# Breve guida alle attivitá di aula
# Elementi di Cartografia Digitale e GIS
# Corso in Scienze Geologiche
# Laurea triennale L-34
# Universitá di Genova

*Marino Vetuschi Zuccolini* [1]

[1] Laboratorio di Geochimica, DISTAV - Universitá di Genova

9 novembre 2020

# Indice



# Elenco delle figure

# Elenco delle tabelle

### Convenzioni grafiche

### Testi di riferimento

### Siti di riferimento

GMT
GRASS
proj
Ghostscript
GSHHG
ETOPO1

### (Bash shell) Comandi citati

source; sleep; awk ; sed ; ls; pwd; cd; echo; head; tail; wc ; more; cat ; paste; gunzip; tar; ps2pdf; which; for-do-done cycle; ">"; "»"; "|"

### (PROJ) Comandi citati

proj

### (GMT) Comandi citati

psbasemap, pscoast, gmtinfo, psxy, pshistogram

### (GRASS7) Comandi citati

**d-group** d.mon; d.rast; d.erase

**g-group** g.region; g.rename; g.remove; g.copy

**v-group** v.in.ascii; v.out.ascii; v.in.ogr; v.to.points; v.overlay; v.surf.idw; v.surf.rst; v.in.region; v.patch; v.proj ; v.concave.hull; v.build.polylines; v.db.addtable; v.to.db; v.db.select; v.in.ascii; v.info; v.digitize; v.build; v.to.points; v.buffer; v.extract

**r-group** r.mapcalc; r.in.gdal; r.to.vect; r.mask; r.contour; r.proj; r.colors; r.patch; r.digit; r.flow; r.plane; r.sum; r.colors, r.surf.fractal; r.to.vect; r.volume

**i-group** i.group; i.target; i.rectify

Genova, 9 novembre 2020

Marino Vetuschi Zuccolini

**Version history**

30 ottobre 2020 - v0.9.2 Aggiornato esercizio Lago Trasimeno con pianificazione posa condotta e dragaggio- distribuita agli studenti

29 ottobre 2020 - v0.9.1 Aggiunto esempio Lago Trasimeno - distribuita agli studenti.

17 ottobre 2020 - v0.9.0 Aggiunta breve introduzione a Virtualbox e ai comandi di shell - distribuita agli studenti.

14 ottobre 2020 - v0.9.0 Prima versione: Processing di dati geologici, creazione della prima cartografia basata su GMT (Atacama Trench), creazione di databases di GRASS in WGS84 e SIRGAS19 - no figures.

# Introduzione

In questo documento sono raccolte alcune delle attivitá pratiche del corso di Elementi di cartografia digitale e GIS del Corso di Scienze Geologiche dell'Universit'į Genova per l'anno 2020/2021. E' da sottolineare che é da considerare come un ausilio alle lezioni pratiche che si tengono in aula informatica o che si possono seguire in remoto (o ancora con in registrazione) e non un manuale esaustivo per cui si rimanda la materiale del corso raccolto in Aulaweb.

Le specificitá della sintassi dei comandi mostrati con altre varianti di utilizzo sono spiegate a voce in maniera piú esaustiva durante le lezioni. L'approfondimento viene demandato alla cura dello studente mediante la ricerca delle informazioni nelle pagine di manuale specifiche o sui siti specifici.

Tra l'altro l'impostazione di un criterio di ricerca di un problema specifico su un motore di ricerca é un esercizio particolarmente efficace per analizzare il problema ed estrarre le informazioni strettamente necessarie per ottenere le risposte a pi'ălto tasso di successo.

L'esercitazione viene condotta `step-by-step` cercando di mettere in luce l'uso diretto operativo dei comandi che piú comunemente si usano. Alcune soluzioni possono apparire eccessivamente semplificate, altre possono sembrare eccessivamente suddivise e poco fluide, ma in questa fase servono a comprendere l'uso di base della sintassi di shell e dei comandi impiegati. Servono anche guidare lo studente nell'impostazione di un problema complesso come somma di problemi deisamente piú semplici da affrontare. Successivamente lo studente troverá un suo modo di risolvere le problematiche con comandi diretti, stilisticamente soddisfacenti ed computazionalmente efficienti.

La parte pratica del corso si prefigge di introdurre lo studente:

- a come creare una macchina virtuale;
- al sistema operativo Linux attraverso l'uso operativo della console generando scripts di shell bash;
- all'uso di software specifico per la cartografia digitale come GMT e GRASS7, mettendo a comune le esperienze di semplice programmazione per la produzione di prodotti cartografici in formato pdf con ghostscript;
- alla produzione di una cartografia digitale sfruttando gli standard di conversione e trasformazione di coordinate mediante proj di comune uso nell'ambito GIS;
- a sviluppare nella parte finale del Corso un progetto, che a partire da materiale cartografico in supporto cartaceo relativo ad un'area sita in un paese extraeuropeo, permetta di generare un modello numerico del terreno;

## 0.2 Note brevissime sulla installazione di un OS Linux su Virtualbox

1. assicurarsi che da BIOS la macchina permetta la virtualizzazione;
2. scaricare VirtualBox;
3. scaricare un OS (Mint, Ubuntu Fedora...) in formato `.iso` e specifico per il processore installato sul pc (es. 32-bit o 64-bit);
4. avviare VirtualBox;
5. definire il tipo di OS che vogliamo installare;
6. definire lo spazio per il disco virtuale, 20 Gb potrebbe essere sufficiente per iniziare anche se sará possibile far variare dinamicamente lo spazio del disco rigido virtuale;
7. definire la RAM che l'emulatore utilizzerá;
8. installare il nuovo OS puntando il file `.iso` scaricato precedentemente;
9. una volta che una versione runtime di Linux sará avviata vi permetterá di avviare l'installer vero e proprio, avviatelo e seguite le istruzioni. **Suggerimento 1**: scegliete la lingua inglese, vi permetterá di fare esercizio e soprattutto vi faciliterá il recupero delle informazioni sulla rete quando incontrerete difficiltá (la maggior parte delle informazioni utili nelle pagine web é in inglese e dovreste altrimenti cercare di tradurre le voci dei menú che talvolta non sono proprio intuitivi). **Suggerimento 2**: l'utente che genererete al moment dell'installazione avrá i diritti di root, cioé di amministratore. Sarebbe bene che utilizzaste questo solo per le operazioni di sistema e appena terminata

l'installazione creaste un utente senza diritti di amministratore, questo dovrebbe proteggervi da possibili catastrofi es. cancellarsi il disco rigido senza accorgersene.

10. con il pacchetto `Synaptic` si puó scaricare ed installare in maniera molto semplice il software necessario;

## 0.3 Introduzione sprint alla shell Bash

Una volta che l'utente, dopo aver inserito user ID (per esempio `info`) e password, é entrato nel proprio computer (`miopc`) e ha disponibilitá del suo desktop deve attivare il Terminale (Console) per eseguire comandi di sistema e non, esclusivamente tramite tastiera.

Di default il terminale indirizza l'utente direttamente nella cartella di sua esclusiva proprietá che generalmente é localizzata nella cartella /`home`) e a seconda del tipo di shell e del profilo dell'utente presenta verrá presentato un prompt. Per essere sicuri di dove ci si trova é sufficiente digitare in corrispondenza del prompt `info@miopc` :~ il comando `pwd` (≡ present working directory)

```
info@miopc:~ pwd
/home/info
info@miopc:~
```

La struttura del filesystem di un sistema Linux é semplice (e rappresenta il modo in cui i dati sono organizzati logicamente sul disco rigido) e fa capo alla ad una directory particolare che indica il punto piú elevato da cui discende l'albero gerarchico di tutte le directory che vengono elencate con `ls` e viene identificato con /.

```
info@miopc:~ cd /
info@miopc:/
```

Ogni directory ha al suo interno altre cartelle che contengono tipologie differenti di dati (es. applicativi, file di configurazione, devices ecc. ecc.). Per rientrare nella propria cartella (chiamata "home") é possibile impiegare diversi modi tra cui specificare come argomento di `cd`,

- il percorso (path) completo ovvero /`home`/`info`, in realtá con questa specifica possiamo andare in ogni directory del filesystem;
- la variabile globale `$HOME`, che é una variabile di sistema e alloca il valore della home dell'utente attivo;
- la variabile globale `$OLDPWD`, che alloca la stringa relativa alla cartella da cui siamo transitati l'ultima volta, nel nostro caso /`home`/`info`.

ad esempio usando la variabile `$HOME`

```
info@miopc:/ cd $HOME
info@miopc:~
```

Siccome i comandi sono molti e possono essere usati semplicemente o con l'impiego (opzionale o obbligatorio) di argomenti e flags se non ci si ricorda che cosa quel comando specifico fa o come si usa é possibile richiamare un aiuto in linea con il comando `man` seguito da un argomento (il comando di cui richiediamo l'help). Per scorrere il testo si puó usare sia la barra delo spazio sia le frecce up-down sulla tastiera mentre per tornare al prompt é sufficiente digitare il carattere `q`.

```
info@miopc:~ man ls
LS(1)                          User Commands                          LS(1)
```

```
NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

 Manual page ls(1) line 1
info@miopc:~
```

La pagina del mannuale del comando ls oltre al NOME e della sua brevissima descrizione riporta la SYNOPSYS ovvero la sintassi di utilizzo. In particolare possiamo vedere la presenza delle OPTIONS tra parentesi quadre, cosí come di un FILE. Le parentesi quadre indicano che sia le opzioni che il nome del file sono opzionali per cui possono non essere messe a seguire del comando, in pratica non vi é obbligo di impiego.

Con ls abbiamo la possibilitá di fare la lista, pura e semplice, dei files (con una accezione molto generica, in quanto sono presenti anche directories e alias - gli analoghi dei Collegamenti di Windows - ad esempio). Aggiungendo alcune flags o argomenti l'output risulta piú completo come ad esempio digitando il comando `ls – lht` otteniamo un elenco a piú colonne ordinato per tempo di creazione, della cartella /usr, ed in cui le dimensioni di files sono in bytes (B, Kb, Mb...)

```
info@miopc:~ ls -lht /usr
total 368K
drwxr-xr-x   2 root root 100K May  5 21:27 bin
drwxr-xr-x 208 root root 132K May  5 21:27 lib64
drwxr-xr-x 256 root root  36K May  5 21:26 include
drwxr-xr-x   2 root root  20K May  5 21:25 sbin
drwxr-xr-x 132 root root  36K May  5 21:24 lib
drwxr-xr-x   9 root root 4.0K May  5 21:24 src
drwxr-xr-x 345 root root  12K Apr  6 15:17 share
lrwxrwxrwx   1 root root   10 Apr  6 14:21 tmp -> ../var/tmp
drwxr-xr-x  19 root root 4.0K Feb 18 22:35 local
drwxr-xr-x   5 root root 4.0K Feb 18 22:35 X11R6
drwxr-xr-x   5 root root 4.0K Feb 18 22:35 x86_64-suse-linux
drwxr-xr-x   2 root root 4.0K Feb 18 22:35 games
drwxr-xr-x   4 root root 4.0K Mar 12  2009 mpi
```

La prima colonna indica i permessi relativi a quel file indicando con la prima lettera se il file é una directory (d) oppure un alias (l) o ancora un file (–) a cui seguono xxx, il proprietario, il gruppo a cui appartiene il proprietario le dimensioni, la data ed il nome del file. La successione dei caratteri cella prima colonna é una ripetizione di tre caratteri in sequenza rwx che sta a significare che, per esempio per la directory bin, essa puŸó essere letta (r), scritta (w) ed eseguita (x, ovvero aperta) dal proprietario. La seconda sequenza r – x, indica che puó essere letta, non modificata (–) e aperta dal gruppo root cosí come gli stessi permessi sono affidati ad uno user che non sia root e che non faccia parte del gruppo root. E' da rimarcare che il precedente comando puó essere scritto in maniera equivalente nella forma ls – l – h – t /etc o anche ls – t – l – t /etc In console vengono digitati molti comandi una volta che ne viene fatto ampio utilizzo e ricordare la giusta sequenza di un comando digitato tempo addietro puó non essere agevole per cui il comando history viene in aiuto. Una delle caratteristiche della shell é quella tipica di molti linguaggi di programmazione per cui forme sintattiche che permettano l'esecuzione di loop ciclici (cicli for) oppure condiionali come (if-then-else). Ad esempio una forma del tipo riportata sotto permette l'esecuzione ripetitiva di un comando, in questo caso echo basandosi su un "contatore" composto da tre elementi

```
info@miopc:~ for i in 1 2 3
> do
> echo $i
> done
1
2
3
info@miopc:~
```

Bisogna anche sottolineare che il contatore 1 2 3 non ha significato numerale per cui la successione di tre numeri conseguenti é assolutamente ininfluente per il completo successo del ciclo-for. Cosa succede se sostituisco 3 2 1 a 1 2 3

E'importante notare come la sintassi di questo ciclo-for veda $i come argomento di echo. Il simbolo $ come nella dichiarazioni delle variabili globali (es. $HOME) viste all'inizio identifica una variabile, in questo caso di sola pertinenza di echo per la durata del ciclo for. La sua presenza Ŕ particolarmente importante inquanto in sua assenza avremmo una situazione di questo tipo:

```
info@miopc:~ for i in 1 2 3
> do
> echo i
> done
i
i
i
info@miopc:~
```

Se eseguiamo questo comando, peraltro abbastanza inutile, diverse volte otterremo sempre lo stesso risultato, per cui proviamo a sostituire alla sequenza numerale 1 2 3 una forma virgolettata di ls in cui le virgolette sono ottenute con la seuqnza di tast "AltGr Maiusc ' "

```
info@miopc:~ for i in ‘ls /usr‘ ; do echo $i; done
```

ottenendo come risultato

```
bin
games
include
```

```
lib
lib64
local
mpi
sbin
share
src
tmp
X11R6
x86_64-suse-linux
info@miopc:~
```

E'un elenco che abbiamo giá visto precedentemente solo che é arrangiato diversamente ma che puó essere customizzato a piacere semplicemente sostituendo la directory. Che cosa succede se sostituisco /var a /usr?

Sino a questo momento il risultato del ciclo for é stato sempre scritto a video, direttamente in console. Talvolta é necessario scrivere un file che contenga l'output di quanto eseguito. In lezioni precedenti si é imparato a re-dirigere l'output di un comando in un file (`fileout`) per cui proviamo ad utilizzarlo anche qui

```
info@miopc:~  for i in ‘ls /usr‘ ; do echo $i > fileout ; done
info@miopc:~
```

il risultato é sorprendente in quanto présente solo $x86_64$ – suse – linux invece che l'elenco completo, malgrado il comando non incontri alcuna difficoltá di esecuzione. Ció é dovuto al fatto che ogni qualvolta `echo` viene eseguito all'interno del ciclo-for esso sovra scrive il file, cancellando quindi quanto scritto prima. Per risolvere questo problema si usa un'altra forma di re-direzione usando il simbolo $\gg$

```
info@miopc:~ for i in ‘ls /usr‘ ; do echo $i >> fileout ; done
info@miopc:~
```

Abbiamo reso un pó piú generale un comando che puó essere riutilizzato per ottenere informazioni e file di output diversi in modo da far parte ad esempio di un medesimo file.

A questo punto impariamo ad eseguire uno o piú comandi di shell, almeno per il momento, scrivendo il primo script che chiameremo `prova.sh` con un editor di testo (es. gedit).

```
info@miopc:~ gedit &
info@miopc:~
```

Avviando un applicazione che fa uso della interfaccia grafica é utile digitare il nome dell'edito seguito dal simbolo $\&$ in modo da eseguire l'editor in background permettendo all'utente di inserire nuovi comandi nella console. Nel nuovo documento che poi salverete avrete cura di scrivere

```
#!/bin/bash
for i in ‘ls /usr‘ ; do echo $i >> fileout ; done
```

in modo da avere il ciclo-for che avete usato sino a questo momento in una file, preceduto da una riga che permetterá l'esecuzione da console una volta che avrete digitato

```
info@miopc:~ . ./prova.sh
info@miopc:~
```

che porterá al medesimo risultato. Il vantaggio di aver creato questo script é evidente nel momento in cui dobbiamo cambiare la directory di cui chiediamo l'elenco. Abbiamo visto difatti che per modificare il target del comando `ls`

dobbiamo scrivere materialmente il comando in console ogni qualvolta si deve eseguire. in un caso semplice come questo ció non é un problema che puó diventare notevole nel momento in cui la sequenza di comandi é particolarmente complessa. Per sopperire a difficoltá di questo genere modifichiamo `prova.sh` in questo modo

```
#!/bin/bash

dir=$1
for i in ‘ls $dir‘ ; do echo $i >> fileout ; done
```

eseguendo lo script nuovamente con l'accortezza di aggiungere la corretta sintassi per indicare una directory di interesse, ad es. `/usr`

```
info@miopc:~ . ./prova.sh /usr
info@miopc:~
```

In questo modo abbiamo ulteriormente generalizzato il comando di partenza senza che si debba modificare lo script tranne che nel caso non vengano aggiunte nuove opzioni. Con `dir = $1` andiamo a definire una variabile (`dir`) che prende il valore del primo argomento (`/usr`) dello script (`prova.sh`). Ed il suo utilizzo (`$dir`) come argomento di `ls` fa si che io possa eseguire la sequenza di comandi in `prova.sh` ottenendo che `fileout` contenga quanto richiesto. Un comando con la struttura fino ad ora modificato ha in se un baco che puó in determinate condizioni addirittura riempire il disco rigido sino a rendere impossibile ogni operazione. Per capire come questo sia possibile provate ad eseguire `prova.sh` con la giusta sintassi 3 volte ed ogni volta che ha terminato la sua esecuzione digitate il comando `wc – l`, che come sapete vi permettere di conoscere il numero di righe presenti in un file. Vedrete che il numero di righe tende ad aumentare sempre piú per via del fatto che >> re-direzione l'output del comando sul medesimo file. Una prima modifica consiste nel cancellare fileout prima del ciclo-for in modo da ricreare il file ad ogni esecuzione del ciclo-for.

Ragionando nello stesso modo visto in precedenza per permettere a `prova.sh` di scrivere dei file di output diversi a seconda della directory da elencare aggiungo una nuova variabile in grado di accettare un secondo argomento di `prova.sh`. Per cui il nuovo script diventerá

```
#!/bin/bash

#
# Uso: . ./prova.sh dir file
#      dove dir=directory da listare ; file=file da scrivere
#
#  Es: . ./prova.sh /etc file_etc
#  Es: . ./prova.sh $HOME $OLDPWD/file_etc
#
#
dir=$1
file=$2

rm $file
for i in ‘ls $dir‘ ; do echo $i >> $file ; done
```

E se volessi scrivere un file contenente non giá i nomi dei files residenti nella cartella `$dir` ma le dimensioni scritte in bytes? E' necessario modificare il comando ls compreso tra le virgolette ed aggiungere un awk uniti dal pipe "|" in questo modo

```
for i in ‘ls -lh $dir | awk ’{print $5}’‘ ; do echo $i >> $file ; done
```

## 0.4 Processing di un database testuale

L'esercizio si basa sull'elaborazione del file Magnitude_5.0_1960-2018.txt estratto mediante Geomapapp dalla repository relativa ai terremoti mondiali. Il catalogo contiene informazioni relative a 1333 eventi del periodo 1960-2018 con magnitudo $\geq 5.0$ nella scala Richter registrati sulla costa del Cile in corrispondenza della zona di subduzione conosciuta come Fossa di Atacama (Atacama Trench) o Fossa Perú-Cile (Perú-Chile Trench). Le informazioni sono arrangiate all'interno del file in formato ASCII (testuale) nel seguente formato a colonne:

```
Time Latitude Longitude Magnitude Depth Location Information
```

**Time:** 2018-04-27T02:01:05.140Z YYYY-MM-DDThh:mm:ss.dddZ é una stringa che riporta il momento dell'evento riferita all'ora UTC, ora di riferimento mondiale riferita al meridiano di Greenwich.

**Latitude:** -24.327 latitudine dell'epicentro in formato sessadecimale riferito al sistema WGS84. Il segno negativo per convenzione indica che siamo nell'Emisfero Sud

**Longitude:** -66.9088 longitudine dell'epicentro in formato sessadecimale riferito al sistema WGS84. Il segno negativo per convenzione indica che siamo ad Ovest del meridiano centrale di Greenwich.

**Magnitude:** 5 Magnitudo dell'evento nella scala Richter.

**Depth:** 163.77 Profonditá della zona ipocentrale.

**Location:** 61km WSW of San Antonio de los Cobres, Argentina localitá piú prossima all'epicentro.

**Information:** http://earthquake.usgs.gov/earthquakes/eventpage/us1000dt5j link del database USGS in cui sono riportate le informazioni specifiche relative all'evento.

## 0.5 Determinare la densitá informativa del database

Prima di iniziare ogni attivtá di creazione di un database relazionale spaziale a partire da sorgenti esterne in un sistema GIS é necessario conoscere le informazioni relative alla consistenza del database ovvero quanti record sono presenti e quali sono i range di variabilitá delle singole informazioni numeriche (le prime 5 colonne nel nostro caso):

```
wc -l Magnitude_5.0_1960-2018.txt
```

la linea richiama il comando `wc` a cui viene aggiunta una flag `-l` ed un argomento `Magnitude_5.0_1960-2018.txt` ed elenca il numero di righe (1334) presenti nel file il che ci suggerisce che oltre ai 1333 eventi é presente anche una riga di intestazione che é visibile con il comando:

```
head Magnitude_5.0_1960-2018.txt
```

che mostra sul terminale le prime 10 righe del file ed in cui si evince che i dati sono elencati con una struttura fissa, il che facilita le attivitá di processing. I tre comandi che seguono permettono di estrarre 1) il contenuto della prima colonna cosí come é presente nel database, 2) il contenuto della prima colonna ordinato in ordine crescente e 3) il contenuto della prima colonna eliminando la prima riga di intestazione e ordinamento in ordine crescente con creazione di un file di outoput `mag_chile.txt`.

```
awk '{print $4}' Magnitude_5.0_1960-2018.txt
awk '{print $4}' Magnitude_5.0_1960-2018.txt | sort
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $4 }' | sort -r  > mag_chile.txt
```

Con l'ultimo comando abbiamo modo di determinare anche i valori minimo e massimo della magnitudo scorrendo il file dalla prima all'ultima. In questa serie di comandi compaiono due caratteri che | e > che permettono di concatenare rispettivamente comandi in serie e ri-direzionare l'output in un file. Naturalmente la scansione di un file che puó essere diversi Gb di dimensione diventa problematico per cui per ottenere questa informazione immediatamente possiamo usare i comandi che seguono:

```
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $4 }' | ${GMT} gmtinfo
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $4 }' | ${GMT} gmtinfo -C
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $4,$5 }' | ${GMT} gmtinfo -C
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $4,$5 }' | \
gmt gmtinfo -C > mag_minmax.txt
```

in cui abbiamo implementato uno dei pacchetti di GMT, `gmtinfo` che restituisce immediatamente i valori richiesti. Per una parte dell'esercizio viene richiamata la versione 5 di GMT cosí come viene richiamato in Mac OS X. Linux ad esempio richiama gmt senta cifra identificativa della versione. In particolare il terzo comando restituisce anche i valori minimo e massimo delle profonditá in km dell'ipocentro ed il quarto scrive l'output in un file. Se si sostituisce `awk 'print 2,3 '` ad `awk print 4,5'` si determina l'estensione areale in termini di latitudine e longitudine. Sempre nell'ultimo comando il carattere \ dopo il secondo | i che permette di continuare il comando nella riga successiva.

Per consultare in maniera rapida il file direttamente in console é possibile utilizzare diversi comandi, lo studente puó provare con i comandi che seguono e osservare l'output in console:

```
cat Magnitude_5.0_1960-2018.txt
more Magnitude_5.0_1960-2018.txt
head Magnitude_5.0_1960-2018.txt
head -3 Magnitude_5.0_1960-2018.txt
tail Magnitude_5.0_1960-2018.txt
```

## 0.6 Creare una prima cartografia

Una volta che si sono ottenute le informazioni relative alle variabili di interesse (magnitudo e profonditá) possiamo affrontare l'aspetto geografico del problema determinando la proiezione piú adatta per rappresentare cartograficamente i dati presenti nel database. Con il comando che segue possiamo determinare analogamnente a prima i valori estremi di longitudine e latitudine e scriverli in un file:

```
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $2,$3 }' | \
gmt gmtinfo -C > mag_minmax.txt
```

Il pacchetto `psbasemap` si incarica di disegnare il frame che delimita l'area geografica di interesse in una scala adeguata e riportando una suddivisione dei bordi per aumentare la leggibilitá. I due blocchi che seguono sono perfettamente equivalenti grazie all'uso del ; che permette di eseguire il secondo comando in modalitá seriale una volta che il primo é stato completato, invece di dover digitare il tasto invio:

```
gmt psbasemap -R-71/-61/-25/-18 -B1g1/1g1 \
-Jm1:10000000 -V > frame.ps
ps2pdf frame.ps
```

Questo comando genera un frame in scala 1:10000000 secondo la proiezione di Mercatore (-Jm) con una suddivisione di 1 grado di ampiezza.

```
gmt psbasemap -R-71/-61/-25/-18 -B1g1/1g1 \
-Jm1:10000000 -V > frame.ps ; ps2pdf frame.ps
```

Sino ad ora abbiamo creato il frame geografico e con il prossimo comando dobbiamo rendere disponibili le informazioni alla morfologia costiera a risoluzione variabile presente nel file `gshhg-shp-2.tar.gz`. Il nome puó sembrare criptico ma puó essere letto in questo modo:

- gshhg: Global Self-Consistent, Hierarchical, High Resolution Geography Database e deriva dall'analisi, sintesi e processing di diversi database mondiali
- shp: formato grafico vettoriale ESRI shape;
- 2: e' la versione maggiore del database;
- .tar: ci suggerisce che non é un file unico ma una cartella composta da altri formattato in modo da essere visto dal filesiystem come un file unico;
- .gz: compresso in formato GNU zip.

Per riportarlo allo stato originale si procede prima decompattando il file con il primo comando e trasformando l'archivio nella cartella usuale con il secondo.

```
gunzip /home/docente/Documents/ECGIS_2020/gshhg-shp-2.tar.gz
tar -xvf /home/docente/Documents/ECGIS_2020/gshhg-shp-2-.tar
```

Il comando che segue oltre a generare il frame di riferimento aggiunge (grazie a `-K` nella prima linea e a `-O` e `>` nella seconda genera anche la costa a massima risoluzione (`-Df`) basata sullo standard GSSHS in un nuovo file che poi viene convertito in formato pdf grazie a `ps2pdf` del pacchetto Ghostscript.

```
gmt psbasemap -R-71/-61/-25/-18 -B1g1/1g1 -Jm1:10000000 -V -K > area.ps
gmt pscoast -Df -G100/200/1 -R -Jm -O >> area.ps
ps2pdf area.ps
```

nel terzo comando notiamo anche la sostituzione dei caratteri » al carattere > a significare che il file area.ps é generato aggiungendo via via che i comandi vengono eseguiti blocchi di informzioni codificate in Postscript Language che poi verranno convertiti in Portable Document Format.

Creato il frame geografico con i prossimi comandi possiamo associare ad esso le informazioni relativi a tutti gli eventi sismici presenti nel catalogo

```
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $3,$2,$4 }'  > chile.txt
gmt psbasemap   -R-71/-61/-26/-18 -B1g1/1g1 -Jm1:10000000 -V -K > chile.ps
gmt pscoast  -Df -G100/100/100 -R -Jm -O -K -V >> chile.ps
gmt psxy  chile.txt -R -Jm -Sc0.1 -G255/0/0 -O -V >> chile.ps
ps2pdf chile.ps
```

é necessario notare che nel primo comando che estrae le informazioni dal database, nella sezione `awk` latitudine e longitudine sono scambiate in quanto la convenzione all'interno di GMT é di avere la longitudine prima della latitudine in mancanza dell'inserimento della flag `-:`.

Il comando che segue é simile al precedente e genera una immagine globale del pianeta centrato sulla zona coperto dal catalogo su cui abbiamo lavorato sino ad ora. Come si puó immaginare nel momento si debba generare una immagine simile ma con caratteristiche differenti bisogna intervenire ad-hoc.

```
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '{print $3,$2,$4 }'  > chile.txt
gmt psbasemap -Rg -JG-76/-26/6i -Bafg -B+tOrthographic -P -K > ortho.ps
gmt pscoast  -Rg -JG -Di -G100/100/100 -O -V -K >> ortho.ps
gmt psxy  chile.txt -Rg -JG -Sc0.01 -G255/0/0 -O -V >> ortho.ps
ps2pdf ortho.ps
```

Per rendere il comando piú flessibile si puó generare uno script personalizzabile. Ma prima di passare alla scrittura di uno script introduciamo il concetto di variabile di sistema. Tutti i comandi che abbiamo scritto ed eseguito sino ad ora richiamavano il file `Magnitude_5.0_1960-2018.txt` che andava scritto ogni volta essendo il file principale sorgente dei dati originali. E' possibile sostituirlo nei singoli comandi assegnando ad una variabile il suo valore come nella prima riga della sequenza di comandi che seguono e richiamandolo come nella seconda riga:

```
filein="Magnitude_5.0_1960-2018.txt"
sed ' 1 d' ${filein} | awk '{print $3,$2,$4 }'  > chile.txt
gmt psbasemap -Rg -JG-76/-26/6i -Bafg -B+tOrthographic -P -K > ortho.ps
gmt pscoast  -Rg -JG -Di -G100/100/100 -O -V -K >> ortho.ps
gmt psxy  chile.txt -Rg -JG -Sc0.01 -G255/0/0 -O -V >> ortho.ps
ps2pdf ortho.ps
```

La variabile `filein` una volta che é definita all'interno della console é sempre disponibile per altri comandi quindi puó essere dichiarata una sola volta e solo se il suo contenuto varia deve essere nuovamente definita. Se vogliamo che la variabile sia eseguita da ogni console aperta ovvero disponibile dal computer a qualsiasi livello diventando una variabile globale (scritta generalmente con lettere maiuscole, ma é una convenzione non una necessitá) é necessario dichiararla in maniera diversa attraverso la keyword di shell export:

```
export FILEIN="Magnitude_5.0_1960-2018.txt"
```

facendo attenzione a non sovrascrivere una variabile magari giá definita con lo stesso nome per cui si puó procedere con la sequenza di comandi:

```
echo ${FILEIN}
export FILEIN="Magnitude_5.0_1960-2018.txt"
```

per cui se il primo comando restituisce una variabile vuota, posiamo procedere senza problemi alla definizione della variabile globale. Gli script sono stati scritti immaginando che la versione di sviluppo di GMT sia la 5, ma l'ultima versione reperibile sul sito e in vari OS é la 6 (v.6.1.1). un'utente potrebbe avere entrambe le versioni sul proprio computer e potrebbe avere l'esigenza di comparare i risultati dello stesso pacchetto sviluppati nelle due versioni e quindi dovrebbe modificare tutti gli script sostituendo `gmt6`, `gmt5` oppure `gmt`. Ad esempio si pu'ødefinire in console la versione di `gmt` secondo quanto desiderato dal sistema e poi usare il suo identificativo di variabile come in:

```
export GMT=gmt6
${GMT} psbasemap -R-71/-61/-25/-18 -B1g1/1g1 \
-Jm1:10000000 -V > frame.ps
```

## 0.7 Creare uno script

Scrivere uno script significa introdurre in un file di testo i comandi che abbiamo eseguito in console aggiungendo qualche specfica per far si che da console possa essere eseguito. Generalmente non riesco a scrivere uno script di getto ma per approssimazioni successive controllando sempre ogni singola aggiunta in termini di qualitá del risultato intermedio. Lo script che generiamo si chiama `make_ortho.sh` ed é equivalente in termini di risultati al comando appena visto ma puó essere personalizzato attraverso l'uso di alcuni argomenti specifici.

```
#/bin/bash

# Usage: . ./make_ortho.sh input map lat long fileout
#      . ./make_ortho.sh Magnitude_5.0_1960-2018.txt ortho -76 -26 chile.txt
input=$1
map=$2
lat=$3
long=$4
fileout=$5

sed ' 1 d' ${input} | awk '{print $3,$2,$4 }'  > ${fileout}
${GMT} psbasemap -Rg -JG${at}/${long}/6i -Bafg -B+tOrthographic -P -K > ${map}.ps
${GMT} pscoast  -Rg -JG -Di -G100/100/100 -O -V -K >> ${map}.ps
${GMT} psxy  ${fileout} -Rg -JG -Sc0.01 -G255/0/0 -O -V >> ${map}.ps
ps2pdf ${map}.ps
```

Con esso possiamo generare una proiezione a scala globale (`-JG`) delle coste a scala intermediate (`-Di`) con il plot dei punti presenti nel catalogo rendendo disponibile la tripletta di valori `${fileout}` per processing successivi.

## 0.8 Creare uno script complesso

Abbiamo introdotto una quantitá sufficiente e necessaria per poter creare uno script piuttosto articolato che converta la conoscenza riassunta in un singolo catalogo sismico in una serie di prodotti grafici/cartografici che approfondisca la nostra conoscenza delle zone di subduzione. Il file `seismo.sh` é a vostra disposizione e qui viene analizzato per una migliore comprensione.

La prima riga dichiara che quanto é riportato nel file é uno script di shell bash. Il carattere # in questo e solo in questo caso fa parte della dichiarazione iniziale. Nelle altre righe dello script # invece indica l'inizio di una riga di commento, ció che lo segue non é un comando e non viene eseguito.

```
#/bin/bash

# Usage: source seismo.sh Magnitude_5.0_1960-2018.txt  chile.txt  trench
#             source seismo.sh     Seismological_db       lat_long_mag   map
```

nello specifico le due righe di commento suggeriscono all'utente come il comando `seismo.sh` deve essere eseguito, preceduto da source che é un comando della shell e seguito da tre argomenti, un file di input e due file di output:

- `seismological.db`: [input] `Magnitude_5.0_1960-2018.txt` dove sono riportati i dati originali del catalogo;
- `lat_long_mag`: [output] file in cui vengono salvati latitudine, longitudine e magnitudo
- `map`: [output] nome del file grafico in formato pdf

Nel blocco che segue vengono dichiarate le variabili che verranno utilizzate in seguito e che permettono all'utente di utilizzare il medesimo script in diverse situazioni, le prime tre vengono direttamente acquisite come parametri posizionali e argomenti del comando, mentre le altre tre sono dichiarate in maniera statica all'interno dello script:

```
file=$1
file2=$2
ps=$3
region="-71/-61/-26/-18"
```

```
proj_t="R" #Winkel Tripel -- questo è un commento
proj_s="-66/10"
```

Questi due comandi non sono nulla di nuovo in quanto li abbiamo giá visti in cui il primo scrive sulla console i valori minimo e massimo delle tre colonne relative a latitudine, longitudine e magnitudo, mentre il secondo estrae i tre valori relativi ai 1333 record del catalogo:

```
# Determino min/max delle colonne
sed ' 1 d' ${file} | awk '{print $3, $2, $4 }' | ${GMT} gmtinfo
# Creo file di dati per plot
sed ' 1 d' ${file} | awk ' {print $3, $2, $4 }'  > ${file2}.txt
```

Ora si genera il primo file pdf in cui gli epicentri dei sismi sono plottati sulla base cartografica scelta:

```
# Genero cartografia
${GMT} psbasemap -R${region} -B10g10/10g10 -J${proj_t}${proj_s} -V -K > ${ps}.ps
${GMT} pscoast -Df -G100/100/100 -R -J${proj_t} -O -K -V >> ${ps}.ps
${GMT} psxy ${file2}.txt -R -J${proj_t} -Sc0.1 -G0/255/0 -O -V >> ${ps}.ps

# Converto il file postscript in formato PDF
ps2pdf ${ps}.ps ${ps}.pdf
```

Se sul piano planimetrico abbiamo acquisito una certa abilitá ora dobbiamo cercare di acquisire qualche informazione sulla distribuzione dei sismi in profonditá. Non abbiamo ancora un file in cui venga riportata l'informazione della profonditá ipocentrale per cui dobbiamo crearlo, ma possiamo anche avere interesse a distinguere i sismi in funzione della loro magnitudo per cui il blocco che segue ci permette separare i sismi nell'area in due subset con M$<5.5$ e M$\geq 5.5$. Il tipo di diagramma bivariato che verrá generato avrá sull'asse delle ascisse la longitudine e sull'asse delle ordinate la profonditá in km e rappresenta la proiezione di tutti gli ipocentri su un unico piano verticale. I due subset verranno plottati con simboli differenti.

```
# Piano di Benioff
sed ' 1 d' ${file} | awk ' {print $3, -$5 }'  > ${file2}_2.txt
sed ' 1 d' ${file} | awk ' $4>=5.5 {print $3, -$5 }'  > ${file2}_3.txt
sed ' 1 d' ${file} | awk ' $4<5.5 {print $3, -$5 }'  > ${file2}_4.txt
```

Per poter determinare i limiti del diagramma che possono variare di volta in volta in base alla regione geografica ed in base al catalogo sismico vengono generate delle variabili con i quattro comandi in cui `min_l`, `max_l`, `min_p` e `max_p` rappresentano i valori minimi e massimi di longitudine e profonditá. Oltre ad essere generate le variabili utilizzabili all'interno di questo script ogni volta che sará necessario sono anche scritte con il comando echo sulla console in maniera tale che durante l'esecuzione l'utente puó rendersi conto di eventuali inconsistenze.

```
min_l=`${GMT} gmtinfo -C ${file2}_2.txt | awk '{print $1}'`
max_l=`${GMT} gmtinfo -C ${file2}_2.txt | awk '{print $2}'`

min_p=`${GMT} gmtinfo -C ${file2}_2.txt | awk '{print $4}'`
max_p=`${GMT} gmtinfo -C ${file2}_2.txt | awk '{print $3}'`

echo
echo "Valore minimo della longitudine  : "${min_l}
echo "Valore massimo della longitudine  : "${max_l}
```

```
echo "Valore minimo della profondita  : "${min_p}
echo "Valore massimo della profondita  : "${max_p}
```

Abbiamo tutto il necessario per generare il diagramma in cui i sismi con M$<5.5$ sono plottati con punti verdi e sismi con M$\geq 5.5$ con punti rossi:

```
${GMT} psbasemap -R${min_l}/${max_l}/${max_p}/${min_p} -B5/50 -JX10 -V -K > benioff.ps
${GMT} psxy ${file2}_4.txt -R -Sc0.1 -G0/255/0 -JX -V -O -K >> benioff.ps
${GMT} psxy ${file2}_3.txt -R -Sc0.1 -G255/0/0 -JX -V -O >> benioff.ps
ps2pdf benioff.ps
```

Sino ad ora abbiamo elaborato il file senza particolari conversioni, estrazione e impiego in comandi specifici. Per consolidare le ablitá di movimento nella console, proviamo a convertire le coordinate lat-long in un sistema proiettato che per il Cile equivale secondo spatialreference.org ai codici **EPSG**:31979 e **EPSG**:31980. Nel blocco di comandi oltre a definire i valori min e max di longitudine con l'ausilio delle librerie proj e secondo la stringa relativa al codice **EPSG** relativo creiamo due files in cui gli le coordinate ipocentrali in lat long sono sostituite da coordinate metriche.

```
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '($3<-66 && $3>=-72) {print $3, $2, $4, $5 }' | \
proj +proj=utm +zone=19 +south +ellps=GRS80 \
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs > Mag_SIRGAS_19.dat
sed ' 1 d' Magnitude_5.0_1960-2018.txt | awk '($3<-60 && $3>=-66) {print $3, $2, $4, $5 }' |
proj +proj=utm +zone=20 +south +ellps=GRS80
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs > Mag_SIRGAS_20.dat
```

Lo script si conclude (anche se é ancora in work in progress) con due cartografie separate, una per ogni zona, con il frame di riferimento consistente.

```
${GMT} gmtinfo -C Mag_SIRGAS_20.dat
#19J 208422.4 808228.61 7140005.39 7921494.76 5 8.2 3.32 273.9
#20J: 192663.71 657190.99 7146226.32 7890736.11 5 7.2 1 601.2
```

Il catalogo sismico lo abbiamo elaborato per ottenere una serie di prodotti cartografici che possono essere di utilizzo in rapporti o per avere un visione sintetica della loro disposizione spaziale statica.

Quello che affrontiamo adesso é il passaggio successivo ovvero creare un db spaziale basato su GRASS utilizzando il medesimo catalogo sismico, includendo dati indipendenti e derivando un database in coordinate proiettate a partire da un database basato su un sistema id riferimento in coordinate geografiche.

**Consolidiamo quanto appreso** Aggiungere al plot :

- Apparati vulcanici
- Profonditá della discontinuitá di Mohorovichich

## 0.9 Creare un db in GRASS

**Work in progress** Al momento fa riferimento la registrazione video o le innumerevoli pagine dedicate a GRASS

## 0.10 Popolare un database

Per produrre la cartografia basata su GMT é stato sufficiente processare il catalogo testuale con gli strumenti base senza fare troppo sforzo. Adesso dobbiamo pre-processare il catalogo in modo che possa essere inserito dentro un database spaziale relazionale, in cui i dati saranno effettivamente georiferiti e potranno essere messi in relazione all'interno di un database.

Lo script `script_01.sh` deve essere eseguito nella console che dipende da GRASS, ovvero quella in cui abbiamo attivato GRASS via comando oppure che é stata aperta nel caso abbiamo attivato GRASS da interfaccia grafica.

Le prime righe hanno il solito significato informativo e di dichiarazione di variabili, che sono in pratica gli argomenti del comando.

```
#!/bin/bash

# USAGE: source script_01.sh Magnitude_5.0_1960-2018.txt temp Mag2grass.dat

file_in=$1
file_out=$2
file_def=$3
```

Il comando successivo genera un file in cui la stringa del tempo T in cui é avvenuto il sisma viene separato nelle sue unitá di tempo anno, mese, giorno, ore, minuti, secondi. Questo file sará eliminato al termine dell'esecuzione dello script.

```
awk 'FS="-" {print $1, $2,$3}' ${file_in} | \
awk 'FS=":" {print $1, $2, $3, $4, $5}' | sed -e 's/T/ T /g' -e 's/Z/ Z/g' | \
awk 'OFS="\t" {print $1, $2, $3, $4, $5, $6, $7, $8}' >  ${file_out}
```

Successivamente il file precedente composto da 8 colonne viene appaiato al file originale, eliminando la prima colonna, e aggiornando le intestazioni delle singole colonne sostituendone il nome.

```
paste ${file_out} ${file_in} | \
awk ' FS="\t", OFS="|" {print $1, $2, $3, $4, $5, $6, $7, $8, $10, $11, \
$12, $13, $14, $15, $16}' | \
sed 's/T ime Latitude Longitude Time/ Yr mo day T hr mn secs Z/' | \
sed 's/|$//g' > ${file_def}
```

Malgrado i dati siano inseriti correttamente, la visualizzazione in 3D in GRASS non avviene correttamente e ció édovuto ad una mancanza di consistenza delle unitá di misura in cui sono espresse le profonditá ipocentrali. Per cui é sufficiente moltiplicare per 1000 il valore delle profonditá e cambiandolo di segno, ottenendo il comando:

```
paste ${file_out} ${file_in} | \
awk ' FS="\t", OFS="|" {print $1, $2, $3, $4, $5, $6, $7, $8, $10, $11, \
$12, -$13*1000, $14, $15, $16}' | \
sed 's/T ime Latitude Longitude Time/ Yr mo day T hr mn secs Z/' | \
sed 's/|$//g' > ${file_def}
```

Quello che vediamo di seguito é un comando specifico di GRASS (`v.in.ascii`) che viene eseguito via console con gli argomenti che permettono di scrivere i punti sotto forma di vettore e creando una tabella specificando la tipologia di informazione.

```
v.in.ascii --overwrite -z in=${file_def} out=earthquakes \
sep="|" x=10 y=9 z=12 skip=1 col='yr integer, mo integer, day integer, \
T varchar(1), hr integer, min integer, secs double precision, UTC varchar(1), \
y double precision, x double precision, mag double precision, \
depth double precision, location varchar(50), info varchar(70)'

rm ${fileout}
```

**Giocare con GRASS per visualizzazione** E' possibile mostrare alcune informazione grazie al db generato con `v.in.ascii`

Una volta scritto ed analizzato per eventuali problemi sintattici si esegue ad esempio come:

```
source script_1.sh Magnitude_5.0_1960-2018.txt temp Mag2grass.dat
```

dove il primo argomento é il catalogo sismico, il secondo é un file temporaneo che verrá cancellato (lo studente potrebbe invece cercare di evitare questo per vedere le informazioni all'interno), ed il terzo é il file ASCII a seguito del pre-processamento prima di essere caricato dentro GRASS.

Ora abbiamo bisogno di caricare nel nuovo database la linea di costa che troviamo nella cartella `gshhg-shp-2` con il comando (avendo cura di sostituire il path completo con quello adeguato alla vostra postazione):

```
v.in.ogr input=/home/docente/Documents/ECGIS_2020/gshhg-shp-2/GSHHS_shp/c layer=GSHHS_c_L1
```

`v.in.ogr` é un comando di GRASS che gestisce i vettori mediante la libreria ogr in grado di esportare/importare un gran numero di formati georiferiti. In questo caso importa un vettore nel formato ESRI shape relativo ad una porzione della costa mondiale ad un grado di risoluzione grossolana (`c = coarse`). Con il comando precedente effettuiamo solo un'importazione parziale relativo ad una sezione denominata L1, mentre nella cartella competente sono presenti 6 sezioni. Abbiamo due possibilitá per importare tutto o ripetiamo manualmente il comando precedente 6 volte oppure facciamo eseguire automaticamente l'operazione con la sintassi seguente:

```
for i in 1 2 3 4 5 6
do v.in.ogr input=/home/docente/Documents/ECGIS_2020/\
gshhg-shp-2/GSHHS_shp/c layer=GSHHS_c_L${i}
done
```

Al termine dell'esecuzione il file sará popolato da 6 files distinti GSHHS_i_L$i con i che varia da 1 a 6, riportando vettori relativi ad acque interne, laghi ecc ecc. Se volessimo importare non solo le coste mondiali a descrizione grossolana (`c`) ma tutto il set comprendente tutte le risoluzioni potremmo modificare il comando precedente nel seguente modo:

```
for j in c l i h f
# resolution: coarse / low / intermediate / high / full
do
   for i in 1 2 3 4 5 6
do v.in.ogr input=/home/docente/Documents/\
ECGIS_2020/gshhg-shp-2/GSHHS_shp/${j} layer=GSHHS_${j}_L${i}
   done
done
```

Analogamente popoliamo il database con uno dei modelli numerici del terreno globali **ETOPO1** grid-centered scon il comando seguente prima decomprimendo il file compresso e poi con `r.in.gdal` [https://ngdc.noaa.gov/mgg/global/global.html] (sempre modificando il path completo che punta al file `ETOPO1_Bed_g_gdal.grd`:

```
gunzip /home/docente/Documents/ECGIS_2020/ETOPO1_Bed_g_gdal.grd.gz
r.in.gdal -e -l -o input=/home/docente/Documents/ECGIS_2020/ETOPO1_Bed_g_gdal.grd output=ETOPO1
```

`r.in.gdal` é il comando di GRASS analogo a `v.in.ogr` per importare ed esportare i files digitali in formato raster. Come nota possiamo sottolineare che la denominazione dei comandi di GRASS come si puó intuire ha una struttura semplice e mnemonica in cui i comandi che cominciano con v.* sono dedicati all'elaborazione dei files vettoriali, `r.*` per i files raster, `i.*` per le immagini, `g.*` sono comandi generali e cosí via. `r.to.vect` estrae solo il tratto di costa che rientra nell'area appena isolata sulla base della estensione dei punti relativi agli eventi sismici presenti nel catalogo.

A questo punto dobbiamo focalizzare la nostra attenzione sull'area della Fossa di Atacama per cui definiamo esplicitamente i limiti E, W, S, e N con il comando che segue specificando anche la risoluzione spaziale che per il modello **ETOPO1** é di 1' di ampiezza. La definizione dei limiti geografici al contrario di quanto avevamo visto all'interno del catalogo sismico usa la convenzione per cui la posizione rispetto al meridiano centrale di Greenwich viene esplicitato con i punti cardinali.

```
g.region n=18:00:00S s=26:00:00S w=72:00:00W e=61:00:00W res=0:00:01
```

**ETOPO1** é un modello altimetrico descrittivo di tutto il globo (circa 1Gb di spazio) mentre noi siamo interessati alla zona limitata dal comando precedente, per cui per limitare effettivamente l'area su cui eseguiamo le prossime operazioni digitiamo i due comandi:

```
r.mapcalc expression="chile=1"
r.to.vect in=chile out=chile type=area
r.mask rast=chile
```

con cui rendiamo attiva l'area a cui abbiamo dato valore 1 e `null()` tutto il resto. Il primo comando `r.mapcalc` permette un gran numero di operazioni matematiche sulle immagini raster mentre il secondo converte il file raster in vettore. Infine il terzo opera su di esso rendendo tutto ció che é al di fuori come inattivo. Ora possiamo riutilizzando uno dei comandi digitati precedentemente estrarre in un file pi' facilmente gestibile [`ETOPO1_red`, per reduced] un modello numerico del terreno con la delimitazione esatta sulla base delle opzioni del comando `g.region`.

```
r.mapcalc expression="ETOPO1_red=ETOPO1"
```

```
v.overlay ainput=GSHHS_i_L1@PERMANENT binput=chile@PERMANENT operator=and output=coast_chile
```

Possiamo replicare per importare altri due dataset sempre acquisiti tramite Geomapapp per quanto riguarda due dataset differenti:

- Depth to Moho (relative to sea-level): profonditá della discontinuitá di Mohorovichich.
- Smithsonian Global Volcanism Programs: elenco degli apparati vulcanici con alcune specifiche vulcanologiche;

Per importare il primo file l'impegno non é particolarmente elevato essendo composto semplicemente da tre colonne numeriche con le solite convenzioni per longitudine e latitudine e un reale per quanto riguarda la profonditá espressa in km.

```
moho=Depth_to_Moho_(relative_to_sea-level).txt
sed '1 d' ${moho} | awk 'OFS="|" {print $1, $2, $3*1000} > moho.dat
v.in.ascii --overwrite in=moho.dat out=moho \
sep="|" x=1 y=2 z=3 col='long double precision, lat double precision, z double precision'
```

Per il momento lo lasciamo li e lo riprendiamo tra poco.

Il file relativo al database degli apparati vulcanici é piú complesso e deve essere elaborato in maniera simile al catalogo sismico. Le lunghezze dei vari campi testuali sono variabili e le lunghezze precise vengono suggerite dal comando in quanto non procede a caricare campi che siano della lunghezza inferiore a quanto suggerito dall'utente:

```
volcanoes=Smithsonian_Global_Volcanism_Program_List_v4.3.4
sed 's/\t/;/g' ${volcanoes} > volcanoes.dat
v.in.ascii --overwrite -z in=volcanoes.dat out=volcanoes skip=1 sep=";" x=11 y=10 z=12 \
col='V_number integer, URL varchar(50), V_name varchar(24), Country varchar(20), \
V_type varchar(20), Activity varchar(20), Last_eruption varchar(20), Region varchar(20), \
, Subregion varchar(37), Latitude double precision, Longitude double precision, \
Elevation integer, Rock_type varchar(41), T_setting varchar(44), dum varchar(1)'

moho=Depth_to_Moho_\(relative_to_sea-level\).txt
sed 's/\t/;/g' ${moho} | awk -F';' '{print $1";"$2";"$3*1000";"}' > moho.dat
v.in.ascii --overwrite -z in=moho.dat out=moho sep=";" x=1 y=2 z=3 \
skip=1 col='longitude double precision, latitude double precision, \
depth double precision, dummy varchar(1)'
```

I punti relativi alla profonditá della Moho sono derivati da modelli geofisici a scala mondiale, uno dei molti disponibili, e possono essere utlizzati per generare una superficie che materializzi la effettiva morfologia della discontinuitá.

```
v.surf.idw --overwrite --verbose input=moho@PERMANENT column=depth output=moho

v.surf.rst --overwrite --verbose input=moho@PERMANENT zcolumn=depth elevation=moho_rst

r.contour --verbose --overwrite input=moho_rst@PERMANENT output=moho_rst \
step=5000.0 minlevel=-75000.0 maxlevel=0.0
```

Un database inquadrato in un sistema globale **EPSG**:4326 é stato popolato da alcune informazioni derivate da diversi database indipendenti ora é il momento di creare un database nuovo di GRASS basato sul sistema **EPSG**:31979, seguendo le medesime procedure seguite precedentemente.

Prima di passare al nuovo database rendiamo visibili i meridiani interessati del sistema UTM che impiegheremo successivamente introducendo anche una delle modalitá di inserimento di dati vettoriali basati sulla conoscenza delle semplici coordinate e della topologia che in questo caso é costituita semplicemente da linee limitate ad una latitudine N-S di 90°.

Il file é composto da una sezione header in cui sono riportate alcune info descrittive ed una sezione dati in cui 4 linee sono identificate da tre linee specifiche.

```
echo ORGANIZATION: LGCH - UNIGE
DIGIT DATE:   26/10/2020
DIGIT NAME:   -
MAP NAME:     ECGIS 2020
MAP DATE:     2020
MAP SCALE:
OTHER INFO:   Test v.in.ascii
ZONE:  0
MAP THRESH:   0.500000
VERTI:"
L  2
 78:00:00W 90:00:00N
 78:00:00W 90:00:00S
L  2
 72:00:00W 90:00:00N
```

```
 72:00:00W 90:00:00S
L  2
 66:00:00W 90:00:00N
 66:00:00W 90:00:00S
L  2
 60:00:00W 90:00:00N
 60:00:00W 90:00:00S
B 5
 72:00:00W 90:00:00N
 66:00:00W 90:00:00N
 66:00:00W 90:00:00S
 72:00:00W 90:00:00S
 72:00:00W 90:00:00N
C 1 1
 69:00:00W 00:00:00S
1 19
```

L'ultima parte della sezione dati riporta la definizione di un area identificata dalla lettera `B` che materializza il fuso identificato dal centroide `C` con categoria 19. E' una maniera sintetica di creare un vettore custom da parte dell'utente senza digitalizzare mantenendo l'esatta valorizzazione dei nodi determinati magari per altra via. Con il comando che segue possiamo importare i dati stocati nel file appena creato:

```
v.in.ascii --verbose input=UTM_bounds.dat output=UTM_bounds format=standard
```

## 0.11 Trasformare il contenuto di un database in sistema di coordinate differente

Una volta creato il database il primo step é importare i dati presenti nel database WGS84 in questo nuovo database SIRGAS19. In questo primo blocco di comandi eseguiti all'interno della console e di SIRGAS19 importiamo alcuni files vettoriali e la raster limitata all'area all'interno della quale ricadono i sismi del catalogo definendo precedentemente la regione specifica in coordinate metriche che sono competenti alla zona 19JS:

```
g.region -p
g.region n=7930000 s=7130000 e=810000 w=200000 res=100
g.region -p

# Importare vettori
v.proj location=WGS84 mapset=PERMANENT input=earthquakes output=earthquakes
v.proj location=WGS84 mapset=PERMANENT input=coast_chile output=coast_chile
v.proj location=WGS84 mapset=PERMANENT input=chile output=chile
v.proj location=WGS84 mapset=PERMANENT input=volcanoes output=volcanoes
v.proj location=WGS84 mapset=PERMANENT input=moho_rst output=moho_rst
```

La serie di comandi appena eseguiti puó essere risolta elegantemente con un ciclo `for` del tipo:

```
for map in earthquakes coast_chile chile volcanoes moho_rst
do
v.proj --overwrite location=WGS84 mapset=PERMANENT input=${map} output=${map}
done
```

```
# Importare raster
r.proj location=WGS84 mapset=PERMANENT input=ETOPO1_red output=ETOPO1_red resolution=100
r.proj location=WGS84 mapset=PERMANENT input=moho output=moho resolution=100
r.proj location=WGS84 mapset=PERMANENT input=moho_rst output=moho_rst resolution=100
```

`v.proj` e `r.proj` sono comandi di GRASS che permettono di convertire e/o trasformare oggetti georiferiti tra sistemi di riferimento differenti le cui specifiche sono all'interno dei singoli database precedentemente creati. In `r.proj` specifichiamo anche la risoluzione a 100m. Dall'analisi della tabella di database che riporta le medesime informazioni che abbiamo inserito nel database WGS84 si può notare che non sono presenti le informazioni che caratterizzano il nuovo database ovvero le coordinate nel sistema specifico del SIRGAS19 per cui per includerle é sufficiente digitare:

```
v.to.db --verbose map=earthquakes opetion=coor col='E,N'
```

in cui il comando trasferisce dati che sono parte del file vettoriale nella tabella del DB evidenziando il fatto che non tutte le informazioni del vettore sono presenti nel DB relazionale. Esso va popolato in funzione delle necessitá del progetto.

I comandi che seguono ci permettono di vedere un riassunto dei metadata relativi alla raster del modello numerico appena importato e poi di definire una palette di colori, una tra le molte disponibili.

```
r.info map=ETOPO1_red@PERMANENT
r.colors -r map=ETOPO1_red@PERMANENT color=grass
v.in.region out=sirgas_19
r.mask vect=sirgas_19
```

Gli ultimi due comandi permettono di generare una maschera a partire da un vettore che ha la medesima estensione della regione (denominata ora `sirgas_19`) appena specificata poche linee sopra.

**Giocare con la visualizzazione 3D del DEM e degli ipocentri** Provare a visualizzare l'andamento del Piano di Benioff in relazione alla Fossa di Atacama e il bacino back-arc.

`r.mapcalc` lo abbiamo visto precedentemente e ora lo usiamo per segmentare il DEM in base a criteri altimetrici, ovvero sezionando l'altimetria suddividendo tre zone: `low`, `middle` e `high`. Abbiamo bisogno di un criterio basato su una condizionalitá che viene espressa tramite il ciclo `if-then-else` implementato tipicamente nei linguaggi di programmazione in quanto rispecchia un tipico approccio cognitivo di enorme portata:

```
r.mapcalc expression='low=if(ETOPO1_red < 0 , 1 , null())'
r.mapcalc expression='middle=if(ETOPO1_red > 0 && ETOPO1_red < 2500, 2 , null())'
r.mapcalc expression='high=if(ETOPO1_red > 2500, 3 , null())'
r.patch in=low,middle,high out=zone
r.colors map=zone col=rainbow
d.mon start=wx0
```

Le prime tre righe segmentano il DEM in tre zone cui vengono assegnate 3 valori categorici (da 1 a 3) assegnando il valore `null()` altrove. I tre criteri determinano tre raster con copertura areale mutualmente esclusiva per cui la sovrapposizione determina tramite il comando `r.patch` una raster unica che con `r.colors` viene colorata assegnando un colore diverso per ogni categoria. La raster ottenuta non é formalmente corretta in quanto i valori $= 0$ non sono contemplati per cui si ó modificare sostituendo il simbolo di diseguglianza $\leq$:

```
r.mapcalc --overwrite expression='low=if(ETOPO1_red <= 0 , 1 , null())'
r.mapcalc --overwrite expression='high=if(ETOPO1_red >= 2500, 3 , null())'
```

Combinando le linee scritte in un unico script (`script_02.sh`) possiamo generare diverse segmentazioni senza modificare alcuna linea di script aggiungendo due argomenti numerici corrispondenti a valori limite delle tre aree:

```
#!/bin/bash

# source script_02.sh 0 2750

d.erase --quiet

lowh=$1
middleh=$2

d.mon -x select=wx0
g.remove -f type=rast name=low,middle,high,zone
r.mapcalc --overwrite expression="low=if(ETOPO1_red <= ${lowh} , 1 , null())"
d.rast map=low --quiet
r.mapcalc --overwrite expression="middle=if(ETOPO1_red > \
${lowh} && ETOPO1_red < ${middleh}, 2 , null())"
d.rast map=middle --quiet
r.mapcalc --overwrite expression="high=if(ETOPO1_red >= ${middleh}, 3 , null())"
d.rast map=high --quiet
r.patch --overwrite in=low,middle,high out=zone
r.colors map=zone col=rainbow
d.rast map=zone
```

Se volessimo poi avere in rapida successione diversi scenari su una finestra interrogabile da console (wx0 creata precedentemente) allora possiamo inglobare lo `script_02.sh` nello `script_03.sh`.

```
#!/bin/bash

for i in 1000 2000 3000 4000
do
source script_02.sh 0 $i
sleep 3
done
```

## 0.12 Importare una batimetria: lago Trasimeno

Talvolta puó succedere che le informazioni di cui necessitiamo siamo difficilmente acquisibili oppure se reperibili, con un contenuto informativo parziale. A tal proposito riportiamo la situazione per cui nella pianificazione di uno studio geochimico necessitiamo della linea di costa e della batimetria del Lago Trasimeno. Il lago é impostato su una depressione tettonica sede di un importante prisma sedimentario derivante dalla detrizione della catena montuosa e attualmente ha una batimetria molto ridotta con una profonditá massima di circa 5 m. Dopo qualche ricerca si é giunti ad un file in formato ESRI shape sul sito della regione Umbria OpenDataUmbria. Il materiale é reperibile in due sistemi di riferimento distinti, il primo in coordinate latitudine-longitudine basato sullo sferoide geocentrico WGS84 **EPSG**:4326 e l'altro secondo nel sistema di coordinate UTM secondo i parametri del codice **EPSG**:32633 - ERTF2000/UTM33N.

Il file é disponibile nella sezione files del corso e una volta scaricato sul proprio pc é sufficiente copiarlo nella directory di lavoro dove sono preenti gli altri files che stiamo utilizzando per il corso. I dati sono all'interno din una

cartella che si chiama UTM33 e sono in formato ESRI .shp e i file sono nominati in maiera da riconoscere quelli riferiti alla batimetria con i valori delle isoipse relative alla quota assoluta o relativa allo zero idrometrico che é posto a 257.33 m sul livello del mare (SLM) come riportato sul sito di riferimento.

Dopo aver creato un database sfruttando il codice **EPSG** é possibile importare il vettore e definire subito il limite di calcolo:

```
v.in.ogr --verbose input=UTM_33 layer=TRASI_CONTOUR025_U33_WGS84_SLM,\
TRASI_CONTOUR025_U33_WGS84_ABS output=d_trasimeno
g.region vect=d_trasimeno res=10
```

Le convenzioni per la denominazione dei files (raster e vector) é la seguente [L_trasimeno_T]:

- L: tipo di file:
  - d: dati originali
  - h: convex hull
  - p: punti
  - c: costa
  - i: isole
  - l: lago nella configurazione finale
  - b,s: batimetria, slope (pendenza)
- T: tipo di isobata
  - SLM: altimetria assoluta sul livello del mare (0 m slm)
  - ABS: profonditá rispetto allo zero idrometrico (257.33 m slm)

Possiamo osservare che i files contengono le isobate ma non ci sono informazioni relativamente alla linea di costa del lago o delle 3 isole presenti (Minore, Maggiore, Polvese). Il file creato possiede al suo interno entrambe le valorizzazioni ed entrambe hanno equidistanza di .25 m, spazialmente non coincidono, per il diverso valore della quota di riferimento ma dopo adeguato processing possono essere utilizzate entrambe per determinare un modello numerico piú dettagliato. Per ottenere il modello numerico della batimetria e l'andamento della linea di costa, il problema va quindi suddiviso in diversi steps:

- convertire le isobate in punti di coordinate [x,y,z]
- determinare una modalitá per ottenere la linea di costa che sará approssimata e che in realtá ricopre il significato di area massima di accesso del natante;
- intervenire manualmente sui set dei dati puntuali;
- calcolare un modello numerico della batimetria, per poter essere utilizzato successivamente.

Per prima cosa convertiamo il vettore costituito da linee ed aree in un vettore di punti interpolando ogni 50 m per entrambe le distribuzioni, SLM (sopra il livello del mare, valori positivi) e ABS (assoluta rispetto allo zero idrometrico, valori negativi):

```
#convert vector lines to vector points
v.to.points -i --verbose --overwrite input=d_trasimeno \
layer=1 output=p_trasimeno_SLM use=vertex dmax=50
v.to.points -i --verbose --overwrite input=d_trasimeno \
layer=2 output=p_trasimeno_ABS use=vertex dmax=50
```

Non essendo disponibile la linea di costa viene utilizzato un codice specifico (`concave.hull = guscio concavo`) per determinare con alcune approssimazioni il contorno minimo che contenga i punti cercando di appoggiare il contorno al maggior numero di punti esterni mimando il contorno:

**Nota** Inizio delle operazioni manuali

```
#generates reference lake coastline
v.concave.hull --overwrite --verbose input=p_trasimeno_ABS \
output=c_trasimeno_ABS threshold=0
```

Una volta che il vettore é stato generato si deve ricorrere ad una pulizia del vettore per evitare tutto ció che il codice ha generato ma ai nostri fini non è necessario per cui con `v.digitize` si ricorre al cleaning e alla definizione del centroide per l'area rappresentativa della linea di costa del lago. All'interno dello specchio lacustre abbiamo 3 isole, la Minore, la Maggiore e la Polvese che a loro volta hanno una linea di costa che viene approssimata nello stesso modo, partendo da un file copiato con i comandi che seguono:

```
g.copy vector=p_trasimeno_ABS,p_polvese_ABS
g.copy vector=p_trasimeno_ABS,p_maggiore_ABS
g.copy vector=p_trasimeno_ABS,p_minore_ABS
```

Si ripete la medesima operazione delimitazione tramite il guscio concavo, cleaning e di designazione del centroide per i vettori rappresentativi delle tre isole.

```
v.concave.hull --overwrite --verbose input=p_polvese_ABS output=c_polvese_ABS threshold=0
v.concave.hull --overwrite --verbose input=p_maggiore_ABS output=c_maggiore_ABS threshold=0
v.concave.hull --overwrite --verbose input=p_minore_ABS output=c_minore_ABS threshold=0
```

Dopo l'elaborazione manuale per ottenere un modello vettoriale per ogni singolo oggetto dei 4 necessari per ricreare la morfologia del Lago nella sua interezza, ricorriamo ad una ricostruzione della topologia del vettore e ad aggiungere una tabella del database relazionale.

**Nota** Fine delle operazioni manuali

```
v.build.polylines --verbose input=c_trasimeno_ABS output=trasimeno_ABS
v.db.addtable map=trasimeno table=coast
v.build.polylines --verbose input=c_polvese_ABS output=polvese_ABS
v.db.addtable map=polvese table=coast
v.build.polylines --verbose input=c_minore_ABS output=minore_ABS
v.db.addtable map=minore table=coast
v.build.polylines --verbose input=c_maggiore_ABS output=maggiore_ABS
v.db.addtable map=maggiore table=coast
```

I comandi che seguono permettono di fare 2 cose distinte, la prima fa si che le 3 isole che sono poligoni chiusi interni al lago vengono uniti in un unico file ed il secondo tramite l'operatore `xor` di esclusione del comando `v.overlay` permette di intersecare le due informazioni evidenziando la distinzione netta tra lo specchio lacustre e la terraferma.

```
v.patch in=minore_ABS,maggiore_ABS,polvese_ABS out=i_trasimeno_ABS
v.overlay --verbose ainput=trasimeno_ABS \
binput=i_trasimeno_ABS operator=xor output=l_trasimeno_ABS
```

Con `v.to.rast` creiamo una raster dalla vettoriale che serve come maschera di calcolo al comando successivo `v.surf.rast`, che a partire dai valori puntuali derivati dal file delle isobate permette di ottenere la prima versione del modello numerico della batimetria del Lago Trasimeno.

**Nota** E' possibile osservare che dalla sovrapposizione tra il modello numerico e le isoipse il codice del guscio concavo fallisce nel ricostruire precisamente i contorni delle isole, per cui potrebbe essere necessario intervenire manualmente sui vettori delle isole per apportare le necessarie modifiche e aumentare la coerenza e la capacitá descrittiva della batimetria, oppure trovare un approccio algoritmico piú flessibile in grado di superare la difficoltá.

```
v.to.rast --verbose input=l_trasimeno_ABS output=l_trasimeno_ABS use=cat
v.surf.rst --overwrite --verbose input=p_trasimeno_ABS \
mask=l_trasimeno_ABS slope=s_trasimeno_ABS layer=1 zcolumn=ELEVATION elevation=b_trasimeno_ABS
```

I due layers relativi alla isobate identificate ABS e SLM non possono al momento essere utilizzate contemporaneamente in quanto sono rappresentative di isolinee con una superficie di riferimento differente, che conosciamo e queste sono informazioni necessarie e sufficienti per poter creare due dataset completi riferiti ai due sistemi di riferimento altimetrici che esportiamo in formato ASCII e quantifichiamo:

```
v.out.ascii --overwrite --verbose input=p_trasimeno_ABS output=p_ABS.dat \
columns=ELEVATION format=point precision=2
v.out.ascii --overwrite --verbose input=p_trasimeno_SLM output=p_SLM.dat
columns=ELEVATION format=point precision=2
```

- 0 mslm per SLM : 26821 punti tra 251.75 m e 257.25 m
- 257.33 mslm per ABS : 26675 punti tra 0 m e -5.50 m

Sulla base di queste due convenzioni altimetriche determiniamo i valori della variabile altimetrica nei due dataset per tutti e due i sistemi in maniera da ottenere una database puntuale completo nei due sistemi di riferimento altimetria assoluto e relativo allo zero idrometrico.

```
awk -F'|' 'FS="|", OFS="|" {print $1,$2,$3,$4-257.33}' p_SLM.dat > p_SLM_2.dat
awk -F'|' 'FS="|", OFS="|" {print $1,$2,$3,257.33+$4,$4}' p_ABS.dat > p_ABS_2.dat
cat p_SLM_2.dat p_ABS_2.dat > p_TOT.dat
```

Il dataset così creato puó essere importato con le corrette denominazioni delle colonne della tabella del database relazionale:

```
v.in.ascii --overwrite --verbose input=p_TOT.dat output=p_trasimeno_TOT \
col='E double precision, N double precision, \
typ integer, SLM double precision, ABS double precision'
```

e successivamente utilizzato per computare la batimetria definitiva basata sulle quote assolute SLM sfruttando 4 processi di calcolo parallelo (il numero dipende dai cores disponibili dalla singola macchina) in modo da aumentare l'efficienza e ridurre il tempo di computazione e i parametri di default:

```
g.region res=10.0

v.surf.rst --overwrite --verbose -t   input=p_trasimeno_TOT mask=l_trasimeno_TOT \
slope=s_trasimeno_TOT treeseg=quad_trasimeno_TOT layer=1 zcolumn=SLM  elevation=b_trasimeno_TOT \
nprocs=4 tens=35 smooth=.1 npmin=500 segmax=100 dev=dev_trasimeno_TOT

v.surf.rst --overwrite --verbose -t   input=p_trasimeno_TOT mask=l_trasimeno_TOT \
slope=s_trasimeno_TOT treeseg=quad_trasimeno_TOT layer=1 zcolumn=SLM  elevation=b_trasimeno_TOT \
nprocs=4 tens=35 smooth=10 npmin=500 segmax=100 dev=dev_trasimeno_TOT

r.colors -e --verbose map=s_trasimeno_TOT color=water
r.colors -e --verbose map=b_trasimeno_TOT color=elevation
```

I due comandi di calcolo si differenziano solo per un parametro, lo smoothing, e danno risultati decisamente differenti che mettono in evidenza l'impatto sul risultato finale, sta all'utente decidere quale é utile al suo lavoro se no altro dopo aver per esempio controllato quale é lo scarto medio delle differenze tra punti originali e superficie calcolata (anche se in realtá nel nostro caso sono dati giá processati da altri senza avere avuto la possibilitá di accedere ai dati originali).

```
v.univar -e --verbose map=dev_trasimeno_TOT col='flt1'
v.out.ascii --verbose in=dev_trasimeno_TOT out=deviations.dat col='flt1' format=point prec=3
```

e magari averne rappresentato lo spread attorno alla media con il comando successivo:

```
sed 's/|/ /g' deviations.dat | awk '{print $5}' | \
gmt pshistogram -Bxa.1f.05+l"depth deviations [m]" -Bya1000f500+l"Punti [#]" \
-BWSen+t"Lago Trasimeno"+glightblue -JX10 -R-.3/.3/0/8000 \
-Gorange -T.01 -F -W0.5p -V > dev_histo.ps ; ps2pdf dev_histo.ps
```

che mostra come gli scarti si concentrino (frequenza maggiore, la moda) attorno allo zero cioé non ci sono deviazioni e molto rapidamente gli scarti tendono a diminuire.

Puó essere di un certo aiuto alla comprensione dell'operazione di merging e shifting fatta fino ad ora dei dati riferiti a diversi "zero" di riferimento e soprattutto all'impatto dell'algoritmo, computando ulteriomente la batimetria con i soli punti SLM e poi comparare le due popolazioni delle deviazioni. In tal modo possiamo osservare se i due dataset portano con se bias dei valori alti o bassi.

```
v.surf.rst --overwrite --verbose input=p_trasimeno_SLM mask=l_trasimeno_ABS \
slope=s_trasimeno_SLM layer=1 zcolumn=ELEVATION elevation=b_trasimeno_SLM \
nprocs=4 dev=dev_trasimeno_SLM
v.univar -e --verbose map=dev_trasimeno_SLM col='flt1'
v.out.ascii --verbose in=dev_trasimeno_SLM out=deviations_SLM.dat col='flt1' format=point prec=3
sed 's/|/ /g' deviations_SLM.dat | awk '{print $5}' > dev_SLM.dat

v.surf.rst --overwrite --verbose input=p_trasimeno_ABS mask=l_trasimeno_ABS \
slope=s_trasimeno_ABS layer=1 zcolumn=ELEVATION elevation=b_trasimeno_ABS \
nprocs=4 dev=dev_trasimeno_ABS
v.univar -e --verbose map=dev_trasimeno_ABS col='flt1'
v.out.ascii --verbose in=dev_trasimeno_ABS out=deviations_ABS.dat col='flt1' format=point prec=3
sed 's/|/ /g' deviations_ABS.dat | awk '{print $5}' > dev_ABS.dat

gmt pshistogram -Bxa.05f.01+l"depth deviations [m]" -Bya1000f500+l"Punti [#]" \
-BWSen+t"Lago Trasimeno"+glightblue dev_SLM.dat -JX10 -R-.05/.05/0/8000 \
-Gorange -T.005 -F -W0.5p -V > dev_SLM.ps ; ps2pdf dev_SLM.ps

gmt pshistogram -Bxa.05f.01+l"depth deviations [m]" -Bya1000f500+l"Punti [#]" \
-BWSen+t"Lago Trasimeno"+glightblue dev_ABS.dat -JX10 -R-.05/.05/0/8000 \
-Gorange -T.005 -F -W0.5p -V > dev_ABS.ps ; ps2pdf dev_ABS.ps
```

Possiamo ora utilizzare il modello batimetrico per sviluppare un altro modello in cui si evidenzino le linee di minimo costo che corrispondono, in maniera approssimata, alle linee lungo cui ad esempio i sedimenti tendono a scorrere verso la zona piú depressa del bacino endoreico. A titolo informativo la struttura tettonica che ospita il lago Trasimeno é attiva dal Pliocene con successioni marine di pertinenza tirrenica e successivamente con spiccate caratteristiche endoreiche fluvio-lacustri ed i cui spessori raggiungono anche i 200 m

```
r.flow -3 --overwrite elev=b_trasimeno_TOT flowlen=path_trasimeno_TOT \
flowline=dens_trasimeno_TOT flowacc=acc_trasimeno_TOT skip=20
```

### 0.12.1 Intervento tecnico: posa di una condotta e dragaggio del fondo per navigazione

**Nota: TODO - Posa condotta** Estensione dell'esercitazione
Ora che abbiamo generato un modello batimetrico del Lago, proviamo ad utilizzarlo per una serie di attivitá puramente fittizie (e necessariamente approssimate per questa esercitazione) di intervento tecnico quali:

- la posa di un una serie di condotte per l'ampliamento del reticolo acquedottistico e fognario della zona:
- attivitá di dragaggio sulle rotte di navigazione dei battelli a maggior pescaggio tra la costa ed alcune isole:

Dal committente abbiamo ricevuto l'incarico per la progettazione di un piano di intervento per la posa di una condotta posata sul fondo del lago e per la definizione di un perimetro di inaccessibilitá del suo intorno. I dati a disposizione sono i punti di approdo (e stazioni di pompaggio) elencati

**Nota: TODO** Elenco dei punti
Per prima cosa possiamo evidenziare sulla mappa le stazioni di pompaggio cosí come importiamo anche il tracciato per intero:

```
Elenco stazione di pompaggio
Leg 1:
Castiglione del Lago: 260138 / 4779018
                      261934 / 4778675
                      264174 / 4777182
                      267039 / 4776274
                      267887 / 4777081
Is. Polvese           267322 / 4777727
Leg 2:
Castiglione del Lago: 260138 / 4779018
                      264012 / 4779785
Is. Maggiore          263628 / 4784306
                      264759 / 4783700
Passignano            267483 / 4785415
```

i punti vengono inglobati in un file ASCII in formato standard in cui si differenziano le tracce teoriche e il posizionamento delle stazioni di pompaggio:

```
v.in.ascii --overwrite --verbose input=acquedotto.dat output=acquedotto format=standard
v.db.addtable map=minore table=coast
```

Il tracciato teorico viene ulteriormente corredato di due aree con differenti significati:

- una fascia di inaccessibilitá permanente pari a 100 m;
- una fascia di inaccessibilitá permanente per le attivitá di posa e manutenzione pari a 500 m;

**Nota:** Uso del -t interessante
con i comandi che seguono generiamo un buffer intorno all'asse della posa dell'acquedotto e lo limitiamo con `v.overlay` alla sola area lacustre, convertendola poi in una raster:

```
v.buffer ty=line in=acquedotto out=buf_acq_100 distance=100
v.overlay --overwrite --verbose  atype=area ainput=buf_acq_100 bty=area binput=l_trasimeno \
operator=and output=buf_100
v.to.rast --verbose input=buf_100 output=buf_100 use=cat
```

ripetiamo la medesima operazione con un buffer piú ampio per delimitare le aree di manutenzione e posa:

```
v.buffer ty=line in=acquedotto out=buf_acq_400 distance=400
v.overlay --overwrite --verbose  atype=area ainput=buf_acq_400 bty=area binput=l_trasimeno\
 operator=and output=buf_400
```

Ora bisogna quantificare la precisa pertinenza dei singoli tratti di condotta per poter permettere la quantificazione del budget e delle operazioni di posa e scavo che possono riassumersi in in tratti:

- su terraferma;
- tra lo zero idrometrico (257.33 m SLM ) e ~-5 m (252.33 m slm approssimata in maniera conservativa secondo un limite batimetrico di 252.25 m slm);
- al di sotto dei ~-5 m dallo zero idrometrico ($<$ 252.25 m slm).

Infatti per i tratti di condotta in terraferma e tra lo 0 idrometrico e ~-5 m (252.33 m slm) l'infrastruttura deve essere interrata per cui si ricorre ad una serie di intersezioni tra domini vettoriali ed alla successiva riclassificazione dei singoli tratti per poi essere nuovamente ricomposti in un nuovo vettore:

```
v.extract --overwrite --verbose input=d_trasimeno@PERMANENT \
where="ELEVATION = 252.25" output=isobath_under_5
v.db.addtable map=isobath_under_5 table=area
v.overlay --overwrite --verbose ainput=acquedotto@PERMANENT \
atype=line binput=l_trasimeno@PERMANENT operator=not output=acq_land
v.overlay --overwrite --verbose ainput=acquedotto@PERMANENT \
atype=line binput=l_trasimeno@PERMANENT operator=and output=acq_lake
```

dopo aver isolato l'isobata di interesse con un'operazione `out-in` un pó ridondante ma necessaria ai fini dell'esercitazione, operiamo una pulitura delle isobate parassite, per semplificare il vettore e ció che segue:

```
v.out.ascii --verbose input=isobath_under_5 output=isobath_under_5.dat \
format=standard precision=1

v.in.ascii --overwrite --verbose input=isobath_under_5.dat \
output=isobath_under_5 format=standard

v.overlay --overwrite --verbose  atype=line ainput=acq_lake bty=area \
binput=isobath_under_5@PERMANENT operator=not output=acq_over_5

v.overlay --overwrite --verbose  atype=line ainput=acq_lake bty=area \
binput=isobath_under_5@PERMANENT operator=and output=acq_under_5
```

con gli ultimi comandi isoliamo i singoli tratti come preventivato. Il dato che ciinteressa non é ancora presente per cui con i successivi, introduciamo nella tabella di database i valori in metri dei singoli tratti con caratteristiche specifiche costruittive:

```
v.to.db --verbose map=acq_land@PERMANENT option=length columns=length units=meters
```

```
v.to.db --verbose map=acq_lake@PERMANENT option=length columns=length units=meters
v.to.db --verbose map=acq_over_5@PERMANENT option=length columns=length units=meters
v.to.db --verbose map=acq_under_5@PERMANENT option=length columns=length units=meters
```

i comandi appena eseguiti possono essere automatizzati con il ciclo `for-do-done` sottostante:

```
for i in land lake over_5 under_5
do
v.to.db --verbose map=acq_${i}@PERMANENT option=length columns=length
done
```

In definitiva sulla base della della lunghezza dei singoli tratti conoscendo il costo lineare per tipologia di operazione si puó valutare il budget necessario per la posa definitiva.

**Nota: TODO** Campionamento e valutazione dello status ambientale
Per poter avere una conoscenza preliminare di quali poterebbero essere le quanttá di elementi potenzialmente pericolosi derivanti da un dragaggio previsto come attivitá di manutenzione delle lrotte dei traghetti che attraversano lo specchio lacustre sono stati acquisiti i punti analisi da una campagna di campionamento condotta da ARPA Umbria.

Le coordinate sono inquadrate nel sistema di coordinate Gauss-Boaga **EPSG:**3004 per cui devono essere necessariamente convertite per poter essere coerenti con il sistema usato sino ad ora. La directory dove la suite di `proj` viene installata dipende talvolta da OS oppure dal fatto che il programma é stato compilato da noi e quindi installato a nostro piacimento, per cui con il comando `which` possiamo determinare la posizione della directory dove `proj` é installato e definirlo direttamente nella variabile `PROJ` (notare le virgolette, l'incubo di Linux...).

```
PROJ=‘which proj‘
cat TR_GB.dat | ${PROJ}/proj -I +proj=tmerc +ellps=intl +lon_0=15dE  +x_0=2520000 \
+k=0.9996 | ${PROJ}/proj -v +proj=utm +ellps=WGS84 +zone=33
```

```
v.in.ascii --overwrite --verbose input=TR_samples.dat output=TR_samples sep=space skip=1 \
x=1 y=2 columns="E_UTM double precision, N_UTM double precision, label varchar(4), \
E_GB integer, N_GB integer, N_tot double precision, TOC double precision, C_tot double precision, \
P_tot double precision, RS double precision, Arsenic double precision, Cd double precision, \
Cr_tot integer, CrVI double precision, Fe integer, Mn integer, Hg double precision, \
Ni double precision, Pb double precision, Cu double precision, Se double precision, Zn integer"
```

```
v.surf.rst --overwrite --verbose input=TR_samples \
mask=l_trasimeno_ABS layer=1 zcolumn=Hg elevation=TR_Hg segmax=600 where="Hg > 0" tens=35 smo=10
```

**Note:** Diagrammi bivariati - aggiungere script

```
v.db.select map=TR_samples col=Ni,Cr sep=" " file=TR\_Ni\_Cr.dat
v.db.univar map=TR_samples col=Ni
v.db.univar map=TR_samples col=Cr
```

```
v.db.select map=TR_samples col=N_tot,P_tot sep=" " file=TR\_N\_P.dat
v.db.univar -e --verbose map=TR_samples_2@PERMANENT column=Cr_tot
```

```
gmt psxy -Bxa10f5+l"Ni [mg/kg]" -Bya10f50+l"Cr_tot [mg/kg]" \
-BWSen+t"Lago Trasimeno"+glightblue TR_Ni_Cr.dat -JX10 -R-0/100/0/100 \
-Gorange -Sc.3 -W0.5p -V -K > two_biv_diag.ps
```

```
gmt psxy -Bxa.1f.05+l"N_tot [mg/kg]" -Bya.01f.05+l"P_tot [mg/kg]" \
-BWSen+t"Lago Trasimeno"+glightblue TR_N_P.dat -JX10 -R-0/.8/0/.06 \
-Gorange -Ss.3 -W0.5p -V -O -Xc7>> two_biv_diag.ps ; ps2pdf two_biv_diag.ps
```

**Nota: TODO - Dragaggio** Estensione dell'esercitazione

In questa sezione affrontiamo un'altro tipo i intervento, direttamente sul modello numerico della batimetria del lago, ovvero andiamo a determinare i volumi di materiale fine da dragare nell'intorno degli approdi da cui partono i battelli sulle linee commerciali tra la costa e le isole. Il principio di base é creare una zona sicura per evitare durante le operazioni di routine possibili problemi alla navigazione per cui si decide di dragare lungo le rotte per un'ampiezza reale che dipenda dalla profonditá per un massimo di 300 m di ampiezza massima teorica, in modo da avere almeno 3 metri battente d'acqua libero.

Supponiamo che le rotte tra i principali approdi siano:

- San Feliciano - Castiglione - Passignano
  269336E - 4777920N / 265649E - 4779709N / 260389E - 4779645N / 266976E - 4785448N

e vengono connesse con rotte rettilinee.

```
v.in.ascii --overwrite --verbose input=dragaggio.dat output=dragaggio format=standard
v.buffer ty=line in=dragaggio out=buf_dra_300 distance=300
v.overlay --overwrite --verbose  atype=area ainput=buf_dra_300 bty=area binput=l_trasimeno \
operator=and output=dra_buf_300
v.to.rast --verbose input=dra_buf_300 output=dra_buf_300 use=cat
```

la conversione da vettore a raster sará utile nel moment in cui dovremo effettuare alcune operazioni successive. C'é ancora una questiona in sospeso che deriva dalla possibile interferenza tra le operazioni di posa della condotta e degli scavi derivati dal dragaggio, per cui necessita conoscere le aree di sovrapposizione dei due interventi tecnici. Con `v.overlay` determiniamo le aree di sovrapposizione:

```
v.overlay --overwrite --verbose  atype=area ainput=dra_buf_300 bty=area \
binput=buf_100 operator=and output=overlap_dra_acq
v.to.db --verbose map=overlap_dra_acq@PERMANENT option=area columns=area units=meters
```

e aggiungiamo con l'ultimo comando una colonna relativa alle aree relative.

Il blocco di comandi a seguire si occupano di operare calcoli sulle raster per ottenere gli spazi in cui dovranno operare le draghe che usufruiranno di un file vettoriale con le delimitazioni da implementare nel GPS di bordo (es. con formato GPX):

```
r.mask dra_buf_300
r.mapcalc --overwrite expr="dig=if(isnull(mask_3 && dra_buf_300),1,null())"
r.mapcalc --overwrite expr="dig_area=dig"
r.to.vect
v.out.ogr
```

il comando `r.mapcalc` grazie a `r.mask` ha effetto solo in un'area limitata e usa una condizione `if-then-else` sulle celle che hanno valore `null()`. Cambiando maschera di calcolo ed estendendola all'intero specchio lacustre si determina una batimetria di risultante in cui le aree da scavare posseggono il valore di progetto che é fissato a 253.33 m slm , ovvero rendendo disponibile 4 m di battente ai battelli in navigazione:

```
r.mask -r
r.mask raster=b_trasimeno
r.mapcalc --overwrite expr="dig_vol=if(isnull(dig_area),b_trasimeno,(253.33))"
r.colors -e --verbose map=dig_vol@PERMANENT color=elevation
```

Il passaggio finale concerne il calcolo del volume di materiale da sottoporre a scavo utile poi anche per la valutazione della quantitá di metalli presenti nei sedimenti ammettendo che la concentrazione dei metalli derivante dai calcoli espressi in precedenza validi per la superficie acqua-sedimento valga anche per la profonditá.

```
r.mapcalc --overwrite expr="volume=b_trasimeno-dig_vol"
r.volume --verbose input=volume@PERMANENT
```

**Nota:** Calcolo del quantitativo totale dei metalli dispersi nei sedimenti

## 0.13 Ricostruzione di un modello 3D di un sistema fagliato

## 0.14 Calcolo flussi $CO_2$ centro italia

## 0.15 DEM di Mayluu-suu