Tutorial de introducción a GRASS

Francisco Alonso Sarría

Presentación

Desde el año 1999, me he encargado de la docencia de diversas asignaturas relacionadas con Sistemas de Información Geográfica en las licenciaturas de Geografía y Ciencias Ambientales en la Universidad de Murcia. Salvo algunos intentos con Idrisi, debido a la imposibilidad de contar con ordenadores con Unix, he utilizado GRASS (Geographic Ressources Analysis Support System) en casi todos los casos.

Siempre he considerado que la enseñanza teórica de como funciona un *Sistema de Información Geográfica* y de la *Ciencia de la Información Geográfica* que tiene detrás debe ser, en la medida de lo posible, independiente del programa que se maneja en las prácticas. De esta manera se le facilita al alumno la posibilidad de migrar a otro sistema cuando empiece su carrera investigadora o profesional al evitar que las particularidades del software utilizado puedan confundirse con elementos básicos en el trabajo con un SIG.

La dificultad que supone este planteamiento es que, en cualquier caso, hay que elegir un programa para hacer prácticas y, aunque se separe la enseñanza teórica de los detalles del manejo de este, el alumno deberá tener una referencia del mismo.

Por ello parecía útil escribir un tutorial de GRASS que los alumnos pudieran utilizar desde el comienzo de la asignatura como material de lectura previo a las prácticas y de referencia durante el desarrollo de las mismas.

Así pues lo que tienes entre las manos es una primera versión de un tutorial de GRASS, por el momento incompleta y centrada en las características del programa necesarias para el desarrollo de las clases prácticas en las asignaturas que imparto. En un futuro se completará y se procurarán subsanar los errores y fragmentos poco claros. Para ello cuento con tu ayuda, envía tus comentarios a alonsarp@um.es

El área de estudio

En la figura 1 aparece la porción del mapa del Instituto Geográfico Nacional a escala 1:200000 correspondiente al área de trabajo.

como puedes ver se trata de una zona de la Región de Murcia que incluye el valle del Guadalentín entre Lorca y Alcantarilla incluyendo el Parque Natural de Sierra Espuña y parte del Parque Natural de Carrascoy.

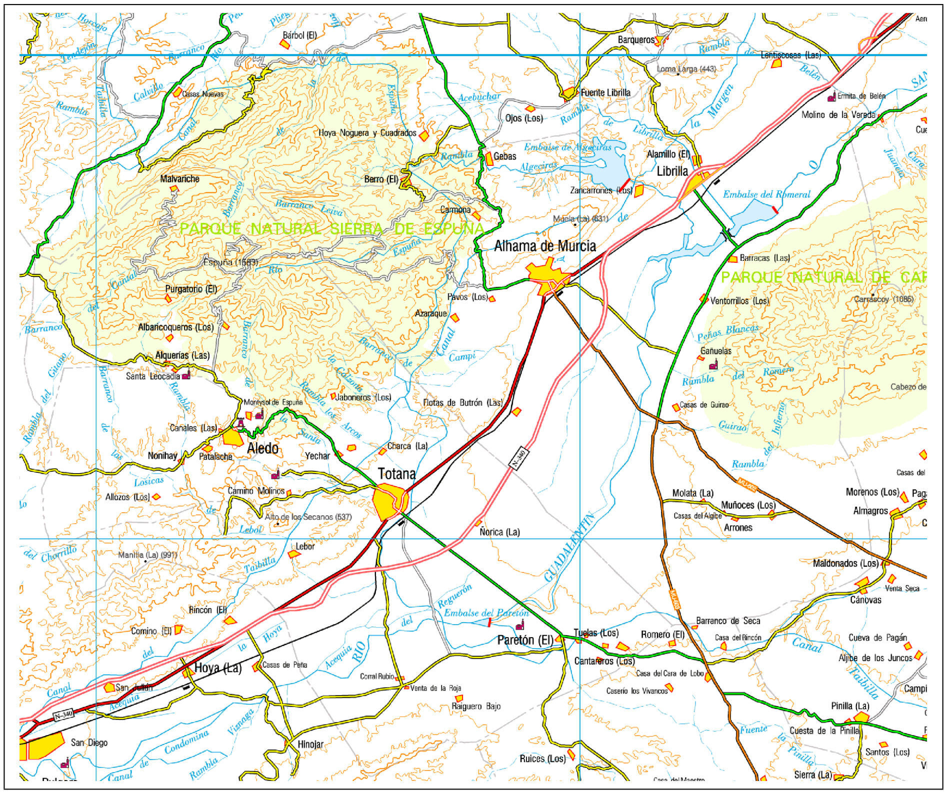


Figura 1: El área de estudio

Las capas de información que incluye son:

- **ign** El mapa que acabas de ver escaneado e incorporado como capa raster
- **mdecm**. Capa raster de elevaciones en centímetros
- **suelos**. Capa raster de tipos de suelo
- **usos**. Capa raster de usos del suelo
- **urbanos**. Capa raster con los principales núcleos urbanos
- **acuiferos**. Capa vectorial con los acuíferos de la Región (polígonos)
- **carreteras**. Capa vectorial con la red de carreteras de la Región de Murcia (lineas)
- **municipios**. Capa vectorial con los límites municipales (polígonos)
- **red_segura**. Capa vectorial con la red de drenaje (lineas)

- **urbanos**. Capa vectorial con los límites de los principales núcleos urbanos (polígonos)
- **carrascoy**. Capa vectorial con las curvas de nivel de Carrascoy (lineas)

Como puedes ver **urbanos** esta repetido como raster y como vectorial.

Además se utilizarán durante el curso 2 bases de datos gestionadas con PostgreSQL

- **suelos** Base de datos que acompaña al mapa de suelos
- **murcia** Base de datos con diversas variables socioeconómicas a nivel municipal
- **clima** Base de datos con información climática

Convenciones tipográficas

GRASS es, como casi todos los SIG, un programa modular. Esto significa que en realidad esta compuesto por múltiples programas pequeños (módulos) que llevan a cabo tareas muy concretas de forma altamente eficiente[1].. Una de las constantes a lo largo del desarrollo de los SIG ha sido tratar de facilitar al usuario la interacción con estos módulos mediante el desarrollo de interfaces, gráficas o no, sencillas e intuitivas. En el caso de GRASS, aparte de algunos intentos de desarrollar interfaces gráficas (tcltkgrass por ejemplo) el planteamiento ha sido dar a los módulos nombres que resulten intuitivos y los clasifiquen según su funcionalidad.

En este tutorial se va a distinguir entre los conceptos de *módulo* (programa que forma parte del SIG GRASS) y *orden* (conjunto de instrucciones que incluyen un módulo y las opciones y parámetros que se le pasan al módulo).

A lo largo del manual se utilizarán las siguiente convenciones tipográficas:

- Cuando se presente una orden se utilizará un tipo de letra ligeramente más grande
- Los conceptos importantes se escriben en *cursiva*, tal como has visto anteriormente con los conceptos de *módulo* y *orden*
- Los nombres de los módulos se escriben en **negrita**
- Los nombres de los parámetros y opciones, pero no sus valores, y en general cualquier elemento de GRASS se escriben también en **negrita**
- Los valores de los parámetros que debe introducir el usuario se escriben en letra *inclinada*[2] cuando hagan referencia a un tipo de dato genérico y en letra normal cuando expresen el dato concreto que hay que introducir

[1] En realidad algunos módulos de GRASS son altamente potentes y constituyen programas en si mismos

[2] Date cuenta de que la letra inclinada no es igual que la letra cursiva

- Cuando se haga referencia a una tecla concreta del teclado del ordenador se escribirá en mayúsculas y entre ángulos, por ejemplo <ENTER>.
- Cuando se presentan ejemplos de los comandos, el texto aparece indentado y antes de la orden se escribe un *prompt* semejante al prompt de GRASS, por ejemplo:

  GRASS: /path > **d.vect** *nombre_mapa* **color=***color* <ENTER>

implica que **d.vect** y **color=** deben teclearse tal cual ya que son el nombre del comando y el nombre de un parámetro, mientras que *nombre_mapa* y *color* son tipos genéricos de datos que deben sustituirse por sus valores correspondientes. Por otro lado:

GRASS: /path > **d.vector** termuni **color=**green <ENTER>

implica que tanto **d.vect** y **color=** como "termuni" y "green" deben teclearse tal cual ya que estos últimos son el mapa (límites de términos municipales) que hay que representar y el color con que se quiere representar.

A lo largo del texto se insertan contenidos de tipo avanzado que se caracterizan por el encabezado **Temas avanzados** y por un tipo de letra más pequeño. El objetivo es introducir algunas cuestiones complejas en el lugar donde deben aparecer pero dejando claro al lector que, en una primera lectura, puede saltárselos para lograr de forma más sencilla una visión de conjunto del capítulo. Cuando el alumno vuelva a repasar el capítulo, debería leerlos para así conseguir una visión más completa del tema tratado.

Con el encabezado **Truco**, letra más pequeña y menor anchura de linea, aparecen algunos consejos para facilitar el trabajo con GRASS.

Finalmente, los mensajes que el programa presenta en el monitor de texto se presentan en un cuadro centrado:

Mensaje del programa: Seguir (s/n)

Los menús de ratón se muestran en el texto como un cuadro ladeado a la izquierda:

Menú de ratón: Seguir (s/n)

estos menús se utilizan, en algunos módulos de GRASS, para permitir que el usuario interaccione con el monitor gráfico donde se muestran los mapas para hacer zoom, consultar determinados valores asociados a las entidades representadas, etc.

wxGRASS

De forma experimental se va a trabajar con un módulo desarrollado para GRASS que genera una Interfaz Gráfica de Usuario más amigable que el sistema de comandos de GRASS.

Este es, sin embargo, un tutorial de GRASS que se ha completado con referencias a wxGRASS, este se describe de manera general en el capítulo 2 mientras que en los siguientes capítulos aparecerá, al final, una sección dedicada a como afrontar con wxGRASS las tareas planteadas en el capítulo.

0.1. GRASS 6.0

Recientemente ha aparecido una nueva versión del programa que incorpora importantes novedades en cuanto al manejo de datos vectoriales. Si quieres profundizar en el puedes descargar un pequeño documento en castellano sobre el mismo en http://www.um.es/geograf/sigmur.

Capítulo 1

El sistema operativo Unix-Linux

GRASS es un programa desarrollado para trabajar en entornos Unix, recientemente se han lanzado versiones que pueden ejecutarse en Windows utilizando un programa, Cygwin, que emula un sistema operativo Unix dentro de Windows. Sin embargo es preferible utilizar la versión de Unix por varias razones:

- Un emulador de Unix no será nunca tan potente como el Unix real.
- La complejidad de Unix más la complejidad de Cygwin es, posiblemente, demasiado para un ordenador convencional cargado con Windows
- Si se trabaja con programas de código abierto[1] (la gran mayoría de los disponibles para Linux) la facilidad de comunicación entre programas se incrementa enormemente mejorando la capacidad de un SIG basado en GRASS.

1.1. Que son Unix y Linux

Unix fue uno de los primeros sistemas operativos en desarrollarse, y lo hizo de forma modular, es decir pequeños programas, módulos, ejecutaban tareas concretas y sencillas de forma correcta y eficiente. La interrelación de todos estos módulos en ejecución (*procesos*) es lo que da a Unix su potencia.

Unix fue en principio software libre hasta que sus derechos fueron adquiridos por una empresa y pasó a comercializarse. Debido a su elevado precio, se desarrollaron clones de Unix para estudiantes como Minix y el más famoso: Linux.

Linux empezó casi como un entretenimiento de un estudiante finlandés que compartió su código a través de Internet. El elevado interés que causó provocó una avalancha de colaboradores que ha conducido al

[1] Código abierto significa que las instrucciones del programa son públicas y cualquiera tiene acceso a ellas, de este modo si alguien quiere crear un programa que trabaje, por ejemplo, con datos de GRASS, le resultará más sencillo ya que sabrá exactamente como trabaja GRASS

Linux[2] actual, una alternativa libre, gratuita, segura y fiable a Unix y a Windows que cuenta con el pleno apoyo de gobiernos y empresas como IBM o HP.

Linux no es Unix sino un clon, es decir algo que funciona igual pero que se ha desarrollado desde cero sin mirar el código del programa clonado[3]. Por tanto cualquier manual de Unix vale para Linux y viceversa (al menos manuales de usuario).

1.2. Linux. Herramientas del sistema

Evidentemente este no es el lugar para desarrollar un manual del usuario de Linux, tienes a montones en librerías y en Internet, sin embargo durante este curso utilizaremos un conjunto de módulos y programas del sistema operativo que hay que conocer.

El hecho de que tanto Linux como GRASS como muchos de los programas desarrollados para Linux sean libres implica que el código y estructuras de datos de unos son conocidos por los desarrolladores de los otros. De esta manera se simplifica enormemente la programación de herramientas de intercambio de datos. Esta idea se expresa metafóricamente en la presentación de GRASS con la frase: *GRASS es parte de un jardín*. Por tanto resulta útil conocer algunas de estas herramientas y programas que se utilizarán a lo largo del curso.

ls

Un disco, en cualquier sistema operativo, se organiza mediante directorios. Estos son compartimentos lógicos que permiten guardar los ficheros de forma organizada. Cada directorio puede contener otros directorios y así sucesivamente de manera que la estructura lógica de un disco duro se asemeja a un árbol.

El comando **ls** produce un listado de los archivos y subdirectorios presentes en un directorio. Podemos modificar su actuación mediante diversas opciones:

GRASS: /path > **ls** mapa* <ENTER>

listará sólo los ficheros que comiencen por mapa

GRASS: /path > **ls** *.txt <ENTER>

listará sólo los ficheros que terminen con .txt

[2] Realmente Linux es sólo el núcleo del sistema, el resto de las herramientas proceden del proyecto GNU desarrollado por la *Free Software Foundation* por tanto en realidad el sistema debiera llamarse GNU-Linux

[3] Recientemente la empresa SCO, propietaria del código de Unix, ha acusado a IBM de pasar a los desarrolladores de Linux fragmentos de su código. Por otra parte la empresa Novell reclama ser ella, y no SCO, la legítima propietaria de los derechos de Unix. En fin un auténtico culebrón

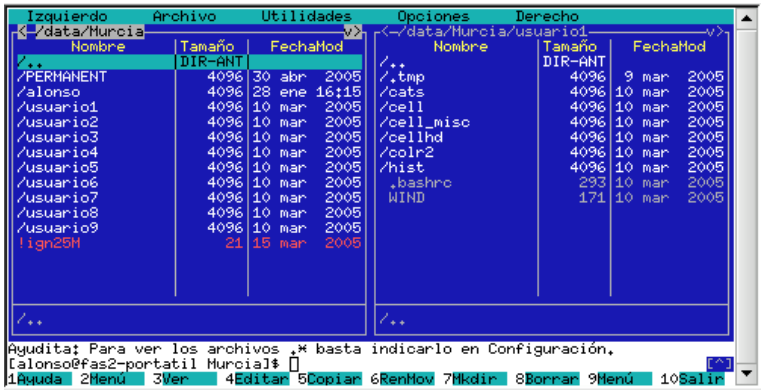


Figura 1.1: Pantalla del programa **mc**

GRASS: /path > **ls** -l <ENTER>

hará un listado incluyendo todos los detalles de cada uno de los archivos

cd

Permite cambiar de directorio

GRASS: /path > **cd** usuario <ENTER>

cambia al subdirectorio usuario

GRASS: /path > **cd** .. <ENTER>

cambia al subdirectorio superior

mc

Se trata de un programa de gestión de archivos y directorios similar al explorador de windows aunque más espartano y eficiente. Al ejecutar el programa se abren, en el terminal de texto desde el que se ha ejecutado, dos ventanas con sendas vistas del sistema de archivos.

Si pinchamos en un fichero o subdirectorio lo seleccionamos y si pinchamos dos veces, o pulsamos <Return>, sobre un directorio seleccionado entraremos en el. Si hacemos lo propio con el subdirectorio **/..** iremos al directorio de nivel superior.

Muchas de las acciones de **mc** se ejecutan mediante teclas:

- La tecla <Ins> permite marcar ficheros o directorios
- La tecla <F3> permite ver el archivo sobre el que está el cursor
- La tecla <F4> permite editar el archivo sobre el que está el cursor
- La tecla <F5> copia los archivos marcados a la otra ventana
- La tecla <F6> mueve los archivos marcados a la otra ventana
- La tecla <F7> crea un nuevo directorio
- La tecla <F8> borra los archivos marcados

Para terminar, ten en cuenta que no es lo mismo marcar (tecla <Ins>) que seleccionar (pinchar con el ratón). Sólo puede haber un fichero o directorio seleccionado pero puede haber varios marcados.

kedit

Es un editor de texto que nos servirá para crear pequeños archivos de texto con los que introducir información a algunos de los módulos de GRASS. Un editor de texto es un pequeño programa (similar al bloc de notas de windows) que sirve para escribir y guardar texto sin ningún tipo de adorno (colores, tipos, fuentes, etc.) esta es la diferencia fundamental con los procesadores de texto.

La manera más simple de ejecutarlo es

GRASS: /path > **kedit** *nombre_del_fichero* <ENTER>

Aparecerá la pantalla que ves en la figura 1.2, en la que podemos empezar a escribir. En la parte alta de la pantalla aparece un menú, en el menú **Archivo** aparecen las habituales opciones de manejo de ficheros búsqueda o sustitución de caracteres así como las opciones para salir o cerrar archivo; en el menú **Editar** aparecen las opciones de cortar, copiar y pegar texto, así como las de búsqueda y sustitución de cadenas de caracteres. Otros menús permiten hacer corrección ortográfica, configurar el programa y obtener ayuda.

En Unix se dispone de un gran número de editores además de kedit como pueden ser **gedit**, **emacs** o **vi**. El primero de ellos tiene, al igual que kedit, una interfaz de usuario basada en menús y botones, mientras que emacs y sobre todo vi son más sencillos y se basan en la utilización de diferentes combinaciones de teclado para dar las órdenes al programa.

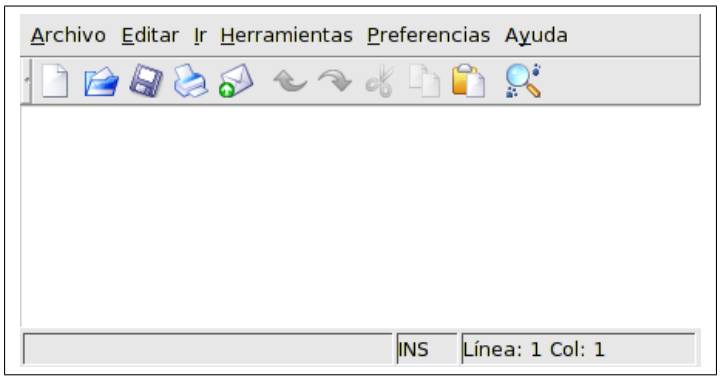


Figura 1.2: Ventana de kedit

Tuberías y redirecciones

La gran potencia de Unix no está sólo en sus comandos sino en la capacidad de interrelacionar unos comandos con otros. Para ello se cuenta, entre otros, con dos mecanismos básicos las tuberías y las redirecciones.

Una **tubería** permite enlazar la salida de un comando como entrada del siguiente. Se escribe como una barra vertical "|" entre ambos comandos.

Como ejemplo vamos a pasar el resultado de la orden **ls -l** que ya hemos visto que produce un listado de los ficheros al comando **sort** que ordena la información de entrada en función de diversos criterios (normalmente **ls** ordena los ficheros alfabéticamente por sus nombres):

ls -l|sort -k 5 **-n**

Con esta orden se ordena la salida de ls -l en función del tamaño del fichero (la opción **-k** 5 hace referencia a la quinta columna de la tabla) y la opción **-n** indica que el criterio de ordenación es numérico no alfabético.

Un ejemplo aún más interesante es el siguiente:

ls -l|sort -k 6 **-M**

que ordena el listado en función de la sexta columna (el mes) desde Enero a Diciembre (la opción **-M** indica que la columna que se va a utilizar para ordenar contiene meses).

La **redirección** dirige la salida de un comando a un fichero o sitúa el contenido de un fichero como entrada de un comando. Por ejemplo:

ls -l>listado.txt

crea un nuevo fichero llamado **listado.txt.** que contiene el listado ordenado. Ahora vamos a ordenar el contenido de este fichero en función del tamaño:

sort -k 5 <listado.txt

Evidentemente nada impide combinar tuberías y redirecciones:

ls -l|sort -k 5 **-n** >listado.txt

1.3. Conexiones remotas

Tanto Unix como Linux son sistemas operativos multiusuario, es decir permiten el trabajo de un elevado número de usuarios al mismo tiempo. Para ello es necesario disponer de una red de ordenadores en el que uno de ellos actúe como servidor Unix y los usuarios se conecten con el a través de terminales que pueden funcionar con Unix o con Windows. Téngase en cuenta además que los casos que se exponen a continuación son los habituales en la Universidad de Murcia, pueden ser diferentes en otros entornos de trabajo.

Si eres un alumno de una asignatura de SIG en la Universidad de Murcia, el nombre de usuario, número de sesión y contraseñas se te proporcionaran al comienzo de curso.

Terminales Unix

En el caso de que las conexiones se hagan desde terminales Unix, la orden de conexión puede ser una de estas dos:

GRASS: /path > **Xwrapper -query** *servidor.dom.um.es* <ENTER>

GRASS: /path > **X -query** *servidor.dom.um.es* <ENTER>

donde *servidor.dom.um.es* hace referencia al nombre o número IP del servidor.

A continuación recibiremos la pantalla de entrada al servidor que nos pedirá nuestro nombre de usuario en dicho ordenador y nuestra contraseña, ten en cuenta que la contraseña no se verá mientras la tecleas así que tendrás que escribirla correctamente a ciegas. Si introducimos nuestros datos correctamente, aparecerá nuestro escritorio en el servidor.

Terminales Windows

En el caso de conexiones desde Windows la posibilidad más sencilla es utilizar el programa **vncviewer**,estro nombre previamente deberá haberse abierto una sesión en el servidor Unix con **vncservr**. Al ejecutar **vncviewer** este nos preguntará por la IP o el nombre del servidor al que hay que realizar la conexión. Debemos pasársela seguido del número de sesión separado por dos puntos, generalmente cada usuario tiene una sesión propia. Por ejemplo si vamos a conectarnos a la sesión 2 en el ordenador 155.54.0.0 deberemos escribir:

```
155.54.0.0:2
```

el programa nos pedirá la clave de la sesión y si es correcta conectará con el servidor y podremos empezar a trabajar, ten en cuenta que la contraseña no se verá mientras la tecleas así que tendrás que escribirla correctamente a ciegas.

Capítulo 2

Dando un primer vistazo

2.1. Objetivos

- Entrar y salir de GRASS
- Entrar y salir de wxGRASS
- Entender la nomenclatura y el modo de funcionamiento básico de los módulos de GRASS
- Analizar los diferentes elementos que forman parte de una base de datos de GRASS
- Aprender el manejo de los módulos de GRASS incluidos en la tabla 2.1

g.manual	Obtiene las páginas de manual de los módulos
g.list	Produce un listado de las diferentes capas de información espacial presentes en la base de datos
d.mon	Activa un monitor gráfico
d.rast	Visualiza una capa raster
d.vect	Visualiza una capa vectorial
exit	Para salir del programa

Cuadro 2.1: Módulos que van a utilizarse en esta práctica

2.2. Entrada y salida rápida al programa

Dependiendo de la distribución de GRASS que estemos manejando y del modo en que esté instalado, el comando para ejecutar GRASS puede variar (grass, grass4.1, grass4.2, grass5, etc). Para confirmar cual es la orden de entrada basta con escribir:

grass <TAB>

ya que Unix, si comenzamos a escribir una orden y pulsamos el tabulador, la completa o, en caso de que existan varias posibilidades, nos las presenta en pantalla.

Una vez inicializado GRASS, en la primera pantalla que aparece debe indicarse la base de datos con la que se va a trabajar y los directorios de trabajo dentro de la misma (**LOCATION** y **MAPSET**). La figura 9.2 nos muestra un ejemplo de este tipo de pantalla, en realidad la apariencia de esta depende de la versión de GRASS con que se este trabajando y de la configuración específica de GRASS para cada usuario. Por ejemplo, la versión 6.0 del programa dispone de una pantalla gráfica para la entrada en el programa (figura 2.2).

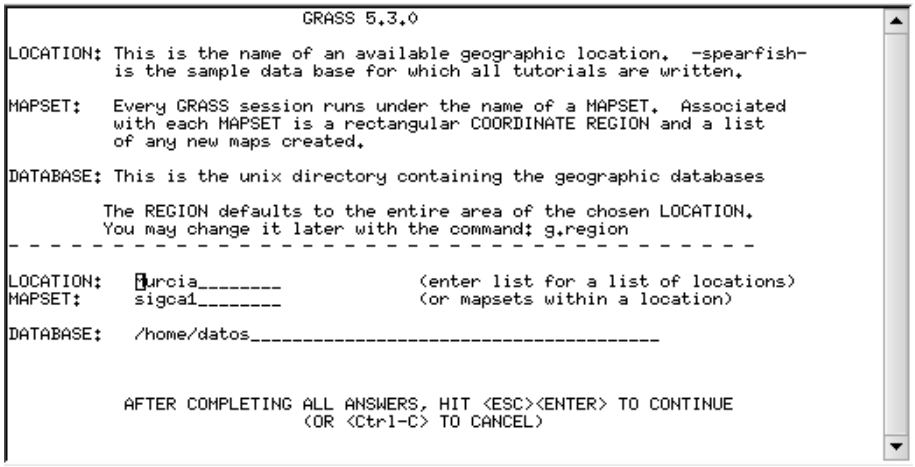


Figura 2.1: Pantalla de introducción a GRASS 5.3.0

Temas avanzados: El sentido de las LOCATION y los MAPSET

La primera dificultad con que se encuentra alguien que comienza a experimentar con GRASS es el sentido de estos conceptos. En realidad se trata de decirle al programa algo tan sencillo como en que directorios están los datos con los que queremos trabajar.

GRASS es un entorno multiusuario en el que se asume que varias personas van a trabajar en varios proyectos al mismo tiempo accediendo al mismo tiempo a un ordenador que actúa como *servidor* almacenando datos y programas. Los usuarios acceden a él a través de *terminales* conectadas a una red local o a Internet. La necesidad de evitar que varios usuarios manipulen a la vez un mismo mapa (con el consiguiente riesgo de pérdida de información), es la razón por la que los datos se almacenan en una estructura de directorios que puede parecer a primera vista algo compleja.

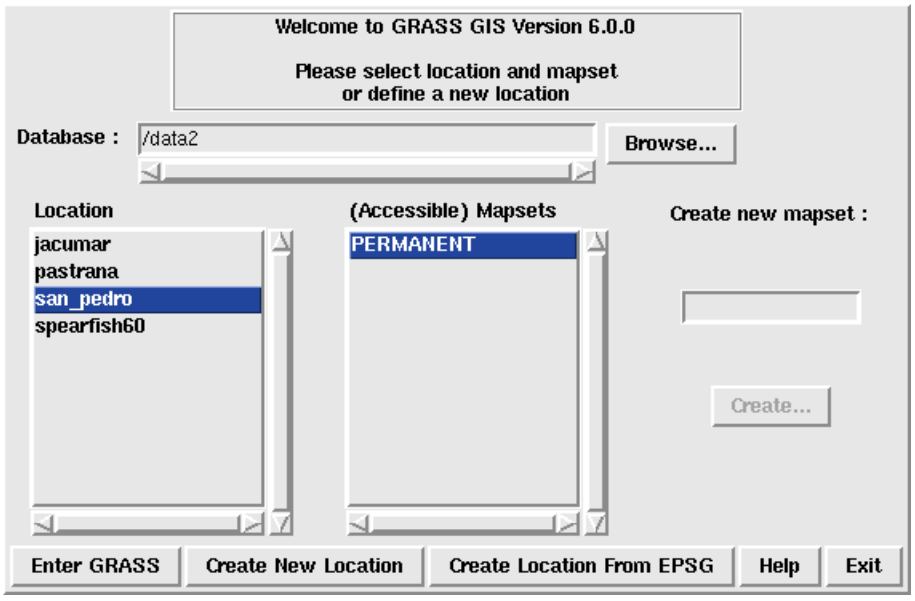


Figura 2.2: Pantalla de introducción a GRASS 6.0

Existe un directorio principal llamado **DATABASE** que, en principio, contiene todas las bases de datos de todos los proyectos gestionados por GRASS.

De esta **DATABASE** cuelgan varias **LOCATION**, cada una de ellas es un directorio que corresponde a un proyecto de trabajo. Cada **LOCATION** utiliza un único sistema de proyección para sus mapas y, en principio, todos los mapas que contiene son, más o menos, de la misma zona.

Puesto que cada **LOCATION** corresponde a un proyecto en el que trabajan varias personas, cada una de ellas tiene un directorio propio llamado **MAPSET** (conjunto o carpeta de mapas en inglés), que contiene todos los mapas que esa persona a hecho y que, por tanto, puede modificar.

Cada **LOCATION** tiene un **MAPSET** llamado **PERMANENT** que contiene todos aquellos mapas que se consideran acabados, que no van a sufrir posteriores modificaciones y que, por tanto, ningún usuario podrá modificar aunque todos pueden leer. Por tanto un usuario puede leer mapas de su **MAPSET** y de **PERMANENT** pero solo puede escribirlos en su **MAPSET**. Si se conceden los permisos oportunos, el usuario podrá además leer, pero no modificar, mapas de todos los demás **MAPSETS**. En definitiva la filosofía de GRASS es que cualquier usuario puede consultar todos los mapas pero sólo puede alterar los suyos.

En realidad los **MAPSET** no tienen porque corresponder con usuarios físicos, también pueden existir mapsets por áreas temáticas (geología, hidrología, etc.) o por tareas (interpolación, modelos, etc.), siempre con la limitación de que cada **MAPSET** tiene un único propietario y que cada usuario del sistema sólo puede acceder a GRASS mediante un **MAPSET** del que sea propietario.

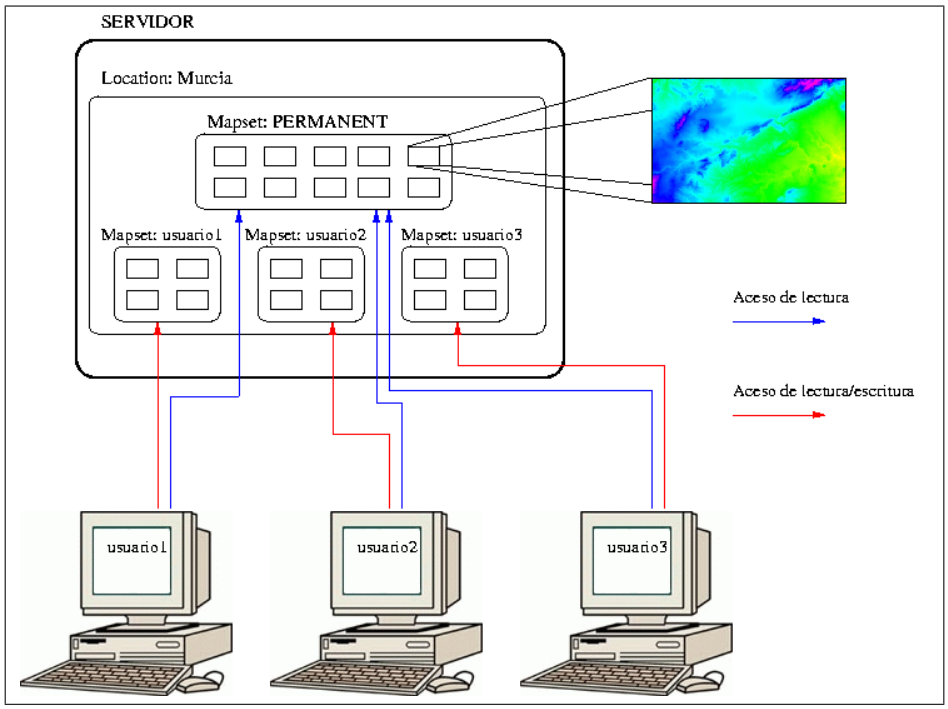


Figura 2.3: Conexiones remotas multiusuario a un Servidor en que se aloja una Location con varios mapset

Para la realización de este tutorial, las entradas adecuadas son:

LOCATION: murcia
MAPSET: *nombre_usuario*
DATABASE: /home/grassdata

Si todo ha salido bien el único cambio apreciable es la sustitución del *promt* anterior por uno nuevo:

GRASS: /path >

que indica que se ha entrado en el programa.

Para salir de GRASS debe teclearse

GRASS: /path > **exit** <ENTER>

Es conveniente salir con exit ya que en otro caso la estructura de datos podría resultar dañada.

2.3. *Modus operandi*: Monitores y linea de comandos

Uno de los criterios que pueden utilizarse para clasificar diferentes programas de gestión de información geográfica, es el predominio de los botones o el predominio de la linea de comandos. GRASS es un programa basado eminentemente en la linea de comandos. Para la visualización de datos permite "abrir" monitores que pueden utilizarse para visualizar información gráfica a lo largo del desarrollo de una sesión con GRASS. Los módulos de consulta sobre el mapa o zoom también requieren interactuar con el mapa mostrado en el monitor. La última distribución de GRASS, incluye un módulo de visualización 3D basado en una interfaz gráfica de usuario con ventanas, cuadros de dialogo, etc. basado en la librería Mesa, una de las librerías gráficas más potentes desarrolladas para entornos Unix.

El hecho de ser un sistema basado en la linea de comandos, permite una integración natural de GRASS con todas las facilidades de proceso de datos de Unix.

La orden de GRASS para abrir un monitor gráfico es **d.mon**, sus parámetros básicos son **start**, **select** y **stop**. De momento baste con saber que para abrir un monitor gráfico para comenzar a ver mapas hay que ejecutar:

GRASS: /path > **d.mon** *x1* <ENTER>

Temas avanzados: Los monitores en GRASS

El sistema de pantallas gráficas de GRASS suele ser el siguiente obstáculo. En la mayoría de los programas que el usuario está acostumbrado a utilizar las ventanas son algo obvio por lo que no hay que preocuparse.

En GRASS existe un número limitado de monitores gráficos (en realidad podría haber muchísimos pero no merece la pena) y el usuario debe especificar con cual de ellos quiere trabajar. Los monitores gráficos se nombran con un x seguida de un número (x1, x2, x3, x4, x5, x6, . . .) por lo que si queremos activar el primero de ellos la orden será:

GRASS: /path > **d.mon start=**x1 <ENTER>

Si sólo tenemos un monitor activo este será automáticamente seleccionado. Para comprobarlo puedes ejecutar la orden:

GRASS: /path > **d.rast** mdecm <ENTER>

y ya de paso veras el mapa de elevaciones, la primera de las capas de información espacial que componen este tutorial. Sin embargo cuando haya más de un monitor abierto será necesario que el usuario le comunique al sistema a cual de ellos debe mandar el resultado de los subsiguientes comandos gráficos. Para ello la orden adecuada es:

GRASS: /path > **d.mon select=**x1 <ENTER>

Finalmente cuando el usuario quiera cerrar el monitor bastará con hacer:

GRASS: /path > **d.mon stop=**x1 <ENTER>

En GRASS, salvo excepciones, no existen iconos, cuadros de diálogo, etc. Todas las ordenes se introducen por linea de comandos. Si bien al principio puede parecer un poco pesado, especialmente a usuarios acostumbrados a trabajar en windows, tiene varias ventajas como la posibilidad de combinar módulos de GRASS con comandos de Unix para automatizar tareas y construir programas más potentes.

2.4. Reconocimiento de la base de datos y la lista de comandos

Una base de datos de GRASS contiene diversos tipos de capas de información espacial almacenados en diversos directorios. Por otro lado GRASS, al ser un SIG modular dispone de una multitud de módulos. Para evitar que el usuario se pierda debido al elevado número de mapas y de módulos se establecen ciertas convenciones.

2.4.1. Identificando los distintos módulos

El trabajar con comandos tecleados en lugar de menús o botones puede recordar al viejo MS-DOS[1]. Sin embargo Unix incorpora una linea de comandos más potente que facilita el trabajo. En primer lugar, si tras teclear algunas letras presionamos la tecla de tabulación, el sistema, tal como se ha visto anteriormente, listará todos los comandos cuyo nombre empieza con esa particular combinación de caracteres.

Los nombres de los módulos de GRASS están constituidos por una letra minúscula que hace referencia al tipo del comando, un punto y el nombre del comando. Los tipos de comando, y sus letras identificadoras son:

- **g.*** -Módulos de gestión de ficheros
- **d.*** -Módulos de salida gráfica
- **r.*** -Módulos para análisis de capas raster
- **v.*** -Módulos para análisis de capas vectoriales
- **i.*** -Módulos para tratamiento de imágenes de satélite
- **s.*** -Módulos de procesamiento de *sites*, los sites son mapas de pùntos, en GRASS 6.0 desaparecen y pasan a formar parte de las capas vectoriales

[1] Aunque en realidad Unix es anterior a MSDOS, este fue sólo una burda imitación

- **m.*** -Miscelanea de módulos con propósitos diversos
- **p.*** y **ps.*** Creación de cartografía y creación de cartografía postscript

Esta particular notación evita la confusión de los módulos de GRASS con otros comandos Unix y además permite echar un vistazo rápido a todos los módulos de GRASS, si tecleamos por ejemplo **r.** y seguidamente pulsamos la tecla de tabulación, obtendremos un listado de todos los módulos relacionados con el tratamiento de ficheros raster.

En algunos casos aparece una segunda palabra que nos da algo más de información sobre el propósito del programa, como ejemplos:

- **in**, son módulos de importación de otros formatos a GRASS
- **out**, son módulos de exportación de GRASS a otros formatos,
- **surf**, interpola superficies a partir de mapas de puntos o lineas
- **what**, permite pinchar en un monitor gráfico y obtener las características de la celdilla, objeto o punto seleccionado
- **to**, realiza diversas operaciones de paso de formato entre raster, vectorial y sites

Existen diferentes combinaciones de estas palabras con las iniciales antes vistas, por ejemplo **r.in**, **v.in**, **s.in**, **r.out**, **v.out**, **s.out**, **r.out**, **v.out** o **s.out**, **d.what.rast**, **d.what.vect**, **d.what.sites**.

El orden en el que se han presentado anteriormente los diferentes tipos de comandos de GRASS implica también el orden en que se van a ir utilizando en el proceso de aprendizaje del programa así como lo habitualmente que se utilizan trabajando con el. De hecho los módulos **p.*** y **ps.*** apenas se utilizan ya que es más conveniente y sencillo utilizar el **driver PNG** para la generación de cartografía.

2.4.2. Manuales y ayuda

Cada módulo de GRASS es casi un programa independiente y como tal cuenta con un manual propio. El módulo con el que podemos pedir al programa que nos muestre el manual de cualquier módulo es **g.manual**. Puede invocarse seguido del nombre del módulo del que queremos ayuda:

GRASS: /path > **g.manual** d.rast <ENTER>

con lo que se obtiene directamente en el monitor de texto el manual del módulo **d.rast**.

Truco:

Manuales en el navegador y en castellano
La salida por defecto de **g.manual** es el monitor de texto en el que suele ser incomodo leer cualquier cosa (y más un manual en inglés). En realidad los manuales de los módulos están escritos también en HTML con lo que pueden leerse en un navegador. Para ello basta con ejecutar:

GRASS: /path > **export GRASS_TEXT_BROWSER=***navegador* <ENTER>

sustituyendo *navegador* por tu navegador preferido.
Si tu distribución de GRASS lo soporta puedes utilizar la opción -e de **g.manual** para disponer de las páginas de manual en castellano. Si no sabes como hacerlo conéctate a http:www.um.es/geograf/sig.

Puede también obtenerse una breve ayuda de un módulo utilizando el parámetro help del módulo correspondiente:

GRASS: /path > **d.rast** help <ENTER>

De este modo el módulo nos presentará un resumen del modo en que debemos invocarlo. En realidad esta información aparecerá cada vez que llamemos al módulo de forma errónea, junto con el mensaje de error correspondiente.

Por tanto, cada vez que ejecutes una orden de GRASS y el programa te conteste con un mensaje de ayuda, no lo dudes: **TE HAS EQUIVOCADO EN ALGO**. Busca al principio del mensaje de ayuda y verás el error (figura 2.4). En este ejemplo, la segunda linea nos dice que -k no es una opción válida y a continuación nos da la ayuda explicándonos cuales son las opciones y parámetros válidos.

2.4.3. Modo interactivo y linea de comandos

La mayor parte de los módulos pueden utilizarse de dos modos diferentes:

Modo interactivo El usuario simplemente teclea el nombre el módulo y espera a que el programa le pregunte por los valores de los parámetros.

GRASS: /path > **d.rast** <ENTER>

Modo en linea de comandos El usuario debe teclear, tras el nombre del módulo, todas las opciones y parámetros de ejecución del mismo.

GRASS: /path > **d.rast** mdecm <ENTER>

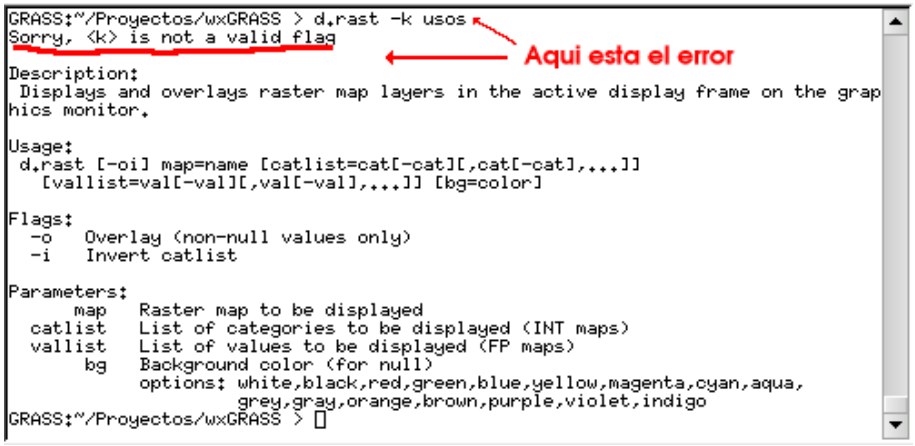


Figura 2.4: Mensaje de error del módulo d.rast

Aunque en principio el modo interactivo parece más sencillo hay que tener en cuenta que, si lo utilizamos, GRASS nos preguntará por todos[2] los parámetros (voluntarios y optativos) y opciones. De este modo la ejecución de un módulo sencillo como **d.rast** se hace realmente farragosa. Por tanto será preferible utilizar la linea de comandos salvo en las escasas ocasiones en las que un módulo solo admita la ejecución en modo interactivo. Por otro lado el modo en linea de comando resulta más útil cuando se va a repetir una misma orden con diversas variaciones o cuando se trabaja con ficheros de guión de Unix.

En el modo linea de comandos, las opciones constan de una sola letra precedida por un guión y los parámetros tienen un nombre. Por ejemplo en el siguiente caso:

GRASS: /path > **d.rast** -*o* **rast**= urbanos <ENTER>

aparece la opción *o* y el parámetro **rast**. La opción *o* indica que aquellas celdillas de la capa raster que tengan valor nulo no se pinten de manera que pueda seguir viéndose una capa raster que hubiera pintada anteriormente. El parámetro **rast** indica la capa raster que va a pintarse. En algunos casos, como el que nos ocupa ahora, es válido introducir el parámetro sin su nombre, por ejemplo **d.rast** *mdecm* tal como se ha visto anteriormente. En el siguiente capítulo se estudiarán más en detalle estos módulos y el tipo de salida gráfica que producen.

Temas avanzados: Interfaz de usuario de los módulos de GRASS

Los módulos de GRASS son programas independientes y, en muchos casos, programados por personas sin ningún de relación. Lo que da unidad al programa es que todos los módulos comparten:

[2] y en algunos casos pueden llegar a ser muchos

- El tipo de estructuras de datos (formatos de los ficheros) a los que acceden y que crean
- La interfaz y las comunicaciones con el usuario tienen un diseño común
- Un conjunto de librerías que incluyen funciones SIG en lenguaje C que simplifican la programación y aseguran que los módulos se comporten de forma consistente aunque hayan sido programados por equipos diferentes.

En el nivel de usuarios noveles en que nos encontramos, lo que nos interesa es conocer algo más de como se van a comunicar con nosotros los diversos módulos de GRASS.

La mayor parte de los módulos transforman información de entrada (capas raster, vectoriales o de puntos) en información de salida (nuevas capas, representaciones gráficas, estadísticas o informes), esta transformación viene condicionada por un conjunto de parámetros y opciones que deben ser introducidos por el usuario. Los parámetros pueden ser obligatorios, es decir aquellos sin cuya presencia el módulo no puede funcionar, u optativos (los introducimos si queremos y si no queremos no), las opciones son como su propio nombre indica siempre optativas.

Por ejemplo, el módulo **d.rast** pinta una capa raster en el monitor gráfico que esté seleccionado, su parámetro obligatorio es, lógicamente, la capa que queremos pintar. Pero **d.rast** tiene algunos parámetros y opciones más que puedes consultar tecleando **d.rast -help**.

2.5. La base de datos

Existen 3 tipos básicos de capas espaciales en GRASS:

rast , corresponden a capas de datos en formato raster. Es en el tratamiento de este tipo de capas donde GRASS se muestra más potente

vect , corresponden a capas de lineas o de polígonos en formato vectorial. El formato de almacenamiento y gestión de este tipo de datos se está modificando profundamente para que las próximas versiones del programa superen algunas de las limitaciones de las versiones anteriores.

sites , corresponden a capas de puntos, tienden a desaparecer en las últimas versiones de GRASS.

Temas avanzados: Otros tipos de datos en GRASS

En una base de datos de GRASS existen más elementos que los tres tipos de capas anteriormente presentados

paint icon files iconos para ser usados con módulos *paint* para crear mapas (poco utilizados).

paint label files etiquetas para ser usados con módulos *paint* para crear mapas (poco utilizados).

region definition files ficheros de definición de regiones. Las regiones son un elemento básico en el manejo de GRASS

imagery group files definición de un *grupo de imagen* se trata de un conjunto de capas raster que son tratados en conjunto por algunos de los módulos de tratamiento de imágenes de satélite.

3D view parameters fichero de parámetros de visualización para el módulo **d.3d**

El comando **g.list** presenta un listado de cualquiera de los 8 elementos de información antes mencionados. Para ello hay que ejecutar el módulo seguido de un solo parámetro, el nombre del tipo de elemento del que quieres obtener un listado, estos nombres son: *rast, vect, sites, icon, labels, sites, region, group y 3dview*. Por ejemplo, para obtener un listado de los mapas vectoriales disponibles debes ejecutar:

GRASS: /path > **g.list** *tipo* <ENTER>

Este comando nos presenta un listado con todas las capas vectoriales presentes en el **MAPSET PERMANENT** y en nuestro **MAPSET**. De momento sólo aparecen capas en el **MAPSET PERMANENT** por que no hemos creado ninguna en el nuestro. Por ejemplo la salida de la orden:

GRASS: /path > **g.list** rast <ENTER>

será

```
raster files available in mapset PERMANENT:
ign mdecm suelos usos
```

Figura 2.5: Salida de la orden **g.list rast**

Capítulo 3

Visualización de mapas

3.1. Objetivos

- Utilizar diferentes monitores gráficos
- Manejar leyendas
- Entender y manejar paletas de color
- Modificar el área de trabajo
- Aprender el manejo de los módulos de GRASS incluidos en la tabla 3.1

d.erase	Borra un monitor gráfico
d.zoom	Permite definir una ventana para hacer zoom
d.pan	Desplaza la región de trabajo
d.legend	Obtiene una leyenda de un mapa en un monitor gráfico
g.region	Modifica la región de trabajo
r.patch	Crea una nueva capa raster superponiendo capas de información como si fueran transparentes
r.mask	Crea una máscara para limitar la visualización y cualquier operación de álgebra de mapas a un área irregular de la región de trabajo
r.colors	Asigna una paleta completa a un mapa
d.vect.line	Pinta un vectorial de lineas
d.vect.areas	Pinta un vectorial de polígonos

Cuadro 3.1: Módulos que van a utilizarse en esta práctica

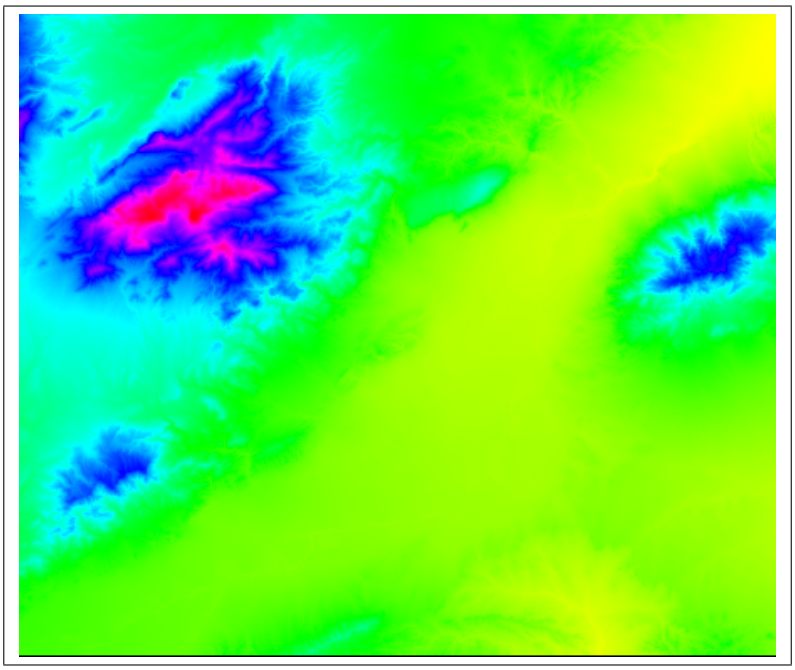


Figura 3.1: Mapa de elevaciones

3.2. Monitores gráficos y leyendas

Para visualizar cualquier gráfico en GRASS, es necesario abrir un *monitor gráfico*. Para ello se utiliza el comando **d.mon start=x0**. Pueden abrirse hasta 7 monitores diferentes cuyos nombres van de $x0$ a $x6$. El monitor puede moverse por la pantalla y redimensionarse utilizando el ratón siempre que no se este a mitad de ejecución de un módulo que necesita acceso al monitor..

A continuación, con el módulo **d.rast** podemos visualizar cada uno de los mapas que aparecían en el listado obtenido previamente. Si ejecutamos **d.rast** sin parámetros, pide el nombre del mapa que queremos visualizar, dándonos la opción de introducir *list* para ver los mapas raster disponibles. Podemos también introducir el módulo seguido del nombre de la capa que se quiere visualizar. Por ejemplo la orden:

GRASS: /path > **d.rast** mdecm <ENTER>

visualiza el mapa de elevaciones en el monitor gráfico (figura 3.1) y la orden:

GRASS: /path > **d.rast** urbanos <ENTER>

visualiza el mapa de núcleos urbanos que aparece en la figura 3.2

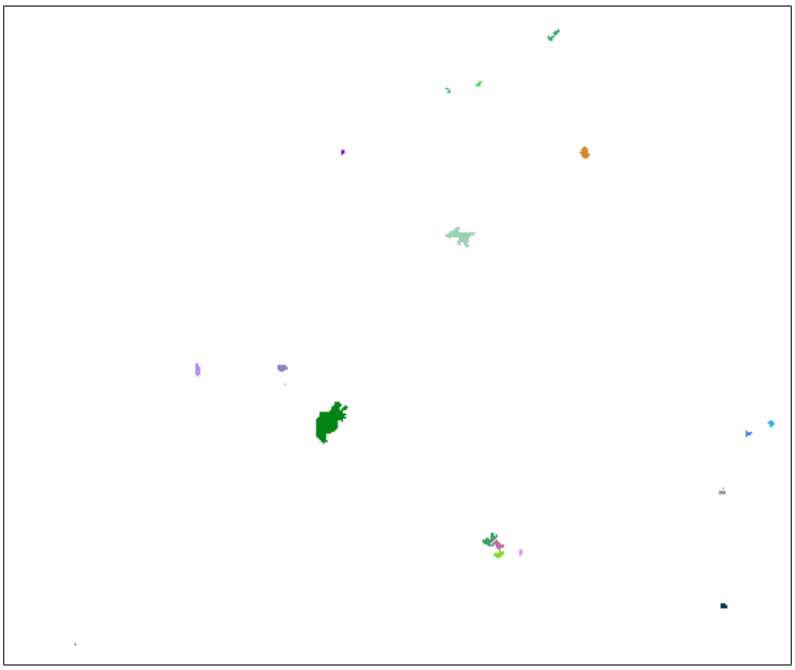


Figura 3.2: Mapa de núcleos urbanos

Resulta interesante comparar estos dos mapas pues, a pesar de estar ambos en formato raster, son de naturaleza distinta. El primero representa una superficie (una variable cuantitativa autocorrelacionada) mientras que el segundo es una representación raster de un mapa de objetos (los núcleos urbanos). Por tanto el espacio entre los nucleos no tiene, en este último mapa ningún valor; en GRASS se dice que tiene valor nulo (*NULL*).

Cuando encontremos valores nulos en una capa raster de objetos, significa que o no existe en esas celdillas ningún objeto del tipo descrito en la capa raster. Cuando aparezcan en una capa representando una superficie significa falta de datos. En cualquier caso el valor nulo es, evidentemente, distinto de valor cero.

En GRASS las celdillas con valor nulo se pintan por convención de blanco, salvo que apliquemos la opción **-o** de **d.rast**. En este caso las celdillas con valor nulo no se pintaran y, si hubiese un mapa pintado anteriormente, lo seguiremos viendo. Prueba a visualizar el mapa raster urbanos superpuesto al mapa mdecm :

GRASS: /path > **d.rast** mdecm <ENTER>

GRASS: /path > **d.rast** -o urbanos <ENTER>

El resultado de estas ordenes aparece en la figura 3.3

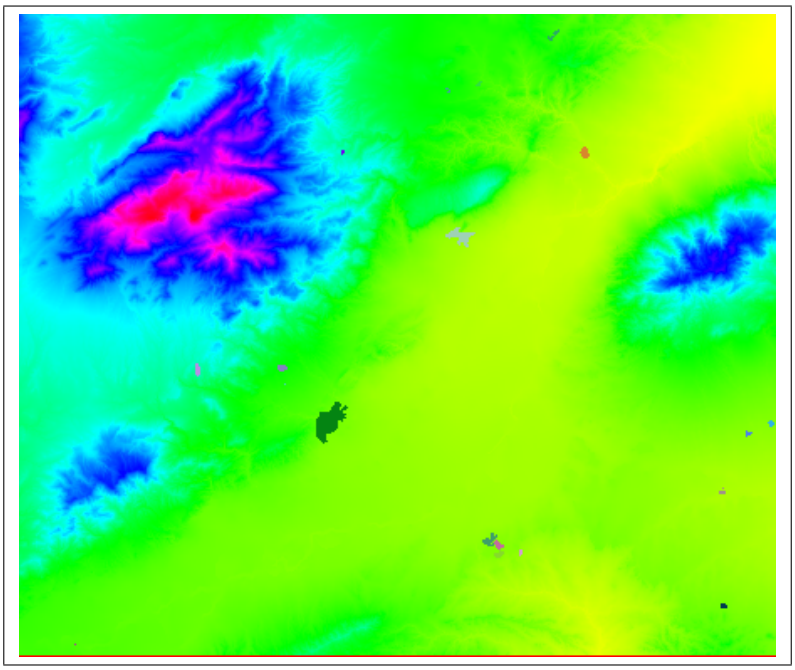


Figura 3.3: Ejemplo de overlay de mapas raster

De este modo las sucesivas capas de información que se dibujan con la opción **-o** de **d.rast** se superpondrán como si se tratara de una superposición de transparencias, en la que las celdillas con valor nulo se tornan transparentes. Esta superposición sobre el monitor gráfico no crea un nuevo mapa, para ello debe utilizarse el módulo **r.patch**. Este permite combinar diferentes capas manteniendo la paleta de colores y las etiquetas asociadas a cada categoría, solo las celdillas con valor no nulo se reemplazan.

> GRASS: /path > **r.patch input=***mapa1,mapa2,mapa3,...* **output=***mapa_compuesto* <ENTER>

El orden en que las capas a combinar se pasan por el parámetro **input** es importante ya que las capas que se seleccionen primero serán las últimas en pintarse y por tanto taparán a las demás. Por ejemplo, con la siguiente orden, la capa mdecm tapa totalmente la capa urbanos

> GRASS: /path > **r.patch input=**urbanos,mdecm **output=**otra_vez_mdecm <ENTER>

sin embargo con la orden:

> GRASS: /path > **r.patch input=**mdecm,urbanos **output=**urbanos_sobre_mdecm <ENTER>

la capa urbanos se superpone a mdecm y como aquella tiene muchos huecos permite ver esta sin dificultad (figura 3.3).

El módulo **d.rast** hace una conversión de los valores almacenados en la matriz de datos de la capa raster visualizada en colores de acuerdo a una serie de criterios, posteriormente se verá como el usuario puede especificar estos criterios. De momento nos conformaremos con verlos mediante una leyenda. Para ello utilizamos el módulo **d.legend**.

Para ello vamos en primer lugar a visualizar el mapa de usos del suelo:

GRASS: /path > **d.rast** usos <ENTER>

Para visualizar su leyenda debemos abrir un nuevo monitor gráfico

GRASS: /path > **d.mon start=**x2 <ENTER>

GRASS: /path > **d.legend** usos <ENTER>

Uno de los problemas que ocasiona la visualización de leyendas es que el número de categorías puede ser muy elevado o muy pequeño para el tamaño de la pantalla. Para solventar este problema se puede llamar a **d.legend** con la opción **lines=***n* donde *n* es el número de categorías que deberían entrar en la pantalla. Si el número de categorías reales es mayor que *n*, entonces sólo *n* categorías serán mostradas. Si es inferior, se situarán en la parte alta de la ventana gráfica dejando espacio para el resto de las *n teóricas* categorías.

No hay que olvidar que GRASS es un sistema de Información Geográfica no un programa de cartografía automática por lo que en las salidas gráficas se prima la sencillez sacrificando la estética. La producción de mapas para presentaciones o para su impresión en papel se deja a otros módulos específicos.

3.3. Visualización de vectoriales

El módulo **d.vect** permite visualizar un mapa vectorial, se trata de un módulo antiguo y bastante simple ya que sólo nos permite ver los puntos, lineas o límites de los polígonos que contiene un mapa vectorial pintados todos con un solo color seleccionado por el usuario entre la lista de colores estándard de GRASS (ver tabla 3.3) o mediante una tripleta RGB.

GRASS: /path > **d.vect** *mapa* **color=**red <ENTER>

GRASS: /path > **d.vect** *mapa* **color=**255:255:0 <ENTER>

red	rojo
orange	naranja
yellow	amarillo
green	verde
blue	azul oscuro
indigo	azul claro
white	blanco
black	negro
brown	marrón
magenta	púrpura
aqua	azul claro
gray o grey	gris

Cuadro 3.2: Lista de colores predefinidos en GRASS

esta última orden pintará las lineas de color amarillo.

El módulo **d.vect.line** mejora el módulo d.vect ya que permite seleccionar que entidades van a representarse en función de sus categorías.

GRASS: /path > **d.vect.line** *mapa* **color=**red **catnum**=1,3 <ENTER>

se pintarán de color rojo sólo aquellas lineas que tengan cuya categoría sea un valor numérico de 1 o 3.

El módulo **d.vect.area** pinta mapas de polígonos, pudiendo elegir, al igual que con **d.vect.line**, que entidades se pintarán y de que color, pueden pintarse de un color diferente los bordes y el interior de los polígonos (parámetros **linecolor** y **fillcolor** respectivamente. Incluye la opción -r para pintar cada polígono de un color diferente de forma aleatoria.

GRASS: /path > **d.vect.area** *mapa* **linecolor=**black **fillcolor=**red **catnum**=1,3 <ENTER>

Temas avanzados: Modificaciones en la codificación de la información vectorial

La versión 5 de GRASS esta suponiendo una transformación importante en la manera de almacenar y trabajar con la información vectorial. Hay una tendencia a almacenar cada vez más porciones de esta información en forma de bases de datos. El carácter abierto de GRASS y de algunos de los programas de bases de datos con los que se coordina (postgresql o MySQL por ejemplo) permite que, puesto que son públicas las estructuras de datos que manejan los programas y las funciones para acceder a las mismas, puedan programarse fácilmente módulos de GRASS para acceder a información, espacial o temática, almacenada en las bases de datos.

Uno de los formatos que más fácil y eficientemente pueden trasladarse a una base de datos son los mapas de sites, por lo que apenas se van a mencionar en este tutorial.

Para borrar el monitor gráfico podemos utilizar la orden **d.erase** seguida de uno de los colores estándard de GRASS (tabla 3.3, página 3.3). La opción por defecto es **black** aunque va a cambiar a **white** en la versión 5.3 del programa. Si los resultados se van a visualizar en la pantalla del ordenador es preferible un fondo negro, pero si las imágenes se van a capturar para su inclusión en un texto es más adecuado utilizar un fondo blanco.

3.4. Cambiando el punto de vista. Zoom, pan, cambio de región y máscaras

3.4.1. Hacer zoom en la región de trabajo (d.zoom)

El comando **d.zoom** es un módulo de manejo sencillo que proporciona la posibilidad de manejar una herramienta de zoom con sus tres posibilidades básicas:

1. Agrandar el zoom
2. Reducir el zoom
3. Hacer zoom sobre un rectángulo definido por el usuario en el monitor gráfico

Este módulo utiliza un **menú de ratón**, se trata de un tipo de menú muy característico de GRASS en el que las opciones se manejan en grupos de tres pinchando con uno de los tres botones del ratón sobre el monitor gráfico[1]. Por desgracia, el uso de **d.zoom** y sus menús de ratón varían mucho de unas versiones a otras de GRASS, es posible que en tu versión sea diferente de cómo se utiliza aquí, en todo caso será bastante intuitivo.

Al ejecutar el programa el menú de ratón que aparece es:

Buttons: Left: 1. corner Middle: Unzoom Right: Quit

[1] Recuerda que en Unix el botón central del ratón si se utiliza aunque en algunos casos se sitúa en el lateral izquierdo del ratón. En caso de que tu ratón sólo tenga dos botones existe l posibilidad de emular el central con una pulsación simultanea de los botones derecho e izquierdo (esta posibilidad se activa en la configuración del sistema gráfico de linux)

Si pulsamos el botón derecho el programa interpreta que se trata de la primera esquina de una ventana, nos indica las coordenadas del punto pinchado y pide una segunda esquina o bien que definamos de nuevo la primera esquina y vuelve al menú inicial:

```
4199280(N) 635375(E)

Buttons:
Left: 1. corner (reset)
Middle: 2. corner
Right: Quit
```

Si en el menú inicial se pulsa el botón central desharemos el zoom (haremos un zoom negativo).

3.4.2. Cambiar el centro de la región de trabajo (d.pan)

Complementario a **d.zoom** es el módulo **d.pan** que desplaza la región de trabajo. Al ejecutar este módulo nos aparece un menú de ratón con las siguientes opciones:

```
Buttons:
Left: Mark point to be at the center of the new region
Middle: Where am I?
Right: Quit
```

Si pulsamos el botón izquierdo resituamos el centro de la región de trabajo, pulsando el botón central obtenemos las coordenadas del punto pinchado y finalmente salimos con el botón derecho.

La ejecución de **d.zoom** y **d.pan** no sólo modifica los parámetros de visualización del monitor gráfico sino que también modifica la **región de trabajo**, este es un concepto fundamental en GRASS.

3.4.3. La región, rectangular, de trabajo (g.region)

Una capa raster está formada por una matriz de datos georreferenciada. Esta georreferenciación consiste en asignar coordenadas a los límites superior (Norte), inferior (Sur), derecho (Este) e izquierdo (Oeste) de la matriz. A partir de esos valores y del número de filas y columnas de la matriz, se deduce la resolución (tamaño de pixel) de la capa raster. En programas menos potentes, Idrisi por ejemplo, todas los capas raster que constituyen una base de datos deben tener exactamente la misma georreferenciación (filas, columnas, Este, Norte, Oeste, Sur y resolución). En GRASS las capas de una base de datos pueden ubicarse en el mismo lugar, en lugares diferentes o ser parcialmente coincidentes.

En sistemas tipo Idrisi el área de trabajo coincide, evidentemente, con la de las diferentes capas raster que constituyen el mapa. Sin embargo en GRASS es independiente de estas por lo que debe ser establecida por el usuario y puede modificarse en todo momento (aunque estas modificaciones han de hacerse con precaución puesto que van a modificar cualquier tipo de análisis o visualización posterior). En GRASS el área de trabajo se denomina **region** y los comandos para modificarla son básicamente **d.zoom** y **g.region**.

El comando **g.region** es bastante complejo y dispone de múltiples opciones y parámetros. Pueden distinguirse entre aquellos parámetros que modifican directamente las propiedades básicas de la región:

e= modifica la coordenada X del límite este de la región

w= modifica la coordenada X del límite oeste de la región

n= modifica la coordenada Y del límite norte de la región

s= modifica la coordenada Y del límite sur de la región

res= modifica la resolución o tamaño del pixel

nsres= modifica la resolución vertical o tamaño del pixel en sentido Norte-Sur

ewres= modifica la resolución horizontal o tamaño del pixel en sentido Este-Oeste

y aquellos parámetros que toman como parámetros de la región de trabajo los equivalentes de una capa espacial

rast= copia los parámetros de una capa raster

vect= copia los parámetros de una capa vectorial

sites= copia los parámetros de una capa de sites

puede también guardarse la región activa con un nombre (parámetro **[save=]**), cargarse una región previamente guardada (parámetro **[region=]**) o cargar la región por defecto con la opción **-d**. Puede obtenerse un listado de las regiones previamente guardadas con la orden:

GRASS: /path > **g.list region** <ENTER>

Finalmente las opciones **-p** y **-g** escriben en la pantalla de texto las características de la región activa con dos formatos diferentes.

3.4.4. El área de trabajo (r.mask)

Es necesario distinguir entre región de trabajo (rectangular y definida por unos límites y una resolución) y área de trabajo[2] como entidad geográfica en la que se lleva a cabo nuestro estudio y con límites irregulares (un municipio, una provincia, una cuenca hidrográfica, etc.). La utilización de **d.zoom** y **g.region**

[2]Denominar área y región de trabajo al territorio de estudio y al rectángulo que lo enmarca respectivamente resulta en realidad bastante arbitrario y quizás incorrecto. Sin embargo se ha adoptado este criterio por similitud con el uso del término region en GRASS

modifican la región de trabajo pero podemos determinar con exactitud cual es nuestra área de trabajo (para que GRASS la considere como tal) con el comando **r.mask**.

Se trata de un comando que se utiliza en modo interactivo; tras su ejecución aparece una pantalla que nos indica cual es la máscara que existe en ese momento (*current mask:*) y nos da la opción de eliminarla o introducir una nueva. Si se selecciona la segunda opción debemos elegir una capa raster a partir de la cual se definirá la máscara. Esta definición se hace asignando valores de 1 en la máscara a aquellos valores que queremos que entren dentro del área de trabajo. En definitiva se genera una variable binomial.

```
MASK: Program for managing current GIS mask

current mask: none


Options:
1 Remove the current mask
2 Identify a new mask
RETURN Exit program

>
```

que nos indica cual es la máscara actual *current mask*, si no hay ninguna definida muestra *none*. A continuación presenta dos opciones, la primera desactiva la máscara actual, la segunda define una nueva máscara. Si se selecciona la segunda opción debe elegirse el mapa que actuará como máscara y a continuación determinar que valores del mapa seleccionado se consideran incluidos en el área de trabajo y cuales no. En definitiva se construye una variable binomial de pertenencia o no al área de trabajo.

3.4.5. El módulo r.digit

En algunos casos puede interesar el utilizar cómo máscara un área irregular que no existe como mapa y que no puede obtenerse de forma sencilla a partir de operaciones de álgebra de mapas. Un ejemplo pueden ser áreas homogéneas en un análisis visual de una imagen de satélite o fotografía aérea digital. La manera de generar un mapa que contenga ese área y que, por tanto, pueda utilizarse posteriormente cómo máscara, es utilizar el comando **r.digit** que permite digitalizar directamente polígonos de forma arbitraria sobre una capa raster. El manejo es sencillo, antes que nada hay que definir el objeto que se va a digitalizar: (polígono, círculo o linea), hay que asignar un objeto y una categoría a cada objeto creado y, finalmente decirle al programa cómo se va a llamar el mapa creado.

```
Please choose one of the following
A define an area
C define a circle
L define a line
Q quit (and create map)
>
```

el menú de ratón posterior permite digitalizar uno a uno los vértices de la linea o polígono, o bien definir el centro y radio de un círculo. Al terminar de digitalizar el objeto se nos pide un identificador numérico y una etiqueta de texto para el objeto creado y se vuelve al menú inicial. Si en este se pulsa la letra Q el programa termina preguntando por el nombre de la capa raster que va a contener los objetos digitalizados.

Una cuestión importante a recordar acerca de las máscaras es que no afectan a las capas vectoriales.

Truco:

Aunque **r.mask** funciona sólo en modo interactivo, podemos generar una máscara en linea de comandos. En realidad **r.mask** lo único que hace es generar una capa raster llamada **MASK** que contiene valores 1 en las celdillas que deben visualizarse/analizarse y NULL en las que no. El resto de los módulos de GRASS *saben* que si existe una capa raster con ese nombre, deben centrarse en las celdillas especificadas en ella. Pues bien, la capa **MASK** puede generarse en linea de comandos con el módulo **r.mapcalc**:

GRASS: /path > **r.mapcalc** 'MASK=if(*mapabase*<=4 && *mapabase*>=2,1,null())' <ENTER>

Esta orden genera una máscara tal que sólo aquellas celdillas con valores de 2, 3 y 4 en el mapa *mapabase* se consideran incluidos dentro del área de trabajo. Para eleminarla basta con la orden:

GRASS: /path > **g.remove** MASK <ENTER>

Se trata de una opción muy útil cuando un script debe crear y eliminar máscaras. Ver más adelante los detalles sobre el manejo de **r.mapcalc**.

3.5. Manipulación de colores

La visualización de una capa raster en una pantalla de ordenador requiere la conversión de los valores de la variable almacenada en la matriz de datos en colores. Para ello se requiere una descripción cuantitativa

del color. La mayor parte de los sistemas de vídeo codifican los colores mediante el sistema RGB (Rojo, Verde, Azul); a cada uno de estos tres colores básicos se le asigna una intensidad que va de 0 a 255, de manera que el número de colores representables es de $256^3 = 16777216$ colores diferentes (los 16 millones de colores que prometen los folletos de las tarjetas gráficas que en realidad son bastantes más de los que pueden apreciarse con la vista). Estos valores se almacenan en un fichero especial de colores

Temas avanzados: colr y colr2

En versiones anteriores de GRASS, las paletas de colores se almacenaban en exclusiva en el directorio **colr** del MAPSET correspondiente. Esto significaba que el usuario sólo podía cambiar los colores de los mapas situados en su MAPSET (ya que sólo se tienen permisos de escritura en estos directorios). Sin embargo los colores de la presentación de una capa raster son algo bastante personal y, en todo caso, no es información fundamental que deba permanecer siempre constante.

Para permitir a los usuarios modificar la paleta de colores de mapas no presentes en su MAPSET se ideo el directorio **colr2**. Este se sitúa en el MAPSET del usuario, para que este pueda modificar su contenido, y contiene referencias a capas raster situadas en otros MAPSETS junto con la paleta de color que el usuario desea para ellos.

Por tanto para visualizar una capa raster debe establecerse una correspondencia entre valores e intensidades en estos tres colores. Esta correspondencia puede manipularse con los módulos **d.colors** y **r.colors**

Puede cambiarse globalmente una paleta con el comando **r.colors** que permite asignar a un mapa una de las siguientes opciones:

- **aspect**: paleta de niveles de gris optimizada para representar orientaciones
- **grey**: niveles de gris
- **grey.eq**: niveles de gris con histograma ecualizado
- **gyr**: verde - amarillo - rojo
- **rainbow**: paleta basada en la secuencia de colores del arcoíris
- **ramp**
- **ryg**: rojo - amarillo - verde
- **random**: paleta aleatoria, útil para variables cualitativas con pocos niveles)

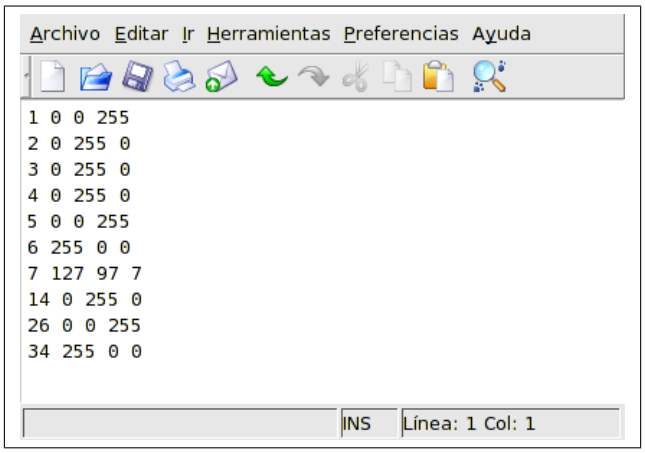


Figura 3.4: Reglas de asignación de colores

- **wave**: repite los mismos colores varias veces, útil para variables cuantitativas con un rango muy amplio
- **rules**: crear los colores interactivamente o mediante un fichero de texto.)

Se trata de una serie de paletas predefinidas excepto la última opción que nos permite definir una paleta. Si simplemente tecleamos:

GRASS: /path > **r.colors map=***mapa_raster* **color=***paleta*<ENTER>

La última de las paletas anteriormente listadas (rules) permite introducir a mano la definición de la paleta de colores, sin posibilidad de corregir si nos equivocamos y teniendo que empezar de cero si queremos probar otra combinación de colores. Por ello es preferible crear un fichero de texto con las reglas (con *emacs* por ejemplo) y *redirigir* este fichero al comando **r.colors**. Por ejemplo:

GRASS: /path > **emacs** elevation.color & <ENTER>

GRASS: /path > **r.colors map=** mdecm **color=**rules<elevation.color<ENTER>

Hay diferentes maneras de escribir un fichero de reglas de asignación de colores. La más simple es la que aparece en la figura 3.4.

Se trata básicamente de 4 columnas, la primera hace referencia a los valores de la variable contenida en la capa raster (en este caso elevaciones), la segunda a la intensidad del rojo (0-255), la tercera a la intensidad del verde (0-255) y la cuarta a la intensidad del azul (0-255). Resulta imprescindible terminar el fichero con la palabra *end*.

En este ejemplo se asigna al rango 1000 - 1200 metros colores que varían suavemente del rojo (255 0 0) al amarillo (255 255 0). Para ver más posibilidades, consultad la ayuda o el manual de **r.colors**.

Capítulo 4

Consultas y gestión de la base de datos

Además del almacenamiento y visualización de información espacial, uno de los objetivos fundamentales de un SIG es la consulta de los valores presentes en las capas de información manejadas, así como la extracción de informes y estadísticas relativas a las mismas. Los módulos encargados de estas tareas pueden clasificarse por un lado en función del tipo de información (raster, vectorial o sites) que contiene la capa consultada y por otro en función del nivel de detalle de la información que proporcionan:

- Módulos que proporcionan información global acerca del mapa, de sus propiedades geométricas (filas, columnas, coordenadas de los bordes y resolución) y de sus valores (**r.info**, **v.info**, **s.info**);
- Módulos que proporcionan estadísticas sobre la variable representada en una capa (**r.stats**, **r.report**, **d.histogram**, **r.univar**, **r.covar**);
- Módulos que permiten, pinchando sobre el mapa, conocer el valor de una o varias variables en posiciones concretas del espacio o la etiqueta asociada a determinados elementos en un mapa vectorial (**d.what.rast**, **d.what.vect**, **d.what.sites**);
- Módulos que enlazan con una base de datos temática para ampliar las capacidades de la base de datos espacial. Estos últimos se verán en el tema correspondiente a bases de datos.

Toda aplicación informática necesita, además, unas funcionalidades básicas de gestión de ficheros (copiar, borrar, renombrar, etc.). En GRASS cada capa, raster o vectorial, está compuesta por diversos ficheros, por tanto cada una de estas operaciones sobre una capa requiere hacer lo propio con todos los ficheros involucrados. Para facilitar la tarea existen tres comandos (**g.copy**, **g.rename** y **g.remove**.

Finalmente, en un SIG se trabaja con información codificada en diferentes formatos lógicos (en el caso de GRASS raster, vectorial y sites); se necesitan por tanto herramientas para que la información pueda pasar de unos formatos a otros.

4.1. Objetivos

- Obtener información sobre los mapas almacenados
- Copiar, borrar y renombrar mapas
- Consultar los valores de las distintas celdillas de una capa raster y los valores asociados a distintos objetos de una capa vectorial.
- Obtener informes sobre los contenidos de los mapas
- Cambiar el modelo lógico en que se basan las capas de datos
- Aprender el manejo de los módulos incluidos en la tabla 4.1

r.info	Obtiene información sobre capas raster
v.info	Obtiene información sobre capas vectoriales
s.info	Obtiene información sobre mapas de sites
g.remove	Borra mapas
g.rename	Renombra mapas
g.copy	Copiar mapas
d.where	Muestra las coordenadas de un punto pinchado en pantalla
d.what.rast	Muestra los valores de uno o varios mapas raster en el punto pinchado en pantalla
d.what.vect	Muestra las coordenadas de un punto pinchado en pantalla
d.what.sites	Muestra las coordenadas de un punto pinchado en pantalla
d.measure	Permite medir objetos sobre el monitor gráfico
r.stats	Contabiliza las celdillas que pertenecen a cada categoría
r.report	Obtiene una tabla resumen de los datos de un fichero raster a partir del resultado de r.stats
d.histogram	Muestra el histograma de un mapa raster en el monitor activo
r.univar	Obtiene estadísticos sencillos de la variable representada en la capa
r.covar	Obtiene una matriz de covarianzas o correlaciones de un conjunto de capas
r.contour	Transforma mapas raster de superficies a isolineas
r.poly	Transforma mapas de polígonos o de variable cuantitativa en raster a vectorial
r.line	Transforma mapas raster representando lineas en vectoriales
r.to.sites	Transforma mapas raster en mapas de sites
v.to.rast	Transforma mapas vectoriales en raster independientemente del tipo de objetos que contengan
v.to.sites	Transforma mapas vectoriales con puntos en mapas de sites
s.to.rast	Transforma mapas de sites en raster
s.to.vect	Transforma mapas de sites en vectoriales

Cuadro 4.1: Módulos que van a utilizarse en esta sección

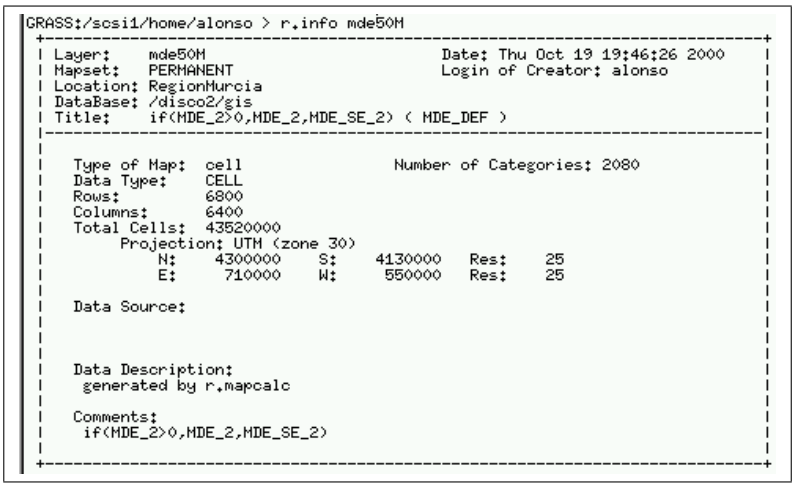


Figura 4.1: Salida del módulo r.info

4.2. Módulos para obtener información global

Los módulos **r.info** y **v.info** proporcionan información general sobre capas raster y vectoriales respectivamente.

GRASS: /path > **r.info** *mapa_raster*<ENTER>

GRASS: /path > **v.info** *mapa_vectorial*<ENTER>

Las figuras 4.1 y 4.2 muestran las salidas de ambos programas. El módulo **s.info** se comporta de manera similar para capas de puntos.

4.3. Módulos para la obtención de gráficos descriptivos y estadísticas

El contenido de una capa raster puede ser:

- Un conjunto de objetos (por ejemplo núcleos urbanos)
- Una variable cualitativa (por ejemplo usos del suelo)
- Una variable cuantitativa, una superficie (por ejemplo elevaciones)

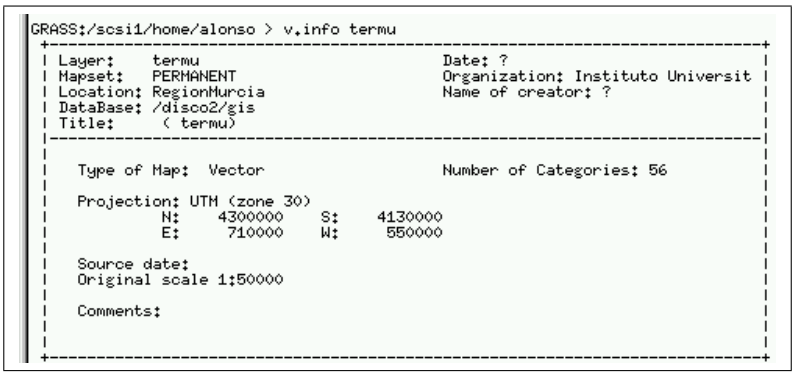


Figura 4.2: Salida del módulo v.info

En los dos primeros casos, la obtención de estadísticos básicos puede limitarse al área ocupada por cada objeto o cada valor de la variable cualitativa. En el último caso podemos aplicar los estadísticos clásicos de la estadística multivariante (media, desviación típica, etc.)

El comando básico de cálculo de estadísticos es **r.stats**, básicamente extrae la frecuencia en celdillas de cada uno de los valores de la variable representada por una capa raster. Sin embargo sus resultados se presentan de forma bastante espartana por lo que su no suele utilizarse directamente y si utilizando otros módulos intermedios que extraen la información de **r.stats** y la presentan de forma más adecuada. De este modo el uso de **r.stats** suele reservarse para la programación de scripts que es un aspecto excesivamente avanzado para ser objeto de este tutorial.

Uno de estos módulos que producen una salida más agradable es **r.report**. Permite obtener el área ocupada por las diferentes categorías en un mapa o por diferentes combinaciones de categorías; pueden seleccionarse diferentes unidades para medir las áreas.

GRASS: /path > **r.report** usos **units=**k <ENTER>

Nos dará las áreas ocupadas por diferentes usos del suelo en kilómetros cuadrados. Las unidades disponibles son: millas cuadradas (mi), metros (m), kilometros (k), acres (a), hectáreas (h), celdillas (c) y porcentaje (p). En la figura 4.3 aparece la salida de esta orden en el monitor de texto.

Para incluir varios mapas en un informe, la sintaxis es:

GRASS: /path > **r.report map=***mapa_raster1*, *mapa_raster2*,...,*mapa_rastern* **units=***unidades* <ENTER>

El comando **d.histogram** es el equivalente gráfico a **r.report**, presenta en pantalla un histograma de frecuencias de los valores almacenados en el mapa al que se hace referencia. Presenta la peculiaridad de que dibuja las barras con los mismos colores que aparecen en la imagen por lo que resulta un buen

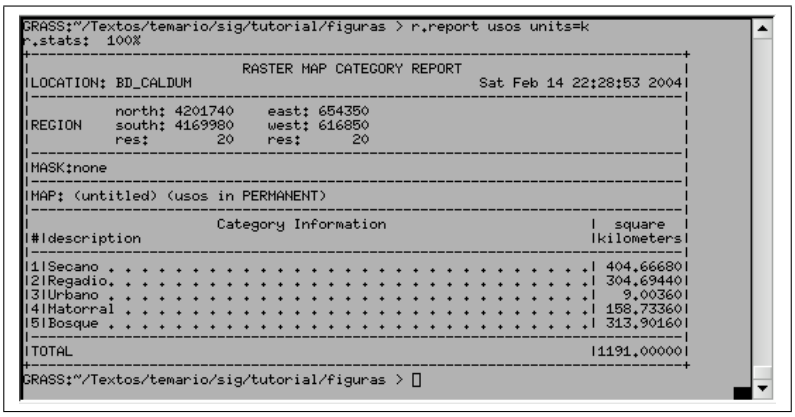


Figura 4.3: Salida del módulo r.report

método alternativo a **d.legend** para ver la relación entre valores y colores en una capa raster. Al igual que ocurre con **d.legend**, es necesario borrar la pantalla con **d.erase** antes de ejecutar este comando ya que si no el histograma se superpondría con el contenido anterior de la pantalla.

El comando **r.univar** realiza un cálculo de estadísticos básicos sobre la variable cuantitativa almacenada en una capa raster. Los estadísticos calculados son:

- Number of cells, número total de celdillas (N)
- Minimum, valor mínimo
- Maximum, valor máximo
- Rango, rango de valores= máximo - mínimo
- Arithmetic Mean, media aritmética (m)
- Variance, varianza (s^2)
- Standard deviation, desviación típica ($s = \sqrt{s^2}$)
- Variation coefficient, coeficiente de variación ($V = 100xs/m$)

Un ejemplo típico de la salida de **r.univar** podría ser:

```
Number of cells: 1405744
Minimum: 10
Maximum: 255
Range: 245
Arithmetic Mean: 94.0227
Variance: 527.623
Standarddeviation: 22.97
Variation coefficient: 24.4303 %
```

Finalmente el módulo **r.covar** permite obtener la matriz de varianzas-covarianzas o de correlaciones correspondiente a un conjunto de mapas raster:

GRASS: /path > **r.covar** *mapa1,mapa2,...* <ENTER>

devolverá la matriz de varianzas covarianzas y

GRASS: /path > **r.covar -r** *mapa1,mapa2,...* <ENTER>

la matriz de correlaciones.

4.4. Obteniendo información puntual sobre el contenido de los mapas

Una vez visualizados los mapas, resulta interesante saber algo sobre el tipo de información que contienen. Para ello vamos a utilizar diferentes comandos.

El más simple es **d.where** que nos indica las coordenadas de los diferentes puntos donde pinchemos dentro del monitor gráfico. Al ejecutar el comando aparece un menú de ratón que nos indica las diferentes opciones.

GRASS: /path > **d.where**<ENTER>

```
Buttons:
Left: where am i
Middle: draw to/from here
Right: quit this

EAST: NORTH:
```

a partir de aquí cada vez que pinchemos en un punto del monitor gráfico con el botón izquierdo nos devolverá las coordenadas X e Y en las columnas EAST y NORTH respectivamente.

Sin embargo esta orden no nos da información sobre el contenido de los mapas. Para obtener esta información tenemos los comandos **d.what.rast**, **d.what.vect** y **d.what.sites** para obtener información acerca de mapas raster, vectoriales y de sites respectivamente.

GRASS: /path > **d.what.rast map=***mapa_raster*<ENTER>

Buttons Left: what's here Right: quit

Pinchando con el botón derecho del ratón nos dará, además de las coordenadas del punto pinchado el valor de la variable contenida en *mapa_raster*

GRASS: /path > **d.what.vect map=***mapa_vectorial* <ENTER>

Genera el mismo menú de ratón que el módulo anterior pero ahora devuelve información de distinto tipo.

Si estos comandos se utilizan sin argumentos devolverán los datos del mapa representado en el monitor gráfico; se le puede añadir map=seguido de varios mapas y obtendremos información de todos ellos.

El comando **d.measure** permite medir longitudes y áreas de objetos digitalizados en el monitor gráfico. Cuando se ejecuta, aparece el siguiente menú de ratón:

Buttons: Left: where am i Middle: set FIRST vertex Right: quit this

El botón izquierdo nos dirá las coordenadas del punto en que pinchamos, con el derecho salimos del programa, y el central determina el primer vértice y entra en un nuevo menú:

Left: where am i Middle: set NEXT vertex Right: FINISH

Pinchando con el botón central iremos definiendo una linea y un polígono interior a la misma. Cada nuevo vértice significa la adición de un nuevo tramo y el aumento de la longitud de la linea, tal como va mostrando el programa

LEN: 26.45 meters LEN: 56.97 meters LEN: 93.97 meters LEN: 127.69 meters LEN: 162.43 meters LEN: 186.48 meters LEN: 199.90 meters

Cuando finalmente pinchemos con el botón derecho (no es necesario cerrar explícitamente el polígono, el programa asume la existencia de una linea de cierre desde el punto final al inicial) aparecerán la longitud y el área del polígono en diferentes unidades junto a la posibilidad de volver a definir otro polígono o salir del programa.

Buttons: Left: DO ANOTHER Middle: Right: quit this LEN: 199.90 meters AREA: 0.36 hectares 0.0014 square miles 3644.32 square meters

4.5. Gestión de la base de datos

Los comandos **g.copy**, **g.rename** y **g.remove** son similares en cuanto a su utilización y se usan para copiar y renombrar mapas. el funcionamiento es bastante similar en cualquiera de los tres casos. Con diversos ejemplos resultará más sencillo entenderlo.

GRASS: /path > **g.copy** rast=acuiferos,misacuiferos<ENTER>

GRASS: /path > **g.remove** vect=red_drenaje <ENTER>

En el primer caso se crea una capa raster llamada *misacuiferos* igual a la capa *acuiferos* preexistente, en el segundo se borra la capa vectorial red_drenaje .

En algunos casos se necesitará copiar una capa de un MAPSET a otro con el mismo nombre. En estos casos es necesario añadir al nombre de la capa el nombre del MAPSET

GRASS: /path > **g.copy** rast=acuiferos@PERMANENT,acuiferos@mimapset<ENTER>

El último ejemplo copiará el mapa *acuiferos* a nuestro MAPSET dándole el mismo nombre.

Con ellos podemos manipular no sólo mapas sino tambien otro tipo de ficheros:

- *rast* Ficheros que componen un mapa raster
- *vect* Ficheros que componen un mapa vectorial
- *icon* Fichero con definición de iconos
- *labels* Fichero con definición de etiquetas
- *sites* Mapa de sitios
- *region* Fichero que define una región
- *group* Fichero que define un *grupo* (en el análisis de imágenes de satélite)
- *3dview* Fichero que contiene los parámetros para una visualización 3D

4.6. Transformaciones entre formatos

Todo programa para gestionar información espacial debe tener una serie de utilidades para convertir esta información de unos formatos a otros. En GRASS existen 3 formatos básicos:

- raster
- vectorial
- sites

El formato *sites* hace referencia a un conjunto de datos puntuales definidos por una tabla que incluye las coordenadas de cada punto, un identificador y una serie de datos asociados a cada punto. Este formato resulta en realidad redundante con el vectorial por lo que tiende a desaparecer en las últimas versiones del programa.

Todos estos módulos son de manejo muy sencillo, sus dos parámetros básicos son el mapa de entrada y el de salida que, puesto que tendrán formatos distintos y se almacenarán en directorios diferentes, pueden tener el mismo nombre. En algunos casos aparecen parámetros complementarios en función del propósito del módulo.

4.6.1. Módulos para transformar datos raster en vectorial

Variables cuantitativas: r.contour

Este módulo genera un mapa vectorial de isolineas a partir de un raster que contiene una superficie:

> GRASS: /path > **r.contour input=**mdecm **output=**mdecm **levels=**20000,40000,60000,80000,100000,12
> <ENTER>

el parámetro **levels** indica los valores de las isolineas que se van a crear. Una forma alternativa a la utilización del parámetro **levels** son los parámetros **minlevel**, **maxlevel** y **step** que definen el valor de la curva de nivel mínima y máxima y la equidistancia[1] de curvas de nivel.

Variables cualitativas y polígonos: r.poly

Este módulo transforma una capa raster con polígonos o con una variable cualitativa en su equivalente en formato vectorial.

> GRASS: /path > **r.poly input=**usos **output=**usos <ENTER>

[1] diferencia de valor entre cada par de isolineas

Lineas: r.line

Este módulo transforma una capa raster con lineas en su equivalente vectorial. Antes de ejecutarlo es recomendable comprobar que las lineas de la capa raster tienen una sola celdilla de anchura. Para garantizarlo se utiliza el módulo **r.thin**.

> GRASS: /path > **r.line input=***mapa_raster_de_lineas* **output=***mapa_raster_de_lineas* <ENTER>

4.6.2. Módulos para transformar datos vectoriales en raster

Se aplica a cualquier mapa vectorial, en función de cuales sean los datos de entrada (lineas, polígonos o puntos) se generará su correspondiente versión en formato raster.

> GRASS: /path > **v.to.rast input=** urbanos **output=** urbanos <ENTER>

Módulos para transformar datos de sites

El propósito de los cuatro módulos que transforman datos de sites debería quedar suficientemente explicado con sus nombres:

- s.to.rast
- s.to.vect
- r.to.sites
- v.to.sites

Solamente uno de estos módulos (**s.to.rast** se utilizará posteriormente para pasar de mapas de puntos a formato raster al aplicar técnicas de interpolación:

> GRASS: /path > **s.to.rast input=** *mapa_de_puntos* **output=** *mapa_de_puntos* <ENTER>

Capítulo 5

Álgebra de mapas

Los capítulos anteriores se han centrado en la visualización y la consulta de la información espacial, es ya tiempo de entrar en el análisis y la manipulación de datos espaciales.

El *álgebra de mapas* constituye un conjunto de herramientas que van a permitir generar nuevas capas de información a partir de las capas preexistentes, siempre en formato raster. El fundamento del *álgebra de mapas* es la consideración de las diferentes capas raster como matrices de datos de manera que los diferentes operadores de no son más que operaciones sobre una o varias matrices/capas que generan una nueva matriz/capa.

Algunos de estos operadores se corresponden con operaciones habituales del álgebra matricial, pero otras son específicas del *álgebra de mapas.*

En el capítulo anterior se ha visto ya el comando **r.mapcalc** y el lenguaje de álgebra de mapas asociado a él. En este se van a repasar algunos de los módulos que implementan aquellos operadores excesivamente complejos para ser programados con mapcalc.

Objetivos:

- Entender el concepto de operador en un SIG raster
- Entender las diferencias entre los distintos tipos de operadores
- Aprender el manejo de los módulos incluidos en la tabla 5.1
- Revisar y poner en contexto algunos de los módulos vistos anteriormente

5.1. Introducción

Un operador de álgebra de mapas genera una o más capas de raster a partir de una o más capas raster de entrada. Existen varios tipos fundamentales de operadores en álgebra de mapas[1]:

[1] Esta clasificación es hasta cierto punto personal, en la bibliografía podéis encontrar otras

r.reclass	Reclasifica los valores de un mapa
r.neighbors	Da a cada celda un valor que es función de los valores de las celdas circundantes. Incluye varias funciones predefinidas
r.mfilter	Permite crear operadores de vecindad que calculan una media ponderada de los valores de los pixels circundantes
r.thin	Considera que los objetos representados en la capa raster son lineales y los adelgaza hasta un grosor de una sola celdilla
r.grow	Agranda en una celdilla cualquier objeto representado en formato raster
r.clump	Reclasifica los datos en un mapa raster agrupando celdas contiguas que contienen el mismo valor.
r.buffer	Crea zonas buffer (áreas tampón) alrededor de un objeto
r.cost	Genera mapas de coste acumulado
r.drain	Determina la ruta de máxima pendiente que seguiría una bola rodando por un mapa de elevaciones
r.average	Crea un mapa con los valores medios de una variable cuantitativa para distintos valores de una variable cualitativa
r.statistics	Calcula estadísticos de una variable cuantitativa para diferentes valores de una variable cualitativa
r.buffer	Genera áreas situadas a determinadas distancias de un objeto u objetos representados en una capa raster
r.distancias	Genera una capa de distancias a objetos y además una capa con el objeto más cercano a cada punto. Admite como entrada mapas de coste

Cuadro 5.1: Módulos que van a utilizarse en esta sección

- Operador local
- Operador de vecindad
- Operadores de bloque
- Operadores extendidos
- Operadores de área
- Operadores globales

5.2. Operadores locales

Son aquellos que asignan a cada celdilla de las capas de salida un valor obtenido de los valores de la misma celdilla en las capas de entrada. El más sencillo es **r.reclass** que hace una reclasificación de mapas.

5.2.1. Reclasificación

La modificación de una paleta de colores no evita el problema de las leyendas demasiado grandes en el caso de variables cuantitativas. Para solventarlo lo más efectivo es reclasificar los valores del mapa para transformar un mapa de variable cuantitativa en un mapa de variable cualitativa con una leyenda razonablemente corta para su visualización con **d.legend**.

El comando que hace reclasificación es **r.reclass**, se trata evidentemente de un operador local de álgebra de mapas que asigna a cada celdilla del nuevo mapa un valor en función del rango de valores al que pertenezca esa misma celdilla en el mapa de entrada. La ejecución del módulo es sencilla:

GRASS: /path > **r.reclass input=***mapa_entrada* **output=***mapa_salida* <ENTER>

Si lo ejecutamos tal cual deberemos introducir una a una las reglas de reclasificación, de modo similar a la opción *rules* de **r.colors**, y, al igual que en ese caso, es más eficiente crear un fichero con las reglas de reclasificación y redireccionárselo. Como ejemplo de fichero de reglas de reclasificación:

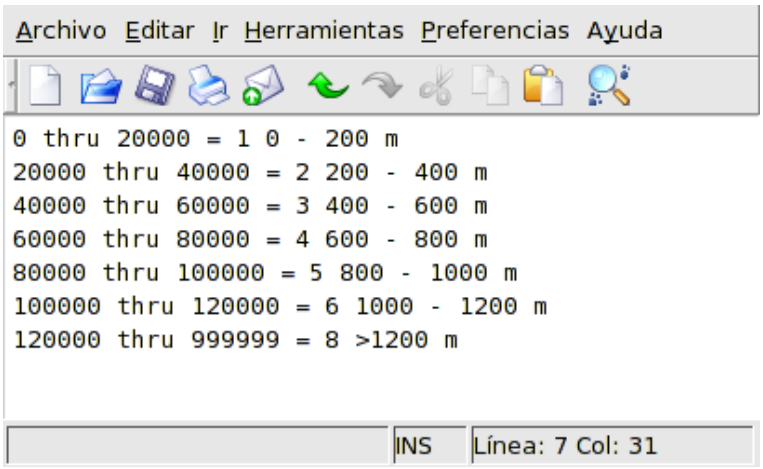


Figura 5.1: Ejemplo de reglas de reclasificación

Suponiendo que este fichero se denomina reclass.txt, la orden quedaría:

GRASS: /path > **r.reclass input=***mapa_entrada* **output=***mapa_salida*<reclas.txt <ENTER>

Que, por ejemplo asigna a los valores entre 0 y 1000 el valor 1 (que será el que se almacena en la celdilla correspondiente de la matriz de datos) y la etiqueta de texto “0 -1000 m” que aparecerá cuando ejecutemos **r.what.rast** o **d.legend**.

5.3. Operadores de vecindad

El valor de cada celdilla de la capa de salida es función de los valores de un conjunto de celdillas, situadas en torno a ella, en la capa de entrada. Generalmente se trabaja con grupos cuadrados de celdillas y el tamaño de la vecindad corresponde a la anchura (o altura) en celdillas del cuadrado y será siempre un número impar. Por ejemplo

1	2	3
4	5	6
7	8	9

representa una vecindad de tamaño 3 centrada sobre la celdilla número 5, mientras que

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

representa una vecindad de tamaño 5 en torno a la celdilla número 13.

5.3.1. r.neighbors

Da a cada celda un valor que resulta del cálculo de un estadístico a partir de los valores de las celdillas circundantes en la capa de entrada.

> GRASS: /path > **r.neighbors input=***capa_entrada* **output=***capa_salida*
> **method=***estadístico* **size=***tamaño_de_la_vecindad* <ENTER>

Los estadísticos que pueden utilizarse son:

- **average**, calcula la media aritmética de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		3.66	
7	5	3			

- **median**, calcula la mediana de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		3	
7	5	3			

- **mode**, calcula la moda de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		3	
7	5	3			

- **minimum**, calcula el máximo de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		1	
7	5	3			

- **maximum**, calcula el mínimo de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		7	
7	5	3			

- **stddev**, calcula la desviación típica de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		1.8	
7	5	3			

- **variance**, calcula la varianza de los valores del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		3.25	
7	5	3			

- **diversity**, calcula el número de valores diferentes del mapa de entrada en las celdillas de la vecindad y la asigna a la celdilla central en el mapa de salida

1	2	3			
4	5	3		6	
7	5	3			

- **interspersion**, porcentaje de celdillas que contienen valores diferentes al de la celdilla central

1	2	3			
4	5	3		78	
7	5	3			

5.3.2. r.mfilter

Permite crear operadores de vecindad que calculan una media ponderada de los valores de las celdillas circundantes

GRASS: /path > **r.mfilter input=***mapa* **output=***mapa2* **filter=***archivo* <ENTER>

Los parámetros **input** y **output** indican los mapas de entrada y salida y el parámetro **filter** un fichero que contiene los criterios de filtrado escritos siguiendo unas reglas sencillas. Así, este comando funciona de modo similar a **r.colors** o **r.reclass** en el sentido de que debe utilizarse un pequeño archivo de texto en el que se especifique el filtro a utilizar. En la figura 5.2 se presentan dos ejemplos de filtro.

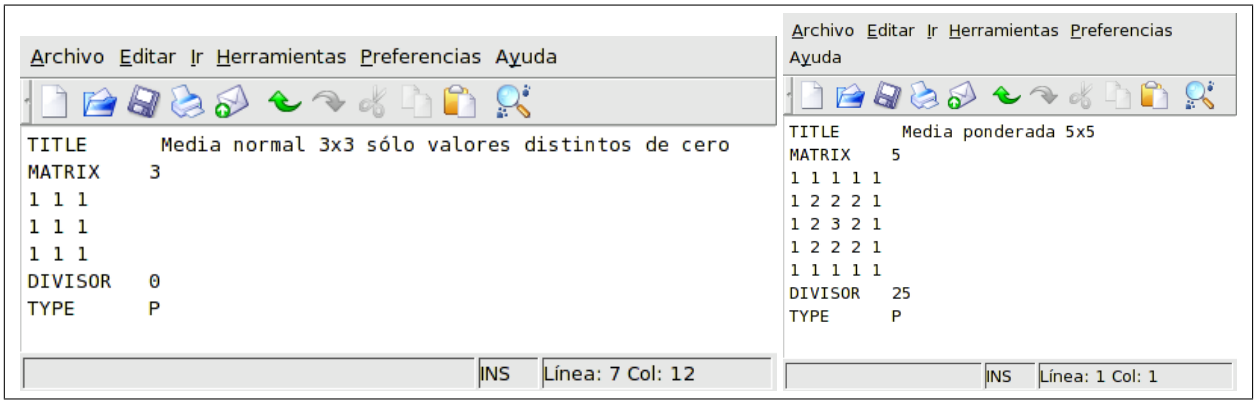


Figura 5.2: Dos ejemplos de filtros

Con estos ficheros vamos a pasar a **r.mfilter** los siguiente parámetros:

- **TITLE:** Es un texto descriptivo del filtro y su objetivo

- **MATRIX:** El tamaño de la matriz de filtrado n y a continuación n lineas, con n enteros separados por al menos un espacio, cada una. El valor n debe ser un número impar igual o mayor que 3. Representa la matriz de coeficientes de ponderación por los que se multiplicará cada una de las celdillas de la ventana.
- **DIVISOR:** El número por el que se va a dividir el resultado de sumar los valores hallados en le mapa multiplicados por los especificados en la matriz de filtrado, si no se especifica el valor por defecto es 1. Si se especificó 0 como divisor, el programa calcula el divisor como la suma de los valores de la matriz en los que el valor de la celdilla era diferente de cero. Se consigue así una media ponderada.
- **TYPE:** Filtro de tipo S significa que es *secuencial* mientras que el tipo P significa que es *paralelo*. En un filtro secuencial, los valores ya procesados se utilizan para procesar los siguientes, mientras que en un filtro paralelo los valores del mapa de entrada se conservan para realizar todos los cálculos.

El proceso de filtrado produce un nuevo valor para cada celdilla en el fichero de entrada multiplicando los valores de las celdillas incluidas en la ventana n x n, sumando todos los resultados y dividiendo esa suma entre el divisor especificado. Si se especificó 0 como divisor, el programa calcula el divisor como la suma de los valores de la matriz en los que el valor de la celdilla era diferente de cero. Se consigue así una media ponderada.

Si se incluye más de un filtro en el fichero, o si el valor asignado a repeat es mayor de 1, los filtros se ejecutan secuencialmente uno detrás de otro hasta que, al finalizar el proceso, los resultados se escriben en el fichero de salida.

Algunos ejemplos del resultado de diversas matrices de filtrado puede verse a continuación:

1	2	3	1	1	1			
4	5	3	1	1	1		3.66	
7	5	3	1	1	1			

1	2	3	1	1	1			
4	5	3	1	2	1		3.8	
7	5	3	1	1	1			

1	2	3	-1	1	-1			
4	5	3	1	2	1		1	
7	5	3	-1	1	-1			

Este tipo de operadores de vecindad se utilizan mucho en teledetección para resaltar u ocultar elementos en la visualización de imágenes de satélite.

5.3.3. r.grow y r.thin

El módulo **r.grow** expande un pixel los objetos que aparezcan en un mapa raster. Se define como objeto en un mapa raster un conjunto de celdillas adyacentes con valores no nulos. CELDILLAS CON VALOR CERO DEFINEN POR TANTO UN OBJETO. Para transformar las celdillas con valor cero en nulos puede utilizarse el módulo **r.null** o hacer un programa para **r.mapcalc**[2].

GRASS: /path > **r.grow input=***mapa* **output=***mapab* <ENTER>

0	0	0	0	0		0	1	1	1	0
0	0	1	0	0		0	1	1	1	0
0	0	1	0	0	r.grow	0	1	1	1	0
0	0	1	0	0		0	1	1	1	0
0	0	0	0	0		0	1	1	1	0

El módulo **r.thin** realiza la operación contraria a **r.grow**. Adelgaza los objetos encontrados hasta una anchura de una celdilla. Este módulo resulta útil para, por ejemplo, procesar un mapa escaneado en el que los objetos lineales aparezcan con grosor variable. A continuación con el módulo **r.line** las lineas pasarían a formato vectorial.

GRASS: /path > **r.thin input=***mapa* **output=***mapab* <ENTER>

1	1	1	0	0		0	1	0	0	0
0	1	1	1	0		0	0	1	0	0
0	1	1	1	0	r.thin	0	0	1	0	0
0	0	1	1	0		0	0	1	0	0
0	0	1	1	1		0	0	0	1	0

El módulo **r.thin** podría clasificarse más apropiadamente como de vecindad extendida pero por su carácter complementario a **r.grow** se ha presentado aquí.

5.4. Operadores de vecindad extendida

5.4.1. r.clump

Reclasifica los datos en un mapa raster agrupando celdas contiguas que contienen el mismo valor dándoles un valor único.

[2] $capa2 = if(capa1 == 0, null(), capa1)$

GRASS: /path > **r.clump input=***nombre_mapa* **output=***nombre_mapa2* <ENTER>

Este módulo resulta útil cuando se necesita individualizar manchas homogéneas. Por ejemplo cuando queremos analizar cada una de las parcelas de matorral por separado en lugar de todo el suelo ocupado por matorral en conjunto. En el siguiente ejemplo aparecen 3 clases, cada una de ellas define un valor en una variable cualitativa, al aplicar r.clump se convierten en 7 polígonos con identificador distinto.

1	1	1	2	2		1	1	1	2	2
3	3	1	1	2		3	3	1	1	2
3	1	1	2	2	r.clump	3	1	1	2	2
2	1	1	1	1		4	1	1	1	1
1	2	3	3	3		5	6	7	7	7

Date cuenta, en la esquina inferior izquierda de la tabla de que **r.clump** sólo hace *movimientos de torre* no *movimientos de rey* es decir que se considera que dos celdillas que sólo comparten una esquina no forman parte del mismo objeto.

5.4.2. r.drain

Genera la ruta que seguiría una pelota dejada en un punto del terreno (parámetro **coordinate**) siguiendo el camino de máxima pendiente. El mapa de elevaciones se le pasa al módulo mediante el parámetro **input**. El parámetro **output** indica cual será el mapa que contiene la ruta.

GRASS: /path > **r.drain input=***name* **output=***name* **coordinate=***x,y* <ENTER>

Los parámetros **input** y **output** hacen referencia al mapa de entrada y salida respectivamente, el parámetros **coordinate** indica el punto desde el que se inicia la modelización.

La siguiente tabla muestra el resultado que tendría **r.drain** en un modelo de elevaciones partiendo de la esquina superior izquierda.

8	8	9	8	6		1	0	0	0	0
7	6	5	6	7		0	1	1	0	0
7	6	6	4	5	r.drain	0	0	0	1	0
6	5	4	2	3		0	0	0	1	0
6	5	3	2	1		0	0	0	0	1

5.5. Operadores de área

5.5.1. r.statistics

El comando básico para obtener estadísticos de una variable cuantitativa para diferentes valores de una variable cualitativa (o para diferentes polígonos) es **r.statistics**. Este módulo permite calcular diversos estadísticos:

- **distribution** Distribución de valores
- **average** Media
- **mode** Moda
- **median** Mediana
- **avedev** Desviación media
- **stddev** Desviación típica
- **variance** Varianza
- **skewness** Sesgo
- **kurtosis** Curtosis
- **min** Valor mínimo
- **max** Valor máximo

Todas estas opciones generan una capa raster salvo la primera que genera un listado con la frecuencia de aparición de cada combinación de variable cualitativa y cuantitativa

La orden para determinar, por ejemplo, la altitud media en distintos tipos de uso de suelo sería:

> GRASS: /path > **r.statistics base=**usos **cover=**mdecm **method=**average **output=** dev_altitud <ENTER>

El parámetro **cover** indica la capa raster que contiene la variable cuantitativa, el parámetro **base** la capa raster que contiene la variable cualitativa y el parámetro **method** el estadístico que se va a calcular. Finalmente en **output** se consigna el nombre de la capa de salida que va a contener los valores del estadístico.

Este módulo no genera, por tanto, una tabla de datos sino una nueva capa raster en la que a cada celdilla le corresponde el estadístico calculado para su valor en la capa de variable cualitativa. En la siguiente tabla se muestra un ejemplo de la orden anterior. La parte de la izquierda muestra el mapa base, la central el mapa cover y la de la derecha el resultado:

1	1	1	2	2		10	18	19	12	12		21.3	21.3	21.3	23.1	23.1
3	3	1	1	2		31	23	21	21	12		26.8	26.8	21.3	21.3	23.1
3	1	1	2	2		13	12	18	25	22		26.8	21.3	21.3	23.1	23.1
2	1	1	1	1		24	21	31	31	32		23.1	21.3	21.3	21.3	21.3
2	2	3	3	3		34	44	35	29	30		23.1	23.1	26.8	26.8	26.8

5.5.2. r.areas.sh

Otro operador de área de interés es **r.areas.sh**. Este simplemente calcula el área ocupada por cada polígono o por cada clase de una variable cualitativa y asigna este valor a los polígonos o clases correspondientes en el mapa de salida. Permite seleccionar la unidad de medida de superficie (utilizando las mismas abreviaturas que con el comando **r.report**. Por ejemplo:

GRASS: /path > **r.areas.sh input=**urbanos **output=**urbanos **units**=k <ENTER>

nos dará un mapa en el que cada núcleo urbano recibirá su extensión (lógicamente las zonas no urbanas cuyas celdillas contienen valor NULL seguirán con este valor. En el siguiente ejemplo se aprecia el comportamiento de **r.areas.sh** asumiendo que las unidades pedidas son celdillas (**units**=c).

1	1	1	2	2		11	11	11	8	8
3	3	1	1	2		6	6	11	11	8
3	1	1	2	2	r.areas	6	11	11	8	8
3	1	1	1	1		6	11	11	11	11
2	2	2	3	3		8	8	8	6	6

Este módulo no está disponible en la distribución oficial de GRASS, lo puedes bajar de las páginas del proyecto yerba[3] en la Universidad de Murcia.

5.6. Operadores de bloque

En GRASS no existe ningún módulo que implemente operadores de bloque como tales, pero pueden ejecutarse con un conjunto de órdenes similar al siguiente:

GRASS: /path > **g.region** res=*tb* <ENTER>

GRASS: /path > **g.region -p** <ENTER>

GRASS: /path > **r.mapcalc** 'bloques=col()+(row()-1)**cols*' <ENTER>

[3] www.um.es/ geograf/sigmur/yerba

GRASS: /path > **g.region** res=*r* <ENTER>

GRASS: /path > **r.statistics base**=bloques **cover==***capa* **method=**average **output**=*capa_salida* <ENTER>

donde tb es el tamaño del bloque en metros, r el tamaño de la celdilla en el mapa original (conviene que tb sea múltiplo de r) y $cols$ es el número de columnas que se obtiene fácilmente ejecutando la orden **g.region -p**. La llamada a **r.mapcalc** genera una capa en la que a cada bloque se asigna un identificador único, para cada uno de estos bloques.

Tras volver al tamaño de celdilla original, se calcula la media de la variable contenida en $capa$ y se asigna, en la capa $capa_salida$, a todas las celdillas incluidas en el bloque.

Los cambios en la resolución permiten trabajar a "escala de bloque" y a "escala de celdilla". En realidad, podría trabajarse con cualquier operador de vecindad con lo que este tipo de operadores de bloque resultan útiles para llevar a cabo cambios de escala (*upscalling*) en los datos de partida.

5.7. Operadores globales

5.7.1. r.cost

Produce un mapa del coste acumulado que conlleva viajar desde un punto del mapa a cualquier otro punto del mapa:

GRASS: /path > **r.cost -k input=** *mapa* **output=***mapa* **coordinate=***x,y* <ENTER>

La opción **-k** fuerza al programa a usar "movimientos Rey" en lugar de "movimientos de torre", de este modo se fuerza al programa a tener en cuenta posibles desplazamientos en la diagonal, es un procedimiento más lento pero más exacto. Los parámetros **input** y **output** hacen referencia al mapa de entrada y salida respectivamente.

En la siguiente tabla aparece un ejemplo de obtención de un mapa de coste acumulado desde la celdilla central a partir de un mapa de costes de entrada.

1	1	1	2	2		5	3	2	3	4
3	3	1	1	2		5	4	2	2	4
3	1	1	2	2	r.cost	5	2	1	3	4
2	1	1	1	1		4	2	2	2	3
1	2	3	3	3		3	4	5	5	5

Como puede verse, la capa resultante se asemeja a un embudo deformado, si pensamos en ella como en una capa de elevaciones vemos que las rutas de mayor pendiente marcan la ruta de mínimo coste. Este hecho es importante ya que permite utilizar el módulo **r.drain** visto anteriormente para construir rutas de mínimo coste a través de una capa de coste acumulado.

5.7.2. r.buffer

Crea zonas buffer (áreas tampón) alrededor de un objeto o conjunto de objetos. Se define como objeto en un mapa raster un conjunto de pixel adyacentes con valores no nulos. CELDILLAS CON VALOR CERO DEFINEN POR TANTO UN OBJETO.

> GRASS: /path > **r.buffer input=***mapa* **output=***mapa* **distances=***valor[,valor,...]* [**units=***nombre*] <ENTER>

El parámetro **input** es el mapa de entrada que contiene los objetos, el parámetro **output** es el mapa de salida que contiene los puntos situados entre distintos umbrales de distancia, finalmente el parámetro **units** indica las unidades en las que se calcula la distancia (la opción por defecto es meters).

5.7.3. r.distancias

Ese módulo calcula las distancia a un objeto o conjunto de objetos. Las diferencias respecto a **r.buffer** son:

- Genera un mapa de distancias (expresado en metros) y no de zonas situadas entre determinados umbrales de distancia
- Genera además un mapa en el que a cada celdilla le asigna el identificador del objeto más cercano (existen otros posibles mapas de salida pero no van a ser tratados aquí)
- Puede utilizar como entrada una capa con el coste de atravesar una determinada celdilla (parámetro **cost**), de esta manera se genera una capa de coste acumulado similar a la que se obtiene con **r.cost**
- Este módulo no está disponible en la distribución oficial de GRASS, lo puedes bajar de las páginas del proyecto yerba[4] en la Universidad de Murcia.

> GRASS: /path > **r.distancias input=***mapa* **dist_map=***mapa* **alloc_map=***mapa* <ENTER>

El parámetro **input** es el mapa de entrada que contiene los objetos, el parámetro **dist_map** es el mapa de salida que contiene los puntos situados entre distintos umbrales de distancia, finalmente el parámetro **alloc_map** indica el nombre del mapa que contendrá el identificador del objeto más cercano. El parámetro **cost** es opcional y hace referencia al mapa de coste que se va a utilizar.

En la figura 5.3 aparece en la parte alta el mapa de núcleos urbanos y el resultado de la ejecución de **r.buffer**. En la parte baja el mapa de distancias y el de núcleo más cercano resultado de la ejecución de **r.distancias**.

[4] www.um.es/ geograf/sigmur/yerba

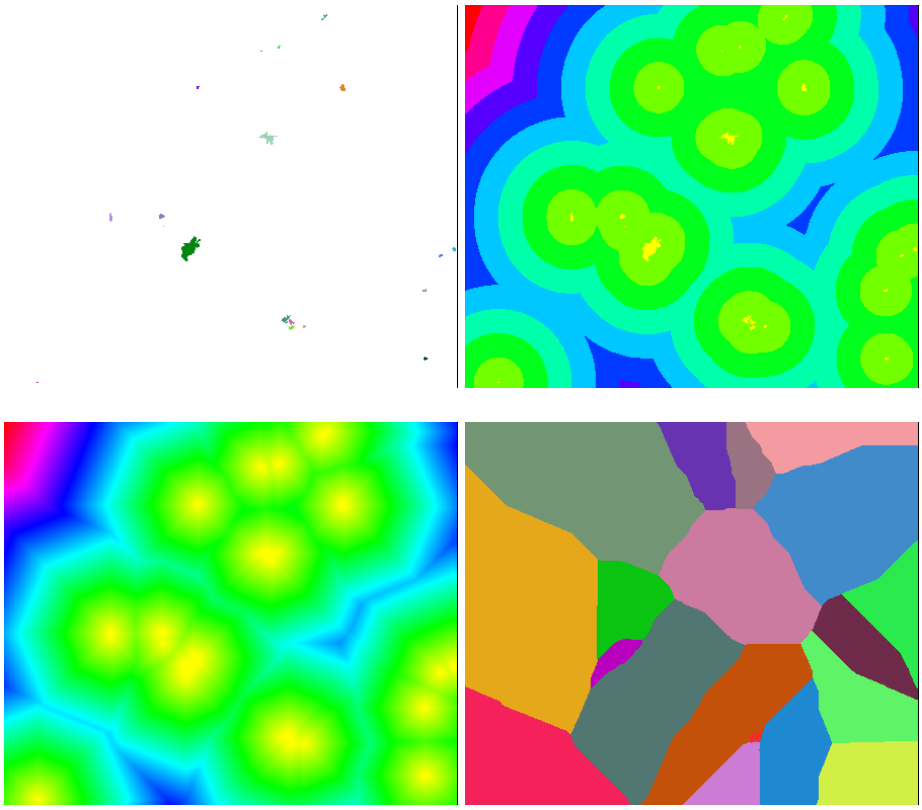


Figura 5.3: Ejemplo de salida de **r.buffer** y **r.distancias**

Capítulo 6

Algebra de mapas con r.mapcalc

El álgebra de mapas comprende un amplio conjunto de operadores cuyo objetivo es la ejecución de operaciones matemáticas, más o menos complejas, entre capas raster. El número de posibles operadores es prácticamente infinito y , lógicamente, no todos van a implementarse como módulos de un SIG. Los operadores más simples o los menos habituales no se incorporan como módulo para evitar sobrecargar el sistema de módulos. Para resolver estos casos se proporciona al usuario un lenguaje de programación de operadores de álgebra de mapas y un módulo capaz de interpretar y ejecutar estos programas. En el caso de GRASS este módulo es **r.mapcalc**.

Los programas para **r.mapcalc** admiten un amplio conjunto de funciones y operadores de tipo aritmético y lógico que permiten programar cualquier tipo de operador local o de vecindad. En esta práctica se verán sólo operadores locales debido a que los de vecindad son bastante más complejidad.

Los programas para **r.mapcalc** tienen la misma estructura que una ecuación matemática. Por ejemplo el programa:

```
mdecm=100*mde
```

Genera una nueva capa llamada *mdecm* que contiene los mismos valores que la capa *mde* pero ahora multiplicados por 100. Si *mde* es una capa de elevaciones en metros, *mdecm* será una capa de elevaciones en centímetros.

Lo que hace **r.mapcalc** es aplicar el programa (la ecuación en definitiva) a cada una de las celdillas del mapa original y asignar el resultado a la misma celdilla del mapa de salida.

6.1. Ejecución de programas con r.mapcalc

Existen tres modos de ejecución de **r.mapcalc**

- *Modo interactivo*. El usuario teclea:

GRASS: /path > **r.mapcalc**<ENTER>

y a continuación el sistema presenta un nuevo *prompt*:

mapcalc>

tras el que deberemos teclear el programa en una sola linea. Tras teclear [RETURN] el programa se ejecuta.

Este módulo tiene el inconveniente de que cualquier error al teclear el programa nos obligará a repetir todo el proceso, por lo cual es el menos recomendable.

- *Modo linea de comandos*. El usuario teclea el programa entero entrecomillado tras **r.mapcalc**. Por ejemplo:

  GRASS: /path > **r.mapcalc** 'mdecm=100*mde'<ENTER>

  Si nos hemos equivocado al introducir la orden bastará con recuperarla con la tecla [flecha hacia arriba] y editarla. El inconveniente es que programas un poco largos resultarán difíciles de editar.

- *Modo archivo*. Consiste en escribir el programa en un fichero aparte con un editor (por ejemplo emacs) y a continuación llamar al programa

  GRASS: /path > **r.mapcalc**<mde100.mc[1]<ENTER>

  De este modo pueden editarse o modificarse los programas con gran facilidad.

6.2. Operadores y funciones aritméticas en r.mapcalc

El lenguaje de programación de **r.mapcalc** admite gran número de operadores aritméticos y funciones, así como combinaciones entre diversos mapas. Los operadores aritméticos son:

- **%**. Módulo, resto de la división
- /. División
- *****. Multiplicación
- +. Suma
- -. Resta

En cuanto a las funciones, **r.mapcalc** admite varias de ellas:

[1] La extensión no es necesaria, pero resulta útil para distinguir los archivos con programas para **r.mapcalc** del resto de los archivos

- **abs()** Valor absoluto
- **float()** Conversión a como flotante
- **int()** Parte entera
- **exp()** Exponencial
- **exp(x,y)** x^y
- **log ()** Logaritmo
- **max()** Valor máximo
- **min()** Valor mínimo
- **median()** Mediana
- **round()** Redondeo
- **sqrt()** Raíz cuadrada
- **sin()**[2] Seno
- **cos()** Coseno
- **tan()** Tangente
- **atan()** Arcotangente

Las diferentes funciones y los operadores pueden combinarse a voluntad. Por ejemplo la siguiente expresión:

$$C = log(sqrt(B)/sin(A)) \tag{6.1}$$

tendría el siguiente efecto:

A						B						C				
14	12	9	8	4		400	410	390	370	350		4.41	4.58	4.84	4.93	5.59
11	9	7	4	4		350	340	320	350	340		4.58	4.77	4.99	5.59	5.58
2	1	1	1	4		300	320	310	330	345		6.21	6.93	6.92	6.95	5.58
2	2	2	1	4		290	300	305	325	340		6.19	6.21	6.22	6.94	5.58
2	2	1	1	4		285	285	300	305	350		6.18	6.18	6.9	6.91	5.59

[2]las funciones trigonométricas operan con ángulos medidos en grados

6.3. Operadores y funciones lógicas con r.mapcalc

Los operadores lógicos implican evaluar una condición y ejecutar unas acciones u otras en función de que la condición se haya cumplido o no. La función de **r.mapcalc** para la evaluación de condiciones y la posterior ejecución selectiva de acciones es **if**. Esta función sigue el esquema:

$$A = if(condicion, accion1, accion2) \tag{6.2}$$

donde A es el mapa que se va a crear, $condicion$ es la condición a evaluar, $accion1$ la función que creará el mapa A si la condición es cierta y $accion2$ la función que creará el mapa A si es falsa. Los operadores lógicos que pueden incluirse en la condición a evaluar son:

- $==$ Igual
- $!=$ No igual
- $>$ Mayor que
- $>=$ Mayor o igual que
- $<$ Menor que
- $<=$ Menor o igual que
- **&&** Y lógico
- || O lógico

La siguiente instrucción creará un mapa con valores 1 para las altitudes (mapa mde) mayores de 500 y 0 en el resto:

$$A = if(mde > 500, 1, 0) \tag{6.3}$$

En el siguiente ejemplo se trata de obtener una capa de elevaciones en el supuesto de una elevación de 300 metros del nivel del mar:

$$A = if(mde > 300, mdecm - 300, 0) \tag{6.4}$$

en este caso, el valor de 300 en el mapa *mde* indica el punto de corte. A los valores superiores se les resta 300 (ya que al subir el nivel del mar todos los valores de altitud sobre el nivel del mar se reducen) y los inferiores se transforman en cero ya que este es el valor de elevación en el nivel del mar.

350	340	320	290	280		50	40	20	0	0
335	333	310	295	275		35	33	10	0	0
335	330	320	305	270		35	30	20	5	0
320	310	305	290	280		20	10	5	0	0
325	315	310	300	290		25	15	10	0	0

Este último ejemplo refleja además como $accion1$ y $accion2$ pueden ser, no sólo un número, sino funciones, incluso otra función **if()** como en el siguiente ejemplo:

$$usosb = if(usos < 3, 1, if(usos == 3, 3, 2)) \tag{6.5}$$

El cual parte del mapa *usos* de usos del suelo con los siguiente valores (puedes utilizar **r.report** para verlos):

1	secano
2	regadío
3	improductivo
4	matorral
5	bosque

para crear un mapa *ususosb* que contiene los siguiente valores:

1	cultivo (incluye los secanos y regadíos del mapa anterior)
2	vegetación natural (incluye el matorral y el bosque del mapa anterior)
3	improductivo

6.4. Operadores lógicos con varias condiciones

En la mayor parte de los casos las condiciones a evaluar son condiciones compuestas de varias subcondiciones. Estas deben concatenarse con los operadores $\&\&$ y $||$. Por ejemplo, la instrucción

$$A = if(B > 0 \ \ \&\& \ \ C < 500, 1, null()) \tag{6.6}$$

creará una capa A que contendrá unos en aquellas celdillas en que B sea mayor de 0 y C menor de 500 y *NULL* en el resto.

Este tipo de operadores multicondicionales se utiliza en problemas de evaluación multicriterio, es decir cuando se requiere extraer las zonas que cumplen un conjunto complejo de criterios que las hacen adecuadas para determinadas actividades, por ejemplo la instrucción:

$$balsa = if(usos == 2 \ \ \&\& \ \ mde > 300, 1, null()) \tag{6.7}$$

seleccionará zonas de regadío con elevación mayor de 300 metros como los lugares ideales para, por ejemplo, la construcción de una balsa de riego.

usos mde balsa

4	4	4	4	4		400	410	390	370	350		NULL	NULL	NULL	NULL	NULL
1	4	3	4	4		350	340	320	350	340		NULL	NULL	NULL	NULL	NULL
2	1	1	1	4		300	320	310	330	345		NULL	NULL	NULL	NULL	NULL
2	2	2	1	4		290	300	305	325	340		NULL	NULL	1	NULL	NULL
2	2	1	1	4		285	285	300	305	350		NULL	NULL	NULL	NULL	NULL

6.5. Valores nulos

En las primeras versiones de GRASS no existía una valor para indicar *ausencia de valor* y se debía utilizar para ello el 0. En las más recientes se ha introducido el valor *NULL* para distinguir entre valor 0 y ausencia de valor. En general es preferible utilizar *NULL* cuando el cero no represente un valor real de una variable concreta, así la anterior instrucción se codificaría como:

$$A = if(mde > 50000, 1, null()) \tag{6.8}$$

ya que **null()** es la función que genera un valor *NULL*.

Por otra parte hará falta una función especial para determinar si un valor es nulo o no (si es nulo no tiene sentido preguntar si es mayor o menor que un número ya que la respuesta estará indeterminada). La función para ello es **isnull()**. Por ejemplo si se quiere obtener un mapa con la elevación de los diferentes núcleos urbanos y con valor NULL en el resto la fórmula sería:

$$mde_urb = if(isnull(urbanos), null(), mde) \tag{6.9}$$

de manera que en las celdillas con valor NULL en el mapa de núcleos urbanos se mantuviera dicho valor y en aquellas con valor no nulo se tomará el valor del mapa de elevación (mde).

6.6. Otras funciones

Existen una serie de funciones que nos permiten incluir la posición de cada celdilla individual en los cálculos, así como tener en cuenta la resolución de la capa raster:

- **x()** devuelve la coordenada X
- **y()** devuelve la coordenada Y
- **col()** devuelve la columna

- **row()** devuelve la fila
- **ewres()** devuelve la resolución este-oeste
- **nsres()** devuelve la resolución norte-sur

Finalmente la función **eval** permite incluir una serie de cálculos intermedios sin necesidad de hacer varias llamadas a **r.mapcalc**. Por ejemplo:

$$mapa = eval(a = mde/500, b = a * a) \tag{6.10}$$

dará a mapa el valor de $mde/500^2$. Esta opción resulta útil cuando los cálculos son complejos.

Como ejemplo de las funciones vistas en este apartado, el programa:

$$f = eval(xx = (x() - 636000)/1000, yy = (y() - 4185000)/1000, \tag{6.11}$$

$$-250 * xx + yy + 2 * xx * xx + 3 * yy * yy + 0{,}5 * exp(xx, 3) \tag{6.12}$$

da lugar a la superficie matemática que aparece en la figura 6.1.

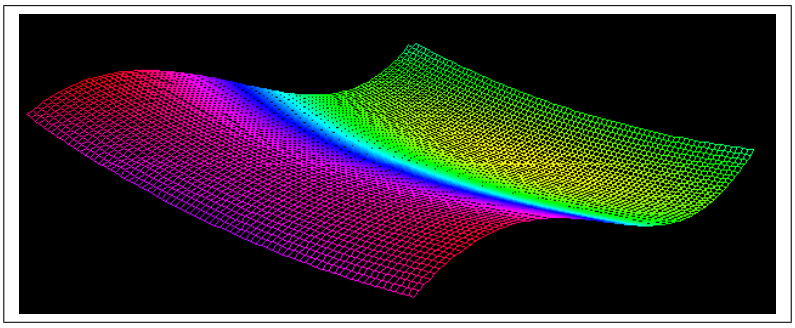


Figura 6.1: Superficie matemática generada con r.mapcalc

y que resulta de la siguiente función polinómica ($Z = -250 * x() + y() + 2 * exp(x(), 2) + 3 * exp(y(), 2) + 0{,}5 * exp(x(), 3)$ donde x e y son coordenadas relativas en kilómetros respecto al punto $X = 636, Y = 4185$.

6.7. Cuestiones

1. Modifica el mapa de elevaciones para que sus valores estén en metros en lugar de en centímetros

2. Sabiendo que el mapa de pendientes está expresado en tantos por ciento (es decir $100 * tan(s)$ donde s es la pendiente en grados), calcular un mapa del coseno de la pendiente

3. Con r.mapcalc crea un mapa que cumpla:

   - Pendiente baja o moderada (según los criterios anteriores)
   - El uso del suelo debe ser matorral o bosque
   - Estar a menos de 500 m de la carretera

4. Se sabe que la temperatura aumenta, por termino medio, $0{,}65^oC$ cada 100 metros. Suponiendo que la temperatura a nivel del mar es de 20^oC crea un mapa de temperaturas

5. En un episodio de precipitación determinado, la cantidad de lluvia aumentó desde la costa hacia el interior de la Región de Murcia a razón de:

$$\frac{\delta P}{\delta X} = 0{,}5mm/Km \tag{6.13}$$

$$\frac{\delta P}{\delta Y} = 0{,}75mm/Km \tag{6.14}$$

$$\tag{6.15}$$

   Sabiendo que en Totana cayeron 24 mm de precipitación genera un mapa de precipitación para toda la zona de trabajo.

Capítulo 7

Modelización Cartográfica. Resolución de problemas con SIG raster

El conjunto de operaciones de álgebra de mapas en un SIG está constituido por un gran número de pequeñas herramientas capaces de ejecutar operaciones concretas sobre una o varias capas de información para generar uno o varias nuevas capas. En definitiva, cada operador constituye un **componente** del **sistema** de información que transforma una serie de **entradas** de información en una serie de **salidas**. Las entradas y salidas no son sólo capas espaciales sino que también pueden ser tablas o visualizaciones en pantalla o impresora.

En ese capítulo del tutorial se va a trabajar con varios ejemplos de problemas espaciales concretos que pueden resolverse mediante modelización cartográfica utilizando herramientas de álgebra de mapas. Para ello deberemos utilizar varios módulos combinados (que pueden corresponderse o no con operadores de álgebra de mapas) para transformar un conjunto de mapas de entrada y obtener la solución. Esta solución puede consistir en un único dato, una tabla de datos o, lo más habitual, una o más capas de salida.

Utilizaremos sólo los módulos que se han visto anteriormente, pero en un caso real pueden llegar a necesitarse muchos más, algunos de los cuales no se han visto todavía y otros no van a llegar a verse en este curso.

La resolución de estos problemas pasará por una fase de **conceptualización** (creando diagramas de flujo) y otra de **formulación** (en la que el diagrama se transforma en un algoritmo formado por un conjunto de órdenes. En los diagramas de flujo las diferentes capas de información se representan mediante rectángulos en los que se escribe el nombre del mapa y una letra (r, v o s) indicando el tipo de capa de que se trata (raster, vectorial o sites) y las funciones mediante óvalos en los que se escribe el nombre del módulo de GRASS correspondiente.

7.1. Localización de áreas óptimas

Esta es una de las aplicaciones básicas de un SIG en ordenación territorial, se trata de buscar aquellas áreas que cumplan con un conjunto de condiciones que las hacen apropiadas para la instalación de una determinada actividad.

Se requiere la selección de áreas con una elevación inferior a 200 m y pendiente inferior a 5 %. Estas áreas deben a su vez estar a menos de 100 m de una vía de comunicación y no estar en cultivo ni ser casco urbano

Para resolver este problema se dispone de las siguientes capas de información:

- Un mapa vectorial de carreteras (**carreteras**)
- Un mapa raster de usos del suelo (**usos**)
- Un mapa raster de elevaciones (**mdecm**)
- Un mapa raster de pendientes (**pendiente**)

El resultado debe ser una capa raster en la que las celdillas que cumplen el conjunto de condiciones aparezcan con un valor no **NULL** y las que no lo cumplen aparezcan como **NULL**. Para resolverlo debemos partir de esta capa resultado y determinar cual es la herramienta adecuada para obtenerla, sin duda se trata de un programa para **r.mapcalc** (herramienta **MAP-calc** en **wxGRASS**).

Este programa incluye tres condiciones referidas a cuatro variables diferentes lo que implica cuatro capas de información:

- Elevaciones
- Pendiente
- Usos de suelo
- Distancia a la carretera

De estas cuatro capas sólo nos falta el mapa de distancias a la carretera, para obtener este necesitamos los módulos **r.buffer** o **r.distancias** y un mapa raster de carreteras. Puesto que el mapa de carreteras es vectorial habrá que transformarlo en raster con **v.to.rast**. Por tanto el problema se resuelve en tres fases tal como muestra el diagrama de flujo que aparece en la figura 7.1.

1. Conversión de la capa de carreteras de formato vectorial a formato raster

GRASS: /path > **v.to.rast input=**carreteras **output=**carreteras <ENTER>

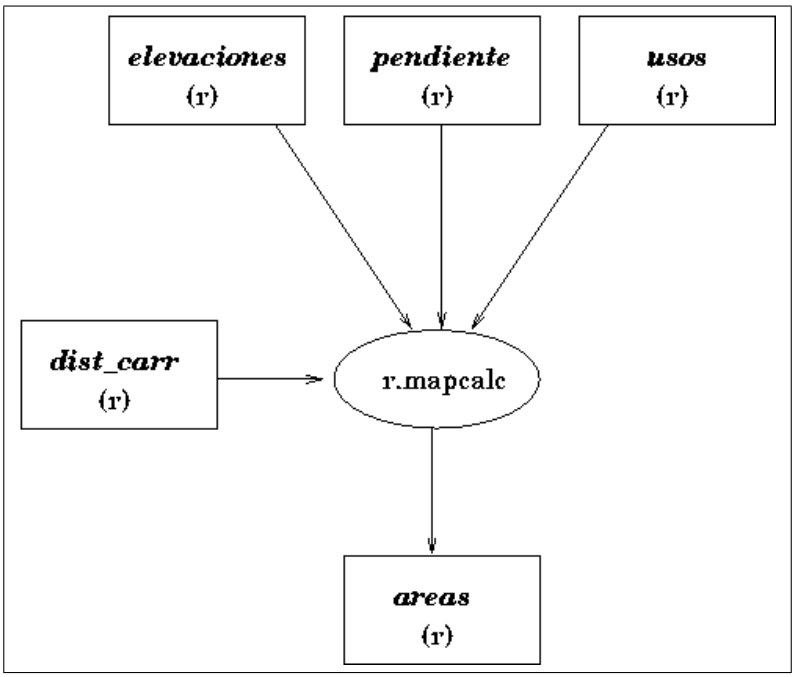


Figura 7.1: Primera aproximación a la resolución del primer problema planteado

El resultado es una capa en formato raster en la que las celdillas atravesadas por una carretera aparecen con el identificador asociado a esa carretera y el resto con valor NULL.

2. Obtención de una capa raster con las distancias a la carretera

   GRASS: /path > **r.distancias input=**carreteras **dist_map=**dist_carr **alloc_map**=alloc_carr <ENTER>

3. Integración de las 3 capas de información con **r.mapcalc** para seleccionar las áreas que cumplen las tres condiciones

   GRASS: /path > **r.mapcalc 'area=if(**mdecm <20000 **&&** pendiente <5 **&&** dist_carr<100 **&&** usos >3,1,null())' <ENTER>

   Cuando se incluye una variable cualitativa como el uso del suelo hay que trabajar con los identificadores numéricos no con las etiquetas de texto. En este caso los suelos cultivados corresponden a los identificadores 1 y 2 y las zonas urbanas al 3; por tanto las zonas válidas serán aquellas cuyo identificador sea mayor de 3.

La figura 7.2 muestra el esquema completo de su resolución.

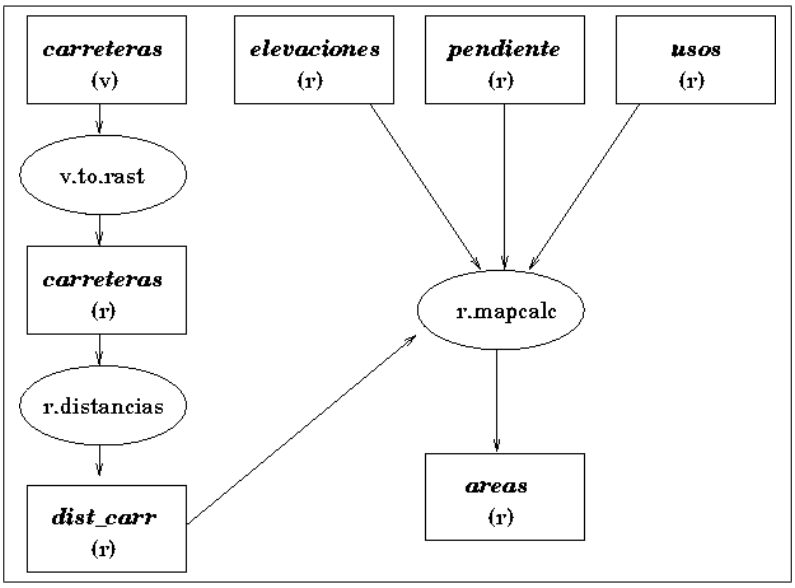


Figura 7.2: Esquema de resolución del primer problema planteado

Se busca un área adecuada para la construcción de un vertedero. Debe ser un polígono de matorral, no situado sobre un acuífero. Al mismo tiempo este campo debe estar como mucho a 1 Km de distancia de una carretera y lo suficientemente alejada (5 Km) de un espacio natural protegido. Ese polígono debe además tener un tamaño mínimo (10 Has), una pendiente media menor de 5 % y la pendiente máxima no puede superar el 10 %

Se parte para ello de los siguiente mapas:

- Un mapa vectorial con los acuíferos **acuiferos**
- Un mapa raster de usos del suelo **usos**
- Un mapa vectorial de carreteras **carreteras**
- Un mapa raster con los espacios protegidos **espacios**
- Un mapa raster de pendientes **pendiente**

Según el planteamiento del problema tenemos 5 condiciones directamente extraíbles de los mapas raster disponibles:

1. El uso de suelo debe ser matorral

2. En la capa de acuíferos el valor de las celdillas debe ser NULL
3. La distancia a las carreteras debe ser menor o igual a 1000 m
4. La distancia a los espacios protegidos debe ser mayor de 5000 m
5. Pendiente menor o igual a 10 %

y dos condiciones que no se refieren a las celdillas sino a las agrupaciones de celdillas válidas consideradas como polígonos (objetos) individuales:

1. La pendiente media debe ser inferior a 5 %
2. El tamaño mínimo es de 10 Has.

Para resolver este tipo de problemas deben en primer lugar resolverse las condiciones referidas a las celdillas mediante un programa de mapcalc:

> GRASS: /path > **r.mapcalc 'area=if(**usos ==4 **&&** isnull(acuiferos) **&&** dist_carr<=1000 **&&** dist_esp>5000 **&&** pendiente <=10 ,1,null())' <ENTER>

Las otras condiciones no hacen referencia a las celdillas sino a los grupos de celdillas contiguas (polígonos) que han cumplido las condiciones anteriores. Por tanto el siguiente paso será convertir la capa resultante de la ejecución anterior de **r.mapcalc** que contiene una variable cualitativa (1 si la celdilla cumple las condiciones y **NULL** si no las cumple) en una capa de polígonos en la que cada uno de ellos tenga un identificador único:

> GRASS: /path > **r.clump input=**areas **output=**poligonos <ENTER>

Cada uno de estos polígonos deberá ser ahora analizado mediante operadores de área para determinar su extensión (**r.areas.sh**) y su pendiente media (**r.statistics**)

> GRASS: /path > **r.areas input=**poligonos **output=**polig_ext **units=**h <ENTER>

> GRASS: /path > **r.statistics base=**poligonos **cover=**pendiente **method=**average **output=**pend_med <ENTER>

Finalmente debemos verificar cual de los polígonos cumplen con estas condiciones mediante una nueva llamada a **r.mapcalc**

> GRASS: /path > **r.mapcalc 'final=if(**pend_med<5 **&&** polig_ext>10,poligonos,null())' <ENTER>

De este modo el mapa resultante (**final**) será igual al mapa **poligonos** salvo que ahora sólo se representan aquellos polígonos que. además de cumplir las condiciones referidas a las celdillas, cumplen también las relativas a los polígonos.

En este caso concreto habría sido además necesario obtener algunos de los mapas a los que se refieren las condiciones. En primer lugar habría que rasterizar los vectoriales y calcular el mapa de distancias a carreteras, si hemos resuelto el ejemplo anterior bastará con rasterizar el mapa de acuíferos:

GRASS: /path > **v.to.rast input=**acuiferos **output=**acuiferos <ENTER>

Por otro lado deberemos obtener un mapa de distancias a espacios protegidos:

GRASS: /path > **r.distancias input=**espacios **dist_map=**dist_esp **alloc_map**=alloc_esp <ENTER>

con lo que ya dispondríamos de todos los mapas para la ejecución del primer programa de **mapcalc**. El conjunto de todas las operaciones se refleja en la figura 7.3

7.2. Cálculo de la población amenazada por un accidente químico

Se supone que debemos organizar el transporte de una carga de material químico peligroso. En previsión de accidentes, debe determinarse la cantidad de población que debería ser evacuada en tal eventualidad. Se asume que esta incluye a todos los habitantes que viven a menos de 2 Km de la carretera que va a ser utilizada para el transporte

Para resolverlo disponemos de los siguientes mapas:

- **transporte** Capa vectorial con la ruta que seguirá el transporte
- **urbanos** Mapa de núcleos urbanos y población total

En la figura 7.4 aparece el diagrama de flujo de la solución a este problema.

En primer lugar hay que rasterizar la ruta del transporte y calcular las distancias.

GRASS: /path > **v.to.rast input=**transporte **output=**transporte <ENTER>

GRASS: /path > **r.distancias input=**transporte **dist_map=**distancia **alloc_map**=alloc <ENTER>

Si representamos el mapa de distancias y le superponemos el de núcleos urbanos:

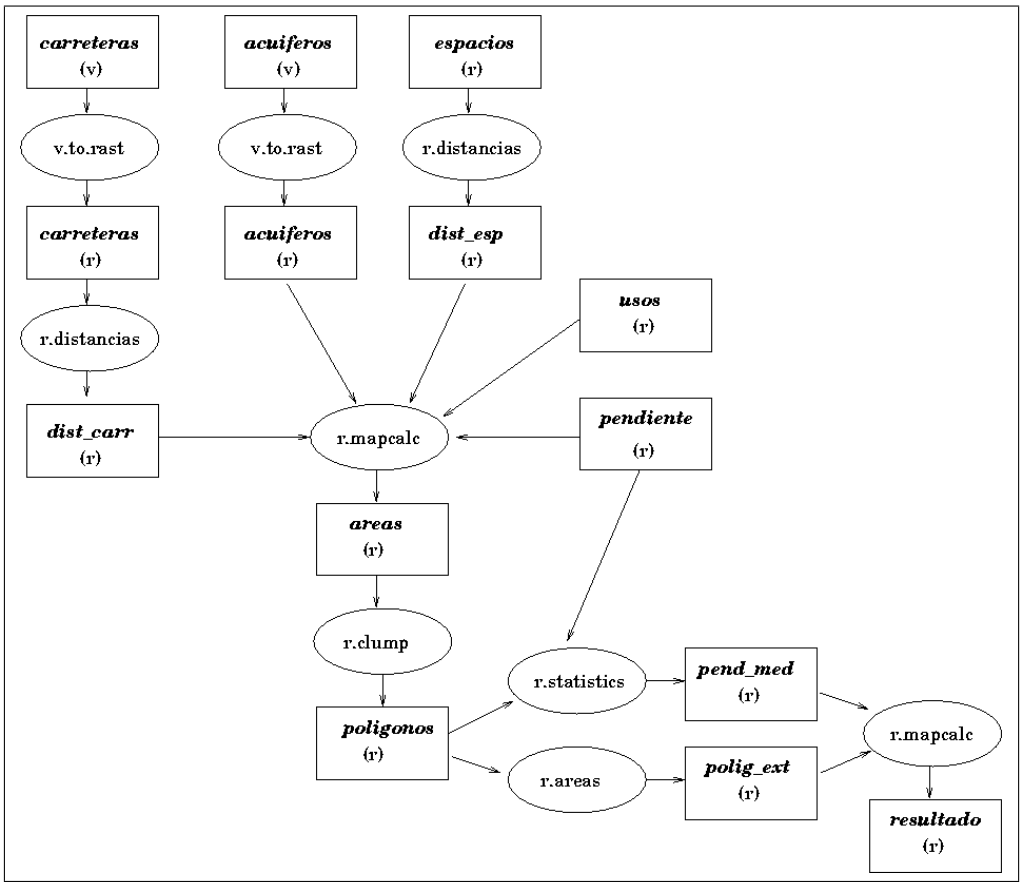


Figura 7.3: Esquema de resolución del segundo problema planteado

GRASS: /path > **d.rast** distancia <ENTER>

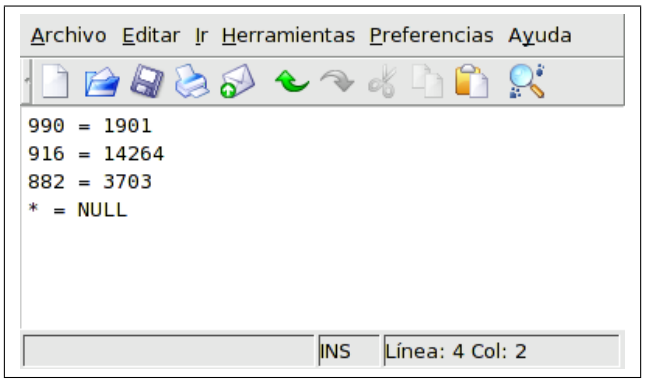
GRASS: /path > **d.rast -o** urbanos <ENTER>

podremos identificar que núcleos de población pueden ser afectados utilizando el módulo **d.what.rast**. Puesto que en las etiquetas de texto asociadas a los núcleos urbanos aparece la población de cada uno de ellos obtenemos una primera aproximación a la población potencialmente afectada.

Puede utilizarse **r.reclass** para obtener un mapa en el que a cada núcleo de población se le asigne su población total:

GRASS: /path > **r.reclass input=**urbanos **output=**pob<rec_pob.txt <ENTER>

El contenido del fichero rec_pob.txt aparece en la siguiente figura:



El mapa **pob** asigna a cada celdilla el total de población residente en el núcleo urbano al que pertenece la celdilla. Un enfoque más preciso al problema que se plantea sería hacer una estimación del número de habitantes por celdilla, para esto basta con utilizar **r.areas** y dividir la población total por el número de celdillas.

GRASS: /path > **r.areas input**=urbanos **output**=urb_area **units**=c <ENTER>

GRASS: /path > **r.mapcalc 'pob2=pob/urb_area'** <ENTER>

Para terminar con el ejercicio es necesario una nueva llamada a **r.mapcalc** para crear un mapa en el que aparezca la población estimada en cada una de las celdillas afectadas por el riesgo:

GRASS: /path > **r.mapcalc 'riesgo=if(distancia<2000,pob2,null())'** <ENTER>

Una llamada a **r.report** nos dará el número de celdillas afectadas y el número de habitantes por celdilla afectada.

7.3. Selección de rutas óptimas

Determinar cual es la ruta óptima entre las localidades de *Ermita de Santa Leocadia* **y** *Librilla*

Ermita de Sta. Leocadia:	**623585,4186592**
Librilla:	**645119, 4194828**

asumiendo que el coste de atravesar cada celdilla es:

- **si la pendiente es mayor que 40 grados o el uso urbano, el coste es infinito (celdilla no apropiada para carretera)**

- **en otro caso el coste es igual a la tangente de la pendiente por el factor C_u que se presenta en la siguiente tabla:**

Secano:	**1**
Regadio:	**3**
Matorral:	**3**
Bosque:	**4**
Improductivo:	**0**

Una de las aplicaciones clásicas de los SIG es la obtención de rutas óptimas entre dos puntos en función de diversos criterios. La resolución de este tipo de problemas con GRASS se basa fundamentalmente en los módulos **r.cost** y **r.drain**.

El esquema de resolución de este tipo de problemas es el siguiente:

1. Elaborar un mapa con el coste de atravesar cada celdilla
2. Hacer, con **r.cost**, un mapa con el coste total de llegar desde cada celdilla individual a la celdilla destino
3. Dejar que **r.drain** determine cual es la ruta apropiada desde el punto de partida al punto de destino (aunque **r.drain** es un módulo destinado en principio a aplicaciones topográficas puede utilizarse con cualquier tipo de variable)

Comenzaremos por hacer un mapa de coste debido al uso de suelo:

GRASS: /path > **r.reclass input=**usos **output=**coste_suelo<rec_coste.txt <ENTER>

para a continuación crear el mapa de coste total:

GRASS: /path > **r.mapcalc ' coste=if(pendiente<40,tan(pendiente)*coste_suelo,null())'** <ENTER>

en la práctica se sustituye el coste infinito por un valor nulo lo que imposibilita el que la futura ruta pase por esa celdilla.

Una vez que se dispone de este mapa de costes es sencillo crear un mapa de coste acumulado desde, por ejemplo, Librilla:

GRASS: /path > **r.cost input=**coste **output=**coste_acu **coordinate=**645119,4194828 <ENTER>

El resultado es un mapa que si lo viéramos en 3 dimensiones sería algo así como un embudo deformado con su punto más bajo en Librilla. La ruta de coste mínimo desde cualquier punto de este **embudo** hasta el punto más bajo será la de pendiente máxima, por lo que resulta adecuado utilizar el módulo **r.drain**.

> GRASS: /path > **r.drain input=**coste_acu **output=**ruta **coordinate=**623585,4186592 <ENTER>

Finalmente queda la opción de transformar el mapa resultante a formato vectorial lo que nos facilitará la visualización

> GRASS: /path > **r.line input=**ruta **output=**ruta <ENTER>

La figura 7.5 muestra el esquema de resolución de este problema.

7.4. ¿Cómo hacerlo con wxGRASS?

En wxGRASS los esquemas de resolución de problemas presentados en este capítulo son perfectamente válidos, la única diferencia es que en lugar de introducir las órdenes directamente en la linea de comandos se construyen introduciendo los parámetros correspondientes en una ventana de diálogo activada al pulsar una opción de menú.

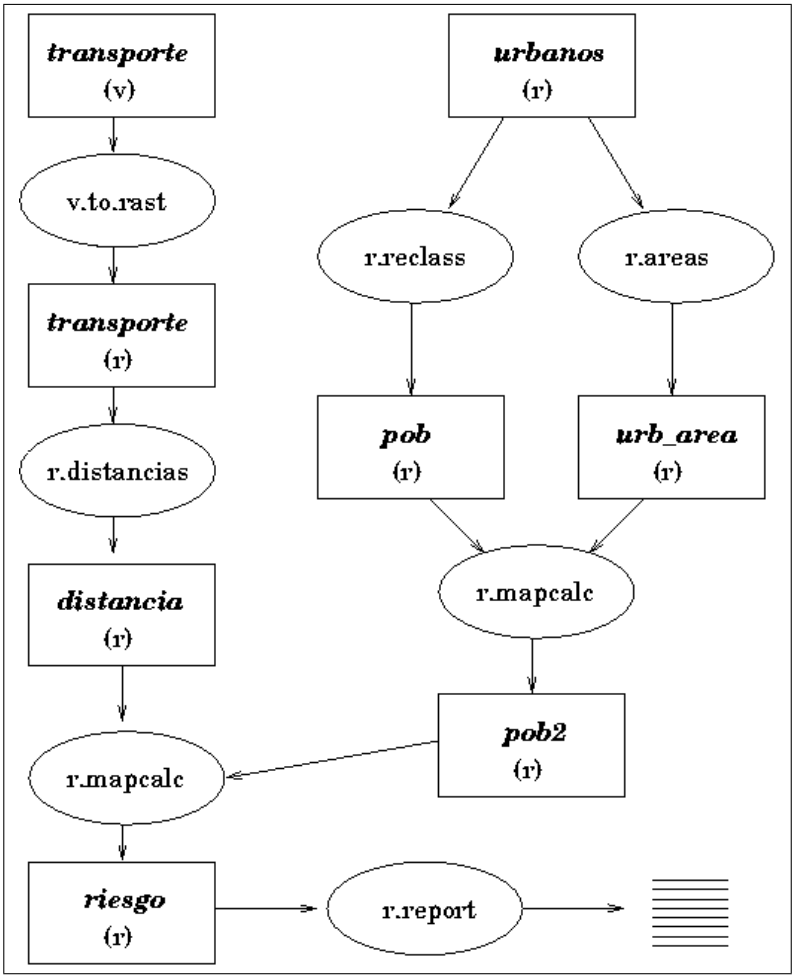
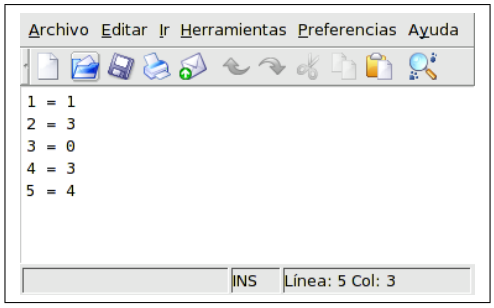


Figura 7.4: Esquema de resolución del tercer problema planteado



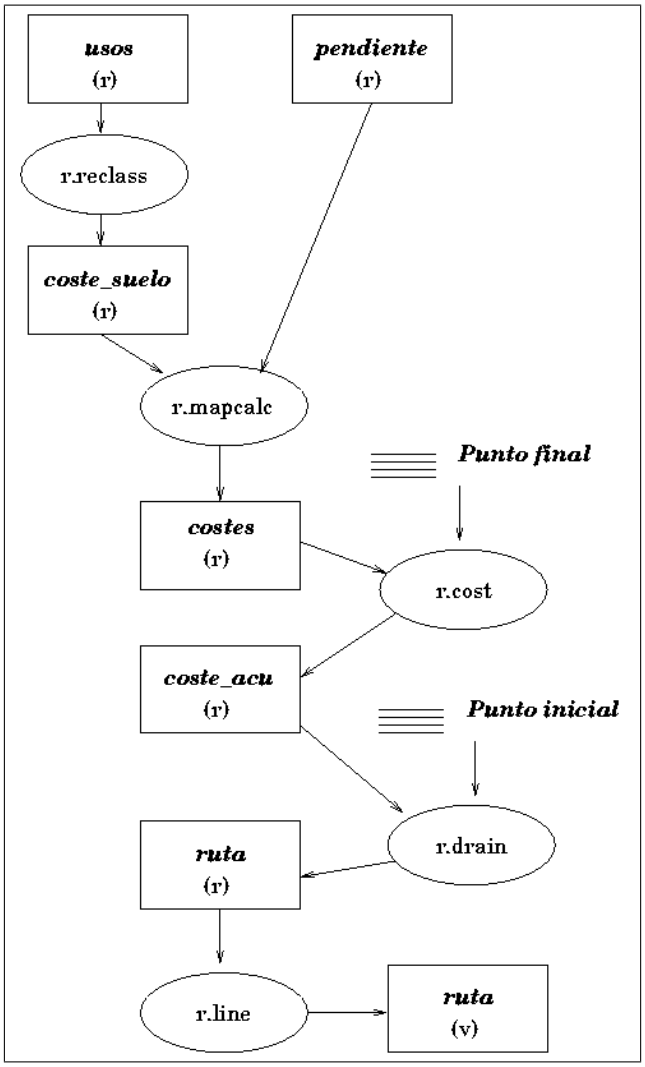


Figura 7.5: Resolución del cuarto problema planteado

Capítulo 8

Interpolación

Todo programa de SIG dispone de un conjunto de módulos o funciones de interpolación. El objetivo de la interpolación espacial es obtener una capa raster de una variable Z a partir de sus valores conocidos en un conjunto de puntos de muestreo. Por tanto se hace necesario estimar el valor de $Z_{x,y}$ para cada celdilla con coordenadas x, y.

Existen infinidad de métodos de interpolación, muchos de los cuales se encuentran implementados en GRASS, cuyo comportamiento puede además modificarse dando valores a un conjunto de parámetros que se le pasan al módulo que ejecuta el método. La consecuencia es que el número de métodos es virtualmente infinito.

Es importante tener en cuenta que diferentes métodos van a generar diferentes superficies (a veces muy diferentes) y que no siempre es fácil decidir cual es el método más adecuado. Por todo ello llevar a cabo una interpolación espacial es mucho más complejo de lo que parece debido a la sencillez de manejo de los módulos que la llevan a cabo.

Los métodos que pueden utilizarse en GRASS son:

- Métodos globales
  - Interpolación por clasificación
  - Interpolación por regresión
- Métodos locales
  - Media ponderada por el inverso de la distancia elevado a un exponente
  - Splines
  - Kriggeado
  - Interpolación a partir de TIN (Red Irregular de Triángulos)
- Interpolación a partir de curvas de nivel

En algunos casos es necesario utilizar programas específicos enlazados con GRASS (por ejemplo *gstat* para interpolar por kriggeado).

8.1. Los ficheros de sites

En GRASS, junto a los conocidos formatos raster y vectorial, existe un tercer formato menos utilizado y que desaparecerá en las próximas versiones del programa. Se trata de las listas de *sites* que son tablas en las que para un conjunto de puntos (*sitios*) en el espacio se codifican una serie de propiedades. Son ficheros en formato ASCII cuya estructura es similar a:

```
name|pluv10_90
desc|s.in.ascii sites=pluv10_90 d=3 fs=space
644562|4194552|168|#1 %24 @Observatorio 1
639123|4191098|210|#2 %32.1 @Observatorio 2
677053|4162161|219|#3 %31.7 @Observatorio 3
668978|4187402|160|#4 %32 @Observatorio 4
632857|4213906|390|#5 %36.2 @Observatorio 5
652874|4210090|110|#6 %28 @Observatorio 6
602054|4198168|780|#7 %14 @Observatorio 7
599123|4189050|645|#8 %19.5 @Observatorio 8
  ...    ...  ..  ..   ...
599625|4218960|629|#44 %31.5 @Observatorio 96
613600|4214160|604|#45 %34.5 @Observatorio 97
632258|4204839|381|#46 %28 @Observatorio 98
660683|4207830|67|#47 %27.6 @Observatorio 99
664329|4205947|57|#48 %30.5 @Observatorio 100
```

Las dos primeras filas están ocupadas por el nombre y la descripción del fichero (en este caso el comando con el que se creó). A continuación aparece una lista en la que cada fila representa un sitio. En este caso cada sitio se define por 3 coordenadas (X, Y y Z), a continuación aparece un identificador único precedido del carácter **#** y una serie de variables. Las variables numéricas se preceden por el carácter **%** y las de texto por **@**.

Los ficheros de sites pueden tener dos o tres coordenadas (X,Y en el primer caso y X,Y,Z en el segundo). Algunos módulos sólo funcionan con ficheros de sites de dos coordenadas.

Resulta evidente que esta tabla podría almacenarse de forma más eficiente en una Sistema de Gestión de Bases de Datos enlazada al SIG[1] y de hecho es lo que suele hacerse (en las últimas versiones de GRASS, desde la 5.7, por ejemplo). Sin embargo muchos módulos de GRASS siguen trabajando con ficheros de sites, especialmente los de interpolación y esa es la razón por la que se presentan aquí.

[1] más adelante se verá como manejar bases de datos y SIG de forma conjunta

8.2. Interpolación por clasificación

Si se dispone de una capa de *sitios* que contiene puntos de muestreo para interpolar por el método de clasificación, lo más adecuado es rasterizar esos puntos con el comando **s.to.rast** y utilizar el comando **r.statistics** para obtener una capa con las medias de los valores muestreados para los diferentes valores de una variable cualitativa que se utiliza como variable de apoyo en la clasificación y de la que también se dispone de capa.

Por ejemplo, el mapa de puntos *mo* contiene un conjunto de puntos de muestro con el contenido en materia orgánica (expresado en tantos por ciento) en el área de trabajo. Utilizando este mapa y el mapa de suelos como entrada a **r.statistics** se consigue un mapa con la estimación del contenido en materia orgánica interpolado por clasificación para cada uno de los tipos de suelo.

GRASS: /path > **s.to.rast input=**mo **output=**mo <ENTER>

GRASS: /path > **r.mapcalc** 'mo100=int(mo*100)' <ENTER>

GRASS: /path > **r.statistics base=**suelos **cover=**mo100 **method=**average **output=**mo100_av <ENTER>

GRASS: /path > **r.mapcalc** 'mo_av=mo100_av/100' <ENTER>

Recuerda que **r.statistics** necesita que el mapa cover contenga números enteros, por ello se ha transformado el mapa original, multiplicándolo por 100, con una llamada a **r.mapcalc**, al final hay que deshacer el cambio.

La adecuación del procedimiento de clasificación puede comprobarse generando mapas de desviación típica, mínimo y máximo con **r.statistics** para determinar si las medias para los diferentes valores de la variable cualitativa son significativamente diferentes.

Un procedimiento alternativo, si no se tienen datos puntuales pero se dispone de valores adecuados de la variable cuantitativa a interpolar para las diferentes clases en la variable de apoyo (por ejemplo, podríamos haber encontrado en la bibliografía una tabla con los valores medios de materia orgánica en los diferentes tipos de suelo de la Región de Murcia), es utilizar **r.reclass** para reclasificar el mapa de la variable cualitativa y asignar a cada clase su valor correspondiente.

8.3. Interpolación por regresión

Con **r.mapcalc** puede programarse cualquier ecuación de regresión para una variable dependiente respecto a una o varias variables independientes, por ejemplo:

GRASS: /path > **r.mapcalc '**$V_1 = a + b * V_2$**'**<ENTER>

Asume que hay una variable independiente V_2 a partir de la que puede estimarse V_1 mediante los coeficientes de regresión a y b.

En muchos modelos de regresión espacial las variables independientes van a ser las coordenadas espaciales (X UTM e Y UTM por ejemplo). En **r.mapcalc** se dispone de funciones que devuelven el valor de estas variables, por ejemplo:

GRASS: /path > **r.mapcalc '**$V_1 = a + b * x() + c * y()$**'**<ENTER>

Tanto en el caso de la interpolación por regresión como en el de la clasificación, el análisis estadístico hay que hacerlo previamente, lo más adecuado es utilizar un programa de estadística para ello.

El fichero de sites **pluv10_90** contiene información acerca de la precipitación total en octubre de 1990 en los observatorios de la Región de Murcia. A partir de un análisis estadístico se ha comprobado que la precipitación tiende a aumentar con la coordenada Y según el siguiente modelo de regresión: $P = 0{,}0009829 * Y - 0{,}004088$ o sea casi un litro por metro cuadrado por cada kilómetro que nos desplazamos hacia el Norte. Este modelo de regresión puede transformarse en un modelo de interpolación con la siguiente orden:

GRASS: /path > **r.mapcalc '**$P = 0{,}0009829 * y() - 0{,}004088$**'**<ENTER>

8.4. Cálculo del semivariograma

Se han desarrollado diversos proyectos para la implementación de la interpolación por kriggeado en GRASS, el más desarrollado en la actualidad implica la utilización de un programa externo (gstat, www.gstat.org/) por lo que, por simplicidad no se va a entrar en detalle y nos limitaremos a estudiar el módulo **s.sv2svfit** que calcula la función semivariograma y le ajusta un modelo seleccionado por el usuario.

GRASS: /path > **s.sv2svfit -p -w sites=**pluv10_90 **lag=**2000 **range=**20000 **model=**3 <ENTER>

El parámetro **input** es el fichero de sites a partir del que se va a hacer la interpolación, **model** es el modelo de semivariograma que se va a ajustar, el modelo puede ser:

- 1 Lineal
- 2 Esférico
- 3 Exponencial
- 4 Gaussiano

- 5 Cuadrático
- 6 Efecto agujero

el **rango** define el parámetro de alcance, es decir la distancia máxima entre puntos para la que se considera que todavía hay influencia de uno a otro, finalmente el parámetro **lag** define los intervalos significativos de distancia. Así la siguiente orden:

> GRASS: /path > **s.sv2svfit** -p -w **sites=**pluv10_90 **model=**3 **lag=**2000 **range=**20000 <ENTER>

calcula la función semivariograma utilizando como distancia significativa 2 Kilómetros lo significa que, por ejemplo, todos los pares de puntos separados entre 1000 y 3000 metros se utilizarán para el cálculo de $\gamma(2000)$, los pares separados entre 3000 y 5000 metros para calcular $\gamma(4000)$, etc. Una vez calculada la función se ajustará el modelo exponencial con un rango de 20000 metros.

La opción **-p** le indica al programa que debe devolvernos una representación gráfica del resultado (para ello debes tener instalado en tu ordenador el programa gnuplot). A partir de esta representación podremos, mediante prueba error, ajustar mejor el modelo.

8.5. Inverso de la distancia

Este método hace una estimación del valor de cada una de las celdillas como la media ponderada de los valores medidos en un conjunto de puntos de muestreo situados alrededor. Los coeficientes de ponderación son inversamente proporcionales a la distancia elevada a un exponente p.

En GRASS se dispone del módulo **r.surf.idw** que utiliza un valor de $p = 2$. El usuario puede determinar el número de puntos de muestreo alrededor que se van a tener en cuenta a la hora de hacer la interpolación (el valor por defecto es 12). La siguiente orden genera un mapa interpolado de la variable precipitación cuyas medidas puntuales aparecen en el mapa pluviómetros utilizando los 10 pluviómetros más cercanos a cada celdilla a interpolar.

> GRASS: /path > **r.surf.idw input=**pluv10_90 **output=**precipitacion **npoints=**10 <ENTER>

Si observamos la figura 6.3 del temario, verás el gráfico de la función semivariograma para este mapa de sitios. Como puedes ver la diferencia entre puntos situados a menos de unos 20000 metros tiende a decrecer con la distancia mientras que por encima de esta distancia es más o menos constante. Por tanto parece más adecuado definir esos 20000 metros como distancia umbral que seleccionar un número de puntos para interpolar.

8.6. splines

Este método simula la adaptación de una membrana flexible a un conjunto de valores de Z en el espacio. Es el método más general y exacto de que dispone GRASS (Neteler & Mitasova), es además uno de los módulos de GRASS que mayor desarrollo ha experimentado en los últimos años, pero requiere un cuidadoso ajuste de sus parámetros para conseguir la máxima exactitud.

El módulo **s.surf.rst** lleva a cabo este tipo de análisis y, opcionalmente, calcula un conjunto de parámetros topográficos de la superficie generada. Existe una versión de este módulo para interpolación en 3 dimensiones (**s.vol.rst**) y un módulo experimental (**s.volt.rst**) para interpolación en 4 dimensiones. No existe en GRASS una versión similar que utilice como entrada datos raster, aunque si existe un módulo **v.surf.rst** para hacer interpolación a partir de isolineas.

Puesto que se utilizan sites, es posible seleccionar cual es la columna que contiene la variable a interpolar mediante el parámetro **field**, en el ejemplo que se muestra a continuación no es necesario ya que el fichero de sites sólo contiene un campo.

A pesar de la necesidad de ajustar apropiadamente los parámetros, una primera aproximación a la interpolación por splines en dos dimensiones puede hacerse dejando las opciones por defecto de todos los parámetros.

GRASS: /path > **s.surf.rst input=**pluv10_90 **elev=**precipitacion <ENTER>

8.7. Interpolación a partir de TIN

Este tipo de interpolación en GRASS requiere dos pasos:

1. Crear la Red Irregular de Triángulos con el módulo **s.delaunay**

   GRASS: /path > **s.delaunay sites=**pluv10_90.2d **vect=**tin_pluv10_90 <ENTER>

   GRASS: /path > **d.vect** tin_pluv10_90 <ENTER>

   GRASS: /path > **d.sites** pluv10_90 <ENTER>

   Si observas el conjunto de órdenes anterior verás que para crear el TIN se ha utilizado un fichero de sites diferente, esto se debe a que **s.delaunay** no admite ficheros de sites tridimensionales por lo que se ha utilizado una versión bidimensional del mismo.

   Tras ejecutar la triangulación es conveniente ejecutar el comando **v.support** con la capa vectorial resultante:

   GRASS: /path > **v.support** tin_pluv10_90 <ENTER>

2. Creación de una capa raster a partir del TIN y los sites:

   GRASS: /path > **v.surf.tin input=**tin_pluv10_90 **sites=**pluv10_90.2d **output=** pluv10_90_rast; <ENTER>

8.8. Interpolación mediante polígonos de Thiessen

Además de para interpolar mediante TIN, la triangulación de delaunay es el paso previo para crear polígonos de Thiessen. Estos pueden crearse directamente a partir del módulo **s.voronoi**.

GRASS: /path > **s.voronoi sites=**pluv10_90.2d **vect=**thie_pluv10_90 <ENTER>

GRASS: /path > **d.vect** thie_pluv10_90 <ENTER>

GRASS: /path > **d.sites** pluv10_90 <ENTER>

Para llevar a cabo la interpolación, es preciso rasterizar los polígonos de Thiessen y convertir en raster el mapa de sites asignando a cada celdilla el valor de la variable que se va a interpolar :

GRASS: /path > **v.to.rast input=**thie_pluv10_90 **output=**thie_pluv10_90 <ENTER>

GRASS: /path > **s.to.rast input=**pluv10_90.2d **output=**pluv10_90.2d <ENTER>

A continuación basta con calcular el valor medio del mapa pluv10_90.2d para cada uno de los polígonos representados en el mapa thie_pluv10_90:

GRASS: /path > **r.statistics base=**thie_pluv10_90 **cover=**pluv10_90.2d **method=**average <ENTER>

8.9. Interpolación a partir de curvas de nivel

Los más habituales son **r.surf.contour** y **v.surf.rst**. El primero utiliza curvas de nivel rasterizadas con **v.to.rast** y un algoritmo que completa los espacios vacíos entre las celdillas a las que se ha asignado valor al rasterizar. Aunque no resulte del todo intuitivo, este hecho hace que cuanto mayor sea la densidad de curvas de nivel en la región de trabajo más rápido se hará la interpolación debido a la mayor cantidad de información disponible.

El módulo **v.surf.rst** utiliza un algoritmo de interpolación por **splines** a partir de curvas de nivel. Si no se sabe cual utilizar, es preferible **r.surf.contour**.

El primer paso es disponer de un fichero vectorial con curvas de nivel digitalizadas y con cota. En la LOCATION *pbx2* se dispone de un mapa llamado carrascoy con las curvas de esta sierra. Para rasterizar estas curvas puedes utilizar el módulo **v.to.rast**.

> GRASS: /path > **v.to.rast** input=carrascoy output=carrascoy_curvas <ENTER>

Los resultados serán mejores si se adoptan las siguiente precauciones:

- No existen curvas cortadas
- El tamaño de celdilla utilizado para rasterizar las curvas de nivel es lo suficientemente pequeño para que las curvas no se superpongan (resolución espacial adecuada)
- Una vez rasterizadas los valores de las curvas de nivel se multiplican por cien (resolución vertical adecuada).
- No falta ninguna curva

Para obtener una resolución vertical adecuada basta con el siguiente programa de **r.mapcalc**:

> GRASS: /path > **r.mapcalc** 'carrascoy_curvas_cm=100*carrascoy_curvas' <ENTER>

Prueba a ejecutar este módulo con diferentes resoluciones y verifica los posibles errores que puedan producirse.

Para obtener el modelo de elevaciones a partir de estas curvas de nivel rasterizadas utiliza **r.surf.contour**. Para ello haz primero un zoom a una zona pequeña ya que se trata de un módulo que requiere bastante tiempo para ejecutarse, por tanto es preferible empezar haciendo modelos pequeños para practicar. Una vez hecho el zoom ejecuta:

> GRASS: /path > **r.surf.contour** -f input=carrascoy_curvas_cm output=carrascoy_dem_cm <ENTER>

La opción -f ejecuta el módulo en modo rápido . Ya puedes visualizar el MDE generado.

8.10. Cuestiones

1. Se sabe que los valores de permeabilidad para los diferentes usos de suelo son:

   - Bosque=50
   - Matorral=30
   - Secano=10
   - Regadío=40

   Obtén un mapa de permeabilidades

2. Dibuja el mapa raster de suelos y superponle el mapa de sites denominado **mo**.

3. Obtén información sobre el mapa de sites con los comandos **s.info** y **d.what.sites**. ¿Que información contiene?

4. Obtén, por clasificación, un mapa raster del contenido en materia orgánica.

5. Calcula y representa el semivariograma de la variable materia orgánica. Ajusta el semivariograma experimental con diferentes modelos. ¿Cual crees que sería una distancia adecuada para llevar a cabo la interpolación por el método IDW? ¿Cual sería el equivalente de esta distancia expresado como número de puntos más cercanos a considerar en la interpolación? Dada la distancia entere los puntos de interpolación, cual crees que sería una resolución adecuada para los mapas raster interpolados? Asigna esa resolución a la región de trabajo con la orden **g.region res=***n*.

6. Interpola mediante polígonos de Thiessen el contenido en materia orgánica.

7. Interpola el contenido de materia orgánica mediante el método TIN. Antes de ejecutar el comando **v.surf.TIN** debes ejecutar **v.support**.

8. Interpola el contenido de materia orgánica mediante el método de splines.

9. Compara los cinco métodos empleados (clasificación, splines, TIN, thiessen e IDW) tanto de forma global (**r.report**) como local (**d.what.rast**)

10. En un episodio de precipitación determinado, la cantidad de lluvia aumentó desde la costa hacia el interior de la Región de Murcia a razón de:

$$\frac{\delta P}{\delta X} = 0{,}5mm/Km \tag{8.1}$$

$$\frac{\delta P}{\delta Y} = 0{,}75mm/Km \tag{8.2}$$

$$\tag{8.3}$$

    Sabiendo que en Totana cayeron 24 mm de precipitación genera un mapa de precipitación para toda la zona de trabajo.

Capítulo 9

Análisis de Modelos Digitales de Terreno

9.1. Visualización 3D

Una de las aplicaciones más habituales cuando se trabaja con un Modelo Digital de Elevaciones (capa raster de elevaciones) es obtener representaciones tridimensionales del terreno. El módulo de GRASS más habitual para ello es **d.3d**

GRASS: /path > **d.3d map=***mapa_colores* **elevation**=*mapa_mde* <ENTER>

Donde *mapa_mde* es el mapa que contiene el valor de Z que se va a levantar en 3D (generalmente un modelo digital de elevaciones) y *mapa_colores* es el mapa con cuyos colores se va a pintar el bloque tridimensional creado.

En la figura 9.2 aparece una visualización 3D de una porción de Sierra Espuña y Valle del Guadalentín utilizando el mapa **mapa200** para dar colores. La orden sería:

GRASS: /path > **d.3d map=**mapa200 **elevation**=mdecm <ENTER>

Los parámetros de la visualización 3D pueden modificarse posteriormente de modo interactivo.

Estos parámetros son ajustados en una ventana como esta:

incluyen la posición del ojo, el punto al que este mira, el campo de visión, la resolución, etc. La mejor manera de entender el significado de los distintos parámetros es probarlos. Cuando se hayan modificado pulsar <Esc>y <Enter>. Si el parámetro **RUN?** está como Y se visualizará el resultado, si está en N se sale del módulo.

La figura 9.3 muestra Sierra Espuña utilizando el mapa de usos del suelo para dar colores. Pueden apreciarse en color verde oscuro los bosques, en amarillo el matorral, en naranja el secano y en verde claro el regadío.

```
------------------------------------------------------------------------------
            VIEWING REGION                | RUN? Y/N              Y_
            N: 4300008.75147295           | Erase Color           black___
   W: 549989.41179575-- E: 710012.91489642Vertical Exaggerat.   2______
            S: 4129991.24906267           | Field of View (deg)   30.00__
                                          | Lines Only? Y/N       Y_
          VIEW COORDINATES:               | Line Color            color___
Eye Position              Center of view| Line Frequency        1____
 3959973.75<- Northing (y) -> 4215000.00| Resolution            13312.27__
 389965.91_<- Easting  (x) -> 630001.16_| Plot null elev? Y/N   N_
 176834.90_<- Height   (z) -> 1724.00___| Box color             none____
                                          | Average elevs? Y/N    N_
------------------------------------------------------------------------------
Eye -----                              | Colors: red orange yellow green blue
 \                    N                |   indigo violet brown gray white black
  \     /MAP----------/                 |
   \   /      X      /                  | Special 'colors':
     W/_____________/E                  |   'None' available for 'Erase Color'
                   S                    |   'color' available for 'Line Color'
------------------------------------------------------------------------------

          AFTER COMPLETING ALL ANSWERS, HIT <ESC><ENTER> TO CONTINUE
```

Figura 9.1: Menú de manejo del programa d.3d

La figura 9.4 contiene la misma visualización pero ahora obtenida con **NVIZ**, se trata de un nuevo módulo de visualización 3D más potente que **d.3d** pero no disponible en todas las distribuciones de GRASS por su dependencia de diversas librerías. Es un sistema algo más complejo que **d.3d** pero que produce resultados de mucha mayor calidad que este, tal como se aprecia comparando las figuras 9.3 y 9.4.

9.2. Visualización de perfiles

Otra posibilidad de visualización es la definición de perfiles mediante lineas sobre el mapa para, después, obtener una representación gráfica de los cambios de elevación a lo largo de estas lineas

En GRASS existen dos comandos que hacen perfiles: **d.profile** y **r.profile**. Para ejecutar el primero es conveniente maximizar el tamaño del monitor gráfico ya que se va a dividir en varias porciones que, en caso contrario no se visualizarían de forma correcta. Una vez ajustado el tamaño del monitor gráfico basta con llamar al módulo seguido del nombre de la capa raster de la que se quieren extraer los perfiles, generalmente un Modelo Digital de Elevaciones:

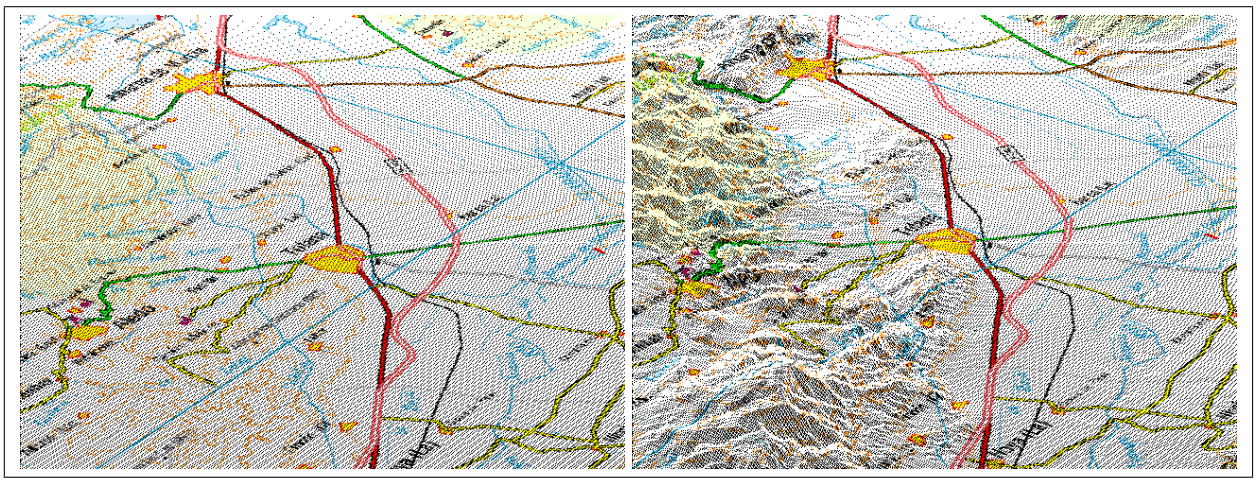


Figura 9.2: Vista 3D

GRASS: /path > **d.profile** mdecm <ENTER>

El monitor gráfico se divide en dos sectores (figura 9.5) a la izquierda aparece el MDE y un menú de ratón en la parte alta, vasta con seguir sus instrucciones. A la derecha aparecen cuatro rectángulos que contendrán hasta 4 gráficos generados a partir de los perfiles.

Otra posibilidad es utilizar el módulo **r.profile**, esté módulo obtiene los perfiles como una lista de valores numéricos útiles para pasarselos a un programa de gráficos que se encargue de su visualización. Dentro del proyecto yerba se incluye el módulo **r.perfiles** que hace esto de manera automática, para ejecutar este módulo basta con escribir:

GRASS: /path > **r.perfiles input**=mdecm **titulo=**”mi titulo” **output**=fichero <ENTER>

A continuación el sistema genera un menú de ratón que nos permite pinchar en cuantos puntos del monitor gráfico queramos para generar una linea a partir de la que se construirá el perfil. El resultado será un fichero de texto con el perfil y un gráfico con su visualización (figura 9.6)

9.3. Obtención de información derivada

Un MDE contiene información altimétrica de la que se pueden derivar diversos mapas con parámetros topográficos y geomorfológicos (pendientes, orientaciones, formas, áreas drenadas, visibilidad, etc.), el conjunto de estos mapas se conoce como Modelo Digital de Terreno (MDT).

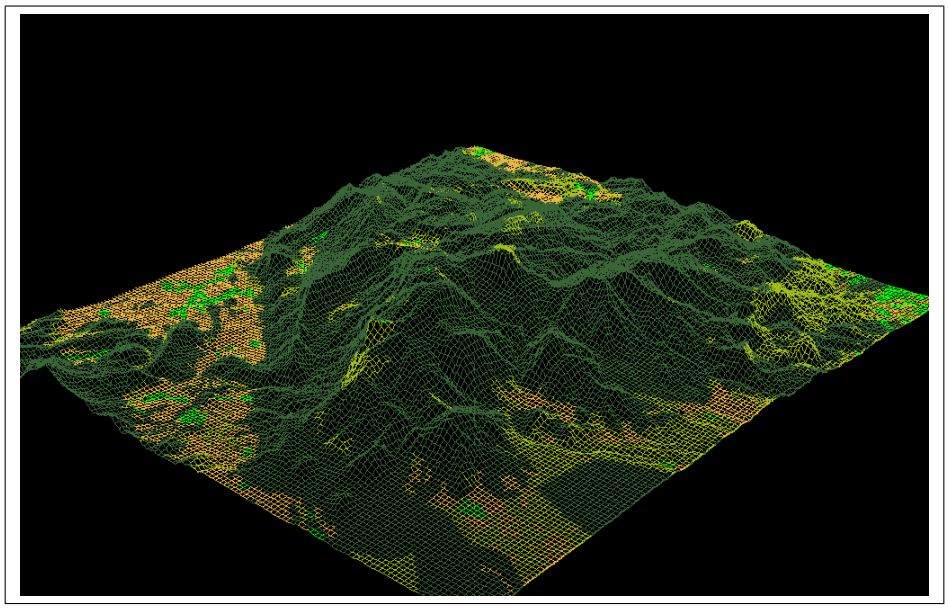


Figura 9.3: Vista 3D de Sierra Espuña con d.3d

La práctica totalidad de los mapas derivados puede obtenerse mediante el módulo **r.slope.aspect**. Si ejecutas **r.slope.aspect help** verás que es necesario introducir una capa de entrada (elevaciones) y podemos seleccionar una o varias capas de salida (los diferentes parámetros).

De momento vamos a ser generar un mapa de pendientes y otro de orientaciones que son los más frecuentemente utilizados:

> GRASS: /path > **r.slope.aspect elevation=**mdecm **slope=**pend **aspect=**asp **pcurv=**cur_perf **tcurv=**cur_trans **zfactor=**0.01 **min_slp_allowed=**5 <ENTER>

El parámetro **zfactor** indica cual es el factor por el que hay que multiplicar las unidades del mapa de elevaciones para obtener metros, puesto que el mapa mdecm está en centímetros el valor será 0.01. El parámetro **min_slp_allowed**, que por defecto no se utiliza, sirve para determinar un valor mínimo de pendiente para calcular la orientación, es conveniente utilizarlo porque no tiene mucho sentido calcular orientaciones en celdillas con pendiente cercana a cero, estando además este cálculo más sujeto a errores cuanto menor sea la pendiente.

9.4. Análisis de un MDT

La visualización de los mapas de elevaciones, pendientes y orientaciones permite comprobar la calidad tanto del MDT en general como del MDE en particular. Los problemas habituales son:

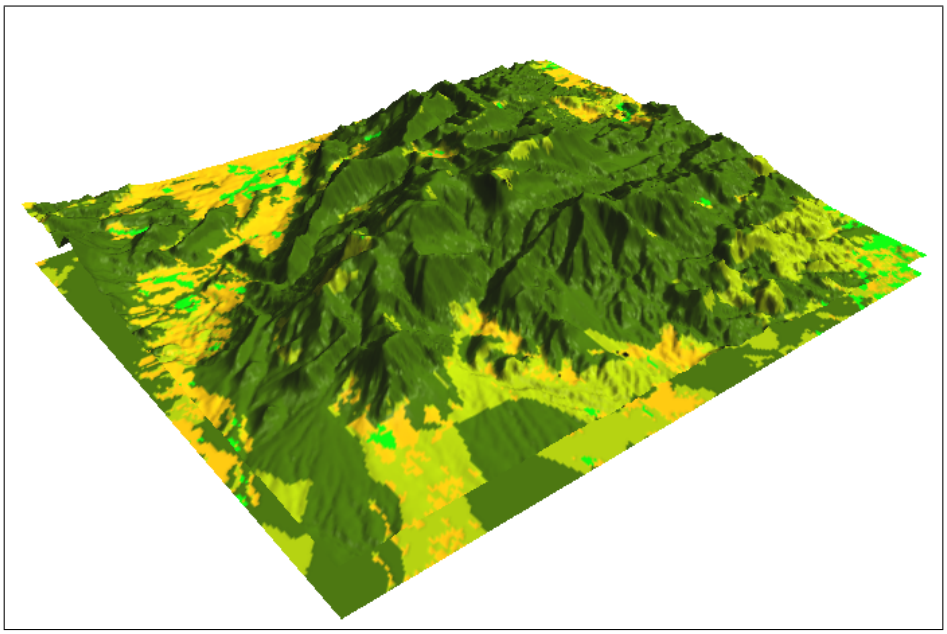


Figura 9.4: Vista 3D de Sierra Espuña con NVIZ

1. **En el mapa de orientaciones aparecen sectores con una variación caótica coincidiendo con sectores de baja pendiente**. Si la pendiente es 0 no existe orientación por tanto no tiene mucho sentido adjudicar un valor a sectores con pendientes inferiores a determinado umbral. Además errores muy pequeños asociados al MDE pueden producir cambios radicales e impredecibles en la orientación de las celdillas. La solución es utilizar la opción **min_slp_allowed** del módulo **r.slope.aspect**.

2. **Aparecen** *artefactos***, formas extrañas en el MDE que se reproducen en los mapas de pendiente y orientaciones**. Esto sucede cuando el tamaño de las celdillas es demasiado grande y se han superpuesto diversas curvas en una misma celdilla al rasterizarlas con **v.to.rast**. La solución es disminuir el tamaño de la celdilla. Sin embargo hay que tener cuidado para no acabar generando un raster de proporciones inmanejables.

   El problema anterior podría deberse también a la existencia de curvas de nivel no completadas. Siempre hay que chequear los mapas de curvas de nivel para localizar estos errores.

3. **Aparece un bandeado y pendientes nulas en sectores con las curvas de nivel algo separadas**. La mayoría de los programas de generación de MDT trabajan con números enteros, por tanto si el número de celdillas entre dos curvas de nivel es mayor que el desnivel entre estas, se repetirán los valores produciendo un efecto de aterrazamiento. La solución sería multiplicar los valores de altitud de las curvas de nivel por una potencia de diez y generar el MDE en decímetros, centímetros o la unidad que sea necesaria.

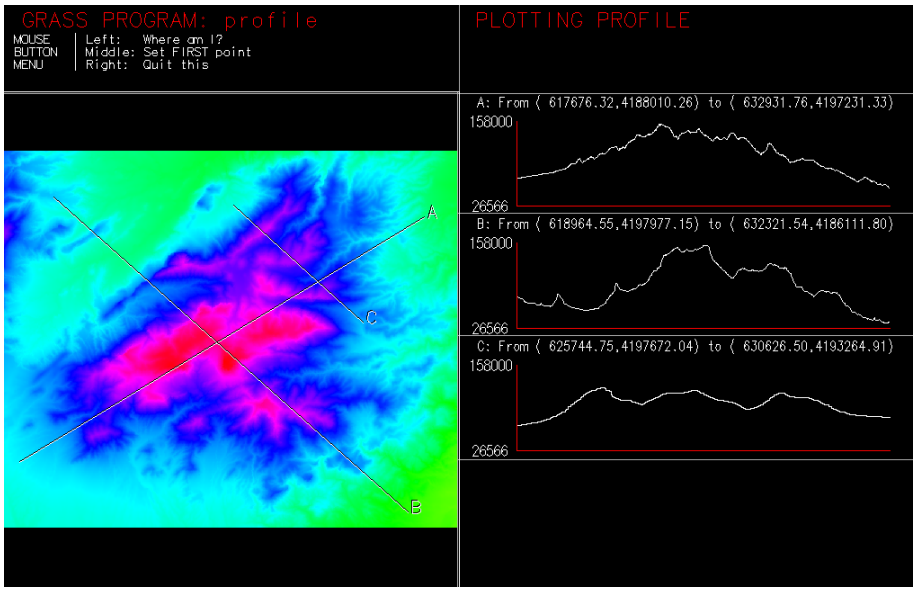


Figura 9.5: Ejemplo de la utilización de **d.profile**

9.5. Modelización a partir de MDT

Cualquier proceso variado en el espacio que ocurra sobre la superficie terrestre, y en el que la topografía suponga un factor del mismo, puede ser modelizado a partir de un MDT. La introducción de la variable tiempo permitirá el desarrollo de modelos dinámicos y espacialmente distribuidos, una introducción a los mismos se verá posteriormente.

9.5.1. Iluminación e irradiancia

Uno de los factores ambientales fundamentales en cualquier medio es el efecto solana-umbría. Con GRASS, y a partir de un MDE, podremos generar capas de diversas variables relacionadas con este efecto como irradiancia total (radiación recibida durante un determinado período de tiempo), ángulo de incidencia solar y duración de la insolación. Estas capas no sólo van a ser útiles como factores ecológicos sino que también resultan especialmente útiles en estudios de teledetección. El comando para generar estos mapas es **r.sun**, se trata de un módulo con un gran número de parámetros que deberemos utilizar o no en función del tipo de mapa que queramos obtener. Los parámetros necesarios en cualquier caso son:

- **elevin**, el MDE
- **aspin**, el mapa de orientación
- **slopein**, el mapa de pendientes

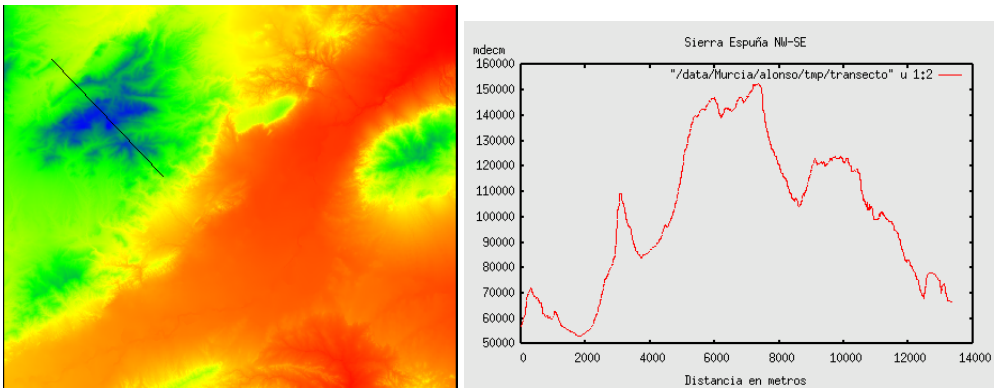


Figura 9.6: Ejemplo de la utilización de **r.perfiles**

- **lat**, latitud aproximada del lugar en grados. Si la región de trabajo es muy grande y se considera que los cambos de latitud pueden afectar significativamente al resultado es preferible introducir un mapa de latitudes con el parámetro **latin**
- **day**, día para el que se calcula el mapa

Si lo que queremos es un mapa de ángulo incidente de radiación en un instante preciso habrá que introducir también la hora:

r.sun -s elevin=mde **aspin=**asp **slopein=**pend **day=**32 **lat=**38 **incidout=**angulo **time=**12

La opción **-s** es para que el módulo tenga en cuenta el efecto de las sombras.

Podemos también calcular la energía recibida a lo largo de todo un día, para ello basta con especificar que la salida que queremos es un mapa de energía y no hacer ninguna referencia horaria

r.sun -s elevin=mde **aspin=**asp **slopein=**pend **day=**32 **lat=**38 **beam_rad=**radiacion

9.5.2. Visibilidad

El objetivo de este tipo de análisis es determinar desde que zonas es visible un punto (o que zona es visible desde ese punto). Para ello se utiliza el módulo **r.los**

> GRASS: /path > **r.los input=**mde **output=**mde_vis_vis **coordinate=**x,y **obs_elev=**1.7 **max_dist=**4000 <ENTER>

Las coordenadas x e y son las del punto que se quiere analizar. Se introduce también la altura del observador (esta variable será realmente importante cuando se trate de una garita de vigilancia o algún otro tipo de estructura) y la distancia máxima a la que se considera que puede alcanzarse a ver en función de las condiciones atmosféricas.

9.5.3. Modelización hidrológica

Es esta una de las aplicaciones más desarrolladas en GRASS, existen varios módulos relacionados con diversos aspectos de la modelización hidrológica. A continuación se verán algunos de ellos.

r.watershed

El módulo de GRASS **r.watershed** permite obtener información hidrológica de varios tipos a partir de un modelo digital de elevaciones. Se trata de un módulo bastante complejo que permite extraer capas raster de diversas propiedades hidrológicas. Requiere un estudio detallado para evaluar sus posibilidades. Como ejemplo sencillo y asumiendo que el mapa de elevaciones se denomina mdecm:

> GRASS: /path > **r.watershed elevation=**mdecm **threshold=**100 **accumulation=**acumulado **drainage=**drenaje **basin=**cuencas <ENTER>

Producirá a partir del MDE un mapa de área drenada acumulada en cada pixel (parámetro **accumulation**) es decir cuantas celdillas desaguan a ella, un mapa con las direcciones de drenaje (parámetro **drainage**) y una subdivisión en cuencas (parámetro **basin**). El parámetro **threshold** es obligatorio para obtener la subdivisión en cuencas ya que determina el tamaño mínimo que tendrán estas.

Los mapas obtenidos con este módulo son útiles para llevar a cabo análisis más detallado de modelización hidrológica. Por ejemplo a partir del mapa de drenaje acumulado pueden obtenerse mapas de redes de drenaje seleccionando un valor umbral de drenaje acumulado por encima del cual se considera que las celdillas forman parte de un cauce. Para ello basta con una ejecución de **r.mapcalc**:

> GRASS: /path > **r.mapcalc** 'cauces=if(acumulado>2000,1,null())' <ENTER>

Este mapa raster de cauces puede rasterizarse mediante la orden:

> GRASS: /path > **r.line input=**drenaje **output=**drenaje <ENTER>

r.drain

Establece el camino de máxima pendiente que seguiría un objeto cayendo en un mapa de elevaciones. El resultado es un mapa en que todos los valores son nulos excepto los de la ruta seguida por el objeto. Este módulo ya se ha visto a la hora de determinar rutas óptimas, ahora va a servir para determinar cual es la linea de flujo de una celdilla determinada.

> GRASS: /path > **r.drain input=**mdecm **output=***mapa_linea* **coordinate=***x*,*y* <ENTER>

r.water.outlet

Genera una cuenca de drenaje partiendo de un mapa de direcciones de flujo y un punto señalando la desembocadura de la cuenca.

r.water.outlet drainage=n **basin=**name **easting=**x **northing=**y

9.6. Cuestiones

1. Crea un MDE a partir de las curvas de nivel que aparecen en el mapa carrascoy. Debes tener en cuenta:
   - Para evitar que el proceso dure demasiado haz un zoom a una zona pequeña
   - Ajusta una resolución vertical y horizontal adecuada a la distancia horizontal e intervalo de valores de las curvas de nivel
2. Visualiza el mapa de elevaciones con el módulo d.3d
3. Visualiza cuatro perfiles transversales del mapa creado
4. Obtén un mapa de pendientes y otro de orientaciones
5. Intenta crear un mapa de cauces

Capítulo 10

Formato vectorial y análisis espacial

Con el modelo de datos vectorial, los objetos geográficos (puntos, lineas o polígonos) se describen mediante la codificación explícita de sus coordenadas (información geométrica), en la versión actual de GRASS estas coordenadas se limitan a dos dimensiones. El modelo digital de datos vectoriales utilizado en GRASS codifica, además de la geometría, la topología de los objetos y asigna a cada uno un identificador numérico[1] y, si el usuario lo considera necesario, una etiqueta de texto.

La información temática se almacena en tablas situadas en bases de datos externas. Una de las columnas de estas tablas debe contener los identificadores de los objetos para poder enlazar la capa vectorial con las tablas según el modelo geo-relacional.

El modelo de datos vectorial es el que más cambios va a experimentar en futuras versiones de GRASS[2], incorporando vectores tridimensionales y la posibilidad de almacenar no sólo la información temática sino también la geométrica y la topológica en bases de datos externas.

10.1. El modelo de datos vectorial en GRASS

Las versiones de GRASS hasta la 5.4, admiten 3 tipos de objetos vectoriales:

- Puntos
- Lineas
- Polígonos

[1] En GRASS el identificador numérico no necesita ser único para cada objeto ya que cada objeto almacena su propio identificador invisible para el usuario

[2] En 2005 ha salido ya la versión 6.0.0 del programa que incorpora todas estas novedades

Las lineas están formadas por **vértices** (pares de coordenadas) los vértices inicial y final se denominan **nodos**. Los polígonos están formados por una o varias lineas denominadas **arcos**. Un polígono completamente incluido dentro de otro se denomina **polígono isla**.

Al igual que en formato raster, los diferentes elementos que componen una capa vectorial se almacenan en varios ficheros que comparten el mismo nombre (el nombre de la capa) y que se sitúan en diferentes subdirectorios que cuelgan del correspondiente *MAPSET*. La información geométrica se almacena en un archivo con el nombre de la capa vectorial almacenado bajo el directorio */dig*. Existe además la posibilidad de almacenar la información geométrica en ficheros ASCII de intercambio situados bajo el directorio */dig_ascii*. Los módulos **v.in.ascii** y **v.out.ascii** permiten transformar la información de ASCII a binario y de binario a ASCII respectivamente. Debe quedar claro que la información en formato ASCII no es válida para ser tratada con los diferentes módulos de análisis vectorial.

Los identificadores numéricos de cada objeto se almacenan en un fichero ASCII bajo el directorio */dig_att* y las etiquetas de texto, también en formato ASCII, bajo el directorio */dig_cats*. La información topológica se almacena en un fichero binario bajo el directorio */dig_plus* y se genera ejecutando el módulo **v.support**.

10.2. Importación y exportación de ficheros vectoriales

Hoy en día, los SIG con mayor introducción en la administración trabajan en formato vectorial, por tanto resulta importante para cualquier SIG disponer de una amplia capacidad de importación y exportación de otros formatos.

Uno de los SIG vectoriales más extendidos es **ArcView**. Este programa nació como un visualizador de mapas capaz de leer los datos de **ArcInfo** programa mucho más completo de la misma compañía. La tendencia en este momento es a la unificación de ambos en un nuevo producto denominado **ArcGIS**. Debido a esta introducción en el mercado resulta importante poder leer y generar archivos compatibles con estos formatos. El más utilizado, hasta el punto de haberse erigido en estandard *de facto* de la industria es el formato **shapefile**. Otros formatos compatibles con programas de **ESRI** que pueden ser incorporados a GRASS son **e00** y los **ungenerate** que son ficheros de intercambio en formato ASCII.

10.2.1. Importación y exportación de *shapefiles*

En GRASS se dispone de los módulos **v.in.shape** y **v.out.shape** para importar/exportar este formato.

Sin embargo hay que tener en cuenta que los ficheros **shapefile** no almacenan topología por lo que GRASS debe reconstruirla. Esta reconstrucción la lleva a cabo el módulo **v.in.shape** pero si la capa en cuestión no cumple las condiciones mínimas para la creación de la topología esta no se creará correctamente.

10.2.2. Importación y exportación de e00

Los ficheros en formato e00 almacenan objetos en formato vectorial y tablas en las que a cada objeto se asocian diferentes categorías. Para importar estos ficheros se utiliza el módulo **m.in.e00** que si se acompaña de la opción **-s** reconstruirá la topología. Este módulo importa la información geométrica y creará tantos ficheros de categorías (los que se almacenan bajo el directorio */dig_cats*) como columnas existieran en la tabla del fichero original y les pone el nombre de la correspondiente columna.

Posteriormente puede crearse un mapa para cada variable sin más que hacer tantas copias del fichero que contiene la información geométrica como columnas en la tabla original y asignarles los correspondientes nombres.

La exportación de capas vectoriales a e00 se hace con el módulo **v.out.e00**

10.2.3. Formato ungenerate

Es un formato ASCII de Esri utilizado habitualmente para el intercambio de información espacial. Consiste en un número variable de ficheros (dependiendo del tipo de objetos almacenados). Si se trata de una capa (cobertura en la jerga de Esri) de lineas tendremos dos ficheros, uno con extensión *.lin* que contiene la geometría y otro de extensión *.dat* que contiene los atributos. En el caso de polígonos tendremos ficheros con extensión *.pol*, *.pnt* y *.dat*.

Los comandos **v.in.arc** y **v.in.arc.poly** permiten importar este formato.

10.2.4. Ficheros DXF

DXF es un formato de intercambio para programas de CAD, suele utilizarse también para intercambio de ficheros vectoriales. Los programas de CAD permiten tener varias capas de información en un sólo fichero de manera que pueden visualizarse por separado o en conjunto. Tradicionalmente estas capas contenían sólo información geométrica, aunque la convergencia entre programas de CAD y SIG lleva a la cada vez mayor incorporación de información temática y topológica. Cada una de las capas de información en CAD pueden, más o menos, asimilarse a una capa de información vectorial.

El módulo para importar DXF es **v.in.dxf** que almacena cada una de las capas del fichero DXF como una capa vectorial independiente. Si la información original no está adecuadamente separada por capas las capas vectoriales resultantes no van a almacenar la información de forma coherente, de manera que aparecerán por un lado capas con información de distinto tipo y por otro tipo información repartida entre distintas capas.

El módulo **v.in.dxf** no importa correctamente la coordenada Z de las entidades almacenadas en el fichero DXF, el caso más habitual es el de las curvas de nivel. Para ello se utiliza el módulo **v.in.dxf3d**.

La exportación se hace con el módulo **v.out.dxf**.

10.3. Digitalización

La entrada de información vectorial a un Sistema de Información Geográfica, suele requerir la digitalización de mapas en papel. Existen dos procedimientos básicos. El primero es la digitalización con tableta gráfica y el segundo la digitalización en pantalla, con el ratón, a partir de una imagen escaneada.

10.3.1. Imágenes escaneadas

En el proceso de escaneado de la imagen debe procurarse elegir la resolución adecuada. Esta no debe ser ni demasiado pequeña, para poder ver bien los detalles del mapa, ni demasiado grande de manera que la imagen resultante sea enorme. Al almacenar la imagen escaneada, los formatos más adecuados son GIF o PNG.

Una vez que se tiene el fichero gráfico, debe importarse a GRASS. (Cuando se tiene el fichero como capa raster, el siguiente paso es georreferenciarlo, para ello pueden utilizarse los módulos **i.points** e **i.rectify**.

Una vez que tenemos un mapa raster con las coordenadas adecuadas lo podremos utilizar como mapa de fondo (Menú *Customize* opción *Backdrop map* en el módulo **v.digit** que es la herramienta de digitalización de GRASS.

v.digit

Para ejecutar este módulo, simplemente hay que teclear el nombre del módulo **v.digit** ya que sólo puede trabajarse con el en modo interactivo. Durante toda la ejecución de v.digit, en el terminal aparece una pantalla de texto que nos permite trabajar con las diferentes opciones del módulo. Debe evitarse cambiar el tamaño del monitor virtual ya que podría causar problemas.

```
                Available  Digitizers

          Name                    Description
          ----                    -----------
  [1]    Calcomp              Calcomp digitizer, format 23 (binary)
  [2]    Altek                Altek digitizer, model AC30, format 8 (binary)
  [3]    none                   Run digit without the digitizer.


 Hit return to use digitizer in brackets
 or type in number or name of other digitizer.

 Select digitizer [none] :
```

En esta pantalla indicamos el tipo de tableta digitalizadora que se va a utilizar o ninguna si se va a trabajar con el ratón.

Posteriormente el programa pregunta que fichero se va a utilizar para trabajar, si el fichero existe, comprobará que se dispone de todos los ficheros necesarios, si no es así el programa nos dice que debemos ejecutar primero v.support y aborta. Si el fichero no existe nos pregunta si queremos crear uno nuevo.

A continuación nos muestra la siguiente pantalla:

```
Provide the following information:

Your organization   US Army Const. Eng. Rsch. Lab
Todays date (mon,yr)\_\_\_\_\_
Your name           \_\_\_\_\_
Map's name          \_\_\_\_\_
Map's date          \_\_\_\_\_
Map's scale         1:0\_\_\_\_\_
Other info          \_\_\_\_\_
Zone                0\_\_\_\_
West edge of area   550000\_\_\_\_\_
South edge of area  4130000\_\_\_\_\_
East edge of area   710000\_\_\_\_\_
North edge of area  4300000\_\_\_\_\_

              AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
```

El único dato que resulta imprescindible introducir es la escala que debe ser el del mapa que se digitaliza. Este parámetro sirve para calcular el factor de *span* es decir la distancia máxima que debe separar dos nodos digitalizados por el usuario para que el sistema los considere el mismo nodo y los pegue. Una vez presionada la tecla <ESC>al terminar, pregunta si se debe continuar si se responde afirmativamente aparece la siguiente pantalla que es ya la pantalla principal del módulo.

```
l-------------------------------------------------------------------------k
| GRASS-DIGIT Version  4.10                                    Main menu  |
|-------------------------------------------------------------------------|
| MAP INFORMATION                          | AMOUNT DIGITIZED             |
|   Name:                                  |   \# Lines:         0        |
|   Scale:        1                        |   \# Area edges:    0        |
|   Person:                                |   \# Sites:         0        |
|   Dig. Thresh.: 0.0300 in.               | - - - - - - - - - - - - - - -|
|   Map Thresh.:  0.0008 meters            |   Total points:    0         |
|                                          |                              |
```

```
|-------------------------------------------------------------------------|
| OPTIONS:                                  |                             |
|                                           |                             |
|     Digitizer:     Disabled               |                             |
|                                           |                             |
|                                           |                             |
|                                           |                             |
|                                                                         |
m-------------------------------------------------------------------------j
| Digitize  Edit  Label  Customize  Toolbox  Window  Help  Zoom  Quit  *  !   |
|                                                                         |
|                                                                         |
|GLOBAL MENU: Press first letter of desired command. [Upper Case Only]    |
m-------------------------------------------------------------------------j
```

Si el fichero introducido existe, nos lo presentará en el monitor gráfico.

Tecleando la primera letra (en mayúscula) de cada uno de los elementos del menú que aparecen en la pantalla de texto entramos al submenú correspondiente.

1. *Digitize*, diversas opciones para la digitalización de objetos en pantalla.
2. *Edit*, herramientas de modificación de los objetos en pantalla (borrado, inserción de puntos, *snap*, etc.).
3. *Label*, etiquetado y desetiquetado de objetos. Los polígonos sólo se etiquetan si puede reconstruirse su topología.
4. *Customize*, personalización, modificación de colores, elección de una capa raster o vectorial sobre la que digitalizar, etc.
5. *Toolbox*, diversas herramientas para resolver problemas con la geometría que puedan estar dando problemas a la hora de crear topología.
6. *Window*, selección de los elementos que queremos ver en pantalla
7. *Help*, ayuda
8. *Zoom*, opción accesible desde cualquier punto del sistema de menús
9. *Quit*

El módulo **v.digit** es muy completo y cada uno de los submenús tiene un gran número de opciones, puedes tratar de explorarlas.

10.4. Consulta

Existen tres módulos básicos para la consulta de datos en capas vectoriales cuyo funcionamiento es similar a sus homólogos en raster:

- **v.info**
- **v.report**
- **d.what.vect**

El módulo **v.report** requiere, además del nombre de la capa vectorial que se va a analizar y las unidades de medida de los resultados, el tipo de objetos (polígonos, lineas o puntos) que contiene la capa.

10.5. Manipulación de capas vectoriales para generar información topológica

Para conseguir en GRASS una adecuada creación de topología, se requiere que la información geométrica cumpla una serie de consideraciones:

- Las lineas no deben cruzarse consigo mismas
- Dos lineas están conectadas sólo si comparten un mismo nodo, es decir si las coordenadas de los vértices correspondientes son iguales.
- Un polígono sólo existe si está rodeado por lineas conectadas entre si de manera que se genere un circuito cerrado

Durante el trabajo de digitalización pueden producirse diversos errores, de forma que la capa resultante incumpla algunos de los principios anteriores. Para solventarlo existen diversos módulos:

- **v.spag** Busca todos los cruces de lineas y los sustituye por nodos, de manera que parte las lineas implicadas
- **v.rm.dangles** Elimina las lineas de un tamaño inferior a determinado umbral, también da la posibilidad de eliminar todas las lineas que, en alguno de sus extremos, no estén conectadas con otras lineas (lineas exteriores).
- **v.cutter** Permite extraer sólo aquellos elementos que se encuentran dentro de un conjunto de un polígonos, que actúan como máscara binaria, incluidos en otra capa.

  Si los objetos representados no tienen etiqueta, puede asignársele una de forma automática con los comandos **v.alabel** y **v.llabel** dependiendo de que los objetos almacenados en la capa vectorial

sean lineas o polígonos. Estos módulos permiten elegir el valor de la etiqueta que va a asignarse o etiquetar de modo incremental (es decir a cada objeto se le asigna una etiqueta superior en una unidad al anterior) con la opción **-i**.

- **v.reclass** modificará los identificadores de los objetos de la misma forma como lo hace **r.reclass**. Sin embargo **v.reclass** no elimina los límites entre polígonos a los que se pueda adjudicar la misma etiqueta. Para ello hay que ejecutar el módulo **v.rmedge**.

10.6. Análisis espacial

El análisis espacial incluye una serie de técnicas de manejo de datos vectoriales que incluyen el cálculo de diversos estadísticos, distinguiéndose entre técnicas orientadas al manejo de mapas de puntos, mapas de lineas (redes), o mapas de polígonos.

10.6.1. Análisis de mapa de puntos

En este caso tenemos:

- Determinación del punto central y el círculo o elipse de desviación media
- Análisis del vecino más próximo

EL módulo **s.spatial.sh** realiza estos cálculos. Como variables de entrada hay que pasarle el mapset en el que está el mapa de sites y el nombre de este. La salida del programa es similar a:

```
/data/Murcia/PERMANENT/site_lists/mo
Número de puntos: 120 Area=1191 ha
Punto central: X=636531 Y=4185304  Desviación típica: Xs=11240 Ys=8781 r=14263
Angulo de desviación=87.661417 Radio mayor=8776 Radio menor=11229

distancia media al vecino más próximo: d0=2825
d0 esperada: da=1575.198400
ratio: r1=1.793603

Intervalos de confianza:
Nivel de Significación   Intervalos de confianza
0.005                       0.921493 - 1.078507
0.001                       0.888992 - 1.111008
0.0005                      0.877061 - 1.122939
0.0001                      0.852530 - 1.147470
```

Donde **N** es el número de puntos, **Area** el área entendida como el rectángulo mínimo que abarca todos los puntos, **X** e **Y** las coordenadas del punto central, **Xs** e **Ys** las desviaciones típicas de las coordenadas X e Y, **r** es el radio equivalente a una desviación típica, **Angulo** el ángulo que forma la orientación principal de los sites respecto al Norte y las desviaciones típicas respecto al eje de orientación principal (**Radio mayor** y respecto al eje transversal (**Radio menor**).

Además el programa dibuja la circunferencia y la elipse de desviación en color verde.

EL resto son parámetros relativos al análisis del vecino más próximo: **do** es la media de las distancias al punto más cercano, **da** el valor correspondiente a una distribución al azar y **r1** la ratio entre ambas magnitudes.

A continuación, para varios valores de nivel de significación, se dan los umbrales correspondientes por arriba y por abajo. En este caso se concluiría que la distribución es dispersa ya que para cualquier nivel de significación, el parámetro **r1** está por encima del intervalo de confianza lo que implica que las distancias al vecino más próximo son significativamente mayores de lo esperable al azar.

10.7. Cuestiones

1. Utilizando los módulo **d.vect** y **d.area** representar el mapa de municipios.
2. Pinta de color rojo el municipio de Mula y de Azul el de Murcia. Deja en verde todos los demás. Los límites entre municipios deben quedar en negro.
3. ¿Que ocurre con Alcantarilla? £‘Por que no ocurre lo mismo con Pliego? Plantea una posible solución
4. Utiliza **v.report** para obtener la extensión de cada uno de los municipios en hectáreas.
5. Representar el mapa de sites que contiene la precipitación en octubre de 1990 y, utilizando el módulo **s.spatial.sh**. Hacer un análisis espacial de la distribución de puntos. Discutir los resultados en clase.
6. Conecta la base de datos murcia y obtén estadísticos básicos de la variable población
7. Haz un mapa en color y una leyenda de la variable población.

Capítulo 11

Conexión a bases de datos

Una de las grandes ventajas de que GRASS sea un sistema abierto (es decir que su código sea público) es la facilidad con que puede interrelacionarse con otros sistemas abiertos. En concreto resulta muy sencilla la integración de GRASS con una base de datos en un modelo de integrado o escasamente integrado[1].

En GRASS existen diversos comandos para enlazar mapas vectoriales y bases de datos. Hay varios proyectos en marcha para la utilización de diferentes sistemas de gestión de base de datos *Open Source* con GRASS. Nos centraremos en PostgreSQL, uno de los SGBD de código abierto más populares, completas y flexibles pero a la vez sencillas de manejar como usuario.

11.1. Servidores y clientes de bases de datos

La gestión de bases de datos se basa en la existencia de un **programa servidor**; que organiza los datos, recibe las consultas, las ejecuta y las devuelve; y un **programa cliente** que el usuario ejecuta y que lanza las consultas creadas por este al servidor. El programa cliente y el servidor no tienen siquiera porque ejecutarse en el mismo ordenador.

Existen diferentes clientes para conectar al servidor de bases de datos de **PostgreSQL**. Vamos a utilizar en principio uno sencillo (**psql**). Si tecleamos:

GRASS: /path > **psql -l** <ENTER>

obtendremos un listado de todas la bases de datos disponibles para el servidor. Si queremos conectarnos a una de ellas se le especificará al teclear el comando:

GRASS: /path > **psql murcia** <ENTER>

[1] Estos apuntes se refieren a GRASS hasta su versión 5.4. Tanto en GRASS 5.7 como en su versión estable (6.0) se ha modificado completamente el modelo de datos vectorial así como los accesos a bases de datos. En la página web de GRASS pueden encontrarse tutoriales al respecto.

En este caso hemos especificado la base de datos a la que queremos conectarnos. El mensaje de bienvenida de psql será algo parecido a:

```
Welcome to psql 7.3.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

murcia=#
```

Disponemos de una serie de comandos formados por una barra y una letra que realizan operaciones sencillas:

- \h para pedir ayuda sobre comandos SQL
- \'? para pedir ayuda sobre los comandos de *barra y letra*
- \q para salir del programa
- \d para obtener un listado de las tablas que forman la base de datos

  Así que una vez dentro del cliente de bases de datos basta con teclear **d** para obtener la siguiente tabla:

```
           Listado de relaciones
 Schema |     Nombre      | Tipo  | Dueño
--------+-----------------+-------+--------
 public | edades          | tabla | alonso
 public | general         | tabla | alonso
 public | municipio       | tabla | alonso
 public | olivares        | tabla | alonso
 public | poblacion       | tabla | alonso
 public | puntos_prot_civ | tabla | alonso
 public | sectores        | tabla | alonso
 public | uso_suelo       | tabla | alonso
(8 filas)
```

La tabla *municipio* contiene los nombres de los términos municipales; la tabla *población* la de la evolución de la población entre 1900 y 1991; la tabla *uso_suelo* los usos del suelo por municipio; la tabla llamada *general* contiene, finalmente, un resumen de indicadores socioeconómicos de la Región de Murcia. Comprueba cuales son las variables que contiene cada una de ellas con el comando **d** seguido del nombre de la tabla. Por ejemplo la orden **d general** produce el siguiente resultado:

```
                      Tabla "public.general"
          Columna           |  Tipo     | Modificadores
----------------------------+-----------+---------------
| ident                     | int4      |           4 |
| municipio                 | varchar   |          60 |
| poblacion                 | int4      |           4 |
| variacion9698             | float8    |           8 |
| matrimonios               | int4      |           4 |
| nacimientos               | int4      |           4 |
| defunciones               | int4      |           4 |
| baseimponible             | int4      |           4 |
| paro                      | int4      |           4 |
| secano                    | int4      |           4 |
| regadio                   | int4      |           4 |
| viviendalibre             | int4      |           4 |
| viviendaprotegida         | int4      |           4 |
| parcelacatastral          | int4      |           4 |
| superficiecatastral       | int4      |           4 |
| correspondencianacida     | float8    |           8 |
| correspondenciadistribuid| float8    |           8 |
| vehiculos                 | float8    |           8 |
| lineastelefonicas         | float8    |           8 |
| farmacias                 | int4      |           4 |
| presupuestos              | int4      |           4 |
| hoteles                   | int4      |           4 |
| extrahotelera             | int4      |           4 |
```

Como ves tenemos 3 tipos de variables. El tipo *int4* corresponde a números enteros, el tipo *float8* corresponde a números reales y *varchar* a cadenas de caracteres. De todos estos atributos el más importante es **ident** ya que asigna a cada municipio un identificador único que coincide con el identificador del polígono correspondiente a dicho municipio en el mapa vectorial.

A partir de aquí hay que continuar utilizando el lenguaje SQL

11.2. El Lenguaje Estructurado de Consultas (SQL)

El lenguaje SQL dispone de instrucciones para la creación de tablas (**CREATE TABLE**), la inserción de datos en las tabla (**INSERT INTO**) la edición de los datos (**UPDATE**, **DELETE**) y la consulta (**SELECT**). Como puedes ver los nombres de las instrucciones corresponden a lo que se espera de ellas tal como es de esperar en un lenguaje declarativo.

```
 ident |           nombreg
-------+----------------------------
     1 | Abanilla
     2 | Abarán
     3 | Águilas
     4 | Albudeite
     :       :
     :       :
    41 | Totana
    42 | Ulea
    43 | Unión (La)
    44 | Villanueva del Río Segura
    45 | Yecla
```

Figura 11.1: Resultado de consulta

En este capítulo nos centraremos en la instrucción **SELECT**, que permite hacer consultas sobre una base de datos que supondremos que otros han creado y mantienen para que podamos acceder a ella. Puedes encontrar una introducción más completa de SQL (en castellano) en:

www.sindominio.net/ayuda/postgresql/tutorial-postgresql-1.html

El comando **SELECT** permite seleccionar objetos de una o varias tablas, en función de las relaciones establecidas entre ellas, que cumplan determinadas condiciones. Vamos a dividir esta sección en consultas a una sola tabla y consultas a varias tablas

11.2.1. Consultas a una sola tabla

Por ejemplo dada una base de datos que contenga una tabla de los municipios de la Región de Murcia con un identificador y un nombre, la siguiente consulta[2]

```
SELECT *
FROM general;
```

producirá la tabla que aparece en la figura 11.1.

Todas las órdenes de SQL acaban con un punto y coma. El asterisco indica que se quieren ver todas las columnas de la tabla municipios, evidentemente también pueden seleccionarse unas si y otras no.

[2]En realidad las consultas en lenguaje SQL se escriben en una sola linea, en este caso se han partido y se ha resaltado (mayúscula y negrita) las palabras clave del lenguaje por claridad

```
 ident |    municipio     | poblacion
-------+------------------+-----------
    17 | Cartagena        |    175628
    25 | Lorca            |     69930
    28 | Molina de Segura |     42008
    31 | Murcia           |    349040
(4 rows)
```

Figura 11.2: Resultado de consulta condicional

Consultas sujetas a una condición

Pueden ejecutarse órdenes más complejas en la que seleccionemos sólo algunas de las filas (también llamadas tuplas) que forman la tabla. Vamos a seleccionar, a partir de la tabla **general**, aquellos municipios que tenían en 1998 (columna **población**) una población inferior a 40000 habitantes. a la tabla anterior se añade una columna con la población de cada municipio, podemos introducir la siguiente orden:

```
SELECT ident,municipio,poblacion
FROM general
WHERE poblacion>40000;
```

que producirá una tabla con las tres columnas, pero sólo con aquellos municipios con una población superior a 40000 habitantes (figura 11.2).

Date cuenta que la columna **ident** de la tabla **general** tiene los mismos valores que la columna **ident** de la tabla **municipios**. Esta es una característica fundamental de las bases de datos relacionales que va a permitir establecer relaciones entre las tablas.

Consultas con resultados ordenados

Si no se ordena lo contrario, el orden en que se presentan los resultados coincide con el orden en que se almacenan en la base de datos. Si se requiere otro orden se especifica mediante:

```
SELECT ident,municipio,poblacion
FROM general
WHERE poblacion>40000
ORDER BY poblacion;
```

y el resultado aparece en la figura 11.3.

```
 ident |    municipio     | poblacion
-------+------------------+-----------
    28 | Molina de Segura |     42008
    25 | Lorca            |     69930
    17 | Cartagena        |    175628
    31 | Murcia           |    349040
(4 rows)
```

Figura 11.3: Resultado de una consulta con resultados ordenados

Consultas utilizando operadores y funciones

Una última opción dentro de las consultas a una sola tabla sería el uso de operadores. En el siguiente ejemplo se calcula la tasa de natalidad en tantos por mil (número de nacimientos por 1000 entre población total) y se le asigna el nombre de tasa con la instrucción **AS**. El **.** tras el 1000 indica que es un número real no entero y por tanto el resultado ha de ser tambien real.

```
SELECT ident,municipio,1000.*nacimientos/poblacion AS tasa
FROM general
ORDER BY tasa;
```

Otra posibilidad es utilizar funciones.

```
SELECT ident,municipio,AVG(poblacion) AS media,MIN(poblacion) AS mínimo, MAX(poblacion)
AS máximo
FROM tabla
WHERE poblacion>40000;
```

Consultas anidadas

En algunos casos es necesario que, al imponer una condición, esta haga referencia a otra consulta sobre la misma tabla u otra diferente. En el siguiente ejemplo se trata de localizar el municipio con menor número de habitantes:

```
 ident | municipio | poblacion
-------+-----------+-----------
    32 | Ojós      |       594
(1 fila)
```

Figura 11.4: Resultado de una consulta anidada

```
SELECTident,municipio,poblacion
FROM general
WHERE poblacion=
(SELECT MIN(poblacion)
FROM general);
```

Como ves, la subconsulta debe situarse entre paréntesis. El resultado aparece en la figura 11.4.

11.2.2. Consultas a varias tablas

Pueden combinarse varias tablas mediante operaciones de SQL más complejas. Cuando se combinan tablas suele introducirse un *alias* simplificado para las tablas (la primera letra por ejemplo). En el siguiente ejemplo se va a utilizar, además de la tabla **general**, una nueva tabla llamada **edades** que contiene la población de cada municipio dividida por grupos de edad. Puedes ver los contenidos de esta tabla con la orden **edades** y comprobar que existe una columna denominada **ident** que es equivalente a la columna **ident** de la tabla **general**.

Vamos a obtener el tanto por ciento de la población murciana que se sitúa entre 20 y 24 años. Asumimos que se dispone de una segunda tabla con datos de inmigración anual llamada *inmigración* que contiene, entre otras, una columna de identificadores *ident* (obviamente los mismos que en la tabla anterior) y otra con la estructura por edades. Puede obtenerse un índice demográfico del grado de envejecimiento de la población en tantos por cien combinando ambas tablas y utilizando operaciones aritméticas y lógicas para construir operaciones más complejas. Finalmente visualizaremos los resultados en orden ascendente:

```
SELECT g.ident,g.municipio,100.*(e.d0a1+e.d1a4+e.d5a9+e.d10a14+e.d15a19)/g.poblacion AS indice
FROM general g, edades e
WHERE d.ident=g.ident
ORDER BY indice;
```

11.3. Enlazando GRASS con bases de datos

Al ser PostgreSQL un sistema abierto, sus desarrolladores ponen a disposición de otros programadores un conjunto de librerías de funciones para facilitar el desarrollo de clientes de bases de datos. Utilizando estas librerías se han desarrollado diversos módulos de GRASS capaces de actuar como clientes de bases de datos.

En GRASS existen diversos comandos para enlazar mapas vectoriales y bases de datos. Hay varios proyectos en marcha para la utilización de diferentes sistemas de gestión de base de datos *Open Source* con GRASS. Nos centraremos en PostgreSQL que es una de las más completas y flexibles pero a la vez sencillas de manejar como usuario. Los módulos de GRASS que enlazan con Postgres son:

- Módulos generales:
  - **g.select.pg**, conecta GRASS con una base de datos almacenada en postgres
  - **g.table.pg**, devuelve un listado de las tablas disponibles en la base de datos seleccionada
  - **g.column.pg**, devuelve un listado de las columnas disponibles en una tabla
  - **g.stats.pg**, calcula los estadísticos básicos de una columna
- Módulos de visualización:
  - **d.rast.pg** Permite visualizar pixels cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones
  - **d.site.pg** Permite visualizar puntos cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones
  - **d.vect.pg** Permite visualizar puntos cuyo identificador corresponda a un objeto (registro en la base de datos) que cumplen determinadas condiciones
- Módulos de consulta interactiva
  - **d.what.r.pg**
  - **d.what.s.pg**
  - **d.what.v.pg**
- Módulos para crear mapas con categorías obtenidas de la base de datos:
  - **r.reclass.pg** cuyos identificadores se obtienen de una columna o índice en una tabla de la base de datos.
  - **r.rescale.pg** crea un mapa de polígonos cuyos identificadores se obtienen reescalando los valores de una columna o índice a un nuevo rango de valores.

11.3.1. Consultas generales

El primer paso será ejecutar el módulo de GRASS con el que seleccionamos la base de datos temática (conjunto de tablas almacenadas en PostgreSQL) con los mapas de la LOCATION que tenemos seleccionada, de esta manera GRASS pasará a actuar como programa cliente de bases de datos. Suponiendo que nuestra base de datos se llama murcia, la orden será:

GRASS: /path > **g.select.pg database=murcia** <ENTER>

Este módulo no dará ningún mensaje (salvo que se haya producido algún error) a continuación podemos obtener información acerca de como está organizada la base de datos, de que tablas dispone y cuales son los atributos (columnas) de cada una de ellas:

GRASS: /path > **g.table.pg** <ENTER>

nos dará la lista de las tablas presentes en la base de datos (puesto que la base de datos ya está seleccionada no hace falta especificarla en esta orden). Como verás aparecen bastantes tablas, de momento nos vamos a quedar con la tabla **general** que incluye un resumen de información socioeconómica de los distintos municipios de la región. Para averiguar cuales son las columnas disponibles teclea:

GRASS: /path > **g.column.pg -v general** <ENTER>

La opción -v aparece en muchos de los módulos de GRASS, su efecto es que la salida del módulo nos proporciona información no imprescindible acerca de la ejecución del mismo. En este caso el efecto es que, junto a los nombres de las columnas, nos da el *tipo atómico* de la columna y su tamaño en bytes.

Podemos también obtener estadísticos muy básicos acerca de los atributos con el módulo **g.stats.pg**

GRASS: /path > **g.stats.pg -v table=general column=poblacion** <ENTER>

producirá el siguiente resultado:

```
Executing
SELECT min(poblacion) as Min, max(poblacion) as Max, avg(poblacion) as Mean FROM

       Min,        Max,       Mean
-------------------------------------
       594,     349040, 24779.288888888889
```

En este caso la opción **-v** resulta útil ya que nos da la orden en SQL que el módulo le ha pasado al programa servidor de base de datos.

Otra opción de este módulo es **-f** que dará las frecuencias en lugar de estos estadísticos, probad:

GRASS: /path > **g.stats.pg -f table=general column=poblacion** <ENTER>

Puede también limitarse el cálculo de estadísticos a aquellos registros que cumplan una determinada condición con el parámetro **where**. Este recibe entre comillas una condición:

GRASS: /path > **g.stats.pg table=***general* **column=***poblacion* **where=**"poblacion>10000"<ENTER>

11.3.2. Visualización de información de la base de datos como mapas

La utilidad real de enlazar una base de datos espacial con una base de datos gestionada con un SGBD estriba en poder visualizar y consultar la información de esta de un modo gráfico.

El módulo **d.vect.pg** permite visualizar un mapa vectorial de polígonos asignando colores a aquellos polígonos que cumplan determinadas condiciones:

GRASS: /path > **d.erase white**

GRASS: /path > **d.vect.pg -f map=municipios tab=general key=ident where="poblacion>=50000" color=blue**

GRASS: /path > **d.vect termuni color=black**

Asegúrate de haber abierto previamente un monitor gráfico con **d.mon**. La primera orden borra la pantalla y le pone fondo blanco, la última dibuja en negro los límites de los polígonos con lo que se consigue un resultado más vistoso. La orden **d.vect.pg** utiliza como parámetros, en primer lugar el nombre del mapa vectorial que contiene los polígonos, en segundo lugar la tabla que contiene los datos, en tercer lugar la columna de la tabla que corresponde al identificador único de cada polígono. Finalmente el parámetro **where** nos permite establecer una condición para pintar o no los polígonos del color especificado en el parámetro **color**, la opción **-f** significa en este módulo que los polígonos se rellenarán con el color asignado, si no se específica se pinta sólo el contorno.

Este módulo tiene un pequeño defecto. En el caso de polígonos que cumplen la condición y contienen un polígono isla que no la cumple, el módulo pinta todo el polígono. Por ejemplo, en el caso anterior Murcia cumple la condición pero Alcantarilla no y se pinta de todos modos.

En GRASS existen diversos comandos para enlazar mapas vectoriales y bases de datos. Hay varios proyectos en marcha para la utilización de diferentes sistemas de gestión de base de datos *Open Source* con GRASS. Uno de ellos, desarrollado en la Universidad de Murcia, utiliza **postgres** como motor de base de datos.

En este momento el proyecto tiene 3 módulos de uso sencillo que permiten hacer consultas a una base de datos y visualizar los resultados en forma de mapa. Estos son:

- **d.vect.pg.cc**
- **d.legend.sql.sh**

- **r.in.sql.sh**

  GRASS: /path > **d.vect.pg.cc map=***nombre_mapa* **sql=***consulta_SQL_entrecomillada* **fichero=***archivo_con_las_reglas_de_color*<ENTER>

Por ejemplo, si tenemos un fichero llamado población con el siguiente contenido:

```
0 255 0 0
1000 0 255 0
5000 0 0 255
```

asignará colores entre rojo y verde a los polígonos cuyo valor de la variable consultada esté entre 0 y 1000 y colores del verde al azul a los polígonos con valores entre 1000 y 5000. Si se utiliza la opción **-l** en lugar de pintar el mapa se pintará la leyenda. Este módulo no se encuentra en las distribuciones normales de GRASS.

Por ejemplo las ordenes:

GRASS: /path > **d.vect.pg.cc map=**municipios **sql=**"select ident,p1991 from pob" **fichero=**paleta.txt<ENTER>

GRASS: /path > **d.vect.pg.cc -l map=**municipios **sql=**"select ident,p1991 from pob" **fichero=**paleta.txt<ENTER>

producirán respectivamente el mapa y la leyenda que aparecen en la figura 11.5 asumiendo que el contenido de colores es el que aparece un poco más arriba.

Es conveniente utilizar un editor (**emacs** o **kedit** por ejemplo) para escribir el archivo de colores para poder dejarlo abierto y modificarlo para ajustar las paletas a los valores devueltos por la base de datos.

En definitiva la única diferencia entre el trabajo de un gestor tradicional de bases de datos y el enlace de un SIG a una base de datos estriba en el modo de presentación (tabla o mapa). Casi todo el trabajo lo hace el gestor de bases de datos y GRASS, el Sistema de Información Geográfica, se limita a presentar los resultados. La auténtica novedad de los SIG vectoriales está en la yuxtaposición de mapas de diverso tipo para realizar análisis complejos del territorio.

11.4. Creación de mapas raster con SQL

Un mapa de polígonos puede representarse tanto en formato raster como en vectorial. En el formato raster, cada celdilla almacena el identificador del polígono.

Existe, de este modo, la posibilidad de reclasificar un mapa raster de polígonos en base a los resultados de una consulta SQL.

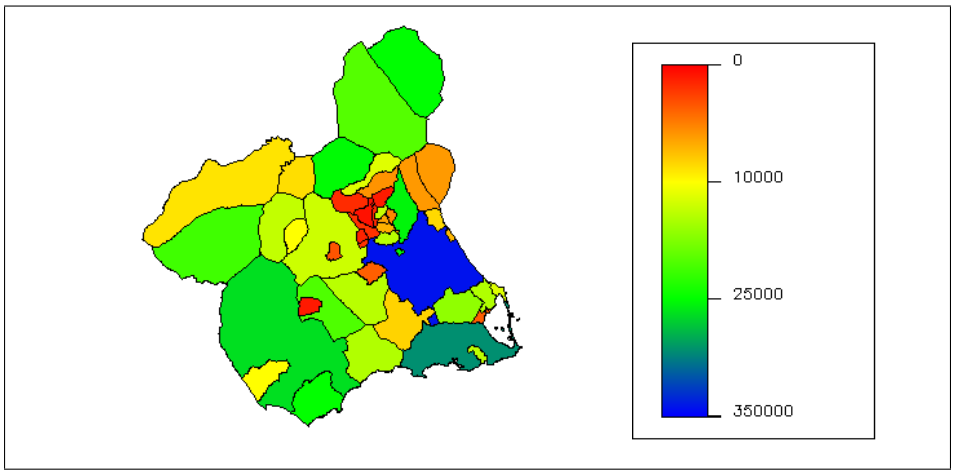


Figura 11.5: Población por municipios en la Región de Murcia en el año 1991

El módulo **r.in.sql.sh** crea un mapa raster de salida a partir de otro de entrada y un fichero de texto con una consulta SQL en la que, al igual que antes, se piden dos columnas: los identificadores y la variable que va a almacenarse en el mapa de salida. Se trata por tanto de una reclasificación.

11.5. Ejercicios

1. Obtener una tabla con el tanto por ciento de superficie de olivar en 1996 a partir de los valores de la tabla *olivares* y de la tabla *area*
2. Obtener una tabla con el tanto por ciento de hectáreas forestales (tablas *area* y *uso_suelo*
3. Crear mapas y leyendas con ambas tablas