



**INSTITUTO POTOSINO DE INVESTIGACIÓN
CIENTÍFICA Y TECNOLÓGICA, A.C.**

POSGRADO EN CIENCIAS APLICADAS

**Integración de un modelo de flujo a un Sistema de
Información Geográfica (SIG)**

Tesis que presenta

Tania Oyuki Chang Martínez

Para obtener el grado de

Maestro(a) en Geociencias Aplicadas

Director (Codirectores) de la Tesis:

Dr. Jaime Carrera Hernández

Dr. Alfredo Ramos Leal

San Luis Potosí, S.L.P., Septiembre de 2012



Constancia de aprobación de la tesis

La tesis **Integración de un modelo de flujo a un Sistema de Información Geográfica (SIG)** presentada para obtener el Grado de de Maestro(a) en Geociencias Aplicadas fue elaborada por **Tania Oyuki Chang Martínez** y aprobada el **11 de septiembre de 2012** por los suscritos, designados por el Colegio de Profesores de la División de Geociencias Aplicadas del Instituto Potosino de Investigación Científica y Tecnológica, A.C.



Créditos Institucionales

Esta tesis fue elaborada en la División de Geociencias Aplicadas del Instituto Potosino de Investigación Científica y Tecnológica, A.C., bajo la dirección del Dr. Jaime Carrera Hernández.

Durante la realización del trabajo el autor recibió una beca académica del Consejo Nacional de Ciencia y Tecnología 250297 y del Instituto Potosino de Investigación Científica y Tecnológica, A. C.



Instituto Potosino de Investigación Científica y Tecnológica, A.C.

Acta de Examen de Grado

El Secretario Académico del Instituto Potosino de Investigación Científica y Tecnológica, A.C., certifica que en el Acta 001 del Libro Primero de Actas de Exámenes de Grado del Programa de Maestría en Ciencias Aplicadas en la opción de Geociencias Aplicadas está asentado lo siguiente:

En la ciudad de San Luis Potosí a los 11 días del mes de septiembre del año 2012, se reunió a las 10:30 horas en las instalaciones del Instituto Potosino de Investigación Científica y Tecnológica, A.C., el Jurado integrado por:

Dr. José Noel Carbajal Pérez	Presidente	IPICYT
Dr. José Alfredo Ramos Leal	Secretario	IPICYT
Dra. Birgit Steinich	Sinodal	IPICYT
Dr. Jaime Jesús Carrera Hernández	Sinodal externo	UNAM

a fin de efectuar el examen, que para obtener el Grado de:

**MAESTRA EN CIENCIAS APLICADAS
EN LA OPCIÓN DE GEOCIENCIAS APLICADAS**

sustentó la C.

Tania Oyuki Chang Martínez

sobre la Tesis intitulada:

Integración de un modelo de flujo a un Sistema de Información Geográfica (SIG)

que se desarrolló bajo la dirección de

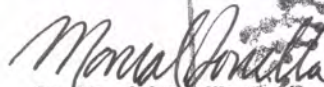
Dr. José Alfredo Ramos Leal
Dr. Jaime Jesús Carrera Hernández (UNAM)

El Jurado, después de deliberar, determinó

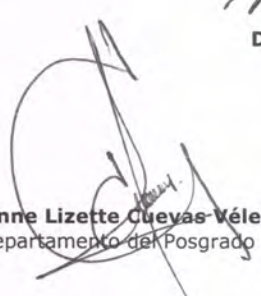
APROBARLA

Dándose por terminado el acto a las 11:45 horas, procediendo a la firma del Acta los integrantes del Jurado. Dando fe el Secretario Académico del Instituto.

A petición de la interesada y para los fines que a la misma convengan, se extiende el presente documento en la ciudad de San Luis Potosí, S.L.P., México, a los 11 días del mes de septiembre de 2012.


Dr. Marcial Bonilla Marín
Secretario Académico




Mtra. Ivonne Lizette Cuevas Vélez
Jefa del Departamento del Posgrado

Dedicatorias

*A mis padre
Alfonso Chang Moreno
y a mi madre
Martha E. Martínez Fernández*

Agradecimientos

A mis padres, por el apoyo incondicional que me han brindado a lo largo de mi vida.

A mis hermanos Alfonso, Karen y Laura, con quienes puedo contar siempre y a Luis y Pipo, por ser parte de la familia.

A mis compañeros y amigos de la maestría Angeles, Manuel, Simon y Pablo por compartir esta etapa de nuestro desarrollo académico, así como también de otras divisiones de IPICYT, Yahana, Arturo y Lorena.

A mis amigos Beto, Toño, Maya, Rafa, Erandi y todos los que compartieron y me apoyaron siempre durante este proceso. En especial a David por su apoyo incondicional.

A todos mis maestros de IPICYT, por haberme introducido en el área de las geociencias aplicadas.

Al Dr. Alfredo Ramos Leal, que ha sido un amigo y un gran apoyo a lo largo de mi estancia en IPICYT.

Al Dr. Noel Carvajal por su consejo para finalizar con éxito esta etapa, por su tiempo y su paciencia.

Al Dr. Jaime Carrera por sus enseñanzas sobre la investigación.

A IPICYT Y CONACYT por los fondos que recibí de su parte y por seguir impulsando la investigación.

A todas las personas que interesadas en mi trabajo y que lo estén leyendo.

Índice general

Constancia de aprobación de la tesis	III
Créditos Institucionales	V
Acta de examen	VII
Dedicatorias	IX
Agradecimientos	XI
Índice general	XIII
Resumen	XVII
Abstract	1
1. Introducción	3
Objetivos	7
2. Antecedentes	9
2.1. Ecuación de flujo para aguas subterráneas	9
2.1.1. Ley de Darcy	9
2.1.2. Derivación de la ecuación de flujo	10
2.2. MODFLOW	12

2.2.1.	MODFLOW: entradas y salidas	14
2.2.2.	MODFLOW y SIG	15
2.3.	SIG GRASS	16
2.3.1.	SIG: Un software de código libre	17
2.3.2.	PostgreSQL	18
3.	Metodología	19
3.1.	Módulo en GRASS	19
3.1.1.	Interfaz del módulo	20
3.2.	Información vectorial	21
3.3.	Información a partir de mapas ráster	21
3.4.	MODFLOW en el módulo	23
4.	Modelación numérica de aguas subterráneas	25
4.1.	Descripción general	26
4.1.1.	Condiciones históricas	28
4.1.2.	Modelo conceptual	30
4.2.	Creación del modelo matemático	33
5.	Resultados y discusión	39
5.1.	Acuífero libre de una capa	39
5.2.	Sistema de un acuífero con río	45
5.3.	Ejemplo de MODFLOW 2005	47
5.4.	Validez de los resultados	50
6.	Conclusiones	55
	Apéndices	56

<i>ÍNDICE GENERAL</i>	XV
A. Programa en C	59
B. Instalación de MODFLOW para el módulo <i>r.gws</i>	97
B.1. <i>rddown.f</i> y <i>rheads.f</i>	97
Bibliografía	99

Resumen

Para conocer el comportamiento de un sistema de flujo subterráneo, es necesario conocer las ecuaciones que lo describen, así como también las variables necesarias para resolverlas. Generalmente, los sistemas que describen el flujo subterráneo, no permiten una solución analítica, por lo que los métodos para resolver las ecuaciones que describen la dinámica de flujos subterráneos son de carácter numérico, ésto hace necesario utilizar herramientas computacionales. En este trabajo se desarrolló un software libre que hace uso de dos herramientas que se encuentran disponibles en la web, por una parte el software Open Source Geographic Resources Analysis Support System (GRASS) GIS que, por medio de Sistemas de Información Geográfica (GIS en inglés) permite crear, manipular y visualizar mapas de datos georeferenciados, teniendo como ventaja que los datos se encuentran en un plano o el espacio. Estos datos son representados por un mapa que cumple las características de una malla cuadrículada (grid). Por otra parte se tiene un código libre llamado MODFLOW, que resuelve las ecuaciones de flujo subterráneo por medio del método de diferencias finitas, método en el cual es necesario tener un mallado (grid). MODFLOW está escrito en Fortran y es de libre acceso. El módulo que se busca crear en este trabajo pretende integrar MODFLOW con GRASS GIS por medio de una interfaz gráfica, la cual permitirá que los datos guardados en GRASS sean los parámetros de entrada para el modelo de flujo utilizado por MODFLOW. Las ventajas que tiene el módulo es que será de código libre, para hacer el modelo de flujo no será necesario hacer la conversión de formatos necesarios para ambos softwares, por lo tanto, el proceso de modelación será más eficiente. Este desarrollo tecnológico puede ser muy útil en el manejo de información asociada a modelos numéricos.

Abstract

To understand the behavior of a groundwater flow system it is necessary to know the equations that describe it, as well as the variables needed to solve it. Generally, systems that describe groundwater flow, do not allow analytical solution, so the methods used to solve these equations are of numeric nature, so it is necessary to use computational tools. This project seeks to develop an open source software that makes use of two tools that are available on the web, first the open source software Geographic Resources Analysis Support System (GRASS) GIS, that by means of Geographical Information Systems (GIS) is able to create, manipulate and display maps of georeferenced data, with the advantage that the data is in square gridded plane or space. Additionally we used the open source software MODFLOW, which solves the groundwater flow equations using a finite difference method, which makes use of a mesh (grid). MODFLOW is written in FORTRAN and is freely accessible. The module we seek to create intends to integrate MODFLOW with GRASS GIS through a graphic interface, which will allow data stored in GRASS to be input parameters for the flow models used by MODFLOW. The advantages of this module is that it will be open source, in order to make the flow model it will no longer be necessary to make format conversions required by both software, therefore making the modeling processes more efficient. This tecnologic development can be very useful in the management of information associated with numerical models.

Capítulo 1

Introducción

Los Sistemas de Información Geográfica (SIG) han sido una herramienta importante en el desarrollo de estudios de diversa índole. En los últimos años, ha aumentado su uso debido a la capacidad que tiene de manipular una gran variedad de datos, en especial datos de tipo espacial.

Debido a que una de las características de los SIG, es el manejo de información a distintas escalas, se ha podido aplicar a estudios con distintos tipos de datos como son geológicos, uso de suelo, variabilidad espacial de escorrentías, etc., esto para ser estudiados por una amplia variedad de análisis. Desde hace algunos años se ha manejado el uso de los SIG para estudios de obtención de mapas de clima más precisos (El-Kenawy et al., 2010), modelación de temperaturas para procesos ambientales (Cristóbal et al., 2008; Ustrnul y Czekierda, 2005), prevención y protección de la salud pública (Croner et al., 1998) y estudios de erosión en cuencas (De-Roo, 1998) por mencionar algunas.

En el estudio de aguas subterráneas e hidrogeología, los SIG han tenido un gran impacto debido a la información que se puede obtener a partir de imágenes como son los Modelos de Elevación Digital (DEM) (figura 1.1), ya que es posible, mediante algoritmos, la manipulación de los datos para obtención de nuevos mapas con información relevante para cada estudio.

Existen diversos softwares para los SIG, como son gvSIG, ArcGIS, Ilwis, GRASS GIS etc., algunos de ellos son capaces de interactuar con herramientas como la digitalización de mapas, modelación matemática, métodos estadísticos, etc. Sin embargo, en su mayoría esto representa un problema debido al cambio en los formatos de cada uno, por lo que para el usuario esto se vuelve más tardado.

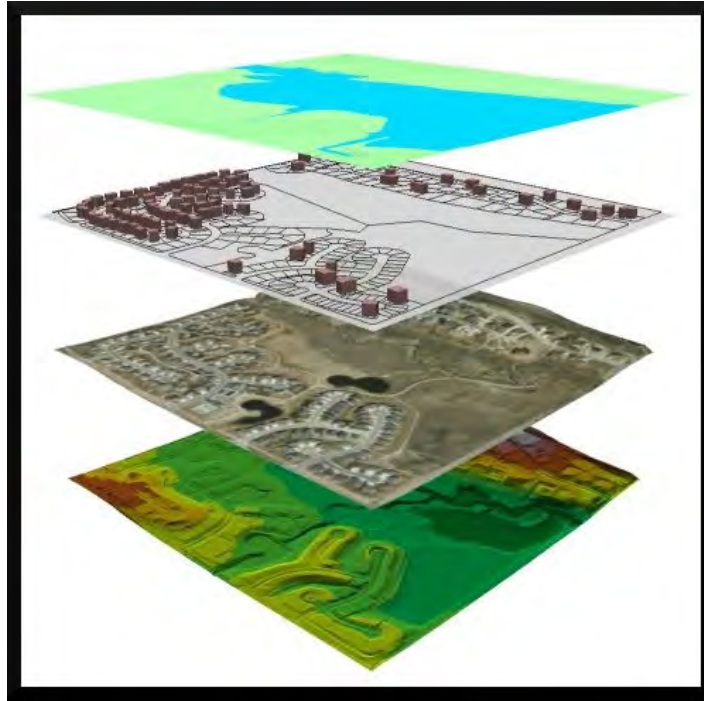


Figura 1.1: Obtención de mapas a partir de un Modelo de Elevación Digital (DEM)

El SIG GRASS (Geographic Resources Analysis Support System) es un software libre, de código abierto que permite al usuario crear sus propios programas en caso de que así se requiera. SIG GRASS, también cuenta con algoritmos que permiten que el usuario pueda manipular libremente sus mapas, usando como herramienta *r.mapcalc* para el álgebra de mapas, por mencionar un ejemplo. El gestor de Sistemas de Información Geográfica GRASS es un software compatible con los sistemas operativos Linux, MacOS y Windows.

En este trabajo se busca integrar el software SIG GRASS con un modelo matemático llamado MODFLOW (3D Finite-Difference Groundwater Flow Model) que simula el flujo subterráneo.

En la figura 1.2 se muestra la interacción del SIG GRASS con herramientas externas, el manejo de los datos por medio de R (software para análisis geoestadístico), digitalización de mapas por medio de Quantum GIS, el gestor de bases de datos relacionales PostgreSQL que se maneja desde SIG GRASS y la visualización de mapas que ofrece este software. Debido a la facilidad de la manipulación de información en SIG GRASS el módulo desarrollado en este trabajo *r.gws* enlaza el modelo matemático para flujo subterráneo MODFLOW con SIG GRASS, aprovechando todas estas ventajas por parte de GRASS.

MODFLOW es un programa desarrollado en fortran, de acceso libre, que mediante la solución de la ecuación de flujo con el método de diferencias finitas, determina la

evolución del sistema que se estudia. Este programa, al igual que GRASS, es compatible con Linux, MacOS y Windows.

El resultado final de este trabajo es un módulo de SIG GRASS *r.gws* que crea todos los archivos de entrada de MODFLOW a partir de mapas hechos en GRASS y los importa para su simulación, una vez obtenidos los resultados de la simulación por parte de MODFLOW, el módulo lo interpreta para que GRASS permita la visualización del resultado.

Este desarrollo tecnológico puede contribuir a simplificar la entrada de datos al modelo MODFLOW y la visualización de los resultados, con un correspondiente ahorro de tiempo.

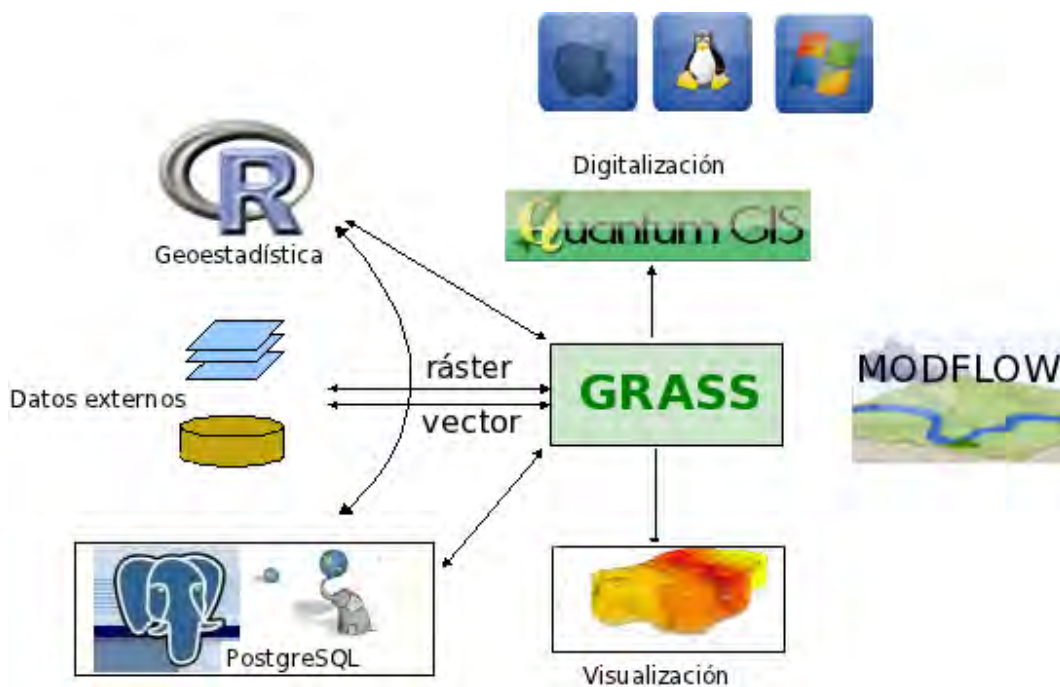


Figura 1.2: Integración de MODFLOW a SIG GRASS

Objetivos

El objetivo principal de este trabajo es el desarrollo de un software que permita la interacción del SIG GRASS con el modelo matemático MODFLOW. El software se desarrolló en el ambiente de SIG GRASS. Ésto permite realizar simulaciones matemáticas de modelos conceptuales, desarrollados en SIG GRASS, haciendo uso de MODFLOW. En este trabajo se presentan algunos ejemplos de simulaciones, realizadas con el módulo *r.gws* desarrollado para ilustrar cómo se utiliza, también se muestra un modelo conceptual de un sistema mucho más complejo, ésto para una explicación más extensiva de como se lleva a cabo una simulación matemática utilizando los paquetes de MODFLOW, ya que el módulo trabaja con los paquetes de manera interna.

Capítulo 2

Antecedentes

2.1. Ecuación de flujo para aguas subterráneas

2.1.1. Ley de Darcy

Henry Darcy en 1856, formuló la ley fundamental que describe el movimiento del agua en la zona saturada a través de un medio poroso. Mientras diseñaba los filtros de arena para el agua potable de la ciudad de Dijon, llegó a la conclusión de que el volumen de agua que fluye a través de un medio poroso por unidad de tiempo (caudal), es proporcional a la sección transversal A , a la diferencia entre cargas del fluido Δh en las superficies de entrada y de salida de la muestra e inversamente proporcional a la longitud de la muestra de arena o trayectoria del flujo (Villón-Béjar, 2006). Es decir

$$Q = -KA \frac{\Delta h}{L} = -KA \frac{h_1 - h_2}{L} \quad (2.1)$$

donde Q es el volumen de agua que atraviesa la muestra por unidad de tiempo, A es el área de la sección transversal, L es la longitud de la muestra, h_1 y h_2 son los potenciales en los puntos 1 y 2, Δh representa la pérdida de carga y K es la constante de proporcionalidad (conductividad hidráulica), este último término depende de la naturaleza del medio y el fluido. En la figura 2.1, se muestran las variables del sistema. El signo negativo se asigna por convención, ya que el gradiente hidráulico va de mayor a menor y $h_1 - h_2$ será negativo.

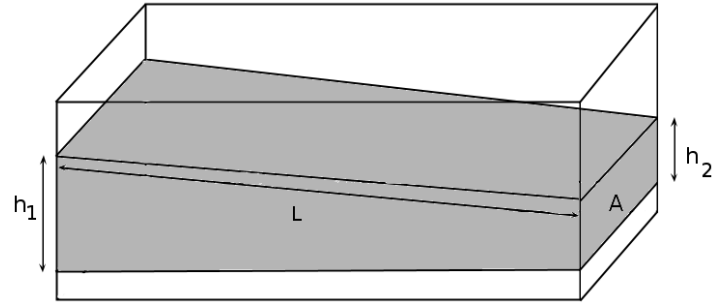


Figura 2.1: Ley de Darcy.

Se considera $v = \frac{Q}{A}$ como la velocidad de descarga, que es el caudal por unidad de sección transversal. Entonces

$$v = \frac{Q}{A} = K \frac{\Delta h}{L} \quad (2.2)$$

2.1.2. Derivación de la ecuación de flujo

El movimiento tridimensional del flujo de agua subterránea está descrito por la ecuación de flujo, la cual se deriva de la ley de Darcy y la ecuación de continuidad o conservación de masa (Martínez-Alfaro et al., 2006). Se considera un elemento diferencial de acuífero, con dimensiones dx , dy y dz , como se muestra en la figura 2.2

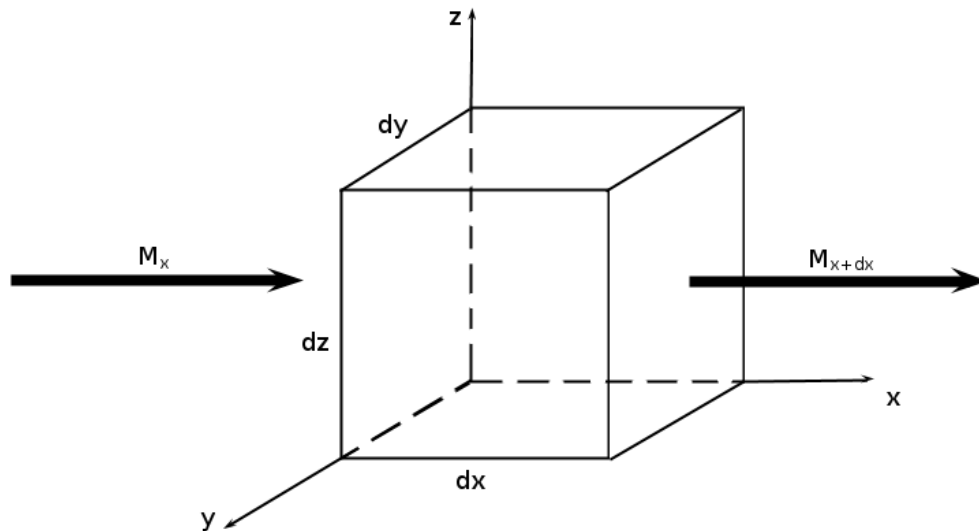


Figura 2.2: Volumen diferencial de acuífero.

Si aplicamos el principio de conservación de masa, la masa que entra y la que

sale deben ser iguales a la variación sobre el almacenamiento en el elemento. Sean las masas que entran y salen por unidad de tiempo

$$M_x = dy \cdot dz \cdot \vec{v}_x \cdot \rho \quad (2.3)$$

$$M_{x+dx} = dy \cdot dz \cdot \vec{v}_{x+dx} \cdot \rho \quad (2.4)$$

Tomando la diferencia de ambas masas

$$\Delta M_x = \Delta V_x \cdot \rho = \frac{\partial \vec{v}_x}{\partial x} \cdot dx \cdot dy \cdot dz \cdot \rho \quad (2.5)$$

Pero como se considera un elemento de acuífero unitario $dx \cdot dy \cdot dz = 1$

$$\Delta V_x = \frac{\partial \vec{v}_x}{\partial x} \quad (2.6)$$

Utilizando la ley de Darcy para la velocidad (ec. 2.2) donde $L = dx$ y considerando un medio homogéneo e isótropo ($k_x = k_y = k_z$)

$$\Delta V_x = \frac{\partial(K \cdot \frac{\partial h}{\partial x})}{\partial x} = K \frac{\partial^2 h}{\partial x^2} \quad (2.7)$$

Para las direcciones x y y se hace el mismo análisis

$$\Delta V_y = \frac{\partial(K \cdot \frac{\partial h}{\partial y})}{\partial y} = K \frac{\partial^2 h}{\partial y^2} \quad (2.8)$$

$$\Delta V_z = \frac{\partial(K \cdot \frac{\partial h}{\partial z})}{\partial z} = K \frac{\partial^2 h}{\partial z^2} \quad (2.9)$$

Por lo tanto para las tres direcciones finalmente se obtiene

$$\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} + \frac{\partial^2 h}{\partial z^2} \right) \cdot K = \Delta V \quad (2.10)$$

En el término de la derecha se representa la variación en el volumen almacenado, si tomamos en cuenta el coeficiente de almacenamiento específico S^* , como estamos considerando un acuífero de espesor unidad, el producto $S^* \frac{\partial h}{\partial t}$ expresa el volumen de agua que se gana o se pierde según la variación del potencial hidráulico a lo largo del tiempo. Por lo tanto la ecuación 2.10 puede ser expresada como sigue

$$\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} + \frac{\partial^2 h}{\partial z^2} \right) \cdot K = S^* \frac{\partial h}{\partial t} \quad (2.11)$$

Para un medio homogéneo y anisotrópico en estado transitorio, $k_x \neq k_y \neq k_z$

$$K_x \frac{\partial^2 h}{\partial x^2} + K_y \frac{\partial^2 h}{\partial y^2} + K_z \frac{\partial^2 h}{\partial z^2} = S^* \frac{\partial h}{\partial t} \quad (2.12)$$

Finalmente se considera la expresión más general de la ecuación de flujo en el que se considera un medio heterogéneo, anisotrópico y en estado transitorio. Se considera un término W que representa fuentes externas que agregan o extraen agua del elemento unidad del acuífero.

$$\frac{\partial}{\partial x} \left(K_x \frac{\partial h}{\partial x} \right) + \frac{\partial}{\partial y} \left(K_y \frac{\partial h}{\partial y} \right) + \frac{\partial}{\partial z} \left(K_z \frac{\partial h}{\partial z} \right) + W(x, y, z, t) = S^* \frac{\partial h}{\partial t} \quad (2.13)$$

2.2. MODFLOW

MODFLOW es un programa que resuelve el modelo matemático que describe el comportamiento de sistemas de flujo de aguas subterráneas. Fue desarrollado por el Servicio Geológico de los Estados Unidos (USGS por sus siglas en inglés) por Michael G. McDonald y Arlen W. Harbaugh, siendo el modelo de agua subterránea más usado en el mundo. Sus aplicaciones son ampliamente usadas en la hidrogeología para simulación de acuíferos en general. El programa está desarrollado en el lenguaje de programación fortran, es de código libre y puede ser utilizado en varios sistemas operativos, incluyendo Windows y Unix.

La primera versión de MODFLOW fue desarrollada entre los años 1981 y 1983 y fue llamada originalmente USGS Modular Three-Dimensional Finite-Difference Ground-Water Flow Model, pero algunos años después fue conocida comúnmente como MODFLOW. Este programa fue desarrollado originalmente en Fortran 66 y fue llamado MODFLOW-88 (Harbaugh, 2005). Desde entonces el programa ha sido ampliamente utilizado para estudios hidrogeológicos.

Existen hasta la fecha varias versiones de MODFLOW, en las que cada vez se pretende mejorar las capacidades del modelo y hacerlo cada vez más fácil de manejar para el usuario. La primera actualización fue MODFLOW-96, en el que se buscaba principalmente mayor facilidad de uso. Posteriormente se agregaron nuevos paquetes al modelo, principalmente el que permitía modelar también transporte de partículas, entre otras modificaciones importantes, como resultado se obtuvo MODFLOW-2000. Finalmente como una actualización a MODFLOW-2000 surge MODFLOW-2005, donde el principal cambio es el manejo de datos interno, pasando de ser manejado por medio de subrutinas a la utilización de módulos, los cuales son usados para poder ser compartidos entre subrutinas.

MODFLOW resuelve la ecuación de flujo por medio de un método numérico llamado diferencias finitas, este método es uno de los más conocidos, en soluciones

numéricas, para la resolución de ecuaciones diferenciales (ecuación 2.13). El método consiste en la discretización de un medio continuo, es decir, para resolverla se necesita discretizar el espacio y el tiempo, de tal forma que se tiene un conjunto de celdas a partir de las cuales se tiene un sistema lineal de ecuaciones (figura 2.4). Básicamente, las derivadas son aproximadas para formar ecuaciones lineales en cada celda. Estas aproximaciones están definidas por el espacio que hay entre celdas, convirtiendo el problema de ecuaciones diferenciales a uno de carácter algebraico (Zill, 2001).

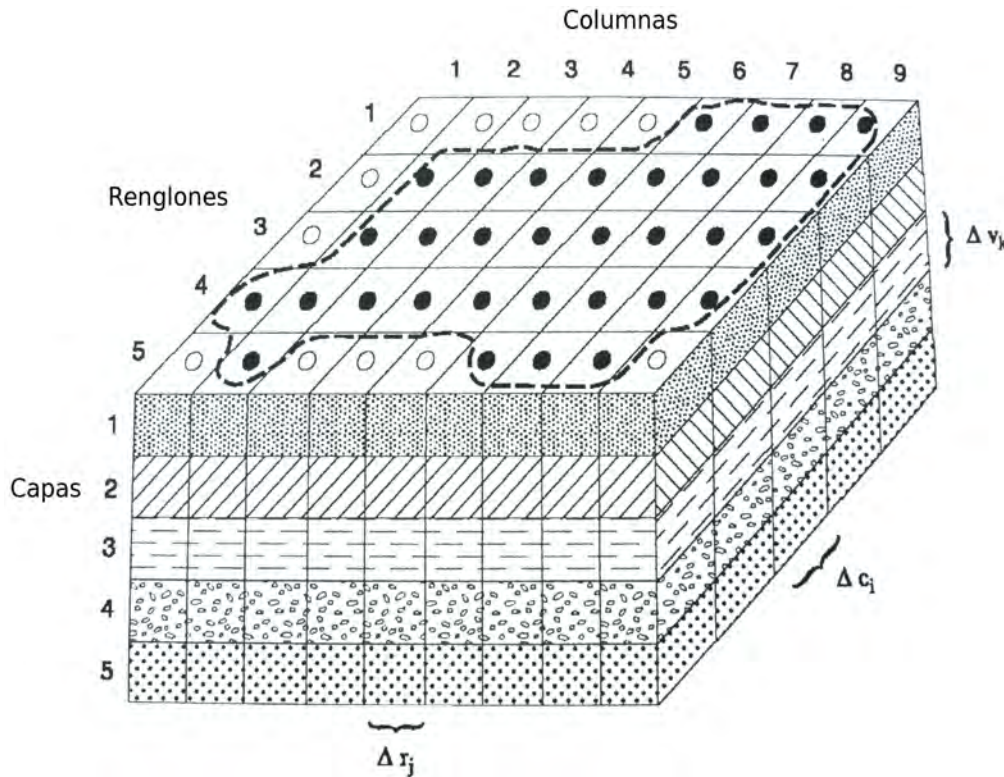


Figura 2.3: Discretización del espacio en MODFLOW.

Existen una gran cantidad de programas que realizan la simulación de sistemas de aguas subterráneas, sin embargo, existen varias razones por las cuales se escogió MODFLOW. Su código es libre y se encuentra disponible en la web, lo cual permite que no sea costoso. Este programa tiene amplias capacidades en la modelación, las cuales permiten simulación en estado transitorio y simulación de varios acuíferos, aparte de que es ampliamente usado y aceptado por la comunidad científica. Actualmente se han desarrollado diferentes interfaces para utilizar MODFLOW de manera más rápida y visual para el usuario, ejemplo de esto son Processing MODFLOW for Windows PMWIN (www.pmwin.net), UNCERT (www.uncert.com), Graundwater Vistas (www.groundwater-vistas.com), Visual MODFLOW (www.visual-modflow.com) y Groundwater Modeling System (www.ems-i.com), los cuales se pueden adquirir con un cierto costo. Sin embargo, ninguno de estos programas trabaja dentro de un SIG,

por lo que siendo SIG un importante complemento de trabajo para estudio de agua subterránea es necesario importar los datos de un programa a otro.

La idea es realizar un módulo en GRASS, que realice la simulación devolviendo como resultado, la distribución espacial de las cargas hidráulicas o sus velocidades. Una de las grandes ventajas de este módulo es que es de libre acceso, lo cual promueve la investigación sin necesidad de comprar software que requiere grandes cantidades de dinero. El módulo está desarrollado bajo MODFLOW-2005, el cual es compilado en gfortran versión 99. Gfortran es el compilador del lenguaje de programación FORTRAN, lenguaje en el que está escrito MODFLOW.

2.2.1. MODFLOW: entradas y salidas

MODFLOW está dividido en paquetes hidrológicos y de solución (Harbaugh, 2005) (cuadro 2.2.1). Existen dos distintos tipos de paquetes hidrológicos, los internos, los cuales simulan el flujo entre las celdas adyacentes, el otro tipo de paquete son los paquetes de estrés, los cuales simulan algún tipo de estrés individual (pozos, ríos y recarga por ejemplo). Por otro lado, MODFLOW incorpora múltiples paquetes para la solución del sistema de ecuaciones lineales derivadas del método de diferencias finitas, cada uno conforma un paquete distinto. Cada paquete requiere cierta información para poder ser ejecutado, por lo que para cada uno se debe tener un archivo de texto, el cual contiene una información específica.

Como se mencionó, para poder hacer uso de MODFLOW el usuario tiene que incorporar un archivo de texto para cada uno de los paquetes, para una simulación simple son necesarios seis diferentes archivos, los llamados NAM, BAS, DIS, BCF o LPF, PCG2 y OC. El archivo NAM (name file) es el archivo principal que lee MODFLOW, ya que dentro de él se describen las instrucciones para llamar a todos los demás paquetes. El archivo BAS (basic) es el que provee información acerca de las condiciones de frontera y las cargas iniciales. El archivo DIS provee toda la información acerca de la discretización del espacio para la resolución de la ecuación, mientras que los archivos BCF y LPF proveen la información necesaria del enfoque para formular los términos de flujo interno, es decir, como se determinará el flujo entre celdas contiguas. El archivo PCG2 contiene las instrucciones para el método de convergencia de gradiente conjugado precondicionado, como el número de iteraciones que se realizarán o el criterio de convergencia para detener la iteración. El archivo OC contiene toda la información de salida sobre como serán guardadas las cargas y el abatimiento o si serán guardados o impresos.

En el caso de que la simulación contenga otras características como pozos, ríos, lagos o exista evapotranspiración, otros archivos deben ser especificados tales como el archivo WEL, el cual contiene toda la información sobre los pozos tales como tasa de extracción y la celda en la que se ubica cada uno, así como también la capa de la que se extrae el agua; en el caso de los ríos el archivo es RIV. Para la recarga existe el archivo

Nombre del paquete	Abreviación	Categoría
Basic	BAS	Control del programa
Block-Centered Flow	BCF	Hidrológico/interno
Layer-Property Flow	LPF	Hidrológico/interno
Discretization	DIS	Hidrológico/interno
Horizontal Flow Barrier	HFB	Hidrológico/interno
Well	WEL	Hidrológico/interno
Recharge	RCH	Hidrológico/interno
River	RIV	Hidrológico/interno
General-Head Boundary	GHB	Hidrológico/interno
Drain	DRN	Hidrológico/interno
Evapotranspiration	EVT	Hidrológico/interno
Strongly Implicit Procedure	SIP	Solución
Precondicioned Conjugate Gradient	PCG	Solución
Direct Solution	DE4	Solución

Cuadro 2.1: Lista de paquetes para la simulación de agua subterránea.

RCH, el cual contiene la información sobre las zonas de recarga, así como también la cantidad. El archivo EVT, tiene información acerca de la transpiración de las plantas y evaporación de agua en el suelo. Para las especificaciones de cada paquete consultar la documentación de MODFLOW-2005 (Harbaugh, 2005).

Los archivos para poder utilizar MODFLOW son muy específicos, por lo que es costoso en tiempo trasladar la información de SIG, es decir, la descripción geoespacial del modelo, a archivos que MODFLOW pueda interpretar, para después volver con el resultado de MODFLOW y ser interpretados en SIG de nuevo. El proyecto pretende evitar la transferencia de datos, realizando todo esto de manera interna y sin salir de SIG, de forma que no haya que trasladar datos de un software a otro.

2.2.2. MODFLOW y SIG

La variabilidad espacial de los datos requeridos para la modelación de flujo subterráneo hace evidente la necesidad de utilizar los SIG. Algunos autores utilizan SIG para el procesamiento de todos los datos de la modelación para después usarlos para ser introducidos a MODFLOW. Esto tiene una desventaja muy grande, ya que es necesario pasar de un formato de datos al necesario para MODFLOW, después de esto el paso inverso es necesario para poder representar el resultado de una simulación en SIG. Esto hace importante la existencia de un software que integre estas dos herramientas para evitar la exportación e importación de datos.

Existen algunos autores que han ligado SIG con MODFLOW: (Orzol, 1997)

realizó un procesador de MODFLOW que trabaja con ARCINFO a través de FORTRAN y The Arc Macro Language (AML), (Winston, 2000) describe la interfaz GUI desarrollada usando Argus GIS. Los SIG que han sido ligados a MODFLOW requieren licencia, por lo que no son flexibles para ser obtenidos, a diferencia de GRASS, el cual es de libre acceso y se puede obtener de manera gratuita.

Hace seis años, Carrera-Hernández (2006) desarrolló el primer modelo del módulo en GRASS que utiliza MODFLOW. El módulo se desarrolló, en ese momento, con la versión MODFLOW-96, el compilador de fortran g77 y GRASS 6.0. La principal diferencia, entre el desarrollado anteriormente y el que se presenta en este trabajo, es la utilización de PostgreSQL para las bases de datos en las tablas asociadas a los mapas vectoriales del SIG GRASS. Por otra parte, se incluyen algunos otros paquetes como el de discretización (DIS), evapotranspiración (EVT) y el de resolución con propiedades de capa (LPF).

2.3. SIG GRASS

El SIG GRASS es un software creado por CERL (U.S. Army Construction Engineering Research Laboratory) en Champaign, Illinois en el año de 1982, inicialmente por la necesidad del gobierno de mejorar el análisis y manejo de sus tierras. Junto con el rápido desarrollo de las computadoras y los procesos de cada vez mayor velocidad, fue posible comenzar a utilizar esta capacidad de las computadoras para ser aplicado a procesos de información geográfica.

El uso principal de SIG GRASS fue militar, ya que el CERL tenía la misión de desarrollar la tecnología para el manejo de las instalaciones del departamento de defensa, las cuales necesitaban millones de acres de tierras para entrenamiento y pruebas militares (Neteler y Mitasova, 2007). Otros usos fueron para el manejo de vida silvestre, caza, pesca, pasto y producción agrícola. Posteriormente se agregaron algunos otros usos de prioridad como la protección de especies y hábitats, protección de sitios culturales y delimitación de zonas de impacto de ruido, artillería, contaminantes y sedimentos.

En el año de 1991 se publica el código en internet, lo cual permite la utilización de este software por distintas industrias, universidades y agencias gubernamentales, haciéndose cada vez más popular. Sin embargo, en 1997 CERL anuncia que dejaría de dar soporte, por lo que las Universidades de Hannover en Alemania, Baylor, en Texas y recientemente el ITC-irst (Centro per la Ricerca Scientifica e Tecnologica) en Italia se hacen cargo y continúan coordinando el desarrollo de SIG GRASS por un grupo de personas alrededor del mundo.

Actualmente SIG GRASS forma parte importante del desarrollo científico, proporcionando mayor facilidad para análisis de diferentes tipos de estudios de fenómenos geoespaciales. Esto es posible debido a que SIG GRASS tiene la capacidad de integrar vectores, rásters y procesamientos geoespaciales en un solo software, incluyendo her-

ramientas para el análisis estadístico, modelación, procesamiento de imágenes y una visualización sofisticada.

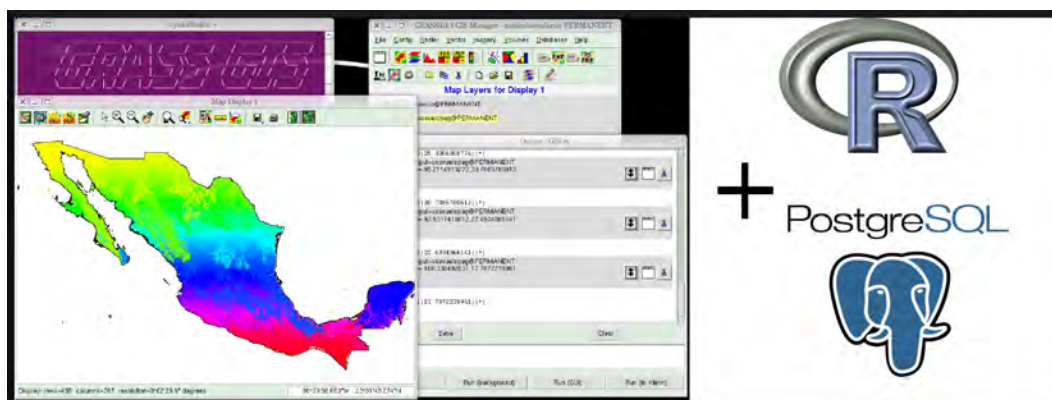


Figura 2.4: Desarrollo de nuevas tecnologías para la interacción de SIG GRASS con distintas herramientas de análisis.

2.3.1. SIG: Un software de código libre

Durante la última década los SIG se han desarrollado como una tecnología que ya no sólo se aplica a cierta rama de la ciencia, sino que es aplicable a casi cualquier aspecto de nuestra vida. Hasta hace algunos años era sólo utilizado por un grupo de investigadores, planificadores y trabajadores de gobierno que se preocupaban por algún tema específico, como la gestión de recursos naturales o manejo de desastres. Actualmente casi todo mundo puede crear mapas o hacer superposiciones teniendo acceso a SIG. Por otro lado, el rápido desarrollo de nuevos temas que se pueden tratar con SIG exige la creación de nuevas herramientas y conocimientos. Por lo tanto, la tecnología de los SIG abarca una amplia gama de aplicaciones, desde ver mapas e imágenes en la web hasta hacer análisis espacial, modelaciones y simulaciones.

La idea de un software de código abierto ha estado presente al menos desde que se hace el desarrollo de software. La investigación hecha en universidades y centros de investigación, han hecho posible la disponibilidad de software al público. Richard M. Stallman define software libre como cuatro libertades (Neteler y Mitsova, 2007):

- La libertad de correr el programa, para cualquier propósito.
- La libertad de estudiar como funciona el programa y adaptarlo a tus necesidades.
- La libertad de redistribuir copias.
- La libertad de mejorar el programa y publicarlo de tal manera que toda la comunidad se beneficie.

Bajo esta definición SIG GRASS es un software libre, por lo que para los propósitos de este trabajo es de gran utilidad, ya que se pretende precisamente realizar la modificación y mejoramiento del software. MODFLOW también cumple con las características de software libre, por lo que es posible tener como resultado final un software libre, el cual trabajará con dos programas de la misma naturaleza simultáneamente.

2.3.2. PostgreSQL

PostgreSQL es un sistema de gestión de base relacional de datos orientada a objetos, es libre y está publicado bajo la licencia BSD (Berkeley Software Distribution), esto quiere decir que permite la distribución libre y flexible.

En los últimos años la utilización de grandes bases de datos para manipulación y análisis ha ido aumentando rápidamente, por lo que se han creado distintos programas para gestión de las mismas. En el caso de PostgreSQL se tiene la ventaja de ser gratuito y de código abierto, característica por la cual interactúa con SIG GRASS. En las últimas versiones de SIG GRASS viene incluido PostgreSQL como un gestor de las tablas asociadas a los mapas vectoriales, los cuales contienen toda la información de cada punto en el mapa en forma de tabla PostgreSQL.

Bases de datos relacionales. Como ya se mencionó anteriormente, PostgreSQL maneja las bases de datos relacionales, esto quiere decir que se puede tener una cierta cantidad de tablas relacionadas entre sí por un identificador propio de cada dato. Esto permite tener distintos tipos de tablas asociadas a una serie de datos, clasificadas de forma temática, lo cual es útil cuando se cuenta con una gran cantidad de información, esto permite mantener un mayor orden y manejo de los datos.

En SIG GRASS los mapas vectoriales tienen una tabla con información asociada, que a su vez, tienen cada uno de los datos georeferenciados en la zona de estudio, esto da grandes ventajas en el área del flujo subterráneo, muestra de ello es la gran cantidad de pozos que se pueden tener en una ciudad o localidad determinada. Gracias a esto es posible desarrollar el software al que va dirigido este trabajo basando la información vectorial en tablas PostgreSQL, para ellos es necesario tener conectado SIG GRASS al gestor de datos PostgreSQL y asociar a los vectores la tabla de tipo PostgreSQL (Matthew y Stones, 2005).

Capítulo 3

Metodología

3.1. Módulo en GRASS

El módulo desarrollado en el software SIG GRASS, es una herramienta que tiene como finalidad mostrar de manera gráfica las simulaciones que se pueden llevar a cabo de flujo subterráneo, específicamente en acuíferos de una zona determinada. Es por esto que dentro de SIG GRASS el módulo lleva como nombre *r.gws* (GroundWater Simulation, por su nombre en inglés).

Existen dos tipos de mapas en GRASS (figura 3.1), los vectores que son representados por líneas, puntos y polígonos en el espacio y los mapas ráster que la información en forma de matrices con entradas que representan a cada celda.



Figura 3.1: Tipos de mapas que representan información georeferenciada.

Con estos dos tipos de mapas se va a adquirir la información que pide el módulo. Una característica importante es la capacidad de SIG GRASS de convertir un mapa vectorial en un mapa ráster y viceversa.

La figura 3.3 muestra de manera general el comportamiento del módulo que se desarrolló. La información de los mapas realizados en GRASS serán los encargados de proporcionar los datos que MODFLOW procesaría para correr una simulación.

3.1.1. Interfaz del módulo

La figura 3.2 muestra la interfaz gráfica, que forma parte del SIG GRASS. Mediante la interfaz, el usuario introduce la información en los espacios correspondientes, en el enunciado superior al espacio, se indica la información que se debe introducir, así como también el tipo de dato que debe contener la información.

A la vez que el usuario integra la información de su modelo, la interfaz va creando la orden de línea de comando, por lo que existe la opción de correr la simulación mediante la terminal. En la pestaña *output*, se muestra la evolución del procedimiento que se está llevando a cabo, así como en caso de presentarse algún error.



Figura 3.2: Interfaz mediante la cual, se introduce la información del modelo.

Para poder desarrollar un módulo dentro de GRASS, se necesita realizar la programación en C. El programa del módulo *r.gws*, utiliza una serie de librerías del SIG GRASS. Mediante las librerías, es posible utilizar las funciones que permiten que

el programa interactúe con mapas vectoriales y ráster en GRASS, bases de datos en PostgreSQL y correr simulaciones con MODFLOW desde *r.gws* (Ver apéndice A).

3.2. Información vectorial

En el caso de los mapas de tipo vectorial, se tiene la ventaja de trabajar con tablas y las bases de datos de PostgreSQL, esto desde las versiones 6.x de SIG GRASS. Estas bases de datos permiten el manejo de grandes cantidades de información, que a su vez, puede ser manipulada geoestadísticamente por medio del programa R (software diseñado para el análisis geoestadístico), ya que dicho software interactúa con SIG GRASS. Este tipo de mapa también permite algunos otros algoritmos, como la de interpolación de datos, en especial con análisis Kriging, el cual se utiliza en el caso de datos georeferenciados.

Un ejemplo sistematizado del manejo de vectores se muestra en la figura 3.3 (esquina superior derecha), donde se tiene un vector que contiene la información de la medición puntual de cargas hidráulicas, para utilizar el módulo es necesario conocer la distribución espacial de las cargas hidráulicas, por lo que es posible dentro de SIG GRASS, utilizar una herramienta de métodos de interpolación (kriging, spline, etc.) que obtenga la mejor distribución espacial para ser introducida al módulo en forma de un mapa ráster de las cargas hidráulicas.

Por otra parte, los datos puntuales como pozos o zonas de drenaje, se ingresan al módulo mediante mapas vectoriales, con la herramienta PostgreSQL, que es un gestor de bases de datos relacionales, y que puede ser manipulado desde el SIG GRASS. Esto se debe a que la información de cada uno de los puntos en el mapa vectorial tiene asociada una o varias tablas, estas tablas tienen la información de cada punto en forma de columnas, en estos casos el módulo *r.gws* pide el nombre del mapa vectorial en el que se encuentran los pozos o las zonas de drenaje, así como también el nombre de las columnas que contienen la información, por ejemplo, en el caso de los pozos, se pide la columna que contiene la tasa de extracción, la capa de donde se extrae el agua de cada pozo, si el pozo es activo en cada período de estrés; en el caso de drenaje, se pide la columna con la elevación de drenaje, la capa donde está el drenaje y la conductividad hidráulica entre el drenaje y el acuífero (ver cuadro 5.1).

3.3. Información a partir de mapas ráster

En el caso de los mapas ráster (izquierda de la figura 3.3), se tiene el manejo de datos en forma de matrices, que en términos de un mapa en SIG GRASS equivale a información para cada celda de la región, esto permite tener una distribución espacial de alguna característica específica de la zona.

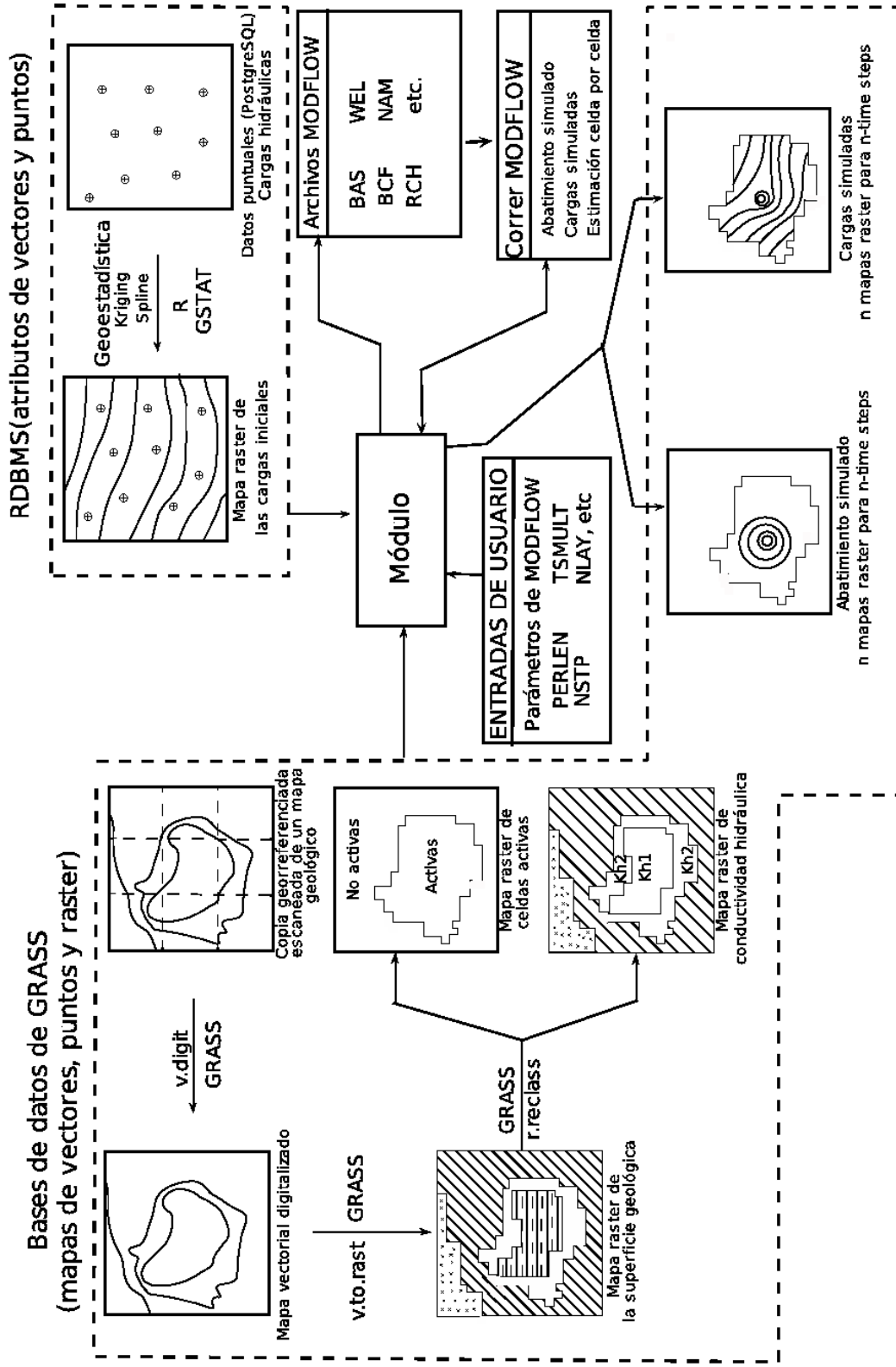


Figura 3.3: Módulo desarrollado para SIG GRASS usando MODFLOW (adaptado de Carrera-Hernández (2006)).

El ejemplo que se presenta es la superficie geológica de la zona, extraída de un mapa físico y escaneado para ser introducido en SIG GRASS. Esto es posible mediante la herramienta de digitalización de mapas de SIG GRASS *v.digit*, el resultado es un mapa vectorial con la distribución de la geología en forma de polígonos. Este mapa se transforma en ráster para finalmente clasificar las superficies de acuerdo a lo que el módulo exige, de manera análoga es posible obtener mapas de distribución de conductividad hidráulica y las condiciones de frontera.

Una vez obtenidos los mapas que el módulo exige en SIG GRASS, éstos se introducen, ya sea de manera gráfica o mediante línea de comandos para realizar la simulación. En el cuadro 5.1, en el capítulo 5, se muestra las entradas que exige el módulo en forma de ráster, en forma de vector o como valores escalares. Los valores escalares son entradas que el usuario tiene que especificar en el módulo como PERLEN, que es el número de períodos de estrés, NSTP, que es el número de pasos de tiempo para cada período de estrés, NLAY, el número de capas del modelo, etc.

3.4. MODFLOW en el módulo

El módulo *r.gws* interactúa con MODFLOW de manera interna, es decir, una vez que se han desarrollado cada uno de los mapas ráster y vectorial necesarios para una simulación, basta con introducirlos en la interfaz de SIG GRASS del módulo para obtener como resultado el abatimiento simulado y las cargas simuladas para cada paso de tiempo en cada período de estrés.

Como ya se ha mencionado anteriormente, MODFLOW requiere de diferentes paquetes para funcionar, los paquetes para cada modelo (BAS, WEL, BCF, RCH, etc.) dependen de la simulación que se desea llevar a cabo. Estos paquetes son creados por el módulo *r.gws* gracias a la información introducida por el usuario, el siguiente paso es llamar a MODFLOW (que ya debe estar instalado), esto arrojará resultados que el mismo módulo interpretará y convertirá para importarlos a SIG GRASS. Así el usuario podrá visualizarlos en forma de ráster, estos mapas son el abatimiento y las cargas en la figura 3.3.

Los paquetes que el módulo *r.gws* puede crear internamente en base a la información del usuario son: BAS, DIS, BCF, LPF, OC, PCG, WEL, RCH, DRN, EVT y RIV.

Existen algunos requerimientos para poder instalar MODFLOW, de tal manera que interactúe con el SIG GRASS. Para utilizarlo, es necesario contar con la instalación de GRASS versión 6.4, el compilador de FORTRAN gfortran y MODFLOW-2005. En el apéndice B, se detalla la instalación de MODFLOW-2005 en linux, para ser utilizado por el módulo *r.gws*.

Capítulo 4

Modelación numérica de aguas subterráneas

Uno de los principales propósitos del proyecto es que se puedan realizar simulaciones de flujo de agua subterránea en sistemas reales, ya que MODFLOW ha sido utilizado para modelar diferentes tipos de acuíferos, más adelante se mencionan algunos de los trabajos ya realizados. En esta sección se dará una explicación de la aplicación de los paquetes de MODFLOW a cada una de las propiedades de un sistema, partiendo del modelo conceptual, el cual tiene el propósito de brindar una idea general del funcionamiento del acuífero, esto permitirá modelar matemáticamente un sistema de flujo subterráneo.

Los modelos matemáticos, han sido aplicados para conocer el comportamiento de un acuífero, es por esto que el desarrollo de MODFLOW significó un gran paso para poder hacer modelación de flujo subterráneo. A partir de ese momento se han creado un gran número de reportes sobre estudios de acuíferos con esta herramienta, ejemplo de ello se muestra en artículos y tesis de posgrado realizados en distintas partes del mundo.

Calvache-Quesada y Pulido-Bosch (1990) desarrollaron un modelo para simulación matemática en dos dimensiones del flujo subterráneo en Río Verde, Granada. El objetivo de su trabajo consiste en conocer el comportamiento del acuífero para determinar si la calidad de agua se puede ver afectada por la disminución en los niveles piezométricos. Llevan a cabo simulaciones en estado estacionario y transitorio utilizando la versión de MODFLOW-88. Unos años más adelante, se realizó otro estudio en la misma zona por *García-Aróstegui et al. (2001)*, donde simulan el comportamiento de los recursos hídricos en una época de sequía que se lleva a cabo entre los años 1990/1991-1993/1994, el objetivo principal de su trabajo es la cuantificación y caracterización de la zona durante el período de sequía, este último estudio se realiza con el software Visual MODFLOW v.2.81.105 (<http://www.ground-water-models.com>), el cual es una plataforma visual para realizar las simulaciones de MODFLOW. Ambos

estudios se realizan en la misma zona de estudio, utilizando MODFLOW para dos propósitos específicos distintos.

MODFLOW, por ser un modelo robusto, se ha utilizado ampliamente, demostración de esto es la serie de artículos publicados en distintas revistas de ciencias en el mundo. En la India, los autores *Ahamed y Umar (2009)*, utilizaron la simulación matemática para conocer el comportamiento del flujo subterráneo, y así evaluar el balance hidrológico, se hace uso del software Visual MODFLOW, Pro 4.1. Recientemente se publicaron trabajos como el de *Flores-Márquez et al. (2011)*, en donde se hace la simulación matemática del acuífero de San Luis Potosí utilizando un modelo conceptual de siete capas. Otros trabajos de modelación matemática, con uso de MODFLOW como herramienta, son los de *Bandani y Moghadam (2011)*, *AL-Fatlawi (2011)*, *Al-Hassoun y Mohammad (2011)* y *Panagopoulos (2012)*.

Existen también estudios realizados como tesis de maestría, que detallan el desarrollo de una simulación matemática en distintas condiciones, como *Jackson (2007)* en Australia, o *Shribru-Wake (2008)* en Bélgica y más recientemente *Gallegos (2011)* en Florida, donde cada uno de estos describe un sistema de acuífero con propiedades completamente diferentes.

Estudios como el de *Flugel y Michl (1995)*, demuestran la importancia del trabajo de un SIG como complemento de MODFLOW en el estudio de agua subterránea. En dicho artículo se trabaja con SIG IDRISI, para ser ligado a MODFLOW, y así modelar el acuífero aluvial de Río Sieg en Alemania. Todos estos estudios están dirigidos al buen manejo de los recursos hídricos en cada una de las zonas descritas, es por esto que algunas instituciones también utilizan este medio para desarrollar algunos de los escenarios que se podrían presentar para hacer mejor uso de la disponibilidad de agua, ejemplo de esto es un reporte realizado por la Corporación Autónoma del Valle de Cauca (CVC) en Colombia (*Gutiérrez-Enríquez y Aristizabal-Rodríguez (2006)*).

Debido al potencial de la herramienta desarrollada en esta tesis, para ayudar en el adecuado manejo de los recursos hídricos, en esta sección se menciona cómo se podría modelar de forma numérica, el acuífero de San Luis Potosí, mostrando la utilidad del módulo *r.gws*, para realizar simulaciones numéricas en MODFLOW.

4.1. Descripción general

Debido a que la finalidad de este trabajo es el desarrollo del software de SIG con MODFLOW, el ejemplo que se presenta en este capítulo muestra, de manera general, cómo se realizaría modelación matemática para un sistema tan complejo como el del Valle de San Luis Potosí.

El Valle de San Luis Potosí es un sistema ampliamente estudiado, sin embargo, para realizar una simulación del mismo se requiere de amplios conocimientos de la

zona, así como una gran cantidad de información, superior a la que se presenta en esta explicación, por lo cual la simulación matemática escapa de los objetivos principales del trabajo. Sin embargo, el propósito de mostrar el ejemplo, es analizar con más detalle, desde el desarrollo de un modelo conceptual, hasta la descripción de los paquetes que utiliza MODFLOW para llevar a cabo la resolución de las ecuaciones.

El Valle de San Luis Potosí (SLP) se encuentra en el suroeste del estado del mismo nombre, la zona comprende la capital del estado y los municipios de Soledad de Graciano Sánchez y Cerro de San Pedro, así como también una pequeña parte de los municipios de Mexquitic de Carmona y Zaragoza. El valle cuenta con una altura promedio de 1850 msnm, y tiene una extensión de 1980 km². La zona está limitada al oeste por la Sierra de San Miguelito y al este por la Sierra de Álvarez, con alturas entre 2780 msnm y 2300 msnm, respectivamente. Al norte limita la Sierra de la Melada y al sur se forma un parteaguas entre la cuenca de Jaral de Berrios-Villa de Reyes y la sierra de San Miguelito.

El Valle de San Luis Potosí presenta un clima templado, con lluvia principalmente en los meses de verano, que alcanza un promedio cercano a 360 mm/año en la planicie y hasta 450 mm/año en la sierras. El período con un máximo de precipitación comprende los meses de junio a septiembre, donde se presentan un 65 % de la precipitación total anual. La temperatura media anual es de 17.5°C, con un máximo en Mayo y Junio de 21°C, y un mínimo de 13°C en enero (Hergt, 2009). La zona se encuentra en una región semiárida, se calcula que existe una evaporación potencial de 2038.7 mm (López-Álvarez, 2012).

El Valle de San Luis Potosí cuenta con una distribución geológica de rocas sedimentarias y volcánicas (figura 4.1), en la zona este se encuentra una formación de calizas del Cretácico, las cuales conforman las formaciones Indidura y cuesta de Cura, las rocas de esta formación tienen una estratificación delgada, con origen de cuenca marina con algunas capas arcillosas, que hace que la conductividad hidráulica en esa zona sea muy baja, debido a estas características, algunos autores consideran que estas rocas constituyen el basamento hidrogeológico. La geología de la zona, provee información necesaria para el modelo matemático, tal como la distribución espacial de las conductividades hidráulicas o las zonas donde no hay flujo.

Sobre la zona de calizas se encuentran rocas volcánicas del Terciario, tienen un espesor máximo de 500m y están cubiertas por material granular semi-consolidado de espesor variable (López-Álvarez, 2012). En la zona oeste se encuentra la sierra de San Miguelito, constituida principalmente por roca volcánica con relleno de material granular, este material se extiende a lo largo del valle, donde se encuentra material aluvial sobre el material granular. En la zona centro del Valle se encuentra un horizonte constituido por arena fina y limo, con un espesor de 50 a 150m (Hergt, 2009). Sobre este material se encuentran aluviones del cuaternario, las rocas que conforman esta capa son arenas, gravas y limos que se intercalan con basaltos.

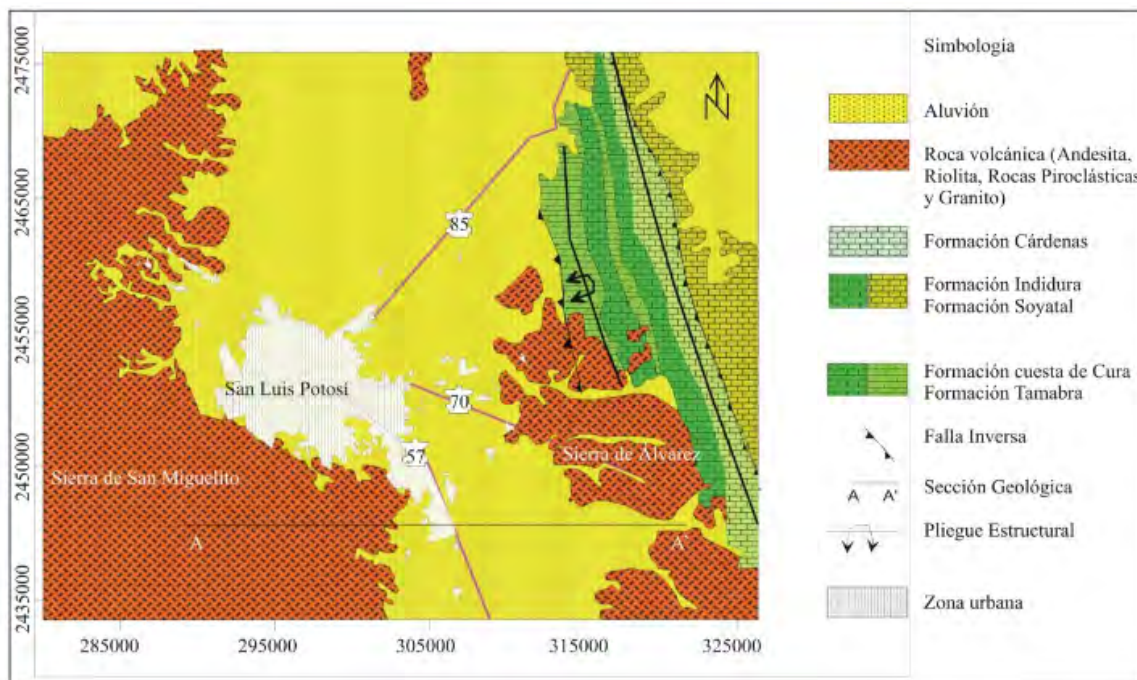


Figura 4.1: Geología del Valle de San Luis Potosí (Extraído de *López-Álvarez (2012)*).

4.1.1. Condiciones históricas

Después de que se establecieron los primeros grupos étnicos en el Valle de San Luis Potosí, se comienza a poblar la zona debido al descubrimiento de algunos de los yacimientos mineros (plata y oro) más importantes del país, así como también la presencia de cuerpos de agua. Ésto da comienzo al desarrollo y crecimiento de la ciudad, con lo que se crea una de las formas más importantes de consumo de agua en el siglo XVII, la agricultura, que en aquellos momentos se lleva a cabo en zona urbana con la creación de huertos, principalmente de nopal, maíz, tomate, avena, chile, frijol y pastizales replantados, posteriormente. Una tercera actividad aparece en el siglo XX, la industria, que surge para formar parte importante de la economía de la ciudad (López-Álvarez, 2012). Inicialmente la ciudad se abastecía por cuerpos de aguas superficiales, hasta que el crecimiento comenzó a necesitar de más fuentes de agua, con lo que se comienza a extraer el agua subterránea.

A partir de entonces en la ciudad de San Luis Potosí comienza el mayor crecimiento en sus actividades económicas y como consecuencia la población aumenta. Actualmente se tiene conocimiento de la alta tasa de extracción de agua subterránea y que debido a eso, se han creado algunos conos de abatimiento, siendo el principal en la zona urbana.

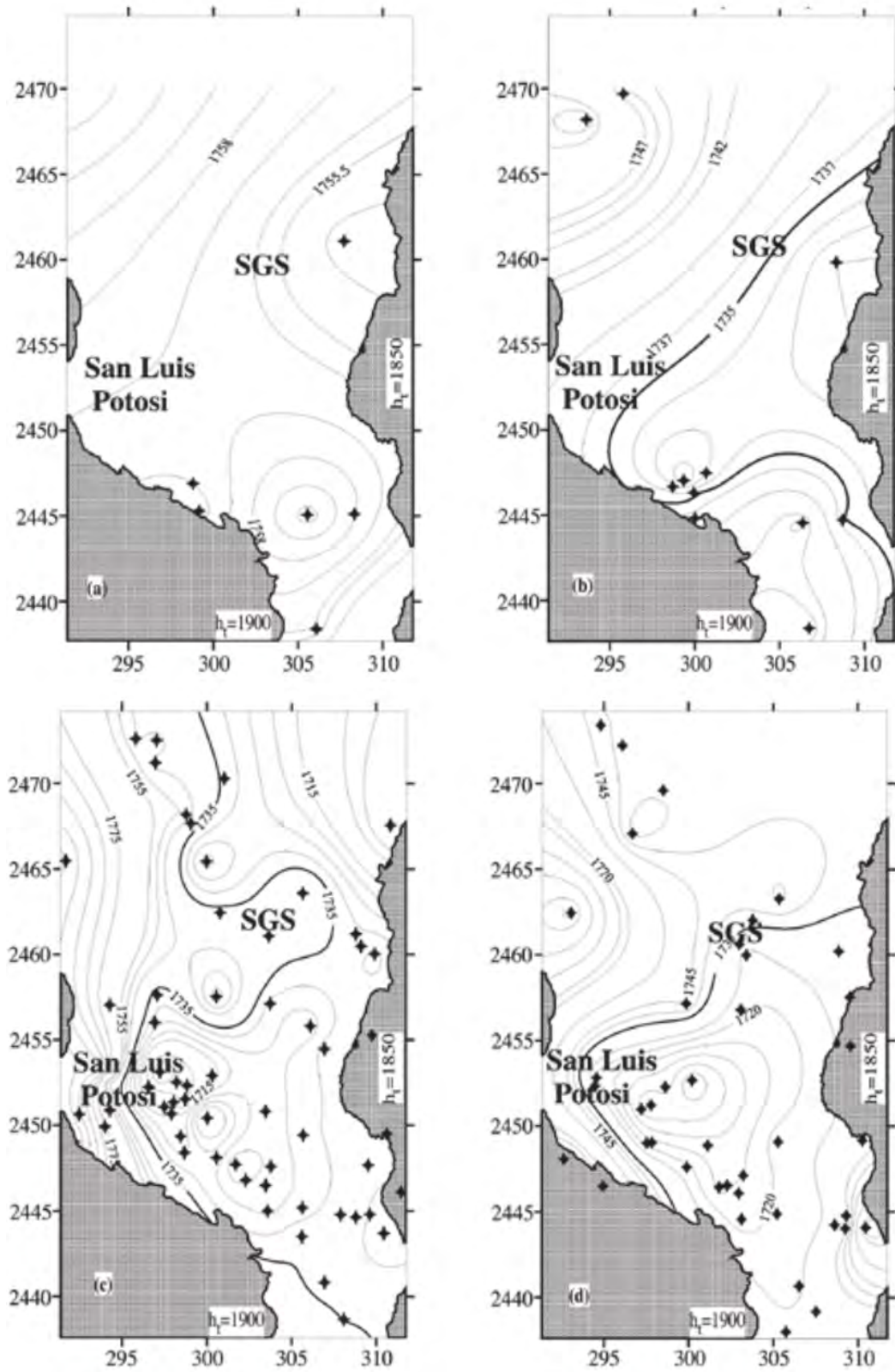


Figura 4.2: Niveles potenciales en los años a)1977, b)1992, c)1999 y d)2003 (extraído de *Flores-Márquez et al. (2011)*).

El crecimiento poblacional sigue aumentando y el acuífero sigue siendo explotado, de acuerdo a censos realizados en el estado de San Luis Potosí, en el 2010 (INEGI,2012), se contabilizaron 2 millones 585 mil 518 personas, de las que en el área de estudio 772, 604 viven en el municipio de San Luis Potosí, 267, 839 en Soledad de Graciano Sánchez, 53, 442 en Mexquitic de Carmona y 4, 021 en Cerro de San Pedro. En el año 2005, la población sólo en el municipio de San Luis Potosí fue de 730, 950 personas, ésto quiere decir 40, 000 personas más en cinco años, lo cual indica un crecimiento constante en la demanda de agua para consumo humano. Por ésta razón, es importante conocer el comportamiento del acuífero, ya que es necesario un manejo correcto de los recursos hídricos del valle.

Para estudiar el acuífero de la ciudad de San Luis Postosí, es importante conocer los registros históricos de extracción de agua. Se sabe que la extracción de agua del acuífero profundo comienza en los años cuarenta, sin embargo, no existe mucha información sobre el suministro de agua en esa época. Actualmente existen algunos datos históricos recolectados sobre los niveles de agua en los pozos en los años 1977, 1992, 1999 y 2003, estos datos fueron extraídos de *Flores-Márquez et al. (2011)* y se muestran en la figura 4.2.

En la figura 4.2 se observan los niveles piezométricos en el acuífero para distintos años, para 1977 prácticamente no existe extracción del acuífero, los niveles son muy estables y el número de pozos es muy pequeño. En el año 1992 se tiene un aumento pequeño en el número de pozos, sin embargo se puede distinguir una disminución en los valores de los niveles. El cambio a partir de 1999 es más notorio, ya que el número de pozos aumenta considerablemente y los niveles disminuyen en mayor proporción, en el registro para el año 2003 se puede observar una variación pequeña en comparación con 1999.

4.1.2. Modelo conceptual

Con base a la geología del valle, se distinguen dos acuíferos principales, los cuales son denominados acuífero colgado y acuífero profundo; ambos acuíferos están separados por una capa impermeable de arcilla y limo. El acuífero profundo consta de dos materiales distintos, por lo que se puede subdividir en dos acuíferos, uno de material granular que se encuentra sobre uno de roca volcánica fracturada, el cual tiene la mayor conductividad. Los cálculos con base de geotermómetros indican que el basamento se encuentra a una profundidad de 1400 m a 2100 m (Hergt, 2009).

El acuífero colgado tiene comportamiento libre y está compuesto de material clástico, se encuentra ubicado sobre este cuerpo compacto con un espesor máximo de 100 m y cubre un área de 300 km², el nivel de agua tiene una profundidad de entre 5 y 40m. El acuífero profundo está compuesto por material sedimentario que rellena la fosa, denominado TGI (tobas y sedimentos clásticos) y rocas volcánicas fracturadas; este acuífero es de tipo semiconfinado, encontrándose bajo la capa de material fino de

limo y arena. La elevación del nivel estático en el acuífero colgado es de 1815 a 1880 msnm, mientras que la del acuífero profundo es de 1715 a 1760 msnm. Se considera que la capa compacta que divide a los acuíferos tiene una conductividad muy baja que impide la conexión entre ellos. La figura 4.3, muestra la distribución vertical del nivel de los acuíferos y las rocas que los conforman. Debido a que la distribución, a lo largo de la superficie del valle del acuífero colgado, es limitada, el acuífero profundo en su parte granular se comporta como un acuífero libre en las zonas donde no existe la capa confinante superior.

Según la información de la figura 4.3, es posible considerar tres distintas capas en el modelo, la primera será la correspondiente al material de aluvi3n, en el que se encuentra el acuífero colgado; a esta capa se le asociará un lecho confinante, que corresponde a la capa de limo y arena con conductividad hidráulica muy baja. La segunda capa, será la compuesta por el material granular (TGI), que forma parte del acuífero profundo; finalmente, se toma una capa correspondiente a la roca volcánica fracturada.

EL acuífero colgado tiene una conductividad hidráulica de 1.2×10^{-3} m/s, y un coeficiente de almacenamiento de 0.15. El acuífero profundo tiene una conductividad que va de 0.1 m/d a 50 m/d, en la parte granular, y de 2 m/s a 10 m/s, en la parte fracturada (L3pez-3lvarez, 2012).

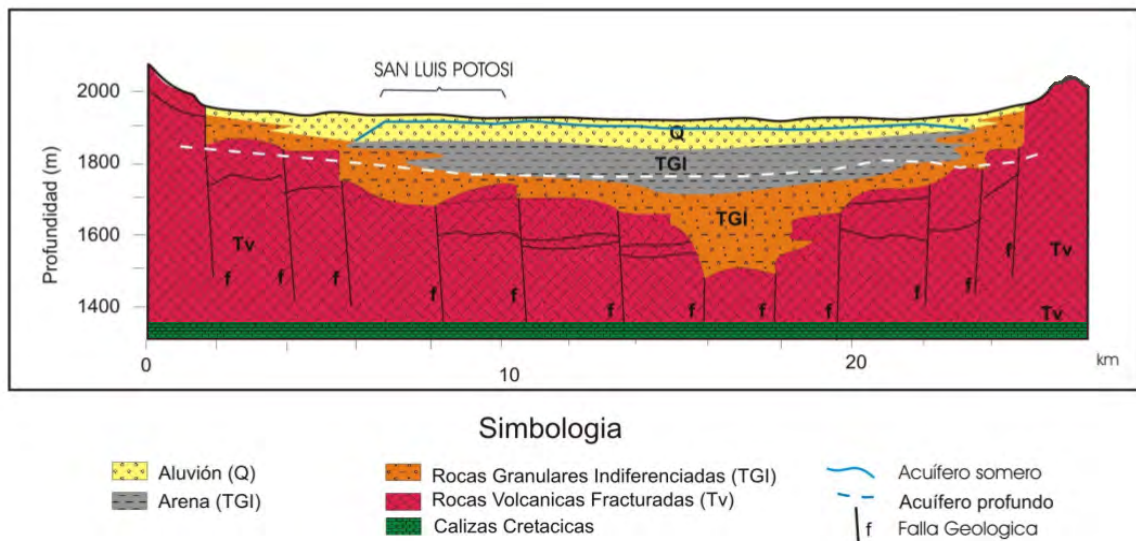


Figura 4.3: Distribuci3n vertical de los acuíferos (extraído de *Hergt (2009)*).

El acuífero colgado tiene un comportamiento muy dinámico, debido a que su recarga depende directamente de la precipitaci3n, evapotranspiraci3n, escurrimiento, pendiente del terreno y en algunos casos de fugas de agua potable o retornos de riego (L3pez-3lvarez, 2012).

El acuífero granular ha sido el más explotado, es donde se encuentra la mayoría de los pozos y alcanza profundidades de hasta 350 m de material sedimentario. Se encuentra entre 100 y 150 m de profundidad. Este acuífero tiene comportamiento confinado en el centro del valle y libre en las proximidades.

El acuífero profundo, en su parte de roca volcánica, debido a las fosas y pilares que lo conforman, tiene una distribución espacial muy irregular. Algunos de los pozos que extraen agua de este acuífero, tienen unas profundidades de 350 a 450 m, existiendo unos de hasta 800 a 1200 m.

La recarga del acuífero profundo del Valle de San Luis Potosí, es un tema discutido, ya que no existe evidencia clara de la procedencia de la misma. Por una parte, CONAGUA presenta valores de hasta 78.1 Mm^3 anuales, mientras que López-Álvarez (2012) estima una recarga de 0.042 Mm^3 anuales. En la actualidad, existen algunos estudios donde se dan a conocer algunas evidencias del comportamiento del flujo en el acuífero profundo. Cardona-Benavides et al. (2006), muestran las edades obtenidas de las aguas que conforman el acuífero profundo concluyendo que se trata de agua antigua, es decir que data de más de 1000 años de antigüedad; ésto implicaría que la recarga local del acuífero profundo es mínima, mientras que la recarga regional proviene de lugares muy lejanos o es muy lenta. Por otra parte, la recarga que podría proceder de las sierras, tendría que ser muy pequeña, ya que ambas Sierras presentan, por razones distintas, características de baja permeabilidad (Noyola-Medrano et al., 2009). La Sierra de San Miguelito, constituida por roca volcánica cuenta con un relleno en las fracturas que impide el paso del agua, mientras que la Sierra de Álvarez está conformada por formaciones de calizas que impiden que el agua se infiltre hacia el valle. En la figura 4.4 se observan las formaciones geológicas de las Sierras y el valle.

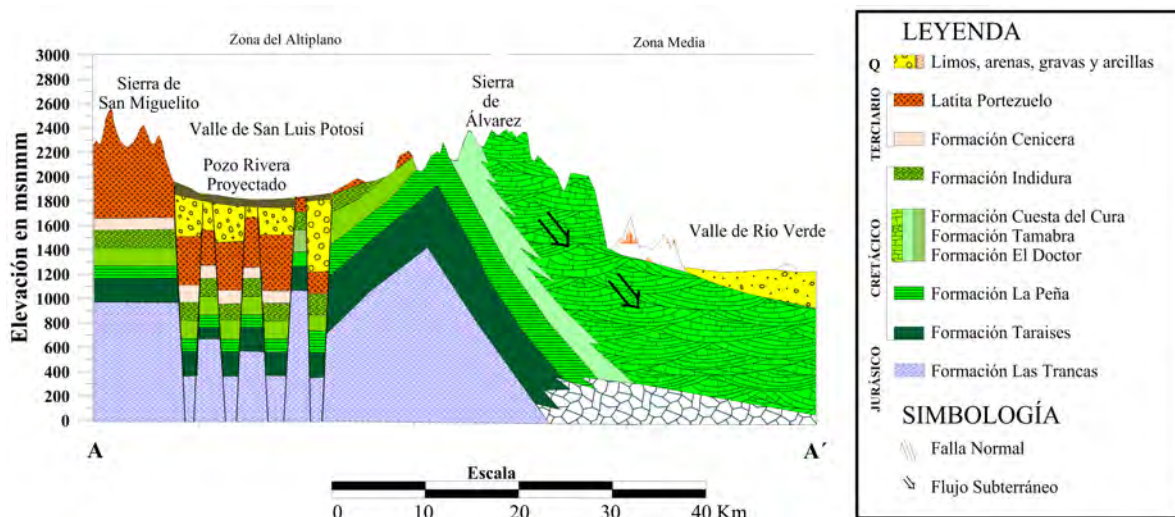


Figura 4.4: Sección geológica del Valle de San Luis Potosí

El modelo conceptual, representa el primer y más importante paso para la

simulación matemática de flujo subterráneo, ya que en base a éste se determinará información importante como las zonas de recarga y descarga, las condiciones de frontera, el número de capas, etc. Para un sistema hidrogeológico es posible tener más de un modelo conceptual, esto quiere decir que la determinación del modelo no es definitiva, sino que depende en gran medida de la persona que lo lleve a cabo, así como también de toda la información extra con la que se cuenta, los sistemas hidrogeológicos son perfectibles, debido a que continuamente se están estudiando y obteniendo nuevos datos, los modelos pueden ser mejorados. En el caso específico de San Luis Potosí, se tiene al menos dos modelos distintos del acuífero, en el caso de *Flores-Márquez et al. (2011)*, se menciona un modelo con siete capas, mientras que en *López-Álvarez (2012)* se realiza el modelo con dos capas, así es posible encontrar una serie de diferencias para un mismo sistema, es por esto que la experiencia de quien realiza un estudio de esta naturaleza influye mucho en la creación del mismo.

4.2. Creación del modelo matemático

Para poder desarrollar una simulación matemática con MODFLOW, es necesario identificar los datos que el modelo requiere para resolver la ecuación. Como ya se mencionó anteriormente, MODFLOW requiere de un conjunto de paquetes, los cuales corresponden a un archivo de texto cada uno. Para generar los archivos, es necesario conocer las propiedades del sistema, es por esto que primero se desarrolla un modelo conceptual en el que se presente una idea de como funciona el sistema del que se pretende un modelo matemático. Los archivos que se crean, a partir de la información, contienen un formato muy estricto, por lo que es importante que cada dato se encuentre en el lugar indicado (figura 4.5). En el caso del Valle de San Luis Potosí, una simulación con MODFLOW, requeriría, al menos, los paquetes BAS, DIS, BCF, RCH, WEL y EVT. Los períodos de estrés y los pasos de tiempo podrán ser definidos por el usuario dependiendo de los propósitos del estudio que se realiza.

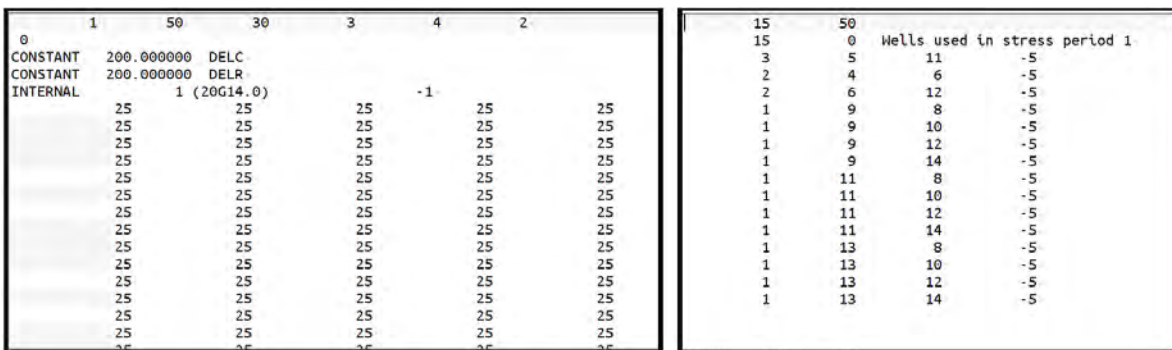


Figura 4.5: Archivos creados por r.gws: a) Archivo de discretización y b) Archivo de pozos.

Paquete de discretización (file.dis)

Como ya se mencionó anteriormente, la zona de estudio debe ser discretizada, para el archivo que corresponde al paquete de MODFLOW, DIS, se tiene que determinar la resolución a la que se desea trabajar, así como también los límites de la zona, esto dará la información del número de celdas del modelo. Para la discretización del tiempo, se escribe el número de períodos de estrés, ya que este paquete le brinda la información sobre la discretización del espacio y el tiempo a MODFLOW. En este paquete están incluidos los datos como número de capas (NLAY), unidades de tiempo (ITMUNI) y distancia (LENUNI), indicación para las capas con un lecho confinante (LAYCBD), número de pasos de tiempo (NSTP) y multiplicador (TSMULT) para cada período de estrés.

Este archivo, también contiene la información de las profundidades de las capas, partiendo de la capa más superficial, incluyendo las capas confinantes y finalizando con la profundidad de la capa que se encuentra junto al basamento. La información que corresponde a este tipo de datos espaciales, se introduce en forma de mallado, indicando un valor para cada celda del espacio ya discretizado.

Discretización del tiempo. La simulación se divide en períodos de estrés, es decir, intervalos de tiempo durante los cuales todas las acciones externas son constantes. Estos períodos de estrés, se dividen a su vez en pasos de tiempo. En cada período, los pasos de tiempo forman una progresión geométrica (figura 4.6). El multiplicador es la razón entre la duración de cada paso de tiempo y el anterior.

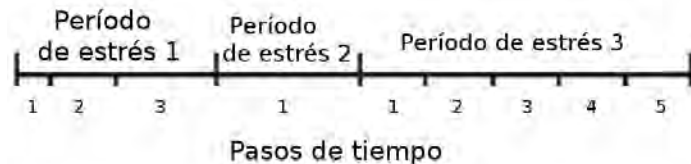


Figura 4.6: Discretización del tiempo (adaptado de *Harbaugh (2005)*).

Para el Valle de San Luis Potosí, se tiene un área de 1980 km², los límites del valle son las coordenadas en UTM de 2,434,250 a 2,474,750 Norte y de 281,750 a 326,250 Este. Si se discretiza el espacio, de tal manera que cada celda tenga de lado 500 m, se tendrá una malla de 90 columnas y 82 renglones, con un total de 7,380 celdas. Como ya se mencionó, el modelo considera tres capas en total, donde se indicará para la primera capa un lecho confinante.

Una vez definidos todos estos parámetros es posible crear el archivo.dis, el cual será llamado por el archivo.nam, para realizar la simulación matemática. Ésto se hace con todos los demás paquetes que se utilicen en el modelo.

Paquete básico (file.bas)

El paquete básico de MODFLOW, contiene información de las condiciones de frontera. Es necesario conocer cuales de las celdas del modelo son celdas activas, cuales son inactivas y cuales tienen una carga constante, la información se introduce en el archivo por cada celda, donde el valor será 1 para celdas activas, -1 para inactivas y 0 para celdas con carga constante. También contiene la información sobre el valor asignado a las celdas inactivas (HNOFLO), esto es para indetificarlas del resto. Finalmente, se indica el valor de las cargas al inicio de la simulación (STRT), esta información se incluye celda por celda, o en caso de que sea constante en toda el área, se puede introducir un sólo valor especificado.

Para el caso de la zona de estudio de San Luis Potosí, las condiciones de frontera corresponderán a zonas de celdas inactivas en toda la parte correspondiente al cerro de San Miguelito y la sierra de Álvarez, ya que en estas zonas no existe flujo y no contribuye a la solución de la ecuación. Las celdas restantes serán celdas activas, esto se aplica para todas las capas del modelo. Las cargas iniciales, que se pueden considerar para la simulación, son valores conocidos que se puedan o hayan medido. López-Álvarez (2012) presenta unas cargas iniciales del período de 1986, donde se tiene valores de 1740 a 1757 msnm. El menor valor se encuentra en el centro del Valle de San Luis Potosí, en la zona urbana. Posteriormente se realizan simulaciones con cargas iniciales de 1995 y 2007.

Paquete de flujo centrado en bloque (file.bcf)

Este paquete es el encargado de calcular las conductancias de la ecuación en diferencias finitas, éstas determinan el flujo entre celdas contiguas. También calcula el flujo desde y hacia el almacenamiento. El cálculo se lleva a cabo bajo la condición de que los nudos de las celdas se encuentran en el centro de la misma. Este paquete y el paquete de flujo por propiedades de capa (LPF), no se pueden usar al mismo tiempo, ya que ambos cumplen con la misma función, la diferencia radica en las características del sistema con el que trabaja cada uno, que son por celda o por capa respectivamente.

Para poder crear el archivo que maneja este paquete, se necesita conocer el tipo de acuífero para cada capa, existen cuatro tipos de acuíferos: confinado, libre, confinado convertible con transmisividad constante y confinado convertible con transmisividad variable. También son necesarios algunos otros valores, como los que se enlistan a continuación:

- La transmisividad del acuífero a lo largo de los renglones, esto es para el caso de acuíferos confinados o convertibles con T (transmisividad) constante.
- La conductividad hidráulica horizontal, esto en el caso de acuíferos libres o convertibles con T variable.

- Los coeficientes de almacenamiento primario y secundario para acuíferos convertibles con T variable o constante.
- Conductividad hidráulica vertical, dividida por el grosor de la capa a la capa inferior.

Para el caso de San Luis potosí, se definiría la primer capa como el acuífero libre (acuífero colgado), la segunda capa sería un acuífero convertible, esto para el acuífero profundo con material granular y la tercer capa sería definido como un acuífero confinado, que es el acuífero compuesto por roca volcánica. Se necesitaría un mallado con información en cada celda de acuerdo a los valores enlistados anteriormente.

Para el acuífero libre, se conoce que la conductividad hidráulica horizontal es igual en toda la capa, tiene un valor de 1.2×10^{-3} m/s, por lo que esto se puede indicar en el archivo, al igual que la conductividad hidráulica vertical. En el caso de las otras dos capas es necesario conocer la transmisividad del medio, y para la capa dos también la conductividad vertical. Con esta información es posible escribir el archivo para utilizar este paquete.

Paquetes de estrés

Los paquetes de estrés constituyen toda la información extra del modelo, los paquetes de estrés son:

- RCH : paquete de recarga
- WEL : paquete de pozos
- EVT : paquete de evapotranspiración

El paquete de Recarga (file.rch) simula la recarga distribuida en superficie del acuífero. Lo más usual, es que este tipo de recarga ocurra como consecuencia de la lluvia que se infiltra hasta el sistema acuífero (Cruces-de Abia, 2006/2007). Este paquete necesita la información de recarga de cada una de las celdas en cada uno de los períodos de estrés.

La recarga para el acuífero colgado de San Luis Potosí se debe principalmente a la infiltración de la precipitación de las lluvias.

Existen distintos modelos que presentan diversas formas de recarga del acuífero profundo. Flores-Márquez et al. (2011), presentan una recarga principalmente proveniente de la sierra de San Miguelito y la Sierra de Álvarez, que son las zonas más altas y con mayor cantidad de precipitación. Sin embargo, López-Álvarez (2012), presenta un trabajo donde la recarga es proveniente del noroeste, considerando que las zonas

de Sierras tienen poca permeabilidad y no existe una infiltración importante hacia el flujo regional. Existen distintos métodos por los cuales se miden los valores de recarga, éstos deben ser calculados y expresados en toda la superficie para ser utilizado por este paquete.

El paquete de pozos (file.wel) permite simular características tales como pozos que bombean agua de un acuífero (o recargan) a un caudal fijo durante un período determinado (Cruces-de Abia, 2006/2007). Para que el paquete de pozos sea llamado en la simulación debe estar indicado en el archivo name (file.nam). La información que debe contener el archivo es la siguiente:

- Q : la cantidad extraída o inyectada al acuífero. Valores negativos corresponden a pozos de bombeo y valores positivos a pozos de recarga.
- La fila en la que se encuentra cada pozo.
- La columna que le corresponde a cada pozo.
- La capa de la cual se está extrayendo o inyectando agua.

Cada uno de estos datos se lee para cada uno de los pozos, también se determina a que período de estrés corresponde, es decir que cada pozo tiene que tener asignado un valor Q para cada período de estrés.

En el caso de San Luis Potosí, existe una cantidad de pozos que extraen agua, de los cuales, algunos tienen una profundidad tal que llega hasta el acuífero fracturado, mientras que la mayoría extraen agua del acuífero granular, esto se indica con el número de capa a la que corresponde cada pozo. En total se calcula una extracción de 136×10^6 m³/año (López-Álvarez, 2012). Cada pozo es representado de forma puntual por medio de las coordenadas de su ubicación en el valle.

El paquete de evapotranspiración (file.evt) simula los efectos de la transpiración de las plantas y de la evaporación directa, extrayendo agua desde la zona saturada del sistema. El modelo está basado en las siguientes aproximaciones: 1) cuando la superficie libre está por encima de una cota especificada, llamada superficie de evapotranspiración (ET surface), las pérdidas por evapotranspiración desde la superficie libre, ocurren al valor máximo especificado por el usuario; 2) cuando la profundidad de la superficie libre bajo la superficie de ET, excede de un valor especificado por el usuario, término llamado profundidad de extinción (extinction depth o cutoff depth), momento en que cesa la evapotranspiración; y 3) entre estos límites, la evapotranspiración varía linealmente con la cota de la superficie (Cruces-de Abia (2006/2007)).

La información que conforma este archivo es la siguiente:

- SURF: La elevación de la superficie de evapotranspiración.

- EVTR: El máximo flujo de evapotranspiración (flujo volumétrico por unidad de área [LT^{-1}]).
- EXDP: La profundidad de extinción.
- IEVT: Capa de la que se remueve la evapotranspiración.

En una zona como la del Valle de San Luis Potosí, la evapotranspiración es un factor muy importante, ya que según algunos estudios realizados, la evapotranspiración es superior a la recarga de agua subterránea en esta zona, por tener características áridas. *Flores-Márquez et al. (2011)*, toman una evapotranspiración de 0.918×10^{-3} m/d con una profundidad de extinción de 5 m, estos valores se definieron basados en el clima, vegetación de la región y estudios previos.

Archivo nombre. El archivo.nam, es el archivo encargado de llamar a todos los paquetes que se van a utilizar en una simulación. Una vez que se corre una simulación en MODFLOW, éste pedirá el archivo de entrada, éste archivo es el archivo nombre. Si el usuario cuenta con la información suficiente, y se encuentra bien escrita en cada uno de los archivos de cada paquete, el programa MODFLOW, regresará un archivo llamado list, es decir, *archivo.lst*, en este archivo se muestra la descripción del proceso de simulación.

Simulación en régimen estacionario y transitorio. La primer simulación que se lleve a cabo, es en estado estacionario, esta primera simulación proporcionará la información de las cargas iniciales para el estado transitorio. Se realiza la simulación en régimen estacionario, en un período de tiempo en el que se tenga la información necesaria para poder calibrar el modelo, en el caso del modelo conceptual de SLP, se tomaría el año 1977 en la figura 4.2, donde se cuenta con un modelo de como lucían los niveles piezométricos en la época y así poder ser comparados con el resultado del modelo que se tiene. Ésto se puede repetir para los siguientes años, para comparar y tener mayor precisión. Una vez que se tiene el resultado estacionario, con un margen pequeño de error, se corre una simulación en estado transitorio, primero para años con datos conocidos y posteriormente para desarrollar nuevos posibles escenarios para los acuíferos, dependiendo de las necesidades del estudio que se está realizando.

Capítulo 5

Resultados y discusión

En este capítulo se presentan algunos ejemplos realizados con el módulo *r.gws*, desarrollados para ilustrar el manejo del mismo. Los diferentes casos considerados tienen distintas características que permiten mostrar la aplicación del módulo. Los casos son: 1) La simulación de un acuífero libre en estado estacionario y en estado transitorio; 2) Un modelo de dos capas, una de las capas es un acuífero libre, separado de uno confinado por una capa de limo, se simula en estado estacionario; y 3) Un ejemplo obtenido de MODFLOW-2005, que consiste en un modelo de tres capas, simulando las cargas en estado estacionario.

Los primeros dos ejemplos fueron desarrollados por primera vez por Chiang (2005), quien utilizó el software PMWIN. Este software tiene la capacidad de modelar acuíferos, utilizando como herramienta MODFLOW. PMWIN fue utilizado con la versión de MODFLOW-2000 y compilado con Fortran 95, el módulo *r.gws*, utiliza la versión de MODFLOW-2005 y se compila con Fortran 99.

5.1. Acuífero libre de una capa

El acuífero de este ejemplo es una capa de arena de grano grueso, con conductividad hidráulica isotrópica de 160 m/d, tiene un rendimiento específico de 0.06. El acuífero se recarga durante la temporada húmeda, que tiene una duración de cuatro meses, la tasa de recarga es de 7.5×10^{-4} m/d. Fuera de la temporada húmeda no existe recarga en el acuífero.

La capa tiene una elevación de 25 m. El área del modelo es de 1000 m de largo por 6000 m de ancho. Al este y oeste se encuentran zonas donde no hay flujo, existiendo una zona de montañas de roca volcánica en la esquina sureste. Al norte, existe una zona de carga constante con un valor de 15 m. Al sur, la frontera tiene una afluencia de $0.0672 \text{ m}^3/\text{d}$ por metro. Un total de nueve pozos en toda el área extraen a

3888 m³/d cada uno, durante 8 meses de la temporada seca, ésto suministra agua para riego y propósitos domésticos. La figura 5.1 muestra la zona de estudio del ejemplo.

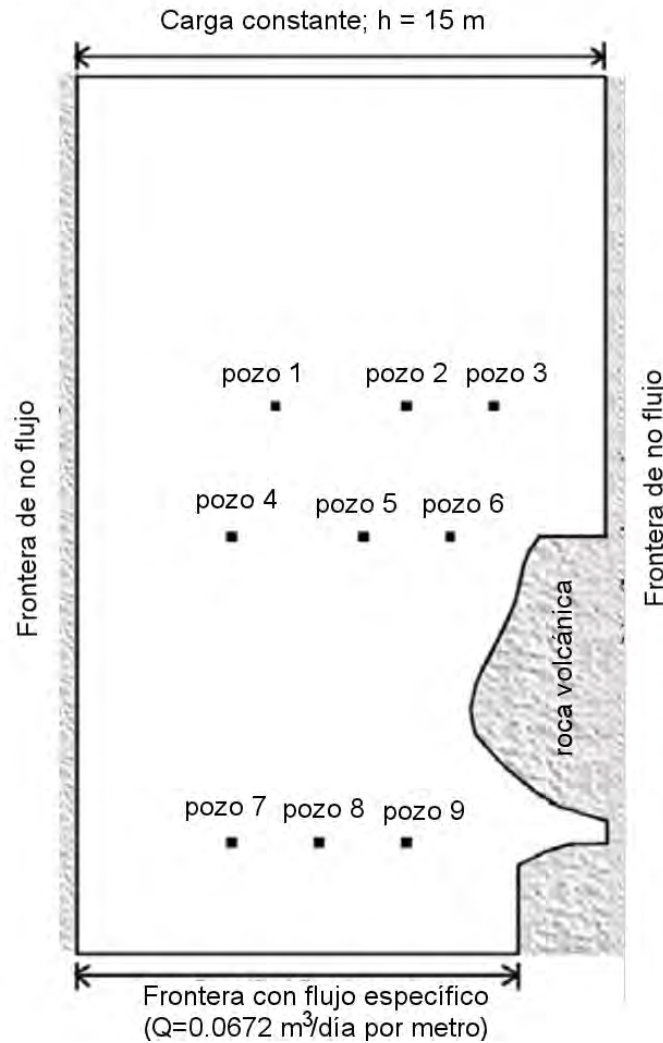


Figura 5.1: Configuración del modelo.

El propósito de este ejemplo, es evaluar los niveles de agua en el acuífero en distintas condiciones. La primera es en estado estacionario, con una recarga de 2.5×10^{-4} m/d, sin extracción. La segunda es después de ocho meses de extracción durante la temporada seca y la tercera, es conocer los niveles de agua al final de los siguientes cuatro meses de temporada húmeda.

Creando el modelo en SIG GRASS

Como ya se ha mencionado en este trabajo, los SIG tienen la ventaja de trabajar con datos georeferenciados, por lo que primero se crea una región en GRASS donde se encuentra la zona de estudio. Para este ejemplo, se ha creado una región xy de 10000 metros de largo por 6000 metros de ancho. Se trabaja con una resolución de 200 metros, por lo que la región contará con un total de 50 renglones y 30 columnas que conforman 1500 celdas en el área, en términos de MODFLOW, esto corresponderá a la discretización del espacio.

Una vez creada la región, es posible comenzar a crear los mapas que tendrán la información del sistema (recarga, conductividad hidráulica, pozos, etc).

Los datos para la primer simulación serán las propiedades de la capa, condiciones de frontera, geometría del acuífero, cargas iniciales, parámetros del tiempo y tasas de recarga. Las condiciones de frontera se determinan en un mapa ráster que contendrá el número de categoría de cada celda de acuerdo a la siguiente convención (ver figura 5.2):

- celdas activas=1
- celdas inactivas=0
- celdas con carga constante=-1



Figura 5.2: Condiciones de frontera.

Los valores para cada mapa pueden ser asignados con la herramienta de SIG GRASS *r.mapcalc*, que permite realizar el algebra de mapas.

Este método fue utilizado para desarrollar los mapas de las propiedades físicas del medio mencionadas anteriormente, como son la conductividad hidráulica, la recarga del acuífero y las cargas iniciales, esta información tiene una distribución espacial determinada (homogénea en este primer ejemplo).

Para establecer el tipo de acuífero, el módulo *r.gws* cuenta con la opción *aqtype* (que vale uno para acuíferos libres). De manera análoga se determinan condiciones como períodos de estrés, pasos de tiempo para cada período de estrés, unidades de tiempo, unidades de longitud, tipo de simulación, número de capas, etc., como se verá más adelante (Cuadro. 5.1).

Integrando la información de los pozos. En el caso de los pozos, se tiene una tabla en la que se conocen las coordenadas de cada pozo, se asocia un identificador para cada pozo y se conoce la tasa de extracción, en este caso se tiene un archivo de texto con esta información, el cual será importado a SIG GRASS, donde se reconocerá al archivo como una serie de puntos en la zona de estudio y así obtendremos un mapa vectorial de los pozos que se encuentran en la zona de estudio.

Una vez que se tiene el mapa vectorial en GRASS, se hace la asociación del vector a una tabla en PostgreSQL mediante la siguiente sentencia

```
g.copy vect=well, wellpg
```

para ésto es necesario que GRASS se encuentre conectado a la base de datos de PostgreSQL, la cual tiene que estar inicializada (ver Matthew y Stones, 2005), para conectar GRASS a PostgreSQL, desde la terminal se usa el comando

```
db.connect driver=pg database="host=localhost,dbname=dbexample"
```

Así ya tenemos un vector asociado a una tabla de tipo PostgreSQL. Esta tabla se puede modificar desde PostgreSQL o desde GRASS, ya que funciona como una interfaz entre ambos. En el caso de los pozos, es necesario conocer el nombre de las columnas para poder satisfacer los campos que requiere el módulo *r.gws*. En este caso se tiene una columna asociada a la tasa de extracción de cada pozo llamada *extr*, otras asociadas a los pozos activos en cada período de estrés llamadas *stress1* para el período 1 y 3, y *stress2* para el período de estrés 2, por último una de las columnas con los datos de la capa en la que se encuentra cada pozo llamada *layer*. Para los siguientes dos ejemplos el procedimiento con la base de datos es la misma en el caso de los pozos y drenaje en la zona, ya que ambos requieren de un vector de entrada para leer la información de sus tablas.

Se corre la simulación tomando en cuenta tres períodos de estrés, el primero es en un caso estacionario que tiene una longitud de un día con un paso de tiempo, esto

nos dará la información necesaria de las cargas iniciales que se tomarán en cuenta para los siguientes períodos, que serán en estado transitorio. En el primer período de estrés se tiene una recarga homogénea de 2.5×10^{-4} m/d y no existe extracción, sin embargo, existe una condición de frontera en la que hay un flujo entrante de las celdas de la zona sur, este flujo se determina aplicando un flujo positivo para cada celda de $13.44 \text{ m}^3/\text{d}$, dando un total de $0.0672 \text{ m}^3/\text{d}$ por metro.

El segundo período de estrés es en estado transitorio y corresponde a ocho meses (240 días) de sequía, donde no existe recarga y sí hay extracción. Este período se dividirá en un total de 12 pasos de tiempo, ésto permitirá ver la evolución del sistema a lo largo de este tiempo.

Finalmente un tercer período de estrés, en estado transitorio, que corresponde a los siguientes cuatro meses (120 días) de temporada húmeda, en este caso se tiene una recarga de 7.5×10^{-4} m/d y una extracción en cada pozo de $-3888 \text{ m}^3/\text{d}$. La sentencia para ejecutar el programa con todas estas condiciones es la siguiente:

```
r.gws layers=1 region=boundaries@PERMANENT
heads=initial.heads@PERMANENT stress=3 simulation=1,0,0
aqtype=1 K=H.conductivity@PERMANENT top=top@PERMANENT
tunits=4 lunits=2 bottom=bottom@PERMANENT length=1,240,120
steps=1,12,6 tsmult=1,1,1 recharge=rch.0@PERMANENT,
rch.1@PERMANENT,rch.2@PERMANENT headsin=h2 drawdownin=d2 bed=0
pack=0 well=wellfluxt@PERMANENT extr=extraccion layer_wel=layer
Sy=Sy@PERMANENT welluse=stres1,stres2,stres1
```

En la línea de comandos se distinguen los mapas ráster y vectores por la terminación *@PERMANENT*, que es el mapset (Neteler y Mitasova (2007)) donde se encuentra localizado el mapa, dentro de GRASS. Las otras opciones son valores directos que entran en el programa y se muestran en el Cuadro. 5.1. Si se utiliza la interfaz gráfica del módulo, es posible ver la descripción más detallada de cada uno de los datos de entrada, esto es de ayuda cuando no se conocen bien las características que debe cumplir cada uno de los datos. En el Cuadro.5.1 se describen también los datos que el módulo requiere, mostrando de que tipo debe ser el dato, la cantidad de datos a ingresar, separandolos por comas y el valor introducido por cada uno de los ejemplos que aquí se plantean.

Si se observa en la figura 5.3 ,a comparación de los resultados, después de adaptar los datos del ejemplo en PMWIN a GRASS, el resultado final no presenta mayor diferencia entre el resultado de Chiang y el obtenido con *r.gws*. Esto muestra que el módulo es consistente con otros softwares, brindando resultados similares cuando se desarrolla en GRASS.

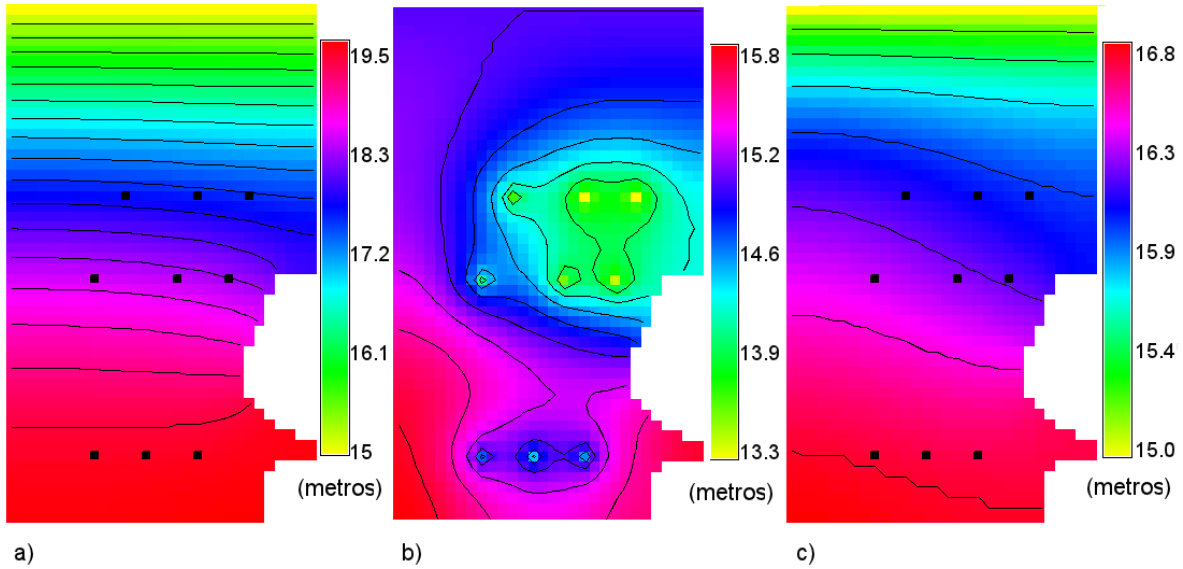


Figura 5.3: Simulaciones obtenidas con el módulo *r.gws* para el ejemplo 1. a) Simulación en estado estacionario; b) Simulación en estado transitorio para el período de estrés 1, paso de tiempo 12; c) Simulación en estado transitorio para el período de estrés 2, paso de tiempo 6.

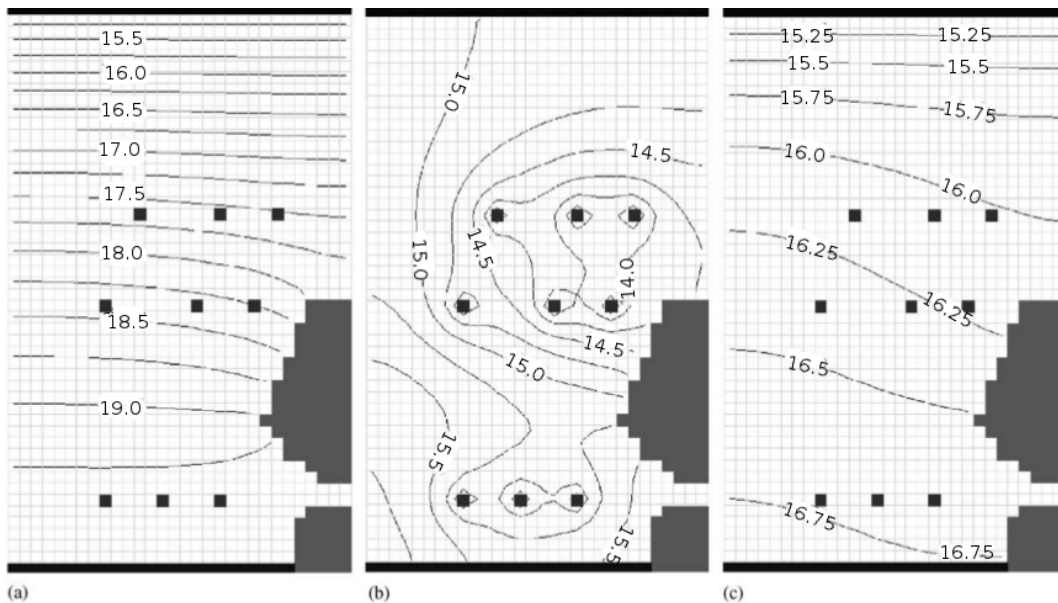


Figura 5.4: Simulaciones obtenidas con PMWIN para el ejemplo 1. a) Simulación en estado estacionario; b) Simulación en estado transitorio para el período de estrés 1, paso de tiempo 12; c) Simulación en estado transitorio para el período de estrés 2, paso de tiempo 6.

5.2. Sistema de un acuífero con río

En este ejemplo, se presenta un caso en el que un río fluye a través de un valle (fig. 5.5). El valle está bordeado al norte y al sur por intrusivos de granito, que son impermeables. Las cargas hidráulicas corriente arriba y corriente abajo son conocidas. El río forma parte de un acuífero libre permeable con conductividad hidráulica horizontal $HK = 5$ m/d, conductividad vertical $VK = 0.5$ m/d, rendimiento específico de $S_y = 0.05$ y porosidad efectiva de $n_e = 0.2$. Este acuífero se encuentra sobre un acuífero confinado de anchura variable, con conductividad hidráulica horizontal $HK = 2$ m/d, conductividad vertical $VK = 1$ m/d, almacenamiento específico $S_s = 5 \times 10^{-5}$ y porosidad efectiva $n_e = 0.2$. Una capa limosa, de espesor 2 m, conductividad hidráulica horizontal de $HK = 0.5$ m/d, conductividad vertical $VK = 0.05$ m/d y porosidad efectiva $n_e = 0.25$, separa los dos acuíferos. Las elevaciones de las partes inferior y superior de las capas son conocidas. Tres pozos extraen agua a una tasa de 500 m³/d, los tres pozos extraen agua del acuífero confinado.

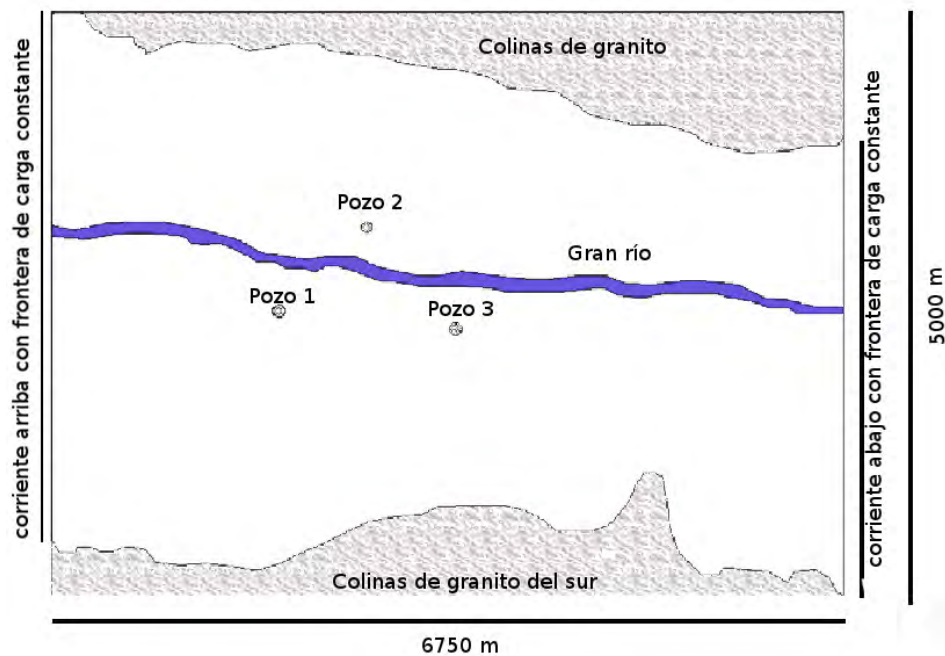


Figura 5.5: Configuración del sistema de un acuífero con río.

El río atraviesa la zona de oeste a este. En la zona oeste, corriente arriba, tiene una elevación total de 19.4 m, donde la altura de la parte inferior del río es de 17.4 m, corriente abajo la elevación total es de 17 m, con una altura de la parte inferior del río de 15 m. El río tiene un ancho de 100 m, mientras que la del lecho del río es de 1 m, la conductividad hidráulica del río es de 2 m/d. Con esta información, es posible construir un modelo que muestre las zonas de captura de los pozos y obtener los niveles de agua

después de un período de estrés de una simulación en estado estacionario.

Creando el modelo en SIG GRASS

De igual forma que en el ejemplo anterior, se crearon los mapas que contienen la información para correr el modelo, el primer mapa a crear es el de las condiciones de frontera para celdas activas e inactivas, para las capas 1 y 2 el mapa queda como se ve en la figura 5.6.



Figura 5.6: Celdas activas para las dos capas y cargas constantes, ejemplo 2.

Los siguientes mapas con información de las alturas de las capas y las cargas iniciales fueron obtenidas a partir de un ejemplo en archivos de PMWIN (la carpeta llamada *tutorials* contiene información en archivos de texto del modelo), los cuales fueron exportados a SIG GRASS mediante la herramienta *r.in.ascii*. Los mapas restantes se crearon de manera análoga. En el Cuadro 5.1 se muestran todas las opciones para el ejemplo 2. La sentencia en la terminal, finalmente queda así:

```
r.gws layers=2 pack=0 region=ibound1y3@PERMANENT,
ibound1y3@PERMANENT heads=i.heads1.2.3@PERMANENT,
i.heads1.2.3@PERMANENT stress=1 simulation=1 aqtype=1,3
K=H.cond1@PERMANENT,H.cond3@PERMANENT
```

```

vcond=V.cond1@PERMANENT,V.cond2@PERMANENT top=top@PERMANENT
tunits=4 lunits=2 bed=1,0 bottom=bottom1@PERMANENT,
bottombed1@PERMANENT,bottom2@PERMANENT length=1 steps=1
tsmult=1 drawdownin=dr12 headsin=hd12
river_heads=riverheads@PERMANENT river_cond=rivercond@PERMANENT
river_elev=riverelev@PERMANENT well=wellpg@PERMANENT
extr=extr layer_wel=layer welluse=stress1

```

El resultado final de la simulación se muestra en la figura 5.7, donde también se hace una comparación con el resultado obtenido con PMWIN, obteniendo un resultado consistente, al igual que en el caso anterior.

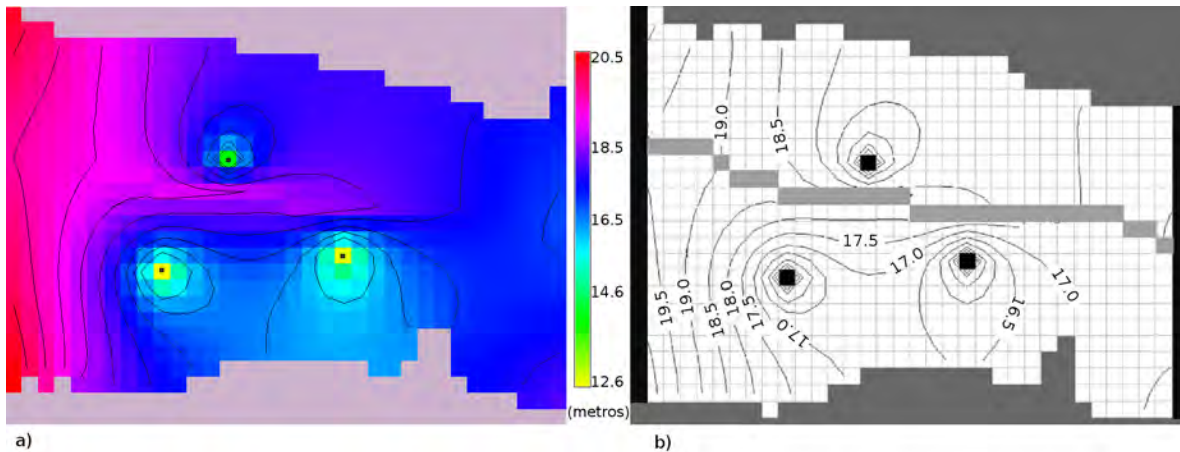


Figura 5.7: a) Resultado de la simulación con *r.gws*, b) Resultado de la simulación usando PMWIN.

5.3. Ejemplo de MODFLOW 2005

El ejemplo que se muestra a continuación fue obtenido de MODFLOW, este ejemplo viene con el paquete a descargar MODFLOW-2005 y sus archivos se encuentran dentro de la carpeta *test – run*. Este problema es descrito y desarrollado por primera vez en 1988 por Harbaugh (McDonald y Harbaugh, 1988), para ejemplificar los archivos y el uso de MODFLOW-88.

El ejemplo consiste en la simulación de un sistema con tres acuíferos, uno libre y dos confinados (fig. 5.8). Los acuíferos se encuentran separados por dos capas confinantes intermedias. La zona de estudio es un cuadrado de 75000 ft de lado. El archivo *twri.lst* se encuentra en la carpeta *test – out*, donde se encuentran los resultados a las simulaciones hechas por MODFLOW-2005, por lo que para comparar, se utilizan

los resultados de dicho archivo y se exportan a SIG GRASS, para poder ser comparados con los resultados que se obtendrán a partir de archivos generados por el módulo *r.gws*.

El sistema tiene una recarga en la capa superior de 3×10^{-8} ft/s. El flujo entra al sistema por medio de la infiltración de la precipitación. El flujo que sale del sistema es debido a los tubos de drenaje enterrados, la extracción de los pozos y un lago representado como una frontera con carga constante (fig. 5.8). El acuífero superior, tiene una conductividad hidráulica $HK = 0.001$ ft/s y tiene una elevación de 200 ft, en la parte inferior cuenta con lecho confinante a -150 ft con un ancho de 50 ft, el acuífero confinado en el centro (segunda capa) tiene una transmisividad $T = 0.01$ ft²/s y su parte inferior se encuentra a una elevación de -300 ft con un lecho confinante de 50 ft, finalmente el acuífero inferior tiene una transmisividad de $T = 0.02$ ft²/s con una elevación de la parte inferior de -450 ft. Las cargas iniciales en toda la zona es de 0.0 ft.

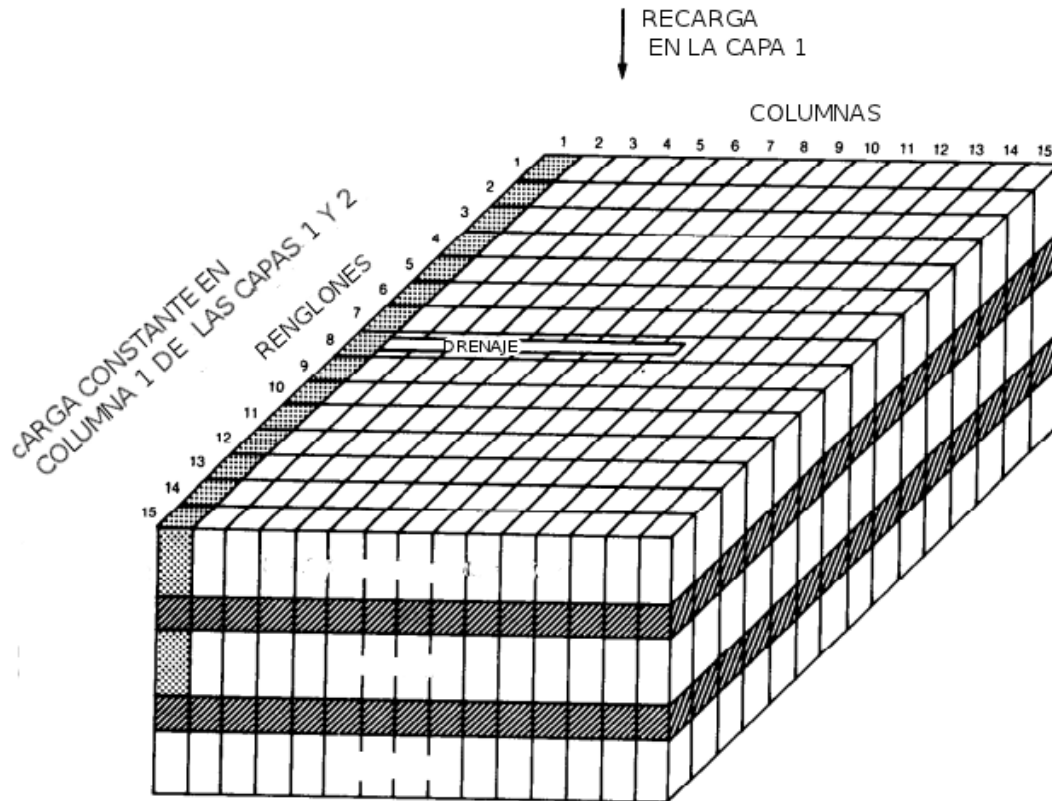


Figura 5.8: Configuración del ejemplo 3, los acuíferos 1 y 2 tienen las mismas celdas activas.

Creando el modelo en SIG GRASS

Primero se genera la región donde se va a hacer todo el estudio, se determina una zona con coordenadas xy con las características de una malla de 15 columnas y 15 renglones con una resolución de 5,000 ft en ambos sentidos del plano xy como se describe en el ejemplo.

Para poder generar los archivos del modelo con SIG GRASS, como en los casos anteriores, se realizaron los mapas correspondientes con *r.mapcalc*, también se exportaron a GRASS algunos de los archivos con información por celda, como el de las profundidades de las capas, las cargas del río o las elevaciones del mismo. Para el desarrollo de los mapas vectoriales, se realizó el mismo procedimiento que en los ejemplos anteriores con tablas de tipo PostgreSQL asociados a mapas vectoriales en GRASS.

La figura 5.8, muestra las celdas activas, para la capa uno y dos éstas corresponden a todas las de la zona, exceptuando las de la primer columna, mientras que en la tercer capa todas las celdas son activas. Se desarrollaron los mapas para cada propiedad física del sistema por cada capa, esto da toda la información para realizar la simulación (Cuadro. 5.1). La simulación se realiza con un período de estrés que dura 86400 s en estado estacionario y con un paso de tiempo.

En la línea de comando (también se puede hacer de manera gráfica) de dan las indicaciones del módulo para hacer la simulación, incorporando la información de los mapas (ver Cuadro. 5.1) como se muestra a continuación:

```
r.gws layers=3 pack=0 region=boundaries1y2@PERMANENT,
boundaries1y2@PERMANENT,boundaries3@PERMANENT
heads=initial.heads@PERMANENT,initial.heads@PERMANENT,
initial.heads@PERMANENT stress=1 simulation=1 aqtype=1,0,0
T=transm2@PERMANENT,transm3@PERMANENT K=h.cond1@PERMANENT
vcond=vcond1@PERMANENT,vcond2@PERMANENT top=top@PERMANENT
tunits=1 lunits=1 bed=1,1,0 bottom=bottom1@PERMANENT,
bottombed1@PERMANENT,bottom2@PERMANENT,bottombed2@PERMANENT,
bottom3@PERMANENT length=86400 steps=1 tsmult=1
recharge=rch1@PERMANENT drawdownin=dr headsin=hd
well=wellpg@PERMANENT extr=rate layer_wel=layer
welluse=stress1 drain=drainpg elev_drn=elev
layer_drn=layer drnuse=layer condrn=conductance
```

El resultado de la simulación para este ejemplo se muestra en la figura 5.9, en la cual se puede observar una similitud enorme entre ambos resultados. La diferencia entre ambas simulaciones consiste en que MODFLOW utiliza en este ejemplo el paquete SIP(Strongly Implicit Procedure), mientras que para el resultado obtenido

por el módulo en SIG GRASS se utiliza el paquete PCG (Preconditioned Conjugate-Gradient), ambos paquetes son utilizados para procedimientos de resolución del sistema de ecuaciones.

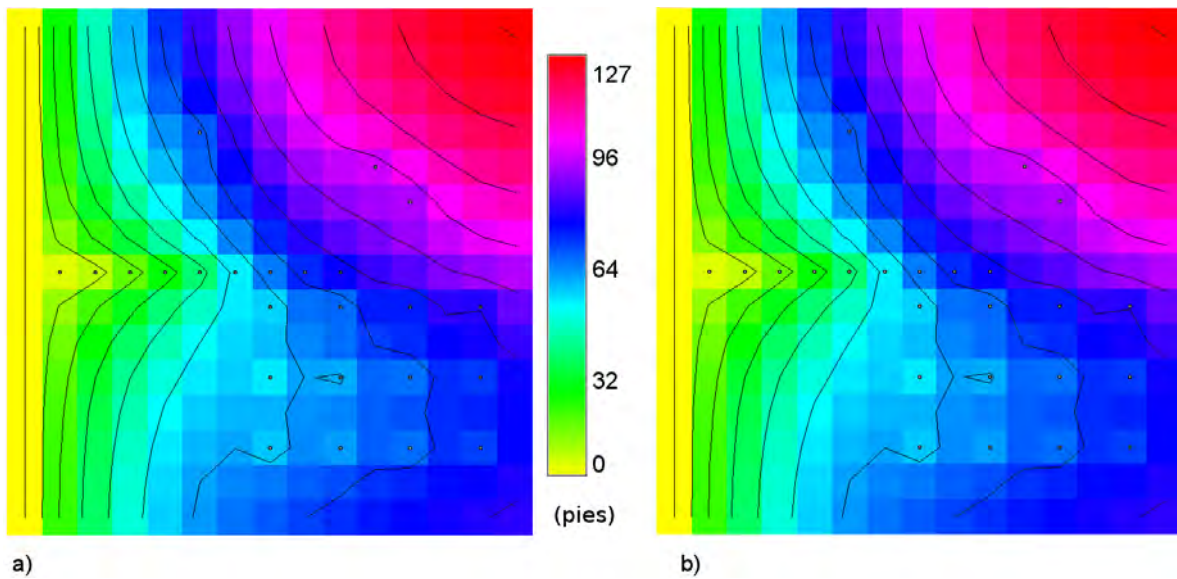


Figura 5.9: a) Simulación obtenida con el módulo *r.gws* utilizando el paquete PCG , b) Simulación obtenida de MODFLOW utilizando el paquete SIP.

5.4. Validez de los resultados

Los ejemplos aquí presentados, muestran que el módulo *r.gws* puede ser utilizado en simulaciones en estado estacionario y transitorio, al igual que en sistemas con una o más capas.

Para mostrar la validez de los resultados obtenidos, en el caso del ejemplo del acuífero libre y el sistema de acuífero con río, se han comparado los resultados de *r.gws* con los obtenidos con PMWIN (Chiang, 2005). En el caso del ejemplo 3 de MODFLOW-2005, se comparó con el resultado que viene en el archivo *twri.lst*, dentro de la carpeta de ejemplos de MODFLOW-2005. En todos los casos, el resultado es satisfactorio, por lo que el módulo cuenta con las características para poder desarrollar simulación de flujo de agua subterránea multicapa, en estado estacionario y transitorio.

Parámetro	Descripción	Valor	Ejemplo 1	Ejemplo 2	Ejemplo 3
Layers	Número de capas en el modelo	entero 1=segundos 2=minutos 3=horas 4=días 5=años 0=no definido	1	2	3
tunits	Unidades de tiempo	1=pies 2=metros 3=centímetros	4	4	1
lunits	Unidades de longitud	entero 0=transitorio 1=estacionario	2	2	1
stress	Número de períodos de estrés	entero	3	1	1
simulation	Tipo de simulación para cada período de estrés	0=transitorio 1=estacionario	1,0,0	1	1
length	Longitud de cada período de estrés	entero	1,240,120	1	86400
steps	Pasos de tiempo para cada período de estrés	entero	1,12,6	1	1
tsmult	Multiplicador de los pasos de tiempo para cada período de estrés	flotante	1,1,1	1	1
aqtype	Tipo de acuífero por cada capa	0=confinado 1=libre 2=variable con T constante 3=variable con T variable	1	1,3	1,0,0
bed	Indica si existe capa inferior confinante	0=no existe 1=si existe	0	1,0	1,1,0
pack	Paquete de solución a utilizar	1 = LPF 0=BCF	0	0	0

Parámetro	Descripción	Valor	Ejemplo 1	Ejemplo 2	ejemplo 3
region	Mapa(s) ráster con celdas activas	caracteres	boundaries	ibound1y3, ibound1y3	boundaries1y2, boundaries1y2, boundaries3
heads	Mapa(s) ráster con cargas iniciales	caracteres	initial.heads	i.heads1.2.3, i.heads1.2.3	initial.heads, initial.heads, initial.heads
K	Mapa(s) ráster con las conductividades hidráulicas	caracteres	H.conductivity	H.cond1,H.cond3	h.cond1
T	Mapa(s) ráster con los valores de transmitividad	caracteres			transm2, transm3
top	Mapa(s) ráster con la elevación máxima del acuífero	caracteres	top	top	top
bottom	Mapa(s) ráster con las profundidades de las capas contiguas	caracteres	bottom	bottom1, bot- tombed1, bot- tom2	bottom1, bot- tombed1, bot- tom2, bot- tombed2, bot- tom3
recharge	Mapa(s) ráster con los valores de las recargas para cada período	caracteres	rch.0,rch.1,rch.2		rch1
vcond	Mapa(s) ráster con los valores de conductividad vertical	caracteres		V.cond1,V.cond2	vcond1,vcond2
Sy	Mapa(s) ráster con los valores de coeficiente de almacenamiento primario	caracteres	Sy		
Syb	Mapa(s) ráster con los valores de coeficiente de almacenamiento secundario	caracteres			
evt_elev	Mapa(s) ráster con los valores de la elevación de la superficie de evapotranspiración (EVT) para cada período de estrés	caracteres			

Parámetro	Descripción	Valor	Ejemplo 1	Ejemplo 2	Ejemplo 3
evt_flow	Mapa(s) ráster con valores máximos de flujo de EVT para cada período de estrés	caracteres			
evt_ext	Mapa(s) ráster con los valores de profundidad de extinción para cada período de estrés	caracteres			
evt_lay	Mapa(s) ráster con la capa de donde se extrae EVT en cada período de estrés	caracteres			
well	Nombre del vector con información de los pozos	caracteres	wellflux	wellpg	wellpg
extr	Nombre de la columna con el valor de la extracción de cada pozo	caracteres	extraccion	extr	rate
layer_well	Nombre de columna con el número de capa que contiene al pozo	caracteres	layer	layer	layer
welluse	Nombre de la columna con los pozos activos para cada período 1=activo 0=activo	caracteres	stres1,stres2,stres1	stress1	stress1
drain	Nombre del vector con información de drenaje	caracteres			drainpg
elev_drn	Nombre de la columna con información de la elevación del drenaje	caracteres			elev
layer_drn	Nombre de la columna de la capa donde está el drenaje	caracteres			layer
drnuse	Nombre de la columna con la información de celdas con drenaje 1=activo 0=activo	caracteres			layer

Parámetro	Descripción	Valor	Ejemplo 1	Ejemplo 2	Ejemplo 3
condrn	Nombre de la columna con información de la conductividad hidráulica entre el drenaje y el acuífero	caracteres			conductance
river_heads	Mapa(s) ráster con los valores de las cargas del río	caracteres		riverheads	
river_cond	Mapa(s) ráster con el valor de la conductividad del río	caracteres		rivercond	
river_elev	mapa(s) ráster con el valor de la elevación del lecho del río	caracteres		riverelev	

Cuadro 5.1: Parámetros usados en cada uno de los ejemplos para correr una simulación en *r.gws* (Adaptado de *Carrera-Hernández (2006)*).

Capítulo 6

Conclusiones

Después de realizar las simulaciones a forma de ejemplo, con el nuevo módulo en el SIG GRASS *r.gws*, es posible concluir que funciona satisfactoriamente para realizar estudios de flujo subterráneo. Se mostró que el módulo se puede usar para modelos multicapa, en estado tanto estacionario como transitorio.

El módulo descrito en este trabajo, cuenta con algunas características que muchos software no reúnen, por falta de una u otra de las siguientes particularidades:

- *r.gws* es un software que interactúa con MODFLOW, desde la interfaz del SIG GRASS, lo cual lo hace práctico para evitar el cambio de un software a otro. Ésto significa que el usuario no tendría que cambiar el formato de la información almacenada en GRASS, para poder hacer uso de MODFLOW, sino que el programa realizará estas tareas de manera interna.
- Es un programa que se desarrolla bajo condiciones de software libre, evitando altos costos para su utilización.
- Es un software de código libre, lo cual permite que el usuario modifique el programa para el beneficio propio de su trabajo.
- Es un módulo que forma parte de un software (GRASS GIS), que a su vez interactúa con programas que se ocupan de herramientas como las de geoestadística (R) o digitalización de mapas (Q-GIS), lo cual hace que el usuario pueda desarrollar todo desde una sola interfaz.

Es por ésto, que el módulo desarrollado representa un avance tecnológico para el mejoramiento de los estudios de agua subterránea, en diversos sistemas de acuíferos.

Apéndice A

Programa en C

Módulo r.gws:

```
1
2     #define MAIN
3     #include <stdio.h>
4     #include <stdlib.h>
5     #include <string.h>
6     #include <unistd.h>
7     #include <math.h>
8     #include <ctype.h>
9     #include <grass/gis.h>
10    #include <grass/raster.h>
11    #include <grass/glocale.h>
12    #include <grass/Vect.h>
13    #include <grass/dbmi.h>
14    int MXRIVR;
15    int main(int argc, char *argv[])
16    {
17    /*BAS & GENERAL FILES*/
18    int IAPART, ISTRT;
19    int fd, NLAY, i, NROW, NCOL, dp, CNSTNT, IPRN, LOCAT, n, more, colcnt, PACK;
20    char **bound, **ihead;
21    char *name, *mapset, ewres[128], nsres[128];
22    float HNOFLO;
23    void *raster;
24    RASTER_MAP_TYPE out_type, map_type;
25    FILE *fp;
26    pid_t pid; /* Used to call MODFLOW and related programs */
27    /*BCF FILE*/
28    int rc, LAYCON[100], NPER, st, usehy, usetran;
29    int usesf2;
30    int IBCFCB, HDRY, IWDFLG, WEIFCT, IWETIT, IHDWET, TRPY, ISS[100];
31    char **ptr, **sf1, **std, **tran, **hcond, **vcond, **sf2, **ptr2;
32    /*DIS FILE*/
33    int j, k, col, row, ITMUNI, LENUNI, state;
34    char **bot, **stp, **mlt, **lst;
35    float tope, bottom;
36    float ts[100];
37    int NSTP[100], LAYCBD[100], STATE[100], PERLEN[100];
38    /*import modflow*/
39    char *aux;
40    FILE *fa;
41    /* well file*/
42    int type, ncols, nrows, cat, ycell, xcell, ctype, vint[100], IWELCB, MXACIW, d;
43    float x, y, z, q, coy, cox, ewr, nsr, pwest, pnorth;
44    char *vect, sql[200];
```

```

45 static struct line_pnts *Points;
46 char regwest [128], regnorth [128];
47 char query [1024], buf [2000];
48 struct line_cats *Cats;
49 dbTable *table;
50 dbColumn *column;
51 dbCursor cursor;
52 dbDriver *driver;
53 dbString table_name, stmt, value_string;
54 dbHandle handle;
55 dbValue *value;
56
57 struct GModule *module;
58 struct
59 {
60     struct Option *ibd;
61     struct Option *ihd;
62     struct Option *layers;
63     struct Option *type;
64     struct Option *pstor;
65     struct Option *state;
66     struct Option *stress;
67     struct Option *trans;
68     struct Option *hycond;
69     struct Option *vcondu;
70     struct Option *sstor;
71     struct Option *tunits;
72     struct Option *lunits;
73     struct Option *topelev;
74     struct Option *bottelev;
75     struct Option *length;
76     struct Option *steps;
77     struct Option *tsmult;
78     struct Option *river;
79     struct Option *rivcond;
80     struct Option *rivelev;
81     struct Option *extr;
82     struct Option *layer;
83     struct Option *wel;
84     struct Option *rch;
85     struct Option *drawdown;
86     struct Option *headsln;
87     struct Option *sy;
88     struct Option *spes;
89     struct Option *bed;
90     struct Option *vkcb;
91     struct Option *pack;
92     struct Option *use;
93     struct Option *drn;
94     struct Option *elevdrn;
95     struct Option *laydr;
96     struct Option *usedr;
97     struct Option *condrn;
98     struct Option *surf;
99     struct Option *evtr;
100    struct Option *exdp;
101    struct Option *ievt;
102 }parm;
103 struct field_info *Fi;
104 struct Map_info In;
105
106     G_gisinit (argv [0]);
107
108     module = G_define_module ();
109     module->description = _("Preprocessor_and_postprocessor_for_MODFLOW");
110
111     parm.layers = G_define_option ();
112     parm.layers->key = "layers";

```

```

113     parm.layers->type          = TYPE_INTEGER;
114     parm.layers->required      = YES;
115     parm.layers->description= "Number_of_layers_in_the_model";
116
117     parm.pack = G_define_option();
118     parm.pack->key            = "pack";
119     parm.pack->type           = TYPE_INTEGER;
120     parm.pack->required       = YES;
121     parm.pack->description= "Use_BCF=0_LPF=1";
122
123     parm.ibd = G_define_option();
124     parm.ibd->key            = "region";
125     parm.ibd->type           = TYPE_STRING;
126     parm.ibd->required       = YES;
127     parm.ibd->multiple       = YES;
128     parm.ibd->gisprompt      = "old , cell , raster" ;
129     parm.ibd->description= "Name_of_raster_map_with_boundaries_information" ;
130
131     parm.ihd = G_define_option();
132     parm.ihd->key            = "heads";
133     parm.ihd->type           = TYPE_STRING;
134     parm.ihd->required       = YES;
135     parm.ihd->multiple       = YES;
136     parm.ihd->gisprompt      = "old , cell , raster";
137     parm.ihd->description= "Name_of_raster_map_with_initial_heads";
138
139     parm.stress = G_define_option();
140     parm.stress->key="stress";
141     parm.stress->type = TYPE_INTEGER;
142     parm.stress->required= YES;
143     parm.stress->description="Number_of_stress_periods";
144
145     parm.state = G_define_option(); /* BCF file*/
146     parm.state ->key = "simulation";
147     parm.state ->type = TYPE_INTEGER;
148     parm.state ->required = YES;
149     parm.state -> multiple = YES;
150     parm.state ->description = "Type_of_simulation_Transient=0,_Steady_state=1
151     ****for_each_stress_period";
152
153     parm.type = G_define_option();
154     parm.type ->key = "aqtype";
155     parm.type -> type = TYPE_INTEGER;
156     parm.type -> required = YES;
157     parm.type -> multiple = YES;
158     parm.type -> description = "Type_of_aquifer_0=Confined_1=Unconfined_2=Variable
159     ****with_constant_T_3=Variable_with_variable_T";
160
161     parm.pstor = G_define_option();
162     parm.pstor->key          = "Sy";
163     parm.pstor->type         = TYPE_STRING;
164     parm.pstor->required     = NO;
165     parm.pstor->multiple     = YES;
166     parm.pstor->gisprompt    = "old , cell , raster";
167     parm.pstor->description = "Name_of_an_existing_raster_map_with_primary_storage
168     ****coefficient_values";
169
170     parm.trans = G_define_option();
171     parm.trans->key          = "T";
172     parm.trans->type         = TYPE_STRING;
173     parm.trans->required     = NO;
174     parm.trans->multiple     = YES;
175     parm.trans->gisprompt    = "old , cell , raster";
176     parm.trans->description = "Name_of_an_existing_raster_map_with_Transmissivity .
177     ****REQUIRED_for_CONFINED_aquifers_or_VARIABLE_confinement_aquifers_with_CONSTANT_T";
178
179     parm.hycond = G_define_option();
180     parm.hycond->key        = "K";

```



```

181     parm.hycond->type      = TYPE_STRING;
182     parm.hycond->required  = NO;
183     parm.hycond->multiple  = YES;
184     parm.hycond->gisprompt = "old , cell , raster";
185     parm.hycond->description= "Name_of_an_existing_raster_map_with_Horizontal_Conductivity_
186     Required_for_UNCONFINED_aquifers_and_VARIABLE_confinment_aquifers_with_VARIABLE_T";
187
188     parm.vcondu = G_define_option();
189     parm.vcondu->key      = "vcond";
190     parm.vcondu->type     = TYPE_STRING;
191     parm.vcondu->required  = NO;
192     parm.vcondu->multiple  = YES;
193     parm.vcondu->gisprompt = "old , cell , raster";
194     parm.vcondu->description= "Name_of_an_existing_raster_map_with_Vertical_conductance";
195
196     parm.sstor = G_define_option();
197     parm.sstor->key      = "Syb";
198     parm.sstor->type     = TYPE_STRING;
199     parm.sstor->required  = NO;
200     parm.sstor->multiple  = YES;
201     parm.sstor->gisprompt = "old , cell , raster";
202     parm.sstor->description= "Name_of_an_existing_raster_map_with_Secondary_Storage_Values";
203
204     parm.spes = G_define_option(); /*lpf options*/
205     parm.spes->key      = "Ss";
206     parm.spes->type     = TYPE_STRING;
207     parm.spes->required  = NO;
208     parm.spes->multiple  = YES;
209     parm.spes->gisprompt = "old , cell , raster";
210     parm.spes->description= "LPF_:Name_of_an_existing_raster_map_with_Specific_Storage_Values";
211
212     parm.sy = G_define_option();
213     parm.sy->key      = "Syl";
214     parm.sy->type     = TYPE_STRING;
215     parm.sy->required  = NO;
216     parm.sy->multiple  = YES;
217     parm.sy->gisprompt = "old , cell , raster";
218     parm.sy->description= "LPF_:Name_of_an_existing_raster_map_with_Specific_Yield_Values";
219
220     parm.vkcb = G_define_option();
221     parm.vkcb->key      = "vkcb";
222     parm.vkcb->type     = TYPE_STRING;
223     parm.vkcb->required  = NO;
224     parm.vkcb->multiple  = YES;
225     parm.vkcb->gisprompt = "old , cell , raster";
226     parm.vkcb->description= "LPF_:Name_of_an_existing_raster_map_with_Vertical_Hydraulic
227     Conductivity_Values_of_Quasi3D_confining_bed";
228
229     parm.topelev = G_define_option(); /*DIS FILE*/
230     parm.topelev->key      = "top";
231     parm.topelev->type     = TYPE_STRING;
232     parm.topelev->required  = NO;
233     parm.topelev->multiple  = YES;
234     parm.topelev->gisprompt = "old , cell , raster";
235     parm.topelev->description= "Name_of_an_existing_raster_map_with_Top_of_aquifer";
236
237     parm.tunits = G_define_option();
238     parm.tunits->key="tunits";
239     parm.tunits->type=TYPE_INTEGER;
240     parm.tunits->required=YES;
241     parm.tunits->description="Time_units_Seconds=_1,_minutes=_2,_hours=3,_days=4,_years=5: ";
242
243     parm.lunits = G_define_option();
244     parm.lunits->key="lunits";
245     parm.lunits->type=TYPE_INTEGER;
246     parm.lunits->required=YES;
247     parm.lunits->description="Length_units_undefined=_0,feet=_1,meters=_2,centimeters=_3:";
248

```

```

249     parm.bed = G_define_option();
250     parm.bed->key="bed";
251     parm.bed->type=TYPE_INTEGER;
252     parm.bed->multiple = YES;
253     parm.bed->required=YES;
254     parm.bed->description="For_each_layer_l1=confining_bed_l0=no_confining_bed";
255
256     parm.bottelev = G_define_option();
257     parm.bottelev->key      = "bottom";
258     parm.bottelev->type     = TYPE_STRING;
259     parm.bottelev->required = NO;
260     parm.bottelev->multiple = YES;
261     parm.bottelev->gisprompt = "old , cell , raster";
262     parm.bottelev->description= "Name_of_an_existing_raster_map_with_aquifer_bottom
263     and_confining_bed_bottom";
264
265     parm.length=G_define_option();
266     parm.length->key="length";
267     parm.length->type=TYPE_INTEGER;
268     parm.length->required=NO;
269     parm.length->multiple=YES;
270     parm.length->description="Length_of_each_stress_period";
271
272     parm.steps = G_define_option();
273     parm.steps->key="steps";
274     parm.steps->type= TYPE_INTEGER;
275     parm.steps->required=NO;
276     parm.steps->multiple=YES;
277     parm.steps->description= "Number_of_time_steps_for_each_stress_period";
278
279     parm.tsmult=G_define_option();
280     parm.tsmult->key="tsmult";
281     parm.tsmult->type=TYPE_STRING;
282     parm.tsmult->required=NO;
283     parm.tsmult->multiple=YES;
284     parm.tsmult->description="Time_step_multiplier_for_each_stress_period";
285
286     parm.river = G_define_option() ; /*river options*/
287     parm.river->key      = "river_heads";
288     parm.river->type     = TYPE_STRING;
289     parm.river->required = NO;
290     parm.river->gisprompt = "old , cell , raster" ;
291     parm.river->description= "Name_of_the_raster_map_containing_the_head_values_in_rivers" ;
292
293     parm.rivcond = G_define_option() ;
294     parm.rivcond->key      = "river_cond";
295     parm.rivcond->type     = TYPE_STRING;
296     parm.rivcond->required = NO;
297     parm.rivcond->gisprompt = "old , cell , raster" ;
298     parm.rivcond->description= "Name_of_the_raster_map_containing_the_conductance_values
299     in_rivers";
300
301     parm.rivelev = G_define_option() ;
302     parm.rivelev->key      = "river_elev";
303     parm.rivelev->type     = TYPE_STRING;
304     parm.rivelev->required = NO;
305     parm.rivelev->gisprompt = "old , cell , raster" ;
306     parm.rivelev->description= "Name_of_the_raster_map_containing_the_elevation_values" ;
307
308     parm.rch=G_define_option();
309     parm.rch->key      = "recharge";
310     parm.rch->type     =TYPE_STRING;
311     parm.rch->required =NO;
312     parm.rch->multiple = YES;
313     parm.rch->gisprompt = "old , cell , raster";
314     parm.rch->description = "Name_of_an_existing_raster_map_with_recharge_values
315     for_each_stress_period";
316

```

```

317     parm.drawdown = G_define_option ();
318     parm.drawdown ->key      = "drawdownin";
319     parm.drawdown ->type     = TYPE_STRING;
320     parm.drawdown ->required = YES;
321     parm.drawdown ->description= "Name_of_raster_map_for_DRAWDOWN_simulation_values";
322
323     parm.headsin = G_define_option ();
324     parm.headsin ->key      = "headsin";
325     parm.headsin ->type     = TYPE_STRING;
326     parm.headsin ->required = YES;
327     parm.headsin ->description= "Name_of_raster_map_for_HEAD_simulation_values";
328
329     parm.surf=G_define_option ();
330     parm.surf ->key      = "evt_elev";
331     parm.surf ->type     =TYPE_STRING;
332     parm.surf ->required =NO;
333     parm.surf ->multiple =YES;
334     parm.surf ->gisprompt = "old , cell , raster";
335     parm.surf ->description = "Name_of_an_existing_raster_map_with_elevation_of_the
336     ET_surface_values_for_each_stress_period";
337
338     parm.evtr=G_define_option ();
339     parm.evtr ->key      = "evt_flow";
340     parm.evtr ->type     =TYPE_STRING;
341     parm.evtr ->required =NO;
342     parm.evtr ->multiple =YES;
343     parm.evtr ->gisprompt = "old , cell , raster";
344     parm.evtr ->description = "Name_of_an_existing_raster_map_with_maximum_ET_flux
345     values_for_each_stress_period";
346
347     parm.exdp =G_define_option ();
348     parm.exdp ->key      = "evt_ext";
349     parm.exdp ->type     =TYPE_STRING;
350     parm.exdp ->required =NO;
351     parm.exdp ->multiple =YES;
352     parm.exdp ->gisprompt = "old , cell , raster";
353     parm.exdp ->description = "Name_of_an_existing_raster_map_with_extinction_depth
354     values_for_each_stress_period";
355
356     parm.ievt =G_define_option ();
357     parm.ievt ->key      = "evt_lay";
358     parm.ievt ->type     =TYPE_STRING;
359     parm.ievt ->required =NO;
360     parm.ievt ->multiple =YES;
361     parm.ievt ->gisprompt = "old , cell , raster";
362     parm.ievt ->description = "Name_of_an_existing_raster_map_with_layer_from_which
363     ET_is_removed_for_each_stress_period";
364
365     parm.wel = G_define_standard_option (G_OPT_V_INPUT);
366     parm.wel->key=" well";
367     parm.wel->description = _("Input_vector_map_containing_wells_information");
368     parm.wel->required= NO;
369
370     parm.extr = G_define_option ();
371     parm.extr->key = "extr";
372     parm.extr->type= TYPE_STRING;
373     parm.extr->required= NO;
374     parm.extr->multiple=NO;
375     parm.extr->description=_("column_name_with_Q_value");
376
377     parm.layer = G_define_option ();
378     parm.layer->key = "layer_wel";
379     parm.layer->type= TYPE_STRING;
380     parm.layer->required= NO;
381     parm.layer->multiple=NO;
382     parm.layer->description=_("column_name_with_layer_number_of_the_well");
383
384     parm.use= G_define_option ();

```

```

385     parm.use->key = "welluse" ;
386     parm.use->type= TYPE_STRING;
387     parm.use->required= NO;
388     parm.use->multiple=YES;
389     parm.use->description=_( "column_name_for_using_well_for_each_stress_period_0=NO
390     _____USE_1=USE" );
391     /* drain options */
392     parm.drn = G_define_standard_option (G_OPT_V_INPUT);
393     parm.drn->key="drn";
394     parm.drn->description = _( "Input_vector_map_containing_drain_information" );
395     parm.drn->required= NO;
396
397     parm.elevdrn = G_define_option ();
398     parm.elevdrn->key = "elev_drn";
399     parm.elevdrn->type= TYPE_STRING;
400     parm.elevdrn->required= NO;
401     parm.elevdrn->multiple=NO;
402     parm.elevdrn->description=_( "column_name_with_the_elevation_of_the_drain" );
403
404     parm.laydr = G_define_option ();
405     parm.laydr->key = "layer_drn";
406     parm.laydr->type= TYPE_STRING;
407     parm.laydr->required= NO;
408     parm.laydr->multiple=NO;
409     parm.laydr->description=_( "column_name_with_layer_number_of_the_drain" );
410
411     parm.usedr= G_define_option ();
412     parm.usedr->key = "drnuse";
413     parm.usedr->type= TYPE_STRING;
414     parm.usedr->required= NO;
415     parm.usedr->multiple=YES;
416     parm.usedr->description=_( "column_name_for_using_drain_for_each_stress
417     _____period_0=NO_USE_1=USE" );
418
419     parm.condrn= G_define_option ();
420     parm.condrn->key = "condrn";
421     parm.condrn->type= TYPE_STRING;
422     parm.condrn->required= NO;
423     parm.condrn->multiple=YES;
424     parm.condrn->description=_( "column_name_for_the_hydraulic_conductance_between
425     _____the_aquifer_and_the_drain" );
426
427     if ( G_parser (argc , argv) )
428         exit (EXIT_FAILURE);
429
430     /*BAS FILE*/
431     bound=parm.ibd->answers;
432     ihead=parm.ihd->answers;
433     sscanf (parm.layers->answer , "%d" ,&NLAY);
434     sscanf (parm.pack->answer , "%d" ,&PACK);
435     HNOFLO=-999.99; /*head of cells that become dry */
436     LOCAT = 1;
437     CONSTN= 1;          /* This value is 1, as the array is read from a raster file*/
438     IPRN= -1;
439     IAPART=0;          /* General value, consult MODFLOW documentation */
440     ISTRT=1;          /* initial heads are saved to compute drawdown */
441                      /* Default value. If ISTRT ==0 they are not saved*/
442     /*BCF FILE*/
443     sscanf (parm.stress->answer , "%d" ,&NPER);
444
445     ptr=parm.type->answers;
446     for ( i=1; i<=NLAY; i++)
447     {
448         sscanf (*ptr , "%d" ,&LAYCON[ i ] );
449         *ptr++;
450     }
451     IBCFCB=50;
452     HDRY=0;

```

```

453     IWDFLG=0;
454     WEIFCT=1;
455     IWETIT=1;
456     IHDWET=1;
457     TRPY=1;
458     sfl=parm.pstor->answers;
459     std=parm.state->answers;
460     for (i=1;i<=NPER;i++)
461     {
462         sscanf(*std,"%d",&ISS[i]);
463         *std++;
464     }
465
466     /*DIS FILE*/
467     sscanf(parm.tunits->answer,"%d",&ITMUNI);
468     sscanf(parm.lunits->answer,"%d",&LENUNI);
469     IPRN=-1;
470     CNSTNT= 1;
471     ptr2=parm.length->answers;
472     stp=parm.steps->answers;
473     mlt=parm.tsmult->answers;
474     std=parm.state->answers;
475
476     for (i=1;i<=NPER;i++)
477     {
478         sscanf(*ptr2,"%d",&PERLEN[i]);
479         sscanf(*stp,"%d",&NSTP[i]);
480         sscanf(*std,"%d",&STATE[i]);
481         ts[i]=atof(*mlt);
482         ptr2++;
483         stp++;
484         mlt++;
485         std++;
486     }
487
488     int cbd=0;
489     ptr=parm.bed->answers;
490     for (i=1;i<=NLAY;i++)
491     {
492         sscanf(*ptr,"%d",&LAYCBD[i]);
493         if (LAYCBD[i]==1){
494             cbd=cbd+1;
495         }
496         ptr++;
497     }
498
499     struct Cell_head region;
500     G_get_window (&region);
501     G_format_northing (region.ew_res , ewres , region.proj);
502     G_format_northing (region.ns_res , nsres , region.proj);
503     NROW = G_window_rows ();
504     printf("NROW:_%d_\n",NROW);
505     NCOL = G_window_cols ();
506     printf("NCOL:_%d_\n",NCOL);
507
508     /******
509     /******funciones para escribir los archivos *****/
510     /******
511     /*BAS FILE*/
512
513     int writeOPTIONS(FILE *fp, int NLAY, int NROW, int NCOL, int NPER,
514     int ITMUNI, int IAPART, int ISTRT)
515     {
516     /*      fprintf(fp,"#Active cells in the region\n");*/
517
518     //fprintf(fp," \n");
519     fprintf(fp,"FREE_\n");
520     return(0);

```

```

521     }
522
523     int write2IBOUND(FILE *fp, int NLAY, int NROW, int NCOL)
524     {
525         void *raster;
526         int row=1,col;
527         for (i=1;i<=NLAY;i++)
528             {
529                 name=*bound;
530                 mapset=G_find_cell2(name,"");
531                 map_type=G_raster_map_type(name, mapset);
532                 out_type=CELL.TYPE;
533                 fd=G_open_cell_old(name, mapset);
534                 raster=G_allocate_raster_buf(out_type);
535                 if (fd < 0)
536                     exit(1);
537
538                 fprintf(fp, "INTERNAL...%10i...(2013).....%9i.....IBOUND_layer...%\n", CONSTNT, IPRN, i);
539                 for (row = 0; row < NROW; row++)
540                     {
541                         G_percent(row, NROW, 2);
542                         if (G_get_raster_row(fd, raster, row, out_type) < 0) return(row);
543                         colcnt=0;
544                         for (col = 0, ptr = raster; col < NCOL; col++, tiny
545                             ptr = G_incr_void_ptr(ptr, G_raster_size(out_type)))
546                             {
547                                 if (G_is_null_value(ptr, out_type)*((CELL *)ptr)=0;
548                                     fprintf(fp, "%3d", *((CELL *)ptr));
549                                     colcnt+=1;
550                                     if (colcnt >=20)
551                                         {
552                                             colcnt=0;
553                                             fprintf(fp, "\n");
554                                         }
555                             }
556                         if (colcnt > 0) fprintf(fp, "\n");
557                         if (rc)
558                             {
559                                 G_fatal_error("Read_failed_at_row...[%d]\n", rc);
560                             }
561                     }
562                 G_close_cell(fd);
563                 *bound++;
564             }
565         return (0);
566     }
567
568     int write4STRT(FILE *fp, int NLAY, int NROW, int NCOL)
569     {
570         void *raster;
571         for (i=1;i<=NLAY;i++)
572             {
573                 name=*ihead;
574                 mapset=G_find_cell2(name,"");
575                 map_type=G_raster_map_type(name, mapset);
576                 out_type=map_type;
577                 fd=G_open_cell_old(name, mapset);
578                 if (out_type==DCELL.TYPE) dp=10;
579                 if (fd < 0)
580                     exit(1);
581                 int row, col, colcnt;
582                 char cell_buf[128];
583                 float flo;
584                 raster=G_allocate_raster_buf(out_type);
585                 fprintf(fp, "INTERNAL...%10i...(20G14.0).....%10i.....Initial_Head_layer...%\n", CONSTNT, IPRN, i);
586                 for (row = 0; row < NROW; row++)
587                     {
588                         colcnt=0;

```

```

589     G_percent(row, NROW, 2);
590     G_percent(row, NROW, 2);
591
592     if (G_get_raster_row(fd, raster, row, out_type) < 0)
593         return(row);
594
595     for (col = 0, ptr = raster; col < NCOL; col++,
596         ptr = G_incr_void_ptr(ptr, G_raster_size(out_type)))
597     {
598
599         if(out_type == CELL_TYPE)
600         {
601             if(G_is_null_value(ptr, out_type))*((CELL *)ptr)=0;
602             flo=(float)*((CELL *)ptr);
603             fprintf(fp, "%14f", flo);
604         }
605         else if(out_type == FCELL_TYPE)
606         {
607             if(G_is_null_value(ptr, out_type))*((FCELL *)ptr)=0;
608             fprintf(fp, "%14.*f", dp,*((FCELL *)ptr));
609         }
610         else if(out_type == DCELL_TYPE)
611         {
612             if(G_is_null_value(ptr, out_type))*((DCELL *)ptr)=0;
613             fprintf(fp, "%14f", dp,*((DCELL *)ptr));
614         }
615         colcnt+=1;
616         if(colcnt>=20)
617         {
618             colcnt=0;
619             fprintf(fp, "\n");
620         }
621     }
622     if(colcnt>0)fprintf(fp, "\n");
623     if(rc)
624     {
625         G_fatal_error("Read_failed_at_row_[%d]\n", rc);
626     }
627 }
628 G_close_cell(fd);
629 *ihead++;
630 }
631 return (0);
632 }
633 /*BCF FILE*/
634 int write_BCF(int fd, FILE *fp, int nrows, int ncols, int out_type, int dp)
635 {
636     int row, col, colcnt;
637     void *ptr, *raster;
638     char cell_buf[300];
639     raster=G_allocate_raster_buf(out_type);
640     for (row = 0; row < nrows; row++)
641     {
642         G_percent(row, nrows, 2);
643
644         if (G_get_raster_row(fd, raster, row, out_type) < 0)return(row);
645
646         colcnt=0;
647         for (col = 0, ptr = raster; col < ncols; col++,
648             ptr = G_incr_void_ptr(ptr, G_raster_size(out_type)))
649         {
650             if(out_type == CELL_TYPE)
651             {
652                 if(G_is_null_value(ptr, out_type))*((CELL *)ptr)=0;
653                 fprintf(fp, "%14d", *((CELL *)ptr));
654             }
655             else if(out_type == FCELL_TYPE)
656             {

```

```

657         if (G_is_null_value(ptr, out_type))*((FCELL *)ptr)=0;
658         fprintf(fp, "%14.*f", dp, *((FCELL *)ptr));
659     }
660     else if (out_type == DCELL_TYPE)
661     {
662         if (G_is_null_value(ptr, out_type))*((DCELL *)ptr)=0;
663         fprintf(fp, "%14.*f", dp, *((DCELL *)ptr));
664     }
665     colcnt+=1;
666     if (colcnt >= 20)
667     {
668         colcnt=0;
669         fprintf(fp, "\n");
670     }
671 }
672 if (colcnt > 0) fprintf(fp, "\n");
673 }
674 return (0);
675 }
676
677 /*BAS FILE*/
678 printf(".....Processing _BAS_ file _now.....\n");
679
680 fp=fopen("model.ba6", "w");
681
682 write1OPTIONS(fp, NLAY, NROW, NCOL, NPER, ITMUNI, IAPART, ISTRT);
683
684 write2IBOUND(fp, NLAY, NROW, NCOL);
685
686 fprintf(fp, "%10f.....HNOFLO\n", HNOFLO);
687
688 write4STRT(fp, NLAY, NROW, NCOL);
689
690 fclose(fp);
691
692 /******
693      /*fin de bas file*/
694      *****/
695 /*BCF FILE*/
696 if (PACK==0){
697
698 printf(".....Processing _BCF_ file _now.....\n");
699
700 fp=fopen("model.bc6", "w");
701 /*ITEM 1*/
702 fprintf(fp, "%10i _%9i _", IBCFCB, HDRY);
703 fprintf(fp, "%9i _%9i _%9i _%9i\n", IWDFLG, WEIFCT, IWETIT, IHDWET);
704 /*ITEM 2*/
705 for (i=1; i<=NLAY; i++)
706 {
707     fprintf(fp, "%2i _", LAYCON[i]);
708 }
709     fprintf(fp, "\n");
710 /*ITEM 3*/
711 fprintf(fp, "CONSTANT_%9i _TRPY\n", TRPY);
712 /*ITEM 4*/
713 st=1;
714 for (i=1; i<=NPER; i++)
715 {
716     if (ISS[i]==0){
717         st=0;
718     }
719 }
720 if (st==0)
721 {
722     if (!parm.pstor->answers)
723     {
724         G_fatal_error("Storage_coefficient_is_needed_for_transient_simulations");

```



```

725     }
726     sf1=parm.pstor->answers;
727 }
728 /*ITEM 5*/
729 usestran=0;
730 for (i=1;i<=NLAY;i++)
731     {
732         if (LAYCON[i]==0||LAYCON[i]==2)
733             {
734                 usestran=1; /* To use transmissivity values */
735             }
736     }
737     if (usestran==1)
738     {
739         if (!parm.trans->answers)
740         {
741             G_fatal_error(" Transmissivity _values _are _needed");
742         }
743         else
744             tran=parm.trans->answers;
745     }
746
747 /*ITEM 6*/
748 for (i=1;i<=NLAY;i++)
749     {
750         if (LAYCON[i]==1||LAYCON[i]==3)
751             {
752                 if (!parm.hycond->answers)
753                 {
754                     G_fatal_error(" Horizontal _conductance _values _are _needed\n");
755                 }
756                 hcond=parm.hycond->answers;
757             }
758     }
759 /*ITEM 7*/
760     if (NLAY>1)
761     if (!parm.vcondu->answers)
762     {
763         G_fatal_error(" Vertical _conductance _values _are _needed\n");
764     }
765     else
766     {
767         vcond=parm.vcondu->answers;
768     }
769
770 /*ITEM 8*/
771 usesf2=0;
772 if (LAYCON[i]==2||LAYCON[i]==3)
773     {
774         if (st==0)
775             {
776                 usesf2=1;
777             }
778     }
779
780     if (usesf2==1)
781     if (!parm.sstor->answers)
782     {
783         G_fatal_error(" Secondary _storage _values _are _needed");
784     }
785     else
786     {
787         sf2=parm.sstor->answers;
788     }
789 /*PARA CADA CAPA*/
790 for (i=1;i<=NLAY;i++)
791     {
792         if (st==0)

```

```

793     {
794         fprintf(fp,"INTERNAL...%10i_(20G14.0).....%10i_\n",CONSTNT,IPRN);
795         name=*sf1; /* Array for line 6 */
796         mapset=G_find_cell2(name,"");
797         if (mapset == NULL)
798             {
799                 G_fatal_error (" Cell_file _[%s] _not_found\n",name);
800             }
801         map_type=G_raster_map_type(name, mapset);
802         out_type=map_type;
803         if(out_type==DCELL.TYPE)dp=10;
804         fd=G_open_cell_old(name, mapset);
805         if (fd < 0)
806             exit(1);
807         rc=write_BCF(fd,fp,NROW,NCOL,out_type,dp);
808         if(rc)
809             {
810                 G_fatal_error("Read_failed_at_row_[%d]\n",rc);
811             }
812         G_close_cell(fd);
813         *sf1++;
814     }
815
816 if(LAYCON[i]==0||LAYCON[i]==2)
817     {
818         fprintf(fp,"INTERNAL...%10i_(20G14.0).....%10i_\n",CONSTNT,IPRN);
819         name=*tran;
820         mapset = G_find_cell2 (name,"");
821         if (mapset == NULL)
822             {
823                 G_fatal_error (" Cell_file _[%s] _not_found\n",name);
824             }
825         map_type = G_raster_map_type(name, mapset);
826         out_type = map_type;
827         if(out_type==DCELL.TYPE)dp=10;
828         fd = G_open_cell_old (name, mapset);
829         if (fd < 0)
830             exit(1);
831         rc=write_BCF(fd,fp,NROW,NCOL,out_type,dp);
832         if(rc)
833             {
834                 G_fatal_error("Read_failed_at_row_[%d]\n",rc);
835             }
836         G_close_cell(fd);
837         *tran++;
838     }
839
840 if(LAYCON[i]==1||LAYCON[i]==3)
841     {
842         fprintf(fp,"INTERNAL...%10i_(20G14.0).....%10i_\n",CONSTNT,IPRN);
843         name=*hcond;
844         mapset = G_find_cell2 (name,"");
845         if (mapset == NULL)
846             {
847                 G_fatal_error (" Cell_file _[%s] _not_found\n",name);
848             }
849         map_type = G_raster_map_type(name, mapset);
850         out_type = map_type;
851         if(out_type==DCELL.TYPE)dp=10;
852         /* open raster file */
853         fd = G_open_cell_old (name, mapset);
854         if (fd < 0)
855             exit(1);
856         rc=write_BCF(fd,fp,NROW,NCOL,out_type,dp);
857         if(rc)
858             {
859                 G_fatal_error("Read_failed_at_row_[%d]\n",rc);
860             }

```

```

861         G_close_cell ( fd );
862         *hcond++;
863     }
864     if (NLAY>1){
865     if (i<NLAY)
866     {
867     fprintf ( fp , "INTERNAL__%10i_(20G14.0)_____ %10i_\n" ,CONSTNT, IPRN);
868     name=*vcond;
869     mapset = G_find_cell2 ( name, "" );
870     map_type = G_raster_map_type ( name, mapset );
871     out_type = map_type;
872     if (out_type==DCELL.TYPE) dp=10;
873     if ( mapset == NULL)
874     {
875     G_fatal_error ( " Cell_file _[%s] _not_found\n" , name );
876     }
877     fd = G_open_cell_old ( name, mapset );
878     if ( fd < 0)
879     exit ( 1 );
880     rc=write_BCF ( fd , fp ,NROW,NCOL, out_type , dp );
881     if (rc)
882     {
883     G_fatal_error ( " Read_failed_at_row_[%d]\n" , rc );
884     }
885     G_close_cell ( fd );
886     *vcond++;
887     }
888     }
889
890     if (st==0){
891     if (LAYCON[ i ]==2||LAYCON[ i ]==3)
892     {
893     fprintf ( fp , "INTERNAL__%10i_(20G14.0)_____ %10i_\n" ,CONSTNT, IPRN);
894     name=*sf2;
895     mapset = G_find_cell2 ( name, "" );
896     if ( mapset == NULL)
897     {
898     G_fatal_error ( " Cell_file _[%s] _not_found\n" , name );
899     }
900     map_type = G_raster_map_type ( name, mapset );
901     out_type = map_type;
902     if (out_type==DCELL.TYPE) dp=10;
903     fd = G_open_cell_old ( name, mapset );
904     if ( fd < 0)
905     exit ( 1 );
906     rc=write_BCF ( fd , fp ,NROW,NCOL, out_type , dp );
907     if (rc)
908     {
909     G_fatal_error ( " Read_failed_at_row_[%d]\n" , rc );
910     }
911     G_close_cell ( fd );
912     }
913     }
914 }
915 fclose ( fp );
916 }
917 /*****
918     /*fin de BCF file*/
919     *****/
920 int ILPFCB, NLPFCB, LAYTYP[100], LAYAVG[100], CHANI[100], LAYVKA[100], LAYWET[100], cb;
921 char **spes, **sy, **vkcb;
922 /*LPF FILE*/
923 if (PACK==1){
924
925     st=1;
926
927     for ( i=1; i<=NPER; i++)
928     {

```

```

929     if (ISS[i]==0){
930         st=0;
931     }
932 }
933     printf(" Processing LPF file now\n");
934     fp=fopen("model.lpf","w");
935
936     ILPFCB=50;
937     NPLPF=0;
938
939     fprintf(fp," %10i %10i %10i\n",ILPFCB,HDRY,NPLPF);
940
941     for (i=1;i<=NLAY;i++)
942     {
943         LAYTYP[i]=LAYCON[i];
944         LAYAVG[i]=0;
945         CHANI[i]=1;
946         LAYVKA[i]=0;
947         LAYWET[i]=0; /*humectacion activa*/
948         fprintf(fp," %2i\n",LAYTYP[i]);
949         fprintf(fp," %2i\n",LAYAVG[i]);
950         fprintf(fp," %2i\n",CHANI[i]);
951         fprintf(fp," %2i\n",LAYVKA[i]);
952         fprintf(fp," %2i\n",LAYWET[i]);
953     }
954 }
955
956 /*HK hydraulic conductivity*/
957 if (!parm.hycond->answers)
958 {
959     G_fatal_error(" Horizontal conductance values are needed\n");
960 }
961     hcond=parm.hycond->answers;
962     for (i=1;i<=NLAY;i++)
963     {
964         fprintf(fp,"INTERNAL %10i (20G14.0) %10i\n",CONSTNT,IPRN);
965         name=*hcond;
966         mapset = G_find_cell2 (name,"");
967         if (mapset == NULL)
968         {
969             G_fatal_error (" Cell file [%s] not found\n",name);
970         }
971         map_type = G_raster_map_type(name, mapset);
972         out_type = map_type;
973         if (out_type==DCELL.TYPE)dp=10;
974         /* open raster file */
975         fd = G_open_cell_old (name, mapset);
976         if (fd < 0)
977             exit (1);
978         rc=write_BCF (fd, fp ,NROW,NCOL, out_type , dp);
979         if (rc)
980         {
981             G_fatal_error (" Read failed at row [%d]\n",rc);
982         }
983         G_close_cell (fd);
984         *hcond++;
985     }
986 /*vka vertical hydraulic conductivity */
987 if (!parm.vcondu->answers)
988 {
989     G_fatal_error(" Vertical hydraulic conductance values are needed\n");
990 }
991     vcond=parm.vcondu->answers;
992
993     for (i=1;i<=NLAY;i++)
994     {
995
996

```

```

997     fprintf(fp, "INTERNAL...%10i...(20G14.0) .....%10i...\n", CONSTNT, IPRN);
998     name=*vcond;
999     mapset = G_find_cell2 (name, "");
1000    map_type = G_raster_map_type(name, mapset);
1001    out_type = map_type;
1002    if(out_type==DCELL_TYPE)dp=10;
1003    if (mapset == NULL)
1004    {
1005        G_fatal_error (" Cell_file_[%s]_not_found\n", name);
1006    }
1007    fd = G_open_cell_old (name, mapset);
1008    if (fd < 0)
1009        exit (1);
1010    rc=write_BCF (fd, fp, NROW, NCOL, out_type, dp);
1011    if(rc)
1012    {
1013        G_fatal_error (" Read_failed_at_row_[%d]\n", rc);
1014    }
1015    G_close_cell(fd);
1016    *vcond++;
1017 }
1018 /* specific storage Ss*/
1019
1020     if (!parm.spes->answers)
1021     {
1022         G_fatal_error (" Specific_Storage_is_needed\n");
1023     }
1024     spes=parm.spes->answers;
1025
1026 for (i=1; i<=NLAY; i++)
1027 {
1028     fprintf(fp, "INTERNAL...%10i...(20G14.0) .....%10i...\n", CONSTNT, IPRN);
1029     name=*spes;
1030     mapset = G_find_cell2 (name, "");
1031     map_type = G_raster_map_type(name, mapset);
1032     out_type = map_type;
1033     if(out_type==DCELL_TYPE)dp=10;
1034     if (mapset == NULL)
1035     {
1036         G_fatal_error (" Cell_file_[%s]_not_found\n", name);
1037     }
1038     fd = G_open_cell_old (name, mapset);
1039     if (fd < 0)
1040         exit (1);
1041     rc=write_BCF (fd, fp, NROW, NCOL, out_type, dp);
1042     if(rc)
1043     {
1044         G_fatal_error (" Read_failed_at_row_[%d]\n", rc);
1045     }
1046     G_close_cell(fd);
1047     *spes++;
1048 }
1049 /*SPECIFIC YIELD*/
1050 if (st==0){
1051
1052     if (!parm.sy->answers)
1053     {
1054         G_fatal_error (" Specific_Yield_for_transient_simulation_is_needed\n");
1055     }
1056
1057     sy=parm.sy->answers;
1058
1059 for (i=1; i<=NLAY; i++)
1060 {
1061     fprintf(fp, "INTERNAL...%10i...(20G14.0) .....%10i...\n", CONSTNT, IPRN);
1062     name=*sy;
1063     mapset = G_find_cell2 (name, "");
1064     map_type = G_raster_map_type(name, mapset);

```

```

1065         out_type = map_type;
1066         if (out_type==DCELL.TYPE)dp=10;
1067         if (mapset == NULL)
1068             {
1069                 G_fatal_error (" Cell_file [%s] not found\n",name);
1070             }
1071         fd = G_open_cell_old (name, mapset);
1072         if (fd < 0)
1073             exit (1);
1074         rc=write_BCF (fd , fp ,NROW,NCOL, out_type , dp);
1075         if (rc)
1076             {
1077                 G_fatal_error (" Read_failed_at_row [%d]\n" , rc);
1078             }
1079         G_close_cell (fd);
1080         *sy++;
1081     }
1082 }
1083
1084 /*vertical hydraulic conductivity of a quasy 3d confining bed BKCB*/
1085 for (i=1;i<=NLAY;i++)
1086     {
1087         if (LAYCBD[i]==1){
1088
1089         if (!parm.vkcb->answers)
1090             {
1091                 G_fatal_error (" Vertical_Conductance_for_confining_bed is needed\n");
1092             }
1093         vkcb=parm.vkcb->answers;
1094
1095         for (i=1;i<=NLAY;i++)
1096             {
1097                 fprintf (fp , "INTERNAL_ %10i (20G14.0) _ _ _ _ _ _ _ _ _ _ %10i \n" ,CONSTNT,IPRN);
1098                 name=*vkcb;
1099                 mapset = G_find_cell2 (name,"");
1100                 map_type = G_raster_map_type(name, mapset);
1101                 out_type = map_type;
1102                 if (out_type==DCELL.TYPE)dp=10;
1103                 if (mapset == NULL)
1104                     {
1105                         G_fatal_error (" Cell_file [%s] not found\n",name);
1106                     }
1107                 fd = G_open_cell_old (name, mapset);
1108                 if (fd < 0)
1109                     exit (1);
1110                 rc=write_BCF (fd , fp ,NROW,NCOL, out_type , dp);
1111                 if (rc)
1112                     {
1113                         G_fatal_error (" Read_failed_at_row [%d]\n" , rc);
1114                     }
1115                 G_close_cell (fd);
1116                 *vkcb++;
1117             }
1118     }
1119 }
1120
1121 fclose (fp);
1122 }
1123
1124 /*DIS FILE*/
1125 ewr =strtof(ewres , NULL);
1126 nsr =strtof(nsres , NULL);
1127 float flo;
1128 printf (" _ _ _ _ _ Processing _DIS_ file _now_ _ _ _ _ _ _ _ _ _ \n");
1129 fp=fopen ("model.dis" ,"w");
1130 //fprintf (fp,"# discretization file\n");
1131 fprintf (fp , " %10i %10i %10i %10i %10i %10i \n" , NLAY, NROW, NCOL, NPER, ITMUNI, LENUNI);
1132

```

```

1133     for (i=1;i<=NLAY;i++)
1134     {
1135     fprintf(fp,"%2i \n",LAYCBD[i]);
1136     }
1137     fprintf(fp,"CONSTANT...%f...DELCL\n",nsr);
1138     fprintf(fp,"CONSTANT...%f...DECLR\n",ewr);
1139
1140     /*ITEM 5*/
1141     if (!parm.topelev->answers)
1142     {
1143     G_fatal_error("Top_elevation_values_are_needed");
1144     }
1145     else
1146     {
1147     fprintf(fp,"INTERNAL...%10i...(20G14.0).....%10i...\n",CONSTNT,IPRN);
1148     bot=parm.topelev->answers;
1149     name=*bot;
1150     mapset=G_find_cell2(name,"");
1151     map_type=G_raster_map_type(name,mapset);
1152     out_type=CELL_TYPE;
1153     fd=G_open_cell_old(name,mapset);
1154     raster=G_allocate_raster_buf(out_type);
1155     if (fd < 0)
1156     exit(1);
1157     for (row = 0; row < NROW; row++)
1158     {
1159     G_percent(row, NROW, 2);
1160     if (G_get_raster_row(fd, raster, row, out_type) < 0) return(row);
1161     colcnt=0;
1162     for (col = 0, ptr = raster; col < NCOL;
1163     col++,
1164     ptr = G_incr_void_ptr(ptr, G_raster_size(out_type)))
1165     {
1166     if (G_is_null_value(ptr, out_type))*((CELL *)ptr)=0;
1167     //flo=(float)*((CELL *)ptr);
1168     fprintf(fp,"%14d",*((CELL *)ptr));
1169     colcnt+=1;
1170     if (colcnt >=20)
1171     {
1172     colcnt=0;
1173     fprintf(fp," \n");
1174     }
1175     }
1176     if (colcnt > 0) fprintf(fp," \n");
1177     }
1178     }
1179     /*ITEM 6*/
1180     if (!parm.bottelev->answers)
1181     {
1182     G_fatal_error("Bottom_elevation_values_are_needed");
1183     }
1184     else
1185     {
1186     bot=parm.bottelev->answers;
1187     for (i=1;i<=(NLAY+cbd); i++)
1188     {
1189     fprintf(fp,"INTERNAL...%10i...(20G14.0).....%10i...\n",CONSTNT,IPRN);
1190     name=*bot;
1191     mapset=G_find_cell2(name,"");
1192     map_type=G_raster_map_type(name,mapset);
1193     out_type=CELL_TYPE;
1194     fd=G_open_cell_old(name,mapset);
1195     raster=G_allocate_raster_buf(out_type);
1196     if (fd < 0)
1197     exit(1);
1198     for (row = 0; row < NROW; row++)
1199     {
1200     G_percent(row, NROW, 2);

```

```

1201     if (G_get_raster_row(fd, raster, row, out_type) < 0)
1202         return(row);
1203     colcnt=0;
1204     for (col = 0, ptr = raster; col < NCOL;
1205         col++,
1206         ptr = G_incr_void_ptr(ptr, G_raster_size(out_type)))
1207     {
1208         if (G_is_null_value(ptr, out_type))*((CELL *)ptr)=0;
1209         flo=(float)*((CELL *)ptr);
1210         fprintf(fp, "%14f", flo);
1211         colcnt+=1;
1212         if (colcnt >=20)
1213             {
1214                 colcnt=0;
1215                 fprintf(fp, "\n");
1216             }
1217     }
1218     if (colcnt > 0) fprintf(fp, "\n");
1219
1220 }
1221 *bot++;
1222 }
1223 }
1224
1225     for (i=1; i <= NPER; i++)
1226     {
1227         if (STATE[i]==1)
1228             fprintf(fp, "%10i_%9i_%9f_ _Ss\n", PERLEN[i], NSTP[i], ts[i]);
1229         if (STATE[i]==0)
1230             fprintf(fp, "%10i_%9i_%9f_ _Tr\n", PERLEN[i], NSTP[i], ts[i]);
1231     }
1232 }
1233     fclose(fp);
1234 /******
1235     /* fin de DIS file */
1236 /******
1237 /* OC FILE */
1238     printf(".....Processing_OC_file_now.....\n");
1239     fp=fopen("model.oc", "w");
1240     int IHEDFM, IDDNFM, IHEDUN, IDDNUN, INCODE, IHDDFL, IBUDFL, ICBCFL, HDPR, DDP, HDSV, DDSV;
1241     INCODE=1;
1242     IHEDFM=9;
1243     IDDNFM=9;
1244     IHEDUN=51;
1245     IDDNUN=52;
1246     fprintf(fp, "#output_control_file\n");
1247     fprintf(fp, "%10i_%9i_%9i_%9i\n", IHEDFM, IDDNFM, IHEDUN, IDDNUN);
1248     IHDDFL=1;
1249     IBUDFL=1;
1250     ICBCFL=1;
1251     HDPR=0;
1252     DDP=0;
1253     for (i=1; i <= NPER; i++)
1254     {
1255         for (j=1; j <= NSTP[i]; j++)
1256         {
1257             fprintf(fp, "%10i_%9i_%9i_%9i_---PERIOD_%9i_ _TIMESTEP_%9i\n",
1258                 INCODE, IHDDFL, IBUDFL, ICBCFL, i, j);
1259             for (k=1; k <= NLAY; k++)
1260             {
1261                 HDSV=k;
1262                 DDSV=k;
1263                 fprintf(fp, "%10i_%9i_%9i_%9i_-----LAYER_ _%9i\n", HDPR, DDP, HDSV, DDSV, k);
1264             }
1265         }
1266     }
1267     fclose(fp);
1268 /******

```



```

1269      /*fin de OC file*/
1270  /******
1271  /*PCG2 FILE*/
1272  printf(" \n\n\nProcessing_PCG2 file \nnow\n\n");
1273  fp=fopen("model.pcg","w");
1274  //fprintf(fp,"#comment\n");
1275  int MXITER, ITER1,NPCOND,NBPOL,IPRPCG,MUTPCG,IPCGCD;
1276  float HCLOSE,RCLOSE,RELAX;
1277
1278  /* Modify so the user can modify these values*/
1279  /*   if (LAYCON==0)
1280      {
1281          MXITER=1;
1282          ITER1=30;
1283      }
1284  else
1285      {*/
1286          MXITER=50;
1287          ITER1=7;
1288      // }
1289  NPCOND=1;
1290  HCLOSE=RCLOSE=0.001;
1291  RELAX=1;
1292  NBPOL=2;
1293  IPRPCG=1;
1294  MUTPCG=0;
1295  IPCGCD=0;
1296  fprintf(fp," %10i_%9i_%9i\n",MXITER,ITER1,NPCOND);
1297  /*fprintf(fp," 0.100E-02 0.100E-02 0.100E+01\n");
1298  fprintf(fp,"          2          1          0          0");*/
1299
1300  fprintf(fp," %10f_%9f_%9f_%9i_%9i_%9i_%9i\n", HCLOSE,RCLOSE,RELAX,NBPOL,
1301  IPRPCG,MUTPCG,IPCGCD);
1302
1303  fclose(fp);
1304  /******
1305  /* RCH file */
1306  /******
1307
1308  int NRCHOP, IRCHCB, INRECH, INIRCH;
1309  char **rech;
1310  fp=fopen("model.rch","w");
1311  // fprintf(fp,"#comment\n");
1312  if(parm.rch->answers)
1313  {
1314      printf(" \n\n\nProcessing_RCH file \n\n");
1315      NRCHOP=1; /* =1 Recharge occurs in the top layer*/
1316      IRCHCB=50; /* unit number for cell by cell flow terms */
1317      INRECH=1; /* If >=1 then the RCH array is read; */
1318      INIRCH=0;
1319      LOCAT=18;
1320
1321      fprintf(fp," %10i_%9i\n", NRCHOP, IRCHCB);
1322      rech = parm.rch->answers;
1323  for (i=1;i<=NPER;i++)
1324      {
1325          fprintf(fp," %10i_%9i\n", INRECH, INIRCH);
1326          fprintf(fp," %10i_%9i (20G14.0) %8i\n", LOCAT, CNSTNT, IPRN);
1327          fprintf(fp,"--Recharge_rates_for_stress_period_%i\n",i);
1328          name=rech;
1329          mapset = G_find_cell2 (name,"");
1330          map_type = G_raster_map_type(name, mapset);
1331          out_type = map_type;
1332          if(out_type==DCELL.TYPE)dp=10;
1333          if (mapset == NULL)
1334              {
1335                  G_fatal_error (" Cell file [%s] not found\n",name);
1336              }

```

```

1337         /* open raster file */
1338         fd = G_open_cell_old (name, mapset);
1339         if (fd < 0)
1340             exit (1);
1341         rc=write_BCF (fd , fp ,NROW,NCOL,out_type ,dp);
1342
1343         G_close_cell (fd);
1344         *rech++;
1345     }
1346     fclose (fp);
1347 }
1348
1349 /******
1350 /*          EVT file          */
1351 /******
1352 int NEVTOP, IEVTCB;
1353 int INSURF, INEVTR, INEXDP, INIEVT;
1354 char **etsurf, **etevtr, **etexdp, **etievt;
1355 fp=fopen ("model.evt", "w");
1356 if (parm.surf->answers)
1357     {
1358     printf ("\n-----Processing EVT file -----\n");
1359     NEVTOP=2;
1360     IEVTCB=50;
1361     INSURF=0;
1362     INEVTR=0;
1363     INEXDP=0;
1364     INIEVT=0;
1365
1366     fprintf (fp, "%10i_%9i\n", NEVTOP, IEVTCB);
1367     etsurf = parm.surf->answers;
1368     etevtr = parm.evtr->answers;
1369     etexdp = parm.exdp->answers;
1370     etievt = parm.ievt->answers;
1371
1372     for (i=1;i<=NPER;i++)
1373     {
1374         fprintf (fp, "%10i_%9i_%9i_%9i\n", INSURF, INEVTR, INEXDP, INIEVT);
1375         fprintf (fp, "%10i_%9i (20G14.0) -----%8i ----", LOCAT, CNSTNT, IPRN);
1376         fprintf (fp, "-----elevation_ET_for_stress_period_%a\n", i);
1377         name=*etsurf;
1378         mapset = G_find_cell2 (name, "");
1379         map_type = G_raster_map_type (name, mapset);
1380         out_type = map_type;
1381         if (out_type==DCELL.TYPE)dp=10;
1382         if (mapset == NULL)
1383             {
1384             G_fatal_error (" Cell_file_ [%s]_not_found\n", name);
1385             }
1386         /* open raster file */
1387         fd = G_open_cell_old (name, mapset);
1388         if (fd < 0)
1389             exit (1);
1390         rc=write_BCF (fd , fp ,NROW,NCOL,out_type ,dp);
1391
1392         G_close_cell (fd);
1393
1394         fprintf (fp, "%10i_%9i (20G14.0) -----%8i ----", LOCAT, CNSTNT, IPRN);
1395         fprintf (fp, "-----maximun_ET_flux_for_stress_period_%a\n", i);
1396         name=*etevtr;
1397         mapset = G_find_cell2 (name, "");
1398         map_type = G_raster_map_type (name, mapset);
1399         out_type = map_type;
1400         if (out_type==DCELL.TYPE)dp=10;
1401         if (mapset == NULL)
1402             {
1403             G_fatal_error (" Cell_file_ [%s]_not_found\n", name);
1404             }
1405         /* open raster file */

```

```

1405     fd = G_open_cell_old (name, mapset);
1406     if (fd < 0)
1407         exit (1);
1408     rc=write_BCF (fd, fp, NROW, NCOL, out_type, dp);
1409
1410     G_close_cell (fd);
1411
1412     fprintf (fp, "%10i %9i (20G14.0) %8i\n", LOCAT, CNSTNT, IPRN);
1413     fprintf (fp, "--extinction_depth_for_stress_period_%i\n", i);
1414     name=*etexdp;
1415     mapset = G_find_cell2 (name, "");
1416     map_type = G_raster_map_type (name, mapset);
1417     out_type = map_type;
1418     if (out_type==DCELLTYPE) dp=10;
1419     if (mapset == NULL)
1420     {
1421         G_fatal_error (" Cell_file [%s] not found\n", name);
1422     }
1423     /* open raster file */
1424     fd = G_open_cell_old (name, mapset);
1425     if (fd < 0)
1426         exit (1);
1427     rc=write_BCF (fd, fp, NROW, NCOL, out_type, dp);
1428
1429     G_close_cell (fd);
1430
1431     fprintf (fp, "%10i %9i (20G14.0) %8i\n", LOCAT, CNSTNT, IPRN);
1432     fprintf (fp, "--layer_form_which_ET_is_remove_for_stress_period_%i\n", i);
1433     name=*etievt;
1434     mapset = G_find_cell2 (name, "");
1435     map_type = G_raster_map_type (name, mapset);
1436     out_type = map_type;
1437     if (out_type==DCELLTYPE) dp=10;
1438     if (mapset == NULL)
1439     {
1440         G_fatal_error (" Cell_file [%s] not found\n", name);
1441     }
1442     /* open raster file */
1443     fd = G_open_cell_old (name, mapset);
1444     if (fd < 0)
1445         exit (1);
1446     rc=write_BCF (fd, fp, NROW, NCOL, out_type, dp);
1447
1448     G_close_cell (fd);
1449
1450     *etievt++;
1451     *etexdp++;
1452     *etevtr++;
1453     *etsurf++;
1454 }
1455 fclose (fp);
1456 }
1457 /******
1458 /*      end EVT file      */
1459 /******
1460 /******
1461 /*      RIV file      */
1462 /******
1463
1464 int write_RIVER (FILE *fp, int riv, int elv, int cond, int NROW, int NCOL, int out_type_rivh,
1465 int out_type_rivcond, int out_type_rivbott, int dp)
1466 {
1467     int row, col, colcnt;
1468     void *ptr, *raster;
1469     char cell_buf [128];
1470     int IRIVCB;
1471     int *colriver, *rowriver, *headint, *condint, *bottint;
1472     float *headfl, *condfl, *bottfl;

```



```

1541         G_trim_decimal(cell_buf);
1542         printf(" cell_buf= %f\n", cell_buf);
1543         *(headfl+i)=atof(cell_buf);
1544         printf("%f\n",*(headfl+i));*/
1545         i=i+1;
1546     }
1547 }
1548 } /*End of column loop */
1549 }/*end of row loop*/
1550 /******
1551 /* The file with the BOTTOM ELEVATIONS is read */
1552 /******
1553 /* Memory allocation for each type of map */
1554 /******
1555 raster=G_allocate_raster_buf(out_type_rivbott);
1556
1557     if(out_type_rivbott == CELL.TYPE)
1558     {
1559         bottint=(int *)malloc(MXRIVR*sizeof(int));
1560         b=0;
1561     }
1562     else if (out_type_rivbott == FCELL.TYPE)
1563     {
1564         bottfl=(float *)malloc(MXRIVR*sizeof(float));
1565         b=1;
1566     }
1567     else if (out_type_rivbott == DCELL.TYPE)
1568     {
1569         bottfl=(float *)malloc(MXRIVR*sizeof(float));
1570         b=1;
1571     }
1572     i=0; /*This counter is needed to increment the value of the pointer */
1573 for (row = 0; row < NROW; row++)
1574 {
1575     G_percent(row, NROW, 2);
1576     /******
1577     /* The map with river bottom elevations is read */
1578     /******
1579     if (G_get_raster_row(elv, raster, row, out_type_rivbott) < 0)return(row);
1580     colcnt=0;
1581     for (col = 0, ptr = raster; col < NCOL; col++,
1582         ptr = G_incr_void_ptr(ptr, G_raster_size(out_type_rivbott)))
1583     {
1584         if(out_type_rivbott == CELL.TYPE)
1585         {
1586             if(G_is_null_value(ptr, out_type_rivbott)*((CELL *)ptr)=0;
1587             if(*(CELL *)ptr)!=0)
1588             {
1589                 *(bottint+i)*((CELL *)ptr);
1590                 i=i+1;
1591             }
1592         }
1593         else if(out_type_rivbott == FCELL.TYPE)
1594         {
1595             if(G_is_null_value(ptr, out_type_rivbott)*((FCELL *)ptr)=0;
1596             if(*(FCELL *)ptr)!=0)
1597             {
1598                 *(bottfl+i)*((FCELL *)ptr);
1599                 i=i+1;
1600             }
1601         }
1602         else if(out_type_rivbott == DCELL.TYPE)
1603         {
1604             if(G_is_null_value(ptr, out_type_rivbott)*((DCELL *)ptr)=0;
1605             if(*(DCELL *)ptr)!=0)
1606             {
1607                 *(bottfl+i)*((DCELL *)ptr);
1608

```

```

1609             i=i+1;
1610         }
1611     }
1612     } /*end of column loop for second map */
1613 } /*end of row loop for second map */
1614 raster=0;
1615 raster=G_allocate_raster_buf(out_type_rivcond);
1616
1617     if(out_type_rivcond == CELLTYPE)
1618     {
1619         condint=(int *)malloc(MXRIVR*sizeof(int));
1620         c=0;
1621     }
1622     else if (out_type_rivcond == FCELLTYPE)
1623     {
1624         condfl=(float *)malloc(MXRIVR*sizeof(float));
1625         c=1;
1626     }
1627     else if (out_type_rivcond == DCELLTYPE)
1628     {
1629         condfl=(float *)malloc(MXRIVR*sizeof(float));
1630         c=1;
1631     }
1632
1633     i=0; /* The counter has to be reseted to zero */
1634     for (row = 0; row < NROW; row++)
1635     {
1636         G_percent(row, NROW, 2);
1637         /******
1638         /* The map with river conductance values is read */
1639         /******
1640         if (G_get_raster_row(cond, raster, row, out_type_rivcond) < 0) return(row);
1641         colcnt=0;
1642         for (col = 0, ptr = raster; col < NCOL; col++,
1643             ptr = G_incr_void_ptr(ptr, G_raster_size(out_type_rivcond)))
1644         {
1645             if(out_type_rivcond == CELLTYPE)
1646             {
1647                 if(G_is_null_value(ptr, out_type_rivcond)*((CELL *)ptr)=0;
1648                 if(*((CELL *)ptr)!=0)
1649                 {
1650                     *(condint+i)*((CELL *)ptr);
1651                     i=i+1;
1652                 }
1653             }
1654             else if(out_type_rivcond == FCELLTYPE)
1655             {
1656                 if(G_is_null_value(ptr, out_type_rivcond)*((FCELL *)ptr)=0;
1657                 if(*((FCELL *)ptr)!=0)
1658                 {
1659                     *(condfl+i)*((FCELL *)ptr);
1660                     i=i+1;
1661                 }
1662             }
1663             else if(out_type_rivcond == DCELLTYPE)
1664             {
1665                 if(G_is_null_value(ptr, out_type_rivcond)*((DCELL *)ptr)=0;
1666                 if(*((DCELL *)ptr)!=0)
1667                 {
1668                     *(condfl+i)*((DCELL *)ptr);
1669                     i=i+1;
1670                 }
1671             }
1672         }
1673     } /*end of row loop for the third map */
1674
1675     /******
1676     /* The values are written to the RIV.DAT file */

```

```

1677 /* The format depends on the type of maps used */
1678 /*****
1679 for (i=0;i<MXRIVR;i++)
1680 {
1681     if(h==0 && b==0 && c==0) /* FIRST CASE: Stage INT conductance INT elevation INT */
1682     {
1683         fprintf(fp, "%10d_%9d_%9d_%9d_%9d_%9d\n", layer, *(rowriver+i), *(colriver+i), *(headint+i),
1684             *(condint+i), *(bottint+i));
1685     }
1686     else if(h==0 && b==0 && c==1) /* SECOND CASE: Stage INT conductance FLOAT elevation INT */
1687     {
1688         fprintf(fp, "%10d_%9d_%9d_%9d_%9.*f_%9d\n", layer, *(rowriver+i), *(colriver+i), *(headint+i),
1689             dp, *(condfl+i), *(bottint+i));
1690     }
1691     else if(h==0 && b==1 && c==1) /* THIRD CASE: Stage INT conductance FLOAT elevation FLOAT */
1692     {
1693         fprintf(fp, "%10d_%9d_%9d_%9d_%9.*f_%9.*f\n", layer, *(rowriver+i), *(colriver+i),
1694             *(headint+i), dp, *(condfl+i), dp, *(bottfl+i));
1695     }
1696     else if(h==1 && b==1 && c==1) /* FOURTH CASE: Stage FLOAT conductance FLOAT elevation FLOAT */
1697     {
1698         fprintf(fp, "%10d_%9d_%9d_%9.*f_%9.*f_%9.*f\n", layer, *(rowriver+i), *(colriver+i), dp, *(headfl+i),
1699             dp, *(condfl+i), dp, *(bottfl+i));
1700     }
1701     else if(h==0 && b==0 && c==0) /* FIFTH CASE: Stage FLOAT conductance INT elevation FLOAT */
1702     {
1703         fprintf(fp, "%10d_%9d_%9d_%9.*f_%9d_%9.*f\n", layer, *(rowriver+i), *(colriver+i), dp, *(headfl+i),
1704             *(condint+i), dp, *(bottfl+i));
1705     }
1706     else if (h==1 && b==0 && c==0) /* SIXTH CASE: Stage FLOAT conductance INT elevation INT */
1707     {
1708         fprintf(fp, "%10d_%9d_%9d_%9.*f_%9d_%9d\n", layer, *(rowriver+i), *(colriver+i), dp, *(headfl+i),
1709             *(condint+i), *(bottint+i));
1710     }
1711     else if(h==1 && b==0 && c==1) /* SEVENTH CASE: Stage FLOAT conductance FLOAT elevation INT */
1712     {
1713         printf(fp, "%10d_%9d_%9d_%9.*f_%9.*f_%9d\n", layer, *(rowriver+i), *(colriver+i), dp, *(headfl+i),
1714             dp, *(condfl+i), *(bottint+i));
1715     }
1716     else if(h==0 && b==1 && c==0) /* EIGHTH CASE: Stage INT conductance INT elevation FLOAT */
1717     {
1718         fprintf(fp, "%10d_%9d_%9d_%9d_%9d_%9.*f\n", layer, *(rowriver+i), *(colriver+i), *(headint+i),
1719             *(condint+i), dp, *(bottfl+i));
1720     }
1721     else if(h==1 && b==1 && c==0) /*NINTH CASE: Stage FLOAT conductance INT elevation FLOAT */
1722     {
1723         fprintf(fp, "%10d_%9d_%9d_%9.*f_%9d_%9.*f\n", layer, *(rowriver++), *(colriver++), dp,
1724             *(headfl++), *(condint++), dp, *(bottfl++));
1725     }
1726 }
1727 return(0);
1728 }
1729 int read_rivercells(FILE *fp, int riv, int NROW, int NCOL, int out_type_rivh, int IRIVCB)
1730 {
1731     int row, col, colcnt;
1732     void *ptr, *raster;
1733     char cell_buf[300];
1734     extern int MXRIVR;
1735     raster=G_allocate_raster_buf(out_type_rivh);
1736     MXRIVR=0;
1737     IRIVCB=50;
1738     for (row = 0; row < NROW; row++)
1739     {
1740         G_percent(row, NROW, 2);
1741         if (G_get_raster_row(riv, raster, row, out_type_rivh) < 0) return(row);
1742         colcnt=0;
1743         for (col = 0, ptr = raster; col < NCOL; col++,
1744             ptr = G_incr_void_ptr(ptr, G_raster_size(out_type_rivh)))

```

```

1745     {
1746         if(out_type_rivh == CELL.TYPE)
1747         {
1748             if(G_is_null_value(ptr, out_type_rivh))*((CELL *)ptr)=0;
1749             if *((CELL *)ptr)!=0)
1750             {
1751                 MXRIVR=MXRIVR+1;
1752             }
1753         }
1754         else if(out_type_rivh == FCELL.TYPE)
1755         {
1756             if(G_is_null_value(ptr, out_type_rivh))*((FCELL *)ptr)=0;
1757             if *((FCELL *)ptr)!=0)
1758             {
1759                 MXRIVR=MXRIVR+1;
1760             }
1761         }
1762         else if(out_type_rivh == DCELL.TYPE)
1763         {
1764             if(G_is_null_value(ptr, out_type_rivh))*((DCELL *)ptr)=0;
1765             if *((DCELL *)ptr)!=0)
1766             {
1767                 MXRIVR=MXRIVR+1;
1768             }
1769         }
1770     }
1771 }
1772
1773     /******
1774     /* The header of the RIV.DAT file is written */
1775     /******
1776     fprintf(fp, "%10i_%9i\n", MXRIVR, IRIVCB);
1777     fprintf(fp, "%10i\n", MXRIVR);
1778     return(0);
1779 }
1780
1781
1782     RASTER_MAP_TYPE out_type_rivh, out_type_rivcond, out_type_rivbott, map_type_rivh,
1783     map_type_rivcond,
1784     map_type_rivbott;
1785     char *rivheads;
1786     char *rivcond;
1787     char *rivbot;
1788     int riv, cond, elv;
1789     int IRIVCB;
1790
1791     if(parm.river->answer)
1792     {
1793
1794         printf("\n~~~~~Processing RIV file ~~~~~\n");
1795         rivheads = parm.river->answer;
1796         rivcond = parm.rivcond->answer;
1797         if (rivheads==NULL)
1798             G_fatal_error("River conductance values are required");
1799         rivbot = parm.rivelev ->answer;
1800         if (rivheads==NULL)
1801             G_fatal_error("River stage values are required");
1802         mapset = G_find_cell2 (rivheads, "");
1803         map_type_rivh = G_raster_map_type(rivheads, mapset);
1804         if (mapset == NULL)
1805             {
1806                 G_fatal_error (" Cell file [%s] not found\n", rivheads);
1807             }
1808         riv = G_open_cell_old (rivheads, mapset);
1809         if (riv < 0)
1810             exit(1);
1811         mapset = G_find_cell2 (rivcond, "");
1812         map_type_rivcond = G_raster_map_type(rivcond, mapset);

```



```

1813     if (mapset == NULL)
1814     {
1815         G_fatal_error ("Cell_file_%s_not_found\n", rivcond);
1816     }
1817     cond = G_open_cell_old (rivcond, mapset);
1818     if (cond < 0)
1819         exit(1);
1820     mapset = G_find_cell2 (rivbot, "");
1821     map_type_rivbott= G_raster_map_type(rivbot, mapset);
1822     if (mapset == NULL)
1823     {
1824         G_fatal_error ("Cell_file_%s_not_found\n", rivbot);
1825     }
1826     elv= G_open_cell_old (rivbot, mapset);
1827     if (elv < 0)
1828         exit(1);
1829     out_type_rivh = map_type_rivh;
1830     out_type_rivcond = map_type_rivcond;
1831     out_type_rivbott=map_type_rivbott;
1832     if(out_type_rivcond==DCELL.TYPE|| out_type_rivh==DCELL.TYPE|| out_type_rivbott
1833         ==DCELL.TYPE)dp=6;
1834
1835     fp = fopen("model.riv", "w"); /* *fp */
1836     /* The number of cells containing rivers has to be computed first */
1837     /* The map containing the heads will be used to this end (name)*/
1838
1839     rc=read_rivercells(fp, riv, NROW,NCOL, out_type_rivh, IRIVCB);
1840
1841     if(rc)
1842     {
1843         G_fatal_error("Read_failed_at_row_%d\n", rc);
1844     }
1845     rc=write_RIVER(fp, riv, elv, cond, NROW,NCOL, out_type_rivh, out_type_rivcond, out_type_rivbott, dp);
1846     if(rc)
1847     {
1848         G_fatal_error("Read_failed_at_row_%d\n", rc);
1849     }
1850
1851     /* The three raster maps and the ASCII file are closed */
1852     G_close_cell(riv);
1853     G_close_cell(elv);
1854     G_close_cell(cond);
1855     fclose(fp); /*Change river to fp */
1856 }
1857 /******
1858 /* WEL file */
1859 /******
1860 /* archivo de pozos well*/
1861 int NP, size, l;
1862 char usewell[100];
1863 char *usew2, *palabra;
1864 int val[1000], val_l[1000], val_cond[1000];
1865 int ITMP, count, c, rows_extr, rows_lay;
1866
1867
1868     if (parm.wel->answer){
1869
1870     IWELCB=50;
1871     NP=0;
1872
1873         printf("\n.....Processing_WELL_file.....\n");
1874
1875         fp=fopen("model.wel", "w");
1876
1877     usew2=parm.use->answer;
1878     size=strlen(usew2);
1879     char usew[size];
1880     strcpy(usew, usew2);

```

```

1881
1882 palabra=strtok(usew, ", ");
1883
1884     for (k=1;k<=NPER;k++)
1885     {
1886
1887         Points = Vect_new_line_struct ();
1888         Cats = Vect_new_cats_struct ();
1889
1890         if ((mapset = G_find_vector2 (parm.wel->answer, "")) == NULL)
1891             G_fatal_error (_("Vector_map<%s>_not_found"), parm.wel->answer);
1892
1893         if (Vect_set_open_level (2))
1894             G_fatal_error (_("Unable_to_set_predetermined_vector_open_level"));
1895
1896         name=parm.wel->answer;
1897
1898         mapset = G_find_vector2 (parm.wel->answer, "");
1899         vect = Vect_open_old (&In, parm.wel->answer, mapset);
1900
1901         type = Vect_read_next_line (&parm.wel, Points, Cats);
1902         name = Vect_get_name (&In);
1903
1904         Fi = Vect_get_field (&In, 1);
1905             if (!Fi) {
1906                 Vect_close (&In);
1907                 G_fatal_error (_("Database_connection_not_defined_for_layer_%d"), 1);
1908             }
1909         G_debug (1, "Field_number:%d;_Name:<%s>;_Driver:<%s>;_Database:<%s>;_Table:<%s>;_Key:<%s>;\n",
1910             Fi->number, Fi->name, Fi->driver, Fi->database, Fi->table, Fi->key);
1911         db_init_string (&table_name);
1912         db_init_handle (&handle);
1913         db_init_string (&value_string);
1914
1915
1916         G_get_window (&region);
1917         G_format_northing (region.north, regnorth, region.proj);
1918         G_format_easting (region.west, regwest, region.proj);
1919
1920             G_trim_decimal (regwest);
1921             G_trim_decimal (regnorth);
1922             G_trim_decimal (ewres);
1923             G_trim_decimal (nsres);
1924
1925             pnorth = strtod (regnorth, NULL);
1926             pwest = strtod (regwest, NULL);
1927
1928         MXACTIW=Vect_cidx_get_num_cats_by_index (&In, 0);
1929
1930     if (k==1){
1931         fprintf (fp, "%10i - %9i\n", MXACTIW, IWELCB);
1932     }
1933
1934     c=0;
1935     while ((type = Vect_read_next_line (&In, Points, Cats)) > 0) {
1936
1937
1938         coy = *Points->y;
1939         cox = *Points->x;
1940         ycell=(int)abs((pnorth-coy)/nsr)+1;
1941         xcell=(int)abs((pwest-cox)/ewr)+1;
1942
1943         for (i=0; i < Cats->n_cats; i++){
1944
1945             Fi= Vect_get_field (&In, Cats->field [i]);
1946             driver = db_start_driver_open_database ( Fi->driver, Fi->database );
1947
1948             if (c==0){

```

```

1949     sprintf(buf,"SELECT_*_FROM_%_WHERE_%=1\n",Fi->table ,palabra );
1950     db_init_string(&stmt);
1951     db_append_string(&stmt ,buf);
1952     if(db_open_select_cursor(driver , &stmt , &cursor , DB_SEQUENTIAL) != DB_OK)
1953         G_fatal_error ( "Cannot_open_select_cursor:_%s'", db_get_string(&stmt));
1954
1955         table=db_get_cursor_table(&cursor);
1956         column=db_get_table_column (table ,0);
1957         ctype= db_sqltype_to_Ctype(db_get_column_sqltype (column));
1958         ITMP=db_get_num_rows(&cursor);
1959
1960         fprintf(fp , " %10i %10i _ _ Wells_used_in_stress_period_%a\n" ,ITMP,NP,k);
1961     }
1962     sprintf(buf,"SELECT_%_FROM_%_WHERE_%=%d_AND_%=1\n" ,parm.extr->answer ,
1963     Fi->table ,Fi->key ,Cats->cat [i] ,palabra );
1964
1965     db_init_string(&stmt);
1966     db_append_string(&stmt ,buf);
1967
1968     if(db_open_select_cursor(driver , &stmt , &cursor , DB_SEQUENTIAL) != DB_OK)
1969         G_fatal_error ( "Cannot_open_select_cursor:_%s'", db_get_string(&stmt));
1970     table=db_get_cursor_table(&cursor);
1971     column=db_get_table_column (table ,0);
1972     ctype= db_sqltype_to_Ctype(db_get_column_sqltype (column));
1973     value=db_get_column_value (column);
1974     rows_extr=db_get_num_rows(&cursor);
1975
1976
1977     count=0;
1978     while(1){
1979         if(db_fetch(&cursor , DB_NEXT, &more) != DB_OK)
1980             G_fatal_error ( "Unable_to_fetch_data_from_table<_%s>" ,Fi->table );
1981         if (!more) break;
1982
1983         if( count == 0 )
1984             val[c] = db_get_value_as_double (value ,ctype);
1985         count ++;
1986     }
1987     sprintf(buf,"SELECT_%_FROM_%_WHERE_%=%d_AND_%=1\n" ,parm.layer->answer ,
1988     Fi->table ,Fi->key ,Cats->cat [i] ,palabra );
1989
1990     db_init_string(&stmt);
1991     db_append_string(&stmt ,buf);
1992
1993     if(db_open_select_cursor(driver , &stmt , &cursor , DB_SEQUENTIAL) != DB_OK)
1994         G_fatal_error ( "Cannot_open_select_cursor:_%s'", db_get_string(&stmt));
1995     table=db_get_cursor_table(&cursor);
1996     column=db_get_table_column (table ,0);
1997     ctype= db_sqltype_to_Ctype(db_get_column_sqltype (column));
1998     rows_lay=db_get_num_rows(&cursor);
1999
2000     value=db_get_column_value (column);
2001
2002     count=0;
2003     while(1){
2004         if(db_fetch(&cursor , DB_NEXT, &more) != DB_OK)
2005             G_fatal_error ( "Unable_to_fetch_data_from_table<_%s>" ,
2006             Fi->table );
2007         if (!more) break;
2008
2009         if( count == 0 )
2010             val_l[c] = db_get_value_as_double (value ,ctype);
2011         count ++;
2012     }
2013     db_close_cursor(&cursor);
2014     db_free_string(&stmt);
2015     if(ITMP!=0 & rows_extr!=0 & rows_lay!=0){
2016         fprintf(fp , " %10i _ %9i _ %9i _ %9d\n" , val_l[c] , ycell , xcell , val[c]);

```

```

2017 //      printf("c= %a\n",c);
2018     }
2019     c++;
2020     }
2021     db_close_database(driver);
2022     db_shutdown_driver(driver);
2023 }
2024
2025 palabra = strtok(NULL, ",");
2026 }
2027     Vect_close(&In);
2028
2029     fclose(fp);
2030 }
2031     /******
2032     /* DRAIN file */
2033     /******
2034     if (parm.drn->answer){
2035
2036         IWELCB=50;
2037         NP=0;
2038
2039         printf("\n-----Processing DRAIN file -----
2040\n");
2041         fp=fopen("model.drn","w");
2042         usew2=parm.usedr->answer;
2043         size=strlen(usew2);
2044         char usew1[size];
2045         strcpy(usew1,usew2);
2046
2047         palabra=strtok(usew1,",");
2048
2049         for (k=1;k<=NPER;k++)
2050             {
2051
2052             Points = Vect_new_line_struct();
2053             Cats = Vect_new_cats_struct();
2054
2055             if ((mapset = G_find_vector2(parm.drn->answer, "")) == NULL)
2056                 G_fatal_error(_("Vector_map<%s>_not_found"), parm.wel->answer);
2057
2058             if (Vect_set_open_level(2))
2059                 G_fatal_error(_("Unable_to_set_predetermined_vector_open_level"));
2060
2061             name=parm.drn->answer;
2062
2063             mapset = G_find_vector2(parm.drn->answer, "");
2064             vect = Vect_open_old(&In,parm.drn->answer, mapset);
2065
2066             type = Vect_read_next_line(&parm.wel, Points, Cats);
2067             name = Vect_get_name(&In);
2068
2069             Fi = Vect_get_field(&In, 1);
2070                 if (!Fi) {
2071                     Vect_close(&In);
2072                     G_fatal_error(_("Database_connection_not_defined_for_layer_%d"), 1);
2073                 }
2074                 G_debug(1,"Field_number:%d;_Name:<%s>;_Driver:<%s>;_Database:<%s>;
2075 _Table:<%s>;_Key:<%s>;\n", Fi->number, Fi->name, Fi->driver, Fi->database, Fi->table, Fi->key);
2076                 db_init_string(&table_name);
2077                 db_init_handle(&handle);
2078                 db_init_string(&value_string);
2079                 G_get_window(&region);
2080                 G_format_northing(region.north, regnorth, region.proj);
2081                 G_format_easting(region.west, regwest, region.proj);
2082
2083                 G_trim_decimal(regwest);
2084                 G_trim_decimal(regnorth);

```

```

2085         G_trim_decimal(ewres);
2086         G_trim_decimal(nsres);
2087
2088         pnorth =strtof(regnorth, NULL);
2089         pwest =strtof(regwest, NULL);
2090
2091         MXACTW=Vect_cidx_get_num_cats_by_index(&In,0);
2092
2093         //printf(" number of cats: %d \n",MXACTW);
2094         if(k==1){
2095             fprintf(fp, "%10i_%9i\n",MXACTW,IWELCB);
2096         }
2097         c=0;
2098         d=0;
2099         while((type = Vect_read_next_line(&In, Points, Cats)) > 0) {
2100
2101             if (type == GV_LINE || type == GV_POINT || type == GV_CENTROID) {
2102                 if (Vect_cat_get(Cats, Fi->number, &cat) == 0) {
2103                     Vect_cat_set(Cats,Fi->number, i);
2104                     i++;
2105                 }
2106             }
2107             coy = *Points->y;
2108             cox =*Points->x;
2109             ycell=(int)abs((pnorth-coy)/nsr)+1;
2110             xcell=(int)abs((pwest-cox)/ewr)+1;
2111
2112             for(i=0;i < Cats->n_cats; i++){
2113
2114                 Fi= Vect_get_field(&In,Cats->field[i]);
2115                 driver = db_start_driver_open_database ( Fi->driver, Fi->database );
2116
2117                 if(c==0){
2118                     sprintf(buf,"SELECT_*_*_FROM_%s_WHERE_%s=1\n",Fi->table, palabra);
2119                     db_init_string(&stmt);
2120                     db_append_string(&stmt, buf);
2121                     if(db_open_select_cursor(driver, &stmt, &cursor, DB_SEQUENTIAL) != DB_OK)
2122                         G_fatal_error ("Cannot_open_select_cursor:_%s", db_get_string(&stmt));
2123
2124                     table=db_get_cursor_table(&cursor);
2125                     column=db_get_table_column(table,0);
2126                     ctype= db_sqltype_to_Ctype(db_get_column_sqltype(column));
2127                     ITMP=db_get_num_rows(&cursor);
2128
2129                     fprintf(fp, "%10i %10i ___Drains_used_in_stress_period_%d_\n",ITMP,NP,k);
2130                 }
2131
2132                 sprintf(buf,"SELECT_%s_FROM_%s_WHERE_%s=%d_AND_%s=1\n",parm.elevdrn->answer,
2133 Fi->table,Fi->key,Cats->cat[i], palabra);
2134
2135                 db_init_string(&stmt);
2136                 db_append_string(&stmt, buf);
2137
2138                 if(db_open_select_cursor(driver, &stmt, &cursor, DB_SEQUENTIAL) != DB_OK)
2139                     G_fatal_error ("Cannot_open_select_cursor:_%s", db_get_string(&stmt));
2140                 table=db_get_cursor_table(&cursor);
2141                 column=db_get_table_column(table,0);
2142                 ctype= db_sqltype_to_Ctype(db_get_column_sqltype(column));
2143                 value=db_get_column_value(column);
2144
2145                 count=0;
2146                 while(1){
2147                     if(db_fetch(&cursor, DB_NEXT, &more) != DB_OK)
2148                         G_fatal_error ("Unable_to_fetch_data_from_table< %s>"),Fi->table);
2149                     if (!more) break;
2150
2151                     if( count == 0 )
2152                         val[c] = db_get_value_as_double(value, ctype);

```

```

2153         count ++;
2154     }
2155
2156     sprintf(buf, "SELECT_%s_FROM_%s_WHERE_%s=%d_AND_%s=1\n", parm.condrn->answer, Fi->table,
2157 Fi->key, Cats->cat[i], palabra);
2158
2159         db_init_string(&stmt);
2160         db_append_string(&stmt, buf);
2161
2162         if(db_open_select_cursor(driver, &stmt, &cursor, DB_SEQUENTIAL) != DB_OK)
2163             G_fatal_error ("Cannot_open_select_cursor:_'%s'", db_get_string(&stmt));
2164
2165         table=db_get_cursor_table(&cursor);
2166         column=db_get_table_column(table, 0);
2167         ctype= db_sqltype_to_Ctype(db_get_column_sqltype (column));
2168         value=db_get_column_value(column);
2169
2170         count=0;
2171         while(1){
2172     if(db_fetch(&cursor, DB_NEXT, &more) != DB_OK)
2173         G_fatal_error (_("Unable_to_fetch_data_from_table<%s>"), Fi->table);
2174     if (!more) break;
2175
2176     if( count == 0 )
2177         val_cond[d] = db_get_value_as_double(value, ctype);
2178     count ++;
2179 }
2180     sprintf(buf, "SELECT_%s_FROM_%s_WHERE_%s=%d_AND_%s=1\n", parm.laydr->answer, Fi->table,
2181 Fi->key, Cats->cat[i], palabra);
2182
2183         db_init_string(&stmt);
2184         db_append_string(&stmt, buf);
2185
2186         if(db_open_select_cursor(driver, &stmt, &cursor, DB_SEQUENTIAL) != DB_OK)
2187             G_fatal_error ("Cannot_open_select_cursor:_'%s'", db_get_string(&stmt));
2188         table=db_get_cursor_table(&cursor);
2189         column=db_get_table_column(table, 0);
2190         ctype= db_sqltype_to_Ctype(db_get_column_sqltype (column));
2191
2192         value=db_get_column_value(column);
2193
2194         count=0;
2195         while(1){
2196     if(db_fetch(&cursor, DB_NEXT, &more) != DB_OK)
2197         G_fatal_error (_("Unable_to_fetch_data_from_table<%s>"),
2198 Fi->table);
2199     if (!more) break;
2200
2201     if( count == 0 )
2202         val_l[d] = db_get_value_int(value);
2203     count ++;
2204 }
2205
2206         db_close_cursor(&cursor);
2207         db_free_string(&stmt);
2208
2209     if(ITMP!=0){
2210     fprintf(fp, "%10i_%9i_%9i_%9d_%9d\n", val_l[d], ycell, xcell, val[c], val_cond[c]);
2211     }
2212     }
2213     c++;
2214     d++;
2215     }
2216 }
2217
2218 palabra = strtok(NULL, ",");
2219 }
2220

```

```

2221     Vect_close(&In);
2222
2223         fclose(fp);
2224     }
2225
2226     /******
2227     /* NAM file is written now */
2228     /******
2229
2230     fp=fopen("model.nam","w");
2231     fprintf(fp,"LIST_____7_____model.lst_\n");
2232     fprintf(fp,"BAS6_____1_____model.ba6_\n");
2233     fprintf(fp,"DIS_____10_____model.dis_\n");
2234
2235     if(PACK==0){
2236     fprintf(fp,"BCF6_____11_____model.bc6_\n");
2237     }
2238
2239     if(PACK==1){
2240     fprintf(fp,"LPF_____20_____model.lpf_\n");
2241     }
2242
2243     if(parm.wel->answer)
2244     {
2245         fprintf(fp,"WEL_____12_____model.wel_\n");
2246     }
2247     if(parm.drn->answer)
2248     {
2249         fprintf(fp,"DRN_____13_____model.drn_\n");
2250     }
2251     if(parm.river->answer)
2252     {
2253         fprintf(fp,"RIV_____14_____model.riv_\n");
2254     }
2255     if(parm.rch->answer)
2256     {
2257         fprintf(fp,"RCH_____18_____model.rch_\n");
2258     }
2259
2260         if(parm.surf->answer)
2261         {
2262             fprintf(fp,"EVT_____15_____model.evt_\n");
2263         }
2264
2265     fprintf(fp,"OC_____22_____model.oc_\n");
2266     fprintf(fp,"PCG_____23_____model.pcg_\n");
2267     fprintf(fp,"DATA(BINARY)___50___BUDGET.DAT\n");
2268     fprintf(fp,"DATA(BINARY)___51___HEADS.DAT_\n");
2269     fprintf(fp,"DATA(BINARY)___52___DDOWN.DAT_\n");
2270
2271     fclose(fp);
2272
2273     /******
2274     /* MODFLOW and import utilities are called */
2275     /******
2276     int importMODFLOW(FILE *fa,int NLAY, int NPER, int NSTP[], int NCOL, int NROW, char *aux)
2277     {
2278         void *rast;
2279         void *rast_ptr; /*These variables are used by GRASS*/
2280         int cf; /* used to create the raster map to be written */
2281         int i,j,k;
2282         int row,col;
2283         char *temp;
2284         FILE *ft;
2285         char output[128];
2286         char mapinput[128];
2287         char program[128];
2288         float head;

```



```

2357         fseek(ft,0,SEEK_CUR);
2358     }
2359     fclose(ft);
2360     unlink(temp);
2361     G_close_cell(cf);
2362     rast_ptr=0;
2363     rast=rast_ptr;
2364     /******
2365     /***  Values < -900 are assigned NULL values  ***/
2366     /******
2367     pid = fork();
2368     sprintf(program,"r.mapcalc");
2369     if(pid==0)
2370     {
2371         sprintf(mapinput,"%s=if(%s<-900,null(),%s)",output,output,output);
2372         execlp(program,program,mapinput,NULL);
2373     }
2374     pid = wait();
2375
2376     /******
2377     /***  end of new code  ***/
2378     }
2379     }
2380     }
2381     }
2382
2383 pid = fork();
2384 if(pid<0)
2385 {
2386     printf("ERROR: _Not_able_to_execute_MODFLOW\n");
2387 }
2388 if(pid==0)
2389 {
2390     printf("*****\n");
2391     printf("_Running_MODFLOW_\n");
2392     printf("*****\n");
2393     execlp("mf2005grass",0);
2394 }
2395 pid = wait();
2396
2397 pid = fork();
2398 if(pid<0)
2399 {
2400     printf("ERROR: _Not_able_to_open_import_module\n");
2401 }
2402 if(pid==0)
2403 {
2404     printf("Importing_HEADS\n");
2405     execlp("./rheads",0);
2406 }
2407
2408 pid=wait();
2409 /******
2410 /*  DDOWN values are imported  */
2411 /******
2412 pid = fork();
2413 if(pid<0)
2414 {
2415     printf("ERROR: _Not_able_to_open_import_module\n");
2416 }
2417 if(pid==0)
2418 {
2419     printf("Importing_DRAWDOWN\n");
2420     execlp("./rddown",0);
2421 }
2422 pid=wait();
2423 if(parm.drawdown->answer)
2424 {

```

```

2425     aux=parm.drawdown->answer;
2426 }
2427 else
2428 {
2429     G_fatal_error("Name_for_simulated_drawdown_maps_is_needed");
2430 }
2431 fa=fopen("DDOWN.ASC","r"); /* The ascii file is opened */
2432 if (fa==NULL)
2433 {
2434     exit(-1);
2435 }
2436
2437 importMODFLOW( fa ,NLAY,NPER,NSTP,NCOL,NROW, aux);
2438 fclose( fa );
2439
2440 printf("*****\n");
2441 printf("Drawdown_values_imported_successfully\n");
2442 printf("*****\n");
2443
2444 if (parm.headsin->answer)
2445 {
2446     aux=parm.headsin->answer;
2447 }
2448 else
2449 {
2450     G_fatal_error("Name_for_simulated_head_maps_is_needed");
2451 }
2452
2453 fa=fopen("HEADS.ASC","r");
2454 if (fa==NULL)
2455 {
2456     printf("File_HEADS.ASC_not_found\n");
2457     exit(-1);
2458 }
2459
2460 importMODFLOW( fa ,NLAY,NPER,NSTP,NCOL,NROW, aux);
2461 fclose( fa );
2462
2463 printf("\n");
2464 printf("*****\n");
2465 printf("Head_values_imported_successfully\n");
2466 printf("*****\n");
2467 printf("\n");
2468 /******
2469 /* A color table is created now using MAX & MIN values */
2470 /******
2471 int colortable(int NLAY, int NPER, int NSTP[], char *aux)
2472 {
2473     DCELL minv; /*There's something wrong here */
2474     DCELL maxv;
2475     char output[128];
2476     int i,j,k;
2477     char *mapset;
2478     struct FPRange range;
2479     struct Colors colors;
2480     DCELL min;
2481     DCELL max;
2482
2483     for (j=1;j<=NLAY;j++)
2484     {
2485         for (k=1;k<=NPER;k++)
2486         {
2487             for (i=1;i<=NSTP[k];i++)
2488             {
2489                 if (NSTP[k]<10)
2490                 {
2491                     sprintf(output,"%s.lay %a .stp %a .tst %a",aux,j,k,i);
2492                 }

```

```

2493     if(NSTP[k]>=10 && NSTP[k]<100)
2494     {
2495         if(i<=9)
2496         {
2497             sprintf(output,"%.lay %ã .stp %ã .tst0 %ã",aux,j,k,i);
2498         }
2499         else
2500         {
2501             sprintf(output,"%.lay %ã .stp %ã .tst %ã",aux,j,k,i);
2502         }
2503     }
2504     if(NSTP[k]>=100 && NSTP[k]<1000)
2505     {
2506         if(i<=9)
2507         {
2508             sprintf(output,"%.lay %ã .stp %ã .tst00 %ã",aux,j,k,i);
2509         }
2510         if(i>9 && i<=99)
2511         {
2512             sprintf(output,"%.lay %ã .stp %ã .tst0 %ã",aux,j,k,i);
2513         }
2514         else
2515         {
2516             sprintf(output,"%.lay %ã .stp %ã .tst %ã",aux,j,k,i);
2517         }
2518     }
2519     mapset=G_find_cell2(output,"");
2520     if (G_read_fp_range (output, mapset, &range) < 0)
2521         G_fatal_error ("could_not_read_range_file");
2522     /* I need to reclassify the values <-900 to NULL values */
2523     G_get_fp_range_min_max(&range,&min,&max);
2524     G_init_colors (&colors);
2525     /* It's not working for head maps */
2526     G_make_rainbow_fp_colors(&colors ,min,max);
2527     G_write_colors (output ,mapset,&colors);
2528     }
2529     }
2530     }
2531     }
2532     aux=parm.drawdown->answer;
2533     colortable(NLAY,NPER,NSTP,aux);
2534
2535     aux=parm.headsin->answer;
2536     colortable(NLAY,NPER,NSTP,aux);
2537
2538     /* Values <=-999 should be reclassified as NULL data */
2539     /* The ASCII files are deleted */
2540
2541     remove("HEADS.ASC");
2542     remove("DDOWN.ASC");
2543     remove("BUDGET.DAT");
2544     remove("HEADS.DAT");
2545     remove("DDOWN.DAT");
2546     exit(0);
2547 }

```

Apéndice B

Instalación de MODFLOW para el módulo *r.gws*

Para instalar MODFLOW en linux se tiene que descargar el paquete de la pagina de internet

`http://water.usgs.gov/nrp/gwsoftware/modflow2005/modflow2005.html`

donde se encontrará el archivo con terminación *tar.Z*. Al descomprimirlo, en el archivo de texto Makefile, se tiene que modificar de tal manera que las líneas 10 y 17 queden como sigue:

```
F90= gfortran
CC= gcc
```

Para que MODFLOW pueda trabajar con el módulo de GRASS, es necesario cambiar, dentro del archivo *mf2005.f* la línea 54, tal que quede:

```
FNAME='model.nam'
```

Una vez realizado ésto, se corre el comando *make* en la terminal y se coloca el archivo binario *mf2005* dentro de la carpeta de ejecutables del sistema. Para que el módulo funcione correctamente, es necesario compilar los programas *rddown.f* y *rheads.f*.

B.1. *rddown.f* y *rheads.f*

Estos programas son creados para la interpretación de la información de salida de MODFLOW. El programa *rddown.f* es el siguiente.

```

      DIMENSION A(300000)
c      DIMENSION HEADNG(32)
      LENA=300000
C
      OPEN(1, file='model.dis')
c      READ(1,20) HEADNG
20  FORMAT(20A4)
      READ(1,30) NLAY,NROW,NCOL,NPER,ITMUNI,LENUNI
30  FORMAT(8I10)
      CLOSE(1)
      NRC=NROW*NCOL
      NRCL=NROW*NCOL*NLAY
      LCNRC=1
      LCNRCL=NRC+1
      NTOTAL=NRC+NRCL
      IF (NTOTAL.GT.LENA) THEN
          WRITE(*,*) 'LENGTH_OF_MASTER_ARRAY_LENA_IN_THE_PROGRAM'
          WRITE(*,*) 'BIN2ASC.FOR_IS_TOO_SMALL'
          STOP
      ENDIF

      CALL CONVERT(NLAY,NROW,NCOL,A(LCNRC),A(LCNRCL))
      stop
      end
C
C
      SUBROUTINE CONVERT(NLAY,NROW,NCOL,HEAD,BUDG)
      DIMENSION HEAD(NCOL,NROW),BUDG(NCOL,NROW,NLAY)
      CHARACTER*16 TEXT
C
C
      SET ALL VALUES TO ZERO
      DO I=1,NCOL
          DO J=1,NROW
              HEAD(I,J)=0
              DO K=1,NLAY
                  BUDG(I,J,K)=0
              END DO
          END DO
      END DO
C
      OPEN(2, file='DDOWN.DAT',FORM='UNFORMATTED')
      OPEN(3, file='DDOWN.ASC')
33  DO NL=1,NLAY
c      READ(2) KSTP,KPER,PERTIM,TOTIM,TEXT,NC,NR,K
C          WRITE(3,*) KSTP,KPER,PERTIM,TOTIM,TEXT,NC,NR,K
          READ(2) HEAD
          DO J=1,NR
              write(3,50) (HEAD(I,J),I=1,NC)
          END DO
      END DO
      GO TO 33
50  FORMAT(10000F9.2)
51  CONTINUE
      CLOSE(2)
      CLOSE(3)

      RETURN
      END

```

El programa *rheads.f*, es el mismo, sólo se tiene que substituir *DDOWN.DAT* y *DDOWN.ASC* por *RHEADS.DAT* y *RHEADS.ASC*.

Bibliografía

- AHAMED, I. y UMAR, R.: «Groundwater flow modelling of Yamuna Krishna interstream, a part of central Ganga Plain Uttar Pradesh». *J. Earth Syst. Sci.*, 2009, **118**, pp. 507–523.
- AL-FATLAWI, A. N.: «The application of the mathematical model (MODFLOW) to simulate the behavior of groundwater flow in Umm Er Radhuma unconfined aquifer». *Euphrates Journal of Agriculture Science*, 2011, **3**, pp. 1–16.
- AL-HASSOUN, S. A. y MOHAMMAD, T. A.: «Prediction of Water Table in an Alluvial Aquifer Using Modflow». *Pertanika J. Sci. & Technol.*, 2011, **19**, pp. 45–55.
- BANDANI, E. y MOGHADAM, M. A.: «Application of Groundwater Mathematical Model for Assessing the Effects of Galoogah Dam on the Shooro Aquifer- Iran». *European Journal of Scientific Research*, 2011, **54**, pp. 449–511.
- CALVACHE-QUESADA, M. L. y PULIDO-BOSCH, A.: «Simulación matemática del flujo subterráneo en el acuífero del Río Verde (Almuñecar, Granada)». *Estudios geol.*, 1990, **46**, pp. 301–316.
- CARDONA-BENAVIDES, A.; MARTÍNEZ-HERNÁNDEZ, J. E.; CASTRO-LARRAGOITIA, J. y ALCALDE-ALDERETE, R.: «La edad del agua subterránea que abastece la región de San Luis Potosí». *Universitarios Potosinos*, 2006, **7**, pp. 20–25.
- CARRERA-HERNÁNDEZ, S. J., J. J. AND GASKIN: «The groundwater modeling tool for GRASS (GMTG): Open source groundwater flow modeling». *Computers and Geosciences*, 2006, **32**, pp. 339–351.
- CHIANG, WEN-HSING: *3D-Groundwater Modeling with PMWIN*. Springer, second edition edición, 2005.
- CRISTÓBAL, J.; NINYEROLA, M. y PONS, X.: «Modeling air temperature through a combination of remote sensing and GIS data». *Journal of Geophysical Research*, 2008, **113**, pp. 1–13.
- CRONER, C. M.; SPERLING, J. y BROOME, F. R.: «Geographic Information Systems (GIS): New perspectives in understanding human health and environmental relationships». *Statistics in Medicine*, 1998, **15**, pp. 1961–1977.

- CRUCES-DE ABIA, J.: *Contaminación ambiental, modelo MODFLOW*, 2006/2007.
- DE-ROO, A. P. J.: «Modelling runoff and sediment transport in catchments using GIS». *Hydrological Processes*, 1998, **12**, pp. 905–922.
- EL-KENAWY, A.; LOPEZ-MORENO, J.; VICENTE-SERRANO, S. M. y MORSI, F.: «Climatological modeling of monthly air temperature and precipitation in Egypt through GIS techniques». *Climate Research*, 2010, **42**, pp. 161–176.
- FLORES-MÁRQUEZ, E. L.; LEDESMA, I. K. y ARANGO-GALVÁN, C.: «Sustainable geohydrological model of San Luis Potosí aquifer, Mexico». *Geofísica Internacional*, 2011.
- FLUGEL, W. A. y MICHL, C.: «Using MODFLOW/MODPATH combined with GIS analysis for groundwater modelling in the alluvial aquifer of the River Sieg, Germany». *Models for Assessing and Monitoring Groundwater Qu*, 1995, **227**, pp. 117–123.
- GALLEGOS, J. J.: *Modeling Groundwater Flow in Karst Aquifers: An Evaluation of MODFLOW-CFP at the Laboratory and Sub-Regional Scales*. Tesina o Proyecto, Florida State University, 2011.
- GARCÍA-ARÓSTEGUI, J. L.; HEREDIA, J.; MURILLO, J. M.; RUBIO-CAMPOS, J. C.; GONZÁLEZ-RÁMON, A. y LÓPEZ-GETA, J. A.: «Contribución desde la modelización del flujo subterráneo al conocimiento del acuífero del Río Verde (Granada)», 2001.
- GUTIÉRREZ-ENRÍQUEZ, M. M. y ARISTIZABAL-RODRÍGUEZ, H. F.: «Modelación del flujo subterráneo en el sector comprendido entre el Piedemonte de la Cordillera Central, los ríos Cauca y Tulua y la Laguna de Sonso». *Informe técnico*, Corporación autónoma regional del valle de Cauca, 2006.
- HARBAUGH, A. W.: *MODFLOW-2005, The U.S. Geological Survey Modular Ground-Water Model of the Ground-Water Flow Process*, 2005.
- HERGT, T.: *Diseño optimizado de redes de monitoreo de la calidad del agua de los sistemas de flujo subterráneo en el acuífero 2411 San Luis Potosí: hacia un manejo sustentable*. Tesis doctoral, Universidad Autónoma de San Luis Potosí, 2009.
- JACKSON, J. M.: *Hidrogeology and groundwater flow model, central catchment of Bribie Island, southeast Queensland*. Tesina o Proyecto, School of natural resource sciences, Queensland University of Technology, 2007.
- LÓPEZ-ÁLVAREZ, B.: *Cambios de uso de suelo y su impacto en el sistema acuífero del valle de San Luis Potosí, aplicando modelación numérica*. Tesis doctoral, Instituto Potosino de Investigación Científica y Tecnológica, 2012.
- MARTÍNEZ-ALFARO, P.E; MARTINEZ-SANTOS, P. y CASTAÑO CASTAÑO, S.: *Fundamentos de hidrogeología*. Mundi-Prensa, 2006.

- MATTHEW, N. y STONES, R.: *Beginnning Databases with PostgreSQL*, 2005.
- MCDONALD, M. G. y HARBAUGH, A. W.: *Techniques of water-Resources Investigations of the United States Geological Survey, Chapter A1*. En: McDonald y Harbaugh (1988), 1988.
- NETELER, M. y MITASOVA, H.: *Open source GIS : A GRASS GIS Approach*. Springer, third edition edicin, 2007.
- NOYOLA-MEDRANO, M. N.; RAMOS-LEAL, J. A.; DOMÍNGUEZ-MARIANI, E.; PINEDA-MARTÍNEZ, L. F.; LÓPEZ-LOERA, H. y CARBAJAL, N.: «Factores que dan origen al minado de acuíferos en ambientes áridos: caso Valle de San Luis Potosí». *Revista Mexicana de Ciencias Geológicas*, 2009, **26**, pp. 395–410.
- ORZOL, L. L.: *User's guide for MODTOOLS: Computer programs for translating data of MODFLOW and MODPATH into geographic information system files*. U.S. Geological Survey (Portland, Or. and Denver, CO), 1997.
- PANAGOPOULOS, G.: «Application of MODFLOW for simulating groundwater flow in the Trifilia karst aquifer, Greece». *Environ Earth Sci*, 2012.
- SHRIBRU-WAKE, J.: *Groundwater Surface Water Interaction Modelling Using Visual MODFLOW and GIS*. Tesina o Proyecto, Universiteit Gent Vrije Universiteit Brussel Belgium, 2008.
- USTRNUL, Z. y CZEKIERDA, D.: «Application of GIS for the development of climatological air temperature maps: an example from Poland». *Meteorological Applications*, 2005, **12**, pp. 43–50.
- VILLÓN-BÉJAR, M.: *Drenaje*. Cartago, 2006.
- WINSTON, R. B.: *Graphical User Interface for MODFLOW, Version 4*. U.S. Geological Survey, 2000.
- ZILL, D. G.: *Equaciones Diferenciales*, 2001.