

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS



SOLUCIÓN NUMÉRICA DEL PROBLEMA DE LA MEMBRANA VIBRANTE CON EL MODELO DE OBJETOS

TESIS

QUE PARA OBTENER EL TÍTULO DE

FÍSICO

PRESENTA:

LAURA CRUZ VAZQUEZ

ENSENADA B. C.

NOVIEMBRE DE 1997.

D e d i c a t o r i a

A mi madre C. Alicia Vázquez Cervantes

. . . por su amor. . .

A mi padre Alejandro I. Cruz Rodríguez

. . . por su comprensión y protección. . .

A mis hermanos Alejandro, Gaby y Karina

. . . por su sinceridad y honestidad. . .

A Cynthia Victorio Cuellar

. . . por su incondicional amistad. . .

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS

SOLUCIÓN NUMÉRICA DEL PROBLEMA DE LA MEMBRANA VIBRANTE
CON EL MODELO DE OBJETOS

TESIS PROFESIONAL

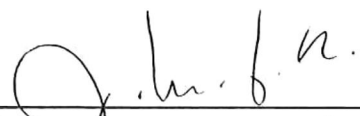
QUE PRESENTA

LAURA CRUZ VAZQUEZ

APROBADO POR:



DR. JOSÉ GÓMEZ VALDÉS
PRESIDENTE DEL JURADO



M. C. JESUS MANUEL FIGUEROA RODRÍGUEZ
PRIMER VOCAL



FÍS. FRANCISCO JUÁREZ GARCÍA
SECRETARIO

Agradecimientos

Gracias

A Dios por todo lo dado y lo no dado.

A mis padres por todo su esfuerzo.

A José Gómez Valdés por dirigir esta Tesis.

A Manuel Figueroa y Francisco Juárez por leer y atender este trabajo.

A E. Cota, J. Siqueiros, Silvia Gómez, Alberto Rubio †, Alejandro Parés, . . . por darme las bases.

Al CONACYT por financiar esta tesis a través del proyecto 225008-5-3523T.

Al CICESE por el equipo facilitado.

A los compañeros del CICESE por sus aportaciones y amistad.

A mis compañeros de Facultad por su buen compañerismo y apoyo profesional.

A Gloria Rubí Vázquez por su amistad, consejos, . . .

A Blanca por su paciencia y ayuda en todos estos trámites.

A Martha Rivera Hernández y Jaime Girón por todas sus atenciones y amistad.

A René Campos por su confianza y apoyo de trabajo.

A Joel Rodríguez y Gladys por su incondicional apoyo profesional y amistad.

A Estela, Isela, Sofía, María Luisa, Lilia, Guillermina, Cuca, Ana, María Elena, Alfonso, Raúl, Juventino, Ayala, Francisco y Víctor . . . por su interés y sincera amistad.

Resumen de la Tesis de LAURA CRUZ VAZQUEZ presentada como requisito parcial para la obtención de la Licenciatura en Física. Ensenada, Baja California, México. Noviembre de 1997.

SOLUCIÓN NUMÉRICA DEL PROBLEMA DE LA MEMBRANA VIBRANTE
CON EL MODELO DE OBJETOS.

Resumen aprobado:



DR. JOSÉ GÓMEZ VALDÉS.

Se hace orientación por objetos de la solución numérica por el método explícito de diferencias finitas de la ecuación de onda bidimensional. Se usa el lenguaje de programación C++. Ejemplos seleccionados para membranas vibrantes en regiones finitas se instalaron en el programa orientado a objetos. Resultó una organización más eficiente que contar con programas por separado. El programa se puede usar en la enseñanza de la física de la membrana vibrante.

ÍNDICE

a)Contenido

I INTRODUCCIÓN	1
II ANTECEDENTES	
II.1 Ecuación de onda	3
II.1.1 Notación	3
II.1.2 Problema de Cauchy	5
II.1.3 Frontera	6
II.1.4 Desigualdad de dependencia	7
II.1.5 Problema de la membrana	8
II.1.5.1 Método de separación de variables	9
II.2 Método Numérico	15
II.2.1 Método explícito de diferencias finitas	15
II.2.2 Código procedural	16
III MODELO DE OBJETOS	20
IV RESULTADOS	23
IV.1 Código orientado a objetos	23
IV.2 Ejemplos	30
V DISCUSIÓN	42
VI CONCLUSIONES	46
VII LITERATURA CITADA	47
VIII ANEXOS	48
VIII.1 Archivo de implantación: constantes.li	48
VIII.2 Archivo de implantación: vector.li	48
VIII.3 Archivo de implantación: matrix.li	50

VIII.4 Archivo de implantación: update.li	53
VIII.5 Archivo de implantación: cfrontera.li	53
VIII.6 Archivo de implantación: explicito.li	54
VIII.7 Archivo de implantación: cinicial.li	54

b) Figuras

Fig. 1 Representación de la solución del programa MEDF	18
Fig. 2 Archivos de cabecera del programa TUTORIAL	25
Fig. 3 Ejemplo 1. Membrana cuadrada unitaria con condición inicial $\sin \pi x_1 \sin \pi x_2$	32
Fig. 4 Ejemplo 2. Membrana rectangular con condición inicial $\sin \pi x_1 \sin \pi x_2$	33
Fig. 5 Ejemplo 3. Membrana cuadrada con condición inicial $\sin \pi x_1 \sin 2\pi x_2$	34
Fig. 6 Ejemplo 4. Membrana cuadrada con condición inicial $\sin 8\pi x_1 \sin \pi x_2$	35
Fig. 7 Ejemplo 5. Membrana cuadrada con condición inicial $10^{-5} x_2 (20-x_2) x_1 (20-x_1)$	36
Fig. 8 Ejemplo 6. Membrana cuadrada con el plano inicial $x_1 + x_2$	37
Fig. 9 Ejemplo 7. Membrana cuadrada con condición inicial $f(x_1, x_2)=1$	38
Fig. 10 Ejemplo 8. Membrana cuadrada con forma inicial $x_1^6 x_2$	39
Fig. 11 Ejemplo 9. Membrana cuadrada con forma inicial $x_1^2 + x_2^2$	40
Fig. 12 Ejemplo 10. Membrana cuadrada con forma inicial $\cos 10\pi x_1 \sin 3\pi x_2$	41
Fig. 13 Aplicación alternativa con el modelo de objetos	43

I INTRODUCCIÓN

Una razón importante para estudiar la ecuación de onda es que ésta describe muchos fenómenos de vibración y propagación de perturbaciones en la física. Es un prototipo en el estudio matemático de ecuaciones diferenciales parciales de tipo hiperbólico.

Definimos una membrana como una hoja delgada de material flexible y elástico. La membrana es ideal si el material que la constituye es uniforme y perfectamente flexible. Se considera que la membrana se estira sobre un bastidor formando un tambor. Luego se hace vibrar desde su posición de equilibrio. El marco del bastidor se representa por una curva C que limita al dominio $\mathfrak{R}$ sobre el plano (x, y) . La membrana se mantiene en ese plano mientras esté en reposo, en ausencia de fuerzas externas. Las fuerzas de estiramiento, que se aplican en la frontera, se transmiten a toda la membrana. La membrana se estira uniformemente en cada punto y en todas direcciones.

Algunos problemas de membranas vibrantes se pueden resolver analíticamente. Aquí se propone encontrar la solución numérica usando programación orientada a objetos (OOP, por sus siglas en inglés). OOP es un paradigma de punta en el desarrollo de software, con el cual un sistema real es modelado por objetos que interactúan entre sí. Los objetos son elementos del dominio del problema. Cada objeto pertenece a una clase, la cual define a las variables y a los métodos empleados. Debido a su propiedad de extensibilidad, OOP se está haciendo muy popular en los proyectos de modelado que requieren muchas líneas de código.

A la fecha, no se ha explorado sistemáticamente la utilidad de la OOP en las aplicaciones científicas, aunque sí se pueden encontrar trabajos (Gómez-Valdés y Favela, 1996). En particular, en la física las aplicaciones con OOP son casi nulas. Como la modelación de la ecuación de onda, tridimensional digamos, requiere de computación de alto rendimiento, cualquier técnica de programación nueva o tecnología de hardware de punta debe inmediatamente ser explorada en trabajos piloto.

En este trabajo se estudia numéricamente la ecuación de onda para el problema de una membrana vibrante, en un ambiente orientado a objetos. Podemos considerar al conjunto de ejemplos numéricos que se usan comúnmente en la enseñanza como un sistema. Podemos también suponer que con OOP puede trabajarse más eficientemente a la membrana vibrante que con el procedimiento tradicional. Este estudio puede ser usado como una base para posteriores investigaciones de OOP en aplicaciones científicas y como un tutorial en el auxilio de la enseñanza de la ecuación de onda.

II ANTECEDENTES

El objetivo de este capítulo es introducir los conceptos básicos de la ecuación de onda generalizada. La razón es tratar el problema de la ecuación de onda independientemente de la dimensión del espacio. Asimismo se plantea el problema de la membrana y su solución por un método analítico y numérico.

II.1 Ecuación de onda

II.1.1. Notación

Para la solución de la ecuación de onda n -dimensional se requiere un espacio de $n+1$ dimensiones o R^{n+1} , donde n representa el número de variables espaciales y $t \in R^1$. La notación R^n representa un espacio euclídeo n -dimensional donde $x = (x_1, \dots, x_n)$ es un punto de R^n . Un dominio R^n o dominio n -dimensional es un conjunto abierto y conexo (no vacío) de puntos pertenecientes a R^n . Consideremos un dominio n -dimensional $\mathfrak{R}$. Si $\mathfrak{S} \subset \mathfrak{R}$, se dice que $\mathfrak{S}$ es estrictamente interior respecto a $\mathfrak{R}$, si $\bar{\mathfrak{S}} \subset \mathfrak{R}$, donde $\bar{\mathfrak{S}}$ indica clausura del conjunto $\mathfrak{S}$.

Definiremos también el uso de la notación $C^k(\mathfrak{R})$ como el conjunto de todas las funciones con derivadas parciales continuas hasta un orden k , donde k es un número entero no negativo. El conjunto $C^k(\bar{\mathfrak{R}})$ es un subconjunto del conjunto $C^k(\mathfrak{R})$, compuesto por todas las funciones cuyas derivadas parciales son continuas hasta el k -ésimo orden en $\bar{\mathfrak{R}}$. Podemos escribir $C^\infty(\mathfrak{R}) = C^k(\mathfrak{R})$, donde $k = 0, 1, 2, 3, \dots$. Del mismo modo representamos $C^\infty(\bar{\mathfrak{R}})$ como el conjunto de todas las funciones que pertenecen a todos los $C^k(\bar{\mathfrak{R}})$.

Para simplificar representamos el operador diferencial parcial por

$$L_i = \partial / \partial x_i, \quad (1)$$

donde x_i representa la i -ésima variable e $i = 1, 2, 3, \dots, n$. El operador para la ecuación de onda bidimensional es:

$$\wp(L) = L_1^2 + L_2^2 - L_3^2, \quad (2)$$

donde x_1 y x_2 son las variables espaciales, x_3 es la variable temporal y $\square(L)$ es el operador de onda.

El operador de onda para la ecuación de onda n-dimensional es

$$\square(L) = L_1^2 + \dots + L_n^2 - L_t^2, \quad (3)$$

en la cual t es la variable temporal. La ecuación característica

$$v_1^2 + \dots + v_n^2 - v_t^2 = 0, \quad (4)$$

describe una dirección donde $v = (v_1, \dots, v_n, v_t)$. Si v es un vector unitario entonces $v_t = \pm 1/(2)^{1/2}$.

Las direcciones forman un ángulo de 45° con el eje t . La superficie n-dimensional es característica respecto a la ecuación de onda, es una superficie cónica doble con ápice en un punto $(x^\circ, t^\circ) = (x_1^\circ, \dots, x_n^\circ, t^\circ)$. La ecuación para este cono es:

$$(x_1 - x_1^\circ)^2 + \dots + (x_n - x_n^\circ)^2 - (t - t^\circ)^2 = 0. \quad (5)$$

El cono característico para $t \leq t^\circ$, es decir el cono de la parte inferior podemos representarlo por medio de la siguiente desigualdad

$$(x_1 - x_1^\circ)^2 + \dots + (x_n - x_n^\circ)^2 - (t - t^\circ)^2 \leq 0, \quad t \leq t^\circ. \quad (6)$$

Consideremos un número τ tal que $\tau \leq t^\circ$ y el plano $t = \tau$ en un espacio R^{n+1} . Por ahora, el plano, está en un espacio R^n en $t = \tau$ e intersecta al cono inferior ($t \geq t^\circ$) con ápice en (x°, t°) . La superficie en $t = \tau$ satisface la desigualdad siguiente:

$$(x_1 - x_1^\circ)^2 + \dots + (x_n - x_n^\circ)^2 \leq (t^\circ - \tau)^2, \quad t = \tau. \quad (7)$$

Representemos el conjunto de dichos puntos por $\bar{I}(x^\circ, t^\circ - \tau)$. El conjunto $\bar{I}$ para el caso $n=2$ es un círculo limitado por el cono inferior ($t \leq t^\circ$). Generalizando, en el espacio R^n para $t = \tau$, podemos decir que $\bar{I}$ tiene centro en x° y radio en $t^\circ - \tau$. Aunque como sabemos una representación geométrica adecuada es más ilustrativa en los casos $n=1$ y $n=2$. Los conjuntos $\bar{I}$ son representados por intervalos y círculos para $n=1$ y $n=2$ respectivamente.

II.1.2 Problema de Cauchy

Sea $x \in R^n$ y $t \in R^1$, donde $x = (x_1, \dots, x_n)$; $x_1, \dots, x_n$ son las variables espaciales y t es el tiempo. Podemos representar la ecuación de onda n -dimensional por,

$$\xi_{x_1 x_1} + \dots + \xi_{x_n x_n} - \xi_{tt} = q(x, t), \quad (8)$$

donde $\xi = \xi(x_1, \dots, x_n, t)$; $\xi_{x_1 x_1}, \dots, \xi_{x_n x_n}$ y ξ_{tt} son las segundas derivadas parciales respecto a $x_1, \dots, x_n$ y t respectivamente; $q(x, t)$ es una función dada. La función ξ está definida en un espacio R^{n+1} . No existe problema en representar gráficamente el espacio (x, t) para $n=1$ y $n=2$. Para $n=1$ se tiene un plano formado por los ejes x y t . Para $n=2$ se forma la tríada de ejes x_1, x_2 y t . Para $n \geq 3$ ya no es posible hacer representaciones gráficas. Sin embargo es posible usar una notación para el espacio (x, t) con $n+1$ dimensiones, representando el x -espacio por un eje y t en otro. Si $q(x, t)=0$, resulta la ecuación de onda homogénea

$$\xi_{x_1 x_1} + \dots + \xi_{x_n x_n} = \xi_{tt}. \quad (9)$$

El problema de Cauchy consiste en la búsqueda de una solución $\xi(x_1, \dots, x_n, t)$, que satisfice las condiciones iniciales dadas por

$$\xi(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n), \quad (10)$$

$$\xi_t(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n). \quad (11)$$

Las funciones f y g son funciones dadas y definidas en el x -espacio R^n en $t=0$. La superficie inicial del problema (9)-(11) es el plano $t=0$ en el espacio (x, t) R^{n+1} .

Si suponemos que f y g son analíticas en el origen del x -espacio, entonces el problema de valores iniciales tiene solución única. Dicha solución está definida y es analítica en una vecindad del origen (x, t) , por el teorema de Cauchy-Kovalevsky. Sin embargo, se debe encontrar una solución para todo (x, t) en R^{n+1} , independientemente de la analiticidad de f y g , no sólo en una vecindad del origen. Para esto debe resolverse el problema para $t \geq 0$, es decir, la mitad superior del espacio. Esto, debido a que para $t \leq 0$ (mitad inferior del espacio) podemos hacer la transformación $t' = -t$ y reducir el problema al caso en que $t \geq 0$ sin alterar la

ecuación de onda. Otra transformación que puede hacerse es cuando el plano inicial es $t=t^*$, transformando con $t'=t-t^*$, obtenemos el problema (9)-(11) con el plano inicial $t=0$ sin alterar la ecuación.

II.1.3 Frontera

Hay tres tipos de condiciones de frontera comunes en la descripción de un fenómeno físico. Cuando una función dada φ se especifica en cada punto de la frontera de una región, ésta queda limitada por una curva en el plano o una superficie si el dominio es tridimensional. En estos casos hablamos de las condiciones de Dirichlet. Por otro lado, cuando los valores de la derivada normal de una función se definen en la frontera, tenemos condiciones de frontera de Neumann. Finalmente, las condiciones de Cauchy ocurren si una de las variables independientes es t y los valores de una función dada y su derivada parcial respecto a t , están dados sobre la frontera en $t=0$.

Hemos tratado, hasta ahora, un problema de valores iniciales para la ecuación de onda. Físicamente la ecuación de onda podemos asociarla al movimiento de una cuerda infinita estirada que vibra, donde el desplazamiento y velocidad están dados como condiciones iniciales. En dos dimensiones podemos pensar en el problema de una membrana infinita estirada, donde de manera similar requerimos el desplazamiento y velocidad como datos iniciales. Sin embargo, es necesaria la presencia de fronteras en el x -espacio.

Plantearemos el caso general para el problema de valores iniciales y en la frontera de la ecuación de onda. Sea $\mathfrak{R}$ un dominio en R^n , queremos encontrar una función $\xi(x,t)$ definida para $x \in \mathfrak{R}$ y $t \geq 0$, que satisface la ecuación de onda n -dimensional para $x \in \mathfrak{R}$ y $t > 0$. Las condiciones iniciales son las planteadas en la sección II.1.2, para $x \in \overline{\mathfrak{R}}$. Consideramos además a una de las condiciones de frontera siguientes:

$$\xi=0, \quad x \in \partial\mathfrak{R}, \quad t \geq 0, \quad (12)$$

$$\partial\xi/\partial n=0, \quad x \in \partial\mathfrak{R}, \quad t \geq 0, \quad (13)$$

donde $\xi = \xi(x, t)$. En esta última condición de frontera la diferenciación es en la dirección de la normal externamente a $\partial\mathfrak{R}$. La solución debe satisfacer sólo a una de las condiciones de frontera homogéneas dadas.

II.1.4 Desigualdad de dependencia

Antes de describir dicha desigualdad, consideremos un punto (x^0, t^0) en el espacio R^{n+1} para $t^0 > 0$. Sea $\mathfrak{R}$ un dominio en R^{n+1} que se encuentra limitado por el cono característico inferior ($t \leq t^0$) y el plano $t=0$. El cono tiene su ápice en (x^0, t^0) .

Ahora consideremos una solución $\xi = \xi(x_1, \dots, x_n, t)$ a la ecuación de onda n -dimensional. Podemos representar la desigualdad de dependencia por

$$\int_{\bar{\mathfrak{R}}(x^0, t^0-\tau)} (\xi_{x_1}^2 + \dots + \xi_{x_n}^2 + \xi_t^2) \big|_{t=\tau} dx \leq \int_{\bar{\mathfrak{R}}(x^0, t^0)} (\xi_{x_1}^2 + \dots + \xi_{x_n}^2 + \xi_t^2) \big|_{t=0} dx, \quad (14)$$

donde ξ satisface la ecuación de onda en un dominio $\mathfrak{R}$ y pertenece al conjunto $C^2(\bar{\mathfrak{R}})$. El integrando en ambas integrales está dado por la suma de los cuadrados de las primeras derivadas parciales de ξ . Como vimos en la sección II.1.1, $\bar{\mathfrak{R}}$ es un dominio n -dimensional con la propiedad de clausura.

Sea una región D en el espacio R^n . Definimos la integral siguiente como:

$$E(\xi; D, \tau) = (1/2) \int_D (\xi_{x_1}^2 + \dots + \xi_{x_n}^2 + \xi_t^2) \big|_{t=\tau} dx, \quad (15)$$

que es la energía de $\xi(x, t)$ en la región D al tiempo $t=\tau$. Físicamente, $\xi(x, t)$ describe el fenómeno de propagación de la onda y E es la energía de dicha onda en la región D en el tiempo $t=\tau$. Por ello la desigualdad de dependencia se conoce también como desigualdad de energía. Podemos escribirla como sigue:

$$E(\xi; \bar{\mathfrak{R}}(x^0, t^0-\tau), \tau) \leq E(\xi; \bar{\mathfrak{R}}(x^0, t^0), 0). \quad (16)$$

Enunciaremos el siguiente teorema relacionado a la conservación de energía.

TEOREMA. Sea ξ una solución del problema de valores iniciales n -dimensional, donde $\xi \in C^2$,

$x \in R^n$ y $t \geq 0$. Los datos iniciales f y g se anulan en $I(0, R)$ para R^n . La energía de ξ contenida en todo el espacio R^n es constante independientemente de t , para toda $t \geq 0$, es decir,

$$(1/2)(\xi_{x_1}^2 + \dots + \xi_{x_n}^2 + \xi_t^2)|_{t=\tau} dx = (1/2)(\xi_{x_1}^2 + \dots + \xi_{x_n}^2 + \xi_t^2)|_{t=0} dx, \quad (17)$$

que se simplifica por

$$E(\xi; R^n, \tau) = E(\xi; R^n, 0). \quad (18)$$

II.1.5 Problema de la membrana

La ecuación de onda en dos dimensiones,

$$\xi_{tt}(x_1, x_2, t) = c^2[\xi_{x_1 x_1}(x_1, x_2, t) + \xi_{x_2 x_2}(x_1, x_2, t)], \quad (19)$$

donde x_1 y x_2 son dos variables espaciales, t es el tiempo, $c = (T/\sigma)^{1/2}$ es una constante y $\xi = \xi(x_1, x_2, t)$. La tensión se denota por T y la densidad de área de la membrana por σ .

El problema se sujeta a las siguientes condiciones iniciales y de frontera,

$$\left. \begin{aligned} \xi(x_1, x_2, t) &= 0 \text{ dentro de un dominio } \mathfrak{R} \text{ para } t > 0, \\ \xi(x_1, x_2, 0) &= f(x_1, x_2) \text{ dentro del dominio } \mathfrak{R}, \\ \xi_t(x_1, x_2, 0) &= g(x_1, x_2) \text{ dentro del dominio } \mathfrak{R}. \end{aligned} \right\} \quad (20)$$

donde $\mathfrak{R}$ es el dominio de la membrana.

Sea una membrana rectangular de longitudes a y b cuyo dominio esta dado por $\mathfrak{R} = \{x | 0 < x_1 < a, 0 < x_2 < b\}$. La frontera esta formada por las rectas $x_1 = 0$, $x_1 = a$, $x_2 = 0$ e $x_2 = b$.

El problema a resolver en la siguiente sección es

$$\left. \begin{aligned} \text{Ecuación:} & \quad c^2[\xi_{x_1 x_1} + \xi_{x_2 x_2}] = \xi_{tt}, \quad 0 < x_1 < a, \quad 0 < x_2 < b, \quad t > 0, \\ \text{Condiciones de frontera:} & \quad \xi(0, x_2, t) = 0, \quad 0 < x_2 < b, \quad t > 0, \\ & \quad \xi(a, x_2, t) = 0, \quad 0 < x_2 < b, \\ & \quad \xi(x_1, 0, t) = 0, \quad 0 < x_1 < a, \\ & \quad \xi(x_1, b, t) = 0, \quad 0 < x_1 < a, \\ \text{Condiciones iniciales:} & \quad \xi(x_1, x_2, 0) = f(x_1, x_2), \quad 0 < x_1 < a, \quad 0 < x_2 < b, \\ & \quad \xi_t(x_1, x_2, 0) = g(x_1, x_2), \quad 0 < x_1 < a, \quad 0 < x_2 < b. \end{aligned} \right\} \quad (21)$$

II.1.5.1 Método de separación de variables

Debido a la simplicidad de la región seleccionada es posible encontrar su solución analítica por el método de separación de variables, proponiendo una solución,

$$\xi(x_1, x_2, t) = \Theta(t) \zeta(x_1, x_2). \quad (22)$$

Sustituimos la solución anterior en la ecuación (1.19) y obtenemos,

$$\Theta''(t) \zeta(x_1, x_2) = c^2 [\zeta_{x_1 x_1}(x_1, x_2) \Theta(t) + \zeta_{x_2 x_2}(x_1, x_2) \Theta(t)], \quad (23)$$

luego separamos variables,

$$(1/c^2) \Theta''(t) = [\zeta_{x_1 x_1}(x_1, x_2) + \zeta_{x_2 x_2}(x_1, x_2)] / \zeta(x_1, x_2) = -\lambda, \quad (24)$$

donde λ es la constante de separación. De aquí obtenemos,

$$\Theta''(t) + c^2 \lambda \Theta(t) = 0, \quad (25)$$

$$\zeta_{x_1 x_1} + \zeta_{x_2 x_2} + \lambda \zeta = 0. \quad (26)$$

Resolvemos primero la Ec. (26) con las condiciones de frontera correspondientes,

$$\left. \begin{aligned} \xi(0, x_2, t) = \Theta(t) \zeta(0, x_2) = 0 &\Rightarrow \zeta(0, x_2) = 0, \\ \xi(a, x_2, t) = \Theta(t) \zeta(a, x_2) = 0 &\Rightarrow \zeta(a, x_2) = 0, \\ \xi(x_1, 0, t) = \Theta(t) \zeta(x_1, 0) = 0 &\Rightarrow \zeta(x_1, 0) = 0, \\ \xi(x_1, b, t) = \Theta(t) \zeta(x_1, b) = 0 &\Rightarrow \zeta(x_1, b) = 0, \end{aligned} \right\} \quad (27)$$

de donde obtenemos las condiciones de frontera para ζ que nos permiten obtener soluciones no triviales de ξ .

Resolvemos nuevamente por separación de variables, proponiendo la solución,

$$\zeta(x_1, x_2) = X_1(x_1) X_2(x_2). \quad (28)$$

Sustituimos en la Ec. (26) y obtenemos

$$X_2(x_2) X_1''(x_1) + X_2''(x_2) X_1(x_1) + \lambda X_1(x_1) X_2(x_2) = 0. \quad (29)$$

Separando variables resulta,

$$[X_1''(x_1) + \lambda X_1(x_1)]/X_1(x_1) = -X_2''(x_2)/X_2(x_2). \quad (30)$$

Como cada lado de la Ec. (30) depende de variables distintas, la igualdad se cumple siempre que ambos lados sean igual a una constante de separación μ ,

$$-[X_1''(x_1) + (\lambda - \mu)X_1(x_1)]/X_1(x_1) = X_2''(x_2)/X_2(x_2) = -\mu. \quad (31)$$

De la Ec. (31) se obtienen

$$X_1''(x_1) + (\lambda - \mu)X_1(x_1) = 0, \quad (32)$$

$$X_2''(x_2) + \mu X_2(x_2) = 0. \quad (33)$$

Aplicamos las condiciones de frontera:

$$\left. \begin{aligned} \zeta(0, x_2) &= X_1(0)X_2(x_2) = 0 \Rightarrow X_1(0) = 0, \\ \zeta(a, x_2) &= X_1(a)X_2(x_2) = 0 \Rightarrow X_1(a) = 0, \\ \zeta(x_1, 0) &= X_1(x_1)X_2(0) = 0 \Rightarrow X_2(0) = 0, \\ \zeta(x_1, b) &= X_1(x_1)X_2(b) = 0 \Rightarrow X_2(b) = 0. \end{aligned} \right\} \quad (34)$$

Las soluciones de las Ecs. (32) y (33) se determinan fácilmente y ellas son

$$X_1(x_1) = A \cos[x_1(\lambda - \mu)^{1/2}] + B \sin[x_1(\lambda - \mu)^{1/2}], \quad (35)$$

$$X_2(x_2) = C \cos[x_2(\mu)^{1/2}] + D \sin[x_2(\mu)^{1/2}], \quad (36)$$

donde A, B, C y D son constantes. De las condiciones $X_2(0) = 0$ y $X_2(b) = 0$ obtenemos

$$C \cos(0) + D \sin(0) = 0, \quad (37)$$

$$C \cos[b(\mu)^{1/2}] + D \sin[b(\mu)^{1/2}] = 0. \quad (38)$$

De aquí obtenemos $C = 0$. Sustituimos en (36) para obtener la ecuación característica,

$$D \sin[b(\mu)^{1/2}] = 0, \quad (39)$$

lo cual se cumple cuando $b(\mu)^{1/2} = n\pi$, donde n es un entero positivo, por lo que los eigenvalores μ son

$$\mu_n = n^2 \pi^2 / b^2, \quad (40)$$

donde $n=1,2,3, \dots$, que corresponden a las soluciones no triviales.

Por otro lado con las condiciones de frontera $X_1(0)=0$ y $X_1(a)=0$ y la Ec. (35), se obtienen,

$$A \cos(0) + B \sin(0) = 0, \quad (41)$$

$$A \cos[a(\lambda-\mu)^{1/2}] + B \sin[a(\lambda-\mu)^{1/2}] = 0, \quad (42)$$

donde A y B son constantes.

De (41) vemos que $A=0$. Por lo que (42), nos da la ecuación característica correspondiente,

$$B \sin[a(\lambda-\mu)^{1/2}] = 0. \quad (43)$$

La ecuación se satisface para $a(\lambda-\mu)^{1/2} = m\pi$, donde m es un entero positivo. Los eigenvalores son,

$$\lambda_{mn} = \pi^2 [(m^2/a^2) + (n^2/b^2)], \quad (44)$$

donde $n, m = 1,2,3, \dots$, para las soluciones no triviales. Las correspondientes eigenfunciones están dadas por $\zeta_{nm}(x_1, x_2) = X_{1n}(x_1)X_{2m}(x_2)$, donde

$$\zeta_{nm}(x_1, x_2) = \sin(n\pi x_1/a) \sin(m\pi x_2/b), \quad (45)$$

$$\Theta_{nm}(t) = A \cos[ct(\lambda_{nm})^{1/2}] + B \{ \sin[ct(\lambda_{nm})^{1/2}] / c(\lambda_{nm})^{1/2} \}. \quad (46)$$

La solución para $\xi(x_1, x_2, t)$ está dada por,

$$\xi(x_1, x_2, t) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} \Theta_{nm}(t) \zeta_{nm}(x_1, x_2), \quad (47)$$

$$\xi(x_1, x_2, t) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} \{ A_{nm} \cos[ct(\lambda_{nm})^{1/2}] + B_{nm} \sin[ct(\lambda_{nm})^{1/2}] / c(\lambda_{nm})^{1/2} \} \zeta_{nm}(x_1, x_2). \quad (48)$$

Con las condiciones iniciales para $t=0$, se obtiene

$$\begin{aligned} \xi(x_1, x_2, 0) &= \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} [A_{nm} \cos(0) + B_{nm} \sin(0/c(\lambda_{nm})^{1/2})] \zeta_{nm}(x_1, x_2) \\ &= f(x_1, x_2), \end{aligned} \quad (49)$$

pero,

$$f(x_1, x_2) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} A_{nm} \zeta_{nm}(x_1, x_2), \quad (50)$$

$$\xi_f(x_1, x_2, 0) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} B_{nm} \zeta_{nm}(x_1, x_2) = g(x_1, x_2). \quad (51)$$

Los coeficientes A_{nm} y B_{nm} son determinados usando la relación de ortogonalidad. La integral doble,

$$\iint_{\mathcal{R}} \zeta_{nm}(x_1, x_2) \zeta_{pq}(x_1, x_2) dx_1 dx_2 \quad (52)$$

es cero cuando $m \neq p$ o $n \neq q$, por lo que sólo contribuyen aquellos términos para $m=p$ y $n=q$, donde la integral tiene el valor de $ab/4$.

Multiplicamos (50) por $\zeta_{pq}(x_1, x_2)$, se obtiene

$$\iint f(x_1, x_2) \zeta_{pq}(x_1, x_2) dx_1 dx_2 = \iint \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} A_{nm} \zeta_{nm}(x_1, x_2) \zeta_{pq}(x_1, x_2) dx_1 dx_2 \quad (53)$$

$$\iint f(x_1, x_2) \zeta_{pq}(x_1, x_2) dx_1 dx_2 = \left\{ \iint \zeta_{nm}(x_1, x_2) \zeta_{pq}(x_1, x_2) dx_1 dx_2 \right\} A_{nm}. \quad (54)$$

Como sólo nos interesan los casos distintos de cero, únicamente contribuyen los términos en que $m=p$ y $n=q$,

$$\iint f(x_1, x_2) \zeta_{nm}(x_1, x_2) dx_1 dx_2 = \left\{ \iint [\zeta_{nm}(x_1, x_2)]^2 dx_1 dx_2 \right\} A_{nm}, \quad (55)$$

de donde,

$$A_{nm} = (4/ab) \iint f(x_1, x_2) \zeta_{nm}(x_1, x_2) dx_1 dx_2. \quad (56)$$

Del mismo modo obtenemos B_{nm}

$$B_{nm} = (4/ab) \iint g(x_1, x_2) \zeta_{nm}(x_1, x_2) dx_1 dx_2. \quad (57)$$

Para una región elástica como el caso de una membrana cuadrada o rectangular podemos imaginar el movimiento de un medio continuo. Puede tratarse el problema de la membrana como un conjunto de cuerdas paralelas.

Considerando al problema de la membrana cuadrada, las frecuencias no son múltiplos de

la frecuencia fundamental. Aquí debemos tomar en cuenta que existen muchos modos de vibración que están manifestándose simultáneamente. Pero varias de esas frecuencias no son múltiplos de la misma frecuencia. Considerando estas frecuencias en el tono de un tambor musical. Al menos en teoría es posible excitar únicamente un solo modo de vibración. En el problema de la cuerda las frecuencias son múltiplos de la primera frecuencia. Sin embargo en la membrana, las frecuencias de los modos normales más grandes de una membrana cuadrada no son múltiplos de la frecuencia fundamental.

Para estudiar los modos de vibración, consideramos el problema de eigenvalores anteriormente descrito cuando $a=b$, es decir, la membrana considerada es cuadrada.

Cuando $m=n$ existe solamente una eigenfunción linealmente independiente

$$\zeta_{mm}(x_1, x_2) = \text{sen}(m\pi x_1/a) \text{sen}(m\pi x_2/a), \quad (58)$$

que corresponde al eigenvalor $\lambda_{mm} = 2\pi^2 m^2 / a^2$.

Si $m \neq n$ hay dos eigenfunciones linealmente independientes

$$\zeta_{nm}(x_1, x_2, 0) = \text{sen}(n\pi x_1/a) \text{sen}(m\pi x_2/a), \quad \zeta_{mn}(x_1, x_2) = \text{sen}(m\pi x_1/a) \text{sen}(n\pi x_2/a), \quad (59)$$

que corresponde al mismo eigenvalor

$$\lambda_{nm} = \lambda_{mn} = (m^2 + n^2)\pi^2 / a^2. \quad (60)$$

Un modo de vibración de la membrana correspondiente al eigenvalor λ_{nm} es $\xi_{nm}(x_1, x_2)$ tomando $B_{nm} = 0$ para simplificación. Por lo que un modo normal correspondiente a la membrana para un eigenvalor λ_{nm} es

$$\xi_{nm}(x_1, x_2, t) = \zeta_{nm}(x_1, x_2) \cos(ct(\lambda_{nm})^{1/2}). \quad (61)$$

De esta última relación podemos obtener los diversos modos de oscilación. La membrana tiene un movimiento armónico simple en cada punto, con frecuencia

$$f_{nm} = c(\lambda_{nm})^{1/2} / 2\pi = c(m^2 + n^2)^{1/2} / 2a. \quad (62)$$

La amplitud es la eigenfunción ζ_{nm} en cada punto. La frecuencia más baja es llamada

frecuencia fundamental de la membrana y está dada por,

$$f_{11} = c(2)^{1/2}/2a. \quad (63)$$

Las curvas nodales resultan de la siguiente ecuación

$$\zeta_{nm}(x_1, x_2) = 0, \quad (64)$$

donde ζ representa la amplitud del movimiento de los puntos (x_1, x_2) de la membrana. Los puntos en la membrana que corresponden a las curvas nodales, permanecen estacionarios.

Consideremos como primer ejemplo la ecuación de onda,

$$\xi_{x_1 x_1}(x_1, x_2, t) + \xi_{x_2 x_2}(x_1, x_2, t) = \xi_{tt}(x_1, x_2, t), \quad (65)$$

y las condiciones iniciales siguientes:

$$\xi(x_1, x_2, 0) = \text{sen}(\pi x_1) \text{sen}(\pi x_2), \quad (66)$$

$$\xi_t(x_1, x_2, 0) = 0, \quad (67)$$

para $x_1 \geq 0, x_2 \leq 1, t=0$. La condición de frontera es,

$$\xi(x_1, x_2, t) = 0, \quad (68)$$

sobre la frontera de un cuadrado unitario, para $t > 0$. Por el método de separación de variables que descrito anteriormente se obtiene,

$$\xi_{nm}(x_1, x_2, t) = \text{sen}(n\pi x_1) \text{sen}(m\pi x_2) \cos[(m^2 + n^2)^{1/2} \pi t]. \quad (69)$$

Calculando los coeficientes A_{nm} y B_{nm} se obtiene la solución

$$\xi(x_1, x_2, t) = \text{sen}(\pi x_1) \text{sen}(\pi x_2) \cos[(2)^{1/2} \pi t]. \quad (70)$$

II.2 Método numérico

Describiremos el método numérico para resolver la ecuación de onda bidimensional. Se utilizará un código procedural para resolver un ejemplo particular para una membrana cuadrada.

II.2.1 Método explícito de diferencias finitas

Consideremos una ecuación hiperbólica lineal homogénea arbitraria de segundo orden

$$\xi_{tt} - A\xi_{x_1x_1} - 2B\xi_{x_1x_2} - C\xi_{x_2x_2} + D\xi_t + E\xi_{x_1} + F\xi_{x_2} + G\xi = 0, \quad (71)$$

donde $\xi = \xi(x_1, x_2, t)$ y los coeficientes A, B, C, D, E, F y G , son constantes o funciones de las variables independientes t, x_1 y x_2 , $A > 0, C > 0$ y $AC - B^2 > 0$. En particular la ecuación de onda bidimensional es

$$\xi_{tt} = c^2(\xi_{x_1x_1} + \xi_{x_2x_2}). \quad (72)$$

Las condiciones iniciales están dadas por,

$$\xi(x_1, x_2, 0) = f(x_1, x_2), \quad (73)$$

$$\partial \xi(x_1, x_2, 0) / \partial t = g(x_1, x_2), \quad (74)$$

para $x_1 > -\infty$ y $x_2 < +\infty$. Uno de los métodos más comunes de aproximación es el método de diferencias finitas. Dicho método será utilizado para resolver la ecuación de onda sobre un dominio regular en dos dimensiones. La aproximación explícita es

$$U^{n+1}_{l,j} = 2(1 - 2p^2)U^n_{l,j} + p^2c^2(U^n_{l+1,j} + U^n_{l-1,j} + U^n_{l,j+1} + U^n_{l,j-1}) - U^{n-1}_{l,j}, \quad (75)$$

donde $p = k/h$, $k = \Delta t$, $h = \Delta x_1 = \Delta x_2$. El espacio y el tiempo se determinan por $x_1 = ih$, $x_2 = jh$ y $t = nk$, donde i, j y n son enteros positivos. Con esta fórmula obtenemos el valor de ξ en un punto (X_1, X_2, T) de la malla.

La parte principal del error de truncamiento se obtiene a partir del teorema de Taylor

alrededor de un punto (lh, jh, nk) , es

$$(1/12) k^2 h^2 \{ (p^2 - 1) [U_{x_1 x_1 x_1} + U_{x_2 x_2 x_2}] + 2p^2 U_{x_1 x_1 x_2} \}. \quad (76)$$

Podemos aproximar los valores de arranque de U en los planos iniciales por,

$$U^0_{l,j} = f(lh, jh) \quad (77)$$

$$U^1_{l,j} = (1 - 2p^2) f(lh, jh) + (1/2)p^2 [f((l+1)h, jh) + f((l-1)h, jh) + f(lh, (j+1)h) + f(lh, (j-1)h)] + kg(lh, jh), \quad (78)$$

en $t=0$ y $t=k$ respectivamente. La estabilidad del método para la Ec. (72), está sujeta a la condición de Courant-Friedrichs-Lewy, la cual es,

$$p \leq (2)^{1/2} \quad (79)$$

Como ejemplo resolveremos un problema para una membrana cuadrada representado por,

$$\xi_{tt}(x_1, x_2, t) = \xi_{x_1 x_1}(x_1, x_2, t) + \xi_{x_2 x_2}(x_1, x_2, t). \quad (80)$$

Las condiciones iniciales son:

$$\begin{aligned} \xi(x_1, x_2, 0) &= \sin(\pi x_1) \sin(\pi x_2) \\ \xi_t(x_1, x_2, 0) &= 0 \end{aligned} \quad (81)$$

Las condiciones de frontera están dadas por,

$$\xi(x_1, x_2, t) = 0, \quad (82)$$

en una región dada por $\{(x_1, x_2) | 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$, para $t > 0$.

Consideremos una malla cuadrada unitaria, donde $h = \Delta x_1 = \Delta x_2 = .02$, $k = .01$ y $p = k/h = .5$. Particularmente la tensión y densidad de área son unitarias, por lo que $c = 1.0$.

II.2.2 Código procedural

Para resolver el ejemplo de la sección anterior utilizamos un programa procedural de nombre MEDF (Método Explícito de Diferencias Finitas) para obtener la solución numérica.

El código hace una aproximación de segundo orden para el tiempo. Los tiempos o planos

iniciales U^{n-1} y U^n son representados por los arreglos U_{pasado} y U_{presente} respectivamente. La solución se actualiza en U_{futuro}. La Fig. 1 es una representación gráfica de la solución generada en el instante de tiempo $t = 1.0$. Las constantes del problema se especifican en un archivo de cabecera ("constantes.h"). El código procedural es el siguiente:

//CÓDIGO PROCEDURAL "MEDF"

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "constantes.h"

typedef double TYPE;
typedef TYPE **OBJECT;
void allocate (OBJECT);

int onda () {

    OBJECT Upasado, Upresente, Ufuturo;
    Upasado=(OBJECT) calloc (rows, sizeof (TYPE *));
    allocate (Upasado);
    Upresente=(OBJECT) calloc (rows, sizeof (TYPE *));
    allocate (Upresente);
    Ufuturo=(OBJECT) calloc (rows, sizeof (TYPE *));
    allocate (Ufuturo);

    for (j=0; j<ny; j++)
        for (i=0; i<nx; i++) {
            Upasado[j][i]=sin(M_PI*i*h)*sin(M_PI*j*h);
        }
    for (j=1; j<ny-1; j++)
        for (i=1; i<nx-1; i++){
            Upresente[ny-1][i]=0.0;
            Upresente[j][nx-1]=0.0;
            Upresente[0][i]=0.0;
            Upresente[j][0]=0.0;

            Upresente[j][i]=(1-2*p*p)*Upasado[j][i]+.5*p*p*(Upasado[j][i+1]
                +Upasado[j][i-1] + Upasado[j+1][i] + Upasado[j-1][i]);
        }
}
```

```

for (int n=2; n=100; n++) {
for (j=1; j<ny-1; j++)
for (i=1; i<nx-1; i++){

    Ufuturo[ny-1][i]=0.0;
    Ufuturo[j][nx-1]=0.0;
    Ufuturo[0][i]=0.0;
    Ufuturo[j][0]=0.0;

    Ufuturo[j][i]= (2. -4.*p*p)*Upresente[j][i] +p*p*c*(Upresente[j][i+1]+Upresente[j][i-1]
+Upresente[j+1][i] + Upresente[j-1][i] -Upasado[j][i];
    }
//ACTUALIZACIÓN.

for (j=0; j<ny; j++)
for (i=0; i<nx; i++){
    Upasado[j][i]=Upresente[j][i];
    Upresente[j][i]=Ufuturo[j][i];
    }
}

    FILE *fptr;
    fptr = fopen ("MEDF.out", "w");
    for (j=0; j<ny; j++){
        fprintf (fptr, "\n\n");
        for ( i=0; i<nx; i++) fprintf (fptr, "%10.5f ",
            Ufuturo[j][i] );
        fclose (fptr);
        de_allocate(Upasado);
        de_allocate(Upresente);
        de_allocate(Ufuturo);
        return 0;
    }

void de_allocate (OBJECT x)
{
    unsigned int i;
    for (i=0; i<rows; i++)
        free ((char*) x[i]);
    free((char*) x);
}

int main ()
{
    onda();
    return 0;
}

```

III MODELO DE OBJETOS

El siguiente paso es hacer un modelo de objetos que fundamenta el diseño orientado a objetos. El modelo utiliza cuatro principios básicos: abstracción, encapsulación, módulos y jerarquías. La abstracción significa pensar en términos de conceptos u objetos. Con la abstracción se extraen las características relevantes y comunes del sistema a modelar. La encapsulación consiste en edificar bloques de información. Los clientes no ven la estructura interna en que se implanta el comportamiento del objeto. Es decir, la mayor parte de la complejidad respecto a la implantación de los objetos se oculta. Las interfaces de una clase capturan únicamente la vista externa.

Un sistema puede descomponerse en bloques, paquetes o módulos. En un paquete puede haber varios objetos que interactúan intercambiando mensajes. La labor más importante es definir cuáles serán los módulos o paquetes apropiados para simular el sistema. En un grupo de objetos se pueden desarrollar jerarquías de clases, y además es fundamental en OOP.

Algunos conceptos fundamentales en OOP son: clases, objetos, mensajes, métodos, herencia y polimorfismo. Las clases "encapsulan" datos y funciones. Éstas definen una categoría específica de objetos. Los objetos interactúan por medio de un intercambio de "mensajes" generando un comportamiento colectivo. La respuesta del objeto ante un mensaje es llamar a una función miembro o método apropiado. De esta forma se establece la comunicación entre objetos. Regularmente la información sobre el funcionamiento de los objetos es desconocida por otros. Un objeto tiene los mismos atributos y funcionalidades con otros*

* Los términos "método y función miembro" son equivalentes.

objetos de la misma categoría o clase. Las clases nos permiten modelar objetos que poseen atributos (datos) y comportamientos (funciones miembro u operaciones). Herencia significa que una clase se puede obtener de otra, es decir, que todas las características o parte de ellas pueden ser transmitidas de clase en clase. Polimorfismo permite que diferentes tipos de objetos que comparten algo en común puedan usarse en esa parte del programa que requiere ese comportamiento común. Estos conceptos son sustentados por el lenguaje C++, el cual es el más usado para hacer OOP.

Esta nueva forma de programar representa una ventaja sobre la forma tradicional de programación, permite por ejemplo que la escritura de códigos de muchas líneas (del orden de 10^5 o mayor) sea más eficiente. A la fecha, no se ha estudiado sistemáticamente la importancia de la OOP en las aplicaciones científicas, aunque en otras ramas del conocimiento esta nueva tecnología ha tenido una amplia aceptación.

La forma usual de programar consiste en decidir el procedimiento y utilizar el mejor algoritmo disponible. Esta metodología ha dado excelentes resultados como se puede comprobar en los avances tecnológicos de las últimas décadas. OO es una técnica de programación nueva diseñada especialmente para el modelado de sistemas. En programación orientada a objetos se trabaja con módulos o paquetes que contienen la información requerida. Con OOP es posible utilizar las tecnologías de vanguardia: computación paralela y realidad virtual, ésta última con enorme potencial de aplicaciones. Hay que resaltar que los proyectos de gran envergadura, requieren de muchas de líneas de código. Usando OOP se pueden organizar esas líneas de código por clases, lo cual facilita su manejo.

Cada paquete "encapsula" datos (atributos) y funciones (comportamientos u operaciones). Particularmente, dichas funciones están asociadas a los conjuntos de condiciones de frontera, condiciones iniciales y el método de diferencias finitas explícito y la función de actualización.

Algunos datos como la constante c (que está relacionada con la tensión y densidad por unidad de área de la membrana), el tamaño del paso de las coordenadas, entre otros, se integran en la clase "ConstantesFisicas".

Antes de hacer OOP se seleccionaron ejemplos comunes en la enseñanza de la ecuación de onda aplicada a una membrana cuadrada y rectangular. Estos ejemplos se probaron en el programa procedural MEDF. Con la técnica de OO integramos el sistema para que funcione como un tutorial para el usuario. Los resultados obtenidos del programa de objetos se comparan previamente con los obtenidos por MEDF. Los ejemplos se muestran en el siguiente capítulo.

IV RESULTADOS

IV.1. Código orientado a objetos.

Para crear el código orientado a objetos declaramos entre otras, las siguientes clases;

***class** CondicionesFrontera,*

***class** CondicionesIniciales,*

***class** DiferenciasFinitas,*

***class** Actual,*

***class** ConstantesFisicas.*

Estas clases se declaran y definen en los archivos constantes.ll y update.ll. Los archivos matrix.ll y vector.ll son adaptados y tomados de Barton y Nackman (2). Estos archivos contienen a las clases "SimpleFloatArray" y "matrix" para arreglos unidimensionales y matrices respectivamente.

//CÓDIGO ORIENTADO A OBJETOS: "TUTORIAL".

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include "constantes.ll"
#include "vector.ll"
#include "matrix.ll"
#include "update.ll"
```

```
matrix Upasado (51, 51), Ufuturo (51, 51), Upresente (51, 51);
```

```
int onda () {
```

```
    CondicionesIniciales *CI;
    CI = new CondicionesIniciales;
```

```

DiferenciasFinitas *FD;
FD = new DiferenciasFinitas;

CondicionesFrontera *CF;
CF = new CondicionesFrontera;

Actual *UP;
UP = new Actual;

CI -> SenpIxSenply (Upasado);

CI -> Present (Upasado, Upresente);

for (int n=2; n=100; n++) {

    FD -> Explicito (Upasado, Upresente, Ufuturo);

    CF -> Cauchy (Ufuturo);

    UP -> Actualizacion (Ufuturo, Upasado, Upresente);

    }

    FILE *fptr;
    fptr = fopen ("TUTORIAL.out", "w");
    for(int j=0; j<ny; j++){

        fprintf (fptr, "\n\n");
        for (int i=0; i<nx; i++) fprintf (fptr, "%15.5f",
            Ufuturo.mval (j,i));

        fprintf (fptr, "\n");
        fclose (fptr);
        return 0;
    }

    int main ()
    {

        onda();
        return 0;

    }

```

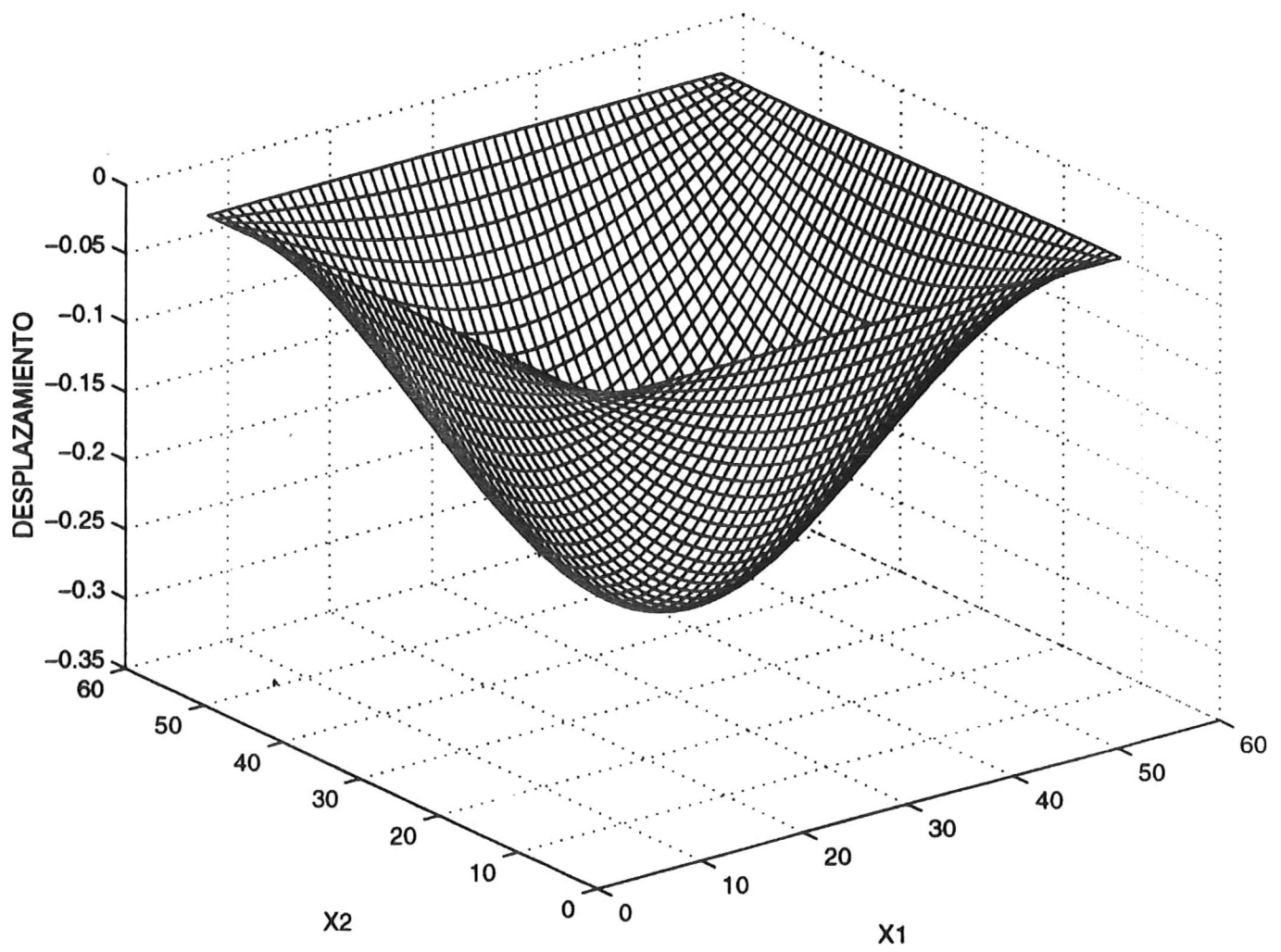


Fig. 1 Representación de la solución del programa MEDF.

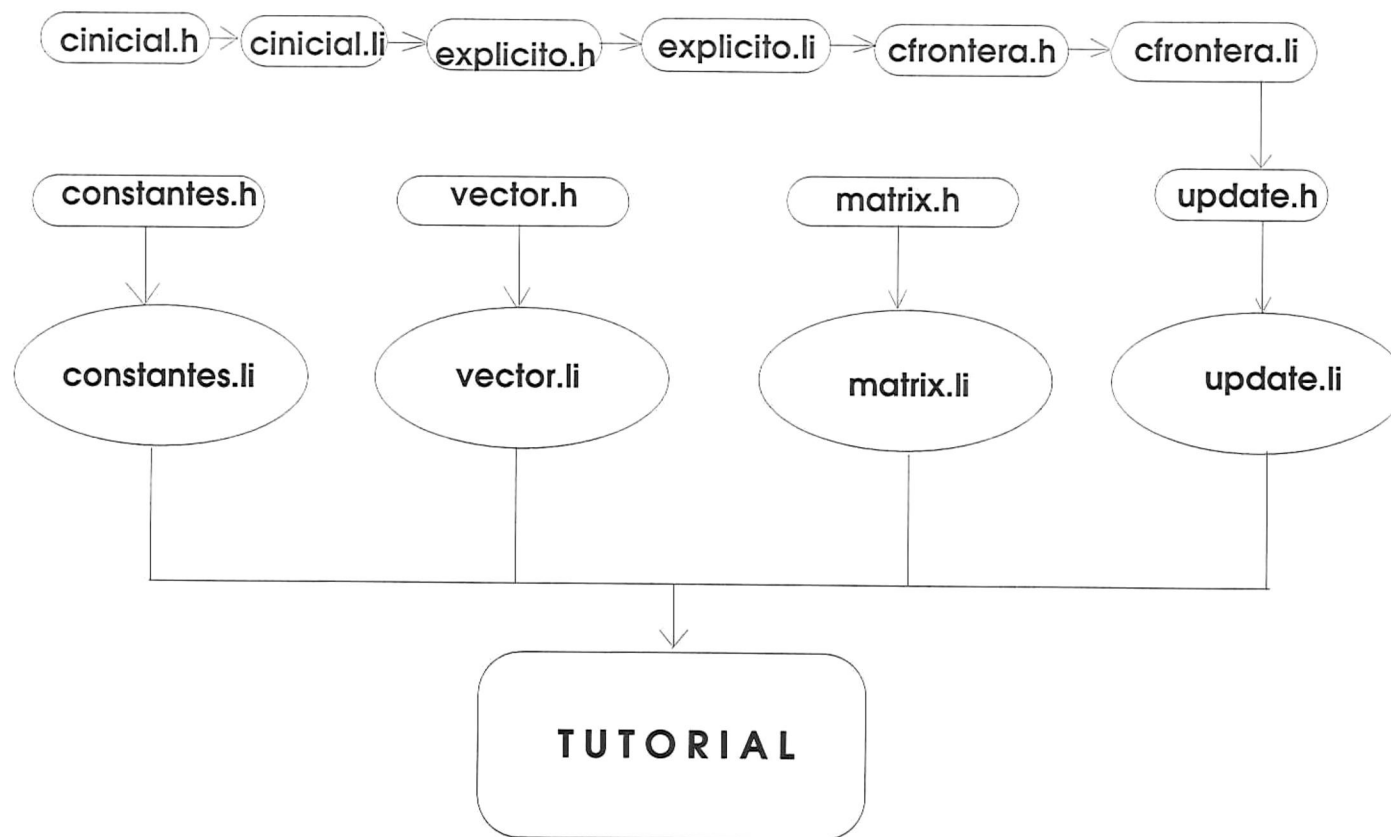


FIG. 2 Archivos de cabecera del programa TUTORIAL.

Los archivos de cabecera del programa TUTORIAL se encuentran ubicados como se muestra en la Fig. 2 y se describen enseguida. La implantación puede verse en los anexos.

//constantes.h

```
extern const int nx;
extern const int ny;
extern const double T;
extern const double sigma;
extern const double h;
extern const double p;
extern const double k;

class ConstantesFisicas {

public:

    static const int nx;
    static const int ny;
    static const double T;
    static const double sigma;
    static const double h;
    static const double p;
    static const double k;

};
```

//vector.h

```
/*
Example programs from the book Scientific and Engineering Programming
in C++: An Introduction with Advanced Techniques and Examples,
Addison-Wesley, 1994.
```

(c) COPYRIGHT INTERNATIONAL BUSINESS MACHINES
CORPORATION 1994. ALL RIGHTS RESERVED.

```
See README file for further details.
```

```
*/
```

```
class SimpleFloatArray {
```

public:

```
SimpleFloatArray(int n);
SimpleFloatArray();
SimpleFloatArray(const SimpleFloatArray);
~SimpleFloatArray();
float& operator[] (int i);
int numEls();
SimpleFloatArray& operator=(const SimpleFloatArray&);
SimpleFloatArray& operator=(float);
void setSize(int n);
```

private:

```
int num_elts;
float* ptr_to_data;
void copy (const SimpleFloatArray& a);
};
```

//matrix.h

```
#ifndef MATRIX
#define MATRIX
```

```
class matrix {
    struct matrep {
        double **m;
        int r, c;
        int n;
    } *p;
    void error (char *msg1, char *msg2 = " ");
```

public:

```
matrix (int mrows = 1, int columns = 1, double initval = 0);
matrix (int mrows, int columns, double* initvalues);
matrix (char* matfile);
matrix (matrix& x);
~matrix();
int rows() const { return p->r;};
int cols() const {return p->c;};
matrix operator=(const matrix& rval);
void write_file(char *filename, char *msg = " ");

matrix operator+ (matrix& rval);
matrix operator+ (const double rval);
```

```

        double & val (int row, int col);
        double & mval(int row, int col) {
            return (p->m[row][col]);
        }

        void print (char * msg = " ");
    };

#endif

```

//update.h

```

#ifndef UPDAT
#define UPDAT
#include "cfrontera.li"

class Actual {

public:

    void Actualizacion (matrix &Ufuturo, matrix &Upasado, matrix &Upresente);
};

#endif

```

//cfrontera.h

```

#ifndef FRONTERA
#define FRONTERA
#include "explicito.li"

class CondicionesFrontera {

public:

    void Cauchy (matrix &Ufuturo);
};

#endif

```

//explicito.h

```

#include "cinicial.li"
#ifndef EXPLI
#define EXPLI
class DiferenciasFinitas : public CondicionesIniciales
{
    private:
        static double c;
    public:
        void Explicito (matrix &Upasado, matrix &Upresente, matrix &Ufuturo);
};

#endif

```

```

//cinicial.h

```

```

#ifndef INICIAL
#define INICIAL

class CondicionesIniciales{

public:

    void SenpixSenply(matrix &Upasado);

    void SenpixSen2ply(matrix &Upasado);

    void Sen8pixSenply(matrix &Upasado);

    void xSEXTAy(matrix &Upasado);

    void x2MASy2(matrix &Upasado);

    void Cos10pixSen3ply(matrix &Upasado);

    void Algebraico (matrix &Upasado);

    void Constante (matrix &Upasado);

    void Present (matrix &Upasado, matrix &Upresente);

};

#endif

```


Con el especificador de acceso precedido **public** : cualquiera de las condiciones iniciales definidas por las funciones miembro declaradas es accesible a cualquier función. Estos servicios públicos son la interfaz pública de la clase.

En el archivo de cabecera *explicito.h* la clase base *CondicionesIniciales* hereda sus miembros públicos, que representan el conjunto de condiciones iniciales. La clase derivada *Diferencias Finitas* contiene la función del método usado para resolver el problema de la membrana. Este procedimiento se conoce como herencia pública y se realiza dado que el método de diferencias finitas puede ser usado por cualquiera de las funciones que se encuentran en la clase base. La herencia pública se especifica como

```
class DiferenciasFinitas : public CondicionesIniciales {
    .....
};
```

El archivo *update.ll* (Anexo VIII.4) obtiene a siete archivos que se representan en la Fig. 2. En éstos se declaran y definen las condiciones iniciales, condiciones de frontera y el método explícito de diferencias finitas.

IV.2 Ejemplos

Algunas soluciones generadas por el programa TUTORIAL de la sección anterior se presentarán en esta sección. El primer ejemplo (Fig. 3) se plantea como sigue

$$C. I. : \xi(x, y, 0) = \sin(\pi x) \sin(\pi y), \quad \xi_t(x, y, 0) = 0.$$

$$C. F. : \xi(x, y, t) = 0.$$

$$REGION: \{(x, y) | 0 \leq x \leq 1, 0 \leq y \leq 1\}$$

$$c = 1.0.$$

El programa llama este ejemplo por

CI -> SenpixSenply (Upasado);

FD -> Explicito (Upasado, Upresente, Ufuturo);

CF -> Cauchy (Ufuturo);

UP -> Actualizacion (Ufuturo, Upasado, Upresente);

Con estas líneas el usuario define el primer ejemplo. En este ejemplo se trata el problema de una membrana cuadrada unitaria.

En el segundo ejemplo (Fig. 4) queremos resaltar la influencia que tiene la curva que limita la frontera en el comportamiento de la membrana. Se utilizó el mismo ejemplo anterior, sujeto a una frontera de forma rectangular. Los lados del rectángulo están dados por $a=1$ y $b=1/2$.

Para generar la solución del tercer ejemplo (Fig. 5), se sustituye la línea

CI -> SenpixSen2ply (Upasado);

en el programa TUTORIAL. La condición inicial es uno de los posibles modos normales de oscilación del primer ejemplo. Es una membrana cuadrada unitaria con condiciones de frontera de Cauchy. Otros modos normales se utilizan para obtener el ejemplo de la Fig. 6. La Fig. 12 es otro ejemplo con condición inicial de tipo trigonométrico.

Las Figs. 7, 8, 10 y 11 se sujetan a condiciones iniciales de tipo algebraico. La Fig. 9 es una representación de un problema restringido a una condición inicial de tipo constante e igual a la unidad. En el ejemplo de la Fig. 7 se utilizan los siguientes valores constantes: $T=10^6$, $\sigma=1.0$ y $c=10^3$. Por esta razón el período de vibración se reduce significativamente respecto a los demás casos, donde se ha considerado $c=1$.

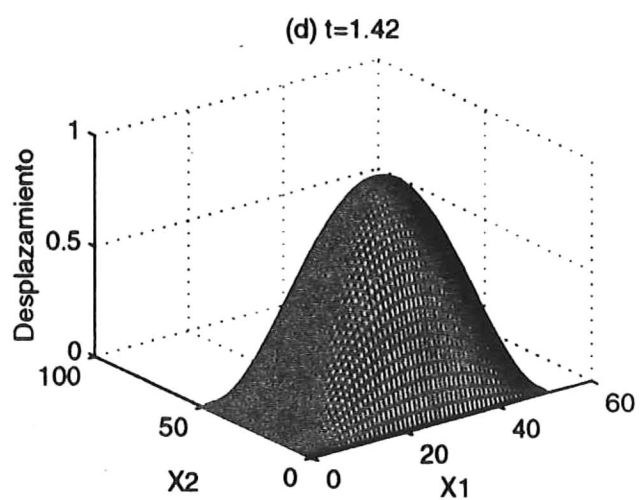
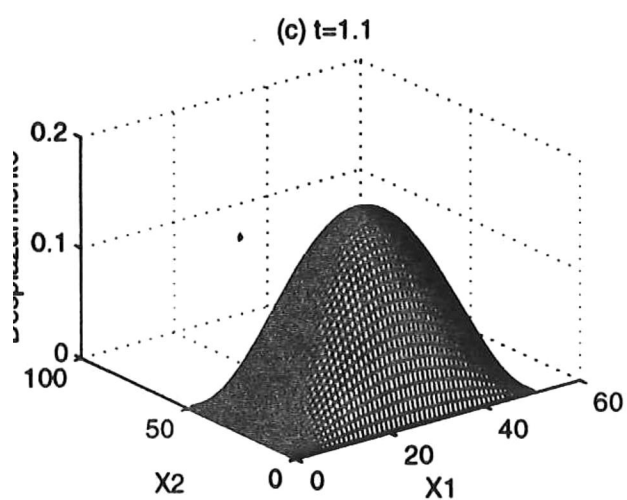
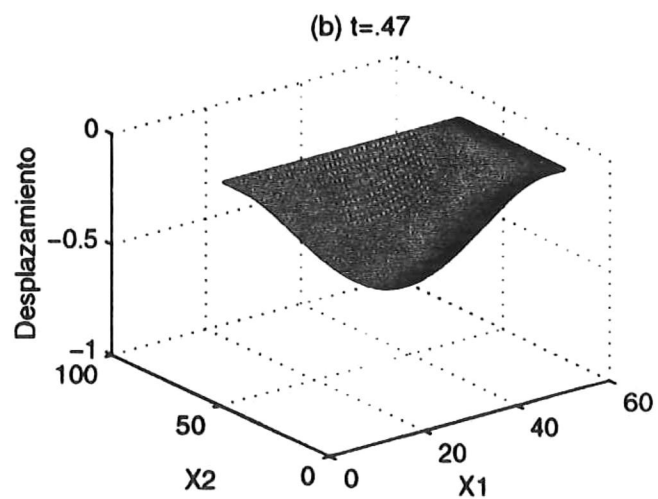
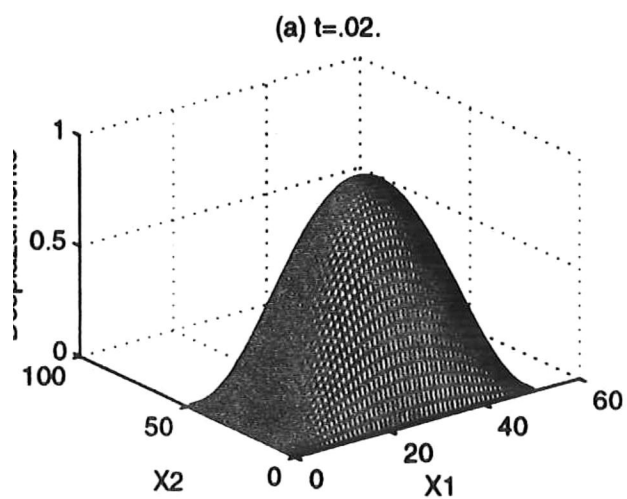


Fig. 3 Ejemplo 1. Membrana cuadrada unitaria con condición inicial: $\sin\pi x_1 \sin\pi x_2$

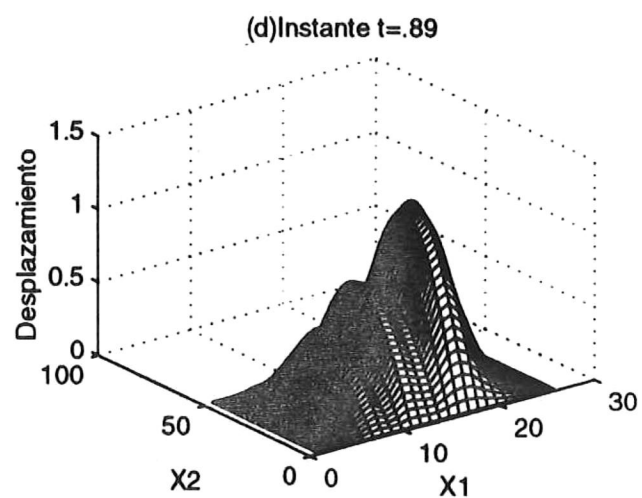
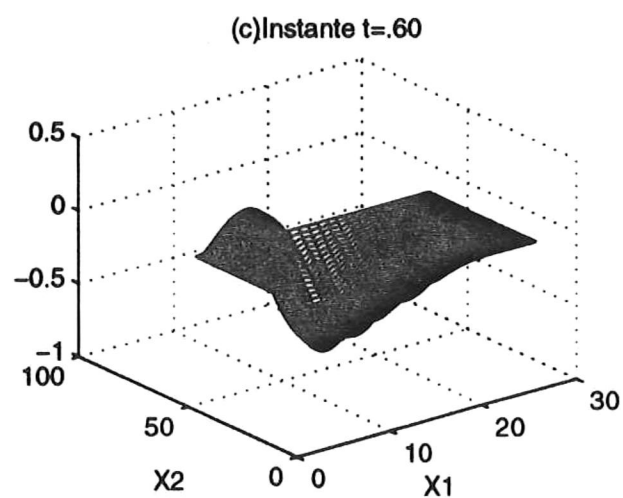
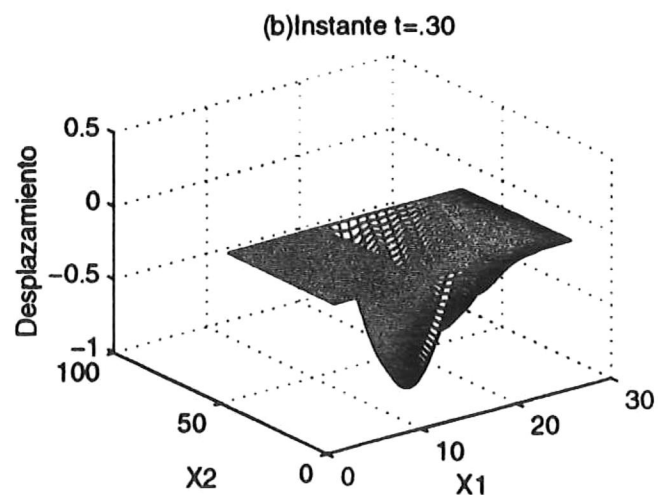
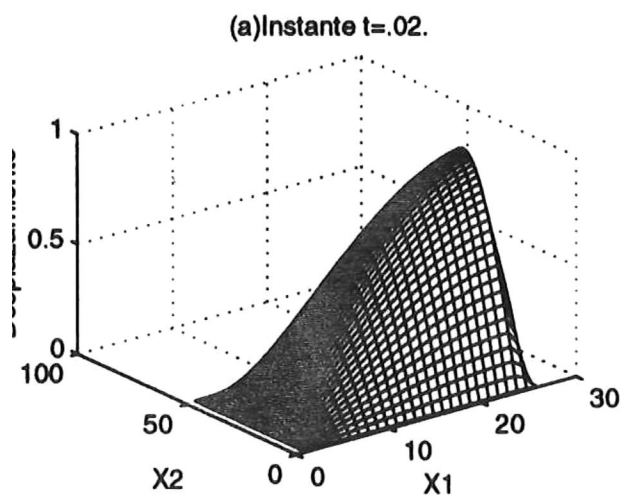


Fig. 4 Ejemplo 2. Membrana rectangular con condición inicial: $\text{sen}\pi x_1 \text{sen}\pi x_2$.

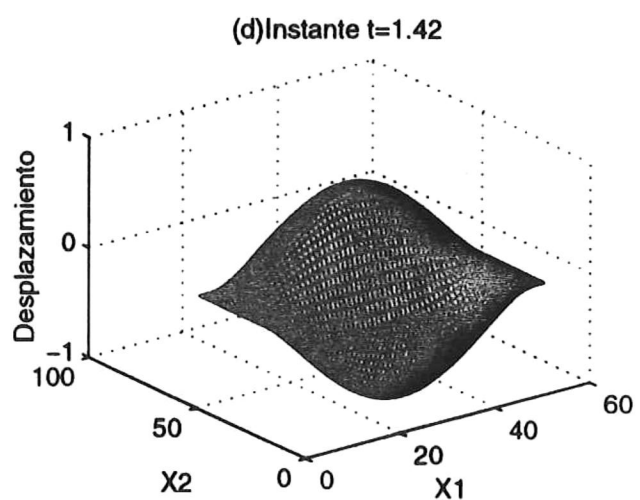
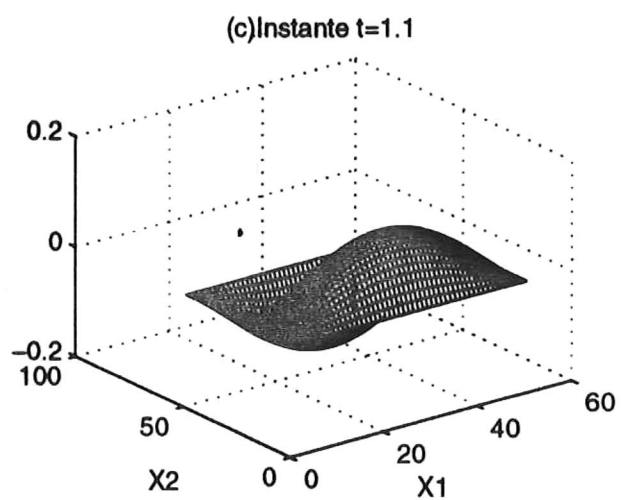
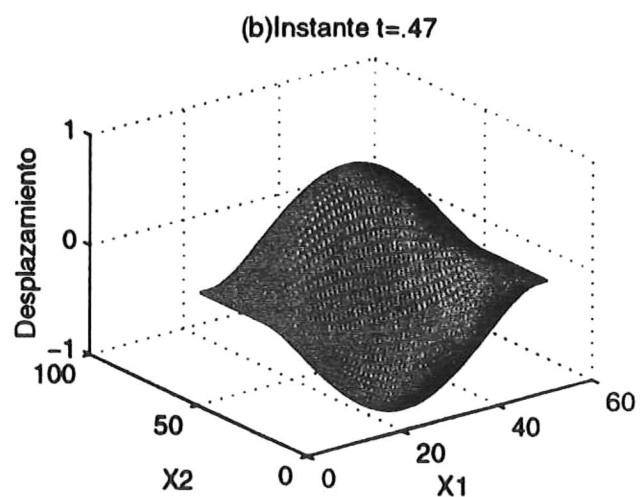
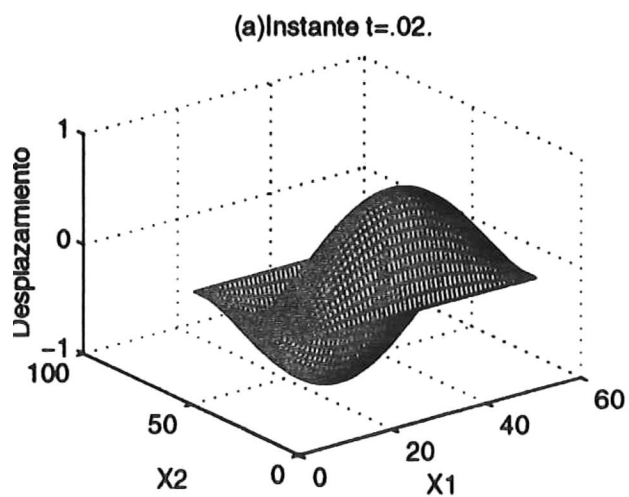


Fig. 5 Ejemplo 3. Membrana cuadrada con condición inicial: $\sin \pi x_1 \sin 2\pi x_2$.

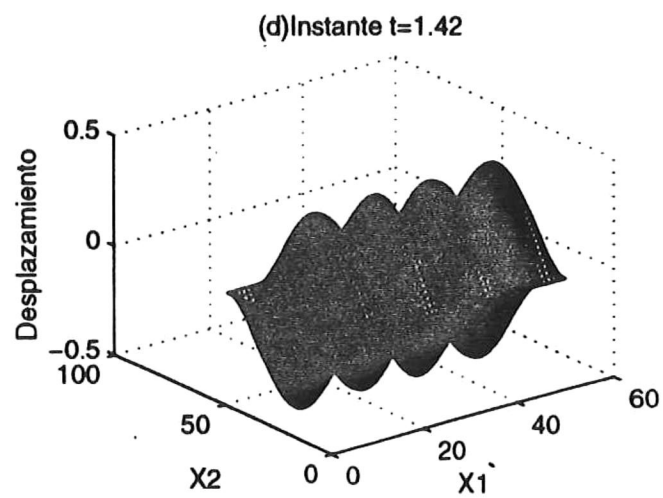
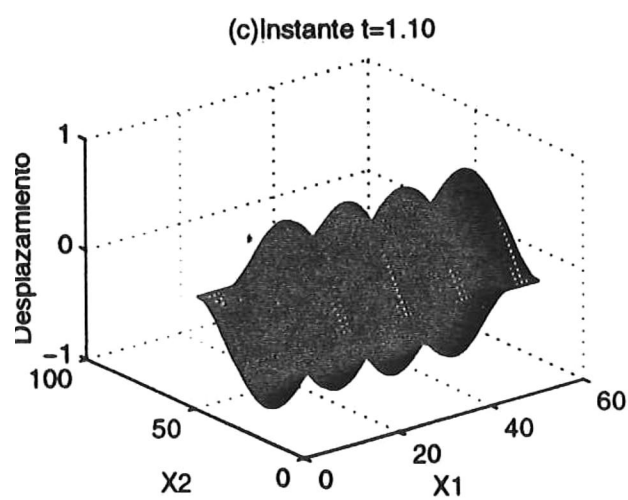
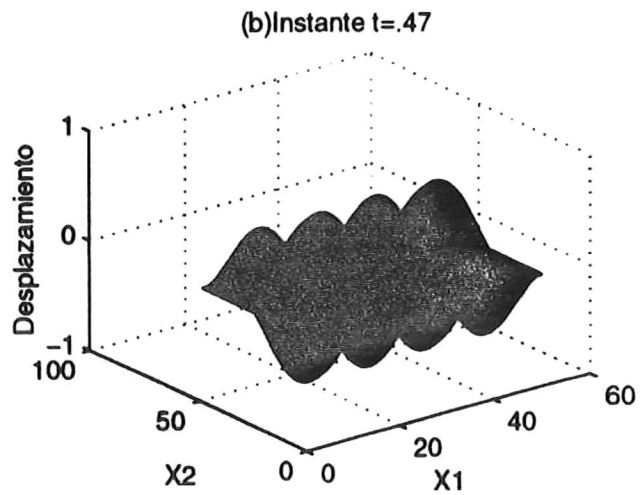
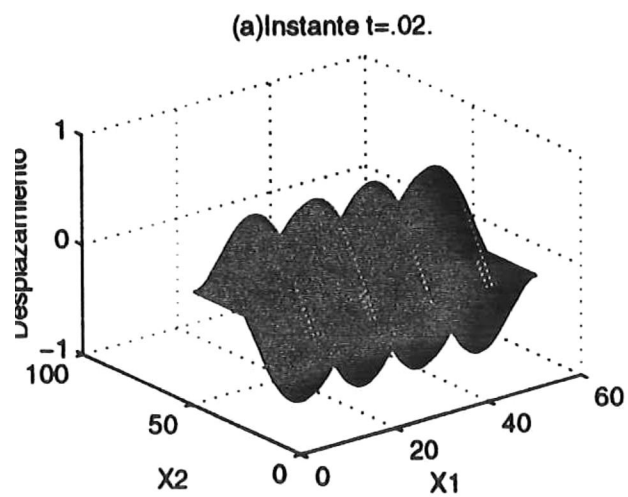


Fig. 6 Ejemplo 4. Membrana cuadrada con condición inicial: $\sin 8\pi x_1 \sin \pi x_2$.

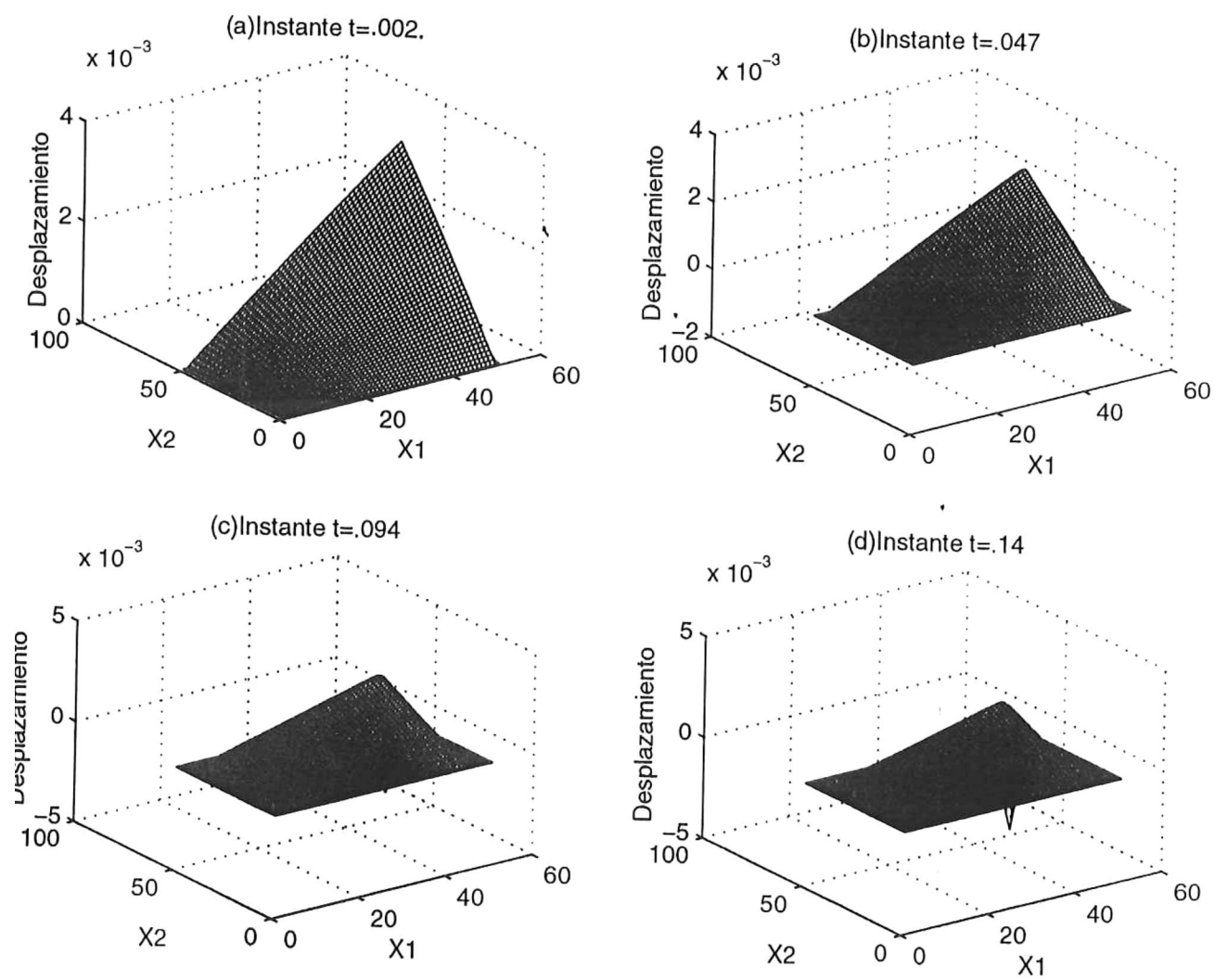


Fig. 7 Ejemplo 5. Membrana cuadrada con condición inicial: $10^{-5} x_2 (20-x_2) x_1 (20-x_1)$.

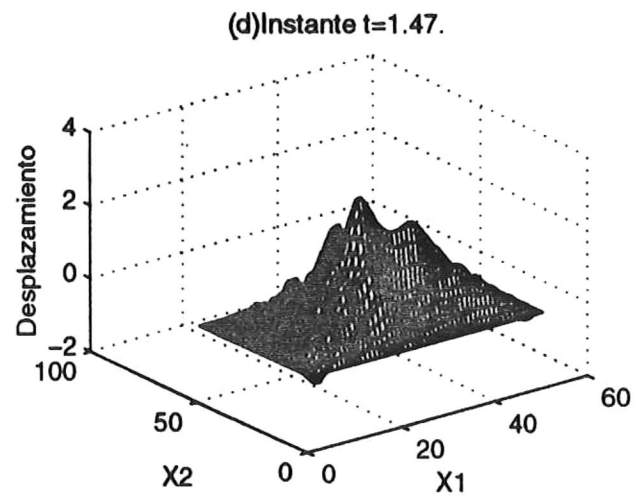
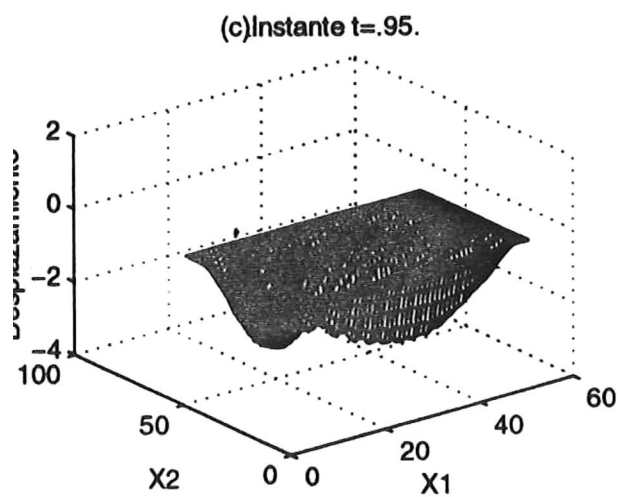
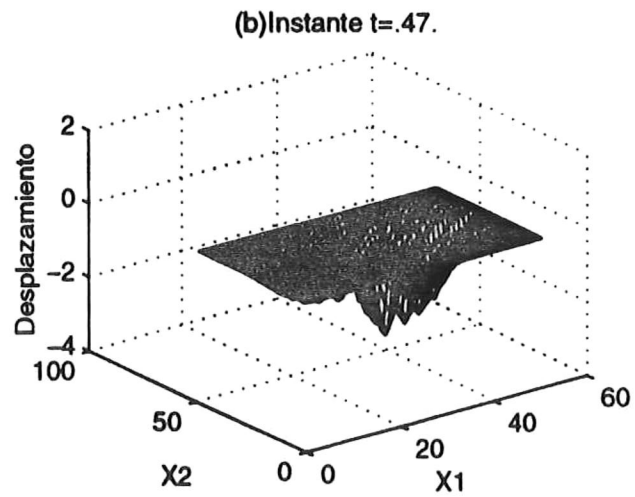
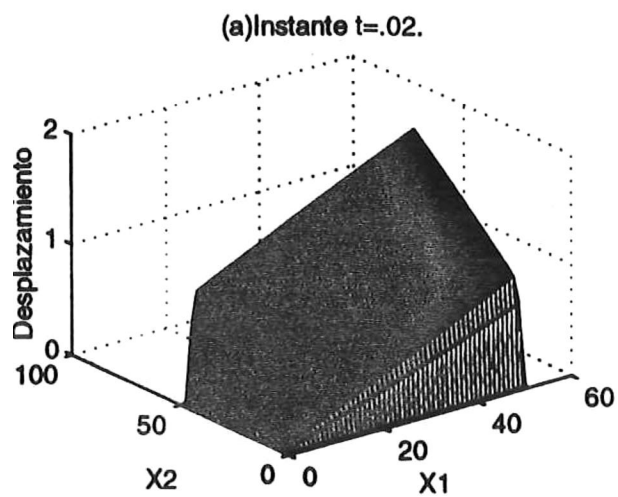


Fig. 8 Membrana cuadrada con el plano inicial: $x + y$.

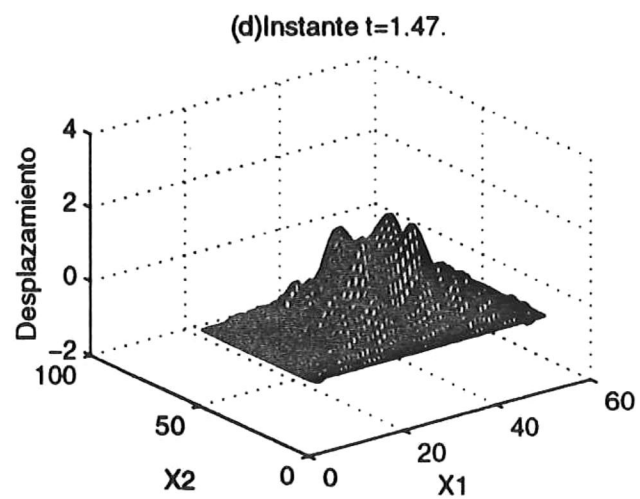
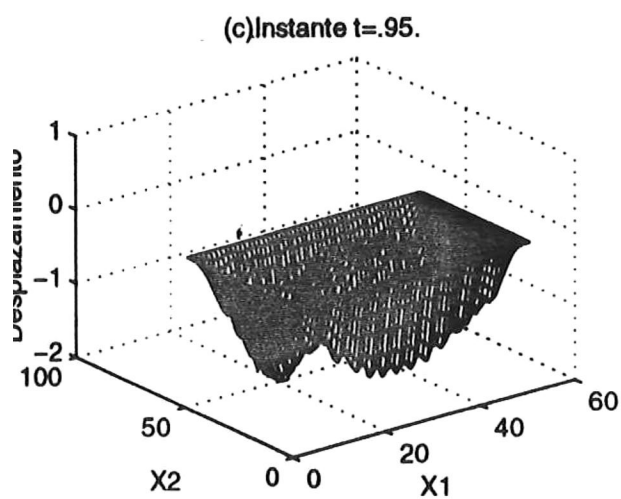
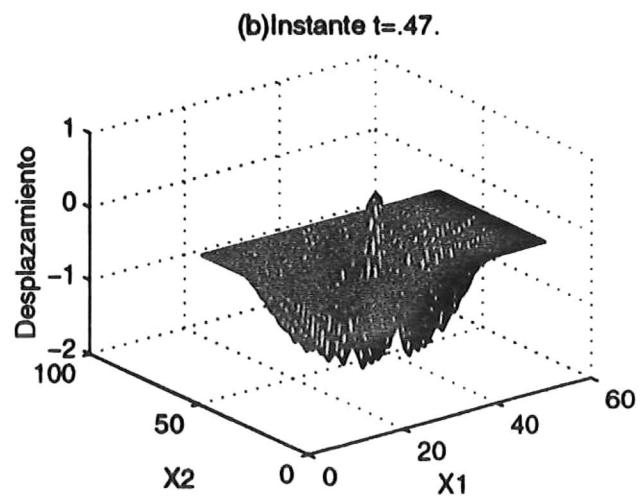
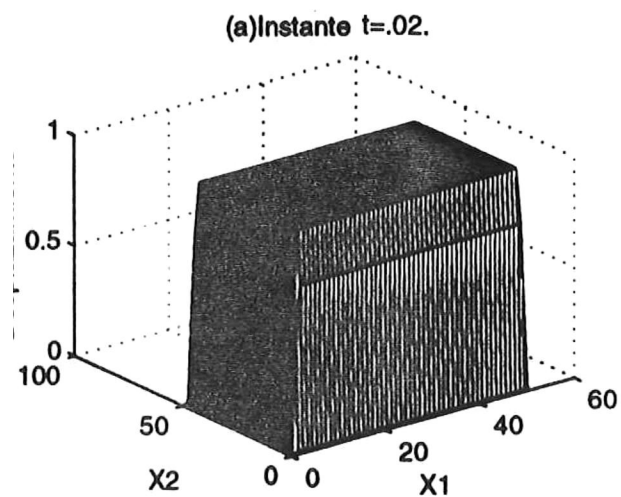


Fig. 9 Membrana cuadrada con condición inicial $f(x,y)=1$.

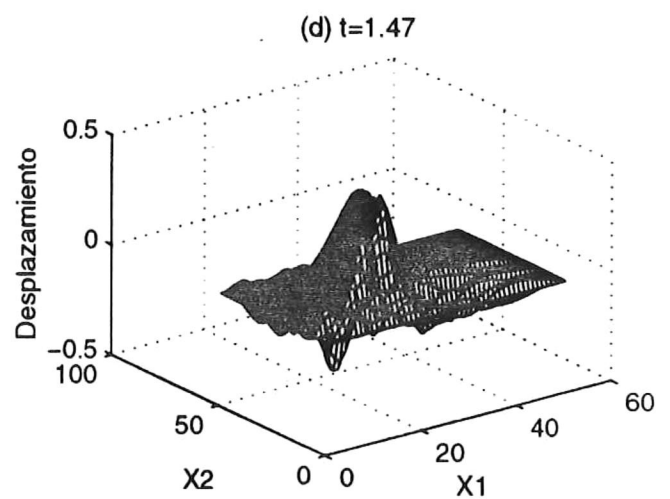
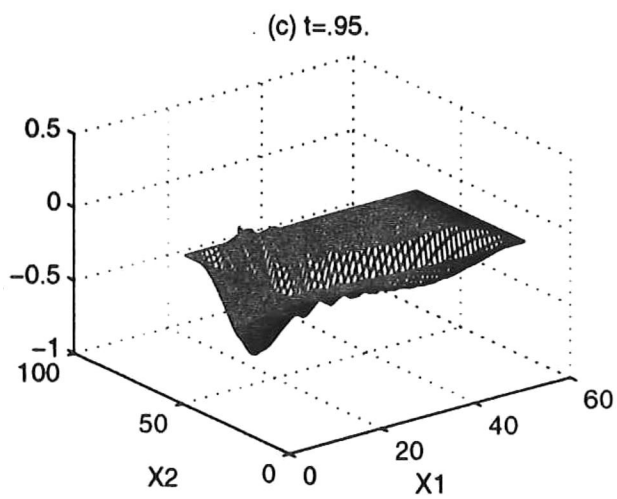
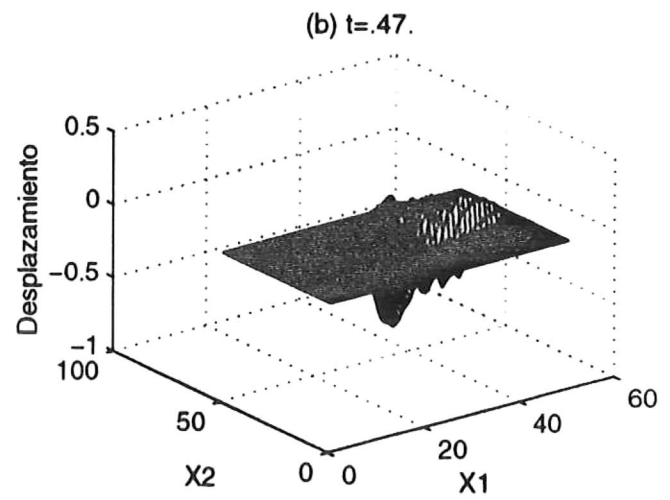
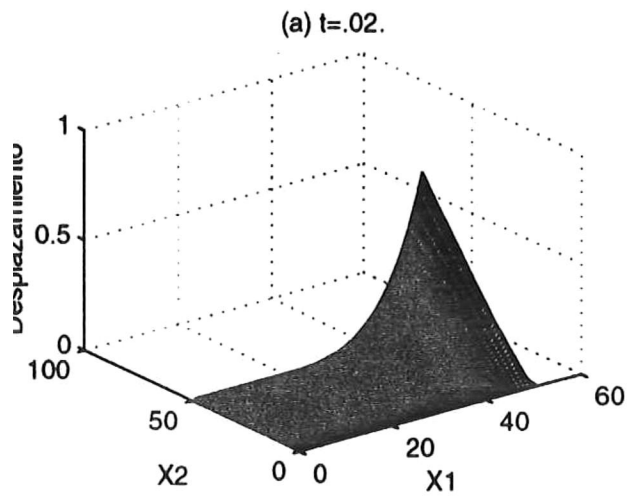


Fig. 10 Ejemplo 8. Membrana cuadrada con forma inicial $x_1^6 x_2$

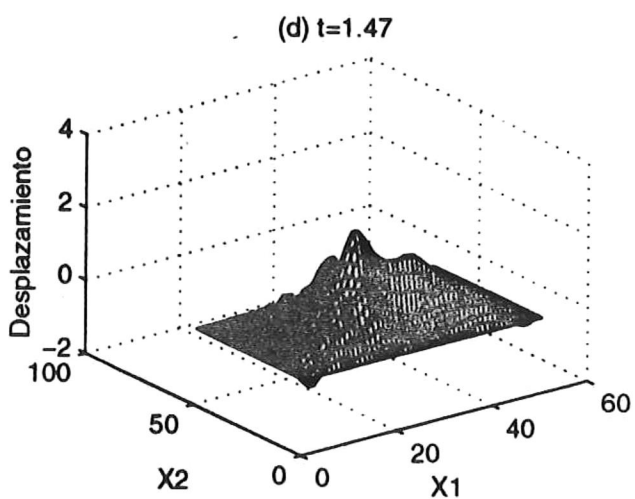
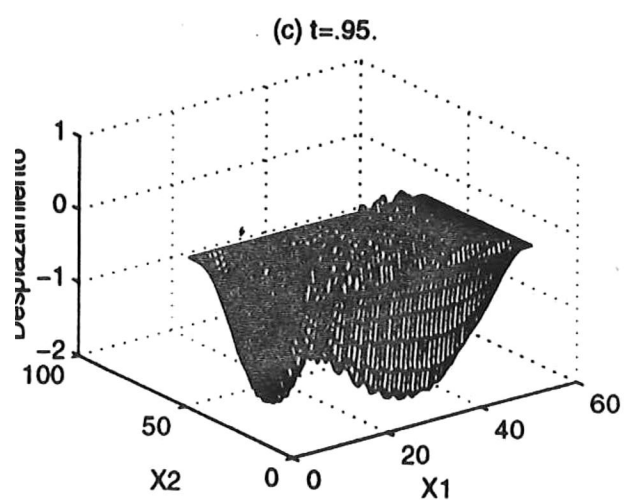
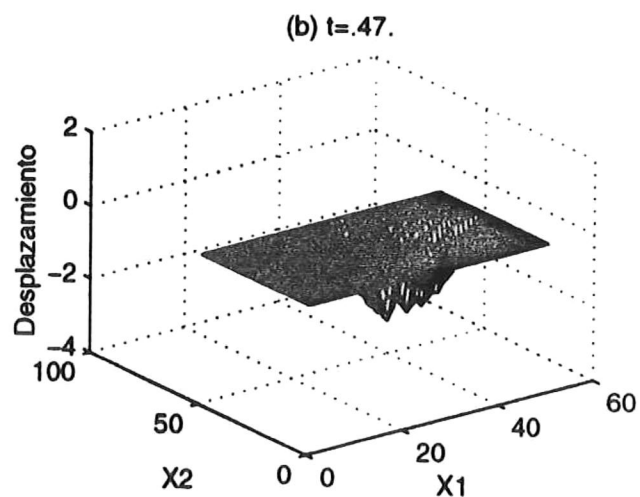
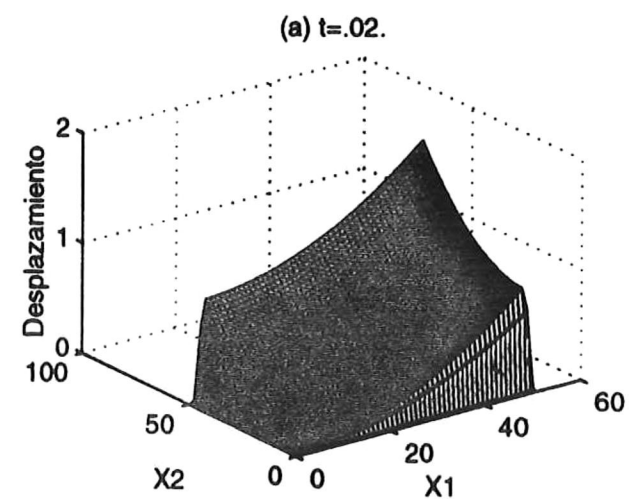


Fig. 11 Ejemplo 9. Membrana cuadrada con forma inicial $x_1^2 + x_2^2$

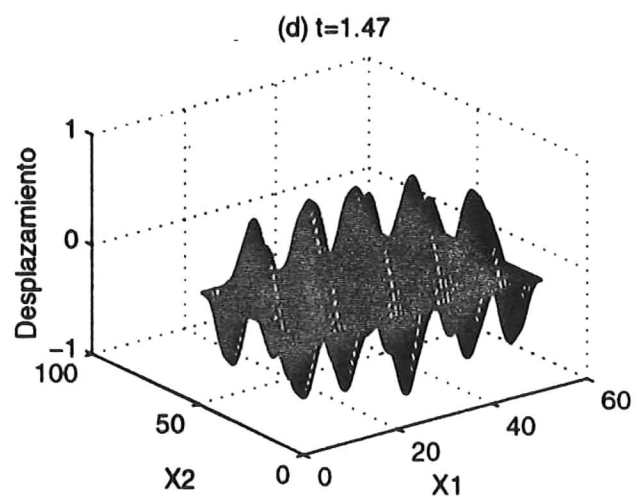
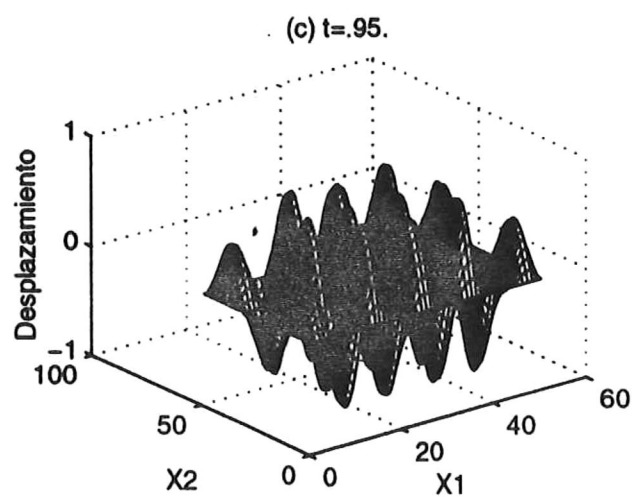
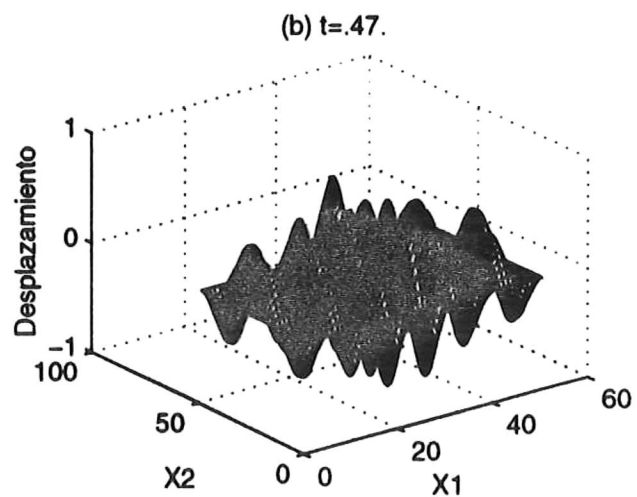
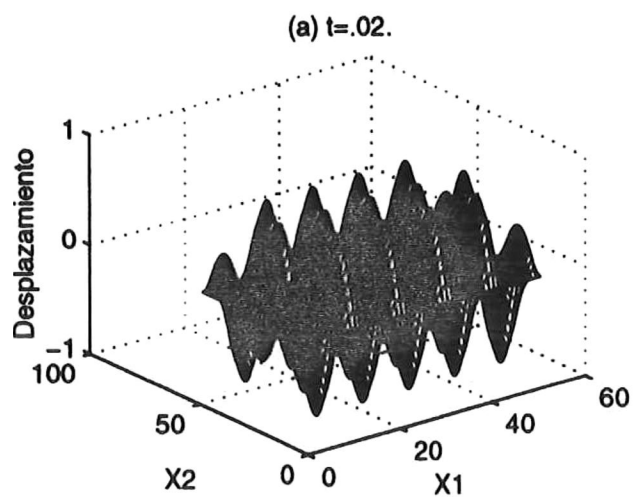


Fig.12 Ejemplo 10. Membrana cuadrada con forma inicial $\cos 10\pi x_1 \sin 3\pi x_2$

V DISCUSIÓN

El sistema está construido a base de clases. Dentro de cada clase se declaran las funciones que están asociadas a las condiciones de frontera, condiciones iniciales y método. Las declaraciones de las clases se encuentran en los archivos de cabecera de terminación ".h". La implantación del comportamiento de éstas se oculta en los archivos de terminación ".ll". Esta forma de trabajar permite establecer la relación objeto-sistema sin permitir al usuario conocer la implantación del objeto.

Por los resultados obtenidos, el modelo objetos da mayor comprensión del proceso de programación. El ocultamiento de información y la encapsulación nos permitió escribir un código comprensible al programador. Esto es muy importante, dado que cualquier programa está sujeto a modificaciones. La capacidad que tiene un programa para ser modificado está dada en este caso por el uso de clases en la construcción del código. Aunque nuestro código se limita a un poco más de 600 líneas, existen programas industriales que están sujetos a cambios. A este nivel la tarea de mantenimiento de dichos programas se transforma no sólo en costo sino en tiempo y falta de eficiencia, cuando no se tiene acceso a la metodología de orientación por objetos. Esta tesis es un trabajo piloto que nos permitió probar las ventajas de OOP en el área científica y extender la posibilidad de trabajar con otras alternativas con OOP. Un ejemplo opcional se representa en la Fig. 13. El usuario en este caso, puede optar por ver la solución para cada aplicación, con distintos métodos, condiciones iniciales y de frontera. Por esto, antes de obtener esta representación (Fig. 13), se buscan propiedades comunes de las diferentes aplicaciones y se utilizan los conceptos de herencia y polimorfismo. Esta opción se construye

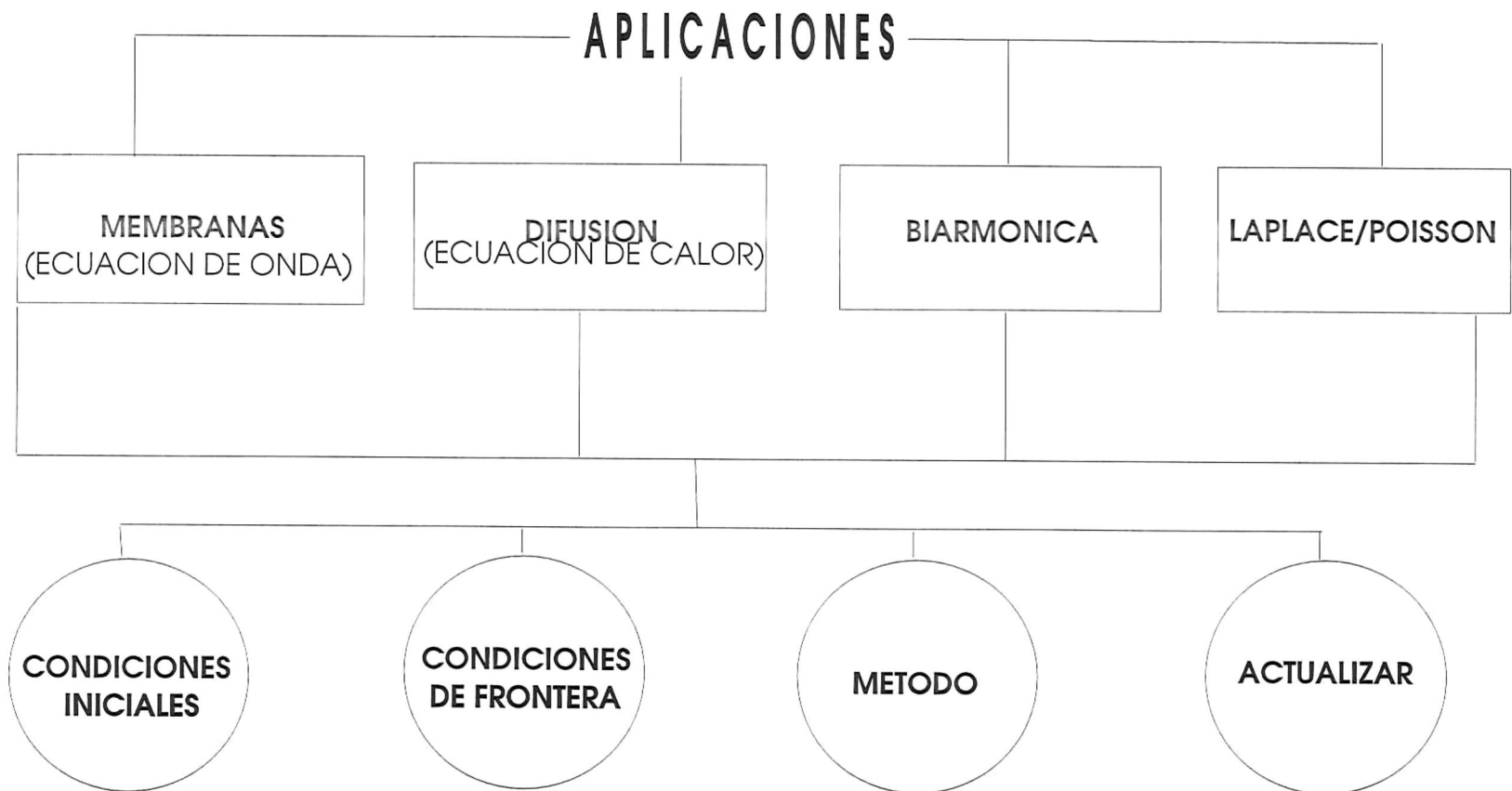


FIG. 13 Aplicaciones alternativas con el modelo de objetos.

bajo los mismos principios utilizados en el programa TUTORIAL de la membrana vibrante. Se hace una función para cada método. Las condiciones iniciales y de frontera se pueden declarar y definir en los archivos *cinicial* y *cfrontera*, donde se integran las condiciones de la membrana.

Por lo que hemos podido observar, la labor importante en OOP se encuentra en la elección apropiada de los objetos. El modelo de objetos inicia con la abstracción y encapsulación de la información. Finalmente se establece una jerarquía de módulos.

El programador procedural obtiene mayor eficiencia en sus códigos con la técnica de orientación por objetos. Cuando científicamente se requiere la elaboración de modelos numéricos con muchas líneas de código. Aunado a esto surge la necesidad de probar numerables experimentos que pueden organizarse de manera más eficiente como ha podido probarse para el problema de la membrana.

Disminuir la complejidad del código, ocultando la implantación del objeto permite un mejor aprovechamiento del código. Por las razones ya mencionadas OO contribuye significativamente en la tecnología. Al nivel de la comunicación que se requiere dentro de la ciencia, puede observarse que la orientación por objetos como una metodología también contribuye al avance científico. En muchas ocasiones los modelos numéricos de fenómenos físicos resueltos por otros programadores del área científica son retomados para posteriores investigaciones.

Usando OOP fue posible trabajar más eficientemente el problema de la membrana vibrante que con el programa procedural MEDF. Aunado a esto las ventajas que se logran respecto a la implantación y estabilidad del código. Esta observación se hace considerando que se trata de un trabajo piloto. Pero en base a este trabajo podemos decir que en códigos industriales las ventajas de usar OOP se ven reflejadas en la reducción de los costos de mantenimiento

provocados por su compleja implantación.

Este trabajo puede utilizarse como base para posteriores investigaciones de OOP en aplicaciones científicas. Así como puede ser usado como un tutorial para esta aplicación de la ecuación de onda, o bien puede extenderse como un tutorial de aplicaciones para cursos de ecuaciones diferenciales parciales. Por las características del modelo deseado es fundamental el uso de OOP, independientemente de la cantidad de líneas de código. Recordando que la utilidad de OOP se hace más evidente en programas de alrededor de 100, 000 líneas de código o más.

VI CONCLUSIONES

• Los programas orientados a objetos permiten un mejor aprovechamiento del código y pueden ser reutilizados fácilmente en trabajos posteriores.

• Como se prueba en el problema de la membrana resuelto en un ambiente orientado a objetos, el uso del mecanismo de herencia facilita los procesos de programación.

• La extensibilidad de los códigos orientados a objetos se logra independientemente de la cantidad de líneas que tiene el programa.

• El carácter tutorial que se logra para el problema de la membrana vibrante es de utilidad en la enseñanza de la Física.

VII LITERATURA CITADA

- [1]. Ames, W. F. 1997. *Numerical methods for partial differential equations*. Academic Press, Inc. San Diego. 365 pp.
- [2] Barton, J. J. y L. R. Nackman. 1994. *Scientific and Engineering C++: An introduction with advanced techniques and examples*. Addison-Wesley Publishing Company, Inc. Reading, Mass. 671 pp.
- [3]. Berg, P. W. y J. L. McGregor. 1966. *Elementary partial differential equations*. Holden-Day, Inc. San Francisco. 421 pp.
- [4]. Elmore, W.C. y M.A. Heald. 1985. *Physics of waves*. Dover Publications, Inc. New York. 477 pp.
- [5]. Gómez- Valdés, J. and J. Favela. 1996. *Gulf of California Numerical Studies: An Object-Oriented Approach*. (abstract), Eos Trans. AGU, April 23.
- [6]. Lippman, S.B. 1991. *C++ Primer*. Addison-Wesley Publishing Company, Inc. Reading, Mass. 464 pp.
- [7]. Martin, J. y J. J. Odell. 1994. *Análisis y diseño orientado a objetos*. Prentice Hall Hispanoamericana, S. A. México. 546 pp.
- [8]. Mijáilov, V. P. 1978. *Ecuaciones Diferenciales Parciales*. Editorial Mir Moscú. URSS. 421pp.
- [9]. Mitchell, A. R. y D. F. Griffiths. 1980. *The finite difference method in partial differential equations*. John Wiley & Sons Ltd. Chichester. 272 pp.
- [10]. Morse, P.M. 1948. *Vibration and sound*. McGraw-Hill Book Company, Inc. New York. 468 pp.
- [11]. Stephenson G. 1985. *Partial Differential Equations for Scientists and Engineers*. Third Edition. Longman. New York. 161 pp.
- [12]. Zachmanoglou, E. C. y D. W. Thoe. 1986. *Introduction to partial differential equations with applications*. Dover Publications, Inc. New York. 405 pp.

VIII ANEXOS

VIII.1 Archivo de implantación: **constantes.li**

```
//constantes.li
#include "constantes.h"
const double h=.02;
const double T=1.0;
const double sigma=1.;
const double p=.5;
const int nx=51;
const int ny=51;
const double k=0.01;
const double ConstantesFisicas::h=0.02;
const double ConstantesFisicas::p=0.5;
const double ConstantesFisicas::T=1.;
const double ConstantesFisicas::sigma=1.0;
const double ConstantesFisicas::k=0.01;
const int ConstantesFisicas::nx=51;
const int ConstantesFisicas::ny=51;
```

VIII.2 Archivo de implantación: **vector.li**

```
//vector.li
/*
Example programs from the book Scientific and Engineering Programming
in C++: An Introduction with Advanced Techniques and Examples,
addison-Wesley, 1994.
```

(c) COPYRIGHT INTERNATIONAL BUSINESS MACHINES
CORPORATION 1994. ALL RIGHTS RESERVED.

See README file for further details.

```
*/
```

```
#include "vector.h"
```

```
SimpleFloatArray::SimpleFloatArray (int n) {
    num_elts=n;
    ptr_to_data= new float [n];
}
SimpleFloatArray::SimpleFloatArray(){
    num_elts=0;
    ptr_to_data=0;
}
```

```

SimpleFloatArray::SimpleFloatArray (const SimpleFloatArray& a) {
    num_elts = a.num_elts;
    ptr_to_data = new float [num_elts];
    copy(a); // Copy a's elements
}

void SimpleFloatArray::copy (const SimpleFloatArray& a) {
    float* p = ptr_to_data + num_elts;
    float* q = a.ptr_to_data + num_elts;
    while (p > ptr_to_data) *--p = *--q;
}

SimpleFloatArray::~SimpleFloatArray() {
    delete [] ptr_to_data;
}

float& SimpleFloatArray::operator[] (int i) {
    return ptr_to_data[i];
}

int SimpleFloatArray::numElts() {
    return num_elts;
}

SimpleFloatArray& SimpleFloatArray::operator=(const SimpleFloatArray& rhs) {
    if ( ptr_to_data != rhs.ptr_to_data) {
        setSize ( rhs.num_elts);
        copy(rhs);
    }
    return *this;
}

void SimpleFloatArray::setSize(int n) {
    if (n != num_elts) {
        delete [] ptr_to_data;
        num_elts= n;
        ptr_to_data= new float [n];
    }
}

SimpleFloatArray& SimpleFloatArray::operator = (float rhs) {
    float* p=ptr_to_data + num_elts;
    while (p>ptr_to_data) *--p=rhs;
}

```

```

    return *this;
}

```

VIII.3 Archivo de implantación: **matrix.li**

```

//matrix.li
#ifndef MATRIXLI
#define MATRIXLI
#include "matrix.h"

matrix::matrix (int nrows, int ncolumns, double initval)
{
    p=new matrep;
    p->n=nrows;
    p->c=ncolumns;
    p->m=new double *[nrows];
    for(int x=0; x<nrows; x++)
        p->m[x]= new double [ncolumns];
    p->n=1;
    for(int j=0; j<nrows; j++){
        for(int i=0; i<ncolumns; i++){
            mval(j, i) = initval;
        }
    }
}

matrix::matrix(int nrows, int ncolumns, double* initvalues)
{
    p=new matrep;
    p->r=nrows;
    p->c=ncolumns;
    p->m= new double *[nrows];
    for(int x=0; x<nrows; x++)
        p->m[x]=new double [ncolumns];

    p->n=1;
    int c=0;
    for(int j=0; j<nrows; j++)
        for(i=0; i<ncolumns; i++)
            mval(j, i) = initvalues [c++];
}

void matrix::error(char *msg1, char *msg2) {
    fprintf(stderr, "matrix error: %s %s\n", msg1, msg2);
    exit(1);
}

```

```

}

matrix::matrix(char *initfile)
{
    const int BSIZE = 120;
    FILE *from;
    if ((from = fopen (initfile, "r")) == 0)
        error("error al abrir archivo");
    char buf[BSIZE], *cp, *cp2;
    int rfound = 0, cfound=0, colonsfound = 0;
    p=new matrep;
    while (fgets(buf, BSIZE, from)) {
        if ((cp=strpbrk (buf, "#")) !=NULL)
            *cp = '\0';
        if ((cp=strpbrk(buf, "r")) !=NULL)
            if((cp2=strpbrk(cp, "=") != NULL)
                if ((cp = strpbrk(cp2, "0123456789")) != NULL) {
                    p->r = atoi(cp);
                    rfound++;
                }
        if ((cp=strpbrk(buf, "c") !=NULL)
            if ((cp2=strpbrk(cp, "=") !=NULL)
                if((cp=strpbrk(cp2, "0123456789")) !=NULL){
                    p->c = atoi(cp);
                    cfound++
                }
        if(strstr(buf, ";;;") !=NULL) {
            colonsfound++;
            break;
        }
    }

    if (lrfound || lcfound || lcolonsfound)
        error ("error en la lectura del archivo\n");
    p->m = new double *[p->r];
    for (int x=0; x<p->r; x++)
        p-> m[x] = new double [p->c];
    p->n=1;
    for(int row =0; row<p->r; row++)
        for(int col=0; col< p->c; col++){
            char nb[20];
            fscanf(from, "%s", nb);
            mval(row, col) = atoi(nb);
            if (ferror (from))
                error ("problema al inicializar archivo");
        }
}

```

```

}
}
matrix::matrix(matrix& x){
    x.p->n++;
    p=x.p;
}

matrix::~~matrix(){
    if (--p->n ==0){
        for(int x=0; x<rows(); x++)
            delete p->m[x];
        delete p->m;
        delete p;
    }
}

matrix matrix::operator=(const matrix& rval) {
    if (--p->n ==0){
        for( int x=0; x<rows(); x++)
            delete p-> m[x];
        delete p->m;
        delete p;
    }
    rval.p->n++;
    p=rval.p;
    return *this;
}

void matrix::write_file(char *filename, char *msg) {
    FILE * to;
    if ((to = fopen(filename, "w")) ==NULL)
        error("error al abrir archivo de escritura");
    fprintf(to, "# %s: matrix file\n",filename);
    fprintf(to, "# %s\n", msg);
    fprintf(to, "rows = %d columns = %d\n", rows(), cols());
    fprintf(to, ":\n");
    for(int row=0; row<rows(); row++){
        for(int col=0; col<cols(); col++)
            fprintf(to, "%6.6g", mval(row, col));
        fprintf(to, "\n");
    }
}

matrix matrix:: operator+(matrix& arg){
    if ((rows() != arg.rows()) || (cols() !=arg.cols()))
        error("las matrices no son conformables para la suma!");
    matrix sum (rows(), cols());
    for(int j=0; j<rows(); j++)

```

```

        for(int i=0; i<cols(); i++)
            sum.mval(j, i) = mval(j,i) + arg.mval(j, i);
    return sum;
}

matrix matrix::operator+(const double arg) {
    matrix sum (rows(), cols());
    for (int j=0; j<rows(); j++)
        for(int i=0; i<cols(); i++)
            sum.mval(j, i) = mval(j, i) + arg;
    return sum;
}

void matrix::print(char *msg) {
    if (*msg) printf ("%s:\n", msg);
    for( int row =0; row<rows(); row++) {
        for(int col=0; col<cols(); col++)
            printf("%6.6f ", mval(row,col));
        printf("\n");
    }
}

#endif //matrix.ll

```

VIII.4 Archivo de implantación: update.ll

```

//update.ll
#include "update.h"

void Actual:: Actualizacion(matrix &Ufuturo, matrix &Upasado, matrix &Upresente)
{
    for(int j=0; j<ny; j++)
        for(int i=0; i<nix; i++){
            Upasado.mval(j, i) = Upresente.mval(j, i);
            Upresente.mval(j, i) = Ufuturo.mval(j, i);
        }
}

```

VIII.5 Archivo de implantación: cfrontera.ll

```

//cfrontera.ll
#include "cfrontera.h"

void CondicionesFrontera::Cauchy (matrix &Ufuturo)
{

```



```

for(int j=1; j<ny-1; j++)
    for(int i=1; i<nx-1; i++) {
        Ufuturo.mval(j, nx-1)=0.;
        Ufuturo.mval(ny-1, i)=0.;
        Ufuturo.mval(0, i)=0.;
        Ufuturo.mval(j, 0)=0.;}

```

VIII.6 Archivo de implantación: **explicito.li**

```

//explicito
#include "explicito.h"

void DiferenciasFinitas::Explicito (matrix &Upasado, matrix &Upresente, matrix &Ufuturo)
{
    c=sqrt(T/sigma);

    for(int j=1; j<ny-1; j++)
        for(int i=1; i<nx-1; i++) {

            Ufuturo.mval(j, i) = (2. - 4.*p*p)*Upresente.mval(j, i) +p*p*c*c*(Upresente.mval(j, i+1)
            +Upresente.mval(j, i-1) + Upresente.mval(j+1, i) +Upresente.mval(j-1,i)) - Upasado(j, i);

        }
    }

```

VIII.7 Archivo de implantación: **cinicial.li**

```

//cinicial.li
#include "cinicial.h"

void CondicionesIniciales::SenplxSenply (matrix &Upasado)
{
    for(int j=0; j<ny; j++)
        for(int i=0; i<nx; i++){
            Upasado.mval(j, i) = sin(M_PI*j*h)*sin(M_PI*i*h);
        }
}

void CondicionesIniciales::SenplxSen2ply (matrix &Upasado)
{
    for(int j=0; j<ny; j++)
        for(int i=0; i<nx; i++){
            Upasado.mval(j, i) = sin(2*M_PI*j*h)*sin(M_PI*i*h);
        }
}

```

```

}
}
void CondicionesIniciales::Sen8plxSenply (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){
    Upasado.mval(j, i) = sin(M_PI*j*h)*sin(8*M_PI*i*h);
}
}
void CondicionesIniciales::xSEXTAy (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){

    Upasado.mval(j, i) = (i*h)*(i*h)*(i*h)*(i*h)*(i*h)*(i*h)*(j*h);
}
}

void CondicionesIniciales::x2MASy2 (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){
    Upasado.mval(j, i) = (i*h)*(i*h) + (j*h)*(j*h);
}
}

void CondicionesIniciales::Cos10plxSen3ply (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){
    Upasado.mval(j, i) = sin(3*M_PI*j*h)*cos(10*M_PI*i*h);
}
}
void CondicionesIniciales::Algebraico (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){
    Upasado.mval(j, i) = .00001*j*h*(20-j*h)*i*h*(20-i*h);
}
}

void CondicionesIniciales::xMASy (matrix &Upasado)
{
for(int j=0; j<ny; j++)
for(int i=0; i<nx; i++){

```

```

        Upasado.mval(j, i) = i*h+j*h;
    }
}

void CondicionesIniciales::Constante (matrix &Upasado)
{
    for(int j=0; j<ny; j++)
        for(int i=0; i<nx; i++){
            Upasado.mval(j, i) = 1.0;
        }
}

void CondicionesIniciales::Present(matrix &Upasado, matrix &Upresente)
{
    for(int j=1; j<ny-1; j++)
        for(int i=1; i<nx-1; i++){
            Upresente.mval(ny-1, i)=0.0;
            Upresente.mval(j, nx-1)=0.0;
            Upresente.mval(0,i)=0.0;
            Upresente.mval(j,0)=0.0;
            Upresente.mval(j, i)= (1-2*p*p)*Upasado.mval(j, i) + .5*p*p*(Upasado.mval(j, i+1)
                +Upasado.mval(j, i-1) + Upasado.mval(j+1, i) + Upasado.mval(j-1, i));
        }
}

```