SERVEI DE SISTEMES D'INFORMACIÓ GEOGRÀFICA I TELEDETECCIÓ
Universitat de Girona

II JORNADAS DE SIG LIBRE

Taller de GRASS

Franciso Alonso Sarría

Gerona, 3 de marzo de 2008

Creative Commons

Reconocimiento – No comercial – Compartir Igual España

CONDICIONES

Usted es libre de

Copiar, distribuir y comunicar públicamente esta obra
hacer obras derivadas

Bajo las condiciones siguientes:

Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

No comercial. No puede utilizar esta obra para fines comerciales.

Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Presentación

En este documento se describen cuatro prácticas cuyo objetivo es mostrar como se realizan con GRASS algunas tareas habituales en un Sistema de información Geográfica. Los asistentes al taller de GRASS de las II Jornadas de SIG libre podrán realizarlas en la segunda parte del mismo tras haber realizado una primera parte común organizada en forma de exposición.

Cuando se haga referencia a un módulo o mapa en concreto se utilizará **negrita**, cuando se menciona una orden concreta se utilizará el tipo de letra `terminal`.

Cuando las órdenes de **GRASS** aparezcan precedidas de **#** significa que son órdenes para escribir directamente en el terminal de texto. Cuando una de estas órdenes sea demasiado grande para entrar en una sola linea se partirá y se añadirá el símbolo **** para indicar que la orden continua en la siguiente linea. Las órdenes de **R** aparecen precedidas de **>**. Las salidas de los programas en el terminal aparecen en tipo de letra `terminal` sin ningún *prompt*.

Cuando una orden es excesivamente larga para escribirse en una sola linea, se parte en dos. En este caso, la primera parte de la linea se termina con un signo \, para indicar que lo que aparece en la siguiente linea debe escribirse a continuación.

Algunas órdenes aparecerán seguidas de un comentario en letra normal y precedido por un símbolo **#** que, lógicamente, no se debe teclear.

Los contenidos de ficheros aparecen en tipo de letra de `terminal` encuadrados y centrados en el texto.

Las prácticas tratadas son las siguientes:

1. **Sistemas de referencia espacial**
2. **Gestión de tablas y bases de datos**
3. **Producción de cartografía en papel**
4. **Interpolación**

Finalmente quisiera agradecer a José Antonio Palazón Ferrando y Fulgencio Cánovas García la ayuda en la edición y revisión del texto y los comentarios acerca de su contenido.

1 Sistemas de referencia espacial

Todas las funcionalidades de proyección, cambio de sistema de proyección y cambio de datum se llevan a cabo en GRASS utilizando la librería PROJ 4. Los módulos que la utilizan, y cuyo funcionamiento se estudiará en esta práctica son:

- **g.proj**
- **g.setproj**
- **m.proj**
- **v.proj**
- **r.proj**

1.1 Definición y consulta del sistema de referencia espacial

La definición del sistema de referencia de una **location** se hace en dos ficheros llamados **PROJ_INFO** y **PROJ_UNITS** dentro del mapset **PERMANENT**. El contenido del primer fichero es más o menos este:

```
name: Universe Transverse Mercator
proj: utm
datum: eur50
towgs84: -131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39
a: 6378388
es: 0.006722670022333321
zone: 30
no_defs: defined
```

Incluyendo información del elipsoide, el datum y los 7 parámetros de la transformación de Bursa Wolf.

y el del segundo:

```
unit: Meter
units: Meters
meters: 1
```

Estos ficheros pueden ser consultados de forma sencilla mediante el módulo **g.proj** y la información puede ser extraída en diversos formatos:

- `g.proj -p` formato de GRASS (simplemente devuelve en pantalla el contenido de los ficheros).
- `g.proj -j`, utiliza el formato de PROJ 4

```
+proj=utm
+no_defs
+zone=30
+a=6378388
+rf=297
+towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39
+to_meter=1
```

- `g.proj -w`, utiliza el formato WKT

```
PROJCS["UTM Zone 30, Northern Hemisphere",
    GEOGCS["international",
        DATUM["European_Datum_1950",
            SPHEROID["International_1924",6378388,297],
            TOWGS84[-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-3],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
```

- `g.proj -e`, formato de Esri (prácticamente igual al WKT)

La salida de **g.proj** se divide en varias lineas para hacerla más legible al usuario, pero con la opción **-f** se obtendrá en una sola linea para permitir su utilización en scripts.

```
# g.proj -fj

+proj=utm +no_defs +zone=30 +a=6378388 +rf=297\
+towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39 +to_meter=1
```

La información referente al sistema de referencia puede modificarse mediante el módulo **g.setproj** o editando directamente **PROJ_INFO** y **PROJ_UNITS** para ello debes ser propietario del mapset **PERMANENT**. Esta modificación no afectará a los mapas pero luego GRASS generará errores cuando intentemos utilizar mapas con un sistema de referencia en una location que tiene definido un sistema de referencia diferente.

1.2 Conversión entre sistemas de referencia espacial

El módulo **v.proj** importa mapas vectoriales de otra location a la location de trabajo. Si ambas locations tienen un sistema de referencia espacial, se realiza la necesaria conversión. El módulo **r.proj** hace lo propio con capas raster, permitiendo remuestrear el contenido de las celdillas utilizando el método del vecino más próximo, interpolación bilineal o convolución cúbica.

El módulo **m.proj** convierte el sistema de referencia de un listado de coordenadas que el usuario pasa por el teclado o desde un fichero ASCII.

1.3 Material para la práctica

Para hacer esta práctica tenemos que abrir dos sesiones de GRASS al mismo tiempo, una con la location **murcia** (sistema de referencia ED50) y otra con la location **murciaetrs89** (sistema de referencia ETRS89).

En la location **murcia** tenemos:

- El mapa vectorial **vertices50** que contiene vértices de la red REGENTE de la Región de Murcia
- El mapa vectorial **Red_Gral_carreteras** que contiene la red de carreteras

En la location **murciaetrs89** tenemos:

- El mapa vectorial **vertices89**

En el directorio **/home/datos/ficheros** tenemos el fichero **qb_murcia.tif** que contiene una imagen Quik bird de la ciudad de Murcia con el sistema de referencia ED50.

1.4 Cambio de sistema de referencia en capas vectoriales

1. Asegúrate de que la región de trabajo coincide con la de los mapas y abre un monitor gráfico para cada sesión de GRASS

   ```
   #g.region vect=vertices50 # En la location murcia
   #g.region vect=vertices89 # En la location murciaetrs89
   #d.mon x1 # En ambas location
   ```

2. Representa en ambos monitores los vértices de la red REGENTE

   ```
   # d.vect vertices50 # En la location murcia
   # d.vect vertices89 # En la location murciaetrs89
   ```

3. Importa el fichero **vertices50** a la location **murciaetrs89**

   ```
   # v.proj input=vertices50 location=murcia mapset=PERMANENT \
   output=vertices50b # En la location murciaetrs89
   ```

4. Importa el fichero vértices89 a la location murcia

   ```
   # v.proj input=vertices89 location=murciaetrs89 mapset=PERMANENT \
   output=vertices89b # En la location murcia
   ```

5. Representa en ambos monitores los vértices transformados
6. Consulta los valores de los mismos vértices en ambos sistemas con **d.what.vect** y comprueba que coinciden con sus coordenadas con **d.where** calcula, más o menos la diferencia en X y en Y de un mismo vértice de un sistema de referencia a otro.

Visto que la conversión funciona con un mapa vectorial vamos a convertir ahora el mapa de carreteras al sistema ETRS89:

```
# v.proj input=Red_Gral_carreteras location=murcia mapset=PERMANENT \
output=Red_Gral_carreterasb
```

1.5 Cambio de sistema de referencia en capas raster

Para comprobar la conversión de mapas raster, vamos a importar el fichero **qb_murcia.tif** a las dos locations:

```
# r.in.gdal -o input=/home/datos/ficheros/qb_murcia.tif output=qb_murcia
```

En principio tendríamos en ambos casos un mensaje de error ya que este fichero, aunque tiene coordenadas UTM, no tiene la información del sistema de referencia de forma explícita. GRASS nos obliga en este caso a utilizar la opción **-o** para forzar la importación o el parámetro **location** para crear una nueva **LOCATION** a partir del sistema de referencia del fichero. Es la manera que tiene GRASS de decirnos *bajo tu responsabilidad*.

Una vez importados los mapas podremos comprobar su adecuación a los sistemas de referencia superponiéndoles la red de carreteras:

```
# g.region rast=qb_murcia
# d.erase
# d.rast qb_murcia
# d.vect Red_Gral_carreteras color=red
```

Comprobarás el desplazamiento que se produce en la location **murciaetrs89** (figura 1).

Ahora vamos a importar correctamente el mapa qb_murcia de la location murcia a la location murciaetrs89:

```
# r.proj input=qb_murcia location=murcia output=qb_murcia_b
```

y comprobamos el resultado:

```
# g.region rast=qb_murcia_b
# d.erase
# d.rast qb_murcia_b
# d.vect Red_Gral_carreteras color=red
```

Puedes ver en la figura 2 que ahora se ajusta perfectamente.

1.6 Series de coordenadas

Vamos a utilizar un listado de vértices de la red REGENTE en formato ED50 y vamos a intentar transformarlos a ETRS89 con el módulo **m.proj**.

Los vértices aparecen en la tabla 1 en formato ED50 y ETRS89. La lista de coordenadas ED50 está en el fichero /home/datos/ficheros/vertices.txt

El módulo **m.proj** necesita que le especifiquemos el datum de entrada y salida en formato de PROJ4. Para obtener esta información debemos ejecutar en ambas location (murcia y murciaetrs89) la siguiente orden:

```
# g.proj -jf
```

En la location **murcia** podemos incluso asignar esta especificación como valor de una variable:

```
# ED50=$(g.proj -jf)
# echo $ED50
```

Figura 1: Mapa raster importado incorrectamente

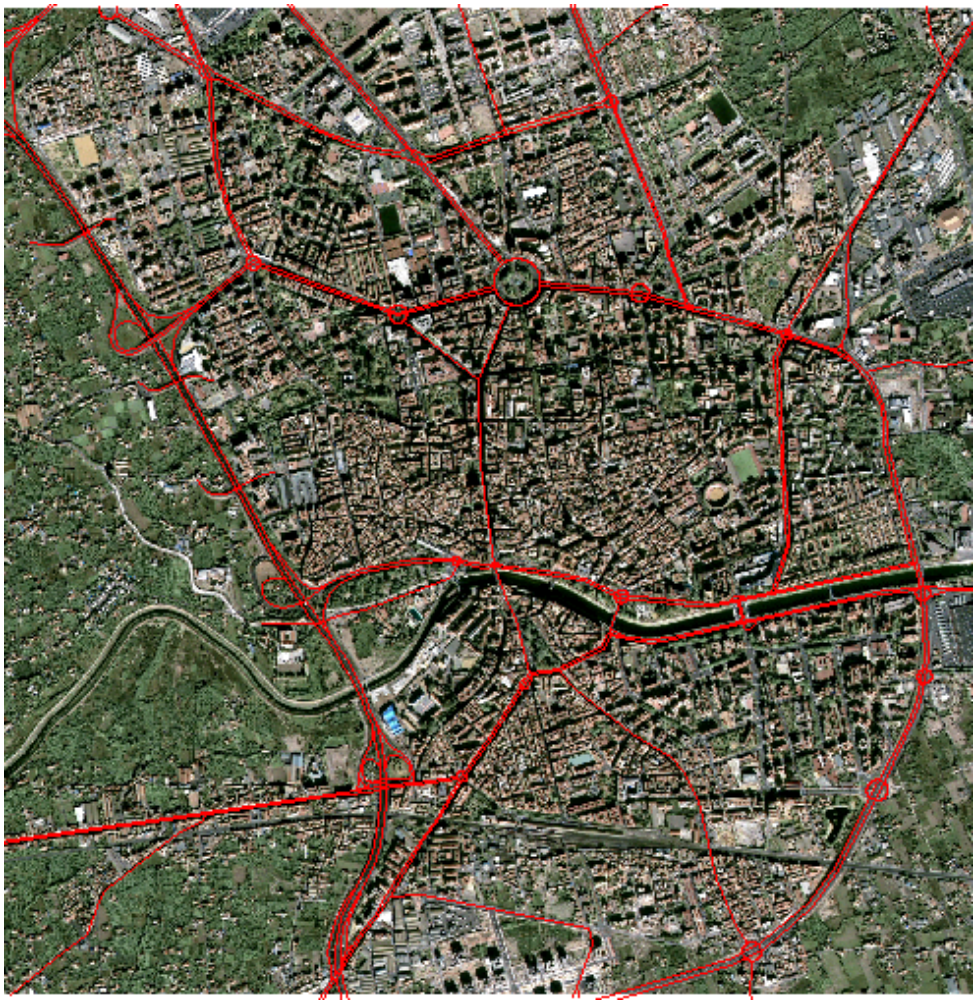

Figura 2: Mapa raster importado correctamente

X_ED50	Y_ED50		X_ETRS89	Y_ETRS89
639094	4290364		638984	4290155.93
671301	4289739		671190.37	4289531.23
643368	4272629		643257.57	4272421.58
671854	4274587		671743.82	4274379.31
619742	4256419		619631.11	4256211.74
647718	4254257		647607.52	4254049.11
660373	4255074		660262.28	4254866.42
561932	4228799		561820.82	4228592
583534	4229886		583422.98	4229679.28
612911	4236896		612800.76	4236688.4
635033	4239863		634921.99	4239655.09
674099	4227386		556171.01	4210032.53
556283	4210240		556171.01	4210032.53
593504	4209357		593393.09	4209149.09
608847	4215772		608736.56	4215564.36
641667	4214640		641556.14	4214432.74
662706	4217599		662595.43	4217391.67
559609	4190445		559496.96	4190237.58
581926	4194095		581814.1	4193887.19
620096	4201852		619984.98	4201644.13
644383	4199146		644271.79	4198937.84
673378	4193488		673266.84	4193280.23
694181	4202766		694070.05	4202557.67
579711	4176625		579599.16	4176417.08
621429	4178337		621317.58	4178129.54
645369	4183867		645258.16	4183659.64
673335	4181332		673223.92	4181124.41
697789	4183847		697677.98	4183639.17
580774	4154137		580662.59	4153929.3
609585	4163460		609474	4163252.39
643460	4169151		643349.09	4168943.87
670862	4167345		670750.72	4167137.35
694603	4168229		694492.19	4168021.41
589802	4142281		589690.87	4142073.82
625761	4140576		625650.03	4140368.17

Tabla 1: Listado de vértices

y el resultado será:

```
+proj=utm +no_defs +zone=30 +a=6378388 +rf=297
+towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39 +to_meter=1
```

Podemos definir una variable equivalente para ETRS89 copiando y pegando la especificación de la otra location:

```
# ETRS89="+proj=utm +no_defs +zone=30 +a=6378137 +rf=298.257222101 \
+towgs84=0,0,0,0,0,0,0 +to_meter=1"
```

Finalmente podemos crear otra variable que contenga el nombre del fichero a importar:

```
# fic=/home/datos/ficheros/verticesed50.txt
```

De esta manera evitaremos tener que teclear órdenes demasiado largas.

```
# m.proj input=$fic proj_in="$ED50" proj_out="$ETRS89"
```

Tras ejecutar la orden verás el listado resultante. Comprueba los errores.

También podemos almacenar los resultados en otro fichero:

```
#m.proj input=$fic proj_in="$ED50" proj_out="$ETRS89">verticesetrs89.txt
```

2 Gestión de tablas y bases de datos

En esta práctica vamos a trabajar con la location **murcia**. Partiremos de un fichero DXF con las lineas de los términos municipales de la Región de Murcia y de una tabla de etiquetas con las coordenadas X e Y en UTM y los nombres de los municipios en el fichero.

El fichero DXF es /home/datos/ficheros/terminos.dxf y el de etiquetas /home/datos/ficheros/terminos.txt

Se trata de un ejemplo no trivial ya que incluye municipios divididos entre varios polígonos, algunos de ellos islas (en sentido topológico y geográfico).

A continuación incorporaremos a la base de datos una tabla con información socioeconómica municipal y finalmente incluiremos información procedente de capas raster en la base de datos.

La gestión de bases de datos relacionales, con tablas enlazadas a mapas vectoriales, se hace por defecto en GRASS mediante ficheros dbf. Esta opción resta versatilidad al sistema ya que no permite aplicar toda la potencia de SQL.

Una opción alternativa es utilizar una base de datos PostgreSQL, si se dispone de ella debemos notificárselo a GRASS mediante la orden **db.connect** especificando el driver (pg, es decir PostgreSQL) que gestionará los accesos a la base de datos y el nombre de la misma.

Para esta práctica el nombre de la base de datos es grass seguido del número de usuario, por ejemplo si eres el usuario 25, tu base de datos se llama **grass25** y la orden será

```
# db.connect driver=pg database=grass25
```

En esta práctica se utilizará un SQL básico fácil de entender con sólo traducir las órdenes del inglés. Aunque SQL admite indistintamente mayúsculas y minúsculas, por claridad en las órdenes se ha utilizado mayúsculas para las clausulas, funciones y operadores SQL y minúsculas para los nombres de tablas y variables.

2.1 Creación del mapa vectorial

En primer lugar debemos importar el fichero con las lineas:

```
# v.in.dxf input=/home/datos/ficheros/terminos.dxf output=lineas
# v.type lineas output=bordes type=line,boundary
```

A continuación importamos las etiquetas:

```
# v.in.ascii input=/home/datos/ficheros/terminos.txt fs=";" \
output=puntos
# v.type puntos output=centroides type=point,centroid
```

El final de la salida de texto de **v.in.ascii** nos indica el número de elementos importados y que se ha generado la topología:

```
Building topology ...
55 primitives registered
Building areas:  100%
0 areas built
0 isles built
Attaching islands:
```

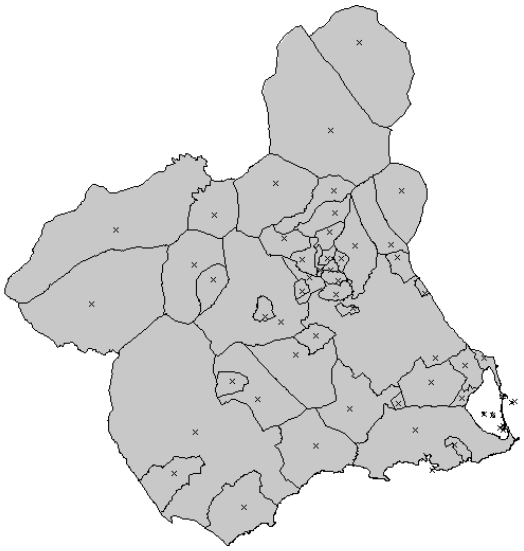

Figura 3: Mapa vectorial de municipios

```
Attaching centroids:  100%
Topology was built.
Number of nodes     :   55
Number of primitives:   55
Number of points    :   55
Number of lines     :   0
Number of boundaries:   0
Number of centroids :   0
Number of areas     :   0
Number of isles     :   0
```

Esto es así en muchos de los programas de manejo de datos vectoriales, ya que la librería de GRASS para manejo de vectoriales tiene una función para generación de topología que utilizan prácticamente todos los módulos.

En ambos casos ha sido necesario modificar con **v.type** el tipo de datos del fichero (puntos por centroides y lineas por límites de area).

Ahora ya podemos ejecutar **v.patch** para integrar ambas capas. El mismo módulo resolverá la topología y generará un mapa topológicamente correcto a la espera de que enlacemos la tabla.

```
# v.patch input=bordes,centroides output=terminos
```

2.2 Creación y enlace de una tabla al mapa vectorial

Para este paso vamos a utilizar **db.copy** para copiar la tabla **centroides** (que se creo con el módulo **type**) y la llamamos **terminos**. En realidad podríamos enlazar **centroides**, pero es más claro si las tablas se llaman como los ficheros vectoriales, además al borrar el mapa **centroides** eliminaríamos también la tabla.

```
# db.copy from_table=centroides to_table=terminos
```

El módulo **v.in.ascii** no permite modificar los nombres de las columnas y crea unos genéricos. Para modificarlos podemos mandar órdenes SQL a la base de datos a través del módulo **db.execute**. En primer lugar eliminamos las columnas que contiene las coordenadas de los centroides y en segundo lugar renombramos las columnas que si interesan.

```
# echo "alter table terminos drop column dbl_1"|db.execute
# echo "alter table terminos drop column dbl_2"|db.execute
# echo "alter table terminos rename column int_1 to codigo"|db.execute
# echo "alter table terminos rename column str_1 to nombre"|db.execute
```

Finalmente enlazamos la tabla con la base de datos:

```
# v.db.connect terminos table=terminos
```

No está demás, una vez obtenido el mapa que queríamos, hacer un poco de limpieza:

```
# g.remove vect=puntos,centroides,lineas,bordes
```

Ahora vamos a poblar la tabla con nueva información. Para ello tenemos que añadir columnas a la tabla, especificando el formato de los datos, con **v.db.addcol**. La opción más obvia es calcular área y perímetro de cada polígono (ojo, la superficie total por municipio habrá que recalcularla después ya que cada municipio está formado por varios polígonos).

```
# v.db.addcol terminos columns="area real, perimetro real"
```

Para incluir en la tabla información procedente de la geometría debes utilizar el módulo **v.to.db**.

```
# v.to.db terminos option=area column=area units=k
# v.to.db terminos option=perimeter column=perimetro units=k
```

2.3 Incorporación de otras tablas

Para incorporar otro tipo de información podemos añadir tablas a la base de datos. A continuación vamos a añadir como ejemplo una pequeña tabla con información socioeconómica (**/home/datos/ficheros/tabla.txt**) que contiene las siguientes columnas:

1. Código de municipio
2. Población (censo de 1991)
3. Superficie de cultivo en secano (hectáreas)
4. Superficie de cultivo en regadío (hectáreas)
5. Renta familiar disponible bruta (1995)
6. Renta familiar disponible bruta (2000)

Para crear la tabla hay que pasar la orden SQL correspondiente a través de **db.execute**

```
# echo "CREATE TABLE estadisticas (codigo int, poblacion int, secano float,\
regadio float, renta95 float, renta2000 float)"|db.execute
```

y posteriormiente copiar los datos del fichero a la tabla. Para ello la orden se le pasa a **psql** en lugar de a **db.execute** ya que **db.execute** no admite la contrabarra en una orden SQL, al ejecutar **psql** es necesario indicar la base de datos con la que se trabaja.

```
# echo "\COPY estadisticas FROM '/home/datos/ficheros/tabla.txt' \
WITH DELIMITER ' '"|psql grassn # donde n es tu número de usuario
```

Ya puedes consultar los contenidos de la tabla, las consultas se hacen a través del módulo **db.select** mientras que, como has visto, otro tipo de órdenes SQL se pasan a través del módulo **db.execute**.

```
# db.select sql="select * from estadisticas"
```

A esta nueva tabla municipal pueden incluirse columnas que contengan información geométrica agregada procedente del mapa vectorial. Por ejemplo el área total de los municipios como la suma de las áreas de los polígonos que los forman.

Para ello debemos generar una tabla intermedia que contendrá las superficies municipales:

```
# db.select sql="SELECT codigo,SUM(area) AS areatotal INTO areatotal \
FROM terminos GROUP BY codigo"
```

y que nos servirá para actualizar la tabla de estadisticas:

```
# echo "ALTER TABLE estadisticas ADD COLUMN area FLOAT"|db.execute
# echo "UPDATE estadisticas e SET area=a.areatotal FROM areatotal a \
WHERE e.codigo=a.codigo"|db.execute
```

Finalmente borramos la tabla intermedia.

```
# echo "DROP TABLE areatotal"|db.execute
```

2.4 Agregar información procedente de mapas raster

La tabla enlazada al mapa vectorial puede recibir también información procedente de una capa raster, más concretamente, estadísticos de variables almacenadas en formato raster para los diferentes polígonos que componen el mapa.

```
# g.region rast=mde500
# v.rast.stats vector=terminos raster=mde500 colprefix=mde
```

Este módulo tiene el inconveniente de que no permite seleccionar los estadísticos necesarios y genera diversas columnas con información no necesariamente útil. Puedes comprobarlo representando en pantalla el mapa términos y consultándolo con **d.what.vect** (figura 4).

2.5 Visualización

En las últimas versiones de GRASS se ha incorporado un módulo para la realización de cartografía temática a partir de mapas vectoriales. Para utilizarlo vamos a crear en primer lugar una columna en la tabla **terminos** que actualizaremos con valores de población de la tabla **estadisticas**.

```
# echo "alter table terminos add variable float"|db.execute
# echo "update terminos t set variable=e.poblacion \
from estadisticas e where e.codigo=t.codigo"|db.execute
```

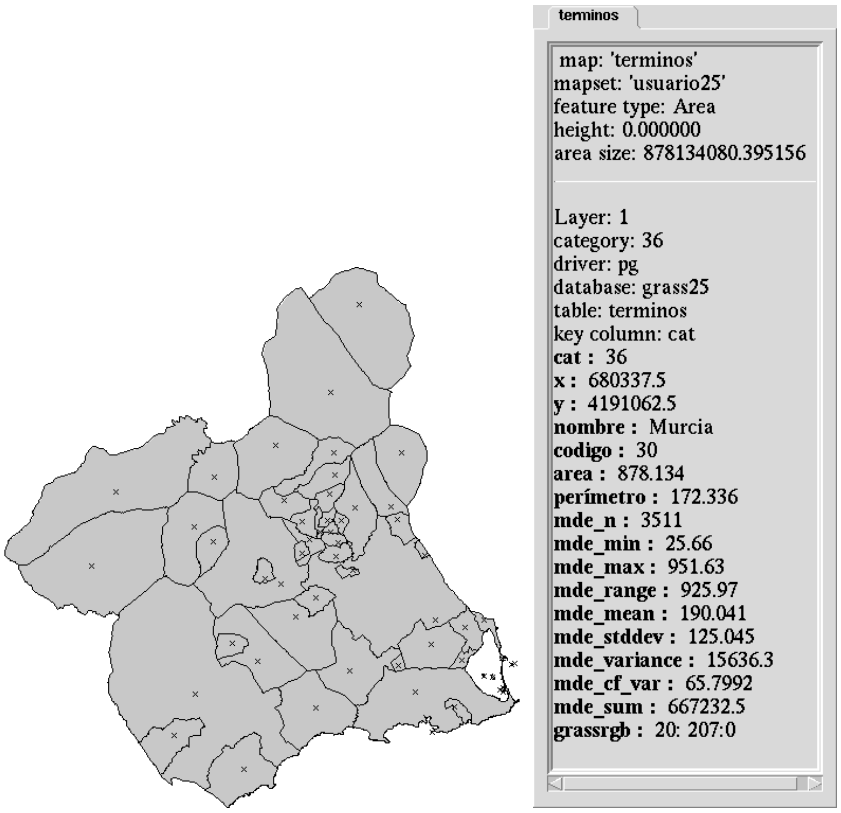

Figura 4: Mapa vectorial de municipios

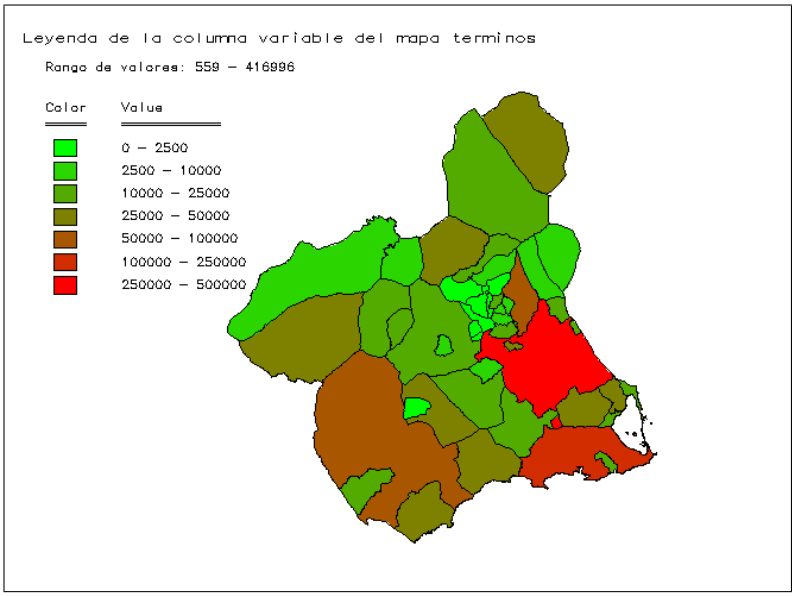

Figura 5: Creación de mapas temáticos. Mapa de población

A continuación utilizamos el módulo **d.vect.thematic**:

```
#d.vect.thematic -l terminos column=variable themecalc=custom_breaks \
breakpoints="0 2500 10000 25000 50000 100000 250000 500000" \
colorscheme=green-red
```

```
#d.vect.thematic terminos column=variable themecalc=custom_breaks \
breakpoints="0 2500 2500 10000 25000 50000 100000 250000 500000" \
colorscheme=green-red
```

La primera de estas órdenes utiliza la opción **-l** crea una leyenda y la segunda superpone el mapa (figura 5). La figura 6 muestra un mapa similar, pero en este caso utilizando la elevación media como variable a cartografiar.

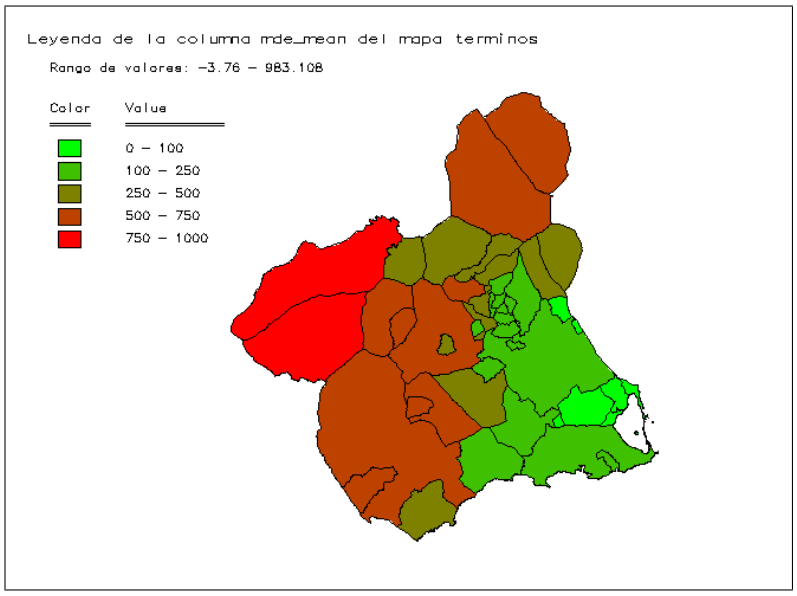

Figura 6: Creación de mapas temáticos. Mapa de elevación media

3 Producción de cartografía en papel

En esta práctica se va a utilizar GMT y el driver PNG de GRASS para generar cartografía con el objetivo de imprimirla en papel. El planteamiento básico consiste en que cada mapa se define y configura como un pequeño script que resulta fácil editar y corregir.

3.1 El driver PNG

El sistema de visualización de GRASS utiliza por defecto el entorno X11. Sin embargo puede utilizarse como alternativa el driver PNG, utilizando las librería libpng[1] y zlib[2].

Este driver se activa como si fuese un monitor gráfico (`d.mon start=PNG`) y se cierra del mismo modo (`d.mon stop=PNG`). Todas las órdenes de visualización que se ejecuten entre tanto iran generando capas en un fichero PNG en lugar de en un monitor gráfico.

Existe un conjunto de variables de entorno que permiten modificar las características del fichero PNG resultante:

- GRASS_WIDTH=xxx
  Establece la anchura de la imagen en puntos (por defecto 640).
- GRASS_HEIGHT=yyy
  Establece la altura de la imagen (por defecto 480).

[1] http://www.libpng.org/pub/png/
[2] http://www.info-zip.org/pub/infozip/zlib/

- GRASS_PNGFILE=filename
  Nombre del fichero que se va a generar (por defecto map.png). Si el fichero tiene extensión ,ppm se creará un PPM en lugar de un fichero PNG.
- GRASS_BACKGROUNDCOLOR=RRGGBB
  Especifica el color de fondo en notación RGB hexadecimal (por defecto 000000 o sea negro).
- GRASS_TRUECOLOR=[TRUE|FALSE]
  Habilitar o no la opción TRUECOLOR, es necesario habilitarla para conseguir colores de 24 bytes.

3.2 GMT

GMT[3] (Generic Mapping Tools) pone a disposición del usuario un conjunto de módulos orientados a la producción de cartografía a partir de datos codificados según los modelos lógicos (raster, vectorial, lista de puntos) habituales en los SIG. Los datos raster se almacenan por defecto en formato propio (GMT/ netCDF) pero también se admiten ficheros binarios con datos en coma flotante, enteros, bytes o bits. Los datos vectoriales se almacenan como simples pares de coordenadas, no se incluye por tanto información topológica pero de hecho no tiene sentido incluirla si el resultado que espera obtenerse es un mapa para imprimir en papel.

El manejo de estos módulos en linea de comandos y la posibilidad de combinarlos (entre si y con otras herramientas UNIX) así como el elevado número de opciones de cada uno de ellos, convierte GMT en un entorno de maquetación de mapas extremadamente flexible. La contrapartida de esta flexibilidad es un lenguaje que puede llegar a ser bastante complejo y hermético.

Los módulos de GMT más utilizados para crear cartografía son:

- **psbasemap**, prepara la base cartográfica teniendo en cuenta la extensión de los datos y la escala.
- **grdimage**, incluye un fichero raster.
- **psimage**, incluye un fichero de imagen (fichero gráfico *SUN rasterfile*).
- **grdcontour**, genera isolineas a partir de un mapa raster.
- **psxy**, dibuja elementos vectoriales (puntos, lineas, polígonos).
- **pstext**, escribe textos.
- **pslegend**, Incluye una leyenda.

Existen muchos más módulos pero, o bien no son directamente de propósito cartográfico, o bien hacen cosas que se pueden hacer igual con GRASS. En parte por simplificar la práctica y en parte porque muchas de las cosas que hacen estos módulos se pueden hacer de forma más sencilla con GRASS, vamos a utilizar sólo los módulos **psbasemap** para obtener marcos con coordenadas en los bordes con una determinada escala y adaptados a un tamaño de papel, y **psimage** para introducir gráficos previamente generados con el driver **PNG** en GRASS.

[3] gmt.soest.hawaii.edu/

El módulo **grdcontour** es útil para introducir isolineas etiquetadas a partir de un mapa raster.

GMT genera mapas en formato postscript mediante la inclusión progresiva de diversos elementos al mapa/fichero por parte de diferentes módulos. Un fichero postscript se caracteriza por tener:

- Cabecera
- Código para la representación de diversos elementos
- Cierre

Generalmente cuando un programa escribe un fichero postscript debe escribir los tres elementos. En el caso de GMT, puesto que diversos módulos escriben código de forma sucesiva en un mismo fichero de salida, el módulo que inicia el mapa debe escribir la cabecera pero no el cierre; por el contrario, el último en hacerlo debe escribir el cierre y no la cabecera; finalmente los módulos intermedios no deben escribir ni cabecera ni cierre.

Para que un módulo de GMT no escriba la cabecera se le pasa la opción **-O** y para que no escriba el cierre la opción **-K**

3.3 Crear un mapa en formato PNG

Vamos a crear un fichero PNG con un mapa sencillo. En primer lugar es necesario dar valores a las variables que van a definir la imagen:

```
# g.region rast=mde500
# export GRASS_WIDTH=1000
# export GRASS_TRUECOLOR=TRUE
# export GRASS_BACKGROUNDCOLOR=255255255
# export GRASS_PNGFILE=mimapa.png
```

La anchura se expresa en pixels y la altura se calcula a partir de la anchura para mantener las proporciones.

A continuación activamos el driver:

```
# d.mon start=PNG
```

ejecutamos las órdenes de visualización

```
# d.rast mde500
```

y cerramos el driver:

```
# d.mon stop=PNG
```

Podemos ya visualizar el mapa creado. Vamos a utilizar el programa display que forma parte del paquete Imagemagick que suele estar presente en las distribuciones de linux.

```
# display mimapa.png
```

3.4 Crear un mapa en formato postscript

Cuando el objetivo es conseguir un mapa en papel, es importante ajustar bien la resolución en puntos por pulgada y la escala. Para ello puede utilizarse el siguiente script que, a partir de la anchura en papel deseada para el mapa resultante y la resolución en puntos por pulgada (dpi), calcula el ancho y alto de la imagen en píxeles y la altura en centímetros.

Como ves se utiliza la salida de **g.region -p** para procesarla mediante pequeños scripts de awk y obtener los valores de las variables. Las variables **alto** y **ancho** se expresan en píxeles, mientras que **altocm** y **anchocm en centímetros**

```
# dpi=300
# i2c=0.39
# anchocm=22

# ancho=$(echo $dpi|awk '{print '$anchocm'*'$i2c'*'$dpi'}')

# minx=$(g.region -p|awk '$1=="west:" {print $2}')
# miny=$(g.region -p|awk '$1=="south:" {print $2}')
# maxx=$(g.region -p|awk '$1=="east:" {print $2}')
# maxy=$(g.region -p|awk '$1=="north:" {print $2}')

# dx=$(echo $minx|awk '{print '$maxx'-'$minx'}')
# dy=$(echo $minx|awk '{print '$maxy'-'$miny'}')

# alto=$(echo $ancho,$dy,$dx|awk -F "," '{ print $1*$2/$3}')
# altocm=$(echo $alto,$i2c,$dpi|awk -F "," '{print $1/($2*$3)}')
```

Script para ajustar los valores de anchura y altura en papel y de imagen.

A continuación das a las variables los resultados obtenidos con el script

```
# export GRASS_WIDTH=$ancho
# export GRASS_HEIGHT=$alto
```

y repetimos lo hecho en el ejercicio anterior:

```
# export GRASS_BACKGROUNDCOLOR=25525255
# export GRASS_PNGFILE=mimapa.png
# d.mon start=PNG
# d.rast mde500
# d.mon stop=PNG
```

Convertimos el fichero PNG a formato SUN rasterfile para que lo entienda GMT.

```
# pngtopnm mimapa.png|pnmtorast -standard>mimapa.ras
```

damos valor a algunas variables de entorno de GMT

```
# gmtset D_FORMAT %7.0f PAPER_MEDIA a4 ANNOT_FONT_SIZE_PRIMARY 10
```

Hemos modificado el tamaño de las anotaciones y su formato y hemos especificado que queremos obtener un mapa en A4.

Preparamos la base para el mapa, observa que utilizamos las coordenadas de los límites de la región y la escala la definimos con el ancho y alto en centímetros, todos estos parámetros los obtuvimos con el script anterior. Observa también la presencia de la opción **-K** para que el programa no cree el cierre del fichero postscript.

```
# psbasemap -R${minx}/${maxx}/${miny}/${maxy} -JX${anchocm}c/${altocm}c
-B0 -K >mimapa.ps
```

Añadimos el fichero gráfico creado. Observa también la presencia de las opciones **-K** y **-O** para que el programa no cree ni la cabecera ni el cierre del fichero postscript.

```
# psimage mimapa.ras -W${anchocm}c/${altocm}c -C0c/0c -O -K>>mimapa.ps
```

Cerramos el mapa, ahora sólo usamos la opción **-O** para que el programa no cree la cabecera.

```
# psbasemap -R -J -B0 -O >>mimapa.ps
```

El resultado ha quedado almacenado en el fichero **mimapa.ps** y lo visualizas con **gv**:

```
# gv mimapa.ps
```

3.5 Un mapa algo más sofisticado

En este ejercicio haremos lo mismo que antes pero ahora añadiendo contenido al mapa , tanto desde GRASS como desde GMT.

El programa está en el fichero `mapa_grande.sh` en el directorio `/home/datos/ficheros`. Ábrelo con **emacs** u otro editor que te guste.

```
# emacs /home/datos/ficheros/mapa_grande.sh
```

y ejecútalo por pasos copiando y pegando órdenes. El resultado será algo similar al que aparece en la figura 7.

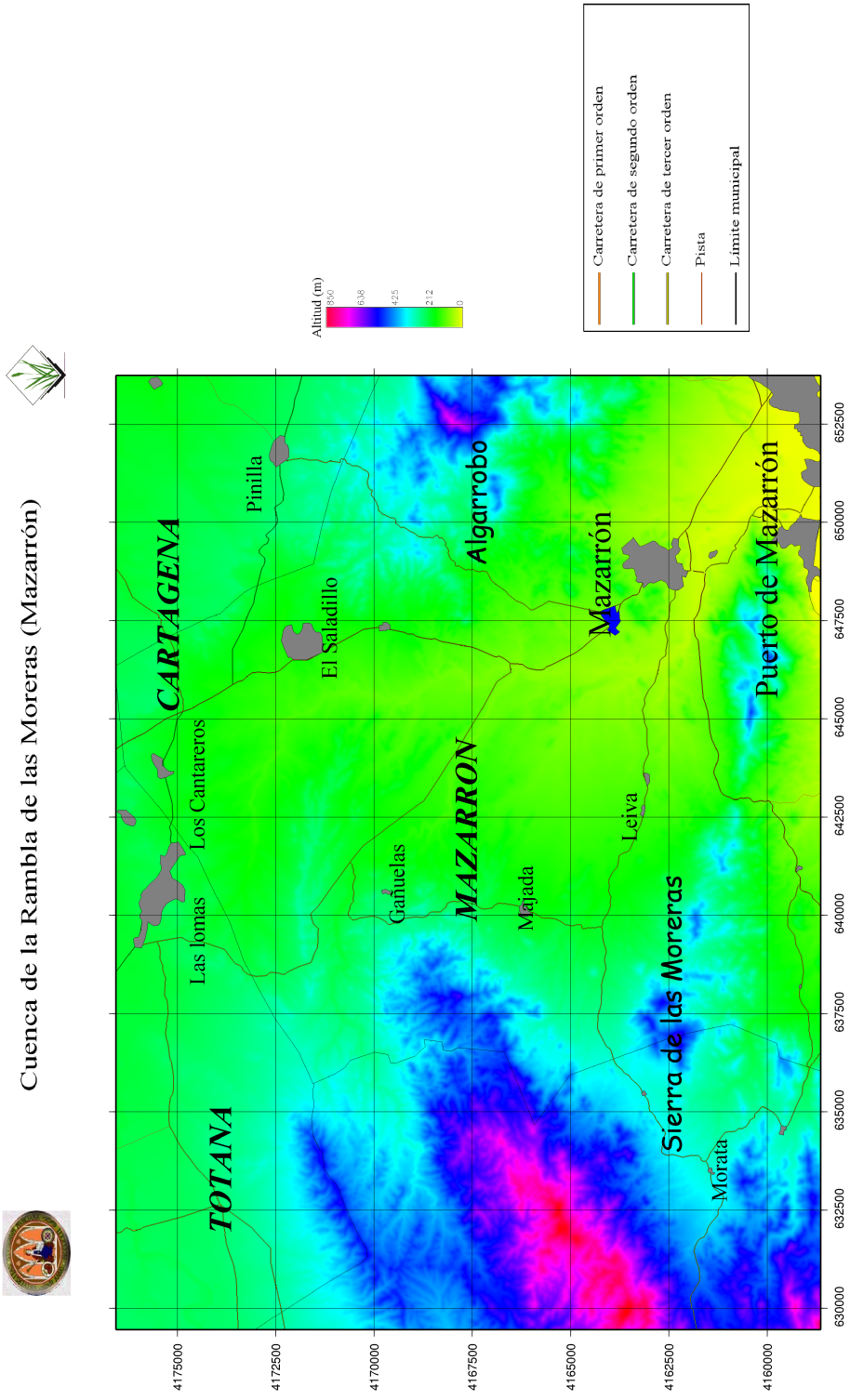

Figura 7: Mapa creado con GRASS y GMT

4 Interpolación

En esta práctica vamos a utilizar tres programas para obtener capas raster de variables climáticas utilizando métodos de interpolación por regresión y krigeado. Estos programas son GRASS[4] para almacenar los datos raster, álgebra de mapas e interpolación; R[5] para los análisis estadísticos y PostgreSQL[6] para almacenar los datos climáticos. Además dentro de R utilizaremos el paquete gstat[7] que implementa la práctica totalidad de las funciones de este programa de geoestadística.

De los tres programas, R es el que proporciona un lenguaje más flexible y mayores posibilidades de acceso a los otros dos, así que R va a ser el programa desde el que manejaremos toda la operación. Más que una práctica de GRASS con R es una práctica de R con GRASS.

4.1 Entrada a los programas

El fichero **/home/datos/ficheros/R_SQL.R** contiene un script con todas las órdenes de R que forman la práctica.

El primer paso es entrar en GRASS y después entraremos en R:

`grass63`
GRASS 6.3.0RC4 (murcia): >`R`

De este modo tendremos acceso tanto a las órdenes de R como a las de GRASS.

A continuación debemos cargar el conjunto de librerías y paquetes externos:

```
> library(RPgSQL)
> library(spgrass6)
> library(gstat)
> source("/home/datos/ficheros/funciones.R")
```

Este último fichero incluye las funciones **consulta** que ejecuta las consulta a PostgreSQL encapsulando las funciones de la librería RPgSQL, y **creagrid** que crea un objeto grid de R a partir de la información de la región de GRASS.

El primero aporta las funciones para conectar con PostgreSQL vía consultas SQL; el segundo permite acceder a los ficheros de GRASS desde R para lectura y escritura; el tercero incorpora las funciones del programa **gstat** que implementa diversas herramientas de geoestadística. Al cargar **spgrass6** se cargarán los paquetes **sp**[8] y **rgdal**, si no estuvieran disponibles R daría un error.

El fichero **funciones.R** contienen funciones que simplifican, de cara al usuario, la interacción entre GRASS, R y PostgreSQL.

[4] grass.itc.it
[5] cran.r-project.org
[6] www.postgresql.org/
[7] www.gstat.org/
[8] este paquete se ha convertido en una especie de *lingua franca* para integrar los diversos paquetes de R que manejan información espacial

4.2 R y PostgreSQL. Consulta a la base de datos

En primer lugar vamos a obtener los datos necesarios para hacer esta práctica de la base de datos, para ello hay que establecer la conexión:

```
db.connect(dbname="clima_segura",host="localhost")
```

A continuación vamos a crear una orden SQL que nos devolverá los observatorios de INM disponibles en la base de datos y representará su distribución espacial (figura 8):

```
> datos=consulta("select codigo,x,y,z from clima_obs where x>0 and y>0;")
> plot(datos$x,datos$y,cex=datos$z/500)
```

La función **consulta** toma como parámetro una orden en lenguaje SQL y devuelve la tabla resultante evitando al usuario utilizar las funciones de la librería RPgSQL (ver el fichero R_SQL.R)

Una consulta algo más compleja va a permitir extraer los datos de precipitación un mes concreto (octubre de 1990):

```
> orden="SELECT o.x,o.y,o.z,p.pluv FROM clima_obs o, menspluv p \
        WHERE o.codigo=p.codigo and p.mes=10 and p.ano=1990 and o.x>0 and o.y>0;"

> datos=consulta(orden)
```

Podemos ver el histograma de los valores de precipitación así como su distribución espacial (figura 9).

```
> hist(datos$pluv)
> tam=(datos$pluv-min(datos$pluv)+1)/50
> plot(datos$x,datos$y,cex=tam,col=datos$z/200,pch=16)
```

4.3 Análisis estadístico

Ahora viene la parte más compleja desde el punto de vista estadístico, tenemos tres variables independientes (x,y,z) que representan las tres coordenadas y una variable que asumimos que puede ser dependiente de las anteriores (pluv). De manera orientativa podemos representar gráficamente pluv contra cada una de las variables independientes (figura 10)

```
> attach(datos)
> plot(datos$x,datos$pluv)
> plot(datos$y,datos$pluv)
> plot(datos$z,datos$pluv)
```

Parece *a priori* que existe una relación respecto a Y pero no respecto a X y Z, aunque si pudiera ser respecto a X^2. Para comprobarlo vamos a generar y analizar una serie de modelos lineales de regresión múltiple:

En primer lugar, y para simplificar la notación, utilizaremos la función **attach** de R que extrae las columnas de la tabla datos y las incorpora como objetos de R.

```
> x2=x^2
> summary(lm(pluv~x*x2*y*z))
```

con esta orden le decimos a R que queremos determinar la relación de pluv respecto a X, X^2, Y y Z, y todas sus posibles interacciones (por eso las variables independientes aparecen multiplicándose. La respuesta de R nos da una estimación de los coeficientes para cada variable junto a su nivel de significación, vamos a rechazar niveles de significación menores de 0.05.

El resultado final es que la única variable independiente con un nivel de significación alto es Y, así que ejecutamos dicho modelo para obtener sus coeficientes y sus residuales:

```
> modelo=lm(pluv ~ y)
> coef=coefficients(modelo)    # Obtención de los coeficientes
```

El modelo resultante sería P=-1650 + 0.0004031303 Y

4.4 R y GRASS. Creación de una capa raster con los resultados del modelo lineal

Hasta ahora se ha estado trabajando con funciones de R, ha llegado el momento de utilizar módulos de GRASS. Si recuerdas hemos entrado en GRASS y luego en R, así que para ejecutar ordenes de R basta con introducirlas mediante la orden **system** que admite como parámetro ordenes para el sistema operativo.

Con los coeficientes anteriormente obtenidos, ya podemos hacer una capa de precipitaciones con **r.mapcalc**.

```
> system("g.region interpolacion")
> system("r.mapcalc 'modelo=-1650 + 0.0004031303*y()'")
```

Ten precaución de no equivocarte con las comillas simples y dobles, además deberás haber verificado que la región de trabajo de GRASS sea la correcta para hacer la interpolación.

La capa creada es un estimación de la precipitación (p') que cumple que $p = p' + \epsilon$ donde ϵ son los residuales. A continuación van a analizarse estos residuales.

4.5 Análisis de los residuos

Podemos acceder a los residuales con la orden:

```
> res=residuals(modelo)  # Obtención de residuales
```

y visualizarlos con las siguientes órdenes:

```
> plot(x,y,pch=16,col="blue",cex=0.25)
> points(x,y,cex=abs(res/10),col=(res<0)+1,pch=16 )
```

Los resultados aparecen en la figura 11, el diámetro de los puntos es proporcional al valor absoluto de los residuales, en negro aparecen los residuales negativos, en rojo los positivos y en azul los más próximos a cero.

Como ya se ha dicho, estos residuales (ϵ) muestran la discrepancia entre el modelo (p')y el dato real (p) en aquellos puntos en que se dispone de este.

4.6 R y GRASS. Exportar la capa de puntos

Podemos ahora crear un objeto de tipo SpatialPointsDataFrame tal como se definen en la librería **sp** que contenga los puntos que representan los observatorios y que obtuvimos con la consulta SQL más los residuales del modelo.

```
> datos_sv=data.frame(x,y,pluv,res)
> coordinates(datos_sv)=~x+y
```

Esta capa de puntos puede ahora exportarse a GRASS para crear una capa vectorial llamada **inm**:

```
> writeVECT6(datos_sv,"inm", v.in.ogr_flags="-o --overwrite")
> system("d.mon x1;d.rast mde500;d.vect inm")
```

El resultado de esta última orden aparece en la figura 12

Si la orden **writeVECT6** da problemas puedes probar hacer lo mismo en dos pasos, primero guardar la tabla como fichero de texto y a continuación crear una capa de GRASS a partir de este fichero.

```
> write.table(datos_sv,"inm.txt",col.names=F,row.names=F)
> system("v.in.ascii inm.txt output=inm fs=' '")
```

4.7 R y GRASS. Añadir capas raster como variables independientes

Lógicamente, puede haber muchas otras variables que explique la distribución espacial de las variables climáticas y que, por tanto, permitan obtener un mejor modelo de interpolación. Si estas variables estuviran disponibles en forma de capas raster, no es el caso de esta práctica, podríamos acceder a sus valores en los mismos puntos en los que tenemos datos climáticos.

Vamos a suponer que tuvieramos dos capas raster con las orientaciones (**aspect_20**) y la distancia a la costa (**dist_costa**) y que las quisieramos incluir en nuestro modelo.

Tendríamos que muestrear las dos capas raster que nos interesan y crear dos nuevas capas de puntos (**orient** y **distan** para guardar los valores muestreados:

```
> system("v.sample input=inm output=orient rast=aspect_20")
> system("v.sample input=inm output=distan rast=dist_costa")
```

Finalmente recuperaríamos estos valores creando para ello dos nuevas variables **ori** y **dist**:

```
> datos2=getSites6("orient");ori=datos2$rast_val
> datos3=getSites6("distan");dist=datos3$rast_val
```

4.8 GSTAT. Interpolación de residuales mediante kriggeado

Puesto que como hemos visto antes $p = p' + \epsilon$ y hemos creado una capa raster de p' con **r.mapcalc** sería buena idea interpolar los valores de ϵ para después sumarlos a los de p' y obtener una mejor estimación de p. Vamos a utilizar para ello el procedimiento de kriggeado que implementa la librería gstat.

Para obtener una capa con los residuales interpolados mediante el método del krigeado, debemos en primer lugar calcular la función semivariograma:

```
> v=variogram(res~1,datos_sv)
> plot(v)
```

La nube de puntos resultante debe ajustarse a un semivariograma teórico. Existen diversos modelos, los más habituales aparecen en la figura **??**.

Una vez escogido el modelo que mejor parece aproximarse a nuestros datos, debemos determinar unos valores iniciales de los parámetros para que la función **fit.variogram** obtenga unos parámetros optimizados. Esta primera aproximación se pasa con la la orden **vgm(pepita, modelo, alcance, meseta)**. En la figura 14 se presentan las magnitudes a que corresponden estos parámetros.

El semivariograma teórico se almacena en el objeto fv y, finalmente, la orden **variogramLine** permite superponerlo al semivariograma muestral (figura 14).

```
> fv=fit.variogram(v,vgm(100,"Sph",40000,500))
> plot(variogramLine(fv,50000))
> plot(v)
> lines(variogramLine(fv,50000))
```

El siguiente paso es ya hacer el krigeado. La función **gmeta** importa a R un objeto que contiene la región de trabajo. Con este objeto podemos, con la función **creagrid** que está en el fichero **R_SQL.R** y la función **gridded**, crear un objeto **SpatialGrid** definido en la librería **sp** que contendrá la interpolación y que posteriormente podrá exportarse como capa raster de GRASS.

```
> G=gmeta6()
> test.grid=creagrid(G,500)
> gridded(test.grid) = ~x+y
> kk <- krige(res~1, datos_sv, test.grid, model = fv)
> summary(kk)
```

La función **summary** presentará los estadísticos del objeto que contiene la interpolación.

4.9 R y GRASS. Exportación a GRASS de las capas raster y ajustes finales

Podemos ya terminar el proceso exportando los resultados a GRASS con la función **writeRAST6** de la librería **spgrass6** y sumando la capa del modelo lineal a la capa con los residuales interpolados.

```
> writeRAST6(kk,"residual",zcol='var1.pred')
> system("r.mapcalc 'precipitacion=modelo+residual'")
```

A continuación podemos verificar resultados ejecutando a través de R las siguientes órdenes de GRASS:

```
> system("d.rast precipitacion")
> system("d.vect datos")
> system("d.legend -m precipitacion")
> system("d.what.vect")
```

Con lo que se obtendrá el resultado que aparece en la figura 15.

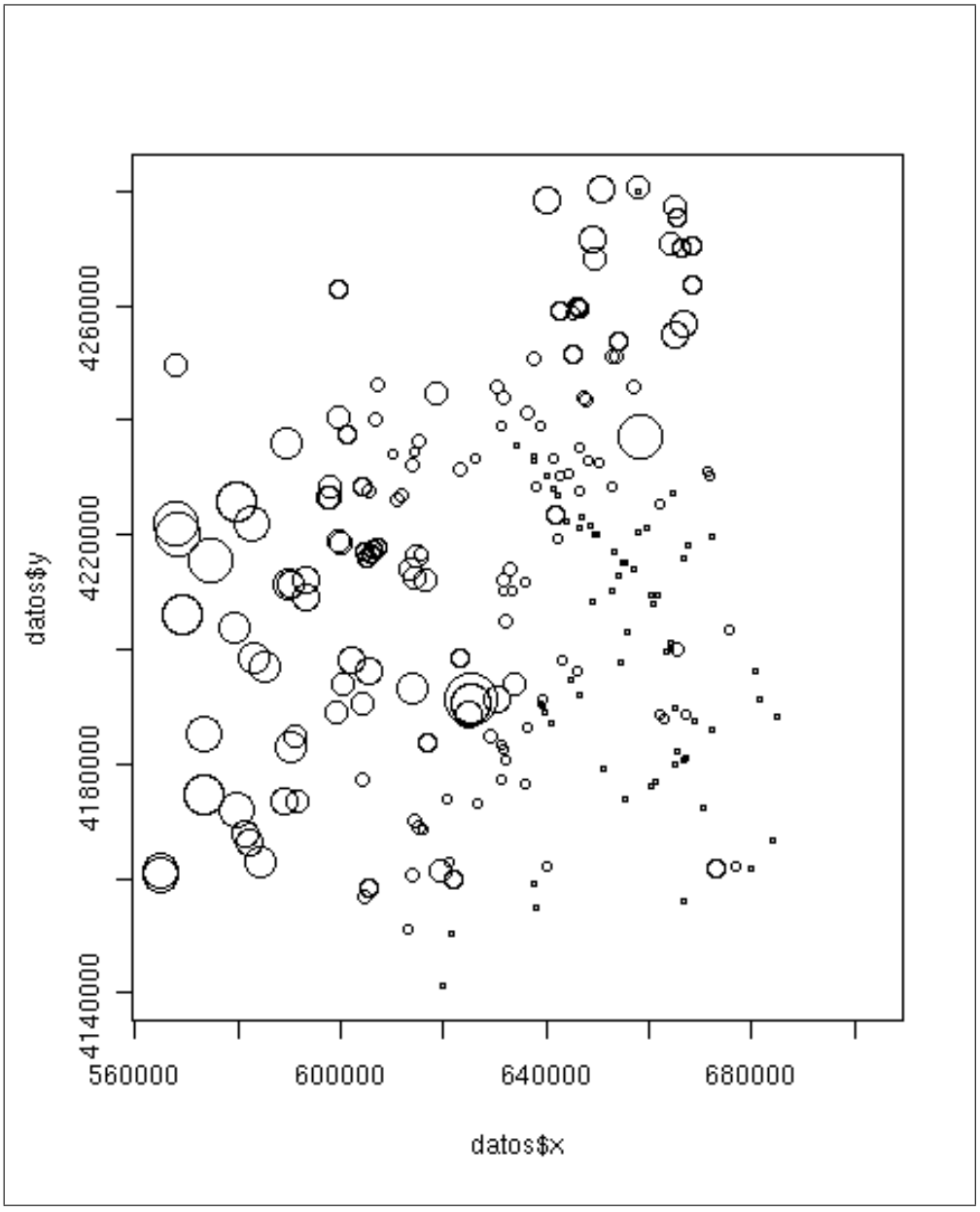

Figura 8: Distribución de observatorios, el diámetro de los círculos es proporcional a la altitud

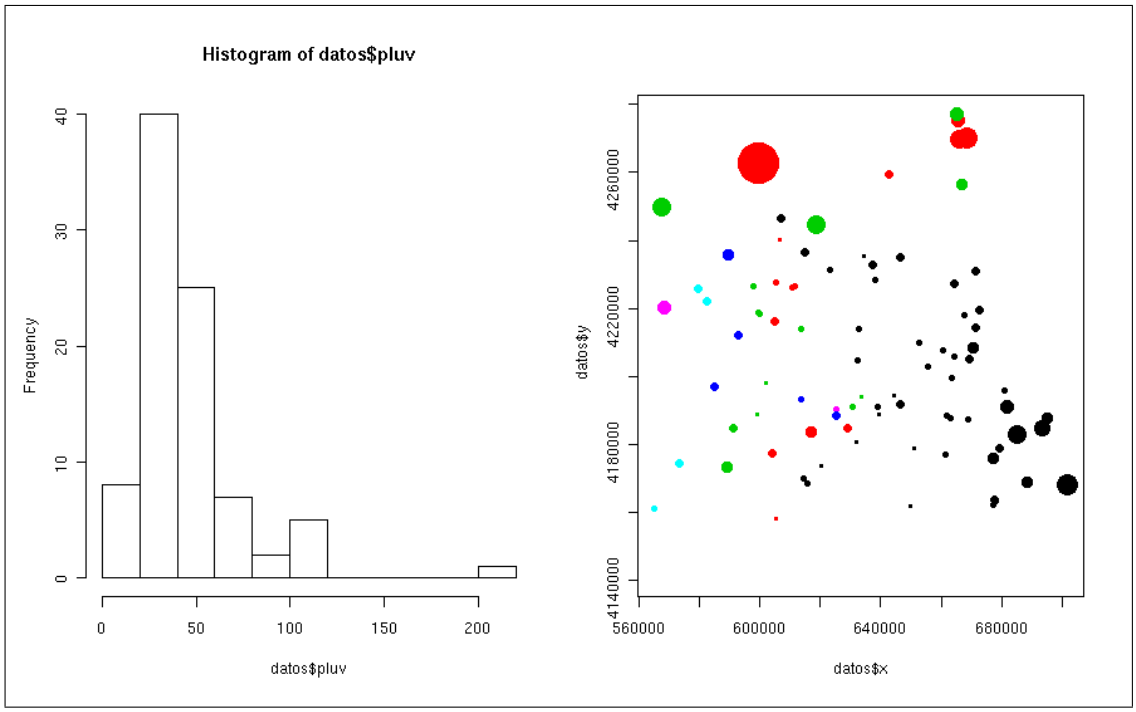

Figura 9: Distribución de observatorios, el diámetro de los círculos es proporcional a la precipitación y el color representa clases de altitud

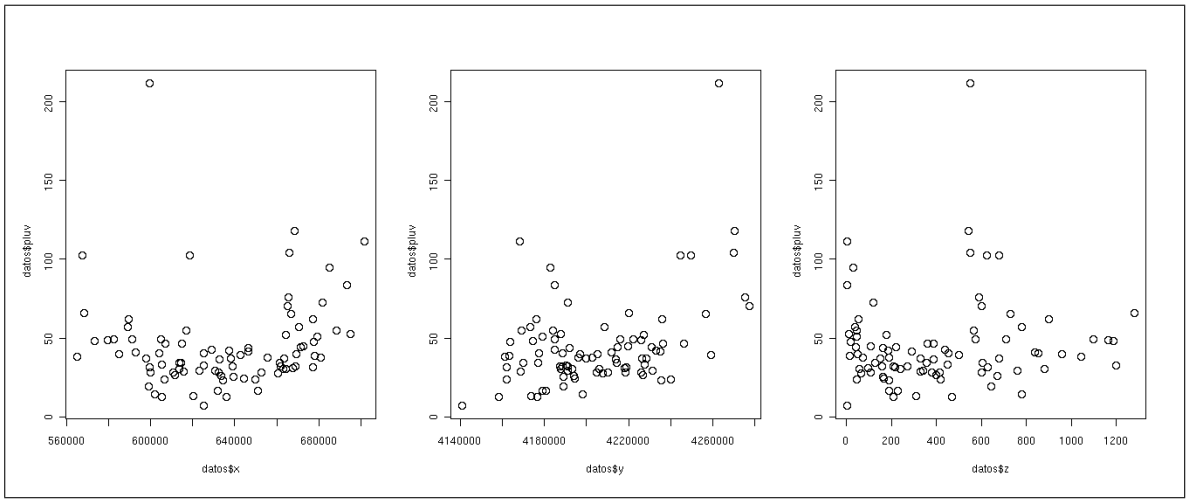

Figura 10: Relación entre las coordenads (X,Y,Z) y la precipitación

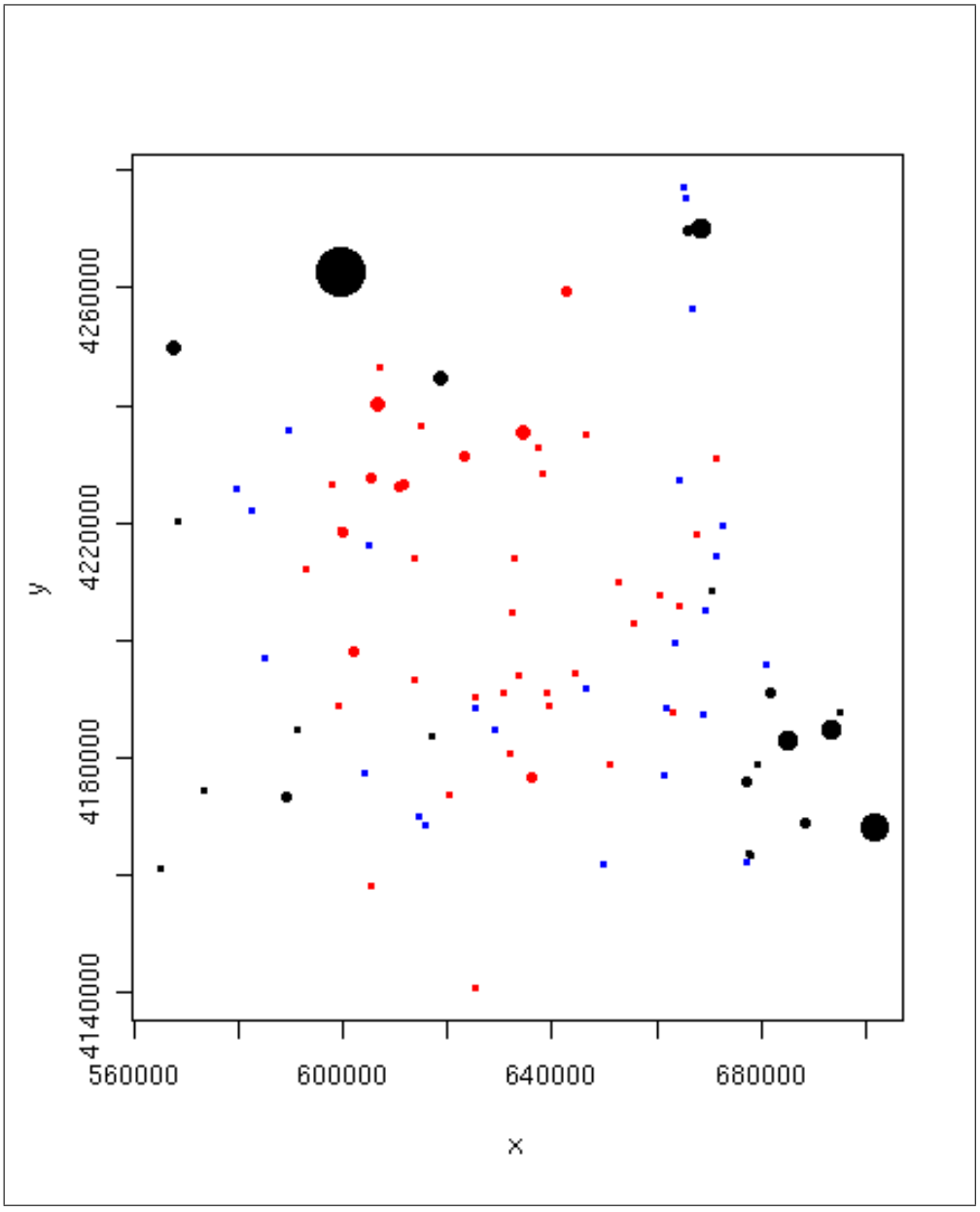

Figura 11: Residuales

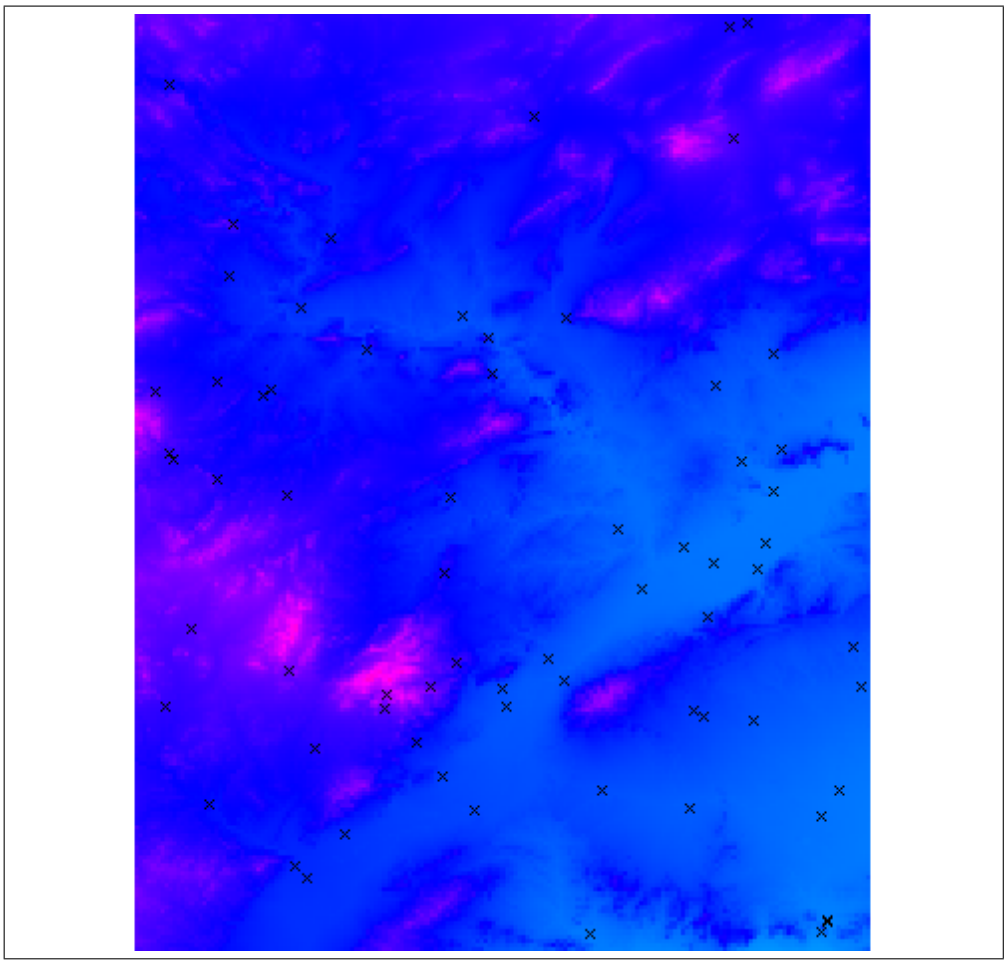

Figura 12: Modelo de Elevaciones y puntos

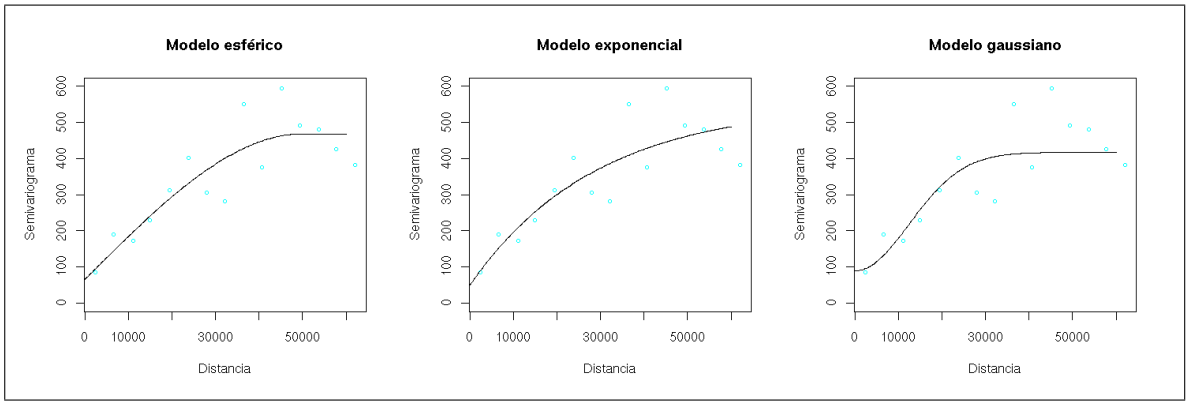

Figura 13: Modelos de semivariograma teórico

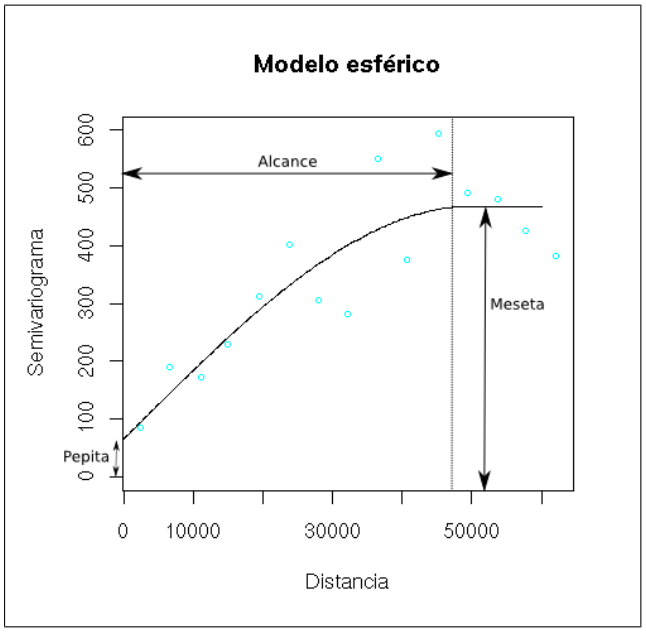

Figura 14: Parámetros del semivariograma

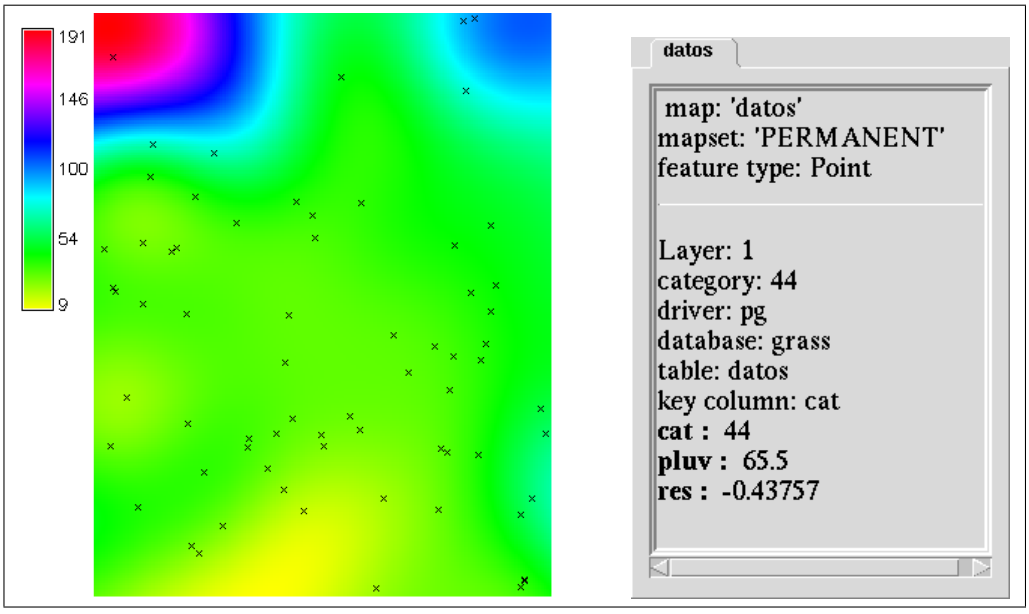

Figura 15: Visualización y comprobación