Master di II livello in Calcolo Scientifico 2011-2012
Laboratorio di Visualizzazione

In questa pagina sono raccolti i principali argomenti da me trattati nel corso *Laboratorio di Visualizzazione* per il *Master di II livello in Calcolo Scientifico* 2011-2012. L'obiettivo del corso è fornire agli studenti gli strumenti di base per la visualizzazione e la gestione di dati tramite i software `GNUPLOT` e `MATLAB`.

GNUPLOT

Installazione

- **Linux (Ubuntu)**: aprite *Ubuntu Software Center*, cercate *gnuplot* e cliccate su installa (dovete digitare la password di *root*). In alternativa aprite un terminale e digitate
  `sudo apt-get install gnuplot` (anche qui dovete digitare la password di *root*).

- **Windows**: scaricate *gnuplot* da qui, decomprimete il file gp444win32.zip e posizionate la cartella decompressa *gnuplot* dove vi pare (in questo esempio in C:\). A questo punto cliccate col pulsante destro del mouse su *Risorse del Computer -> Proprietà -> Avanzate -> Variabili di Ambiente*. Nell'elenco *Variabili di Sistema* selezionate *Path* e cliccate su *Modifica*. Aggiungete in coda alla stringa (es. C:\windows\system32;...) un punto e virgola *;* e digitate
  `C:\gnuplot\binary\`
  Cliccate *ok* e ritornate al desktop. A questo punto *gnuplot* è riconosciuto come comando globale, accessibile da qualunque cartella. Per verifica cliccate su *Start -> Esegui*. Digitate *cmd* per aprire un terminale. Infine digitate *gnuplot*.

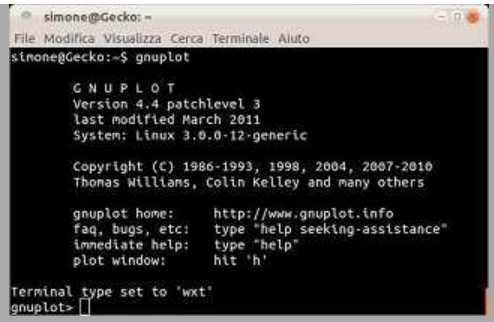
Torna all'indice

Apertura

- **Linux**: aprite un terminale e digitate
  `gnuplot`

- **Windows**: cliccate su *Start -> Esegui* e digitate *cmd* per aprire un terminale.
  Infine digitate
  `gnuplot`

Una volta avviato *gnuplot* vi mostra alcune informazioni (tipo il numero di versione) ed è pronto a ricevere comandi (notate il prompt *gnuplot*>).



Torna all'indice

Per cominciare

Tenete sempre a portata di mano il manuale di *gnuplot* (lo trovate qui in formato pdf), oppure chiedete aiuto direttamente a *gnuplot*, digitando

```
help
```

inserendo successivamente un argomento o un sotto argomento tra quelli disponibili (provate provate...).

In particolare potete interrogare *gnuplot* sulla sintassi di uno specifico comando, digitando

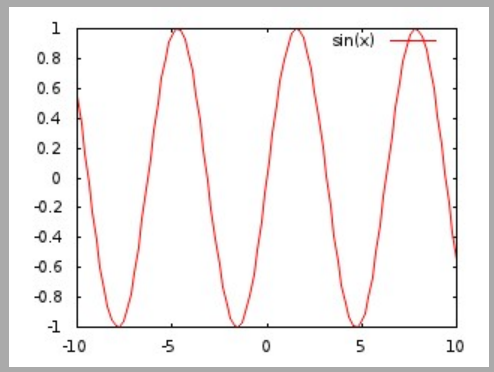
```
help comando
```

Torna all'indice

I primi plot: funzioni di una variabile

Il comando principale di *gnuplot* è sicuramente `plot`. Come vedremo esso può essere usato in molti modi differenti a seconda dei parametri che riceve in input.

Il modo più semplice consiste nel *plottare* il grafico cartesiano di una funzione di una variabile, definita da una espressione analitica. Ad esempio




```
plot sin(x)
```

Nota: la variabile `x` (così come `y,t,u,v`) è una variabile riservata (detta *dummy variable*) che permette a *gnuplot* di capire che volete plottare il grafico di una funzione di `x`. Se non avete un buon motivo per cambiare i nomi delle variabili dummy, utilizzate sempre `x` (e `y,t,u,v` per altri tipi di grafici che vedremo in seguito). Se invece `x` proprio non vi piace, provate col comando `set dummy nomevariabile`. Ad esempio

```
set dummy homer
plot homer**3
```
(in *gnuplot* il doppio asterisco ** indica l'elevamento a potenza)

Attenzione: da questo momento in poi `x` non è più definita, quindi (ad esempio) il comando `plot tan(x)` produrrà l'errore `undefined variable: x`. Nel dubbio, digitate il comando

```
show dummy
```

per sapere quali sono le variabili dummy correnti:

```
dummy variables are "homer" and "y"
```

Nota: la y si usa per i grafici cartesiani di funzioni di due variabili, per modificarla come variabile dummy si usa (occhio alla virgola!)

```
set dummy , nomevariabiley
```

oppure

```
set dummy nomevariabilex, nomevariabiley
```

per cambiare sia `x` sia `y`. Per ripristinare le variabili dummy di default (ma anche *tutti* gli altri parametri di *gnuplot* che avete eventualmente modificato col comando `set`) digitate

```
reset
```

A questo punto potete provare a plottare funzioni sempre più complicate, scrivendo una espressione che contenga composizioni di funzioni elementari. Ad esempio

```
plot x*cos(log(1+abs(x**2)))*exp(-sin(x))
```

A pagina 25 del manuale trovate una lista delle funzioni implementate in *gnuplot*.

Osservate che i grafici finora prodotti hanno tutti per dominio l'intervallo [-10,10]. Possiamo ovviamente cambiarlo utilizzando il comando `set xrange[xmin:xmax]`. Ad esempio

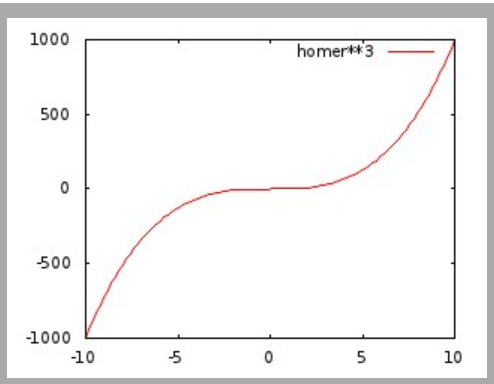
```
set xrange[0:2*pi]      (la variabile pi è riservata e vale pi greco)
plot cos(x)
```

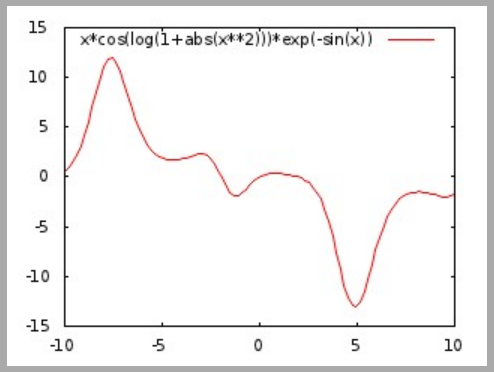
Il range per la variabile `y` viene per default impostato tra il valore minimo ed il valore massimo che la funzione plottata assume nel dominio (in questo caso tutti i valori in [-1,1]). Per cambiare anche questo range possiamo usare il comando `set yrange[ymin:ymax]`, ricordando però che, se la funzione assume valori al di fuori dal range impostato, non tutto il grafico sarà visibile. In ogni momento potete chiedere a *gnuplot* quali siano i valori di range utilizzando i comandi `show xrange` e `show yrange`.

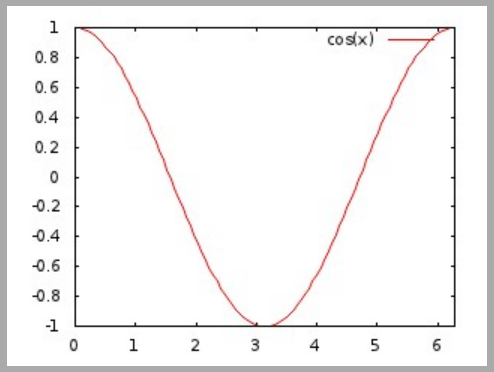
Supponiamo ora di voler plottare più di un grafico alla volta. È sufficiente elencare tutte le funzioni in un singolo comando `plot` separando le varie espressioni con delle *virgole*. Ad esempio

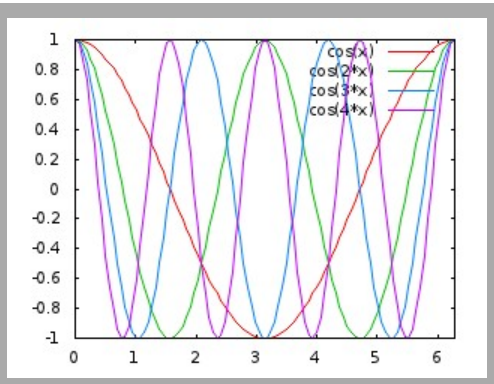
```
plot cos(x), cos(2*x), cos(3*x), cos(4*x)
```

Notate come *gnuplot* assegni ad ogni funzione un colore differente e riporti il nome ed il colore in una legenda in alto a destra. Vedremo in seguito come assegnare ai nostri plot colori arbitrari. Se invece volete disabilitare la legenda (che può rendere difficile la lettura del grafico) utilizzate il comando `unset key`. Per ulteriori informazioni sul comando `key` consultate il manuale a pagina 114 oppure digitate in *gnuplot* il comando `help key`.









Torna all'indice

Variabili, funzioni e iterazione

gnuplot vi offre la possibilità di definire *variabili* e *funzioni* personalizzate, il cui utilizzo vi permette di semplificare notevolmente la scrittura dei vostri *codici gnuplot*. Come vedremo in seguito, esse permettono inoltre di utilizzare *gnuplot* come un (semplice) linguaggio di programmazione, ad esempio per ottenere grafici animati.

Per definire una variabile è sufficiente dichiararla con il comando `nomevar=valorevar`, facendo attenzione (tutto si può fare) a non sovrascrivere una variabile già esistente o una variabile riservata. Ad esempio

```
a=0.5
b=1.5
e=exp(1)
log2=log(2)
```

Ogni variabile può essere utilizzata in una qualunque espressione, anche come parametro di un comando. Ad esempio
`set xrange[a:b]` equivale in questo caso a `set xrange[0.5:1.5]`.

Per controllare in ogni momento quali variabili avete definito, digitate il comando `show variables` ed otterrete un output come questo

```
User and default variables:
pi = 3.14159265358979
NaN = NaN
GNUTERM = "wxt"
a = 0.5
b = 1.5
e = 2.71828182845905
log2 = 0.693147180559945
```

Analogamente, per definire una funzione è sufficiente dichiararla con il comando `nomefunz(nomevar)=espressione`. Ad esempio

```
runge(x)=1/(1+x**2)
```

Potete anche definire una funzione che dipende da più variabili e/o parametri con
`nomefunz(nomevar1,nomevar2,...,nomeparam1,nomeparam2,...)=espressione`. Ad esempio

```
retta(x,m,n)=m*x+n
```

A questo punto potete plottare i grafici delle vostre funzioni utilizzando (notate in questo caso che `x` è la variabile dummy, mentre i parametri `m,n` vengono assegnati)

```
plot runge(x), retta(x,0,0.5), retta(x,1,0)
```

Supponete ora di voler plottare 11 rette passanti per l'origine con un coefficiente angolare che varia da 0 a

10 con passo 1. Il modo delirante per farlo è

```
plot retta(x,0,0), retta(x,1,0), retta(x,2,0), retta(x,3,0), retta(x,4,0),
retta(x,5,0), retta(x,6,0), retta(x,7,0), retta(x,8,0), retta(x,9,0), retta(x,10,0)
```

Il modo furbo invece consiste nell'utilizzare il comando `for[parametro=minimo:massimo]` combinato con `plot` in questo modo

```
plot for[m=0:10] retta(x,m,0)
```

Nota: la variabile di iterazione (`m` in questo caso) deve essere intera, così come i valori `minimo` e `massimo`. Per iterare valori non interi dovete inventarvi qualche trucco, tipo questo

```
f(x,p)=x**p
plot for[m=1:100] p=0.1*m f(x,p)
```

che plotta tutte le potenze di `x` con esponente `p` tra 0.1 e 10 con passo 0.1.

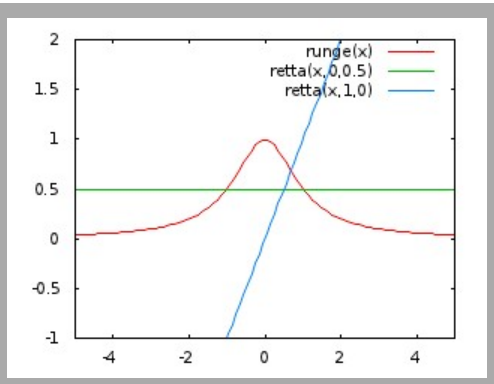
Attenzione: l'iterazione termina sempre alla prima virgola separatrice! Ad esempio il seguente codice

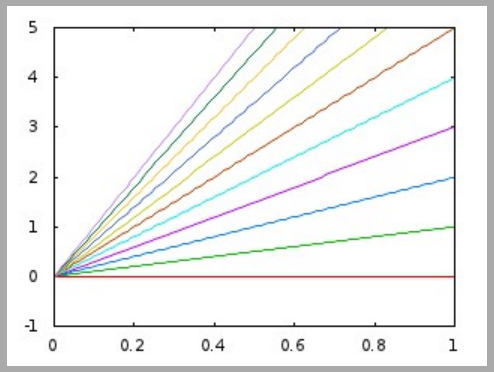
```
plot for[m=1:3] sin(m*x), cos(m*x)
```

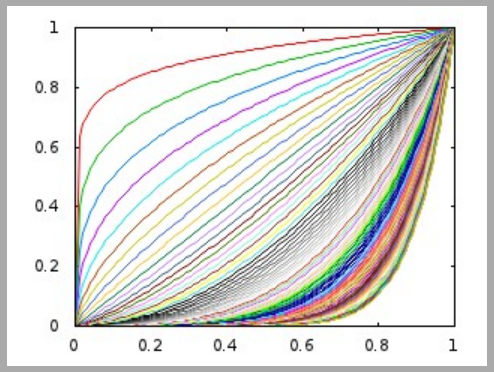
produrrà tre sinusoidi ed una sola cosinusoide, corrispondente all'ultimo valore di iterazione `m=3`, proprio perché l'iterazione termina con la prima virgola. Per ottenere tre sinusoidi e tre cosinusoidi dovremmo invece scrivere

```
plot for[m=1:3] sin(m*x), for[m=1:3] cos(m*x)
```

Per ulteriori approfondimenti sul comando di iterazione `for` consultate il manuale a pagina 84.







Torna all'indice

Curve parametriche nel piano

Una curva nel piano può essere definita tramite una coppia di equazioni parametriche del tipo `(x(t),y(t))`, dove `x` e `y` sono funzioni del parametro `t` che varia in un intervallo `[t0,t1]`. Per plottare curve in *gnuplot* occorre passare in modalità parametrica utilizzando il comando

```
set parametric
```

Notate l'output:

```
dummy variable is t for curves, u/v for surfaces
```

gnuplot vi informa che in modalità parametrica `t` è la variabile dummy per le curve, mentre `u` e `v` sono le variabili dummy per le superfici parametriche (vedi

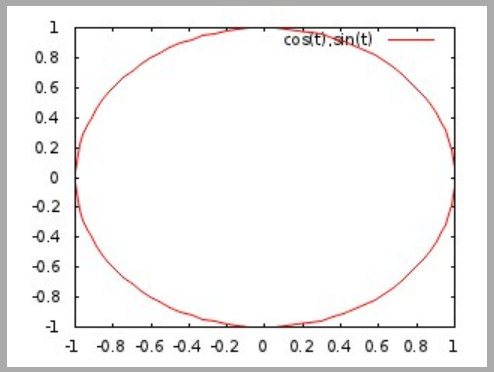
dopo), esattamente come x è la variabile dummy per le funzioni di una variabile ed x e y sono variabili dummy per le funzioni di due variabili (vedi dopo).

Attenzione: per tornare in modalità cartesiana occorre digitare il comando

```
unset parametric
```

Un semplice esempio di curva nel piano è la circonferenza di raggio unitario, che può essere parametrizzata tramite la coppia di equazioni

```
x(t)=cos(t)
y(t)=sin(t)
```



dove il parametro `t` varia in `[0,2pi]`. Impostiamo dunque il range per `t` utilizzando il comando (analogo a `xrange`)

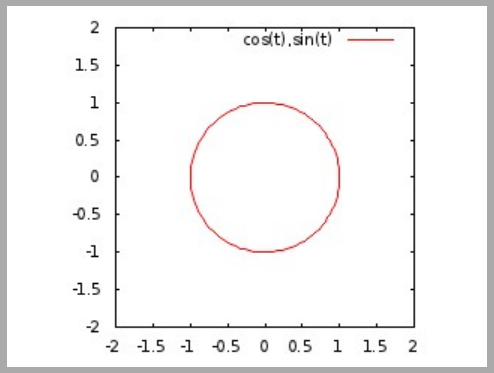
```
set trange[0:2*pi]
```

Plottiamo quindi la curva fornendo al comando `plot` le due equazioni parametriche separate da una virgola

```
plot cos(t), sin(t)
```

Notate che il risultato sembra un'ellisse e non una circonferenza. Questo dipende dal fatto che gli assi cartesiani non sono scalati di default in maniera uniforme. D'altra parte la circonferenza è tangente ai bordi del *bounding box* della figura, proprio perché `xrange` e `yrange` sono settati di default per contenere tutto il plot. Per risolvere il problema della scalatura non uniforme utilizziamo il comando

```
set size square
```



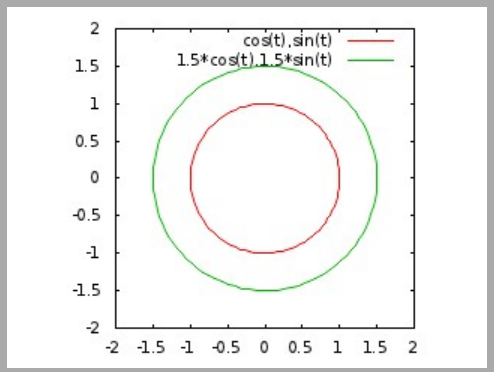
Per cambiare invece il bounding box utilizziamo (come già visto precedentemente)

```
set xrange[-2:2]
set yrange[-2:2]
```

e riplottiamo il grafico usando il comando

```
replot
```

Nota: `replot` ripete l'ultimo comando `plot` inserito (nel nostro caso `plot cos(t), sin(t)`). Esso può anche essere utilizzato per aggiungere un nuovo grafico al precedente, fornendo le nuove equazioni. Ad esempio, aggiungiamo una circonferenza di raggio 1.5 a quella già disegnata:




```
replot 1.5*cos(t), 1.5*sin(t)
```

che equivale, nel nostro caso, ad una forma compatta di (notate il ruolo differente tra le virgole che definiscono le coppie di equazioni e la virgola che separa i due grafici)

```
plot cos(t), sin(t) , 1.5*cos(t), 1.5*sin(t)
```

Proviamo ora qualcosa di più complicato. Disegnamo una spirale che faccia 2 giri completi, con un raggio variabile da 0 a 2. Innanzitutto impostiamo `trange` opportunamente:

```
set trange[0:4*pi]
```

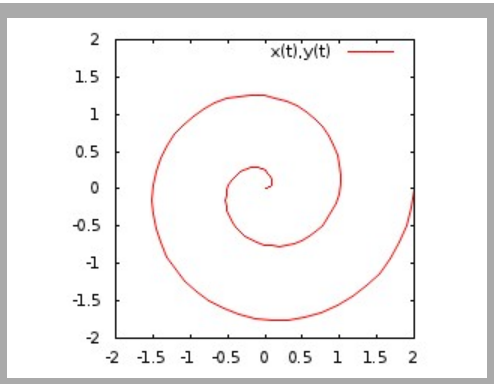
Utilizziamo due funzioni personalizzate per definire le due equazioni parametriche:

```
x(t)=t/(2*pi)*cos(t)
y(t)=t/(2*pi)*sin(t)
```

Notate che la definizione di `x(t)` e `y(t)` non sovrascrive le variabili dummy `x` e `y`, proprio perché le prime

sono funzioni, le seconde variabili. Infine notate che il fattore di riscalamento `t/(2*pi)` varia linearmente tra 0 e 2 al variare di `t` tra 0 e 4pi, come rischiesto. Plottiamo:

```
plot x(t), y(t)
```

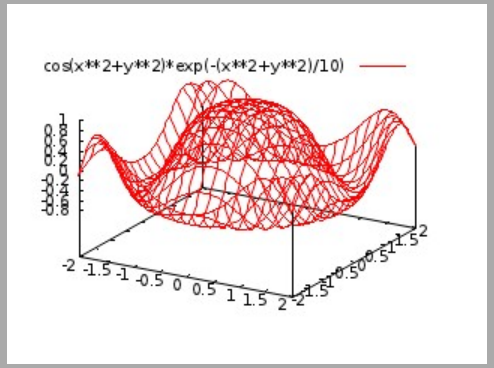


Torna all'indice

Funzioni di due variabili

Per plottare il grafico di funzioni di due variabili si utilizza il comando `splot` (che sta per *surface plot*), la cui sintassi è analoga a quella per i plot di funzioni di una variabile. Ricordate che in modalità cartesiana (`unset parametric`) le variabili dummy sono sempre `x` e `y` (a meno che non le abbiate modificate). Vediamo un esempio di `splot`

```
splot cos(x**2+y**2)*exp(-(x**2+y**2)/10)
```



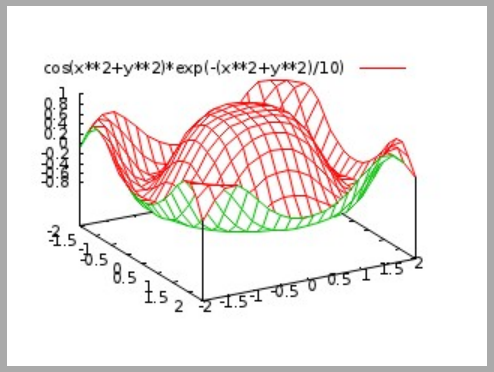
Il grafico viene plottato come una maglia di fil di ferro (*wireframe*). Potete ruotare la figura 3D tenendo premuto il pulsante sinistro del mouse e trascinando il cursore all'interno della finestra grafica.

Talvolta la visualizzazione wireframe può risultare poco chiara. Per ovviare a questo problema potete abilitare/disabilitare (tramite `set/unset`) un algoritmo di rimozione delle linee nascoste, in grado appunto di camuffare le linee che non sarebbero visibili dalla vostra visuale se il grafico fosse *solido*, utilizzando colori diversi per le due facce del grafico. Utilizzate il comando `hidden3d`

```
set hidden3d
replot
```

Notate che `replot` si applica anche in 3D e ripete l'ultimo comando `splot` inserito. Tuttavia esso va utilizzato in maniera coerente. Infatti se l'ultimo comando inserito è ad esempio un `plot` di una funzione di una variabile, digitando `replot x**2-y**2` otterrete un errore

```
undefined variable: y
```



proprio perché il grafico 2D precedente è incompatibile con quello 3D appena inserito.

Anche per i grafici di due variabili possiamo utilizzare funzioni personalizzate, eventualmente dipendenti da uno o più parametri. Ad esempio possiamo definire

```
sella(x,y)=x**2-y**2
piano(x,y,a,b)=a*x+b*y
```

e plottare, anzi, *splottare*

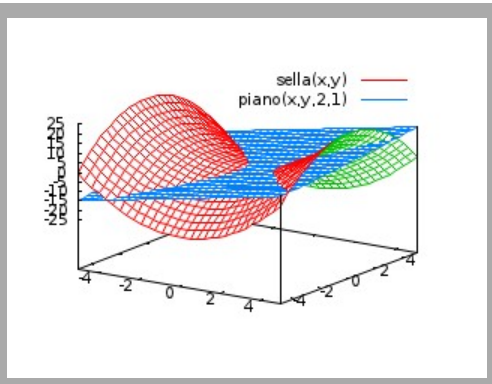
```
splot sella(x,y), piano(x,y,1,1)
```

Come per le funzioni di una variabile, anche per i grafici di funzioni di due variabili possiamo riscalare gli assi cartesiani in maniera uniforme. In questo caso però non si utilizza `set size square`, ma

```
set view equal xy
```

E se volessimo disegnare 10 piani orizzontali paralleli di equazione `z(x,y)=c` con `c=0,...,9` ? Provate con

```
z(x,y,c)=c
splot for[c=0:9] z(x,y,c)
```



Torna all'indice

Curve e superfici parametriche nello spazio

Una curva nello spazio può essere definita tramite una terna di equazioni parametriche del tipo `(x(u),y(u),z(u))`, dove `x`, `y` e `z` sono funzioni del parametro `u` che varia in un intervallo `[u0,u1]`. Come parametro ho scelto di usare `u` invece di `t`, proprio perché il primo è per default la variabile dummy per il comando `splot` in modalità parametrica. Ricordiamoci sempre di passare in modalità parametrica quando vogliamo plottare questo tipo di grafici

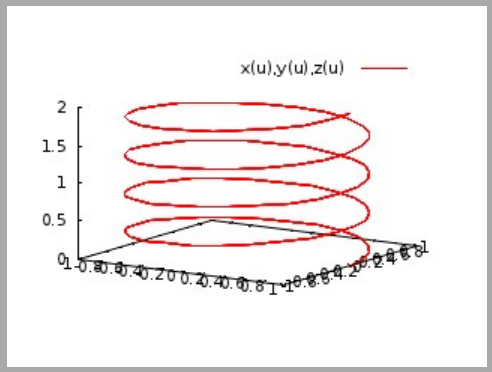
```
set parametric
```

Come esempio di curva nello spazio, proviamo a disegnare un'*elica* che faccia 4 giri completi, con un raggio che decresca da 1 e 0 e che abbia un'altezza complessiva di 2. Partiamo dalla solita circonferenza di raggio unitario che giace nel piano `z=0`, di equazioni parametriche

```
x(u)=cos(u)
y(u)=sin(u)
z(u)=0
```

e facciamo variare il parametro `u` come richiesto

```
set urange[0:8*pi]
```



Per default *gnuplot* plotta le superfici al di sopra del piano xy. Per riportare lo zero dell'asse z sul piano xy utilizziamo il comando

```
set ticslevel 0
```

A questo punto modifichiamo l'equazione in `z` in modo che la circonferenza non sia percorsa 4 volte, ma si sollevi fino all'altezza desiderata:

```
z(u)=u/(4*pi)
```

Splottiamo!

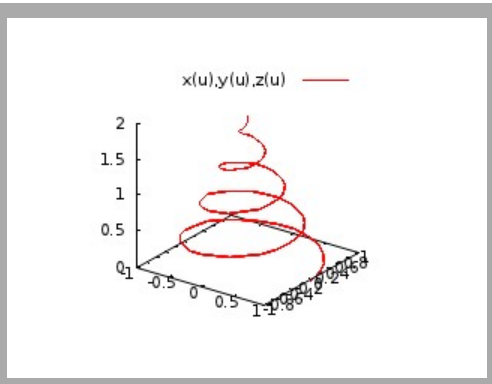
```
splot x(u),y(u),z(u)
```

Infine modifichiamo il raggio della circonferenza in modo che decresca linearmente da 1 a 0 al crescere di z tra 0 e 2. Otteniamo le seguenti equazioni (notate l'utilizzo della funzione personalizzata `radius(u)` per semplificare la scrittura del codice):

```
radius(u)=(2-u/(4*pi))/2
x(u)=radius(u)*cos(u)
y(u)=radius(u)*sin(u)
z(u)=u/(4*pi)
```

Vediamo il risultato con

```
replot
```



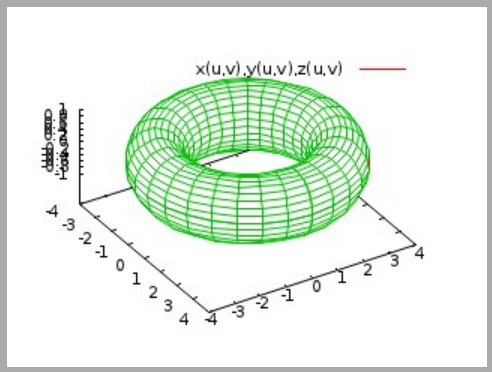
Passiamo alle superfici.

Il principio è lo stesso delle curve, ma in questo caso abbiamo bisogno di una terna di equazioni parametriche del tipo `(x(u,v),y(u,v),z(u,v))`, dove `x`, `y` e `z` sono funzioni dei parametri `u` e `v` che variano rispettivamente negli intervalli `[u0,u1]` e `[v0,v1]`. Ricordiamo che in modalità parametrica `u` e `v` sono le variabili dummy per il comando `splot`.

Vediamo due esempi di superfici nello spazio.

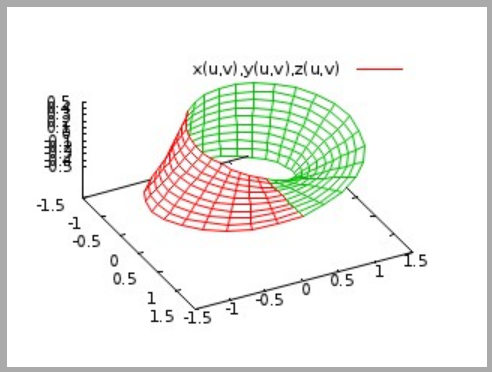
Il primo esempio è un *toro*, noto anche come *ciambella*!

```
x(u,v)=(3+cos(u))*cos(v)
y(u,v)=(3+cos(u))*sin(v)
z(u,v)=sin(u)
set urange[0:2*pi]
set vrange[0:2*pi]
splot x(u,v),y(u,v),z(u,v)
```



Il secondo esempio è un nastro di *Möbius*.

```
x(u,v)=(1+.5*v*cos(u/2))*cos(u)
y(u,v)=(1+.5*v*cos(u/2))*sin(u)
z(u,v)=.5*v*sin(u/2)
set urange[0:2*pi]
set vrange[-1:1]
splot x(u,v),y(u,v),z(u,v)
```



Nota: dal momento che il nastro di Möbius è una superficie non orientabile (*con una sola faccia!*), i due colori generati dal comando di rimozione delle linee nascoste `hidden3d` si incontrano a metà della superficie.

Torna all'indice

Lettura di dati da file

Tipicamente i grafici che si vogliono visualizzare sono il risultato di misurazioni o di simulazioni numeriche. Difficilmente (leggi *praticamente mai*!) si ha a disposizione l'espressione analitica di una funzione che descriva correttamente un fenomeno o rappresenti la soluzione di un problema. Abbiamo invece a che fare con *dati*, che possono rappresentare qualsiasi cosa, siano essi punti, posizioni, velocità, pressioni, temperature,...

gnuplot è in grado di leggere e visualizzare dati da file, purché rispettiate alcune regole di base e purché i vostri dati siano organizzati in maniera opportuna.

Consiglio: raccogliete i vostri file di dati in una cartella apposita e assicuratevi di essere in tale cartella quando lavorate con *gnuplot*. Ecco due modi per farlo, sia sotto linux, sia sotto windows.

1) Quando aprite il terminale, prima di digitare `gnuplot`, raggiungete la cartella prescelta utilizzando il comando `cd` (change directory), ad esempio nel mio caso

`cd /home/simone/Scrivania/laboratorioVis` su linux, oppure
`cd c:\laboratorioVis` su windows

A questo punto avviate *gnuplot* e controllate di essere nella cartella giusta col comando `pwd`, che restituisce appunto il percorso corrente all'interno dell'albero delle cartelle.

2) Direttamente da gnuplot, una volta avviato, usate il comando `cd` seguito dal percorso della cartella racchiuso tra apici singole `'` o doppie `"`, ad esempio nel mio caso

`cd "/home/simone/Scrivania/laboratorioVis"` su linux, oppure
`cd "c:\laboratorioVis"` su windows

Per sicurezza controllate di essere nella cartella giusta col comando `pwd`.

Consideriamo ora un esempio semplice. Partendo dal tempo `t=0` misuriamo la nostra temperatura corporea ad intervalli di 1 secondo per 9 secondi. Otteniamo una sequenza di numeri possibilmente compresa tra 36 e 40. Apriamo un nuovo file di testo che nominiamo `dati1.txt` e riportiamo in colonna tutti i numeri registrati, *andando a capo* dopo ogni record.

Il file `dati1.txt` si presenta come segue (lo trovate qui):

```
# dati1.txt - temperature a intervalli di un secondo
36.5
36
36.5
37
38
38.5
37.5
36.5
36
36.5
```

Il carattere cancelletto **#** viene riconosciuto da *gnuplot* come l'inizio di un commento e pertanto *viene ignorato* in fase di caricamento *insieme a tutta la riga che lo segue*.

Quando *gnuplot* legge un file di questo tipo, per default interpreta i dati come come punti del piano xy in cui l'ascissa è uguale all'indice di riga del record letto (partendo da 0), mentre l'ordinata è il valore del record letto.

Nel caso del file `dati1.txt`, le coppie di coordinate generate saranno quindi `(0,36.5) (1,36) (2,36.5) (3,37) ...`

Per plottare tale file digitiamo il comando `plot "nomefile"` oppure `plot 'nomefile'`, ricordando che le apici singole e quelle doppie sono praticamente equivalenti.

Come già detto precedentemente, *gnuplot* modifica automaticamente il *bounding box* del grafico per contenere tutti i dati plottati. Poiché i punti che cadono sui bordi della figura potrebbero non essere ben visibili, regoliamo prima i range e poi plottiamo il grafico:

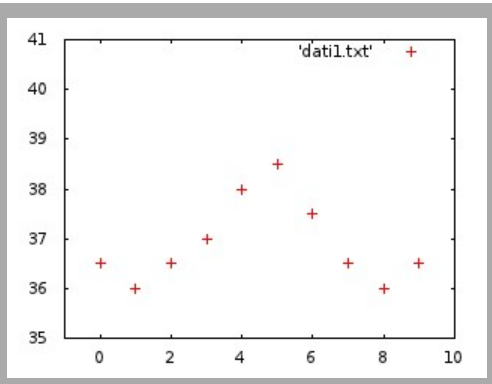
```
set xrange[-1:10]
set yrange[35:41]
plot 'dati1.txt'
```

Per default *gnuplot* disegna i singoli dati come punti. Se invece vogliamo che ogni dato sia unito al successivo con una linea (ovvero che i dati vengano *interpolati linearmente*) possiamo usare il comando

```
plot 'dati1.txt' with lines
```

oppure, abbreviando

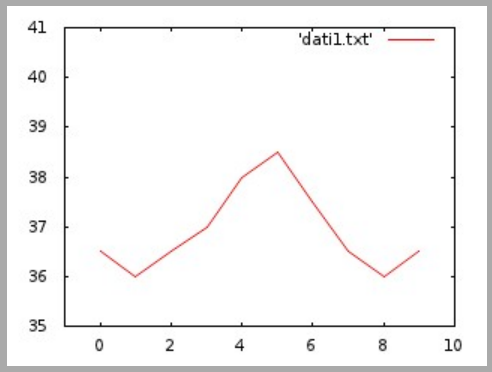
```
plot 'dati1.txt' w l
```



In *gnuplot* ogni comando o parametro può essere abbreviato (in alcuni casi anche con una singola lettera), purché non ci sia ambiguità con un altro comando o parametro (in tal caso viene segnalato un errore).

Investite sulla vostra pigrizia: trovate il numero minimo di lettere per scrivere tutti i comandi studiati finora (ad esempio `set hid` abbrevia `set hidden3d`).

Vedremo in seguito altri utilizzi del parametro `with`.



Supponiamo ora di voler registrare le temperature di un'altra persona e di volerle confrontare con le nostre. Possiamo senz'altro creare un nuovo file (diciamo `dati1b.txt`) che contiene i nuovi valori e plottarli insieme, come nel caso di più funzioni, col comando

```
plot 'dati1.txt' w l, 'dati1b.txt' w l
```

Tuttavia, se le persone di cui volete monitorare la temperatura sono tante, può essere conveniente utilizzare un solo file, memorizzando più colonne di valori, una per ogni persona.

Apriamo ad esempio un nuovo file `dati2.txt` e scriviamo due colonne di valori, la prima presa da `dati1.txt`.

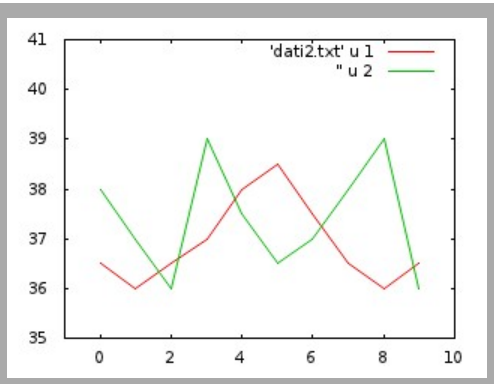
Attenzione: i record di ogni riga *devono* essere separati da *almeno* uno spazio.

Il file `dati2.txt` si presenta come segue (lo trovate qui):

```
# dati2.txt - 2 temperature a intervalli di un secondo
36.5 38
36 37
36.5 36
37 39
38 37.5
38.5 36.5
37.5 37
36.5 38
36 39
36.5 36
```

A questo punto vogliamo che *gnuplot* generi 2 sequenze di coppie di coordinate, rispettivamente

`(0,36.5) (1,36) (2,36.5) (3,37) ...` e `(0,38) (1,37) (2,36) (3,39) ...`



Si utilizza il parametro `using` seguito dal numero della colonna da cui vogliamo leggere i dati (partendo da 1). Quindi

```
plot 'dati2.txt' using 1 with lines, 'dati2.txt' using 2 with lines
```

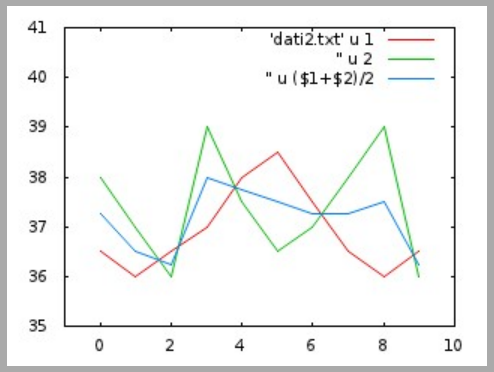
oppure, abbreviando

```
plot 'dati2.txt' u 1 w l, '' u 2 w l
```

dove il doppio apice vuoto `''` è una abbreviazione per l'*ultimo file caricato* (in questo caso `dati2.txt`).

E se volessimo calcolare la media aritmetica delle due temperature ad ogni tempo? Potremmo inserire nel file una terza colonna contenente tutti i valori `(T1+T2)/2`, dove `T1` e `T2` sono i valori presi rispettivamente dalla prima e dalla seconda colonna. In tal caso, il comando di plot sarebbe

```
plot 'dati2.txt' u 1 w l, '' u 2 w l, '' u 3 w l
```



Tuttavia *gnuplot* ci semplifica la vita nel caso in cui serva un dato derivato da altri tramite espressioni o funzioni elementari. Infatti, ogni record letto viene associato ad una variabile speciale `$n`, dove `n` è il numero della colonna cui il record appartiene. Possiamo pertanto utilizzare queste variabili speciali per eseguire i calcoli che ci interessano *al volo*. Nel caso della media aritmetica della prima (`$1`) e della seconda (`$2`) colonna procediamo come segue:

```
plot 'dati2.txt' u 1 w l, '' u 2 w l, '' u ($1+$2)/2 w l
```

Notate la differenza tra l'indice di colonna (`1,2,...`) e la variabile speciale (`$1,$2,...`) che accede al valore contenuto nella colonna.

Consideriamo ora un caso più generale, in cui i nostri dati siano ad esempio punti nel piano, ovvero coppie di coordinate del tipo `(x0,y0) (x1,y1) (x2,y2) ...` Apriamo un nuovo file `dati3.txt` e riportiamo una coppia di coordinate per riga, separando le coordinate con (*almeno*) uno spazio.

Il file `dati3.txt` si presenta come segue (lo trovate qui):

```
# dati3.txt - coppie di coordinate nel piano
0 0
0.1 0.5
0.25 0.7
0.35 1
0.5 2
0.7 0
1 1
1.3 2.7
1.5 3
2 1
```

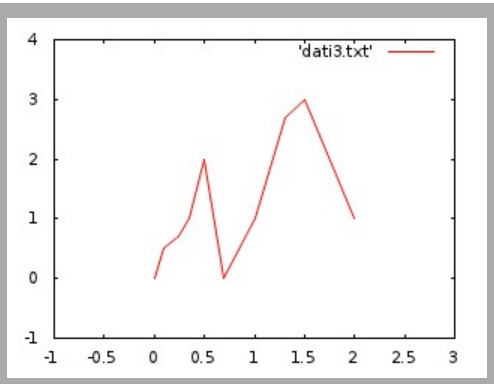
A questo punto *gnuplot* è in grado di riconoscere automaticamente il nuovo formato di dati. È sufficiente digitare (modifichiamo i range per una migliore visualizzazione)

```
set xrange[-1:3]
set yrange[-1:4]
plot 'dati3.txt' w l
```

Proseguiamo aprendo un nuovo file `dati4.txt` (lo trovate qui) in cui aggiungiamo una terza colonna di

valori. La struttura del file è la seguente:

```
x0 y10 y20
x1 y11 y21
x2 y12 y20
...
```

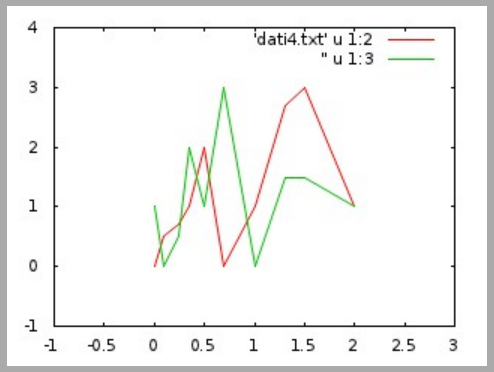


Utilizzando il parametro `using colonnax:colonnay` possiamo chiedere a *gnuplot* di generare le due sequenze di coordinate `(x0,y10) (x1,y11) (x2,y12) ...` e `(x0,y20) (x1,y21) (x2,y22) ...`:

```
plot 'dati4.txt' u 1:2 w l, '' u 1:3 w l
```

E se volessimo confrontare la differenza in valore assoluto tra le due sequenze? Utilizziamo le variabili speciali di colonna:

```
plot 'dati4.txt' u 1:(abs($2-$3)) w l
```



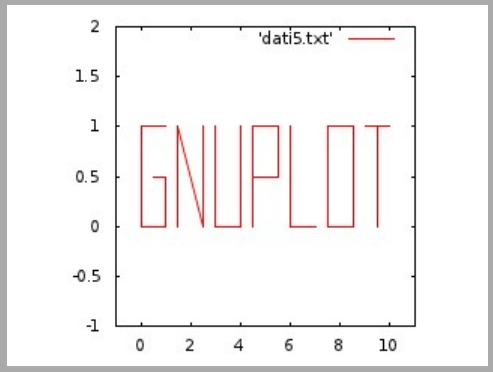
vale a dire *plotta il file dati4.txt utilizzando la colonna 1 per l'ascissa ed il valore assoluto della differenza tra i valori della colonna 2 ed i valori della colonna 3 per l'ordinata*.

Attenzione: racchiudete sempre tra parentesi l'espressione che coinvolge le variabili speciali di colonna, altrimenti *gnuplot* potrebbe commettere un errore di interpretazione.

Abbiamo visto come il parametro `with lines` produca un plot che interpola linearmente tutti i dati letti da file. Supponiamo ora di voler creare delle interruzioni di linea, ovvero di voler interpolare linearmente solo gruppi di dati. Per farlo è sufficiente *separare* tali gruppi nel file *tramite UNA riga vuota*.

Aprite quindi un nuovo file `dati5.txt` (lo trovate qui) che rispetti questa struttura:

```
x0 y0
x1 y1
x2 y2
...
xn yn

xn+1 yn+1
xn+2 yn+2
...
xm ym
```



Provate quindi a plottare con il comando

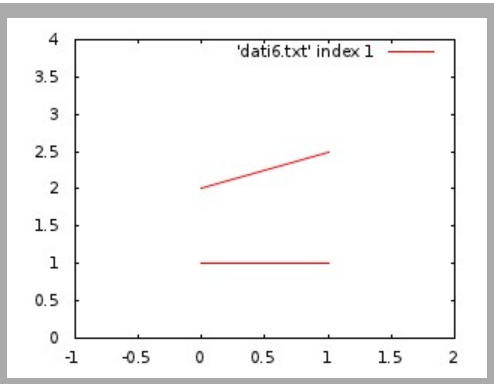
```
plot 'dati5.txt' w l
```

Se invece volete che i vostri dati siano suddivisi *concettualmente* in sottogruppi (non solo per quanto riguarda l'interpolazione) è sufficiente *separarli* nel file *tramite DUE righe vuote*.

Aprite quindi un nuovo file `dati6.txt` (lo trovate qui) che rispetti questa struttura:

```
x0 y0
x1 y1
x2 y2
```

```
...
xn yn


xn+1 yn+1
xn+2 yn+2
...
xm ym
```



Per accedere ad un gruppo di dati si utilizza il parametro `index indicegruppo`, dove `indicegruppo` è appunto l'indice (partendo da 0) che identifica gruppi di dati compresi tra coppie di righe vuote successive.

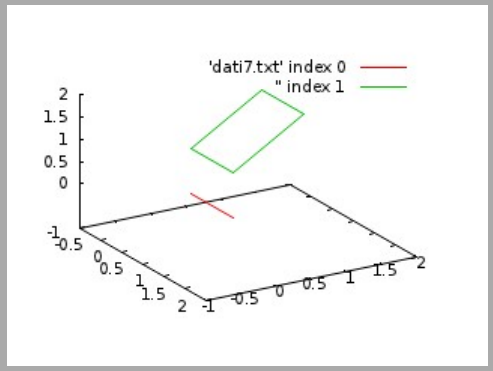
Ad esempio, per accedere al *secondo* gruppo di dati nel file `dati6.txt` digitate (ricordate sempre che `index` parte da 0!)

```
plot 'dati6.txt' index 1 w l
```

Vediamo ora come utilizzare il *separatore di linea* e il *separatore di indice* nel caso di dati tridimensionali.

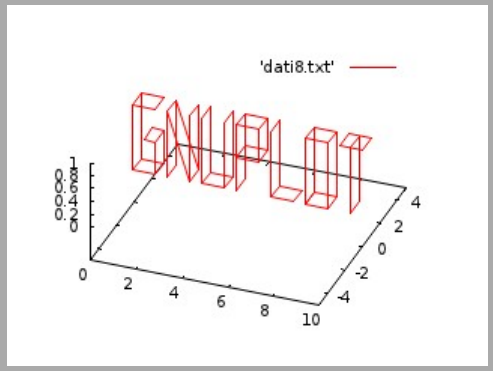
Aprite un nuovo file `dati7.txt` (lo trovate qui) del tipo

```
x0 y0 z0
x1 y1 z1
x2 y2 z2
...
xn yn zn


xn+1 yn+1 zn+1
xn+2 yn+2 zn+2
...
xm ym zm

xm+1 ym+1 zm+1
...
xp yp zp
```



In questo caso il file presenta due gruppi di dati (accessibili tramite `index 0` e `index 1`).
Il primo gruppo di dati consiste in due vertici che definiscono una linea orizzontale.
Il secondo gruppo di dati consiste invece in quattro vertici, divisi a coppie da un separatore di linea.
L'effetto di tale separatore è quello di creare non due linee separate, ma una cella che ha per vertici i 4 vertici del gruppo:

```
splot 'dati7.txt' index 0 w l, '' index 1 w l
```



Tenendo a mente queste semplici regole sintattiche potete organizzare gruppi di dati sempre più complessi. Se il vostro file si presenta strutturato ma non specificate nessun indice di gruppo in fase di plot, *gnuplot* disegnerà tutti i dati contemporaneamente, rispettando però i separatori (come nel caso del file `dati8.txt` che trovate qui) :

```
splot 'dati8.txt' w l
```

Torna all'indice

Curve di livello

IN COSTRUZIONE

Dettagli grafici: campioni, stili, vista

IN COSTRUZIONE

Comandi utili

IN COSTRUZIONE

File di comandi gnuplot

IN COSTRUZIONE

Animazioni

IN COSTRUZIONE

C++ e gnuplot

IN COSTRUZIONE

Elenco dei comandi studiati

IN COSTRUZIONE

Torna all'indice