSSZÚ MONDAT NEM TARTOZIK AZ EREDETI SZÖVEGBE, CSAK UTÓLAG ÍRTUK HOZZÁ, HOGY A GYAKORLATON KI LEHESSEN VÁGNI. További jelentőségét az ún. Church-Turing-tézis adja, amely szerint univerzális algoritmikus modell (ld. lentebb). Az ilyen egyszerű számítógépmodellek matematizált elméleteivel a matematika számítógép-tudománynak nevezett eléggé fiatal tudományágának olyan részterületei foglalkoznak, mint például a számításelmélet.
>
> A Turing-gép fogalmát Alan Turing angol matematikus dolgozta ki 1936-ban megjelent cikkében a matematikai számítási eljárások, algoritmusok precíz leírására, tágabb értelemben pedig mindenfajta „gépies” problémamegoldó folyamat, például az akkoriban még nem létező számítógépek működésének modellezésére. Erre az időszakra, a II. világháború környékére tehető az ilyesfajta, a számítási eljárásokat azok különféle modelljein keresztül vizsgáló kutatások fellendülése, melyek végül a valódi számítógépek építésébe torkollottak (Turing maga is részt vett egy valódi gép, a Colossus megépítésében).
>
> — Forrás: wikipedia.hu

F1.5. Írjunk egy makrót, amely egy paragrafus elejére és végére is tesz egy-egy idézőjelet!

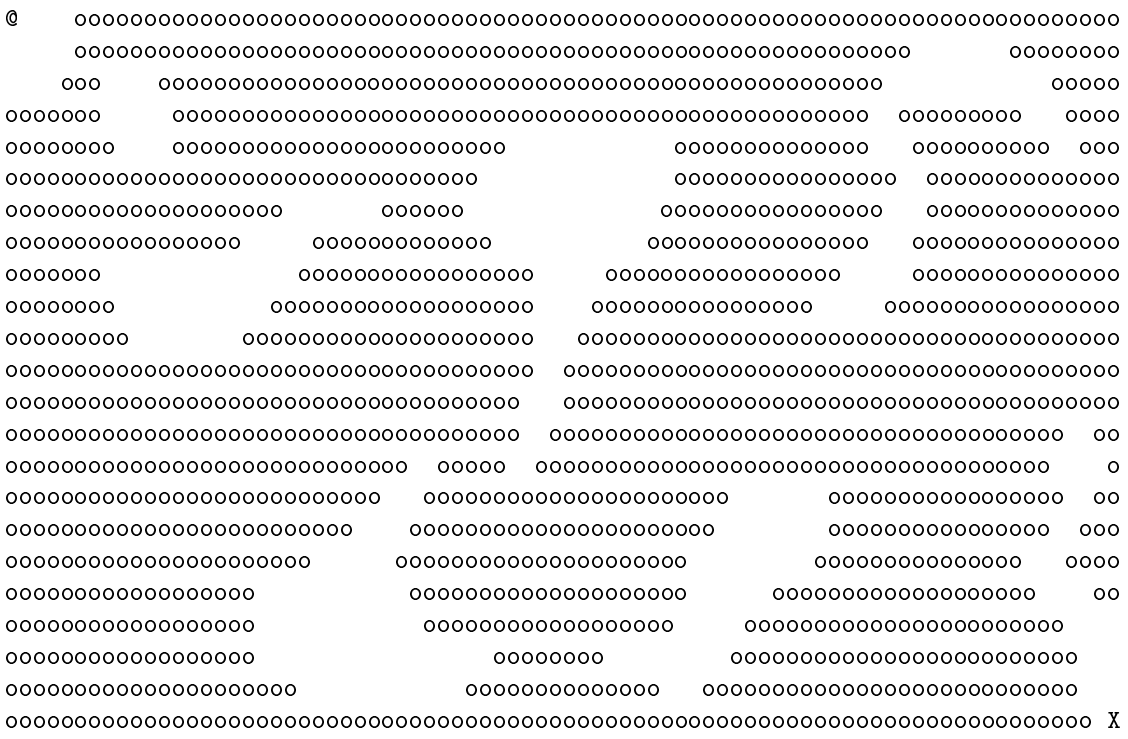
F1.6. Adjuk meg azokat a parancsokat, amikkel a nevünket a lehető legkevesebb billentyűleütéssel leírhatjuk 20 sorban, soronként 10-szer! (Útmutatás a parancsok rögzítéséhez: használjunk egy makró-regisztert (pl. `qa`), majd a regiszter tartalmát írassuk ki a makró felvétele után (pl. a `"ap`).)



1.3. ábra. Labirintus az órai feladathoz.

F1.7. Adjuk meg azokat a parancsokat, amikkel egy 20×20-as `o`-ból álló négyzetet `x`-ekkel bekeretezhetünk minél kevesebb billentyűleütéssel! (Útmutatás: használjuk a Block-visual módot!)

F1.8. Adjuk meg azt a makrót, amely egy sor utolsó szavát megcseréli a következő sor első szavával.



1.4. ábra. „Szigetek” az órai feladathoz.

1.1. táblázat. Ex módbeli parancsok. Az Ex parancsokat Normal módból mindig a kettőspont (`:`) billentyűvel kell kezdeni.

`:h[elp]`	segítség kérése
`:q[uit]`	kilépés a *vim*ből
:[range] d	a [range] tartományba eső sorok törlése
`:r[ead]` [file]	a [file] tartalmának beszúrása a kurzor alá
`:w[rite]`	a puffer mentése

1.2. táblázat. Kurzormozgatási parancsok

N `h`	N-szer balra
N `l`	N-szer jobbra
N `k`	N-szer fel
N `j`	N-szer le
`0`	ugrás a sor elejére
`^`	ugrás a sor első nem üres karakterére
`$`	ugrás a sor végére
N `gg`	ugrás az első (N-ik) sorra
N `G`	ugrás az utolsó (N-ik) sorra
N `w`	ugrás a következő (N-ik) szó elejére
N `W`	ugrás a következő (N-ik) üres hellyel elválasztott szó elejére
N `e`	ugrás a következő (N-ik) szó végére
N `E`	ugrás a következő (N-ik) üres hellyel elválasztott szó végére
N `b`	ugrás az előző (N-ik) szó elejére
N `B`	ugrás az előző (N-ik) üres hellyel elválasztott szó elejére
N `ge`	ugrás az előző (N-ik) szó végére
N `gE`	ugrás az előző (N-ik) üres hellyel elválasztott szó végére
N `f`{char}	ugrás a {char} következő (N-ik) előfordulásá*ra* jobbra
N `F`{char}	ugrás a {char} következő (N-ik) előfordulásá*ra* balra
N `t`{char}	ugrás a {char} következő (N-ik) előfordulásá*ig* jobbra
N `T`{char}	ugrás a {char} következő (N-ik) előfordulásá*ig* balra
N `;`	az előző `f`, `F`, `t`, `T` ismétlése (N-szer)
N `,`	az előző `f`, `F`, `t`, `T` ismétlése az ellenkező irányban (N-szer)
N `)`	ugrás a következő (N-ik) mondat elejére
N `(`	ugrás előző (N-ik) mondat elejére
N `}`	ugrás a következő (N-ik) paragrafus elejére
N `{`	ugrás előző (N-ik) paragrafus elejére

1.3. táblázat. Szöveg bevitele

Normal módból	
N `a`	szöveg beszúrása a kurzor után (N-szer)
N `A`	szöveg beszúrása a sor vége után (N-szer)
N `i`	szöveg beszúrása a kurzor elé (N-szer)
N `I`	szöveg beszúrása a sor első nem üres karaktere elé (N-szer)
N `o`	új sor nyitása az aktuális sor alatt (N-szer)
N `O`	új sor nyitása az aktuális sor felett (N-szer)
Visual block módból	
`A`	ugyanannak a szövegnek a hozzáadása minden kijelölt sor után
`I`	ugyanannak a szövegnek a beszúrása minden kijelölt sor elé

1.4. táblázat. Szöveg törlése

N `x`	(N) karakter törlése a kurzor alatt
N `X`	(N) karakter törlése a kurzor előtt
N `dd`	(N) sor törlése
N `D`	törlés a kurzortól a sor végéig (N-szer)
`d`{motion}	szöveg törlése, amelyen a kurzor áthalad
{visual} `d`	a kijelölt szöveg törlése (Visual módban)
N J	(N) sor összefűzése (sorvégek törlése)
{visual} J	a kijelölt sorok összefűzése (Visual módban)

1.5. táblázat. Szöveg másolása és mozgatása

`"`{char}	a {char} regiszter használata a következő törléshez, másoláshoz vagy beszúráshoz
N `y`{motion}	a kurzor mozgatásával érintett szöveg bemásolása egy regiszterbe
N `yy`	N sor bemásolása egy regiszterbe
N `Y`	N sor bemásolása egy regiszterbe
N `p`	egy regiszter tartalmának bemásolása a kurzor után (N-szer)
N `P`	egy regiszter tartalmának bemásolása a kurzor elé (N-szer)
{visual} `y`	a kijelölt szöveg bemásolása egy regiszterbe

1.6. táblázat. Parancsok ismétlése

N `.`	a legutóbbi parancs ismétlése N-szer
`q`{a-z}	a leütött karakterek mentése az {a-z} regiszterbe (makró felvétel)
`q`{A-Z}	a leütött karakterek hozzáfűzése az {a-z} regiszterhez
`q`	a makró mentés befejezése
N `@`{a-z}	az {a-z} regiszter tartalmának végrehajtása (N-szer) (makró lejátszás)
N `@@`	az előző `@`{a-z} ismétlése (N-szer)
N `u`	az előző (N) parancs visszavonása (undo)
N `<Ctrl-R>`	az előző (N) visszavont parancs ismételt elvégzése (redo)
`U`	a legutóbb módosított sor visszaállítása

1.7. táblázat. Szövegmódosító parancsok

N `r`{char}	N karakter kicserélése {char}-ra
N `R`{char}	belépés Replace módba (a szöveg ismétlése N-szer)
N `c`{motion}	a kurzor által érintett szöveg kicserélése
{visual} `cc`	a kijelölt szöveg kicserélése
N `cc`	N sor kicserélése
`C`	a szöveg kicserélése a kurzortól a sor végéig

2. fejezet

Dokumentumok készítése

A tipográfia, azaz a nyomdai tervezőmunka egy önálló szakma. A nyomdai minőség számos olyan apróságon múlik, mint például a különbség a „fizika” és a „fizika” között[1], a tördelés, a címek betűmérete, a szavak közötti távolság, stb. Ebben a fejezetben a LaTeX „nyomdai rendszer”[2] használatát ismertetjük, amellyel nyomdai minőségű dokumentumokat készíthetünk anélkül, hogy ki kellene tanulnunk a nyomdász szakmát.

A LaTeX filozófiája szerint ketté kell választani a szerzői és a tipográfusi feladatokat. A szerzőnek nem az a feladata, hogy a nyomdászkodjon, hanem hogy a szöveg *struktúráját* definiálja, azaz megmondja mi a cím, mik a fejezetcímek, hol vannak képletek, listák, stb.

Ennek a megközelítésnek az előnyei különösen komplex, matematikai képletekkel, kereszthivatkozásokkal, ábrákkal teli szövegek esetén jelentkeznek. A komplexitást növelheti továbbá az indexálás, bibliográfia, vagy a nagy oldalszám. A LaTeX ezekkel a feladatokkal is kitűnően megbirkózik.

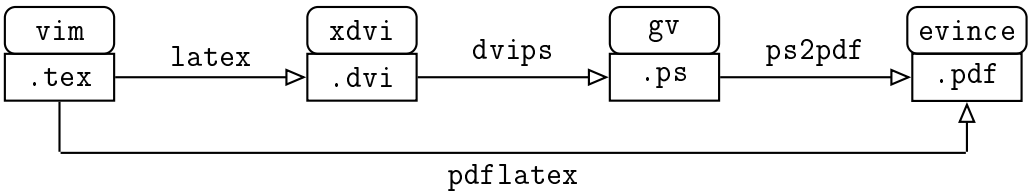
2.1. LaTeX olvasnivalók

A LaTeX-hez rengeteg könyv, útmutató, segédanyag áll rendelkezésre. Többségük angol nyelvű, de rendelkezésre áll két magyar nyelvű összefoglaló is:

- az Egy nem túl rövid bevezető a LaTeX 2_ε használatába – avagy LaTeX 2_ε 78 percben[1, 2], és
- a LaTeX kezdőknek és haladóknak[3] a Panem Kiadó gondozásában.

Ebben a jegyzetben csak a példák megértéséhez és a feladatok megoldásához szükséges alapfogalmakat ismertetjük. A LaTeX részletesebb ismertetéséhez az Internetről szabadon letölthető fenti leírásokat ajánljuk.

[1] Figyeljük meg, hogy az első esetben az f és az i betű nem különül el egymástól.
[2] angolul: *typesetting system*



2.1. ábra. A LaTeX fordítás fázisai. Szögletes dobozban az adott fájl kiterjesztése, kerekített dobozban pedig az adott fájl megjelenítéséhez használandó Linux program. A nyilakon az adott fájl létrehozásához használandó program neve látható.

2.2. A LaTeX használata

A LaTeX bemenete egy egyszerű szöveges fájl, általában `.tex` kiterjesztéssel. Ezt a fájlt a továbbiakban *forrásfájlnak* fogjuk nevezni. A forrásfájl együtt tartalmazza a szöveget, és a szöveg struktúráját leíró parancsokat. Ahhoz, hogy a dokumentumot használni tudjuk, például kinyomtassuk vagy a képernyőn megnézzük, az elkészített forrásfájlt le kell fordítani. A fordításnak több útvonala és lépése lehetséges, a dokumentumba ágyazott tartalomtól és a kívánt végső dokumentum formátumától függően. Egy lehetséges út első lépése a `latex` paranccsal tehető meg:

```
user@host:~$ latex valami.tex
```

A fordítás után egy `.dvi`[3] kiterjesztésű fájl jön létre. Ezt Linuxban az *xdvi* vagy az *evince*, míg Windowsban a *yap* programmal nézhetjük meg. Ha elégedettek vagyunk az eredménnyel, akkor következő lépésben a *dvips* paranccsal létrehozhatunk egy nyomtatásra alkalmas `.ps`[4] kiterjesztésű fájlt. Ezt Linuxon a *gv* vagy az *evince*, Windowson a *GSview* nevű programmal nézhetjük meg, illetve nyomtathatjuk ki. Lehetőség van arra, hogy a `.ps` fájlból `.pdf`[5] fájlt hozzunk létre a *ps2pdf* paranccsal. A forrásfájlhoz tartozó külső hivatkozások megfelelő előkészítése után lehetséges a *pdflatex* paranccsal közvetlenül `.pdf` dokumentumot építeni. A fordítás egyes fázisai a 2.1. ábrán követhetők nyomon.

A LaTeX forrásfájlok szerkesztéséhez egy egyszerű karakteres szövegszerkesztő elegendő. A gyakorlaton az előző fejezetben bemutatott *vim* szövegszerkesztőt fogjuk használni. Alapesetben az angol billentyűzet karakterkészlete használható a forrásfájlban, azaz az ékezet nélküli kis- és nagybetűk, a számok, a szóköz, a tabulátor, és az sorvége karakter, valamint az alábbi jelek:

```
‘ ’ " ; : . , ? ! @ # $ % ^ & * ( ) ~ - _ = + [ ] { } < > \ / |
```

[3] A *device independent* fájlformátum jellemző kiterjesztése
[4] A *PostScript* formátum kiterjesztése
[5] A *portable document format* fájlok kiterjesztése

Fontos megjegyezni, hogy a LaTeX forrásfájlban a szavak közti szóközök számának és a sortörés helyének nincs jelentősége. A paragrafusokat legalább egy üres sor választja el egymástól.

A fenti karakterek közül tíznek speciális jelentése van, ezeket csak külön parancsokkal lehet megjeleníteni:

```
\ { } % $ _ ^ ~ & #
```

2.3. A gyakori speciális jelek jelentése

Az alábbiakban röviden összefoglaljuk a leggyakoribb speciális jelek funkcióját, így a `\ { } % $ _ ^` jelekét.

Parancsok A szöveg struktúráját parancsokkal és környezetekkel (egy speciális parancspárral) lehet megadni. Minden parancs a \ (backslash) jellel kezdődik. A parancsokat három csoportba lehet osztani:

Alfabetikus parancsok egy \-jelből és az angol ABC betűiből állnak, például

`\section{...}`.

Csillagos parancsok az alfabetikus parancsok csillaggal végződő változatai, amelyek az alap alfabetikus parancstól kissé eltérően viselkednek. Például a

`\section*{...}`

parancs a `\section{...}` parancstól eltérően egy sorszám nélküli fejezetet nyit.

Kétjeles parancsok két jelből, a \-jelből és egy nem-alfabetikus jelből állnak, így például a `\'` parancs egy vesszőt tesz az utána következő karakterre.

Blokkok A kapcsos zárójelek ({-jel és }-jel) olyan blokkokat hoznak létre, amelyeket a LaTeX egy egységként tud kezelni. Kapcsos zárójelekkel tudjuk például megadni, hogy a fejezetcím meddig tart a `\chapter{...}` parancs után. Ennek a szakasz címe például így van megadva:

```
\section{A gyakori speciális jelek}
```

Blokkokat hozhatunk létre a `\begin{`*név*`}` `\end{`*név*`}` párokkal is. Ezeket a blokkokat *környezeteknek* hívjuk. Környezetekben adhatjuk meg például a kiemelt matematikai képleteket (`equation`), felsorolásokat (`itemize`), stb.

Megjegyzések A LaTeX mindent figyelmen kívül hagy a %-jeltől (százalékjel) a sor végéig[6], így megjegyzéseket tehetünk a forrásfájlba. Ha egy szó végét rögtön %-jel követi, akkor a szó összeolvad a következő sor első szavával.

Szövegközi matematikai mód A szövegközi képleteket $-jelek közé kell zárni. Az $y = ax + b$ képlet például a forrásfájlban így néz ki: `$y=ax+b$`.

Index jelek matematikai módban Matematikai módban, legyen szó szövegközi kifejezésekről vagy kiemelt formulákról, a képletekben gyakran szerepelnek a felső és az alsó indexben jelek, melyeket a ^-jel illetve a _-jel segítségével varázsolhatunk elő. Ezek használatát az alábbi példával illusztráljuk. A forrásfájlban megadott

```
$f(x_i) = a x_i^2 + b x_i + c$
```

kifejezés képe a dokumentum fordítása után így fest: $f(x_i) = ax_i^2 + bx_i + c$.

Megjegyezzük még, hogy a LaTeX parancsoknak kétféle argumentuma lehetséges, úgymint kötelező vagy opcionális. A kötelező argumentumokat a fent bemutatott kapcsos zárójelek közé, míg az opcionális argumentumokat szögletes zárójelek közé kell tenni, mint `\section[`*rövid cím*`]{`*teljes cím*`}`.

2.4. Az ékezetes karakterek használata

A dokumentum bevezető részében betöltött `inputenc` csomag gondoskodik arról, hogy a forrásfájl karakterkészletét a fordító helyesen értelmezze. Előfordulhat olyan helyzet, amikor a szöveg szerkesztője, a forrásfájl gondozója nem választhatja meg önkényesen a karakterkódolást. Hasonlóan elképzelhető, hogy a forrásfájl bevitelére használt eszköz nem támogatja az ékezetes betűkészletet. Ha korrektül szeretném leírni például az autóm márkáját, és nem lennének ékezetparancsok, bajban lennék, mert *Škoda* a típusa... Ilyenkor tehát az ékezetek elővarázslására speciális parancsot kell használnunk. Legtöbb ékezet parancsa intuitív, a repülő ékezetek mintájára képződnek, csak a karaktert meg kell előzze a parancs. A 2.1. táblázatban összefoglaljuk a leggyakrabban használt ékezetparancsokat, előre sorolva a magyar nyelvben hasznosakat.

Az ékezet parancsok használatával kapcsolatban két dolgot kell megjegyezni. A parancs nem értelmezi a mögötte szereplő karaktert, így egzotikus „betűket” is alkothatunk, mint például a `\'m` hatására ḿ betűt szedünk ki. A másik tudnivaló, hogy a pontozott betűkre feltett ékezetek nem tüntetik el a pontot, így nem meglepő, hogy a `\'j` eredménye j́ lesz. Az i illetve a j betű ponttalan változatai: `\i` illetve `\j` (kiszedve: ı, ȷ), tehát a í helyesen `\'\i` írandó.

[6] Beleértve a sor vége jelet is.

2.1. táblázat. Az ékezetparancsok rövid összefoglalója

Parancs	Példa	Kép
`\'`	`\'o`	ó
`\"`	`\"e`	ë
`\H`	`\H{u}`	ű
`\``	`\`o`	ò
`\^`	`\^e`	ê
`\~`	`\~o`	õ
`\c`	`\c{c}`	ç
`\k`	`\k{a}`	ą
`\=`	`\={o}`	ō
`\b`	`\b{o}`	o̲
`\.`	`\.{o}`	ȯ
`\d`	`\d{u}`	ụ
`\r`	`\r{a}`	å
`\u`	`\u{o}`	ŏ
`\v`	`\v{s}`	š
`\t`	`\t{oo}`	o͡o

2.5. A forrásdokumentum szemantikája

Ahogy a nyelvnek megvannak a saját szabályai, úgy a LaTeX dokumentumok forrásainak is követnie kell egy szabványosított formalizmust. A legegyszerűbb LaTeX forrásfájl a következő elemekből áll:

```
\documentclass{article}

\begin{document}
Hello World!
\end{document}
```

A forrásfájl elején a `\documentclass[`*opciók*`]{`*osztály*`}` parancs deklarálja a dokumentum egészének formátumát. A `\documentclass{...}` és a `\begin{document}` közötti részt bevezető résznek[7] nevezik. Ezt követi a dokumentum törzse[8] a `\begin{document}` és a `\end{document}` közötti részben, ide kell a dokumentum szövegét írni.

A LaTeX legfontosabb beépített dokumentumosztályai az `article`, a `report`, a `book` és a `letter`. Attól függően, hogy milyen dokumentumosztállyal dolgozunk eltérő pa-

[7] angolul: *preamble*
[8] angolul: *body*

rancsok és környezetek állnak alapértelmezéseben rendelkezésünkre. A kiadók általában saját dokumentumosztályt használnak.

A 2.2. ábrán egy összetettebb dokumentum szerkezete látható. A példában az `article` dokumentumosztályt használtunk és az osztály megjelölése mellett néhány opcionális paraméterét is megadtunk. A leggyakoribb opciók a

`10pt, 11pt, 12pt` a normál betűméret kiválasztása,

`onecolumn, twocolumn` egyoszlopos vagy két oszlopos mód,

`a4paper, letterpaper` a lapméret kiválasztása,

`oneside, twoside` egyoldalas vagy kétoldalas kép kiválasztása, és a

`landscape` fektetett lapválasztás.

A `\usepackage[`*opciók*`]{`*csomag*`}` paranccsal kiegészítő csomagokat töltünk be. Segítségükkel bővítjük a rendelkezésre álló parancs- és környezetkészletet, illetve finomíthatjuk a már definiáltak hatását. Rengeteg LaTeX csomag létezik, de talán a legfontosabb csomagokat összefoglalja az alábbi felsorolás.

`inputenc` a forrásfájl karakterkészletének, így például az ékezetes betűk, helyes dekódolásához használható csomag. A magyar ékezetes karakterek értelmezéséhez ma leggyakrabban előforduló rendszerbeállítások mellett tipikusan használt kódolások, melyeket opcióként kell megadni, az `utf8` és a `latin2`.

`fontenc` a karakterek megjelenítését szabályozó csomag, a magyar ékezetes betűk helyes kiszedését a `T1` opció feltüntetése mellett érjük el.

`babel`[4] nyelvi csomag nemzeti fejezetcímekkel, dátumformátummal, ábra- és táblázat aláírás címkékkel, valamint szabályozza az adott nyelvben használt szavak helyes elválasztását.

`graphicx`[5] grafikai csomag, mely lehetővé teszi, hogy a dokumentum forrásfájljától független ábrákat emeljünk a dokumentumba.

A bevezető részben adhatjuk meg a dokumentum egyes formai elemeit beállító parancsokat is. Így például a dokumentum címlapjának definíciói, azaz a cím (`\title{...}`), a szerzőlista (`\author{...}`) valamint a dokumentum készítésének dátuma (`\date{...}`). Ilyen továbbá az oldalak stílusát szabályozó `\pagestyle` parancs is, melynek legfontosabb opciói a következők lehetnek:

`empty` az oldalszámozás nélküli dokumentumlapok,

`plain` hatására az oldalszámozás a lap alján jelenik meg, és a

`headings` amikor az oldalszámozás és a fejezetcím a lap tetejére kerül.

A fentieken kívül célszerű a bevezető részben megadni a parancsdefiníciókat és rövidítéseket. A fenti példában az `\ELTE` parancs az „Eötvös Loránd Tudományegyetem" kifejezést helyettesíti. Fontos megjegyezni, hogy a parancsok érzékenyek a kis- és a nagybetű közötti különbségre, tehát az `\elte` parancs nincs definiálva.

A dokumentum törzsében a `\maketitle` parancs létrehozza a címlapot. Ezután a dokumentumot különböző szintű fejezetcímekkel tagolhatjuk. Az `article` osztályban a következő szintek értelmezettek:

```
\section{...}
\subsection{...}
\subsubsection{...}
\paragraph{...}
\subparagraph{...}
```

A `report` és a `book` osztályokban használhatjuk még a `\chapter{...}` és a `\part{...}` parancsokat is. A LaTeX a fordítás során a fejezetcímeknél automatikusan elkészíti a tartalomjegyzéket, melyet a `\tableofcontents` paranccsal illeszthetünk be a dokumentumba. Figyeljünk arra, hogy a hivatkozások (mint például a tartalomjegyzék) helyes behelyezéséhez a LaTeX-et többször is le kell futtatnunk. Erre a LaTeX figyelmeztet is a fordítás során.

A példában két környezet is szerepel. Az első az `itemize` környezet, amely egy egyszerű felsoroláslistát készít. A lista elemeit az `\item` parancs különbözteti meg. Az `itemize` környezethez hasonlóak az `enumerate` és a `description` környezetek is. Az előbbi automatikusan sorszámozza a lista elemeit, az utóbbi pedig az `\item`[*címke*] paranccsal címkéket tesz a listaelemek elé. A második környezet az `equation` környezet. Ez a szövegből kiemelt képletek megjelenítésére szolgál. A matematikai képletek szerkesztésével a későbbi 6. fejezetben foglalkozunk.

2.5.1. Hivatkozások használata a dokumentumban

Dokumentumaink átláthatóságát segíti a hivatkozások használata. Ez gyakorlatilag azt jelenti, hogy a dokumentum objektumai sorszámot kapnak és ezekre a sorszámokra tudunk a folyó szövegben utalni. A sorszámokat a fordító kezeli, a szöveg szerkesztőjének csak arra kell ügyelni, hogy a hivatkozáshoz megfelelő címkéket társítson. Később az objektumokat – mint fejezetek, ábrák, táblázatok és képletek – bátran átrendezhetjük, a fordítás során a sorszámok automatikusan újragenerálódnak. Két dologra kell odafigyelnünk a dokumentum fordítása során. Mindenek előtt érdemes a terminálon megjelenő üzeneteket nyomon követni, keletkező naplót megvizsgálni[9], hiszen a hiányzó címkékkel

[9] A naplófájl kiterjesztése `.log` és neve a forrásából készül a `.tex` lehasításával

kapcsolatosan figyelmeztető jelzést kapunk. Ha erre nem ügyelünk, a dokumentum folyó szövegében `??` jelet fogunk találni. A másik szempont, hogy a fordító első menetben a folyószövegbe elhelyezendő referencia számok helyeit hagyja ki és ezzel párhuzamosan a címkézett objektumokról katalógust épít. További fordításra van szükség, hogy a tényleges, helyes számok kerüljenek a szövegbe. Az ökölszabály, hogy háromszor fordítsuk a forrást annak változtatása után.

A hivatkozások készítésének három alapparancsa van. A `\label{`*címke*`}` paranccsal címkéket kell hozzárendelni az objektum közelében, így például a fejezet `\section{`*cím*`}` mögött szerepelhet `\label{`*címke*`}`. A címkére a `\ref{`*címke*`}` vagy az őt ölelő oldalra a `\pageref{`*címke*`}` paranccsal lehet hivatkozni. Célszerű a címkének tömörnek és egyértelműnek lennie, ezt segíti, ha konvencionálisan *chap:, sec:, fig:, tab:, eq:*, stb. szócskával kezdődnek címkéink, melyek rendre főfejezetre, fejezetre, ábrára, táblázatra vagy egyenletre vonatkoznak sokat könnyítünk szerkesztői munkánkon.

A magyar nyelvi környezetben további hivatkozást segítő parancsok vannak definiálva. A `\aref{`*címke*`}` és `\Aref{`*címke*`}` a kiszedett szám elé kis illetve nagy betűvel kezdődően határozott névelőt tesz. Segítségével elkerülhető a magyar nyelvű szövegben előforduló banális hiba, mint *„A 54. ábrán jól látszik, hogy. . . ”*[10]

2.6. A *vim* és a LaTeX

A *vim* szövegszerkesztő a LaTeX fájlok szerkesztését számos hasznos funkcióval segíti. Például különböző színnel emeli ki a LaTeX parancsokat, a megjegyzéseket vagy a fejezetcímeket (*syntax highlighting*). Ezenkívül telepíthetünk további kiegészítéseket (*plugin*) is a *vim*hez. A `latex-suite` kiegészítés további hasznos funkciókat nyújt a LaTeX fájlok szerkesztéséhez.

Ebben a szakaszban a *vim* néhány olyan funkcióját ismertetjük, melyek igen hasznosak lehetnek például LaTeX fájlok szerkesztésénél.

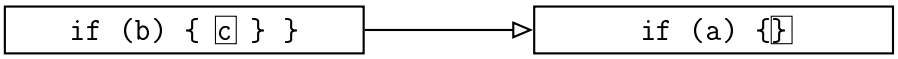
2.6.1. Szöveg objektum kiválasztása

Előzőekben, az 1. fejezetben bemutattuk, hogy ha operátorok után mozgás parancsokat adunk ki, akkor az adott operátor a szövegnek arra a részére vonatkozik, amelyen a kurzor áthalad. Például a `dw` paranccsal a kurzortól a szó végéig, a `d)` paranccsal a kurzortól a mondat végéig, valamint a `d}` paranccsal a kurzortól a paragrafus végéig törölhetjük a szöveget. Gyakran azonban nem a kurzortól szeretnénk törölni, hanem azt az adott szövegobjektumot, amiben a kurzor van. Ilyenkor eddig körülményes módon az adott szövegobjektum elejére kellett vinnünk a kurzort, és onnan elvégezni a parancsot.

A fenti nehézség kiküszöbölésére operátorok után vagy vizuális módban a mozgásparancsok helyett megadhatunk szövegobjektumokat is. A 2.2. táblázatban található

[10] Helyesen kiszedve: *„Az 54. ábrán jól látszik, hogy. . . ”*

parancsok közül az `a`-val kezdődő[11] szövegobjektum parancsok az üres helyekkel *együtt*, az `i`-vel kezdődőek[12] az üres karakterek *nélkül* választják ki a szövegobjektumokat. Tehát az `i`-vel kezdődő parancsok mindig kevesebbet választanak ki, mint az `a`-val kezdődőek. A táblázat második felében a kijelölés művelete blokkokra hat. A különböző típusú zárójelárral határolt blokkok, mint kijelölő műveletére kényelmesen használhatjuk a zárójel párját, így a `d2i{` és a `d2i}` hatása ugyanaz, az alábbi szövegre alkalmazva követhetjük nyomon a hatását:



2.6.2. Az előző művelet ismétlése

Ha az előző parancsot meg akarjuk ismételni, akkor nyomjuk meg a `.` (pont) billentyűt (ld. 1.6. táblázat). Az előző makró ismétléséhez adjuk ki a `@@` parancsot.

2.6.3. Parancshívás *vim*ből

A LaTeX dokumentum fordításához meg kell hívnunk a `latex` parancsot a parancssorból. Ehhez eddig vagy ki kellett lépni a *vim*ből, vagy egy másik terminálablakot kellett nyitni. A *vim*ből azonban lehetőség van közvetlenül is parancsokat kiadni. Ehhez üssük le a `:!` billentyűket, majd írjuk be a kívánt parancsot (pl.: `:!latex valami.tex`). A parancs bevitelénél két segítségünk is van. A `<Tab>` billentyűre a *vim* (és a parancssor is) kiegészíti a részben begépelt parancsot vagy fájlnevet. Másrészt a fel (↑) megnyomására a korábban megadott parancsok között válogathatunk. Ugyanezeket a billentyűket a parancssorban is használhatjuk.

2.6.4. Block-visual mód

Gyakran előfordul, hogy egymás alatti sorokba ugyan azt a szöveget akarjuk egymás alá beírni. Például, ha a forrásfájl egy részét nem akarjuk, hogy nyomtatásba bekerüljön, de nem is akarjuk teljesen kitörölni, akkor az érintett rész elé %-jelet kell tenni, így azt a LaTeX figyelmen kívül hagyja. A legegyszerűbb megoldás, ha `<Ctrl-v>`-vel block-visual módba kapcsolunk, majd kijelöljük az érintett szövegrészt. Végül az `I` paranccsal a kijelölés elejére, `A` paranccsal pedig a kijelölés végére beírhatjuk a kívánt szöveget. Ha végeztünk, nyomjunk `<Esc>`-et, és ekkor a szövegszerkesztő a kijelölt rész minden sorába beírja a szöveget.

[11] Az angol *an* határozatlan névelőből lehet megjegyezni.
[12] Az angol *inner* szó kezdőbetűjéből.

2.2. táblázat. Szöveg objektumok

N aw	kiválaszt N szót *(a word)* a kezdő vagy bezáró üres helyeket is beleértve, de az üres helyeket nem számítva N-be.
N iw	kiválaszt N belső szót *(inner word)*, az üres helyeket is beszámítva.
N aW	kiválaszt N SZÓT a kezdő vagy bezáró üres helyeket is beleértve, de az üres helyeket nem számítva N-be.
N iW	kiválaszt N belső SZÓT, az üres helyeket is beszámítva.
N as	kiválaszt N mondatot *(a sentence)* a kezdő vagy bezáró üres helyeket is beleértve, de az üres helyeket nem számítva N-be.
N is	kiválaszt N belső mondatot *(inner sentence)*, az üres helyeket is beszámítva.
N ap	kiválaszt N bekezdést *(a paragraph)* a kezdő vagy bezáró üres helyeket is beleértve, de az üres helyeket nem számítva N-be.
N ip	kiválaszt N belső bekezdést *(inner paragraph)*, az üres helyeket is beszámítva.
N a[	kiválaszt N „[]” blokkot, a zárójeleket beleértve.
N i[	kiválaszt N „[]” blokkot, a zárójeleket leszámítva.
N a(	kiválaszt N „()” blokkot, a zárójeleket beleértve.
N i(	kiválaszt N „()” blokkot, a zárójeleket leszámítva.
N a{	kiválaszt N „{}” blokkot, a zárójeleket beleértve.
N i{	kiválaszt N „{}” blokkot, a zárójeleket leszámítva.
N a<	kiválaszt N „<>” blokkot, a zárójeleket beleértve.
N i<	kiválaszt N „<>” blokkot, a zárójeleket leszámítva.
N a'	kiválaszt N két '-jel blokkot, az idézőjeleket beleértve.
N i'	kiválaszt N két '-jel blokkot, az idézőjeleket leszámítva.

2.7. Példák és feladatok

Gyakorló példák

Gy2.1. Fordítsuk le a 2.2. ábrán látható fájt LaTeX-hel, és nézzük meg egy `.dvi` nézővel a kapott fájlt. Készítsünk PostScript és `.pdf` fájlt, és azokat is nézzük meg.

Gy2.2. Módosítsuk az iménti feladatban letöltött fájlt. Használjunk más dokumentumosztályt, kétoszlopos szedést, sorszámozott listát, használjunk a `\chapter` és a `\subsection` parancsokat, helyezzük át a tartalomjegyzéket a dokumentum végére.

Gy2.3. Rendezzük megfelelő sorrendbe a 2.3. ábrán látható forrásfájl sorait.

Gy2.4. Írjuk be a megfelelő helyekre a 2.4. ábrán látható forrásfájl sorait.

Gy2.5. A 2.5. mellékletben szereplő `itemize` környezetben minden sor elé szúrjuk be az `\item` parancsot. Cseréljük ki mellékelt szövegben a zárójelben álló részeket IGEN-re (vagy NEM-re). Pl. (Igen / Nem) $\Longrightarrow$ (IGEN)

Feladatok

F2.1. Készítsünk a mellékelt adatfájlból egy címkézett listát (`description`) tartalmazó LaTeX dokumentumot. A címkék legyenek az első oszlopban található kémiai vegyjelek.

F2.2. Készítsünk egy LaTeX dokumentumot, amelyet figyelemfelkeltő címmel látunk el. Szerzőként adjuk meg a nevünket és a tanulmányi azonosítónkat. A dokumentumban szerepeljen három fejezet (`chapter`), és pár bekezdésnyi folyószöveg. Az első fejezet szövegében hivatkozzunk a harmadik fejezetre, és viszont, a harmadikból az elsőre. Továbbá ugyanitt hivatkozzunk arra is, hogy hányadik oldalon kezdődik a második fejezet.

F2.3. Készítsünk egy LaTeX dokumentumot, amelyben az alábbi képlet szerepel a szövegben és kiemelve is:

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$$

```
\documentclass[12pt, twoside]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[magyar]{babel}

\title{Ez a cím}
\author{Szerző}
\date{2009. szeptember 21.}

\pagestyle{plain}
\newcommand{\ELTE}{Eötvös Loránd Tudományegyetem}

\begin{document}
  \maketitle
  \tableofcontents

  \section{Bevezetés}
  Ez egy példa, amely egyszerre
  \begin{itemize}
    \item rövid, és
    \item és tartalmas.
  \end{itemize}

  \section{Konklúzió}
  Még egy képlet:
  \begin{equation}
    \sum_{n=0}^{N-1} \rho^n=\frac{1-\rho^N}{1-\rho}
    \label{eq:geom_sum}
  \end{equation}
  \dots és a \ref{eq:geom_sum} képlettel már vége is.
\end{document}
```

2.2. ábra. Egy összetett LaTeX forrásfájl tartalma.

```
% Gy2.3 gyakorló feladat: rendezze az alábbi dokumentumot a
% Latex-nek megfelelő struktúrába.

\author{Gipsz Jakab}
\title{Gy2.3 gyakorló feladat}
\date{}

\end{document}

\section{Az ékezetes betűk}
Az ékezetes betűk használatához be kell tölteni a
\textsf{fontenc} és az \textsf{inputenc} kiegészítő csomagokat.

\maketitle
\documentclass[12pt]{article}

\subsection{Ékezetek \LaTeX{} parancsokkal}
Ha nem töltöttük volna be a szükséges csomagokat, akkor \LaTeX{}
parancsokkal \'\i{}gy kellene magadni az \'ekezetes bet\H{u}ket a
forr\'asf\'ajlban.  Ez a megold\'as hosszabb sz\"oveg eset\'en
k\'ets\'egtelen\"ul el\'eg k\'enyelmetlen lenne.  Ha viszont
valaki ismeri az \'ekezetek bevitel\'ere alkalmas parancsokat,
akkor az k\'epes tetsz\H{o}leges bet\H{u}re tetsz\H{o}leges
\'ekezetet fel\'\i{}rni: p\'eld\'aul \~n, \`e,\"\i, \c{o}, \dots

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[magyar]{babel}
```

2.3. ábra. Egy összekevert LaTeX forrásfájl.

```
% Gy2.4 gyakorló feladat: Írja az alábbi kifejezéseket a
% megfelelő helyekre

% chapter twoside babel utf8 usepackage document book document
% itemize enumerate babel fontenc

\documentclass[.........]{.........}
\usepackage[T1]{.........}
\.........[.........]{inputenc}
\usepackage[magyar]{.........}

\author{Gipsz Jakab}
\title{Gy2.4 gyakorló feladat}
\date{}
\begin{.........}
\maketitle
\.........{Bevezetés}
Ez a feladat az alapvető \LaTeX{} dokumentum struktúra
felépítésének gyakorolását segíti.

\section{Betűtípusok}
Az alábbi betűtípusok közül választhatunk:
\begin{.........}
  \item \textrm{antikva}
  \item \textit{kurzív}
  \item \texttt{írógép}
  \item \textsf{groteszk}
\end{itemize}

\section{Betűváltozatok}
Minden betűtípushoz választhatjuk az alábbi változatokat,
akár többet is:
\begin{enumerate}
  \item \textup{álló}
  \item \textsl{döntött}
  \item \textmd{félkövér}
  \item \textbf{kövér}
  \item \textsc{kiskapitális}
\end{.........}

\textsl{\textbf{\textsf{Ez a mondat tehát döntött kövér és
groteszk.}}}

\end{.........}
```

2.4. ábra. Egy hiányos LaTeX forrásfájl.

```
% Gy2.5 gyakorló feladat: Az itemize környezetben minden
% sor elé szúrjuk be az \item parancsot.  Cseréljük ki
% szövegben a zárójelben álló részeket IGEN-re (vagy NEM-re).
% Pl. (Igen / Nem) => (IGEN)


\documentclass[12pt]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[magyar]{babel}

\author{Gipsz Jakab}
\title{Gy2.5 gyakorló feladat}
\date{}
\begin{document}
\maketitle
\section{Bevezetés}
Ez a feladat a parancsismétlés és a Block-visual mód
gyakorolását segíti.

\section{Parancsismétlés és a Block-visual mód használata}
A \LaTeX{} dokumentumok
\begin{enumerate}
szépek (Igen / Nem)
rendezettek (Igen / Nem)
struktúráltak (Igen / Nem)
karakteres szövegfájlból állnak (Igen / Nem)
rendszerfüggetlenek (Igen / Nem)
matematikai képletekhez valók (Igen / Nem)
nyomdai minőségűek (Igen / Nem)
tipográfiailag megtervezettek (Igen / Nem)
egyszerűek (Igen / Nem)
könnyen megtanulhatóak (Igen / Nem)
\end{enumerate}

\end{document}
```

2.5. ábra. A parancsismétlés és a visual-block mód használata.

3. fejezet

Ábrakészítés

A dokumentumok készítésének egyik alapvető feladata az igényes ábrák alkotása és beillesztése a dokumentumba. Ennek a feladatnak az ellátására fejlesztették ki a *gnuplot* nevű alkalmazást. A *gnuplot* egy parancssor vezérelt interaktív adat- és függvényábrázoló program. A program szinte minden használatban lévő operációs rendszerre megtalálható és ingyenesen letölthető.

A *gnuplot* segítségével képesek leszünk pl. analitikus alakban ismert függvényeket kirajzolni, adatfájlokat tartalmát grafikusan megjeleníteni, két és háromdimenziós grafikonokat készíteni, fájlok oszlopaival egyszerű matematikai műveleteket végezni, adatokra görbét illeszteni, animációt levetíteni, számos grafikus formátumban elmenteni az elkészült ábrát.

A *gnuplot* az összes felsorolt feladatot nem csak interaktív módban, hanem szkriptek alapján végezve is képes ellátni.

3.1. Gnuplot olvasnivalók

A *gnuplot* megismeréséhez szintén hatalmas mennyiségű online elérhető segédlet áll rendelkezésre. Ezek közül az alábbi magyar és angol nyelvűeket ajánljuk az olvasó figyelmébe:

- a Gnuplot howto[6] oldalt egy fizikus kolléga *tollából*,
- a Gnuplot használata[7] című segédanyagot, valamint
- a gnuplot – An Interactive Plotting Program[8] című kézikönyvet.

3.2. A *gnuplot* interaktív használata

A *gnuplot* programot *interaktív* vagy *szkript értelmező* üzemmódban használhatjuk. Az utasításkészletet tekintve nincs gyakorlati különbség a két üzemmód között, különbség csak a program futtatásának módjában van. Míg interaktív módban sorról-sorra, parancsról-parancsra adhatjuk meg az utasításokat, addig a szkript értelmező módban a már előre megírt utasításokat egy fájlból olvastatjuk be az ábrázoló programmal. Az interaktív módban könnyen kísérletezhetünk az ábra részleteivel, szkript üzemmódot pedig akkor célszerű használnunk, ha több hasonló ábra létrehozását szeretnénk automatizálni, vagy ha egy ábránk elkészítését szeretnénk rögzíteni. Szkriptek használatával lényegesen lerövidítjük a munkánkra szánt időt. Ebben a fejezetben elsőként az interaktív használattal ismerkedünk meg.

A *gnuplot* programot az alábbi parancs kiadásával tudjuk elindítani:

```
user@host:~$ gnuplot
```

Ennek hatására a terminálablakban megjelenik a *gnuplot* prompt, ahova a utasításokat fogjuk begépelni. Kilépni az `exit`, a `quit` parancsokkal, vagy a `<Ctrl-d>` billentyűkombináció leütésével tudunk.

Az ábrákat néhány paranccsal és azok egzotikus paraméterezésével készíthetjük el. Jellemzően a paramétereknek csak kis töredékét tartja az átlagos felhasználó fejben, a részleteket a dokumentációban lehet megkeresni.

Általános segítséget a már elindított *gnuplot* programban a `help` paranccsal kérhetünk. A `help` parancs segítségével rengeteg részletet tudhatunk meg az utasításokról, illetve azok paraméterezéséről. Ha valamely konkrét parancsról szeretnénk többet megtudni akkor a `help` parancs után adjuk meg a parancs nevét. Például a rövidesen bemutatásra kerülő `plot` parancsról a `help plot` paranccsal tudhatunk meg részleteket.

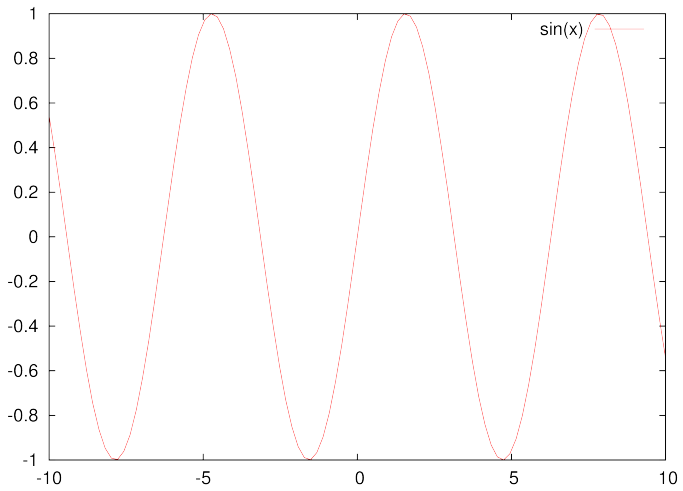
3.2.1. A `plot` parancs

A `plot` utasítás a *gnuplot* legtöbbet használt parancsa. A `plot` parancs segítségével tudunk adatokat illetve függvényeket ábrázolni. Számos, az ábrázolással kapcsolatos, esztétikai beállítás is megadható ennek a parancsnak a paramétereiként.

Az alábbi, legegyszerűbb példával szemléltethetjük a `plot` (ábrázolás) parancs használatát:

```
gnuplot>plot sin(x)
```

Ennek hatására a 3.1 ábrán látható *szinusz* függvény jelenik meg $x \in [-10; 10]$ tartományon, folytonos piros vonallal.



3.1. ábra. A $\sin(x)$ függvény grafikonja *gnuplot*-tal.

Függvények ábrázolása

A `plot` parancs segítségével lehetőségünk van zárt, illetve paraméteres alakban megadott függvények ábrázolására. A 3.1. táblázatban a leggyakrabban használt beépített függvényeket gyűjtjük össze. Fontos megjegyezni, hogy a szögfüggvények radiánban értelmezettek. A program kezeli a komplex számokat, így a függvények megengedik a komplex értékű változókat is.

A beépített függvényekből saját függvényeket is definiálhatunk. Próbáljuk ki a következő *gnuplot* parancsokat:

```
gnuplot> f(t) = sin(t) + a * cos(t)
gnuplot> a = 2
gnuplot> plot f(x)
```

Az első sorban adtuk meg a függvény definícióját. A definícióban először megadtuk az új függvény nevét, majd zárójelben felsoroltuk annak *változóit*. Az értékadó operátor „=” után szerepel a függvénydefiníció kifejtése.

A függvénydefiníció kifejtésében szerepelhetnek a változó- és függvényneveken kívül további címkék is, melyeket *paramétereknek* nevezünk. Ahhoz, hogy egy függvényt használhassuk – például ábrázoljuk –, a benne szereplő paramétereknek külön parancsban értéket kell adnunk, hasonlóan a fenti példa második sorához.

A fenti példában `f` a függvény neve, `t` a változó neve, `a` pedig egy paraméter. Természetesen több változót is megadhatunk, ezeket a függvény neve utáni zárójelben vesszővel kell elválasztanunk egymástól.

3.1. táblázat. A *gnuplot* által támogatott alapfüggvények

Függvényalak	A függvény értelme
`abs(x)`	abszolútérték
`acos(x)`	árkusz-koszinusz
`asin(x)`	árkusz-szinusz
`atan(x)`	árkusz-tangens
`cos(x)`	koszinusz
`cosh(x)`	koszinusz-hiperbolikus
`erf(x)`	hibafüggvény
`exp(x)`	természetes kitevő függvény
`inverf(x)`	hibafüggvény inverze
`invnorm(x)`	Gauss-függvény inverze
`log(x)`	természetes alapú logaritmus
`log10(x)`	tízes alapú logaritmus
`norm(x)`	normált 1-szórású Gauss-függvény
`rand(x)`	álvéletlen
`sgn(x)`	előjelfüggvény
`sin(x)`	szinusz
`sinh(x)`	hiperbolikus-szinusz
`sqrt(x)`	négyzetgyök
`tan(x)`	tangens
`tanh(x)`	hiperbolikus-tangens

A függvények, a változók és a paraméterek nevei egy vagy több betűből, számból, illetve aláhúzásból állhatnak, például `f_1`, azzal a megkötéssel, hogy a nevek nem kezdődhetnek számmal.

A változók és a paraméterek különböző szerepet játszanak a függvények definícióiban. A függvények ábrázolásánál csak változókat használhatunk, függvények adatokra való illesztésénél pedig paramétereket. Ezen kívül a paramétereket globális változóként is használhatjuk, azaz ha több függvény definíciójában ugyanazt a paramétert használjuk, akkor ennek a paraméternek az értéke az összes függvény értékét befolyásolja.

A fenti példa harmadik sorában a `plot` paranccsal ábrázoltuk a függvényt. Vegyük észre, hogy a `plot` utasításánál az `x` változónevet adtuk meg, nem a függvény definíciójában szereplő `t` vátlozót. Ez nem véletlen, a normál egyváltozós ábrázolásnál kötelezően `x` jelöli a függvény változóját, kétváltozós (térbeli) ábrázolásnál pedig kötelezően `x` és `y`.

Adatsor ábrázolása

A *gnuplot* használata során nagyon gyakran nem analitikus függvényeket, hanem valamilyen számszerű adatsort szeretnénk ábrázolni. Ehhez szükségünk van az adatokat tartalmazó ASCII fájlra, amiben az adatok táblázatszerűen vannak elrendezve. Az adatfájlban az egy sorban lévő adatokat szóközzel, vagy tabulátorral kell elválasztani.[1]

Adatok ábrázolásánál leggyakrabban az adatfájl egyik oszlopában található mennyiségeket ábrázoljuk egy másik oszlopának függvényében. Látni fogjuk, hogy az adatokon függvénytranszformációt is végezhetünk. Ilyen transzformációk segítségével akár több oszlop értékeit is felhasználhatjuk.

Az adatsorok ábrázolásának kipróbálásához töltsük el a gyakorlat weboldaláról a mintafájlt, melyet mentsünk el `sinusadatok.dat` néven. Az adatfájlban lévő adatokat szintén a `plot` paranccsal ábrázolhatjuk. Az ábrázoláshoz a `plot` parancs után a fájlnevet idézőjelek között kell megadni:

```
gnuplot>plot "sinusadatok.dat"
```

Ha az adatfájl nem abban a könyvtárban van, ahonnan a *gnuplot*-ot elindítottuk, akkor meg kell adnunk az elérési útvonalat is (ld. 9.4. fejezet).

Ezen a ponton meg kell jegyezni, hogy a *gnuplot*-ban a parancsokat és az opcionális paramétereket gyakran addig rövidíthetjük, amíg a rövidítés egyértelmű. A `plot` és a `help` parancs esetén elegendő az kezdőbetűt megadnunk. Ezt a lehetőséget a következő példákban ki is próbálhatjuk:

```
gnuplot>p "sinusadatok.dat"
```

[1] Lehetőségünk van az adatfájlban egyéb szeparátort is használni, lásd `help set datafile separator`

A *gnuplot* a több oszlopos a fájlok esetén alapértelmezett esetben az első oszlop függvényében ábrázolja a második oszlop adatait. Amennyiben csak egy oszlopa van az adatfájlnak akkor ezt az oszlopot ábrázolja az adott sor sorszámának függvényében.

Adatoszlopok használata

A `plot` parancs működését számos opcionális kapcsoló segítségével szabályozhatjuk. A fontos megjegyeznünk, hogy kapcsolók csak meghatározott sorrendben adhatók meg.

Az első, és egyik leggyakrabban használt kapcsoló a `using`, mellyel megadhatjuk, hogy a megadott adatfájl mely oszlopait ábrázolja a *gnuplot*. Ehhez a legegyszerűbb esetben a `using` kapcsoló után két számot kell megadnunk kettősponttal elválasztva; az első szám az x, a második pedig az y adatokat tartalmazó oszlop sorszámát jelenti. A „nulladik" oszlopnak speciális jelentése van, ezzel hivatkozhatunk arra, hogy az adott adat hanyadik sorban szerepel.

A többoszlopos fájlok alapértelmezett ábrázolását jeleníti meg például a

```
gnuplot>plot "sinusadatok.dat" using 1:2
```

parancs.

Nemcsak az oszlopok sorszámát adhatjuk meg azonban, hanem tetszőleges függvényt is alkalmazhatunk az adatokra. Ábrázoljuk például a harmadik oszlop értékeinek kétszeresét a második oszlop függvényében! Ezt a következő paranccsal tehetjük meg:

```
gnuplot>p "sinusadatok.dat" u 2:(2*$3)
```

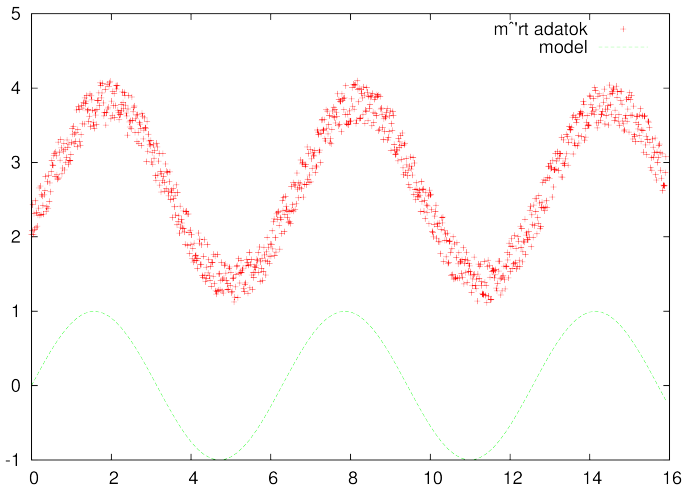
Vegyük észre, hogy amikor matematikai műveletet kívánunk végezni az adott oszlop elemeivel, akkor a kifejezést zárójelbe kell tenni, valamint az oszlopokra a „$" jel beiktatásával kell hivatkoznunk, ellenkező esetben számkonstansként és nem az oszlop azonosítójaként értelmezi a program.

Egyetlen `plot` parancs kiadásával egyszerre több függvényt illetve adatgörbét is ábrázolhatunk. Ilyenkor az ábrázolandó görbéket vesszővel („,") kell elválasztanunk egymástól. Hasznos megjegyezni, hogy ha egy állomány adataiból egyszerre több görbét is készítünk egyszerre, akkor másodjára már elhagyhatjuk a fájl nevét arra „üres" fájlnévvel hivatkozhatunk:

```
gnuplot>plot "sinusadatok.dat" u 2:(2*$3), "" u (0.7*$2):(2*$3), sin(x)
```

Jelmagyarázat

Ha több függvényt vagy adatot ábrázolunk ugyanazon az ábrán, akkor fontos megadni, hogy a különféle vonal- vagy ponttípusok mit jelentenek. Ezt az információt a jelmagyarázatban adhatjuk meg. Alapesetben a *gnuplot* az ábra jobb felső sarkában minden



3.2. ábra. *gnuplot* ábra jelmagyarázattal. A jelmagyarázat az ábra belső részének jobb felső sarkában található.

vonal- és ponttípus mellé kiírja a megadott analítikus függvényeket illetve a megadott fájlnevet és a `using` opciót.

Amennyiben módosítani szeretnénk a jelmagyarázatot, akkor a `using` kapcsoló utáni `title` kapcsolóval tehetjük meg:

```
gnuplot>plot "sinusadatok.dat" u 1:3 t "adatok", sin(x) t ""
```

A fenti példa eredménye látható a 3.2. ábrán. Amennyiben valamely vonalhoz vagy szimbólumhoz nem kívánunk jelmagyarázatot fűzni, a `title` kapcsoló után adjunk meg üres címet, vagy használjuk a `notitle` kapcsolót.

Az ábrázolási tartomány

A korábbi példákban az x- és y-tengelyek ábrázolási tartományát minden esetben automatikusan állította be a program. Amennyiben másként nem rendelkezünk, a *gnuplot* a függvényeket a $[-10; 10]$ intervallumon, az adatsorokat pedig azok értelmezési tartományán jeleníti meg. Az y-tengelyt alapesetben úgy állítja be a *gnuplot*, hogy az ábrázolt függvény vagy adat teljes egészében látható legyen az ábrán.

Az ábrázolási tartományokat természetesen szabadon megváltoztathatjuk. Az alapértelmezett ábrázolási tartományokat a `plot` parancs után megadott, `[`*tól*`:`*ig*`]` formátumú opcionális argumentumokkal lehet felülbírálni. Az első ilyen argumentummal az x tartományt, a másodikkal az y tartományt, stb. lehet beállítani, például

```
gnuplot>plot [100:200] "sinusadatok.dat"
```

3.2. táblázat. A `with` kapcsolóval megadható fontosabb vonal-stílusok

Stílus	Kapcsoló	Rövid forma
vonal	`lines`	`l`
pontsor	`points`	`p`
vonal és pontok	`linespoints`	`lp`
kicsi pontok	`dots`	`d`
tüskék	`impulses`	`i`
pontok hibasávval	`errorbars`	`e`

Ha valamely határnak a $*$ jelet adjuk meg, akkor a *gnuplot* az adott határt automatikusan fogja beállítani, ha pedig bármely határt elhagyhatjuk, akkor a *gnuplot* az éppen alapértelmezett határt fogja használni.

```
gnuplot>plot [100:]  "sinusadatok.dat"
gnuplot>plot [100:*] "sinusadatok.dat"
```

Előfordulhat, hogy csak az y tengely ábrázolási tartományát akarjuk beállítani. Mivel az egyértelműség megköveteli, hogy az x-tengelyre vonatkozó tartomány is szerepeljen, ezért ha az x tengelyhez az alapértelmezett értékeket szeretnénk használni, akkor mind az alsó, mind a felső határokat hagyjuk üresen:

```
gnuplot>plot [][2:4] "sinusadatok.dat"
```

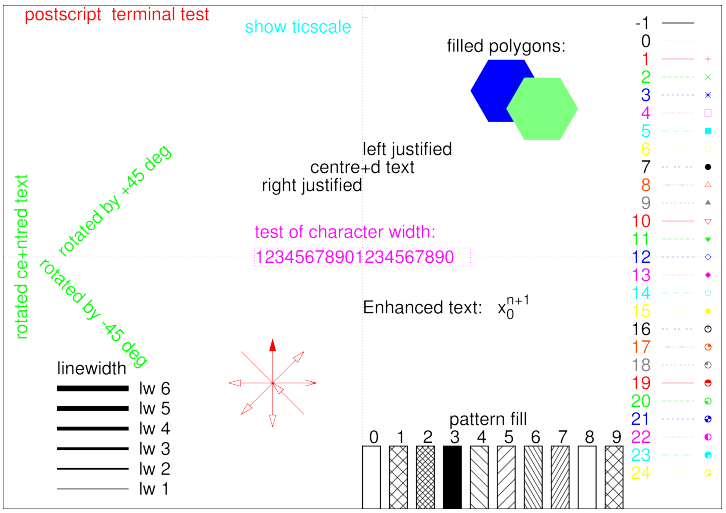
Görbék megjelenésének formázása

A *gnuplot* szinte minden állítható tulajdonságához vannak alapértelmezett értékek, így nem szükséges minden egyes részletet beállítanunk, hogy gyorsan ábrázolni tudjunk egy görbét. Azonban amennyiben az eredeti beállítások nekünk nem felelnek meg, akkor ezeket át tudjuk állítani. Ilyenek az ábrázolt görbék vonal, vagy szimbólum típusai, színei és egyéb jellemzői is.

Vonalstílus Az ábrázolt mennyiségeket a *gnuplot* alapvetően kétféle módon jeleníti meg: a függvényeket vonalakkal, az adatsorokat pedig szimbólumokkal. Ettől természetesen eltérhetünk a `with` (röviden `w`) kapcsoló használatával:

```
gnuplot>plot "sinusadatok.dat" with lines
gnuplot>plot "sinusadatok.dat" w l
```

A 3.2. táblázatban a leggyakrabban használt görbe ábrázolási stílusokat foglaljuk össze, a teljes készletet a `help plot with` paranccsal nézhetjük meg a dokumentációban.



3.3. ábra. A *gnuplot* `test` parancsának kimenete postscript terminálon. A jobb oldalon látható az adott sorszámhoz tartozó vonal- és ponttípus.

Vonalszín és vonaltípus A vonalak, szimbólumok színét a `plot` parancs `linecolor` kapcsolójával lehet beállítani. A lehetséges értékek 0 és 7 közötti egész számok, amik a különféle színeknek felelnek meg. A következő parancs a kék vonallal köti össze az adatpontokat:

```
gnuplot>plot "sinusadatok.dat" w l lc 3
```

A vonalakat színük mellett szaggatottságukkal is megkülönböztethetjük egymástól. Ez a lehetőség különösen fekete-fehér nyomtatás esetén fontos. A vonalak szaggatottságát a `plot` parancs `linetype` kapcsolója után megadott egész számmal szabályozhatjuk, például:

```
gnuplot>plot "sinusadatok.dat" w l lt 2
```

Fontos megjegyeznünk, hogy a `linetype` kapcsoló hatása a megjelenítő terminál típusa szerint más és más lehet. A későbbiekben ismertetett `eps` kimenet esetében például a szín és a vonal szaggatottság egyszerre változik. Egy adott terminál lehetséges megjelenési beállításaihoz nyújt segítséget a `test` parancs, melynek `eps` kimenete a 3.3. ábrán látható.

A `linetype` kapcsolónak 0 vagy -1 értéket is adunk. Az előbbivel vékony sűrűn pontozott vonalat, az utóbbival vastag folytonos vonalat kapunk, mindkettőt fekete színnel.

Vonalvastagság Az ábrázolt vonalak vastagságát is állíthatjuk, ami különösen fontos a nyomtatásra szánt ábrák esetén. Általában a képernyőn még jól látszó vonalak

nyomtatásban nehezen észrevehetőek, túlságosan vékonyak lesznek. Ennek elkerülésére a nyomtatásra szánt dokumentumokban szükséges lehet a vonalak vastagságát megnövelni. Ezt a `plot` parancs `linewidth` (röviden `lw`) kapcsolójának használatával lehet elérni:

```
gnuplot>plot sin(x) w l lw 5
```

A fenti példában ötös vastagságú vonallal ábrázoljuk a szinusz-függvényt.

A pontok típusa A mérési adatokat általában pontokkal ábrázoljuk. Az ezekhez használt szimbólumokat a `plot` parancs `pointtype` (`pt`) kapcsolójával állíthatjuk be. A kapcsoló után megadott számmal adhatjuk meg, hogy melyik szimbólumot szeretnénk használni. A lehetséges szimbólumokat a *gnuplot* `test` parancsára megjelenő ábra jobb szélén láthatjuk felsorolva.

```
gnuplot>plot sin(x) w p pt 6
```

A fenti példa a hatos sorszámú (belül üres kör alakú) szimbólum típust használja.

A pontok mérete Az adatpontokat reprezentáló szimbólumok méretének beállításához használjuk a `plot` parancs `pointsize` kapcsolóját. Itt lehetőségünk van lebegőpontos számértékeket is megadni a finom beállítás érdekében.

```
gnuplot>plot sin(x) w p pt 6 ps 2.5
```

A fenti példa 2.5-ös méretű körökkel ábrázolja az adatpontokat. Próbáljunk ki több különböző méretet, akár 1-nél kisebb értékeket is!

3.2.2. Újrarajzolás

A `replot` paranccsal megismételhetjük az előzőleg kiadott `plot` parancsot. Ez akkor hasznos például, ha szeretnénk a következő fejezetben ismertetett `set` parancs beállításaival az ábrát megnézni, illetve ha szeretnénk ugyanazt az ábrát különböző terminálokon ábrázolni.

A `replot` parancs után az ábrázolási tartományon kívül a `plot` parancs bármely argumentuma megadható. Ennek hatása megegyezik azzal, mintha az előző `plot` parancs után, az elválasztó vessző után, adtuk volna a meg a `replot` parancs argumentumát.

3.2.3. A `set` parancs

A *gnuplot* program egyik leghasznosabb tulajdonsága, hogy az ábra minden apró részlete beállítható. A `plot` parancs opcionális argumentumai segítségével az ábra számos elemét szabadon beállíthatjuk, de ezek a beállítások mindíg az adott plot parancsra vonatkoznak. Ha egy új `plot` parancsot adunk ki, akkor ezeket a beállításokat újra meg kell adni.

A `set` paranccsal lehetőségünk van arra, hogy az alapértelmezett beállításokat változtassuk meg. Az így megadott beállítások az összes későbbi `plot` parancsra vonatkoznak, hacsak nem módosítjuk a `plot` parancson belül a beállításokat.

A `set` paranccsal számos olyan beállítás is megadható, amelyet a `plot` parancs segítségével nem tudunk megváltoztatni, például a kimenet, a terminál típusa, vagy a tengelyfeliratok.

A `set` parancsnak hozzávetőlegesen 120 változata van. Ezek közül számos nagyon hasonló – minden tengelyre például hasonló parancs vonatkozik, amelynek csak egy vagy két kezdőbetűje tér el egymástól – néhány parancsot pedig valószínűleg elvétve fogunk használni. Az alábbiakban a `set` parancsnak csak a legfontosabb változatait tekintjük át. Mindazonáltal fontos megjegyezni, hogy ha az ábra valamely különleges elemét meg szeretnénk változtatni, akkor azt minden bizonnyal a `set` paranccsal megtehetjük.

3.2.4. Az ábra címe

A `set` parancs segítségével az ábráknak adhatunk címet. A cím alapesetben az ábrák kerete felett jelenik meg:

```
gnuplot>set title "szinus meresi adatsor"
```

A címet az `unset title` parancs segítségével vehetjük le az ábráról.

3.2.5. Tengelyfeliratok

Egy grafikon tengelyein fontos feltüntetni, hogy milyen mennyiségek értékét ábrázoljuk, milyen más mennyiség értékeinek függvényében. Ha az ábra mérési adatokat tartalmaz, akkor a mértékegységeket is szerepeltetni kell.

Az x- és az y-tengelyek feliratait a `xlabel` és a `ylabel` (röviden `xla` és `yla`) változók beállításával adhatjuk meg. A beállítások után a `replot` paranccsal ábrázolhatjuk újra az utolsó `plot` parancsban foglaltakat:

```
gnuplot>set xlabel "ido [sec]"
gnuplot>set yla "kiteres [m]"
gnuplot>replot
```

Jelmagyarázat elhelyezése

Amint azt a 3.2.1. szakaszban láttuk, az ábrázolt görbékhez a jelmagyarázatban megjelenő felirat a `plot` parancs kiadásakor állítható be a `title` (röviden `t`) kapcsolóval:

```
gnuplot>plot "sinusadatok.dat" title "inga kitérése"
gnuplot>p "sinusadatok.dat" t "mért adatok", sin(x) t "elméleti görbe"
```

A jelmagyarázatot a `set key` paranccsal szabhatjuk testre. A jelmagyarázatot kikapcsolhatjuk a `set key off` vagy az `unset key` parancsokkal. A `set key` parancsnak rengeteg opcionális argumentuma van, ezek közül most csak az elhelyezésre vonatkozókat említjük.

Alapesetben a jelmagyarázat a bal felső sarokban van. Előfordul azonban, hogy ezen a részen az ábra más részei is vannak, ezért a jelmagyarázatot az ábra más részeire kell mozgatni.

A jelmagyarázat lehet az ábrán belül és kívül. Az ábrán belül a `left`, `right`, `center`, illetve `top`, `bottom`, `center` parancsok bármely kombinációját megadhatjuk. Az ábrán kívüli jelmagyarázatot a `tmargin`, `bmargin`, `lmargin` és `rmargin`, valamint az előző kulcsszavak megfelelő kombinációival adhatunk meg. A

```
gnuplot>set tmargin left
```

parancs például az ábrá felső margójának bal oldalára helyezi a jelmagyarázatot. A `set lmargin left` parancs esetén azonban a `left` kulcsszót figyelmen kívül hagyja a *gnuplot*, mivel ez nem gyeztethető össze a `lmargin` kapcsolóval..

3.2.6. Az ábrázolási tartomány

A 3.2.1. szakaszban bemutattuk, hogy az ábrázolási tartományokat hogyan állíthatjuk be a `plot` parancs első opcionális argumentumával. Említettük azt is, hogy ez a beállítás csak az adott `plot` parancsra vonatkozik, és mindíg felülírja az alapértelmezett beállítást.

Ezzel szemben, a `set` parancs segítségével magát az alapértelmezett ábrázolási tartományokat változtathatjuk meg. Például a

```
gnuplot>set xrange [100:200]
```

parancs az x tengelyen a $[100, 200]$ intervallumra állítja az alapértelmezett ábrázolási tarományt. A többi tengely beállításához értelemszerűen a `set yrange` és a `set zrange`, stb. parancsokat kell használnunk.

Amennyiben szeretnénk visszaállítani az automatikus tartománybeállítást, használjuk `set xrange [*:*]` vagy a `set autoscale` parancsokat.

3.2.7. A kész ábra elmentése

A *gnuplot* többféle „terminálon” képes az ábrákat megjeleníteni. Amikor a *gnuplot*ot elindítjuk, az alapértelmezett kimenet a számítógép monitora. Az ennek megfelelő terminál a használt operációs rendszertől függ, Linuxos környezetben a `wxt`, vagy `x11` terminálok használhatók, MS Windows környezetben a `wxt` wagy `windows` terminálok, MacOS rendszeren pedig az `aqua` terminál.

Az éppen kiválasztott terminál típusa a `show terminal` paranccsal olvasható ki. Természetesen legtöbb esetben nem elégszünk meg azzal, hogy a monitoron vizsgáljuk az elkészült ábrát. A képernyőkimeneten kívül különféle képformátumokat lehet terminálként megadni, például Postscript, JPG, PNG, TIFF, GIF, stb, melyeket elmenthetünk, hogy más program számára is elérhetők legyenek.

A *gnuplot*ban az elkészült ábrák elmentése egy több lépésből álló folyamat. Ennek során első lépésben át kell állítanunk a megjelenítő terminált a kívánt ábraformátumra. Ezt a `set terminal` paranccsal tehetjük meg. Ezután a `set output` paranccsal meg kell adni a kimenetet. Abban az esetben, ha a `set output` parancs után megadunk egy fájlnevet, akkor minden további `plot` parancs kimenete a megadott fájlba kerül, ha pedig nem adunk meg fájlnevet, akkor a kimenet a terminálától függően a képernyő vagy a standard kimenet lesz. Végezetül az ábrát a megfelelő `plot` vagy `replot` paranccsal újra ki kell rajzolni. Ezáltal az ábra az újonnan megadott terminál típusban, a megadott fájlba íródik ki.

Lássunk erre egy példát, melyben színes *png*[2] ábrát készítünk a szinusz-függvényről:

```
gnuplot>plot sin(x) w p pt 6 ps 2.5
gnuplot>set terminal png color
gnuplot>set output "abra.png"
gnuplot>replot
gnuplot>set output
gnuplot>set terminal wxt
```

Figyeljük meg, hogy a `replot` parancs kiadása után nem a képernyőn jelenik meg újra az ábra, hanem a parancs hatására a megadott fájlba íródik az ábrát leíró tartalom.

Figyeljünk arra, hogy a `set output` utasítással le kell zárnunk a kimeneti fájlt, különben a későbbi `plot` parancsokkal felülírjuk a kész ábrát.[3] Ha szeretnénk újra a képernyőn megjeleníteni az ábrát, vissza kell állítani a célterminált.

A fenti utasítássorozat eredményeként elkészül egy `abra.png` nevű képfájl a felhasználói területünkön. Megjegyezzük, hogy egyes LaTeX-sablonok használata esetén a dokumentumokba legkönnyebben *eps* formátumú ábrát tudunk beilleszteni. Ezt a formátumot a `postscript` terminál kiválasztásával kaphatjuk meg:

[2] Az angol portable network graphics rövidítése

[3] Postscript terminál használatakor a többszöri `plot` parancsra külön oldalakra készülnek el az ábráink.

```
gnuplot> set terminal postscript enhanced eps color
```

vagy tömörebben:

```
gnuplot> set term po enh eps c
```

3.3. A *gnuplot* szkriptek használata

Legtöbb *gnuplot* implementáció a kiadott utasítások sorozatát naplózza. Így az előző fejezetben megismert interaktív használatkor a régebben kiadott parancsok a „felfele”-gomb segítségével előhozhatók, egyesével újra kiadhatók. Ezen túlmenően lehetőség van a *gnuplot* teljes pillanatnyi állapotának elmentésére, valamint újbóli betöltésére is.

Mint arról már az előző fejezetben is szót ejtettünk, a megjelenő ábra számos változó (pl. tengely feliratok, betűtípus, színek) pillanatnyi értékétől függ. A `save` parancs segítségével a elmenthetjük a *gnuplot* pillanatnyi állapotát:

```
gnuplot> save "abra.plt"
```

A parancs eredményeként egy `abra.plt` nevű szöveges állomány keletkezik. Ez az állomány az elmentett állapot összes beállítását tartalmazó `set` parancsokat, az összes függvényt és globális változót, valamint a legutóbbi `plot` parancsot tartalmazza. Az így kimentett állapot a `load` paranccsal tölthető vissza, melynek hatására fájlban felsorolt utasítások rendre lefutnak:

```
gnuplot> load "abra.plt"
```

A *gnuplot* parancsokat tartalmazó állományokat – mint például a `save` paranccsal elmentett fájlok – *gnuplot* szkripteknek nevezzük. A *gnuplot* szkriptjeinket a program indításakor argumentumként is megadhatjuk:

```
user@host:~$ gnuplot abra.plt
```

Ekkor a *gnuplot* szkriptfeldolgozó módban indul el. A *gnuplot* ilyenkor nem vár interaktív parancsokra, nem ad prompt-ot, hanem lefuttatja az `abra.plt` tartalmát, majd vissza is kapjuk a parancsértelmező promptját.

Egy ilyen szkriptállományt bármilyen szövegszerkesztővel létre lehet hozni. Például *vim*mel készítsünk el egy ilyen szkriptet. A `vim abra2.plt` paranccsal új szöveges állományt nyitunk, melynek legyen a következő a tartalma:

```
set title "cosinus gorbe"
set xlabel "ido [s]"
plot [0:][-0.5:1] cos(x)
pause -1
```

A *gnuplot* állapotát leíró összes változó mindegyikének van alapértelmezett értéke, így ha valamit nem állítunk be, akkor annak az alapértelmezett értéke marad érvényben. Az utolsó sorban a `pause -1` parancs segítségével egy `<Enter>` gomb nyomásáig megállítjuk a *gnuplot* futását, így az elkészült ábra tetszőleges ideig a képernyőn marad. A `pause mouse keypress` parancs egy az ábrán történő egérgomb, vagy más billentyű megnyomásáig tarja a képernyőn a grafikont. Próbáljuk is ki, indítsuk el a programot ezzel az új szkripttel:

```
user@host:~$ gnuplot abra2.plt
```

A szkript mód segítségével végleges ábrafájlokat is készíthetünk. Módosítsuk a korábbi scriptet a következőképpen, és mentsük el `abra3.plt` néven:

```
set title "cosinus gorbe"
set xlabel "ido [s]"
plot [0:][-0.5:1] cos(x)
set term po enh eps col
set output "sin.eps"
```

Ezután futtassuk le a szkriptet és győződjünk meg arról, hogy helyesen elkészült a `sin.eps` állomány.

3.4. Az ábrák beillesztése LaTeX-be

Az előző módon elkészített *eps* formátumú ábrát a LaTeX dokumentumokba a `figure` környezet használatával tudjuk beilleszteni. Ehhez a LaTeX dokumentum fejlécébe a a `\usepackage{`*graphicx*`}` paranccsal a `graphicx` csomagot be kell töltenünk. Ezután a dokumentum törzsében a kívánt helyre beszúrhatjuk az ábránkat a következő módon:

```
\begin{figure}
 \centerline{
  \includegraphics[width=.8\textwidth, angle=-90]{sin.eps}
 }
 \caption{Ez itt az ábránk ábraaláírása.}
 \label{fig:abra1}
\end{figure}
```

A fenti példában a következő LaTeX-parancsokat alkalmaztuk:

- a `\begin{`*figure*`}` az ún. úsztatott (floating) ábrák létrehozására alkalmas környezetet nyitja meg. Az úsztatott ábrának nincs rögzített helye a dokumentumban, hanem a LaTeX helyezi el azokat. Az úsztatott ábrák általában a lapok tetejére kerülnek, de ezt a `figure` környezet opcionális argumentumán keresztül módosíthatjuk.

- a `\centerline` parancs középre rendezi a zárójelei közé tett objektumot, ami esetünkben éppen az ábra lesz.
- Az ábrát magát az `\includegraphics[`*opciók*`]{`*fájlnév*`}` parancs segítségével illeszthetjük be a dokumentumba. A bemutatott példában az ábra szélességének megadására a `width`, az elforgatására a `angle` opciókat használtuk. A szélességet a szöveg szélességének 0,8-szorosára állítottuk, így automatikusan igazodik a papírmérethez. Az *fájlnév* helyére az ábrafájl neve, azaz `sin.eps` került.
- a `\caption` paranccsal az ábraaláírást tudjuk megadni.
- a `\label{`*címke*`}` paranccsal egy egyedi azonosítót adhatunk az ábrának. A `\label` paranccsal felcimkézett környezetekre a `\ref{`*cimke*`}` paranccsal tudunk hivatkozni, például így:

```
A(z) \ref{fig.abra1}.~ábrán látható elméleti modell ...
```

  Fontos, hogy a `\label` parancs a `\caption` parancs után legyen, különben helytelen számozást kapunk a cimkére való hivatkozáskor. A könnyebb áttekinthetőség kedvéért érdemes a cimkéket az adott környezetre jellemző résszel kezdeni, pl. „`fig:`”, „`sec:`”, stb, de ez nem kötelező.
- A `figure` környezetet végül az `\end{`*figure*`}` paranccsal zárjuk be.

3.5. Példák és feladatok

A példákhoz tartozó adatsorok letölthetőek a gyakorlat honlapjáról. A fájlok nevei a feladatok leírásánál szerepelnek. A feladatoknál mindig mutassuk be az elkészítés módját és a végeredményként készült ábrákat is.

Gyakorló példák

Gy3.1. Töltsük le a gyakorlat weboldalától a `sinusadatok.dat` adatsort. A *vim* szövegszerkesztő segítségével ismerkedjünk meg az adatfájl felépítésével. Készítsünk belőle `eps` formátumú ábrát, amin az adatfájl első oszlop értékeinek függvényében a második oszlop értékeit ábrázoltuk.

Gy3.2. A `sinusadatok.dat` adatfájlból készítsünk egy ábrát, amin

1. az adatsor második oszlopában található értékek függvényében ábrázoljuk az első és a harmadik oszlop adatainak összegét, illetve
2. ugyanebben az ábrában rajzoljuk ki a $13 + 4.5 * x$ egyenest is.

Formázásnak adjuk meg a következőket: az adatsorból származó pontok legyenek közepes méretű, kék színű háromszögekkel ábrázolva, míg az egyenes legyen fekete folytonos vonallal. Adjunk neveket az x és y tengelyeknek.

Gy3.3. Az előző feladatban készített ábrát mentsük el PostScript formátumban és illesszük be az egyik múlt órai LaTeX dokumentumba. Adjunk meg ábraaláírást, hivatkozzunk a szövegben az ábrára!

Feladatok

F3.1. Töltsük le a gyakorlat weboldaláról a `sinusadatok.dat` adatsort. A *vim* szövegszerkesztő segítségével ismerkedjünk meg az adatfájl felépítésével. Készítsünk belőle PostScript formátumú ábrát, amin az adatfájl első oszlopában található értékek függvényében a második oszlop értekeinek négyzetgyökét ábrázoljuk. (Útmutatás: a négyzetgyök kiszámításához használjuk a *gnuplot* `sqrt()` függvényét!).

F3.2. Az előző adatsort felhasználva készítsünk egy ábrát, amiben az x-tengelyen a ráközelítettünk a $[10; 20]$, az y-tengelyen az $y \geq 0$ tartományra. Az eredményből készítsünk PostScript ábrát, amit az `evince` parancs segítségével mutassunk be a gyakorlatvezetőnek.

F3.3. Az előző adatsort felhasználva készítsünk ábrát, amiben az ábrázolási tartományon túl számunkra tetsző színekkel, vonalvastagsággal ábrázoljuk az $f(t) = A\cos(wt) + c$ függvényt $A = 1.5$, $w = 2$, $c = 1$ paraméterek mellett, valamint jól látható szimbólumokkal az adatsor első oszlopának függvényében a harmadik és második oszlop különbségét. Az eredményből készítsünk PostScript ábrát.

F3.4. Az előző feladatban elkészített ábrát helyezzük el egy LaTeX dokumentumban (ez lehet valamelyik `tex` fájl az előző gyakorlatról), adjunk ábrafeliratot az ábrához, valamint helyezzünk el rá hivatkozást a szövegben.

F3.5. Készítsünk egy *gnuplot* szkriptet *vim*mel, melyben az $f(x) = 1/x$ és a $g(x) = \exp(x)$ függvényeket ábrázoljuk a $(0, 3)$ intervallumon. Az y-tengelyt válasszuk úgy, hogy a két függvény metszéspontja közelítőleg könnyen leolvasható legyen. Adjunk nevet a tengelyeknek, és a kész ábrát mentsük el PNG formátumban!

4. fejezet

Szöveges adatfájlok feldolgozása

Az ábrázolni és értelmezni kívánt adatsorok a legritkább esetben állnak a rendelkezésünkre abban a formában, amiben azt fel tudjuk használni. Általában több-kevesebb átalakítást kell rajtuk végrehajtani, hogy megfeleljenek a céljainknak. Ezek az átalakítások magukba foglalják mind a formai átalakításokat, mind a tartalmiakat. Ezekhez az átalakításokhoz kiváló segédeszköz az *awk* adatfeldolgozó program.

Az *awk* adatfeldolgozó program működésének alapja, hogy az adatfájlban megadott mintájú sorokat keres és azokon a felhasználó által definiált átalakításokat hajt végre. Ebben a fejezetben az GNU *awk*, azaz a *gawk* működésének alapjaival ismerkedünk meg.

4.1. *awk* olvasnivalók

Természetesen az *awk* programhoz is rengeteg információ lelhető fel a weben. Egy magyar nyelvű ezek közül:

- Hatékony *Awk* programozás – A GNU *Awk* felhasználói kézikönyve

A webes információkon kívül minden Linux parancshoz rendelkezésre áll a működést leíró kézikönyv (angolul *manual*). A kézikönyvet a terminálba beírt `man` paranccsal érhetjük el, ha utána írjuk miről kérünk leírást. Az *awk* GNU Linux variánsáról a

```
user@host:~$ man gawk
```

paranccsal kérhetünk tehát leírást.

4.2. A *gawk* futtatása

A *gawk*ot – akár csak a *gnuplot*ot – lehet parancssorból interaktívan futtatni, vagy lehetséges egy fájlból a megírt szkriptprogramot betölteni. Egyelőre mi az utóbbit fogjuk használni, melynek szintaxisa a következő:

```
user@host:~$ gawk -f programfájl  adatfájl > kimenetfájl
```

ahol a *programfájl* a *gawk* utasításokat tartalmazó fájl, az *adatfájl* az a fájl, amit a *gawk*kal feldolgozunk, a *kimenetfájl* pedig egy fájl, amibe a kimeneti adatokat el akarjuk menteni. A fenti parancssorban az `-f` a *gawk* program egy kapcsolója, `>` pedig a standard kimenetet átirányító operátor. A parancssor ezen elemeiről a 9. fejezetben lesz szó.

4.3. Az *awk* működése

Az *awk* az adatfeldolgozás során előre megadott mintákat keres az adatfájl rekordjaiban (alapesetben soraiban), és azokra a rekordokra (sorokra), amikben egyezést talált a megadott mintával, végrehajtja a megadott utasításokat. Azokra a rekordokra (sorokra), amikben az adott minta nem található meg, nem hajt végre műveletet. A feldolgozás során végighalad az adatfájl minden rekordján (során), majd befejezi a működését. A feldolgozás során számokkal, karakterekkel, szövegrészekkel végezhetünk műveleteket, beleértve matematikai és logikai kifejezések használatát is.

Az *awk* program alapvetően minta–tevékenység utasítások sorozatából áll:

```
Minta { tevékenység utasítások }
```

Az *awk* sor orientált nyelv. Előbb jön a minta, majd a tevékenység. A `Minta` egy jelsorozat, ami meghatározza, hogy mely rekordokra legyenek végrehajtva az `tevékenység utasítások` részben megadott parancsok. A tevékenység-utasítások `{` és `}` közé vannak zárva. Vagy a minta, vagy a tevékenység elmaradhat, de természetesen mindkettő nem. Ha a minta hiányzik, a tevékenység minden egyes bemenő rekordon végrehajtódik. A hiányzó tevékenység ugyanaz, mint a

```
{ print }
```

utasítás, amely kiírja az egész rekordot.

4.3.1. Minták

Az *awk* minták a 4.1. táblázatban felsoroltak lehetnek. A `BEGIN` és az `END` két speciális minta, amely nem inputfüggő. A `BEGIN` blokkok azelőtt kerülnek végrehajtásra, mielőtt bármilyen bemenet beolvasása megtörténne. Ehhez hasonlóan az `END` blokkok akkor hajtódnak végre, amikor minden bemenet beolvasása véget ért (vagy amikor exit utasítás hajtódik végre). A `BEGIN` és az `END` blokkok nem kombinálhatók más mintákkal a mintakifejezésekben. A `BEGIN` és az `END` minták után nem hiányozhat a tevékenység rész.

A */reguláris kifejezés/* mintákhoz rendelt utasítás minden olyan rekordra végrehajtódik, amely illeszkedik a reguláris kifejezésre. A reguláris kifejezések összefoglalását lásd a 4.5. szakaszban.

4.1. táblázat. A *gawk* minták

`BEGIN`
`END`
/reguláris kifejezés/
relációs kifejezés
$minta_1$ `&&` $minta_2$
$minta_1$ `||` $minta_2$
$minta_1$ `?` $minta_2$ `:` $minta_3$
`(`*minta*`)`
`!` *minta*
$minta_1$`,` $minta_2$

A *relációs kifejezésekben* szerepelhet az alábbi, tevékenységekről szóló részben definiált operátorok közül bármelyi. Ezek rendszerint azt ellenőrzik, hogy bizonyos mezők illeszkednek-e bizonyos reguláris kifejezésekre.

Az `&&`, `||`, és `!` operátorok rendre a logikai ÉS, logikai VAGY és logikai NEM (mint a C programnyelvben) jelölésére szolgálnak. Csak addig értékelődnek ki, ameddig egyértelműen eldől, hogy logikai igaz vagy hamis az értékük. Egyszerűbb mintakifejezések összekapcsolására szolgálnak. Mint a legtöbb nyelvben, zárójelek használhatók a kiértékelés sorrendjének megváltoztatására.

A `?:` operátor hasonlít a C ugyanezen operátorához. Ha a $minta_1$ igaz, akkor $minta_2$ kerül felhasználásra az ellenőrzéshez, egyébként a $minta_3$. A $minta_2$ és $minta_3$ közül csak az egyik értékelődik ki.

A $minta_1$`,` $minta_2$ kifejezésforma neve tartományminta. Ez illeszkedik a $minta_1$-re illeszkedő rekorddal kezdve folytatólagosan minden bemenő rekordra, egy, a $minta_2$-re illeszkedő rekordig – beleértve a két határmintára illeszkedő rekordokat is. Nem működik együtt másfajta mintakifejezésekkel.

4.3.2. Változók, rekordok és mezők

Az *gawk* változók dinamikusan, első használatukkor jönnek létre. Értékük lehet lebegőpontos szám, karakterlánc, vagy mindkettő, a használatuk módjától függően. Az *awk*-ban egydimenziós tömböket is létre lehet hozni. A tömbök asszociativak, azaz kulcs–érték párokból állnak, ahol a kulcsok egyediek. A kulcsok tetszőlegesek lehetnek, ezért a többdimenziós tömbök könnyen szimulálhatók.

Rekordok

Az *gawk* az adatfájlt rekordonként olvassa be, ahol a rekordokat az úgynevezett *rekordelválasztó* választja el egymástól. Ez alapesetben az **newline** karakter, de a rekordelválasztás módja az `RS` nevű beépített változó értékének megváltoztatásával módosítható. Ha az *RS* változó értéke egyetlen karakter, akkor ez a karakter lesz a rekordelválasztó, egyébként *RS* reguláris kifejezésként értelmezi a program. Ebben az esetben minden, erre a reguláris kifejezésre illeszkedő szöveg rekordelválasztó lesz. Ha `RS` üres karakterláncra van beállítva, akkor a rekordokat üres sorok választják el. Ilyenkor az **newline** karakter mezőelválasztóként viselkedik, bármilyen értékű is FS.

Mezők

A *gawk* a beolvasott rekordokat úgynevezett *mezőkre* bontja a *mezőelválasztók* mentén, melyet a `FS` változó értékeként lehet megadni. Ha `FS` egyetlen karakter, akkor ez a karakter választja el a mezőket. Ha `FS` üres karakterlánc (`FS=""`), akkor minden egyes karakter külön mező lesz. Egyébként a program `FS`-t, mint reguláris kifejezést kezeli. Abban a speciális esetben, ha `FS` egyetlen **space**, akkor a mezőket nemcsak a **space**, hanem a **tab** és/vagy az **newline** karakterek is elválasztják.

A mezők értékeire a `$1`, `$2`, stb. kifejezésekkel lehet hivatkozni. Ezek szintén változók, melyeknek akár új értéket is adhatunk. A `$0` mező az egész rekordot (sort) jelenti.

A mezőhivatkozásnak nem szükséges konstansnak lennie:

```
n = 5
print $n
```

kiírja a bemeneti rekord ötödik mezőjét. Az `NF` változó a bemeneti rekord mezőinek számát tartalmazza, tehát tetszőleges rekord utolsó mezőjére a `$NF` kifejezéssel hivatkozhatunk.

Beépített változók

A *gawk* legfontosabb beépített változói:

`RS` A bemeneti rekordelválasztó *(Record Separator)*. Ez a változó írja elő az *awk* programnak, hogy a bemenetként feldolgozandó állományban milyen karakterelemek válasszák el a rekordokat. A feldolgozás közben átállítható, a változást a *gawk* figyelembe veszi. `RS` alapértelmezésben az **newline** karakter.

`FS` Feladata hasonló, mint a rekordelválasztó változónak. Az aktuálisan beolvasott rekord mezőelválasztó *(Field Separator)* elemét jelöli ki. Az alapértelmezett értéke a **space**.

`NR` Ez a változó a rekord beolvasása után frissül, és a már beolvasott rekordok számát *(Number of Records)* mutatja.

`NF` Dinamikusan frissülő változó, mely a a mezők számát *(Number of Fields)* mutatja az adott rekordban. Értékét minden rekord beolvasása után automatikusan kapja.

`IGNORECASE` A bemenet feldolgozásához használt kapcsoló, ha az értéke nem nulla, akkor a kis és nagybetűket a *gawk* nem különbözteti meg.

4.4. A *gawk* legegyszerűbb utasításai

A *gawk* program számos összetett funkciót ellátó paranccsal rendelkezik, de most egyelőre csak a legegyszerűbbekkel ismerkedünk meg. A `{Utasítások}` szakaszban több *gawk* utasítás is szerepelhet, azonban több utasítás esetén azokat egymástól `;`-vel el kell választani.

- `print` – kiíratás, a parancs után vesszővel felsorolt változókat és konstansokat kiírja a kimenetre. Pl. `print $2,$1` kiírja az adatfájl minden sorára az adott sor második és első elemét, tehát az összes adatsorra lefutva megkapjuk az adatfájl második és első oszlopának adatait felcserélve.

- `printf()` – kiíratás `C` szintaxissal. A `printf()` parancs a `C` nyelv szintaxisát követi, azaz a zárójelek között idézőjelek közé helyezzük a kiírni szánt mennyiségek formázását, majd az idézőjel bezárása után vesszővel elválasztva megadjuk a kiírni szánt mennyiségeket. Pl. `printf("%d %.2f\n",$1,$2)` kiírja az első oszlop elemeit (`$1`) egész értékekként (`%d`), majd egy szóköz után a második oszlop elemeit (`$2`) két tizedesjegyet ábrázolva lebegőpontos számként (`%.2f`).

- matematikai operátorok: `+,-,*,/,^,%` – az összeadás, kivonás, szorzás, osztás, hatványozás, modulo szimbólumai. Az `a=$2*2-$1` utasítás például az `a` változóhoz hozzárendeli az adott rekord második mezőjének kétszeresét, levonva belőle az első oszlop értékét. A *gawk* ismeri a `C` nyelvben gyakran használt jelöléseket is: `++,--,+=,-=,/=,*=`. Ezek közül a `++` eggyel növeli, `--` pedig eggyel csökkenti a változó értékét. A `+=` utasítással a `a=a+b` alakú értékadásokat rövidíthetjük `a+=b` alakúra. A többi értékadó utasítás ennek analógja.

4.5. Reguláris kifejezések

Szöveges adatfájlok feldolgozásánál, szövegszerkesztésnél, dokumentáció olvasása közben és számos más helyzetben szükséges lehet bizonyos szöveges minták keresésre. A reguláris kifejezés egy minta, amely a karakterláncoknak (stringeknek) egy halmazát írja

le. A reguláris kifejezések az aritmetikai kifejezésekhez hasonlóan konstruálhatók, azaz különböző operátorok segítségével egyszerűbb kifejezésekből építhetők fel.

A reguláris kifejezésekben előforduló operátoroknak ugyanúgy definiált a műveleti sorrendje, mint mondjuk a számtanban az összeadásnak és a szorzásnak. A műveleti sorrendet zárójelekkel tudjuk megváltoztatni, hasonlóan a közönséges összeadáshoz és szorzáshoz.

A reguláris kifejezéseknek kétféle nyelvtana (szintaxisa) terjedt el: az „alap” (basic) és a „bővített” (extended) változat. Az alap reguláris kifejezések általában kevésbé hatékonyak. A következő leírás a bővített reguláris kifejezésekre vonatkozik; az alap reguláris kifejezéseknek ettől való különbségeit ezután összegezzük.

4.5.1. Az reguláris kifejezések elemi építőkövei

A reguláris kifejezések elemi építőkövei (atomjai) a következők lehetnek:

- `(...)` zárójelbe zárt reguláris kifejezés,
- üres `()` zárójel,
- `[...]` szögletes zárójel-kifejezés,
- pont „`.`”,
- kalapjel „`^`”,
- dollárjel „`$`”,
- backslash „\”, melyet a „`^.[$()|*+?{`\” speciális karakterek valamelyike követ,
- backslash, melyet valamely tetszőleges más karakter követ, vagy
- egy egyszerű önálló karakter.

Az egyes atomok jelentését a 4.2. táblázatban foglaltuk össze.

Normál karakterek

A legtöbb karakter (pl. a betűk és a számok) olyan reguláris kifejezés, amely önmagához illeszkedik. Vannak azonban speciális jelentéssel rendelkező metakarakterek („`^.[$()|*+?{`\”), melyek elé backslash-t (\) kell írni, hogy a speciális jelentésük helyett az adott karakterhez való illeszkedést vizsgálhassuk. (Pl. a mintában levő `\[` fog illeszkedni a szövegbeli nyitó szögletes zárójelre.) Másrészről, ha normál karakterek elé írunk backslash-t, akkor azok speciális jelentést vehetnek fel. (Pl. a mintában lévő `\t` fog illeszkedni a szövegben lévő tabulátor karakterre.)

Horgonyok

A `^` (kalap) és a `$` (dollár) jelek metakarakterek, melyek rendre a sor elejére és végére illeszkednek. Hasonlóan, a `\<` és `\>` szimbólumok a szavak elejére illetve végére illeszkednek.

Helyettesítő karakter

A . (pont) jel egy olyan metakarakter, amely a sorvégén kívül minden karakterre illeszkedik. Ez amolyan Jolly Joker, de vigyázzunk a használatával, mert sokkal több dologra illeszkedhet, mint amire gondolunk.

Zárójel-listák

A `[` és `]` jelek közé írt karakterlista illeszkedik a listában szereplő bármely karakterhez. Amennyiben a lista a `^` jellel kezdődik, az illeszkedés a listában *nem* szereplő karakterekre áll fenn. Például a `[0123456789]` reguláris kifejezés bármely számjegyhez illeszkedik, míg a `[^0123456789]` reguláris kifejezés bármire illeszkedik, ami *nem* számjegy.

ASCII karakterek tartománya az első és utolsó karakterrel adható meg, ha ezeket '-' jel választja el. (Pl. `[a-f]` ugyanaz, mint az `[abcdef]`, vagy a korábbi példában említett a számjegyeket a `[0-9]` reguláris kifejezéssel adhatjuk meg.)

A karakterek néhány speciális osztálya előre definiált névvel rendelkezik. (Ezen nevek többségének jelentése angolul magától értetődő, itt kifejtjük őket.)

`[:alnum:]`	betű vagy szám
`[:alpha:]`	betű
`[:cntrl:]`	vezérlőkarakter
`[:digit:]`	számjegy
`[:graph:]`	grafikus karakter
`[:lower:]`	kisbetű
`[:print:]`	nyomtatható karakter
`[:punct:]`	elválasztó, központozó jel (.,;?!)
`[:space:]`	szóköz
`[:upper:]`	nagybetű
`[:xdigit:]`	hexadecimális szám

Például `[[:alnum:]]` jelentése: `[0-9A-Za-z]`. Ezen osztálynevekben szereplő szögletes zárójelek a szimbolikus nevek részei, és a zárójeles listákat határoló zárójelek mellett ezeket is meg kell adni.

A legtöbb metakarakter elveszti speciális jelentését egy listán belül. Egy `]` jelet a minta első elemeként szerepeltetve csatolhatunk a listához. Hasonlóan, a `^` jel az első hely kivételével bármely helyre kerülve a `^` jelet fogja jelenteni. Végül a '-' jelet a

4.2. táblázat. Reguláris kifejezésekben szereplő leggyakoribb atomok.

Atomok *(atom)*

`(...)`	A zárójelben álló reguláris kifejezésre illeszkedik.
`()`	Az üres string-re illeszkedik.
`[...]`	A `[` és `]` közötti listában szereplő karakterekre illeszkedik.
`[^...]`	A `[` és `]` közötti listában *nem* szereplő karakterekre illeszkedik.
`.`	Minden karakterre illeszkedik (Használjuk óvatosan!).
`^`	A sor elejére illeszkedik.
`$`	A sor végére illeszkedik.
`\`■	A ■ jelölt speciális karakter mint normál karakter illeszkedik, lehetséges elemek: `\^`, `\.`, `\[`, `\$`, `\(`, `\)`, `|`, `\*`, `\+`, `\?`, `\{`, `\\`
■	Speciális jelentés nélküli karakterre illeszkedik. Lehetséges elemek nem teljes listája: `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`, `!`, `@`, `#`, `%`

Darabok *(piece)*

atom`*`	Az *atom* tetszőleges (0 vagy több) alkalommal illeszkedik.
atom`?`	Az *atom* legfeljebb egy (0 vagy 1) alkalommal illeszkedik.
atom`+`	Az *atom* legalább egy (1 vagy több) alkalommal illeszkedik.
atom`{n}`	Az *atom* pontosan `n` alkalommal illeszkedik.
atom`{n,}`	Az *atom* `n` vagy több alkalommal illeszkedik.
atom`{,m}`	Az *atom* legfeljebb `m` alkalommal illeszkedik.
atom`{n,m}`	Az *atom* legalább `n` de legfeljebb `m` alkalommal illeszkedik.

Ágak *(branch)*

$piece_1 piece_2 \ldots$	Egy vagy több egymás utáni *piece*-re illeszkedik.

Reguláris kifejezések

$branch_1 \mid branch_2 \mid \ldots$	Bármire illeszkedik, amire legalább az egyik *branch* illeszkedik.

lista utolsó elemeként kell írni, ha nem (karaktertartományt megadó) metakarakterként akarjuk értelmezni.

4.5.2. Ismétlő operátorok

Az elemi építőköveket a következő ismétlési operátorok egyike követheti:

`?`	Az előző tag opcionális, és legfeljebb egyszer illeszkedik.
`*`	Az előző tag nulla vagy több alkalommal illeszkedik.
`+`	Az előző tag egy vagy több alkalommal illeszkedik.
`{n}`	Az előző tag pontosan `n` alkalommal illeszkedik.
`{n,}`	Az előző tag `n` vagy több alkalommal illeszkedik.
`{,m}`	Az előző tag legfeljebb `m` alkalommal illeszkedik.
`{n,m}`	Az előző tag legalább `n` de legfeljebb `m` alkalommal illeszkedik.

4.5.3. Összetett reguláris kifejezések

Az atomokat és az azokat követő ismétlési operátorokat együtt daraboknak *(pieces)* nevezzük. Az egyes darabok összefűzhetőek. Az összefűzött darabokat ágnak *(branch)* nevezzük. Az ág minden olyan karakterlánchoz illeszkedik, amely az ágat alkotó darabokhoz illeszkednek a sorrend megtartása mellett (logikai ÉS, de nem felcserélhető). Például az `ab` egy ág, amely az `a` és `b` atomokból áll, és illeszkedik az „ablak” szó első két karakterére, vagy a „baba” szó középső két betűjére.

Két ág összekapcsolható a `|` infix operátorral; a kapott reguláris kifejezés minden karakterlánchoz illeszkedik, amelyikhez valamelyik ág illeszkedik (logikai VAGY).

Az ismétlés („hatványozás”) nagyobb precedenciájú, mint az összefűzés („szorzás”), ami viszont a választóoperátornál (azaz `|`-nál) nagyobb precedenciájú („összeadás”). Egy részkifejezés zárójelbe tehető, hogy felülbíráljuk a precedenciát. Például a „baba” szónak csak az első két karakterére illeszkedik a `ba*` kifejezés, de az egész szóra illeszkedik a `(ba)*`.

4.5.4. Visszahivatkozás

A zárójeles kifejezések nemcsak az operátorok sorrendiségét szabályozzák, hanem a hivatkozhatunk is a zárójeles reguláris kifejezéssel illeszkedő mintára. A visszahivatkozást *(backreference)* $\backslash n$ jelöli, ahol n egy számjegy. Ez illeszkedik ahhoz a karakterlánchoz, amely a reguláris kifejezés ezt megelőző n-edik zárójeles alkifejezéshez illeszkedett. Például a `(.a)\1` kifejezés illeszkedik az „papa”, „baba”, stb. párokra.

4.5.5. Alap reguláris kifejezések

Az alap reguláris kifejezésekben a `?`, `+`, `{`, `|`, `(`, és `)` metakarakterek elvesztik speciális jelentésüket; helyettük a backslash-es változatukat kell használni: `\?`, `\+`, `\{`, `\|`, `\(`, és `\)`.

4.6. Reguláris kifejezések a *vim*-ben

A reguláris kifejezések nemcsak az *awk* nyelvben használhatóak, hanem számos más programban. A *vim*-ben például elsősorban kereséshez illetve cseréhez használhatjuk. A *vim*ben alapesetben az alap reguláris kifejezéseket használhatjuk, azaz a `*`-on kívül minden metakarakter elé backslash-t kell tenni. A *vim* reguláris kifejezéseiről a `:h regex` paranccsal kérhetünk bővebb információt.

4.6.1. Keresés

A keresést Normal módból a `/` illetve `?` billentyűkkel indíthatunk. Ekkor a legalsó parancssorban megjelenik a kurzor és megadhatjuk azt a reguláris kifejezést, amire keresni szeretnénk. Ha a következő találatra szeretnénk ugrani, akkor az `n`-nel a keresés irányában ugorhatunk a következő találatra, míg az `N` billentyűvel a keresés indításával ellentétes irányban ugorhatunk.

4.6.2. Csere

Szövegrészek cseréjére a `:s[substitute]` parancs szolgál (az angol *substitute*-ből), ahol a szögletes zárójel azt jelenti, hogy a benne lévő részt elhagyhatjuk. A parancs formája a következő:

:[*tartomány*]s[ubstitute]/*minta*/*string*/[*kapcsolók*] [*darab*]

A szögletes zárójeles részek megadása itt sem kötelező. A parancs a megadott *tartomány* minden sorában kicseréli a *mintá*-ra illeszkedő részt a megadott (akár üres) *string*-re. Ha a *tartomány*-ra nem adunk meg semmit, akkor csak a kurzor aktuális sorára végzi el a cserét. Ha az egész szövegre el akarjuk végezni a helyettesítést, akkor a *range*-nek adjuk meg a `%` karaktert.

A legfontosabb *kapcsolók* a következők:

- `c` Minden csere előtt megerősítést vár.
- `g` Az adott sorban minden előfordulásra elvégzi a cserét. Ennek a kapcsolónak a hiányában csak az első előfordulásra végzi el a cserét.

```
# Számoljuk ki a mellékelt adatfájlban a második adatoszlop
# átlagát,  azokra a sorokra, amelyek nem #-jellel kezdődnek,
# és az első oszlopban található szám nagyobb, mint négy.
1       4
2       6
3       8
4       10
5       12
6       14
7       16
8       18
9       20
10      22
```

4.1. ábra. Adatfájl a Gy4.1. feladathoz.

4.7. Példák és feladatok

Gyakorló példák

Gy4.1. Számoljuk ki *gawk*kal a 4.1. ábrán látható adatfájlban a második adatoszlop átlagát, azokra a sorokra, amelyek nem #-jellel kezdődnek, és az első oszlopban található szám nagyobb, mint négy.

Gy4.2. Töröljük ki *gawk*kal a 4.2. ábrán mellékelt szövegben azoknak a sorokat az utolsó szavát, amelyek #-jellel kezdődnek, de nem tartalmaznak $-jelet.

Gy4.3. Keressük meg *vim*mel és *gawk*kal a 4.3. ábrán mellékelt szövegben azokat a szavakat, amelyekben szerepel nagybetű, de nem az első helyen.

Gy4.4. Helyettesítsünk be *vim*mel a mellékelt a 4.4. ábrán található szövegben minden négyjegyű szám után egy–egy pontot.

Feladatok

F4.1. Írjunk *gawk* szkriptet, amely kiszámolja a mellékelt szövegben a 7. sortól kezdve a verssorokban található szavak számának átlagát, és az eredményt kiírja a terminálra.

F4.2. Számoljuk össze *gawk*kal a mellékelt szövegben azoknak a szavaknak a számát, amelyekben szerepel az „e” magánhangzó, illetve azoknak a szavaknak a számát, amelyben *csak* az „e” magánhangzó szerepel, más magánhangzó nem. A két eredményt írassuk ki a terminálra.

F4.3. Keressük meg *vim*mel a mellékelt szövegben a „()” zárójeles részeket. Az egymásba ágyazott zárójelek közül csak a legbelsőt vegyük figyelembe, de figyeljünk arra, hogy a zárójelek párosítása helyes legyen.

F4.4. A mellékelt fájl második oszlopában az első oszlop gyakoriság-eloszlása található. Az első oszlop értékei monoton növekvők. Készítsünk egy *gawk* szkriptet, amely a gyakoriság-eloszlásból kumulatív eloszlást készít, azaz az első oszlop minden értékhez kiírja a korábbi gyakoriságok összegét. A kumulatív gyakoriságok kerüljenek a harmadik oszlopba, és az eredményt mentsük el egy új adatfájlba. Ábrázoljuk *gnuplot*tal egy ábrán az eloszlást és a kumulatív eloszlást is. (Útmutatás: vegyük észre, hogy nem kell minden i-ik sorra a teljes összegzést elvégeznünk, hanem elegendő csupán az előző $c(i)$ kumulatív értékhez hozzáadnunk a következő $p(i+1)$ gyakoriság-értéket: $c(i+1) = c(i) + p(i+1)$.)

F4.5. Keressük meg *vim*mel a mellékelt LaTeX szövegben azokat az ismétlődő szavakat (pl. „... és és...”), amelyek nem megjegyzésként szerepelnek (tehát nem `%` jel után). Cseréljük ki a kettőzött vagy többszörözött szavakat egyszeri előfordulásra és a kijavított szöveget fordítsuk le LaTeX-hel.

F4.6. Helyettesítsünk be *gawk*kal egy tetszőleges szövegben a szövegben található számok után egy–egy pontot, feltéve, hogy az adott szám után a helyettesítés előtt még nem volt pont.

F4.7. Írjunk egy *gawk* szkriptet, amely kiszámolja egy tetszőleges szövegben a magánhangzók és mássalhangzók számának arányát.

F4.8. Adjuk meg, hogyan kereshetjük meg *vim*mel egy tetszőleges szövegben szereplő LaTeX parancsokat a hozzájuk tartozó opcionális és kötelező argumentumokkal együtt.

```
     A dollár története

Az amerikai dollár több, mint 200 éves múltra tekint vissza.
Az első dollárérméket az United States Mint (amerikai
pénzverde) bocsátotta ki.  Ezeknek az érméknek a súlya és
összetétele megegyezett a spanyol dollaréval. Az Amerikai
függetlenségi háború után egyaránt volt forgalomban spanyol
és amerikai ezüstdollár is.

Érmék

# 1 cent - Abraham Lincoln; EZEKET
# 5 cent (nickel) - Thomas Jefferson; A
# 10 cent (dime) - Franklin D. Roosevelt; SZAVAKAT
# 25 cent (Quarter) - George Washington; KELL
# 50 cent (fél dollár) - John F. Kennedy; GAWKKAL
# Egy dollár - Amerikai elnökök; KITÖRÖLNI

Bankjegyek

# 1 $ - George Washington
# 2 $ - Thomas Jefferson
# 5 $ - Abraham Lincoln
# 10 $ - Alexander Hamilton
# 20 $ - Andrew Jackson
# 50 $ - Ulysses S. Grant
# 100 $ - Benjamin Franklin
# 500 $ - William McKinley
# 1000 $ - Grover Cleveland
# 5000 $ - James Madison
# 10000 $ - Salmon P. Chase

     -- Forrás: wikipedia.hu
```

4.2. ábra. Adatfájl a Gy4.2. feladathoz.

```
                    Rövidítések a Hálón


Mindenkivel előfordult már, hogy levelezés vagy IRC-zés közben
különböző, számára nem teljesen érthető jeleket, betűhalmazokat
olvasott. Összegyűjtöttem pár, szinte minden nap előforduló
rövidítést, és azok jelentését. Íme.

Rövidítés      Jelentése              Magyar fordítás
IMHO     In My Humble Opinion         Szerény véleményem szerint
TIA      Thanks In Advance            Előre is köszönöm
ASAP     As Soon As Possible          Amint lehetséges
AFAIK    As Far As I Know             Tudtommal
BTW      By The Way                   Erről jut eszembe
FYI      For Your Information         Csak hogy tudd
IMO      In My Opinion                IMHO csak szerénytelenül
JAM      Just A Minute                Egy pillanat
LOL      Laughing Out Loud            Hangosan nevetni
ROTLF    Rolls On The Floor Laughing  A földön fetreng a nevetéstől
THX      Thanx                        Köszi
TTYL     Talk To You Later            Később megbeszéljük
WYSIWYG What You See Is What You Get Azt kapod amit látsz

  -- Forrás: http://www.csatolna.hu/hu/erdekes/Atti/rovid.html
```

4.3. ábra. Adatfájl a Gy4.3. feladathoz.

```
    Egy dátum kiszámolása a XXI. században

Számoljuk ki, hogy milyen napra esik 2073 július 14.  Először
kiszámoljuk, hogy mikorra esik 2073 január 1.  (újév). Ez egy
egyszerű trükk: vesszük a 73 (2073 utolsó 2 számjegyét) 25%át (1
negyedét).  (73/4=18,25) és levesszük a 25 századot mert csak a
szám egész része kell. Hozzáadjuk ezt a 73-hoz: 73+18=91. És
91-ból kivonjuk a 7 legnagyobb többszörösét ami még nem ad
negatívat szóval a 91-7x13=0.  Itt most ez 0=vasárnap. Tehát:
2073 január 1. az vasárnap.  Ha az eredmény 1=hétfő 2=kedd
3=szerda 4=csütörtök 5=péntek 6=szombat 7=vasárnap 0=vasárnap.
Most az eredményhez hozzáadunk 6-ot mivel a július hónapkódja:6.
Tehát: VASÁRNAP+6=SZOMBAT Tehát 2073 július 1 az SZOMBAT.  És ha
1 szombat akkor 14-e péntek. Tehát 2073 július 14. péntekre fog
esni.

 -- Forrás: http://fejszamolas.trukkok.hu
```

4.4. ábra. Adatfájl a Gy4.4. feladathoz.

5. fejezet

Komplex feladatok

Ezen a gyakorlaton a korábbi gyakorlatok anyagát ismételjük át, kiegészítve néhány hasznos funkcióval.

5.1. Programok párhuzamos indítása

A korábban bemutatott programok mindegyikét egy különálló terminálablakból értük el. Gyakran szükségünk lehet arra, hogy több program egyszerre fusson, például ha egy LaTeX dokumentum szerkesztése mellett szeretnénk a lefordított `.dvi` állományt is folyamatosan ellenőrizni. Eddig azonban egyszerre csupán egy programot tudtunk elindítani a terminál parancssorából, mivel a program futásának végéig a terminál nem adta vissza a promptot, ahova a következő parancsot be tudtuk volna vinni.

Egy nyilvánvaló megoldás lehet, ha annyi terminálablakot indítunk, amennyi programot futtatni kívánunk. Ennek a módszernek nyilvánvaló hátránya, hogy rövid időn belül rengeteg terminálablak lesz megnyitva. Érdemes azonban megjegyeznünk, hogy a Linux `gnome-terminal`[1] vagy `konsole` terminálkezelői több terminál együttes kezelését is támogatják, melyek fülei között kényelmesen tudunk váltani. A következő módszerrel lehetőség van arra is, hogy a terminálablakban egy parancs kiadása után azonnal visszakapjuk a promptot. Ehhez a programokat a „háttérben" (*background*) kell futtatnunk. Ezt úgy érhetjük el legkönnyebben, ha a parancsok végére egy `&`-jelet írunk, például így:

```
user@host:~$ xdvi valami.dvi &
```

Figyeljünk arra, hogy csak olyan programokat indítsunk háttérben, amelyek nem várnak a terminálról semmilyen bemenetet, ellenkező esetben lehetséges, hogy a programunk a háttérben folyamatosan várakozik. Így például a LaTeX parancsot mindig az „előtérben" (*foreground*) indítsuk, hiszen a fordító környezet a futás közben észlelt hibák feloldására gyakran felhasználói választ vár a standard bemenetről.

[1] Új parancsértelmező `<Ctrl-Shift-t>` gombnyomással lehetséges

Egy háttérben futó programot az `fg` paranccsal hozhatunk az előtérbe. Fordított esetben, ha egy előtérben futó programot kívánunk a háttérbe vinni, akkor először üssük le a `<Ctrl-Z>` billentyűket, majd adjuk ki a `bg` parancsot. Egy parancsértelmező számos háttérben futó szál kezelésére képes, ezek nyomon követését segíti a `jobs` parancs, amelynek hatására a futó *background* parancsok listáját kérdezhetjük le. A válaszlista első eleme egy azonosító, pl. `[1]+`. Az azonosító segít kiválasztani a megfelelő szálat, melyre pl. a `%1` mintával hivatkozhatunk. Ennek értelmében a

```
user@host:~$ kill %4
user@host:~$ fg %1
```

parancsok rendre megszakítják a 4. háttérszálat és az elsőt előtérbe hozzák.

5.2. Több állomány szerkesztése *vim*mel

Természetesen a *vim*ben is lehetőség van több dokumentum párhuzamos szerkesztésére. A memóriában lévő dokumentumokat puffernek nevezik, míg a puffernek a képernyőn megjelenített részét ablaknak. Különböző ablakokban megjeleníthetjük egyetlen puffer különböző részeit vagy különböző puffereket is.

Az ablakok megnyitására és bezárása szolgáló legfontosabb parancsokat a 5.1. táblázatban foglaltuk össze. Az ablakok között a `hjkl` kurzormozgató billentyűkhöz analóg módon válthatunk, ha előbb lenyomjuk a `<Ctrl-w>` billentyűt. Ha az ablakokat kívánjuk átrendezni, akkor a `<Ctrl-w>` billentyű után a kurzormozgató parancsok nagybetűs változatát használhatjuk, azaz a `HJKL` billentyűket. A vonatkozó parancsokat a 5.2. táblázatban foglaltuk össze.

5.3. Külső parancsok futtatása *vim*ből

Gyakran előfordul, hogy mikőzben *vim*ben dolgozunk, valamely más programot is el szeretnénk indítani. Tipikus példa, hogy miközben egy LaTeX dokumentumot szerkesztünk *vim*mel, azt le szeretnénk fordítani, hogy megnézzük a lefordított változatot. Pontosan erre szolgál a `:!` parancs. A *vim* a `:!` parancs után beírt sort átadja a parancsértelmezőnek, és az eredményt addig megmutatja, ameddig az `<Enter>` billentyűt le nem nyomjuk.

Egy másik rendkívül hasznos funkciója a *vim*nek, hogy külső parancsokat szűrőként is alkalmazhatunk. Ehhez Visual módban jelöljük ki a szöveget, majd adjuk ki a `:!` parancsot, utána az alkalmazandó parancsot.

5.4. Külső parancsok futtatása *gnuplot*ból

Külső parancsokat *gnuplot*ból hasonló módon lehet elindítani, mint *vim*ből. Ehhez a *gnuplot* prompt elejére írjunk `!` jelet, majd a lefuttadandó parancsot. Tegyük fel például, hogy létrehoztunk egy `abra.eps` fájlt, és szeretnénk az eredményt megnézni. Ehhez nem kell kilépni a *gnuplot*ból, elég a következő parancsot kiadni:

```
gnuplot> !evince abra.eps&
```

Mivel a parancs végére kitettük a `&` jelet, ezért ennek hatására az `evince` program a háttérben fog elindulni, így folytathatjuk a munkát a *gnuplot*ban, nem kell ehhez az `evince` programból kilépni.

Külső programokat szürőként *gnuplot*ban is használhatunk. Ehhez a fájlnév elejére tegyünk egy `<` jelet. Ha például az első oszlop szerint rendezve szeretnénk ábrázolni az `adatok.dat` adatfájlt, adjuk ki a következő parancsot:

```
gnuplot> p "<sort -n adatok.dat"
```

Ennek hatására a *gnuplot* a shell-ben lefuttatja a `<` jel utáni parancssort, majd annak a kimenetét használja fel. A `sort` paracsról bővebben a 10.3 szakaszban lesz szó.

5.5. Több adatsor ábrázolása *gnuplot*ban

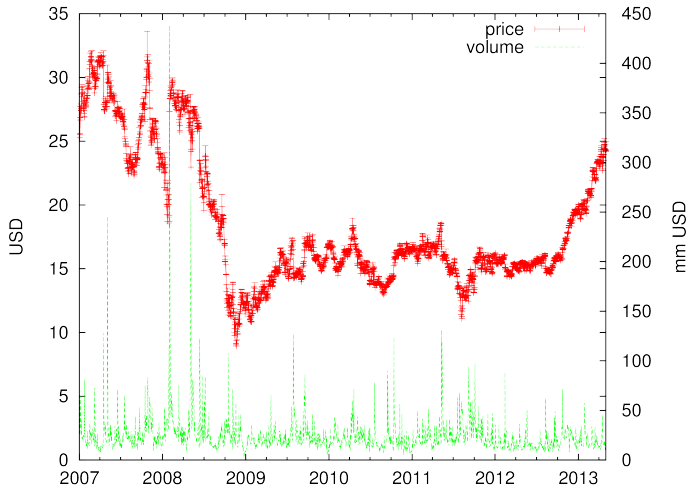
5.5.1. Másodlagos tengelyek

A 3. fejezetben bemutattuk, hogy hogyan lehet *gnuplot*ban az x és y tengelyeken feliratokat elhelyezni, valamint az ábrázolási tartományokat beállítani. Az x tengely az ábra alján, az y tengely pedig az ábra bal oldalán található.

A *gnuplot*ban lehetőségünk van arra is, hogy másodlagos tengelyeket használjuk. Ezek a másodlagos tengelyek az elsődleges tengelyekkel szemközt találhatóak, tehát az `x2` tengely az ábra tetején, az `y2` tengely pedig az ábra jobb oldalán. Ezek a tengelyek különösen akkor hasznosak, ha ugyanazon az ábrán több mennyiséget szeretnénk ábrázolni, melyek különböző skálán vagy mértékegységben vannak megadva.

Az 5.1. ábrán egy példát mutatunk másodlagos tengelyek használatára. Az adatfájl oszlopaiban a kereskedési napok, a nyitóár, napi legmagasabb és legalacsonyabb ár, a záróár, valamint az osztalékkal korrigált árak vannak felsorolva.

A `plot` parancsban az `axes` kapcsoló után adhatjuk meg, hogy a *gnuplot* melyik tengelyeken ábrázolja az adatokat. A lehetséges értékek `x1y1`, `x1y2`, `x2y1`, `x2y2`, melyek értelemszerűen a bal–alsó, jobb–alsó, bal–felső, valamint a jobb–felső tengelypárokra hivatkoznak. Az alábbi példában a 2. oszlopban található napi nyitóárakat az alapértelmezett `x1y1` tengelyeken, a 6. oszlopban található forgalmi adatokat pedig a `x1y2` tengelyeken ábrázoljuk:



5.1. ábra. Példa idősor ábrázolására és másodlagos tengely használatára *gnuplot*ban. A bal oldali tengelyen a Yahoo NASDAQ részvényárfolyama látható 2007. január 1-től napjainkig, a jobb oldali tengelyen pedig a napi forgalma millió USD-ban. Az árfolyamot hibasávval ábrázoltuk, melynek maximuma és minimuma a napi maximum és minimum ár.

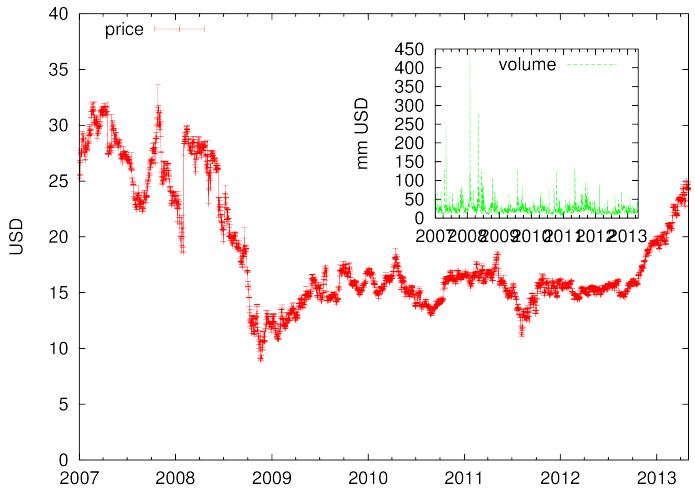
```
gnuplot>p "yahoo.dat" u 1:2 t "price", "" u 1:6 t "volume" w l axes x1y2
```

5.5.2. Multiplot

Az előző szakaszban bemutattuk, hogyan lehet a másodlagos tengelyek segítségével több adatot egyazon ábrán ábrázolni. Előfordulhat azonban, hogy ez a megoldás nem megfelelő. A `set multiplot` parancs segítségével lehetőségünk van arra is, hogy több `plot` parancs kimenetét, beleértve a tengelyeket, a különböző feliratokat, stb., ugyanazon az ábrán ábrázoljuk. Ezt főként akkor használjuk, ha egy nagyobb ábrán belül egy kisebb, ún. „inset” ábrát szeretnénk elhelyezni.

Az egyes `plot` parancsok kimenetének elhelyezését kétféleképpen szabályozhatjuk. Egyrészt a `set size` és `set origin` parancsokkal manuálisan beállíthatjuk, hogy a kirajzolt ábra mekkora legyen és, hogy az ábra hova kerüljön a vásznon. A méretet a vászon méretéhez viszonyítva kell megadni. A pozíciót a vászon koordinátarendszerében kell megadni, ahol `0, 0` koordináta a vászon bal alsó sarkát jelenti, a `1, 1` koordináta pedig a jobb felső sarkot. Ahhoz például, hogy egy felére kicsinyített ábra bal az eredeti ábra bal felső sarkába kerüljön, a

```
gnuplot>set size 1
gnuplot>set origin 0, 0
```



5.2. ábra. Példa `multiplot` használatára *gnuplot*-ban. A fő ábrán a Yahoo NASDAQ részvényárfolyama látható 2007. január 1-től napjainkig, a belső ábrán pedig a napi forgalma millió USD-ban. Az árfolyamot hibasávval ábrázoltuk, melynek maximuma és minimuma a napi maximum és minimum ár.

```
gnuplot> set multiplot
multiplot> plot ...
multiplot> set size 0.5
multiplot> set origin 0, 0.5
multiplot> plot ...
multiplot> unset multiplot
```

parancsokat kell kiadni. Figyeljük meg, hogy a `set multiplot` parancs kiadása után megváltozik a prompt `multiplot>`-ra.

Az 5.2. ábrán egy példát láthatunk egy kicsinyített ábra manuális elhelyezésére. Amennyiben az inset ábra mögötti részt le akarjuk törölni, akkor a `plot` parancs előtt adjuk ki a `clear` parancsot is.

A `set multiplot` parancs `layout` kapcsolójának segítségével lehetőség van arra is, hogy a *gnuplot* automatikusan, táblázatszerűen helyezze el a kisebb ábráinkat.

```
gnuplot> set multiplot layout 2,3
multiplot> [itt most legfeljebb 2x3=6 plot parancs következik]
multiplot> unset multiplot
```

Figyeljünk arra, hogy a `multiplot` környezetben nem állíthatjuk át sem a kimenetet, sem pedig a terminált. Ezen kívül a `replot` paranccsal sem tudjuk az egész multiplot ábrát újrarajzoltatni, mivel a `replot` parancs csak az előző `plot` parancsot ismétli.

Ezért, ha az ábránkat egy fájlba szeretnénk menteni, akkor a `set multiplot` parancs kiadása *előtt* állítsuk be a kívánt kimenetet és terminált a `set output` és `set terminal` parancsokkal.

5.6. Idősorok ábrázolása

Az időt tartalmazó adatok formátuma rendkívül sokféle lehet. Ahhoz, hogy az idősorokat ábrázolni tudjuk, meg kell adnunk, hogy az időadatok hogyan vannak ábrázolva. *gnuplot*ban az időadatokat egy karakterlánccal megadott mintával írhatjuk le, amelyben a dátum és az idő különböző részei helyett helyettesítő karaktereket használunk. A helyettesítő karakterek a „%” jelből és egy betűből állnak. Például a „Jan 1, 2001” dátumra a `"%b %d, %Y"` minta illeszkedik. Ahhoz, hogy a *gnuplot* az x tengelyen a megadott formátumban adatokat ábrázoljon, ki kell adni a

```
gnuplot>set xdata time
gnuplot>set timefmt "%b %d, %Y"
```

parancsokat. Fontos megjegyezni, hogy ha a `set xdata time` paranccsal bekapcsoltuk az időformátumot, akkor a `plot` parancsban kötelezően meg kell adnunk a `using` kapcsolót. A normál adatformátumra a

```
gnuplot>set xdata
```

paranccsal térhetünk vissza.

Arra is lehetőség van, hogy az ábrán a beolvasottól eltérő formátumban feliratozzuk a tengelyeket. Ahhoz, hogy az x tengelyen csak az évszámok legyenek feltüntetve, adjuk ki a

```
gnuplot>set format x "%Y"
```

parancsot. A helyettesítő karakterek listája a az 5.3. táblázatban található.

5.7. Példák és feladatok

Gyakorló példák

Gy5.1. A mellékelt szövegben távolság, idő, és tömeg mennyiségek szerepelnek, pl. 3,5 km. A mennyiségek a következő részekből állnak: Egy szám, melyben szerepelhet legfeljebb egy tizedes vessző. A tizedesvessző után nulla vagy több számjegy állhat még. Ezt egy szóköz követi, majd legfeljebb egy előtag, (pl. *kiló-* (k), *milli-* (m), vagy *nano* (n)), és végül egy mértékegység (m, s, vagy g). Nyissuk meg a szöveget *vim*mel és keressük meg az összes mennyiséget, amelyre illik a fenti minta.

Gy5.2. A mellékelt adatfájl minden sorában az előző feladatban leírt mintára illeszkedő távolság, idő és tömeg mennyiségek találhatóak. Számoljuk össze *gawk*kal, hogy összesen hány méter, hány másodperc és hány kilogramm van az adatsorban, és az eredményt írassuk ki a terminálra. (Útmutatás: mivel az angolban nem tizedesvessző, hanem tizedespont van, ezért *vim*mel előbb cseréljük le a vesszőt pontra!)

Gy5.3. Ábrázoljuk *gnuplot*tal a $\sin(1/x)$ függvényt logaritmikus skálán a $[0.01, 10]$ intervallumon. Feliratozzuk az x és az y tengelyeket. Hasonlítsuk össze a program által kirajzolt görbét azzal, amit várnánk. Növeljük a függvényből vett minták (*sample*) számát tíz, majd százszorosára. Mit tapasztalunk? (Útmutatás: A logaritmikus skálát és a minták számát az értelmezési tartományhoz hasonlóan a `set` paranccsal állíthatjuk be (pl. `set log`). A parancsok használatához kérjünk segítséget a `help` paranccsal!)

Gy5.4. A mellékelt adatfájlban az első oszlop a dátumot, a második oszlop New York, a harmadik pedig Los Angeles napi hőmérsékletadatait tartalmazza. Vizsgáljuk meg *vim*mel az adatok formátumát, majd *gnuplot*tal ábrázoljuk a hőmérsékletadatokat 2000. jan. 1. és 2002. dec. 31. között úgy, hogy a bal oldali tengelyen Fahrenheit, a jobb oldalin pedig Celsius skála legyen[2]. A tengelyeket feliratozzuk, adjunk meg jelmagyarázatot, és az x tengelyen jelenítsük meg a dátumokat a magyar írásmód szerint. Azokon a napokon, amlyeken nem volt mérési adat, -99 van megadva. Ezeket *gawk* segítségével szűrjük ki.

Gy5.5. A mellékelt adatfájlban az előző feladathoz hasonlóan hőmérsékletadatok vannak, de most a harmadik oszlopban Los Angeles helyett Chicago hőmérsékleti adatai vannak. Ezen kívül a dátum formátuma is kissé más.

Vizsgáljuk meg az adatkat, majd írjunk egy *gawk* szkriptet, amely kiszámolja New York és Chicago napi átlaghőmérsékleteit. Az eredményt ábrázoljuk *gnuplot*tal. Az adatokat ábrázoljuk ponttal. Adjunk az ábrához egy inset ábrát is, amelyen a két város átlaghőmérsékleteinek különbségét ábrázoljuk.

Feladatok

F5.1. A mellékelt szövegben az első blokkban helyesen írt dátumok, a másodikban helytelen vagy helytelenül írt dátumok szerepelnek. Nyissuk meg a fájlt *vim*mel, és adjunk meg olyan reguláris kifejezést keresésnek, amely a helyes dátumokra illeszkedik, a helytelenekre pedig nem.

Tegyük fel, hogy a dátumokat a következőképpen kell írni:

- az évszám után pontot kell tenni,

[2] A hőmérsékletskálák között a $C = \frac{5}{9}(F - 32)$ képlettel tudunk váltani.

- a hónap első betűjét írhatjuk kis- vagy nagybetűvel,
- a hónapot három betűvel rövidíthetjük, mely után pontot kell tenni,
- a nap után szintén pontot kell tenni, és
- az egyes tagok közé legalább egy szóközt kell tenni.

F5.2. A mellékelt adatfájl elején különböző pénznemek közötti keresztárfolyamok vannak felsorolva `USD/CCY = ZZZ` formátumban, ahol `CCY = EUR, CHF`, vagy `JPY`, `ZZZ` pedig a keresztárfolyam, amely azt adja meg, hogy 1 `USD` mennyibe kerül az adott pénznemben.

A keresztárfolyamok utáni sorokban különböző összegek szerepelnek a fenti pénznemekben, pl. 100 EUR. A pénznemek jele előtt szerepelhet a *mm* vagy *bn* előtag, melyek 10^6 illetve 10^9 szorzófaktort jelentenek.

Írjunk egy *gawk* szkriptet, amely összeadja és kiírja a különböző pénznemekben megadott összegeket, valamint kiírja összegek összegét USD-ben is.

F5.3. Készítsünk LaTeX dokumentumot a mellékelt `.tex` kiterjesztésű fájlból. Az utasításoknak megfelelően másoljuk át a megfelelő helyre a `.tb1`, `.tb2`, és `.tb3` kiterjesztésű fájlokban található táblázatokat. Fordítsuk le a kész dokumentumot!

F5.4. A mellékelt adatfájlból készítsük el az 5.1. és az 5.2. ábrán látható grafikonokat. Az inset ábra legyen a `0,48, 0,48` pontba mozgatva, és `0,45`-szörös méretre kicsinyítve.

F5.5. 2x2-es multiplot ábrán ábrázoljuk a $\sin(x)$, $\cos(x)$,$\tan(x)$, $\sin(x)\cos(x)$ függvényeket a $[0, 2\pi]$ intervallumon. Minden görbét ábrázoljunk folytonos vonallal és más-más színnel. A tengelyekre írjuk fel az ábrázolt függvények neveit.

5.1. táblázat. Parancsok ablakok kezelésére. Az Ex parancsokat Normal módból mindig a kettőspont (:) billentyűvel kell kezdeni. A [] jelek közötti részeket nem kötelező megadni.

:sp[lit] [file] <Ctrl-W> s	Vízszintesen kettéosztja az aktuális ablakot. Ha a [file] nincs megadva, akkor az aktuális puffert mutatja az új ablakban.
:vs[plit] [file] <Ctrl-W> v	Függőlegesen kettéosztja az aktuális ablakot. Ha a [file] nincs megadva, akkor az aktuális puffert mutatja az új ablakban.
:new [file] <Ctrl-W> n	Vízszintesen kettéosztja az aktuális ablakot. Ha a [file] nincs megadva, akkor egy üres puffert mutat az új ablakban.
:vne[w] [file]	Függőlegesen kettéosztja az aktuális ablakot. Ha a [file] nincs megadva, akkor egy üres puffert mutat az új ablakban.
:q[uit][!]	Bezárja a kurzort tartalmazó ablakot. Ha a !-jel is meg van adva, akkor a módosított puffereket is bezárja, egyébként ebben az esetben hibát jelez.
:on[ly][!]	A kurzort tartalmazó ablak kivételével az összes ablakot bezárja. Ha a !-jel is meg van adva, akkor a módosított puffereket is bezárja, egyébként ebben az esetben hibát jelez.

5.2. táblázat. Kurzormozgatási parancsok

N <Ctrl-w> h	Az N-nel balra lévő ablakba mozgatja a kurzort.
N <Ctrl-w> l	Az N-nel jobbra lévő ablakba mozgatja a kurzort.
N <Ctrl-w> k	Az N-nel feljebb lévő ablakba mozgatja a kurzort.
N <Ctrl-w> j	Az N-nel lejjebb lévő ablakba mozgatja a kurzort.
N <Ctrl-w> H	Az ablakot bal szélre mozgatja.
N <Ctrl-w> L	Az ablakot jobb szélre mozgatja.
N <Ctrl-w> K	Az ablakot legfelülre mozgatja.
N <Ctrl-w> J	Az ablakot legalulra mozgatja.

5.3. táblázat. Helyettesítő karakterek *gnuplot*ban

`%d`	a hónap napja, 1–31
`%m`	hónap, 1–12
`%y`	év, 0–99
`%Y`	év, 4-számjeggyel
`%j`	az év napja, 1–365
`%H`	óra, 0–24
`%M`	perc, 0–60
`%s`	a Unix időszámítás kezdete óta eltelt idő másodpercekben (1970-01-01, 00:00 UTC)
`%S`	másodpercek, 0–60
`%b`	a hónap nevének három betűs angol rövidítése
`%B`	a hónap angol neve

6. fejezet

Képletek és táblázatok

A LaTeX rendszer egyik legnagyobb erőssége a matematikai képletek nyomdai minőségű megjelenítése. A képletek megjelenítésének alapvetően kétféle módja van: a szövegközi mód (pl. $E = mc^2$), és a kiemelt mód, amit az alábbi diffúziós egyenlettel szemléltetünk:

$$\frac{\partial \phi(\vec{r},t)}{\partial t} = \nabla \left(D(\phi,\vec{r}) \nabla \phi(\vec{r},t) \right) \tag{6.1}$$

A LaTeX beépített matematikai képességein túl számos kiegészítő csomag használható. Az egyik legfontosabb és leggyakrabban használt matematikai csomag az `amsmath`[9] csomag, melyet a korábban bemutatott módon a dokumentum bevezetőjébe elhelyezett `\usepackage{`*amsmath*`}` paranccsal tölthetünk be.

A LaTeX matematikai környezetein túl egy másik fontos és gyakran használt környezetet, a táblázatokat is bemutatjuk ebben a fejezetben.[1]

6.1. Matematikai képletek

A matematika képletek nyomdai szedése az egyik legösszetettebb feladat. A képletek szedéséhez a LaTeX-et ún. matematikai módba kell kapcsolni. A szövegközi matematikai módba a dollár jellel (pl. `$`*képlet*`$`) lehet kapcsolni, a 6.1. képletben bemutatott kiemelt matematikai módba pedig az `equation` környezettel lehet átváltani:

```
\begin{equation}
    ...
\end{equation}
```

Látható, hogy a kiemelt módban az egyenletek sorszámot is kapnak. A sorszámozott képletekre a LaTeX forrásállományában a `\label{`*címke*`}` – `\ref{`*címke*`}` párossal kell

[1] Ennek a gyakorlatnak a feldolgozása előtt mindenképpen ismételjük át a 2. gyakorlat anyagát.

6.1. táblázat. A leggyakoribb matematikai módú „ékezetek”.

$\hat{a}$	`\hat{`a`}`
$\tilde{a}$	`\tilde{`a`}`
$\bar{a}$	`\bar{`a`}`
$\vec{a}$	`\vec{`a`}`
$\dot{a}$	`\dot{`a`}`
$\ddot{a}$	`\ddot{`a`}`

hivatkozni. Ha még sincs szükség a képlet sorszámára, akkor a `displaymath` környezettel sorszám nélküli kiemelt képletet is készíthetünk.

Bizonyos parancsok – például a hatványozás („^”) vagy az indexelés („_”) – csak matematikai módban használhatóak. Ennek a fordítottja is igaz, azaz vannak olyan parancsok, amelyek csak a normál szövegben használhatóak, ha matematikai módban szerepeltetjük, a fordítás során hibát emel. Ilyenek például a betűtípust állító parancsok. (`\textrm`, `\textit`, stb.).

6.1.1. Formulák betűkészlete

Matematikai formulákhoz általában sokkal többféle betűtípusra[10] van szükség, mint a normál szöveghez. A változókat döntött betűvel, a függvényneveket álló betűvel kell szedni. Szükség lehet ezen kívül görög, héber, gót, kalligrafikus vagy duplázott betűkre, hogy a különböző típusú mennyiségek jól elkülöníthetőek legyenek. Egy másik lényeges különbség a normál szöveg és a matematikai képletek között, hogy az utóbbi esetén a szóközöknek semmi jelentőségük, az egyes matematikai elemek közötti távolságokat azok funkciója dönti el. Például a változók között más a távolság, mint a relációjelek mellett.

Latin betűk

Változók megjelenítésére leggyakrabban az angol ábécé betűit használjuk. Ezekre is lehet „ékezeteket” tenni, de ezeknek más a jelentésük, mint a normál szöveg ékezetes betűi; például az $\dot{x}$ általában idő szerinti deriváltat jelent, $\bar{x}$ pedig átlagot. A leggyakrabban előforduló „ékezeteket” a 6.1. táblázatban gyűjtöttük össze. Ha több ékezetet szeretnénk egymásra helyezni, akkor használjuk az `amsmath`[9] csomagban található parancsok nagybetűvel kezdődő változatait (pl. `\Hat{`a`}`), amely helyesen egymás fölé igazítja az ékezeteket. A `\wide` kezdetű parancsokkal több betűn átnyúló ékezetet tehetünk (pl. a `\widehat{`abc`}` paranccsal kapjuk: $\widehat{abc}$).

A betűk megjelenését a normál szöveghez hasonlóan változtathatjuk meg azzal a különbséggel, hogy „text” helyett „math”-ot kell írni, pl. a `\mathrm{`*formula*`}` álló betűket

6.2. táblázat. A leggyakoribb matematikai módban használt betűrajzolatok.

Rajzolat neve	Parancs	Példakészlet
Kalligrafikus betű	`\mathcal{A}`	$\mathcal{ABC}$
Duplázott betű	`\mathbb{A}`	$\mathbb{ABC}$
Folyó kalligrafikus	`\mathfrak{A}`	$\mathfrak{ABCabc123}$
Talpatlan betűk	`\mathsf{A}`	$\mathsf{ABCabc123}$
Kövér betűk	`\mathbf{A}`	$\mathbf{ABCabc123}$

szed ki, a `\mathbf{`*formula*`}` kövér betűket eredményez, stb.

Görög betűk

Matematikai formulákban a második leggyakrabban használt betűtípus a görög betűk. Azok a görög betűk, amelyek különböznek minden latin betűtől, a saját nevükből képzett paranccsal adhatóak meg. Például az `$\alpha$` parancs adja az α betűt. A kis görög betűk közül csak az omikronnak van latin megfelelője („o”), ezért ezt az `$o$` paranccsal adhatjuk meg. A nagy görög betűk közül a következőknek van latin megfelelője: ABEZHIKMNOPTX. A többit nagy kezdőbetűs paranccsal kell megadni, pl. a `$\Lambda$` adja a Λ betűt.

A görög betűk alapesetben „állók”. A döntött változatot a parancsok elé írt `\var` előtaggal érhetjük el, amennyiben lehetséges (alapesetben a kis epsilon, theta, pi, rho, sigma és phi betűknek van döntött változata).

A kalligrafikus, gót és duplázott betűk

Leggyakrabban különböző halmazok, operátorok jelölésére kalligrafikus, gót vagy duplázott betűrajzolatokat használunk, vektorokat kövéren szedünk. A 6.2. táblázat felsorolásban erre mutatunk példákat.

6.1.2. Hatványozás, indexek

A 2. fejezetben már említettük, hogy a hatványozás és az indexelés jelölésére két speciális karakter szolgál: a kitevőket a kalap jellel („^”), az indexeket pedig az alulvonás jellel („_”) adhatjuk meg. Mindkét parancs tetszőleges sorrendben, de csak matematikai módban használható. Ha az indexbe egynél több tag kerül, akkor azokat blokkosítani kell, az azonos szinten lévő elemeket kapcsos zárójellel csoportosítjuk: pl. `$x_{i,j}^2$` paranccsal adhatjuk meg a $x_{i,j}^2$ kifejezést. Amennyiben képleteink kiszedése során többszintű indexelésre van szükségünk, akkor is a blokkhatárok helyes megadásával kell az indexeket csoportosítanunk, pl. `$x_{i_k}$` adja a x_{i_k} sorközi képletet.

Az alsó indexek kerülhetnek egy adott matematikai objektum alá, vagy mellé is. Szövegközi képletek esetén alapesetben az adott objektum mellé kerülnek (pl. $\lim_{x\to 0} 1/x$). Kiemelt módban függvények esetén (pl. sin, cos, exp, stb.) szintén a függvény mellé, azonban operátorok esetén (pl. lim, max, $\sum$, stb.) az operátor alá kerül az index:

$$\lim_{x\to 0} \frac{1}{x}.$$

6.1.3. Törtek, gyökvonás

Egyszerűbb törtek megjelenítésére használhatjuk a `/` jelet (pl. `$x/y$` adja a x/y kifejezést). Komplikáltabb esetekben, amikor a számlálót és a nevezőt egymás alá akarjuk írni, használjuk a `\frac{`*számláló*`}{`*nevező*`}` parancsot. Jól látszik, hogy ennek a parancsnak két kötelező argumentuma van: az elsőben a számlálót, a másodikban pedig a nevezőt kell megadnunk.

Gyökvonás jelölésére szolgál a `\sqrt{`*gyök*`}` parancs. A parancs opcionális argumentumaként megadható, hogy hányadik gyököt veszünk.

6.1.4. Operátorok, függvények

A legfontosabb operátorok a szumma ($\sum$) a produktum ($\prod$), az integrálás ($\int$) és a limesz (lim), amiket sorrendben a `\sum`, `\prod`, `\int` és a `\lim` parancsokkal érhetünk el. Ezeknek a operátoroknak az alsó és felső határát az alsó és felső indexszel adhatjuk meg: pl. `$\sum_{i=0}^nx_i$` paranccsal adhatjuk meg a

$$\sum_{i=0}^{n} x_i$$

kifejezést.

A több betűből álló függvényneveket is parancsként kell megadni, különben a LaTeX változóként kezeli a függvény betűit. Hasonlítsuk össze például a $sinx$ és a $\sin x$ kifejezéseket. A leggyakrabban használt függvénynevek: `\exp`, `\log`, `\ln`, `\sin`, `\cos`.

6.1.5. Relációjelek

A legegyszerűbb relációjeleket ($<$, $=$, $>$) közvetlenül megadhatjuk a billentyűzetről. Egyéb relációjeleket parancsokkal adhatunk meg. A legfontosabb relációjelek a 6.3. táblázatban találhatók.

6.1.6. Zárójelek

Az igényesen kiszedett matematikai kifejezésekben a zárójelek mérete függ attól, hogy mekkora a bennük foglalt képletrész mérete. Annak érdekében, hogy a LaTeX pontosan

6.3. táblázat. Legfontosabb relációjelek

$<$	$<$	$\equiv$	\equiv	$\leq$	\le
$=$	$=$	$\neq$	\neq	$\geq$	\ge
$>$	$>$	$\in$	\in	$\ll$	\ll
$\approx$	\approx	$\ni$	\ni	$\gg$	\gg
$\sim$	\sim	$\notin$	\notin	$\propto$	\propto

6.4. táblázat. Legfontosabb egyéb jelek

$\pm$	\pm	∂	\partial	$\forall$	\forall
$\mp$	\mp	∞	\infty	$\exists$	\exists
$\times$	\times	∇	\nabla	$\Re$	\Re
$\cdot$	\cdot	$\emptyset$	\emptyset	$\Im$	\Im
$\to$	\to	$\hbar$	\hbar	$\|$	|

meg tudja állapítani az összetartozó zárójeleket, a zárójelek előtt adjuk ki a `\left` és `\right` parancsokat. Ezeknek a parancsoknak mindig párban kell állniuk. Például a

$$\left(\sum_{i=0}^{n} x_i\right)^2 \tag{6.2}$$

kifejezést a következőképpen kell megadni: `\left(\sum_{i=0}^n x_i\right)^2`. Ha az egyik zárójelet nem akarjuk kitenni, akkor az adott zárójel helyett írjunk egy pontot.

6.1.7. Egyéb jelek

Matematikai képletekben számtalan egyéb jelre lehet szükség. Ezeknek a felsorolása meghaladja ennek a jegyzetnek a kereteit. A legfontosabb jeleket azonban röviden összefoglaltuk a 6.4. táblázatban.

6.2. Táblázatok

Tudományos munkák elkészítésekor – legyen az egy szakdolgozat vagy egy folyóiratcikk – szokás táblázatokat használni, ami könnyen átlátható formában összegzi egy vizsgálat eredményeit. Szükséges tehát szót ejtenünk a táblázatkészítés csínjáról, hogy jó minőségű munkát készíthessünk, amelynek eredményeképpen igényes táblázatokat nyomtathatunk.

Mivel a LaTeX nem *Excell-jellegű* környezet, tehát ha túl bonyolult táblázatot kell készíteni – (illetve a cellák tartalma egymás függvényei – akkor előfordulhat olyan eset, amikor célszerű egy külső programmal elkészíteni azt és ábra formátumban csatolni a forrásban. Az egyszerű táblázatok kiszedése könnyű feladat. Komplexebb feladatok megoldására standard LaTeX-kiegészítő csomagok állnak rendelkezésre, melyek közül említésre méltó `tabu` csomag, ami számos egyéb táblázatkészítéssel kapcsolatos csomag[2] jó tulajdonságát ötvözi. Használatához elengedhetetlen az alapok ismerete, ebben a fejezetben az lapokat ismertetjük.

6.2.1. Normál szövegbe ágyazott táblázatok

A szövegkörnyezet függvényében a LaTeX-ben más-más környezet segítségével hozhatunk létre táblázatokat. A `tabular` környezettel a normál szövegben dolgozhatunk, a környezet nyitásakor kell megadnunk az oszlopok kiszedéséhez szükséges alapvető információt, tehát a táblázat a `\begin{`*tabular*`}{`*oszlopleíró*`}` paranccsal kezdődik és a táblázat végét a `\end{`*tabular*`}` forma zárja. Alapesetben az oszlopleíróban a következő jelek szerepelhetnek:

1. a függőleges vonal és a dupla függőleges vonal (`|` és a `||`) a cellák oszlopait határoló függőleges kihúzást illetve dupla kihúzást kapcsolja be,
2. az `l` betű hatására balra igazított oszlopot,
3. míg az `r` betű hatására jobbra igazított oszlopot definiálunk,
4. a `c` betű középre igazított oszlopot ad a táblázathoz.

Az oszlopok szélességét a bennük foglalt tartalom határozza meg, automatikusan. Előfordul olyan eset, hogy meg kell követelnünk a fix oszlopszélességet, mely esetben a `p{...}` forma használható, a pontok helyén szerepeltetve valamilyen nyomdai vagy a felhasználó által definiált hosszúságot (pl. `1pt`, `3em`, ...). A környezet megengedi, hogy az oszlopokat elválasztó minta ne vonal legyen, erre tipikusan akkor van szükségünk, amikor különböző számú tizedes jegyet tartalmazó számokat szeretnénk a tizedesvessző köré rendezni. Az oszlopelválasztó megadására a `@{.}` kifejezést vezették be, ahol a pont jelöli az elválasztó mintát. A táblázat sorait határoló vízszintes vonalat a `hline` paranccsal adhatjuk meg.

A táblázat elemeinek feltöltése soronként történik, a táblázat sorait a duplázott repjel (\\) mutatja. Egy soron belül az oszlopokat a `&` jel választja el. Például a forrásban szereplő következő kifejezés:

[2] A teljesség igénye nélkül táblázatkészítéssel kapcsolatos csomagok még: `longtable`, `tabularx`, `hhline` és `multirow`

```
\begin{tabular}{l|r@{,}l}
Név & \multicolumn{2}{c}{Átlag}\\
\hline
Gizi & 5&0 \\
Géza & 4&5 \\
Pityu& 3&25\\
Vilma& 3&12415
\end{tabular}
```

a dokumentumban fordítás után így néz ki:

Név	Átlag
Gizi	5,0
Géza	4,5
Pityu	3,25
Vilma	3,1415

A felvázolt példában felfedezhető, hogyan lehet előre definiált oszlopokat összeejteni a `multicolumn` parancs segítségével.

Amennyiben bonyolultabb táblázatokat kell kiszedetni LaTeX-ben lehetőségünk van a `tabular` környezetek egymásba ágyazására, ám megfontolandó, hogy olyan esetekben, amikor egy oszlop sorait kell egybenyitni, akkor az erre kifejlesztett `multirow`[11] csomagot használjuk, melyet a dokumentumforrás bevezetőjében a `\usepackage{`*multirow*`}` kell betöltenünk. Cizelláltabb táblázatkeretezéshez a `hhline`[12] csomag használatát javasoljuk.

Az előbb ismertetett `tabular` környezet az `includegraphics` analógja. A korábbi 3.4. fejezetben megtanultuk, hogy az ábrákat lehet feliratozni, és azokra a folyószövegből hivatkozhatunk. A `figure` környezet mintájára a táblázatok úsztatására a `table` környezetet használjuk, melynek sémája így fest:

```
\begin{table}[úsztatási instrukciók]
\begin{center}
\begin{tabular}{oszlopdefiníció}
 ... & ... \\
\end{tabular}
\end{center}
\caption{Ez itt a táblázatunk ábraaláírása.}
\label{tab:table1}
\end{table}
```

A fenti példában a következő LaTeX környezeteket alkalmaztuk:

- a `\begin{`*table*`}` nyitja meg és a `\end{`*table*`}` zárja le az úsztatott táblablokkot tartalmazó környezetet,

- a `\begin{`*center*`}` és a `\end{`*center*`}` parancsokkal átölelt objektumok a fordítás során középre rendeződnek, ami esetünkben azt jelenti, hogy a táblázat jobb és bal margótól vett távolsága egyforma.
- a `tabular` környezetben a szakasz elején ismertetett módon készítjük el a táblázatot,
- a `\caption{`*összegzés*`}` parancs segítségével rövid, figyelemfelkeltő táblázat aláírást adhatunk meg,
- a `\label{`*címke*`}` parancs hivatkozási pontot szúr a táblázat sorszámához, melyre később a szövegben a `\ref{`*címke*`}`, stb. módon hivatkozhatunk.

Meg kell jegyezni, hogy a `table` környezetben is használhatjuk az `includegraphics` parancsot, így a táblázatoknál megszokott feliratot kapunk egy külső fájlból hivatkozott kép alatt.

6.2.2. Táblázatok a matematikai módban

Matematikai módban az `array` környezet segítségével tudunk táblázatokat megadni. Erre tipikusan akkor van szükség, ha mátrixok elemeit akarjuk szemléltetni. A következő torzító nyújtást leíró operátort leíró LaTeX-részlet eredménye a 6.3. képletben látható.

```
\begin{equation}
S_\mathbf{a} = \left(%
 \begin{array}{ccc}
 \alpha_x&\delta&0\\
 0&\alpha_y&0\\
 0&0&1
 \end{array}
\right),\label{eq:proj}
\end{equation}
```

$$S_{\mathbf{a}} = \begin{pmatrix} \alpha_x & \delta & 0 \\ 0 & \alpha_y & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{6.3}$$

6.3. Példák és feladatok

Feladatok

F6.1. Készítsünk *gnuplot* segítségével ábrát egy olyan tetszőleges paraméterű parabolá-ról, ami metszi az $y = 0$ egyenletű egyenest. Írjunk LaTeX dokumentumot, amely-hez csatolja ezt az ábrát és az ábraaláírásban szerepelteti a parabola képletét. A szövegben szedje ki a másodfokú egyenlet megoldóképletét:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{6.4}$$

Végezetül egy táblázatban foglaljuk össze a parabola paramétereit és a gyököket, az alábbi formában:

a	
b	
c	
x_1	
x_2	

F6.2. Írjunk LaTeX dokumentumot, amelyben megadja a Fourier-sorfejtés és a Fourier-transzformáció képleteit. Hivatkozzunk a szövegben a képletekre. A T-periodikus függvények sorba fejthetők a periodikus függvények bázisán, $\omega_0 := 2\pi/T$ jelöléssel:

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\omega_0 k t},$$

ahol az együtthatók:

$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-i\omega_0 k t} \mathrm{d}t.$$

Az L^2-integrálható függvényeken értelmezhető Fourier-transzformáció képletei:

$$\begin{aligned} \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} \mathrm{d}t \\ f(t) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} \mathrm{d}\omega. \end{aligned}$$

F6.3. Írjunk LaTeX dokumentumot, amelyben megadja a normális eloszlás sűrűségfüggvényét,

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

valamint a sűrűségfüggvény maximumát:

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=\mu} = 0.$$

F6.4. Ismerkedjen meg a LaTeX `hhline`, `multirow` csomagjaival és a `multicolumn` paranccsal. Próbálja meg kiszedni a következő táblázatot:

Csoport	Név	1. feladat	
		Elmélet	Gyakorlat
A	Okos Orsolya	5	5
	Rombusz Róbert	5	3
	Kalapács Kornél	2	5
B	Gáz Géza	5	5
	Puskázó Petra	4	1

F6.5. Az `array` parancs segítségével szedjük ki az alábbi képletet:

$$f(x) = \begin{cases} \sin\left(\frac{2\pi t}{T}\right), & \text{ha} \quad x \in [0; T/2] \\ 0, & \text{ha} \quad x \in [T/2; T]. \end{cases}$$

7. fejezet

Haladó *gnuplot*

7.1. Függvényillesztés

A fejezet feldolgozása előtt mindenképpen ismételjük át a 3. fejezet anyagát.

A *gnuplot* program az adatsorok és függvények ábrázolásán túl nagyon jól használható függvényillesztésre is. Ennek során egy adatsor pontjaihoz az általunk megadott függvény paramétereit állítja be úgy, hogy az eredményül megkapott paraméterekkel leírt függvény a lehető legjobban illeszkedjen az adatsor pontjaihoz.

Az ábrák elkészítéséhez hasonlóan a függvények illesztése is egyaránt lehetséges interaktív módban és szkriptek segítségével, azonban bonyolultabb függvények illesztése során az eredmények helyes értelmezéséhez érdemes interaktív üzemmódot használni.

7.2. A legkisebb négyzetek módszere

Általában a függvények legjobb illeszkedéséhez, azaz a függvényhez tartozó paraméterek a meghatározására számos módszer ismert a szakirodalomban. A *gnuplot*ban a *nem lineáris legkisebb négyzetek (NLLS*[1]*)* módszert implementálták, ezt a módszert használja mind az egyenesek, mind a bonyolultabb függvények numerikus paraméterbecslése során.

A módszer alapja a következő. Adott egy modell függvény $f(x; a, b, c, \dots)$, melynek x a független változója, $a, b, c, \dots$ pedig a függvény ismeretlen paraméterei. A mérési adatoknak a modelltől való eltérésének mértékére bevezethetjük a következő mennyiséget:

$$\chi^2(a, b, c, \dots) = \sum_i \frac{(y_i - f(x_i, a, b, c, \dots))^2}{\sigma_i}, \tag{7.1}$$

ahol a x_i jelenti a független változó értékeit, amelyeket szabadon változtathatunk, y_i jelöli az i-edik, azaz az x_i-hez tartozó mérési adatot, és a σ_i az y_i függő változó szórássa.

[1] *Non-linear least squares*

A fenti összefüggés az illesztés hibáját az elméleti modell szerinti $f(x_i, a, b, c, \dots)$ érték és a mért y_i értékek közötti négyzetes eltérések összegével méri, adott $a, b, c, ...$ paraméterek mellett. A módszer elnevezése onnan származik, hogy a legjobb illesztést azok mellett a paraméterértékek mellett kapjuk, ahol χ^2 a legkisebb.

A numerikus számítás során konkrét $a, b, c, \dots$ paraméterértékkel kezdődik az illesztés. A χ^2 érték meghatározását követően a *gnuplot* kicsit módosított paraméterértékekkel újraszámolja χ^2 értékét. Amennyiben χ^2 értéke csökkent, akkor a paraméterek jó irányban változtak, amennyiben nőtt, akkor rossz irányban. Az eredménynek megfelelően újra módosítjuk a paramétereket majd újraszámoljuk a χ^2 értékét. Ezt a lépéssort hajtja végre a *gnuplot* újra és újra amíg a kívánt pontosságot el nem érjük.

Valójában az illesztés nemcsak akkor ér véget, ha a kívánt pontosságot elértük, hanem akkor is, ha az egyes lépések során a négyzetes eltérés változása egy megadott küszöbérték alá csökken. Ebből következik, hogy egy amúgy pontatlan, rosszul illeszkedő görbe illesztése is befejeződik, ha már nem lehet elég gyorsan javítani a pontosságon.

A legkisebb négyzetek módszere erősen függ a kezdeti paraméterek minőségétől. Bonyolult, nem-lineáris függvények illesztésekor számos probléma származhat a kezdeti paraméterek nem megfelelő megválasztásából. Előfordulhat, hogy valamely paraméter változtatására a χ^2 érték nem változik jelentősen, mert esetleg egy másik paraméter függvénybeli szerepe elnyomja a paraméter hatását. Esetleg χ^2 értéke egy amúgy helytelen *lokális minimumba* kerül, és a paramétereket változtató algoritmus leállítja a keresést a lokális minimumban, és ennek következtében a valódi minimumot nem tudja elérni a program. Ezen esetek elkerülése érdekében figyelni kell a kezdeti paraméterek közelítőleg helyes megadására, amit aztán az illesztés pontosítani tud.

A módszerről bővebben a Wikipédia oldalain lehet olvasni.

A *gnuplot* a függvényillesztés során a fenti képlet számítását és a minimális χ^2-hez tartozó paraméterek megkeresését automatikusan végzi, azonban a kezdeti paraméterek megadásra és az eredmény helyességének eldöntésére magunknak kell odafigyelnünk. Ennek legegyszerűbb módja, ha minden illesztés után ábrázoljuk az adatokat és az illesztett függvényt.

7.3. Függvények illesztése *gnuplot*ban

A *gnuplot* függvény illesztésre használt parancsa a `fit`, amelynek három kötelező argumentuma van. Ezek sorrendben az illesztendő függvény, az adatsor amire az illesztést végezzük és az illesztendő paraméterek nevei, melyet a `via` kulcsszó után kell vesszővel elválasztva felsorolni:

```
gnuplot>fit a+b*x "adatok.dat" via a,b
```

A fenti példában a `fit` parancs első argumentuma egy egyenes egyenlete, `a+b*x`, amelynek tengelymetszete `a`, meredeksége `b`. Az idézőjelek között adjuk meg az adatfájl

nevét, esetleg a `plot` parancsnál már látott módon (ld. a 3.2.1. fejezetben) használhatjuk a `using` kapcsolót az adatfájl különböző oszlopaiban használatára, illetve az adatok transzformációjára. A `via` kapcsoló után vesszőkkel felsorolva adjuk meg az illeszteni kívánt paramétereket. A fenti példában mindkét paramétert illesztjük, tehát `a, b`.

A `fit` parancs kiadása után a *gnuplot* kiírja a képernyőre az iterációs lépéseket, amelyekben lépésről lépésre változtatja a program az illesztési paramétereket. Az alábbi részlet az illesztési eredményeket mutatja be:

```
After 5 iterations the fit converged.
final sum of squares of residuals : 30.0347
rel. change during last iteration : -4.68348e-06

degrees of freedom    (FIT_NDF)                        : 997
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.173566
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0301251

Final set of parameters    Asymptotic Standard Error
=======================    ==========================
a        = 0.711585         +/- 0.1917      (26.94%)
b        = -0.0144642       +/- 0.003379    (23.36%)
```

Amikor az egymást követő lépésekben kiszámolt χ^2 értékek már alig változnak (a különbségük alapbeállításban már kisebb, mint 10^{-5}) az iteráció leáll és megkapjuk az illesztett paramétereket és azok becsült hibáit.

Az illesztés „jóságát” a négyzetes eltérésből kaphatjuk meg. A négyzetes eltérés illesztéskor kapott értéke („final sum of squares of residuals”) elméletben χ^2 eloszlást követ. A szabadsági fokok száma a „degrees of freedom” mellett található. Egy adott konfidenciaszint mellett meghatározhatjuk a χ^2 eloszláshoz tartozó kritikus értéket, és ha ez nagyobb, mint az illesztésből kapott négyzetes eltérések összege, akkor az illesztés statisztikai értelemben jó, ellenkező esetben pedig rossz.

Kevésbé precíz módon a megadott hibából is következtethetünk az illesztés minőségére. Amennyiben a hiba alacsony (néhány, esetleg néhány tíz százalék), akkor az illesztett paraméter érték nagy valószínűséggel helyes, azonban ha a hiba nagy (több ezer, esetleg millió százalék) akkor az illesztés szinte biztosan hibás eredményre vezetett. Ebben az esetben próbáljunk meg jobb becslést adni a kezdeti paraméterekre, és futtassuk újra az illesztést.

Amennyiben a kezdeti paraméterértékeket, amelyekre az első χ^2 számítást fogjuk elvégezni, nem adjuk meg, akkor a *gnuplot* ezeknek a paramétereknek automatikusan 1-et állít be kezdeti értékül. A paraméterekben lineáris függvények és „normális” adatok esetén az illesztés a kezdeti értékektől függetlenül a legkisebb négyzetek módszere a globális minimumhoz konvergál, de bonyolultabb esetekben érdemes az adatokat egy `plot` paranccsal ábrázolni, és szemmel megbecsülni a paraméterek kezdeti értékét, és azokat

a `fit` parancs kiadása előtt megadni (esetleg újabb `plot`-tal ellenőrizni a becslésünk helyességét):

```
gnuplot> a=3.5
gnuplot> b=-0.0001
gnuplot> fit a+b*x "adatok.dat" via a, b
gnuplot> plot a+b*x, "adatok.dat"
```

Magát a függvényt is lehet előre definiálni, ami bonyolultabb függvények esetén egyszerűsíti a parancsok begépelését:

```
gnuplot> f(x)=1/(sqrt(2*pi)*s)*exp((-(x-m)**2)/(2*s**2))
gnuplot> m=2
gnuplot> s=1
gnuplot> fit f(x) "adatok.dat" via m, s
```

Felhívjuk a figyelmet, hogy a *gnuplot*ban a hatványozás jele a `**` (pl. `x**2`), ellenben pl. a LaTeX-ben megszokott kalap (^) jellel.

7.4. Három dimenziós ábrázolás

A 3. fejezetben bemutattuk a `plot` parancsot, mellyel kétdimenziós ábrákat lehet készíteni. A *gnuplot*ban lehetőség van háromdimenziós ábrázolásra is a `splot` parancs segítségével. A `splot` parancsot a `plot` parancshoz hasonlóan lehet paraméterezni, például a `using`, `with`, `axes`, stb. kapcsolókkal. Adatok ábrázolásánál eggyel több adatoszlopot kell megadni, mint a `plot` parancsnál, analítikus függvények ábrázolásánál pedig nemcsak az x változót lehet használni, hanem az y változót is:

```
gnuplot> splot [-pi:pi][-pi:pi] cos(x)+cos(y)
```

A parancs kimenete a 7.2. ábrán látható. A fenti parancsnál a megadott ábrázolási tartományok értelemszerűen az x és az y tengelyre vonatkoznak. A z tengely ábrázolási tartományát egy harmadik tartománnyal állíthatjuk be.

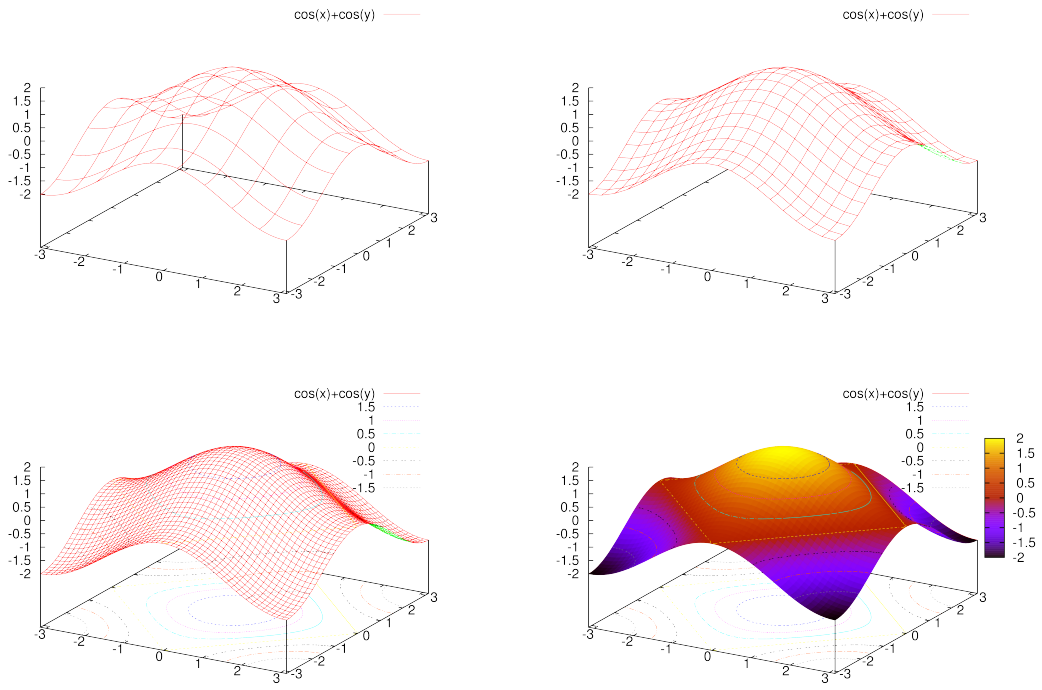
A `set` parancsnak számos olyan beállítása van, amelyek csak háromdimeniós esetén értelmezettek. Ezek főként arra szolgálnak, hogy segítsék a háromdimenziós ábrázolást. Az alábbiakban röviden összefoglaljuk ezeket.

Ha kiadjuk a

```
gnuplot> set hidden3d
```

parancsot, akkor a *gnuplot* úgy ábrázolja a háromdimenziós felületeket, mintha azokon nem lehetne átlátni. Ilyenkor a *gnuplot* a felületet síkidomokkal közelíti.

A `set isosamples [x][,y]` paranccsal tudjuk beállítani, hogy az ábrázoláshoz használt rács milyen sűrű legyen. Például a



7.1. ábra. Háromdimenziós grafikonok *gnuplot*tal. A második ábrán az eltakart vonalak nem látszanak. A harmadik ábrán kontúrvonalakat is látunk, míg a negyedik ábrán a felületet színekkel ábrázoltuk.

```
gnuplot>set isosamples 20
```

parancs kiadása után a felületet 20×20-as rácson ábrázolja. Amennyiben a `set hidden3d` parancsot is kiadtuk, akkor érdemes a `set isosamples` értékét legalább `20`-ra beállítani az alapértelmezett, `10` helyett, különben a felület nem lesz sima.

A `set contour` parancs segítségével kontúrvonalakat rajzolhatunk az ábrára. A kontúrvonalak lehetnek az alapon (`base`), a felületen (`surface`), vagy mindkettőn (`both`). A kontúrvonalak beállításait a `set cntrparam` parancs segítségével állíthatjuk be.

Lehetőség van arra is, hogy a felületet a magasságtól függően szinezzük. Ehhez a `set pm3d` parancsot kell használnunk. A szinezést kérhetjük az alapra, az ábra tetejére, vagy a felületre az `at` kapcsolóval. Például a

```
gnuplot>set pm3d at bs
```

parancs kiadása után a szinezés mind az alapon, mind a felületen meg fog jelenni.

7.5. Paraméteres görbék ábrázolása

A 3. fejezetben bemutattuk a *gnuplot* `plot` parancsának legfontosabb alkalmazását, amelyben az adatokat és a függvényeket derékszögű koordinátarendszerben ábrázoltuk. Előfordulhat azonban, hogy nem ez az ábrázolási mód a legmegfelelőbb.

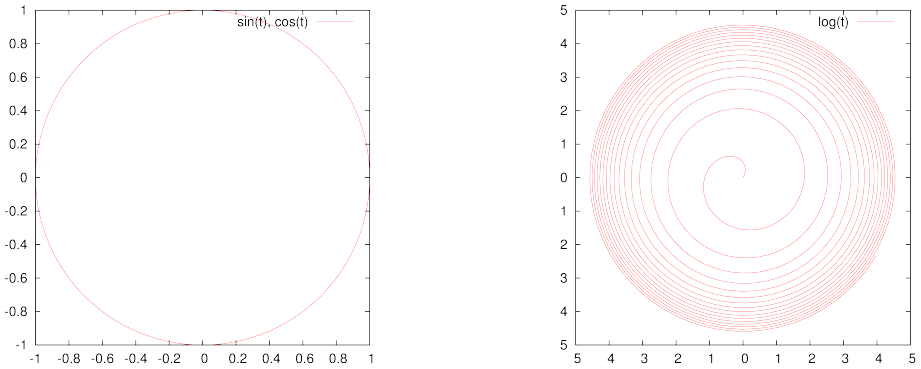
A *gnuplot*ban lehetőség van derékszögű koordinátarendszertől eltérő rendszerben is ábrázolni az adatokat. A `set parametric` parancs hatására a *gnuplot* parametrikus módba kapcsol. Ekkor a függvényábrázoláskor használt segédváltozók is megváltoznak. Kétdimenziós esetben ilyenkor a `t`, háromdimenziós ábrázolásnál az `u`, és a `v` segédváltozókat kell használni.

Parametrikus ábrázolásnál megváltozik a `plot` parancs szintaktikája is. Ilyenkor minden koordinátához egy–egy függvényt kell megadnunk. Például az alábbi paranccsal egy kört tudunk kirajzolni:

```
gnuplot>set parametric
gnuplot>set size square
gnuplot>plot sin(t), cos(t)
gnuplot>unset parametric
```

Polárkoordináták szerint is ábrázolhatunk függvényeket. Ilyenkor a segédváltozó a `t`, ami a polárkoordinátarendszerben az elfordulás szögét edja meg. A megadott függvény az origóról mért távolságot adja meg. Például logaritmikus spirált az alábbi paranccsal tudunk kirajzolni:

```
gnuplot>set polar
gnuplot>set size square
gnuplot>plot log(t)
gnuplot>unset polar
```



7.2. ábra. Az első ábrán parametrikus ábrázolási módban egy kör látható. A második ábrán polárkoordinátákban megadott logaritmikus spirál látható.

7.6. Példák és feladatok

Gyakorló példák

Gy7.1. Töltsük le a gyakorlat weboldalától a `fitadatok1.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk $f(x) = a + bx$ egyenest a fájl első két oszlopában lévő adatokra, majd készítsünk PostScript formátumú ábrát, amin az adatsor és az illesztett egyenes is látható.

Gy7.2. Töltsük le a gyakorlat weboldalától a `fitadatok2.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk $f(x) = a + bx$ egyenest a fájlban lévő adatokra az $x \in [20; 60]$ tartományra, majd készítsünk PostScript formátumú ábrát, amin az adatsor és az illesztett egyenes is látható.

Gy7.3. Töltsük le a gyakorlat weboldalától az `fitadatok3.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk

$$f(x) = a + bx + cx^2 \tag{7.2}$$

polinomot a fájlban lévő adatokra. Figyeljünk a paraméterek kezdeti értékének helyes megadására. Készítsünk postScript formátumú ábrát, amin az adatsor és az illesztett görbe is látható.

Gy7.4. Töltsük le a gyakorlat weboldalától az `fitadatok4.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk

$$f(x) = p\sin(rx - s) \tag{7.3}$$

függvényt a fájlban lévő adatokra. Figyeljük meg az illesztés eredményét amennyiben nem állítunk be kezdeti paramétereket, majd illesszük a függvényt helyes kezdeti paraméterek megadásával. Készítsünk PostScript formátumú ábrát, amin az adatsor és az illesztett görbe is látható.

Feladatok

F7.1. Töltsük le a gyakorlat weboldalától a `fitadatok5.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk $f(x) = a + bx$ egyenest a fájl első két oszlopában lévő adatokra, majd készítsünk POstScript formátumú ábrát `F7.1.eps` néven, amin az adatsor és az illesztett egyenes is látható.

F7.2. Töltsük le a gyakorlat weboldalától a `fitadatok6.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk $f(x) = a + bx$ egyenest a fájlban lévő adatokra $x = 40$-ig, majd készítsünk PostScript formátumú ábrát, `F7.2.eps` néven, amin az adatsor és az illesztett egyenes is látható.

F7.3. Töltsük le a gyakorlat weboldalától a `fitadatok7.dat` adatsort. *vim* segítségével ismerkedjünk meg az adatfájl felépítésével. Illesszünk

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{7.4}$$

Gauss-görbét (Normális eloszlást) a fájlban lévő adatokra. Figyeljünk a paraméterek kezdeti értékének helyes megadására (a π értékét ismeri a *gnuplot* és a `pi` változó használatával hivatkozhatunk rá: ld. a `print pi` parancsot). Készítsünk PostScript formátumú ábrát `F7.3.eps` néven, amin az adatsor és az illesztett görbe is látható.

F7.4. Töltsük le a gyakorlat weboldalától a `fitadatok8.dat` adatsort. Illesszünk négyzetgyökfüggvényt az adatsort megfelelő intervallumán. Ismételjük meg az illesztést úgy is, hogy előtte *gnuplot*ban az adatokat egyenessé transzformáljuk. Készítsünk PostScript formátumú ábrákat az illesztésről.

F7.5. Töltsük le a gyakorlat weboldalától a `fitadatok9.dat` adatsort, ami a egy izotóp bomlási folyamatát modellezi. Az adatsor pontjai a keletkező γ-részecskék energiájának gyakoriságsűrűségét mutatják. Az eloszlás azonos szórású Gauss-görbék kombinációjával jellemezhető, a háttér lineáris járulékot ad. Végezzük el az illesztési feladatot. Nagyon fontos a paraméterek kezdeti értékének helyes megadására. Készítsünk PostScript formátumú ábrát, amin az adatsor és az illesztett görbe egyaránt jól látható.

F7.6. Készítsünk egy LaTeX dokumentumot, amiben elhelyezzük a fenti ábrákat és a hozzájuk tartózó képleteket az illesztési paraméterekkel. Írjunk egy kis szöveget, amiben hivatkozzunk minden ábrára és képletre a megfelelő helyen.

8. fejezet

Programozás

Ebben a fejezetben néhány alapvető programozási fogalmat mutatunk be a *gawk* programnyelven keresztül. A fejezetnek a feldolgozása előtt mindenképpen ismételjük át a 4. fejezet anyagát.

8.1. Bevezetés

Mielőtt részletesebben ismertetnénk a *gawk* programozását, röviden áttekintjük a számítógépes programokkal kapcsolatos néhány alapvető fogalmat. Ezek a fogalmak annyira alapvetőek, hogy csaknem minden magasabb szintű programnyelvben megtalálhatóak. De mi is a programnyelv?

8.1.1. Programnyelvekről

A *programnyelv* egy olyan nyelv, amelyben a programozó az emberi nyelvhez hasonlóan lépésről–lépésre megfogalmazhatja, hogy mit csináljon a számítógép. Egy adott feladatnak egy adott programnyelvben megadott leírása a *forrásprogram*. Természetesen nincs olyan programnyelv, amely minden szempontból jobb lenne a többi programnyelvnél; bizonyos feladatokra az egyik, más feladatokra egy másik programnyelv alkalmasabb. Ideális esetben az adott feladathoz legjobban illeszkedő programnyelvet kellene használjuk. Általában két ellentétes szempontot, kell figyelembe venni:

- Mennyi időbe telik a program megírása, és
- mennyi időbe telik a program futtatása?

A forrásprogramokat alapvetően az emberek tudják értelmezni, a számítógép nem. Ahhoz, hogy a számítógép futtasson egy programot, azt előbb a számítógép által értelmezhető ún. gépi kóddá kell alakítani. A gépi kóddá alakításhoz egy programot kell futtatni, amely értelmezi a forrásprogramot. A gépi kóddá alakításnak két módja van:

Fordító (compiler) Egy fordítóprogram előre elkészíti a számítógép által futtatható állományt, amelyet aztán bármikor önállóan futtathatunk. Ilyen programnyelvek pl. a C/C++, a Pascal és a Fortran.

Értelmező (interpreter) Egy értelmezőprogram sorról–sorra értelmezi és futtatja a forrásprogramot. Ilyen programnyelv pl. a *bash*, a *gnuplot*, és a *gawk*. Az interpreter nyelvek forrásprogramját *szkript*nek is nevezik.

A fordító előnye, hogy a fordításkor ellenőrzi, hogy a forrásprogram szintaktikailag helyes-e, valamint hogy a lefordított program gyorsabban fut, mint a szkriptek, és a programok interpreter nélkül is futtathatók. A hátránya viszont, hogy általában hosszabb időbe telik egy forrásprogram megírása és tesztelése, mint egy szkriptnek, mivel mindig újra és újra le kell fordítani a forrásprogramot.

Az ebben a fejezetben bemutatott *awk* programnyelv egy interpretert használ a forrásprogramok futtatásához.

8.1.2. Adatok tárolása – változók, konstansok, tömbök

A legegyszerűbb programokhoz is szükség van bizonyos adatok tárolására. Az adatok tárolására általában többféle adatstruktúra áll rendelkezésre. A leggyakoribb adatstruktúra a *változó*, melyben egyszerre egy adatot tárolhatunk.

A változó értékét a program tetszőleges sokszor írhatja és olvashatja. Néha szükségünk lehet olyan „változóra”, amelynek csak egyszer adunk értéket, és biztosítani akarjuk, hogy ezt az értéket később nem változtatjuk meg véletlenül. Az ilyen „változót” *konstansnak* nevezzük. Ilyen lehet például a `pi=3.141592653`.

A *tömbök* változókból képzett egy vagy többdimenziós vektorok. A tömbök elemeit a tömb dimenziójának megfelelő számú egész számmal címezhetjük meg, melyeket *indexeknek* nevezzük.

Megjegyezzük, hogy magasabb szintű programnyelvekben számos más adatstruktúra is létezik, például a lista (list), a halmaz (set), a verem (stack), a sor (queue), stb.

8.1.3. Adattípusok

Ahhoz, hogy egy programnyelv az adatokat kezelni tudja meg kell határozni, hogy a változók milyen típusúak. A változókat az alábbi főbb típusokba szokás sorolni:

integer Egész számok tárolására szolgáló változó. A változó tárolásához szükséges memória méretétől függően az alábbi speciális típusok is lehetségesek:

- **bool** Egy bit információ (igaz-hamis) tárolására alkalmas adattípus.
- **char** Egy byte információ tárolására alkalmas adattípus. Mivel az angol ABC-t (írásjelekkel, és egyéb speciális jelekkel együtt) az ASCII szabvány egy byte-on tárolja, ez az adattípus elsősorban egy karakter tárolására szolgál.

float Lebegőpontos számok tárolására szolgáló adattípus, amellyel lehetőség van törtek ábrázolására. Mivel a számítógép a számokat kettes számrendszerben ábrázolja, ezért a tízes számrendszerbeli számokat általában csak közelíteni lehet ezzel az adattípussal.

string Szöveges karaktersorozatok tárolására alkalmas adattípus.

A fordítókon alapuló programnyelvekben (pl. C, Pascal) a változók adattípusát a használatuk előtt deklarálni kell, azaz pontosan meg kell a típust határozni. A szigorú típusegyeztetés lehetővé teszi, hogy számos programozási hibát a fordítóprogram kiszűrjön.

8.1.4. Vezérlés

A programok futása során az egyes utasítások egymás után kerülnek végrehajtásra. Azt, hogy melyik utasítás kerül végrehajtásra legközelebb, a vezérlőutasítások határozzák meg. Kétfajta vezérlőutasítás minden programnyelvben megtalálható: a feltételválasztó és a ciklusszervező vezérlőutasítás.

A feltételválasztó utasítás (általában az „`if`" utasítás) egy kifejezés igazságtartalmát vizsgálja meg. Ha a kifejezés igaz, akkor a vezérlést egy adott utasításhoz továbbítja, ha hamis, akkor pedig egy másik utasításhoz.

A ciklusszervező utasítások egy adott utasítást hajtanak végre ismételten, ameddig egy megadott feltétel nem teljesül.

8.1.5. Függvények

Ha valamilyen feladatot gyakran akarunk egy programban elvégezni, akkor célszerű azt egy külön *függvényként* megvalósítani. A matematikából ismert függvényekhez hasonlóan a programok függvényeinek is vannak argumentumai, és visszatérési értéke.

A programozó által megírt függvényeken túl, a programozás megkönnyítésére, minden programnyelvhez különböző függvénykönyvtárak állnak rendelkezésre. Ezek a függvénykönyvtárak általában tartalmaznak matematikai függvényeket, string függvényeket, kimeneti és bemeneti függvényeket, stb.

8.2. Változók az *awk*ban

Az *awk* változók dinamikusak; akkor jönnek létre, amikor először használjuk őket. A változók értékei vagy *float* vagy *string* típusúak. Ezen túlmenően az *awk* programnyelvben nincs szükség típusegyeztetésre, mivel az *awk* a kontextustól függően határozza meg, hogy miként értelmezze egy adott változó típusát. Ha például egy változó értékének két

szám összegét adjuk, akkor a változó típusa szám lesz. Ha ezután meg akarjuk határozni, hogy az adott változó hány karakter hosszú, akkor az *awk* automatikusan string-gé konvertálja a változót, és utána adja vissza a z eredményt.

A tömbök indexei szögletes zárójelben ([és]) megadott kifejezések. Ha a kifejezés egy kifejezéslista (kif, kif ...) akkor a tömbindex az egyes kifejezések karakterlánc-értékének összefűzéséből álló karakterlánc, ahol az egyes részeket a `SUBSEP` változó értéke szerinti karakter választja el. Például:

```
i = "A"; j = "B"; k = "C"
x[i, j, k] = "hello, world\n"
```

a „`hello, world\n`” karakterláncot rendeli az `x` tömb „`A\034B\034C`” karakterlánccal indexelt eleméhez. Az *awk*ban minden tömb asszociatív, azaz karakterláncokkal indexelt.

8.3. Operátorok

A változók és konstansok közötti műveleteket operátorokkal végezhetünk. A *gawk*ban használható operátorokat a 8.1. táblázatban foglaltuk össze.

8.4. Vezérlő utasítások a *gawk*ban

A *gawk* vezérlő utasításainak összefoglalása a 8.2. táblázatban található.

8.4.1. Feltételvizsgálat

Az `if` utasítással egy feltételt vizsgálhatunk. Az utasítás szerkezete a következő:

```
if (feltétel) utasítás [ else utasítás ]
```

Ha a feltétel igaz, akkor az `if` utáni utasítás hajtódik végre, ha nem igaz, és adott az opcionális `else` vezérlőutasítás, akkor az `else` utáni utasítás következik.

8.4.2. A `while`-ciklus

A `while` ciklusnak két formája van:

```
while (feltétel) utasítás
do utasítás while (feltétel)
```

Mindkét esetben addig ismétli a `while`-ciklus az *utasítás* végrehajtását, amíg a feltétel igaz. Az első esetben a feltételt a ciklus elején vizsgáljuk, ezért lehetséges, hogy az *utasítás* egyszer sem hajtódik végre. A második esetben viszont a feltételt a ciklus végén vizsgáljuk, ezért az *utasítás* legalább egyszer végrehajtódik.

8.1. táblázat. Az *awk* operátorai csökkenő műveleti sorrend szerint.

Operátor	Leírás.
`(...)`	Csoportosítás. A matematikai zárójelezésnek megfelelően először a zárójelen belüli operátorok kerülnek kiértékelésre.
`$`	Mezőhivatkozás.
`++ --`	Inkrementálás és dekrementálás, azaz egy egész értékű változó növelése és csökkentése eggyel. Mindkettő prefix és postfix, azaz tehetjük a változó elé és mögé. Ha egy változó mögött van, akkor a műveleti sorrendben az utolsó helyre kerül.
`^`	Hatványozás (** szintén használható, **= pedig értékadó operátorként).
`+ - !`	Egyoperandusú plusz/mínusz és logikai tagadás.
`* / %`	Szorzás, osztás és maradékképzés.
`+ -`	Összeadás és kivonás.
space	Karakterláncok összekapcsolása (konkatenáció).
`< >` `<= >=` `!= ==`	A megszokott relációs operátorok: kisebb, nagyobb, kisebb-egyenlő, nagyobb-egyenlő, nemegyenlő és egyenlő. Figyeljünk arra, hogy két egyenlőségjel jelenti azt, hogy az egyenlőséget vizsgáljuk; egy zárójel értékadást jelent.
`~ !~`	Reguláris kifejezés illeszkedése, nem-illeszkedése. FONTOS: Ne használjunk konstans reguláris kifejezést (`/foo/`) `~` vagy `!~` baloldalán, csakis a jobbon! A `/foo/ ~ exp` kifejezés jelentése ugyanaz, mint a `(( $0 ~ /foo/) ~ exp)` kifejezésé. Rendszerint nem ezt várják.
`in`	Tömbhöz tartozás.
`&&`	Logikai ÉS.
`||`	Logikai VAGY.
`? :`	A C feltételes kifejezése. Ennek formája `kif1 ? kif2 : kif3`. Ha `kif1` igaz, a kifejezés értéke `kif2`, egyébként `kif3`. Csak egy értékelődik ki `kif2` és `kif3` közül.
`= += -=` `*= /=` `%= %^=`	Értékadás. Úgy az abszolút értékadás (var = value) mint az operátor-értékadás (a többi forma) egyaránt támogatott. Az operátor-értékadás jelentése az, hogy a változó aktuális értékével elvégezzük a operátor-értékadásban szereplő műveletet, majd változóhoz hozzárendeljük az eredményt.

8.4.3. A `for`-ciklus

A `for`-ciklus abban különbözik a `while`-ciklustól, hogy ebben az esetben meg lehet adni egy ciklusváltozót. A ciklusváltozó minden ciklusban automatikusan frissül. A `for`-ciklusnak is két formája van:

```
for (kif1 ; kif2 ; kif3) utasítás
for (var in tömb) utasítás
```

Az első esetben a ciklus előtt végrehajtódik a *kif1* kifejezés. Ezután mindaddig végrehajtódik az *utasítás*, ameddig *kif2* igaz. Minden ciklusban végrehajtódik a *kif3* kifejezés is, amely általában a ciklusváltozó frissítését végzi.

A második formában egy *var* változó egy adott tömb indexhalmazán fut végig, így az *utasításban* elérhetjük a tömb elemeit.

Ha a vezérlés a `break` utasításhoz ér, akkor a program azonnal kilép a legbelső ciklusból. Ha a program a `continue` utasításhoz ér, akkor felfüggeszti az *utasítás* futását, és a ciklusváltozó új értékével folytatja a program futtatását.

8.4.4. Tömbök törlése, és kilépés

Tömbök elemeit és magukat a tömböket is a `delete` paranccsal törölhetjük. A futó programból az `exit` paranccsal léphetünk ki.

8.4.5. Utasítások csoportosítása

Utasításblokkokat kapcsos zárójelekkel hozhatunk létre. Ez különösen hasznos amikor pl. egy ciklusban több utasítást akarunk végrehajtani.

8.5. Példák és feladatok

Gyakorló példák

Gy8.1. Írjunk egy *awk* szkriptet, amely egy bemenő adatfájl minden sorára megszámolja a pozitív és a negatív számokat, és a két számot soronként kiírja a képernyőre.

Gy8.2. A mellékelt adatfájlban hallgatói azonosító kódok és érdemjegyek vannak felsorolva. Írjunk egy *awk* szkriptet, amely megszámolja, hogy hány hallgató kapott ötöst, négyest, stb.

Feladatok

8.2. táblázat. A *gawk* vezérlő utasításai

```
if (feltétel) utasítás [ else utasítás ]
while (feltétel) utasítás
do utasítás while (feltétel)
for (kif1; kif2; kif3) utasítás
for (var in tömb) utasítás
break
continue
delete tömb[index]
delete tömb
exit [ kifejezés ]
{ utasítások }
```

F8.1. Írjunk egy *awk* szkriptet, amely megkeresi egy adatfájlban a legnagyobb és a legkisebb elemet, és kiírja a képernyőre.

F8.2. Írjunk egy *awk* szkriptet, amely egy adott adatfájlra kiszámolja a *sorok* átlagát.

F8.3. Íjunk egy *awk* szkriptet, amely adott gyakoriságeloszlásból kiszámolja a relatív gyakoriságokat. (Útmutatás: olvassuk be a sorokat egy tömbbe, és közben számítsuk ki a normálófaktort. Az adatfájl beolvasása után fussunk végig a tömbön, és írassuk ki a normált elemeket.)

F8.4. Írjunk egy *awk* szkriptet, amely kiírja a képernyőre a

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} \tag{8.1}$$

sorösszeg első N elemét addig, ameddig a *gawk* beépített exp() függvényéből számított értéktől vett eltérés kisebb nem lesz, mint 10^{-6}. Hány tagot kellett kiíratni?

F8.5. Tegyük fel, hogy nem létezik sem a `tail` sem a `head` parancs a rendszerünkön, csak *awk* áll rendelkezésünkre. Implementáljuk ezeket, a lehető leghatékonyabban! (Tegyük fel, hogy terrabájt nagyságú állományokon kellene ezt használni, gondoskodjunk róla, hogy fusson ilyen méretek mellett is a szkriptünk.)

9. fejezet

Linux Shell parancsok

A Linux operációs rendszer egyik jellegzetessége, hogy igen nagy szabadságot ad a programok bemenő és kimenő adatainak kezelésében. Az éppen futó feladatok ki és bemenetei össze is kapcsolhatóak (pipe), illetve a bemenet és kimenet tetszőleges eszközre átirányítható.

A Linux nyílt forráskódú operációs rendszer, aminek következtében se szeri se száma az elérhető Linux parancsoknak. Ráadásul mindegyiknek vannak kapcsolói, melyekkel egy-egy parancs hatását lehet módosítani. Természetesen az ilyen nagyszámú parancs használata, illetve fejben tartása már igen körülményes lehet, ezért a Linux rendszerek része egy beépített kézikönyv (ld. a 9.3.1. fejezetben a `man` parancsot).

9.1. Shell olvasnivalók

Az alábbi néhány forrás segítségével már el lehet indulni a Linux shell parancsok megismerésének útján.

- Bevezetés a UNIX rendszerekbe egyetemi jegyzet[13],
- Bevezetés a LINUX használatába[14],
- Linux Man oldala magyarul.

9.2. A Linux shell

A Linux shell egy olyan interaktív program, amely egyszerre tölti be a parancsértelmező (command interpreter) és az adatközvetítő szerepét. A shell-t közvetlenül, interaktívan a korábbi gyakorlatokon megszokott módon, a terminálon keresztül érhetjük el. A shell ún. prompttal jelentkezik be, ami általában a $ jel, ez után lehet a parancsokat begépelni. A parancsokat általában interaktív módon, egy sorban adjuk be: először a parancsot,

majd az esetleges módosító kapcsolókat és argumentumokat. A parancsot az `<Enter>` billentyűvel zárjuk. Lehetőség van a shell parancsokat nem-interaktív módon is meghívni, ha a parancsokat egy fájlba írjuk. Az ilyen fájlokat shell szkriptnek hívják.

Vigyázzunk, mert a Linux rendszer megkülönbözteti a nagy és kisbetűket! A parancsok nagy részét kisbetűvel kell írni.

9.2.1. Shell gyorsbillentyűk

A Linux shell hatékony használatához mindenképpen érdemes néhány hasznos gyorsbillentyűt megjegyeznünk (ld. 9.1. táblázat).

A parancs végrehajtásának megszakítása

Bizonyos esetekben szükség lehet a parancs leállítására annak futása közben: például ha egy parancs már túl hosszú ideje fut, vagy közben eszünkbe jut, hogy nem ezt az utasítást akartuk futtatni. A parancs leállítására a `Ctrl` és `C` (röviden `<Ctrl-C>`) billentyűk egyidejű lenyomása szolgál. Ezzel megszakítjuk a parancs futását, és visszakapjuk az eredeti shell promptot.

A parancs végrehajtásának felfüggesztése

Lehetőség van arra is, hogy egy parancs futását ne megszakítsuk, hanem csak felfüggesszük (suspend). A parancsok felfüggesztésére a `<Ctrl-Z>` billentyűkombináció szolgál. A felfüggesztés után szintén visszakapjuk a promptot, de a megszakítástól eltérően a felfüggesztett programok bármikor tovább futtathatóak. A felfüggesztett programok folytatásához a `fg` és a `bg` parancsok szolgálnak attól függően, hogy az előtérben *(foreground)* vagy a háttérben *(background)* kívánjuk folytatni a programot.

Parancskiegészítés

Az egyik leghasznosabb shell szolgáltatás a parancsok és fájlnevek dinamikus kiegészítése. Ha beírjuk egy parancs vagy fájlnév első néhány betűjét, akkor a `<Tab>` billentyű megnyomására a shell megpróbálja az addig begépelt szót kiegészíteni. Ha egyértelmű a kiegészítés, tehát ha pontosan egy parancs vagy fájlnév kezdődik a megadott módon, akkor a shell beilleszti a kiegészített szót. Ha több lehetőség is van, akkor először csak egy hangjelzést kapunk, a második `<Tab>` leütésére pedig egy listát is a lehetséges kiegészítésekről.

Naplózás

Egy másik igen hasznos szolgáltatás a korábban kiadott parancsok megjegyzése. A „fel” billentyűvel (⇑) az eggyel korábbi, a „le” billentyűvel (⇓) pedig az eggyel későbbi bevitt

<Ctrl-C>	parancs megszakítása
<Ctrl-Z>	parancs felfüggesztése
<Ctrl-R>	parancs keresése a history-ban
⇑	az előző parancs a history-ban
⇓	a következő parancs a history-ban
<Tab>	parancs kiegészítése

9.1. táblázat. A legfontosabb shell gyorsbillentyűk.

parancssor hívható elő. Ezen túl lehetőség van arra is, hogy a korábban kiadott parancsok között keressünk. Ehhez üssük le a <Ctrl-R> billentyűket, és gépeljük be azt a szövegrészletet, amit meg szeretnénk keresni. Ha megtaláltuk a keresett parancssort, akkor azt az <Enter>-rel ismét lefuttathatjuk, illetve a bal (⇐) vagy jobb (⇒) billentyű megnyomása után szerkeszthetjük.

9.2.2. A parancsok szintaxisa

A Linux/Unix parancsok szintaxisa, néhány kivételtől eltekintve, egységes. Elsőként a parancs nevét kell megadni. Ezt követhetik opcionálisan a parancs kapcsolói, majd a parancs argumentuma(i). A parancsnevet, a kapcsolókat illetve az argumentumokat szóközök választják el egymástól.

9.2.3. Az utasítások kapcsolói

A kapcsolók az alaputasítás végrehajtását módosítják. Kétféle kapcsoló létezik: a rövid és a hosszú kapcsoló.

A rövid kapcsoló

A rövid kapcsolók egy mínusz (-) jellel kezdődnek és egy betűből állnak. Szinte minden utasítás rendelkezik rövid kapcsolóval, de vigyázzunk arra, hogy ugyanaz a kapcsoló más más parancs esetén teljesen különböző jelentéssel bírhat.

A kapcsolókat az utasítástól illetve annak argumentumaitól szóköz választja el. Ha az egyes kapcsolókat külön-külön adjuk meg, akkor azok között szintén szóköznek kell állnia, és minden kapcsoló elé külön-külön mínusz jelet kell tenni. Lehetőség van a rövid kapcsolók összevonására is. Ilyenkor a mínusz jelet csak egyszer kell begépelni, és utána szóközök *nélkül* kell a kapcsolókat megadni.

A következő példákban egy feltételezett felhasználói könyvtárban nézzük meg az állományok listáját: Az `ls` parancs az aktuális könyvtárban lévő állományok felsorolását adja:

```
$ ls
```

```
Asztal        examples.desktop  Képek       Nyilvános  Videók
Dokumentumok  Letöltések  Sablonok   Zenék
```

A `-l` kapcsoló módosítja az `ls` utasítás alapértelmezését, és állományok részletes listáját írja ki (ami oszloponként a következőket tartalmazza: hozzáférési státusz, link szám, a fájl birtokosa (felhasználó és csoport), méret byte-ban, az utolsó változtatás ideje és végül a fájl neve):

```
$ ls -l
összesen 40
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Asztal
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Dokumentumok
-rw-r--r-- 1 usernev usergroup  167 2009-11-20 15:50 examples.desktop
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Képek
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Letöltések
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Nyilvános
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Sablonok
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Videók
drwxr-xr-x 2 usernev usergroup 4096 2009-11-20 19:56 Zenék
```

A `-a` kapcsoló a könyvtárban lévő összes állományt kilistázza, beleértve a ponttal (`.`) kezdődőeket is, amelyek általában rejtve maradnak. Az `-a` és az `-l` kapcsolók kombinálhatóak is: az `ls -a -l`, vagy az `ls -al`) parancsok kiadásával. Ekkor mindkét kapcsoló hatása érvényes lesz, azaz az *összes fájlt részletesen* kilistázzuk.

Figyeljünk rá, hogy amennyiben mindkét kapcsoló elé írunk - jelet, akkor szóköz karaktert is tegyünk közéjük:

```
$ ls -a-l
ls: érvénytelen kapcsoló
```

helytelen szintaxis, míg a következő helyes módon megadott parancs:

```
$ ls -a -l
összesen 172
drwxr-xr-x 31 usernev usergroup 4096 2009-11-21 21:42 .
drwxr-xr-x  3 root root 4096 2009-11-20 15:50 ..
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Asztal
-rw-r--r--  1 usernev usergroup  220 2009-11-20 15:50 .bash_logout
-rw-r--r--  1 usernev usergroup 3180 2009-11-20 15:50 .bashrc
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Dokumentumok
-rw-r--r--  1 usernev usergroup  167 2009-11-20 15:50 examples.desktop
drwx------  4 usernev usergroup 4096 2009-11-20 19:56 .gconf
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Képek
```

```
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Letöltések
drwx------  4 usernev usergroup 4096 2009-11-20 20:24 .mozilla
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 .nautilus
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Nyilvános
-rw-r--r--  1 usernev usergroup 4096 2009-11-20 15:50 .profile
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Sablonok
drwx------  2 usernev usergroup 4096 2009-11-20 19:56 .update-notifier
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Videók
-rw-------  1 usernev usergroup 7882 2009-11-21 21:41 .viminfo
drwxr-xr-x  2 usernev usergroup 4096 2009-11-20 19:56 Zenék
```

Hosszú kapcsolók

A hosszú kapcsolók abban különböznek a rövid kapcsolóktól, hogy két mínusz jellel kezdődnek (`--`), és általában több karakterből álló értelmes angol kifejezések. Vannak olyan hosszú kapcsolók, amelyek pontosan megfelelnek egy rövid kapcsolónak (pl. az `ls` parancs `-a` és `--all` kapcsolói ekvivalensek). Vannak azonban olyan rövid és hosszú kapcsolók is, amelyeknek nincs hosszú illetve rövid változatuk.

A hosszú kapcsolókat nem lehet a rövid kapcsolókhoz hasonlóan összevonni.

9.2.4. Az utasítások argumentumai

A kapcsolók mellett a Linux operációs rendszer utasításainak nagy része paramétereket is elfogad. Amíg a kapcsolók módosítják az adott parancs hatását, addig a paraméterek határozzák meg, hogy mire vonatkozik az utasítás. A parancs után először a kapcsoló(ka)t, majd a paramétereket adjuk meg szóközökkel elválasztva. Általában több paraméter is megadható. Például az `echo` parancs hatására a parancs argumentuma megjelenik a képernyőn.

```
$ echo haliho
haliho
```

ahol az „`echo`” a parancs és a „`haliho`” a parancs által megjelenített argumentum.

9.3. Linux parancsok

Egy teljes Linux környezetben kb. 100 shell parancs és több ezer futtatható program van. Az összes parancs és kapcsoló megtanulása nyilvánvalóan reménytelen feladat. Ebben a szakaszban összegyűjtöttük a legfontosabb Linux parancsokat, amelyek mindenképpen fontos megismernünk.

9.3.1. Segítségkérés

A `man` parancs

A Linux operációs rendszerben minden parancsról részletes leírást kaphatunk a `man` (*manual*) parancs használatával. A `man` parancs után megadva a kérdéses utasítás nevét megkapjuk az adott utasítás leírását. Például a

```
$ man man
```

paranccsal magáról a `man` parancsról kérhetünk leírást. Ha egy kulcsszót szeretnénk keresni a kézikönyvben, akkor a `-f` kapcsoló után megadva azoknak a fejezeteknek a rövid listáját kapjuk meg, amiben a keresett kulcsszó szerepel.

A `man` parancs nagyon fontos ahhoz, hogy eligazodjunk a Linuxban, ezért tekintsük át egy manual oldal felépítését. Minden oldal különböző szakaszokra van bontva. Az első szakasz, a `NÉV` (`NAME`) mindig kötelező. Ebben található a parancs neve, és egy egysoros rövid leírás. A következő szakasz általában az `ÁTTEKINTÉS` (`SYNOPSIS`), amelyben a parancs hívásának változatai vannak felsorolva a lehetséges kapcsolókkal.

Az `ÁTTEKINTÉS` szakaszban az alábbi konvenciók érvényesek, melyek iránymutatók a többi szakaszra is:

`kövér`	pontosan a megadott szerint írandó.
`dőlt`	a megfelelő kifejezéssel helyettesítendő. A szöveges terminálon a dőlt szedés helyett lehetséges `aláhúzott` is.
`[-abc]`	bármelyik kifejezés a `[ ]`-n belül opcionális. A szögletes zárójeleket nem kell bevinni.
`-a|-b`	a `|`-vel elválasztott kapcsolók együtt nem használhatóak.
`argumentum` `...`	a *`argumentum`* megismételhető
`[`*`kifejezés`*`] ...`	a *`kifejezés`* a `[ ]`-n belül megismételhető.

Ezután következik általában egy hosszabb `LEÍRÁS` (`DESCRIPTION`) a parancsról, valamint a `KAPCSOLÓK` (`OPTIONS`) részletes ismertetése.

Keresés a kézikönyvekben

Minden manual oldalnak van egy rövid leírása. Az `apropos` parancs a parancsnevekben és a rövid leírásokban keres `kulcsszavakat`.

9.3.2. Állományok kezelése

Listázás

Az állományok kilistázására az `ls` parancs szolgál. A lista formátumát sokféle kapcsolóval módosíthatjuk.

Példák:

```
$ ls
Asztal        examples.desktop  Képek       Nyilvános  Videók
Dokumentumok  Letöltések  Sablonok   Zenék
```

Leellenőrizzük, hogy az adott `examples.desktop` fájl megtalálható-e az adott könyvtárban:

```
$ ls examples.desktop
examples.desktop
```

Ha az adott fájl nincs benne a könyvtárban, akkor hibaüzenetet kapunk:

```
$ ls probafile
ls: probafile nem érhető el: Nincs ilyen fájl vagy könyvtár
```

A parancs a `-l` kapcsoló hatására kiírja az állományok, ill. könyvtárak (könyvtár esetén az első betű `d`, míg közönséges állománynál `-`) hozzáférési jogait, a rájuk vonatkozó kapcsolatok számát, a tulajdonosuk nevét és felhasználói csoportját, a hosszát byteokban, az utolsó módosításuk idejét, végül a nevét:

```
$ ls -l
```

A parancs a `-a` kapcsoló hatására kiírja az aktuális könyvtárban lévő összes fájlt, beleértve a rejtett, azaz ponttal kezdődő nevű állományokat is:

```
$ ls -a
```

Másolása

Állományokról másodpéldány készítésére a `cp` parancs szolgál. Például a

```
$ cp gyakorlas masolat
```

a `gyakorlas` állomány tartalmát átmásolja a `masolat` nevű állományba.

Vigyázzunk a `cp` parancs használatakor, mivel a másolás során — egy régebbi állományra való másolással — könnyen megsemmisíthetünk adatokat!

A `cp` parancsnak egyszerre több állománynevet is megadhatunk, sőt a shell ún. kifejtő mechanizmusa révén helyettesítő karaktereket is használhatunk. A parancs így használható pl. egy könyvtár tartalmának átmásolására egy másik könyvtárba. Például a

```
$ cp * masikkonyvtar
```

átmásolja az aktuális könyvtárban lévő összes állományt a `masikkonyvtar` nevű alkönyvtárba. A parancs a könyvtárakat nem másolja.

Könyvtárakat a teljes alkönyvtárrendszerükkel együtt a `-r` kapcsoló segítségével másolhatunk át. Például a

```
$ cp -r /usr/user /tmp/user/
```

átmásolja a `/usr/user` könyvtár összes állományát és az abból nyíló összes alkönyvtárat a `/tmp/user` könyvtárba. Ezáltal a teljes `/usr/user` könyvtárban lévő állománystruktúra megjelenik a `/tmp/user` könyvtárban is.

A `cp` utasítás – ugyanúgy, mint a többi állománykezelő parancs – hibajelzéssel áll meg, ha nincs jogosultságunk arra, hogy az adott helyre írjunk.

Mozgatás, átnevezés

Állományok átnevezésére vagy átmozgatására az `mv` parancs szolgál. A parancs kérdezés nélkül felülírja a már létező állományokat. Ha meg szeretnénk tartani a fájl eredeti nevét más könyvtárba való másolásnál, akkor elegendő, ha az `mv` utasítás céljaként a könyvtárat adjuk meg. Például a

```
$ mv level /tmp/test
```

átmozgatja a `level` állományt az aktuális könyvtárból a `/tmp/test` könyvtárba az eredeti `level` néven.

Az `mv` parancsnak — a `cp` parancshoz hasonlóan — több állománynevet is megadhatunk, de ilyenkor a legutolsó argumentumnak (a célnak) könyvtárnak kell lennie.

Törlés

Állományokat az `rm` paranccsal törölhetünk. Egyszerre több állományt is megadhatunk, illetve akár helyettesítő karaktereket is használhatunk. Az `rm` parancs kérdezés nélkül töröl minden megadott állományt.

A `-r` kapcsoló rekurzív törlést eredményez, azaz a megadott könyvtártól lefelé a teljes alkönyvtár rendszerben sor kerül a törlésre, az utasítás még a könyvtárakat is kitörli. A

```
$ rm -r *
```

a Linux egyik „leghatásosabb” parancsa, az adott könyvtártól kezdve felszámolja a teljes állománystruktúrát, és még az alkönyvtárakat is kitörli! Csak akkor adjuk ki, ha valóban törölni akarunk mindent!

9.3.3. Állományok tartalmának megjelenítése

Sokszor szükségünk van az állományok tartalmának megjelenítésére a képernyőn. Erre a célra ugyan szövegszerkesztőket is használhatunk (pl. `vim`-t), de sokszor hatékonyabb a `cat`, `head`, vagy `tail` parancsok használata.

A `cat` parancs

Kisebb állományok megjelenítésére a `cat` parancsot használjuk az állomány(ok) nevével, mint paraméterrel. Egyszerre több állománynév is megadható, ilyenkor a rendszer a megadott sorrendben jeleníti meg az állományok tartalmát. A paraméterlistában használhatjuk a shell helyettesítő karaktereit, a csillagot (*), amely egy tetszőleges stringet helyettesít, vagy a kérdő jelet (?), amely egy tetszőleges betűt helyettesít.

```
cat gyakorlofile.txt
```

A `head` parancs

A `head` a megadott fájlok első részét (alapértelmezésben első 10 sorát) írja ki. Leggyakrabban használt kapcsolója segítségével megadhatjuk, hogy a bemeneti fájl hány sorát írja ki:

```
$ head -n 5 gyakorlofile.txt
```

a fájl első 5 sorát írja ki. Megjegyezzük, hogy a `head -5 gyakorlofile.txt` parancs is működik az `n` kapcsoló kiírása nélkül is.

A `tail` parancs

A `tail` a megadott fájlok végét (alapértelmezésben az utolsó 10 sorát) írja ki. Leggyakrabban használt kapcsolója segítségével megadhatjuk, hogy a bemeneti fájl hány utolsó sorát írja ki:

```
$ tail -n 5 gyakorlofile.txt
```

a fájl utolsó 5 sorát írja ki. Megjegyezzük, hogy a `tail -5 gyakorlofile.txt` parancs is működik az `n` kapcsoló kiírása nélkül is.

9.3.4. Szöveg keresése: a `grep` parancs

A `grep` paranccsal gyorsan megkereshetünk egy adott szöveget (karaktermintát) az állományokban. A keresett minta lehet egy szó, vagy betűk és számok csoportja csoportja, vagy akár reguláris kifejezés (ld. 4.5. szakasz). Ha a `grep` parancs megtalálja a mintát, akkor a mintát tartalmazó sort kiírja a képernyőre. A parancs használatához meg kell adni a keresett karaktersort és azt az állományt (vagy állományokat), amely(ek)ben keresni akarunk. Ha a karaktersor több szóból áll, vagy szóköz(öke)t tartalmaz, akkor aposztrófok közé kell tenni (nem tévedhetünk, ha mindig aposztrófok közé tesszük a keresett szót). Példák:

```
$ grep ebben gyakorlofile.txt
Most mindent együtt szerepeltetunk ebben a sorban 'vege' $1 * 42
```

A parancs kilistázza azt a sort a `gyakorlofile.txt` nevű fájlból, amelyben az „ebben” szó szerepel. Természetesen a `grep` parancs is megkülönbözteti a keresett karaktersorban a kis és nagybetűket, ezt a különbségtételt azonban a `-i` kapcsolóval kikapcsolhatjuk:

```
$ grep -i ebben gyakorlofile.txt
Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42
Ebben a sorban nehany furcsa karakter van % @ !
Most mindent egyutt szerepeltetunk ebben a sorban 'vege' $1 * 42
```

A `grep` parancs alapesetben kiírja a képernyőre az állományok azon sorait, amelyekben a keresett karaktersorozatot megtalálta. Amennyiben a `-n` kapcsolót megadjuk, a sorok előtt azok sorszáma is megjelenik.

Néha arra van szükség, hogy csak a állományok nevét kapjuk meg, amelyekben ez a minta előfordul. Ilyenkor a `-l` kapcsolót használjuk.

9.4. Könyvtárak használata

A könyvtárak olyan speciális állományok, amelyek más állományok szervezésére és csoportosítására használatosak. A Linux operációs rendszer könyvtárai egy hierarchikus fastruktúrát alkotnak.

A Linux könyvtárrendszerének tetején helyezkedik el a gyökér könyvtár, amit a `/` jel jelöl. Innen kiindulva minden fájl elérhető a rendszerben. Az elérési utat *path*-nak (ösvény, útvonal) hívjuk. A könyvtárak más könyvtárakat vagy állományokat tartalmazhatnak. Minden könyvtárnak van szülő könyvtára, amelyik a struktúrában felette álló könyvtár. Az egyedüli kivétel a gyökér (/) könyvtár.

A felhasználó `HOME` könyvtára és a hozzá tartozó fájlok és az alkönyvtárrendszer csak egy kis részét alkotja a Linux teljes könyvtárrendszerének. A `HOME` könyvtár az a könyvtár, amibe a felhasználó bejelentkezés után automatikusan bekerül. A `HOME` könyvtár alatt a felhasználó létrehozhatja saját könyvtárrendszerét.

Az éppen használt könytár gyökér könyvtárhoz viszonyított útvonalat lekérdezhetjük a

```
$ pwd
/home/usernev
```

paranccsal.

9.4.1. Abszolút és relatív útvonal

Az útvonal megadásánál kétféle leírást használhatunk. Az abszolút útvonal-név megadásakor a gyökér (/) könyvtárból indulunk és leírjuk a teljes útvonalat. Ilyen útvonal lehet pl. a `/usr/local/bin`.

A relatív útvonal-név az éppen aktuális könyvtárból adja meg az elérési útvonalat. Például ha a `usernev` felhasználó a `HOME` könyvtárában van benn, akkor a `Mail/incoming` adja meg a `Mail` alkönyvtárbeli `incoming` állomány elérését.

Minden könyvtárban használhatjuk a `".."` jelölést, ami egy olyan speciális állomány, ami az egy szinttel feljebb levőkönyvtárra mutat. Amennyiben a felettünk lévő könyvtárból nyíló alkönyvtárra akarunk hivatkozni, akkor a `"../"` után írjuk be az alkönyvtár nevét. Például, ha a `/home/usernev1/proba` könyvtárban vagyunk benn, akkor a `../../usernev2/jegyzet` adja meg a `/home/usernev2/jegyzet` könyvtár elérését.

9.4.2. A könyvtárrendszer használata

A könyvtárak között a `cd` paranccsal lehet mozogni. Ha egy könyvtárból annak egy Alkönyvtárába mozgunk akkor elég a `cd alkönyvtár` parancsot kiadni. Ha más könyvtárba mozgunk, ki kell írni a teljes elérési útvonalat. Két szinttel feljebb pl. a

```
$ cd ../..
```

paranccsal juthatunk.

9.4.3. Könyvtárak létrehozása és törlése

Könyvtárakat a `mkdir` paranccsal hozhatunk létre. Argumentumként akár több könyvtárnevet is megadhatunk. Például a:

```
$ mkdir test test/artur /u/lovag/galahad/tmp
```

létrehozza az aktuális könyvtárból nyíló `test`, az abból nyíló `artur`, valamint a bejelentkezési könyvtár (itt a `/u/lovag/galahad`) alatti `tmp` könyvtárakat.

Természetesen nem hozhatunk létre könyvtárat egy "normális" fájl alatt. Ugyanígy sikertelen a parancs, ha nincs engedélyünk alkönyvtár létrehozására (beleírására) az adott könyvtárban. Ha egy idő után nem lesz többé szükség egy Könyvtárra, és célszerű azt kitörölni, akkor erre a `rmdir Könyvtár` parancsot használhatjuk. Az `rmdir` parancs a `mkdir` parancshoz hasonlóan több argumentumot is elfogad. Amennyiben olyan könyvtárat akarunk kitörölni, amelyik nem üres, az operációs rendszer hibajelzést ad. Ilyenkor — kellő óvatossággal — használhatjuk az `rm -r` parancsot, ami a törli az adott könyvtár teljes tartalmát, majd végül magát a könyvtárat is.

9.5. Ki- és bemenetek átirányítása, különleges shell parancsok

A shell program fogadja a parancsokat és – részben átalakítva – továbbadja azokat az egyes speciális programoknak. A shell-en belül megteremtették a programok összekapcsolásának lehetőségét is, mivel általában minden program innen indul el.

Az átirányítás lehetővé teszi, hogy egy program végeredményét egy másik program bemenetként használja, külön állomány létrehozása nélkül. Egy parancssorban több átirányítást is használhatunk, pl. az első utasítás a bemenetét egy megadott állományból veszi, majd a második parancs az első kimenetét (eredményét) dolgozza fel, végül a harmadik a második kimenetét továbbalakítva a végeredményt egy kimeneti állományba írja. Mindez nagy közbülső állományok létrehozása nélkül történik, ezért sokkal gyorsabb a parancsok egyenkénti futtatásánál.

9.5.1. Parancs kimenetének átirányítása

Egy program végeredményét egy állományba is beleírhatjuk a képernyő helyett. Ehhez a parancs után egy `>` jelet, majd a kívánt fájl nevét kell írnunk. Például a

```
$ ls -l > filelista
$ cat filelista
```

parancsok hatására az `ls` parancs kimenet egy `filelista` nevű állományba kerül, amit aztán a `cat` parancs segítségével írunk a képernyőre.

Az átirányítás felülírja a már létező állományt. Ha ehelyett az állomány végéhez akarjuk az eredményt fűzni, akkor a `>>` jelet kell használnunk. A shell ekkor ellenőrzi, hogy létezik-e a megadott nevű fájl. Ha igen, akkor a végéhez fűzi az új adatokat, ha pedig nem létezik a fájl, akkor létrehozza azt. Az előző példát folytatva:

```
$ ls > filelista
$ ls -l >> filelista
```

Az első `ls` felülírja a fájllistát, míg az `ls -l` a végéhez fűzi a részletes állománylistát.

9.5.2. A cső (pipe) használata

A Linuxban lehetőség van arra, hogy közbülső fájl létrehozása nélkül kapcsoljunk össze két (vagy több) programot. Ilyenkor az első program kimenete a második bemenete lesz. A két összekötni kívánt programot az ún. pipe jellel (ez a `|`) kell a parancssorban elválasztani. Egyszerre több parancsot is összekapcsolhatunk a `|` jellel, ezt pipeline-nak (csővezeték) nevezzük.

Példaként listázzuk ki a `/bin` könyvtár tartalmát az `ls` parancs segítségével és a könyvtárban lévő fájlok számát a `wc` segítségével megjelenítjük a képernyőn:

```
$ ls -l /bin | wc -l
```

9.6. Példák és feladatok

Feladatok

F9.1. Hozzuk létre a `szamalap-9` nevű könyvtárat a felhasználói területünk gyökerében[1], és mozgassuk át ebbe a `Munkaasztal` könyvtárban található összes állományt.

F9.2. Írassuk ki a `/etc/fstab` állomány első olyan sorát, amely tartalmazza az `ext3` kifejezést.

F9.3. Listázzuk ki a `/proc/meminfo` fájl első 10 sorát egy az F9.1. feladatban készített könyvtár alatti új fájlba, majd a terminálra írassuk ki az `M` betűvel kezdődő sorokat.

F9.4. Írassuk ki a `/proc/cpuinfo` fájlból a számítógépében található processzormodell nevét, majd számoljuk meg ez hány szóból áll.

F9.5. A `touch` parancs manuál oldalai alapján állítsuk át egy tetszőleges fájl létrehozásának dátumát 2001. január 1.-re. Ellenőrizzük, hogy sikeres volt-e a módosítás.

F9.6. Listázzuk ki a `/usr` könyvtárban a fájlok méretét

1. rekurzívan,
2. SI egységekben, és
3. a fájlok mérete szerint csökkenő sorrendben.

Az eredményt írassuk be egy fájlba.

F9.7. Keressük meg azt a parancsot, amely egy számtani sorozatot ad vissza. Listázzuk ki ezzel a paranccsal egy sorban a páros számokat egy és száz között.

F9.8. Adjuk meg mely parancsokkal kereshetjük meg azt az alkönyvtárat, amely a legmélyebben található a `/usr` könyvtárban.

F9.9. Készítsük el a `/usr` könyvtárban található összes fájl kumulatív eloszlását, és ábrázoljuk log-log skálán az adatokat *gnuplot*tal. Illesszünk hatványfüggvényt egy megfelelő tartományon. Az eredményekről készítsünk LaTeX jegyzőkönyvet.

F9.10. Értelmezzük és találjuk ki, hogy mit csinál az alábbi shell szkript:

[1] azaz a *home*-könyvtárban

```
#!/bin/bash

R=~/szamalap-9

if [ ! -d $R ] ; then
 echo ejnye-bejnye >&2
 exit 1
fi

T=‘mktemp -d --tmpdir=$R‘
cat /proc/cpuinfo > $T/cpu
grep bogo $T/cpu | tee $R/.cpu
```

10. fejezet

Haladó shell parancsok

Ezen a gyakorlaton az előző gyakorlat anyagát bővítjük. A jegyzetben bemutatunk néhány hasznos programot is. A programokról nem közlünk azonban részleteket, mivel mindegyik bemutatott programról részletes leírást találunk a manuál oldalakon.

10.1. Speciális karakterek

A shell rendelkezik néhány olyan karakterrel, amelynek speciális jelentése van. Ezek közül néhánnyal már korábban is tárgyaltunk (pl. `|`, `&`), illetve különösebb magyarázat nélkül használtunk is (pl. **space**, **newline**). Mielőtt részletesebben is áttekintjük az egyes speciális karakterek használatát, elsőként rögzítsük az alábbiakban előforduló kifejezések jelentését:

Üres hely Az üres helyeknek – angolul *blank* – a **space** és **tab** karaktereket illetve ezek láncolatát nevezzük.

Szó Szavaknak – angolul *word* vagy *token* – olyan karaktersorozatot nevezünk, amelyet a shell egy egységként értelmez.

Név/Azonosító A név – angolul *word* vagy *identifier* – olyan **szó**, amely kizárólag alfanumerikus karakterekből vagy alulvonásból áll, és amely betűvel vagy alulvonással kezdődik.

Metakarakter A metakarakter – angolul *metacharacter* – olyan karakter, amelyet ha nem teszünk idézőjelek közé, akkor elválasztja a **szavakat**. Ezek az alábbiak lehetnek:

`|` `&` `;` `(` `)` `<` `>` **space** **tab**

Vezérlő operátor A vezérlő operator – angolul *control operator* – olyan token, amely lezárja illetve elválasztja az egyszerű parancsokat. Az alábbiak lehetnek:

`||` `&` `&&` `;` `(` `)` `|` `|&` **newline**

Látható, hogy bizonyos karaktereknek többféle jelentése is lehet a kontextustól függően.

Az üres karaktereknek (**space**, **tab**) nincs különösebb jelentésük azon kívül, hogy elválasztják a szavakat.

A `|`, `<` és a `>` karakterek a pipe-ra, valamint standard bemenet és kimenet átirányítására szolgálnak (ld. 9. fejezet). A `&` karakter módosíthatja ezeknek a metakaraktereknek a jelentését, a részletes leírás a *bash* manuál oldalán található.

A `;` metakarakter szolgál például a shell `for` ciklusokban az egyes kifejezések elválasztására.

10.1.1. Egyszerű parancsok

Az *egyszerű parancsok* opcionális változó-értékadások sorozatából, valamint az azt követő üres karakterekkel elválasztott szavakból és átirányításokból állnak, melyet egy vezérlő operátor zár. Például

```
user@host:~$ PATH=/home/username/bin myprog arg1 > datafile &
```

Itt a `PATH=/home/username/bin` a path változó értékét ideiglenesen átállítja az adott könyvtárra, majd végrehajtja a `myprog` parancsot egy `arg1` argumentummal. A program standard kimenetét a `>` átirányítja a `datafile` fájlba. A parancsot lezáró `&` vezérlő operátor végül a háttérbe helyezi a programot.

10.1.2. Összetett parancsok

Az egyszerű utasításokból igen bonyolult parancsokat építhetünk. Ennek egyik módja a pipe-ok (`|`) használata, amikor a első program standard kimenetét átirányítjuk a következő program standard bemenetére. A `|&` operátorral a standard hiba kimenetéből készíthetünk pipe-ot.

Listákat is létrehozhatunk egy vagy több összefűzött, `;`-vel, `&`-tel, `&&`-tel vagy `||`-sal elválasztott pipe-pal. Ha egy parancsot a `&` vezérlőoperátor zár le, akkor a shell a parancsot a háttérben hajtja végre, azaz a shell nem vár a parancs befejezésére, hanem azonnal visszaadja az irányítást. A `;` vezérlőjellel elválasztott parancsok egymás után szekvenciálisan hajtódnak végre.

ÉS és VAGY listáknak azokat nevezzük, amikor egy vagy több pipe-ot `&&` vagy `||` karakterek választanak el egymástól. Ezek a listák balról jobbra hajtódnak végre, és a következő a jelentésük:

parancs1 && *parancs2*

esetén *parancs2* pontosan akkor hajtódik végre, ha *parancs1* 0 értékkel (azaz hibátlanul) tér vissza;

parancs1 || *parancs2*

esetén pedig pontosan akkor, ha *parancs1* nem 0 értékkel (azaz hibával) tér vissza.

Ezekkel a vezérlőoperátorokkal logikai műveleteket hajthatunk végre. Például

```
user@host:~$ latex valami.tex && dvips valami.dvi
```

esetén a `dvips` parancs csak akkor hajtódik végre, ha a `latex` parancs sikeresen lefutott.

10.1.3. Megjegyzések

Megjegyzéseket a #-jellel tehetünk a shell szkriptekbe. A #-jeltől a sor végéig figyelmen kívül hagyja a shell a parancsokat. (Tipp: Ha egy hosszú, de félkész parancsot már begépeltünk, és eszünkbe jut, hogy pl. egy másik parancsot le kell futtatnunk, akkor illesszünk be egy #-jelet a sor elejére. Ha `<Enter>`-t ütünk, akkor a shell nem hajtja végre az utasítást, viszont megjegyzésként bekerül a hisztory-ba. Ha lefuttattuk a kívánt programot, keressük ki a beütött parancsot a history-ból, és vegyük ki a megjegyzést jelet.)

10.1.4. Idézőjelek

Néha szükségünk lehet arra, hogy a shell figyelmen kívül hagyjon egyes speciális jelentésű karaktereket, például ha olyan fájlnevet akarunk megadni, amelyben van szóköz. A karakterek „szó szerinti” jelentésének eléréséhez háromféle lehetőségünk van:

1. Az „idézőjeltelen” \ karakter megszünteti az azt követő karakter speciális jelentését.
2. A szimpla idézőjel közti karakterek megőrzik szó szerinti jelentésüket. Szimpla idézőjelek között nem állhat szimpla idézőjel még akkor sem, ha \ előzi meg.
3. A dupla idézőjelek között a karaktereknek szó szerinti jelentésük lesz, kivéve a $, a \, a ‘, és a ! karaktereket. A $ és a ‘ megőrzik speciális jelentésüket. A \ csak a $, ‘, ", \ és a **newline** karakterek előtt tartja meg speciális jelentését. Dupla idézőjel között állhat dupla idézőjel, ha egy \-jel áll előtte.

10.1.5. Változók

A shell változóinak értékét a $ jellel érhetjük el. A shellnek számos beépített változója van, de természetesen mi is létrehozhatjuk később. A beépített változók közül a legfontosabbak a `HOME` és a `PATH`. Az előbbi tárolja a hozzánk rendelt könyvtárat, míg a második azokat az útvonalakat, amelyekben a shell a futtatható programokat tárolja.

A változók leggyakrabban parancsok argumentumaként jelennek meg. A változók értékét legegyszerűbben az `echo` paranccsal jeleníthetjük meg:

```
user@host:~$ echo $HOME
```

10.1.6. Kiegészítések

A shell-ben számos kiegészítő funkció található, gondoljunk csak a `<Tab>` billentyűre. Az alábbiakban megismerkedünk további két kiegészítési móddal.

Útvonalkiegészítés

Ha egy *szóban* szerepel a `*`, `?` vagy `{` karakter, akkor a shell az adott szót mintaként kezeli, és a helyére ABC sorrendben azok a fájlnevek kerülnek, amelyekre illik a minta. Ha nincs egyetlen egyezés sem, akkor az adott szót változatlanul hagyja a shell.

A minta illesztésekor az alábbi speciális karakterek kivételével minden karakter önmagára illeszkedik. A speciális karakterek jelentése a következő:

`*` A `*` karakternek hasonló funkciója van, mint a reguláris kifejezésekben, de nem teljesen ugyan az: a `*` helyén az útvonalkiegészítéskor tetszőleges számú, tetszőleges fajta karakter állhat, ellentétben a reguláris kifejezésekkel, ahol a `*` az előtte álló kifejezés tetszőleges számú ismétlését jelentette.

`?` A `?` karakter egy tetszőleges karaktert helyettesít.

`[...]` A zárójelen belül minden karakterre illeszkedik. A reguláris kifejezésekhez hasonlóan egy kötőjellel elválasztott karakterpár egy intervallumot jelent. Ha a zárójelekben az első karakter egy `!`-jel, vagy egy `^`-jel, akkor azok a karakterek illeszkednek, amelyek nincsenek a zárójelben.

Használhatóak a reguláris kifejezéseknél megismert karakterosztályok is:

```
[:alnum:] [:alpha:] [:ascii:] [:blank:] [:cntrl:] [:digit:]
[:graph:] [:lower:] [:print:] [:punct:] [:space:] [:upper:]
[:word:] [:xdigit:]
```

A karakterosztályok használatánál ügyelni kell a szögletes zárójelek megfelelő használatára, pl. egy útvonalkiegészítési így fest:

```
user@host:~$ echo IMG[[:digit:]]*.JPG
```

Karakter	jelentés
#	megjegyzés
$	változó helyettesítés
\	escape karakter (a következő karakter elveszti speciális jelentését)
’	szimpla idézőjelek: a köztük lévő karakterek elvesztik speciális jelentésüket
"	dupla idézőjelek: a köztük lévő karakterek elvesztik speciális jelentésüket, kivéve a $, a \, a ‘, és a !
*	tetszőleges számú karaktert helyettesít a fájlnevekben
?	egyetlen karaktert helyettesít a fájlnévben
&	parancs futtatása a háttérben
;	parancsok szekvenciális futtatása
|	pipe
>	standard kimenet átirányítása
<	standard bemenet átirányítása
/	könyvtárakat elválasztó jel

10.1. táblázat. Leggyakoribb speciális shell karakterek.

Kapcsos zárójel kiegészítés

A kapcsos zárójel kiegészítés egy olyan mechanizmus, amellyel tetszőleges string előállítható. A mechanizmus hasonló az *útvonalkiegészítéshez*, de ebben az esetben a fájlnévnek nem kell léteznie. A kapcsos zárójel kiegészítés egy opcionális előtaggal kezdődik, amelyet egy kapcsos zárójelek közé írt, vesszővel elválasztott lista követ. Végül az egészet egy opcionális utótag követi.

A kifejtéskor a közös előtaghoz a shell beírja egyesével a listában található kifejezéseket, majd mindegyikhez hozzáfűzi az utótagot. Például `a{b,c,d}e` eredménye: `abe ace ade` lesz.

10.2. Adatfájlok karaktereinek és oszlopainak manipulálása

10.2.1. Karakterenkénti „fordítás” vagy törlés

A `tr` parancs a standard bemenetről érkező karaktersorozatot lefordítja vagy törli, és az eredményt kiírja a standard kimenetre. Paraméterként egy vagy két karakterhalmazt vár a program. A `-d` *(delete)* kapcsolóra az első karakterhalmaz elemeit törli a szövegből. A fordítás akkor történik, ha a `-d` kapcsoló nincs bekapcsolva, és mind a két karakterhalmaz adott. Ekkor az első karakterhalmaz minden előfordulását kicseréli a második

karakterhalmaz megfelelő elemére. Ha az `-s` kapcsoló esetén a program összevonja az egymás után álló egyforma karaktereket.

10.2.2. Oszlop kivágása, összefűzése

Az adatfájloknak gyakran túl sok oszlopa van. Ilyenkor az adatfeldolgozás során ki kell törölni a felesleges sorokat. Erre szolgál a `cut` parancs. A `-d` kapcsolóval megadhatjuk, hogy az oszlopokat mi választja el egymástól, a `-f` kapcsoló pedig kijelöli a szükséges oszlopokat.

Előfordulhat az is, hogy a szükséges adatok több fájlban vannak. Ekkor használhatjuk a `paste` programot, amely soronként összepárosítja a különböző adatfájlokat.

10.3. Adatok rendezése

A `sort` parancs

A `sort` utasítás minden sor legelső karaktere szerint működik. Ha az első karakterek megegyeznek, akkor a második, harmadik, stb. karaktereket hasonlítja össze. A rendezés a szokásos ASCII kódok esetén az írásjelek – számok – nagybetűk – kisbetűk sorrend szerint történik. Az `-r` kapcsolóval a rendezési sorrendet teljesen megfordíthatjuk. Az `-n` kapcsolóval numerikusan rendezhetjük a sorokat. Az állomány sorai néha oszlopokba rendezett adatok mezőit tartalmazzák. A mezők között valamilyen határoló vagy elválasztó jelnek kell állnia, ez alapesetben a szóköz vagy a tabulátor. Ha a `sort` parancsot a *+Szám* kapcsolóval használjuk, a *Szám* által meghatározott mezőt kihagyva végzi a sorba rendezést. A *-Szám* hatására a rendezés a *Számnál* megadott mezőnél ér véget.

A `sort` parancsnak számos alkalmazási lehetősége van. Az egyik legfontosabb az adatok mediánjának megkeresése. A medián az az érték, aminél az adatok fele kisebb, fele nagyobb. Páratlan számú adatnál ez éppen a sorba rendezett adatsor középső eleme, páros számú adatnál pedig a két középső elem.

Másik lehetőség az adatok ún. kumulatív gyakoriságeloszlásának meghatározása.

A `wc` parancs

A betűk, szavak, és sorok leszámlálására használjuk a `wc` *(word count)* parancsot. Ha tehát a középső elemet keressük, akkor először számláljuk le az adatokat a `wc` paranccsal, majd osszuk el a kapott értéket kettővel.

A `uniq` parancs

A `uniq` parancs kiszűri az egy állományban található egymás után ismétlődő sorokat, és csak egyszer írja ki az ismétlődő sort (célszerű lehet ezért az állományt először a `sort`

utasítással rendezni). A parancsot olyankor használjuk, amikor például egy listában ki akarjuk szűrni az ismétléseket. A parancsnak a kapcsolókon kívül két argumentuma lehet, a bemeneti és a kimeneti állomány. Ezek hiányában a standard kimenetre illetve bemenetről dolgozik. A parancs alapértelmezésben teljes sorokat hasonlít össze. Lehetőség van arra is, hogy az összehasonlítás során ne vegyen figyelembe megadott mezőket (a mező egy olyan karaktersorozat, melyet egy vagy több elválasztókarakter – általában szóköz vagy tabulátor – választ el egy másik karaktersorozattól).

A `-c` kapcsoló hatására a sorok elé kiírja az ismétlések számát is. Ha ezt a kapcsolót használjuk egy sorba rendezett adatfájlon, akkor könnyen megkaphatjuk a leggyakoribb elemet (az adatok módusza). Ehhez csupán a gyakoriság szerint kell sorba rendezni az adatokat.

10.4. Példák és feladatok

Feladatok

F10.1. Írassuk ki az *echo* paranccsal a következő mondatokat:

1. `"$HOME, sweet $HOME"`
2. `"$HOME, sweet [`*`$HOME értéke`*`]"`
3. `'Azt mondta az öreg tímár:  "Ez az ulti nem ulti már."'`

F10.2. Listázzuk ki, és számoljuk meg a `/usr/lib` könyvtárban és az összes alkönyvtárában az összes olyan fájlt, amelynek utolsó előtti betűje „e”.

F10.3. A `sort` és a `uniq` parancsok segítségével keressük meg a mellékelt adatfájl módusza, azaz a leggyakoribb elemet. Rendezzük gyakoriság szerint sorba az adatokat és ábrázoljuk a gyakoriság-eloszlást *gnuplot*tal. Próbáljuk meg az x tengely címkézését az egyes kategóriáknak megfelelően megváltoztatni (Útmutatás: a *gnuplot* interaktív használatával kérjünk segítséget a `help set xlabel` paranccsal).

F10.4. A `sort`, a `uniq` és a `wc` parancsok használatával keressük meg a mellékelt adatfájl mediánját, azaz azt az elemet, aminél az adatok fele kisebb, illetve nagyobb.

F10.5. A mellékelt adatfájlban a Mauna Loa obszervatóriumban 1974 és 1987 között mért CO_2 koncentrációk találhatóak havi felbontással (`http://itl.nist.gov`). Készítsünk kumulatív gyakoriságeloszlást, azaz az koncentráció függvényében ábrázoljuk, hogy az adott értéknél hányszor fordul elő kisebb mért érték. Az eloszlást ábrázoljuk *gnuplot*tal, és illesszük be egy LaTeX dokumentumba.

F10.6. Számoljuk meg, hány olyan fájl van az `/usr` könyvtárban ami `c` vagy `v` betűvel kezdődik, és `o` betűre végződik. Azok az útvonalak, amit nincs jogunk olvasni hibát emelhetnek, gondoskodjunk róla, hogy a hibaüzenet a `keres.err` állományba kerüljön.

F10.7. Adott egy tömörített naplófájl, ami egy számítógép hálózati forgalmának csomagjait indulási és érkezési eseményeit tartalmazza. Ismerkedjünk meg a naplófájl szerkezetével. Az első oszlopban az időbélyegek olvashatóak. Készítsük el külön a kimenő illetve külön a bejövő csomagok között eltelt időtartamok eloszlását. Készítsünk ezekről *gnuplot* ábrákat. Határozzuk meg, hány bájt adat érkezett a `sportgeza.hu` szerverről a kliensre.

11. fejezet

Szimulációs feladatok

11.1. Példák és feladatok

Feladatok

F11.1. Piszkos Fred, a kapitány, miután meglehetősen sok rumot fogyasztott, végtelen bolyongásba kezd: 1/2–1/2 valószínűséggel balra vagy jobbra indul az Origó nevű vendéglátóipari egységből. Minden lépés után megpihen, majd ismét azonos valószínűséggel lép egyet jobbra vagy balra. Szimuláljuk a kapitány mozgását *gawk*kal!

1. Írjunk egy *gawk* szkriptet, amely $T = 10$ időlépés után kiírja a kapitány pozícióját. (Útmutatás: a programot a `BEGIN` blokkba írjuk. Véletlen számokat `rand()` függvénnyel kaphatunk, melyet a `srand()` függvénnyel inicializálhatunk.)
2. Módisítsuk az előző szkriptet úgy, hogy ne egyszer, hanem $N = 10\,000$-szer írja ki a kapitány pozícióját $T = 10$ lépés után.
3. Az előző szkripttel, valamint a `sort` és `uniq` parancsokkal készítsünk gyakoriságeloszlást, és az adatokat mentsük el egy fájl-ba. *gnuplot*tal ábrázoljuk az adatokat és illesszünk az adatokra normális eloszlást:

$$\mathcal{N}(m,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{11.1}$$

4. Ismételjük meg az előző feladatot $T = 5, 10, 20, 50, 100, 200, 500, 1000$ időlépésre, és írjuk be az illesztett paramétereket egy adatfájlba.
5. Ábrázoljuk az illesztett paramétereket a T függvényében. Mit tapasztalunk? Próbáljunk meg az adatokra hatványfüggvényt illeszteni, és határozzuk meg a hatványkitevőt!

6. Írjunk LaTeX jegyzőkönyvet Piszkos Fred bolyongásáról!

F11.2. Piszkos Fred újra felöntött a garatra az Origó nevű kocsmában. Igen ám, de a múlt alkalom óta fejlesztették az ivó környékét. Piszkos Fred az utóbbi időben már a síkon bármerre botorkálhat. Készítsük el a szimulátort, amivel a trajektóriáját lehet nyomon követni. Vizsgáljuk meg a következő kérdéseket, amiről ábrákkal tarkított jegyzőkönyvet készítünk.

1. N lépés után Piszkos Fred elpihen egy arra alkalmas helyen, mondjuk egy árokban. Hol vannak a gödrök, azaz hogyan alakul Piszkos Fred kocsmától számított távolságának eloszlása?
2. Hogyan változik N-nel ez az eloszlás?
3. Bolyongása során Piszkos Fred néha visszatér egy-egy pohárkára, amikor az ivó ϵ sugarába kerül. Mekkora ennek a valószínűsége? Valamint a visszatérések között megtett utak hossza milyen eloszlást követ?
4. Milyen érdekes kérdést tehetünk még fel Piszkos Fred botorkálásával kapcsolatban?

F11.3. Dr. Watson a Balaton-parton üldögél, amikor a víz egy palackot sodor a partra. A palackban a következő üzenetet találja:

```
RGKKTRJLFKVJABXVVNAXALGJASTMLGMTMFEKVKZFKEKVKZFKGRAZGXOTMXVG
HYEKGPRYVSFRVPAREASARJFXTMOGHJARBFZNVJOVNBVKASEKVHAEEKGEIVSV
RHNVFZEAJVRSVEGIGBGSFZGRGJPGBFHVKVDEGRHGELARHVERVHYXVMZAKVEA
JIFSRAJGMGEKVBBFHSVXFEGHSVBARTNNVGPTMVKGHYJASFJFSVMYFPVMAZVE
AEGOMVZAZZPAOPVXVMASFVEFZGSOGJJGMCTHHAJGSZGJJGMVKVXVZVEFTRRG
PEGHGRVKZNGMKFOVNBVIVSVRHJARHVERGIVRYVHHVEZYVRRVJJARRYLGMVLV
BVEKGOGXVMALVRVKMGEKVKVPGSDVKVRVHYGESGHGRGEGBGJGEPGSDVOFJASX
GHGZGSVKGKGSGXGEPGDIEKGSFVXFEGHSVBVJJASOVSRGOGRRGJVDEAPPJFEA
SEKVHRVJMGEKVEKGJIGMYGIVRGOVBTRVFOVHYVSJAKZVSEVEVHRVJOGMYRGJ
RGHYXVHYAZZGRHGSOAEEVPVSZNVFZBTRVFRVJVKGSZCAHNVJIFXRFVJAKZVS
EVEVHAZRGIAHYAEEKGJGXGSNGJGHYOVEFJJVMVKVMEASVNRVFOVHYVSJAKZV
SEVEVHHVMGKTZALLFZVJJASEGOVHYVSAJMVJNVJOVNBIVRGOJAPAZZAMZAKG
JTGMRYTZZVMEASVNRVFVJVJFJDEVJJVLVMVLAMXGZZGJCAMVOVHYVSRGXGZM
GEGMGIGZFSRFOFMYGRNABAMAHMGEKVJJASOVHYVSRVJMGRRFGMGHZVMVRVRR
YFZOARBVRFIAHYVOVHYVSEKAPAZAOEKVKZFKGRAZGXVMVZZFHGXGXVMZAKFJ
OGMYVBBFHSVOFRBGRGMARYGMXLGCGMEKFXABFJOGHIAKKVJGMMGOGENGMGRZ
GEZVSZVMAOOVMOVHYVSRFCSVRDFVTMPGMBVTMVRRYFZZGEKOVNBOVHVOVZNA
MMGEKAPRFEPVRYAMTMTZDVRPGRKZZVMVMRFGSZGMGIVNAMRFJVZVMVRRYGMX
NVSVELVRJARRYGBGRIVNAMHVZAJVOFAZVJFRKABGSGJKEVLVOLAMJFHYAHYT
MZVOGEIVXVMVJFMARBARLVRFHYEKAMFVOHAFRHOVHYVSRFXVHYFEEKAEKGSF
```

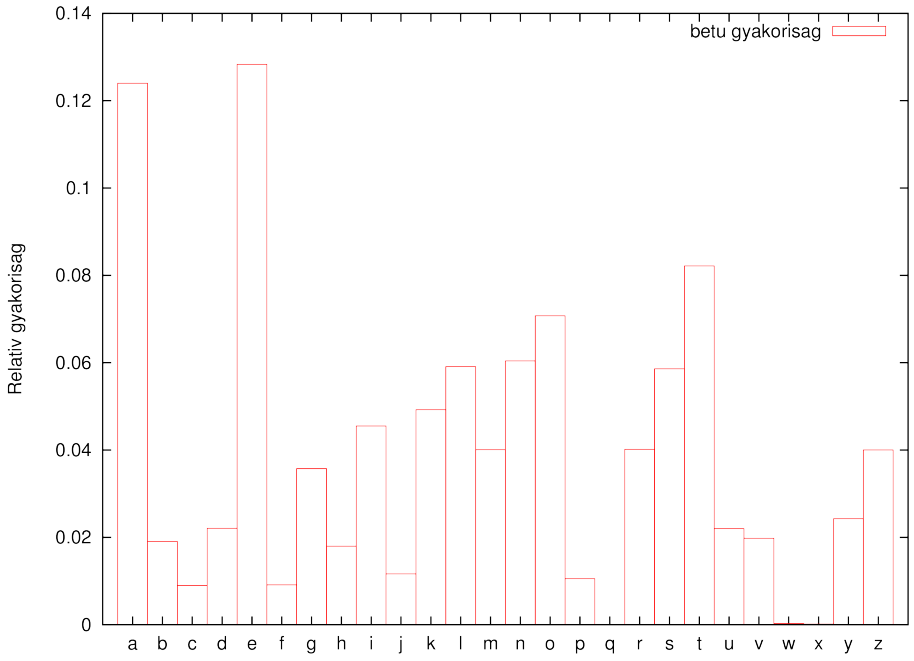
```
RZOGHYGJOVHYVSRFGKVKZNGMGRZFVIIAKVKFEZGRFRAIAKVJFZAZZMVZEKOA
EZABVOGHYGJOGHEKAMFZAOLGMGJVSAMAJIVKVXFEKGOGEFZZGHYDETRYVEKA
JAXGZJGKFJOVEPGMBVGROVHYVSAJZGOVHYVSEKAOVHYVSFJOGSZFJGEMGEKV
KFHGIGZDFXFMFKVMZRYGMXGRRASXGHTMHASAHTMLAMHVSTMLVEKJRYGMXGRE
ZLVKZNGMGRZFOVNBSAPAHAEJVDEVETMZGZGEKGOGEKGMGEKFJFBGFTLASJVE
VMVZVXVMOFJAKLGRYGITBFOGRTIFRVDEVJGHYJFEMVRYZITKKVVCTMGOLGZA
XVLLVVRYTJVOGIGZGJOVHYVSRFOVHYVSIVZEKMGZZTMVKZNGMGRZFIAHYGHY
JFECFTGMJGSGBKJGBFJOAKFLVEVKGBGEVRYNVRGOFIVLAKVETZVRGMGRHGBF
PGBFHVCFMOGZDEVJZFKGRRYAMDGXGRCGMTMFGJMVZAHVZIVZNVJBGIVHYNTJ
VJTMCAMBGZFZZIARFEEAJOFRBGRZOVEJGRZIFXRVJOVNBPGMBVTMVXVRFMFV
IGMYGZZOGMYFBGHGREKAVIVLASTOGHYVZVJAKZTBVZLVOFRZIAHYSGHFNGMG
RZGEGZVOTHYFEGMXGEKZGZZGVXFEGHSVBFDTJSVEKBVLVRZGIVZVCVHYMVMZ
AEPTMZCAMAZZGKMGEKJFFSXVGPGSPTRDEIVLASTDEAJAMVBGFHYCAHTRJGMR
FVBBFHGKZVPVSGXGZJFJGMMLFSRF
```

A jelek szerint a szöveget betűhelyettesítéses titkosítással kódolták, és magyarul íródott. Segítsünk Watsonnak megfejteni az írást!

A megfejtésről készítsünk egy LaTeX dokumentumot, mely tartalmazza a megfejtést, valamint a megfejtéshez írt programot.

(Útmutatás: A gyakorlaton megismert Unix parancsok segítségével a titkosított szöveg betűiről készítsünk egy gyakoriság eloszlást, és hasonlítsuk össze a 11.1. ábrán látható eloszlással, mely egy tipikus magyar nyelvű szövegből készült.

A titkosítás visszafejtését kezdjük a leggyakoribb betűkkel. A `tr` paranccsal a megfejtett betűk helyére helyettesítsünk kis betűket. Ennek segítségével a részben megfejtett szöveg egyes részeiből újabb betűkre következtethetünk.)



11.1. ábra. Betűk gyakoriság eloszlása magyar szövegben.

12. fejezet

Megoldások

Ebben a fejezetben összegyűjtöttük a gyakorló példák megoldásához szükséges legfontosabb lépéseket.

12.1. Az 1. fejezet gyakorló feladatainak megoldásai

M1.1. megoldás: A nem annyira billentyűtakarékos megoldás a következő:
`lljllklljjjhhhhhjjlllllllkkkkklljjllkklljllklljjjhhh`
`hjhhhhjjjhhhhhhhhhjjlllllllllllkkkllllklljjjjhhkhhjj`
Megjegyezzük, hogy az alábbi rövidebb parancssorozat is végigvezet a labirintuson:
`lljllkll3j4hhjj7l5klljjjllkklljllkl`
`l3j4hj4h3j9hjj11l3k4lkll4jhhkhhjj`

M1.2. megoldás: Egy lehetséges megoldás a következőképpen kezdődik:

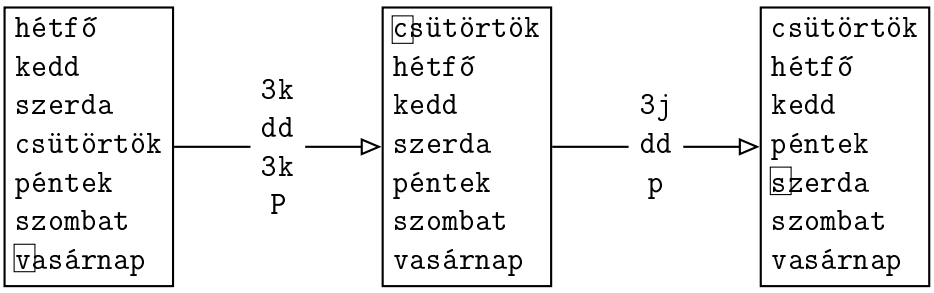
Mit?	Miért?
`to`	Jobbra visszük a kurzort (`t`) az első `o` karakterig.
`5l`	A kurzorral 5 karakterrel jobbra (`l`) ugrunk.
`to`	Ismét jobbra visszük a kurzort az első `o`-ig.
`jjjjj`	Ötször lefelé (`j`) visszük a kurzort.
`To`	Balra visszük a kurzort (`T`) az első `o`-ig.
⋮	⋮

Egy teljes megoldás a következő:
`to5ltojjjjjTojjhh4jhhjhhhjhjjljllll9lt`
`okkkklklll3ktokktoklkto6jhjjjj3jjljljl`

M1.3. megoldás: Az alábbi táblázat mutat egy megoldást:

Mit?	Miért?
`ihétfő`	Insert módba kapcsolunk és begépeljük az első napot.
`<Enter>`	Új sort kezdünk.
`kedd`	Begépeljük a második napot.
⋮	⋮
`vasárnap`	Begépeljük az utolsó napot.
`<Esc>`	Kilépünk az Insert módból.
`3k`	A kurzort 3 sorral feljebb visszük `csütörtök`-re.
`dd`	A `csütörtök`-öt kitöröljük, ami így bekerül egy ideiglenes regiszterbe. A kurzor a következő sorra (`péntek`) ugrik.
`3k`	A kurzort 3 sorral feljebb visszük.
`P`	Az ideiglenes regiszter tartalmát visszamásoljuk `hétfő` *fölé* (`P`).
`3j`	A kurzort 3 sorral lejjebb visszük.
`dd`	A `szerdá`-t kitöröljük, ami így bekerül egy ideiglenes regiszterbe. A kurzor a következő sorra (`péntek`) ugrik.
`p`	Az ideiglenes regiszter tartalmát visszamásoljuk `péntek` *alá* (`p`).

A megoldásmenet kezdeti fázisai az alábbi ábrán láthatók, ahol a segédkeret mutatja a kurzor helyét az egyes műveletek előtt és után.



M1.4. megoldás: Az alábbi táblázat mutat egy megoldást:

Mit?	Miért?
`yy`	Bemásoljuk a címet tartalmazó sort egy ideiglenes regiszterbe
`p`	Visszamásoljuk az ideiglenes regiszter tartalmát a szövegbe, a cím alá. A kurzor automatikusan a bemásolt cím első nem üres karakterére ugrik.
`v`	Visual módba kapcsolunk.
`$`	A cím utolsó karakterére ugrunk.
`r-`	A kijelölt karakterek cseréje a `-` karakterre

M1.5. megoldás: Az alábbi táblázat egy lehetséges megoldást vázát mutatja be:

Mit?	Miért?
`"ayw`	Az „a” regiszterbe (`"a`) bemásoljuk az első szót (`dw`).
`"byw`	Az „b” regiszterbe (`"b`) bemásoljuk a második szót (`dw`).
⋮	⋮
`"fyw`	Az „f” regiszterbe (`"f`) bemásoljuk a hatodik szót (`dw`).
`3G`	A 3. sorra ugrunk.
`W`	A következő SZÓRA[1] ugrunk.
`"aP`	Az „a” regiszter tartalmát (`"a`) beszúrjuk a kurzor elé (`P`).
`dt,`	Kitöröljük (`d`) a karaktereket a vesszőig jobbra (`t,`).
`W`	A következő SZÓRA ugrunk.
`"bP`	Az „b” regiszter tartalmát (`"b`) beszúrjuk a kurzor elé (`P`).
`dt,`	Kitöröljük (`d`) a karaktereket a vesszőig jobbra (`t,`).
⋮	⋮

M1.6. megoldás: Az alábbi táblázat egy lehetséges megoldást mutat be:

Mit?	Miért?
`qa`	A makró rögzítésének indítása az „a” regiszterbe.
`0`	A sor elejére ugrunk.
`x`	A kurzor alatti karaktert kitöröljük, és bemásoljuk az ideiglenes regiszterbe.
`$`	A sor végére ugrunk.
`p`	Az ideiglenes regiszterből a sor végére másoljuk a karaktert.
`q`	A makró rögzítésének befejezése.

A makrót a `@a` billentyűk leütésével indíthatjuk. A makró egymás utáni alkalmazásával „körbeforgathatjuk” a sort. Próbáljuk ki!

12.2. A 2. fejezet gyakorló feladatainak megoldásai

M2.1. megoldás: A fordítás fázisait, a részeredmények és a teljes végleges dokumentum megtekintését az alábbi parancsokkal tehetjük meg:

```
user@host:~$ latex Gy2.1.tex
user@host:~$ xdvi Gy2.1.dvi
user@host:~$ dvips Gy2.1.dvi
user@host:~$ gv Gy2.1.ps
user@host:~$ ps2pdf Gy2.1.ps
```

[1] A SZÓ a következő üres hellyel elválasztott szót jelenti.

```
user@host:~$ evince Gy2.1.pdf
```

Megjegyezzük, hogy csak akkor tudjuk kiadni az `xdvi`-t és a `gv`-t követő parancsokat, ha ezeket a programokat bezártuk, vagy ha új terminálablakot nyitunk.

M2.2. megoldás: A megoldás vázlatos menete a következő:

1. Az `article`-t cseréljük le pl. `report`-ra,
2. A `\documentclass` parancs opcionális argumentumába (a „[]” közé) írjuk be, hogy twocolumn,
3. Az `itemize` környezetet cseréljük le `enumerate`-re.
4. A `\tableofcontents` parancsot helyezzük át az `\end{document}` elé.
5. ...

M2.3. megoldás: A feladat megoldása a 12.1. ábrán látható.

M2.4. megoldás: A megoldás a 12.2. ábrán látható.

M2.5. megoldás: A megoldás kezdő lépései az alábbi táblázatban követhető nyomon:

Mit?	**Miért?**
`:set number`	Bekapcsoljuk a sorok számozását, hogy lássuk melyik sorra kell ugranunk.
`24G`	A 24. sorra megyünk.
`<Ctrl-v>`	Block-visual módba kapcsolunk.
`34G`	A 34. sorra megyünk, és ezzel kijelöljük az első oszlopot.
`I\item`	Beszúrjuk a kijelölés elé az `\item` parancsot.
`<Esc>`	Kilépünk a beszúrás módból.
`24G$`	Visszalépünk a 24. sorra, és a sor végére megyünk.
`cib`	Kicseréljük a zárójelben lévő szöveget ...
`IGEN`	...IGEN-re.
`<Esc>`	Kilépünk a beszúrás módból.
`l$`	A következő sorra lépünk, és a sor végére megyünk.
`.`	Megismételjük a cserét.
⋮	⋮

12.3. A 3. fejezet gyakorló feladatainak megoldásai

M3.1. megoldás: A *vim* segítségével meggyőződünk róla, hogy valóban számok vannak az adatfájlban 3 oszlopba rendezve. A *gnuplot* elindítása után a következő parancsokat adjuk ki:

```
gnuplot> plot "sinusadatok.dat" u 1:2
gnuplot> set terminal postscript enh eps
gnuplot> set output "abra.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

Az elkészült ábrát egy Linux terminál ablakból a *gv* vagy az *evince* paranccsal tudjuk megnézni:

```
user@host:~$ evince abra.eps
```

M3.2. megoldás: A megoldás menete a következő:

```
gnuplot> plot "sinusadatok.dat" u 2:($1+$3) t "adatok" w p
    ps 2 pt 5 lc 3, 13+4.5*x title "egyenes" w line lc 0
gnuplot> set xlabel "ez az x tengely felirata"
gnuplot> set ylabel "a cica merete [koblab]"
gnuplot> replot
gnuplot> exit
```

M3.3. megoldás: Az előző feladat megoldásának lépéseit használjuk fel, természetesen a kilépés nélkül. A folytatás:

```
gnuplot> set terminal postscript enh eps
gnuplot> set output "abra2.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

Nyissuk meg az egyik múlt órai LaTeX feladat fájlját, és szerkeszzük bele a következő részt:

```
\begin{figure}
\centerline{
\includegraphics[width=.8\textwidth, angle=-90]{abra2.eps}
}
\caption{A 2-ik feladat ábrája.}
\end{figure}
```

12.4. A 4. fejezet gyakorló feladatainak megoldásai

M4.1. megoldás: Hozzuk létre *vim*mel az `M4.1.awk` nevű fájlt, majd írjuk be a megoldást:

```
/^[^#]/ && ($1 > 4) {s += $2; n++}
END{print s/n}
```

Lépjünk ki a *vim*ből, és futtassuk le az alábbi parancsot:

```
user@host:~$ gawk -f M4.1.awk Gy4.1
```

ahol `M4.1.awk` a fenti parancsokat tartalmazó szkriptfájl, `Gy4.1` pedig az adatokat tartalmazó fájl neve. A parancsfájl egyes részeinek jelentése a következő:

Mit?	Miért?
`/^[^#]/`	Azt a mintát keressük, amikor az első karakter *nem* `#`. Ehhez elsőként a mintát sor elejére illesztjük (`^`), majd azt a karakter osztályt vesszük, amely *nem* a `#` karakter. Ezzel ekvivalens a `!/^#/` minta is, amely a sor elején lévő `#`-jel illeszkedés negáltját adja.
`&&`	És kapcsolat a következő minta-feltétellel.
`($1 > 4)`	Az első oszlop elemének értéke nagyobb, mint négy.
`{s += $2; n++}`	A feltételnek megfelelő sorokra az `s` változó előző értékéhez hozzáadjuk a második oszlop értékét, és az `n` változó értékét eggyel növeljük.
`END{print s/n}`	Az adatfájl beolvasása után kiírjuk az átlagot.

M4.2. megoldás: Az alábbi *gawk* szkripttel oldhatjuk meg a feladatot:

```
/^[[:space:]]*#[^$]*/{$NF=""}
{print}
```

Mit?	Miért?
`/^[[:space:]]*#[^$]*/`	Azt a mintát illesztjük, amikor a sor elején (`^`) nulla vagy több üres karakter után (`[[:space:]]*`)egy `#`-jel áll (`#`), majd utána nulla vagy több karakter következik, amelyek nem `$`-jelek (`[^$]*`).
`{$NF=""}`	Az adott sor mezőinek számát a `NF` beépített változó tartalmazza. Az értékadással az utolsó mezőt üres stringre változtatjuk, azaz kitöröljük.
`{print}`	Kiíratjuk az adott sort. Mivel ez előtt a feladat előtt nincs minta, ezért ez a parancs minden sorra végrehajtódik, akkor is ha módosítottuk a sort, akkor is, ha nem.

M4.3. megoldás: *vim*ben egy lehetséges megoldás a következő:

```
/[[:alpha:]]\+[[:upper:]][[:alpha:]]*
```

Mit?	Miért?
`/`	Elindítjuk a keresést.
`[[:alpha:]]\+`	A minta először legalább egy betűre illeszkedik.
`[[:upper:]]`	Aztán egy nagybetűre illeszkedik a minta.
`[[:alpha:]]*`	Nulla vagy több nem üres karakter van még a szóban.

*gawk*kal a következő programmal írathatjuk ki a megfelelő szavakat:

```
BEGIN{RS="[[:space:]]"}
/[[:alpha:]]+[[:upper:]][[:alpha:]]*/
```

Mit?	Miért?
`BEGIN`	Az adatfájl beolvasása előtt...
`{RS="[[:space:]]"}`	...a rekordelválasztót beállítjuk minden üres helyre.
`/[[:alpha:]]+`	Hasonló a *vim*ben megadott reguláris kifejezéshez, de
`[[:upper:]]`	itt a `+` ismétlőoperátort nem kell backslash-sel speciális karakterré tenni. A `/.../` jelek a reguláris kifejezés értelmezést adnak a mintának. Mivel nincs a minta után semmi, így az alapértelmezett `print` utasítást végzi a *gawk*.
`[[:alpha:]]*/`	

M4.4. megoldás: A következő *vim* paranccsal

```
:%s/\([0-9]\{4}\)/\1./g
```

Mit?	Miért?
:	*vim* Command-line módba kapcsolunk
%	A tartomány, amire a Command-line parancsot kiadjuk az összes sor.
s	A `substitute` parancs rövidítése.
/\([0-9]\{4}\)/\1./	A minta amit keresünk: négy számjegy. Ezt zárójelezzük, hogy a helyettesítő string-nél visszahivatkozhassunk a számra. A helyettesítő string-ben \1-ként visszahivatkozunk a számra, és egy pontot teszünk utána.
g	A sorokban szereplő minden számra elvégezzük a cserét, nem csak az első találatnál.

12.5. Az 5. fejezet gyakorló feladatainak megoldásai

M5.1. megoldás: Nyissuk meg a fájlt *vim*mel, majd idjuk ki a következő keresési parancsot: `/[0-9]\+,\?[0-9]* [kmn]\?[msg]`

Mit?	Miért?
/	Keresés *vim*ben
[0-9]\+	A minta egy vagy több számra illeszkedik.
,\?	A minta Legfeljebb egy vesszőre illeszkedik.
[0-9]*	A minta nulla vagy több számra illeszkedik.
	Egy szóköz.
[kmn]\?	Ez a minta nulla vagy egy `k`, `m` vagy `n` betűre illeszkedik.
[msg]	A minta pontosan egy `m`, `s` vagy `g` betűre illeszkedik.

Figyeljük meg, hogy *vim*ben a `+` és `?` karaktereknek akkor van speciális jelentésük, ha egy \ jelet teszünk eléjük.

M5.2. megoldás: *vim*ben a következő paranccsal cserélhetjük ki a vesszőket tizedespontokra: `:%s/,/./g`

Ha arra is figyelni szeretnénk, hogy a vessző csak akkor szerepel tizedesvesszőként, ha előtte és utána is szám áll, akkor a cserét következő paranccsal végezhetjük el: `:%s/\([0-9]\),\([0-9]\)/\1.\2/g`

Ezután hozzuk létre *vim*mel az `M5.2.awk` nevű fájlt, majd írjuk be a megoldást:

```
# Először a prefixeket vesszük figyelembe:

/k[smg]/ {$1 *= 1000}
/m[smg]/ {$1 /= 1000}
/n[smg]/ {$1 /= 1000000000}

# Ezután összeadogatjuk a különböző mennyiségeket:

/[0-9]+,?[0-9]* [knm]?m/ {l += $1}
/[0-9]+,?[0-9]* [knm]?s/ {t += $1}
/[0-9]+,?[0-9]* [knm]?g/ {m += $1}

# A végén kiiratjuk az eredményt a terminálra:

END{print l, t, m}
```

Lépjünk ki a *vim*ből, és futtassuk le az alábbi parancsot:

```
user@host:~$ gawk -f M5.2.awk Gy5.2
```

ahol `M5.2.awk` a fenti parancsokat tartalmazó szkriptfájl, `Gy5.2` pedig az adatokat tartalmazó fájl neve. A parancsfájl egyes részeinek jelentése a következő:

Mit?	Miért?
`/k[smg]/`	Azt a sort keressük, amelyben van `k`, és ezt követi `s`, `m`, vagy `g`,
`{$1 *= 1000}`	majd erre a mintára illeszkedő sorokra az első oszlop értékeit megszorozzuk 1000-rel. Hasonló módon kezeljük a milli- és nano- prefixeket is.
`/[0-9]+,?[0-9]* [knm]?m/`	Ezután a azokra a sorokra, amelyekre a mértékegység `m`,
`{l += $1}`	az `l` változó értékéhez hozzáadjuk az első oszlopnak az előtaggal korábbiakban átskálázott értékét. Ezt megismételjük az idő és tömeg adatokra is.
`END{print l, t, m}`	Végül kiiratjuk az összegzett adatokat.

M5.3. megoldás: A megoldás a következő

```
gnuplot> reset
gnuplot> set xrange [0.01:10]
gnuplot> set log x
```

```
gnuplot> set xlab "x tengely"
gnuplot> set ylab "y tengely"
gnuplot> plot sin(1/x)
gnuplot> pause -1
gnuplot> show sample
gnuplot> set sample 1000
gnuplot> replot
gnuplot> pause -1
gnuplot> set sample 10000
gnuplot> replot
```

M5.4. megoldás: Ha *vim*mel megnyitjuk az adatfájlt, akkor láthatjuk, hogy a dátumok „hónap.nap.év” formátumban, a hőmérsékletek pedig Fahrenheitben vannak megadva. Ennek figyelembevételével a megoldás a következő:

```
gnuplot> set timefmt "%m.%d.%Y"
gnuplot> set format x "%Y. %m.  %d."
gnuplot> set xdata time
gnuplot> set y2tics
gnuplot> set ytics nomirror
gnuplot> set xrange ["1.1.2000":"12.31.2002"]
gnuplot> set yrange [-10:100]
gnuplot> set ylab "T [F]"
gnuplot> set y2lab "T [C]"
gnuplot> set y2range [5.0/9*(-10-32):5.0/9*(100-32)]
gnuplot> p "<gawk '!/-99/' Gy5.4" u 1:2 t 'New York' w l
gnuplot> rep "<gawk '!/-99/' Gy5.4" u 1:3 t 'Los Angeles' w l
```

M5.4. megoldás: Miután *vim*mel megvizsgáltuk az adatfájlt, hozzuk létre *vim*mel az `M5.5.awk` nevű fájlt, majd írjuk be:

```
{
  if ($4!=-99) d1[$1"."$2] += $4;
  if ($5!=-99) d2[$1"."$2] += $5;
  n[$1"."$2]++;
}
END {for(i in n) {print i " " d1[i]/n[i] " " d2[i]/n[i]}}
```

Ezután a következő *gnuplot* parancsokkal kirajzolhatjuk az ábrát:

```
gnuplot> reset
gnuplot> set xdata time
```

```
gnuplot> set timefmt "%m.%d"
gnuplot> set format x "%b.  %d."
gnuplot> set ylabel "T [F]"
gnuplot> set multiplot
gnuplot> p '<gawk -f M5.5.awk Gy5.5' u 1:2 t 'New York', '' u 1:3 t 'Chicago'
gnuplot> set origin 0.3, 0.1
gnuplot> set size 0.5
gnuplot> unset key
gnuplot> set format x "%b."
gnuplot> set title 'Temperature Difference'
gnuplot> p '<gawk -f M5.5.awk Gy5.5' u 1:(3-2)
gnuplot> unset multiplot
```

12.6. A 7.1. fejezet gyakorló feladatainak megoldásai

M7.1. megoldás: A *vim* segítségével meggyőződünk róla, hogy valóban számok vannak az adatfájlban két oszlopba rendezve. A *gnuplot* elindítása után a következő parancsokat adjuk ki:

```
gnuplot> plot "fitadatok1.dat" u 1:2
gnuplot> fit a+b*x "fitadatok1.dat" via a,b
gnuplot> plot [][0:150] a+b*x, "fitadatok1.dat"
gnuplot> set terminal postscript enh eps
gnuplot> set output "abra.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

Az elkészült ábrát egy Linux terminál ablakból az *evince* paranccsal tudjuk megnézni:

```
user@host:~$ evince abra.eps
```

M7.2. megoldás: A *vim* segítségével meggyőződünk róla, hogy valóban számok vannak az adatfájlban két oszlopba rendezve. A *gnuplot* elindítása után a következő parancsokat adjuk ki:

```
gnuplot> plot "fitadatok2.dat" u 1:2
gnuplot> fit [20:60] a+b*x "fitadatok2.dat" via a,b
gnuplot> plot a+b*x, "fitadatok2.dat"
gnuplot> set terminal postscript enh eps
```

```
gnuplot> set output "abra.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

Az elkészült ábrát egy Linux terminál ablakból az *evince* paranccsal tudjuk megnézni:

```
user@host:~$ evince abra.eps
```

M7.3. megoldás: A *vim* segítségével meggyőződünk róla, hogy valóban számok vannak az adatfájlban két oszlopba rendezve. A *gnuplot* elindítása után a következő parancsokat adjuk ki:

```
gnuplot> plot "fitadatok3.dat"
gnuplot> fit a+b*x+c*x*x "fitadatok3.dat" via a,b,c
gnuplot> p "fitadatok3.dat",a+b*x+c*x*x
gnuplot> set terminal postscript enh eps
gnuplot> set output "abra.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

Az elkészült ábrát egy Linux terminál ablakból az *evince* paranccsal tudjuk megnézni:

```
user@host:~$ evince abra.eps
```

M7.4. megoldás: A *vim* segítségével meggyőződünk róla, hogy valóban számok vannak az adatfájlban két oszlopba rendezve. A *gnuplot* elindítása után a következő parancsokat adjuk ki:

```
gnuplot> plot "fitadatok4.dat"
gnuplot> fit p*sin(r*x-s) "fitadatok4.dat" via p,r,s
gnuplot> p "fitadatok4.dat",p*sin(r*x-s)
gnuplot> p=1.5
gnuplot> r=0.05
gnuplot> s=0
gnuplot> p "fitadatok4.dat",p*sin(r*x-s)
gnuplot> r=0.1
gnuplot> p "fitadatok4.dat",p*sin(r*x-s)
```

```
gnuplot> fit p*sin(r*x-s) "fitadatok4.dat" via p,r,s
gnuplot> p "fitadatok4.dat",p*sin(r*x-s)
gnuplot> set terminal postscript enh eps
gnuplot> set output "abra.eps"
gnuplot> replot
gnuplot> set terminal wxt
gnuplot> set output
gnuplot> exit
```

A megoldás menete közben észre kellett vennünk, hogy az első illesztési parancs kiadása után az illesztett paraméterek hibái nagyok voltak. Ennek oka, hogy rossz kezdőfeltételekkel indult a paraméteroptimalizálás. Ezért a függvény analízisével és az adatok ismeretében meg kell becsülnünk a modellparamétereket. Ezután célszerű a becsült paraméterek mellett közösen ábrázolni az adatpontjainkat és a modellfüggvényt, hogy lássuk helyes volt-e a becslésünk. Itt fenn ez nem teljesült, a periódusidőn tovább kellett finomítanunk. Az új illesztés elfogadható eredményre vezetett.

Az elkészült ábrát egy Linux terminál ablakból az *evince* paranccsal tudjuk megnézni:

```
user@host:~$ evince abra.eps
```

12.7. A 8. fejezet gyakorló feladatainak megoldásai

M8.1. megoldás: A megoldás a 12.3. ábrán látható.

Mit?	Miért?
`{`	Egy feladatblokk kezdete, amely – mivel nincs minta a blokk előtt – a bemenő fájl minden sorára végrehajtódik.
`pos = 0;`	A `pos` változó értékét lenullázzuk.
`neg = 0;`	A `neg` változó értékét lenullázzuk.
`for(i=1; i<=NF; i++) {`	Egy `for`-ciklus, amelyben az `i` indexváltozó végigfut 1-től az adott sorban található rekordok számáig (NF).
`if ($i > 0) pos++;`	Megvizsgáljuk, hogy az `i`-ik mező pozitív-e. Ha igen, akkor növeljük a `pos` változó értékét eggyel.
`if ($i < 0) neg++;`	Megvizsgáljuk, hogy az `i`-ik mező negatív-e. Ha igen, akkor növeljük a `neg` változó értékét eggyel.
`}`	Bezárjuk a `for`-ciklust.
`print pos, neg`	Kiíratjuk az eredményt.
`}`	Bezárjuk a feladatblokkot.

A szkriptet végül a Gy8.1 tesztfájlon próbáljuk ki:

```
user@host:~$ gawk -f M8.1 Gy8.1
```

paranccsal futtathatjuk le.

M8.2. megoldás: A megoldás a 12.4. ábrán látható.

Mit?	Miért?
`{ jegy[$2]++; }`	A `jegy` tömbben számoljuk a különböző jegyek számát. A tömböt a bemeneti sor második mezőjével indexeljük, és minden találatra növeljük eggyel a tömb megfelelő elemét.
`END {`	Az adatfájl beolvasása után végrehajtandó blokk.
`for(j in jegy)`	`for`-ciklus, amelyben a `j` indexváltozó végigfut a `jegy` tömb indexhalmazán.
`print j, jegy[j];`	Kiíratjuk az adott indexet és a hozzá tartozó jegyek számát.
`}`	Bezárjuk az adatfájl beolvasása után végrehajtandó blokkot.

```
% A Gy2.3 gyakorló feladat megoldása

\documentclass[12pt]{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[magyar]{babel}

\author{Gipsz Jakab}
\title{Gy2.3 gyakorló feladat}
\date{}
\begin{document}
\maketitle
\section{Bevezetés}
Ez a feladat az alapvető \LaTeX{} dokumentum struktúra
felépítésének gyakorolását segíti.

\section{Az ékezetes betűk}
Az ékezetes betűk használatához be kell tölteni a
\textsf{fontenc} és az \textsf{inputenc} kiegészítő csomagokat.

\subsection{Ékezetek \LaTeX{} parancsokkal}
Ha nem töltöttük volna be a szükséges csomagokat, akkor \LaTeX{}
parancsokkal \'\i{}gy kellene magadni az \'ekezetes bet\H{u}ket a
forr\'asf\'ajlban.  Ez a megold\'as hosszabb sz\"oveg eset\'en
k\'ets\'egtelen\"ul el\'eg k\'enyelmetlen lenne.  Ha viszont
valaki ismeri az \'ekezetek bevitel\'ere alkalmas parancsokat,
akkor az k\'epes tetsz\H{o}leges bet\H{u}re tetsz\H{o}leges
\'ekezetet fel\'\i{}rni: p\'eld\'aul \~n, \`e,\"\i, \c{o}, \dots

\end{document}
```

12.1. ábra. Az összekevert LaTeX forrásfájl megoldása.

```
% A Gy2.4 feladat megoldása

\documentclass[twoside]{book}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[magyar]{babel}

\author{Gipsz Jakab}
\title{Gy2.4 gyakorló feladat}
\date{}
\begin{document}
\maketitle
\chapter{Bevezetés}
Ez a feladat az alapvető \LaTeX{} dokumentum struktúra
felépítésének gyakorolását segíti, de megismerhetjük belőle
a különböző betűtípusokat is.

\section{Betűtípusok}
Az alábbi betűtípusok közül választhatunk:
\begin{itemize}
  \item \textrm{antikva}
  \item \textit{kurzív}
  \item \texttt{írógép}
  \item \textsf{groteszk}
\end{itemize}

\section{Betűváltozatok}
Minden betűtípushoz választhatjuk az alábbi változatokat, akár
többet is:
\begin{enumerate}
  \item \textup{álló}
  \item \textsl{döntött}
  \item \textmd{félkövér}
  \item \textbf{kövér}
  \item \textsc{kiskapitális}
\end{enumerate}

\textsl{\textbf{\textsf{Ezt a mondat tehát döntött kövér és
groteszk.}}}

\end{document}
```


12.2. ábra. A hiányos LaTeX forrásfájl megoldása.

```
{
  pos = 0;
  neg = 0;
  for(i = 1; i <= NF; i++) {
    if ($i > 0) pos++;
    if ($i < 0) neg++;
  }
  print pos, neg;
}
```

12.3. ábra. *awk* szkript, amely megszámolja, hogy hány pozitív és negatív szám van egy sorban.

```
{ jegy[$2]++; }

END {
  for(j in jegy)
    print j, jegy[j];
}
```

12.4. ábra. *awk* szkript, amely megszámolja, hány ötös, négyes, stb. van egy évfolyamban.

Irodalomjegyzék

[1] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. The not so short introduction to latexe. http://tobi.oetiker.ch/lshort/lshort.pdf, 2008.

[2] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. Egy nem túl rövid bevezető a latexe használatába – avagy latexe 78 percben. https://www.cs.elte.hu/local/TeX/lrovid.dvi, 2008.

[3] Wettl Ferenc, Mayer Gyula, and Sudár Csaba. *LaTeX kezdőknek és haladóknak*. Panem Kiadó, Budapest, 1998.

[4] Johannes Braams. Babel, a multilingual package for use with latex's standard document classes. http://parokia.kre.hu/lelkesz/latex/babel.pdf, 2006.

[5] D. P. Carlisle and S. P. Q. Rahtz. The graphicx package. http://users.ecs.soton.ac.uk/srg/softwaretools/document/start/graphicx.pdf, 1999.

[6] Borsányi Szabolcs. Gnuplot howto oldal. http://achilles.elte.hu/gnuplot/.

[7] Dr. Horváth András. Gnuplot használata. http://www.fizika.sze.hu/~horvatha/Fi12/Oravazlatok/Gnuplot/gnuplot.pdf.

[8] Thomas Williams and Colin Kelley. gnuplot - an interactive plotting program. http://gnuplot.sourceforge.net/docs_4.0/gnuplot.pdf.

[9] American Mathematical Society. User's guide for the amsmath package. ftp://ftp.ams.org/pub/tex/doc/amsmath/amsldoc.pdf, 1999.

[10] American Mathematical Society. Latex mathematical symbols. http://amath.colorado.edu/documentation/LaTeX/Symbols.pdf, http://mirror.math.ku.edu/tex-archive/fonts/amsfonts/doc/amssymb.pdf, 2013.

[11] Piet van Oostrum, Øystein Bache, and Jerry Leichter. The multirow, bigstrut and bigdelim packages. http://www.math.washington.edu/tex-archive/macros/latex/contrib/multirow/doc/multirow.pdf, 2010.

[12] David Carlisle. The hhline package. http://www.ctex.org/documents/packages/table/hhline.pdf, 1994.

[13] Bagoly Zsolt and Papp Gábor. *Bevezetés a UNIX rendszerekbe.* ELTE egyetemi jegyzet, 1993.

[14] Sikos László. *Bevezetés a LINUX használatába.* BBS-INFO kiadó, 2005.