1. INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

La gestión de los recursos hídricos es una cuestión de importancia capital en toda sociedad moderna, y buena parte de estos recursos se halla almacenada en acuíferos. Es cierto que las aguas subterráneas sufren procesos de filtración y mineralización que pueden aumentar su calidad de manera natural, pero de la misma forma son susceptibles de captar substancias nocivas procedentes de actividades como la agricultura (fertilizantes), la industria química (aceites, carburantes) o la minería (metales pesados). Se convierte, por tanto, en una necesidad conocer el comportamiento de estas substancias en el agua subterránea: determinar su movimiento, su balance de masa a lo largo de éste, etc.

La hidrología subterránea nos proporciona modelos teóricos que describen, por un lado el movimiento del agua en el acuífero, y por otro el movimiento de un determinado soluto en el agua subterránea. Como se explicará en el primer capítulo, estos modelos están descritos por ecuaciones que, incluso en algunos casos simples, pueden no tener una solución analítica factible. Se hace necesario, pues, resolver dichas ecuaciones utilizando técnicas numéricas adecuadas que, dadas las dimensiones de los dominios habituales de trabajo, deberán ser resueltas por un ordenador.

Dichas técnicas numéricas se conocen muy bien hoy en día, por aplicarse en esencia las mismas a diferentes problemas en diferentes ramas de la ciencia. Nos referimos, por supuesto, a los métodos de elementos finitos, de diferencias finitas y algunos otros, que tan utilizados son. Veremos que existen programas comerciales de una versatilidad y potencia considerables para llevar a cabo esta tarea, por ejemplo MODFLOW, MT3D, HST3D, y otros. Estos programas tienen implementados muchos de los métodos numéricos más habituales, para resolver modelos lo más generales posible.

Aparentemente, entonces, lo único que hay que hacer es ir ampliando estos programas, conforme se desarrollen nuevas técnicas matemáticas mejores, maneras diferentes de enfocar el problema físico, o se consideren condiciones más estrictas que hagan el modelo más realista. Es cierto que esto es lo único que hay que hacer, pero no es fácil, y se verá por qué.

Estos programas están en su mayoría construidos mediante técnicas clásicas, lo que se llama programación secuencial. En este tipo de programación, existe una línea de flujo en la que se ejecutan sucesivamente las tareas necesarias para el funcionamiento del programa. Estas tareas pueden ser bloques de código inmersos dentro de la línea de flujo, o bloques separados (también llamados “módulos”). Ahora bien, la separación de estos bloques de la línea principal es una simple cuestión de organización visual. Esto es porque todas las variables existentes en el programa deben ser declaradas al principio de la línea principal de flujo. Aunque es posible declarar otras fuera de ésta o durante la

ejecución, ello tiene muchas limitaciones. Por tanto, para añadir una subrutina, por pequeña que sea, debemos modificar todo el programa, desde la declaración de variables. Las modificaciones pueden no ser demasiado substanciales para una subrutina pequeña, pero sí para una cierta cantidad de código. Por otro lado, resulta difícil combinar diferentes métodos numéricos. Por ejemplo, si un programa se ha desarrollado para diferencias finitas, resulta complejo extenderlo a elementos finitos.
La existencia de estas reglas tan rígidas para los programas comerciales más usados, dificulta notablemente su ampliación. Incluso podemos decir que son tremendamente crípticos para cualquiera que no los conozca, lo que todavía complica más las cosas a la hora de la incorporación de nuevo código por un programador que desconozca el existente.

Ahora bien, existe un tipo de programación, cada vez más utilizada, que soluciona muchos de los defectos de los que peca la programación secuencial. Se trata de la programación orientada a objeto.

En esta clase de programación también existe una línea de flujo, cierto, pero se limita a crear, manipular y destruir una serie de *objetos*. Más adelante se verá lo que es un objeto, aunque en esencia, podemos decir que no es más que un conjunto de variables organizadas (denominadas *atributos*) según una pauta común, y sobre las que se pueden realizar unas operaciones predeterminadas. ¿Quién decide la organización de atributos y operaciones? Cada objeto pertenece a una *clase* determinada, y es esta clase la que determina la estructura de los objetos.

Por hacer una analogía, en un lenguaje común tenemos diferentes tipos de variables que almacenan datos de índole diversa. Por ejemplo, tenemos variables enteras y reales. En cierta manera, las operaciones que se pueden realizar sobre estas variables están limitadas por su tipo: así, una variable entera no puede dividirse por un número por el que no sea divisible. De la misma manera, los objetos contienen datos diferentes, y sobre ellos no se pueden realizar operaciones no definidas en su clase. Sólo que, tanto los datos como las operaciones, ahora pueden ser todo lo complicadas que uno quiera.

Por supuesto, el código que al final la máquina interpretará es igual utilizando reglas clásicas o bien orientación a objeto. Lo único que estamos haciendo es **localizar la información** para tenerla constantemente controlada, de manera que sólo esté al alcance de los objetos que la necesitan. Esto facilita la combinación de funciones (multimorfismo) y la herencia de propiedades (un objeto puede contener otros objetos).

El crecimiento del programa se facilita enormemente con una estructura tan bien organizada, aunque, como se verá, ésta tiene el problema precisamente de que necesita un conocimiento previo importante. Una vez superado este pequeño obstáculo, la versatilidad está garantizada.

1.2. Objetivos

El fin último de este documento es estudiar las ventajas de la programación orientada a objeto, de cara a resolver el problema concreto del transporte de solutos en medios porosos, mediante el desarrollo del programa PROW (PRocess Oriented groundWater)

Para conseguir esto, comenzamos describiendo brevemente el problema físico en cuestión, y las ecuaciones fundamentales que lo describen.

Posteriormente mostramos la manera de implementar estas ecuaciones en un sistema informático, siguiendo diversos métodos. En concreto, explicamos sucintamente los métodos de elementos finitos y de diferencias finitas. Así mismo explicamos como se implementan dos métodos mixtos conocidos como MOC (Method Of Characteristics), y MMOC (Modified Method Of Characteristics). Todos estos métodos han sido implementados en el programa PROW.

Una vez descrito el problema, y las herramientas matemáticas necesarias para su resolución, pasamos a explicar a grandes rasgos lo que es la programación orientada a objeto en términos generales. Debemos tener en cuenta que PROW ha sido programado en FORTRAN90, que NO es un lenguaje de programación estrictamente orientada a objeto. Por eso vamos a explicar la manera de "simular" esta clase de programación en este lenguaje.

Hecho esto, sólo queda describir el programa, y lo más importante, las dificultades que surgieron entorno a su ampliación, así como algunas recomendaciones y advertencias a la hora de utilizar la programación orientada a objeto en un lenguaje no diseñado al efecto.

Para acabar, se resolverá un pequeño problema de transporte de solutos en aguas subterráneas mediante PROW, y se expondrán los resultados obtenidos, comentando lo más relevante acerca de éstos.

2. EL PROBLEMA DEL TRANSPORTE DE SOLUTOS EN AGUAS SUBTERRÁNEAS

Con el fin de introducir la formulación sobre la que posteriormente se desarrollará el programa, en esta sección se hace una breve descripción de los conceptos fundamentales del transporte de solutos en aguas subterráneas.

A continuación se muestran las ecuaciones que gobiernan el flujo de agua y el transporte de solutos. Se presenta, asimismo, el procedimiento de cálculo que se sigue habitualmente.

2.1. Ecuación de flujo

La ecuación de flujo es el resultado de imponer el principio de conservación de masa y suponer que el movimiento de agua obedece la ley de Darcy. La conservación de masa sobre cualquier volumen del acuífero implica:

$$\frac{\partial \rho \phi}{\partial t} = -\nabla(\rho \mathbf{q}) + \rho r \tag{2.1}$$

donde ρ es la densidad del agua, ϕ es la porosidad del acuífero, $\mathbf{q}$ es el flujo de agua y r representa los términos fuente/sumidero. En esta ecuación, el primer miembro representa la variación de masa de agua almacenada por unidad de volumen de acuífero. El primer sumando del segundo miembro representa las entradas menos las salidas de agua sobre cada volumen desde/hacia los volúmenes de acuífero adyacentes (expresadas como masa de agua por unidad de volumen de acuífero y por unidad de tiempo). Por último, el segundo sumando representa las entradas menos las salidas desde/hacia el exterior. El flujo, $\mathbf{q}$, viene dado por la conocida ley de Darcy:

$$\mathbf{q} = -\mathbf{K}\nabla h \tag{2.2}$$

donde $\mathbf{K}$ es el tensor de conductividad hidráulica y la h es el nivel. Por otro lado, es habitual aproximar el primer sumando linealmente como:

$$\frac{\partial \rho \phi}{\partial t} = \rho S_s \frac{\partial h}{\partial t} \tag{2.3}$$

donde S_s es el llamado coeficiente de almacenamiento específico (volumen de agua extraído por unidad de volumen de acuífero y por unidad de descenso de nivel).

Sustituyendo (2.2) y (2.3) en (2.1) y despreciando las variaciones espaciales de densidad, se obtiene la forma básica de la ecuación de flujo:

$$S_s \frac{\partial h}{\partial t} = \nabla(\mathbf{K} \nabla h) + r \tag{2.4}$$

Esta ecuación se resuelve en el dominio del acuífero con las condiciones iniciales y de contorno apropiadas. En principio, representa el flujo de agua en tres dimensiones. Sin embargo, dado que los acuíferos son entidades esencialmente bidimensionales, es habitual eliminar la coordenada vertical integrando en z. La ecuación resultante es análoga a (2.4), excepto que en lugar de expresar la conservación de masa por unidad de volumen lo hace por unidad de superficie de acuífero. Ello implica los cambios correspondientes en los términos de almacenamiento y fuentes. Mención especial requiere la transmisividad T, que esencialmente es la integral de **K** a lo largo de la vertical del acuífero.

La transmisividad y la conductividad hidráulica son parámetros muy variables en el espacio. Tiene sentido hablar de valores de K entre 10^{-10} y 10^{5} m/día. Dado que su variabilidad se expresa en órdenes de magnitud, no sorprende que sus valores puntuales suelan adoptar una distribución logarítmico normal (Davis, 1969).

2.2. Ecuación de transporte

La ecuación de transporte expresa la conservación de la masa de un soluto. Por tanto, se deriva de establecer dicha conservación, así como de suponer que el flujo de soluto puede tener lugar por tres procesos, que son difusión, dispersión y advección.

Esto puede escribirse como:

$$\frac{\partial(\phi c)}{\partial t} = -\nabla(\mathbf{j}) + f \tag{2.5}$$

donde $\mathbf{j}$ es el vector de flujo de masa; c es la concentración. El término f incluye todas las posibles fuentes y sumideros. El vector de flujos $\mathbf{j}$ puede descomponerse en la suma de $\mathbf{j}_a$, flujo debido a advección, $\mathbf{j}_m$, el debido a dispersión mecánica, y $\mathbf{j}_d$ debido a difusión molecular. Este se obtiene mediante la ley de Fick, equivalente a la de Darcy, con una corrección en el coeficiente de difusión en medio continuo, D_d, por la porosidad, ϕ, y el tensor de tortuosidad, $\boldsymbol{\tau}$, es decir:

$$\mathbf{j}_d = -\phi D_d \boldsymbol{\tau} \cdot \nabla c \tag{2.6}$$

El flujo advectivo estará dado por el producto del caudal unitario (velocidad de Darcy), por la concentración, esto es:

$$\mathbf{j}_a = \mathbf{q}\, c \tag{2.7}$$

La caracterización del flujo dispersivo es más compleja. Para ello se suele seguir el método propuesto por Scheidegger y De Josselin De Jong, que está recogido por Bear (1972), y que consiste esencialmente en promediar el transporte a través de capilares aleatoriamente distribuidos. Este enfoque conduce a una expresión similar a la de la difusión molecular, aunque en una escala más grande, es decir:

$$\mathbf{j}_m = -\mathbf{D}_m \cdot \nabla c \tag{2.8}$$

donde $\mathbf{D}_m$ es el tensor de dispersión mecánica. Sus direcciones principales son paralelas y perpendiculares al flujo y sus valores principales son los coeficientes de dispersión longitudinal y transversal, respectivamente. Si se acepta la independencia entres dispersión mecánica y difusión molecular, es intuitivo que el coeficiente de dispersión resulte proporcional a la velocidad (ver figura 2.1). Esto es debido al hecho de que la trayectoria recorrida por las partículas es independiente de la velocidad. Así resulta:

$$\begin{aligned} D_L &= \alpha_L \, v \\ D_T &= \alpha_T \, v \end{aligned} \tag{2.9}$$

donde α_L y α_T son las dispersividades longitudinal y transversal, respectivamente. Estas tienen dimensiones de distancia y, en principio, se consideran características del medio. Existen expresiones más complicadas para el coeficiente de dispersión, pero su uso no está generalizado.
Sustituyendo (2.6), (2.7) y (2.8) en (2.5), y restando la ecuación de continuidad (2.1) multiplicada por c y dividida entre ρ, resulta, tras algunas operaciones:

$$\phi \frac{\partial c}{\partial t} = \nabla(\mathbf{D}\nabla c) - \mathbf{q}\nabla c + f - rc \tag{2.10}$$

donde $\mathbf{D}$ es la suma de los tensores de dispersión y de difusión. Esta ecuación también se resuelve con las condiciones iniciales y de contorno apropiadas.

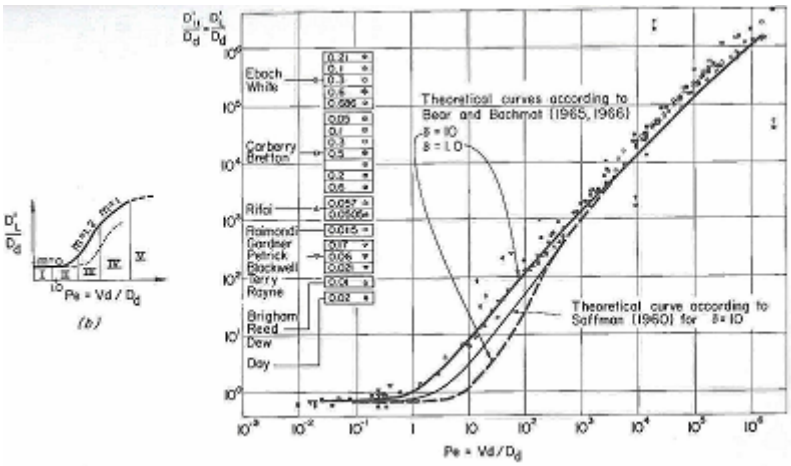

Figura 2.1. Relación entre dispersión (eje vertical adimensionalizado por D_d) y advección. Se puede observar que la dispersión crece con la velocidad (Bear, 1972)

2.3. Sentido físico de la solución

Las ecuaciones anteriores se resuelven de manera sucesiva:

1. Resolver flujo (ec. 2.4 más condiciones iniciales y de contorno) para obtener h
2. Calcular $\mathbf{q}$ mediante la ley de Darcy.
3. Resolver la ecuación (2.10) para calcular c.

La ecuación de flujo suele resolverse mediante métodos numéricos apropiados (elementos finitos, diferencias finitas, etc.). Sin embargo, los niveles son relativamente fáciles de obtener o, mejor dicho, dadas las condiciones de contorno y algunas medidas en pozos, se pueden estimar. De hecho, puede tenerse una idea aproximada de los niveles dibujando una red de flujo. Es razonable, pues, pensar que los niveles se conocen relativamente bien. Por ello, en el cálculo del flujo mediante la ecuación (2), la principal fuente de incertidumbre será la conductividad hidráulica. Es preciso, sin embargo, reconocer que esta incertidumbre puede ser muy grande.
Conocido el flujo, $\mathbf{q}$, se procede a la solución de la ecuación de transporte. Para ello, también es habitual emplear métodos numéricos. Aunque ahora las dificultades computacionales son formidables, se puede obtener una idea aproximada de la solución a partir de la compresión de los procesos de advección y dispersión. Ambos quedan ilustrados en la Figura 2.2.

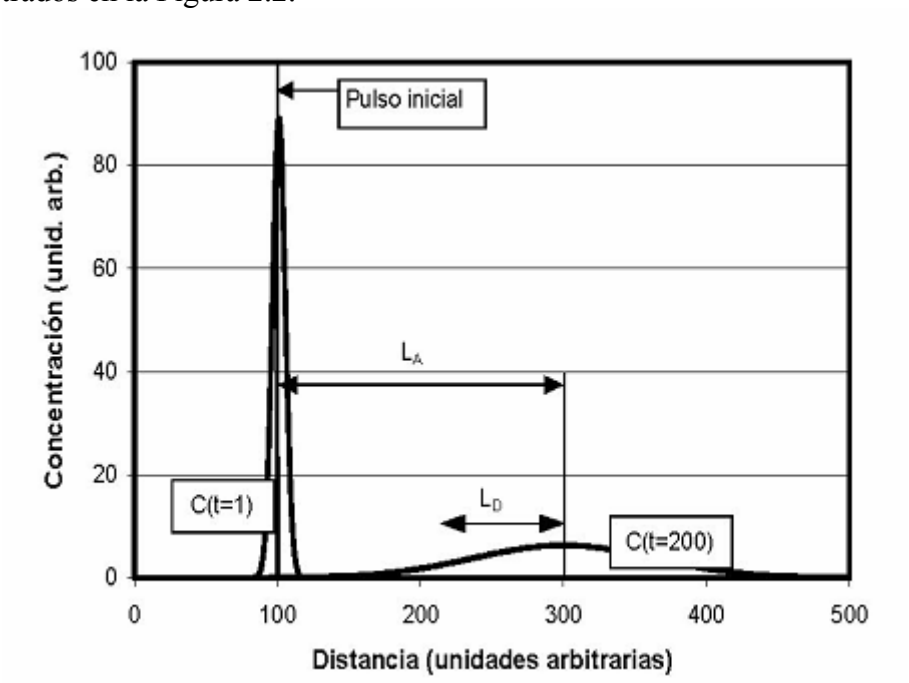

Figura 2.2. Concentraciones calculadas después de 1 y 200 días tras la inyección de un pulso de 1000 g. en un fluido que se mueve hacia la derecha a 1 m/d con un coeficiente de difusión de 10 m^2/d y porosidad de 1. Al cabo de un día, la difusión es apreciable (las concentraciones definen la conocida campana de Gauss), pero el centro de gravedad no parece haberse desplazado. Por el contrario, tras 200 días, el desplazamiento es muy marcado y supera la difusión, de manera que la concentración es despreciable en el punto de inyección.

El penacho se desplaza una distancia dada por la escala de advección, L_A :

$$L_A = \frac{\mathbf{q}\,t}{\phi} = v\,t \tag{2.11}$$

Al mismo tiempo, el contaminante se va diluyendo por dispersión. Ello hace que el tamaño del penacho crezca como:

$$L_D = \sqrt{\frac{2\,\mathbf{D}\,t}{\phi}} = \sqrt{\frac{2\,\alpha\,\mathbf{q}\,t}{\phi}} = \sqrt{2\,\alpha\,v\,t} = \sqrt{2\,\alpha\,L_A} \tag{2.12}$$

donde, implícitamente se ha despreciado la difusión y donde se ha dejado sin especificar si la dispersividad es longitudinal o transversal (la escala de dispersión será una en dirección longitudinal y otra en transversal). La dilución también reduce la concentración máxima (la misma masa ocupa un volumen creciente de agua), que adopta la expresión:

$$c_{\max} = \frac{M}{\phi\sqrt{\left|2\,\pi\,\mathbf{D}\,t/\phi\right|}} = \frac{M}{\phi\left|L_D\right|} \tag{2.13}$$

donde $c_{\max}$ es la concentración, M es la masa inyectada y $\left|.\right|$ representa determinante (así, $\left|L_D\right|$ es $L_{DL}\,L_{DT}\,L_{DT}$, siendo L_{DL} y L_{DT} las longitudes de dispersión longitudinal y transversal, respectivamente).
Lo importante de esta sucesión de cálculos es que pone de manifiesto que, en principio, bastarían tres parámetros: $\mathbf{K}$, α y ϕ. La conductividad hidráulica controla el flujo, $\mathbf{q}$. La velocidad es directamente proporcional a $\mathbf{q}$ (y por tanto a $\mathbf{K}$) e inversamente proporcional a ϕ. La longitud de dispersión crece con la raíz de $\mathbf{q}$ (es decir, $\mathbf{K}$), α y $1/\phi$.

2.4. Condiciones de contorno

Las ecuaciones anteriores no pueden resolverse directamente, es necesaria la imposición de condiciones de contorno, así como el campo de velocidades en todos los puntos del dominio. Además, naturalmente, es necesario el conocimiento de todos los parámetros propios del medio, que deberán ser estimados utilizando las técnicas apropiadas.

Las condiciones de contorno que se imponen normalmente son las típicas de problemas lineales, es decir:

- **Concentración constante** (condición de Dirichlet): Esta condición se impone, por ejemplo, cuando una parte del contorno está en contacto con una fuente de concentración conocida, como un lago, un depósito…

- **Flujo dispersivo constante** (condición de Neumann): En la práctica, esta condición suele utilizarse sólo cuando el flujo dispersivo es nulo, como es el caso de un contorno impermeable, o el de salida de agua.

- **Condición mixta** (condición de Cauchy): Empleada cuando se conoce tanto la fuente de caudal unitario como la concentración.

3. MÉTODOS NUMÉRICOS

3.1. Introducción y clasificación

Los métodos para la simulación de la ecuación de transporte de solutos pueden ser clasificados de acuerdo con la forma de la ecuación que pretenden resolver, como eulerianos, lagrangianos, y mixtos. Seguidamente se hace una pequeña revisión de los tres tipos de métodos.

3.1.1. Métodos eulerianos

Los métodos eulerianos se basan en la discretización de la ecuación de transporte en retícula fija. Son los más antiguos y comunes para la solución de esta ecuación. Entre éstos, una de las formulaciones más ampliamente usadas es la basada en diferencias finitas, que será descrita en el punto 3.2.1.
La otra gran formulación es la basada en elementos finitos, que describiremos en el punto 3.2.2.
A pesar de que ambas formulaciones dan lugar a sistemas de ecuaciones similares, sus filosofías son diferentes, como se verá más abajo.

3.1.2. Métodos lagrangianos

Los métodos lagrangianos no utilizan una retícula fija para discretizar la ecuación de transporte. Esto puede ser, bien utilizando una retícula fija sobre un sistema de coordenadas deformables, bien utilizando una retícula deformable sobre un sistema de coordenadas fijo.

En el primer caso, se puede mencionar el método propuesto por Jensen y Finlayson (Carrera y Melloni, 1987), en el que la ecuación de transporte se resuelve usando una malla cuyo orígen coincide con el centro del frente móvil de contaminación (soluto). Si el campo de velocidades es uniforme, el término advectivo desaparece de la formulación. Si no, es necesario considerar ciertos términos residuales.

La idea en el caso de los métodos de retícula deformable, es desplazar los nudos de la malla a lo largo de las líneas características en cada incremento de tiempo. Éstos métodos proporcionan buenos resultados, pero también sufren ciertas limitaciones importantes. Por ejemplo, su coste de ejecución es muy alto debido a la necesidad de redefinir continuamente las posiciones de los nudos, las matrices del sistema, etc. Además, si la malla se deforma en exceso, puede resultar un problema en caso de permeabilidad muy heterogénea, lo que lleva a una pérdida de precisión. Por otra parte, la definición de los parámetros en medios heterogéneos podría resultar conflictiva.

Estas dificultades limitan la aplicabilidad de los métodos lagrangianos, y son la causa de la aparición de los métodos mixtos, que discutimos en el siguiente subcapítulo.

3.1.3. Métodos mixtos euleriano-lagrangianos

Estos métodos se basan en resolver la ecuación del transporte en una retícula fija. Intentan combinar la precisión de los métodos lagrangianos con la eficiencia y generalidad de los métodos eulerianos. Un ejemplo idealizado de los métodos euleriano-lagrangianos sería uno en el cual la malla se construyera asegurando que los nodos están situados sobre las líneas de corriente del campo de velocidades y que el tiempo de viaje entre nodos vecinos es constante.

Así, el proceso de solución se descompone en dos fases. En la primera, las concentraciones nodales se desplazan un nodo en la dirección del flujo; y en la segunda, se calcula el efecto de la dispersión. La limitación que tendría este método es que grandes variaciones en el campo de velocidades llevarían a mallas demasiado deformadas (como en el caso de los métodos lagrangianos ya descritos).

Hay, naturalmente, métodos menos restringidos, y entre ellos están el método de las características (MOC). Este método, como el anterior, separa en dos fases el proceso de solución, pero para resolver la fase advectiva, utiliza unas partículas que se mueven libremente por el dominio, siguiendo las líneas de velocidad del campo. Este método se describirá en el apartado 3.2.3. Una solución similar, pero que pretende corregir la necesidad de una enorme cantidad de partículas que tiene MOC, es el método modificado de las características (MMOC), que se describirá en el apartado 3.2.4.
Además de los citados métodos, existen otros que podrían considerarse incluídos en este mismo grupo (euleriano-lagrangianos). Entre ellos está el de las celdas de mezcla, que podría describirse como una cadena de depósitos a través de los cuales, el fluido se mueve secuencialmente. Otro método de esta misma clase, que tiene en común con MOC y MMOC el hacer uso de partículas es el método del camino aleatorio (*Random Walk*).

3.2. Descripción de algunos métodos importantes

En este capítulo se describirán cuatro métodos, ya citados en el capítulo precedente, por haber sido implementados en el programa PROW.

3.2.1. Método de diferencias finitas

La discretización utilizada en PROW nos lleva a las siguientes expresiones para los términos de la ecuación de transporte. El término advectivo queda así:

$$\frac{\partial}{\partial x_i}\left(q_i C\right) = q_{xi,j+1/2}\frac{\alpha_{xj+1/2}C_{i,j}^{n+1} + (1-\alpha_{xj+1/2})C_{i,j+1}^{n+1}}{\Delta x_j} - q_{xi,j-1/2}\frac{\alpha_{xj-1/2}C_{i,j-1}^{n+1} + (1-\alpha_{xj-1/2})C_{i,j}^{n+1}}{\Delta x_j} +$$
$$+ q_{yi+1/2,j}\frac{\alpha_{yi+1/2}C_{i,j}^{n+1} + (1-\alpha_{yi+1/2})C_{i+1,j}^{n+1}}{\Delta y_i} - q_{yi-1/2,j}\frac{\alpha_{yi-1/2}C_{i-1,j}^{n+1} + (1-\alpha_{yi-1/2})C_{i,j}^{n+1}}{\Delta y_i} \qquad (3.1)$$

siendo C la concentración, q la velocidad en el sentido de Darcy, y α unos factores de ponderación entre aguas arriba y aguas abajo de la celda considerada (i,j). Aquí, i,j

representan número de fila y número de columna, respectivamente, dentro de la malla, como se muestra en la figura 3.1.

La expresión para el término dispersivo es algo más compleja:

$$
\begin{aligned}
&\frac{\partial}{\partial x_i}\left(D_{ij}\frac{\partial C}{\partial x_j}\right)=\frac{\partial}{\partial x}\left(D_{xx}\frac{\partial C}{\partial x}\right)+\frac{\partial}{\partial x}\left(D_{xy}\frac{\partial C}{\partial y}\right)+\frac{\partial}{\partial y}\left(D_{yx}\frac{\partial C}{\partial x}\right)+\frac{\partial}{\partial y}\left(D_{yy}\frac{\partial C}{\partial y}\right)= \\
&= D_{xxi,j+1/2}\frac{C_{i,j+1}^{n+1}-C_{i,j}^{n+1}}{\Delta x_j(0.5\Delta x_j+0.5\Delta x_{j+1})}-D_{xxi,j-1/2}\frac{C_{i,j}^{n+1}-C_{i,j-1}^{n+1}}{\Delta x_j(0.5\Delta x_{j-1}+0.5\Delta x_i)}+ \\
&+D_{xyi,j+1/2}\frac{\omega_{xj+1/2}C_{i+1,j}^{n+1}+(1-\omega_{xj+1/2})C_{i+1,j+1}^{n+1}-\omega_{xj+1/2}C_{i-1,j}^{n+1}-(1-\omega_{xj+1/2})C_{i-1,j+1}^{n+1}}{\Delta x_j(0.5\Delta y_{i-1}+\Delta y_i+0.5\Delta y_{i+1})}- \\
&-D_{xyi,j-1/2}\frac{\omega_{xj-1/2}C_{i+1,j-1}^{n+1}+(1-\omega_{xj-1/2})C_{i+1,j}^{n+1}-\omega_{xj-1/2}C_{i-1,j-1}^{n+1}-(1-\omega_{xj-1/2})C_{i-1,j}^{n+1}}{\Delta x_j(0.5\Delta y_{i-1}+\Delta y_i+0.5\Delta y_{i+1})}+ \qquad (3.2) \\
&+D_{yxi+1/2,j}\frac{\omega_{yi+1/2}C_{i,j+1}^{n+1}+(1-\omega_{yi+1/2})C_{i+1,j+1}^{n+1}-\omega_{yi+1/2}C_{i,j-1}^{n+1}-(1-\omega_{yi+1/2})C_{i+1,j-1}^{n+1}}{\Delta y_i(0.5\Delta x_{j-1}+\Delta x_j+0.5\Delta x_{j+1})}- \\
&-D_{yxi-1/2,j}\frac{\omega_{yi-1/2}C_{i-1,j+1}^{n+1}+(1-\omega_{yi-1/2})C_{i,j+1}^{n+1}-\omega_{yi-1/2}C_{i-1,j-1}^{n+1}-(1-\omega_{yi-1/2})C_{i,j-1}^{n+1}}{\Delta y_i(0.5\Delta x_{j-1}+\Delta x_j+0.5\Delta x_{j+1})}+ \\
&+D_{yyi+1/2,j}\frac{C_{i+1,j}^{n+1}-C_{i,j}^{n+1}}{\Delta y_i(0.5\Delta y_i+0.5\Delta y_{i+1})}-D_{yyi-1/2,j}\frac{C_{i,j}^{n+1}-C_{i-1,j}^{n+1}}{\Delta y_i(0.5\Delta y_{i-1}+0.5\Delta y_i)}
\end{aligned}
$$

En la figura 3.1. se puede ver un ejemplo de una pequeña malla de diferencias finitas, en la que se pueden distinguir los elementos característicos básicos para su definición (tal y como PROW trata este tipo de mallas). La malla se supone incluida en un rectángulo de referencia. La discretización se define dando los intervalos DX y DY para cada línea de celdas, en dirección horizontal y vertical. Así mismo, se declara qué celdas pertenecen al dominio, y el programa las numera tal y como se ve en la figura.
Los valores DX y DY pueden ser diferentes para cada línea, pero no para dos celdas de una misma línea. De aquí ya podemos deducir que esta malla, salvo casos muy concretos, es mejor usarla con una discretización regular y bastante refinada, dadas las limitaciones de eficiencia del método. NUMNP es una variable de PROW que representa el número de puntos en los que se da la solución. Aquí corresponden a los puntos centrales de cada celda. En el método de elementos finitos, que vamos a ver a continuación, representa el número de nodos.

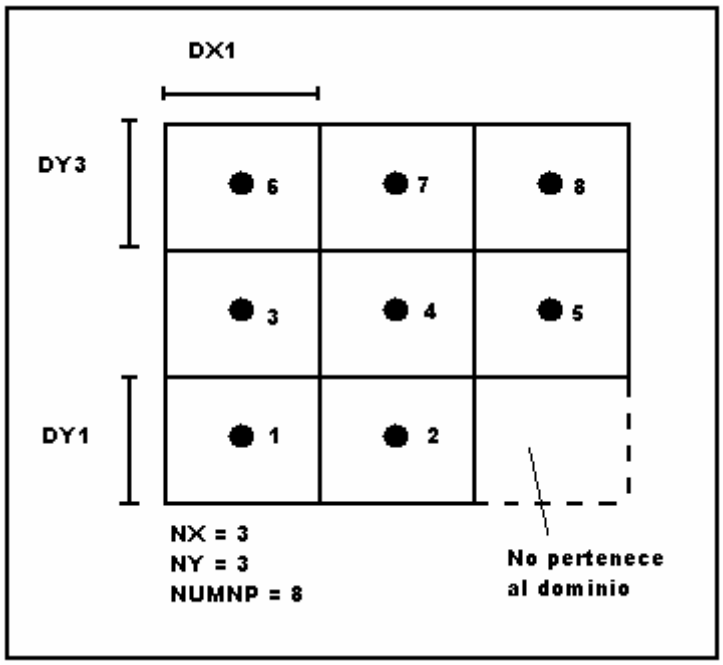

Figura 3.1. Malla de diferencias finitas

3.2.2. Método de elementos finitos (MEF)

El método de elementos finitos es una técnica numérica para resolver problemas que están descritos por ecuaciones diferenciales parciales o que pueden ser formulados como una minimización funcional.

El dominio de interés se representa como el ensamblaje de *elementos finitos*. Por otra parte, se utilizan funciones de aproximación, expresadas en términos de los valores nodales de un campo físico que se busca. Un problema físico continuo se transforma en un problema discreto de elementos finitos cuyas incógnitas son los valores del campo en los nodos (puntos comunes de varios elementos). Si el problema físico es lineal, el problema de elementos finitos resultante es un sistema de ecuaciones lineales. Una vez resueltos los valores en los nodos, cualquier otro valor se obtiene a partir de las funciones de aproximación.

Dos características importantes merecen ser citadas:

- La aproximación a trozos de campos físicos es de gran precisión, incluso con funciones de aproximación simples. Incrementando convenientemente el número de elementos podemos conseguir cualquier precisión.
- La localización de la aproximación lleva a sistemas de ecuaciones dispersos en el problema discretizado. Esto ayuda a resolver problemas con un gran número de incógnitas nodales.

Como el resto de métodos con los que puede trabajar PROW hasta el momento, el método de elementos finitos trabaja en un dominio bidimensional. El tipo de método

que ha sido implementado en PROW utiliza una malla de elementos triangulares lineales, tal y como había sido implementado previamente en TRACONF.

Veamos ahora un ejemplo de malla de elementos finitos, tal y como es tratada por PROW. En la figura 3.2. se muestra un pequeño ejemplo. En este caso, a diferencia del método anterior, primero se definen los nodos, numerados como muestra la figura. Para cada uno de ellos se deben especificar las coordenadas. Una vez definidos los nodos se hace lo mismo con los elementos. Para ello, se especifica qué nudos pertenecen a cada elemento. Por ejemplo, en la figura se puede ver que al elemento número 2 pertenecen los nodos 2, 5 y 4. Se deben numerar en este orden (antihorario) por necesidad del programa. Hay muchas formas de conectar los nodos para formar elementos. En principio, cuanto más se adapte la malla al problema tratado, más precisa la solución. Eso quiere decir que podría refinarse la malla sólo localmente, por ejemplo, porque se fuera a introducir una inyección de soluto en un punto determinado.
Por supuesto, el ejemplo presentado en la figura sólo es ilustrativo, y los nodos podrían estar dispuestos con total libertad de forma irregular en el plano. Se ha hecho así, por analogía con la figura mostrada para diferencias finitas.
La variable NUMNP es en este caso el número de nodos (15). La variable NUMEL es el número de elementos. En diferencias finitas sólo tratabamos con unos entes que eran las celdas, y NUMEL tomaría el mismo valor que NUMNP, para propósitos de unificación de la programación de ambos métodos. Estos temas se verán con más detalle en el capítulo dedicado a PROW.

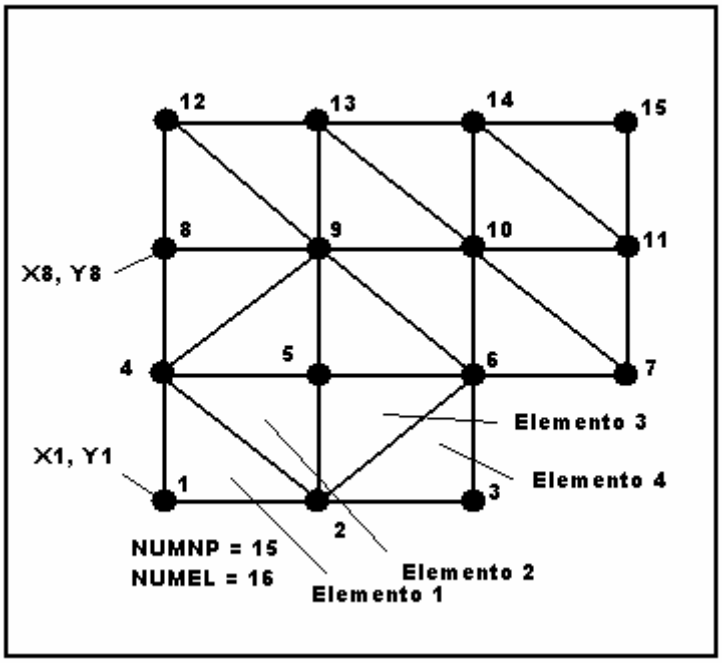

Figura 3.2. Malla de elementos finitos

3.2.3. Método de las características (MOC)

El método de las características (MOC) fue aplicado en su origen al transporte en medios porosos por Garder, Peaceman, Pozzi (Zheng y Wang, 1999) para calcular el

desplazamiento miscible en simulación de presas. Posteriormente se popularizó y ha sido muy utilizado en estudios de campo.

Ya hemos dicho anteriormente que el método MOC comienza por incorporar el término advectivo de la ecuación del transporte (segundo sumando del término derecho de la ecuación 9, capítulo 1) al término de la izquierda de la ecuación (derivada local), para formar la derivada material de la concentración de soluto. Posteriormente, esta ecuación se discretiza (por ejemplo, utilizando una técnica de elementos finitos o diferencias finitas). Se utiliza entonces una técnica convencional de seguimiento de partículas para simular el término advectivo. Al principio de la simulación, un conjunto de partículas móviles es distribuído en el dominio siguiendo un determinado patrón. Éste puede consistir en una disposición fija dentro de los dominios de cada elemento o celda, o bien una disposición aleatoria. A cada partícula se le asocia una posición y una masa de soluto (hay variantes que asocian una concentración). Sin embargo, esta masa la asociamos mediante dos valores: una concentración y el volumen de la celda en la que cada partícula fue generada. Las partículas son desplazadas, durante un intervalo de tiempo pequeño, sobre el campo de velocidades del flujo (figura 3.3).

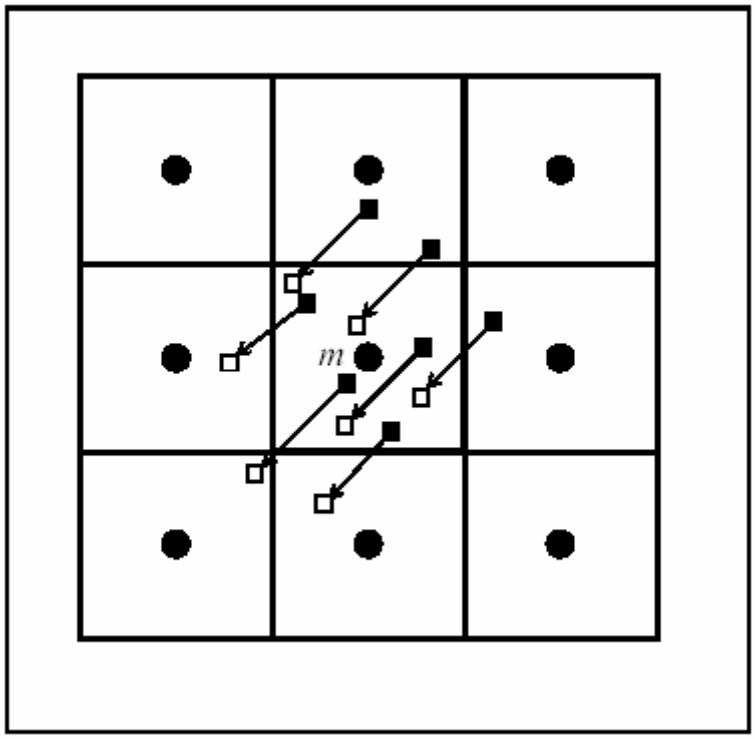

Figura 3.3. Movimiento de las partículas en el método MOC

Entonces se calcula una concentración media para cada celda/elemento a partir de la nueva disposición de masa debida al desplazamiento de las partículas. A esta concentración intermedia la llamamos C^{n*}, y debe tenerse en cuenta que es puramente ficticia. No corresponde ni a C^{n} ni a C^{n+1}, la utilizaremos en la ecuación de transporte, quedando esta de la siguiente manera (fórmula 3.3):

$$\frac{\phi(C^{n+1} - C^{n*})}{\Delta t} = \nabla(D \cdot \nabla C^{n+\theta}) + f \qquad (3.3)$$

Pudiendo entonces ser resuelta según el método empleado (diferencias finitas, elementos finitos), pero sin el término advectivo, que ya va incorporado en la concentración C^{n*}. Para evaluar esta concentración se puede utilizar un método aritmético simple, quedando la siguiente expresión (Zheng y Wang, 1999):

$$C_m^{n*} = \sum_{p=1}^{NP_m} V_p C_p^n / \sum_{p=1}^{NP_m} V_p \qquad (3.4)$$

siempre que $NP_m>0$. NP_m es el número de partículas en la celda m. V_p es el volumen de la celda en la que la partícula p se generó, y C^n_p la concentración asociada a la partícula. Es posible también sustituir la concentración C^{n*} por un valor intermedio entre C^{n*} y C^n, teniendo en cuenta que, en realidad, el proceso de dispersión se da **durante** el incremento de tiempo en el que movemos las partículas (Zheng y Wang, 1999).

Una vez resuelta la ecuación y obtenidas las nuevas concentraciones en las celdas, se actualizan las concentraciones asociadas a las partículas para reflejar los cambios debidos a dispersión y otros procesos. Esto completa un paso de transporte en el método MOC. El proceso se repite durante el periodo de tiempo deseado.

Cuando una partícula sale fuera del dominio, se refleja en el contorno, de manera que todas las partículas siempre queden dentro. Cuando en una determinada celda hay más partículas de un determinado límite prestablecido por el usuario, éstas se destruyen y la masa que portaban se redistribuye entre otras partículas redispuestas por el programa.
De la misma manera, cuando en una celda hay menos partículas de un determinado límite inferior, éstas se destruyen y se redispone un número de partículas igual al mínimo, que lleven la misma masa que las anteriormente existentes.

La ventaja más importante del MOC es que está virtualmente libre de dispersión numérica por errores de truncamiento. Su principal desventaja, sin embargo, es que requiere una gran cantidad de memoria para almacenar las características de las partículas (porque pueden ser muchas), y que puede llegar a ser muy lento.
Por otra parte, podría haber discrepancias en el balance de masa, ya que la naturaleza discreta de los métodos mixtos Euleriano-Lagrangianos no garantiza la conservación local de la masa en un determinado incremento de tiempo.

Para reducir el uso de memoria, se puede utilizar un enfoque dinámico para el seguimiento de las partículas. Esto consiste en asociar un mayor número de partículas a aquellas celdas cuya concentración es superior a un cierto valor umbral, y un menor número al resto. De esta manera, nos ahorramos cargar con un montón de partículas portadoras de una masa nula o cercana a cero, pero que ocupan la misma cantidad de memoria.

3.2.4. Método modificado de las características (MMOC)

La principal desventaja del método de las características (MOC) es el gran sacrificio de eficiencia computacional que implica el mismo, debido al número de partículas cuyas características (posición, concentración, volumen) hay que mantener almacenadas durante todo el proceso de solución.

Para atenuar este problema, se desarrolló el método MMOC, inicialmente por Russel y Wheeler y Neumann (Zheng y Wang, 1999). Esta técnica es análoga al método MOC excepto en cuanto al tratamiento del término advectivo. MMOC sólo utiliza una partícula por cada celda del dominio.
Cada una de estas partículas se supone en su celda correspondiente en el instante n+1. Entonces se mueve la partícula hacia atrás en el tiempo siguiendo el campo de velocidades del flujo, hasta un instante intermedio n^* (Figura 3.4). La concentración asociada a la partícula en la posición que acabe ocupando en ese instante será C^{n*}, y se utilizará de la misma manera que se hizo en el método MOC, siendo a partir de entonces el proceso completamente igual al de dicho método.

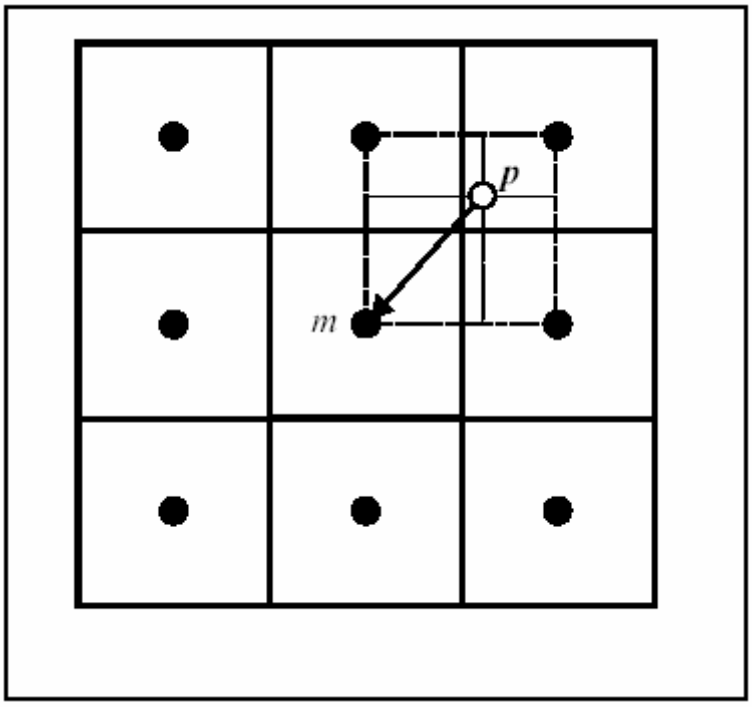

Figura 3.4. Movimiento de las partículas en MMOC

Las ventajas, naturalmente, son la mayor velocidad de ejecución del método, y el menor coste computacional que supone usar sólo una partícula por celda. Y no sólo eso. Las partículas, en cada nuevo incremento de tiempo se redisponen en las celdas para moverlas hacia atrás, lo que implica que no es necesario almacenar sus características a lo largo de todo el proceso, reduciéndose notablemente la cantidad de memoria usada.

Sin embargo, la desventaja de este método es que introduce dispersión numérica, especialmente en las zonas cercanas a frentes abruptos de concentración. Una manera de atenuar este efecto es utilizando esquemas de interpolación de orden superior al lineal (que es el habitualmente usado), como por ejemplo, interpolación cuadrática (Zheng y Wang, 1999).
A su vez, esta interpolación introduce dos problemas: por una parte, puede generar oscilaciones artificiales en la solución, y por otra, reduce la ventaja computacional que presenta MMOC, al utilizar una interpolación de orden superior. Así, conviene este método para los problemas en que no hay presencia de frentes abruptos de concentración.

3.3. Programas comerciales

Existe gran variedad de programas comerciales destinados a la solución del problema del transporte de solutos en medios porosos. Entre ellos, cabe destacar MT3D (Zheng y Wang, 1999) como referente, ya que se ha convertido en uno de los programas más utilizados en la actualidad. MT3D fue desarrollado en origen por Chunmiao Zheng en 1990 en S.S. Papadopoulos & Associates, y actualmente continúa desarrollándose en la Universidad de Alabama. MT3D sólo resuelve transporte de solutos (que puede ser reactivo y con múltiples especies), y necesita un campo de flujo resuelto previamente por algún otro programa. Está particularmente diseñado para trabajar conjuntamente con MODFLOW (Harbaugh, Banta, Hill et al, 2000), clásico programa de flujo en medios porosos, desarrollado por el USGS. Parte de los algoritmos de PROW han sido desarrollados basándose en los implementados en MT3D

Otro importante programa es HST3D (Kipp, 1997), desarrollado por el USGS (Servicio Geológico Estadounidense), que puede resolver flujo, transporte de solutos, y además, transporte de calor.

Algunos de estos programas (en particular, los más utilizados) están construídos utilizando FORTRAN77, y técnicas de programación clásicas. Todavía no existe un entorno ampliamente utilizado para resolver problemas de transporte en medios porosos que haya sido programado mediante técnicas de orientación a objeto.

4. SOBRE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Este capítulo centra algunos conceptos que se emplearán más adelante para describir la estructura del programa PROW.

4.1. Motivación

La gran mayoría de programas de elementos finitos (cito este método por ser el más versátil y difundido en los últimos tiempos) están escritos en *lenguajes de procedimiento* (*procedural languages*). Estos programas consisten esencialmente en una secuencia de llamadas a procedimientos (subrutinas, funciones), a los que se debe proveer de ciertos datos (sus argumentos).

La modularidad de estos procedimientos está limitada por dos deficiencias: la **debilidad de las estructuras de datos**, y la **secuencialidad de las operaciones (**Dubois-Pélerin y Zimmermann, 1992).

Las estructuras de datos en la programación por procedimientos contienen poca información, y provocan una interacción excesiva entre el procedimiento llamado y el resto del programa. Por otra parte, cuando el *cliente* (el código que requiere de un servicio) llama a un procedimiento necesita saber no sólo *qué* debe éste hacer, sino *cómo* debe hacerlo (Cross, Masters, Sukirman et al, 1997).
La secuencialidad implica que una subrutina depende del orden de las operaciones del programa, lo que hace que aquella deba conocer el contexto en cada momento, de nuevo interaccionando con el resto del programa (Dubois-Pélerin y Zimmermann, 1992).

La programación orientada a objetos (Object-Oriented Programming) aparece como una técnica adecuada para promover la modularidad de los programas. Como se verá en los siguientes apartados, los programas de este tipo se componen de objetos que almacenan su propia información, y pueden ser modificados unos a otros mediante ciertas operaciones. Aquí uno de los conceptos clave es que la información concerniente a un objeto sea solamente manipulable por ese objeto, y quede **oculta** al resto (Decyk, Norton y Szymanski, 1997). Se verá tambien que las clases están organizadas jerárquicamente, lo que facilita el control constante del movimiento de la información.

Por supuesto, esta programación no está exenta de sus desventajas: todos los sistemas con un cierto grado de jerarquía necesitan de una cierta "burocracia" para ser construidos. Esto implica que el diseño inicial del programa debe estar muy meditado. **Un diseño pobre en etapas tempranas del proceso de desarrollo puede ser muy difícil de corregir después** (Cross, Masters, Sukirman et al, 1997). Como contrapartida veremos que, sentadas unas bases de diseño coherentes desde el principio, los programas orientados a objeto son mucho más flexibles a la hora de ser ampliados que los programas clásicos y, una vez asumida esta filosofía, las ventajas acaban superando con creces a los inconvenientes.

4.2. Conceptos fundamentales

Pero, ¿qué es un objeto?. La definición de objeto está íntimamente ligada a la definición de clase (nomenclatura del lenguaje C) o módulo (nomenclatura de FORTRAN):

Una clase (módulo) es una estructura abstracta integrada por atributos y servicios (métodos, operaciones) que manipulan estos atributos. Un objeto no es más que una instancia (una representación) de una clase (Cross, Masters, Sukirman et al, 1997).

> *El significado de las palabras clase y objeto es muy cercana al significado que tienen coloquialmente. Por ejemplo:*
> *Los árboles forman una clase, que posee unos atributos (altura, robustez, tipo de hoja...) y realiza unas funciones (crecer, envejecer, perder las hojas...). Cuando el árbol crece está modificando su atributo altura. Cuando envejece puede variar su robustez.*
> *Todos los árboles cumplen las mismas funciones, aunque no todos lo hagan de la misma manera. Cada uno de los árboles es una instancia de la clase "árbol", y por tanto, un objeto de la misma. Así, unos crecen más rápido que otros. Pero todos pueden crecer en algún momento.*

Figura 4.1. : Analogía con los objetos reales

La figura 4.2. muestra un diagrama de una clase, con los elementos de los que se compone. Este diagrama viene a ser también un esquema del orden de los elementos a la hora de programar físicamente la clase.
Un objeto de un programa puede necesitar conocer algún valor que posee un objeto distinto, y por tanto, requerirá del servicio que se lo proporcione. Al objeto que llama a un servicio de otro objeto le llamamos *cliente*. Al que lo proporciona, *servidor* (Cross, Masters, Sukirman et al, 1997).

Figura 4.2. : Los componentes de una clase

Una vez definidos los conceptos básicos de clase y objeto, vamos a ver algunos principios de la programación orientada a objeto.

El primer principio a describir, que está muy relacionado con la ocultación de la información que ya he citado antes, es el del **encapsulamiento de la información**.
Según este principio, los objetos guardan los datos referentes a sus propios atributos. Los métodos que pueden aplicarse a cada objeto pueden ser privados o públicos, pero un objeto sólo podrá acceder a un atributo de un objeto ajeno mediante un método

público (figura 4.3.). Así, la interfaz (métodos públicos) de una clase debe definirse de tal modo que revele tan poco como sea posible de su funcionamiento interno.

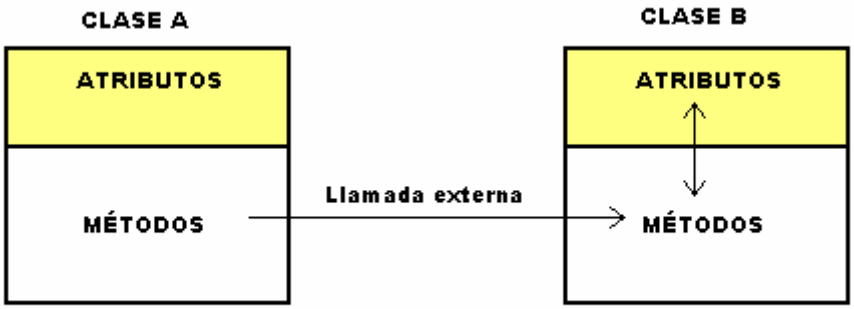

Figura 4.3. : Encapsulamiento de la información. Los atributos de B sólo son accesibles mediante métodos de B.

Dentro de una misma clase, podemos encontrar objetos cuyas funciones se realicen de una forma algo diferente, o cuyos atributos varíen. Por ejemplo, una matriz diagonal y una matriz llena pertenecen claramente a la clase matriz (MATRIX, p.ej.). Pero por otra parte la primera pertenece a la clase matriz diagonal (MT_DIAGONAL), y la segunda a la clase matriz llena (MT_FULL). Decimos que estas dos clases son **subclases** o **clases derivadas** (Cross, Masters, Sukirman et al, 1997) de la clase MATRIX. Volviendo al ejemplo de los árboles, los robles o las palmeras serían subclases de la clase "árbol".
El mecanismo por el cual las clases comparten su estructura y su comportamiento según una relación jerárquica se denomina **herencia** (Decyk, Norton y Szymanski, 1997 b). Mediante este mecanismo, sólo es necesario definir atributos y métodos en la clase de orden jerárquicamente superior (también llamada **superclase** (Dubois-Pélerin, Zimmermann, 1992), o clase **base** (Decyk, Norton y Szymanski, 1997 a). Las subclases que dependan de ella *heredan* estos atributos y métodos, pudiendo modificarlos o extenderlos, según sus necesidades (Decyk, Norton y Szymanski, 1997 a). El comportamiento de una llamada externa según el mecanismo de herencia se muestra en la figura 4.5.

Por ejemplo, cuando un objeto de una clase llamada MESH (que representa una malla), requiera realizar una operación sobre una matriz, no necesitará saber que tipo de matriz es. Solamente tendrá que llamar a dicha operación en la clase MATRIX (que representa las matrices), y será ésta la que decida a qué subclase llamar, en función del tipo de matriz que reciba como argumento. Naturalmente, esta operación puede realizarse de forma muy distinta en una subclase o en otra, siendo así que el mecanismo de herencia posibilita la independencia entre el ***propósito*** *de una función y el* ***modo*** *de llegar hasta él.*

Figura 4.4. : Mecanismo de herencia. Ejemplo.

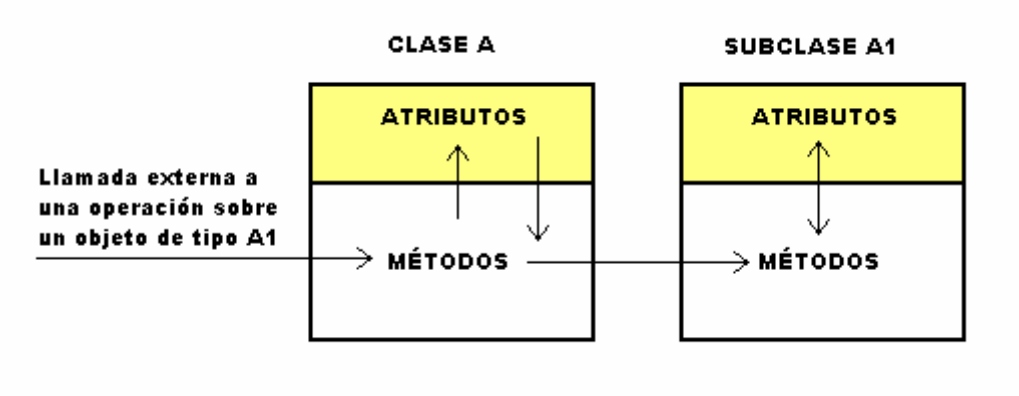

Figura 4.5. : Mecanismo de herencia. Los atributos y métodos de la subclase A1 han sido heredados de los correspondientes en la superclase A

A la propiedad de una operación sobre una clase de trabajar de diferente manera según el tipo de argumento que recibe se le llama **polimorfismo**. En el ejemplo anterior, el polimorfismo estaba integrado en una jerarquía de clases, y por tanto, ligado a la herencia entre las mismas. Pero no tiene por qué ser así.

> *Supongamos ahora que cierto objeto desea crear una malla (objeto de tipo MESH) y una matriz (objeto de tipo MATRIX). Obviamente estas dos operaciones son muy diferentes, y son realizadas por clases diferentes (MESH y MATRIX), sin relación jerárquica entre ellas. Sin embargo, la línea de código que llame a estos métodos puede ser simplemente CREATE para ambos casos. Cuando se quiera crear la matriz se escribe por ejemplo CREATE(matriz1). Como el objeto matriz1 habrá sido declarado previamente bajo el tipo MATRIX, la operación CREATE será realizada por la clase MATRIX, y análogamente para el caso de la malla.*

Figura 4.6. : Ejemplo de polimorfismo

También podría darse polimorfismo cambiando no sólo el tipo de argumentos, sino también el número de los mismos. Veremos en el siguiente apartado que en FORTRAN90 el polimorfismo está ligado a la sentencia INTERFACE.

Existe, además, otra clase de polimorfismo, llamado **polimorfismo en tiempo de ejecución** (*run-time polymorphism, dynamic dispatching*) (Decyk, Norton y Szymanski, 1997 b). Su propósito es construir una sola vez métodos comunes a diferentes subclases en una relación jerárquica de herencia, y evitar la multiplicación innecesaria del código. Sólo es posible cuando los métodos en las diferentes subrutinas realizan funciones análogas y utilizan operadores análogos, declarados bajo el polimorfismo convencional.

Dijimos al principio de este capítulo que uno de los problemas de la programación procedimental es la secuencialidad de las operaciones. En la programación orientada a objeto, esta secuencialidad puede suavizarse mediante el polimorfismo ya citado. Por ejemplo, se desea utilizar un atributo de un objeto, pero no se sabe si ha sido creado o no. Un programa clásico probablemente inicializaría todos los atributos de un objeto, con la expectativa de usarlos después. Pero en caso de no ser así, se estaría desperdiciando una capacidad de almacenamiento. Usando un enfoque orientado a

objeto, la llamada a ese atributo sería atendida por una interfaz que comprobaría antes si el atributo ha sido inicializado previamente, desviando entonces la llamada a la operación correspondiente (crear el atributo o recuperarlo). Esta propiedad de independizar al cliente del contexto temporal (o más bien, secuencial) ha sido llamada **no-anticipación** (Dubois-Pélerin y Zimmermann, 1992).

La ventaja principal que proporciona la programación orientada a objeto, desde el punto de vista del programador, es la claridad en la presentación del programa, que permite distinguir sus elementos sin demasiado esfuerzo. La organización jerárquica que existe entre las clases contribuye a ello. La incorporación de nuevas clases es sencilla, ya que sólo requiere su programación independiente del resto del programa, por una parte; y las declaraciones de uso y llamadas a sus métodos desde otras clases, por otra.

Hasta aquí hemos descrito a grandes rasgos una serie de conceptos, pero quizá de una manera muy abstracta, sin entrar apenas en la realidad de los lenguajes de programación. Debe quedar claro que la implementación de estos conceptos está extraordinariamente ligada al lenguaje usado. Existen lenguajes de programación específicamente diseñados a este efecto (C++, Java...). Sin embargo, dado que el programa PROW ha sido construído en FORTRAN90, en el siguiente apartado se explicará brevemente una manera de implementar (en cierta manera simular) este tipo de programación en el mismo.

4.3. Implementación en FORTRAN90

El lenguaje de programación FORTRAN es (o ha sido) probablemente el más utilizado en el campo científico, y concretamente en el campo que nos ocupa, el del transporte de solutos. Como ya vimos en el capítulo tercero de este documento, algunos de los códigos más utilizados (MODFLOW, MT3D, HST3D) están programados utilizando FORTRAN77. Ésta es una versión bastante antigua del lenguaje, y lo cierto es que se ha desarrollado una ingente cantidad de algoritmos utilizándola. Todos estos programas siguen el paradigma de programación clásico, el sistema por procedimientos, que es terriblemente rígido a la hora de expandir un programa. Además, el código en FORTRAN77 resulta críptico para alguien diferente del programador (si no para el propio programador).

¿Significa esto que todos esos millones de líneas de código pasarán a ser inservibles? En absoluto. FORTRAN, como muchos lenguajes, evoluciona (los lenguajes que no lo hacen... mueren). La versión FORTRAN90 incorpora toda una serie de nuevas ideas, incluidas muchas usadas en la programación orientada a objeto, lo que permite hasta cierto punto simularla. Puesto que FORTRAN90 es compatible con FORTRAN77, es posible utilizar antiguos algoritmos e introducirlos en una estructura general orientada a objeto, donde cumplirán su función exactamente igual que antes. De hecho, como se verá en el capítulo siguiente, PROW fue construido inicialmente utilizando algoritmos del programa TRACONF (programado en FORTRAN77), y disponiéndolos convenientemente en una estructura nueva y mucho más ordenada.

Vamos a ver, pues, las posibilidades que ofrece FORTRAN90 para simular la programación orientada a objeto, para presentar un posible diseño de la misma en este lenguaje.

Una imagen vale más que mil palabras. La siguiente sentencia muestra una llamada a una subrutina de MT3D (programación clásica), que formula los coeficientes de la matriz de dispersión:

```
CALL DSP4FM(NCOL,NROW,NLAY,MCOMP,ICOMP,IX(LCIB),  &
     X(LCDELR),X(LCDELC),X(LCDH),X(LCDXX),X(LCDXY),
     & (LCDXZ),X(LCDYX),X(LCDYY),X(LCDYZ),X(LCDZX),
     & X(LCDZY),X(LCDZZ),X(LCA),NODES,UPDLHS, &
     X(LCCNEW),X(LCRHS),NCRS)
```

La siguiente línea muestra la llamada equivalente en PROW (orientado a objeto):

```
CALL INTGV_DIV_P_GRAD_x_(THIS%pMESH, DISP, THIS%pA)
```

Es posible que el lector, incluso sin conocer PROW, de un vistazo se imagine lo que cada argumento puede representar en la segunda sentencia.
¿Lo puede hacer con la primera?

Figura 4.7. : Comparación entre una sentencia clásica y una en un lenguaje orientado a objeto

4.3.1. Dimensionamiento dinámico

La primera gran ventaja de FORTRAN90 es la posibilidad de declarar y dimensionar matrices durante la ejecución del programa, y no solamente durante la compilación.
Esto tiene dos consecuencias inmediatas:

En general el programa (sea el que sea) no trabaja con modelos siempre de las mismas dimensiones. Lo que quiere decir que debe declarar una serie de matrices de dimensiones deducidas a partir de datos entrados por el usuario (p.ej. a través de un archivo). En FORTRAN77, esto se hacía mediante la declaración de dos inmensos arrays (uno de tipo real y el otro entero), en el que se incluían todas las matrices del programa cuyas dimensiones dependieran del modelo. Estas matrices se declaraban con unas dimensiones estimadas suficientemente grandes para una mayor parte de los modelos (ver figura 4.8.). Si no eran suficiente, entonces se debían ampliar las dimensiones y recomenzar la ejecución. Esto suponía, o bien una dificultad más a la hora de ejecutar el programa, o bien que se reservaba una cantidad de memoria que podía ser mucho mayor que la necesaria.

La posibilidad de dimensionar matrices en plena ejecución, permite reservar para éstas sólo el espacio necesario, optimizando notablemente el uso de la memoria. Esto no es una característica que tenga que ver con la orientación a objeto, pero sí la siguiente.

Una subrutina trabaja con variables, algunas de las cuales son auxiliares, es decir, pueden ser destruidas al terminar la ejecución de dicha subrutina. En caso de que una de esas variables auxiliares fuera una matriz, en FORTRAN77, ésta debía pasar como argumento de la subrutina, ya que debía dimensionarse correctamente durante la

compilación. Utilizando dimensionamiento dinámico, cada subrutina puede crear sus variables auxiliares y destruirlas una vez utilizadas. Esto representa un paso (aunque muy primitivo) hacia el principio de encapsulamiento de la información, ya descrito.

El siguiente comentario aparece en la cabecera del programa principal de MT3D, y muestra las variables que contienen limitaciones sobre el programa, algo típico de FORTRAN77. Los arrays X e IX son los que contendrán toda la información (real o entera).

```
C--SET MAXIMUM ARRAY DIMENSIONS AND FORTRAN UNIT
NUMBERS FOR I/O FILES.
C--LENX AND LENIX: MAXIMUM DIMENSION OF STORAGE
ARRAYS X AND IX;
C--MXPRS:  MAXIMUM NUMBER OF TIMES AT WHICH
RESULTS ARE SAVED;
C--MXSTP:  MAXIMUM NUMBER OF TIME STEPS IN FLOW
MODEL;
C--MXOBS:  MAXIMUM NUMBER OF OBSERVATION POINTS;
C--MXCOMP: MAXMUM NUMBER OF COMPONENTS.
```

Figura 4.8. : Comentario de limitaciones del programa en MT3D

4.3.2. Estructura modular real

Muchos programas escritos en FORTRAN77 simulaban una estructura modular, en la que los módulos no eran más que conjuntos de subrutinas con una cierta relación entre sí. No había, sin embargo, una cohesión en el módulo.

FORTRAN90 permite, mediante un bloque MODULE-END MODULE, agrupar una serie de subrutinas, funciones, parámetros, interfaces, y tipos, sólo disponibles dentro del módulo. Si el módulo A pretende llamar a una subrutina del módulo B, deberá declarar previamente el uso de este módulo mediante la sentencia USE B. Así, todos los datos contenidos en un módulo, sólo son accesibles desde el módulo, excepto que se declare específicamente lo contrario.

Además existe la posibilidad de utilizar sólo ciertos aspectos de un módulo, mediante la instrucción ONLY. Podríamos querer utilizar de un módulo, sólo un tipo derivado que contiene (más abajo se verá lo que es un tipo derivado), pero no llamar a ninguna subrutina del módulo.

Por otra parte, dentro de cada módulo habrá algunas subrutinas públicas (que pueden recibir llamadas externas al módulo), pero habrá otras que sólo el módulo podrá utilizar, que servirán para desarrollar las operaciones internas del módulo (entre las que se podrían encontrar también, subrutinas que provienen de un antiguo código en FORTRAN77 y que han sido incorporadas a una estructura nueva). Estas subrutinas se declararán mediante la palabra clave PRIVATE, que prohíbe su llamada desde fuera del módulo.

4.3.3. Interfaces

La instrucción INTERFACE genera un nombre común a una serie de subrutinas. Las subrutinas citadas dentro de un bloque INTERFACE responderán a cualquier llamada externa por el nombre que se declare tras la instrucción. La ventaja es que, si tenemos varias subrutinas que realizan tareas similares, pero con cantidades o tipos de argumentos diferentes, la llamada externa al módulo no necesita saber a qué subrutina debe llamar. La interfaz, que pertenece al módulo, redirecciona la llamada a la subrutina cuyos argumentos concuerden con los de aquella. La instrucción INTERFACE se convierte así en uno de los instrumentos de FORTRAN90 para representar el polimorfismo, ya explicado.

Una interfaz puede representar no sólo a una serie de subrutinas que actúan sobre un mismo objeto de formas diferentes, sino también a subrutinas que realizan operaciones análogas sobre objetos diferentes. Por ejemplo, en PROW, mediante la interfaz CREATE_(argumentos) se puede crear absolutamente cualquier objeto, dependiendo del tipo de argumentos con los que se la llame.

El polimorfismo en tiempo real, citado en el subcapítulo anterior, no está presente en FORTRAN90, y su simulación requiere la reduplicación de las subrutinas, con posterior declaración conjunta bajo una interfaz. Luego no acaba de ser un principio que aporte nada nuevo a lo visto.

4.3.4. Tipos derivados.

FORTRAN contiene varios tipos de datos intrínsecos (INTEGER, REAL, LOGICAL, etc.) con los que pueden operar diversas funciones y operadores definidos en el lenguaje. Además, en FORTRAN90 es posible crear tipos definidos por el usuario, o **tipos derivados**. Un tipo derivado está compuesto de varios tipos (pudiendo ser éstos intrínsecos, o derivados a su vez). Luego una variable declarada como tipo derivado se compone de un conjunto de variables de diferentes tipos predefinidos, a las que se llama componentes. La instrucción para declarar tipos derivados en FORTRAN90 es TYPE.
En la figura 4.9. se muestra un ejemplo de declaración de un tipo.

```
TYPE T_MESH1

     INTEGER*4 :: index

     REAL*8, POINTER :: data(:)

     TYPE(T_MATRIX2), POINTER :: modii

END TYPE T_MESH1
```

Figura 4.9. : Ejemplo de declaración del tipo MESH1. Consta de tres componentes: una variable entera, un array real, y un puntero a un objeto de tipo MATRIX2, perteneciente a un módulo ajeno.

En general, un tipo sirve para agrupar una serie de datos de diferentes tipos intrínsecos, pero relacionados con un mismo concepto: una malla, una matriz... en definitiva, un objeto cualquiera. Esta herramienta tan útil se convierte así en la materialización de los atributos de un objeto, y por tanto llamaremos *objeto* de tipo X a una variable declarada como tipo X.

4.3.5. Posible diseño de un sistema orientado a objeto

Vamos a ver cómo utilizar las posibilidades que nos ofrece FORTRAN90 para simular la programación orientada a objeto. Pretendemos, antes de nada, definir una clase: MESH. Ya hemos dicho que una clase contiene atributos y métodos, así que encerremos esos atributos y esos métodos en un módulo, el módulo MESH. Así, podemos establecer un paralelismo entre módulo y clase.

¿Cómo caracterizamos los atributos de la clase? Definamos un tipo derivado, cuyas componentes sean de los tipos (intrínsecos o derivados a su vez) correspondientes a los atributos. Llamémosle T_MESH, donde T significa tipo derivado.
Está claro que los métodos de la clase deberán ser subrutinas y funciones incluidas en el módulo. Pero recordemos que los métodos de una clase actúan *sobre objetos de esa misma clase*. Por ejemplo supongamos la subrutina METHOD1_MS, donde MS es una abreviatura de MESH. ¿Cuáles serán los argumentos de esta subrutina? Entre ellos habrá uno que represente al objeto de la clase MESH, luego será una variable de tipo T_MESH. ¿Cómo llamar a esta variable? Parece lógico que, dentro de una misma clase, el objeto sea tratado como algo propio. Así que le llamaremos THIS.

Hasta aquí hemos caracterizado una clase aislada. Pero esta clase debe interactuar con otras, por ejemplo llamando a sus subrutinas. Una vez sepamos con qué clases se relacionará, incluiremos las sentencias USE correspondientes a sus módulos. Por otra parte, al llamar a un método de una clase diferente, deberemos pasar como argumento el objeto sobre el que actúa. Parece lógico, entonces, incluir dentro del tipo derivado de nuestra clase una componente que sea un puntero a dicho objeto, con un nombre suficentemente descriptivo del mismo.
Además, todos los métodos públicos de nuestra clase deberán disponer de una interfaz sencilla. Por ejemplo, supongamos el método READ_MS. Interesa que disponga de una interfaz que se llame simplemente READ_, donde utilizamos el guión bajo pospuesto para indicar que se trata de una interfaz (esta es la costumbre seguida en PROW, cada uno que lo haga a su manera).
Posteriormente, declararemos todas las interfaces mediante la instrucción PUBLIC, y las subrutinas o funciones mediante PRIVATE.
Para completar el módulo podríamos necesitar declarar parámetros, como por ejemplo los nombres de las subclases disponibles.

En la figura 4.10. se muestra un esquema de un módulo tipo siguiendo el diseño expuesto.

Hay que tener en cuenta que si una determinada clase contiene varias subclases (por ejemplo, las matrices que pueden ser diagonales, en banda, etc), esta *superclase* contendrá básicamente métodos públicos. Cada uno de estos métodos llamarán al método descendiente (por el mecanismo de herencia) de la subclase a la que pertenece

el objeto sobre el que se quiere actuar, es decir, el que se pasa como argumento. El método descendiente, a su vez, podría redireccionar el flujo del programa hacia otros métodos descendientes, o bien realizar las operaciones necesarias por sí mismo.

```
MODULE CLASS

Comentarios diversos: nombre y abreviatura (CL), propósito de la clase,
métodos públicos disponibles, etc.

Declaraciones de uso de diferentes módulos, entre ellos los que representan
subclases:
USE class1
USE class2
...

Declaraciones de métodos públicos y privados:
PUBLIC :: &
      METHOD1_, &
      METHOD2_,&
      ...
PRIVATE :: &
      METHOD1_CL, METHOD2_CL, ..., &
      METHOD7, METHOD8...

Declaración de tipos derivados:
TYPE T_CLASS
      declaración de componentes
END TYPE T_CLASS

Declaración de parámetros (si los hay)
Declaración de interfaces:
INTERFACE METHOD1_
      MODULE PROCEDURE METHOD1_CL
END INTERFACE
...

Código de subrutinas y funciones (preferiblemente primero las públicas y
luego las privadas):
CONTAINS

SUBROUTINE METHOD1_CL (THIS, otros argumentos)
      IMPLICIT NONE
      TYPE(T_CLASS) :: THIS
      Resto de declaraciones

      Código de la subrutina
      RETURN
END SUBROUTINE
...

END MODULE CLASS
```

Figura 4.10. : Diseño tipo de una clase en FORTRAN90

5. EL PROGRAMA PROW. DESCRIPCIÓN Y ESTRUCTURA.

5.1. Motivación

La estructura básica del programa PROW fue desarrollada en la sección de Hidrología Subterránea del Departamento de Ingeniería del Terreno y Cartográfica, de la UPC. La idea era utilizar la programación orientada a objeto para resolver problemas en medios porosos, tanto flujo como transporte, procesos químicos, etc.
Se partió de un programa ya existente, TRACONF (Carrera, Galarza y Medina, 1993), programado en FORTRAN77 con técnicas clásicas (*procedural programming*), y se seccionó de manera correcta, generando módulos para los diferentes bloques del programa. Así, se agruparon todas las subrutinas que trataban con la malla en un módulo, y lo mismo con las matrices, el flujo, el transporte... Y se creó una estructura de clases tal y como ha sido descrita en el capítulo anterior.
TRACONF es un programa de resolución únicamente mediante elementos finitos. Así que, partiendo de la base de que la programación orientada a objeto facilita el crecimiento, se pensó en introducir los métodos de resolución que actualmente utilizan programas tan difundidos como MT3D, MODFLOW, etc.
En este apartado se describirá el PROW, y la manera en que se introdujeron tres nuevos métodos de resolución: el método de diferencias finitas, el MOC y el MMOC.

5.2. Estructura del programa

PROW consta de los siguientes elementos:

- el programa principal MAIN
- el módulo de lectura RENG
- una serie de módulos, que representan clases diversas (una malla, una matriz, un intervalo de tiempo…), los cuáles serán descritos uno por uno más adelante.
- una serie de archivos externos, que contienen datos como las instrucciones de ejecución o los datos de entrada.

Veamos a continuación la descripción de cada uno de ellos.

5.2.1. MAIN y RENG

Ya hemos visto en el anterior capítulo que los objetos que componen un sistema como PROW son razonablemente independientes, y que se van llamando unos a otros. Pero tiene que existir algún elemento que inicie la secuencia de procesos. Éste es el programa principal MAIN.

El objetivo principal de MAIN es crear, manipular, y posteriormente destruir el primer objeto: el objeto ENGINE o motor del programa. Se puede encontrar el listado de

MAIN en el anejo A. La secuencia de operaciones que realiza es esencialmente la siguiente:

1. Declaración de uso del módulo ENGINE
2. Declaración de variables auxiliares
3. Carga del archivo ROOT
4. Lectura del tipo de ENGINE
5. Creación del objeto ENGINE
6. Lectura de atributos de ENGINE
7. Ejecución del objeto ENGINE
8. Destrucción del objeto ENGINE
9. Descarga del archivo ROOT
10. Fin del programa

Hay diversas operaciones que deben hacerse de manera mecánica en muchos puntos del programa, como por ejemplo determinar el tipo de archivo que va a leerse, o el nombre del mismo, etc. Para ello, se ha construido el módulo de lectura RENG (Read ENGine). Este módulo no es más que un conjunto de subrutinas y funciones útiles para el programa, y no contiene ningún objeto en el sentido que tratamos en el capítulo anterior. Por ejemplo, el punto 3 de la lista de tareas de MAIN es una llamada a la subrutina LoadEng, contenida en el módulo RENG, en la que se lee el archivo ROOT.ENG, del que hablaremos en el punto 5.2.6. Este archivo contiene los parámetros necesarios para la ejecución del programa (nombres de los archivos de datos, tipos de mallas, ecuaciones...), pero no datos del modelo concreto que se va a resolver. Estos datos estarán contenidos en otros archivos, de los que hablaremos en el punto 5.2.5.

5.2.2. Clases principales de PROW

Una vez creado el objeto ENGINE, éste es realmente el módulo que ejecutará la lista de procesos necesaria para leer los datos, resolver las ecuaciones y escribir los resultados, a base de ir creando y manipulando los objetos necesarios a este efecto.
Pero, con qué clases va a contar ENGINE para trabajar? La figura 5.1 muestra las diferentes clases (superclases) de las que dispone PROW.

ENGINE: Organiza los objetos en el orden necesario
MATRIX : Gestiona matrices
MESH: Gestiona la malla (o discretización espacial)
TIMEDISCR: Gestiona la discretización temporal
FLOW: Gestiona las ecuaciones de flujo
TRANSPORT: Gestiona las ecuaciones de transporte
WOUT: Realiza la escritura de resultados
TIMEINTG: Realiza la integración temporal

Figura 5.1. : Clases públicas de PROW

Las clases que aparecen en dicha figura son aquellas que pueden ser llamadas directamente desde un módulo ajeno. Cada una de estas clases dispone de punteros a una serie de objetos pertenecientes a las subclases sobre las que domina. Por ejemplo, la clase MESH contiene punteros a las clases MS_FEM2DLTRI, MS_FDM2D y MS_PARENT, que representan, respectivamente: malla de elementos finitos triangulares lineales 2D, malla de diferencias finitas 2D, y parámetros comunes a todas las mallas. Podría decirse que la clase PARENT es una clase auxiliar.

Se puede observar que hay dos clases relacionadas con el tiempo, TIMEDISCR y TIMEINTG. A pesar de ello, desempeñan funciones diferentes: la primera se ocupa de gestionar el valor de los incrementos de tiempo de cálculo, y el instante de ejecución. La segunda, sin embargo, realiza la integración del sistema lineal de ecuaciones, conocida habitualmente como integración temporal (aunque en un caso estacionario se realice una sola vez).

Naturalmente, las diferentes clases pueden ser usadas unas por otras, y sus subrutinas llamadas desde cualquier subrutina perteneciente a otra clase. Así, la clase TRANSPORT puede necesitar operar sobre un objeto MATRIX, por lo que tendrá activado el uso del módulo MATRIX, y dispondrá en su tipo derivado de un puntero al objeto de tipo MATRIX en cuestión, tal y como explicamos anteriormente.

La figura 5.2. muestra la organización de clases y subclases en PROW.

Como se puede ver, hay muchas clases que actualmente sólo disponen de una subclase. En muchos casos, ésta lleva el afijo TRACONF. Estas clases son el resultado de la transformación de TRACONF de programación clásica a orientada a objeto. Para la realización de este documento, he realizado básicamente la clase MT_FDM2D, y una gran ampliación del módulo TR_TRACONF. De esta ampliación, resultado de la inclusión de los métodos de partículas, hablaremos más adelante.

Para ilustrar el flujo del programa, veamos la figura 5.3. Esta figura muestra un diagrama de tipo “Rational Rows”, muy utilizado en la programación orientada a objeto. Las columnas representan clases y las flechas horizontales, llamadas a métodos de otras clases (línea continua, llamada; línea discontinua, respuesta). El tiempo de ejecución crece a medida que descendemos por el diagrama. Sólo hemos representado las clases públicas en la figura. Naturalmente, cada llamada a una clase pública va siempre seguida de una llamada al mismo método en la subclase a la que pertenece el objeto argumento. El diagrama corresponde a un paso de tiempo en la resolución de un problema de transporte transitorio con flujo estacionario, dentro del bucle de iteración temporal del método RUN de ENGINE.

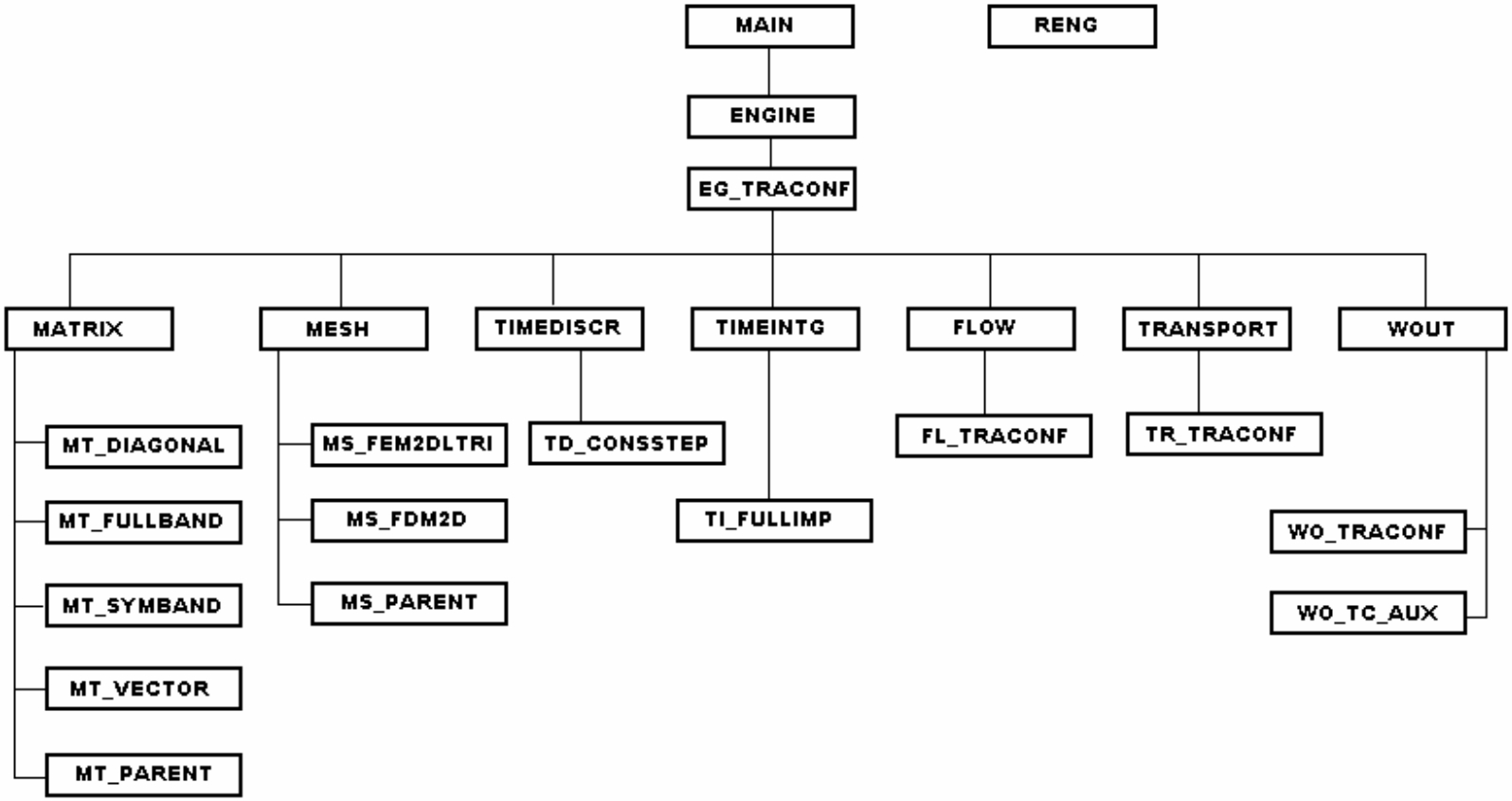

Figura 5.2. : Estructura de clases de PROW

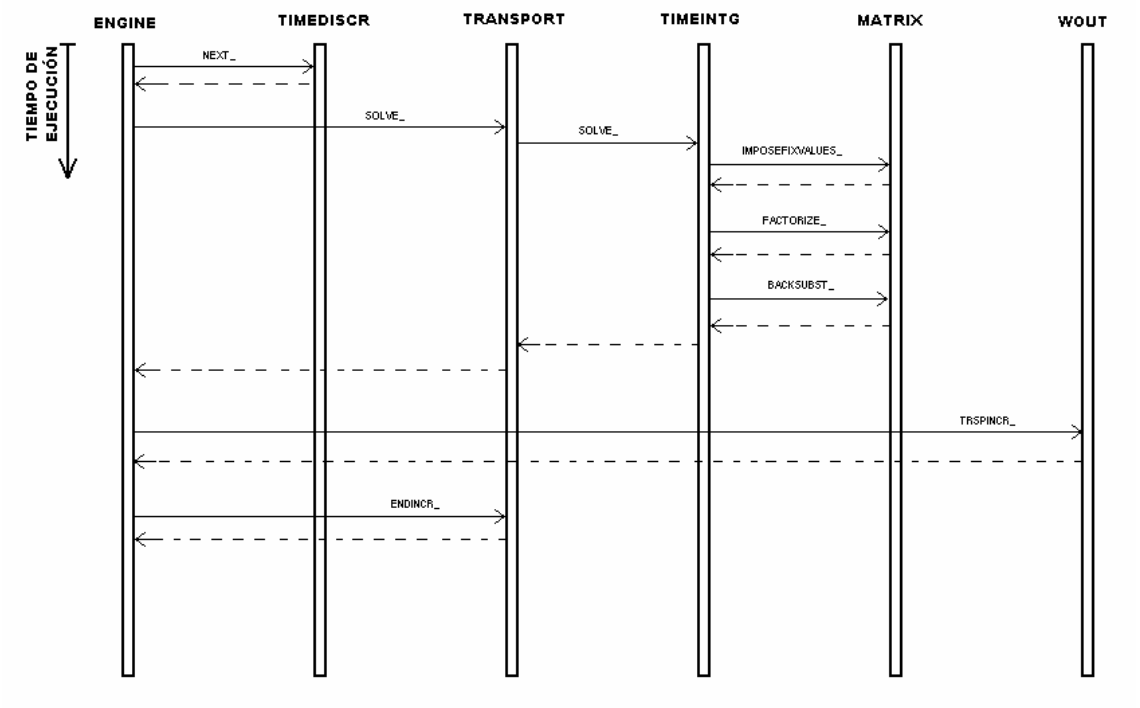

Figura 5.3. Diagrama rational rows de un paso de tiempo en la resolución del transporte con flujo estacionario

Ahora veremos una descripción de cada una de las clases públicas: su objetivo y las operaciones que pueden realizar sobre sus objetos. Antes, decir que todas ellas siempre disponen de unas operaciones básicas y fundamentales para manejar los objetos, que son las siguientes:

- CREATE: Crea el objeto deseado, lo que supone inicializar todas las componentes de su tipo.
- DESTROY: Destruye el objeto deseado, desalocatando todas sus variables, y haciendo nulos los punteros.
- READ: Lee datos necesarios para crear los atributos del objeto. Puede hacerlo desde disco o de la manera deseada. Únicamente la clase MATRIX no dispone de este método, simplemente porque no ha sido necesario hasta ahora. Podría haber un nuevo ENGINE que necesitara disponer de él, y no supondría ningún problema implementarlo.

5.2.2.1. Clase ENGINE: gestión del flujo del programa

El objeto (o los objetos) de tipo ENGINE son los responsables de la ejecución ordenada de los procesos necesarios para resolver el problema. Esto lo hace mediante la creación y realización de operaciones sobre objetos de otras clases. Él es el responsable de las llamadas a módulos para la creación de las mallas y discretizaciones temporales, del manejo de las ecuaciones de transporte y flujo, de la resolución del sistema resultante y de la escritura (o gestión alternativa) de resultados.

En PROW sólo está implementada la subclase EG_TRACONF, que es el motor del programa TRACONF, adaptado a esta estructura.

El tipo derivado T_ENGINE dispone de un puntero hacia el tipo T_EG_TRACONF.

Aparte de las operaciones básicas ya citadas, el objeto ENGINE sólo puede realizar una operación, fundamental, eso sí:

- **RUN**: ejecuta ENGINE.

5.2.2.2. Clase MATRIX: gestión de matrices

Los objetos de tipo MATRIX, como su nombre indica, contienen matrices. Entre sus atributos se encuentran tanto los valores de sus componentes como todas las características que pudieran ser necesarias para realizar alguna operación: tipo de almacenamiento, número de elementos, ancho de banda, etc.

En general, las subclases de MATRIX representan matrices almacenadas según diferentes esquemas. Hasta ahora, MATRIX puede trabajar con matrices en banda (simétricas o no), diagonales y vectores (matrices nx1). Existe una subclase (WATSOLV) para trabajar con matrices dispersas de grandes dimensiones, pero no ha sido incorporada aún al código.

Los módulos que utiliza MATRIX son los correspondientes a las subclases, es decir, los siguientes:

- **M_MT_FULLBAND**: Gestiona matrices en banda (con cualquier simetría).
- **M_MT_SYMBAND**: Gestiona matrices simétricas en banda.
- **M_MT_DIAGONAL**: Gestiona matrices diagonales.
- **M_MT_VECTOR**: Gestiona vectores (matrices nx1).
- **M_MT_PARENT**: Contiene parámetros comunes a todas las subclases.

Más adelante se explicará el porqué de los módulos PARENT.

El tipo derivado (T_MATRIX) contiene punteros a los tipos de cada una de las subclases.

MATRIX puede operar sobre sus objetos con los siguientes métodos (aparte de los básicos ya comentados):

- **DUMP**: Escribe el contenido de la matriz en un archivo.
- **RESET**: Pone a cero los elementos de la matriz.
- **COPY**: Copia los elementos de la matriz a otra del mismo tipo.
- **INDXD_ADD**: Suma una submatriz (subvector, o simplemente un valor) a las componentes indicadas.
- **CALC_DIAG**: Calcula los términos de la diagonal, según ciertos criterios.

- **VANISH_COLROW**: Hace nulas la columna y la fila en la que se encuentra una determinada componente.
- **ADD2DIAG**: Suma un valor a la diagonal.
- **AddMxf**: Suma una matriz multiplicada por el escalar f
- **MxV**: Realiza el producto matriz-vector.
- **FACTORIZE**: Factoriza una matriz.
- **BACKSUBST**: Realiza la substitución hacia adelante y hacia atrás de una matriz (si es posible).
- **SOLVE**:
- **CAN_FACTORIZE**: Determina si una matriz puede ser o no factorizada.
- **GET_DIM**: Devuelve las dimensiones de la matriz.
- **SET_ITEM**: Introduce un valor en una posición determinada.
- **GET_ITEM**: Recupera un valor de una posición especificada.
- **pVEC**: puntero a una matriz nx1.
- **DEASSVEC**: Desasigna el puntero anterior.
- **HASCHANGED**: Determina si la matriz ha cambiado o no.
- **SETHASCHANGED**: Asigna el valor TRUE a HASCHANGED.
- **ISDIAGONALSUM**: Indica si el término de la diagonal es suma del resto de términos de la fila.
- **SETDIAGONALSUM**: Asigna el valor TRUE a ISDIAGONALSUM.

5.2.2.3. Clase MESH: gestión de mallas

Los objetos de tipo MESH representan mallas y, en general, todo lo que hace referencia al tratamiento del espacio en el problema. La clase MESH realiza toda una serie de operaciones físicas y matemáticas como cálculo de volúmenes y áreas, integraciones e interpolaciones de campos, disposición y movimiento de partículas, etc.
Hasta ahora, PROW trata con dos tipos diferentes de mallas: elementos finitos triangulares lineales, y diferencias finitas.

Así, los módulos usados por MESH se reducen a sus propias subclases, más un módulo PARENT, a semejanza del utilizado por MATRIX. Son los siguientes:

- **M_MS_FEM2DLTRI** : Gestiona mallas de elementos finitos triangulares lineales en dos dimensiones.
- **M_MS_FDM2D** : Gestiona mallas de diferencias finitas en dos dimensiones. Esta subrutina ha sido desarrollada especialmente para realizar esta tesina. El listado del código en FORTRAN90 se puede encontrar en el anejo B.
- **M_MS_PARENT** : Contiene parámetros comunes a las dos anteriores subclases.

Además, MESH hace uso del módulo M_RENG (engine de lectura), para operaciones como averiguar qué archivos debe leer, etc.

El tipo derivado (T_MESH) contiene punteros a las dos mallas existentes.

Las operaciones posibles son:

- **INIT**: Inicializa valores de la malla, como por ejemplo, las conectividades entre diferentes conjuntos de puntos (puntos de velocidad, nodos…)
- **pACTH**: es un puntero a la variable ACTH, que representa el espesor de acuífero.
- **DEASSACTH**: desasocia el puntero pACTH
- **INTGV_DIV_P_GRAD_x**: Integra en el dominio de forma abstracta la divergencia del campo P por el gradiente de x, construyendo la matriz resultante según el tipo de malla (elementos finitos, diferencias finitas).
- **INTGV_ P_x**: Integra el campo tensorial P por x en el dominio.
- **INTGV_P_GRAD_x**: Integra el campo vectorial P por el gradiente de x, en el dominio.
- **INTGB_F_x_MINUS_x0**: Integra sobre el contorno el producto $F\cdot(x-x_0)$.
- **INTGB_F**: Integra sobre el contorno el campo F.
- **P_GRAD_s**: Devuelve el producto del campo tensorial P por el gradiente de s (utilizado, por ejemplo, para calcular la velocidad del flujo, a partir de K y h)
- **INTERP_ELEM2VP**: Interpola de elementos a puntos de velocidad.
- **ELEM_VOL**: Calcula el volumen de un elemento.
- **PART_LAY**: Dispone una determinada cantidad de partículas siguiendo diferentes patrones de disposición.
- **MOVE_POINT**: Mueve una partícula según un vector y devuelve la posición final.
- **PART2ELEM**: Interpola entre las posiciones de un conjunto de partículas dado, y el conjunto de los elementos.
- **PART_UPDATE**: Actualiza un conjunto de partículas.

5.2.2.4. Clase TIMEDISCR: discretización temporal.

La clase TIMEDISCR se ocupa de objetos que contienen datos relativos al tiempo en el problema. Controla, por ejemplo, el tipo de intervalo temporal (constante o variable), el instante en que se encuentra la ejecución en cada momento, cuándo debe llamar a la escritura de resultados, etc. En suma, todo lo relativo a la discretización temporal.

De momento, PROW sólo puede trabajar con intervalos de tiempo constantes, así que TIMEDISCR trabaja únicamente con la subclase M_TD_CONSSTEP, aparte del módulo M_RENG de lectura.

Su tipo derivado (T_TIMEDISCR) contiene un puntero a la discretización temporal T_TD_CONSSTEP

TIMEDISCR puede realizar las siguientes operaciones sobre la discretización temporal:

- **INIT**: Inicializa el tiempo.
- **RESET**: Reinicia el contador de tiempo.
- **NEXT**: Continua con el siguiente incremento de tiempo.
- **ISLASTTIME**: Determina si nos encontramos al final del tiempo de resolución.

- **GET_INT**: Obtiene el valor del incremento de tiempo.
- **GET_TIME**: Obtiene el valor del tiempo absoluto.
- **GET_DT**: Devuelve el valor del incremento de tiempo DT.

5.2.2.5. Clase FLOW: ecuaciones de flujo en medio poroso.

Los objetos FLOW contienen los parámetros de las ecuaciones de flujo en medio poroso. A partir de estos parámetros es posible construir la matriz de flujo, el campo de velocidad, calcular el balance de masa, decidir qué condiciones de contorno deben imponerse.

Dado el origen de PROW (el programa TRACONF), FLOW contiene una sola subclase, que es M_FL_TRACONF. Esta subclase realiza las operaciones citadas de la misma manera en que las hacía TRACONF. Además, FLOW puede utilizar el módulo M_RENG de lectura.

El tipo derivado (T_FLOW) contiene un puntero hacia T_FL_TRACONF.

Los siguientes métodos están disponibles sobre objetos FLOW:

- **INIT**: Inicializa las variables de flujo.
- **BUILD**: Construye las matrices de flujo e impone condiciones de contorno.
- **SOLVE**: Llama a la resolución de la ecuación de flujo.
- **ENDINCR**: Termina el incremento de tiempo de flujo.
- **COMVEL**: Calcula el campo de velocidades.
- **COMFLW**: Calcula los valores nodales del flujo.
- **COMBAL**: Calcula el balance de masa del flujo.
- **pH**: Puntero a los valores de altura piezométrica.
- **DEASSH**: Desasigna pH.
- **pMESH**: Puntero a la malla en uso.
- **DEASSMESH**: Desasigna pMESH.
- **ISTRANSIENT**: Determina si el flujo es transitorio o no.

5.2.2.6. Clase TRANSPORT: ecuaciones de transporte en medio poroso.

De una manera completamente análoga a FLOW, TRANSPORT gestiona las ecuaciones de transporte de solutos. Como se puede ver, su estructura es paralela a la de la clase anterior, y realiza exactamente las mismas operaciones, pero sobre las ecuaciones de transporte.

Asimismo, sólo dispone de la subclase TR_TRACONF. Dentro de esta subclase, se ha desarrollado todo un bloque de código, responsable de la gestión de los métodos de partículas. El código de la subclase TR_TRACONF se puede encontrar en el anejo C.

Operaciones posibles:

- **INIT**: Inicializa las variables de transporte
- **BUILD:** Construye las matrices de transporte
- **SOLVE**: Llama a la resolución de las ecuaciones de transporte.
- **ENDINCR**: Termina el incremento de tiempo para el transporte.
- **COMBAL**: Calcula el balance de masa.
- **pC**: Puntero a las concentraciones nodales.
- **DEASSC**: Desasigna pC.
- **pFLOW**: Puntero al objeto de flujo en uso.
- **DEASSFLOW**: Desasigna pFLOW.
- **pMESH**: Puntero a la malla en uso.
- **DEASSMESH**: Desasigna pMESH.
- **ISTRANSIENT**: Determina si el transporte es transitorio.

5.2.2.7. Clase WOUT: gestión de los resultados.

Los objetos de tipo WOUT contienen información relativa a los resultados de un modelo: evolución de procesos en el tiempo y en el espacio, y escritura o almacenamiento adecuado de los datos.
WOUT dispone de una subclase, que escribe los resultados en un archivo, tal y como lo hace el programa TRACONF. Ésta es M_WO_TRACONF.
Es perfectamente posible incorporar nuevas subclases WOUT que almacenen los resultados en un archivo tratable por algún programa comercial de postproceso, o por otro programa de cálculo que los utilice, aunque de momento no ha sido llevado a cabo.

Estas son las operaciones disponibles sobre objetos WOUT:

- **FLOWINI**: Escribe los resultados de flujo en el instante inicial.
- **FLOWINCR**: Escribe los resultados de flujo para un cierto incremento de tiempo.
- **FLOWEND**: Escribe los resultados de flujo en el instante final.
- **TRSPINI**: Escribe los resultados de transporte en el instante inicial
- **TRSPINCR**: Escribe los resultados de transporte para un cierto incremento de tiempo.
- **TRSPEND**: Escribe los resultados de transporte en el instante final.
- **ENDWRITE**: Termina el proceso de escritura.

5.2.2.8. Clase TIMEINTG: integración temporal de ecuaciones.

El objetivo de la clase TIMEINTG es realizar la integración temporal de las ecuaciones construidas por otras clases (por ejemplo FLOW y TRANSPORT).
TIMEINTG contienen de momento una sola subclase, que corresponde a la integración totalmente implícita en el tiempo: M_TI_FULLIMP. Está desarrollándose una subclase

que integra según un esquema general en el tiempo (un parámetro θ controla si la integración es totalmente implícita, o explícita, o bien un caso intermedio).

Las operaciones disponibles son:

- **SET**: Establece los atributos
- **SET_FIX_VALUE**: Fuerza un valor fijo.
- **RESET_FIX_VALUES**: Restablece los valores fijos.
- **SOLVE**: Resuelve una ecuación (por ejemplo, flujo o transporte)
- **SET_pOLD**: Crea un puntero al valor de la variable de estado en el instante anterior.

5.2.3. Subclases. Introducción del método de diferencias finitas.

Ya conocemos las clases de objetos con las que puede trabajar PROW. Cada una de ellas posee unos atributos, contenidos en el tipo derivado. Sin embargo, hemos visto que sus tipos derivados esencialmente contienen una serie de punteros a otros objetos. Cada uno de estos punteros apunta a un objeto de una subclase diferente. Naturalmente, sólo contendrá información el puntero a la subclase asignada al objeto.

Los atributos del objeto se encuentran realmente en el tipo derivado de la subclase a la que pertenece. Estos atributos pueden ser radicalmente diferentes para cada subclase. Esto permite englobar objetos muy diferentes, pero a la vez análogos, en una misma clase.

Sin embargo, las operaciones contenidas en la clase superior deben existir necesariamente en cada una de las subclases. Esto es, naturalmente, porque pueden ser llamadas por cualquier objeto ajeno, que no tiene por qué conocer la subclase del objeto sobre el que pretende operar. Esto, algunas veces puede representar que una determinada subrutina de una subclase no haga absolutamente nada. Sin embargo, debe existir por completitud del programa, ya que puede ser llamada.

Por ejemplo, consideremos las dos mallas existentes en PROW, elementos finitos triangulares y diferencias finitas. En cada una se definen conjuntos de objetos característicos; en MEF, nodos y elementos; en MDF, celdas y puntos de velocidad. Sin embargo, la subclase PARENT común a las mallas, contiene entre sus argumentos tres valores que representan el número de: nodos, elementos y puntos de velocidad. ¿Qué relación guardan con cada una de las subclases de mallas existentes? En elementos finitos, la velocidad se da en elementos, luego se hace una correspondencia entre elementos y puntos de velocidad. En diferencias finitas, se ha convenido hacer la correspondencia directa entre celdas, nodos y elementos. Los puntos de velocidad son los centros de las aristas de la retícula. Así, por ejemplo, una subrutina de interpolación entre nudos y elementos es trivial en la subclase de diferencias finitas, ya que allí estos dos conjuntos son el mismo.

Este ejemplo es una de los problemas que surgieron durante la introducción de la subclase M_MS_FDM2D, que gestiona mallas de diferencias finitas. ¿Cómo se introdujo? Bien, se utilizó mucho la analogía con el módulo M_MS_FEM2DLTRI, de elementos finitos triangulares. Inicialmente se construyeron todas las subrutinas

públicas de que disponía la clase MESH, y que podían ser llamadas por módulos externos. Cada una de estas subrutinas recibía unos argumentos, algunos de los cuales eran dados, y otros resultados que debía devolver. Así que todo consistía en programar operaciones análogas que generaran esos resultados.

La malla de diferencias finitas que se implantó tiene claras diferencias con la de elementos finitos, pero se puede hacer fácilmente una analogía entre las dos. Sin embargo, nos aparece el problema, que ya hemos comentado, de los puntos característicos de las mallas. Dado que estos números pertenecen a la subclase auxiliar PARENT, todas las mallas deben tratar con ellos exactamente en la misma forma.

Los tres conjuntos de puntos representan, genéricamente, lo siguiente:

- Nodos: Puntos donde se dan los resultados (alturas, concentraciones)
- Elementos: Entes que están compuestos de un material
- Puntos de velocidad: Puntos en los que el programa calcula la velocidad.

El paralelismo que se hizo entre ellos fue el siguiente (figura 5.4):

	Diferencias finitas	**Elementos finitos**
Nodos	Celdas	Nodos
Elementos	Celdas	Elementos
Puntos de velocidad	Puntos medios de aristas	Elementos

Figura 5.4. : Paralelismo entre parámetros de las mallas de PROW

Hecha esta correspondencia, solamente hubo que programar operaciones análogas a las de elementos finitos. Ya hemos mencionado la trivialidad de algunas subrutinas de interpolación entre conjuntos de puntos característicos.
Hay que decir también que, en general, las subrutinas que realizan las operaciones necesarias para generar el resultado no son las subrutinas públicas de la subclase. Son subrutinas privadas a las que sólo se puede acceder desde el interior de la subclase, y que se declaran específicamente como privadas en la cabecera del módulo. Solamente algunos métodos muy sencillos o triviales, no llaman a una subrutina privada.

Pero, ¿por qué algunos parámetros de la malla pertenecen a la subclase PARENT? Bien, esto es, en cierta manera, una violación controlada de las reglas de la programación orientada a objeto (cosa que podemos hacer utilizando FORTRAN90, pero no los lenguajes específicamente de este tipo). Algunas clases, como FLOW y TRANSPORT, trabajan **directamente** con el número de nodos, o el número de elementos. Esto, naturalmente, no respeta el principio del encapsulamiento de la información, pero de no hacerlo así, habría resultado un exceso de métodos poco generales en la clase MESH. A veces, un pequeño sacrificio de las reglas, genera una estructura más clara.

La razón por la que estos parámetros se separaron de las subclases de MESH es para no producir algo peor: **la duplicación de la información**. Duplicar la información puede dar lugar a errores, si no se actualiza correctamente en todos los puntos del programa. Por otra parte, no se incluyen directamente en la clase MESH, puesto que las subclases

de MESH pueden requerir su uso, y ello significaría que una subclase llama a un método de la clase de orden superior a la que pertenece, cosa que la jerarquía no debería permitir (aunque en FORTRAN90 es perfectamente posible).

Como se ve, hay que buscar un diseño suficientemente general en los métodos disponibles para cada objeto, a fin de poder incorporar fácilmente una nueva subclase posteriormente. Ya dijimos antes que en la programación orientada a objeto, un diseño pobre de un principio puede ser muy difícil de corregir posteriormente. Un diseño general supone **incorporar sólo aquellas operaciones que sean independientes de cómo se describe el objeto**. Por ejemplo, si el objeto representa una entidad matemática, como una matriz, pensar, como métodos públicos, sólo en operaciones matemáticas que se puedan realizar sobre una matriz cualquiera (y no sobre un tipo concreto de matriz). Por supuesto, cada subclase contendrá, además, todas las subrutinas y funciones auxiliares que sean necesarias para ejecutar los métodos públicos requeridos.
En la práctica, sin embargo, no siempre es fácil decidir que operaciones deben incorporarse a cada objeto. El criterio citado es el que se ha intentado seguir para la construcción de PROW.

Durante la construcción de las subclases aparece además otro problema de diseño. Sabemos que el tipo asociado a la subclase debe contener los atributos de los objetos que representa. Estos atributos, sin embargo, no son únicos. Algunos son deducibles a partir de otros, por tanto podríamos crear un método privado que los obtuviera cuando fuera necesario. Dado que la información relativa a un objeto sólo puede ser tratada por los métodos de su subclase, estos atributos derivados siempre podrían estar disponibles a partir de las componentes del tipo. La disyuntiva está entre elegir un mayor gasto de memoria, o un uso más frecuente del procesador. Para simulaciones pequeñas, los ordenadores personales actuales son adecuados en un sentido y en otro. Pero para grandes modelos, el programa debería adaptarse a las características del ordenador a usar.
En PROW se ha intentado equilibrar la proporción memoria-procesador. Sin embargo, en determinadas ocasiones, no se ha dudado en preferir el abuso de la memoria, antes que una cadena excesivamente larga de llamadas a subrutinas.
Desde luego, lo que nunca se debería hacer es no almacenar en memoria una información que, llegado el momento, deberá leerse desde disco. La lectura de disco se hace mediante el método READ, del que casi todas las clases disponen. Este método es ejecutado por ENGINE en un estadio previo al uso de las variables para la construcción de matrices y otras estructuras de datos necesarias para la solución del problema.

5.2.4. Introducción de los métodos de partículas MOC y MMOC

Los métodos de partículas que, como ya se explicó en el tercer capítulo, son métodos mixtos euleriano-lagrangianos, se basan en resolver la advección utilizando un sistema de partículas móviles en el dominio, y sobre esta advección resuelta, efectuar la dispersión y el resto de procesos que se den en el sistema. Estos últimos procesos se realizan mediante otro método común, como diferencias o elementos finitos.

Por tanto, en esencia, es aprovechable la estructura ya existente en el programa para incorporar los métodos de partículas. Lo único que se ha hecho es hacer un bypass de la llamada en la clase TRANSPORT a la subrutina que calcula las componentes de la matriz de advección (en la clase MESH), y desviar esta llamada hacia otra subrutina

específica de los métodos de partículas. Ésta se ocupa de realizar el movimiento de las partículas y controlar su número y masa portada, para extrapolar al final la concentración "intermedia" típica de estos métodos (como ya se dijo en el capítulo 3). Desde esta concentración intermedia, el programa realiza el flujo normalmente, pero sin utilizar ya matriz de advección alguna.

Los atributos relativos a las partículas (posición, masa, elemento/celda que ocupan, etc.) están almacenados en el tipo T_TR_PARTICLES, perteneciente a la clase TR_TRACONF. Y la pregunta es: ¿por qué las partículas no forman clase propia, dado que disponen de atributos propios y operaciones propias?

Las operaciones realizadas sobre las partículas, en esencia, lo son mediante subrutinas de MESH (movimiento de partículas, interpolación de concentraciones…), lo que implica una llamada continua a esta clase. Ciertamente, cuando se empezó a construir el método MOC, se pensó que las partículas eran algo muy ligado al transporte, y dada esta constancia de llamadas a MESH, no tenía sentido crear una clase nueva, ni convenía por excesivo movimiento en el flujo. Ahora se ha visto que otros métodos (como el metodo del camino aleatorio) utilizan las partículas como medio fundamental, sin estar nunca ligados a diferencias o elementos finitos. Por tanto, dada la generalidad de las partículas, es posible que hubiese sido mejor crear una clase propia para éstas.

Ahora bien, este pequeño error es fácilmente subsanable, dado que realmente consiste en un copiar/pegar de las subrutinas y el tipo que ya existen en TR_TRACONF. Esto es porque ya se hizo una separación muy clara de las subrutinas exclusivas de partículas durante su programación, en previsión (quizá intuitiva) de que acabarían formando una clase independiente.

5.2.5. Entrada y salida de datos

La entrada de datos se lleva a cabo mediante archivos que incorporan todos los parámetros del modelo. Estos archivos tienen un formato determinado, según el tipo al que pertenezcan. PROW trata actualmente con dos tipos. Las subrutinas de lectura, conociendo el tipo de archivo que tienen que leer, lo hacen en el formato adecuado, mediante la llamada a la subrutina privada adecuada.
Para incorporar un nuevo formato de archivo de entrada, no habría más que programar una nueva subrutina privada que leyera en el formato deseado.

Los dos formatos de entrada que lee PROW son:

- TRACONF**:** es el formato de entrada del programa TRACONF. No se ha variado nada. Con este formato, PROW lee los datos siempre que se esté usando una malla de elementos finitos (objeto de tipo T_MS_FEM2DLTRI).
- MOSTSIMPLE**:** es un formato de entrada muy sencillo, creado para la introducción de los datos con una malla de diferencias finitas (objeto de tipo T_MS_FDM2D).

Con los archivos de salida de resultados ocurre exactamente lo mismo. En este caso, sin embargo, PROW sólo trata con un formato de salida, que es ASCII.

PROW no dispone de módulos de preproceso (generación de mallas y condiciones externas) o postproceso (representación gráfica de resultados). En el subcapítulo 5.3. se habla de la posible incorporación de éstos al cuerpo del programa.

5.2.6. Ejecución del programa

Hasta ahora hemos visto como interaccionan los diferentes objetos para conseguir diferentes propósitos, con el fin de resolver las ecuaciones deseadas.
Pero el programa no puede realizar ninguna de estas tareas sin saber, por ejemplo, qué tipo de malla va a usar, o con qué tipo de matriz va a tratar en el transporte. Tampoco puede saber el nombre de los archivos que contienen los parámetros del modelo.
Todo esto está especificado en el archivo ROOT. En este archivo se halla escrito un pequeño código que declara todos y cada uno de los objetos que PROW debe usar, así como los nombres, tipos y unidades de los archivos de entrada de datos, y de salida de resultados. Este código se realiza en un pequeño lenguaje o *scripting*, cuyo intérprete se halla contenido en el módulo de lectura RENG.

En el anejo D se presenta un ejemplo de código ROOT, correspondiente a un modelo de elementos finitos. La primera línea describe que el tipo de ENGINE a usar es TRACONF. A la variable lógica ONLYFLOW se le asigna el valor FALSE, porque se va a resolver tanto la ecuación de flujo, como la de transporte (hay la posibilidad de resolver sólo la ecuación de flujo).

A partir de aquí, el código contiene datos relativos a cada una de las clases (y sus correspondientes objetos) que utilizará PROW. Así, la malla será de elementos finitos (FEM2DLTRI), la discretización temporal será de intervalo constante (CONSSTEP), y la salida de resultados (objeto WOUT) será de tipo TRACONF, así como los objetos FLOW y TRANSPORT.

Tras la declaración de la clase, y entre llaves, se encuentran datos como:

- El archivo de entrada (INPUTFILE): su nombre, su tipo y la unidad en la que debe ser abierto
- El archivo de salida (OUTPUTFILE), con datos análogos al de entrada.
- Los tipos de matrices que deben usarse para el flujo y el transporte, así como el tipo de integración temporal, si el proceso en cuestión es transitorio, etc.

Cualquier nuevo dato que deba introducirse a través del código ROOT, se incorporará fácilmente, introduciendo el código necesario para su lectura en el intérprete RENG.

5.3. Posible crecimiento del programa

Dado que la estructura de PROW está pensada para su fácil crecimiento, en este subcapítulo vamos a citar dos de los aspectos que podrían incorporarse al programa.

5.3.1. Intervalo de tiempo variable

Varios métodos de resolución de la ecuación de transporte hacen uso de partículas para describir, por ejemplo, el fenómeno de advección (MOC, MMOC), o bien el

movimiento completo del contaminante incluyendo advección y dispersión (Random Walk Method). Tradicionalmente, estos métodos utilizan un intervalo de tiempo constante para el seguimiento de las partículas. Fueron desarrollados para acuíferos cuasi-homogéneos en los que las variaciones en el espacio de la velocidad eran suaves y debidos, básicamente, al gradiente de carga hidráulica. El uso de un intervalo de tiempo constante estaba, pues, justificado. Mientras su valor fuera suficientemente pequeño aseguraría que todas las partículas atravesaran una celda en varios pasos y, debido a la pequeña variabilidad del campo de velocidades del flujo, el número de pasos necesarios para todas las partículas fuera similar.
Algunos autores (Wen y Gómez-Hernandez, 1996) han mostrado que el uso de un algoritmo con intervalo de tiempo constante en acuíferos altamente heterógeneos (varianza de la conductividad hidráulica superior a 1.0) resulta en una pérdida de eficiencia computacional. Dado que, en acuíferos heterogéneos, las velocidades del flujo pueden variar en órdenes de magnitud, un esquema con incremento de tiempo constante que asegurara que todas las partículas atravesaran las celdas en un mínimo número de pasos, necesitaría utilizar un paso de tiempo extremadamente pequeño. Es decir, las partículas atravesarían las celdas de alta conductividad en el número de pasos requerido, pero aquellas en las que la conductividad es baja o media, las atravesarían en un número de pasos excesivo.
Para resolver este problema, estos autores han propuesto que cada partícula atraviese cada celda en un número constante de pasos, lo que obliga disminuir el paso de tiempo en las celdas de elevada conductividad, y a reducirlo en las de baja o media.

Esto implica utilizar una evolución en el tiempo diferente para cada partícula. Y para esto, lo que haríamos en PROW sería crear un nuevo tipo de objeto que representara un intervalo de tiempo variable (incluido en la clase TIMEDISCR), al que llamaríamos, por ejemplo, T_TD_VARSTEP (análogamente a T_TD_CONSSTEP).

5.3.2. Preproceso / postproceso

Para generar los archivos de entrada de datos es conveniente disponer de un programa de preproceso, que de una manera gráfica y sencilla permita definir el dominio sobre el que se va a trabajar y sus características físicas. PROW no incorpora un programa de preproceso. Los archivos necesarios para el modelo se crearon mediante sencillos generadores de mallas programados en FORTRAN.
Asimismo, para interpretar los datos de salida es necesario un programa de postproceso, que sea capaz de representar gráficamente los resultados que nos interese. PROW tampoco dispone de postproceso, pudiéndose utilizar cualquier programa comercial al efecto. Los resultados presentados más adelante en este documento se realizaron mediante el programa Surfer. En el anejo F se encuentra la lista de programas utilizados para la redacción de este documento.

La incorporación de pre y postproceso es un paso importante en un programa de esta clase. Además, la estructura orientada a objeto permite construir un objeto que represente el modelo realizado, pudiendo este objeto ser directamente tratado por un módulo de pre o postproceso incorporado a la estructura del programa.
Así, por ejemplo, la clase postproceso podría disponer de varias subclases, cada una de las cuales organizara los resultados en base a un standard.

6. EJEMPLO DE MODELO UTILIZANDO PROW

Una vez descrito el funcionamiento del programa, vamos a resolver un pequeño problema de transporte en medio poroso, utilizando dos de los métodos numéricos implementados en PROW hasta el momento: elementos finitos triangulares lineales (MEF) y diferencias finitas (MDF).

6.1. Planteamiento del problema

Se trata de obtener la solución a un pulso instantáneo en x = 0 para la ecuación:

$$\phi \frac{\partial c}{\partial t} = \nabla \cdot (\mathbf{D} \cdot \nabla c) - \mathbf{q} \cdot \nabla c \tag{6.1}$$

Donde ϕ es la porosidad, c la concentración, $\mathbf{D}$ el tensor de dispersión (cuyas componentes principales son $\alpha_L |\mathbf{q}|$ y $\alpha_T |\mathbf{q}|$, siendo α_L y α_T las dispersividades longitudinal y transversal, respectivamente y , $\mathbf{q}$ el flujo.

La condición inicial corresponde a un pulso de masa M:

$$c(\mathbf{x}, t) = \delta(\mathbf{x}, t) \tag{6.1}$$

Siendo $\delta(\mathbf{x}, t)$ la delta de Dirac. Las condiciones de contorno se imponen a distancia infinita, donde la concentración tiende a cero.

Los parámetros utilizados en el modelo para describir el medio son los siguientes:

- Conductividades hidráulicas (iguales): 100 m/d
- Velocidad del flujo uniforme: 1 m/d
- Coeficiente de almacenamiento: 0.001
- Porosidad: 0.2
- Coeficiente de difusión molecular: 0.001
- Dispersividad longitudinal: 40 m
- Dispersividad transversal: 10 m
- Coeficiente de retardo: 0

6.2. Solución analítica

Suponiendo que el flujo es constante y uniforme, la solución viene dada por:

$$c(\mathbf{x},t) = \frac{M}{\phi(2\pi)^{n/2}\sqrt{|2\mathbf{D}t/\phi|}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{v}t)^t (2\mathbf{D}t/\phi)^{-1}(\mathbf{x}-\mathbf{v}t)\right] \tag{6.3}$$

Donde $\mathbf{v} = \mathbf{q}/\phi$ es la velocidad y n la dimensión (2 en nuestro caso)

La solución, a partir del instante inicial, es una campana de Gauss que se mueve en la dirección del flujo, dispersándose simultáneamente, tal y como se mostraba en la figura 2.2. (en su línea central Y=0).

6.3. Resolución por elementos finitos

Se ha resuelto el problema mediante el método de elementos finitos con una malla de triángulos uniforme. La luz de malla es de 10 m, y se ha utilizado una malla de 41x21 nodos, es decir, de 400x200 metros.

En el punto X=100, Y=0 se sitúa el punto de inyección, al que se le asigna una concentración de 37500 g/m^3. Alrededor del punto de inyección, la malla se ha refinado.

A continuación se muestra la gráfica de evolución temporal de las concentraciones en la línea Y=0 del dominio (figura 6.1), es decir la línea central.

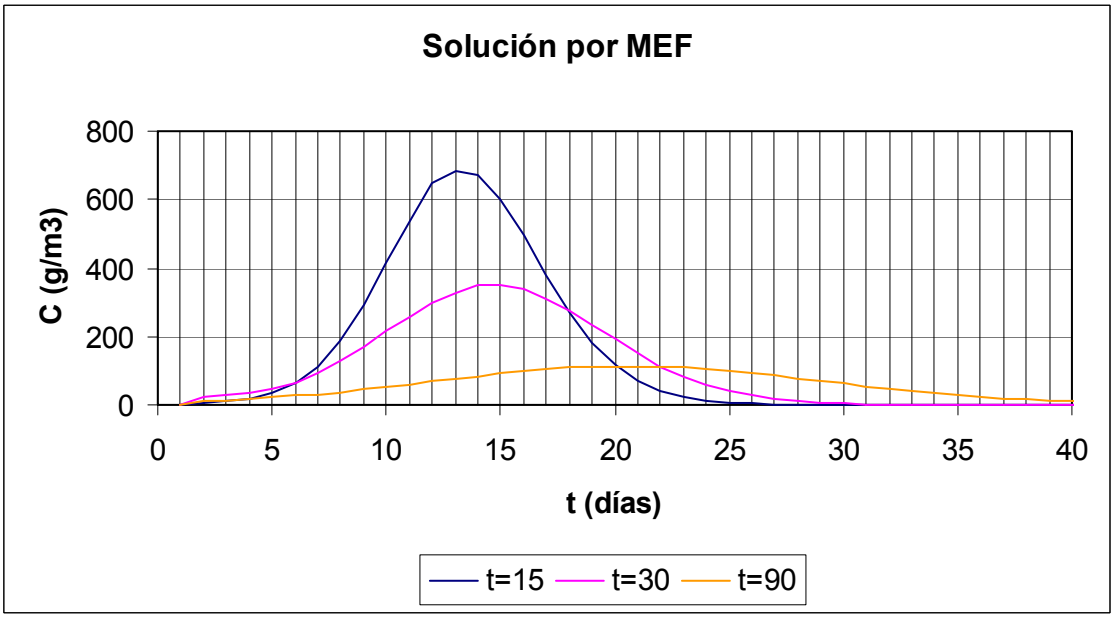

Figura 6.1. : Evolución de la solución por MEF en la línea Y=0 del dominio.

Comparemos ahora en cada uno de los tres instantes considerados, con la solución analítica (figs. 6.2., 6.3., 6.4.):

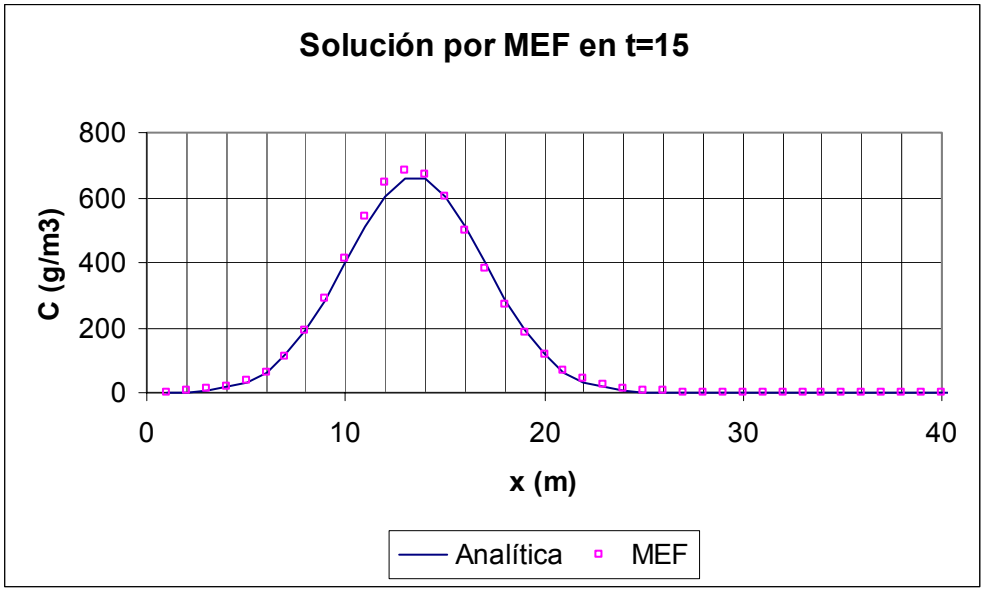

Figura 6.2. : Solución por MEF en t=15

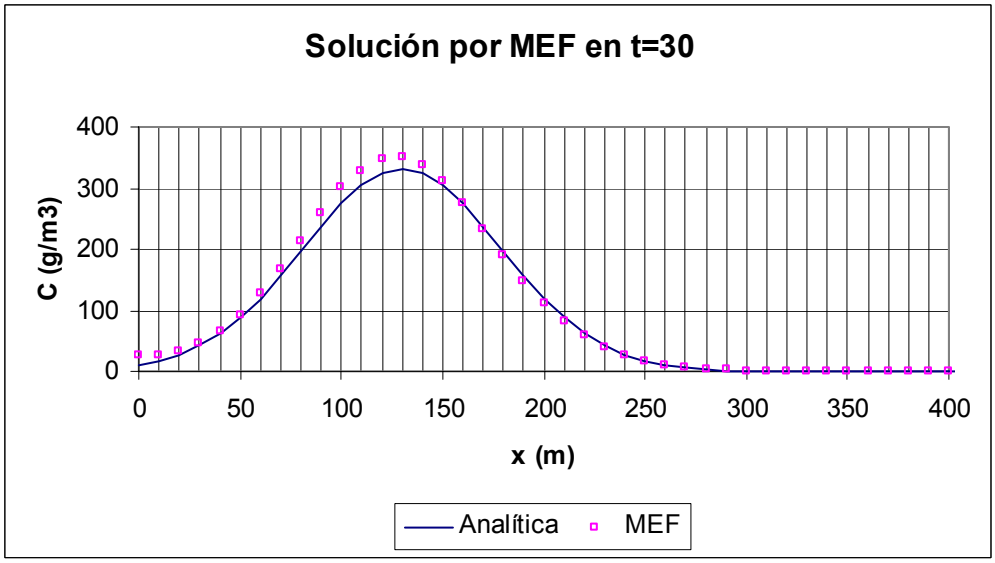

Figura 6.3. : Solución por MEF en t=30

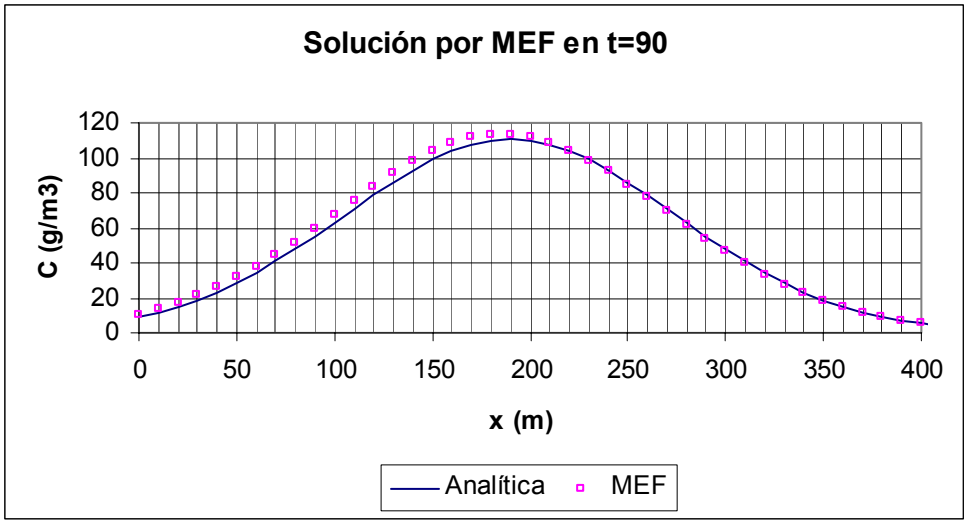

Figura 6.4. : Solución por MEF en t=90

Se muestran a continuación las soluciones para t=15 y t=30, por el método de elementos finitos (figuras 6.5, 6.6. y 6.7):

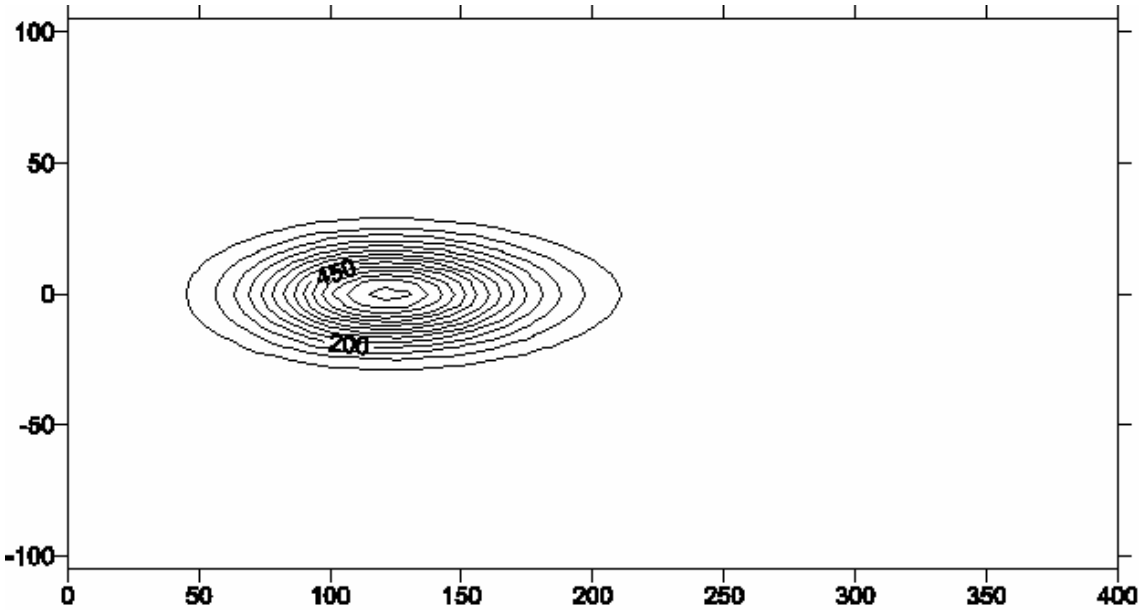

Figura 6.5. : Solución por elementos finitos para t=15 días

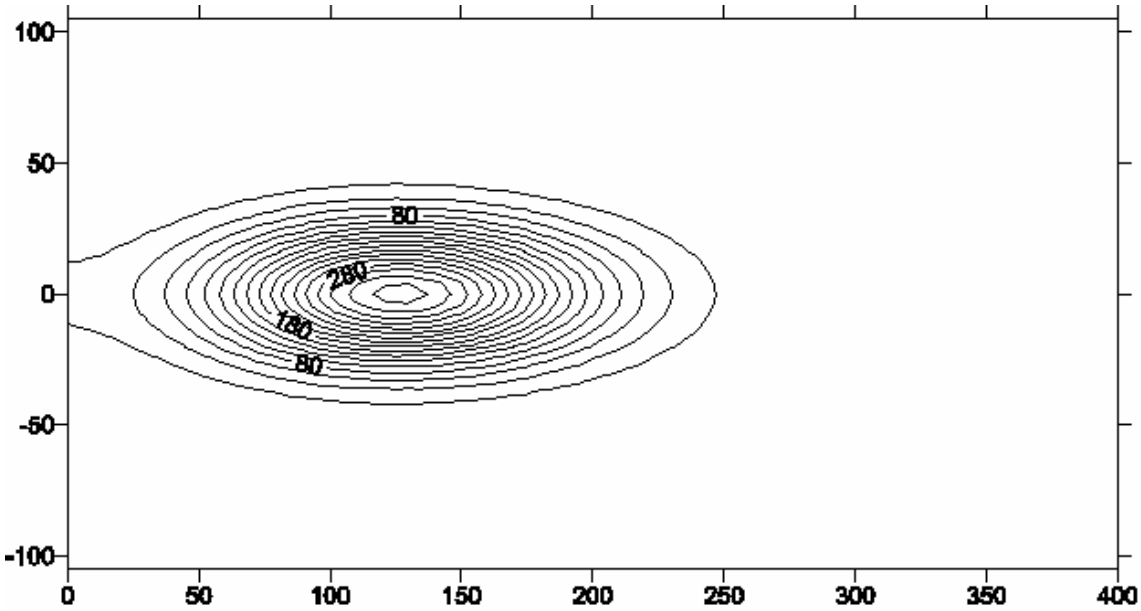

Figura 6.6. : Solución por elementos finitos para t=30 días

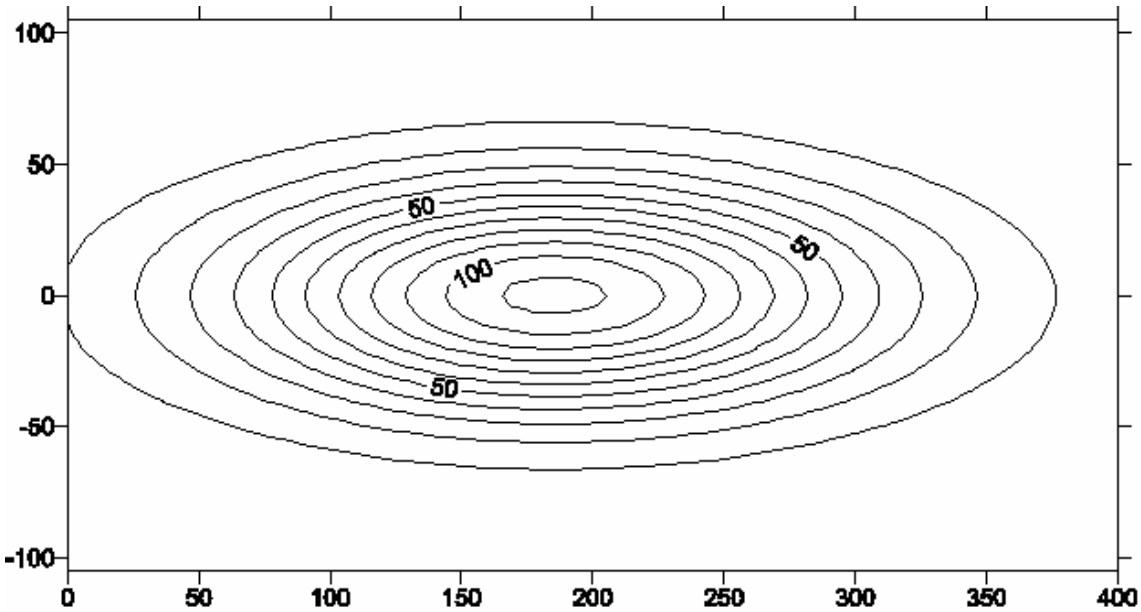

Figura 6.7. : Solución por elementos finitos para t=90 días

6.4. Resolución por diferencias finitas

Se ha utilizado una malla regular de cuadrados, cuyo lado (luz de malla) es de 10 m. El punto de inyección se sitúa, como en el método anterior, en el punto X=100, Y=0, al que se le asigna una concentración inicial de 25000 g/m^3. Las dimensiones de la malla son las mismas que antes (400x200 metros).

La evolución de la solución en el tiempo es la siguiente (figura 6.8) :

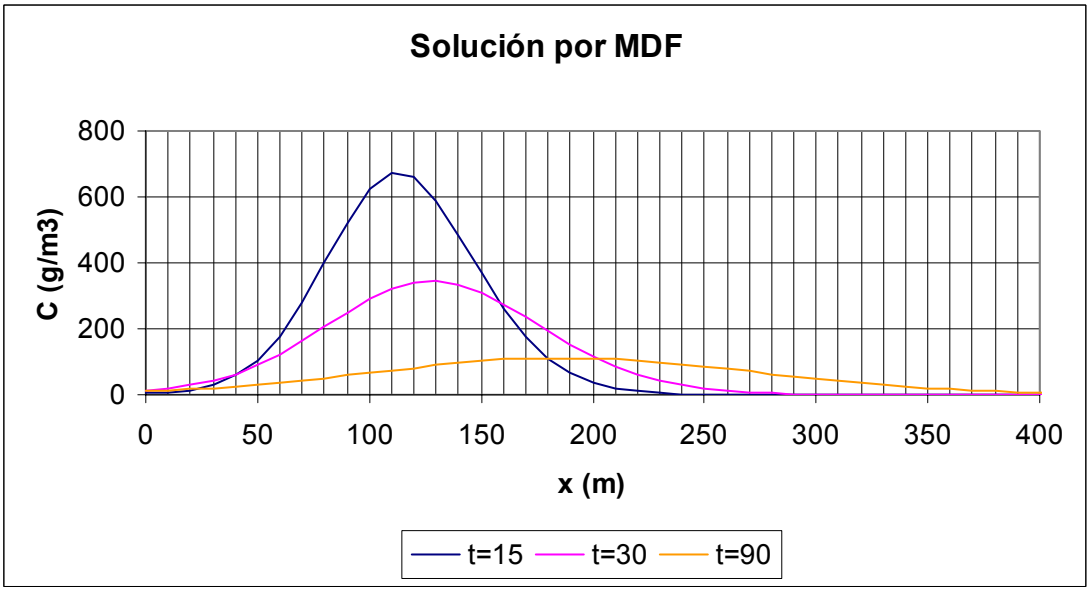

Figura 6.8. : Evolución en el tiempo de la solución por MDF

Y las siguientes figuras (6.9., 6.10 y 6.11) muestran la comparación con la solución analítica en cada instante de tiempo considerado:

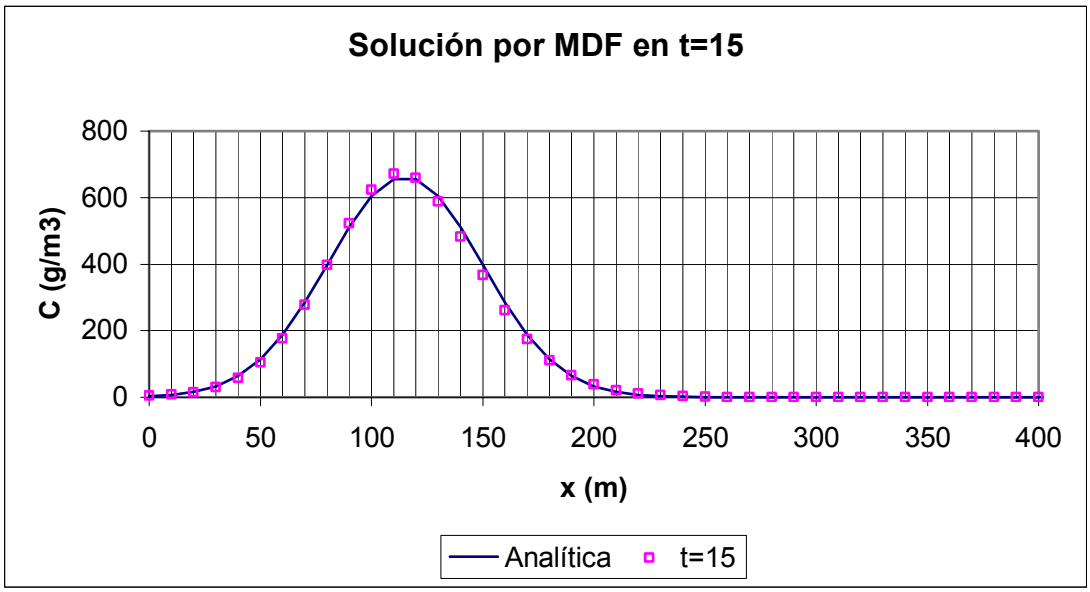

Figura 6.9. : Solución por MDF para t=15 días

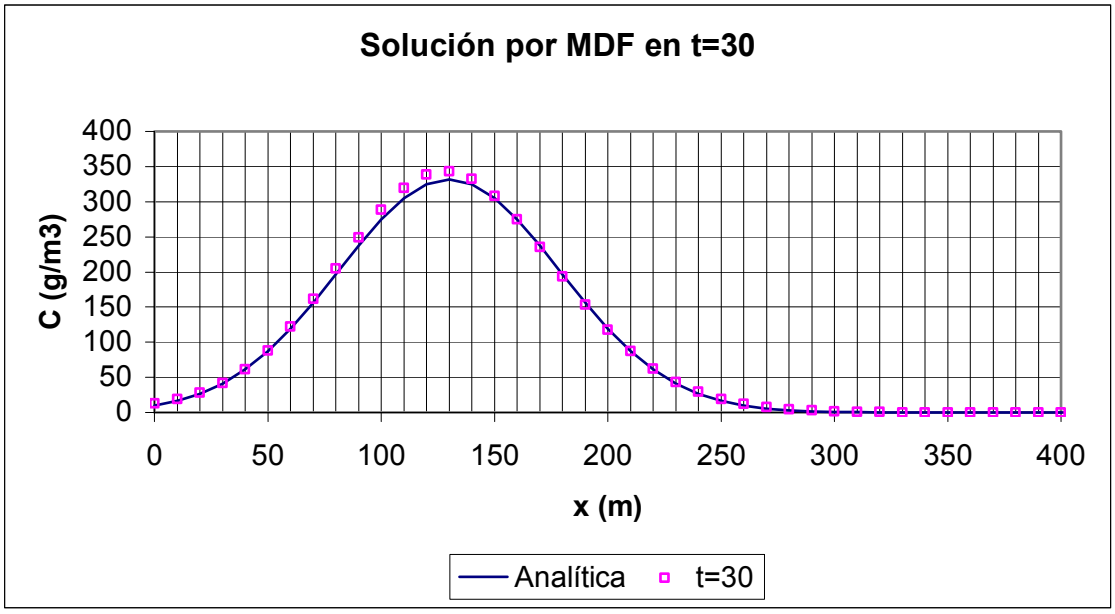

Figura 6.10. : Solución por MDF para t=30 días

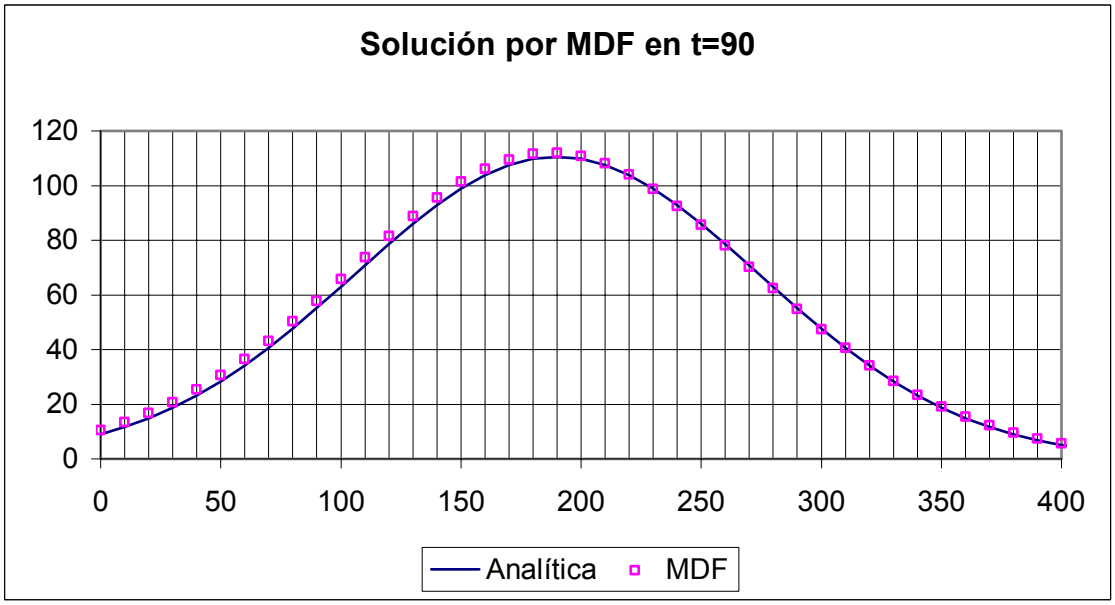

Figura 6.11. : Solución por MDF para t=90 días

Las soluciones completas, en gráficos de curvas nivel se muestran en las siguientes figuras (6.12, 6.13, 6.14):

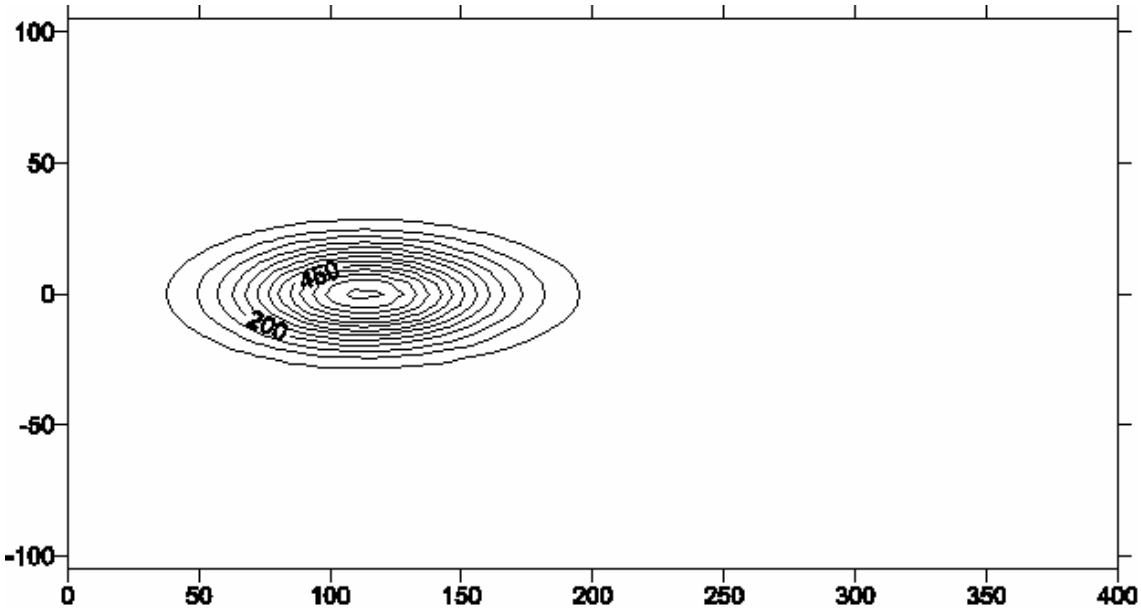

Figura 6.12. : Solución por diferencias finitas para t=15 días

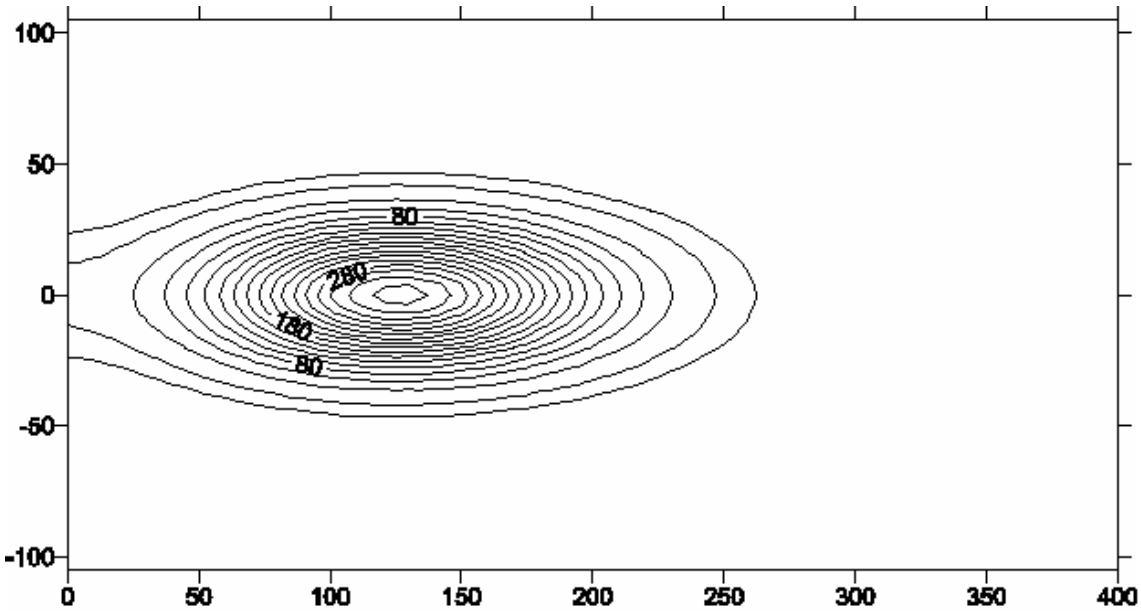

Figura 6.13. : Solución por diferencias finitas para t=30 días

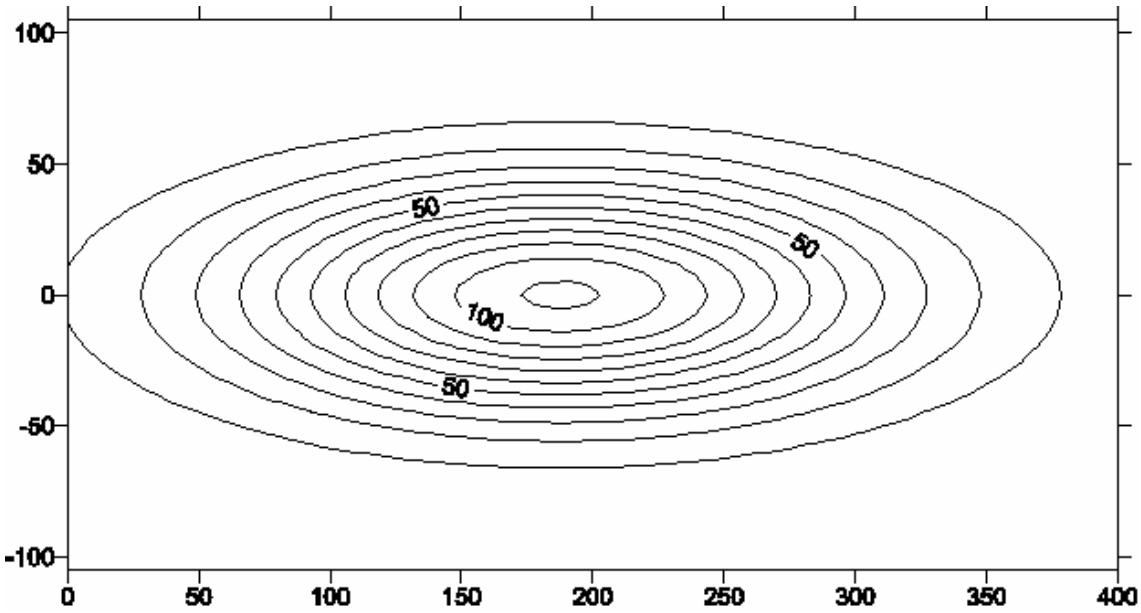

Figura 6.14. : Solución por diferencias finitas para t=90 días

7. CONCLUSIONES

7.1. Respecto a la programación orientada a objetos

- A lo largo de las páginas anteriores, se han intentado explicar las ventajas de la programación orientada a objeto frente a la programación clásica, en los siguientes aspectos:

  - Facilidad de incorporación de nuevos conceptos al programa.
  - Facilidad de comprensión del programa, una vez asumida su estructura.
  - Seguridad en el manejo de la información, debido a su control estricto.

- Es importante pensar un diseño de clases antes de comenzar el programa, que sea suficientemente general, ya que este diseño no debería cambiar significativamente durante la programación. Hay que tener presente que un mal diseño puede dificultar mucho la introducción de nuevos conceptos en el programa en fases avanzadas de la programación. También puede dificultar la corrección (eliminación de errores) del programa.

- Tomarnos la orientación a objeto en un sentido estricto a veces nos puede llevar a un purismo excesivo en el diseño del programa. Es mucho más importante dejar el programa abierto a crecer que, a priori, desglosarlo excesivamente sin necesidad.

- La decisión de qué conceptos merecen un objeto propio a veces no es fácil. Las claves para esta decisión son dos, mutuamente condicionadas:
  - Los objetos del programa deben estar adaptados a las necesidades reales del problema a resolver.
  - Sin embargo, debe ser posible (y fácil) introducir nuevos objetos más concretos, necesarios para nuevos problemas, o nuevos enfoques del mismo.

- Cuando se programa en un lenguaje clásico, hay que aprovechar la flexibilidad del mismo. No es malo violar puntualmente algún principio de la orientación a objeto, si esto nos proporciona una gran ventaja computacional o de otra clase. No se trata de crear obras de arte, sino de construir programas eficientes y flexibles.

7.2. Respecto al programa desarrollado

- Se ha desarrollado un programa que permite resolver las ecuaciones de flujo y transporte de solutos en medios porosos, mediante técnicas de programación orientada a objeto en FORTRAN90.

- A partir de un programa que resolvía el transporte por el método de Galerkin, se han implementado tres nuevos métodos (diferencias finitas, MOC y MMOC). El nuevo código se ha construido siguiendo en la medida de lo posible las reglas de la programación orientada a objetos.

- La implementación de diferencias finitas se ha basado en una analogía entre los entes que componen la malla de elementos finitos y los de la nueva malla a introducir.

- Es posible que las partículas hubieran merecido una clase propia. Sin embargo, la estructura con la que se han diseñado las subrutinas de los métodos MOC y MMOC facilitaría su construcción notablemente.

- Se comprueba que los resultados del modelo ensayado con PROW son satisfactorios, al compararlos con su solución analítica.

8. REFERENCIAS BIBLIOGRÁFICAS.

Bear (1972), *Dynamics of fluids in porous media*. American Elsevier, New York.

Carrera, J., Galarza, G. y Medina, A. (1993), *TRACONF. Programa de elementos finitos para la solución de las ecuaciones de flujo y transporte en acuíferos confinados. Manual del usuario*. ETSECCPB-UPC, Barcelona.

Carrera, J. y Melloni, G. (1987), *The Simulation of Solute Transport: An Approach Free of Numerical Dispersion*. Sandia National Laboratories, Alburquerque, New Mexico.

Cross, J.T., Masters, I. y Lewis, R.W. (1997), *A brief review of object-oriented finite element methods,* editors R.W. Lewis, J.T. Cross, Proc. 10th International Conference on Numerical Methods in Thermal Problems. Pineridge Press, Swansea.

Cross, J.T., Masters, I., Sukirman y Y., Lewis, R.W. (1997), *Object-Oriented Programming Techniques for Finite Element Methods in Heat Transfer,* editors R.W. Lewis, J.T. Cross, Proc. 10th International Conference on Numerical Methods in Thermal Problems. Pineridge Press, Swansea.

Decyk, V.K., Norton,C.D. y Szymanski, B.W. (1997 a), Expressing Object-Oriented Concepts in FORTRAN90. *ACM Fortran Forum*, Vol. 16, nº1.

Decyk, V.K., Norton,C.D.y Szymanski, B.W. (1997 b), *How to Support Inheritance and Run-Time Polymorphism in FORTRAN90.*

Dubois-Pèlerin, Y. y Zimmermann, T. (1992), *Object Oriented Finite Element Programming: A New Modularity*, editores H. Alder, J.C. Heinrich, S. Lavanchy, E. Oñate, B. Suárez, p. 843-851 Numerical Methods in Engineering and Applied Sciences. CIMNE, Barcelona.

Harbaugh, A.H, Banta, E.R., Hill, M.C. y McDonald, M.G. (2000), *MODFLOW-2000, the U.S. Geological Survey Modular Ground Water Model – User guide to modularization concepts and the ground-water flow process.* Open-File Report 00-92. Reston, Virginia.

Kipp, K.L. (1997), *Guide to the revised heat and solute transport simulator: HST3D-Version 2*. Water Resources Investigation Report 97-4157. US Geological Survey, Denver, Colorado.

Wen, X.H. y Gómez-Hernández, J. (1996), The Constant Displacement Scheme for Tracking Particles in Heterogeneous Aquifers. *GROUND WATER*, Vol. 34, 135-142.

Zheng, C. y Wang, P. (1999), *MT3DMS: A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems; Documentation and User's Guide*. US Army Corps of Engineers Contract Report.

9. OTRAS FUENTES DE CONSULTA

Páginas web consultadas:

http://www.cs.rpi.edu/~szymansk/oof90.html

http://www.ltas.ulg.ac.be/klapka/cosyen/cosyen.html

Referencias a software en internet:

MT3D : http://hydro.geo.ua.edu/mt3d/

HST3D :http://water.usgs.gov/software/hst3d.html

MODFLOW :http://water.usgs.gov/software/modflow.html

HYDRUS2D :http://www.ussl.ars.usda.gov/models/hydrus2d.HTM

ANEJOS

A. Listado en FORTRAN 90 del programa principal MAIN.F90

```
PROGRAM PROW
!
!******* PROW MAIN PROGRAM: PRocess Oriented ground WATER modelling
!

!#INI_ONLY_PC#
      USE DFLIB
!#END_ONLY_PC#

      USE M_ENGINE

      IMPLICIT NONE

      TYPE(T_ENGINE) :: &
            ENGINE

      INTEGER*4 :: &
            liError

!#INI_ONLY_PC#
      integer*4 :: &
            numArgs,    &
            iPos
!#END_ONLY_PC#
      CHARACTER*200 :: &
            rootname
      character*25 :: &
            lsEspType
      logical :: &
            lbPoints

      rootname = 'root.eng'

!#INI_ONLY_PC#
      ! Obtains command line arguments: first argument is root
filename
      numargs = NARGS()
      if ( numargs .eq. 2 ) then
            CALL GETARG(1, rootname)
            iPos = index(rootname,'"') -1
            if ( iPos.gt.0) then
                  rootname = rootname(1:iPos)
            end if
      end if
!#END_ONLY_PC#

      !Loads engine file
```

```
      call LoadEng ( rootname, liError)
      if ( liError .ne. 0 ) stop

      !Reads engine type
      lsEspType = 'TRACONF'
      call ReadType ( GT_ENGINE, lsEspType, lbPoints, lsEspType )

      ! No pointer allowed
      if ( lbPoints ) then
            print *,'CANNOT USE POINTER IN MAIN FOR ENGINE TYPE'
            stop
      end if

      ! Creates this engine
      print *,'Creating...'
      CALL CREATE_(ENGINE, lsEspType, liError )
      if ( liError .ne. 0 ) stop

      ! Reads engine attributes
      print *,'Reading...'
      call ChgSect('.|' // GT_ENGINE // CS_BLANK)
      CALL READ_(ENGINE, liError)
      if ( liError .ne. 0 ) stop
      call ChgSect(CS_PREVIOUS)

      ! Everything ok so far, then ...

      ! Runs engine
      print *,'Running...'
      CALL RUN_(ENGINE)

      ! Destroys everyhing gracefully
      print *,'Finishing...'
      CALL DESTROY_(ENGINE)


      ! Unloads engine file
      CALL UnloadEng()

      ! and that's all
      print *,'...OK'
      stop

998 continue
      print *,'FILE NOT FOUND:',rootname
      stop

END PROGRAM PROW
```

B. Listado en FORTRAN de la subclase MS_FDM2D

```
MODULE M_MS_FDM2D

!
!******* MESH MANAGEMENT *********
!
! Purpose: Manages operations on a 2-dimensional finite difference mesh
!
! Assumptions/Shortcomes:
!
! Abreviation: MS_D2
!
! General use:
!                   . Create object (CREATE_)
!                   . Destroy object (DESTROY_)
!                   . Read attributes (READ_)
!                   . Initialize values (INIT_)
!
!
!                   . ... and it's ready to perform operations on mesh:
!                             . INTGV_DIV_P_GRAD_x_
!                             . INTGV_P_x_
!                             . INTGV_P_GRAD_x_
!                             . P_GRAD_x_
!                             . INTGB_F_x_MINUS__x0_
!                             . INTGB_F_
!                             . INTERP_ELEM2VP
!
!
!
! Tested? YES.
!
! Last modified: 17-06-2004
!
!_____________________________________________________________________________

!
! Used modules
!
          USE M_RENG
          USE M_MS_PARENT,    ONLY : T_MS_PARENT

!
! List of public subroutines
!
          PUBLIC :: &
                    CREATE_,                                & ! Creates: allocates and sets certain attributes
                    DESTROY_,                               & ! Destroys: deallocates
                    READ_,                                            & ! Reads attributes
                    INIT_,                                            & ! Initializes values
                    pACTH_,                                           & ! Points to thickness vector
                    DEASSACTH_,                                       & ! Deassigns pointer to thickness vector
                    INTGV_DIV_P_GRAD_x_,          & ! Volume integral{div (P grad x)} (P stands for (ndim,ndim) field
and x are the unknowns)
                    INTGV_P_x_,                                       & ! Volume integral{P*x} (P stands for scalar
field and x are the unknowns)
                    INTGV_P_GRAD_x_,              & ! Volume integral{P grad x} (P stands for (ndim) field and x are
the unknowns)
                    P_GRAD_s_,                                        & ! Computes P*(grad s) (P stands for (ndim,ndim)
field and s stands for scalar field )
                    INTGB_F_x_MINUS_x0_,          & ! Boundary integral{f(x-x0)} (f,x0 stand for scalars)
                    INTGB_F_,                               & ! Boundary integral{f} (f stand for scalar)
                    INTERP_ELEM2VP_,              & ! Interpolates from elements to cell interfaces
                    RELAT_GRAD_,                            &
                    ELEM_VOL_,                                        &
                    PART_DISPOSE_,                          &
                    MOVE_POINT_,                            &
                    PART2ELEM_,                                       &
                    PART_UPDATE_,                           &
                    SCALAR_FIELD_

! List of private subroutines
!

          PRIVATE :: &
                    CREATE_MS_D2, DESTROY_MS_D2, READ_MS_D2, INIT_MS_D2, &
                    pACTH_MS_D2, DEASSACTH_MS_D2, INTGV_DIV_P_GRAD_x_MS_D2, &
                    INTGV_P_x_MS_D2, INTGV_P_GRAD_x_MS_D2, P_GRAD_s_MS_D2, &
                    INTGB_F_x_MINUS_x0_MS_D2, INTGB_F_MS_D2, INTERP_ELEM2VP_MS_D2,&
                    RELAT_GRAD_MS_D2, ELEM_VOL_MS_D2, PART_DISPOSE_MS_D2, &
                    MOVE_POINT_MS_D2, PART2ELEM_MS_D2, PART_UPDATE_MS_D2, &
                    SCALAR_FIELD_MS_D2, &
                    READ_MOSTSIMPLE, CASE5_COMPONENTS, CASE9_COMPONENTS, &
                    GET_DXDY, INTERP_TENSOR_ARIT, INTERP_VECTOR, &
                    VEL_COMPONENTS, INTERP_LEVELS, GET_CELL, POINT_VEC, &
                    IS_OUTSIDE, REFLECT, COORD_X, COORD_Y

! _____________________________________________________________________________
!
! Type definition
!
! Last modified: 06-03-2003
!
```

```
          TYPE T_MS_FDM2D

                    PRIVATE

                    TYPE (T_MS_PARENT),POINTER :: &
                              p                                                   ! Parent attributes (dimension, number
of nodes, elements...)

                    LOGICAL :: &
                              ISASSIGNED                                ! .TRUE. if non-computed values points
                                                                                  ! .FALSE. if allocates the values
                    LOGICAL, POINTER :: &
                              IS_INDOMAIN(:)                  ! .TRUE. if cell is in domain

                    INTEGER*4 :: &
                              MAX_I, &                        ! Maximum number of cells in x direction
                              MAX_J                                     ! Maximum number of cells in y direction

                    INTEGER*4, POINTER :: &
                              ADJACENT_CELLS(:,:), &          ! Cells in domain, adjacent to each cell in domain
                              CELL_INTERFACES(:,:), &         ! Interfaces enclosing a cell
                              CELL_IJ(:,:)                    ! Cell position inside reference rectangle, called (I,J)
                                                                                  ! in private subroutines

                    REAL*8,POINTER :: &
                              DELTA_X(:), &                   ! Delta X values (MAX_I)
                              DELTA_Y(:)                      ! Delta Y values (MAX_J)

                    REAL*8,POINTER :: &
                              ACTH(:)                                   ! Aquifer thickness (numel)


          END TYPE T_MS_FDM2D

!_____________________________________________________________________________
!
! Interfaces
!

          INTERFACE CREATE_
                    MODULE PROCEDURE CREATE_MS_D2
          END INTERFACE

          INTERFACE DESTROY_
                    MODULE PROCEDURE DESTROY_MS_D2
          END INTERFACE

          INTERFACE READ_
                    MODULE PROCEDURE READ_MS_D2
          END INTERFACE

          INTERFACE INIT_
                    MODULE PROCEDURE INIT_MS_D2
          END INTERFACE

          INTERFACE pACTH_
                    MODULE PROCEDURE pACTH_MS_D2
          END INTERFACE

          INTERFACE DEASSACTH_
                    MODULE PROCEDURE DEASSACTH_MS_D2
          END INTERFACE

          INTERFACE INTGV_DIV_P_GRAD_x_
                    MODULE PROCEDURE INTGV_DIV_P_GRAD_x_MS_D2
          END INTERFACE

          INTERFACE INTGV_P_x_
                    MODULE PROCEDURE INTGV_P_x_MS_D2
          END INTERFACE

          INTERFACE INTGV_P_GRAD_x_
                    MODULE PROCEDURE INTGV_P_GRAD_x_MS_D2
          END INTERFACE

          INTERFACE P_GRAD_s_
                    MODULE PROCEDURE P_GRAD_s_MS_D2
          END INTERFACE

          INTERFACE INTGB_F_x_MINUS_x0_
                    MODULE PROCEDURE INTGB_F_x_MINUS_x0_MS_D2
          END INTERFACE

          INTERFACE INTGB_F_
                    MODULE PROCEDURE INTGB_F_MS_D2
          END INTERFACE

          INTERFACE INTERP_ELEM2VP_
                    MODULE PROCEDURE INTERP_ELEM2VP_MS_D2
          END INTERFACE

          INTERFACE RELAT_GRAD_
                    MODULE PROCEDURE RELAT_GRAD_MS_D2
          END INTERFACE

          INTERFACE ELEM_VOL_
                    MODULE PROCEDURE ELEM_VOL_MS_D2
          END INTERFACE

          INTERFACE PART_DISPOSE_
                    MODULE PROCEDURE PART_DISPOSE_MS_D2
          END INTERFACE
```

```
          INTERFACE MOVE_POINT_
                    MODULE PROCEDURE MOVE_POINT_MS_D2
          END INTERFACE

          INTERFACE PART2ELEM_
                    MODULE PROCEDURE PART2ELEM_MS_D2
          END INTERFACE

          INTERFACE PART_UPDATE_
                    MODULE PROCEDURE PART_UPDATE_MS_D2
          END INTERFACE

          INTERFACE SCALAR_FIELD_
                    MODULE PROCEDURE SCALAR_FIELD_MS_D2
          END INTERFACE

!
! Module contains
!

          CONTAINS

!_______________________________________________________________________________________
!
!                                                          PUBLIC SUBROUTINES
!_______________________________________________________________________________________

!_____________________________________________________________ CREATE_MS_D2
!
SUBROUTINE CREATE_MS_D2 (THIS, PARENT, AIERROR)
!         Purpose: creates object of type MESH
!         Arguments:
!                           THIS: FDM2D-type object
!                           PARENT: parent attributes
!                           AIERROR
!
!         Tested? NO
!   Last modified 06-03-2003
!

          IMPLICIT NONE

          TYPE(T_MS_FDM2D) THIS
          TYPE(T_MS_PARENT),POINTER :: PARENT

          INTEGER*4 :: &
                    AIERROR


          AIERROR = 0

          IF ( .NOT. ASSOCIATED(PARENT) ) THEN
                    AIERROR = -1
                    PRINT *,'ERROR: PARENT NOT CREATED IN CREATE_MS_D2'
                    RETURN
          END IF

          THIS%p => PARENT

          ! Default values
          THIS%p%NDIM = 2
          THIS%p%MLOC = 3

          ! Nullifying every component inside type T_MS_FDM2D
          THIS%MAX_I = 0
          THIS%MAX_J = 0

          NULLIFY(THIS%IS_INDOMAIN)
          NULLIFY(THIS%ADJACENT_CELLS)
          NULLIFY(THIS%CELL_INTERFACES)
          NULLIFY(THIS%CELL_IJ)
          NULLIFY(THIS%DELTA_X)
          NULLIFY(THIS%DELTA_Y)
          NULLIFY(THIS%ACTH)

          THIS%ISASSIGNED = .FALSE.

          RETURN
END SUBROUTINE CREATE_MS_D2


!_____________________________________________________________ DESTROY_MS_D2
!
SUBROUTINE DESTROY_MS_D2 (THIS)
!         Purpose: destroys object of type MESH
!         Arguments:
!                           THIS: FDM2D-type object
!
!         Tested? NO
!   Last modified 06-03-2003
!
          IMPLICIT NONE

          TYPE(T_MS_FDM2D) THIS

          NULLIFY(THIS%p)

          ! Default values


          IF ( .NOT. THIS%ISASSIGNED ) THEN
                    IF ( ASSOCIATED(THIS%DELTA_X) ) THEN
                              DEALLOCATE(THIS%DELTA_X)
```

```
                    END IF
                    IF ( ASSOCIATED(THIS%DELTA_Y) ) THEN
                              DEALLOCATE(THIS%DELTA_Y)
                    END IF
                    IF ( ASSOCIATED(THIS%IS_INDOMAIN) ) THEN
                              DEALLOCATE(THIS%IS_INDOMAIN)
                    END IF
                    IF ( ASSOCIATED(THIS%ADJACENT_CELLS) ) THEN
                              DEALLOCATE(THIS%ADJACENT_CELLS)
                    END IF
                    IF ( ASSOCIATED(THIS%CELL_INTERFACES) ) THEN
                              DEALLOCATE(THIS%CELL_INTERFACES)
                    END IF
                    IF ( ASSOCIATED(THIS%CELL_IJ) ) THEN
                              DEALLOCATE(THIS%CELL_IJ)
                    END IF
          END IF

          IF ( ASSOCIATED(THIS%ACTH) ) THEN
                    DEALLOCATE(THIS%ACTH)
          END IF

          THIS%MAX_I = 0
          THIS%MAX_J = 0

          NULLIFY(THIS%DELTA_X)
          NULLIFY(THIS%DELTA_Y)
          NULLIFY(THIS%ACTH)
          NULLIFY(THIS%IS_INDOMAIN)
          NULLIFY(THIS%ADJACENT_CELLS)
          NULLIFY(THIS%CELL_INTERFACES)
          NULLIFY(THIS%CELL_IJ)

          THIS%ISASSIGNED = .FALSE.

          RETURN
END SUBROUTINE DESTROY_MS_D2

!_____________________________________________________________ READ_MS_D2
!
SUBROUTINE READ_MS_D2 (THIS, AIERROR)
!         Purpose: reads object attributes
!         Arguments:
!                              THIS: FDM2D-type object
!                              AIERROR
!
!         Tested? NO
!   Last modified 30-01-2003
!
          IMPLICIT NONE

          TYPE(T_MS_FDM2D) THIS

          INTEGER*4 :: &
                    AIERROR, &
                    IUNIT

          CHARACTER*50 :: &
                    FILETYPE
          CHARACTER*100 :: &
                    FILENAME

! Default values

          FILETYPE='MOSTSIMPLE'
          FILENAME='INPUT_MS.DAT'

! Obtains filename

          IUNIT = 444
          CALL ReadFile ( NA_INPUTFILE, FILETYPE, FILENAME, IUNIT, &
                                                                                    FILETYPE, FILENAME, IUNIT)
          IF ( AIERROR .NE. 0 ) RETURN

! Select file type

          SELECT CASE (FILETYPE)

                    CASE (FT_MOSTSIMPLE)
                              CALL READ_MOSTSIMPLE(THIS, FILENAME, IUNIT)

                    CASE DEFAULT
                              AIERROR = -97
                              PRINT *,'ERROR: FILE TYPE NOT ACCEPTED BY READ_MS_D2 '

          END SELECT

          RETURN
END SUBROUTINE READ_MS_D2

!_____________________________________________________________ INIT_MS_D2
!
SUBROUTINE INIT_MS_D2 (THIS)
!         Purpose: initializes values of attributes
!         Arguments:
!                              THIS: FDM2D-type object
!
!         Troubles:
!                              construction of ACTH array?
!                              bandwidth bad calculated
!         Tested? NO
!   Last modified 06-03-2003
!
```

```
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS

          ! Internal variable declaration
          INTEGER*4 :: &
                    K, KK, &
                    I, J, &                     ! Position of a cell inside reference rectangle
                    TCELLS, &           ! Total number of cells (in&out of domain) inside ref. rectangle
                    MAXADJ, &           ! Maximum number of adjacent cells
                    AP, &                       ! Auxiliar variable
                    M, &                        ! Cell number inside domain
                    BW, BWAUX, &        ! Auxiliar variable for bandwidth calculation
                    BCELL1, BCELL2

          INTEGER*4, POINTER :: &
                    POS(:), &           ! Auxiliar values to define position of adjacent cells
                    DOMAIN_CELL(:)      ! Array of TCELLS components: it values the cell number inside
                                                        ! domain, if cell belongs to domain. Otherwise it values
zero.


! Construction of ADJACENT_CELLS and CELL_IJ arrays

          MAXADJ=8  ! Maximum number of adjacent cells

          ALLOCATE (THIS%ADJACENT_CELLS(THIS%p%NUMNP, MAXADJ))
          ALLOCATE (THIS%CELL_IJ(THIS%p%NUMNP, 2))

          TCELLS = THIS%MAX_I*THIS%MAX_J
          ALLOCATE (DOMAIN_CELL(TCELLS))
          DOMAIN_CELL = 0

! Construction of adjacent positions array
          ALLOCATE (POS(MAXADJ))

          POS(1) = 1
          POS(2) = -THIS%MAX_I+1
          POS(3) = -THIS%MAX_I
          POS(4) = -THIS%MAX_I-1
          POS(5) = -1
          POS(6) = THIS%MAX_I-1
          POS(7) = THIS%MAX_I
          POS(8) = THIS%MAX_I+1


          THIS%ADJACENT_CELLS = 0
          THIS%CELL_IJ = 0

          M=1                 ! Cell number inside domain
          I=1                 ! Cell x position inside reference rectangle
          J=1                 ! Cell y position inside reference rectangle

          DO K=1,TCELLS

                    IF (THIS%IS_INDOMAIN(K)) THEN
                              THIS%CELL_IJ(M,1) = I
                              THIS%CELL_IJ(M,2) = J
                              DOMAIN_CELL(K) = M
                              DO KK=1,MAXADJ
                                        AP = K+POS(KK)
                                        IF ((AP.GT.0).AND.(AP.LE.TCELLS) &

          .AND.(THIS%IS_INDOMAIN(AP))) THEN
                                                  THIS%ADJACENT_CELLS(M,KK)=AP
                                        ELSE
                                                  THIS%ADJACENT_CELLS(M,KK)=0
                                        END IF
                              END DO
                              ! Checks if cell is at the edge of reference rectangle
                              IF (I.EQ.1) THEN
                                        THIS%ADJACENT_CELLS(M,4) = 0
                                        THIS%ADJACENT_CELLS(M,5) = 0
                                        THIS%ADJACENT_CELLS(M,6) = 0
                              END IF
                              IF (I.EQ.THIS%MAX_I) THEN
                                        THIS%ADJACENT_CELLS(M,1) = 0
                                        THIS%ADJACENT_CELLS(M,2) = 0
                                        THIS%ADJACENT_CELLS(M,8) = 0
                              END IF
                              IF (J.EQ.1) THEN
                                        THIS%ADJACENT_CELLS(M,2) = 0
                                        THIS%ADJACENT_CELLS(M,3) = 0
                                        THIS%ADJACENT_CELLS(M,4) = 0
                              END IF
                              IF (J.EQ.THIS%MAX_J) THEN
                                        THIS%ADJACENT_CELLS(M,6) = 0
                                        THIS%ADJACENT_CELLS(M,7) = 0
                                        THIS%ADJACENT_CELLS(M,8) = 0
                              END IF

                              M=M+1
                    END IF

                    IF(I.EQ.(THIS%MAX_I)) THEN
                              J=J+1
                              I=1
                    ELSE
                              I=I+1
                    END IF

          END DO
```

```
          ! Bandwidth
          BW = 0
          DO M=1, THIS%p%NUMNP

                    BCELL1=THIS%ADJACENT_CELLS(M,4)
                    IF (BCELL1.EQ.0) THEN
                              BCELL1 = THIS%ADJACENT_CELLS(M,3)
                    END IF
                    IF (BCELL1.EQ.0) THEN
                              BCELL1 = THIS%ADJACENT_CELLS(M,2)
                    END IF
                    IF (BCELL1.EQ.0) THEN
                              BCELL1 = THIS%ADJACENT_CELLS(M,5)
                    END IF
                    IF (BCELL1.EQ.0) THEN
                              KK=1
                              DO WHILE(DOMAIN_CELL(KK).LT.M)
                                        KK=KK+1
                              END DO
                              BCELL1 = KK
                    END IF

                    BCELL2=THIS%ADJACENT_CELLS(M,8)
                    IF (BCELL2.EQ.0) THEN
                              BCELL2 = THIS%ADJACENT_CELLS(M,7)
                    END IF
                    IF (BCELL2.EQ.0) THEN
                              BCELL2 = THIS%ADJACENT_CELLS(M,6)
                    END IF
                    IF (BCELL2.EQ.0) THEN
                              BCELL2 = THIS%ADJACENT_CELLS(M,1)
                    END IF
                    IF (BCELL2.EQ.0) THEN
                              KK=1
                              DO WHILE(DOMAIN_CELL(KK).LT.M)
                                        KK=KK+1
                              END DO
                              BCELL2 = KK
                    END IF

                    BWAUX=1
                    DO WHILE (BCELL1 .LT. BCELL2)
                              BCELL1=BCELL1+1
                              IF(THIS%IS_INDOMAIN(BCELL1)) THEN
                                        BWAUX=BWAUX+1
                              END IF
                    END DO
                    IF (BWAUX .GT. BW) THEN
                              BW = BWAUX
                    END IF
          END DO
          THIS%p%DIFMX = BW-1


          ! Once assigned adjacent cell numbers referred to ref.rectangle, we
          ! change them by numbers referred to domain, using DOMAIN_CELL array.
          DO M=1,THIS%p%NUMNP
                    DO KK=1,MAXADJ
                              AP = THIS%ADJACENT_CELLS(M,KK)
                              IF (AP.NE.0) THEN
                                        THIS%ADJACENT_CELLS(M,KK) = DOMAIN_CELL(AP)
                              END IF
                    END DO
          END DO

          DEALLOCATE (DOMAIN_CELL, POS)

! Construction of ACTH array

          THIS%ACTH = 1

! Construction of adjacent cells array and NUMVP

          ALLOCATE (THIS%CELL_INTERFACES(THIS%p%NUMNP, 2))

          THIS%CELL_INTERFACES = 0
          J=0
          DO I=1,THIS%p%NUMNP
                    IF(THIS%ADJACENT_CELLS(I, 1).NE.0) THEN
                              J=J+1
                              THIS%CELL_INTERFACES(I, 1)=   J
                    END IF
                    IF(THIS%ADJACENT_CELLS(I, 7).NE.0) THEN
                              J=J+1
                              THIS%CELL_INTERFACES(I, 2)=   J
                    END IF
          END DO

          ! Assignation of value to NUMVP
          THIS%p%NUMVP = J



          THIS%p%ISCALCULATED = .TRUE.

          RETURN
END SUBROUTINE INIT_MS_D2

!_____________________________________________________________ pACTH_MS_D2
!
FUNCTION pACTH_MS_D2 (THIS)
!         Purpose: points to thickness vector
!         Arguments:
```

```
!                           THIS: FDM2D-type object
!
!         Tested? NO
!   Last modified 30-01-2003
!
          IMPLICIT NONE
          TYPE(T_MS_FDM2D) THIS
          REAL*8,POINTER :: &
                    pACTH_MS_D2(:)               ! Thickness (numel)


          IF ( .NOT. THIS%p%ISCALCULATED ) THEN
                    PRINT *,'ERROR: NEITHER MESH NOR THICKNESS ARE CALCULATED'
                    RETURN
          END IF

          pACTH_MS_D2 => THIS%ACTH

          RETURN
END FUNCTION pACTH_MS_D2

!_____________________________________________________________ DEASSACTH_MS_D2
!
SUBROUTINE DEASSACTH_MS_D2 (THIS, ACTH)
!         Purpose: Deassigns pointer to thickness vector
!         Arguments:
!                           THIS: FDM2D-type object
!                           ACTH: array containing aquifer thicknesses
!
!         Tested? NO
!   Last modified 30-01-2003
!

          IMPLICIT NONE
          TYPE(T_MS_FDM2D) THIS ! Used only on polymorphism resolution
          REAL*8,POINTER :: &
                    ACTH(:)             ! Thickness (numel)

          NULLIFY(ACTH)

          RETURN
END SUBROUTINE DEASSACTH_MS_D2

!_____________________________________________________________ INTGV_DIV_P_GRAD_x_MS_D2
!
SUBROUTINE INTGV_DIV_P_GRAD_x_MS_D2 (THIS, P, MAT_RES)
!         Purpose: Computes matrix coefficients for this symbolic integral
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: Array containing a 4-component magnitude for each interface
!                           MAT_RES: Matrix to which components must be assigned
!
!         Tested? YES
!   Last modified 06-03-2003
!
          USE M_MATRIX, ONLY : T_MATRIX, INDXD_ADD_

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          TYPE (T_MATRIX) MAT_RES
          REAL*8 :: &
                    P(:)

          ! Internal variable declaration
          INTEGER*4 :: &
                    NC, NCP, &
                    CELL

          INTEGER*4, POINTER :: &
                    VALPOS_R(:), &
                    VALPOS_C(:)

          REAL*8, POINTER :: &
                    VALUES(:)

          NC = 9              ! Number of components to add to MAT_RES
          NCP = 4             ! Number of components of P for each cell

          ALLOCATE (VALUES(NC), VALPOS_R(NC), VALPOS_C(NC))


          DO CELL=1,THIS%p%NUMNP

                    VALPOS_R = CELL

                    CALL CASE9_COMPONENTS(THIS, P, VALUES, VALPOS_C, CELL)

                    ! Changes sign
                    VALUES = -VALUES

                    CALL INDXD_ADD_(MAT_RES, VALUES, VALPOS_R, VALPOS_C, NC)

          END DO

          DEALLOCATE (VALUES, VALPOS_R, VALPOS_C)

          RETURN
END SUBROUTINE INTGV_DIV_P_GRAD_x_MS_D2

!_____________________________________________________________ INTGV_P_x_MS_D2
!
SUBROUTINE INTGV_P_x_MS_D2 (THIS, P, MAT_RES)
```

```
!         Purpose: Computes matrix coefficients for this symbolic integral
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: Array containing a 1-component magnitude for each cell
!                           MAT_RES: Matrix to which components must be assigned
!
!         Tested? NO
!   Last modified 06-03-2003
!
          USE M_MATRIX, ONLY : T_MATRIX, INDXD_ADD_

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          TYPE (T_MATRIX) MAT_RES
          REAL*8 :: &
                    P(:)

          ! Internal variable declaration
          INTEGER*4 :: &
                    NC, NCP, &
                    CELL

          INTEGER*4, POINTER :: &
                    VALPOS_R(:), &
                    VALPOS_C(:)

          REAL*8, POINTER :: &
                    VALUES(:)

          NC = 1              ! Number of components to add to MAT_RES
          NCP = 1             ! Number of components of P for each cell

          ALLOCATE (VALUES(NC), VALPOS_R(NC), VALPOS_C(NC))

          DO CELL=1,THIS%p%NUMEL

                    VALUES = P(CELL)
                    VALPOS_R = CELL
                    VALPOS_C = CELL

                    CALL INDXD_ADD_(MAT_RES, VALUES, VALPOS_R, VALPOS_C, NC)

          END DO

          DEALLOCATE (VALUES, VALPOS_R, VALPOS_C)

          RETURN
END SUBROUTINE INTGV_P_x_MS_D2

!_____________________________________________________________ INTGV_P_GRAD_x_MS_D2
!
SUBROUTINE INTGV_P_GRAD_x_MS_D2 (THIS, P, MAT_RES)
!         Purpose: Computes matrix coefficients for this symbolic integral
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: Array containing a 2-component magnitude for each cell
!                           MAT_RES: Matrix to which components must be assigned
!
!         Tested? NO
!   Last modified 06-03-2003
!
          USE M_MATRIX, ONLY : T_MATRIX, INDXD_ADD_

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          TYPE (T_MATRIX) MAT_RES
          REAL*8 :: &
                    P(:)

          ! Internal variable declaration
          INTEGER*4 :: &
                    NC, NCP, NTOT, &
                    CELL

          INTEGER*4, POINTER :: &
                    VALPOS_R(:), &
                    VALPOS_C(:)

          REAL*8, POINTER :: &
                    VALUES(:)

          NC = 5              ! Number of components to add to MAT_RES
          NCP = 2             ! Number of components of P for each interface
          NTOT=NCP*THIS%p%NUMVP

          ALLOCATE (VALUES(NC), VALPOS_R(NC), VALPOS_C(NC))

          DO CELL=1,THIS%p%NUMNP

                    VALPOS_R = CELL

                    CALL CASE5_COMPONENTS(THIS, P(1:NTOT), VALUES, VALPOS_C, CELL)

                    ! Changes sign
                    VALUES = -VALUES

                    CALL INDXD_ADD_(MAT_RES, VALUES, VALPOS_R, VALPOS_C, NC)

          END DO

          DEALLOCATE (VALUES, VALPOS_R, VALPOS_C)
```

```
          RETURN
END SUBROUTINE INTGV_P_GRAD_x_MS_D2

!_____________________________________________________________ P_GRAD_s_MS_D2
!
SUBROUTINE P_GRAD_s_MS_D2 (THIS, P, S, RES)
!         Purpose: Computes velocity coefficients
!         Arguments:
!                           P(:) : Transmissivity tensor (ndim*ndim*numvp). Only needed 2*numvp components
!                           S(:) : Water levels (numnp)
!                           RES(:,:) : Velocity field (ndim, numvp)
!
!         Tested? NO
!   Last modified 06-03-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          REAL*8 :: &
                    P(:), &
                    S(:), &
                    RES(:,:)

          ! Internal variable declarations
          INTEGER*4 :: &
                    CELL

          RES=0

          DO CELL=1,THIS%p%NUMNP

                    CALL VEL_COMPONENTS(THIS, P, S, RES, CELL)

          END DO

          ! Changes sign
          RES = -RES

          RETURN
END SUBROUTINE P_GRAD_s_MS_D2

!_____________________________________________________________ INTGB_F_x_MINUS_x0_MS_D2
!
SUBROUTINE INTGB_F_x_MINUS_x0_MS_D2 (THIS, F, LFIX, MAT_RES, RHS_RES)
!         Purpose:
!         Arguments:
!
!
!         Tested? NO
!   Last modified 06-02-2003
!
          USE M_MATRIX, ONLY: T_MATRIX, INDXD_ADD_, ADD2DIAG_

          IMPLICIT NONE

          TYPE(T_MS_FDM2D) THIS

          LOGICAL :: &
                    LFIX(:)
          REAL*8 :: &
                    F(:)
          TYPE(T_MATRIX) :: &
                    MAT_RES,  &
                    RHS_RES

          INTEGER*4 :: &
                    J,                  &
                    CELL,     &
                    NC,                 &
                    TCELLS,   &
                    INDXVALUES(1)
          REAL*8 :: &
                    VALUES(1)

          NC = 1

          TCELLS = THIS%p%NUMNP

          J=1
          DO CELL=1, TCELLS

                    IF ( LFIX(CELL) ) THEN
                              VALUES(CELL) = F(J)*F(J+1)
                              INDXVALUES(CELL) = CELL
                              CALL INDXD_ADD_(RHS_RES, VALUES, INDXVALUES, NC)
                              CALL ADD2DIAG_(MAT_RES, CELL, F(J))
                    END IF

          J=J+2
          END DO

          RETURN
END SUBROUTINE INTGB_F_x_MINUS_x0_MS_D2

!_____________________________________________________________ INTGB_F_MS_D2
!
SUBROUTINE INTGB_F_MS_D2 (THIS, F, LFIX, MAT_RES, RHS_RES)
!         Purpose:
!         Arguments:
!
!
!         Tested? NO
```

```
!   Last modified 06-02-2003
!
USE M_MATRIX, ONLY: T_MATRIX, INDXD_ADD_

          IMPLICIT NONE
          TYPE(T_MS_FDM2D) THIS
          LOGICAL :: &
                    LFIX(:)
          REAL*8 :: &
                    F(:),     &         ! Field properties: 2*NX*NY (Factor, Constant value)
                    SUBVEC(1)
          TYPE(T_MATRIX) :: &
                    MAT_RES,  &
                    RHS_RES
          INTEGER*4 :: &
                    CELL,     &
                    I,                  &
                    TCELLS,   &
                    NC,       &
                    INDXSUBVEC(1)


          TCELLS= THIS%p%NUMNP
          NC = 1

          DO CELL=1,TCELLS


                    I = 2*CELL-1
                    IF ( LFIX(CELL) ) THEN
                              SUBVEC(1) = F(I)
                              INDXSUBVEC(1) = CELL
                              CALL INDXD_ADD_(RHS_RES, SUBVEC, INDXSUBVEC, NC)
                    END IF

          END DO

          RETURN
END SUBROUTINE INTGB_F_MS_D2

!_____________________________________________________________ INTERP_ELEM2VP_MS_D2
!
SUBROUTINE INTERP_ELEM2VP_MS_D2(THIS, F, RES, NC)
!         Purpose: interpolates a field from elements to cell interfaces
!         Arguments:
!                                       F: Scalar field in elements
!                                       RES: Scalar field at interfaces (result)
!
!         Tested? NO
!   Last modified 06-03-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    NC
          REAL*8 :: &
                    F(:), &             ! Field in elements (numel)
                    RES(:)                        ! Field at interfaces (numvp)

          ! Internal variables declaration
          INTEGER*4 :: &
                    CELL, I1, I2, J, K
          INTEGER*4, POINTER :: &
                    ADJ(:,:), &
                    INTF(:,:)
          REAL*8, POINTER :: &
                    DX(:), &
                    DY(:)
          REAL*8 :: &
                    DXT, DYT

          ALLOCATE( DX(3), DY(3))
          INTF => THIS%CELL_INTERFACES
          ADJ => THIS%ADJACENT_CELLS

          DO CELL=1, THIS%p%NUMNP

                    CALL GET_DXDY(THIS, DX, DY, CELL)

                    IF(INTF(CELL, 1).NE.0) THEN
                              DO K=1, NC

                                        I1 = NC*(CELL-1)+K
                                        I2 = NC*(ADJ(CELL, 1)-1)+K
                                        J = NC*(INTF(CELL,1)-1)+K
                                        DXT = DX(2)+DX(3)

                                        RES(J) = (DX(3)*F(I1)+DX(2)*F(I2))/DXT
                              END DO
                    END IF

                    IF(INTF(CELL, 2).NE.0) THEN
                              DO K=1, NC
                                        I1 = NC*(CELL-1)+K
                                        I2 = NC*(ADJ(CELL, 7)-1)+K
                                        J = NC*(INTF(CELL,2)-1)+K
                                        DYT = DY(2)+DY(3)

                                        RES(J) = (DY(3)*F(I1)+DY(2)*F(I2))/DYT
                              END DO
                    END IF
```

```
          END DO

          NULLIFY(INTF)
          NULLIFY(ADJ)
          DEALLOCATE(DX, DY)

          RETURN
END SUBROUTINE INTERP_ELEM2VP_MS_D2

!_____________________________________________________________ RELAT_GRAD_MS_D2
!
SUBROUTINE RELAT_GRAD_MS_D2(THIS, F, RG)
!         Purpose: calculates relative gradient of field F
!         Arguments:
!                                   F: Scalar field in elements
!                                   RG: Relative gradient (result)
!
!         Tested? NO
!   Last modified 01-04-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          REAL*8 :: &
                    F(:), &
                    RG(:)

          INTEGER*4 :: &
                    I, J, K
          REAL*8 :: &
                    FIMAX, FIMIN, &
                    FMAX, FMIN, &
                    DEN


          FMAX = F(1)
          FMIN = F(1)
          DO I=1, THIS%p%NUMEL
                    IF (F(I).GT.FMAX) THEN
                              FMAX = F(I)
                    END IF
                    IF (F(I).LT.FMIN) THEN
                              FMIN = F(I)
                    END IF
          END DO
          DEN = FMAX - FMIN

          DO I=1, THIS%p%NUMEL
                    FIMAX = 0.0
                    FIMIN = FMAX
                    DO J=1, 8
                              K = THIS%ADJACENT_CELLS(I,J)
                              IF ((F(K).GT.FIMAX).AND.(K.GT.0)) THEN
                                        FIMAX = F(K)
                              END IF
                              IF ((F(K).LT.FIMIN).AND.(K.GT.0)) THEN
                                        FIMIN = F(K)
                              END IF
                    END DO
                    RG(I) = (FIMAX-FIMIN)/DEN
          END DO

          RETURN
END SUBROUTINE RELAT_GRAD_MS_D2

!_____________________________________________________________ ELEM_VOL_MS_D2
!
SUBROUTINE ELEM_VOL_MS_D2(THIS, VOL, CELL)
!         Purpose: calculates volume of a cell
!         Arguments:
!                                   CELL
!                                   VOL
!
!         Tested? NO
!   Last modified 01-04-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL
          REAL*8 :: &
                    VOL

          INTEGER*4 :: &
                    I,J
          REAL*8 :: &
                    DX, DY

          I = THIS%CELL_IJ(CELL, 1)
          J = THIS%CELL_IJ(CELL, 2)
          DX = THIS%DELTA_X(I)
          DY = THIS%DELTA_Y(J)

          VOL = DX*DY*THIS%ACTH(CELL)

          RETURN
END SUBROUTINE ELEM_VOL_MS_D2

!_____________________________________________________________ PART_DISPOSE_MS_D2
!
SUBROUTINE PART_DISPOSE_MS_D2(THIS, N, X, Y, CELL, MODE)
!         Purpose: calculates positions of particles in a cell
```

```
!         Arguments:
!                                   N: number of particles
!                                   X: X-coordinate of particles
!                                   Y: Y-coordinate of particles
!                                   CELL: cell considered
!                                   MODE: type of disposal used. Values are:
!                                             1 = CENTERED
!                                             2 = UNIFORM
!                                             3 = RANDOM
!
!         Tested? NO
!   Last modified 18-03-2004
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    N, &
                    CELL, &
                    MODE
          REAL*8 :: &
                    X(:), &
                    Y(:)


          INTEGER*4 :: &
                    CI, CJ, &
                    I, J, K, M
          REAL*8 :: &
                    CX, CY, &
                    DX, DY

          CI=THIS%CELL_IJ(CELL,1)
          CJ=THIS%CELL_IJ(CELL,2)
          CX = COORD_X(THIS, CI)
          CY = COORD_Y(THIS, CJ)
          DX = THIS%DELTA_X(CI)
          DY = THIS%DELTA_Y(CJ)

          SELECT CASE (MODE)

                    CASE (1)
                    ! Disposes all particles at central point of a cell

                              DO K=1, N
                                        X(K) = CX
                                        Y(K) = CY
                              END DO

                    CASE (2)
                    ! Disposes particles in a grid scheme
                    ! NOTE: Number of particles per cell has to be a perfect square number

                              M = N**(0.5)

                              K=1
                              DO I=1, M
                              DO J=1, M

                                        X(K) = CX - DX/2 + (J-0.5)*DX/M
                                        Y(K) = CY - DY/2 + (I-0.5)*DY/M

                                        K=K+1
                              END DO
                              END DO

                    CASE (3)
                    ! Disposes particles randomly
                    ! Not yet built. Now acting as CENTERED

                              DO K=1, N
                                        X(K) = CX
                                        Y(K) = CY
                              END DO

                    CASE DEFAULT
                              PRINT *,'ERROR IN PART_DISPOSE_MS_D2: MODE UNKNOWN'
                              RETURN

          END SELECT

          RETURN
END SUBROUTINE PART_DISPOSE_MS_D2

!_____________________________________________________________ MOVE_POINT_MS_D2
!
SUBROUTINE MOVE_POINT_MS_D2(THIS, N, X, Y, V)
!         Purpose: calculates new positions of points after a movement
!         Arguments:
!                                   N: number of particles
!                                   X: X-coordinate of particles
!                                   Y: Y-coordinate of particles
!                                   V: displacement vector
!
!         Tested? NO
!   Last modified 27-11-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    N
          REAL*8 :: &
```

```
                    X(:), &
                    Y(:), &
                    V(:)

          INTEGER*4 :: &
                    I
          REAL*8 :: &
                    VX, VY, &
                    NEWX, NEWY

          DO I = 1, N
                    CALL POINT_VEC (THIS, V, X(I), Y(I), VX, VY)

                    NEWX = X(I) + VX
                    NEWY = Y(I) + VY

                    IF (IS_OUTSIDE(THIS, NEWX, NEWY)) THEN
                              CALL REFLECT (THIS, X(I), Y(I), VX, VY, NEWX, NEWY)
                    ELSE
                              X(I) = NEWX
                              Y(I) = NEWY
                    END IF
          END DO

          RETURN
END SUBROUTINE MOVE_POINT_MS_D2

!_____________________________________________________________ PART2ELEM_MS_D2
!
SUBROUTINE PART2ELEM_MS_D2 (THIS, N, VOL, EL, CP, C)
!         Purpose: calculates new cell concentracions from particle concentrations
!         Arguments:
!                                       N: number of particles
!                                       VOL: particle initial volumes
!                                       EL: element in which is included a particle
!                                       C: node concentrations at intermediate time
!                                       CP: particle concentrations
!
!         Tested? NO
!   Last modified 17-03-2004
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    N
          INTEGER*4 :: &
                    EL(:)
          REAL*8 :: &
                    VOL(:), &
                    CP(:), &
                    C(:)

          INTEGER*4 :: &
                    I, J, K, &
                    NEL
          REAL*8, POINTER :: &
                    TOTALVOL(:), &
                    CEL(:)                                          ! Concentrations at elements

          NEL = THIS%p%NUMEL

          ALLOCATE (TOTALVOL(NEL))
          ALLOCATE (CEL(NEL))

          ! Creating total volume array
          TOTALVOL=0
          DO I=1, N
                    TOTALVOL(EL(I))=TOTALVOL(EL(I))+VOL(I)
          END DO

          ! Creating element concentration array

          CEL=0
          DO I=1, N
                    CEL(EL(I)) = CEL(EL(I)) + VOL(I)*CP(I)/TOTALVOL(EL(I))
          END DO

          ! Obtaining node concentration array from element concentrations
          ! Provisionally, no change is made.

          C = CEL

          DEALLOCATE (CEL)

          RETURN
END SUBROUTINE PART2ELEM_MS_D2

!_____________________________________________________________ PART_UPDATE_MS_D2
!
SUBROUTINE PART_UPDATE_MS_D2 (THIS, DC, N, PC, X, Y)
!         Purpose: interpolates particle concentrations from cell concentrations
!         Arguments:
!                                       N: number of particles
!                                       DC: cell concentration variation
!                                       PC: particle concentrations
!                                       X: X-coordinate of particles
!                                       Y: Y-coordinate of particles
!
!         Tested? NO
!   Last modified 17-03-2004
!
          IMPLICIT NONE
```

```
          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    N
          REAL*8 :: &
                    DC(:), PC(:), &
                    X(:), Y(:)

          INTEGER*4 :: &
                    I, J


          ! Variant 1: inclusion
          ! Relation between particle concentrations and cell concentrations only takes into
          ! account the cells particles belong to.

          DO I=1, N
                    J = GET_CELL(THIS, X(I), Y(I))
                    PC(I) = PC(I) + DC(J)
                    IF (PC(I).LT.0.0) THEN
                              PC(I) = 0.0
                    END IF
          END DO


          RETURN
END SUBROUTINE PART_UPDATE_MS_D2

!_____________________________________________________________ SCALAR_FIELD_MS_D2
!
SUBROUTINE SCALAR_FIELD_MS_D2 (THIS, F, N, X, Y, PF)
!         Purpose: obtains the value of a node field in some given points
!         Arguments:
!                                        N: number of points
!                                        X: X-coordinate of points
!                                        Y: Y-coordinate of points
!                                        F: given field
!                                        PF: value of the field at the points
!
!         Tested? NO
!   Last modified 20-03-2004
!

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) :: THIS
          INTEGER*4 :: &
                    N
          REAL*8 :: &
                    X(:), &
                    Y(:), &
                    F(:), &
                    PF(:)

          INTEGER*4 :: &
                    I, J, &
                    CELL

          ! Variant used: inclusion.
          DO I = 1, N
                    CELL = GET_CELL(THIS, X(I), Y(I))
                    PF(I) = F(CELL)
          END DO

          RETURN
END SUBROUTINE SCALAR_FIELD_MS_D2


!_____________________________________________________________________________________
!
!                                                        MAIN PRIVATE SUBROUTINES
!_____________________________________________________________________________________


!_____________________________________________________________ READ_MOSTSIMPLE
!
SUBROUTINE READ_MOSTSIMPLE (THIS, FILENAME, IUNIT)
!         Purpose: reads object attributes from MOSTSIMPLE type file
!         Arguments:
!                             THIS: FDM2D-type object
!                             FILENAME: file name
!                             IUNIT: unit number for this file
!                             AIERROR
!
!         Tested? NO
!   Last modified 06-02-2003
!
          IMPLICIT NONE

          TYPE(T_MS_FDM2D):: &
                    THIS

          INTEGER*4 :: &
                    IUNIT,    &
                    AIERROR,  &
                    INDOM,    &
                    K, I,     &
                    NDOM,     &
                    TCELLS,  &
                    NX, NY

          CHARACTER*100 :: &
                    FILENAME
```

```
          CHARACTER*80 :: &
                    DUMMY

          OPEN (UNIT=IUNIT, FILE=FILENAME, STATUS='UNKNOWN')

          READ (IUNIT, 100) NX                              ! Mesh dimension X
          READ (IUNIT, 100) NY                              ! Mesh dimension Y

100       FORMAT (I6)

          THIS%MAX_I = NX
          THIS%MAX_J = NY
          TCELLS = THIS%MAX_I*THIS%MAX_J

          ALLOCATE (THIS%DELTA_X(THIS%MAX_I), THIS%DELTA_Y(THIS%MAX_J))
          ALLOCATE (THIS%IS_INDOMAIN(TCELLS))

          READ (IUNIT, 110) (THIS%DELTA_X(K), K=1,(THIS%MAX_I))      ! X mesh intervals
          READ (IUNIT, 110) (THIS%DELTA_Y(K), K=1,(THIS%MAX_J))      ! Y mesh intervals

110       FORMAT (F6.3)

          NDOM = 0

          DO K=1,TCELLS

                    READ (IUNIT, 115) INDOM                 ! IS_INDOMAIN property

115                 FORMAT (I1)

                    SELECT CASE (INDOM)
                              CASE (1)
                                        THIS%IS_INDOMAIN(K) = .TRUE.
                                        NDOM=NDOM+1
                              CASE (0)
                                        THIS%IS_INDOMAIN(K) = .FALSE.
                              CASE DEFAULT
                                        PRINT *, 'Error: Unknown value for IS_INDOMAIN'
                    END SELECT

          END DO

          THIS%p%NUMEL = NDOM
          THIS%p%NUMNP = NDOM

          ALLOCATE (THIS%ACTH(NDOM))

          I=0
          DO K=1, TCELLS
                    IF (THIS%IS_INDOMAIN(K)) THEN
                              I=I+1
                              READ (IUNIT, 120) THIS%ACTH(I)                    ! Aquifer thicknesses
                    ELSE
                              READ (IUNIT, '(A80)') DUMMY
                    END IF
          END DO

120       FORMAT(F8.3)

          CLOSE (IUNIT)

          RETURN

END SUBROUTINE READ_MOSTSIMPLE


!_____________________________________________________________ CASE9_COMPONENTS
!
SUBROUTINE CASE9_COMPONENTS(THIS, P, VALUES, VALPOS, CELL)
!         Purpose: constructs coefficients of matrix due to flux or dispersion
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: 4-components magnitude for each cell
!                           VALUES:   values to assign to matrix
!                           VALPOS: column position of those values
!                           CELL: Cell number inside domain
!
!         Tested? YES
!   Last modified 24-02-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL, &
                    VALPOS(:), &
                    NC
          REAL*8 :: &
                    P(:), &
                    VALUES(:)

          ! Internal variable declarations
          INTEGER*4 :: &
                    INTF, ADJ
          REAL*8, POINTER :: &
                    M(:,:), &
                    DX(:), &
                    DY(:), &
                    WX(:), &
                    WY(:)

          ALLOCATE( M(4,2), DX(3), DY(3), WX(2), WY(2))
```

```
M=0
DX=0
DY=0
WX=0
WY=0

VALUES=0
VALPOS=0

! Assign values to DX, DY, checking for the edge of the ref. rectangle
CALL GET_DXDY(THIS, DX, DY, CELL)

! Needed components of tensor
! Components Pxx, Pxy in (2)
ADJ = THIS%ADJACENT_CELLS(CELL, 1)
IF (ADJ .NE. 0) THEN
          INTF = THIS%CELL_INTERFACES(CELL,1)
          M(1,1) = P(4*(INTF-1)+1)
          M(1,2) = P(4*(INTF-1)+2)
ELSE
          M(1,1) = 0
          M(1,2) = 0
END IF

! Components Pyy, Pyx in (4)
ADJ = THIS%ADJACENT_CELLS(CELL, 3)
IF (ADJ .NE. 0) THEN
          INTF = THIS%CELL_INTERFACES(ADJ,2)
          M(2,2) = P(4*(INTF-1)+4)
          M(2,1) = P(4*(INTF-1)+3)
ELSE
          M(2,1) = 0
          M(2,2) = 0
END IF

! Components Pxx, Pxy in (6)
ADJ = THIS%ADJACENT_CELLS(CELL, 5)
IF (ADJ .NE. 0) THEN
          INTF = THIS%CELL_INTERFACES(ADJ,1)
          M(3,1) = P(4*(INTF-1)+1)
          M(3,2) = P(4*(INTF-1)+2)
ELSE
          M(3,1) = 0
          M(3,2) = 0
END IF

! Components Pyy, Pyx in (8)
ADJ = THIS%ADJACENT_CELLS(CELL, 7)
IF (ADJ .NE. 0) THEN
          INTF = THIS%CELL_INTERFACES(CELL,2)
          M(4,2) = P(4*(INTF-1)+4)
          M(4,1) = P(4*(INTF-1)+3)
ELSE
          M(4,1) = 0
          M(4,2) = 0
END IF

! Weighting factors (as defined in MT3D)
WX(1) = DX(1)/(DX(1)+DX(2))
WX(2) = DX(3)/(DX(2)+DX(3))

WY(1) = DY(1)/(DY(1)+DY(2))
WY(2) = DY(3)/(DY(3)+DY(2))


! Local scheme of cell and adjacents
!
!                   7 8 9
!                   6 1 2
!                   5 4 3
!
! Cell number 1 corresponds to CELL whose coefficients being calculated

! X-direction. Points 6-1-2. Diagonal coefficients
!         Point 1
!         Point 2
!         Point 6
! X-direction. Points 6-1-2. Mixed coefficients
!         Point 1
!         Point 2
!         Point 6
! X-direction. Values for points 5-3-7-9, due to interpolation
!         Point 3
!         Point 5
!         Point 7
!         Point 9


! Y-direction. Points 4-1-8. Mixed coefficients
!         Point 1
!         Point 4
!         Point 8
! Y-direction. Points 4-1-8. Diagonal coefficients
!         Point 1
!         Point 4
!         Point 8
! Y-direction. Values for points 5-3-7-9, due to interpolation
!         Point 3
!         Point 5
!         Point 7
!         Point 9
```

```
! Component 1 (always existent)

VALUES(1) = -M(1,1)/(DX(2)*(DX(2)+DX(3))/2) &
                            -M(3,1)/(DX(2)*(DX(2)+DX(1))/2) &
                            -M(2,2)/(DY(2)*(DY(2)+DY(3))/2) &
                            -M(4,2)/(DY(2)*(DY(2)+DY(1))/2)

VALPOS(1) = CELL

! Component 2
IF (THIS%ADJACENT_CELLS(CELL, 1).NE.0) THEN
          VALUES(2) = M(1,1)/(DX(2)*(DX(2)+DX(3))/2) &
                                      +WY(2)*M(2,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2))) &
                                      -(1-WY(1))*M(4,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))


          VALPOS(2) = THIS%ADJACENT_CELLS (CELL, 1)
ELSE
          VALUES(2) = 0
          VALPOS(2) = CELL
END IF

! Component 3
IF (THIS%ADJACENT_CELLS(CELL, 2).NE.0) THEN
          VALUES(3) = (1-WX(2))*M(1,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      +(1-WY(2))*M(2,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))

          VALPOS(3) = THIS%ADJACENT_CELLS (CELL, 2)
ELSE
          VALUES(3) = 0
          VALPOS(3) = CELL
END IF

! Component 4
IF (THIS%ADJACENT_CELLS(CELL, 3).NE.0) THEN
          VALUES(4) = WX(2)*M(1,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      -(1-WX(1))*M(3,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      +M(2,2)/(DY(2)*(DY(2)+DY(3))/2)

          VALPOS(4) = THIS%ADJACENT_CELLS (CELL, 3)
ELSE
          VALUES(4) = 0
          VALPOS(4) = CELL
END IF

! Component 5
IF (THIS%ADJACENT_CELLS(CELL, 4).NE.0) THEN
          VALUES(5) = -WX(1)*M(3,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      -(1-WY(2))*M(2,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))

          VALPOS(5) = THIS%ADJACENT_CELLS (CELL, 4)
ELSE
          VALUES(5) = 0
          VALPOS(5) = CELL
END IF

! Component 6
IF (THIS%ADJACENT_CELLS(CELL, 5).NE.0) THEN
          VALUES(6) = M(3,1)/(DX(2)*(DX(2)+DX(1))/2) &
                                      -WY(2)*M(2,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2))) &
                                      +(1-WY(1))*M(4,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))

          VALPOS(6) = THIS%ADJACENT_CELLS (CELL, 5)
ELSE
          VALUES(6) = 0
          VALPOS(6) = CELL
END IF

! Component 7
IF (THIS%ADJACENT_CELLS(CELL, 6).NE.0) THEN
          VALUES(7) = WX(1)*M(3,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      +WY(1)*M(4,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))

          VALPOS(7) = THIS%ADJACENT_CELLS (CELL, 6)
ELSE
          VALUES(7) = 0
          VALPOS(7) = CELL
END IF

! Component 8
IF (THIS%ADJACENT_CELLS(CELL, 7).NE.0) THEN
          VALUES(8) = -WX(2)*M(1,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      +(1-WX(1))*M(3,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      +M(4,2)/(DY(2)*(DY(2)+DY(1))/2)

          VALPOS(8) = THIS%ADJACENT_CELLS (CELL, 7)
ELSE
          VALUES(8) = 0
          VALPOS(8) = CELL
END IF

! Component 9
IF (THIS%ADJACENT_CELLS(CELL, 8).NE.0) THEN
          VALUES(9) = -(1-WX(2))*M(1,2)/(DX(2)*(DY(1)/2+DY(3)/2+DY(2))) &
                                      -WY(1)*M(4,1)/(DY(2)*(DX(1)/2+DX(3)/2+DX(2)))

          VALPOS(9) = THIS%ADJACENT_CELLS (CELL, 8)
ELSE
          VALUES(9) = 0
          VALPOS(9) = CELL
END IF

DEALLOCATE (M, DX, DY, WX, WY)
```

```
          RETURN
END SUBROUTINE CASE9_COMPONENTS

!_____________________________________________________________ CASE5_COMPONENTS
!
SUBROUTINE CASE5_COMPONENTS(THIS, P, VALUES, VALPOS, CELL)
!         Purpose: constructs coefficients of matrix due to advection
!         Arguments:
!                              THIS: FDM2D-type object
!                              P: 2-components magnitude for each interface
!                              VALUES:   values to assign to matrix
!                              VALPOS: column position of those values
!                              CELL: Cell number inside domain
!
!         Tested? NO
!   Last modified 31-01-2003
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL, &
                    VALPOS(:)
          REAL*8 :: &
                    P(:), &
                    VALUES(:)

          ! Internal variable declarations
          INTEGER*4 :: &
                    ADJ, INTF
          REAL*8, POINTER :: &
                    M(:), &
                    DX(:), &
                    DY(:), &
                    WX(:), WY(:)


          ALLOCATE( M(4), DX(3), DY(3), WX(2), WY(2))

          M=0
          DX=0
          DY=0
          VALUES=0
          VALPOS=CELL


          ! Assign values to DX, DY, checking for the edge of the ref. rectangle

          CALL GET_DXDY(THIS, DX, DY, CELL)

          ! Prepare needed coefficients
          ! qx_i+1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 1)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(CELL, 1)
                    M(1) = P(2*(INTF-1)+1)
          ELSE
                    M(1) = 0
          END IF

          ! qx_i-1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 5)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(ADJ, 1)
                    M(3) = P(2*(INTF-1)+1)
          ELSE
                    M(3) = 0
          END IF

          ! qy_j-1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 3)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(ADJ, 2)
                    M(2) = P(2*(INTF-1)+2)
          ELSE
                    M(2) = 0
          END IF

          ! qy_j+1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 7)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(CELL, 2)
                    M(4) = P(2*(INTF-1)+2)
          ELSE
                    M(4) = 0
          END IF


          ! Weighting factors
          WX(2)= DX(3)/(DX(2)+DX(3))
          WX(1)= DX(1)/(DX(1)+DX(2))

          WY(2)= DY(3)/(DY(1)+DY(3))
          WY(1)= DY(1)/(DY(1)+DY(2))


          ! Creation of values
          ! Scheme of cell and adjacents
```

```
          !
          !                     5
          !                   4 1 2
          !                     3
          !
          ! # stands for CELL

          ! X-direction. Points 1-2-4
          !         Point 1
          VALUES(1) = VALUES(1)  &
                                          + WX(2)/DX(2)*M(1) &
                                          - WX(1)/DX(2)*M(3)

          !         Point 2
          VALUES(2) = VALUES(2) &
                                          + (1-WX(2))/DX(2)*M(1)
          IF(THIS%ADJACENT_CELLS(CELL, 1).NE.0) THEN
                    VALPOS(2) = THIS%ADJACENT_CELLS(CELL, 1)
          END IF
          !         Point 4
          VALUES(4) = VALUES(4) &
                                          -(1-WX(1))/DX(2)*M(3)
          IF(THIS%ADJACENT_CELLS(CELL, 5).NE.0) THEN
                    VALPOS(4) = THIS%ADJACENT_CELLS(CELL, 5)
          END IF
          ! Y-direction. Points 1-3-5
          !         Point 1
          VALUES(1) = VALUES(1) &
                                          + WY(2)/DY(2)*M(4) &
                                          - WY(1)/DY(2)*M(2)
          !         Point 3
          VALUES(3) = VALUES(3) &
                                          +(1-WY(1))/DY(2)*M(2)
          IF(THIS%ADJACENT_CELLS(CELL, 3).NE.0) THEN
                    VALPOS(3) = THIS%ADJACENT_CELLS(CELL, 3)
          END IF
          !         Point 5
          VALUES(5) = VALUES(5) &
                                          -(1-WY(2))/DY(2)*M(4)
          IF(THIS%ADJACENT_CELLS(CELL, 7).NE.0) THEN
                    VALPOS(5) = THIS%ADJACENT_CELLS(CELL, 7)
          END IF

          DEALLOCATE (M, DX, DY, WX, WY)

          RETURN
END SUBROUTINE CASE5_COMPONENTS


!_____________________________________________________________ VEL_COMPONENTS
!
SUBROUTINE VEL_COMPONENTS(THIS, P, S, RES, CELL)
!         Purpose: to construct velocity components (P*grad(s))
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: tensor of 2nd order (4*numvp)
!                           S: vector (numnp) (usually heads)
!                           RES: vector to be constructed (2, numvp)
!                           CELL: cell which coefficients are being constructed
!
!         Tested? NO
!   Last modified 17-03-2004
!
          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL
          REAL*8 :: &
                    P(:), &
                    S(:), &
                    RES(:,:)
          ! Internal variable declarations
          INTEGER*4 :: &
                    ADJ, INTF, &
                    INTF1, INTF2
          REAL*8, POINTER :: &
                    M(:,:), &
                    H(:,:), &
                    DX(:), &
                    DY(:)

          ALLOCATE(M(2,2), H(2,3), DX(3), DY(3))

          M=0
          DX=0
          DY=0

          ! Get DX and DY values
          CALL GET_DXDY(THIS, DX, DY, CELL)

          ! Prepare needed coefficients
          ! Components Pxx, Pxy in (2)
          ADJ = THIS%ADJACENT_CELLS(CELL, 1)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(CELL,1)
                    M(1,1) = P(4*(INTF-1)+1)
                    M(1,2) = P(4*(INTF-1)+2)
          ELSE
                    M(1,1) = 0
                    M(1,2) = 0
          END IF
```

```
          ! Components Pyy, Pyx in (8)
          ADJ = THIS%ADJACENT_CELLS(CELL, 7)
          IF (ADJ .NE. 0) THEN
                    INTF = THIS%CELL_INTERFACES(CELL,2)
                    M(2,2) = P(4*(INTF-1)+4)
                    M(2,1) = P(4*(INTF-1)+3)
          ELSE
                    M(2,1) = 0
                    M(2,2) = 0
          END IF

          ! Get necessary S values
          CALL INTERP_LEVELS(THIS, S, H, DX, DY, CELL)

          ! Construct values
          ! RES(2, INTF1) = 0
          ! RES(1, INTF2) = 0

          INTF1 = THIS%CELL_INTERFACES(CELL, 1)
          IF (INTF1.NE.0) THEN
                    RES(1, INTF1) = M(1,1)*(H(1,1)-S(CELL))/((DX(2)+DX(3))/2) &
                                          + M(1,2)*(H(1,2)-H(1,3))/(DY(2))
          END IF

          INTF2 = THIS%CELL_INTERFACES(CELL, 2)
          IF (INTF2.NE.0) THEN
                    RES(2, INTF2) = M(2,2)*(H(2,1)-S(CELL))/((DY(2)+DY(3))/2) &
                                          + M(2,1)*(H(2,2)-H(2,3))/(DX(2))
          END IF


          RETURN
END SUBROUTINE VEL_COMPONENTS
!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________

!_____________________________________________________________________________________
!
!                                                    AUXILIARY PRIVATE SUBROUTINES
!_____________________________________________________________________________________


!_____________________________________________________________ GET_DXDY
!
SUBROUTINE GET_DXDY(THIS, DX, DY, CELL)
!         Purpose: to correct DX and Dy, paying attention to edges of ref. rectangle
!         Arguments:
!                           THIS: FDM2D-type object
!
!         Tested? NO
!   Last modified 06-02-2003
!

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    I, J, &
                    CELL
          REAL*8 :: &
                    DX(:), DY(:)

          ! Position of cell referred to ref.rectangle
          I = THIS%CELL_IJ (CELL, 1)
          J = THIS%CELL_IJ (CELL, 2)


          IF(I.EQ.1) THEN
                    DX(1) = THIS%DELTA_X(I)
          ELSE
                    DX(1) = THIS%DELTA_X(I-1)
          END IF

          IF(I.EQ.THIS%MAX_I) THEN
                    DX(3) = THIS%DELTA_X(I)
          ELSE
                    DX(3) = THIS%DELTA_X(I+1)
          END IF

          IF(J.EQ.1) THEN
                    DY(1) = THIS%DELTA_Y(J)
          ELSE
                    DY(1) = THIS%DELTA_Y(J-1)
          END IF

          IF(J.EQ.THIS%MAX_J) THEN
                    DY(3) = THIS%DELTA_Y(J)
          ELSE
                    DY(3) = THIS%DELTA_Y(J+1)
          END IF

          DX(2) = THIS%DELTA_X(I)
          DY(2) = THIS%DELTA_Y(J)

          RETURN
END SUBROUTINE GET_DXDY


!_____________________________________________________________ GET_CELL
!
FUNCTION GET_CELL(THIS, X, Y)
```

```
!         Purpose: Finds the cell which point X,Y lies on. It returns zero
!                  value if point lies outside the domain
!         Arguments:
!                           THIS: FDM2D-type object
!                           X: x-coordinate
!                           Y: y-coordinate
!
!         Tested? NO
!   Last modified 17-03-2004
!

          IMPLICIT NONE
          TYPE(T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    GET_CELL
          REAL*8 :: &
                    X, Y

          INTEGER*4 :: &
                    I, J, &
                    ABSP, &
                    K, CELL
          REAL*8 :: &
                    CX, CY

          CX=0
          CY=0
          I=0
          J=0

          ! Finding I
          IF (X.LT.0) THEN
                    GET_CELL=0
                    RETURN
          END IF
          DO WHILE (CX.LT.X)
                    I = I + 1
                    IF (I.LE.THIS%MAX_I) THEN
                              CX = CX + THIS%DELTA_X(I)
                    ELSE
                              GET_CELL=0
                              RETURN
                    END IF
          END DO

          ! Finding J
          IF (Y.LT.0) THEN
                    GET_CELL=0
                    RETURN
          END IF
          DO WHILE (CY.LT.Y)
                    J = J + 1
                    IF (J.LE.THIS%MAX_J) THEN
                              CY = CY + THIS%DELTA_Y(J)
                    ELSE
                              GET_CELL=0
                              RETURN
                    END IF
          END DO

          ! Skip if cell does not belong to domain
          ABSP = THIS%MAX_I*(J-1)+I
          IF (.NOT.THIS%IS_INDOMAIN(ABSP)) THEN
                    GET_CELL = 0
                    RETURN
          END IF

          ! Find cell position inside domain
          CELL = 0
          DO K=1, ABSP
                    IF (THIS%IS_INDOMAIN(K)) THEN
                              CELL=CELL+1
                    END IF
          END DO

          GET_CELL = CELL

          RETURN
END FUNCTION GET_CELL


!_____________________________________________________________ COORD_X
!
FUNCTION COORD_X (THIS, CELL)
!         Purpose:
!         Arguments:
!                           THIS: FDM2D-type object
!         Tested? NO
!   Last modified 16-03-2004
!
          IMPLICIT NONE
          TYPE (T_MS_FDM2D) THIS
          REAL*8 :: &
                    COORD_X
          INTEGER*4 :: &
                    CELL

          INTEGER*4 :: &
                    I, &
                    CELL_I
          REAL*8:: &
                    X
```

```
          CELL_I = THIS%CELL_IJ(CELL,1)

          X = 0
          DO I=1,CELL_I
                    X=X+THIS%DELTA_X(I)
          END DO
          X=X-THIS%DELTA_X(CELL_I)/2

          COORD_X = X

          RETURN
END FUNCTION


!_____________________________________________________________ COORD_Y
!
FUNCTION COORD_Y (THIS, CELL)
!         Purpose:
!         Arguments:
!                              THIS: FDM2D-type object
!         Tested? NO
!   Last modified 16-03-2004
!
          IMPLICIT NONE
          TYPE (T_MS_FDM2D) THIS
          REAL*8 :: &
                    COORD_Y
          INTEGER*4 :: &
                    CELL

          INTEGER*4 :: &
                    J, &
                    CELL_J
          REAL*8:: &
                    Y

          CELL_J = THIS%CELL_IJ(CELL, 2)

          Y = 0
          DO J=1,CELL_J
                    Y=Y+THIS%DELTA_Y(J)
          END DO
          Y=Y-THIS%DELTA_Y(CELL_J)/2

          COORD_Y = Y

          RETURN
END FUNCTION


!_____________________________________________________________ INTERP_TENSOR_ARIT
!
SUBROUTINE INTERP_TENSOR_ARIT(THIS, P, M, DX, DY, CELL, NC)
!         Purpose:
!         Arguments:
!                              THIS: FDM2D-type object
!                              P: 2nd order tensor in cells (4*numel)
!                              M: local interpolated tensorial values (4 x 2)
!
!         Tested? NO
!   Last modified 06-02-2003
!

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL, &
                    NC, &
                    ADJ
          REAL*8 :: &
                    P(:), &
                    M(:,:), &
                    DX(:), DY(:), &
                    DAUX

          ! Components Dxx, Dxy in (2)
          ADJ = THIS%ADJACENT_CELLS(CELL, 1)
          IF (ADJ .NE. 0) THEN

                    DAUX = (DX(2) + DX(3))

                    M(1,1) = (P(4*(CELL-1)+1)*DX(3) + P(4*(ADJ-1)+1)*DX(2))/DAUX
                    M(1,2) = (P(4*(CELL-1)+2)*DX(3) + P(4*(ADJ-1)+2)*DX(2))/DAUX

          ELSE

                    M(1,1) = 0
                    M(1,2) = 0

          END IF

          ! Components Dyy, Dyx in (4)
          ADJ = THIS%ADJACENT_CELLS(CELL, 3)
          IF (ADJ .NE. 0) THEN

                    DAUX = (DY(2) + DY(3))

                    M(2,2) = (P(4*(CELL-1)+4)*DY(3) + P(4*(ADJ-1)+4)*DY(2))/DAUX
                    M(2,1) = (P(4*(CELL-1)+3)*DY(3) + P(4*(ADJ-1)+3)*DX(2))/DAUX

          ELSE
```

```
                M(2,1) = 0
                M(2,2) = 0

        END IF

        ! Components Dxx, Dxy in (6)
        ADJ = THIS%ADJACENT_CELLS(CELL, 5)
        IF (ADJ .NE. 0) THEN

                DAUX = (DX(2) + DX(1))

                M(3,1) = (P(4*(CELL-1)+1)*DX(1) + P(4*(ADJ-1)+1)*DX(2))/DAUX
                M(3,2) = (P(4*(CELL-1)+2)*DX(1) + P(4*(ADJ-1)+2)*DX(2))/DAUX

        ELSE

                M(3,1) = 0
                M(3,2) = 0

        END IF

        ! Components Dyy, Dyx in (8)
        ADJ = THIS%ADJACENT_CELLS(CELL, 7)
        IF (ADJ .NE. 0) THEN

                DAUX = (DY(2) + DY(1))

                M(4,2) = (P(4*(CELL-1)+4)*DY(1) + P(4*(ADJ-1)+4)*DY(2))/DAUX
                M(4,1) = (P(4*(CELL-1)+3)*DY(1) + P(4*(ADJ-1)+3)*DX(2))/DAUX

        ELSE

                M(4,1) = 0
                M(4,2) = 0

        END IF


        RETURN
END SUBROUTINE INTERP_TENSOR_ARIT


!_____________________________________________________________ INTERP_LEVELS
!
SUBROUTINE INTERP_LEVELS(THIS, S, H, DX, DY, CELL)
!       Purpose:
!       Arguments:
!                       THIS: FDM2D-type object
!                       S: vector in cells (numnp)
!                       H: local interpolated vectorial values (2,3)
!
!       Tested? NO
!   Last modified 28-02-2003
!
        IMPLICIT NONE

        TYPE (T_MS_FDM2D) THIS
        INTEGER*4 :: &
                CELL
        REAL*8 :: &
                S(:), &
                H(:,:), &
                DX(:), DY(:)

        ! Internal variable declarations
        INTEGER*4, POINTER :: &
                ADJ(:)
        REAL*8 :: &
                AUX1, AUX2

        ADJ => THIS%ADJACENT_CELLS(CELL,:)


        ! Level in (i+1,j)
        IF (ADJ(1).NE.0) THEN
                H(1,1) = S(ADJ(1))
        ELSE
                H(1,1) = 0
        END IF

        ! Level in (i+1/2, j+1/2) : H(1,2)
        H(1,2) = 0

        ! Level in (i+1/2, j-1/2) : H(1,3)
        H(1,3) = 0

        ! Level in (i,j+1)
        IF (ADJ(7).NE.0) THEN
                H(2,1) = S(ADJ(7))
        ELSE
                H(2,1) = 0
        END IF

        ! Level in (i+1/2, j+1/2) : H(2,2)
        H(2,2) = H(1,2)

        ! Level in (i-1/2, j+1/2) : H(2,3)
        H(2,3) = 0

        RETURN
END SUBROUTINE INTERP_LEVELS
```

```
!_____________________________________________________________ INTERP_VECTOR
!
SUBROUTINE INTERP_VECTOR(THIS, P, M, DX, DY, CELL)
!         Purpose:
!         Arguments:
!                           THIS: FDM2D-type object
!                           P: vector in cells (2*numnp)
!                           M: local interpolated vectorial values (4)
!
!         Tested? NO
!   Last modified 26-02-2003
!

          IMPLICIT NONE

          TYPE (T_MS_FDM2D) THIS
          INTEGER*4 :: &
                    CELL, &
                    ADJ
          REAL*8 :: &
                    P(:), &
                    M(:), &
                    DX(:), DY(:), &
                    DAUX

          M=0

          ! Storage scheme
          !                   4
          !           3 # 1
          !                   2
          !

          ! qx_i+1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 1)
          IF (ADJ .NE. 0) THEN
                    DAUX = DX(2) + DX(3)
                    M(1) = (P(2*(CELL-1))*DX(3)+P(2*(ADJ-1))*DX(2))/DAUX
          ELSE
                    M(1) = 0
          END IF

          ! qx_i-1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 5)
          IF (ADJ .NE. 0) THEN
                    DAUX = DX(2) + DX(1)
                    M(3) = (P(2*(CELL-1))*DX(1)+P(2*(ADJ-1))*DX(2))/DAUX
          ELSE
                    M(3) = 0
          END IF

          ! qy_j-1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 3)
          IF (ADJ .NE. 0) THEN
                    DAUX = DY(2) + DY(1)
                    M(2) = (P(2*(CELL-1)+1)*DY(1)+P(2*(ADJ-1)+1)*DY(2))/DAUX
          ELSE
                    M(2) = 0
          END IF

          ! qy_j+1/2
          !
          ADJ = THIS%ADJACENT_CELLS(CELL, 7)
          IF (ADJ .NE. 0) THEN
                    DAUX = DY(2) + DY(3)
                    M(4) = (P(2*(CELL-1)+1)*DY(3)+P(2*(ADJ-1)+1)*DY(2))/DAUX
          ELSE
                    M(4) = 0
          END IF


          RETURN
END SUBROUTINE INTERP_VECTOR


!_____________________________________________________________ POINT_VEC
!
SUBROUTINE POINT_VEC (THIS, V, X, Y, VX, VY)
!         Purpose:
!         Arguments:
!                           THIS: FDM2D-type object
!         Tested? NO
!   Last modified 16-03-2004
!

          IMPLICIT NONE
          TYPE(T_MS_FDM2D) THIS
          REAL*8 :: &
                    VX, VY, &
                    X, Y
          REAL*8 :: &
                    V(:)

          INTEGER*4 :: &
                    CELL, &
                    I, J
          INTEGER*4, POINTER :: &
                    IN(:)

          REAL*8 :: &
```

```
                    DX, DY, &
                    CX, CY, &
                    VXA, VXB, &
                    VYA, VYB, &
                    DISTX, DISTY


          ALLOCATE (IN(4))

          CELL = GET_CELL(THIS, X, Y)
          CX = COORD_X(THIS, CELL)
          CY = COORD_Y(THIS, CELL)
          I = THIS%CELL_IJ(CELL, 1)
          J = THIS%CELL_IJ(CELL, 2)
          DX = THIS%DELTA_X(I)
          DY = THIS%DELTA_Y(J)
          DISTX = X - (CX-DX/2)
          DISTY = Y - (CY-DY/2)

          ! Determining interfaces of the cell

          IN(1) = THIS%CELL_INTERFACES(CELL,1)

          IF (THIS%ADJACENT_CELLS(CELL, 3) .EQ.0) THEN
                    IN(2) = 0
          ELSE
                    IN(2) = THIS%CELL_INTERFACES(THIS%ADJACENT_CELLS(CELL,3),2)
          END IF

          IF (THIS%ADJACENT_CELLS(CELL, 5).EQ.0) THEN
                    IN(3) = 0
          ELSE
                    IN(3) = THIS%CELL_INTERFACES(THIS%ADJACENT_CELLS(CELL,5),1)
          END IF

          IN(4) = THIS%CELL_INTERFACES(CELL,2)


          ! X-Component of velocity
          IF (IN(3).EQ.0) THEN
                    VXA = 0
          ELSE
                    VXA = V(2*(IN(3)-1)+1)
          END IF

          IF (IN(1).EQ.0) THEN
                    VXB = 0
          ELSE
                    VXB = V(2*(IN(1)-1)+1)
          END IF

          IF ((IN(1).EQ.0).OR.(IN(3).EQ.0)) THEN
                    VX = VXA + VXB
          ELSE
                    VX = VXA + DISTX/DX*(VXB-VXA)
          END IF

          ! Y-Component of velocity
          IF (IN(2).EQ.0) THEN
                    VYA = 0
          ELSE
                    VYA = V(2*(IN(2)-1)+2)
          END IF

          IF (IN(4).EQ.0) THEN
                    VYB = 0
          ELSE
                    VYB = V(2*(IN(4)-1)+2)
          END IF

          IF ((IN(2).EQ.0).OR.(IN(4).EQ.0)) THEN
                    VY = VYA + VYB
          ELSE
                    VY = VYA + DISTY/DY*(VYB-VYA)
          END IF

          DEALLOCATE (IN)

          RETURN
END SUBROUTINE POINT_VEC


!_____________________________________________________________ REFLECT
!
SUBROUTINE REFLECT (THIS, XA, YA, VX, VY, XB, YB)
!         Purpose:
!         Arguments:
!                           THIS: FDM2D-type object
!         Tested? NO
!   Last modified 19-05-2003
!
          IMPLICIT NONE
          TYPE (T_MS_FDM2D) THIS
          REAL*8 :: &
                    XA, YA, &
                    VX, VY, &
                    XB, YB

          INTEGER*4 :: &
                    CELL
          REAL*8 :: &
                    CX, CY
```

```
          !CELL = GET_CELL (THIS, XA, YA)
          !CX = COORD_X(THIS, CELL)
          !CY = COORD_Y(THIS, CELL)

          XB = XA             ! Temporarily sets coords at initial position
          YB = YA

          RETURN
END SUBROUTINE REFLECT

!_____________________________________________________________ IS_OUTSIDE
!
FUNCTION IS_OUTSIDE (THIS, X, Y)
!         Purpose:
!         Arguments:
!                             THIS: FDM2D-type object
!         Tested? NO
!   Last modified 19-05-2003
!
          IMPLICIT NONE
          TYPE (T_MS_FDM2D) THIS
          LOGICAL :: &
                    IS_OUTSIDE
          REAL*8 :: &
                    X, Y

          IF (GET_CELL(THIS, X, Y).EQ.0) THEN
                    IS_OUTSIDE = .TRUE.
          ELSE
                    IS_OUTSIDE = .FALSE.
          END IF

          RETURN
END FUNCTION


!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________
!_____________________________________________________________


END MODULE M_MS_FDM2D
```

C. Listado en FORTRAN90 de la subclase TR_TRACONF.

```
MODULE M_TR_TRACONF

!
!******* TRANSPORT EQUATIONS *********
!
! Purpose: Solves transport equations for steady and transient state.
!
! Assumptions/Shortcomes:
!
! Abreviation: TR_TC
!
! General use:
!                   . Create object (CREATE_)
!                   . Reads attributes (READ_)
!                   . Initializes (INIT_)
!                   . Computes system (BUILD_)
!                   . Obtains results (SOLVE_)
!
! additionally:
!                   . computes velocities (COMVEL_)
!                   . computes mass balance (BAL_)
!

          use m_reng

          USE M_TIMEINTG
          USE M_MESH

          USE M_MATRIX

          USE M_FLOW, ONLY : T_FLOW, COMVEL_, COMFLW_

          PUBLIC:: &
                    CREATE_,  & ! Creates: allocates and initalizes
                    DESTROY_, & ! Destroys: deallocates
                    READ_,              & ! Reads TRANSPORT properties
                    SET_,               & ! Sets TRANSPORT properties
                    INIT_,              & ! Initializes problem
                    BUILD_,             & ! Interpolate element properties
                    SOLVE_,             & ! Solves system equation
                    ENDINCR_, & ! Tasks after solving a time increment
                    COMVEL_,  & ! Computes elemental velocities and nodal TRANSPORTs
                    COMBAL_,  & ! Computes mass balance for each node
                    pC_,                & ! Points to head values
                    DEASSC_,  & ! Deassigns pointer to head values
                    pFLOW_,             & ! Points to flow equations
                    DEASSFLOW_,         & ! Deassigns pointer to flow equations
                    pMESH_,             & ! Points pointer to mesh
                    DEASSMESH_,         & ! Deassigns pointer to mesh
                    ISTRANSIENT_  ! Indicates if it's transient o not

          PRIVATE:: &
                    CREATE_TR_TC, READ_TR_TC, INIT_TR_TC, BUILD_TR_TC, &
                    COMBAL_TR_TC, pC_TR_TC, DEASSC_TR_TC, DESTROY_TR_TC, &
                    ENDINCR_TR_TC, ISTRANSIENT_TR_TC, SET_TR_TC, &
                    SOLVE_TR_TC_SS, SOLVE_TR_TC_TR, pFLOW_TR_TC, DEASSFLOW_TR_TC,  &
                    pMESH_TR_TC, DEASSMESH_TR_TC, &
                    GET_DISP, GET_STO, CC_TRA, MAT_TRA, VARRAY2V1D,&
                    READ_TRACONF, READ_MOSTSIMPLE, &
                    ! Subroutines used in particle methods
                    PART_INIT, PART_RUN,  &
                    MOC_INIT, MMOC_INIT, RNDWALK_INIT, &
                    MOC_RUN, MMOC_RUN, RNDWALK_RUN, &
                    PART_RESTRICT

!*************************************************************************

! ____________________________________________________________________________________
!
! PARTICLE TYPE
!
! Last modified: 17-03-2004
!
          TYPE, PRIVATE :: T_ADV_PART

                    LOGICAL :: &
                              IS_USED, &                          ! TRUE if particle method is used
                              IS_FIRST                    ! TRUE if it's first time

                    INTEGER*4 :: &
                              NPART, &                    ! Number of particles
                              NPART2, &                   ! Number of particles after correction
                              NPH, &                              ! High number of particles in an element (user)
                              NPL, &                              ! Low number of particles in an element (user)
                              NPMAX, &                    ! Maximum number of particles in an element (user)
                              NPMIN, &                    ! Minimum number of particles in an element (user)
                              METHOD                              ! Type of method used:
                                                                        ! 0 = NONE
                                                                        ! 1 = MOC
                                                                        ! 2 = MMOC
                                                                        ! 3 = Random Walk

                    INTEGER*4, POINTER :: &
                              PARTSINELEM(:), &   ! Number of particles in an element (numel)
```

```
                    ELEMENT(:)                              ! Element containing particle

          REAL*8 :: &
                    DCEPS                                   ! Threshold concentration (user)

          REAL*8, POINTER     :: &
                    PX(:), &                        ! X-Coordinate of particle
                    PY(:), &                        ! Y-Coordinate of particle
                    PVOL(:), &                              ! Volume associated to particle
                    PC(:), &                        ! Particle concentration
                    A(:,:), &                       ! Particle advection matrix
                    CN(:)                                   ! Node concentration at intermediate time

          CHARACTER*25 :: &
                    METHOD_NAME                             ! Name of method used


     END TYPE T_ADV_PART
! ____________________________________________________________________________


     TYPE, PRIVATE :: T_MATER_TRANSPORT

          REAL*8 :: &
                    POR,      & ! Porosity
                    DFM,      & ! Molecular diffusion
                    DSL,      & ! Longitudinal dispersivity
                    DST,      & ! Transversal dispersivity
                    CRD                   ! Delay coefficient

     END TYPE T_MATER_TRANSPORT


     TYPE T_TR_TRACONF

          PRIVATE

          TYPE(T_MESH),POINTER :: &
                    pMESH                                           ! Mesh linked to it
          TYPE(T_FLOW),POINTER :: &
                    pFLOW                                           ! Flow linked to it

          LOGICAL :: &
                    IsTransient                                     ! Indicates if it's transient (TRUE) or
it's steady state

          LOGICAL :: &
                    USEPARTICLES

          TYPE(T_ADV_PART), POINTER :: &
                    PART                                            ! Particle properties

          TYPE(T_MATRIX),POINTER :: &
                    GMAT(:)                                         ! Global system matrix and RHS
          TYPE(T_MATRIX),POINTER :: &
                    pA,pD,pB                                ! Individual pointers to GMAT

          TYPE(T_TIMEINTG),POINTER :: &
                    TIMEINTG                                ! Time integral manager

          INTEGER*4 :: &
                    NUMMAT                                          ! Total number of materials
          TYPE(T_MATER_TRANSPORT),POINTER:: &
                    MATER(:)                                ! TRANSPORT materials (nummat)
          INTEGER*4,POINTER:: &
                    INDXMAT(:)                                      ! Index of materials (numel)

          REAL*8 :: &
                    XLAM                                            ! Radioactive desintegration constant

          INTEGER*4,POINTER :: &
                    IBTCO(:)                                ! Boundary condition code (numnp)
          REAL*8,POINTER :: &
                    COEC(:),                                & ! External concentration (numnp)
                    COLD(:)                                           ! Initial concentrations (numnp)

          REAL*8,POINTER :: &
                    C(:)                                              ! Computed concentrations (numnp)

                                                                    ! Values computed previously in mesh
and flow module
          REAL*8,POINTER :: &
                    pACTH(:),                               & ! Element thickness (numel) obtained from mesh
                    CAUDAL(:,:),                            & ! Nodal flux (numnp,nphase=1)
                    VD(:,:,:)                                 ! Darcy velocities (ndim,numvp,nphase=1)

          LOGICAL :: &
                    ISASSIGNED                      ! Indicates if non-computed values points (TRUE)
                                                              ! or allocates (FALSE) the values

     END TYPE T_TR_TRACONF

!***************************************************************************
!                                                       PARAMETERS FOR PSEUDO-MODULE
!                                                       PARTICLE MANAGEMENT
!***************************************************************************
! Last modified: 18-03-2004

     INTEGER*4, PARAMETER ::                  & ! Particle method types
          IP_MOC=1, &                                       !       . MOC method
          IP_MMOC=2, &                              ! . MMOC method
```

```
                IP_RNDWALK=3                            ! . Random Walk method
                !...***add more types****

        CHARACTER*25, PARAMETER ::  & ! Particle method names
                SP_MOC='MOC', &                         !       . MOC method
                SP_MMOC='MMOC', &               ! . MMOC method
                SP_RNDWALK='RNDWALK'            ! . Random Walk method
                !...***add more names****


!*****************************************************************************

        INTERFACE CREATE_
                MODULE PROCEDURE CREATE_TR_TC
        END INTERFACE

        INTERFACE DESTROY_
                MODULE PROCEDURE DESTROY_TR_TC
        END INTERFACE

        INTERFACE READ_
                MODULE PROCEDURE READ_TR_TC
        END INTERFACE

        INTERFACE INIT_
                MODULE PROCEDURE INIT_TR_TC
        END INTERFACE

        INTERFACE BUILD_
                MODULE PROCEDURE BUILD_TR_TC
        END INTERFACE

        INTERFACE COMBAL_
                MODULE PROCEDURE COMBAL_TR_TC
        END INTERFACE

        INTERFACE ENDINCR_
                MODULE PROCEDURE ENDINCR_TR_TC
        END INTERFACE

        INTERFACE pC_
                MODULE PROCEDURE pC_TR_TC
        END INTERFACE

        INTERFACE DEASSC_
                MODULE PROCEDURE DEASSC_TR_TC
        END INTERFACE

        INTERFACE pFLOW_
                MODULE PROCEDURE pFLOW_TR_TC
        END INTERFACE

        INTERFACE DEASSFLOW_
                MODULE PROCEDURE DEASSFLOW_TR_TC
        END INTERFACE

        INTERFACE pMESH_
                MODULE PROCEDURE pMESH_TR_TC
        END INTERFACE

        INTERFACE DEASSMESH_
                MODULE PROCEDURE DEASSMESH_TR_TC
        END INTERFACE

        INTERFACE ISTRANSIENT_
                MODULE PROCEDURE ISTRANSIENT_TR_TC
        END INTERFACE

        INTERFACE SOLVE_
                MODULE PROCEDURE SOLVE_TR_TC_SS
                MODULE PROCEDURE SOLVE_TR_TC_TR
        END INTERFACE

        CONTAINS
!_________________________________________________________________________

!_____________________________________________________________ CREATE_TR_TC


SUBROUTINE CREATE_TR_TC (THIS, MESH, FLOW, AIERROR)

        IMPLICIT NONE
        TYPE(T_TR_TRACONF) THIS
        TYPE(T_MESH), POINTER:: &
                MESH                    ! associated mesh
        TYPE(T_FLOW), POINTER:: &
                FLOW                    ! associated flow
        INTEGER*4 :: &
                AIERROR

        AIERROR = 0

        ! Set pointer to mesh

        IF ( .NOT. ASSOCIATED(MESH) ) THEN
                AIERROR = -1
                PRINT *,'ERROR MESH NOT ASSOCIATED IN CREATE_TR_TC'
        END IF

        THIS%pMESH => MESH

        IF ( .NOT. ASSOCIATED(FLOW) ) THEN
                AIERROR = -1
```

```
                PRINT *,'ERROR FLOW NOT ASSOCIATED IN CREATE_TR_TC'
        END IF

        THIS%pFLOW => FLOW

        ! Default values

        THIS%ISTRANSIENT = .FALSE.

        THIS%NUMMAT = 0

        NULLIFY(THIS%MATER)
        NULLIFY(THIS%INDXMAT)

        THIS%XLAM = 0D0

        NULLIFY(THIS%IBTCO)
        NULLIFY(THIS%COEC)
        NULLIFY(THIS%COLD)

        NULLIFY(THIS%C)

        NULLIFY(THIS%pACTH)
        NULLIFY(THIS%VD)
        NULLIFY(THIS%CAUDAL)

        THIS%ISASSIGNED = .FALSE.
        THIS%USEPARTICLES = .FALSE.

        NULLIFY(THIS%TIMEINTG)
        NULLIFY(THIS%GMAT)
        NULLIFY(THIS%pA)
        NULLIFY(THIS%pD)
        NULLIFY(THIS%pB)

        NULLIFY(THIS%PART)

        RETURN
END SUBROUTINE CREATE_TR_TC

!_____________________________________________________________________ DESTROY_TR_TC

SUBROUTINE DESTROY_TR_TC (THIS)

        IMPLICIT NONE
        TYPE(T_TR_TRACONF) THIS

        INTEGER*4 I

        IF ( ASSOCIATED(THIS%pMESH) .AND. ASSOCIATED(THIS%pACTH) ) THEN
                CALL DEASSACTH_(THIS%pMESH, THIS%pACTH)
        END IF

        NULLIFY(THIS%pMESH)
        NULLIFY(THIS%pFLOW)

        IF ( ASSOCIATED(THIS%TIMEINTG) ) THEN
                CALL DESTROY_(THIS%TIMEINTG)
                DEALLOCATE(THIS%TIMEINTG)
        END IF

        IF ( ASSOCIATED(THIS%MATER) ) THEN
                DEALLOCATE(THIS%MATER)
        END IF
        THIS%NUMMAT = 0

        THIS%XLAM = 0D0

        if ( .NOT. this%IsAssigned ) then
                IF ( ASSOCIATED(THIS%INDXMAT) ) THEN
                        DEALLOCATE(THIS%INDXMAT)
                END IF
                IF ( ASSOCIATED(THIS%IBTCO) ) THEN
                        DEALLOCATE(THIS%IBTCO)
                END IF
                IF ( ASSOCIATED(THIS%COEC) ) THEN
                        DEALLOCATE(THIS%COEC)
                END IF
                IF ( ASSOCIATED(THIS%COLD) ) THEN
                        DEALLOCATE(THIS%COLD)
                END IF
        end if

        IF ( ASSOCIATED(THIS%C) ) THEN
                DEALLOCATE(THIS%C)
        END IF
        IF ( ASSOCIATED(THIS%VD) ) THEN
                DEALLOCATE(THIS%VD)
        END IF
        IF ( ASSOCIATED(THIS%CAUDAL) ) THEN
                DEALLOCATE(THIS%CAUDAL)
        END IF

        IF ( ASSOCIATED(THIS%GMAT) ) THEN
                DO I=1,SIZE(THIS%GMAT)
                        CALL DESTROY_(THIS%GMAT(I))
                END DO
                DEALLOCATE(THIS%GMAT)
        END IF

        NULLIFY(THIS%GMAT)
        NULLIFY(THIS%TIMEINTG)
```

```
          NULLIFY(THIS%MATER)
          NULLIFY(THIS%INDXMAT)

          NULLIFY(THIS%IBTCO)
          NULLIFY(THIS%COEC)
          NULLIFY(THIS%COLD)

          NULLIFY(THIS%C)

          NULLIFY(THIS%pACTH)
          NULLIFY(THIS%VD)
          NULLIFY(THIS%CAUDAL)

          THIS%ISTRANSIENT = .FALSE.
          THIS%ISASSIGNED = .FALSE.
          THIS%USEPARTICLES = .FALSE.

          NULLIFY(THIS%pA)
          NULLIFY(THIS%pD)
          NULLIFY(THIS%pB)

          NULLIFY(THIS%PART)

          RETURN
END SUBROUTINE DESTROY_TR_TC

!_____________________________________________________________________ READ_TR_TC

SUBROUTINE READ_TR_TC (THIS, AIERROR)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: AIERROR

          CHARACTER*50 :: FILETYPE
          CHARACTER*100 :: FILENAME

          CHARACTER*25 lsEspType
          character*25,parameter :: &
                    SP_ISTRANSIENT='ISTRANSIENT',   &
                    SP_MATRIXA='MATRIXA',        &
                    SP_MATRIXD='MATRIXD',        &
                    SP_VECTOR='VECTOR',     &
                    FT_MOSTSIMPLE='MOSTSIMPLE'

          LOGICAL lbPoints
          INTEGER*4 IUNIT

          AIERROR = 0

          this%IsTransient = ReadLogical ( SP_ISTRANSIENT, .TRUE. )
          if ( aiError .ne. 0 ) stop

          ! Choose between creating two or four matrices

          if ( this%istransient ) then
                    ALLOCATE(THIS%GMAT(3))
          else
                    ALLOCATE(THIS%GMAT(2))
          end if


          ! Matrix A creation

          lsEspType = 'FULLBAND'
          call ReadType ( SP_MATRIXA, lsEspType, lbPoints, lsEspType )
          if ( lbPoints ) then
                    print *,'CANNOT USE POINTER IN READ_TR_TC FOR GLOBAL SYSTEM'
                    stop
          end if
          THIS%pA => THIS%GMAT(1)
          call CREATE_(THIS%pA, lsEspType, THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)
          CALL SETDIAGONALSUM_(THIS%pA,.FALSE.)

          ! Matrix D creation (transient cases)

          if ( this%IsTransient ) then

                    lsEspType = 'DIAGONAL'
                    call ReadType ( SP_MATRIXD, lsEspType, lbPoints, lsEspType )
                    if ( lbPoints ) then
                              print *,'CANNOT USE POINTER IN READ_TR_TX FOR GLOBAL SYSTEM'
                              stop
                    end if
                    THIS%pD => THIS%GMAT(3)
                    call CREATE_(THIS%pD, lsEspType, THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)

          end if

          ! Vector B creation

          THIS%pB => THIS%GMAT(2)
          call CREATE_(THIS%pB, SP_VECTOR, THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)

          ! Creates time integral

          lsEspType = 'FULLIMP'
          call ReadType ( GT_TIMEINTG, lsEspType, lbPoints, lsEspType )
          if ( lbPoints ) then
                    print *,'CANNOT USE POINTER IN READ_TR_TX FOR GLOBAL SYSTEM'
                    stop
          end if
          call ChgSect('.|' // GT_TIMEINTG // CS_BLANK)
```

```
          ALLOCATE(THIS%TIMEINTG)
          CALL CREATE_(THIS%TIMEINTG, THIS%pMESH, lsEspType, this%istransient, .FALSE., &
                                                                                THIS%GMAT, AIERROR)
          CALL READ_(THIS%TIMEINTG, AIERROR)
          call ChgSect(CS_PREVIOUS)

          ! Obtains filename

          FILETYPE='TRACONF'
          FILENAME='INPUT.INP'
          IUNIT = 444
          call ReadFile ( NA_INPUTFILE, FILETYPE, FILENAME, IUNIT, &
                                                                                  FILETYPE, FILENAME, IUNIT )

          ! Reads

          SELECT CASE (FILETYPE)

                    CASE (FT_TRACONF)
                              CALL READ_TRACONF(THIS, FILENAME, IUNIT, AIERROR)

                    CASE (FT_MOSTSIMPLE)
                              CALL READ_MOSTSIMPLE(THIS, FILENAME, IUNIT, AIERROR)

                    CASE DEFAULT
                              aiError = -97
                              print *,'READ_TR_TC ONLY ACCEPTS CERTAIN FILE TYPES'

          END SELECT

          CALL SET_pOLD_(THIS%TIMEINTG, THIS%COLD)

          RETURN
END SUBROUTINE READ_TR_TC

!_____________________________________________________________________ SET_TR_TC

SUBROUTINE SET_TR_TC (THIS, NUMMAT, XLAM, abIsTransient, &
                                                  asEspTI, asEspTypeMT, asEspTypeGSMT, &
                                                  PORML, DFMML, DSLML, DSTML, CRDML, &
                                                  MTYPEN, &
                                                  IBTCON, COLDN, COECN, &
                                                  abAssigns, AIERROR)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          CHARACTER*25 :: &
                    asEspTI, &
                    asEspTypeGSMT, &
                    asEspTypeMT(:)
          INTEGER*4:: &
                    NUMMAT
          INTEGER*4, TARGET :: &
                    MTYPEN(:),          &
                    IBTCON(:)
          REAL*8:: &
                    PORML(:), DFMML(:), DSLML(:), DSTML(:), CRDML(:)
          REAL*8, TARGET:: &
                    COLDN(:),COECN(:)
          REAL*8 :: &
                    XLAM
          LOGICAL :: &
                    abIsTransient, &
                    abAssigns                              ! see components
          INTEGER*4 :: AIERROR

          INTEGER*4 I
          character*25,parameter :: &
                    SP_GSVECTOR='VECTOR'

          AIERROR = 0

          this%IsTransient = abIsTransient
          THIS%ISASSIGNED = abAssigns

          if ( this%istransient ) then
                    ALLOCATE(THIS%GMAT(3))
          else
                    ALLOCATE(THIS%GMAT(2))
          end if

          ! Create global system

          THIS%pA => THIS%GMAT(1)
          call CREATE_(THIS%pA, asEspTypeMT(1), THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)
          CALL SETDIAGONALSUM_(THIS%pA,.FALSE.)

          if ( this%IsTransient ) then

                    ! Create global system
                    THIS%pD => THIS%GMAT(3)
                    call CREATE_(THIS%pD, asEspTypeMT(2), THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)

          end if

          ! Create global system

          THIS%pB => THIS%GMAT(2)
          call CREATE_(THIS%pB, SP_VECTOR, THIS%pMESH%p%NUMNP, THIS%pMESH%p, .FALSE., AIERROR)

          ALLOCATE(THIS%TIMEINTG)
          CALL CREATE_(THIS%TIMEINTG, THIS%pMESH, asEspTI, this%istransient, .FALSE., &
                                                                                THIS%GMAT, AIERROR)
```

```
          CALL SET_(THIS%TIMEINTG, asEspTypeGSMT, AIERROR)

          IF ( NUMMAT.LE.0 ) THEN
                    PRINT *,'ERROR IN NUMBER OF MATERIALS'
                    AIERROR = -2
                    RETURN
          END IF

          THIS%NUMMAT = NUMMAT

          ! Allocates space

          ALLOCATE ( THIS%MATER(NUMMAT) )

          IF ( .NOT. THIS%ISASSIGNED ) THEN
                    ALLOCATE( THIS%INDXMAT(THIS%pMESH%p%NUMEL) )
                    ALLOCATE( THIS%IBTCO(THIS%pMESH%p%NUMNP), THIS%COEC(THIS%pMESH%p%NUMNP))
                    ALLOCATE( THIS%COLD(THIS%pMESH%p%NUMNP) )
          END IF

          ALLOCATE( THIS%VD(THIS%pMESH%p%NDIM,THIS%pMESH%p%NUMVP,1) )
          ALLOCATE( THIS%C(THIS%pMESH%p%NUMNP) )
          ALLOCATE( THIS%CAUDAL(THIS%pMESH%p%NUMNP,1) )

          DO I=1,NUMMAT
                    THIS%MATER(I)%POR  = PORML(I)
                    THIS%MATER(I)%DFM  = DFMML(I)
                    THIS%MATER(I)%DSL  = DSLML(I)
                    THIS%MATER(I)%DST  = DSTML(I)
                    THIS%MATER(I)%CRD  = CRDML(I)
          END DO

          THIS%XLAM = XLAM

          IF ( THIS%ISASSIGNED ) THEN

                    THIS%IBTCO => IBTCON
                    THIS%COEC  => COECN
                    THIS%COLD  => COLDN
                    THIS%INDXMAT => MTYPEN

          ELSE

                    THIS%IBTCO = IBTCON
                    THIS%COEC  = COECN
                    THIS%COLD  = COLDN
                    THIS%INDXMAT = MTYPEN

          ENDIF

          CALL SET_pOLD_(THIS%TIMEINTG, THIS%COLD)

          RETURN
END SUBROUTINE SET_TR_TC

!_____________________________________________________________ INIT_TR_TC

SUBROUTINE INIT_TR_TC ( THIS )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: AIERROR

          ! Assigns thickness if it's already calculated
          THIS%pACTH => pACTH_(THIS%pMESH)

          ! Initializes particle properties (if used)
          IF (THIS%USEPARTICLES) THEN
                    CALL PART_INIT(THIS, AIERROR)
          END IF


          RETURN
END SUBROUTINE INIT_TR_TC

!_____________________________________________________________ BUILD_TR_TC

SUBROUTINE BUILD_TR_TC ( THIS, DT )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: DT                                                    ! Time increment

          ! Obtains velocities and flows
          CALL COMVEL_(THIS%pFLOW, THIS%VD)
          CALL COMFLW_(THIS%pFLOW, DT, THIS%CAUDAL)

          ! Calcs TRANSPORT matrix
          CALL MAT_TRA(THIS, DT)

          ! Impose boundary conditions for TRANSPORT
          CALL CC_TRA(THIS)

          RETURN
END SUBROUTINE BUILD_TR_TC

!_____________________________________________________________ ENDINCR_TR_TC

SUBROUTINE ENDINCR_TR_TC ( THIS )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
```

```
          if ( this%IsTransient ) then

                    ! Stores the variation in node concentrations to be used by
                    ! a particle method
                    IF (THIS%USEPARTICLES) THEN
                              IF (THIS%PART%METHOD_NAME.EQ.SP_MOC) THEN
                                        THIS%PART%CN = THIS%C - THIS%COLD
                              END IF
                    END IF

                    ! Updates concentrations at nodes
                    THIS%COLD = THIS%C

          end if

          RETURN
END SUBROUTINE ENDINCR_TR_TC

!_____________________________________________________________ pC_TR_TC

FUNCTION pC_TR_TC ( THIS )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8, POINTER :: &
                    pC_TR_TC(:)         ! Concentration values

          pC_TR_TC => THIS%C

          RETURN
END FUNCTION pC_TR_TC

!_____________________________________________________________ DEASSC_TR_TC

SUBROUTINE DEASSC_TR_TC ( THIS, C )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8, POINTER :: &
                    C(:)      ! Concentration values

          NULLIFY(C)

          RETURN
END SUBROUTINE DEASSC_TR_TC

!_____________________________________________________________ pFLOW_TR_TC

FUNCTION pFLOW_TR_TC ( THIS )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          TYPE(T_FLOW),POINTER :: &
                    pFLOW_TR_TC                                     ! Flow equations

          pFLOW_TR_TC => THIS%pFLOW

          RETURN
END FUNCTION pFLOW_TR_TC

!_____________________________________________________________ DEASSFLOW_TR_TC

SUBROUTINE DEASSFLOW_TR_TC ( THIS, FLOW)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          TYPE (T_FLOW),POINTER :: &
                    FLOW                                  ! Flow equations

          NULLIFY(FLOW)

          RETURN
END SUBROUTINE DEASSFLOW_TR_TC

!_____________________________________________________________ pMESH_TR_TC

FUNCTION pMESH_TR_TC ( THIS )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          TYPE(T_MESH),POINTER :: &
                    pMESH_TR_TC                                     ! Mesh

          pMESH_TR_TC => THIS%pMESH

          RETURN
END FUNCTION pMESH_TR_TC

!_____________________________________________________________ DEASSMESH_TR_TC

SUBROUTINE DEASSMESH_TR_TC ( THIS, MESH)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          TYPE (T_MESH),POINTER :: &
                    MESH                                  ! Mesh

          NULLIFY(MESH)

          RETURN
END SUBROUTINE DEASSMESH_TR_TC

!_____________________________________________________________ COMBAL_TR_TC
```

```
SUBROUTINE COMBAL_TR_TC ( THIS, DT, BAL)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: &
                    DT                                                                  ! Time increments
          REAL*8 :: &
                    BAL(:,:)  ! Mass balance (out)

          INTEGER*4 I
          REAL*8 CONZ

          REAL*8 :: &
                    DISP(THIS%pMESH%p%MLOC,THIS%pMESH%p%MLOC), &
                    ADV(THIS%pMESH%p%MLOC,THIS%pMESH%p%MLOC), &
                    V(THIS%pMESH%p%MLOC,THIS%pMESH%p%MLOC), &
                    SD(THIS%pMESH%p%MLOC,THIS%pMESH%p%MLOC)


          DO I=1,THIS%pMESH%p%NUMNP
                    BAL(I,1) = 0D0
                    SELECT CASE ( THIS%IBTCO(I))
                              CASE (1)
                                        BAL(I,1) = BAL(I,1) + THIS%CAUDAL(I,1)*THIS%C(I)
                              CASE (2,3)
                                        IF ( THIS%CAUDAL(I,1) .GT. 0.0D0 ) THEN
                                                  CONZ = THIS%COEC(I)
                                        ELSE
                                                  CONZ = THIS%C(I)
                                        END IF
                                        BAL(I,1) = BAL(I,1) + THIS%CAUDAL(I,1)*CONZ
                              CASE (4)
                                        BAL(I,1) = BAL(I,1) + THIS%COEC(I)
                    END SELECT
          END DO

!FBP pdte.
!         DO I=1,THIS%pMESH%p%NUMEL
!                   IF ( CHECKORELEM_(THIS%pMESH,I,THIS%IBTCO,1) ) THEN
!
!                             if ( this%IsTransient ) then
!                                       CALL MAT_TRA_ELEM_STO_DEC(THIS, I, V, SD )
!                                       CALL ADDBALSTO_(THIS%pMESH, I, V, DT, THIS%C, THIS%COLD, THIS%IBTCO,
BAL(1:,1))
!                             end if
!                             CALL MAT_TRA_ELEM_DIS_ADV(THIS, I, THIS%VD(1:,I,1), DISP, ADV )
!                             CALL ADDBALDISP_(THIS%pMESH, I, DISP, THIS%C, THIS%IBTCO,  BAL(1:,1))
!                             CALL ADDBALADV_ (THIS%pMESH, I, ADV, THIS%C, THIS%IBTCO,  BAL(1:,1))
!
!                   END IF
!         END DO

          RETURN
END SUBROUTINE COMBAL_TR_TC

!_____________________________________________________________________ ISTRANSIENT_TR_TC

FUNCTION ISTRANSIENT_TR_TC ( THIS)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          LOGICAL ISTRANSIENT_TR_TC

          ISTRANSIENT_TR_TC = THIS%ISTRANSIENT

          RETURN
END FUNCTION ISTRANSIENT_TR_TC

!_____________________________________________________________________ SOLVE_TR_TC_SS

SUBROUTINE SOLVE_TR_TC_SS ( THIS, AIERROR )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: AIERROR

          if ( this%istransient ) then
                    print *,'ERROR IN SOLVE_TR_TC_SS STEADY STATE MODULE WITH TRANSIENT MODE'; stop
          else
                    CALL SOLVE_(THIS%TIMEINTG, THIS%C, AIERROR)
          end if

          RETURN
END SUBROUTINE SOLVE_TR_TC_SS

!_____________________________________________________________________ SOLVE_TR_TC_TR

SUBROUTINE SOLVE_TR_TC_TR ( THIS, DT, AIERROR )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: AIERROR
          REAL*8 :: dt                                                        ! Time increment

          ! If particles are used, resets old concentration with the intermediate one
          IF ( THIS%USEPARTICLES ) THEN
                    CALL PART_RUN(THIS, DT, AIERROR)
                    CALL SET_pOLD_(THIS%TIMEINTG, THIS%PART%CN)
          END IF

          ! Calls the general system solver
          if ( this%istransient ) then
```

```
                    CALL SOLVE_(THIS%TIMEINTG, THIS%C, DT, AIERROR)
          else
                    CALL SOLVE_(THIS%TIMEINTG, THIS%C, AIERROR)
          end if

          RETURN
END SUBROUTINE SOLVE_TR_TC_TR

!-------------------------------------------------------------------------------

!_____________________________________________________________ GET_DISP

SUBROUTINE GET_DISP ( THIS, VD,  DISP )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: &
                    VD(:,:),                                          & ! Velocity (Darcy sense)
                    DISP(:)   ! Dispersion matrix (out)

          TYPE(T_MATER_TRANSPORT),POINTER::pMATER
          INTEGER*4 :: &
                    IELEM, K, J, &
                    ND, NE, NV, NC
          REAL*8 :: &
                    QX2,QY2,QNOR,QXY
          REAL*8,ALLOCATABLE, DIMENSION(:) :: &
                    DFML, DFMLX, &
                    DST, DSTX, &
                    DSL, DSLX

          ND = THIS%pMESH%p%NDIM
          NE = THIS%pMESH%p%NUMEL
          NV = THIS%pMESH%p%NUMVP

          ALLOCATE (DFMLX(NE), DSLX(NE), DSTX(NE))
          ALLOCATE (DFML(NV), DSL(NV), DST(NV))

          DO IELEM=1,NE

                    pMATER => THIS%MATER(THIS%INDXMAT(IELEM))

                    DFMLX(IELEM) = pMATER%DFM * pMATER%POR * THIS%pACTH(IELEM)
                    DSLX(IELEM) = pMATER%DSL
                    DSTX(IELEM) = pMATER%DST

          END DO

          NC=1
          CALL INTERP_ELEM2VP_ (THIS%pMESH, DFMLX, DFML, NC)
          CALL INTERP_ELEM2VP_ (THIS%pMESH, DSLX, DSL, NC)
          CALL INTERP_ELEM2VP_ (THIS%pMESH, DSTX, DST, NC)

          DEALLOCATE (DFMLX, DSLX, DSTX)

          K=1
          DO J=1, NV

                    QX2 = VD(1,J)*VD(1,J)
                    QY2 = VD(2,J)*VD(2,J)
                    QNOR = DSQRT(QX2+QY2)

                    IF (QNOR.NE.0.) THEN
                              QX2 = QX2 / QNOR
                              QY2 = QY2 / QNOR
                              QXY = VD(1,J)*VD(2,J)/QNOR
                    ELSE
                              QX2 = 0.
                              QY2 = 0.
                              QXY = 0.
                    END IF

                    DISP(K) = DFML(J) + DSL(J)*QX2 + DST(J)*QY2
                    DISP(K+3) = DFML(J) + DST(J)*QX2 + DSL(J)*QY2
                    DISP(K+1) =(DSL(J)-DST(J))*QXY
                    DISP(K+2) = DISP(K+1)

                    K=K+4
          END DO

          DEALLOCATE (DFML, DSL, DST)

          RETURN
END SUBROUTINE GET_DISP

!_____________________________________________________________ GET_STO

SUBROUTINE GET_STO ( THIS, STO, STODEL )

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          REAL*8,POINTER :: &
                    STO(:),   & ! Storage (out)
                    STODEL(:) ! Storage affected by radioactive decay (out)

          INTEGER*4 IELEM
          TYPE(T_MATER_TRANSPORT),POINTER::pMATER

          DO IELEM=1, THIS%pMESH%p%NUMEL

                    pMATER => THIS%MATER(THIS%INDXMAT(IELEM))

                    STO(IELEM) = (pMATER%CRD + pMATER%POR) * THIS%pACTH(IELEM)
```

```
                    STODEL(IELEM) = STO(IELEM) * THIS%XLAM

          END DO

          RETURN
END SUBROUTINE GET_STO

!_____________________________________________________________________ MAT_TRA

SUBROUTINE MAT_TRA ( THIS, DT )
!
! Builds elemental flow matrixes DISP,V,ADV,SD and assembles it to the A,D global system
!
          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS

          INTEGER*4 ND, NE, NV

          REAL*8 :: DT
          REAL*8, POINTER :: DISP(:), STO(:), STODEL(:), V1D(:)


          ND = THIS%pMESH%p%NDIM
          NE = THIS%pMESH%p%NUMEL
          NV = THIS%pMESH%p%NUMVP

          ALLOCATE(DISP(ND*ND*NV), V1D(ND*NV))
          CALL GET_DISP(THIS, THIS%VD(1:,1:,1), DISP)

          CALL RESET_(THIS%pA)

          ! Auxiliar subroutine to transform array V in an 1-D array
          CALL VARRAY2V1D(THIS, V1D)

          ! Advection process (only for non-particle methods)
          IF (.NOT.THIS%USEPARTICLES) THEN
                    ! This is a weird step: we have to change sign of velocity
                    ! REVISE THIS!
                    V1D = -V1D

                    CALL INTGV_P_GRAD_x_(THIS%pMESH, V1D, THIS%pA)
          END IF

          ! Dispersion process
          CALL INTGV_DIV_P_GRAD_x_(THIS%pMESH, DISP, THIS%pA)

          DEALLOCATE (V1D)

          ! Getting storage
          if ( this%IsTransient ) then

                    ALLOCATE(STO(NE))
                    ALLOCATE(STODEL(NE))
                    CALL GET_STO(THIS, STO, STODEL)

                    CALL RESET_(THIS%pD)
                    CALL INTGV_P_x_(THIS%pMESH, STO, THIS%pD)
                    CALL INTGV_P_x_(THIS%pMESH, STODEL, THIS%pA)

                    DEALLOCATE(STODEL)
                    DEALLOCATE(STO)

          END IF

          DEALLOCATE(DISP)

          RETURN
END SUBROUTINE MAT_TRA

!_____________________________________________________________________ VARRAY2V1D
!
SUBROUTINE VARRAY2V1D(THIS, V)

          IMPLICIT NONE

          TYPE (T_TR_TRACONF) THIS
          REAL*8, POINTER :: &
                    V(:)

          ! Internal variable declaration
          INTEGER*4 :: &
                    ND, NV, &
                    I, J, K

          ND = THIS%pMESH%p%NDIM
          NV = THIS%pMESH%p%NUMVP

          V=0
          K=0

          DO I=1, NV
                    DO J=1, ND
                              K=K+1
                              V(K) = THIS%VD(J, I, 1)
                    END DO
          END DO

          RETURN
END SUBROUTINE

!_____________________________________________________________________ CC_TRA

SUBROUTINE CC_TRA ( THIS )
```

```
!
! Computes boundary conditions:
!                  (1) Fixed values: applied in time integration construction
!                  (2,3) Flow with fixed conc.: term added to diagonal and rhs
!                  (4) Fixed mass: term added to diagonal
!
          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS

          INTEGER*4 I,NP
          LOGICAL ISPRESENTLEAK, ISPRESENTMASS
          LOGICAL, POINTER :: LFIXLEAK(:), LFIXMASS(:)
          REAL*8 ZEROES(THIS%pMESH%p%NUMNP), Q1
          REAL*8, POINTER :: FLEAK(:)

          NP = THIS%pMESH%p%NUMNP

          ZEROES = 0D0
          CALL SET_ALL_(THIS%pB, ZEROES)
          CALL RESET_FIX_VALUES_(THIS%TIMEINTG)

          ALLOCATE(LFIXLEAK(NP))
          LFIXLEAK = .FALSE.
          ISPRESENTLEAK = .FALSE.

          ALLOCATE(FLEAK(2*NP))
          FLEAK = 0D0

          ALLOCATE(LFIXMASS(NP))
          LFIXMASS = .FALSE.
          ISPRESENTMASS = .FALSE.

          DO I=1,NP
                    SELECT CASE ( THIS%IBTCO(I))
                              CASE (1)
                                        CALL SET_FIX_VALUE_(THIS%TIMEINTG, I, THIS%COEC(I))
                              CASE (2,3)
                                        Q1 = THIS%CAUDAL(I,1)
                                        IF ( Q1 .GT. 0.0D0 ) THEN
                                                  ISPRESENTLEAK = .TRUE.
                                                  LFIXLEAK(I) = .TRUE.
                                                  FLEAK(2*I-1) = Q1
                                                  FLEAK(2*I) = THIS%COEC(I)
                                                  THIS%IBTCO(I) = 2 ! ¿?
                                        END IF
                              CASE (4)
                                        ISPRESENTMASS = .TRUE.
                                        LFIXMASS(I) = .TRUE.
                    END SELECT
          END DO

          IF ( ISPRESENTLEAK ) THEN
                    CALL INTGB_F_x_MINUS_x0_(THIS%pMESH, FLEAK, LFIXLEAK, THIS%pA, THIS%pB)
          END IF

          IF ( ISPRESENTMASS ) THEN
                    CALL INTGB_F_(THIS%pMESH, THIS%COEC, LFIXMASS, THIS%pA, THIS%pB)
          END IF

          DEALLOCATE(LFIXMASS)
          DEALLOCATE(FLEAK)
          DEALLOCATE(LFIXLEAK)

          RETURN
END SUBROUTINE CC_TRA

!_____________________________________________________________________ READ_TRACONF

SUBROUTINE READ_TRACONF (THIS, FILENAME, IUINP, AIERROR)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: IUINP
          INTEGER*4 :: AIERROR

          CHARACTER*80 :: DUMMY
          INTEGER*4:: NUMNP, NUMEL, I, NUMMAT, N, L, MTYPEN,IBTCON
          REAL*8:: PORML,DFMML,DSLML,DSTML,CRDML, DEFCOLD, COLDN,COECN, XLAM
          CHARACTER*100 :: FILENAME

          AIERROR = 0

    open(unit=IUINP,file=FILENAME,status='old', err= 998)

          ! Gets dimensions

          READ(IUINP, '(A80)') DUMMY

          READ(IUINP, '(3I5)') NUMNP,NUMEL,NUMMAT

          IF ( NUMMAT.LE.0 ) THEN
                    PRINT *,'ERROR IN NUMBER OF MATERIALS'
                    AIERROR = -2
                    CLOSE(IUINP)
                    RETURN
          END IF

          THIS%NUMMAT = NUMMAT

          ! Allocates space

          ALLOCATE ( THIS%MATER(NUMMAT) )
```

```
          N = THIS%pMESH%p%NUMEL
          IF ( N.LE.0 ) THEN
                    PRINT *,'ERROR IN NUMBER OF ELEMENTS OR MESH NOT INITIALIZED'
                    AIERROR = -3
                    CLOSE(IUINP)
                    RETURN
          END IF
          ALLOCATE( THIS%INDXMAT(N) )
          ALLOCATE( THIS%VD(THIS%pMESH%p%NDIM,N,1) )

          N = THIS%pMESH%p%NUMNP
          IF ( N.LE.0 ) THEN
                    PRINT *,'ERROR IN NUMBER OF NODES OR MESH NOT INITIALIZED'
                    AIERROR = -3
                    CLOSE(IUINP)
                    RETURN
          END IF
          ALLOCATE( THIS%IBTCO(N), THIS%COEC(N), THIS%COLD(N), THIS%C(N) )
          ALLOCATE( THIS%CAUDAL(N,1) )

          ! Read node and element information

          READ(IUINP,'(A80)') DUMMY ! IOFLUST,IOTRAST,IOEQTRA
          READ(IUINP,'(A80)') DUMMY ! KAT,INTER,DT,TIME1,DTC,TMAX

          DO I=1,NUMMAT
                    READ(IUINP,'(30X,5E10.4)') PORML,DFMML,DSLML,DSTML,CRDML
                    THIS%MATER(I)%POR  = PORML
                    THIS%MATER(I)%DFM  = DFMML
                    THIS%MATER(I)%DSL  = DSLML
                    THIS%MATER(I)%DST  = DSTML
                    THIS%MATER(I)%CRD  = CRDML
          END DO

!*.................................................................LEE NUDOS
!*     VALORES POR DEFECTO
          READ(IUINP,'(E10.3,20X,E10.3)') XLAM, DEFCOLD

          THIS%XLAM = XLAM
!* ............................................ COMIENZA CICLO DE LECTURA DE NUDOS
          L=1
          DO WHILE ( L.LE.NUMNP )

                    READ(IUINP,'(I5,20X,3X,I3,28X,2E7.1)')N,IBTCON,COLDN,COECN
                    IF(COLDN.EQ.0D0) COLDN=DEFCOLD

                    THIS%IBTCO(N) = IBTCON
                    THIS%COEC(N)  = COECN
                    THIS%COLD(N)  = COLDN

                    IF ( N.GT.L ) THEN

                              ! Interpolates nodal information

                              THIS%IBTCO(L:N-1) = THIS%IBTCO(L-1)
                              THIS%COEC(L:N-1)   = THIS%COEC(L-1)
                              THIS%COLD(L:N-1) = THIS%COLD(L-1)

                    END IF
                    L = N +1
          END DO
!*.............................................TERMINA CICLO DE LECTURA DE NUDOS

!*.......................................................LEE ELEMENTOS
!* ....................................COMIENZA CICLO DE LECTURA DE ELEMENTOS
          L=1
          DO WHILE ( L.LE.NUMEL )

                    READ(IUINP,'(I5,15X,I5)') N,MTYPEN

                    ! Check errors
                    IF (MTYPEN.LT.0 .OR. MTYPEN.GT.THIS%NUMMAT ) THEN
                              PRINT *,' ERROR: MTYPE FUERA DE LIMITES EN ELEMENTO',N
                              AIERROR=AIERROR-1
                    END IF

                    ! Assign values
                    THIS%INDXMAT(N) = MTYPEN

                    IF ( N.GT.L ) THEN

                              ! Interpolates element information

                              THIS%INDXMAT(L:N-1) = THIS%INDXMAT(L-1)

                    END IF
                    L=N+1
          END DO
!*.....................................TERMINA CICLO DE LECTURA DE ELEMENTOS

          THIS%USEPARTICLES = .FALSE.

          CLOSE(IUINP)
          RETURN

998 continue
          aiError = -1
          print *, 'ERROR WHILE READING TRANSPORT'

          RETURN
END SUBROUTINE READ_TRACONF
```

```
!_____________________________________________________________________ READ_MOSTSIMPLE

SUBROUTINE READ_MOSTSIMPLE (THIS, FILENAME, IUINP, AIERROR)

          IMPLICIT NONE
          TYPE(T_TR_TRACONF) THIS
          CHARACTER*100 :: FILENAME
          INTEGER*4 :: IUINP
          INTEGER*4 :: AIERROR

          CHARACTER*80 :: DUMMY
          INTEGER*4:: NX,NY, TOTALCELLS
          INTEGER*4:: NPH, NPL, NPMAX, NPMIN, METHOD
          REAL*8:: DCEPS

          INTEGER*4:: I, NUMMAT, N, K
          REAL*8 :: PORML,DFMML,DSLML,DSTML,CRDML

          INTEGER*4, ALLOCATABLE, DIMENSION(:):: &
                    ISINDOMAIN

          AIERROR = 0

    open(unit=IUINP,file=FILENAME,status='old', err= 998)

          READ (IUINP, 100) NX                              ! Mesh dimension X
          READ (IUINP, 100) NY                              ! Mesh dimension Y
100       FORMAT (I6)

          TOTALCELLS = NX*NY

          DO I=1,NX+NY
                    READ(IUINP, '(A80)') DUMMY
          END DO

          ALLOCATE(ISINDOMAIN(TOTALCELLS))
          DO I=1,TOTALCELLS
                    READ(IUINP, '(I1)') ISINDOMAIN(I)
          END DO

          N = THIS%pMESH%p%NUMNP

          ALLOCATE( THIS%INDXMAT(N) )
          ALLOCATE( THIS%VD(THIS%pMESH%p%NDIM,THIS%pMESH%p%NUMVP,1) )
          ALLOCATE( THIS%IBTCO(N), THIS%COEC(N), THIS%COLD(N), THIS%C(N) )
          ALLOCATE( THIS%CAUDAL(N,1) )

          K=0
          DO I=1,TOTALCELLS
                    IF (ISINDOMAIN(I).EQ.1) THEN
                              K=K+1
                              READ(IUINP, 200) THIS%INDXMAT(I), THIS%IBTCO(I), &
                                                  THIS%COLD(I), THIS%COEC(I)
                    ELSE
                              READ (IUINP,'(A80)') DUMMY
                    END IF
          END DO

200 FORMAT(8X,I5,45X,I5,2F8.1)


          ! Gets dimensions

          READ(IUINP, '(I5)') NUMMAT

          IF ( NUMMAT.LE.0 ) THEN
                    PRINT *,'ERROR IN NUMBER OF MATERIALS'
                    AIERROR = -2
                    CLOSE(IUINP)
                    RETURN
          END IF

          THIS%NUMMAT = NUMMAT

          ! Allocates space

          ALLOCATE ( THIS%MATER(NUMMAT) )

          DO I=1,NUMMAT
                    READ(IUINP,'(30X,5E10.4)') PORML,DFMML,DSLML,DSTML,CRDML
                    THIS%MATER(I)%POR  = PORML
                    THIS%MATER(I)%DFM  = DFMML
                    THIS%MATER(I)%DSL  = DSLML
                    THIS%MATER(I)%DST  = DSTML
                    THIS%MATER(I)%CRD  = CRDML
          END DO

          ! Particle properties ..................................................................

          ! Reads type of method used. Available ones are:
          !         1 = MOC
          !         2 = MMOC
          !         3 = Random walk

          ALLOCATE (THIS%PART)

          READ(IUINP, '(A80)') DUMMY
          READ(IUINP, '(I1)') METHOD

          SELECT CASE (METHOD)
```

```
                    CASE (0)
                              ! No particles are used

                    CASE (1)
                              THIS%PART%METHOD_NAME = SP_MOC

                    CASE (2)
                              THIS%PART%METHOD_NAME = SP_MMOC

                    CASE (3)
                              THIS%PART%METHOD_NAME = SP_RNDWALK

                    CASE DEFAULT
                              PRINT *,'UNKNOWN PARTICLE METHOD'
                              AIERROR = -2
                              CLOSE (IUINP)
                              RETURN

          END SELECT

          IF (METHOD .NE. 0) THEN
                    THIS%USEPARTICLES = .TRUE.
                    THIS%PART%METHOD = METHOD
                    THIS%PART%IS_FIRST = .TRUE.
          ELSE
                    THIS%USEPARTICLES = .FALSE.
          END IF


          ! Reads necessary properties for particle methods

          IF (THIS%USEPARTICLES) THEN

                    READ(IUINP, '(4I5, E10.4)') NPH, NPL, NPMAX, NPMIN, DCEPS

                    THIS%PART%NPH = NPH
                    THIS%PART%NPL = NPL
                    THIS%PART%NPMAX = NPMAX
                    THIS%PART%NPMIN = NPMIN
                    THIS%PART%DCEPS = DCEPS

          END IF


          CLOSE(IUINP)
          RETURN

998 continue
          aiError = -1
          print *, 'ERROR WHILE READING FLOW'

          RETURN
END SUBROUTINE READ_MOSTSIMPLE


!_____________________________________________________________________________________
!
!                                                                        PSEUDO-MODULE
!                                          PARTICLE MANAGEMENT SUBROUTINES
!_____________________________________________________________________________________
!
!                                                                        MAIN SUBROUTINES
!_____________________________________________________________________________________

!_____________________________________________________________________ PART_INIT
!
SUBROUTINE PART_INIT (THIS, AIERROR)
!         Purpose: initializes particle parameters
!         Arguments:
!                           THIS: TRANSPORT-type object
!
!         Tested? NO
!   Last modified 18-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS
          INTEGER*4 :: AIERROR

          SELECT CASE (THIS%PART%METHOD_NAME)

                    CASE (SP_MOC)
```

```
                    CALL MOC_INIT (THIS)

          CASE (SP_MMOC)
                    CALL MMOC_INIT (THIS)

          CASE (SP_RNDWALK)
                    CALL RNDWALK_INIT (THIS)

          CASE DEFAULT
          AIERROR = -1
          PRINT *, 'ERROR IN PART_INIT: UNKNOWN METHOD NAME'

    END SELECT

    RETURN
END SUBROUTINE PART_INIT

!_____________________________________________________________________ PART_RUN
!
SUBROUTINE PART_RUN (THIS, DT, AIERROR)
!         Purpose: runs particle method
!         Arguments:
!                   THIS: TRANSPORT-type object
!                   DT: time increment
!
!         Tested? NO
!   Last modified 18-03-2004

    IMPLICIT NONE

    TYPE(T_TR_TRACONF) THIS
    INTEGER*4 :: AIERROR
    REAL*8 :: DT

    SELECT CASE (THIS%PART%METHOD_NAME)

          CASE (SP_MOC)
                    CALL MOC_RUN (THIS, DT)

          CASE (SP_MMOC)
                    CALL MMOC_RUN (THIS, DT)

          CASE (SP_RNDWALK)
                    CALL RNDWALK_RUN (THIS, DT)

          CASE DEFAULT
          AIERROR = -1
          PRINT *, 'ERROR IN PART_RUN: UNKNOWN METHOD NAME'

    END SELECT


END SUBROUTINE PART_RUN


!_____________________________________________________________________________________
!
!                                                                          PSEUDO-MODULE
!                                          PARTICLE MANAGEMENT SUBROUTINES
!_____________________________________________________________________________________
!
!                                                   SECONDARY SUBROUTINES
!_____________________________________________________________________________________


!_____________________________________________________________________ MOC_INIT
!
SUBROUTINE MOC_INIT (THIS)
!         Purpose: initializes particle parameters for MOC method
!         Arguments:
!                   THIS: TRANSPORT-type object
!
!         Tested? NO
!   Last modified 17-03-2004
!

    IMPLICIT NONE

    TYPE(T_TR_TRACONF) THIS

    INTEGER*4 :: &
          NPH, NPL, NPR,&
          N, NODES, MAXNPART, &
          I, J, K
    INTEGER*4, POINTER :: &
          PEL(:)
    REAL*8 :: &
          VOL                         ! Element volume
    REAL*8, POINTER :: &
          DC(:), &  ! MT3D's DCCELL
          C1(:), &
          PX(:), PY(:), &
          PC(:), PVOL(:), &
          LPX(:), LPY(:)


    N = THIS%pMESH%P%NUMEL
    NODES = THIS%pMESH%P%NUMNP
    NPH = THIS%PART%NPH
    NPL = THIS%PART%NPL
    MAXNPART = NPH*N

    ALLOCATE (DC(N))
```

```
          ALLOCATE (PX(MAXNPART), PY(MAXNPART))
          ALLOCATE (PEL(MAXNPART))
          ALLOCATE (PVOL(MAXNPART), PC(MAXNPART))
          ALLOCATE (LPX(NPH), LPY(NPH))

          ALLOCATE (THIS%PART%CN(NODES))
          ALLOCATE (THIS%PART%PARTSINELEM(N))

          C1 => THIS%COLD

          ! Calculates relative gradient
          CALL RELAT_GRAD_(THIS%pMESH, C1, DC)

          ! Loop which obtains all particle properties
          K = 0
          DO I=1,N

                    CALL ELEM_VOL_(THIS%pMESH, VOL, I)

                    LPX = 0
                    LPY = 0

                    NPR = NPL
                    IF (DC(I).GT. THIS%PART%DCEPS) THEN
                              NPR = NPH
                    END IF

                    THIS%PART%PARTSINELEM(I) = NPR
                    CALL PART_DISPOSE_(THIS%pMESH, NPR, LPX, LPY, I, 2)

                    DO J = 1, NPR
                              PX(K+J) = LPX(J)
                              PY(K+J) = LPY(J)
                              PVOL(K+J) = VOL
                              PC(K+J) = C1(I)
                              PEL(K+J) = I
                    END DO

                    K= K + NPR

          END DO

          THIS%PART%NPART = K

          ! Assignation of values to TYPE variables

          ALLOCATE(THIS%PART%PX(K), THIS%PART%PY(K))
          ALLOCATE(THIS%PART%PVOL(K), THIS%PART%PC(K))
          ALLOCATE(THIS%PART%ELEMENT(K))

          DO I=1, K

                    THIS%PART%PX(I) = PX(I)
                    THIS%PART%PY(I) = PY(I)
                    THIS%PART%PVOL(I) = PVOL(I)
                    THIS%PART%PC(I) = PC(I)
                    THIS%PART%ELEMENT(I) = PEL(I)

          END DO

          DEALLOCATE (PX, PY, PVOL, PC, LPX, LPY, DC, PEL)

          RETURN
END SUBROUTINE MOC_INIT

!_____________________________________________________________________ MMOC_INIT
!
SUBROUTINE MMOC_INIT (THIS)
!         Purpose: initializes particle parameters for MMOC method
!         Arguments:
!                             THIS: TRANSPORT-type object
!
!         Tested? NO
!   Last modified 20-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS

          INTEGER*4 :: &
                    NPH, NPL, NPR,&
                    NEL, NODES, MAXNPART, &
                    I, J, K
          INTEGER*4, POINTER :: &
                    PEL(:)
          REAL*8 :: &
                    VOL                         ! Element volume
          REAL*8, POINTER :: &
                    DC(:), &  ! MT3D's DCCELL
                    C1(:), &
                    PX(:), PY(:), &
                    PC(:), &
                    LPX(:), LPY(:)


          NEL = THIS%pMESH%P%NUMEL
          NODES = THIS%pMESH%P%NUMNP
          MAXNPART = THIS%PART%NPL*NEL

          ALLOCATE (PX(MAXNPART), PY(MAXNPART))
          ALLOCATE (PEL(MAXNPART))
          ALLOCATE (LPX(NPH), LPY(NPH))
```

```
          ALLOCATE (THIS%PART%CN(NODES))
          ALLOCATE (THIS%PART%PARTSINELEM(NEL))


          K = 0
          DO I=1,NEL

                    ! Only one particle is disposed in every cell, unless a source or
                    ! a sink exists in it.
                    ! This is not intended for finite element method. Bear in mind that
                    ! finite difference method considered places each node at the center
                    ! of the element (cell)
                    ! A solution should be found to determine the mass flux in each ELEMENT,
                    ! not NODE.

                    IF (THIS%CAUDAL(I,1).EQ.0.) THEN
                              NPR = 1
                    ELSE
                              NPR = NPH
                    END IF

                    THIS%PART%PARTSINELEM(I) = NPR

                    CALL PART_DISPOSE_(THIS%pMESH, NPR, LPX, LPY, I, 1)

                    DO J = 1, NPR
                              PX(K+J) = LPX(J)
                              PY(K+J) = LPY(J)
                              PEL(K+J) = I
                    END DO

                    K= K + NPR

          END DO

          THIS%PART%NPART = K

          ALLOCATE(THIS%PART%PX(K), THIS%PART%PY(K))
          ALLOCATE(THIS%PART%PC(K))
          ALLOCATE(THIS%PART%ELEMENT(K))

          DO I=1, K

                    THIS%PART%PX(I) = PX(I)
                    THIS%PART%PY(I) = PY(I)
                    THIS%PART%ELEMENT(I) = PEL(I)

          END DO

          DEALLOCATE (PX, PY, PC, LPX, LPY, PEL)

          RETURN
END SUBROUTINE MMOC_INIT


!_____________________________________________________________________ RNDWALK_INIT
!
SUBROUTINE RNDWALK_INIT (THIS)
!         Purpose: initializes particle parameters for RNDWALK method
!         Arguments:
!                              THIS: TRANSPORT-type object
!
!         Tested? NO
!   Last modified 17-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS


END SUBROUTINE RNDWALK_INIT


!_____________________________________________________________________ MOC_RUN
!
SUBROUTINE MOC_RUN (THIS, DT)
!         Purpose: runs MOC method
!         Arguments:
!                              THIS: TRANSPORT-type object
!                              DT: time increment
!
!         Tested? NO
!   Last modified 17-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: DT

          INTEGER*4 :: &
                    ND, NV, NP, &
                    NPART1, NPART2
          REAL*8, POINTER :: &
                    PC(:), &
                    PVOL(:), &
                    PX(:), &
                    PY(:), &
                    VEL(:)
```

```
          ND = THIS%pMESH%P%NDIM
          NV = THIS%pMESH%P%NUMVP
          NP = THIS%pMESH%P%NUMNP

          NPART1 = THIS%PART%NPART
          NPART2 = NPART1

          ALLOCATE (VEL(ND*NV))

          PC => THIS%PART%PC
          PVOL => THIS%PART%PVOL
          PX => THIS%PART%PX
          PY => THIS%PART%PY

          IF(.NOT.THIS%PART%IS_FIRST) THEN

                    THIS%PART%NPART = THIS%PART%NPART2
                    NPART1 = THIS%PART%NPART
                    NPART2 = NPART1

                    ! Interpolates in order to get particle concentrations
                    ! Here, CN stores C - COLD
                    CALL PART_UPDATE_(THIS%pMESH, THIS%PART%CN, NPART1, PC, PX, PY)

          END IF

          ! Gets velocity in 1-dimensional array form
          CALL VARRAY2V1D(THIS, VEL)

          ! Moves particles inside the domain (reflects them)
          VEL = DT*VEL
          CALL MOVE_POINT_(THIS%pMESH, NPART1, PX, PY, VEL)

          THIS%PART%PX = PX
          THIS%PART%PY = PY

          ! Controls number of particles, extreme concentrations...
          CALL PART_RESTRICT (THIS)

          ! Gets node concentrations, from particle ones
          CALL PART2ELEM_(THIS%pMESH, THIS%PART%NPART2, THIS%PART%PVOL, &
                                                                THIS%PART%ELEMENT, THIS%PART%PC, THIS%PART%CN)

          THIS%PART%IS_FIRST = .FALSE.

          RETURN
END SUBROUTINE MOC_RUN

!_____________________________________________________________________ MMOC_RUN
!
SUBROUTINE MMOC_RUN (THIS, DT)
!         Purpose: runs MMOC method
!         Arguments:
!                           THIS: TRANSPORT-type object
!                           DT: time increment
!
!         Tested? NO
!   Last modified 20-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: DT

          INTEGER*4 :: &
                    I, J, K, &
                    ND, NV, NP, NPR, &
                    NPART
          REAL*8, POINTER :: &
                    PC(:), &
                    PVOL(:), &
                    PX(:), PY(:), &
                    VEL(:), &
                    C(:)

          ND = THIS%pMESH%P%NDIM
          NV = THIS%pMESH%P%NUMVP
          NP = THIS%pMESH%P%NUMNP
          NPART = THIS%PART%NPART

          ALLOCATE (VEL(ND*NV))
          ALLOCATE (PX(NPART), PY(NPART))
          ALLOCATE (C(NP))

          ! Assign initial positions to PX, PY
          PX = THIS%PART%PX
          PY = THIS%PART%PY

          ! Gets velocity in 1-dimensional array form
          CALL VARRAY2V1D(THIS, VEL)

          ! Moves particles backwards inside the domain (reflects them)
          VEL = -DT*VEL
          CALL MOVE_POINT_(THIS%pMESH, NPART, PX, PY, VEL)

          ! Interpolates node concentrations on positions where particles are placed
          IF (THIS%PART%IS_FIRST) THEN
                    C = THIS%COLD
          ELSE
                    C = THIS%C
          END IF
```

```
          CALL SCALAR_FIELD_ (THIS%pMESH, C, NPART, PX, PY, THIS%PART%PC)

          ! Assigns particle concentrations to nodes, interpolating when a mass flux is
          ! present.

          K=0
          DO I = 1, NP

                    NPR = THIS%PART%PARTSINELEM(I)
                    K = K + NPR
                    IF (NPR.GT.1) THEN
                              ! No interpolation used. Just the mean value.
                              ! Obviously, it has to be improved!
                              THIS%PART%CN(I) = 0
                              DO J = 1, NPR
                                        THIS%PART%CN(I) = THIS%PART%CN(I) + THIS%PART%PC(K-NPR+J)/NPR

                              END DO
                    ELSE
                              THIS%PART%CN(I) = THIS%PART%PC(K)
                    END IF
          END DO

          DEALLOCATE (PX, PY, C)

          THIS%PART%IS_FIRST = .FALSE.

          RETURN
END SUBROUTINE MMOC_RUN

!_____________________________________________________________________ RNDWALK_RUN
!
SUBROUTINE RNDWALK_RUN (THIS, DT)
!         Purpose: runs Random Walk method
!         Arguments:
!                           THIS: TRANSPORT-type object
!                           DT: time increment
!
!         Tested? NO
!   Last modified 18-03-2004
!

          IMPLICIT NONE

          TYPE(T_TR_TRACONF) THIS
          REAL*8 :: DT

END SUBROUTINE RNDWALK_RUN

!_____________________________________________________________________ PART_RESTRICT
!
SUBROUTINE PART_RESTRICT (THIS)
!         Purpose: restricts max-min number of particles,
!                           and obtains the first matrix product
!         Arguments:
!                           THIS: TRANSPORT-type object
!
!         Tested? NO
!   Last modified 17-03-2004
!
          IMPLICIT NONE
          TYPE (T_TR_TRACONF) THIS

          LOGICAL :: &
                    CHOSEN
          INTEGER*4 :: &
                    I, J, K, &
                    N, M, NODES, &
                    NPART1, &
                    NPR, &
                    OLD_NPR, &
                    MAXNPART
          INTEGER*4, POINTER :: &
                    OLD_PARTS(:)
          REAL*8 :: &
                    VOL
          REAL*8, POINTER :: &
                    PX(:), PY(:), &
                    PC(:), PVOL(:), &
                    LPX(:), LPY(:), &
                    PEL(:), PC_OLD(:)

          N = THIS%pMESH%p%NUMEL
          NODES = THIS%pMESH%p%NUMNP
          MAXNPART = N*THIS%PART%NPMAX

          ! Initially allocate all arrays with maximum acceptable number of particles

          ALLOCATE (PX(MAXNPART), PY(MAXNPART))
          ALLOCATE (PEL(MAXNPART))
          ALLOCATE (PVOL(MAXNPART), PC(MAXNPART))
          ALLOCATE (LPX(THIS%PART%NPH), LPY(THIS%PART%NPH))
          ALLOCATE (PC_OLD(THIS%PART%NPART))

          PC_OLD = THIS%PART%PC
          NPART1 = THIS%PART%NPART

          ! Loop in elements
          K = 0
          DO I=1,N

                    ! Control maximum and minimum number of particles in a cell
                    LPX = 0
```

```
                    LPY = 0
                    NPR = -1
                    IF (THIS%PART%PARTSINELEM(I).GE.THIS%PART%NPMAX) THEN
                              NPR = THIS%PART%NPH
                    END IF
                    IF (THIS%PART%PARTSINELEM(I).LE.THIS%PART%NPMIN) THEN
                              NPR = THIS%PART%NPL
                    END IF

                    ! If the particles in the cell have to be redisposed, do it.
                    IF (NPR.GE.0) THEN

                              OLD_NPR = THIS%PART%PARTSINELEM(I)
                              THIS%PART%PARTSINELEM(I) = NPR

                              CALL ELEM_VOL_(THIS%pMESH, VOL, I)

                              CALL PART_DISPOSE_(THIS%pMESH, NPR, LPX, LPY, I, 2)

                              DO J = 1, NPR
                                        PX(K+J) = LPX(J)
                                        PY(K+J) = LPY(J)
                                        PVOL(K+J) = VOL
                                        IF (THIS%PART%IS_FIRST) THEN
                                                  PC(K+J) = THIS%COLD(I)
                                        ELSE
                                                  PC(K+J) = THIS%C(I)           ! Should be considered a combination
                                                                                                                                                  ! of
different node concentrations
                                        END IF
                                        PEL(K+J) = I
                              END DO

                    ! If no change has been made on the number of particles, simply
                    ! move their positions, bearing in mind other cells may have changed
                    ELSE

                              NPR =THIS%PART%PARTSINELEM(I)

                              ALLOCATE(OLD_PARTS(NPR))

                              J=0
                              M=1
                              DO WHILE ((J.LE.NPR).AND.(M.LE.NPART1))
                                        IF(THIS%PART%ELEMENT(M).EQ.I) THEN
                                                  J=J+1
                                                  OLD_PARTS(J) = M
                                        END IF
                              M=M+1
                              END DO

                              DO J = 1, NPR
                                        PX(K+J) = THIS%PART%PX(OLD_PARTS(J))
                                        PY(K+J) = THIS%PART%PY(OLD_PARTS(J))
                                        PVOL(K+J) = THIS%PART%PVOL(OLD_PARTS(J))
                                        PC(K+J) = THIS%PART%PC(OLD_PARTS(J))
                                        PEL(K+J) = I
                              END DO

                              DEALLOCATE (OLD_PARTS)
                    END IF

                    K = K + THIS%PART%PARTSINELEM(I)

          END DO

          ! Store the new number of particles in NPART2. The old one still remains
          ! in NPART
          THIS%PART%NPART2 = K

          ! Deallocate particle properties and reallocate to new number K

          DEALLOCATE(THIS%PART%PX, THIS%PART%PY)
          DEALLOCATE(THIS%PART%PVOL, THIS%PART%PC)
          DEALLOCATE(THIS%PART%ELEMENT)

          ALLOCATE(THIS%PART%PX(K), THIS%PART%PY(K))
          ALLOCATE(THIS%PART%PVOL(K), THIS%PART%PC(K))
          ALLOCATE(THIS%PART%ELEMENT(K))

          ! Assign auxiliary variable values to type ones

          DO I=1, K
                    THIS%PART%PX(I) = PX(I)
                    THIS%PART%PY(I) = PY(I)
                    THIS%PART%PVOL(I) = PVOL(I)
                    THIS%PART%PC(I) = PC(I)
                    THIS%PART%ELEMENT(I) = PEL(I)
          END DO

          ! Deallocate auxiliary variables

          DEALLOCATE (PX, PY, PVOL, PC, LPX, LPY, PEL)
          DEALLOCATE (PC_OLD)

          RETURN
END SUBROUTINE PART_RESTRICT


END MODULE M_TR_TRACONF
```

D. Listado del archivo ROOT

El archivo ROOT contiene los parámetros fundamentales relativos a las clases de PROW que se utilizarán en la solución del modelo.

```
(ENGINE=TRACONF)
{
  ONLYFLOW:FALSE


  (MESH=FEM2DLTRI)
  {
   INPUTFILE:<EJE_MEF.DAT|TRACONF|444>
  }


  (TIMEDISCR=CONSSTEP)
  {
   INPUTFILE:<EJE_MEF.DAT|TRACONF|444>
  }


  (WOUT=TRACONF)
  {
   PARAMWFLOW:'    NIVELES    '
   OUTFT:<t9ht.out|ASCII|50>
   OUTFS:<t9hs.out|ASCII|51>
   PARAMWTRSP:'CONCENTRACIONES'
   OUTTT:<t9ct.out|ASCII|52>
   OUTTS:<t9cs.out|ASCII|53>
   OUTPUTFILE:<t9.out|ASCII|5>
   INPUTFILE:<EJE_MEF.DAT|TRACONF|444>
  }


  (FLOW=TRACONF)
  {
     ISTRANSIENT:FALSE

     (MATRIXA=SYMBAND){}

     (MATRIXD=DIAGONAL){}

     (TIMEINTG=FULLIMP)
     {
        (MATRIX=SYMBAND){}
     }

     INPUTFILE:<EJE_MEF.DAT|TRACONF|444>
  }

(TRANSPORT=TRACONF)
  {
     ISTRANSIENT:TRUE

     (MATRIXA=FULLBAND)
     {
     }

     (MATRIXD=DIAGONAL)
     {
     }
```

```
      (TIMEINTG=FULLIMP)
      {
         (MATRIX=FULLBAND){}
      }

      INPUTFILE:<EJE_MEF.DAT|TRACONF|444>
   }
}
```

E. Listados de resultados del modelo

X	Y	MEF T=15	MEF T=30	MEF T=90	MDF T=15	MDF T=30	MDF T=90
0	-105	3.712E-16	2.6692E-07	0.02336241	4.557E-16	1.3766E-07	0.02336241
10	-105	5.0522E-16	2.888E-07	0.03041116	8.1976E-16	1.9539E-07	0.03041116
20	-105	9.3988E-16	3.5987E-07	0.03835532	1.65E-15	2.9117E-07	0.03835532
30	-105	1.9301E-15	4.9144E-07	0.04739178	3.3069E-15	4.3719E-07	0.04739178
40	-105	3.9161E-15	6.9441E-07	0.05769553	6.3399E-15	6.4566E-07	0.05769553
50	-105	7.556E-15	9.7588E-07	0.0694197	1.1465E-14	9.2479E-07	0.0694197
60	-105	1.3659E-14	1.3356E-06	0.08260803	1.9371E-14	1.2763E-06	0.08260803
70	-105	2.2924E-14	1.7638E-06	0.0971949	3.033E-14	1.6907E-06	0.0971949
80	-105	3.5421E-14	2.2317E-06	0.11307093	4.3627E-14	2.1441E-06	0.11307093
90	-105	5.0009E-14	2.6995E-06	0.12995176	5.7298E-14	2.6029E-06	0.12995176
100	-105	6.5383E-14	3.1434E-06	0.1476405	6.8255E-14	3.0181E-06	0.1476405
110	-105	7.5108E-14	3.4448E-06	0.16519654	7.3659E-14	3.3461E-06	0.16519654
120	-105	8.0051E-14	3.6374E-06	0.18261988	7.214E-14	3.5432E-06	0.18261988
130	-105	7.7908E-14	3.6636E-06	0.19910114	6.4445E-14	3.5886E-06	0.19910114
140	-105	6.9808E-14	3.5275E-06	0.21402783	5.2916E-14	3.4769E-06	0.21402783
150	-105	5.803E-14	3.2512E-06	0.22687494	4.0266E-14	3.2274E-06	0.22687494
160	-105	4.5057E-14	2.8723E-06	0.23685499	2.8612E-14	2.875E-06	0.23685499
170	-105	3.2913E-14	2.4358E-06	0.2438586	1.9104E-14	2.4598E-06	0.2438586
180	-105	2.2756E-14	1.9867E-06	0.24766702	1.2075E-14	2.0237E-06	0.24766702
190	-105	1.4973E-14	1.5607E-06	0.24762399	7.2638E-15	1.6054E-06	0.24762399
200	-105	9.4173E-15	1.1828E-06	0.24412401	4.1785E-15	1.229E-06	0.24412401
210	-105	5.6876E-15	8.6628E-07	0.23734279	2.3089E-15	9.0979E-07	0.23734279
220	-105	3.3102E-15	6.1423E-07	0.22793657	1.2301E-15	6.5229E-07	0.22793657
230	-105	1.8632E-15	4.2236E-07	0.21577412	6.3399E-16	4.5341E-07	0.21577412
240	-105	1.0168E-15	2.8199E-07	0.20155543	3.17E-16	3.0635E-07	0.20155543
250	-105	5.3938E-16	1.8318E-07	0.18578362	1.5417E-16	2.0132E-07	0.18578362
260	-105	2.7883E-16	1.1587E-07	0.16898369	7.3086E-17	1.288E-07	0.16898369
270	-105	1.407E-16	7.144E-08	0.15170251	3.3841E-17	8.0407E-08	0.15170251
280	-105	6.9439E-17	4.3021E-08	0.13439945	1.5337E-17	4.9005E-08	0.13439945
290	-105	3.3592E-17	2.5331E-08	0.1175339	6.8106E-18	2.9204E-08	0.1175339
300	-105	1.5937E-17	1.4592E-08	0.10147771	2.9683E-18	1.7039E-08	0.10147771
310	-105	7.4253E-18	8.2368E-09	0.08651527	1.2717E-18	9.7382E-09	0.08651527
320	-105	3.4054E-18	4.5586E-09	0.07282158	5.3588E-19	5.4588E-09	0.07282158
330	-105	1.5371E-18	2.4766E-09	0.06054975	2.2249E-19	3.0024E-09	0.06054975
340	-105	6.8363E-19	1.322E-09	0.04974355	9.104E-20	1.6228E-09	0.04974355
350	-105	2.9995E-19	6.9336E-10	0.04035921	3.6755E-20	8.6234E-10	0.04035921
360	-105	1.2997E-19	3.5788E-10	0.03237487	1.4657E-20	4.508E-10	0.03237487
370	-105	5.5606E-20	1.8192E-10	0.02565927	5.7747E-21	2.3203E-10	0.02565927
380	-105	2.3525E-20	9.1077E-11	0.02010086	2.2501E-21	1.1769E-10	0.02010086
390	-105	9.8446E-21	4.4957E-11	0.01556837	8.6732E-22	5.8844E-11	0.01556837
400	-105	4.0784E-21	2.1898E-11	0.01192183	3.3094E-22	2.903E-11	0.01192183
0	-95	2.9168E-13	7.4822E-06	0.07096872	3.5807E-13	3.8589E-06	0.07096872
10	-95	3.9699E-13	8.0954E-06	0.09238093	6.4414E-13	5.4771E-06	0.09238093
20	-95	7.3853E-13	1.0088E-05	0.11651319	1.2965E-12	8.1619E-06	0.11651319
30	-95	1.5166E-12	1.3776E-05	0.14396353	2.5985E-12	1.2255E-05	0.14396353
40	-95	3.0772E-12	1.9465E-05	0.17526355	4.9817E-12	1.8099E-05	0.17526355
50	-95	5.9373E-12	2.7355E-05	0.21087842	9.0086E-12	2.5924E-05	0.21087842
60	-95	1.0733E-11	3.744E-05	0.25094103	1.5221E-11	3.5777E-05	0.25094103
70	-95	1.8013E-11	4.9441E-05	0.29525202	2.3832E-11	4.7392E-05	0.29525202
80	-95	2.7833E-11	6.2557E-05	0.34347916	3.4281E-11	6.0102E-05	0.34347916
90	-95	3.9295E-11	7.5673E-05	0.39475858	4.5023E-11	7.2963E-05	0.39475858
100	-95	5.1376E-11	8.8114E-05	0.44849224	5.3633E-11	8.4602E-05	0.44849224
110	-95	5.9018E-11	9.6564E-05	0.50182278	5.7879E-11	9.3796E-05	0.50182278
120	-95	6.2902E-11	0.00010196	0.55475021	5.6685E-11	9.9322E-05	0.55475021
130	-95	6.1218E-11	0.0001027	0.60481586	5.0639E-11	0.00010059	0.60481586
140	-95	5.4854E-11	9.8882E-05	0.65015913	4.158E-11	9.7464E-05	0.65015913
150	-95	4.5599E-11	9.1136E-05	0.68918523	3.164E-11	9.0471E-05	0.68918523
160	-95	3.5405E-11	8.0514E-05	0.71950194	2.2482E-11	8.0592E-05	0.71950194
170	-95	2.5862E-11	6.8278E-05	0.74077702	1.5012E-11	6.8953E-05	0.74077702
180	-95	1.7881E-11	5.5691E-05	0.75234597	9.488E-12	5.6728E-05	0.75234597
190	-95	1.1765E-11	4.3749E-05	0.75221528	5.7077E-12	4.5001E-05	0.75221528
200	-95	7.3998E-12	3.3156E-05	0.74158326	3.2833E-12	3.4452E-05	0.74158326
210	-95	4.4692E-12	2.4283E-05	0.72098372	1.8142E-12	2.5503E-05	0.72098372
220	-95	2.601E-12	1.7218E-05	0.69241018	9.666E-13	1.8285E-05	0.69241018
230	-95	1.4641E-12	1.1839E-05	0.65546391	4.9817E-13	1.271E-05	0.65546391
240	-95	7.9897E-13	7.9047E-06	0.61227134	2.4909E-13	8.5874E-06	0.61227134
250	-95	4.2383E-13	5.1348E-06	0.56436081	1.2114E-13	5.6434E-06	0.56436081
260	-95	2.1909E-13	3.2481E-06	0.51332712	5.7429E-14	3.6105E-06	0.51332712
270	-95	1.1056E-13	2.0026E-06	0.46083153	2.6591E-14	2.2539E-06	0.46083153
280	-95	5.4563E-14	1.2059E-06	0.40826949	1.2051E-14	1.3737E-06	0.40826949
290	-95	2.6396E-14	7.1007E-07	0.35703646	5.3515E-15	8.1864E-07	0.35703646
300	-95	1.2523E-14	4.0903E-07	0.30826207	2.3324E-15	4.7764E-07	0.30826207
310	-95	5.8346E-15	2.3089E-07	0.2628102	9.9928E-16	2.7298E-07	0.2628102
320	-95	2.6759E-15	1.2778E-07	0.22121243	4.2108E-16	1.5302E-07	0.22121243
330	-95	1.2078E-15	6.9423E-08	0.18393391	1.7483E-16	8.4162E-08	0.18393391
340	-95	5.3718E-16	3.7059E-08	0.15110756	7.1537E-17	4.549E-08	0.15110756
350	-95	2.357E-16	1.9436E-08	0.12260046	2.8881E-17	2.4173E-08	0.12260046

360	-95	1.0213E-16	1.0032E-08	0.09834617	1.1517E-17	1.2637E-08	0.09834617
370	-95	4.3694E-17	5.0996E-09	0.07794598	4.5376E-18	6.5041E-09	0.07794598
380	-95	1.8485E-17	2.553E-09	0.06106101	1.7681E-18	3.299E-09	0.06106101
390	-95	7.7356E-18	1.2602E-09	0.04729255	6.8151E-19	1.6495E-09	0.04729255
400	-95	3.2047E-18	6.1383E-10	0.03621531	2.6004E-19	8.1375E-10	0.03621531
0	-85	1.1767E-10	0.00015028	0.19291298	1.4446E-10	7.7509E-05	0.19291298
10	-85	1.6016E-10	0.0001626	0.25111742	2.5987E-10	0.00011001	0.25111742
20	-85	2.9795E-10	0.00020262	0.31671568	5.2305E-10	0.00016394	0.31671568
30	-85	6.1183E-10	0.0002767	0.39133343	1.0483E-09	0.00024615	0.39133343
40	-85	1.2414E-09	0.00039097	0.47641573	2.0098E-09	0.00036353	0.47641573
50	-85	2.3953E-09	0.00054945	0.57322698	3.6343E-09	0.00052069	0.57322698
60	-85	4.3299E-09	0.00075201	0.68212844	6.1406E-09	0.00071861	0.68212844
70	-85	7.2671E-09	0.00099305	0.80257821	9.6147E-09	0.00095189	0.80257821
80	-85	1.1228E-08	0.00125649	0.93367316	1.383E-08	0.00120718	0.93367316
90	-85	1.5853E-08	0.00151992	1.07306509	1.8164E-08	0.00146551	1.07306509
100	-85	2.0727E-08	0.00176981	1.21912832	2.1637E-08	0.00169928	1.21912832
110	-85	2.3809E-08	0.00193954	1.36409576	2.335E-08	0.00188394	1.36409576
120	-85	2.5376E-08	0.00204798	1.50796741	2.2868E-08	0.00199494	1.50796741
130	-85	2.4697E-08	0.00206271	1.64405996	2.0429E-08	0.00202048	1.64405996
140	-85	2.213E-08	0.0019861	1.76731575	1.6774E-08	0.00195761	1.76731575
150	-85	1.8396E-08	0.00183051	1.87339968	1.2764E-08	0.00181715	1.87339968
160	-85	1.4283E-08	0.00161717	1.95580905	9.07E-09	0.00161874	1.95580905
170	-85	1.0434E-08	0.00137141	2.01364072	6.0562E-09	0.00138496	2.01364072
180	-85	7.2138E-09	0.00111858	2.04508838	3.8277E-09	0.0011394	2.04508838
190	-85	4.7464E-09	0.00087872	2.04473312	2.3026E-09	0.00090386	2.04473312
200	-85	2.9853E-09	0.00066596	2.0158323	1.3246E-09	0.00069199	2.0158323
210	-85	1.803E-09	0.00048774	1.95983696	7.3192E-10	0.00051224	1.95983696
220	-85	1.0493E-09	0.00034583	1.882166	3.8996E-10	0.00036726	1.882166
230	-85	5.9065E-10	0.0002378	1.78173564	2.0098E-10	0.00025529	1.78173564
240	-85	3.2233E-10	0.00015877	1.66432606	1.0049E-10	0.00017248	1.66432606
250	-85	1.7099E-10	0.00010314	1.53409173	4.8871E-11	0.00011335	1.53409173
260	-85	8.8389E-11	6.5241E-05	1.39536779	2.3168E-11	7.2519E-05	1.39536779
270	-85	4.4603E-11	4.0223E-05	1.25266998	1.0728E-11	4.5272E-05	1.25266998
280	-85	2.2012E-11	2.4222E-05	1.10979154	4.8618E-12	2.7591E-05	1.10979154
290	-85	1.0649E-11	1.4262E-05	0.97052571	2.159E-12	1.6443E-05	0.97052571
300	-85	5.052E-12	8.2155E-06	0.83794319	9.4095E-13	9.5936E-06	0.83794319
310	-85	2.3538E-12	4.6376E-06	0.71439218	4.0314E-13	5.4829E-06	0.71439218
320	-85	1.0795E-12	2.5666E-06	0.60131772	1.6988E-13	3.0734E-06	0.60131772
330	-85	4.8728E-13	1.3944E-06	0.49998421	7.0531E-14	1.6904E-06	0.49998421
340	-85	2.1671E-13	7.4434E-07	0.41075293	2.886E-14	9.1368E-07	0.41075293
350	-85	9.5086E-14	3.9038E-07	0.3332626	1.1651E-14	4.8552E-07	0.3332626
360	-85	4.12E-14	2.015E-07	0.2673326	4.6463E-15	2.5381E-07	0.2673326
370	-85	1.7627E-14	1.0243E-07	0.21187915	1.8306E-15	1.3064E-07	0.21187915
380	-85	7.4574E-15	5.1279E-08	0.16598103	7.1329E-16	6.6262E-08	0.16598103
390	-85	3.1208E-15	2.5312E-08	0.12855447	2.7494E-16	3.3131E-08	0.12855447
400	-85	1.2929E-15	1.2329E-08	0.09844342	1.0491E-16	1.6345E-08	0.09844342
0	-75	2.4373E-08	0.00216286	0.46924645	2.9921E-08	0.0011155	0.46924645
10	-75	3.3173E-08	0.00234014	0.6108244	5.3825E-08	0.00158327	0.6108244
20	-75	6.1713E-08	0.00291605	0.77038727	1.0834E-07	0.00235936	0.77038727
30	-75	1.2673E-07	0.00398222	0.95188941	2.1713E-07	0.00354257	0.95188941
40	-75	2.5713E-07	0.00562684	1.15884575	4.1628E-07	0.00523187	1.15884575
50	-75	4.9613E-07	0.0079076	1.39433189	7.5277E-07	0.00749369	1.39433189
60	-75	8.9684E-07	0.01082281	1.65922657	1.2719E-06	0.01034217	1.65922657
70	-75	1.5052E-06	0.01429187	1.95221168	1.9915E-06	0.01369956	1.95221168
80	-75	2.3257E-06	0.01808325	2.27109036	2.8646E-06	0.0173736	2.27109036
90	-75	3.2836E-06	0.02187462	2.61015083	3.7622E-06	0.02109147	2.61015083
100	-75	4.293E-06	0.02547092	2.96543875	4.4816E-06	0.02445592	2.96543875
110	-75	4.9316E-06	0.02791368	3.31806124	4.8364E-06	0.02711356	3.31806124
120	-75	5.2562E-06	0.02947434	3.66801831	4.7367E-06	0.02871097	3.66801831
130	-75	5.1155E-06	0.02968638	3.99905329	4.2315E-06	0.02907852	3.99905329
140	-75	4.5836E-06	0.02858375	4.29886383	3.4744E-06	0.02817379	4.29886383
150	-75	3.8103E-06	0.02634454	4.55690507	2.6438E-06	0.02615229	4.55690507
160	-75	2.9584E-06	0.02327412	4.75735973	1.8786E-06	0.02329674	4.75735973
170	-75	2.1611E-06	0.0197372	4.89803094	1.2544E-06	0.01993229	4.89803094
180	-75	1.4942E-06	0.0160985	4.97452502	7.9283E-07	0.01639819	4.97452502
190	-75	9.831E-07	0.0126464	4.97366089	4.7694E-07	0.01300829	4.97366089
200	-75	6.1834E-07	0.00958446	4.90336179	2.7436E-07	0.00995907	4.90336179
210	-75	3.7345E-07	0.00701956	4.7671573	1.516E-07	0.00737212	4.7671573
220	-75	2.1734E-07	0.00497713	4.57822848	8.077E-08	0.00528559	4.57822848
230	-75	1.2234E-07	0.00342242	4.33393913	4.1628E-08	0.00367404	4.33393913
240	-75	6.6762E-08	0.002285	4.04834906	2.0814E-08	0.00248235	4.04834906
250	-75	3.5416E-08	0.00148432	3.73156377	1.0122E-08	0.00163134	3.73156377
260	-75	1.8308E-08	0.00093894	3.39412812	4.7988E-09	0.00104369	3.39412812
270	-75	9.2384E-09	0.00057888	3.04702635	2.222E-09	0.00065155	3.04702635
280	-75	4.5593E-09	0.0003486	2.6994852	1.007E-09	0.00039709	2.6994852
290	-75	2.2057E-09	0.00020526	2.36073144	4.4718E-10	0.00023664	2.36073144
300	-75	1.0464E-09	0.00011824	2.03823435	1.949E-10	0.00013807	2.03823435
310	-75	4.8754E-10	6.6743E-05	1.73770572	8.3501E-11	7.8909E-05	1.73770572
320	-75	2.236E-10	3.6938E-05	1.46266052	3.5186E-11	4.4233E-05	1.46266052
330	-75	1.0093E-10	2.0068E-05	1.21617432	1.4609E-11	2.4329E-05	1.21617432
340	-75	4.4887E-11	1.0713E-05	0.99912587	5.9777E-12	1.315E-05	0.99912587
350	-75	1.9695E-11	5.6184E-06	0.81063643	2.4133E-12	6.9876E-06	0.81063643
360	-75	8.5337E-12	2.8999E-06	0.65026662	9.6238E-13	3.6528E-06	0.65026662
370	-75	3.6511E-12	1.4741E-06	0.51538023	3.7916E-13	1.8801E-06	0.51538023
380	-75	1.5446E-12	7.38E-07	0.40373648	1.4774E-13	9.5364E-07	0.40373648

390	-75	6.464E-13	3.6429E-07	0.31269916	5.6948E-14	4.7682E-07	0.31269916
400	-75	2.6779E-13	1.7744E-07	0.23945629	2.173E-14	2.3523E-07	0.23945629
0	-65	2.5919E-06	0.02230402	1.02137587	3.1819E-06	0.01150334	1.02137587
10	-65	3.5277E-06	0.02413208	1.32953866	5.7239E-06	0.01632713	1.32953866
20	-65	6.5627E-06	0.03007107	1.676848	1.1521E-05	0.02433034	1.676848
30	-65	1.3476E-05	0.04106564	2.07191098	2.309E-05	0.03653195	2.07191098
40	-65	2.7344E-05	0.05802545	2.52237835	4.4268E-05	0.05395241	2.52237835
50	-65	5.2759E-05	0.08154526	3.03494454	8.0051E-05	0.07727688	3.03494454
60	-65	9.5372E-05	0.11160758	3.61152222	0.00013526	0.10665113	3.61152222
70	-65	0.00016007	0.14738149	4.24924237	0.00021178	0.14127339	4.24924237
80	-65	0.00024732	0.18647913	4.94332326	0.00030463	0.17916107	4.94332326
90	-65	0.00034918	0.22557677	5.68133242	0.00040008	0.21750067	5.68133242
100	-65	0.00045653	0.26266267	6.45466274	0.00047659	0.25219582	6.45466274
110	-65	0.00052444	0.2878531	7.22219141	0.00051432	0.27960207	7.22219141
120	-65	0.00055895	0.30394698	7.98391844	0.00050371	0.29607497	7.98391844
130	-65	0.00054399	0.30613365	8.70445908	0.00044998	0.2998652	8.70445908
140	-65	0.00048743	0.29476297	9.35703568	0.00036948	0.29053541	9.35703568
150	-65	0.00040519	0.27167175	9.91869598	0.00028115	0.26968917	9.91869598
160	-65	0.00031461	0.24000878	10.3550116	0.00019978	0.24024203	10.3550116
170	-65	0.00022982	0.20353514	10.6612008	0.0001334	0.20554688	10.6612008
180	-65	0.00015889	0.1660119	10.8277001	8.4311E-05	0.1691024	10.8277001
190	-65	0.00010455	0.13041293	10.8258192	5.0719E-05	0.13414485	10.8258192
200	-65	6.5756E-05	0.09883744	10.672804	2.9176E-05	0.10270055	10.672804
210	-65	3.9713E-05	0.07238749	10.3763373	1.6122E-05	0.07602319	10.3763373
220	-65	2.3113E-05	0.05132549	9.96510915	8.5894E-06	0.05450637	9.96510915
230	-65	1.301E-05	0.03529284	9.43338164	4.4268E-06	0.03788768	9.43338164
240	-65	7.0997E-06	0.02356354	8.81175775	2.2134E-06	0.0255986	8.81175775
250	-65	3.7662E-06	0.01530668	8.1222334	1.0765E-06	0.01682277	8.1222334
260	-65	1.9469E-06	0.00968257	7.38776086	5.1032E-07	0.01076278	7.38776086
270	-65	9.8244E-07	0.00596961	6.63224875	2.3629E-07	0.0067189	6.63224875
280	-65	4.8485E-07	0.00359488	5.87578029	1.0709E-07	0.0040949	5.87578029
290	-65	2.3456E-07	0.0021167	5.13843872	4.7554E-08	0.00244032	5.13843872
300	-65	1.1128E-07	0.00121929	4.4364819	2.0726E-08	0.00142381	4.4364819
310	-65	5.1847E-08	0.00068828	3.78234229	8.8797E-09	0.00081373	3.78234229
320	-65	2.3778E-08	0.00038092	3.18367067	3.7418E-09	0.00045614	3.18367067
330	-65	1.0733E-08	0.00020695	2.64716143	1.5536E-09	0.00025088	2.64716143
340	-65	4.7734E-09	0.00011047	2.17472728	6.3568E-10	0.0001356	2.17472728
350	-65	2.0944E-09	5.7938E-05	1.76445551	2.5664E-10	7.2058E-05	1.76445551
360	-65	9.075E-10	2.9905E-05	1.41538978	1.0234E-10	3.7669E-05	1.41538978
370	-65	3.8827E-10	1.5202E-05	1.12179204	4.0321E-11	1.9388E-05	1.12179204
380	-65	1.6426E-10	7.6105E-06	0.87878491	1.5711E-11	9.8342E-06	0.87878491
390	-65	6.874E-11	3.7567E-06	0.68063034	6.056E-12	4.9171E-06	0.68063034
400	-65	2.8477E-11	1.8298E-06	0.52120772	2.3108E-12	2.4257E-06	0.52120772
0	-55	0.00014151	0.16480568	1.98936855	0.00017373	0.0849988	1.98936855
10	-55	0.0001926	0.17831328	2.58958771	0.00031252	0.12064206	2.58958771
20	-55	0.00035831	0.22219683	3.26605393	0.00062902	0.17977822	3.26605393
30	-55	0.00073579	0.30343633	4.03553154	0.00126069	0.26993662	4.03553154
40	-55	0.00149295	0.42875328	4.91292218	0.00241696	0.39865739	4.91292218
50	-55	0.00288056	0.60254248	5.91126479	0.00437066	0.5710032	5.91126479
60	-55	0.00520714	0.82467468	7.03428477	0.00738474	0.7880512	7.03428477
70	-55	0.00873941	1.08901006	8.27639401	0.01156268	1.043877	8.27639401
80	-55	0.0135034	1.37790472	9.62827899	0.01663204	1.32383122	9.62827899
90	-55	0.01906476	1.66679938	11.0657245	0.02184379	1.60712464	11.0657245
100	-55	0.02492583	1.94082921	12.5719663	0.02602078	1.86348903	12.5719663
110	-55	0.0286333	2.12696268	14.0669081	0.0280808	2.06599535	14.0669081
120	-55	0.03051783	2.24588128	15.5505497	0.02750172	2.18771457	15.5505497
130	-55	0.02970098	2.2620387	16.9539713	0.02456833	2.21572077	16.9539713
140	-55	0.02661303	2.17802012	18.2250169	0.02017299	2.14678245	18.2250169
150	-55	0.02212282	2.00739778	19.3189818	0.01535046	1.99274838	19.3189818
160	-55	0.01717709	1.77343834	20.1688085	0.01090765	1.7751618	20.1688085
170	-55	0.01254757	1.50393259	20.7651836	0.00728316	1.51879742	20.7651836
180	-55	0.00867533	1.22667127	21.08948	0.00460324	1.24950709	21.08948
190	-55	0.00570799	0.96362849	21.0858165	0.00276916	0.99120382	21.0858165
200	-55	0.00359014	0.73031535	20.7877837	0.00159295	0.75886013	20.7877837
210	-55	0.00216828	0.53487521	20.2103453	0.00088021	0.56173961	20.2103453
220	-55	0.00126193	0.37924695	19.4093823	0.00046896	0.40275061	19.4093823
230	-55	0.00071032	0.26078075	18.3737185	0.0002417	0.27995422	18.3737185
240	-55	0.00038763	0.17411235	17.1629605	0.00012085	0.18914952	17.1629605
250	-55	0.00020563	0.11310194	15.8199505	5.8772E-05	0.12430441	15.8199505
260	-55	0.0001063	0.07154505	14.3893933	2.7862E-05	0.07952682	14.3893933
270	-55	5.3639E-05	0.04410975	12.9178567	1.2901E-05	0.04964636	12.9178567
280	-55	2.6472E-05	0.0265628	11.4444573	5.8468E-06	0.03025746	11.4444573
290	-55	1.2806E-05	0.01564038	10.008312	2.5964E-06	0.01803168	10.008312
300	-55	6.0756E-06	0.00900938	8.64108681	1.1316E-06	0.01052063	8.64108681
310	-55	2.8307E-06	0.00508571	7.36699684	4.8482E-07	0.00601271	7.36699684
320	-55	1.2982E-06	0.00281462	6.20094374	2.0429E-07	0.00337044	6.20094374
330	-55	5.86E-07	0.00152914	5.15596644	8.4821E-08	0.00185379	5.15596644
340	-55	2.6062E-07	0.00081627	4.23579034	3.4707E-08	0.00100198	4.23579034
350	-55	1.1435E-07	0.00042811	3.43669006	1.4012E-08	0.00053244	3.43669006
360	-55	4.9548E-08	0.00022097	2.75680286	5.5877E-09	0.00027834	2.75680286
370	-55	2.1199E-08	0.00011233	2.18495254	2.2015E-09	0.00014326	2.18495254
380	-55	8.9683E-09	5.6234E-05	1.71163929	8.578E-10	7.2665E-05	1.71163929
390	-55	3.7531E-09	2.7758E-05	1.32568689	3.3065E-10	3.6333E-05	1.32568689
400	-55	1.5548E-09	1.3521E-05	1.01517403	1.2616E-10	1.7924E-05	1.01517403
0	-45	0.00396685	0.87256202	3.46728835	0.00486979	0.45002529	3.46728835

10	-45	0.00539903	0.94407788	4.51341573	0.00876031	0.63873821	4.51341573
20	-45	0.01004401	1.17641891	5.69243478	0.01763238	0.95183399	5.69243478
30	-45	0.02062544	1.60654065	7.03356423	0.03533928	1.42917674	7.03356423
40	-45	0.04184976	2.27002997	8.56277628	0.06775135	2.1106876	8.56277628
50	-45	0.08074683	3.19015517	10.3027966	0.12251659	3.02317075	10.3027966
60	-45	0.14596467	4.3662319	12.2601182	0.20700619	4.17232923	12.2601182
70	-45	0.2449798	5.76575294	14.4250016	0.32412077	5.52679641	14.4250016
80	-45	0.3785222	7.29530282	16.7812141	0.46622298	7.00901123	16.7812141
90	-45	0.53441633	8.82485271	19.2865509	0.61231682	8.50890542	19.2865509
100	-45	0.69871165	10.2757009	21.9117933	0.72940478	9.86622411	21.9117933
110	-45	0.80263798	11.2611827	24.5173407	0.78715044	10.9383918	24.5173407
120	-45	0.85546444	11.8907961	27.1031931	0.77091779	11.582833	27.1031931
130	-45	0.83256664	11.9763414	29.5492291	0.6886901	11.7311116	29.5492291
140	-45	0.74600653	11.5315059	31.764546	0.56548164	11.3661183	31.764546
150	-45	0.62013873	10.6281475	33.6712272	0.43029826	10.5505865	33.6712272
160	-45	0.48150167	9.38945166	35.1523979	0.30575925	9.3985765	35.1523979
170	-45	0.35172874	7.96255613	36.1918254	0.20415883	8.0412578	36.1918254
180	-45	0.24318351	6.49459886	36.7570444	0.12903626	6.61550287	36.7570444
190	-45	0.16000424	5.10192144	36.7506593	0.077624	5.24791875	36.7506593
200	-45	0.10063734	3.86664737	36.2312154	0.04465309	4.01777739	36.2312154
210	-45	0.06078038	2.83189147	35.2247927	0.02467363	2.97412478	35.2247927
220	-45	0.03537386	2.00791918	33.8287871	0.01314578	2.13235907	33.8287871
230	-45	0.01991148	1.38070107	32.0237194	0.00677514	1.48221482	32.0237194
240	-45	0.0108659	0.92183611	29.9134783	0.00338757	1.00145026	29.9134783
250	-45	0.0057641	0.59881707	27.5727341	0.00164748	0.65812848	27.5727341
260	-45	0.00297967	0.37879457	25.0794031	0.00078103	0.42105395	25.0794031
270	-45	0.00150359	0.23353866	22.5146486	0.00036164	0.26285218	22.5146486
280	-45	0.00074206	0.14063647	19.9466476	0.0001639	0.16019782	19.9466476
290	-45	0.00035898	0.08280785	17.4435771	7.2781E-05	0.09546855	17.4435771
300	-45	0.00017031	0.04770006	15.060628	3.172E-05	0.05570139	15.060628
310	-45	7.935E-05	0.02692624	12.8400051	1.359E-05	0.03183426	12.8400051
320	-45	3.6392E-05	0.01490199	10.8076806	5.7267E-06	0.01784475	10.8076806
330	-45	1.6427E-05	0.00809601	8.9863803	2.3777E-06	0.0098149	8.9863803
340	-45	7.3056E-06	0.00432175	7.38259711	9.7289E-07	0.00530495	7.38259711
350	-45	3.2055E-06	0.00226661	5.98983802	3.9278E-07	0.002819	5.98983802
360	-45	1.3889E-06	0.00116992	4.80485652	1.5663E-07	0.00147366	4.80485652
370	-45	5.9424E-07	0.00059471	3.80817344	6.1711E-08	0.0007585	3.80817344
380	-45	2.514E-07	0.00029773	2.98323152	2.4046E-08	0.00038473	2.98323152
390	-45	1.052E-07	0.00014697	2.31055161	9.2686E-09	0.00019236	2.31055161
400	-45	4.3584E-08	7.1584E-05	1.76935595	3.5366E-09	9.4898E-05	1.76935595
0	-35	0.05709062	3.31021051	5.40766439	0.07008568	1.70724648	5.40766439
10	-35	0.07770235	3.58151796	7.03922922	0.12607762	2.42316063	7.03922922
20	-35	0.14455258	4.46294264	8.87805504	0.25376377	3.61094205	8.87805504
30	-35	0.29683966	6.0946817	10.969712	0.50859989	5.42182192	10.969712
40	-35	0.60229818	8.61173981	13.3547071	0.97507178	8.00724778	13.3547071
50	-35	1.16210157	12.1023892	16.0684837	1.76324842	11.4689058	16.0684837
60	-35	2.10071125	16.5640338	19.1211685	2.97921579	15.8284314	19.1211685
70	-35	3.52572869	21.8733518	22.4975714	4.66471889	20.9668301	22.4975714
80	-35	5.4476598	27.6759561	26.1723758	6.70984196	26.5898609	26.1723758
90	-35	7.69127492	33.4785604	30.079758	8.8124123	32.2799613	30.079758
100	-35	10.0557994	38.9825967	34.1741476	10.4975324	37.4291777	34.1741476
110	-35	11.5514985	42.7211874	38.2378206	11.328603	41.4966259	38.2378206
120	-35	12.3117725	45.1097315	42.2707768	11.0949841	43.9414219	42.2707768
130	-35	11.9822292	45.4342619	46.085672	9.9115702	44.5039413	46.085672
140	-35	10.7364634	43.7467036	49.5407323	8.13836424	43.1192781	49.5407323
150	-35	8.92498455	40.3196621	52.5144372	6.19281646	40.0254212	52.5144372
160	-35	6.92973165	35.6204613	54.8245058	4.40046142	35.6550779	54.8245058
170	-35	5.06205057	30.2072936	56.4456213	2.93823673	30.5058616	56.4456213
180	-35	3.49987674	24.6383512	57.3271502	1.85707898	25.0970209	57.3271502
190	-35	2.30276755	19.3549956	57.3171918	1.11715805	19.9088609	57.3171918
200	-35	1.44836413	14.668776	56.5070548	0.64264354	15.2421131	56.5070548
210	-35	0.87474613	10.7432558	54.9374144	0.35510077	11.2828417	54.9374144
220	-35	0.50909763	7.61737854	52.7601713	0.18919303	8.08946215	52.7601713
230	-35	0.2865643	5.23792134	49.9449452	0.09750718	5.62303079	49.9449452
240	-35	0.15638118	3.49714004	46.6537637	0.04875359	3.79916971	46.6537637
250	-35	0.0829564	2.27171309	43.0030839	0.02371041	2.49672087	43.0030839
260	-35	0.04288309	1.4370208	39.1144264	0.01124052	1.59733884	39.1144264
270	-35	0.02163958	0.88596811	35.114375	0.00520472	0.99717387	35.114375
280	-35	0.01067961	0.53352805	31.1092603	0.00235879	0.60773734	31.1092603
290	-35	0.00516645	0.31414547	27.2054127	0.00104746	0.36217597	27.2054127
300	-35	0.00245106	0.18095817	23.4889093	0.00045651	0.21131259	23.4889093
310	-35	0.001142	0.1021492	20.0255736	0.00019559	0.1207686	20.0255736
320	-35	0.00052375	0.0565332	16.8559127	8.2418E-05	0.06769705	16.8559127
330	-35	0.00023641	0.03071356	14.0153699	3.4219E-05	0.03723446	14.0153699
340	-35	0.00010514	0.01639528	11.5140719	1.4002E-05	0.02012521	11.5140719
350	-35	4.6133E-05	0.00859876	9.34189213	5.6528E-06	0.01069436	9.34189213
360	-35	1.9989E-05	0.00443828	7.49376713	2.2542E-06	0.00559058	7.49376713
370	-35	8.5522E-06	0.00225614	5.93931678	8.8813E-07	0.0028775	5.93931678
380	-35	3.6181E-06	0.0011295	4.65271798	3.4606E-07	0.00145952	4.65271798
390	-35	1.5141E-06	0.00055754	3.60359058	1.3339E-07	0.00072976	3.60359058
400	-35	6.2726E-07	0.00027157	2.75952911	5.0898E-08	0.00036001	2.75952911
0	-25	0.42184583	8.99808508	7.54700361	0.51786701	4.64077708	7.54700361
10	-25	0.57414706	9.73557519	9.82403577	0.93159464	6.58683352	9.82403577
20	-25	1.06810713	12.1315359	12.3903239	1.87507474	9.81555817	12.3903239
30	-25	2.19336492	16.5670625	15.3094664	3.75807314	14.73804	15.3094664

40	-25	4.45041505	23.4091358	18.6379951	7.2048601	21.7659561	18.6379951
50	-25	8.58683372	32.8977048	22.4253754	13.0287415	31.1757183	22.4253754
60	-25	15.5222732	45.025712	26.6857403	22.0135926	43.0261375	26.6857403
70	-25	26.0518071	59.4579347	31.3978902	34.4678696	56.9937533	31.3978902
80	-25	40.2530639	75.2310486	36.5264929	49.5793986	72.2787357	36.5264929
90	-25	56.8312619	91.0041624	41.9796839	65.1154088	87.7460323	41.9796839
100	-25	74.3028662	105.965684	47.6938651	77.566856	101.743053	47.6938651
110	-25	85.3546702	116.128227	53.3651775	83.7076833	112.799524	53.3651775
120	-25	90.9723774	122.620963	58.9936213	81.9814599	119.445169	58.9936213
130	-25	88.5373637	123.503129	64.3177364	73.2371482	120.974255	64.3177364
140	-25	79.33233	118.915869	69.1396616	60.13483	117.21035	69.1396616
150	-25	65.9472115	109.600205	73.289801	45.7590683	108.800375	73.289801
160	-25	51.2041759	96.8264528	76.5137615	32.5152563	96.9205504	76.5137615
170	-25	37.4037756	82.1119372	78.7762104	21.710796	82.9235291	78.7762104
180	-25	25.8607856	66.9739823	80.0064831	13.7220607	68.2207757	80.0064831
190	-25	17.0152786	52.6123327	79.9925851	8.25474348	54.1178947	79.9925851
200	-25	10.7020438	39.8738672	78.8619478	4.74852916	41.4323591	78.8619478
210	-25	6.46354822	29.2031969	76.6713381	2.6238595	30.6699436	76.6713381
220	-25	3.76175094	20.7061817	73.6327506	1.39795793	21.989438	73.6327506
230	-25	2.1174397	14.2381464	69.7037861	0.72048601	15.2849824	69.7037861
240	-25	1.15550932	9.50621223	65.1105723	0.360243	10.327214	65.1105723
250	-25	0.61296952	6.17515643	60.0156382	0.17519752	6.78679097	60.0156382
260	-25	0.31686555	3.90622752	54.5885795	0.08305681	4.34201713	54.5885795
270	-25	0.15989608	2.40831101	49.0060581	0.03845799	2.71059962	49.0060581
280	-25	0.07891221	1.45027959	43.4164702	0.0174292	1.65200137	43.4164702
290	-25	0.03817518	0.85393592	37.968212	0.00773971	0.98449637	37.968212
300	-25	0.01811101	0.49189532	32.7814136	0.00337321	0.57440717	32.7814136
310	-25	0.00843828	0.27767032	27.9479394	0.00144522	0.32828308	27.9479394
320	-25	0.00386999	0.15367318	23.5243212	0.00060899	0.18401966	23.5243212
330	-25	0.00174685	0.08348811	19.5600243	0.00025285	0.10121375	19.5600243
340	-25	0.0007769	0.04456699	16.0691818	0.00010346	0.054706	16.0691818
350	-25	0.00034088	0.02337385	13.0376607	4.1769E-05	0.02907028	13.0376607
360	-25	0.0001477	0.01206449	10.4583945	1.6657E-05	0.01519677	10.4583945
370	-25	6.3193E-05	0.00613281	8.2889843	6.5625E-06	0.00782186	8.2889843
380	-25	2.6734E-05	0.00307029	6.49339103	2.5571E-06	0.00396739	6.49339103
390	-25	1.1188E-05	0.00151556	5.02921579	9.8565E-07	0.0019837	5.02921579
400	-25	4.6348E-06	0.0007382	3.85123311	3.7609E-07	0.00097862	3.85123311
0	-15	1.60034298	17.5258766	9.42506692	1.96461544	9.0389995	9.42506692
10	-15	2.17812327	18.9623112	12.268736	3.53416066	12.8293999	12.268736
20	-15	4.05204371	23.6290054	15.473642	7.11341086	19.1180968	15.473642
30	-15	8.32089807	32.2682316	19.1192098	14.2568814	28.7057822	19.1192098
40	-15	16.8833967	45.5947708	23.2760391	27.3328464	42.3942937	23.2760391
50	-15	32.5755954	64.0759795	28.0059048	49.4267184	60.7220078	28.0059048
60	-15	58.8863496	87.698112	33.3264566	83.5122594	83.8034727	33.3264566
70	-15	98.8319041	115.808243	39.2112197	130.75965	111.008673	39.2112197
80	-15	152.706756	146.530074	45.6160693	188.087773	140.779754	45.6160693
90	-15	215.598934	177.251905	52.4262808	247.026236	170.905934	52.4262808
100	-15	281.880398	206.39297	59.5624295	294.262891	198.168409	59.5624295
110	-15	323.807272	226.186902	66.6450416	317.559151	219.703473	66.6450416
120	-15	345.118987	238.833024	73.6741172	311.010433	232.647421	73.6741172
130	-15	335.881354	240.551248	80.3231324	277.837418	235.625674	80.3231324
140	-15	300.960513	231.616487	86.3449881	228.131574	228.294589	86.3449881
150	-15	250.181819	213.47205	91.5278851	173.594708	211.914194	91.5278851
160	-15	194.251638	188.592178	95.5541245	123.352084	188.775455	95.5541245
170	-15	141.897503	159.932215	98.3795813	82.3635498	161.512981	98.3795813
180	-15	98.1072319	130.447505	99.916006	52.0569412	132.875927	99.916006
190	-15	64.5503161	102.474831	99.8986494	31.3157553	105.407266	99.8986494
200	-15	40.6	77.6636885	98.4866543	18.0143426	80.6992162	98.4866543
210	-15	24.5205554	56.8800607	95.7509139	9.95405154	59.7368931	95.7509139
220	-15	14.2708338	40.3301349	91.9561772	5.30338811	42.8295769	91.9561772
230	-15	8.03286301	27.7321224	87.0494945	2.73328464	29.7710806	87.0494945
240	-15	4.38361863	18.5155732	81.3132646	1.36664232	20.1146662	81.3132646
250	-15	2.32540279	12.0275624	74.9504619	0.66464121	13.2188638	74.9504619
260	-15	1.20208268	7.60829232	68.1728857	0.31508996	8.45709458	68.1728857
270	-15	0.60659262	4.69074933	61.2011602	0.14589685	5.27952714	61.2011602
280	-15	0.29936672	2.82475894	54.2206097	0.06612058	3.21765931	54.2206097
290	-15	0.14482397	1.66324005	47.4165586	0.02936188	1.91753709	47.4165586
300	-15	0.06870714	0.95808125	40.9390313	0.01279684	1.1187924	40.9390313
310	-15	0.03201203	0.54082793	34.9027525	0.00548267	0.63940813	34.9027525
320	-15	0.01468144	0.29931448	29.3783219	0.0023103	0.35842136	29.3783219
330	-15	0.00662697	0.16261264	24.4275143	0.00095923	0.19713747	24.4275143
340	-15	0.0029473	0.08680464	20.0679796	0.00039249	0.10655275	20.0679796
350	-15	0.00129317	0.04552604	16.2820679	0.00015846	0.05662118	16.2820679
360	-15	0.00056032	0.02349842	13.0609542	6.319E-05	0.02959926	13.0609542
370	-15	0.00023973	0.01194509	10.3516887	2.4896E-05	0.01523491	10.3516887
380	-15	0.00010142	0.0059801	8.10926404	9.7007E-06	0.00772742	8.10926404
390	-15	4.2442E-05	0.00295191	6.28073046	3.7392E-06	0.00386371	6.28073046
400	-15	1.7583E-05	0.00143781	4.80960812	1.4268E-06	0.00190608	4.80960812
0	-5	3.11704251	24.4593312	10.532692	3.82654836	12.61494	10.532692
10	-5	4.24240483	26.4640371	13.7105464	6.88360503	17.9048699	13.7105464
20	-5	7.89230347	32.9769335	17.29209	13.8550325	26.6814534	17.29209
30	-5	16.2068964	45.033945	21.3660816	27.7686132	40.0621463	21.3660816
40	-5	32.8843665	63.6326286	26.0114176	53.2371154	59.166003	26.0114176
50	-5	63.4485961	89.4252331	31.2971326	96.2701019	84.7443885	31.2971326
60	-5	114.694948	122.392575	37.2429507	162.65967	116.957168	37.2429507

70	-5	192.498264	161.623424	43.8192857	254.685022	154.925084	43.8192857
80	-5	297.432147	204.499192	50.9768273	366.344958	196.473974	50.9768273
90	-5	419.929382	247.374961	58.5873685	481.141409	238.518445	58.5873685
100	-5	549.028047	288.044594	66.5621507	573.14585	276.566293	66.5621507
110	-5	630.690446	315.66925	74.4771048	618.520768	306.620896	74.4771048
120	-5	672.2	333.318336	82.3322308	605.765607	324.685631	82.3322308
130	-5	654.207547	335.71631	89.7626321	541.153397	328.842119	89.7626321
140	-5	586.191037	323.246847	96.4921707	444.339632	318.610765	96.4921707
150	-5	487.287646	297.924245	102.284157	338.116322	295.750083	102.284157
160	-5	378.350528	263.201587	106.783556	240.257053	263.457371	106.783556
170	-5	276.378596	223.203387	109.941058	160.42229	225.409522	109.941058
180	-5	191.086795	182.054159	111.658042	101.393077	185.443295	111.658042
190	-5	125.726848	143.015148	111.638646	60.9947627	147.10769	111.638646
200	-5	79.0780021	108.388409	110.060714	35.0871484	112.624829	110.060714
210	-5	47.7595204	79.3825195	107.003472	19.387845	83.3695503	107.003472
220	-5	27.7957888	56.2852373	102.762782	10.3295896	59.7734897	102.762782
230	-5	15.6458807	38.7032946	97.27947	5.32371154	41.54889	97.27947
240	-5	8.53812322	25.840564	90.8691237	2.66185577	28.0722781	90.8691237
250	-5	4.52926617	16.7858155	83.7585704	1.29454431	18.4484106	83.7585704
260	-5	2.34133735	10.6182273	76.1844998	0.61371144	11.8028263	76.1844998
270	-5	1.18148109	6.54646804	68.3934635	0.28416826	7.36817368	68.3934635
280	-5	0.58308676	3.94226867	60.5925653	0.12878531	4.49060531	60.5925653
290	-5	0.28207858	2.32123849	52.9889084	0.05718912	2.67613859	52.9889084
300	-5	0.13382324	1.3371101	45.7501481	0.02492484	1.56140057	45.7501481
310	-5	0.06235092	0.75478618	39.0044915	0.01067879	0.89236592	39.0044915
320	-5	0.02859554	0.41772701	32.8308349	0.00449985	0.5002173	32.8308349
330	-5	0.01290757	0.22694423	27.298213	0.00186832	0.27512751	27.298213
340	-5	0.00574056	0.12114563	22.4263498	0.00076447	0.14870634	22.4263498
350	-5	0.00251876	0.06353671	18.1955213	0.00030863	0.07902122	18.1955213
360	-5	0.00109136	0.03279469	14.5958653	0.00012308	0.04130909	14.5958653
370	-5	0.00046693	0.01667071	11.5682095	4.8491E-05	0.02126203	11.5682095
380	-5	0.00019754	0.00834591	9.0622572	1.8894E-05	0.01078449	9.0622572
390	-5	8.2667E-05	0.00411972	7.01883605	7.283E-06	0.00539224	7.01883605
400	-5	3.4247E-05	0.00200662	5.37482879	2.779E-06	0.00266015	5.37482879
0	0	3.38792612	25.5	10.68	4.15909091	13.1516667	10.68
10	0	4.61108699	27.59	13.9022992	7.48181818	18.6666667	13.9022992
20	0	8.5781766	34.38	17.5339335	15.0590909	27.8166667	17.5339335
30	0	17.6153414	46.95	21.664903	30.1818182	41.7666667	21.664903
40	0	35.7421511	66.34	26.3752078	57.8636364	61.6833333	26.3752078
50	0	68.962536	93.23	31.7348476	104.636364	88.35	31.7348476
60	0	124.662403	127.6	37.7638227	176.795455	121.933333	37.7638227
70	0	209.227143	168.5	44.432133	276.818182	161.516667	44.432133
80	0	323.280205	213.2	51.6897784	398.181818	204.833333	51.6897784
90	0	456.422946	257.9	59.406759	522.954545	248.666667	59.406759
100	0	596.740807	300.3	67.4930748	622.954545	288.333333	67.4930748
110	0	685.5	329.1	75.5187258	672.272727	319.666667	75.5187258
120	0	730.616902	347.5	83.4837119	658.409091	338.5	83.4837119
130	0	711.060833	350	91.0180332	588.181818	342.833333	91.0180332
140	0	637.133412	337	97.8416898	482.954545	332.166667	97.8416898
150	0	529.634915	310.6	103.714681	367.5	308.333333	103.714681
160	0	411.230722	274.4	108.277008	261.136364	274.666667	108.277008
170	0	300.397016	232.7	111.47867	174.363636	235	111.47867
180	0	207.693012	189.8	113.219668	110.204545	193.333333	113.219668
190	0	136.65302	149.1	113.2	66.2954545	153.366667	113.2
200	0	85.9502007	113	111.6	38.1363636	117.416667	111.6
210	0	51.9100161	82.76	108.5	21.0727273	86.9166667	108.5
220	0	30.2113554	58.68	104.2	11.2272727	62.3166667	104.2
230	0	17.0055711	40.35	98.64	5.78636364	43.3166667	98.64
240	0	9.28012071	26.94	92.14	2.89318182	29.2666667	92.14
250	0	4.92287774	17.5	84.93	1.40704545	19.2333333	84.93
260	0	2.54480905	11.07	77.25	0.66704545	12.305	77.25
270	0	1.28415658	6.825	69.35	0.30886364	7.68166667	69.35
280	0	0.63375936	4.11	61.44	0.13997727	4.68166667	61.44
290	0	0.30659235	2.42	53.73	0.06215909	2.79	53.73
300	0	0.14545302	1.394	46.39	0.02709091	1.62783333	46.39
310	0	0.06776947	0.7869	39.55	0.01160682	0.93033333	39.55
320	0	0.03108061	0.4355	33.29	0.00489091	0.5215	33.29
330	0	0.01402929	0.2366	27.68	0.00203068	0.28683333	27.68
340	0	0.00623944	0.1263	22.74	0.00083091	0.15503333	22.74
350	0	0.00273765	0.06624	18.45	0.00033545	0.08238333	18.45
360	0	0.00118621	0.03419	14.8	0.00013377	0.04306667	14.8
370	0	0.00050751	0.01738	11.73	5.2705E-05	0.02216667	11.73
380	0	0.00021471	0.008701	9.189	2.0536E-05	0.01124333	9.189
390	0	8.9851E-05	0.004295	7.117	7.9159E-06	0.00562167	7.117
400	0	3.7223E-05	0.002092	5.45	3.0205E-06	0.00277333	5.45
0	5	3.11704251	24.4593312	10.532692	3.82654836	12.61494	10.532692
10	5	4.24240483	26.4640371	13.7105464	6.88360503	17.9048699	13.7105464
20	5	7.89230347	32.9769335	17.29209	13.8550325	26.6814534	17.29209
30	5	16.2068964	45.033945	21.3660816	27.7686132	40.0621463	21.3660816
40	5	32.8843665	63.6326286	26.0114176	53.2371154	59.166003	26.0114176
50	5	63.4485961	89.4252331	31.2971326	96.2701019	84.7443885	31.2971326
60	5	114.694948	122.392575	37.2429507	162.65967	116.957168	37.2429507
70	5	192.498264	161.623424	43.8192857	254.685022	154.925084	43.8192857
80	5	297.432147	204.499192	50.9768273	366.344958	196.473974	50.9768273
90	5	419.929382	247.374961	58.5873685	481.141409	238.518445	58.5873685

100	5	549.028047	288.044594	66.5621507	573.14585	276.566293	66.5621507
110	5	630.690446	315.66925	74.4771048	618.520768	306.620896	74.4771048
120	5	672.2	333.318336	82.3322308	605.765607	324.685631	82.3322308
130	5	654.207547	335.71631	89.7626321	541.153397	328.842119	89.7626321
140	5	586.191037	323.246847	96.4921707	444.339632	318.610765	96.4921707
150	5	487.287646	297.924245	102.284157	338.116322	295.750083	102.284157
160	5	378.350528	263.201587	106.783556	240.257053	263.457371	106.783556
170	5	276.378596	223.203387	109.941058	160.42229	225.409522	109.941058
180	5	191.086795	182.054159	111.658042	101.393077	185.443295	111.658042
190	5	125.726848	143.015148	111.638646	60.9947627	147.10769	111.638646
200	5	79.0780021	108.388409	110.060714	35.0871484	112.624829	110.060714
210	5	47.7595204	79.3825195	107.003472	19.387845	83.3695503	107.003472
220	5	27.7957888	56.2852373	102.762782	10.3295896	59.7734897	102.762782
230	5	15.6458807	38.7032946	97.27947	5.32371154	41.54889	97.27947
240	5	8.53812322	25.840564	90.8691237	2.66185577	28.0722781	90.8691237
250	5	4.52926617	16.7858155	83.7585704	1.29454431	18.4484106	83.7585704
260	5	2.34133735	10.6182273	76.1844998	0.61371144	11.8028263	76.1844998
270	5	1.18148109	6.54646804	68.3934635	0.28416826	7.36817368	68.3934635
280	5	0.58308676	3.94226867	60.5925653	0.12878531	4.49060531	60.5925653
290	5	0.28207858	2.32123849	52.9889084	0.05718912	2.67613859	52.9889084
300	5	0.13382324	1.3371101	45.7501481	0.02492484	1.56140057	45.7501481
310	5	0.06235092	0.75478618	39.0044915	0.01067879	0.89236592	39.0044915
320	5	0.02859554	0.41772701	32.8308349	0.00449985	0.5002173	32.8308349
330	5	0.01290757	0.22694423	27.298213	0.00186832	0.27512751	27.298213
340	5	0.00574056	0.12114563	22.4263498	0.00076447	0.14870634	22.4263498
350	5	0.00251876	0.06353671	18.1955213	0.00030863	0.07902122	18.1955213
360	5	0.00109136	0.03279469	14.5958653	0.00012308	0.04130909	14.5958653
370	5	0.00046693	0.01667071	11.5682095	4.8491E-05	0.02126203	11.5682095
380	5	0.00019754	0.00834591	9.0622572	1.8894E-05	0.01078449	9.0622572
390	5	8.2667E-05	0.00411972	7.01883605	7.283E-06	0.00539224	7.01883605
400	5	3.4247E-05	0.00200662	5.37482879	2.779E-06	0.00266015	5.37482879
0	15	1.60034298	17.5258766	9.42506692	1.96461544	9.0389995	9.42506692
10	15	2.17812327	18.9623112	12.268736	3.53416066	12.8293999	12.268736
20	15	4.05204371	23.6290054	15.473642	7.11341086	19.1180968	15.473642
30	15	8.32089807	32.2682316	19.1192098	14.2568814	28.7057822	19.1192098
40	15	16.8833967	45.5947708	23.2760391	27.3328464	42.3942937	23.2760391
50	15	32.5755954	64.0759795	28.0059048	49.4267184	60.7220078	28.0059048
60	15	58.8863496	87.698112	33.3264566	83.5122594	83.8034727	33.3264566
70	15	98.8319041	115.808243	39.2112197	130.75965	111.008673	39.2112197
80	15	152.706756	146.530074	45.6160693	188.087773	140.779754	45.6160693
90	15	215.598934	177.251905	52.4262808	247.026236	170.905934	52.4262808
100	15	281.880398	206.39297	59.5624295	294.262891	198.168409	59.5624295
110	15	323.807272	226.186902	66.6450416	317.559151	219.703473	66.6450416
120	15	345.118987	238.833024	73.6741172	311.010433	232.647421	73.6741172
130	15	335.881354	240.551248	80.3231324	277.837418	235.625674	80.3231324
140	15	300.960513	231.616487	86.3449881	228.131574	228.294589	86.3449881
150	15	250.181819	213.47205	91.5278851	173.594708	211.914194	91.5278851
160	15	194.251638	188.592178	95.5541245	123.352084	188.775455	95.5541245
170	15	141.897503	159.932215	98.3795813	82.3635498	161.512981	98.3795813
180	15	98.1072319	130.447505	99.916006	52.0569412	132.875927	99.916006
190	15	64.5503161	102.474831	99.8986494	31.3157553	105.407266	99.8986494
200	15	40.6	77.6636885	98.4866543	18.0143426	80.6992162	98.4866543
210	15	24.5205554	56.8800607	95.7509139	9.95405154	59.7368931	95.7509139
220	15	14.2708338	40.3301349	91.9561772	5.30338811	42.8295769	91.9561772
230	15	8.03286301	27.7321224	87.0494945	2.73328464	29.7710806	87.0494945
240	15	4.38361863	18.5155732	81.3132646	1.36664232	20.1146662	81.3132646
250	15	2.32540279	12.0275624	74.9504619	0.66464121	13.2188638	74.9504619
260	15	1.20208268	7.60829232	68.1728857	0.31508996	8.45709458	68.1728857
270	15	0.60659262	4.69074933	61.2011602	0.14589685	5.27952714	61.2011602
280	15	0.29936672	2.82475894	54.2206097	0.06612058	3.21765931	54.2206097
290	15	0.14482397	1.66324005	47.4165586	0.02936188	1.91753709	47.4165586
300	15	0.06870714	0.95808125	40.9390313	0.01279684	1.1187924	40.9390313
310	15	0.03201203	0.54082793	34.9027525	0.00548267	0.63940813	34.9027525
320	15	0.01468144	0.29931448	29.3783219	0.0023103	0.35842136	29.3783219
330	15	0.00662697	0.16261264	24.4275143	0.00095923	0.19713747	24.4275143
340	15	0.0029473	0.08680464	20.0679796	0.00039249	0.10655275	20.0679796
350	15	0.00129317	0.04552604	16.2820679	0.00015846	0.05662118	16.2820679
360	15	0.00056032	0.02349842	13.0609542	6.319E-05	0.02959926	13.0609542
370	15	0.00023973	0.01194509	10.3516887	2.4896E-05	0.01523491	10.3516887
380	15	0.00010142	0.0059801	8.10926404	9.7007E-06	0.00772742	8.10926404
390	15	4.2442E-05	0.00295191	6.28073046	3.7392E-06	0.00386371	6.28073046
400	15	1.7583E-05	0.00143781	4.80960812	1.4268E-06	0.00190608	4.80960812
0	25	0.42184583	8.99808508	7.54700361	0.51786701	4.64077708	7.54700361
10	25	0.57414706	9.73557519	9.82403577	0.93159464	6.58683352	9.82403577
20	25	1.06810713	12.1315359	12.3903239	1.87507474	9.81555817	12.3903239
30	25	2.19336492	16.5670625	15.3094664	3.75807314	14.73804	15.3094664
40	25	4.45041505	23.4091358	18.6379951	7.2048601	21.7659561	18.6379951
50	25	8.58683372	32.8977048	22.4253754	13.0287415	31.1757183	22.4253754
60	25	15.5222732	45.025712	26.6857403	22.0135926	43.0261375	26.6857403
70	25	26.0518071	59.4579347	31.3978902	34.4678696	56.9937533	31.3978902
80	25	40.2530639	75.2310486	36.5264929	49.5793986	72.2787357	36.5264929
90	25	56.8312619	91.0041624	41.9796839	65.1154088	87.7460323	41.9796839
100	25	74.3028662	105.965684	47.6938651	77.566856	101.743053	47.6938651
110	25	85.3546702	116.128227	53.3651775	83.7076833	112.799524	53.3651775
120	25	90.9723774	122.620963	58.9936213	81.9814599	119.445169	58.9936213

130	25	88.5373637	123.503129	64.3177364	73.2371482	120.974255	64.3177364
140	25	79.33233	118.915869	69.1396616	60.13483	117.21035	69.1396616
150	25	65.9472115	109.600205	73.289801	45.7590683	108.800375	73.289801
160	25	51.2041759	96.8264528	76.5137615	32.5152563	96.9205504	76.5137615
170	25	37.4037756	82.1119372	78.7762104	21.710796	82.9235291	78.7762104
180	25	25.8607856	66.9739823	80.0064831	13.7220607	68.2207757	80.0064831
190	25	17.0152786	52.6123327	79.9925851	8.25474348	54.1178947	79.9925851
200	25	10.7020438	39.8738672	78.8619478	4.74852916	41.4323591	78.8619478
210	25	6.46354822	29.2031969	76.6713381	2.6238595	30.6699436	76.6713381
220	25	3.76175094	20.7061817	73.6327506	1.39795793	21.989438	73.6327506
230	25	2.1174397	14.2381464	69.7037861	0.72048601	15.2849824	69.7037861
240	25	1.15550932	9.50621223	65.1105723	0.360243	10.327214	65.1105723
250	25	0.61296952	6.17515643	60.0156382	0.17519752	6.78679097	60.0156382
260	25	0.31686555	3.90622752	54.5885795	0.08305681	4.34201713	54.5885795
270	25	0.15989608	2.40831101	49.0060581	0.03845799	2.71059962	49.0060581
280	25	0.07891221	1.45027959	43.4164702	0.0174292	1.65200137	43.4164702
290	25	0.03817518	0.85393592	37.968212	0.00773971	0.98449637	37.968212
300	25	0.01811101	0.49189532	32.7814136	0.00337321	0.57440717	32.7814136
310	25	0.00843828	0.27767032	27.9479394	0.00144522	0.32828308	27.9479394
320	25	0.00386999	0.15367318	23.5243212	0.00060899	0.18401966	23.5243212
330	25	0.00174685	0.08348811	19.5600243	0.00025285	0.10121375	19.5600243
340	25	0.0007769	0.04456699	16.0691818	0.00010346	0.054706	16.0691818
350	25	0.00034088	0.02337385	13.0376607	4.1769E-05	0.02907028	13.0376607
360	25	0.0001477	0.01206449	10.4583945	1.6657E-05	0.01519677	10.4583945
370	25	6.3193E-05	0.00613281	8.2889843	6.5625E-06	0.00782186	8.2889843
380	25	2.6734E-05	0.00307029	6.49339103	2.5571E-06	0.00396739	6.49339103
390	25	1.1188E-05	0.00151556	5.02921579	9.8565E-07	0.0019837	5.02921579
400	25	4.6348E-06	0.0007382	3.85123311	3.7609E-07	0.00097862	3.85123311
0	35	0.05709062	3.31021051	5.40766439	0.07008568	1.70724648	5.40766439
10	35	0.07770235	3.58151796	7.03922922	0.12607762	2.42316063	7.03922922
20	35	0.14455258	4.46294264	8.87805504	0.25376377	3.61094205	8.87805504
30	35	0.29683966	6.0946817	10.969712	0.50859989	5.42182192	10.969712
40	35	0.60229818	8.61173981	13.3547071	0.97507178	8.00724778	13.3547071
50	35	1.16210157	12.1023892	16.0684837	1.76324842	11.4689058	16.0684837
60	35	2.10071125	16.5640338	19.1211685	2.97921579	15.8284314	19.1211685
70	35	3.52572869	21.8733518	22.4975714	4.66471889	20.9668301	22.4975714
80	35	5.4476598	27.6759561	26.1723758	6.70984196	26.5898609	26.1723758
90	35	7.69127492	33.4785604	30.079758	8.8124123	32.2799613	30.079758
100	35	10.0557994	38.9825967	34.1741476	10.4975324	37.4291777	34.1741476
110	35	11.5514985	42.7211874	38.2378206	11.328603	41.4966259	38.2378206
120	35	12.3117725	45.1097315	42.2707768	11.0949841	43.9414219	42.2707768
130	35	11.9822292	45.4342619	46.085672	9.9115702	44.5039413	46.085672
140	35	10.7364634	43.7467036	49.5407323	8.13836424	43.1192781	49.5407323
150	35	8.92498455	40.3196621	52.5144372	6.19281646	40.0254212	52.5144372
160	35	6.92973165	35.6204613	54.8245058	4.40046142	35.6550779	54.8245058
170	35	5.06205057	30.2072936	56.4456213	2.93823673	30.5058616	56.4456213
180	35	3.49987674	24.6383512	57.3271502	1.85707898	25.0970209	57.3271502
190	35	2.30276755	19.3549956	57.3171918	1.11715805	19.9088609	57.3171918
200	35	1.44836413	14.668776	56.5070548	0.64264354	15.2421131	56.5070548
210	35	0.87474613	10.7432558	54.9374144	0.35510077	11.2828417	54.9374144
220	35	0.50909763	7.61737854	52.7601713	0.18919303	8.08946215	52.7601713
230	35	0.2865643	5.23792134	49.9449452	0.09750718	5.62303079	49.9449452
240	35	0.15638118	3.49714004	46.6537637	0.04875359	3.79916971	46.6537637
250	35	0.0829564	2.27171309	43.0030839	0.02371041	2.49672087	43.0030839
260	35	0.04288309	1.4370208	39.1144264	0.01124052	1.59733884	39.1144264
270	35	0.02163958	0.88596811	35.114375	0.00520472	0.99717387	35.114375
280	35	0.01067961	0.53352805	31.1092603	0.00235879	0.60773734	31.1092603
290	35	0.00516645	0.31414547	27.2054127	0.00104746	0.36217597	27.2054127
300	35	0.00245106	0.18095817	23.4889093	0.00045651	0.21131259	23.4889093
310	35	0.001142	0.1021492	20.0255736	0.00019559	0.1207686	20.0255736
320	35	0.00052375	0.0565332	16.8559127	8.2418E-05	0.06769705	16.8559127
330	35	0.00023641	0.03071356	14.0153699	3.4219E-05	0.03723446	14.0153699
340	35	0.00010514	0.01639528	11.5140719	1.4002E-05	0.02012521	11.5140719
350	35	4.6133E-05	0.00859876	9.34189213	5.6528E-06	0.01069436	9.34189213
360	35	1.9989E-05	0.00443828	7.49376713	2.2542E-06	0.00559058	7.49376713
370	35	8.5522E-06	0.00225614	5.93931678	8.8813E-07	0.0028775	5.93931678
380	35	3.6181E-06	0.0011295	4.65271798	3.4606E-07	0.00145952	4.65271798
390	35	1.5141E-06	0.00055754	3.60359058	1.3339E-07	0.00072976	3.60359058
400	35	6.2726E-07	0.00027157	2.75952911	5.0898E-08	0.00036001	2.75952911
0	45	0.00396685	0.87256202	3.46728835	0.00486979	0.45002529	3.46728835
10	45	0.00539903	0.94407788	4.51341573	0.00876031	0.63873821	4.51341573
20	45	0.01004401	1.17641891	5.69243478	0.01763238	0.95183399	5.69243478
30	45	0.02062544	1.60654065	7.03356423	0.03533928	1.42917674	7.03356423
40	45	0.04184976	2.27002997	8.56277628	0.06775135	2.1106876	8.56277628
50	45	0.08074683	3.19015517	10.3027966	0.12251659	3.02317075	10.3027966
60	45	0.14596467	4.3662319	12.2601182	0.20700619	4.17232923	12.2601182
70	45	0.2449798	5.76575294	14.4250016	0.32412077	5.52679641	14.4250016
80	45	0.3785222	7.29530282	16.7812141	0.46622298	7.00901123	16.7812141
90	45	0.53441633	8.82485271	19.2865509	0.61231682	8.50890542	19.2865509
100	45	0.69871165	10.2757009	21.9117933	0.72940478	9.86622411	21.9117933
110	45	0.80263798	11.2611827	24.5173407	0.78715044	10.9383918	24.5173407
120	45	0.85546444	11.8907961	27.1031931	0.77091779	11.582833	27.1031931
130	45	0.83256664	11.9763414	29.5492291	0.6886901	11.7311116	29.5492291
140	45	0.74600653	11.5315059	31.764546	0.56548164	11.3661183	31.764546
150	45	0.62013873	10.6281475	33.6712272	0.43029826	10.5505865	33.6712272

160	45	0.48150167	9.38945166	35.1523979	0.30575925	9.3985765	35.1523979
170	45	0.35172874	7.96255613	36.1918254	0.20415883	8.0412578	36.1918254
180	45	0.24318351	6.49459886	36.7570444	0.12903626	6.61550287	36.7570444
190	45	0.16000424	5.10192144	36.7506593	0.077624	5.24791875	36.7506593
200	45	0.10063734	3.86664737	36.2312154	0.04465309	4.01777739	36.2312154
210	45	0.06078038	2.83189147	35.2247927	0.02467363	2.97412478	35.2247927
220	45	0.03537386	2.00791918	33.8287871	0.01314578	2.13235907	33.8287871
230	45	0.01991148	1.38070107	32.0237194	0.00677514	1.48221482	32.0237194
240	45	0.0108659	0.92183611	29.9134783	0.00338757	1.00145026	29.9134783
250	45	0.0057641	0.59881707	27.5727341	0.00164748	0.65812848	27.5727341
260	45	0.00297967	0.37879457	25.0794031	0.00078103	0.42105395	25.0794031
270	45	0.00150359	0.23353866	22.5146486	0.00036164	0.26285218	22.5146486
280	45	0.00074206	0.14063647	19.9466476	0.0001639	0.16019782	19.9466476
290	45	0.00035898	0.08280785	17.4435771	7.2781E-05	0.09546855	17.4435771
300	45	0.00017031	0.04770006	15.060628	3.172E-05	0.05570139	15.060628
310	45	7.935E-05	0.02692624	12.8400051	1.359E-05	0.03183426	12.8400051
320	45	3.6392E-05	0.01490199	10.8076806	5.7267E-06	0.01784475	10.8076806
330	45	1.6427E-05	0.00809601	8.9863803	2.3777E-06	0.0098149	8.9863803
340	45	7.3056E-06	0.00432175	7.38259711	9.7289E-07	0.00530495	7.38259711
350	45	3.2055E-06	0.00226661	5.98983802	3.9278E-07	0.002819	5.98983802
360	45	1.3889E-06	0.00116992	4.80485652	1.5663E-07	0.00147366	4.80485652
370	45	5.9424E-07	0.00059471	3.80817344	6.1711E-08	0.0007585	3.80817344
380	45	2.514E-07	0.00029773	2.98323152	2.4046E-08	0.00038473	2.98323152
390	45	1.052E-07	0.00014697	2.31055161	9.2686E-09	0.00019236	2.31055161
400	45	4.3584E-08	7.1584E-05	1.76935595	3.5366E-09	9.4898E-05	1.76935595
0	55	0.00014151	0.16480568	1.98936855	0.00017373	0.0849988	1.98936855
10	55	0.0001926	0.17831328	2.58958771	0.00031252	0.12064206	2.58958771
20	55	0.00035831	0.22219683	3.26605393	0.00062902	0.17977822	3.26605393
30	55	0.00073579	0.30343633	4.03553154	0.00126069	0.26993662	4.03553154
40	55	0.00149295	0.42875328	4.91292218	0.00241696	0.39865739	4.91292218
50	55	0.00288056	0.60254248	5.91126479	0.00437066	0.5710032	5.91126479
60	55	0.00520714	0.82467468	7.03428477	0.00738474	0.7880512	7.03428477
70	55	0.00873941	1.08901006	8.27639401	0.01156268	1.043877	8.27639401
80	55	0.0135034	1.37790472	9.62827899	0.01663204	1.32383122	9.62827899
90	55	0.01906476	1.66679938	11.0657245	0.02184379	1.60712464	11.0657245
100	55	0.02492583	1.94082921	12.5719663	0.02602078	1.86348903	12.5719663
110	55	0.0286333	2.12696268	14.0669081	0.0280808	2.06599535	14.0669081
120	55	0.03051783	2.24588128	15.5505497	0.02750172	2.18771457	15.5505497
130	55	0.02970098	2.2620387	16.9539713	0.02456833	2.21572077	16.9539713
140	55	0.02661303	2.17802012	18.2250169	0.02017299	2.14678245	18.2250169
150	55	0.02212282	2.00739778	19.3189818	0.01535046	1.99274838	19.3189818
160	55	0.01717709	1.77343834	20.1688085	0.01090765	1.7751618	20.1688085
170	55	0.01254757	1.50393259	20.7651836	0.00728316	1.51879742	20.7651836
180	55	0.00867533	1.22667127	21.08948	0.00460324	1.24950709	21.08948
190	55	0.00570799	0.96362849	21.0858165	0.00276916	0.99120382	21.0858165
200	55	0.00359014	0.73031535	20.7877837	0.00159295	0.75886013	20.7877837
210	55	0.00216828	0.53487521	20.2103453	0.00088021	0.56173961	20.2103453
220	55	0.00126193	0.37924695	19.4093823	0.00046896	0.40275061	19.4093823
230	55	0.00071032	0.26078075	18.3737185	0.0002417	0.27995422	18.3737185
240	55	0.00038763	0.17411235	17.1629605	0.00012085	0.18914952	17.1629605
250	55	0.00020563	0.11310194	15.8199505	5.8772E-05	0.12430441	15.8199505
260	55	0.0001063	0.07154505	14.3893933	2.7862E-05	0.07952682	14.3893933
270	55	5.3639E-05	0.04410975	12.9178567	1.2901E-05	0.04964636	12.9178567
280	55	2.6472E-05	0.0265628	11.4444573	5.8468E-06	0.03025746	11.4444573
290	55	1.2806E-05	0.01564038	10.008312	2.5964E-06	0.01803168	10.008312
300	55	6.0756E-06	0.00900938	8.64108681	1.1316E-06	0.01052063	8.64108681
310	55	2.8307E-06	0.00508571	7.36699684	4.8482E-07	0.00601271	7.36699684
320	55	1.2982E-06	0.00281462	6.20094374	2.0429E-07	0.00337044	6.20094374
330	55	5.86E-07	0.00152914	5.15596644	8.4821E-08	0.00185379	5.15596644
340	55	2.6062E-07	0.00081627	4.23579034	3.4707E-08	0.00100198	4.23579034
350	55	1.1435E-07	0.00042811	3.43669006	1.4012E-08	0.00053244	3.43669006
360	55	4.9548E-08	0.00022097	2.75680286	5.5877E-09	0.00027834	2.75680286
370	55	2.1199E-08	0.00011233	2.18495254	2.2015E-09	0.00014326	2.18495254
380	55	8.9683E-09	5.6234E-05	1.71163929	8.578E-10	7.2665E-05	1.71163929
390	55	3.7531E-09	2.7758E-05	1.32568689	3.3065E-10	3.6333E-05	1.32568689
400	55	1.5548E-09	1.3521E-05	1.01517403	1.2616E-10	1.7924E-05	1.01517403
0	65	2.5919E-06	0.02230402	1.02137587	3.1819E-06	0.01150334	1.02137587
10	65	3.5277E-06	0.02413208	1.32953866	5.7239E-06	0.01632713	1.32953866
20	65	6.5627E-06	0.03007107	1.676848	1.1521E-05	0.02433034	1.676848
30	65	1.3476E-05	0.04106564	2.07191098	2.309E-05	0.03653195	2.07191098
40	65	2.7344E-05	0.05802545	2.52237835	4.4268E-05	0.05395241	2.52237835
50	65	5.2759E-05	0.08154526	3.03494454	8.0051E-05	0.07727688	3.03494454
60	65	9.5372E-05	0.11160758	3.61152222	0.00013526	0.10665113	3.61152222
70	65	0.00016007	0.14738149	4.24924237	0.00021178	0.14127339	4.24924237
80	65	0.00024732	0.18647913	4.94332326	0.00030463	0.17916107	4.94332326
90	65	0.00034918	0.22557677	5.68133242	0.00040008	0.21750067	5.68133242
100	65	0.00045653	0.26266267	6.45466274	0.00047659	0.25219582	6.45466274
110	65	0.00052444	0.2878531	7.22219141	0.00051432	0.27960207	7.22219141
120	65	0.00055895	0.30394698	7.98391844	0.00050371	0.29607497	7.98391844
130	65	0.00054399	0.30613365	8.70445908	0.00044998	0.2998652	8.70445908
140	65	0.00048743	0.29476297	9.35703568	0.00036948	0.29053541	9.35703568
150	65	0.00040519	0.27167175	9.91869598	0.00028115	0.26968917	9.91869598
160	65	0.00031461	0.24000878	10.3550116	0.00019978	0.24024203	10.3550116
170	65	0.00022982	0.20353514	10.6612008	0.0001334	0.20554688	10.6612008
180	65	0.00015889	0.1660119	10.8277001	8.4311E-05	0.1691024	10.8277001

190	65	0.00010455	0.13041293	10.8258192	5.0719E-05	0.13414485	10.8258192
200	65	6.5756E-05	0.09883744	10.672804	2.9176E-05	0.10270055	10.672804
210	65	3.9713E-05	0.07238749	10.3763373	1.6122E-05	0.07602319	10.3763373
220	65	2.3113E-05	0.05132549	9.96510915	8.5894E-06	0.05450637	9.96510915
230	65	1.301E-05	0.03529284	9.43338164	4.4268E-06	0.03788768	9.43338164
240	65	7.0997E-06	0.02356354	8.81175775	2.2134E-06	0.0255986	8.81175775
250	65	3.7662E-06	0.01530668	8.1222334	1.0765E-06	0.01682277	8.1222334
260	65	1.9469E-06	0.00968257	7.38776086	5.1032E-07	0.01076278	7.38776086
270	65	9.8244E-07	0.00596961	6.63224875	2.3629E-07	0.0067189	6.63224875
280	65	4.8485E-07	0.00359488	5.87578029	1.0709E-07	0.0040949	5.87578029
290	65	2.3456E-07	0.0021167	5.13843872	4.7554E-08	0.00244032	5.13843872
300	65	1.1128E-07	0.00121929	4.4364819	2.0726E-08	0.00142381	4.4364819
310	65	5.1847E-08	0.00068828	3.78234229	8.8797E-09	0.00081373	3.78234229
320	65	2.3778E-08	0.00038092	3.18367067	3.7418E-09	0.00045614	3.18367067
330	65	1.0733E-08	0.00020695	2.64716143	1.5536E-09	0.00025088	2.64716143
340	65	4.7734E-09	0.00011047	2.17472728	6.3568E-10	0.0001356	2.17472728
350	65	2.0944E-09	5.7938E-05	1.76445551	2.5664E-10	7.2058E-05	1.76445551
360	65	9.075E-10	2.9905E-05	1.41538978	1.0234E-10	3.7669E-05	1.41538978
370	65	3.8827E-10	1.5202E-05	1.12179204	4.0321E-11	1.9388E-05	1.12179204
380	65	1.6426E-10	7.6105E-06	0.87878491	1.5711E-11	9.8342E-06	0.87878491
390	65	6.874E-11	3.7567E-06	0.68063034	6.056E-12	4.9171E-06	0.68063034
400	65	2.8477E-11	1.8298E-06	0.52120772	2.3108E-12	2.4257E-06	0.52120772
0	75	2.4373E-08	0.00216286	0.46924645	2.9921E-08	0.0011155	0.46924645
10	75	3.3173E-08	0.00234014	0.6108244	5.3825E-08	0.00158327	0.6108244
20	75	6.1713E-08	0.00291605	0.77038727	1.0834E-07	0.00235936	0.77038727
30	75	1.2673E-07	0.00398222	0.95188941	2.1713E-07	0.00354257	0.95188941
40	75	2.5713E-07	0.00562684	1.15884575	4.1628E-07	0.00523187	1.15884575
50	75	4.9613E-07	0.0079076	1.39433189	7.5277E-07	0.00749369	1.39433189
60	75	8.9684E-07	0.01082281	1.65922657	1.2719E-06	0.01034217	1.65922657
70	75	1.5052E-06	0.01429187	1.95221168	1.9915E-06	0.01369956	1.95221168
80	75	2.3257E-06	0.01808325	2.27109036	2.8646E-06	0.0173736	2.27109036
90	75	3.2836E-06	0.02187462	2.61015083	3.7622E-06	0.02109147	2.61015083
100	75	4.293E-06	0.02547092	2.96543875	4.4816E-06	0.02445592	2.96543875
110	75	4.9316E-06	0.02791368	3.31806124	4.8364E-06	0.02711356	3.31806124
120	75	5.2562E-06	0.02947434	3.66801831	4.7367E-06	0.02871097	3.66801831
130	75	5.1155E-06	0.02968638	3.99905329	4.2315E-06	0.02907852	3.99905329
140	75	4.5836E-06	0.02858375	4.29886383	3.4744E-06	0.02817379	4.29886383
150	75	3.8103E-06	0.02634454	4.55690507	2.6438E-06	0.02615229	4.55690507
160	75	2.9584E-06	0.02327412	4.75735973	1.8786E-06	0.02329674	4.75735973
170	75	2.1611E-06	0.0197372	4.89803094	1.2544E-06	0.01993229	4.89803094
180	75	1.4942E-06	0.0160985	4.97452502	7.9283E-07	0.01639819	4.97452502
190	75	9.831E-07	0.0126464	4.97366089	4.7694E-07	0.01300829	4.97366089
200	75	6.1834E-07	0.00958446	4.90336179	2.7436E-07	0.00995907	4.90336179
210	75	3.7345E-07	0.00701956	4.7671573	1.516E-07	0.00737212	4.7671573
220	75	2.1734E-07	0.00497713	4.57822848	8.077E-08	0.00528559	4.57822848
230	75	1.2234E-07	0.00342242	4.33393913	4.1628E-08	0.00367404	4.33393913
240	75	6.6762E-08	0.002285	4.04834906	2.0814E-08	0.00248235	4.04834906
250	75	3.5416E-08	0.00148432	3.73156377	1.0122E-08	0.00163134	3.73156377
260	75	1.8308E-08	0.00093894	3.39412812	4.7988E-09	0.00104369	3.39412812
270	75	9.2384E-09	0.00057888	3.04702635	2.222E-09	0.00065155	3.04702635
280	75	4.5593E-09	0.0003486	2.6994852	1.007E-09	0.00039709	2.6994852
290	75	2.2057E-09	0.00020526	2.36073144	4.4718E-10	0.00023664	2.36073144
300	75	1.0464E-09	0.00011824	2.03823435	1.949E-10	0.00013807	2.03823435
310	75	4.8754E-10	6.6743E-05	1.73770572	8.3501E-11	7.8909E-05	1.73770572
320	75	2.236E-10	3.6938E-05	1.46266052	3.5186E-11	4.4233E-05	1.46266052
330	75	1.0093E-10	2.0068E-05	1.21617432	1.4609E-11	2.4329E-05	1.21617432
340	75	4.4887E-11	1.0713E-05	0.99912587	5.9777E-12	1.315E-05	0.99912587
350	75	1.9695E-11	5.6184E-06	0.81063643	2.4133E-12	6.9876E-06	0.81063643
360	75	8.5337E-12	2.8999E-06	0.65026662	9.6238E-13	3.6528E-06	0.65026662
370	75	3.6511E-12	1.4741E-06	0.51538023	3.7916E-13	1.8801E-06	0.51538023
380	75	1.5446E-12	7.38E-07	0.40373648	1.4774E-13	9.5364E-07	0.40373648
390	75	6.464E-13	3.6429E-07	0.31269916	5.6948E-14	4.7682E-07	0.31269916
400	75	2.6779E-13	1.7744E-07	0.23945629	2.173E-14	2.3523E-07	0.23945629
0	85	1.1767E-10	0.00015028	0.19291298	1.4446E-10	7.7509E-05	0.19291298
10	85	1.6016E-10	0.0001626	0.25111742	2.5987E-10	0.00011001	0.25111742
20	85	2.9795E-10	0.00020262	0.31671568	5.2305E-10	0.00016394	0.31671568
30	85	6.1183E-10	0.0002767	0.39133343	1.0483E-09	0.00024615	0.39133343
40	85	1.2414E-09	0.00039097	0.47641573	2.0098E-09	0.00036353	0.47641573
50	85	2.3953E-09	0.00054945	0.57322698	3.6343E-09	0.00052069	0.57322698
60	85	4.3299E-09	0.00075201	0.68212844	6.1406E-09	0.00071861	0.68212844
70	85	7.2671E-09	0.00099305	0.80257821	9.6147E-09	0.00095189	0.80257821
80	85	1.1228E-08	0.00125649	0.93367316	1.383E-08	0.00120718	0.93367316
90	85	1.5853E-08	0.00151992	1.07306509	1.8164E-08	0.00146551	1.07306509
100	85	2.0727E-08	0.00176981	1.21912832	2.1637E-08	0.00169928	1.21912832
110	85	2.3809E-08	0.00193954	1.36409576	2.335E-08	0.00188394	1.36409576
120	85	2.5376E-08	0.00204798	1.50796741	2.2868E-08	0.00199494	1.50796741
130	85	2.4697E-08	0.00206271	1.64405996	2.0429E-08	0.00202048	1.64405996
140	85	2.213E-08	0.0019861	1.76731575	1.6774E-08	0.00195761	1.76731575
150	85	1.8396E-08	0.00183051	1.87339968	1.2764E-08	0.00181715	1.87339968
160	85	1.4283E-08	0.00161717	1.95580905	9.07E-09	0.00161874	1.95580905
170	85	1.0434E-08	0.00137141	2.01364072	6.0562E-09	0.00138496	2.01364072
180	85	7.2138E-09	0.00111858	2.04508838	3.8277E-09	0.0011394	2.04508838
190	85	4.7464E-09	0.00087872	2.04473312	2.3026E-09	0.00090386	2.04473312
200	85	2.9853E-09	0.00066596	2.0158323	1.3246E-09	0.00069199	2.0158323
210	85	1.803E-09	0.00048774	1.95983696	7.3192E-10	0.00051224	1.95983696

220	85	1.0493E-09	0.00034583	1.882166	3.8996E-10	0.00036726	1.882166
230	85	5.9065E-10	0.0002378	1.78173564	2.0098E-10	0.00025529	1.78173564
240	85	3.2233E-10	0.00015877	1.66432606	1.0049E-10	0.00017248	1.66432606
250	85	1.7099E-10	0.00010314	1.53409173	4.8871E-11	0.00011335	1.53409173
260	85	8.8389E-11	6.5241E-05	1.39536779	2.3168E-11	7.2519E-05	1.39536779
270	85	4.4603E-11	4.0223E-05	1.25266998	1.0728E-11	4.5272E-05	1.25266998
280	85	2.2012E-11	2.4222E-05	1.10979154	4.8618E-12	2.7591E-05	1.10979154
290	85	1.0649E-11	1.4262E-05	0.97052571	2.159E-12	1.6443E-05	0.97052571
300	85	5.052E-12	8.2155E-06	0.83794319	9.4095E-13	9.5936E-06	0.83794319
310	85	2.3538E-12	4.6376E-06	0.71439218	4.0314E-13	5.4829E-06	0.71439218
320	85	1.0795E-12	2.5666E-06	0.60131772	1.6988E-13	3.0734E-06	0.60131772
330	85	4.8728E-13	1.3944E-06	0.49998421	7.0531E-14	1.6904E-06	0.49998421
340	85	2.1671E-13	7.4434E-07	0.41075293	2.886E-14	9.1368E-07	0.41075293
350	85	9.5086E-14	3.9038E-07	0.3332626	1.1651E-14	4.8552E-07	0.3332626
360	85	4.12E-14	2.015E-07	0.2673326	4.6463E-15	2.5381E-07	0.2673326
370	85	1.7627E-14	1.0243E-07	0.21187915	1.8306E-15	1.3064E-07	0.21187915
380	85	7.4574E-15	5.1279E-08	0.16598103	7.1329E-16	6.6262E-08	0.16598103
390	85	3.1208E-15	2.5312E-08	0.12855447	2.7494E-16	3.3131E-08	0.12855447
400	85	1.2929E-15	1.2329E-08	0.09844342	1.0491E-16	1.6345E-08	0.09844342
0	95	2.9168E-13	7.4822E-06	0.07096872	3.5807E-13	3.8589E-06	0.07096872
10	95	3.9699E-13	8.0954E-06	0.09238093	6.4414E-13	5.4771E-06	0.09238093
20	95	7.3853E-13	1.0088E-05	0.11651319	1.2965E-12	8.1619E-06	0.11651319
30	95	1.5166E-12	1.3776E-05	0.14396353	2.5985E-12	1.2255E-05	0.14396353
40	95	3.0772E-12	1.9465E-05	0.17526355	4.9817E-12	1.8099E-05	0.17526355
50	95	5.9373E-12	2.7355E-05	0.21087842	9.0086E-12	2.5924E-05	0.21087842
60	95	1.0733E-11	3.744E-05	0.25094103	1.5221E-11	3.5777E-05	0.25094103
70	95	1.8013E-11	4.9441E-05	0.29525202	2.3832E-11	4.7392E-05	0.29525202
80	95	2.7833E-11	6.2557E-05	0.34347916	3.4281E-11	6.0102E-05	0.34347916
90	95	3.9295E-11	7.5673E-05	0.39475858	4.5023E-11	7.2963E-05	0.39475858
100	95	5.1376E-11	8.8114E-05	0.44849224	5.3633E-11	8.4602E-05	0.44849224
110	95	5.9018E-11	9.6564E-05	0.50182278	5.7879E-11	9.3796E-05	0.50182278
120	95	6.2902E-11	0.00010196	0.55475021	5.6685E-11	9.9322E-05	0.55475021
130	95	6.1218E-11	0.0001027	0.60481586	5.0639E-11	0.00010059	0.60481586
140	95	5.4854E-11	9.8882E-05	0.65015913	4.158E-11	9.7464E-05	0.65015913
150	95	4.5599E-11	9.1136E-05	0.68918523	3.164E-11	9.0471E-05	0.68918523
160	95	3.5405E-11	8.0514E-05	0.71950194	2.2482E-11	8.0592E-05	0.71950194
170	95	2.5862E-11	6.8278E-05	0.74077702	1.5012E-11	6.8953E-05	0.74077702
180	95	1.7881E-11	5.5691E-05	0.75234597	9.488E-12	5.6728E-05	0.75234597
190	95	1.1765E-11	4.3749E-05	0.75221528	5.7077E-12	4.5001E-05	0.75221528
200	95	7.3998E-12	3.3156E-05	0.74158326	3.2833E-12	3.4452E-05	0.74158326
210	95	4.4692E-12	2.4283E-05	0.72098372	1.8142E-12	2.5503E-05	0.72098372
220	95	2.601E-12	1.7218E-05	0.69241018	9.666E-13	1.8285E-05	0.69241018
230	95	1.4641E-12	1.1839E-05	0.65546391	4.9817E-13	1.271E-05	0.65546391
240	95	7.9897E-13	7.9047E-06	0.61227134	2.4909E-13	8.5874E-06	0.61227134
250	95	4.2383E-13	5.1348E-06	0.56436081	1.2114E-13	5.6434E-06	0.56436081
260	95	2.1909E-13	3.2481E-06	0.51332712	5.7429E-14	3.6105E-06	0.51332712
270	95	1.1056E-13	2.0026E-06	0.46083153	2.6591E-14	2.2539E-06	0.46083153
280	95	5.4563E-14	1.2059E-06	0.40826949	1.2051E-14	1.3737E-06	0.40826949
290	95	2.6396E-14	7.1007E-07	0.35703646	5.3515E-15	8.1864E-07	0.35703646
300	95	1.2523E-14	4.0903E-07	0.30826207	2.3324E-15	4.7764E-07	0.30826207
310	95	5.8346E-15	2.3089E-07	0.2628102	9.9928E-16	2.7298E-07	0.2628102
320	95	2.6759E-15	1.2778E-07	0.22121243	4.2108E-16	1.5302E-07	0.22121243
330	95	1.2078E-15	6.9423E-08	0.18393391	1.7483E-16	8.4162E-08	0.18393391
340	95	5.3718E-16	3.7059E-08	0.15110756	7.1537E-17	4.549E-08	0.15110756
350	95	2.357E-16	1.9436E-08	0.12260046	2.8881E-17	2.4173E-08	0.12260046
360	95	1.0213E-16	1.0032E-08	0.09834617	1.1517E-17	1.2637E-08	0.09834617
370	95	4.3694E-17	5.0996E-09	0.07794598	4.5376E-18	6.5041E-09	0.07794598
380	95	1.8485E-17	2.553E-09	0.06106101	1.7681E-18	3.299E-09	0.06106101
390	95	7.7356E-18	1.2602E-09	0.04729255	6.8151E-19	1.6495E-09	0.04729255
400	95	3.2047E-18	6.1383E-10	0.03621531	2.6004E-19	8.1375E-10	0.03621531
0	105	3.712E-16	2.6692E-07	0.02336241	4.557E-16	1.3766E-07	0.02336241
10	105	5.0522E-16	2.888E-07	0.03041116	8.1976E-16	1.9539E-07	0.03041116
20	105	9.3988E-16	3.5987E-07	0.03835532	1.65E-15	2.9117E-07	0.03835532
30	105	1.9301E-15	4.9144E-07	0.04739178	3.3069E-15	4.3719E-07	0.04739178
40	105	3.9161E-15	6.9441E-07	0.05769553	6.3399E-15	6.4566E-07	0.05769553
50	105	7.556E-15	9.7588E-07	0.0694197	1.1465E-14	9.2479E-07	0.0694197
60	105	1.3659E-14	1.3356E-06	0.08260803	1.9371E-14	1.2763E-06	0.08260803
70	105	2.2924E-14	1.7638E-06	0.0971949	3.033E-14	1.6907E-06	0.0971949
80	105	3.5421E-14	2.2317E-06	0.11307093	4.3627E-14	2.1441E-06	0.11307093
90	105	5.0009E-14	2.6995E-06	0.12995176	5.7298E-14	2.6029E-06	0.12995176
100	105	6.5383E-14	3.1434E-06	0.1476405	6.8255E-14	3.0181E-06	0.1476405
110	105	7.5108E-14	3.4448E-06	0.16519654	7.3659E-14	3.3461E-06	0.16519654
120	105	8.0051E-14	3.6374E-06	0.18261988	7.214E-14	3.5432E-06	0.18261988
130	105	7.7908E-14	3.6636E-06	0.19910114	6.4445E-14	3.5886E-06	0.19910114
140	105	6.9808E-14	3.5275E-06	0.21402783	5.2916E-14	3.4769E-06	0.21402783
150	105	5.803E-14	3.2512E-06	0.22687494	4.0266E-14	3.2274E-06	0.22687494
160	105	4.5057E-14	2.8723E-06	0.23685499	2.8612E-14	2.875E-06	0.23685499
170	105	3.2913E-14	2.4358E-06	0.2438586	1.9104E-14	2.4598E-06	0.2438586
180	105	2.2756E-14	1.9867E-06	0.24766702	1.2075E-14	2.0237E-06	0.24766702
190	105	1.4973E-14	1.5607E-06	0.24762399	7.2638E-15	1.6054E-06	0.24762399
200	105	9.4173E-15	1.1828E-06	0.24412401	4.1785E-15	1.229E-06	0.24412401
210	105	5.6876E-15	8.6628E-07	0.23734279	2.3089E-15	9.0979E-07	0.23734279
220	105	3.3102E-15	6.1423E-07	0.22793657	1.2301E-15	6.5229E-07	0.22793657
230	105	1.8632E-15	4.2236E-07	0.21577412	6.3399E-16	4.5341E-07	0.21577412
240	105	1.0168E-15	2.8199E-07	0.20155543	3.17E-16	3.0635E-07	0.20155543

250	105	5.3938E-16	1.8318E-07	0.18578362	1.5417E-16	2.0132E-07	0.18578362
260	105	2.7883E-16	1.1587E-07	0.16898369	7.3086E-17	1.288E-07	0.16898369
270	105	1.407E-16	7.144E-08	0.15170251	3.3841E-17	8.0407E-08	0.15170251
280	105	6.9439E-17	4.3021E-08	0.13439945	1.5337E-17	4.9005E-08	0.13439945
290	105	3.3592E-17	2.5331E-08	0.1175339	6.8106E-18	2.9204E-08	0.1175339
300	105	1.5937E-17	1.4592E-08	0.10147771	2.9683E-18	1.7039E-08	0.10147771
310	105	7.4253E-18	8.2368E-09	0.08651527	1.2717E-18	9.7382E-09	0.08651527
320	105	3.4054E-18	4.5586E-09	0.07282158	5.3588E-19	5.4588E-09	0.07282158
330	105	1.5371E-18	2.4766E-09	0.06054975	2.2249E-19	3.0024E-09	0.06054975
340	105	6.8363E-19	1.322E-09	0.04974355	9.104E-20	1.6228E-09	0.04974355
350	105	2.9995E-19	6.9336E-10	0.04035921	3.6755E-20	8.6234E-10	0.04035921
360	105	1.2997E-19	3.5788E-10	0.03237487	1.4657E-20	4.508E-10	0.03237487
370	105	5.5606E-20	1.8192E-10	0.02565927	5.7747E-21	2.3203E-10	0.02565927
380	105	2.3525E-20	9.1077E-11	0.02010086	2.2501E-21	1.1769E-10	0.02010086
390	105	9.8446E-21	4.4957E-11	0.01556837	8.6732E-22	5.8844E-11	0.01556837
400	105	4.0784E-21	2.1898E-11	0.01192183	3.3094E-22	2.903E-11	0.01192183

F. Software utilizado y nota de copyright de MT3D.

Software utilizado

Programación de PROW: Compaq Visual Fortran Professional Edition 6.1.0

Postproceso resultados: Golden Software Surfer Version 7.00

Nota de copyright de MT3D

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
C                         MT3DMS
C     a modular three-dimensional multi-species transport model
C    for simulation of advection, dispersion and chemical reactions
C              of contaminants in groundwater systems
C
C               For Technical Information Contact
C                       Chunmiao Zheng
C               Department of Geological Sciences
C                     University of Alabama
C                     Tuscaloosa, AL 35487
C                     E-mail: czheng@ua.edu
C           World-Wide-Web: http://hydro.geo.ua.edu/mt3d
C
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
C MT3DMS is based on MT3D originally developed by Chunmiao Zheng
C at S.S. Papadopulos & Associates, Inc. and documented for
C the United States Environmental Protection Agency.
C MT3DMS is written by Chunmiao Zheng and P. Patrick Wang
C with the iterative solver routine by Tsun-Zee Mai.
C Funding for MT3DMS development is provided, in part, by
C U.S. Army Corps of Engineers Waterways Experiment Station.
C
C Copyright, 1998-2001, The University of Alabama. All rights reserved.
C
C This program is provided without any warranty.
C No author or distributor accepts any responsibility
C to anyone for the consequences of using it
C or for whether it serves any particular purpose.
C The program may be copied, modified and redistributed,
C but ONLY under the condition that the above copyright notice
C and this notice remain intact.
```