UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO
UACPyP IIMAS

Simulación Computacional de una Válvula Coronaria

T e s i s

Que para obtener el grado de

Maestro en Ciencias (Computación)

Presenta

Raymundo Eugenio Peralta Herrera

México, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Indice

Introducción.

La motivación para desarrollar como tema de tesis una simulación de una válvula coronaria, tuvo su origen en largas discuciones con Antonmaria Minzoni acerca de lo que significa estar interesado en un problema y en su solución, y cuando Benito Chen planteó la posibilidad de resolver la ecuación de Navier-Stokes decidí estudiar ese problema. Estas dos situaciones ofrecian la posibilidad de desarrollar una tesis de computación que resolvería un problema en el cual había un interes científico.

Parece ser que para desarrollar un programa no trivial se requiere tener el deseo de resolver un problema no trivial, ejemplos de esta situación son: los compiladores de los lenguajes de programación para las diferentes máquinas; los códigos que se escriben para calcular aviones; las aplicaciones de ingenieria; los sistemas operativos; TEX; **LINPACK**; **AUTO**; **IMSL** etc. Estos códigos siempre tienen como objetivo resolver un problema existente.

Si un programa resuelve algún problema de interés para alguna rama de la ciencia o técnica, tiene posibilidades de crecer, porqué tendrá usuarios que retroalimentarán al diseñador permitiendo que se mejore el software. Por lo general, este tipo de programas son complicados y muchas veces no se sabe como codificarlos, es ahi, donde aparecen los problemas interesantes que tienen alguna oportunidad de desarrollarse, tal vez no para llegar a ser muy importantes o famosos, sino únicamente, para ser útiles y para ayudar a aclarar incognitas.

De entre los programas antes mencionados, algunos le muestran al usuario una interface versátil y poderosa. Hacen uso de las técnicas de compiladores para lograr su poder de comunicación, y algunas veces la interface resulta a primera vista complicada, pero es mejor soportar este pequeño mal y tener versatilidad a no tener versatilidad y gozar de una interface sencilla.

Cuando se trabaja con métodos numéricos para resolver ecuaciones parciales en dominios complicados, generalmente, se trata de usar un sistema interactivo para comunicarle a la máquina la forma geométrica del dominio[15], pero,

¿Porque no usar un lenguaje para definir una forma geométrica?. Un lenguaje tendría la facilidad de verificar que las propiedades importantes de la malla[1] se cumplieran. De tal suerte, que solo se permitiría continuar con el proceso si la malla es adecuada para discretizar la ecuación diferencial. Pequeños cambios en la forma del dominio implicarían compilar la malla cada vez, y se garantizaría que se obtendra una solución correcta al problema una vez que el resto del sistema ha sido puesto a punto.

Este método es el opuesto a lo que ocurre generalmente con los programas de análisis numérico. Estos programas se desarrollan generalmente para resolver una ecuación diferencial en un dominio, y nada más.

Tratando de seguir este camino de ideas, se plantea el problema de desarrollar un código que resuelva ecuaciones diferenciales no-lineales en derivadas parciales, usando la técnica de diferencias finitas. Y que además se le puedan definir diferentes dominios y diferentes ecuaciones, con facilidad.

El sistema propuesto, a primera vista, podría parecer un sistema muy complejo de programar, pero si se diseña de una forma modular y se separan adecuadamente la parte que trabaja con el dominio de la parte que hace el analisis numérico y de la parte que define a la ecuación parcial, el trabajo se convierte en una tarea realizable por una sola persona. Si bien el análisis de una EDP específica puede requerir ayuda de parte de un especialista del area para entender las sutilezas del método particular de solución, la programación requiere únicamente una buena compresión del método numérico (diferencias finitas) y una buena organización de la información.

Para desarrollar un sistema con las habilidades planteadas, se usó la siguiente metodología:

- se planteó un método numérico adecuado para resolver las ecuaciones diferenciales propuestas[2].
- se diseñó un lenguaje para describir los dominios sobre los que se resuelve la ecuación diferencial parcial.
- se programó el compilador del lenguaje de descripción de mallas.
- se desarrollaron los programas necesarios para crear y resolver los sistemas de ecuaciones algebraicas asociados con las discretizaciones de las ecuaciones diferenciales.

[1] El dominio de la ecuación diferencial se discretiza haciendo uso de una malla.

[2] Tal vez un buen método de realizar un programa como este, consista en trabajar desde un principio con una ecuación diferencial que ofrezca una gama grande de dificultades, para que la definición del sistema sea amplia y puedea ser aplicada a otros problemas sin dificultad, si el probelma inicial es muy simple, el sistema resultante será por fuerza muy simple y no se podrá utilizar para otros problemas.

- se programaron algunos ejemplos[3] para probar el sistema[4].
- se describió el método para discretizar una ecuación diferencial y resolverla.

El sistema desarrollado de esta forma, brindará la posibilidad de resolver muchas ecuaciones diferenciales diferentes en diferentes dominios numéricamente de una forma sencilla —por construcción—.

Las habilidades antes mencionadas, se implantaron en el *software* que se desarrolló como parte de este trabajo, demostraron ser útiles y versátiles. Se pudo resolver la ecuación de Navier-Stokes en diferentes dominios[5] lo cual da confianza acerca de que los programas estan trabajando correctamente, no se presume que este sistema sea capaz de resolver cualquier ecuación diferencial en derivadas parciales, únicamente puede trabajar con ecuaciones elípticas que no tengan derivadas cruzadas[6] y que no tengan discontinuidades dentro del dominio[7], las discontinuidades en la frontera del dominio pueden ser manejadas adecuadamente, asi como las condiciones de frontera de diferentes tipos.

Será necesario ver si alguien en el futuro se interesa en resolver algún problema haciendo uso de este programa, para probar las ideas que se mencionan al principio[8].

Esta tesis se presenta en seis capítulos, cuyos contenidos son los siguientes:

- **CAPITULO I.** Se presenta el problema de modelar fluidos biológicos (en particular, modelar una válvula coronaria abierta) y las suposiciones que se hacen para poder simular la situación real con un modelo físico. Al final se plantean los objetivos del trabajo.
- **CAPITULO II.** Se describe a *grosso modo* el sistema que se implantó, para proporcionar un panorama del funcionamiento global.
- **CAPITULO III.** Se presentan las técnicas numéricas propuestas para atacar el problema matemático.

---

[3]Para el desarrollo y prueba de los programas que resuleven la ecuación de Navier-Stokes debo agradecer de forma muy especial a A.A. Minzoni y B. Chen por su apoyo y ayuda, sin la cual no hubiera sido posible presentar soluciones a la ecuación de Navier-Stokes.

[4]No se presentan los problemas iniciales que se usaron para depurar el sistema, porqué son en extremo sencillos, y se prefiere presentar las soluciones de la ecuación de Navier-Stokes, que son mucho más interesantes.

[5]Durante el desarrollo del trabajo, se resolvieron dos ecuaciones diferenciales parciales diferentes: la ecuación de Laplace, la ecuación de transporte.

[6]El problema de las derivadas cruzadas puede ser resuelto dentro del mismo marco de ideas.

[7]Existe una previsión para manejar discontinuidades en el dominio, pero no esta totalmente implantada.

[8]En particular A.A. Minzoni ha expresado su deseo de tratar de resolver una EDP relacionada con magnetohidrodinámica haciendo uso de estos programas.

- **CAPITULO IV.** Se describe el lenguaje diseñado para definir los dominios y la descripción del compilador desarrollado para traducir el lenguaje de descrición de mallas.
- **CAPITULO V.** Se describe el módulo generador de ecuaciones lineales.
- **CAPITULO VI.** Se presenta como aplicación el método usado para resolver el problema de la válvula coronaria abierta y sus resultados, asi como los resultados obtenidos para diferentes dominios en los que se resolvió la ecuación de Navier-Stokes.

Esta tesis, es una mezcla de varias disciplinas, para cada una de ellas, el problema que se desea resolver y los intereses son un poco dispares. La parte de matemáticas, se interesa por el comportamiento de las soluciones, las forma de calcularlas y por que exista la convergencia de la solución numérica a la solución análitica. Para la física, lo importante es el comportamiento del fluido en los diferentes dominios para entender las fuerzas que se dan cita en cada situación. Mientras que para la computación, lo relevante es plantear un método que ayude a resolver el problema que se ha propuesto, que se haga facilmente y que permita resolver otros problemas con pequeñas modificaciones localizadas a los codigos. Por otro lado, el problema técnico desde el punto de vista de computación se reduce a describir mallas, numerarlas y a partir de ellas generar un sistema de ecuaciones. Los puntos de vista y el lenguaje que de cada una de las ramas que estan involucradas, son diferentes, mientras a la parte matemática del problema no le interesa como se puedan programar ni la forma de discretizar el dominio ni el algóritmo de solución de los sistemas de ecuaciones si entregan una solución correcta, a la parte computacional —para hacerla modular— no le interesa el problema matemático que se está resolviendo, únicamente es necesario enfocar el problema de como crear las mallas y como dirigir el proceso de creación de los sistemas de ecuaciones para que después sean resueltos con algún método numérico que es semi-independiente del problema matemático y del dominio del análisis numérico.

El codificar el programa que se presenta como tesis, fue muy interesante, porqué me obligó a entender la física del problema, a aprender algo de las matemáticas asociadas, además de permitirme poner en práctica algunas ideas para resolver este tipo de problemas.

Deseo agradecer a B. Chen por haberme ofrecido un tema de tesis y por haberme brindado su consejo en los momentos necesarios.

Deseo agradecer de una forma muy especial a A.A Minzoni y a B. Chen por haber empleado mucho de su tiempo ayudandome a que este trabajo de tesis tuviera un final feliz. Sin su ayuda no hubiera sido posible obtener soluciones de la ecuación de Navier-Stokes.

Al Arturo Olvera Chavez por sus comentarios que durante la redacción de este trabajo, aclararon la comprensión de la relación entre las distintas partes del trabajo.

Al Departamento de Estadística del IIMAS-UNAM, se le agradece haber permitido que su equipo de computo se empleara para la realización de este trabajo.

Ciudad Universitaria, 7 de julio 1988.

Capítulo I

La válvula coronaria abierta.

El cuerpo humano para su funcionamiento depende de una gran cantidad de fluidos, los cuales pueden llegar a tener comportamientos variados. Cuando el comportamiento de los fluidos vitales es alterado por alguna razón, el cuerpo humano puede sufrir daños irreparables y por lo general simplemente funciona mal. Este aspecto del cuerpo humano es presentado en la siguiente cita de *Thomas J. Muller*:

> Desde hace mucho tiempo, se ha sabido que los fluidos juegan un papel muy importante en los procesos biológicos. Se sabe que la sangre y el aire son de vital importancia, pero tambien existen otros fluidos importantes, como son: la orina, el sudor, las lágrimas y el líquido sinovial de las articulaciones. El aire, la orina, el sudor y las lágrimas presentan un comportamiento Newtoniano, mientras que por el contrario, la sangre y el líquido sinovial presentan un comportamiento no-Newtoniano bajo ciertas condiciones. La sangre entendida como una suspensión, se comporta como un líquido Newtoniano cuando la magnitud de los esfuerzos cortantes es grande, y es no-Newtoniano cuando la magnitud de los esfuerzos cortantes es pequeña. El líquido sinovial es un fluido visco-elástico y al parecer la elasticidad es importante durante el proceso de lubricación de las articulaciones.
>
> Estos fluidos fisiológicos estan sujetos a una gran variedad de movimientos en tres dimensiones, como pueden ser: dilataciones y paso por tubos de menor diámetro. Por ejemplo, la sangre cuando fluye pulsando en un sistema circulario sano, tiene un comportamiento laminar casi en su totalidad, aunque el número de Reynolds maximo es del orden de 10,000. La única excepción al flujo laminar

de la sangre aparece durante unas pequeñas "explosiones" de turbulencia que han sido detectadas en la aorta durante una pequeña fracción de cada ciclo. Como consecuencia de una oclución o de una estenosis en el sistema circulatorio, el flujo normalmente laminar se convierte en un flujo transicional o turbulento. También el flujo de aire en los pulmones que es normalmente laminar, puede llegar a ser turbulento como resultado de una obstrucción o durante la tos, bajo esta situación el número de Reynolds puede alcanzar valores del orden de 50,000.

Pareciera ser que el cuerpo humano ha sido desarrollado como un sistema óptimo. El hecho de que el flujo sea laminar, mantiene la resistencia del flujo al mínimo, asegurando el máximo tiempo de vida de los componentes, i.e. corazón, pulmones y sistema circulatorio. Más aun, la naturaleza pulsante del flujo en el sistema circulatorio elástico, primero forza al corazón y a los vasos sanguíneos y enseguida los relaja. Este ejercicio vascular contribuye a la larga vida del sistema[1].

Es muy interesante estudiar los diferentes regímenes de flujo que presentan los fluidos biológicos, tanto dentro del cuerpo humano (i.e. *in vivo*) como fuera del mismo (i.e. *in vitro*). Dentro del cuerpo, y para el caso del sistema circulatorio, se presentan situaciones donde los fluidos se separan en dos ramas, pasan a través de una válvula, y en ambos casos mantienen su régimen laminar. La comprensión de este fenómeno resulta muy interesante y se cree que es debido a que el sistema circulatorio es elástico. Por otro lado, el entendimiento de los fluidos biológicos *in vitro* tiene importancia en el diseño de equipo médico, porqué permitiría mejorar los diseños de los equipos de diálisis, de los equipos que hacen circular la sangre durante las intervenciones quirúrgicas de corazón abierto, de las válvulas artificiales etc.

Por estas razones, es importante tratar de estudiar los fluidos biológicos. De entre todas las posibilidades, una nos resulta interesante: el flujo en una válvula coronaria totalmente abierta al final de un ciclo de bombeo. Que es una situación que se puede modelar con la ecuación de Navier-Stokes. Su dominio se puede describir adecuadamente para usar diferencias finitas y porqué, se supone que el flujo alcanza un estado estacionario cuando toda la sangre se desplaza en una dirección al final del ciclo de bombeo.

Si bien los fluidos biológicos circulan dentro de tubos elásticos, en este momento no se puede resolver este problema. Para empezar a estudiarlo es necesario comenzar por estudiar estos fluidos en situaciones mucho más simples, como puede ser el flujo con paredes rígidas. Este tipo de modelos son muy útiles para iniciar el entendimiento del comportamiento de los fluidos biológicos que poseen un régimen de flujo muy complejo. Sin embargo no se debe menospreciar a los modelos simples, porqué pueden arrojar luz sobre las propiedades

[1] Tomado del libro [1]

fundamentales del flujo y permitir que se desarrollen modelos cada vez más completos. Si bien, se han desarrollado modelos numéricos que entregan soluciones de este problema, todos comienzan por suponer que trabajan con un fluido Newtoniano, que tienen paredes rígidas, y trabajan por lo general, en un dominio único.

Para simular los fluidos biológicos, se emplea comunmente la ecuación de Navier-Stokes y se supone que el fluido es Newtoniano [1,17]. Los fluidos modelados por esta ecuación son fluidos viscosos, que presentan regiones de separación y capas límite. Estas características poseen la propiedad de que los valores con los que se representan las funciones de interés tienen cambios bruscos o desviaciones muy pequeñas alrededor de un valor.

Las regiones de separación son comunes a todas las situaciones prácticas de flujo viscoso y los flujos biológicos no son la excepción, tanto *in vivo* como *in vitro* se presentan regiones de separación para los fluidos biológicos. *In vivo* las regiones de separación se presentan como consecuencia de malfuncionamientos del sistema circulatorio, causadas por arteroscleroSis (especialmente cerca de las bifurcaciones), ateroma, reducción de la luz de una válvula cardiaca o de los vasos sanguineos y aneurismas. *In vitro* se presentan en válvulas artificiales, y en los aparatos usados para hacer circular los líquidos biológicos fuera del organismo [1].

Las regiones de separación se caracterizan por flujos recirculantes de baja velocidad que atrapan diversos componentes de la sangre produciendo coágulos, y además porqué entre la región de flujo recirculante y la corriente principal se generan esfuerzos cortantes grandes que dañan la pared celular de los glóbulos rojos, lo cual puede producir anemia o intoxicación por hemoglobina libre en el torrente sanguineo. De ahi la importancia que tiene el poder calcular los vórtices generados en estas situaciones. Es necesario aclarar que estas ca- racterísticas de los fluidos no son particulares a los fluidos biológicos, tambien aparecen en muchas otras situaciones.

I.1 Modelo físico.

El problema de los fluidos biológicos es muy complejo, dado que los fluidos reales no se sabe como modelarlos y porqué el comportamiento de los fluidos que si se pueden modelar es en si mismo difícil de estudiar. Tienen un comportamiento no lineal, y solo en los últimos años del desarrollo de la física y de las matemáticas se han permitido iniciar un estudio más cuidadoso de este régimen.

Un fluido Newtoniano viscoso, al circular por un tubo, se pegará a las paredes del mismo y en el centro del tubo alcanzará la maxima velocidad posible. Al hacerlo menos viscoso, la maxima velocidad (es decir, la velocidad en el centro del tubo) crecerá como consecuencia de que el flujo ya no de adhiere tan fuertemente a las capas que estan más cerca de las paredes.

Si en este momento, un obstáculo es introducido en medio del torrente, el

fluido tendrá que rodearlo. Pegandose a las paredes del obstáculo. Si además la velocidad del fluido es alta, se formarán zonas de separación, que se visualizan como vórtices —remolinos— antes y después del obstáculo. En las regiones de separación y dependiendo de la geometría se pueden generar multiples vórtices o zonas con estructura más compleja (estelas). A la fecha no hay soluciones analíticas ni métodos cualitativos para entender estos flujos. Por esto se recurre a métodos numéricos que puedan capturar estas soluciones de gran interés poder empezar a calcular soluciones numéricas que arrojen luz sobre este tema.

Una propiedad importante de estos fluidos es que la velocidad del flujo junto a la pared es cero, ésto forma lo que se llama la capa límite, que es la región en donde la velocidad del líquido cambia muy rápidamente, desde cero en la pared a diferente de cero un poco más adentro del líquido. Esta propiedad la debe satisfacer toda solución numérica que se calcule, y es el hecho físico que se debe entender para poder resolver la ecuación de Navier-Stokes numéricamente, que es el modelo matemático que se emplea para simular el problema físico de los fluidos Newtonianos viscosos incompresibles.

Haciendo uso de la ley de conservación del momento y de las relaciones constitutivas que definen un fluido Newtoniano, se obtiene la ecuación de Navier-Stokes. Para una derivación detallada de la ecuación de Navier-Stokes, se puede consultar el libro [7] primera parte.

En este trabajo, se usara la ecuación de Navier-Stokes expresada en términos de la función de corriente y de la vorticidad. La función de corriente se define como aquella función cuya derivada parcial con respecto a y es la componente horizontal de la velocidad del flujo, y menos su derivada parcial con respecto a x es la componente vertical de la velocidad del flujo. La vorticidad se define como el rotacional del vector velocidad del flujo. En términos de la función corriente, la vorticidad es menos el Laplaciano de ella. Esto se expresa de la siguiente forma.

Si $\bar{v} = (u, v)$ denota al vector velocidad del flujo y sus componentes horizontal y vertical respectivamente, y si ψ denota a la función corriente y ω es la vorticidad, se tiene que:

$$\begin{aligned} u &= \frac{\partial \psi}{\partial y} \\ &= \psi_y \qquad \text{(I.1)} \\ v &= -\frac{\partial \psi}{\partial x} \\ &= -\psi_x \qquad \text{(I.2)} \\ \omega &= \nabla \times \bar{v} \\ &= -\Delta \psi. \qquad \text{(I.3)} \end{aligned}$$

Ahora la ecuación de Navier-Stokes, se expresa de la forma siguiente, siendo ν el coeficiente de viscocidad:

$$\Delta \psi = -\omega \qquad \text{(I.4)}$$

$$(\psi_y, -\psi_x) \cdot \nabla\omega \;=\; \nu\Delta\omega \qquad \text{(I.5)}$$

a la ecuación I.4 se le llama ecuación de conservación de masa, y a la ecuación I.5 que es la ley de conservación del momento se le llama ecuación de Navier-Stokes. En el resto de esta tesis, se hará referencia a la ecuación I.4 como la ecuación de conservación de masa, a la ecuación I.5 se le llamara ecuación de la vorticidad y al sistema se le llamara ecuación de Navier-Stokes, esta última ecuación es una ecuación diferencial parcial (EDP) no lineal.

Las ecuaciones de conservación de masa y de la vorticidad, se acoplan a través de las condiciones de frontera de la segunda, las cuales deben ser expresadas en términos de la función de corriente. Las condiciones de frontera para la función de corriente, se pueden deducir facilmente de la geometría del problema y de las suposiciones físicas que se hacen para simularlo.

El problema de calcular el valor de las condiciones de frontera en las salidas del modelo —condiciones de frontera al infinito—, es un problema delicado, ya que cuando el fluido se vuelve menos viscoso, la influencia de las regiones de separación se extiende sobre todo el dominio y eso puede alterar el comportamiento del flujo a la entrada y a la salida del líquido y hacer que no coincida con las condiciones especificadas. Sin embargo alguna decisión debe ser tomada para modelar el flujo al infinito. Existen varias alternativas com son: especificar que el flujo no cambia a partir de un punto dado; especificar que la $\psi_{xxx} = 0$; extrapolar el valor en la frontera usando 2 o tres puntos interiores etc.

Una forma alternativa de plantear el problema de Navier-Stokes consiste en combinar las ecuaciones I.4 y I.5 para obtener la ecuación biharmonica:

$$(\psi_y, -\psi_x) \cdot \nabla\omega = -\nu\Delta^2\omega \qquad \text{(I.6)}$$

para esta ecuación es necesario plantear condiciones de frontera en la función y en las segundas derivadas de ω primeras, segundas o terceras derivadas de la función de interés. Plantear el problema de esta forma lo hace más díficil de resolver numéricamente, porque para dicretizar una cuarta derivada se necesitan ocho vecinos, pero permite ver que el problema está bien planteado, al menos para ν suficientemente grande, ver [22].

I.2 Objetivos de este trabajo.

A lo largo de este capítulo, se ha planteado el problema de entender los fluidos biológicos, y en particular el flujo de sangre en una válvula coronaria totalmente abierta bajo un régimen laminar (no se modela el flujo pulsante de sangre). Se presenta la ecuación de Navier-Stokes, como el modelo matemático que se puede emplear para simular los fluidos biológicos.

Los flujos que son solución a la ecuación de Navier-Stokes, pueden tener regiones en donde pueden tener estructura (vórtices secundarios en las regiones de se- paración), que sería deseable poder resolver. Además se resolverá la ecuación de Navier-Stokes en dominios diferentes.

El problema planteado anteriormente, será el problema que sirva de motivación para desarrollar un programa de computadora, que permitirá resolver la ecuación diferencial de Navier-Stokes en diferentes dominios. El sistema deberá ser capaz de resolver otras ecuaciones diferenciales con poco esfuerzo por parte del usuario, como resultado adicional de la técnica usada para codificarlo.

Estas consideraciones obligan a programar el sistema computacional de una forma no convencional, porqué la técnica usual no permite cambiar fácilmente la ecuación diferencial ni el dominio. Los programas numéricos que trabajan usualmente con EDPs, resuleven una única ecuación en un único dominio, ésto constituye una restriccion en las posibles aplicaciones del programa desarrollado y del método numérico empleado, porqué el sistema solo es empleado una vez para resolver el problema propuesto (para el cual fue desarrollado). Si se trata de resolver otra ecuación, modificar los codigos suele ser un trabajo no trivial y a veces tan complicado o más que escribir un nuevo codigo que resuelva el nuevo problema, este último método suele tener muchas dificultades, y al menos de que el autor original realice las modificaciones suele ser no confiable. La solución a este último problema consiste en desarrollar un sistema, al cual se le puedan definir las ecuaciones de discretización de la ecuación diferencial en rutinas separadas que se ligan con el cuerpo principal del programa.

Debido a estas necesidades y gracias a la existencia de las ideas de programación modular y de la téoria de lenguajes de programación [4,3,5] se construyó un sistema modular que trabaja con la técnica de diferencias finitas —que es muy fácilmente adaptable a la captura de estructuras especiales—, dentro del cual sea posible definir diferentes dominios sobre los que se desea resolver la EDP y que además se pueda cambiar la ecuación diferencial fácilmente.

Capítulo II

Panorama del sistema.

En este capítulo se describirá el sistema que se desarrolló, para que el lector tenga una visión global de como funciona, de tal forma que le sea posible entender los detalles de las descripciones de cada uno de los módulos. Para ponerlo en condiciones de usar el programa.

Se empieza explicando de una forma breve el método de diferencias finitas el cual convierte una ecuación diferencial parcial (EDP) en un sistema de ecuaciones algebraicas, esto es con la finalidad de definir el problema computacional.

El método de diferencias finitas, implica describir discretizado el dominio de la ecuación parcial de alguna forma, asi como describir la discretización de la ecuación diferencial y de sus condiciones de frontera.

El discretizar el dominio en donde se va a resolver la ecuación diferencial, equivale a sobreponer una cuadrícula al dominio, y darle un número secuencial a cada vértice de la cuadrícula que quedó dentro del dominio. Los vértices podrían ser enumerados usando un sistema coordenado, solo que para explicar el método de diferencias, se usa generalmente la numeración secuencial, porqué corresponde con la numeración que se debe usar dentro del sistema de ecuaciones lineales. La relación que existe entre la posición del nodo en el sistema coordenado y la numeración secuencial se muestra en la figura II.1 (en la figura, los • representan a los vértices de la malla) para facilitar la comprensión de la equivalencia entre la numeración secuencial y la numeración coordenada (ver [6] en la sección en donde habla acerca de la solución de ecuaciones diferenciales parciales).

La función que el lenguaje de descripción y el compilador deben realizar, es por un lado, el describir la malla —en el caso de la figura II.1 es muy simple la descripción— y construir la función biyectiva que relaciona a la numeración secuencial con la numeración coordenada. Más adelante cuando se expliquen las mallas que se necesitan para implantar el método numérico propuesto (para lo cual será necesario complicar la estructura de la numeración de los nodos) tendrá sentido usar el lenguaje y el compilador para tal efecto.

$$\bullet(1,3) = 3 \quad \bullet(2,3) = 6 \quad \bullet(3,3) = 9$$

$$\bullet(1,2) = 2 \quad \bullet(2,2) = 5 \quad \bullet(3,2) = 8$$

$$\bullet(1,1) = 1 \quad \bullet(2,1) = 4 \quad \bullet(3,1) = 7$$

Figura II.1: El dominio esta formado por el rectángulo $(1,1),(3,3)$.

La discretización de la ecuación diferencial por el contrario, se explica haciendo uso de la numeración coordenada. Se considera un nodo (i,j), y alrededor de este nodo se consideran los vecinos que estan hacia: el norte, vecino $(i,j+1)$; el este, vecino $(i+1,j)$; el sur, vecino $(i,j-1)$; el oeste, vecino $(i-1,j)$. A cada uno de estos vértices, se le ha asignado una variable que tiene como subíndice a la coordenada del vértice, y que representa el valor de la función incognita en el punto. Haciendo uso de estas variables, se escriben las ecuaciones que discretizan a las derivadas que aparecen en la ecuación diferencial.

Estas discretizaciones, se ven como polinomios que expresan el valor de las derivadas en el punto (i,j) haciendo uso de los puntos $(i-1,j)$ y $(i+1,j)$ si las derivadas son con respecto a x, y de los puntos $(i,j-1)$ y $(i,j+1)$ si las derivadas son con respecto a y.

El siguiente paso en la discretización de la ecuación diferencial, consiste en sustituir las expresiones discretizadas de las derivadas en la ecuación parcial. Las apariciones de la función incognita, se sustituyen directamente por la expresión de la función dicretizada (que son los valores que toma la función incognita sobre cada uno de los vértices). Después de haber sustituido las expresiones discretizadas de las derivadas en la ecuación diferencial, un subíndice puede aparecer varias veces en la expresión que se obtiene, por lo cual es necesario reagrupar los coeficientes para que aparezca únicamente una sola vez cada subíndice en la expresión. Estos valores se convierten en las incognitas de un sistema de ecuaciones lineales, y se diferencian unos de otros por el sibíndice, que puede ser un elemento de la numeración secuencial o coordenada, según convenga (el lector debe recordar que existe una biyección entre las dos numeraciones). Se genera una ecuación discretizada de este tipo para cada nodo de la malla y es este conjunto de ecuaciones el que forma al sistema lineal.

En los parráfos anteriores, se han descrito dos procesos diferentes: el primero, que corresponde con la discretización del dominio, y el segundo, que corresponde con la discretización de la ecuación diferencial. Cada uno de ellos debe ser abordado de diferente forma para lograr desarrollar el sistema que se tiene en mente.

El proceso de discretización de dominio, por sus características, se puede automatizar a través del uso de un lenguaje. Porqué el problema que se presenta es exactamente el de describir un objeto para el cual se debe generar una

descripción que un computador entienda. Este punto lo resuelven muy bien los compiladores de los lenguajes de alto nivel modernos, cuando traducen las declaraciones de las estructuras de datos. Solo falta diseñar el lenguaje, la forma de compilarlo y las estructuras de datos que se deben usar.

El proceso de descripción del dominio, requiere que se diga cual será el componente básico que se usará. Como el método de diferencias finitas trabaja —al menos en este caso— con coordenadas rectángulares, se puede usar un rectángulo como la unidad elemental para describir el dominio. Por lo tanto, la descripción de dominios se hará usando rectángulos.

El segundo proceso, es aquel de la discretización de la ecuación diferencial y de sus condiciones de frontera. El cual deberá ser resuelto de una forma diferente, porqué no se trata de describir un objeto, sino de describir un conjunto de operaciones aritméticas que deben ser llevadas a cabo cada vez que se desea generar un renglón del sistema de ecuaciones lineales, el cual, está directamente relacionado con un nodo de la malla y por ende con la topología (en este punto, se debe observar que la forma de dicretizar la ecuación diferencial es independiente de la posición del nodo central que se use. Basta con decir cuales son los nodos vecinos del nodo central y a que distancias están para poder expresar completamente el polinomio de discretización. Este hecho será explotado para modularizar al sistema, ya que será suficiente con que el sistema le comunique al usuario la información de cuales son los vecinos y de las distancias a los vecinos, para que este sea capaz de escribir el polinomio de discretización. Algo parecido ocurre en el caso de las condiciones de frontera).

Para poder escribir un programa modular en este caso, es necesario describir la forma de calcular las ecuaciones de cada nodo, dentro de algunas subrutinas que esten separadas del sistema, y que puedan ser "ligadas" con los programas capaces de generar el sistema de ecuaciones a partir de la descripción de la malla (es aqui donde las propiedades de la discretización se explotan para definir una interface que permita separar las rutinas que el usuario escribe). Esto implica que la descripción de la discretización de la ecuación diferencial, deberá ser programada en algún lenguaje de alto nivel que permita tener módulos separados.

En el siguiente paso de la descripción a *grosso modo* del sistema desarrollado, se supone que ya se tienen los programas que forman al sistema, y que el usuario ya ha discretizado el dominio y escrito el programa correspondiente, asi como los programas que definen la discretización de la ecuación diferencial y de las condiciones de frontera.

Para definir el dominio, el usuario usó el lenguaje de descripción, lo que le obligó a escribir un programa que debe almacenar en un archivo dentro de su ordenador. El nombre del archivo, deberá tener la extensión **def**[1] para que el

[1] En la actualidad los nombres de los archivos en casi todas las máquinas se forman de dos partes. La primera que es el nombre que bautiza al archivo, y la segunda que es una extención. El nombre puede variar de longuitud dependiendo de la máquina, mientras que la extención es por lo general de tres letras. La forma de hacer referencia a los archivos en las máquinas consiste en poner el nombre y la extención juntos y entre ellos un punto.

compilador lo encuentre. Una vez almacenado el programa, debe ser compilado para producir los archivos con las listas de vecinos que describen al dominio discretizado, estos archivos son la salida del compilador y constituyen la interface entre la descripción del dominio y el programa que genera y resuelve los sistemas de ecuaciones algebraicos.

Los archivos de salida que el compilador produce, toman el nombre del archivo que contiene el programa de definición del dominio y lo usan para generar cinco archivos cuyos nombres estan formados con el nombre del programa de descripción y con extenciones diferentes.

Para traducir el programa de definición del dominio, el usuario debe ejecutar los programas **malla** y **mlist** en este orden, yuxtaponiendo el nombre del programa y el nombre (sin la extensión) del archivo que contiene el programa de definición de dominio, con lo cual el usuario logra crear la descripción del dominio que tendrá que usar posteriormente.

Como se supuso que el usuario ya ha escrito las rutinas que definen la discretización de la ecuación diferencial y de las condiciones de frontera, se puede proceder a explicar que debe hacer con estos programas para obtener un código ejecutable.

Para programar la parte numérica del sistema, se escogio usar el lenguaje FORTRAN —por razones pragmáticas—, por esta razón, las rutinas que el usuario ha escrito deberán estar programadas en FORTRAN y deberán ser compiladas por el mismo compilador con el que se compile el resto del sistema. Después se deberán ligar las subrutinas del usuarios con el programa **ireg**, con las rutinas que implantan los métodos numéricos y con los algoritmos de solución de la ecuación diferencial[2]. Para asi obtener un programa ejecutable que sepa resolver el problema planteado[3].

Después de crear el programa ejecutable, el usuario deberá correrlo e indicarle al programa en que archivo está almacenada la información del dominio sobre el que se desea resolver la ecuación diferencial que se ha discretizado y para la cual se escribieron los programas mencionados en el parráfo anterior. El nombre de archivo que se le debe entregar al programa cuando lo solicita es el nombre formado con el título del archivo que contiene la descripción del dominio y con la extensión **inf**.

Como resultado de este proceso y si todo funcionó adecuadamente, la solución al problema planteado deberá estal almacenada en un archivo cuya extensión es **sol**[4].

Hasta el momento, se ha descrito únicamente la conceptualización del proceso de diseño del sistema y de su uso en lo que a la discretización del dominio y

[2]Este conjunto de rutinas será discutido más adelante.

[3]Este paso del proceso depende del sistema en el que el usuario este trabajando. En los apéndices aparecen archivos de comandos para realizar el trabajo en un computador VAX 11/750 con sistema operativo VMS 4.7.

[4]En algún punto de la programación del algoritmo de solución de la ecuación parcial, el usuario indica el nombre del archivo que se usa para almacenar la solución

de la ecuación diferencial se refiere. Falta mostrar como es que el sistema podrá trabajar con diferentes ecuaciones diferenciales que pueden requerir métodos diferentes de solución. Por ejemplo: el problema de resolver la ecuación de Laplace, es muy diferente al problema de resolver la ecuación de Navier-Stokes. La primer ecuación requiere generar y resolver únicamente una vez el sistema lineal asociado con el Laplaciano, mientras la segunda requiere iterar con los sistemas asociados con cada una de sus ecuaciones parciales.

¿Como se pueden resolver simultaneamente estas necesidades?.

Para poder presentar una solución al problema antes mencionado, es necesario analizar las funciones que deberán ser realizadas por la parte numérica del sistema y su relación con el problema que se está planteando.

La ecuación diferencial debe ser convertida en un sistema de ecuaciones lineales haciendo uso de las rutinas en las que el usuario especifica la discretización, y de la información que produjo el compilador a partir del programa que describe al dominio.

Este hecho, muestra que la primera tarea que debe realizar la parte numérica del sistema es generar el sistema algebraico. Durante la generación del sistema lineal, solamente se usan las rutinas que definen la discretización de la EDP y la información de salida del compilador. Esto significa que si se cambiara una rutina de discretización por otra rutina de discretización que use la misma interface para comunicarse con el sistema, pero que defina otra EDP; el sistema lineal que se habrá generado será el sistema correspondiente a la nueva ecuación, mientras que la malla de discretización es la misma para ambos casos (no se debe pensar que existe algún problema con las condiciones de frontera, ya que cada vez que se define una discretización de una EDP, se debe definir una discretización para las condiciones de frontera, que estén de acuerdo con la discretización del dominio).

El segundo problema consiste en resolver el sistema algebraico antes mencionado. Este problema es totalmente independiente de los pasos anteriores. En este paso, puede ocurrir que el sistema generado no se pueda resolver con alguno de los métodos ya programados, debido a que esté malcondicionado. En ese caso, el usuario deberá escribir o adaptar algún método numérico para resolver el problema y/o cambiar la técnica de discretización[5] de la EDP para adaptarla mejor al problema y asi lograr que la matriz esté bien condicionada[6].

El último problema fundamental que el sistema debe resolver es imprimir la solución.

Para llevar a cabo estas tareas, es posible que en algún momento el usuario deba escribir algunas rutinas que realicen tareas secundarias que dependerán de su problema específico, las cuales deberán ser programadas de una forma consistente con el sistema.

[5]Esto es sencillo, y se realiza en las rutinas que el usuario escribe.

[6]En el caso de que se tenga que programar un método numérico, se debe emplear la estructura de datos que se usa para almacenar la matriz.

Ya se han mencionado todos los elementos que toman parte en el proceso, solo falta explicar como se programaron para que el lector tenga la visión completa del sistema.

Para facilitar el trabajo del usuario, el programa lleva a cabo todas las labores de bajo nivel que son necesarias para generar los sistemas de ecuaciones, resolverlos e imprimir las soluciones. Estas funciones las desempeña dentro de unas rutinas que le entrega al usuario, para que sea él quien dirija el proceso de creación y solución, indicando en que momento se debe realizar cada operación. A lo largo de este proceso, se requiere usar algunas rutinas secundarias que el usuario debe escribir.

Para realizar el trabajo, el sistema llamará en su momento a la rutina que dirige el proceso de creación y solución, y a las rutinas secundarias. De tal suerte que el usuario se debe preocupar únicamente por enetender como funciona el sistema y no por como se generan y resuelven las ecuaciones.

El método de separar la discretización del dominio, la discretización de la EDP, las funciones que el sistema provee y las rutinas secundarias, permite combinar facilmente el tipo de ecuación diferencial que se desea resolver, gracias a que las rutinas que el usuario escribe y a que la definición del dominio son totalmente independientes del resto del sistema. Esta técnica permite inclusive cambiar la ecuación diferencial que se desea resolver durante el proceso de solución, si el usuario se encarga del problema logístico de manejar la información.

En resumén, el sistema tiene dos entradas de información que se procesan de forma diferente: la primera es el programa que define al dominio; la segunda, las rutinas en FORTRAN —discretización de la ecuación y algoritmo de solución— que se ligan con el generador de ecuaciones. Hasta después de haber llevado a cabo estas dos actividades, compilar el dominio y ligar las rutinas en FORTRAN, se puede resolver el problema planteado.

Se diseño el sistema pensando en que la tarea del usuario fuera lo más simple posible, este enfoque tiene como resultado el ocultar muchos detalles de su vista, porqué trata de relevarlo de todas las tarea laboriosas que es necesario llevar a cabo. Como ejemplo a este comentario, se tienen dos situaciones perfectamente claras.

La primer situación ocurre cuando se trata de resolver una ecuación diferencial con diferencias finitas a mano. Para hacerlo, es necesario discretizar el dominio y numerar todos y cada uno de los nodos que resultan. Del dibujo que se hace (figura II.1), se puede observar directamente cual es la posición de cada nodo en el dominio, por lo que no se necesita usar una numeración coordenada para hacer referencia a los nodos. En el caso de querer hacer la discretización automáticamente es necesario tener las dos numeraciones. En este sistema se releva al usuario del problema de numerar los nodos, se le proporciona una numeración coordenada para hacer referencia a cada nodo del dominio y se le oculta todo el manejo de información que el compilador lleva a cabo para poder crear las descripciones del dominio que hacen uso de los dos tipos de numeración.

El proceso de compilación lo que hace es crear una base de datos con la información necesaria para generar las listas de vecinos. Las llaves de acceso estan formadas con las numeración bidimensional, lo cual integra toda la información geométrica necesaria para crear las listas de vecinos en la base de datos.

La segunda situación se da durante el proceso de formación del sistema de ecuaciones. A primera vista puede parecer engañosa la forma como se le comunica al programa la discretización de la ecuación diferencial, porqué se usan tres arreglos para depositar la información de la discretización y no se ve el proceso de creación del sistema de ecuaciones al momento de programar las ecuaciones discretizadas. El proceso de construcción del sistema algebraico se oculta para lograr que la forma de programar las ecuaciones discretizadas sea cercana a la forma como lo explican los libros. El problema consiste en que las ayudas visuales con las que cuentan los libros no las tiene el programa.

El programa que genera el sistema de ecuaciones, toma las listas de vecinos generadas por el compilador, y usa la discretización de las ecuaciones para generar las ecuaciones lineales llamando a las rutinas de discretización que el usuario provee y que contienen la discretización de la ecuación diferencial, que se llevó a cabo de la forma tradicional, y que es independiente de la posición del nodo, lo cual permite que **la descripción de la ecuación discretizada sea la misma para todo nodo en el dominio, sin importar que tipo de vecinos tenga, ni su posición particular.** Toda la información necesaria para generar los polinomios de cada ecuación, es transmitida al usuario por el programa a través de algunas variables, y es lo suficientemente general para permitir la independencia de la posición. Los polinomios de discretización son almacenados siempre en el mismo juego de arreglos en las rutinas que el usuario escribe, cada vez que la rutina es llamada, el usuario no debe temer por el futuro de su discretización, porqué durante el proceso, la información será transferida al arreglo que contiene al sistema de ecuaciones, después de haber sido ordenada deacuerdo con la numeración secuencial de los nodos.

El sistema de ecuaciones se construye de una forma sistemática, el programa llama para cada renglón de la matriz, a la rutina que el usuario ha escrito, para que le entregue el polinomio que discretiza a la ecuación correspondiente al renglón que está procesando. Acto seguido, el programa procede a sustituir los nodos que no correspondan a nodos de discretización del dominio, por sus valores correspondientes o por los polinomios adecuados que los describen.

Después de sustituir y reagrupar los polinomios, la ecuación obtenida es ordenada para respetar la numeración secuencial de los nodos del dominio (que corresponde con la numeración de los renglones y las columnas de la matriz del sistema lineal) para luego ser insertada en la estructura de datos que contiene a la matriz del sistema.

La salida del sistema, que es la solución, es una lista de números, cada renglón de la lista representa un nodo del dominio (incluidas las fronteras), que está descrito mediante sus coordenadas (x, y) y el valor de la función en el punto.

Puntualizando, el usuario debe:

- escribir el programa que define el dominio en donde desea resolver su ecuación diferencial. Primero debió hacer un dibujo del dominio y diferenciar todas y cada una de las condiciones de frontera para poder escribir el programa de definición. Además de anotar todas las coordenadas de los puntos de interés (todas las esquinas).
- compilar el programa de definición de dominio.
- escribir las rutinas que discretizan a las condiciones de frontera y a la ecuación diferencial.
- escribir las rutinas de control y las rutinas secundarias.
- compilar las rutinas y ligarlas con el resto del sistema.
- finalmente, correr el programa generado en el paso anterior.

si todos los pasos se pudieron realizar satisfactoriamente y el problema matemático que se esta planteando está bien entendido, el usuario habrá obtenido después de correr los programas, una solución a su problema, con los errores numéricos que provienen de la discretización.

Capítulo III

El Método numérico empleado.

Este capítulo comienza mostrando cual es el tipo de mallas refinadas que se tiene en mente ver figura III.1, para resolver los problemas que se mencionaron en el primer capítulo. Esta forma de refinar la malla es la nueva estrategia de discretización que se propone para tratar de resolver la estructura fina de la soluciones a las ecuaciones diferenciales. La idea central de la estrategia es poder crear regiones del dominio en donde se reduzca el error numérico aumentando el menor número de nodos posible.

Al final de la primera sección de este capítulo, se muestran los resultados que se optubieron para la ecuación de transporte.

La malla mostrada en la figura III.1, tiene un paso h, es decir, la distancia horizontal o vertical entre los nodos gruesos (representados con $\cdot$) es de h. En el centro de la malla se ha definido una región de paso de malla $h/2$, que se puede ver claramente en la figura III.1. El problema que se debe resolver para poder usar estas mallas, consiste en calcular los valores de los nodos marcados con $\bullet$, porqué deben ser usados para escribir las ecuaciones dicretizadas de los nodos refinados, marcados con o. Si estos nodos no existieran, no se pordrían escribir las ecuaciones discretizadas de los nodos refinados, mientras que por el contrario, para escribir las ecuaciones discretizadas de los nodos gruesos no existe ningún problema. Los segmentos en donde se encuentran los nodos de interpolación (representados por $\bullet$) son las fronteras de la malla de interpolación y deben se estar localizadas en una zona en donde la tercera derivada de la función sea pequeña para que el acoplamiento de la malla sea adecuado. Más adelante, se explicará como funciona el acoplamiento de la malla para que el usuario sepa lo que debe hacer para poder usar este tipo de mallas.

A continuación, se presenta una simplificación del problema de Navier-Stokes para explicar y desarrollar el método numérico empleado para calcular los nodos

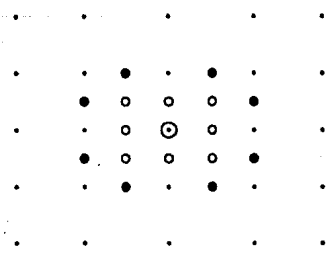


Figura III.1: Malla refinada, los · representan a los nodos gruesos, los ○ a los nodos refinados, los ● a los nodos de interpolación y los ⊙ a los nodos gruesos que han quedado dentro de una región refinada.

de interpolación.

III.1 Simplificación al problema de Navier-Stokes.

Considere las ecuaciones: la de conservación de masa I.4 y la ecuación de la vorticidad I.5

$$\begin{aligned} \Delta\psi &= -\omega \\ (\psi_y, -\psi_x)\cdot\nabla\omega &= \nu\Delta\omega \end{aligned}$$

respectivamente. Se puede observar que la ecuación I.4 es una ecuación de Poisson y el método de diferencias finitas no tiene problema para resolverla. Lo mismo no sucede con la ecuación I.5. Esta ecuación tiene coeficientes variables y sus soluciones presentan capas límite y vórtices, es decir, las soluciones pueden tener regiones de cambios rápidos y regiones donde los valores varian muy poco con respecto a un valor.

Para desarrollar y probar el método numérico que aquí se propone, se va a trabajar inicialmente con una ecuación más simple que la ecuación I.5 y que además tiene una solución analítica contra la cual comparar los resultados numéricos.

Suponga que $\psi_y >> -\psi_x$, entonces se puede despreciar el término $-\psi_x\omega_y$, con lo que la ecuación I.5 se reduce a:

$$\psi_y\omega_x = \nu\Delta\omega \qquad \text{(III.1)}$$

si además se supone que ψ_y es casi constante en el dominio de intéres, la ecuación III.1 se puede escribir de la siguiente forma:

$$\epsilon\Delta u + u_x = 0 \qquad \text{(III.2)}$$

con ϵ un parámetro pequeño. En este caso, el dominio en donde se resuelve la ecuación es el cuadrado $0 \leq x \leq \pi$, $0 \leq y \leq \pi$, con condiciones de frontera $u = \sin(y)$ para $x = 0$ y cero en los otros tres lados.

Esta ecuación imita un comportamiento de la ecuación de Navier-Stokes, posee soluciones analíticas con cambios rápidos pero como consecuencia de que se desprecian las velocidades verticales y no se conserva la masa deja de modelar la física del problema original, pero, se usa para empezar a entender el método numérico propuesto y para desarrollar el lenguaje y el compilador.

La solución analítica de esta ecuación, se puede obtener fácilmente con el método de separación de variables, y es:

$$u(x,y) = \sin(y)\left(Ae^{\lambda_1 x} + Be^{\lambda_2 x}\right) \tag{III.3}$$

con:

$$\begin{aligned} \lambda_1 &= \frac{-1+\sqrt{1+4\epsilon^2}}{2\epsilon} \\ \lambda_2 &= \frac{-1-\sqrt{1+4\epsilon^2}}{2\epsilon} \\ A &= \frac{e^{\lambda_2 \pi}}{e^{\lambda_2 \pi} - e^{\lambda_1 \pi}} \\ B &= 1 - A \end{aligned}$$

para $\epsilon << 1$, el término $\sqrt{1+4\epsilon^2}$ se puede desarrollar en serie alrededor de $\epsilon = 0$, con lo que se obtiene que $\sqrt{1+4\epsilon^2} \approx 1$, lo cual implica que $\lambda_1 = O(\epsilon)$, $\lambda_2 = -\frac{1}{\epsilon} + O(\epsilon)$, $A << O(\epsilon)$ y $B = 1 + \epsilon'$ con $\epsilon' << O(\epsilon)$. Sustituyendo en la solución analítica III.3 resulta que:

$$u(x,y) = \sin(y)\, e^{-x/\epsilon} + O(\epsilon). \tag{III.4}$$

Si se calcula el valor del factor $e^{-x/\epsilon}$ para distintos valores de x, se encuentra que la función ha caido en un 63% cuando $x = \epsilon$ y un 96% cuando $x = 2\epsilon$, i.e. la función varia muy rápidamente en el intervalo $[0, 2\epsilon]$, y después del cual es prácticamente constante —cero—.

III.2 Planteamiento del método numérico.

Si se usa el método de diferencias finitas para tratar de encontrar una solución a una ecuación con este tipo de comportamiento, aparece de inmediato la necesidad de usar mallas con paso muy pequeño, lo cual genera de inmediato sistemas de ecuaciones inmensos. Esta forma de resolver el problema sería muy buena en el caso de que la solución tuviera en todo el dominio cambios importantes, pero en el caso en el que la región de cambio esta concentrada en un intervalo o en una región no tiene sentido refinar la malla uniformemente. Como solución

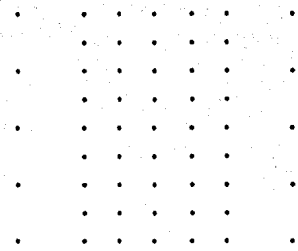


Figura III.2: Malla refinada sobre un intervalo del eje x.

a este problema se puede seguir alguna de las dos alternativas siguientes: usar una malla irregular; refinar la malla por regiones [?], ver figura III.1.

El usar una malla irregular puede ser muy útil en algunas ocasiones, pero sigue adoleciendo del problema de crear más ecuaciones de las necesarias, por ejemplo, piense en un dominio bidimensional y en el refinamiento de un intervalo del eje x, esta acción incrementa el número de nodos a lo largo del eje y sobre la región en la que se refinó el eje x, ver figura III.2.

Ahora bien, si se refina por regiones dentro de un dominio bidimensional, se pueden crear únicamente las ecuaciones necesarias para capturar los cambios de la función en la región de interés (suponga que la región de interes es la región refinada de la figura III.1). Esto presenta el problema de escribir las ecuaciones para los nuevos nodos de la malla que de ahora en adelante se les llamara *nodos de refinamiento* y a los nodos de la malla original, se les llamara *nodos gruesos*.

Al tratar de escribir la ecuación de un *nodo de refinamiento* que posee únicamente como vecinos a *nodos de refinamiento* o *nodos gruesos* que quedaron inmersos dentro de la región de refinamiento no existe problema alguno. Las dificultades empiezan al momento de acoplar la malla refinada con la malla gruesa, ya que para escribir las ecuaciones de los *nodos de refinamiento* vecinos de la frontera es necesario conocer el valor de la función sobre la frontera de las dos mallas, valor para el cual no se puede plantear una ecuación. Para encontrar este valor, se interpola usando valores de la función en puntos de la malla gruesa. De esta forma se obtiene un sistema de ecuaciones que corresponde a la ecuación diferencial que se desea resolver, ver figura III.3.

Para calcular el valor de la función en los *nodos de interpolación*, se hace un desarrollo en series a segundo orden, i.e. se desea conocer el valor $f\left(x_i + \frac{h}{2}\right)$, donde h es el paso de la malla gruesa y $f(x_i)$ representa el valor de la función en un *nodo grueso*, el cual es aun desconocido, pero se tiene una ecuación que lo calcula, y $x_i + \frac{h}{2}$ denota al *nodo de interpolación* marcado con $\bullet_I$ en la

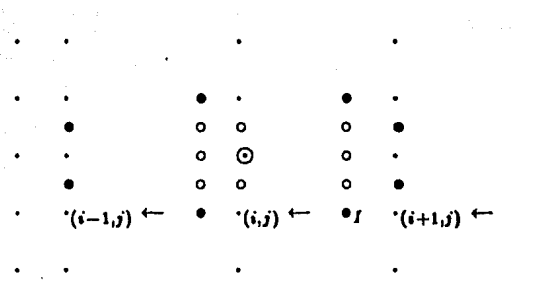


Figura III.3: Para calcular el valor del nodo marcado con I, se usan los nodos marcados con $\rightarrow$.

fugura III.3. Entonces:

$$f\left(x_i+\frac{h}{2}\right)=f(x_i)+\frac{h}{2}f'(x_i)+\frac{1}{2}\left(\frac{h}{2}\right)^2 f''(x_i)+O\left(\left(\frac{h}{2}\right)^3\right) \qquad \text{(III.5)}$$

donde las derivadas son calculadas con las expresiones de diferencias finitas centradas:

$$f'(x_i) = \frac{f(x_{i+1})-f(x_{i-1})}{2h}+O(h^2) \qquad \text{(III.6)}$$

$$f''(x_i) = \frac{f(x_{i+1})-2f(x_i)+f(x_{i-1})}{h^2}+O(h^2) \qquad \text{(III.7)}$$

lo cual una vez sustituído en la ecuación III.5 da como resultado:

$$f\left(x_i+\frac{h}{2}\right)=-\frac{1}{8}f(x_{i-1})+\frac{3}{4}f(x_i)+\frac{3}{8}f(x_{i+1})+O(h^3). \qquad \text{(III.8)}$$

El termino $O(h^3)$ contiene como factor a la f''', razón por la cual, el error de la aproximación se puede mantener bajo control, si el acoplamiento se realiza en una zona donde la tercera derivada sea pequeña, y/o el valor de h es suficientemente pequeño. Este hecho muestra que las mallas se deben de refinar de acuerdo a la idea que se tenga de la solución, y que se debe hacer uso de las dos posibilidades que existen, para controlar el error. Ya sea situando las interfaces de las mallas en los lugares donde f''' es pequeña, o haciendo que el valor del paso de la malla sea pequeño en ese sitio. Estos hechos suguieren dos cosas: a) hacer uso de mallas refinadas varias veces para minimizar el número de nodos; b) usar el método de prueba y error para localizar zonas del dominio donde se pueda realizar el acoplamiento de las mallas.

La forma de numerar estas mallas se ha complicado por necesidad, es necesario hacer distinción entre los nodos de refinamiento, los nodos de intepolación

y los nodos normales para poder generar las descripciones de estas mallas. El ejemplo mostrado es muy sencillo, pero si se piensa un ejemplo complicado, se verá inmmediatamente la necesidad de escribir un programa que le ayude al usario a llevar la contabilidad del proceso, porqué las nuevas descripciones, no necesitarán únicamente la descripción de los vecinos de cada nodo, sino además necesitarán la descripción de la estructura de los nodos de interpolación, en donde la estructura ya no es la estructura de los primeros vecinos, sino de los vecinos de orden mayor. Este nueva propiedad, hará desaparecer la simetría de los sistemas de ecuaciones generados con estas mallas, y tiene la posibilidad de hacer que las matrices asociadas estén mal condicionadas. Sin embargo se ha observado que si la malla se adapta bien al problema los sistemas de ecuaciones converjen correctamente.

III.2.1 Resultados obtenidos con el método.

Las soluciones numéricas que se obtienen son muy buenas cuando la malla que se usa es la apropiada, se han podido calcular soluciones numéricas de la ecuación III.2 con mallas de muy pocos puntos, por ejemplo, se calcularon soluciones a la ecuación III.2 con $\epsilon = 0.1$, 5329 nodos para el caso de la malla gruesa y 7244 en el caso de la malla fina, la malla fina se extiende hasta 4ϵ, mientras que se hubiera refinado todo el dominio, hubiera sido necesario usar de 15000 a 20000 nodos. El número de iteraciones fué de 258 y 372, respectivamente. Si se usa la norma $\|\cdot\|_\infty$ los errores que se encuentran en comparación con la solución analítica son: con la malla gruesa, del orden $5.5X10^{-3}$, mientras que con la malla refinada —a la cual se le han agregado nodos de paso $\frac{h}{2}$ en la región $0 \leq x \leq 2\epsilon, 0 \leq y \leq \pi$— son del orden $1.55X10^{-3}$ como se puede observar en la figura III.5.

Se ha podido obtener una solución a la ecuación III.2 para $\epsilon = 0.1$ usando 466 nodos con un error de $2.8X10^{-2}$, sin embargo la matriz no esta bien condicionada numéricamente y es costoso resolverla, para el caso de la malla fina se usó el método SOR con factor de relajamiento igual a 1.6 y se requieren 372 iteraciones. En el segundo caso no es posible usar el método SOR porqué diverge y se usó el algoritmo del gradiente generalizado para las ecuaciones normales, este último requirio de 5457 iteraciones. Se observa que para la ecuación III.2 el error que se comete en las regiones donde se usan mallas refinadas es menor que el error que se comete si se usa una malla gruesa, lo que parece apoyar las ideas propuestas. Esta técnica de discretización plantea el problema de generar sistemas de ecuaciones no simétricos, generalmente el método **SOR** puede resolverlos sin mayor problema. En el caso de que el método **SOR** no pudiera resolver los sistemas de ecuaciones, se deberá analizar la posibilidad de resolverlos con un método directo, o bien con la familia de métodos derivados del algoritmo del gradiente conjugado, que pueden resolver problemas no-simétricos.

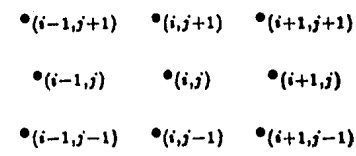


Figura III.4: Los desarrollos en series de Taylor se hacen alrededor del nodo (i,j), generalmente, los desarrollos en series se hacen en una dimensión, y después se cambian los subíndices para obtener las expresiones faltantes.

III.3 Discretización de la ecuaciones diferenciales.

El las dos subsecciones que siguen, se plantearán dos formas alternativas de realizar la discretización de la ecuación de Navier-Stokes. La primera llamada *diferencias finitas centradas* usa las expresiones de las primeras y segundas derivadas centradas en el punto sobre del cual se hacen los desarrollos en series de Taylor para encontrar la discretización de las ecuaciones diferenciales, mientras la segunda, calcula las primeras derivadas no centradas y las segundas derivadas centradas, este método tiene el inconveniente de generar una discretización de orden h —h es el paso de la malla— pero se puede corregir a segundo orden cuando el método para resolver la ecuación de Navier-Stokes ha convergido.

En el resto del capítulo cuando se haga referencia a una figura, será la figura III.4, que muestra a los vecinos del nodo (i,j). En este capítulo, se plantean las ecuaciones para una malla regular, lo que se debe hacer para escribir las ecuaciones para las mallas irregulares es hacer el algebra y calcular los nuevos coeficientes. Las nuevas expresiones se deben simplificar a las anteriores en el caso de que los pasos sean iguales. Las expresiones para los nodos de interpolación y las ecuaciones discretizadas para la malla irregular aparecen en el apéndice A.

III.3.1 Diferencias finitas centradas.

Para deducir la expresión de diferencias finitas centradas, se hace un desarrollo en serie de la función alrededor del punto x_i hacia la derecha y hacia la izquierda (los puntos que se usan para el desarrollo son: (i,j), $(i-1,j)$ e $(i+1,j)$, a los cuales se hara referencia con los subíndices i, $i-1$ e $i+1$ respectivamente):

$$f(x_i+h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + O(h^3) \qquad \text{(III.9)}$$

$$f(x_i-h) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) + O(h^3) \qquad \text{(III.10)}$$

para deducir la expresión de la segunda derivada, se suman las ecuaciones III.9 y III.10, después se despeja la segunda derivada, cuya expresión es:

$$f''(x_i) = \frac{f(x_i + h) - 2f(x_i) + f(x_i - h)}{h^2} + O(h^2). \qquad \text{(III.11)}$$

Para calcular la expresión de la primer derivada, se resta la ecuación III.10 de la ecuación III.9 lo cual da como resultado:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} + O(h^2). \qquad \text{(III.12)}$$

Por facilidad para expresar las ecuaciones discretizadas, se definen los operadores:

$$\begin{aligned}
\Delta_x^0 f_{i,j} &\equiv \frac{f_{i+1,j} - f_{i-1,j}}{2h} + O(h^2) \\
\Delta_y^0 f_{i,j} &\equiv \frac{f_{i,j+1} - f_{i,j-1}}{2h} + O(h^2) \\
\Delta_{xx} f_{i,j} &\equiv \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2} + O(h^2) \\
\Delta_{yy} f_{i,j} &\equiv \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h^2} + O(h^2) \\
\Delta_x^+ f_{i,j} &\equiv \frac{f_{i+1,j} - f_{i,j}}{h} + O(h) \\
\Delta_x^- f_{i,j} &\equiv \frac{f_{i,j} - f_{i-1,j}}{h} + O(h) \\
\Delta_y^+ f_{i,j} &\equiv \frac{f_{i,j+1} - f_{i,j}}{h} + O(h) \\
\Delta_y^- f_{i,j} &\equiv \frac{f_{i,j} - f_{i,j-1}}{h} + O(h)
\end{aligned}$$

Es fácil verificar que las siguientes propiedades se cumplen:

$$\begin{aligned}
\frac{1}{2}(\Delta_x^+ + \Delta_x^-) &= \Delta_x^0 \\
\Delta_x^+ - \Delta_x^- &= h\Delta_{xx} \\
\Delta_x^+ \Delta_x^- &= \Delta_{xx}
\end{aligned}$$

lo mismo ocurre para el caso de la y.

Para discretizar las ecuaciones diferenciales, se sustituyen las derivadas por las expresiones discretizadas. En el caso de la ecuación I.4

$$\Delta\psi = -\omega \qquad \text{(III.13)}$$

se sustituye el operador Δ por $\Delta_{xx} + \Delta_{yy}$, ψ por $\psi_{i,j}$ y ω por $\omega_{i,j}$, se obtiene que:

$$\Delta_{xx}\psi_{i,j} + \Delta_{yy}\psi_{i,j} = -\omega_{i,j} \qquad \text{(III.14)}$$

o sustituyendo las expresiones de los operadores y reagrupando:

$$-4\psi_{i,j} + \psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} = -h^2\omega_{i,j}. \qquad \text{(III.15)}$$

Para discretizar la ecuación I.5

$$(\psi_y, -\psi_x) \cdot \nabla\omega = \nu\Delta\omega \qquad \text{(III.16)}$$

basta con escribir, sustituir las expresiones de cada una de las derivadas que aparecen en la ecuación por su expresión discretizada, como se hizo con la ecuación anterior, lo que arroja en términos de los operadores antes definidos

$$(\Delta^0_y\psi_{i,j}, -\Delta^0_x\psi_{i,j}) \cdot (\Delta^0_x\omega_{i,j}, \Delta^0_y\omega_{i,j}) = \nu\Delta_{xx}\omega_{i,j} + \nu\Delta_{yy}\omega_{i,j} \qquad \text{(III.17)}$$

La ecuación que resulta después de sustituir los operadores es:

$$\begin{aligned} a_{i,j}\psi_{i,j} + a_{i-1,j}\psi_{i-1,j} + \\ a_{i+1,j}\psi_{i+1,j} + a_{i,j-1}\psi_{i,j-1} + \\ a_{i,j+1}\psi_{i,j+1} &= 0 \end{aligned} \qquad \text{(III.18)}$$

con:

$$\begin{aligned} a_{i,j} &= \frac{4\nu}{h} \\ a_{i-1,j} &= -\left(\frac{A}{2} + \frac{\nu}{h}\right) \\ a_{i+1,j} &= \left(\frac{A}{2} - \frac{\nu}{h}\right) \\ a_{i,j-1} &= \left(\frac{B}{2} - \frac{\nu}{h}\right) \\ a_{i,j+1} &= -\left(\frac{B}{2} + \frac{\nu}{h}\right) \\ A &= \frac{1}{2h}(\psi_{i,j+1} - \psi_{i,j-1}) \\ B &= \frac{1}{2h}(\psi_{i+1,j} - \psi_{i-1,j}) \end{aligned}$$

Las expresiones discretizadas III.15 y III.18 son usadas por el usuario para escribir las rutinas que definen a la discretización de la EDP (de la que se habló en el capítulo anterior). Como se puede observar estas expresiones no dependen de la posición del nodo central, únicamente dependen de los vecinos del nodo y de las distancias entre ellos.

Estas expresiones se pueden usar para resolver el problema de Navier-Stokes para números de Reynolds que van de 1 a 50, para números mayores, las matrices que esta discretización produce estan mal condicionadas (dejan de tener la diagonal dominante).

III.3.2 Upwinding correguido a segundo orden.

Para resolver la ecuación de Navier-Stokes para números de Reynolds altos —en este caso mayores a 50—, es necesario cambiar el método de discretización de la ecuación parcial I.5. Mientras que la ecuación discretizada III.15 no es necesario modificarla.

Para tratar de calcular una solución para números de Reynolds altos, lo que han sugerido diferentes autores (ver [8], pag 192) es aprovechar la información del flujo para discretizar la ecuación I.5 y asi obtener una matriz bien condicionada. Para viscosidad baja la ecuación de conservación de momento se vuelve cada vez más hiperbolica y la discretización debe ser consistente con las características. El esquema, lo que hace es calcular las primeras derivadas como derivadas no centradas en las direcciones x y y, luego combinarlas en función de la dirección del flujo. Este esquema por la forma de plantearlo recupera la dominancia diagonal del sistema de ecuaciones, pero resulta ser un esquema a primer orden. Para corregir la última dificultad, se hace una correcccción al método para que en el caso, cuando ya convergio, el resultado tenga un error a segundo orden.

Las ecuación I.5

$$(\psi_y, -\psi_x) \cdot \nabla\omega = \nu\Delta\omega \tag{III.19}$$

se discretiza de la siguiente forma:

$$\left(L(\psi^{m+1}_{i,j} - \nu\Delta\right)\omega^{m+1}_{i,j} = -\Lambda(\psi^{m+1}_{i,j})\omega^{m}_{i,j} \tag{III.20}$$

con:

$$\begin{aligned}
L(\psi_{i,j}) &= (\Delta^0_y\psi_{i,j})\Delta^*_x - (\Delta^0_x\psi_{i,j})\Delta^*_y \\
\Delta^*_x &= \frac{1}{2}(1-\epsilon^u_{i,j})\Delta^+_x + \frac{1}{2}(1+\epsilon^u_{i,j})\Delta^-_x \\
&\equiv \Delta^0_x - \frac{1}{2}h\epsilon^u_{i,j}\Delta_{xx} \\
\Delta^*_y &= \frac{1}{2}(1-\epsilon^v_{i,j})\Delta^+_y + \frac{1}{2}(1+\epsilon^v_{i,j})\Delta^-_y \\
&\equiv \Delta^0_y - \frac{1}{2}h\epsilon^v_{i,j}\Delta_{yy} \\
\Lambda(\psi_{i,j})\omega_{i,j} &= \frac{1}{2}h\epsilon^u_{i,j}u_{i,j}\Delta_{xx}\omega_{i,j} - \\
&\quad \frac{1}{2}h\epsilon^v_{i,j}v_{i,j}\Delta_{yy}\omega_{i,j} \\
u_{i,j} &= \Delta^0_y\psi_{i,j} \\
v_{i,j} &= -\Delta^0_x\psi_{i,j} \\
\epsilon^u_{i,j} &= signo(u_{i,j}) \\
\epsilon^v_{i,j} &= signo(v_{i,j})
\end{aligned}$$

donde los superíndices representan diferentes iteraciones para las incognitas. Se puede observar que cuando $\omega_{i,j}^{m} = \omega_{i,j}^{m+1}$ se recuperan las expresiones para el planteamiento de diferencias centradas. Claramente este criterio se debe entender con un nivel de error numérico.

Este planteamiento, permite calcular soluciones a las ecuación de Navier-Stokes para números de Reynolds mayores.

III.3.3 Discretización de las condiciones de frontera.

Para la ecuación de conservación de masa (I.4), el problema de discretizar las condiciones de frontera es muy simple. Basta con especificar el valor de la función de corriente a la entrada del flujo y en las paredes.

En los casos que se presentan, se usó un flujo de Poiseuille. Las características de este flujo, son que la velocidad del fluido junto a las paredes es cero y que el frente de velocidad es parabólico. Por la primera razón, las paredes constituyen una linea de corriente, y por lo tanto el valor de la función de corriente es constante sobre las paredes.

Para especificar el flujo a la entrada del modelo, se debe usar una función —función de corriente— cuya primera derivada sobre la línea de entrada al modelo del flujo, sea una parabóla (esta derivada es la velocidad del flujo perpendicular a la frontera de entrada al modelo), que los valores de esta derivada evaluados sobre las paredes sean cero, y que los valores de la función de corriente sean los especificados para las paredes.

La segunda derivada negativa sobre la frontera de entrada al modelo, es la función que se debe usar como condición de frontera en la entrada del modelo para la ecuación I.5.

El calculo de la condición de frontera en las paredes para la ecuación I.5 es un poco más complicado. Se usa la ecuación de la continuidad para evaluar el valor de la vorticidad a partir de valores de la función de corriente, que se conoce al momento de resolver la ecuación de la vorticidad. En coordenadas rectangulares y si la pared es paralela a eje de las x, se tiene que:

$$\psi_{xx} + \psi_{yy} = -\omega \tag{III.21}$$

se sabe que $\psi_{xx} = 0$ ya que la función de corriente a lo largo de la pared es constante por tratarse de una linea de flujo y que $\psi_y = 0$ por ser un flujo de Poiseuille, entonces la ecuación III.21 se reduce a:

$$\psi_{yy} = -\omega \tag{III.22}$$

Para calcular el valor de la ψ_{yy} se hacen dos desarrolos en serie alrededor de pared y con uno y dos pasos de malla, si el subíndice p indica la posición de la pared, el subíndice $p+1$ indica un nodo dentro de fluido y $p+2$ dos nodos dentro del fluido (en dirección perpendicular a la pared). Como la pared

se supuso paralela al eje de las x, las derivadas se toman en la dirección y. Entonces se tiene que:

$$\psi_{p+1} = \psi_p + h\psi'_p + \frac{1}{2}h^2\psi''_p \quad \text{(III.23)}$$

$$\psi_{p+2} = \psi_p + 2h\psi'_p + 2h^2\psi''_p \quad \text{(III.24)}$$

De las ecuaciones anteriores se tiene que:

$$\psi_{p+2} - 2\psi_{p+1} = -\psi_p + h^2\psi'' \quad \text{(III.25)}$$

$$\psi_{p+2} - 4\psi_{p+1} = -3\psi_p - 2h\psi' \quad \text{(III.26)}$$

si se restan estas expresiones y se despeja la segunda derivada, se tiene que:

$$\psi''_p = \frac{2\psi_{p+1} - 2\psi_p - 2h\psi'_p}{h^2} \quad \text{(III.27)}$$

que es la expresión ψ_{yy} valuada sobre la pared que se buscaba. Para el caso del flujo de Poiseuille $\psi'_p = 0$ y la expresión que queda es la expresión que se debe usar para calcular la condición de frontera para la ecuación de la vorticidad.

Para las esquinas, se puede decir que existen dos direcciones normales en la discretización, la forma de calcular la condición de frontera de las esquinas es usar la expresión anterior para cada una de las direcciones normales a las esquinas [17].

En la discretización de la condición de salida, se empleará la condición $f_x = 0$ (f puede ser la función de corriente o la vorticidad), esto implica que el valor de la función no cambia más allá de la frontera de salida, eso se expresa diciendo que $f_{frontera\ de\ salida} = f_{un\ nodo\ adentro}$.

Ecuación (4), $\varepsilon = 0.1$, Corte a lo largo del eje X
Arriba malla gruesa, abajo malla fina.

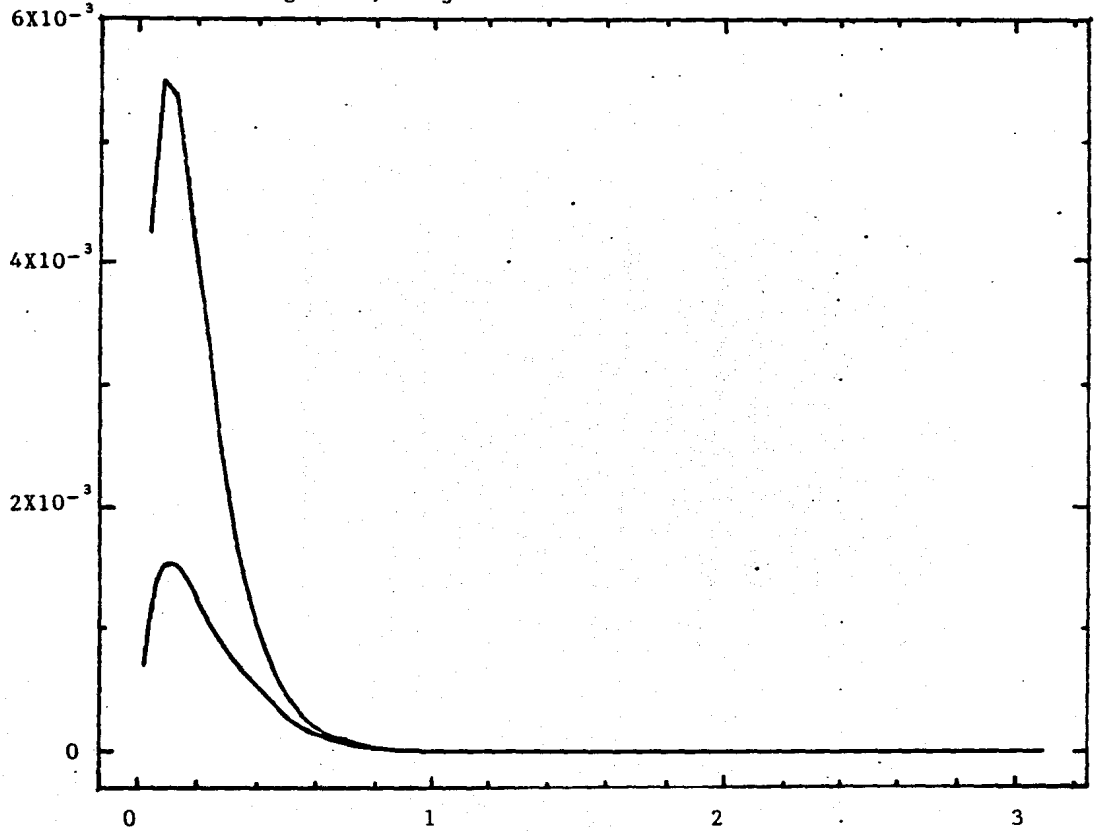


Figura III.5

Capítulo IV

El lenguaje de descripción y el compilador.

A lo largo de los capítulos anteriores se ha planteado el problema físico que se desea resolver, las dificultades que presenta, y los métodos numéricos que se desean aplicar, los cuales obligan a usar un tipo de mallas que se pueden refinar por regiones.

Las mallas refinadas por regiones, tienen mucha más estructura que las mallas que se usan tradicionalmente con la técnica de diferencias finitas. El poder refinar por regiones un número de veces diferente[1] independientemente de la región, hace la descripción de la mallas difícil, si se tratara de hacer la descripción como se hace normalmente en los programas que implantan la técnica de diferencias finitas, el resultado sería catastrófico, porque sería muy difícil poner a punto un programa semejante. Para poder manejar el problema de la descripción de este tipo de mallas, se diseñó un lenguaje sencillo que las describe y con el cual resulta muy sencillo hacer todo el trabajo, además se gana un poder de descripción para definir dominios que no se imaginaba.

A continuación se discutirán las mallas refinadas requeridas por los métodos numéricos, la técnica para describirlas (i.e. la interface con el usuario), y el procedimiento para producir una descripción que pueda ser usada por el sistema para generar y resolver los sistemas de ecuaciones correspondientes a la discretización de una ecuación diferencial.

El problema principal que se desea resolver es el de encontrar los vecinos de cada nodo de discretización del dominio y comunicarle esa información a los otros programas. Este problema puede parecer muy aparte del problema de la resolución de la ecuación diferencial, pero es el que da entrada a lo que se puede llamar el trabajo de computación dentro del contexto de esta tesis.

[1] Este concepto se explica a lo largo de este capítulo.

IV.1 Mallas refinadas.

Para visializar las mallas, piense en una malla regular a la manera de una hoja cuadriculada, la hoja representa el dominio sobre el cual se desea resolver la ecuación diferencial parcial, claramente, falta retirar de esta hoja las regiones que no pertenecen al dominio de la ecuación. La longuitud de las aristas reprenta el paso de la malla más fina que se usará para discretizar la EDP. Si la longuitud de las aristas es la misma para todos los cuadros se esta trabajando con una malla regular, si la longuitud de las aristas es diferente, se esta trabajando con una malla irregular.

Sobre la hoja cuadriculada, se sobrepone una lámina transparente cuadriculada que es la malla más gruesa que se usará para discretizar la EDP, la cual en cada cuadro agrupa 1, 4, 16, 64, 256, 1024, 4096 o 16384 cuadros de la hoja cuadriculada dependiendo del numero de veces que se desee poder refinar la malla. Si no se desea refinar la malla de la lámina transparente, cada cuadro de esta malla debe contener 1 cuadro de la hoja cuadriculada, 4 cuadros de la hoja cuadriculada si se desea refinar la malla transparente 1 vez, 16 cuadros si se desea refinar 2 veces y asi sucesivamente. El número maximo de veces que se puede refinar una malla es 7. Para identificar las mallas, se les asocia el número $n+1$, donde n es el número de veces que se refino para llegar a la malla en cuestión, al número $n+1$ se le llama nivel de refinamiento. Se puede ver que la malla de la hoja transparente está al nivel de refinamiento 1 (no se refina para llegar a esta malla, la malla refinada de la figura III.1, se encuentra a nivel 2.). Debido a esta forma de plantear las mallas refinadas, se ve que es necesario fijar de antemano el maximo número de veces que una malla podrá ser refinada, para que el proceso se pueda llevar a cabo. Si se trata de hacer de otra forma, el algoritmo de numeración y de creación de nodos de la malla debe ser muy complicado.

A los vértices de la hoja cuadriculada, que estan sobre la frontera del modelo (más adelante, se explicará como se define la frontera del modelo, por ahora, basta con suponer que coincide con la frontera de la hoja cuadriculada) se les llama *nodos de frontera*, estos nodos existen siempre y estan definidos al nivel más fino de las mallas. Los vértices de la malla transparente que estan dentro de las fronteras del modelo, reciben el nombre de *nodos normales*.

Sobre la malla transparente se pueden definir regiones que no pertenecen al dominio de la EDP. Para imaginar estas regiones, se debe pensar en un rectángulo de color negro que se sobrepone sobre la malla transparente, este rectángulo debe satisfacer la condicion de que sus esquinas inferior izquierda y superior derecha coninciden con *nodos normales* de la malla transparente. Los nodos de la frontera del rectángulo pasan a ser *nodos de frontera* sin importar su posición y los nodos interiores del rectángulo que eran *nodos normales*, desaparecen del dominio y no serán usados. Cualquier región que no pertenezca al dominio, aunque haya quedado fuera de la frontera definida por el usuario, deberá ser declarada como una región de no-dominio.

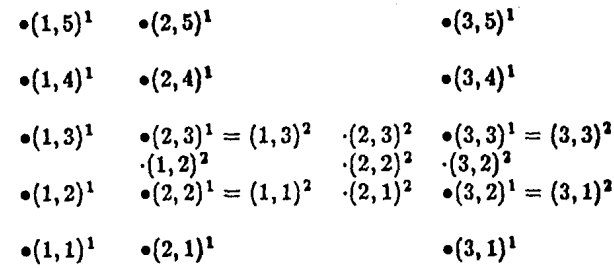
Sobre la malla más gruesa, la de nivel de refinamiento 1, se pueden sobreponer otras mallas de diferentes niveles de refinamiento, sin importar cual sea la diferencia entre sus niveles de refinamiento respectivos. La única condición que se debe satisfacer, es que las esquinas inferior izquierda y superior derecha de la malla refinada coincidan con *nodos normales* de la malla que contenga totalmente a la malla refinada. Esto de puede visualizar como los parches de una ropa de payaso, los diferentes colores de los parches representarian a los diferentes niveles de refinamiento, la única diferencia consiste en que las mallas más finas siempre estan encima de cualquier malla más gruesa. Las regiones que no pertenecen al dominio son ignoradas aunque se haya sobrepuesto a ellas una malla refinada.

Las regiones de mallas refinadas se definen haciendo uso de rectángulos de igual forma que con las regiones que no pertenecen al dominio, pero en este caso las fronteras de las regiones refinadas poseen dos tipos de nodos, aquellos que no estan encima de *nodos normales* de mallas más gruesas y aquellos que si lo estan, a los primeros se les da el nombre de *nodos de interpolación*, y los segundos pasan a ser *nodos normales*. Las coordenadas de las regiones refinadas se definen en función de la malla sobre la cual se esta refinando (la malla que las contiene totalmente), es decir, si se esta refinando sobre la malla transparente, las esquinas de la región refinada deben de coincidir con *nodos normales* de la malla transparente, y si se esta refinando sobre una malla refinada, las esquinas de la nueva región refinada deben de coincidir con *nodos normales* de la malla refinada anterior, aunque estos nodos no pertenezcan a la malla de nivel 1.

En el momento cuando se crea una región refinada, se crea simultáneamente una numeración de nodos relativa a la malla refinada. Para la malla transparente la numeración corresponde con la numeración del plano cartesiano definido por el producto cartesiano de los números naturales sin el cero, i.e. la esquina inferior izquierda recibe como etiqueta a la pareja $(1,1)$, el primer nodo sobre el eje de la X, recibe la etiqueta $(2,1)$, el primer nodo sobre el eje de la Y, recibe la etiqueta $(1,2)$. La nueva malla refinada tiene la misma convención para la numeración de sus nodos, su nodo inferior izquierdo es el $(1,1)$ y asi sucesivamente, ver figura IV.1.

Para hacer fácil la descripción de las mallas refinadas, se permite que las mallas refinadas se encimen entre ellas a diferentes niveles y que se encimen también con las regiones que no pertenecen al dominio del problema. Cuando las mallas refinadas se enciman y son de diferentes niveles, se debe usar la malla sobre la que cada malla refinada esté totalmente contenida para definirla. En el caso cuando se encima con las regiones que no pertenecen al dominio, los nodos de la malla refinada que quedan sobre los no-dominios desparecen. Si un *nodo de interpolación* queda entre dos mallas refinadas, i.e. es *nodo de interpolación* para las dos mallas simultáneamente, se convierte en *nodo normal* automáticamente.

Esta descripción define el tipo de mallas refinadas adecuadas para aplicar las técnicas numéricas explicadas en el capítulo anterior, pero tienen el inconveniente de no ser tan sencillas de describir y crear como lo eran las mallas



En la figura, los exponentes denotan el nivel de refinamiento de la malla, y las coordenadas entre parentesis la coordenada que le toca a cada nodo. Los nodos la malla gruesa • pueden tener dos coordenadas diferentes asociadas, y dependerá del nivel de refinamiento en el que se está trabajando para usar una numeración u otra. La malla refinada queda definida con las coordenadas $(2,2)^1$ *y* $(3,3)^1$ *de la malla gruesa.*

Figura IV.1: Numeración para mallas refinadas.

usadas con la técnica de discretización convencional. Una primera versión de este tipo de mallas aparece en el libro de [7], aunque no se desarrolla la forma de implantarlas ni se plantean las mallas con niveles multiples de refinamiento.

IV.2 Lenguaje de descripción de mallas.

Las mallas que han sido descritas en la sección anterior tienen una estructura complicada. Por esta razón se ha creado un lenguaje sencillo que las describe y que hace uso de algunas técnicas de los lenguajes de alto nivel con el objetivo de poder verificar de antemano que las propiedades fundamentales que cualquier malla debe satisfacer se cumplan, y asi poder garantizar que la malla está bien descrita. Aún con estas precauciones sera posible describir mallas que satisfagan las condiciones minimas para que se pueda crear con ella un sistema de ecuaciones, pero que no correspondan con el dominio que se desea describir. Este es el mismo fenómeno que se presenta en los lenguajes de programación actuales —el programa que se escribe no es necesariamente el que se desea escribir—. De ahi la necesidad de verificar que la descripción que se hace de la malla sea la correcta.

Para la técnica de elementos finitos existen programas basados en técnicas de CAD (computer aided design) que describen los dominios en los que se resulven las EDP, [14,15,16]. En este trabajo se decidió usar un acercamiento diferente, porqué no se contaba con facilidades para desarrollar una aplicación sobre un sistema CAD y porqué las mallas de diferencias finitas se deben definir siempre con rectángulos —hasta el momento— lo cual las hace más sencillas que las ma- llas de elementos finitos. De ahi la idea de usar un lenguaje que us-

ara rectángulos como instrumento de descripción para las mallas. Este lenguaje deberá manejar la información acerca de las dimensiones de la malla, los pasos de malla, considerando que el paso de malla es constante en la direccion X y diferente o igual pero constante en la dirección Y, número de niveles de refinamiento permitido, tipos de fronteras, forma de las fronteras, regiones de no-dominios y regiones de mallas refinadas. Esta información debera ser empacada en archivos que podrán ser usados por los programas que generarán los sistemas de ecuaciones correspondientes a la EDP.

Para desarrollar el lenguaje, es necesario entender como se describen los dominios de las EDP. Lo primero es definir la forma geométrica del dominio, a continuación se definen las condiciones de frontera que serán usadas, de Dirichlet, de Neuman o autoconsistentes etc. las condiciones de Dirichlet especifican el valor de la función en la frontera, las condiciones de Neuman especifican el valor de la derivada normal, las condiciones autoconsistentes especifican que la tercera derivada normal es igual a cero etc. Se pueden especificar diferentes tipos de fronteras en diferentes regiones de la frontera, por lo que el lenguaje que se defina debe ser capaz de manejar esta opción. También es necesario poder especificar los tipos de frontera de las esquinas, en el caso en que sea necesario considerarlas.

Para poder trabajar condiciones de Neuman o autoconsistentes, es necesario especificar que nodos son vecinos de los nodos frontera en la dirección normal, y se deben poder diferenciar los nodos de frontera de diferentes tipos de alguna forma. Las condiciones de Dirichlet son fáciles de manejar porqué solo necesitan que se especifique el valor de la función en la frontera. Para los otros tipos se deben especificar hasta cinco vecinos en la dirección normal.

Como objetivo de este trabajo se tiene el que el sistema desarrollado pueda trabajar con otras ecuaciones diferenciales y con dominios diferentes al usado en la simulación de la válvula, y que sea realmente fácil cambiar de dominio y de ecuación diferencial. Este objetivo implica que la descripción del dominio debe ser lo más general posible e independiente de la discretización de la EDP.

Por otra parte se sabe que la discretización de operadores elípticos de segundo orden en muchos casos requiere únicamente de los nodos vecinos a primer orden que estan al norte, sur, este y oeste del nodo central. De ahi que el compilador del lenguaje requiera crear una descripción de la malla que contenga la información geométrica, para que sea capaz de crear las listas de vecinos que se necesitan para crear la ecuaciones algebraicas.

El lenguaje que se diseñó tiene una estructura tipo PASCAL, i.e. comienza con un **begin**, termina con un **end**, y emplea estructuras anidadas para describir las mallas refinas y los diferentes parámetros de las expresiones que se usan para definir cada una de las partes del dominio. Se usan las palabras **levels, horizontal, vertical, hstep, vstep, frontiertypes, nodetypes, frontier, nodomain, grid** para especificar cual es el número maximo de niveles que se permitiran en una descripción dada, número de puntos en la dirección X a nivel de malla transparente, número de puntos en la dirección Y a nivel de

malla transparente, paso de la malla en la direccción X, paso de la malla en la dirección Y, número y definición de tipos de *nodos de frontera*, número de nodos de tipo especial (no fue necesario emplearlos, se dejo la definición para no modificar el parser del compilador), definición de la forma de le frontera y los tipos usados en cada región, definición de no-dominios y definición de mallas refinadas respectivamente.

A continuación se discutirá como ejemplo una definición de un dominio que corresponde a un tubo con dos obstáculos, el primero adherido a la pared horizontal inferior del tubo, y el segundo adherido a la pared superior del tubo. La distancia de la entrada del tubo al primer obstaculo es de una unidad y de tres cuartos de unidad al segundo obstaculo. El tubo mide dos unidades de largo, una unidad de alto y tiene posibilidad para refinar dos veces las mallas.

Toda definición de dominio, debe comenzar con la palabra:

begin

que marca el comienzo de la definición. La siguiente declaración define cuantos son los niveles de refinamiento que existirán como maximo en la definición de este dominio:

levels = 3.

El número de nodos de la malla transparente en las direcciones horizontal y vertical son definidos en las dos declaraciones siguientes:

horizontal = 62
vertical = 31

estas declaraciones implican que los valores que pueden tomar los índices de la numeración coordenada de la malla gruesa son: para la dirección horizontal de 1 a **horizontal**, para la dirección vertical de 1 a **vertical**.

A continuación se definen los pasos horizontal y vertical respectivamente de la malla de la hoja cuadriculada (malla más fina):

hstep = 0.008333333
vstep = 0.008333333.

Hasta aqui se han definido las características de todas las mallas que toman parte en la definición del dominio, a continuación se definirán las características de las condiciones de frontera, la forma geométrica del dominio, las regiones del no-dominio y la regiones de malla refinada.

Para definir las codiciones de frontera hay que definir cuantos tipos de fronteras existen y cuales son los vecinos que se deben tomar en cuenta para calcular los valores de la frontera. Se debe aclarar que las condiciones de frontera se pueden calcular de dos formas durante la discretización: explícita o implícita. La primera consiste en especificar el valor de la función de interés en la frontera, para la cual no es necesario especificar vecinos. En la discretización del segundo

caso se emplea un polinomio que esta expresado en función de los valores de la función incognita en los puntos vecinos del *nodo de frontera*, esto hace, que no sea posible conocer el valor de la función en la frontera, sino hasta que se ha resuelto el sistema de ecuaciones algebraicas, razón por la cual esta condición de frontera se incorpora en el sistema algebraico como parte de las ecuaciones, lo que hace que el sistema entregue como resultado los valores que se adaptan correctamente a la solución del problema y a las condiciones de frontera.

La palabra **frontiertypes** señala el comienzo de la declaración de tipos de frontera, la lista de tipos con sus características se encierran entre llaves y cada definición de tipo con sus vecinos se encierra entre parentesis cuadrados y se separa de la siguiente con una coma. En la especificación de vecinos de cada tipo de frontera se usan las palabras **nor, sou, wes y eas** —que indican la dirección en la cual se deben buscar los vecinos del nodo frontera—. Las palabras que indican la dirección de busqueda de vecinos van seguidas del un número, el cual indica cuántos vecinos hay que tomar en cuenta en esa dirección, esta descripción se encierra entre parentesis y en cada declaración de tipo de frontera pueden existir cero, una o dos descripciones de vecinos, separadas por un espacio en blanco entre ellas. Por ejemplo la definicón de la forma del dominio que se esta usando como ejemplo, comienza de la siguiente forma:

frontiertypes { [1 (nor 1)],

que especifica que el tipo 1 de frontera tiene 1 vecino hacia el norte, este tipo de frontera se usa para la pared horizontal inferior del tubo. A continuación se declaran los tipos de frontera que se necesitan para la salida del tubo —pared derecha—, pared horizontal superior y entrada del tubo, respectivamente:

[2 (wes 4)],
[3 (sou 1)],
[4],

el tipo de frontera 4, no requiere de vecinos porqué en el problema de la válvula la condición a la entrada del tubo siempre es explícita. Ahora le toca el turno al los tipos de frontera usados en los obstáculos. Se define primero el tipo de la esquina superior derecha del obstáculo de abajo, que tiene dos vecinos uno hacia el este y el otro hacia el norte:

[5 (eas 1) (nor 1)],

el tipo 6 usado para la pared vertical derecha del obstáculo inferior con un vecino hacia el este únicamente:

[6 (eas 1)],

el tipo 7, para la esquina superior izquierda del obtáculo de abajo, con sus dos nodos vecinos, uno hacia el oeste y el otro hacia el norte:

[7 (wes 1) (nor 1)],

la pared vertical izquierda del obstáculo inferior, con un vecino hacia el oeste:

[8 (wes 1)],

la esquina inferior derecha del obstáculo supeior, con dos vecinos:

[9 (eas 1) (sou 1)],

la esquina inferior izquierda del obstáculo superior:

[10 (wes 1) (sou 1)],

y las dos paredes verticales del obstáculo superior:

[11 (eas 1)],
[12 (wes 1)] }.

Las esquinas y las paredes verticales de los obstáculos deben ser consideradas en forma especial en el problema de la válvula coronaria, y es por esa razón que se definen los tipos de frontera de la esquinas y de las paredes.

La declaración de nodos normales especiales, no se uso en la declaración de ningún dominio y parece ser inútil , sin embargo se dejo para no modificar el compilador. No se toma en cuenta más que de una forma sintáctica.

nodetypes = 0.

En la descripción de la frontera, se usa una tortuga —la tortuga de lenguaje logo—, para trazar la forma geométrica de la frontera. Para describir la frontera, se especifica el tipo de nodo que le corresponde, el nodo donde se empieza a trazar la frontera y el nodo en donde termina este primer segmento recto de la frontera, el tipo mencionado al principio, es asignado a todos los nodos de la hoja cuadriculada que estan contenidos en el segmento de recta. Si al final de la definición de un segmento, aparece un signo igual, significa que la tortuga antes de avanzar hacia el siguiente nodo de la hoja cuadriculada que se le indique, cambia el tipo del nodo sobre el que está parada. Por ejemplo, observe estas instrucciones:

frontier { 1 [(1, 1) (62, 1)] =
2 [(62, 1)],

la primera expresa que el segmento que del nodo (1,1) al nodo (62,1) es de tipo 1. El signo igual al final de esta instrucción, obliga a que al nodo (62,1) se le asigne el tipo de frontera 2, cuando la segunda instrucción es procesada. Las tres frases a continuación expresan la forma como se forma la frontera:

2 [(62, 31)],
3 [(1, 31)],
4 [(1, 1)]}

los segmentos de recta, siempre deben ser horizontales o verticales.

Una frontera debe ser cerrada, por esta razón el último nodo que se usa debe ser el igual al primer nodo que se usó. La numeración usada en la definición de la frontera es la numeración de los nodos de la malla transparente.

La definición de los no-dominios es bastante simple, basta con especificar las esquinas inferior izquierda y superior derecha del rectángulo del no-dominio y los valores de los tipos de fronteras del no-dominio, empezando por el de la esquina inferior izquierda, luego el de la pared horizontal inferior, esquina inferior derecha, pared vertical derecha, esquina superior derecha, pared horizontal superior, esquina superior izquierda y finalmente pared vertival derecha. Las posisiones de los nodos de las esquinas se especifican entre parentesis y los valores de los tipos de frontera entre parentesis cuadrados. Como ejemplo de este tipo de descripción se presenta a continuación las dos instrucciones que definen los dos obstáculos del dominio antes mensionado. El ancho de cada obstáculo es de un treintavo, el alto del primero es de media unidad y el alto del segundo es de un tercio de unidad.

```
nodomain  (31, 1)  (32,16)  [1  1  1  6  5  1  7  8]
nodomain  (21,23)  (22,31)  [10 3  9  11 3  3  3  12]
```

Ahora se discutirá la parte más interesante de toda la descripción, el refinamiento local de mallas. En este ejemplo se pondrán dos zonas con mallas refinadas, una después de cada obstáculo y de la mitad del alto. En la malla refinada que va después del obstáculo inferior se colocara una malla más fina de la cuarta parte del area de la anterior, y pegada a la esquina. Es necesario recordar en este punto las convenciones acerca de la numeración de las mallas refinadas, ver figura IV.1.

```
grid    1    (32,   1)  (40,   8)  {
        grid 1     (1,   1)     (8,   8)
}
grid    1    (22,   27)  (26,   31)
```

El número que precede a la palabra **grid** indica cuantos niveles de refinamiento se toman en cuenta para construir la malla refinada. Notese que si sobre la malla transparente se refina una vez en el rectángulo (1,1) (2,2), el nuevo rectángulo refinado tendrá los puntos del producto cartesiano $(1,2,3) \times (1,2,3)$. Si se hubiera refinado dos veces, tendría los puntos del producto cartesiano $(1,2,3,4,5) \times (1,2,3,4,5)$ y asi sucesivamente.

Finalmente, la palabra **end** indica el final de la decripción del dominio.

IV.3 Descripción del compilador.

En esta sección es necesario describir las estructuras de datos y el algoritmo que se usa para compilar el lenguaje de descripción de mallas y generar la

descripción de la discretización del dominio. La compilación de este lenguaje, se lleva a cabo en dos etapas: primero se genera una estructura de datos que contiene toda la información geométrica del dominio discretizado, operación que lleva a cabo el programa llamado *malla*. Segundo se numeran todos los nodos del dominio discretizado y se crea una lista de vecinos que contiene toda la información necesaria para generar los sistemas de ecuaciones, este trabajo lo realiza el programa *mlist*. El primer paso de este proceso se puede entender como la generación de un código intermedio que facilita el resto del trabajo, y la segunda etapa corresponde a la generación de un código objeto que será ejecutado por la máquina virtual que conoce el procedimiento para generar las ecuaciones algebraicas y el algoritmo para resolverlas.

El primer paso de la compilación que corresponde a la generación de un código intermedio, empieza reconociendo sintacticamente el lenguaje antes descrito, se inicializan las variables que describen las mallas que se usarán y se almacena en memoria toda la información geométrica del dominio. La hoja cuadriculada se almacena en memoria haciendo uso de un arbol B^+ (se usó el paquete Btrieve), cuyas llaves estan formadas por tres entradas, la primera que corresponde al índice i en la dirección x, la segunda que corresponde al índice j en la dirección y y finalmente la tercera que es el tipo de nodo del que se trate. Como al inicio del proceso de compilación la base de datos esta vacia, es necesario insertar todos los nodos que pertenecen a la frontera del problema, a continuación se insertan todos los *nodos de frontera* de los no-dominios y enseguida los nodos del no-dominio, estos últimos se deben de considerar al nivel de los *nodos normales* de la hoja cuadriculada, para cancelar cualquier *nodos normales*en la región del no-dominio que pudiera ser usado en el futuro. Una vez creados los *nodos de frontera*y los *nodos de interpolación*, no se les puede cambiar ese atributo.

En el siguiente paso se insertan los *nodos normales* de la malla transparente de nivel 1. Es conveniente hacer notar en este momento, la forma como se ha ido almacenando la información geométrica de la malla. En la base de datos estan insertados únicamente los nodos que pertenecen a la frontera del modelo, los que pertenecen al no-dominio y los que pertenecen a la malla transparente de nivel 1. Para encontrar que nodo es vecino de un nodo (i,j) dado[2], basta con preguntarle a la base de datos si el nodo $(i,j+n)$ esta en la base de datos, para $n = \pm 1, 2, 3\ldots$, o bien preguntar si el nodo $(i+m,j)$ esta en la base de datos, para $m = \pm 1, 2, 3\ldots$. Si la respuesta es afirmativa para algun valor de n o m, el nodos $(i+m,j)$ o $(i,j+n)$ serán vecinos del nodo (i,j) en la dirección y sentidos respectivos. Como se puede observar la información geométrica esta implícita en la combinación de estructura de datos y conjunto de llaves usada, esto fácilita el inserción de nuevos nodos de niveles superiores, porque basta con

[2]Se desea aclarar que la numeración de los nodos siempre es la numeración de los nodos de la hoja cuadriculada, cuando se trabaja con la base de datos, la numeración relativa de todas las mallas transparentes que se usan durante la definición de un dominio, siempre es convertida a la numeración de la hoja cuadriculada

insertar el nodo (i, j) para que la convención de la numeración de nodos lleve la cuenta de los vecinos.

Conocida la estructura de datos que se usa para almacenar la malla discretizada, y explicado el procedimiento para insertar los *nodos normales* de la malla de nivel 1, se puede proceder a explicar como se insertan los *nodos normales* de las mallas refinadas.

En este paso basta con insertar directamente los nodos en la base de datos, y deducir cuales de los nodos de la frontera de las mallas refinadas son *nodos normales* y cuales son *nodos de interpolación*, obviamente este proceso de decisión se debe llevar a cabo después de que se han insertado todos los *nodos normales* de las mallas refinadas para que toda la estructura de la malla sea conocida y se puda deducir con certeza la estructura geométrica y por ende los *nodos de interpolación*.

La segunda etapa en el proceso de compilación es llevada a cabo por el programa llamado *mlist* y consiste en generar a partir de la información almacenada en la base de datos las listas de vecinos. Este proceso es realmente simple. Usando la tercera llave del arbol B^+ se le solicitan a la base de datos uno por uno y en secuencia los *nodos normales*, *nodos de frontera* y *nodos de interpolación* para numerarlos, en cada paso la base de datos es actualizada para que contenga la numeración de los nodos. Al terminar este proceso se buscan los vecinos para cada nodo de toda la malla. Como los nodos han sido numerados en orden, se puede distinguir fácilmente los tipos de nodos por el intervalo en donde está el número que los identifica.

El procedimiento para encontrar los vecinos de cada tipo de nodo es un poco diferente en cada caso. Para los *nodos normales* se buscan los primeros vecinos en las direcciones norte, sur, este y oeste. Para los *nodos de frontera* se respeta la definición para cada tipo, tomando en cuenta que los *nodos de interpolación* no pueden ser vecinos de los nodos de frontera, esto implica que si en la gráfica de la malla un *nodo de interpolación* es vecino de un nodo de frontera es ignorado y se busca por el siguiente nodo. El compilador y los programas que generan las ecuaciones verifican que esta restricción sea respetada de lo contrario lo indican y paran el procesamiento.

Para los *nodos de interpolación* se decide quienes son sus vecinos a partir de la información del nivel de refinamiento. Un nodo vecino de un *nodo de interpolación* siempre está a un nivel inferior de refinamiento, y el nivel de refinamiento siempre se puede deducir a partir de la numeración de la malla de la hoja cuadriculada. Esta es la regla principal para encontrar los nodos vecinos de un *nodo de interpolación*. Otro *nodo de interpolación* cualquiera puede ser nodo vecino de un *nodo de interpolación* si y solo si está a un nivel inferior de refinamiento. Por construcción de la malla, se garantiza que siempre es posible encontrar los nodos vecinos de los *nodos de interpolación*. Este proceso es complejo de explicar, porqué depende del nivel al que se este trabajando, de las direcciones sobre las que se deba interpolar y de la posición que ocupa el *nodo de interpolación* dentro de la malla. Existen muchas convinaciones para encontrar

los vecinos de los *nodos de interpolación* lo cual hace muy difícil explicarlo con detalle. La mejor forma de entender como se encuentran los nodos vecinos de un *nodo de interpolación* consiste en consultar los programas, en particular la rutina *checa_int* que es llamada por la rutina *make_inte*, ambas pertenecen al programa *mlist.c*. La regla básica de decisión ya ha sido mencionada y consiste en que los nodos vecinos de un *nodo de interpolación* siempre deben de pertenecer a un nivel inferior para poder garantizar que el error numérico de las interpolaciones no crece. No importa que nodos sean los nodos vecinos de un *nodo de interpolación* siempre y cuando se satisfaga la primer regla y todos los nodos vecinos esten dentro de la malla.

Capítulo V

Generación y resoluciòn de los sistemas de ecuaciones algebraicas.

En el capítulo anterior se explicó la forma como se generan las listas de vecinos que describen la malla discretizada. En este capítulo se describirá el contenido de los archivos generados por el compilador y se dirá como se usan los archivos para generar los sistemas de ecuaciones algebraicas. Se explicará el procedimiento de generación de los sistemas de ecuaciones y la interface con el método numérico.

V.1 Contenido de los archivos de salida del compilador.

El compilador genera como salida cinco archivos, cuyos nombres se forman con la primera parte del nombre del archivo que contiene la definición del dominio —este archivo siempre debe tener la extención **def**, y el nombre de este archivo es el nombre que el modelo recibe— y cuyas extenciones son **nor, des, inf, h y mss.**

El primer archivo —extención **nor**— contiene la descripción de todos los *nodos normales* del dominio. El formato de este archivo posee 13 columnas, las primera es un número secuencial que identifica al nodo de forma única. La segunda y tercera columnas contienen la numeración i, j sobre la malla de la hoja cuadriculada. La cuarta columna contiene información de si el nodo es un nodo especial —no se usa, siempre es cero—. La quinta columna contiene el nivel de malla refinada al que el nodo pertenece. Las últimas ocho columnas conienen los índices de los nodos vecinos del nodo definido en cada renglón. El

orden de los vecinos es el siguiente SW, S, SE, W, E, NW, N y NE —por el momento las posiciones de los nodos SW, SE, NW y NE contienen ceros porqué el sistema no puede trabajar con nodos de interpolación en esas direcciones—.

El segundo archivo —extención **des**— contiene las decripciones de los *nodos de frontera* y del los *nodos de interpolación*. El formato de los *nodos de frontera* es el siguiente: La primer columna contiene un número secuencial. La segunda y tercer columnas contienen la numeración i, j del nodo en la malla de la hoja cuadriculada. La cuarta columna indica si el nodo es especial o no. La quinta columna tiene el tipo de nodo —uno de los definidos en la declaración **frontiertypes**—. Las cinco columnas siguientes contienen los nodos vecinos del *nodo frontera*, la siguiente columna contiene la dirección que se usó para encontrar a los vecinos anteriores. Las suiguientes seis columnas contienen el mismo tipo de información que las seis anteriores. Estos dos últimos conjuntos de seis columnas estan relacionados con las declaraciones de tipos de fronteras de acuerdo con el tipo de nodo del renglón. Si la declaración de tipo de frontera no contiene vecinos, aparecerán ceros en las doce columnas. Si en la declaración se especifica una dirección con dos vecinos, en el primer conjunto de columnas, aparecerán dos columnas llenas y la especificación de dirección. Si se especificaron dos direcciones, los dos juegos de columnas contendrán información.

La segunda parte de este archivo contiene la información correspondiente a los *nodos de interpolación*, en este caso el archivo tiene una columna con el numero secuencial que identifica al nodo de forma única, dos columnas que contienen la numeración i, j del nodo sobre la malla de la hoja cuadriculada, una columna con la dirección que indica si se interpola hacia la izquierda o derecha del nodo central, tres columnas que contienen a los nodos que se usan para calcular el valor interpolado, una columna con la orientación —horizontal o vertical— del segmento de interpolación y la última columna que almacena el nivel de refinamiento al que el nodo pertenece.

El archivo cuya terminación es **h** contiene los pasos de mallas de la hoja cuadriculada, **en el caso que se quisiera trabajar con mallas irregulares, se deben de cambiar los valores en este archivo para que la nueva malla corresponda con el diseño que se desea.**

El archivo con extención **inf** contiene los nombres de los archivos que contienen el modelo, información acerca del número de nodos, número de niveles y la información para encontrar los nodos vecinos de los nodos de frontera (por tipos).

Finalmente, el último archivo —extensión **mss**— contiene la lista de vecinos de cada uno de los nodos usados en el dominio, que se puede imprimir y puede ser usado para interpretar las salidas del sistema.

El contenido del primer archivo es leido y almacenado en la matriz **norb2**, mientras que el contenido del segundo archivo es leido y almacenado en el arreglo **desarr**. De esta forma el programa que genera el sistema de ecuaciones tiene toda información necesaria referente a todos los nodos de la malla.

V.2 Discretización de la EDP y su interface con el sistema.

Uno de los objetivos iniciales en este trabajo, fue el poder cambiar facilmente la EDP que se modela. Para lograr este objetivo el sistema supone que el usuario programa las subroutinas que discretizan la EDP y las condiciones de frontera —**diffinmX**, **frontmX**—. Para expresar la discretización de la EDP y de las condiciones de frontera, el usuario dispone de dos mecanismos. El primero consiste en expresar el polinomio de la discretización de la ecuación o de la condición de frontera haciendo uso de tres arreglos, a saber **coef**, **cols** y **dirs**. Los polimonios se almacenan en los arreglos, depositando los coeficientes en el arreglo **coef** y las columnas (de la matriz) a las que pertenecen en el arreglo **cols**, esto se hace entrada a entrada.

La entrada i-ésima del primer arreglo contiene el coeficiente correspondiente a la incognita cuyo índice esta almacenado en la entrada i-ésima del arreglo **cols**. En la entrada i-ésima del arreglo **dirs** se guarda un 1 si el nodo que esta en la entrada i-ésima del arreglo **cols** está al norte, un 2 si está al este, un 3 si está al sur y un 4 si está al oeste. El segundo mecanismo consiste en entregar el valor numérico correspondiente a una parte de la ecuación discretizada o a la condición de frontera en una variable para que el sistema generador de ecuaciones lo pueda usar. En la programación de la discretización de la ecuaciones y de las condiciones de frontera, se puede usar uno, otro o los dos mecanismos.

Para la discretización de la EDP, el sistema le envia al usuario toda la información necesaria —acerca de los nodos vecinos que debe tomar en cuenta—, en un arreglo llamado **norb2**, el arreglo es una matriz, y para direccionarlo es necesario usar la variable *actin* en la primera entrada y alguna de las variables **east**, **west**, **nort** o **sur** en la segunda entrada. Al hacer la indirección del arreglo **norb2(actin,*)** se accesa el índice del nodo vecino en la dirección correspondiente. Las distancias entre el nodo central y los nodos vecinos aparecen en las variable **h1** para el nodo de oeste, **h2** para el nodo del este, **v1** para el nodo del sur y **v2** para el nodo del norte. El usuario debe llenar los arreglos **coef**, **cols** y **dirs** de acuerdo a las explicaciones del párrafo siguiente, además debe almacenar el valor del lado derecho de la ecuación discretizada en la variable **ld**. Como el usuario no sabe si alguno de los nodos vecinos que esta considerando es un nodo de frontera, no debe preocuparse de este hecho, el sistema sustituye los valores adecuados de los nodos de frontera automáticamente, de ahí la necesidad de especificar la forma de calcular los valores de las condiciones de frontera para cada uno de los posibles tipos y desde cada una de las posibles direcciones desde donde puede ser visto cada nodo de frontera —el lector debe reflexionar un momento sobre las esquinas de los obstáculos del modelo de la válvula coronaria que tienen dos formas diferentes para ser calculadas—.

El programa genera uno a uno los renglones del sistema de ecuaciones algebraicas, durante este proceso llama a cada paso —cada vez que empieza a

generar un nuevo renglón— la rutina **diffinmX**, esta rutina debe informarle al sistema los valores de los coeficientes de la ecuación discretizada, las columnas a las que cada uno de los coeficientes pertenece y las direcciones hacia donde se ven cada uno de los vecinos cuando el usuario esta parado en el nodo central. En este punto el usuario no se debe preocupar acerca de la estructura de la malla, toda la información de la malla ha sido resumida dentro de la información que se le entrega, por esta razón la discretización es independiente de la estructura de la malla y solo debe tener en cuenta que las distancias a los nodos vecinos desde el nodo central pueden no ser iguales. El número del renglón sobre el que se esta trabajando esta almacenado en la variable **actin**, este valor debe ser almacenado en la primer entrada del arreglo **cols** y el valor del coeficiente de la discretización correspondiente al nodo central —coeficiente $a_{i,j}$ de la discretización de la ecuación— debe ser almacenado en la primer entrada del arreglo **coef**, en este único caso el valor que se guarda en la primera entrada del arreglo **dirs** es cero. A continuación se puede considerar cualquiera de los nodos vecinos de la ecuación discretizada, como ejemplo se considerará el nodo $a_{i,j+1}$, que está en la dirección norte con respecto a la malla. El índice de este nodo se encuentra en la posición **norb2(actin, nort)**, este valor se debe almacenar en la siguiente entrada del arreglo **cols**, el valor del coeficiente $a_{i,j+1}$ se debe guardar en la siguiente entrada del arreglo **coef** y la siguiente entrada del arreglo **dirs** debe contener el valor 1, por estar el nodo hacia el norte del nodo central.

Las direcciones en las que se ven los nodos vecinos $a_{i+1,j}$, $a_{i,j-1}$ y $a_{i-1,j}$ desde el nodo central (i,j) son respectivamente: este, sur y oeste, lo cual implica que las posiciones dentro del arreglo **norb2** son: **norb2(actin,east)**, **norb2(actin,sur)** y **norb2(actin,west)**. Los valores que se deben almacenar en cada caso en la siguiente entrada del arreglo **dirs** son 2, 3 y 4. Por último, el usuario debe almacenar en la variable **indice** el número de entradas que guardo en el arreglo **cols**. Una técnica adecuada para manejar el valor de la variable **indice** consiste en incrementar su valor cada vez que se deposita un valor en el arreglo **coef**, ya que en las rutinas que calculan los valores de frontera, se desconoce el valor inicial de la variable **indice**, aunque en las rutinas **diffinmX** el valor de **indice** es cero sería deseable que el usuario usara esta técnica.

La información necesaria para la discretización de las condiciones de frontera es proporcionada en los arreglos **froi**, **fh**, **froi2** y **fh2** y en los parámetros **dir** y **tipo** de las rutinas **frontmX**.

Para poder expresar los valores de las condiciones de frontera, es necesario darse cuenta que en la misma rutina hay que expresar los valores para todos los tipos de fronteras que se definieron y que se usaron en la descripción del dominio. Si para alguno de estos tipos he han definido dos direcciones desde las cuales se le puede ver, se debe calcular el valor de frontera para cada dirección y el programa debe manejar las cuatro direcciones posibles que le van a aparecer, y no producir ningún tipo de error si se le pide que entrege un valor de frontera para cualquier combinación tipo-dirección que se le pida. Es oportuno aclarar

que los números que se almacenan en el arreglo **dirs** al momento de hacer la discretización de la ecuación en el paso anterior, son los números que aparecen en el parámetro **dir** de las rutinas **frontmX**.

Una vez que se ha explicado que para cada convinación tipo-dir se debe expresar el valor de la frontera, se puede proceder a explicar como se aplican la forma explícita e implícita de calcular los valores de la frontera.

Para expresar el valor de las condiciones de frontera con el método explícito, se cuenta dentro de la rutina con los parámetros x, y, que contienen el valor de las coordenadas del punto de la frontera. Estos valores pueden ser usados para calcular el valor de la función en la frontera —en caso de que se necesite— . Si el valor es constante o bien ha sido calculado, se debe almacenar en la variable vfront, que es a su vez un parámetro de la rutina **frontmX**. En las rutinas **frontmX** aparece otro parámetro llamado **coefi**, que es el coeficiente que multiplica al valor de la frontera en la ecuación lineal, dentro de las rutinas **frontmX no se debe multiplicar el valor de la frontera por coefi, esto lo hace el programa después de llamar a las rutina frontmX.**

Para expresar los valores de las condiciones de frontera haciendo uso del método implícito el sistema provee al usuario con los arreglos y variables siguientes: **froi, fh, cuantos, froi2, fh2** y **cuantos2** que contienen respentivamente los índices de los nodos vecinos al *nodo frontera* en la dirección correspondiente, las distancias entre los nodos y el número de nodos vecinos que aparecen almacenados en el arreglo **froi**. Lo mismo es cierto para el segundo juego de **arreglos**.

El usuario debe almacenar el polinomio que expresa el valor de la frontera en función de los nodos vecinos correspondientes en los arreglo **coef, cols** y **dirs**, incrementando tantas veces como sean necesarias el valor de la variable **indice** para que refleje correctamente el número de entradas que se le han agregado a los arreglos, si por alguna razón aparece un término que se deba poner en el lado derecho de la ecuación sin ser modificado de ninguna forma, este término se debe de sumar o restar a la variable **ld** —en este momento la variable **ld** contiene el valor del lado derecho que será usado en la solución del sistema de ecuaciones—. Si en algo se necesita, es necesario aclarar que la variable **colsi** contiene el índice correspondiente al nodo de frontera con el que se está trabajando.

La forma de expresar los valores de los coeficientes para el polinomio de frontera es la misma que en caso de la discretización de la ecuación diferencial, solo se debe recordar que en este caso es necesario multiplicar estos coeficientes por el valor **coefi**, porqué este valor de la frontera está siendo sustituido dentro de una ecuación y este valor tiene en la ecuación un valor que lo multiplica.

El usuario debe agregar toda la lógica que sea necesaria dentro de las rutinas **diffinmX** y **frontmX** asi como todas las rutinas adicionales que requiera para implartar el control que sobre su problema necesite. Este programa implanta únicamente el control de flujo para gobernar la creación del sistema de ecuaciones a partir de la malla discretizada y el cálculo de la solución, esta implantación es totalmente modular i.e. que las rutinas que hacen este trabajo

pueden ser llamada varias veces con parámetros diferentes para resolver diferentes ecuaciones siempre sobre una misma discretización, pero el usuario deberá programar la lógica de control. El usuario debe programas una rutina **frontmX** y **diffinmX** para cada EDP de un sistema acoplado de ecuaciones y darles los nombres **frontm1**, **frontm2** ... y **diffinm1**, **diffinm2** ... . En el capítulo dedicado a la discución de como se resuelve la ecuación de Navier-Stokes se explicará la forma de usar la difrentes rutinas que están implantadas dentro del sistema.

V.3 Generación del sistema de ecuaciones algebraicas.

Una vez que el usuario ha programado la rutinas que necesita para manejar el control de flujo de su problema específico, y las rutinas que expresan la discretización de la EDP y las condiciones de frontera, las debe *ligar* con el resto del sistema —esta operación puede ser diferente para diferentes máquinas—, y ejecutar el programa respondiendo a todas las preguntas que el sistema le hará.

Después de que el sistema ha sido arrancado, inicializa todas sus variables de control, lee la descripción de la malla y comienza a generar el primer sistema de ecuaciones que se le pida. Para realizar este proceso se fija en el reglón i-ésimo —empieza con el primer renglón— de la matriz **norb2** y llama a la rutina **diffinmX** para conocer los valores de los coeficientes de la discretización para el nodo i-ésimo —la discretización de la ecuación diferencial fue almacenada en un *stack* por facilidad—. En seguida comineza a revisar el tipo de cada nodo que ha sido metido en el stack desde abajo hacia arriba. Si el tipo del nodo j-ésimo es normal lo ignora, si el tipo de nodo es de frontera, el sistema manda llamar a la rutina **frontmX** para conocer el valor de frontera ya sea de forma explícita o implícita. Si la frontera fue expresada de froma implícita, los coeficientes de la discretización son trasladados hasta arriba del stack, como se explicó en la sección referente a la discretización de las fronteras.

La siguiente opción revisa si el nodo que sigue en el stack es de interpolación, si ese es el caso, el programa expande este nodo en función de la definición que para el nodo de interpolación aparece al final de la matriz **desarr** depositando las definiciones del *nodo de interpolación* hasta arriba del stack. Esta expanción de *nodos de interpolación* se realiza por niveles, tomando en cuenta que un *nodo de interpolación* solo puede estar definido en función de nodos de menor nivel. Se expanden primero los nodos de niveles más altos y después los de niveles más bajos. Esta es la regla principal que se debe tomar en cuenta para expander nodos de interpolación, y no importa si las líneas sobre las que están definidas las interpolaciones no son las mismas, basta con que se cumpla la regla de niveles para que la interpolación funcione adecuadamente.

Este proceso se reliza hasta que no quedan *nodos de frontera* o *nodos de interpolación* en el stack, para cada una de las entradas válidas del arreglo **norb2**.

Después de que cada ecuacion ha sido calculada, se inserta en los arreglos que se usarán como entrada de las rutinas que pueden resolver los sistemas de ecuaciones, la estructura de datos que se usa para estos arreglos se explicara en la siguiente sección. Si el lector esta iteresado en entender como se construye la estructura de datos a partir del stack en donde esta guardado el renglón i-ésimo de la discretización despu'es de haber sido expandido, puede consultar la rutina **push** del programa **ireg.for**, esta rutina es realmente clara y no vale la pena explicarla.

V.4 Interface con el método numérico que resuelve el sistema algebraico de ecuaciones.

Para explicar la interface con el método numérico, basta con explicar la forma como se guardan los sistemas de ecuaciones en memoria y decir que cualquier rutina que se trate de usar para resolver el problema deberá usar esta estructura.

En el almacenamiento del sistema de ecuaciones se usan los arreglos reales (double precision) **b**, **an** y enteros **ia**, **ja**. El arreglo **b** contiene el lado derecho del sistema de ecuaciones. En los otros tres arreglos se almacenan únicamente los elementos diferentes de cero de la matriz. El arreglo **ia** contiene la posición dentro de los arreglos **an** y **ja** en donde empieza cada nuevo renglón de la matriz, y el arreglo **ja** contiene el número de columna al que pertenece cada elemento del arreglo **an**, i.e. si en la entrada j-ésima del arreglo **an** se depositó el **coef(i)**, en la entrada j-ésima del arreglo **ja** se debe depositar el contenido de la entrada **cols(i)**. En caso de duda se puede consultar la rutina **push** del programa **ireg.for**, esta rutina es llamada una vez cada vez que se ha terminado de desarrollar un renglón del sistema de ecuaciones.

La rutina que resuelva el sistema lineal de ecuaciones debe aceptar los parámetros **tol** que contiene la tolerancia con la que hay que resolver el problema, **maxi** contiene el número maximo de iteraciones que se deben ejecutar antes de parar, en caso de que se use un método iterativo. También se debe regresar el número de iteraciones efectuadas en la variable **iters**, y un valor que indique como se desempeño el algoritmo en la variable **res**, si el algoritmo trabajo sin problemas se debe regresar un 0, si tuvo cualquier problema menor el valor regresado debe ser menor que cero y finalmente si no pudo concluir el trabajo por alguna razón el valor debe ser mayor que uno. Este último valor se despliega en la tabla de resultados para indicar que paso y si es mayor que cero, para el procedimiento. El usuario puede querer pasar más parámetros a la rutina para controlar su funcionamiento, en ese caso él debe modificar adecuadamente el procedimiento **solve** del programa **ireg.for**, el procedimiento **solve** solamente es llamado en la rutina **mainalgo** que el usuario escribe. En el siguiente capítulo se mostrarán todas la rutinas de las que el usuario dispone para programar un problema, asi como las que debe programar obligatoriamente.

V.5 Resolución del sistema de ecuaciones lineales.

Para encontrar la solución del sistema de ecuaciones lineales, se usa el algoritmo **SOR** [11,6]. Solamente hay que modificarlo para que funcione adecuadamente con la estructura de datos que se esta usando. El el caso particular de la ecuación de Navier-Stokes se usan los parámetros de relajación de 1.6 para la ecuación de continuidad y 1.1 para la ecuación de la vorticidad [18]. Se observa el mismo tipo de comportamiento en la convergencia del método **SOR** que el reportado por [18].

Capítulo VI

Una aplicación: Solución a la ecuación de Navier-Stokes.

Haciendo uso del sistema descrito a lo largo de los capítulos anteriores, se resolvió la ecuación de Navier-Stokes. Durante el proceso, se definieron varios dominios diferentes, y se programó un método iterativo para resolver la EDP haciendo uso de las fácilidades del sistema. Esta es la parte del del trabajo que será descrita a continuación.

La ecuación de Navier-Stokes expresada en términos de la vorticidad está formada por dos ecuaciones diferenciales parciales acopladas a través de la condición de frontera de la ecuación de la vorticidad, la cual, se expresa en términos del Laplaciano de la función de corriente. Esta situación obliga a que se resuelva en primer lugar la ecuación de continuidad, luego la ecuación de la vorticidad calculando las condiciones de frontera con el Laplaciano de la función de corriente, para luego, resolver de nuevo la ecuación de continuidad con el nuevo valor de la vorticidad ... . En el momento cuando este proceso ha calculado una solución para la ecuación de continuidad, que satisface la condición física en las fronteras que corresponde al tipo de flujo que se desea, es entonces el momento de parar la iteración, aunque no sean exactos los valores de la condición de frontera, ya que el método numérico no será capaz de calcular con mayor precisión los valores de las soluciones debido al error de discretización. Esta condición de paro se refleja en el hecho de que los valores de la vorticidad dejan de cambiar en menos que el valor al cuadrado del paso de la malla y esta es la condición de paro que se debe usar. En la práctica se observa que se puede ser menos estricto con la condición de paro, el resultado neto es que el número de cifras significativas de la solución se reduce —si se mantienen las tolerancias para la iteraciones locales, si las tolerancias locales

son reducidas, se compromete la convergencia del algoritmo global—.

Si se analiza el método iterativo del sistema de ecuaciones parciales haciendo uso de un sistema de ecuaciones ordinarias que imita algunas propiedades básica del sistema parcial, se podra observar que el sistema de ordinarias no converge, sino que es periódico. Esta situación se manifiesta para la ecuación parcial en el hecho de que hay que amortiguar los valores de la vorticidad y de las condiciones de frontera de la vorticidad en las iteraciones globales, para lograr que el algoritmo converja [8,1,22,?]. Si estos valores no son amortiguados adecuadamente, se observara que el sistema diverge, no muy rápidamente pero con vigor y en algunos casos las iteraciones locales dejan de converger para los valores antes citados de los parámetros de relajación del **SOR** [1].

A lo largo de este capítulo, se mostrará la técnica que se usó para resolver la ecuación de Navier-Stokes. Desde luego la presentación no corresponde con el desarrollo histórico del proceso, sino con una forma comprensible para describirlo —como de costumbre cuando se explican desarrollos matemáticos y algoritmos, las cosas se hacen al revés de como se presentan—.

VI.1 Descripción del método numérico empleado para resolver la ecuación de Navier-Stokes.

La ecuación de Navier-Stokes se resuleve con un método iterativo que consiste en adivinar la primera vez una solución para la vorticidad, sustituirla en la ecuación de la continuidad y resolverla, tomar la solución de la función de corriente para calcular las condiciones de frontera de la ecuación de la vorticidad, relajar las condiciones de frontera, sustituirlas en la ecuación y resolverla, la nueva vorticidad se debe relajar antes de sustituirla en la ecuación de la continuidad, y asi sucesivamente. A este proceso de le llama "iteración global", dentro de esta iteración existen dos iteraciones locales; las que corresponden a la busqueda de la solución de cada sistema de ecuaciones lineales —método **SOR**—. En la programación del proceso es necesario incluir todos los parámetros para controlar todas y cada una de las iteraciones.

[1] Cuando ésto ocurre, es señal de que algo anda mal con alguna parte del proceso, parámetros de amortiguamiento, condiciones de frontera etc., porque los valores de relajación del SOR son característicos del problema, por lo tanto son constantes para la ecuación de Navier-Stokes.

La iteración global puede ser expresada de la siguiente forma:

i. Se propone ω_0.

ii. Se hace $i = 1$.

iii. Se resuleve la ecuación de continuidad igualada con ω_{i-1}, y se encuentra ψ_i.

iv. Se calculan las condiciones de frontera $\hat{\omega}_i = f(\psi_i)$.

v. Se relajan las condiciones de frontera $\tilde{\omega}_i = (1 - \lambda_f)\tilde{\omega}_{i-1} + \lambda_f \hat{\omega}_i$.

vi. Se resuelve la ecuación de la vorticidad haciendo uso de las condiciones de frontera $\tilde{\omega}_i$, y se encuentra $\breve{\omega}_i$

vii. Se relaja el valor de la vorticidad $\omega_i = (1 - \lambda_\omega)\omega_{i-1} + \breve{\omega}_i$.

viii. Se verifica si $\|\omega_i - \omega_{i-1}\|_\infty < \epsilon$ (donde ϵ es la tolerancia prefijada que depende del ancho de malla) si es cierto el algoritmo termino; si no, se incrementa en 1 la i, y se prosigue la ejecución a partir del paso iii.

Los pasos iii y vi corresponden a las iteraciones locales y es ahi en donde se emplea el algoritmo **SOR.**

Como se puede observar, es necesario mantener en memoria siempre las dos últimas realizaciones de los valores de la vorticidad y de las condiciones de frontera para poder llevar a cabo la iteración global —por facilidad en la programación es conveniente mantener también en memoria las dos últimas realizaciones de la función de corriente—. Las formas de la iteración y de la ecuación de la vorticidad obligan a generar en cada vuelta la ecuación de la vorticidad, mientras que la ecuación de la continuidad puede ser generada al principio del proceso junto con su primer lado derecho como una ecuación de Laplace. En cada paso del ciclo global se puede sumar el valor de la vorticidad al lado derecho de la ecuación de Laplace y asi obtener a cada paso la ecuación de la continuidad, para hacer más rápido el proceso.

Estas características obligan a que el sistema pueda generar de forma independiente y en un número ilimitado de veces cada una de las ecuaciones —siempre sobre la misma malla—. También debe ser capaz de calcular los valores de las fronteras idependientemente de la forma como fueron definidas, i.e. en el caso de que la definición de una frontera sea explícita, no hay ningún problema, si la frontera fué definida en términos de algunos nodos vecinos de la misma función incognita, los valores de interés pueden ser calculados únicamente después de haber resuelto el sistema de ecuaciones, calculando y evaluando el polinomio que los define. Si los valores fronterizos se deben relajar es necesario almacenarlos relajados en cada una de la iteraciones y disponer de toda la información necesaria en el momento oportuno.

También debe ser posible inicializar los valores de las distintas incognitas y de las condiciones de frontera, para poder comenzar el algoritmo de solución con las soluciones iniciales adecuadas.

VI.2 Programación del Algoritmo de solución de la ecuacion de Navier-Stokes.

Esta sección está dedicada a la programación del problema de Navier-Stokes se presentan las rutinas que el sistema ofrece y se describen las rutinas que se programaron.

VI.2.1 Interface del sistema con el usuario.

El sistema provee al usuario con un conjunto de rutinas que realizan las operaciones de creación del sistema de ecuaciones, solución del sistema de ecuaciones, calculo y almacenamiento de las condiciones de frontera y nodos de interpolación, calculo del residuo del sistema de ecuaciones lineales e impresión de la solución y de los valores de frontera que se usaron, esta última información se guarda para poder realizar la graficación de la solución obtenida.

Los nombres de las rutinas empleadas son respectivamente: **create, solve, llenavals, errmul y printsol.** A continuación se describen los parámetros que usan y su función.

En primer lugar se tiene a la rutina **create.** Esta rutina como ya se ha dicho antes, se encarga de la generación del sistema de ecuaciones lineales que corresponde a la forma discretizada de una ecuación diferencial parcial sobre la malla que se le haya especificado al sistema.

La declaración de la rutina create es la siguiente:

```
subroutine create(an, ia, ja, b, x1, x2, problema)
integer ia(1), ja(1), problema
double precision b(1), x1(1), x2(1), an(1).
```

Los parámetros **an, ia, ja** y **b** contendrán a salir de la rutina la matriz del sistema de ecuaciones lineales, su organización corresponde a la de la estructura de datos antes mencionada.

Los parámetros **x1** y **x2** no son modificados durante la llamada a la rutina **create.**

En el capítulo V, página 60, se mencionó que existen las rutinas **diffinmX** y **frontmX**, eso es cierto, pero falta mencionar que se debe escribir una rutina de cada tipo para cada ecuación parcial que se desee trabajar. En el nombre de la rutina la **X** se debe reemplazar por un número secuencial que numera a cada EDP [2], este número identificará que problema se desea resolver, esta información es comunicada a la rutina que crea el sistema de ecuaciones haciendo uso del parámetro llamado **problema.**

[2] Si se desean trabajar más de dos ecuaciones diferenciales parciales, es necesario agregar las llamadas pertinentes en las rutinas **genrow** para **diffinmX** y **desarrolla** para **frontmX** del programa **ireg.** En el programa que trabaja con la ecuación de Navier-Stokes existen las llamadas para cuatro rutinas **diffinm1, diffinm2, frontm1** y **frontm2** que corresponden a la ecuación de continuidad y de la vorticidad.

El segundo lugar le corresponde a la rutina que se encarga de resolver el sistema de ecuaciones lineales. La declaración de la rutina se muestra a continuación:

```
      subroutine solve(an, ia, ja, b, x1, x2, metodo, tol,
     *                 imax, w, res, iters, etime, k)
      integer ia(1), ja(1)
      double precision an(1), b(1), x1(1), x2(1),
      integer metodo, imax, res, iters, k
      double precision tol, w, etime
```

Los parámetros **an, ia, ja y b** contienen el sistema de ecuaciones lineales que fue creado al momento de ejecutar la rutina **create**, en estas dos rutinas se deben emplear los mismos arreglos para generar y resolver el sistema de ecuaciones, aunque en el procedimiento pueden existir varios juegos de arreglos para manejar diferentes ecuaciones.

El arreglo **x1** es un arreglo de trabajo, la información que queda almacenada al momento de terminar la ejecución de la rutina **solve** no tiene porqué tener algún significado.

En el arreglo **x2** se deposita la solución que se encontró.

Durante la depuración del código que resulve la ecuación de Navier-Stokes, fue necesario resolver los sitemas de ecuaciones con diferentes algoritmos, por esa razón se implantó la posibilidad de tener varios métodos numéricos para resolver las ecuaciones lineales.

Existen 4 métodos que se pueden usar:

[3] Método **SOR**. Para este método el parámetro de relajación se almacena en la variable **w**, la tolerancia del error en la variable **tol**, el número maximo de iteraciones en la variable **imax**, la variable **k** no se usa y la rutina regresa en la variable **res** un número que indica como se desarrollo el proceso de solución, el número de iteraciones efectuado en la variable **iters** y finalmente el tiempo que le tomo resolver el sistema de ecuaciones en la vartiable **etime** [3].

[4] Método **CGNE**, es un gradiente conjugado con la ecuaciones normales. En este caso únicamente es necesario establecer cual es la tolerancia con la que se desea resolver el sistema de ecuaciones en la variable **tol**, al final la rutina entrega el número de iteraciones empleadas en la variable **iters**. No regresa ningún valor en la variable **res**.

[5] método Orthomin [23][4]. En este punto únicamente es necesario aclarar que

[3] Si la rutina que resuleve el sistema de ecuaciones necesita conocer la dimención de la matriz, esta información esta alnacenada en la variable *nornodes* que aparece en un common de la rutina **solve** —nornodes corresponde con número de nodos normales—.

[4] Se desea agradecer al M. en C. José Luis Morales Perez, por haber proporcionado los programas que implantan los algoritnos Orthomin.

la variable **k** contiene el número de vectores que se desean guardar para el método orthomin.

[6] método Orthomin con precondicionamiento. Para el método con precondicionamiento se almacena el número de vectores que se emplean en la busqueda de la solución en la variable **k**, y el parámetro de relajación del precondicionamiento en la variable **w**.

La tercer rutina implanta el cálculo y almacenamiento de los valores de las condiciones de frontera y de los valores de los nodos de interpolación, la declaración de esta rutina es realmente simple, como se puede observar:

```
subroutine llenavals(problema)
integer problema
```

El parámetro indica únicamente para que ecuación se deben calcular y almacenar los valores de las fronteras y nodos de interpolación. El funcionamiento correcto de esta rutina depende de algunas rutinas que el usuario debe programar, más adelante se detallaran las rutinas necesarias.

El cuarto lugar lo ocupa la rutina encargada de calcular el residuo de una solución numérica del sistema de ecuaciones, esta rutina se le proporcionan una matriz, una solución y un lado derecho, multiplica la matriz por la solución, y lo resta del lado derecho. El error que reporta corresponde con la $\|\cdot\|\infty$ del vector residuo. El resultado es devuelto como el valor de la función. La declaración va como sigue:

```
double precision function errmul(an, ia, ja, b, x)
integer ia(1), ja(1)
double precision b(1), x(1), an(1)
```

los parámetros **an**, **ia**, **ja** y **b** contienen el sistema de ecuaciones lineales y el vector **x** la solución que multiplicará a la matriz.

El último lugar lo ocupa la rutina **printsol** que se encarga de imprimir la solución encontrada con el formato posición x, posición y y valor de la función $f(x,y)$. La declaración tiene la forma siguiente:

```
subroutine printsol(arch, x, problema)
integer arch, problema
double precision x(1)
```

la variable **arch** debe contener el número de un archivo arbierto en el cual se escribirán los valores de la función y sus coordenadas, **x** contiene el vector solución que se desea imprimir y **problema** se refiere al problema al que corresponde la solución **x**, éste parámetro se emplea únicamente para encontrar los valores de las condiciones de frontera.

Estas son todas las rutinas que el sistema provee para realizar el trabajo, cubren las funciones básicas de discretizar, resolver una ecuación y de probar e imprimir la solución. No es un conjunto de rutinas autosuficientes, se necesita que el usuario escriba las rutinas que completan los procesos de inicialización y descripción del algoritmo numérico que resuelve la ecuación que se está trabajando.

VI.2.2 Rutinas que el usuario debe programar.

Para comenzar esta sección, es necesario explicar la estructura del programa, para que el usuario pueda entender como debe programar las rutinas que almacenan los valores de las condiciones de frontera y de los *nodos de interpolación*, asi como la estructura de datos con la que se maneja el sistema de ecuaciones lineales y los vectores que almacenan la solución.

Se ha mencionado en el capítulo V, que la definición de los *nodos normales* esta almacenada en la matriz **norb2** y que la definición de los *nodos de frontera* y *nodos de interpolación* esta almacenada en el arreglo **desarr**, la estructura de estos arreglos es la misma que la estructura de los archivos de salida del compilador de mallas.

Los sistemas de ecuaciones se almacenan en la estructura de datos formada por los arreglos **an**, **ia**, **ja** y **b** que ya se ha explicado con anterioridad. Las soluciones del sistema de ecuaciones deben ser manejadas en dos arreglos uno de trabajo, **x1** [5] y el otro que almacena la solución del sistema de ecuaciones, **x2** [6]. Esta estructura de datos se debe declarar [7] tantas veces como sistemas de ecuaciones o EDP se desee trabajar simultaneamente, el arreglo de trabajo **x1** puede ser reutilizado o sea declarado una única vez.

En el caso de la ecuación de N-S se debe tener dos arreglos como **x2** — obviamente con nombres diferentes— para almacenar los valores de la solución actual y de la solución anterior. Una manera eficiente de realizar este trabajo consiste en usar un apuntador sobre un único arreglo lineal que contiene los dos vectores que se necesitan y en cada iteración cambiar el valor del apuntador para reflejar el hecho de que lo que era la solución anterior desaparese y el espacio de usa para la nueva solución, y de que lo que era la solución actual pasa a ser la solución anterior en la nueva iteración.

La forma de almacenar los valores de las fronteras y de los nodos de interpolación depende únicamente de la forma como el usuario la programe. En el caso de la ecuación de N-S, se usa una matriz que tiene tantos renglones como *nodos*

[5]Este primer arreglo lo usa el algoritmo **SOR**, su dimensión y contenido final quedan determinados por el uso que le de el algoritmo de solución, como ejemplo, con el algoritmo **CGNE**, el arreglo **x1** no se usa.

[6]Las dimensiones de los arreglos estan especificadas en la declaración **Parameter** de todos los programas **FORTRAN** y en cada una de las subroutinas que los usan. Con la declaración actual se pueden resolver sistemas de hasta 15000 ecuaciones.

[7]En cada declaración se debe declarar cada uno de los arreglos con las dimensiones correspondientes y nombres diferentes.

de frontera y *nodos de interpolación* hay en en modelo y tiene ocho columnas, se ha mencionado antes que la rutina **guardaval** recibe como parámetros de entrada los valores **x**, **nodo**, **dir** y **problema**, el primero es el valor que le corresponde al nodo **nodo** del problema **problema** y visto desde la dirección **dir**. Entonces lo que la rutina debe hacer, es usar los valores **dir** y **problema** para calcular en que columna del renglón **nodo** se debe almacenar el valor **x**. Y el proceso de recuperar el valor de **x** dados los valores de **nodo**, **dir** y **problema** lo debe realizar la rutina **dameval**. Este proceso se emplea indistintamente para los *nodos de interpolación* que para los *nodos de frontera*, en el caso de los *nodos de interpolación*, el valor del parámetro **dir** se hace siempre igual a 1 y se emplea para calcular la columna en donde se almacena el valor del *nodo de interpolación*, esto se puede hacer porque el concepto de dirección no existe para un *nodo de interpolación*.

La rutina **iniguarda** únicamente inicializa a ceros la matriz en donde se almacenan los valores de la fronteras y de los *nodos de interpolación*, lo cual implica que el primer valor que se usa en el cálculo de las condiciones de frontera para la ecuación de la vorticidad es cero.

En el proceso durante el cual el sistema inicializa sus variables, el sistena llama a tres rutinas que el usuario le ha proporcionado, **ininco**g, **iniguarda** y **inipar**. La primer rutina inicializa los valores de la o las soluciones iniciales, en sus parámetros tiene contenida toda la información necesaria para calcular los valores iniciales de cada entrada de los vectores que almacenan las soluciones. La segunda se encarga de inicializar los arreglos que se usan para almacenar los valores de las condiciones de frontera, únicamente debe cumplir con los lineamientos que se han explicado antes. La rutina **inipar** inicializa todos los parámetros que el usuario necesite para especificar su ecuación diferencial.

Después de llamar las dos últimas rutinas mencionadas, el sistema llama a la rutina mainalgo que contiene la programación del algoritmo que resuelve la EDP.

A continuación resta presentar las declaraciones de cada una de la rutinas mencionadas [8].

INIGUARDA
```
subroutine iniguarda
```

GUARDAVAL
```
subroutine guardaval(x, nodo, dir, problema)
 double precision x
 integer nodo, dir, problema
```

DAMEVAL
```
double precision function dameval(nodo, dir,
problema)
integer nodo, dir, problema
```

INIPAR
```
subroutine inipar
```

[8] Las impreisones completas de todas las rutinas estan en los ápendices.

ININCOG

```
      subroutine inincog(x, y, i)
C     (x,y) posicion del nodo i
      double precision x, y
      integer i
```

La siguiente rutina que se discute es **mainalgo** [9], la declaración es muy simple y es como sigue:

MAINALGO

```
      subroutine mainalgo
```

Esta rutina implanta la iteración global antes definida.

Para lograr este objetivo, tiene la necesidad de declarar una matriz llamada **memval**, donde almacena los valores de los *nodos de frontera* y *nodos de interpolación*, usa el algoritmo antes decrito para almacenar y recuperar la información. Adem'as declara dos juegos de arreglos para almacenar los sistemas de ecuaciones lineales, el primer juego **pan, pb1, pb2, nia** y **nja** almacena la discretización de la ecuación de continuidad, el arreglo **pb1** se usa para almacenar el lado derecho de la ecuación de Laplace —que es la ecuación de poison, pero con $\omega = \bar{0}$—. Durante el proceso se suman **pb1** y ω calculada para obtener **pb2** que es el lado derecho con el que se debe resolver el sistema de ecuaciones.

El sistema de ecuaciones que pertenece a la ecuación de la vorticidad, se almacena en los arreglos **nan, nb, pja** y **pia**, en este caso es necesario generar el sistema de ecuaciones cada vez que se va a resolver, raz'on por la cual no se puede reutilizar ninguna parte de la información para acelerar el proceso como se hizó en el caso de la solución de la ecuación de continuidad.

En la rutina **mainalgo**, se declaran dos variables de tipo entero **offpro** y **off1**, estos son los apuntadores que se usan para implantar el proceso iterativo sobre las soluciones de los dos sistemas de ecuaciones. Las cuatro soluciones estan almacenados en dos arreglos **phi** y **omega**, estos arreglos son dos veces mas grandes que los arreglos que almacenan los lados derechos de los sistemas lineales, ya que cada uno tiene espacio para dos soluciones. Las rutinas que calculan los valores de las condiciones de frontera, las discretizaciones y las que almacenan los valores de frontera y de los nodos de interpolación, necesitan conocer a los apuntadores para que puedan recuperar los valores adecuados a cada momento. El usuario debe garantizar que la forma como se manejan los apuntadores permitira recuperar la información correctamente. La rutina que inicializa el primer valor de las soluciones, lo debe almacenar en las dos secciones de cada arreglo —**phi** y **omega**—. El primer valor que toma **offpro** es 1 y **off1** es **NMAX**. Para calcular la primera solución a la ecuación de Laplace, se usa el apuntador **off1** para accesar el arreglo **omega**, lo que equivale a ver la **omega** de la iteración anterior —el lector debe recordar que es necesario iniciar el algoritmo de alguna forma—. Cuando se genera el sistema de ecuaciones de

[9] Si el lector desea analizar con detalle la programación de la rutina **mainalgo**, debe revisar el programa correspondiente.

la ecuación de la vorticidad, se usa **offpro** para mirar dentro del arreglo que duarda a la ψ y tambien se usa **offpro** para accesar el arreglo que guardara a la ω. Cuando se ha terminado una iteración a la variable **offpro** se le da el valor de **NMAX** + **1** y a la variable **off1** se le da el valor 0 y el ciclo se repite. Como los valores de las dos variables han sido invertidos, el resultado dentro del algoritmo es que se simulo el proceso de avanzar el índice de la iteración.

Ahora es muy fácil decir que hace la rutina **mainalgo** [10]:

i. Inicialización de las variables que el usuario define.

ii. Creación del sistema de ecuaciones lineales de la ecuación de Laplace.

iii. Cálculo del lado derecho de la ecuación de Poisson e inicialización del vector de trabajo para el método **SOR**.

iv. Cálculo de la solución de la ecuación de Poisson.

v. Verificación de la solución —se calcula el residuo de la solución—, almacenamiento de los valores de la forntera y de los nodos de interpolación.

vi. Relajación de la ψ —en el caso de que se necesite—.

vii. Creaciaón de la ecuación de la vorticidad.

viii. Inicialización del arreglo de trabajo de la segunda ecuación parcial.

ix. Cálculo de la solución de la segunda ecuación parcial.

x. Almacenamiento de los valores de frontera e interpolación para la vorticidad.

xi. Relajación de la ω.

xii. Cálculo del error en la iteración global, con la rutina **pnorma**, la cual se explica a continuación.

xiii. Si el error global es menor que la tolerancia pedida, se imprime la solución y es el fin del algoritmo, de lo contrario se intercambian los valores **offpro** y **off1** y se repite el proceso desde el paso iii.

En este algoritmo no se menciona la relajación de las fronteras, porqué los valores de las fronteras se relajan durante el proceso de generación del sistema de ecuaciones lineales asociado a la segunda ecuación parcial.

La rutina discutida anteriormente hace uso de la rutina **pnorma** que calcula la $\|\cdot\|_\infty$ del residuo de los vectores ψ y ω entre la iteración anterior y la actual, su declaración es la siguiente:

[10] Unicamente se mencionan los paso importantes de la implantación, los pasos de perapación y despiegle de información se obvian.

PNORMA

```
double precision function pnorma(n, phi,
*phi1, omega, omega1)
integer n, i
double precision phi(1), phi1(1)
double precision omega(1), omega1(1), aux,
*aux1
```

Finalmente algunos comentarios sobre las rutinas que discretizán las condiciones de frontera y las EDP. Los encabezados de las declaraciones se presentan a continuación:

FRONTMX

```
subroutine frontm1(x, y, coefi, colsi,
*indice,tipo,cols,coef,dirs,dir,
*ld,vfront)
double precision x, y, coefi,
*coef(1), ld, vfront
integer indice, tipo, cols(1),
*dir, colsi, dirs(1)
```

DIFFINMX

```
subroutine diffinm1(indice, cols,
*coef, dirs, ld)
double precision coef(1), ld
integer cols(1), indice, dirs(1)
```

Estas rutinas calculan las condiciones de frontera y las discretizaciones de las ecuaciones parciales, las reglas para programarlas ya han sido discutidas, y falta aclarar que en caso de la EDP de Navier-Stokes, no se deben amortiguar las esquinas en donde existe una discontinuidad para la vorticidad, si esto de hace, el problema no converge, si no se amortiguan, el sistema hace un poco lo que quiere con los valores de las esquinas y se las arregla para que converja el problema.

El usuario no debe llamar ninguna de las rutinas que el sistema llama durante su inicialización, si lo hace puede generar problemas en las estructuras de datos y no permitir que el sistema trabaje adecuadamente. Si el usuario escribe rutinas que el sistema desconoce que existen, las puede llamar libremente, siempre y cuando respete la estructura del sistema.

VI.3 Resultados para la ecuación de Navier-Stokes.

Una vez que se logró poner a punto el programa que es capaz de resolver la ecuación de Navier-Stokes, y gracias al generador de mallas, que permite definir diferentes dominios con fácilidad, se pudieron resolver diferentes problemas,

haciendo uso de la misma EDP. En esta sección se presentan algunos de los resultados obtenidos.

Los programas fueron probados contra la solución exacta en un tubo sin obtáculos en el caso de flujo cortante (Couette) y flujo parabólico (Pouseuille). En los casos en los que no se dispone de solución exacta, las soluciones numéricas fueron calculadas para mallas de diferente paso, observandose un comportamiento consistente.

A continuación pasamos a describir en detalle los resultados obtenidos para varias geometrías.

VI.3.1 El tubo con un obstáculo.

El método utilizado fue el de diferencias centradas. El paso de la malla $\frac{1}{30}$. El número de Reynolds es 10. Los parámetros de relajación son: para ω 0.2; para la frontera 0.2. La solución tiene dos cifras significativas.

Se observa la formación de un vórtice detrás del obstáculo y una rápida recuperación del flujo a su valor no perturbado. Se observa la línea de corriente libre claramente, figura VI.1.

En la figura VI.2, se aumentó el número de Reynolds a 30. Se observa el crecimiento del vórtice y el corrimiento de la línea de corriente libre hacia la esquina del objeto tal y como es de esperarse en estas geometrías.

VI.3.2 El tubo con dos obstáculos.

Las figuras VI.3, VI.4 y VI.5 son experimentos para un tubo con dos obstáculos y flujo de Pouseuille a la entrada. El paso de malla es nuevamente de $\frac{1}{30}$ y los parámetros de relajación son de 0.15 y 0.2 respectivamente. Se observa la formación de vórtices detrás de ambos obstáculos, los cuales crecen al aumentar el número de Reynolds. No se alcanza a observar la fusión de los vórtices ni tampoco hay vorticidad convectada aguas abajo.

VI.3.3 La válvula coronaria abierta.

El modelo fisiológico se aproxima por un tubo de paredes rígidas y la válvula también es rígida. La idea es la de simular el flujo cuando la válvula esta abierta. La hipótesis es que el flujo se reajusta al estado estacionario en una escala de tiempo mucho más corta que la del movimiento de la válvula. Este es un hecho experimental. Para facilitar el cálculo, se buscan soluciones simétricas. Es muy posible sobre todo para números de Reynolds altos que aparecieran soluciones no simétricas. Estas se descartan en el presente cálculo. Los resultados se presentan en las figuras VI.6 a la VI.11.

El método empleado es una discretización con upwinding corregido a segundo orden para mallas regulares. El paso de la malla fue de $\frac{1}{30}$. Se observa para

número de Reynolds de 50 la formación de dos vórtices y el flujo se recupera a su valor uniforme rápidamente.

En la figura VI.7 se observa un nuevo vórtice en la pared de arriba. Este se forma por conservación de momento angular, ya que la vorticidad es cero en la línea de simetría.

Al aumentar el número de Reynolds se observa el crecimiento esperado de la misma configuración. Ver las figuras VI.7 a VI.9.

Las figuras VI.9 a VI.11 muestran el crecimiento del vórtice pegado a la pared. Estas indican la necesidad de tener una región de integración más grande ya que en este caso el flujo no se recupera dentro de la región estudiada. Esto no se exploró en el presente trabajo.

VI.3.4 El tubo con cavidad, la historia de dos vórtices.

Un problema de gran interés ha sido el de estudiar la interacción de dos vórtices que al amalgamarse forman una estructura diferente. Un ejemplo de ésta situación se da al estudiar el flujo a lo largo de un tubo con una cavidad. Puede esto pensarse como una primera aproximación al patron de flujo en un aneurisma. La hipótesis es que el aneurisma ha sido formado de alguna manera y que este se comporta como una pared rígida. De hecho estos cálculos no representan la evolución del aneurisma que es un problema dinámico en el cual la elasticidad de las paredes juega un papel fundamental.

Los calculos que se exhiben representan la evolución del flujo en una geometría fija de paredes rígidas para diferentes números de Reynolds.

Las figuras VI.12 a la VI.32 representan la evolución del flujo a medida que el número de Reynolds crece.

Los experimentos se llevaron a cabo con el método de upwinding corregido y un paso de malla de $\frac{1}{30}$ y parámetros de relajación son respectivamente 0.15 y 0.2.

Las figuras VI.12 a VI.23 muestran el crecimiento asimétrico de los vórtices que se forman en las esquinas.

En las figuras VI.24 a VI.26 se ve la fusión de los dos vórtices en el resto de las figuras se observa como el vórtice ocupa toda la cavidad y como para números de Reynolds altos el centro del vórtice es convectado por el flujo. Las figuras VI.27 a VI.32 muestran este proceso.

VI.3.5 Miscelánea de resultados.

Las figuras VI.33 a VI.35 muestran la evolución de un vórtice en una cavidad para un fluido conductor en un campo magnético perpendicular a la cavidad. Observese que el campo magnético impide el crecimiento de los vórtices.

Las figuras VI.36 a VI.45 modelan el flujo en una cavidad cubierta por un rodillo con velocidad preescrita. Observese la formación de tres vórtices al inicio y la posterior fusión de dos de ellos. El proceso puede repetirse para números

de Reynolds más altos esperandose una configuración con más y más vórtices que se generan en las esquinas, y posteriormente se amalgaman.

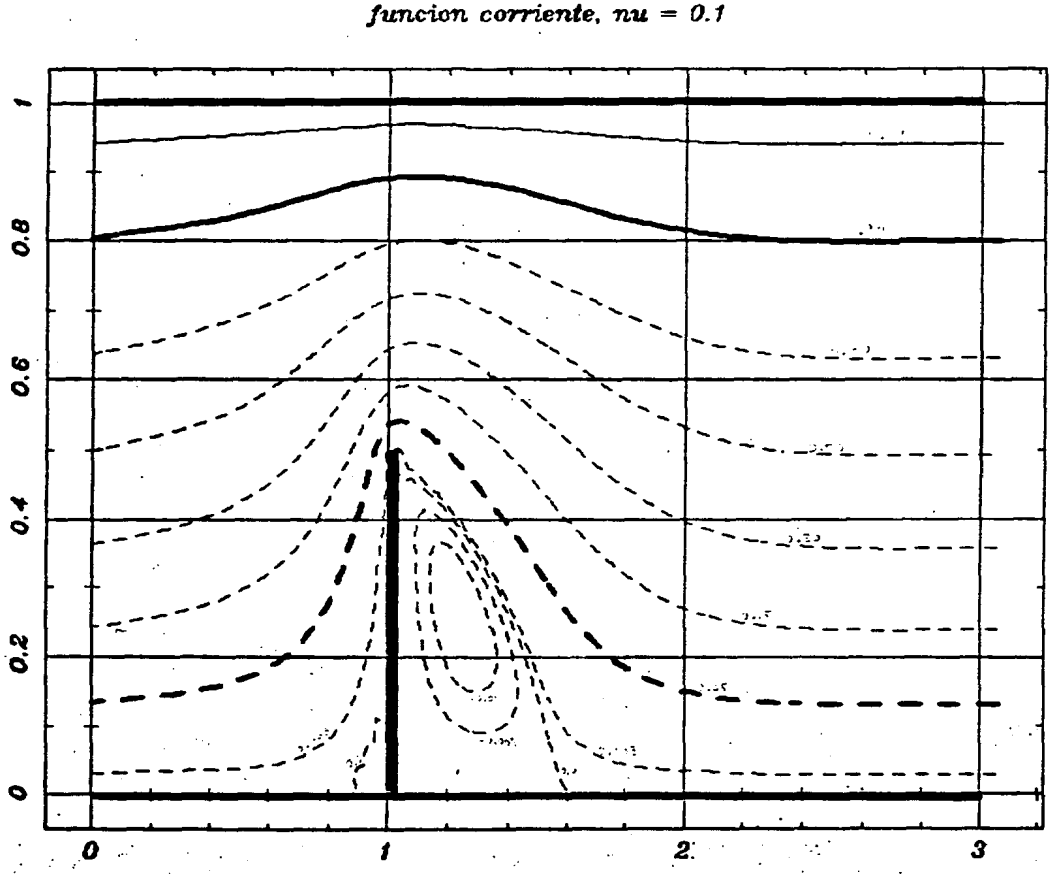


Figura VI.1

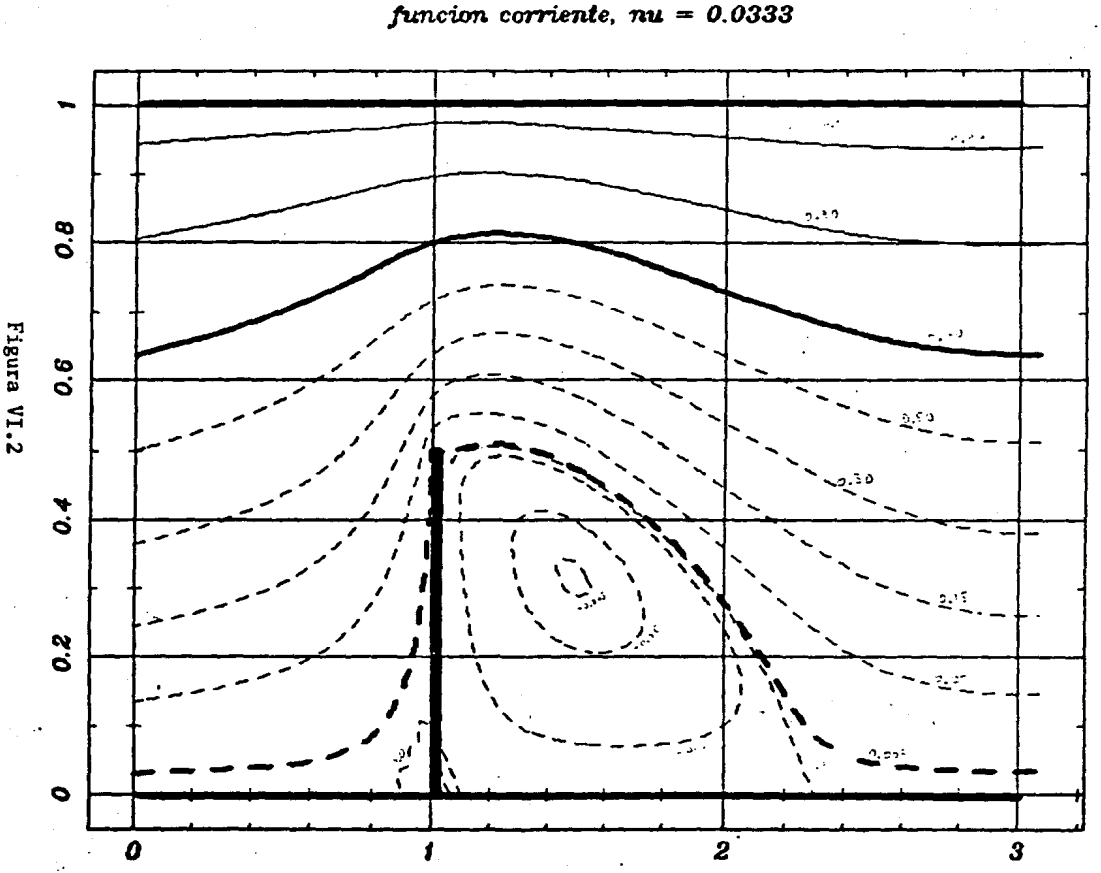


Figura VI.2

funcion corriente, Re = 10

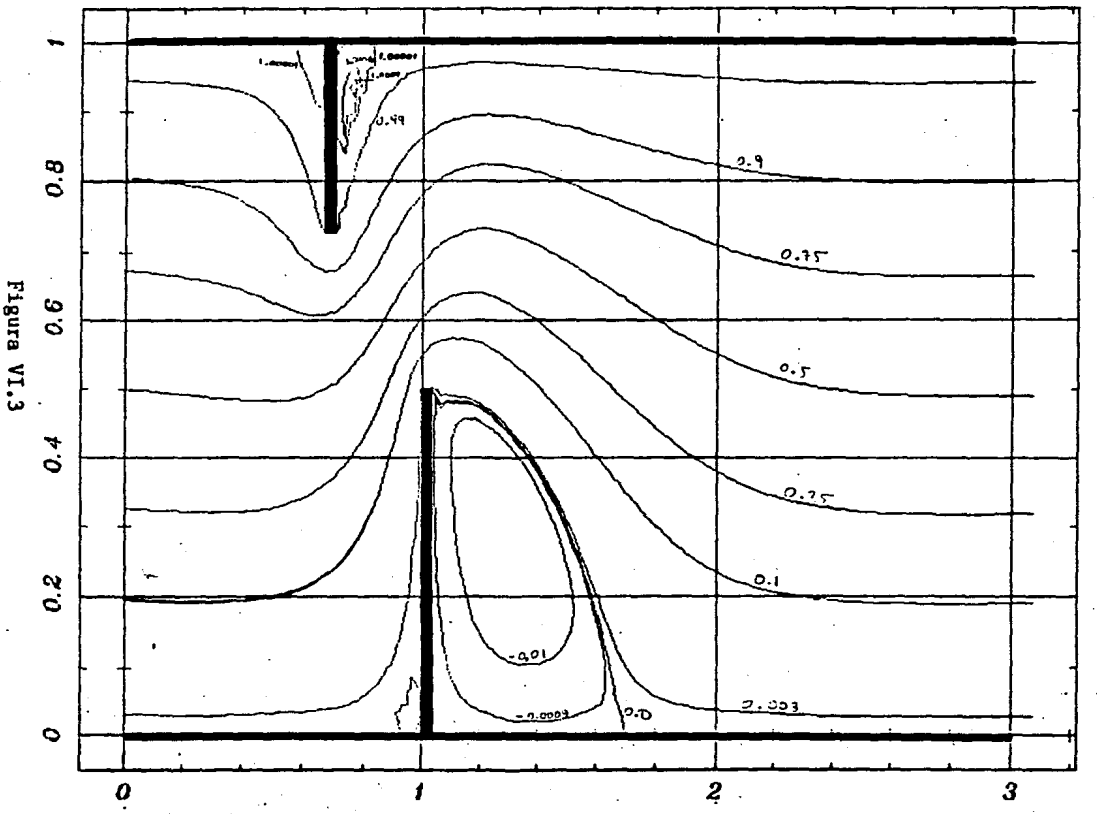


Figura VI.3

funcion corriente, Re = 20

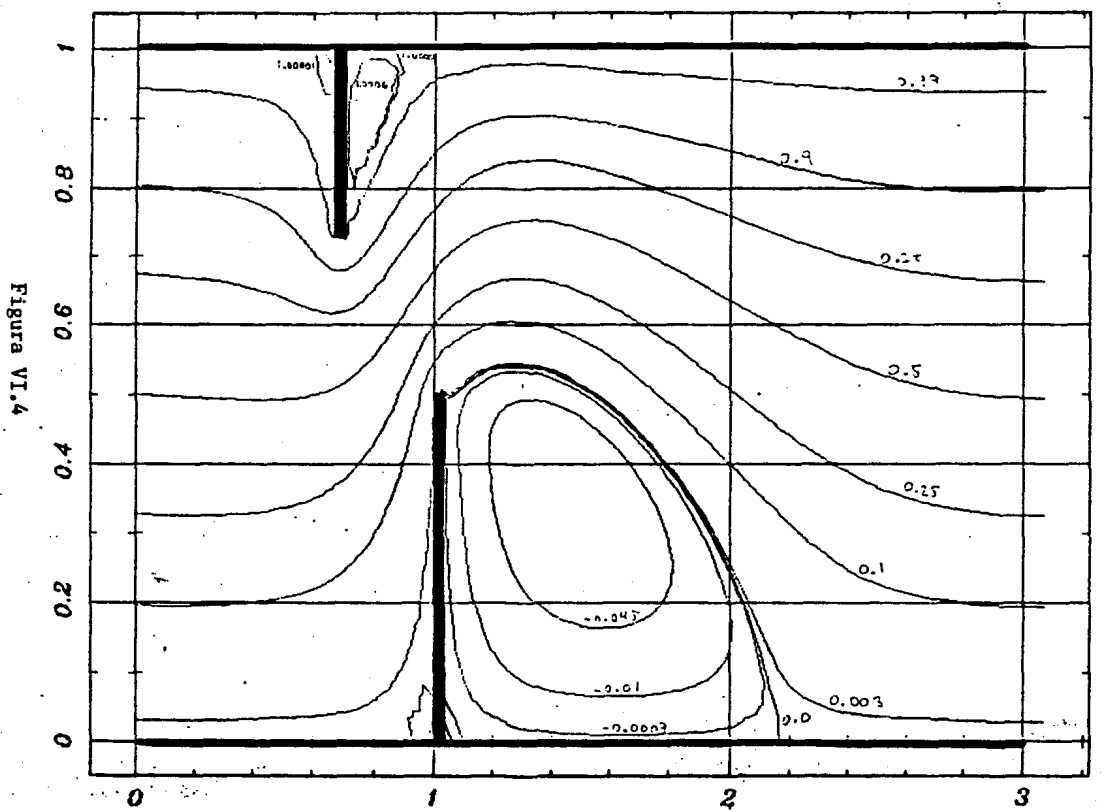


Figura VI.4

funcion corriente, Re = 30

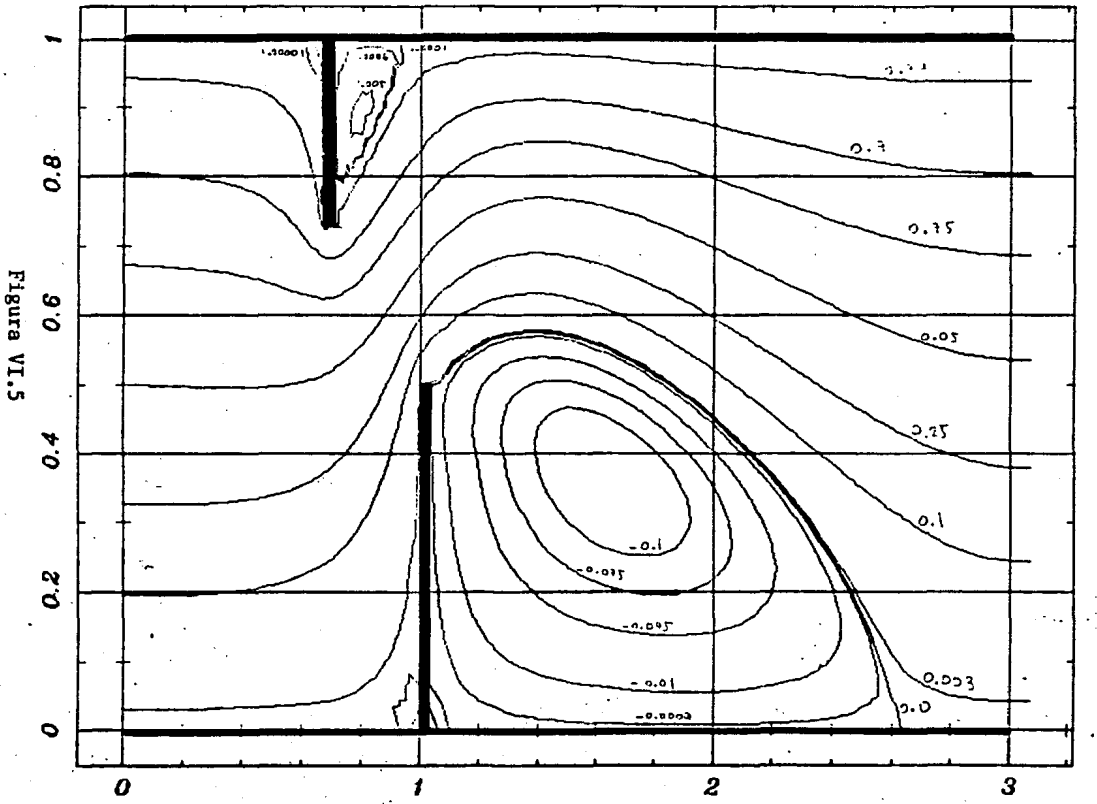


Figura VI.5

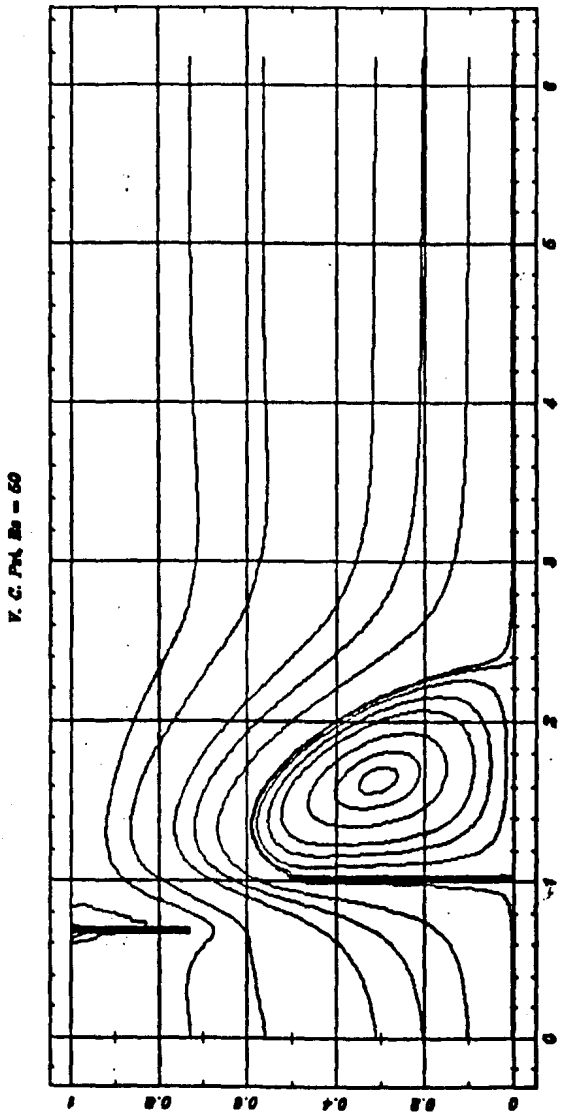


Figura VI.6

Figura VI.7

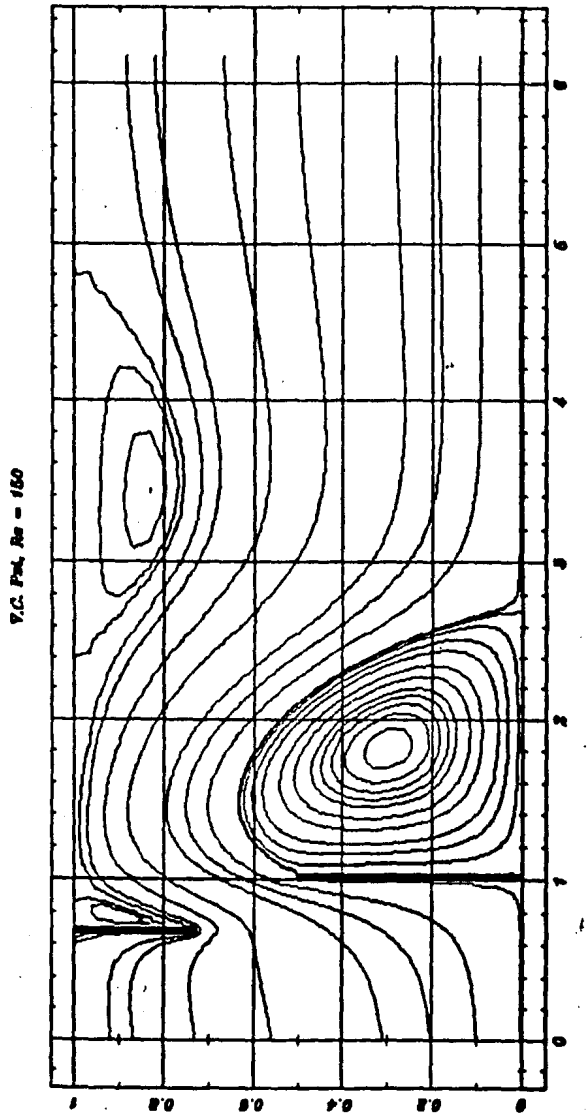


Figura VI.8

Figura VI.9

Figure VI.10

Figura VI.11

Figure VI.12

Figura VI.13

Figura VI.14

Figura VI.15

Figura VI.16

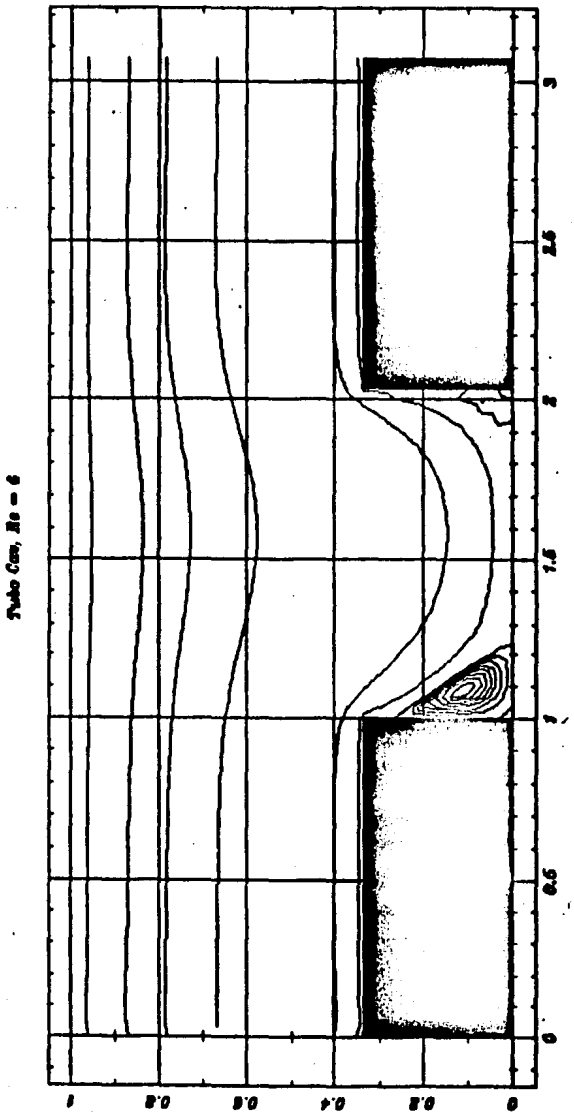


Figura VI.17

Figura VI.18

Figura VI.19

Figura VI.20

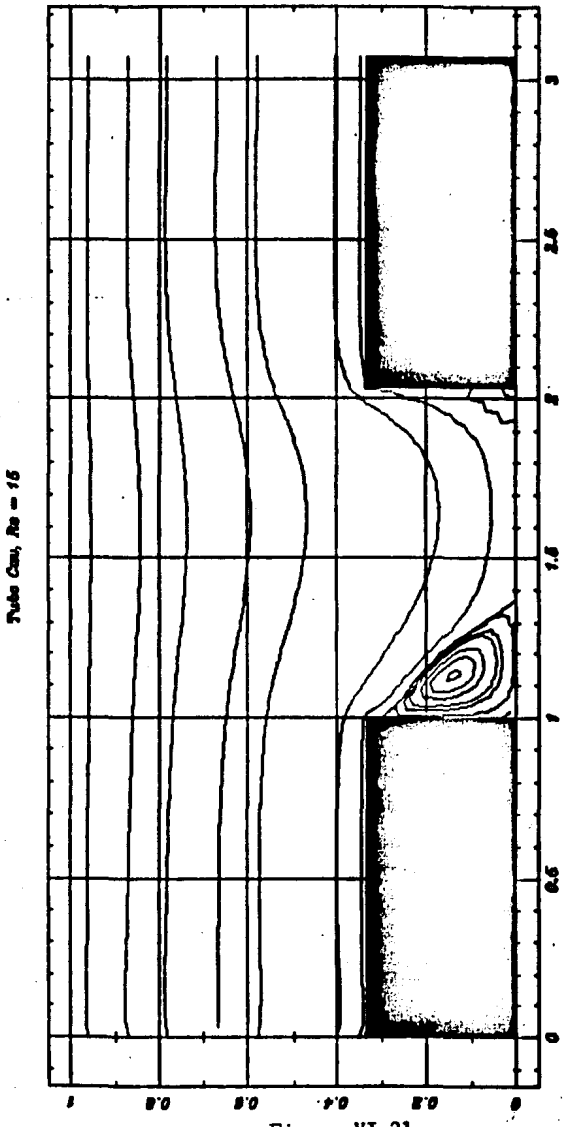


Figura VI.21

Figure VI.22

Figura VI.23

Figura VI.24

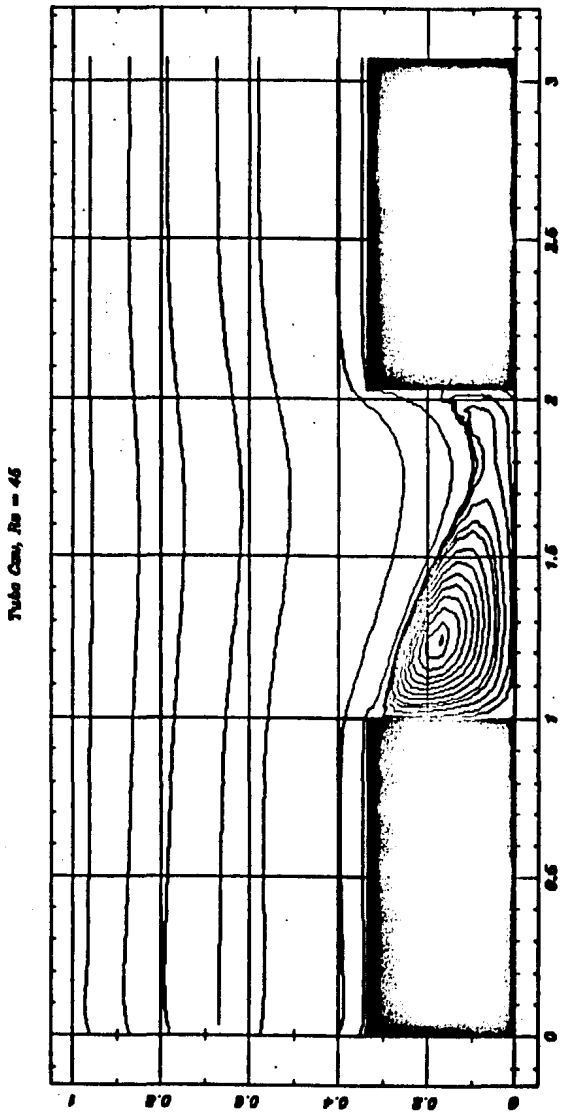


Figura VI.25

Figura VI.26

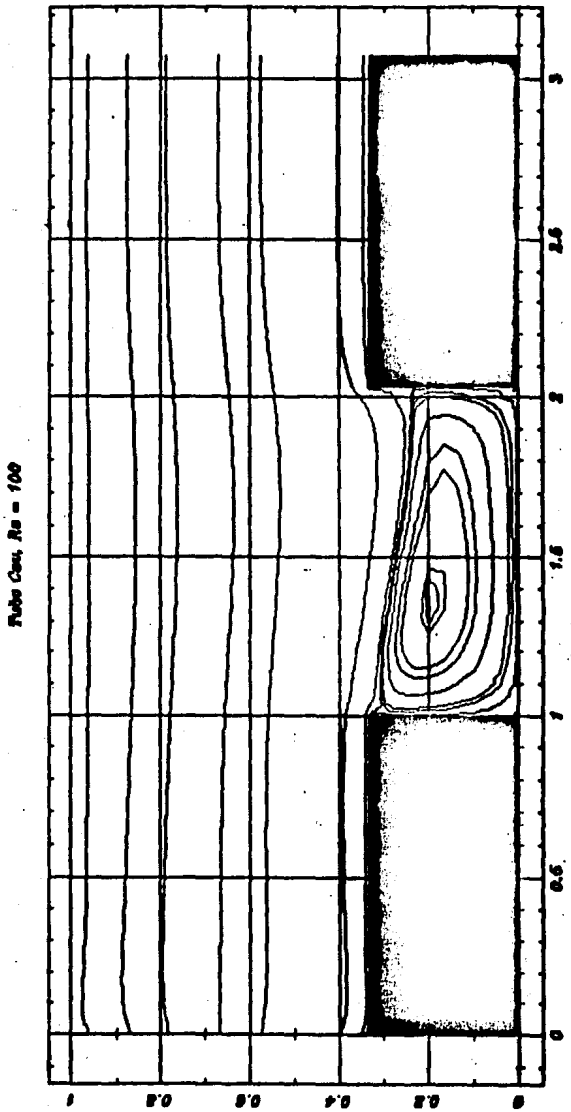


Figura VI.27

Figura VI.28

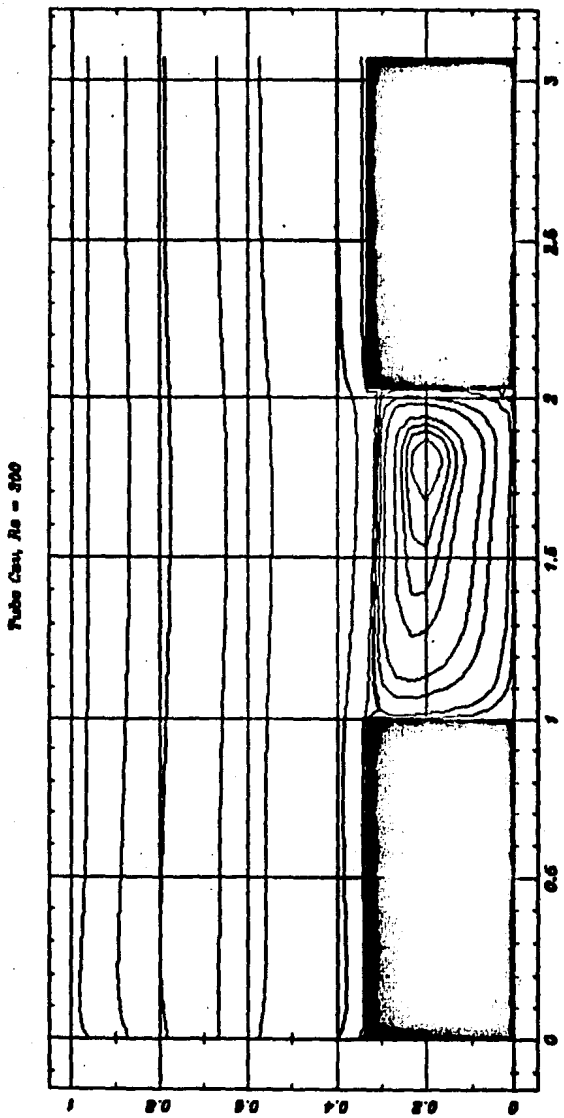


Figura VI.29

Figura VI.30

Figura VI.31

Figure VI.32

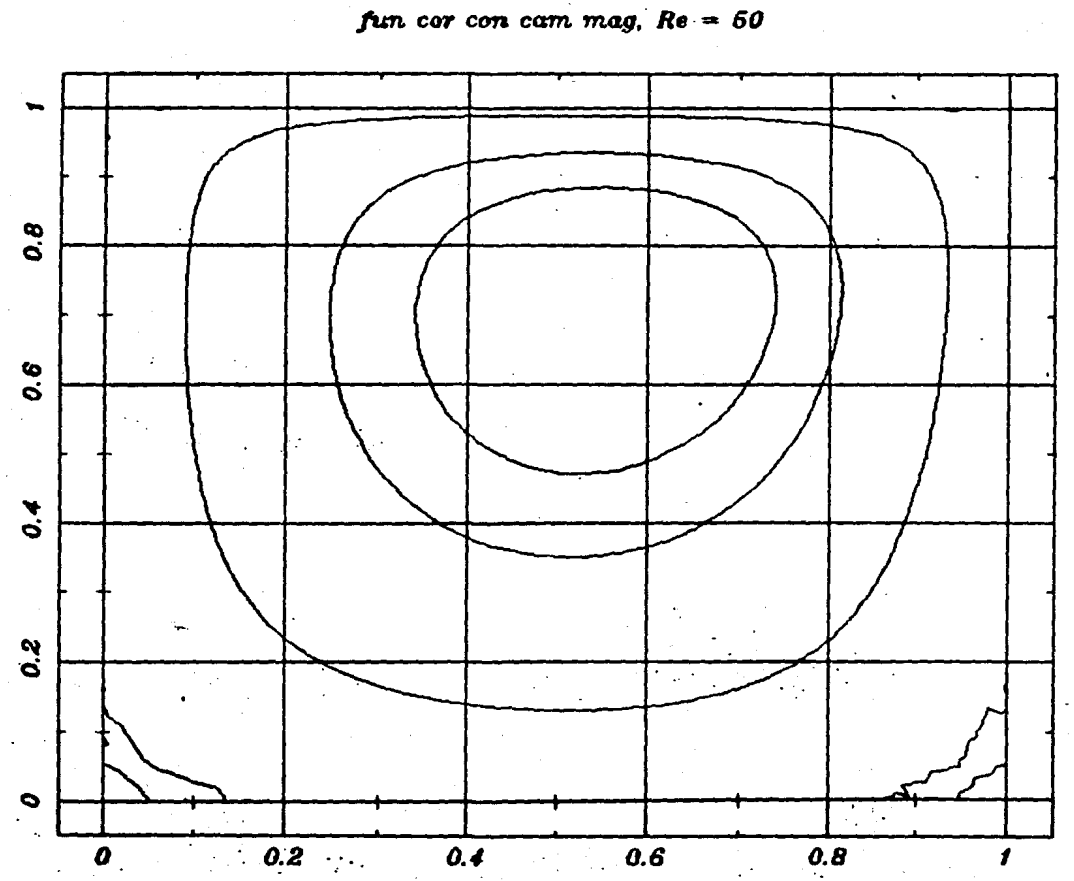


Figura VI.33

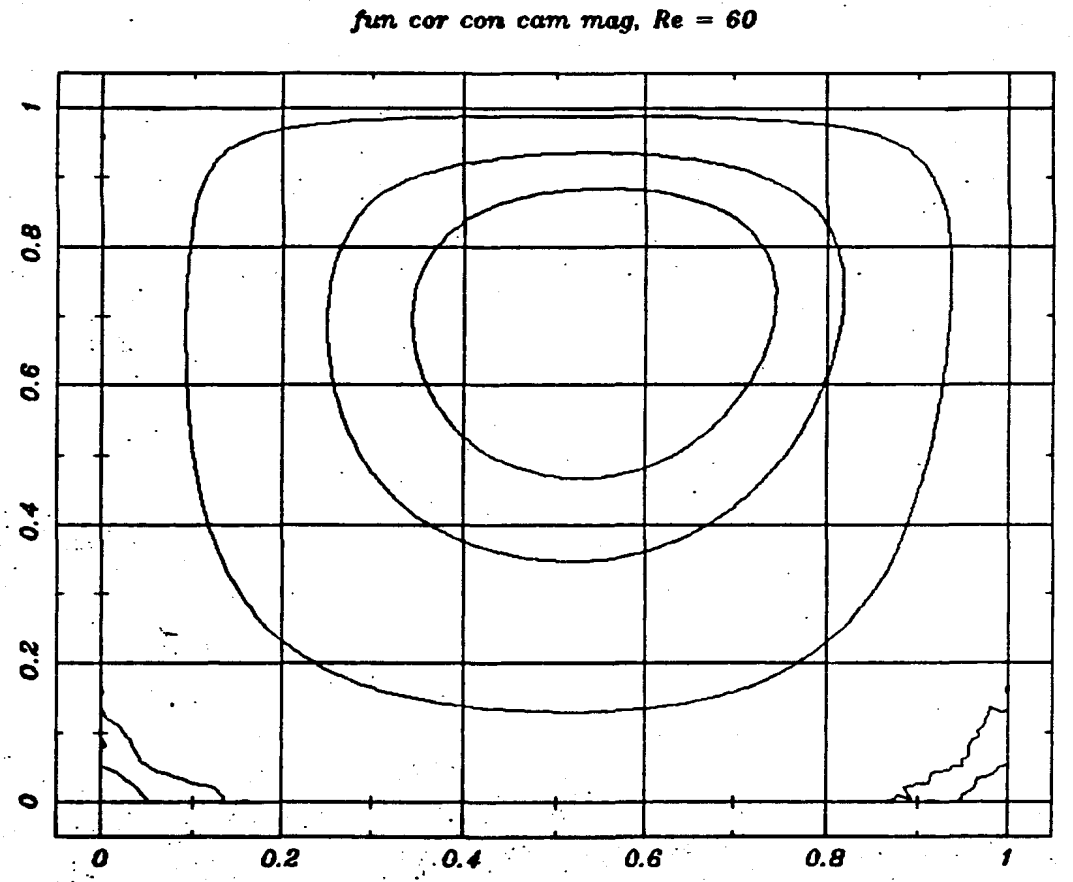


Figura VI.34

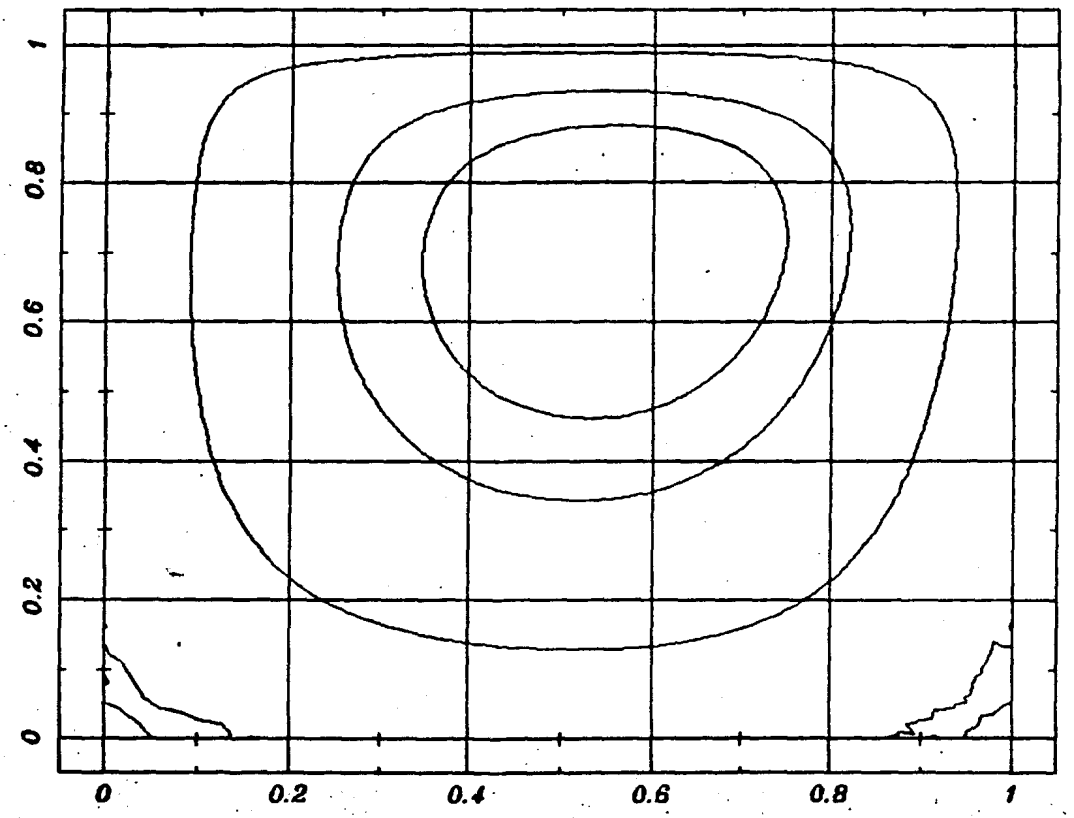


Figura VI.35

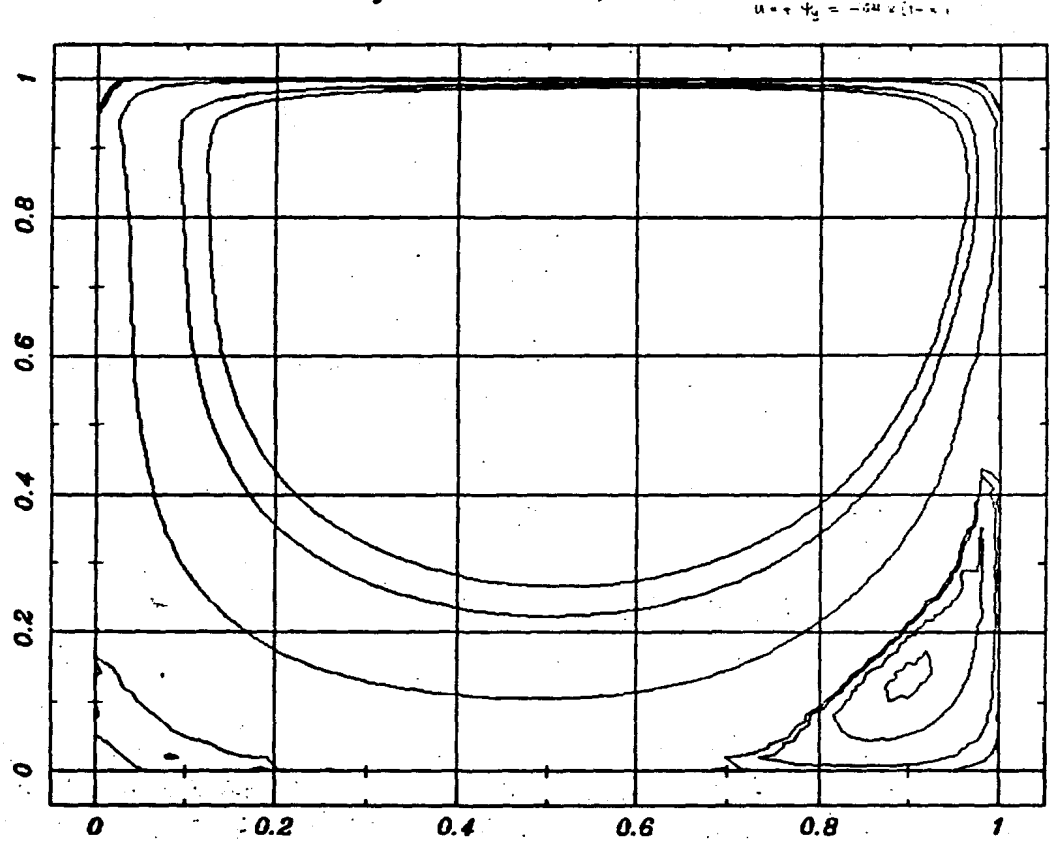


Figura VI.36

funcion corriente, Re = 60

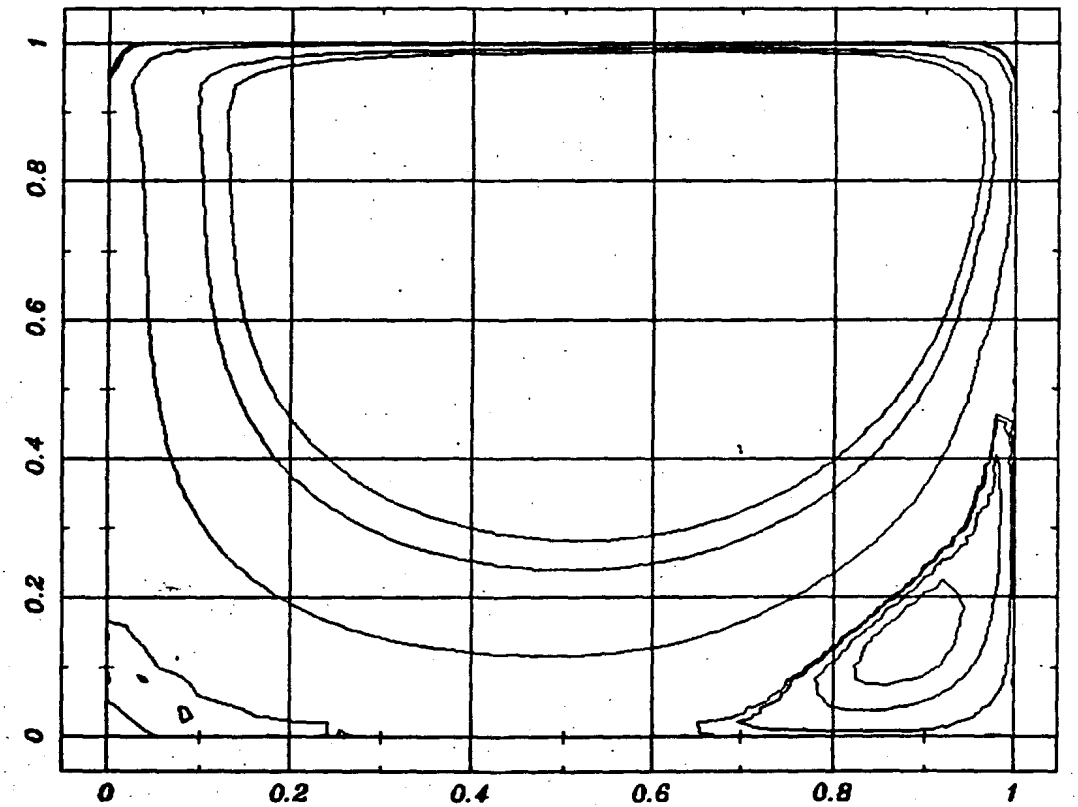


Figura VI.37

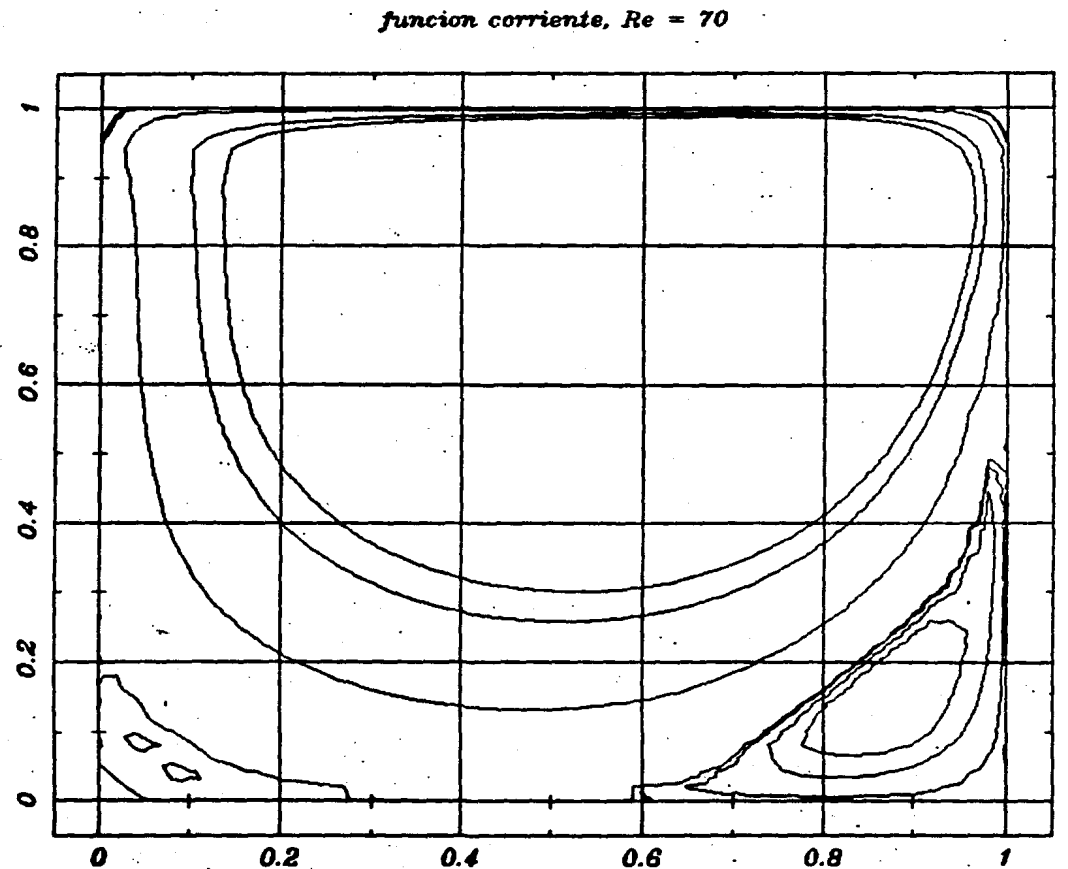


Figura VI.38

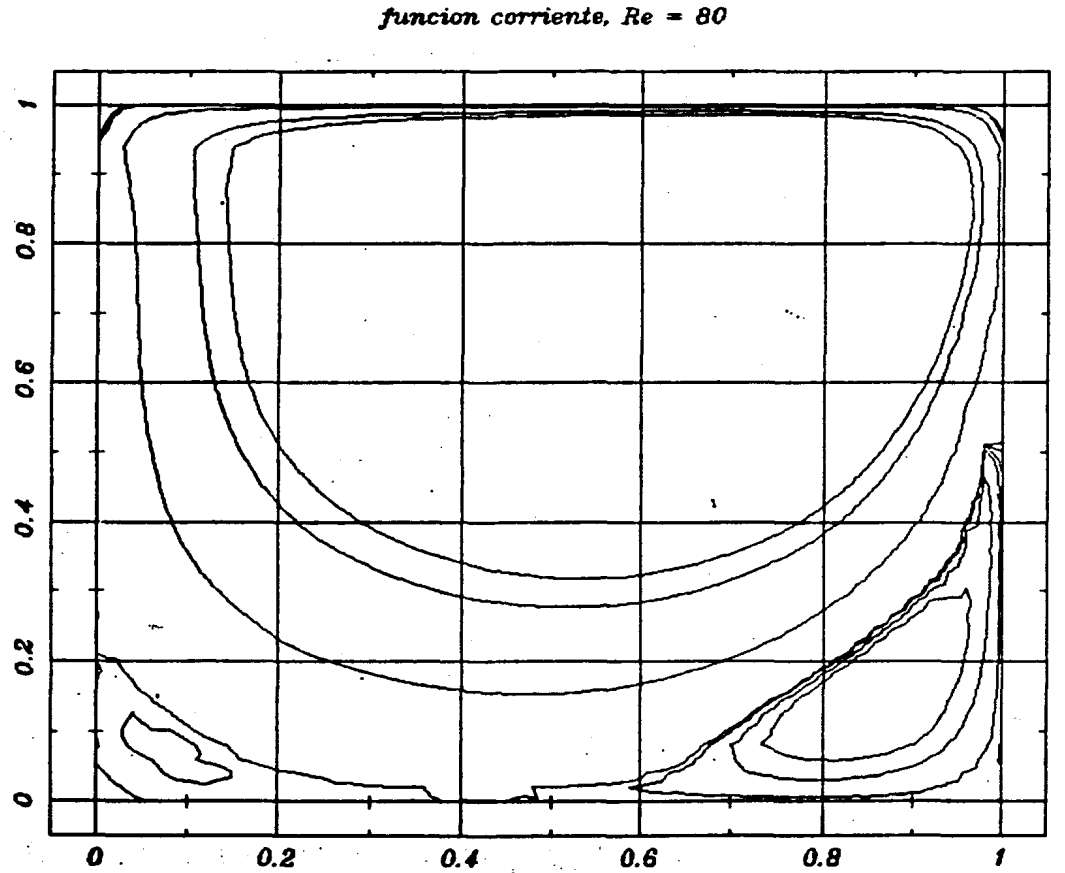


Figura VI.39

funcion corriente, Re = 90

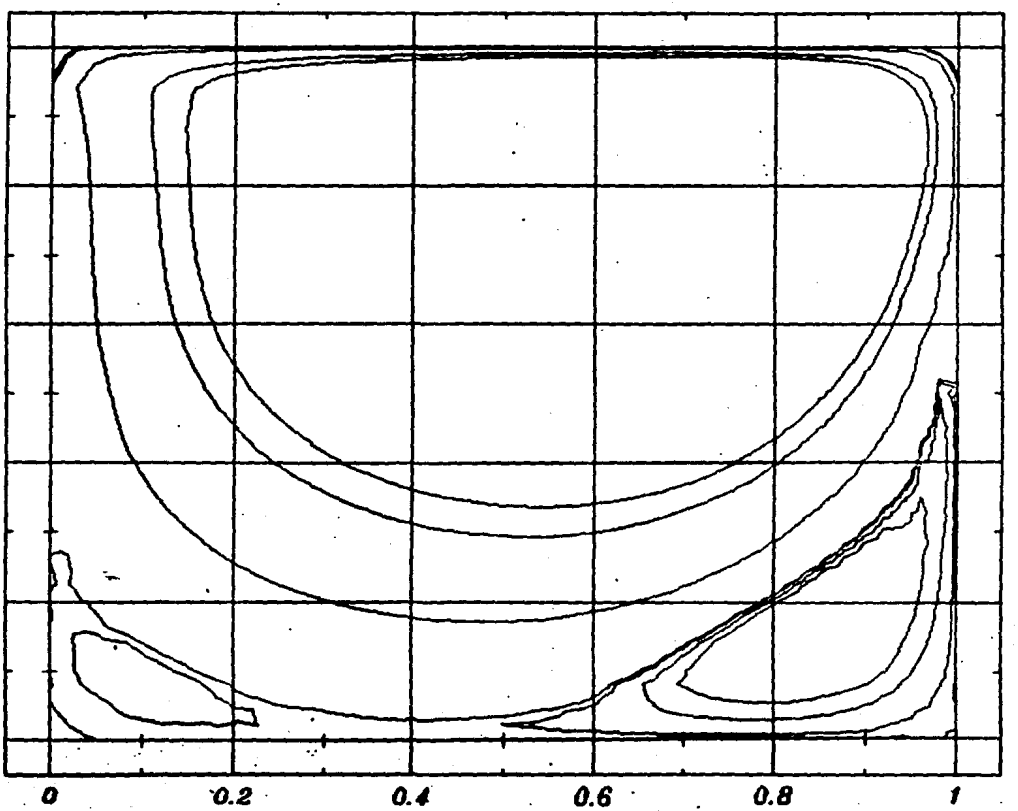


Figura VI.40

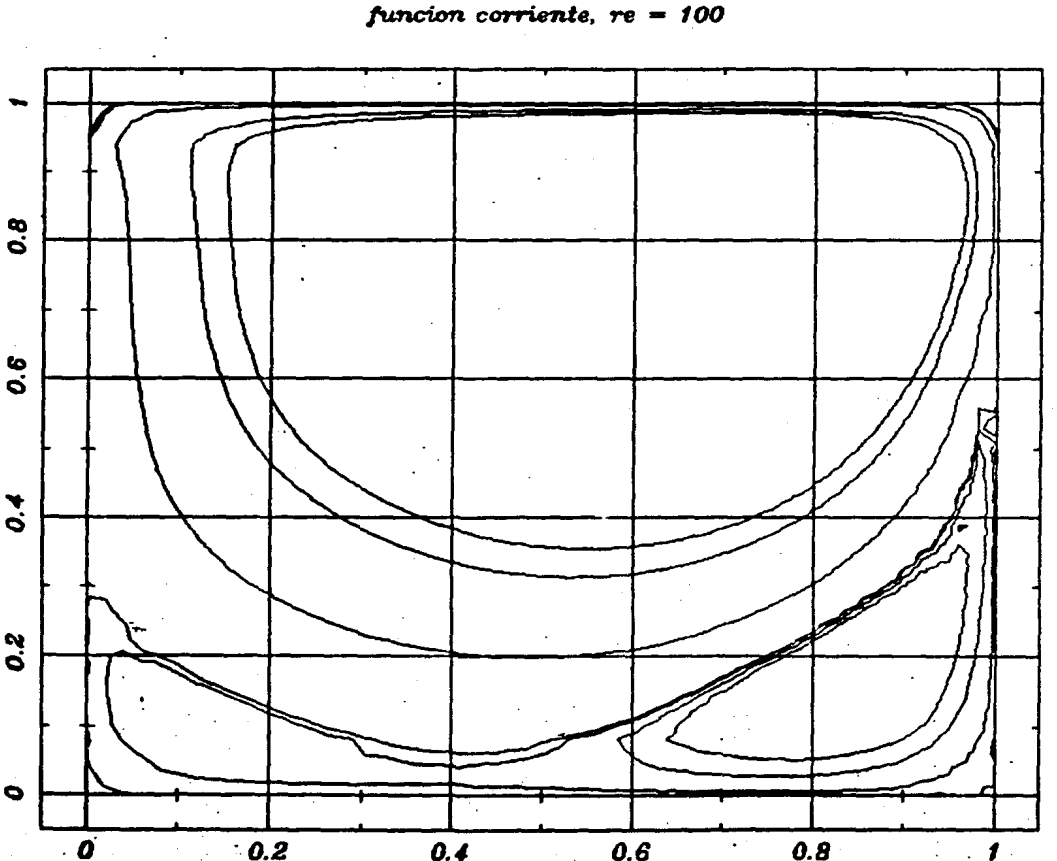


Figura VI.41

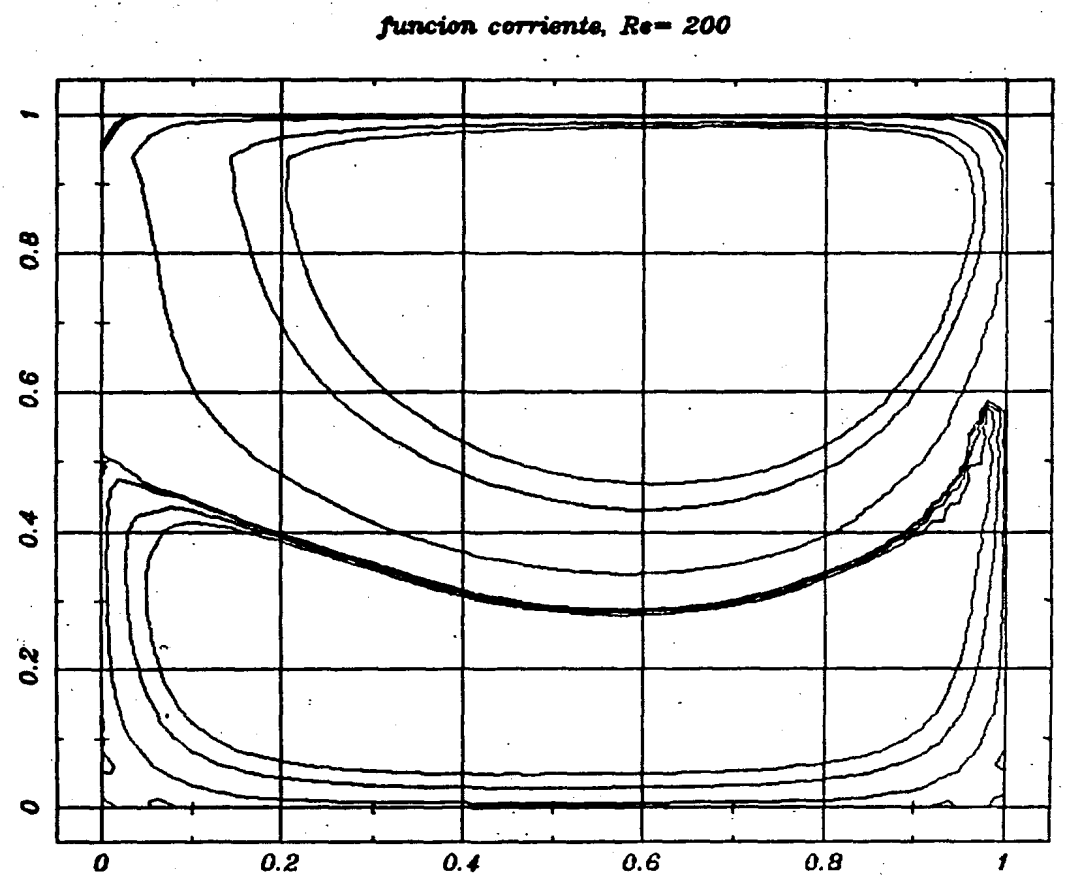


Figura VI.42

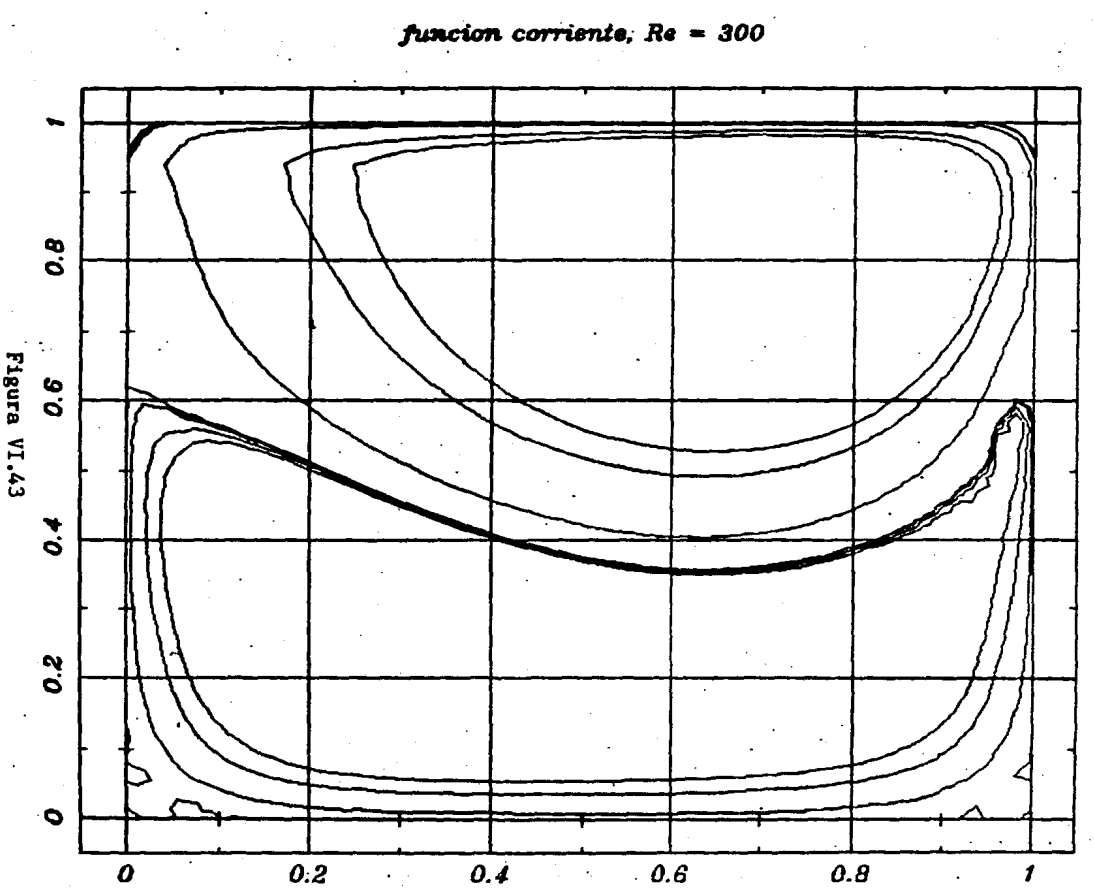


Figura VI.43

funcion corriente, Re = 400

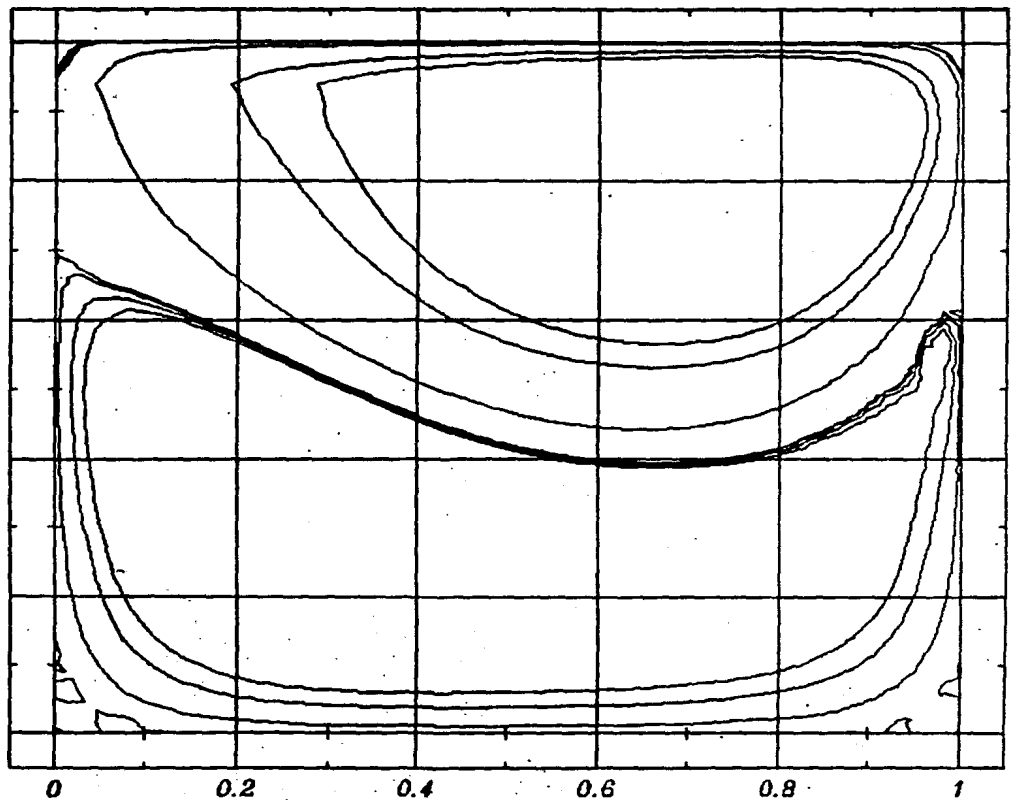


Figura VI.44

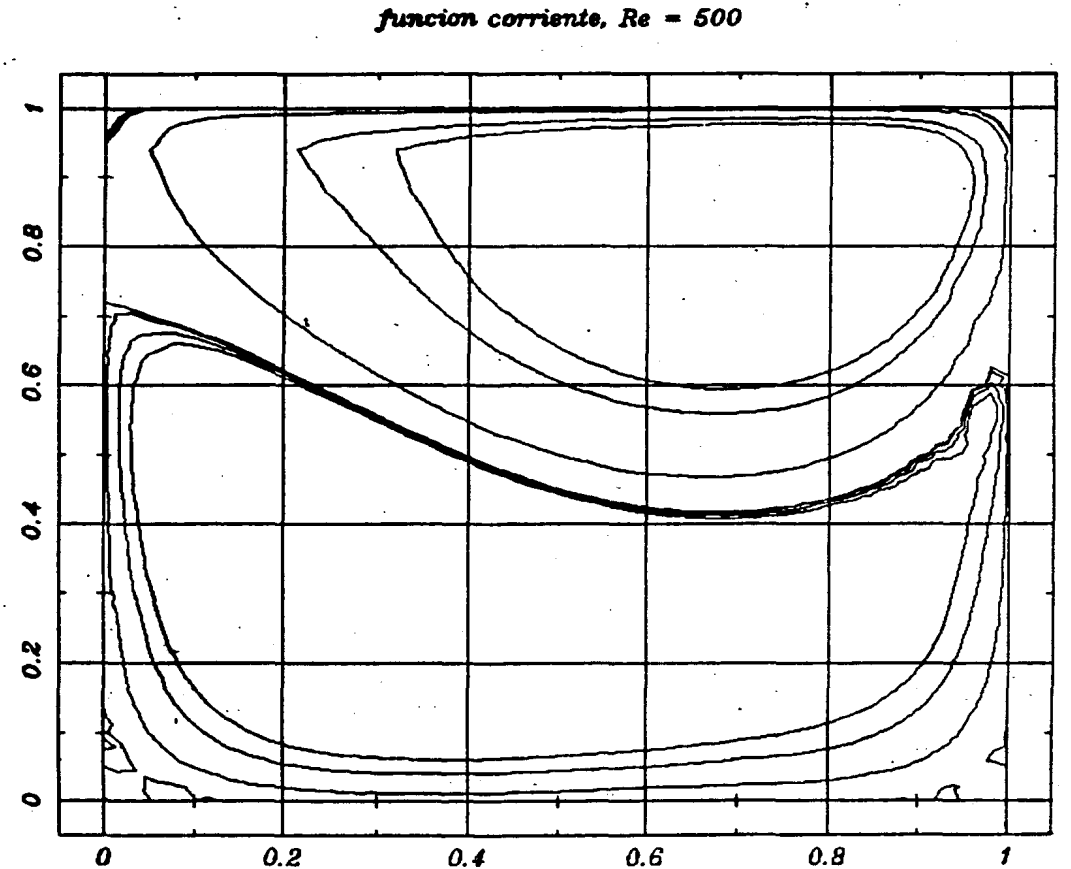


Figura VI.45

Conclusiones.

En el presente trabajo se desarrolló un sistema para resolver ecuaciones elípticas en dominios irregulares utilizando siempre el mismo código cambiando parámetros geométricos de manera apropiada.

La idea de desarrollar este sistema fue la de estudiar flujos viscosos en geometrías complicadas con miras a entender la influencia de obstáculos en la formación e interacción de vórtices.

Este programa se llevó a cabo exitosamente pudiendose calcular por vez primera la formación e interacción de vórtices en un tubo con una cavidad para un amplio rango de números de Reynolds.

Otro resultado interesante fue el de la captura del vórtice aguas abajo que se forma en una válvula.

Este sistema fue diseñado con la versátilidad suficiente para poder cambiar rápidamente de ecuación y geometría. Esto permitió resolver muy facilmente un problema de fluidos conductores que no se habia contemplado originalmente.

En la tesis se describió el sistema y se proporcionan los programas de forma tal que puedan ser utilizados por otros investigadores de mecánica de fluidos.

Bibliografía

[1] H. J. Wirz and J. J. Smolderen (Editors), *Numerical Methods in Fluid Dynamics*, McGraw-Hill, New York, 1978, pp. 89–154.

[2] I. G. Currie, *Fundamental Mechanics of Fluids*, McGraw-Hill, New York, 1974.

[3] Terrence W. Pratt, *Programming Languages, Design and Implementation*, Prentice-Hall, Inc., New Jersey, 1984.

[4] C. Ghezzi and M. Jazayeri, *Programming Language Structures*, John Willey & Sons, Inc., New York, 1982.

[5] O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare, *Programación estructurada*, Editorial Tiempo Contemporáneo, Argentina, 1976.

[6] Richard L. Burden, J. Douglas Faires, *Análisis Numérico*, Grupo Editorial Iberoamérica, México D.F., 1985.

[7] George E. Forsythe, Wolfgang R. Wasow, *Finite-Difference Methods for Partial Differential equations*, John Willey & Sons, Inc., New York, 1960.

[8] Roger Peyret, Thomas D. Taylor, *Computational Methods for Fluid Flow*, Springer-Verlag, Inc., New York, 1983.

[9] Steven E. Koonin, *Computational Physics*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1986.

[10] Sergio Pissanetzky, *Sparse Matrix Technology*, Academic Press, Inc., New York, 1984.

[11] Louis A. Hageman, David M. Young, *Applied Iterative Methods*, Academic Press, Inc., New York, 1981.

[12] G. D. Smith, *Numerical Solution of Partial Differential Equations*, Oxford University Press, Ely House, London, 1965.

[13] J. Stoer, R. Burlirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[14] C. O. Frederick, Y. C. Wong and F. W. Edge, *Two-Dimensional Automatic Mesh Generation for Structural Analysis*, International Journal for Numerical Methods in Engineering., **2**, (1970) pp. 133–144.

[15] Mark A. Yerry and Mark S. Shephard, *A Modified Quadtree Approach to Finite Element Mesh Generation*, IEEE Comp. Graph. Appl., **3**, Jan 1983, pp. 39–46.

[16] Y. T. Lee, A. de Pennington and N. K. Shaw, *Automatic Finite-Element Mesh Generation from Geometric Models-A Point-Based Approach*, ACM Transactions on Graphics., **3**, No. 4, (1984) pp. 287–311.

[17] M. Nallasamy, *Numerical Solution of The Separating Flow Due to an Obstruction*, Computers & Fluids, **14**, (1986) pp. 59–68.

[18] M. Nallasamy and K. Krishna Prasad, *Numerical Studies on Quasilinear and Linear Elliptic Equations*, Journal of Computational Physics, **15**, (1974) pp. 429–448.

[19] Murli M. Gupta and Ram P. Manohar, *Boundary Approximations and Accuracy in Viscous Flow Computations*, Journal of Computational Physics, **31**, (1979) pp. 265–288.

[20] Murli M. Gupta, *A Comparison of Numerical Solutions of Convective and Diverge forms of the Navier-Stokes Equations for the Driven Cavity Problem*, Journal of Computational Physics, **43**, (1981) pp. 260–267.

[21] Murli M. Gupta, *Discretization Error Estimates for Certain Splitting Procedures for Solving First Biharmonic Boundary Value Problems*, SIAM J. Numer. Anal., **12**, No. 3, June 1975, pp. 364–377.

[22] Louis W. Ehrlich and Murli M. Gupta, *Some Difference Schemes for the Biharmonic Equation*, SIAM J. Numer. Anal., **12**, No. 5, October 1975, pp. 773–790.

[23] José Luis Morales Perez, *Métodos Iterativos en Matrices Bandadas*, Tesis de Maestría en Ciencias de la Computación, IIMAS-UNAM, (1985).