Mètodes Numèrics

Guia de supervivència informàtica

Grau de Matemàtiques UAB, 2014–2014

J.M. Mondelo

1 Ús bàsic de Linux des d'una terminal

1.1 Arbre de directoris

En Linux (de fet en qualsevol sistema Unix, com Mac OS X) no hi ha "lletres d'unitat" (com `c:` o `e:`,...), sinó que hi ha un únic arbre de directoris. El directori arrel es representa per `/`, i aquest mateix caràcter fa de separador de directoris. Així, `/bin` és el subdirectori `bin` del directori arrel, i `/bin/ls` és el fitxer `ls` del directori `/bin`.

Existeix una operació anomenada "muntar", que consisteix a associar els continguts d'un cert dispositiu (com un pen–drive, CD–ROM o una partició del disc dur) a un punt de l'arbre de directoris general. L'operació contrària es diu "desmuntar". Vegeu la comanda `df` més endavant.

1.2 Línia de comandes

Si des de l'entorn gràfic obriu una terminal (icona "Konsole" de la carpeta d'escriptori, o bé cliqueu a la icona de la UAB l'esquerra de la barra inferior, pestanya *Aplicacions*, *Sistema→Konsole*), us apareixerà una línia semblant a

```
NIU@acielc-05:~$
```

Aquest grup de caràcters acabats en dòlar s'anomena "prompt", i indica que el sistema està esperant que li entreu una *comanda*. Si escriviu alguna cosa i apreteu `RETURN`, la primera paraula del que hagueu escrit s'interpretarà com el nom d'un programa a executar, i la resta s'interpretarà com arguments, que es passaran al programa en qüestió.

Així, si escriviu

```
ls
```

s'executarà la comanda `ls`, que mostra els fitxers del directori actual (probablement el vostre directori *home*). Si, en canvi, escriviu

```
ls -l
```

s'executa la comanda `ls` amb l'opció `-l`. Aquesta opció (que significa "long") fa que, a més dels noms dels fitxers del directori actual, vegeu informació addicional com els seus permisos, tamany o data de darrera modificació.

1.3 Comandes bàsiques

- `pwd` (Print Work Directory)

  Escriu el nom del directori actual.

- `cd` (Change Directory)

  Canvia el directori actual a l'argument que se li passi. Si no se li passa cap argument, canvia el directori actual al vostre directori *home*. Dins la llista d'arguments, "`..`" fa referència al directori pare del directori actual, i "`.`" fa referència al directori actual. Proveu de fer:

- `ls` (LiSt)

  Sense cap més argument, llista el contingut del directori actual. Si li poseu un nom de de directori o fitxer, llista aquest directori o fitxer (p.ex., feu "`ls /bin`", "`ls /bin/ls`"). Amb l'opció `-l` us mostra informació addicional, com hem vist abans. L'opció `-d` no llist ael contingut dels directoris especificats, sinó els directoris mateixos. Per exemple, si feu

  ```
  ls -ld /bin/ls /bin
  ```

  (noteu que es poden barrejar opcions, "`-ld`" equival a "`-l -d`") obtindreu

  ```
  drwxr-xr-x    2 root    root     4096 Feb 17 13:32 /bin/
  -rwxr-xr-x    1 root    root    43784 Mar 18  2002 /bin/ls
  ```

  En aquestes línies, la primera columna conté els permisos, la tercera el propietari del fitxer, la quarta el grup del fitxer, a continuació ve la data d'última actualització, i finalment el nom.

  La columna de permisos s'interpreta com segueix: la primera lletra ens indica si l'entrada corresponent és un directori (`d`) o un fitxer (`-`)[1].

[1] Hi ha més codis: `l` vol dir *link simbòlic*, `p` vol dir *pipeline*, `s` vol dir *socket*. No les farem servir.

Les tres següents fan referència al permsisos del propietari del fitxer: `r` vol dir que (el propietari) pot veure el seu contingut, `w` vol dir que pot modificar el seu contingut, i `x` vol dir que el pot executar. Per a directoris, "executar" vol dir "poder fer-lo actiu" (amb la comanda `cd`). Les lletres 5–7 fan referència al grup, i les lletres 8–10 fan referència a la resta d'usuaris.[2]

- `cp` (CoPy)

  Copia fitxers. Té dues formes principals:

  - `cp fitx1 fitx2`
    Copia `fitx1` a sobre de `fitx2` (compte!).
  - `cp fitx1 fitx2 directori`
    Copia els fitxers `fitx1` i `fitx2` al directori `directori`.

  Amb l'opció `-r` s'accepten directoris a la llista de fixters, i llavors es copia tot el seu contingut també.

  A la llista d'arguments s'accepten els anomenats *caracters comodí*, que són `*` i `?`. L'intèrpret de comandes,[3] abans d'exectuar cap comanda expandeix els caràcters comodí, d'acord amb les següents regles:

  - `*` representa qualsevol conjunt de caràcters de qualsevol longitud.
  - `?` represeta exactament un caràcter.

  Així, "`*.c`" fa referència a tots els fitxers que acabin en "`.c`", i "`solucio*`" fa referència a qualsevol fitxer que comenci per `solucio`. També, per exemple, si al directori actual teniu uns fitxers `solucio1.txt`, `solucio2.txt`, `solucio3.txt` i els voleu copiar al directori pare del directori actual, heu de fer

  ```
  cp solucio?.txt ..
  ```

  Si hi hagués un fitxer anomentar `soluciok.txt`, la comanda anterior també el copiaria.

  La opció `-i` us demana confirmació abans de fer qualsevol acció potecialment perillosa, per exemple si feu

  ```
  cp -i fitx1 fitx2
  ```

[2] Com heu endevinat, a tot sistema Unix els usuaris estan agrupats en grups.

[3] El programa ens mostra el prompt i interpreta les nostres comandes, també conegut com a *shell*.

i `fitx2` ja existeix.

- `mv` (MoVe)

  Mou fitxers i/o directoris ("moure" vol dir copiar i esborrar l'original, mantenint propietats com permisos i data d'última modificació). Admet els arguments de `cp` que hem vist. També s'usa per canviar de nom un fitxer.

- `df` (Disk Free)

  (vol dir Disk Free): sense arguments, escriu l'espai disponible a tots els dispositus muntats a l'arbre de directoris general (en particular, us permet saber tot el que hi ha muntat i a on). Si li passeu com a argument qualsevol fitxer o directori, us mostra l'espai disponible al dispositiu on és aquest fitxer o directori (a més de quin dispositiu és i a on està muntat).

- `rm fitx1 fitx2` (ReMove)

  Esborra els fitxers `fitx1` i `fitx2`. Amb l'opció `-r`, esborra també directoris i el seu contingut.

- `mkdir` (MaKe DIRectory)

  Crea directori(s).

- `rmdir` (ReMove DIRectory)

  Esborra directori(s).

1.4 Una sessió d'exemple

A continuació teniu una sessió d'exemple per exercitar les comandes que hem vist. Tot el que hi hagi darrera un caràcter `#` és un comentari i no ho heu d'escriure.

```
cd                      # ens posem al directori home
pwd                     # ara el veiem
cd /bin                 # ens canviem al directori /bin
pwd                     # voilà
cd ..                   # canviem al directori pare de /bin
pwd                     #    que és /
cd                      # tornem al directori home
pwd
```

```
ls -ld /bin/ls /bin     # /bin/ls es un fitxer
                        # /bin un directori
unalias cp              # per desactivar la protecció de
                        #    sobreescriptura
mkdir dirproves         # fem un direcori anomenat "dirproves"
touch f1 f2             # creem dos fitxers f1, f2
ls -l f1 f2             # ... que ara veiem
cp f1 f2                # sobreescrivim f2 amb f1
cp -i f1 f2             # pregunta abans de matxacar
ls -l dirproves         # dirproves és buit
cp -i f1 f2 dirproves   # hi copiem f1 i f2
ls -l dirproves         # ... que ara s'han de veure
cp -i f1 f2 dirproves   # f1, f2 ja són a dirproves
                        #    demana confirmació abans de matxacar
cp f1 f2 dirproves      # matxaca sense preguntar
rm dirproves/*          # esborra tots els fitxers de dirproves
touch a1                # fem un fitxer "a1"
ls -l . dirproves       # llista el directori de treball
                        #    i dirproves
cp * dirproves          # copia tots els fitxers del directori
                        #    de treball a dirproves,
                        #    es queixa perquè no pot
                        #    copiar directoris.
ls -l dirproves
rm dirproves/*
cp f? dirproves         # només copiem f1, f2
ls -l dirproves         # ara ho veiem
rm dirproves/
ls -l . dirproves
mv f1 dirproves         # f1 desapareix de . i passa a dirproves
ls -l . dirproves
mv f2 f3                # f2 passa a ser f3, matxacant continguts
ls -l
rmdir dirproves         # no podem!
ls -l dirproves         # és perquè no és buit
rm dirproves/*
rmdir dirproves         # ara sí

touch f1 f2
mkdir dirproves
cp * dirproves
```

```
ls -l . dirproves
rm -r dirproves         # esborrem dirproves recursivament
ls -l                   # ... ja no hi és
```

2 Configuració de les aules del SID

Quan obriu una sessió Linux a un ordinador de les aules del SID, el vostre directori home és de la forma

```
/home/samba/homes/<NIU>
```

on `<NIU>` és el vostre NIU. Aquest directori es troba físicament al disc dur de l'ordinador on esteu treballant. Tot el que guardeu en ell **no** ho podreu veure des de cap altre ordinador, ni des del mateix ordinador sota Windows. A més, el contingut d'aquest directori s'esborra periòdicament.

Podeu accedir al vostre directori personal (permanent) dins el servidor des del subdirectori `smbhome` del vostre directori home. És a dir, a

```
/home/samba/homes/<NIU>/smbhome
```

on `<NIU>` és el vostre NIU. Podreu veure tot el que guardeu aquí des de qualsevol altre ordinador del SID, i també des de Windows. L'espai és limitat, i molt més petit que el que teniu disponible al directori home (`/home/samba/homes/<NIU>`). Si executeu la comanda `df`, podreu verificar aquesta informació.

3 Ús del compilador de C

Emprarem el compilador de C del projecte GNU, que és el compilador standard als sistemes Linux (també el podeu instal·lar a Mac OS X). De fet és el mateix compilador que fa servir Dev–C++ sota Windows. Es diu `gcc`, i una invocació típica és:

```
gcc -g -Wall -pedantic -o executable font.c -lm
```

El significat de cada argument és el següent:

- `-g` indica al compilador que inclogui informació simbòlica, de manera que el programa final es pugui executar sota un debugger.
- `-Wall` indica al compilador que escrigui tots els warnings. Per defecte, el compilador no avisa de moltes construccions sintàcticament correctes però potencialment molt perilloses, que en la major part dels casos han estat escrites per error.

- `-o executable` indica a l'enllaçador que guardi el codi executable produït dins un fitxer anomenat `executable`.
- `-pedantic` indica al compilador que limiti la sintaxi que admet a l'standard ANSI C del 1990.
- `font.c` és el fitxer amb el codi font C.
- `-lm` indica a l'enllaçador que enllaci contra la biblioteca matemàtica (que conté funcions com `sin()` o `fabs()`. Per una qüestió històrica, als sistemes Unix les funcions matemàtiques estan separades de la biblioteca general, tot i que ANSI C especifica que formen part de la biblioteca standard).

Podeu canviar `-g` per `-O3`, que vol dir "optimitzar a nivell 3". Llavors no podreu executar el programa sota el debugger,[4] però, depenent de com sigui el codi, el temps d'execució es reduirà notablement (són freqüents factors entre 2 i 3, de vegades s'aconsegueix més). És aconsellable fer servir `-O3` quan s'està segur de la correcció del codi i es vol fer una execució llarga.

4 Ús del debugger

Disposeu d'un debugger desenvolupat també pel projecte GNU, que s'anomena `gdb`. Funciona amb línia de comandes. Per exemple, per començar una sessió de debugging amb el programa generat a dalt, heu de fer

```
gdb executable
```

Llavors us apareixerà un prompt "`(gdb)`" i podeu entrar comandes.

Per executar pas a pas, heu de començar per posar un breakpoint a la rutina `main()`:

```
b main
```

Llavors podeu engegar l'execució amb:

```
r
```

Observareu com s'atura a `main()`. Podeu veure a quin punt del programa us trobeu fent

[4] Bé, sí que el podreu executar sota el debugger, però només podreu veure instruccions-màquina i no el codi font. Podreu veure el codi font C si useu simultàniament `-g` i `-O3`. No obstant, això no és recomanable, perquè molts processos d'optimització donen lloc a reordinacions del codi, i això pot donar lloc a sessions de depuració molt confuses

```
bt
```

(vol dir "backtrace"). Per a avançar l'execució una línia de codi, hi ha dues comandes: la comanda

```
s
```

(vol dir "step"), i la comanda

```
n
```

(vol dir "next"). Es diferencien en què, quan la línia de codi que executeu conté crides a funcions escrites per vosaltres, la comanda `s` entra dins el cos de totes elles per executar-les pas a pas, mentre que `n` les executa d'una atacada (sense entrar-hi).

Si voleu continuar l'execució del programa fins a un altre lloc, hi heu de posar un altre breakpoint ("`b 150`" per parar a la línia 150, "`b jacobi`" per parar a la rutina `jacobi()`, etc) i llavors escriure la comanda

```
c
```

(vol dir "continue").

Podeu veure el contingut de qualsevol variable amb la comanda

```
p
```

(vol dir "print"). Per exemple, per veure el contingut de la variable `index` escriuríeu "`p index`", i per veure la segona component del vector `u` posaríeu "`p u[1]`". De fet la comanda `p` no serveix només per veure el contingut de variables, sinó que admet qualsevol expressió. Així, si voleu canviar el valor de la segona component del vector `u`, podeu fer "`p u[1]=3`".

Quan engegueu `gdb`, si al directori on ho feu hi ha un fitxer anomenat `.gdbinit`,[5] `gdb` executa les comandes que hi hagi dintre abans de passar-vos el control. Així, si poseu (dins `.gdbinit`)

```
b main
r
```

llavors, només engegar `gdb`, començarà l'execució del programa, que s'aturarà a la rutina `main()`.

Una situació en la qual el debugger és especialment útil és quan el programa acaba anormalment amb el missatge "`Segmentation fault`". Dues

[5]El punt forma part del nom del fitxer. Els fitxers amb noms que comencen per punt són "ocults": la comanda `ls` no els mostra a no ser que empreu l'opció `-a`.

de les causes més freqüents d'aquest error són accedir a un vector amb un índex fora de rang o usar un punter amb una adreça invàlida (p.ex. sense inicialitzar). Si llavors executeu el programa sota el debugger sense cap breakpoint, quan s'aturi podreu veure la línia de codi on s'ha produit el `Segmentation fault` amb la comanda `bt`. Aleshores, amb la comanda `p` podeu veure valors d'índexs, etc., per tal d'esbrinar si és aquesta línia on s'ha produït l'accés incorrecte a la memòria.[6]

Recordeu: la millor manera de depurar un programa és no equivocar-se. Moltes vegades és més eficient repassar el codi un parell de vegades que precipitar-se amb el debugger. Com a mínim, sempre l'heu de rellegir atentament una vegada després d'haver-lo escrit.

Trobareu més informació fent "`info gdb`" des d'una terminal (en particular, trobareu una sessió d'exemple molt instructiva).

5 Ús de gnuplot

És un programa per graficar, basat en línia de comandes. Permet dibuixar funcions definides per expressions que involucrin funcions elementals (com fa Maple, p.ex.), però la seva potència radica en la facilitat amb la que es poden representar fitxers de dades.

5.1 Gràfiques planes

Per a `gnuplot`, un fitxer de dades és un fitxer de text que consta de diverses línies, on cadascuna conté les coordenades d'un punt a representar. Un fitxer de dades pot constar de diversos "datablocks", que estan separats per línies en blanc.

Per a gràfiques bidimensionals s'usa la comanda `plot`. Per exemple, suposem que tenim un fitxer de text anomenat `dades.txt` amb contingut

```
# Aixo es un comentari
# Primer datablock
0 1 0
1 1 1

# Segon datablock
```

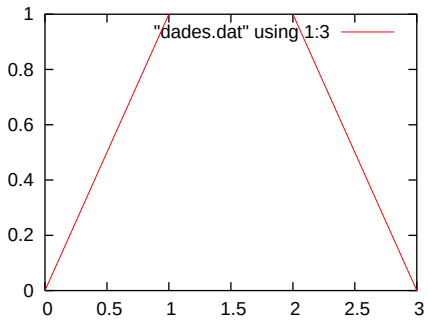
[6]Heu de tenir en compte que això no sempre passa: pot ser que en un punt del programa feu un accès invàlid a memòria que "desestabilitzi" el sistema de gestió de memòria dinàmica. Això pot fer terminar el programa amb un segfault no aquí sinó més endavant. En aquest cas, l'error és molt més difícil de detectar.

```
2 1 1
3 1 0
```

Si entrem dins `gnuplot` i escrivim

```
plot "dades.txt" using 1:3 with lines
```

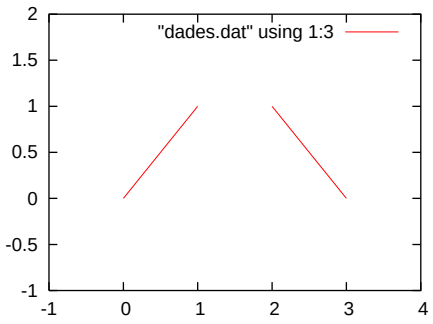
gnuplot recorrerà el fitxer, per a cada línia considerarà el punt que té el valor de la columna 1 com a coordenada x, i el valor de la columna 3 com a coodenada y, i unirà amb línies els punts consecutius del cada datablock (o sigui, a cada línia en blanc "aixequem el llapis"). El dibuix que obtindrem és:



Observeu que els rangs s'han ajustat perquè hi càpiguin exactament els 4 punts que cal representar. Els podem ampliar fent

```
set xrange [-1:4]; set yrange [-1:2]
replot
```

i obtenim



A `with lines` podeu canviar "`lines`" per "`points`", "`linespoints`", "`dots`", "`impulses`" i altres. Podeu veure què fan provant les següents ordres:

```
set xrange [-pi:3*pi]
set autoscale y
plot sin(x) with lines
plot sin(x) with points
plot sin(x) with linespoints
plot sin(x) with dots
plot sin(x) with impulses
```

Sempre i quan no hi hagi lloc a confusió, les comandes de `gnuplot` es poden abreujar. Així, es pot fer

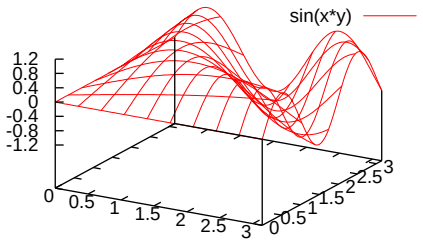
```
set autoscale
plot "dades.txt" u 1:3 w l
```

5.2 Gràfiques tridimensionals

Per representar gràfiques tridimensionals cal fer servir la comanda `splot`. Per exemple, si feu

```
set xrange [0:pi]; set yrange [0:pi]
splot sin(x*y) w l
```

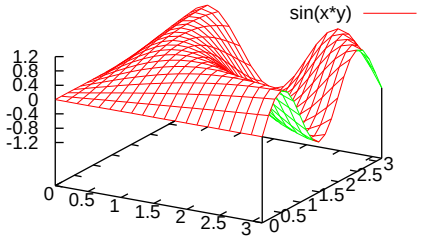
obtindreu



Es veurà una mica millor elimineu superfícies ocultes i augmenteu el número de mostres:

```
set hidden3d
set isosamples 20
replot
```

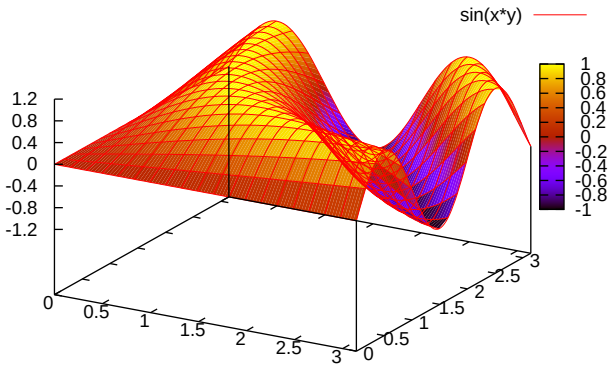
s'obté



L'opció `pm3d` representa un gradient de color a sobre de la superfície d'acord amb el valor de la coordenada z. Així, si feu

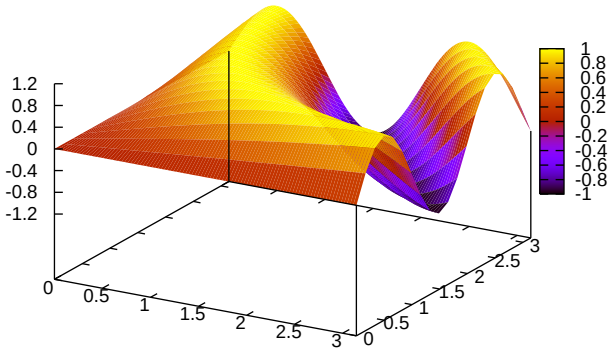
```
unset hidden3d
set pm3d
replot
```

s'obté



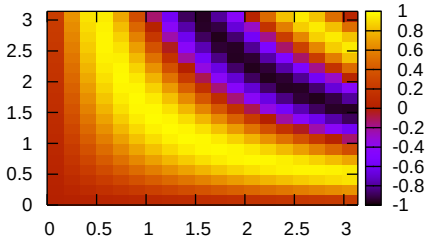
A un dibuix de gradient de color no voldreu veure la malla:

```
unset surface
replot
```



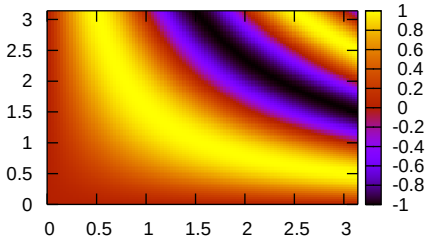
De fet, a un dibuix de gradient de color, habitualment tampoc no voldreu veure la superfície, sinó el gradient de color directament al pla xy. Això es pot fer mitjançant

```
set view map
replot
```



En un cas com aquest, voldreu pujar la resolució.

```
set isosamples 100
replot
```



Per tal de dibuixar fitxers de dades, la sintaxi és anàloga a la de la comanda `plot`, només que cal especificar tres columnes en comptes de dues. Els punts del el fitxer de dades no poden estar de qualsevol manera: els heu de posar ordenats, recorrent la graella de la superfície que voldreu representar per files o per columnes, i heu d'inserir una línia en blanc cada cop que canvieu de fila o de columna.[7] Així, per generar un fitxer per representar la gràfica de $\sin(xy)$ com hem estat fent, podeu escriure, en C

```
#include <stdio.h>
#include <math.h>

int main (void) {
   double dx, dy;
   int i, j;
   dx=M_PI/50; dy=M_PI/50;
   for (j=0; j<=50; j++) {
      for (i=0; i<=50; i++)
         printf("%.16G %.16G %.16G\n", i*dx, j*dy, sin(i*dx*j*dy));
      printf("\n");
   }
   return 0;
}
```

Si guardeu aquest codi dins un fitxer `tmp.c`, compileu amb

```
gcc -o tmp -g -Wall tmp.c -lm
```

executeu amb

```
./tmp > tmp.txt
```

i dins gnuplot feu

```
unset pm3d      # no volem gradient de color
set view        # tornem a vista per defecte
set surface     # si no no la veuriem!!
splot 'tmp.txt' u 1:2:3 w l
```

tornareu a veure la primera superfície que hem dibuixat (però amb més resolució).

[7] Per separar datablocks cal emprar *dues* línies en blanc.

5.3 Sortida a fitxer

Una característica molt pràctica de gnuplot es que no només sap graficar a una finestra, sinó que també pot generar fitxers en diversos formats gràfics. Per exemple, per guardar el dibuix de $\sin(xy)$ dins un fitxer anomenat `dibuix.eps` en PostScript encapsulat, heu d'escriur

```
set pm3d map
unset surface
set term post eps color solid  # Format: PostScript encapsulat
set out 'dibuix.eps'           # Sortida a dibuix.eps
splot 'tmp.txt' u 1:2:3 w l
set out           # Per tancar el fitxer dibuix.eps
set term x11      # Per tornar a dibuixar per pantalla.
                  # Podria ser: wxt , windows , o altres.
                  # Veureu quina heu de fer servir a la darrera
                  # línia que escriu gnuplot quan l'engegueu,
                  # abans de passar-vos el control.
```

Llavors, des d'una terminal, podeu mirar el dibuix que acabeu de generar fent

```
gv dibuix.eps
```

(dins gnuplot podeu escriure "`!gv dibuix.eps`"). També podeu convertir aquest fitxer a qualsevol altre format gràfic amb la comanda `convert`. Per exemple, per convertir-lo en PNG heu de fer (a una terminal, o dins gnuplot però amb "`!`" al davant)

```
convert -density 150 -units PixelsPerInch dibuix.eps dibuix.png
```

Les opcions "`-density 150 -units PixelsPerInch`" fan que la resolució del PNG final sigui de 150 píxels per polçada.[8][9] Podeu visualitzar el PNG que heu generat amb la comanda `display`.

És probable que el PNG que genereu amb la comanda anterior no sigui amb fons blanc, sinó amb fons transparent (en aquest cas `display` us mostrará un fons de tauler d'escacs). Això pot ser útil p.ex. per incloure el gràfic dins una pàgina web, però en moltes ocasions pot ser un inconvenient. Si voleu un PNG amb el mateix fons que l'EPS original, heu de desactivar la transparència:

[8] Una polçada són 2.54 cm.

[9] Per defecte, `gnuplot` fa els dibuixos en PostScript encapsulat de 5 polçades d'amplada i 3.5 polçades d'alçada. Això es pot canviar amb la comanda `set size` (per a més informació, feu "`help set size`" dins gnuplot).

```
convert -density 150 -units PixelsPerInch -alpha off \
    dibuix.eps dibuix.png
```

5.4 Animacions

Anem a veure com fer animacions senzilles amb gnuplot a partir d'un exemple. Suposem que volem fer una pel·lícula en la que s'alternen dos triangles isòscel·les, un apuntant cap a amunt i un altre cap a avall, a intervals de 0.5 segons. Per a això, escrivim un fitxer `triangles.txt` amb dos datablocks, cadascun amb els vèrtexs del triangle corresponent:

```
# vertexs primer triangle
0 0
0.5 1
1 0
0 0      # repetim el primer vertex perque tanqui

# vertexs segon triangle
0 1
0.5 0
1 1
0 1
```

Dins gnuplot, podem dibuixar per separat cadascun dels datablocks fent

```
set xrange [-1:2]; set yrange [-0.1:1.1]
plot 'triangles.txt' every :::0::0 w l  # dibuixa el primer datablock
plot 'triangles.txt' every :::1::1 w l  # dibuixa el segon datablock
```

(per a més informació, feu dins gnuplot "`help plot datafile every`"). Llavors, per a animar-lo, escrivim dins un fitxer `anim.gnu` les comandes necessàries,

```
plot 'triangles.txt' every :::(a%2)::(a%2) w l
pause 0.5
a=a+1
if (a<amax) reread   # nomes volem mostrar amax frames
```

("`reread`" vol dir "saltar al començament del fitxer") i, dins gnuplot, fem

```
a=0
amax=10
load 'anim.gnu'
```