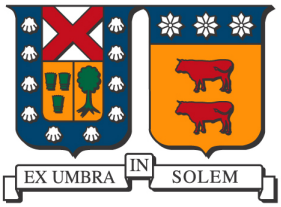


Universidad Técnica Federico Santa María

Departamento de Ingenieria Mecánica

Proyecto 1

Dinámica de fluidos computacional

Nombre : Ignacio Apablaza
Rol : 201141007-6
Profesores : Romain Gers
: Olivier Skurtys
Asignatura : IPM468

Índice

1 Introducción **3**

2 Metodología **4**
2.1 Precisión numérica en Fortran 4
2.2 Método de Diferencias Finitas 4
2.3 Esquema de integración temporal 5
2.3.1 Esquema Euler Implicito 5
2.3.2 Esquema integración de Θ 6
2.3.3 Esquema Leap-Frog 6
2.3.4 Esquema Newmark 6
2.3.5 Método Runge Kutta de orden 4 6
2.4 Analisis Espectral 7
2.4.1 Factor de Amplificación 7

3 Desarrollo y Análisis **8**
3.1 Ejercicios en Fortran 8
3.1.1 Ejercicio 1 8
3.1.2 Ejercicio 2 9
3.2 Ejercicio 3 10
3.3 Estudio del comportamiento mecánico de una arteria 12
3.4 Parte 1: Movimiento de una pared arterial 12
3.4.1 Euler Implicito 13
3.4.2 Crank Nicolson 14
3.4.3 Resultados 16
3.5 Parte 2: Un modelo hiperbólico para la interacción de la sangre con la pared . . . 26
3.5.1 Leap-Frog 26
3.5.2 Newmark 27
3.5.3 Resultados 28
3.6 Atractor de Lorenz 29
3.6.1 Parte 1 29
3.6.2 Parte 2 31
3.6.3 Parte 3 32

4 Conclusiones y Observaciones **35**

5 Codigos implementados **37**
5.1 Ejercicios en Fortran 37
5.1.1 Ejercicio 1 37
5.1.2 Ejercicio 2 38
5.1.3 Ejercicio 3 39
5.2 Estudio del comportamiento mécanico de una arteria 43
5.2.1 Parte 1 43

1 Introducción

En el presente trabajo se estudia el método de diferencias finitas aplicado a la resolución de problemas modelados por ecuaciones diferenciales. El estudio se basa en programar distintas rutinas escritas en lenguaje Fortran que permite implementar los algoritmos a estudiar. Se divide en en tres secciones: La primera parte consiste en el estudio del parámetro `precision` que permite establecer el número de cifras significativas asociada a un objeto de programación, y cómo la elección de este parámetro afecta los cálculos numéricos.

La segunda parte se estudia el comportamiento de una arteria sometida a esfuerzos asociados al bombeo sanguineo. Se modela el radio de la pared de la arteria como cilindros que se deforman radialmente y dicho comportamiento puede modelarse como una ecuación diferencial lineal de primer orden o como una ecuación diferencial hiperbólica, dependiendo de los supuestos escogidos. Se implementan los métodos de integración temporal Euler Implícito, Crank Nicolson, Leap-Frog y Newmark. La idea es estudiar la estabilidad del método teóricamente y comparar los resultados obtenidos por el programa implementado.

La última parte está orientada a la implementación al estudio de la resolución de sistemas dinámicos caóticos, correspondientes a modelos simplificados de la convección de Rayleight-Benand. Se obtiene un sistema de ecuaciones diferenciales de orden 1, tridimensional y no lineal. Se implementan una rutina Runge Kutta 4 (RK4) para la resolución del sistema. Se estudia además el comportamiento del mismo al variar los parametros que lo gobiernan.

Se anexa al final del informe los cógidos y rutina implementadas.

2 Metodología

2.1 Precisión numérica en Fortran

Fortran (*Formula Translator* o *Traductor de Fórmulas*) es un lenguaje de programación orientado a objetos y de alto nivel utilizado para la computación científica en distintas disciplinas del área de las ciencias. Fortran posee distintos tipos de objetos:

character cadena de uno o varios caracteres
integer números enteros positivos y negativos
logical valores lógicos o booleanos (`.true.` o `.false.`)
real números reales positivos y negativos
complex números complejos compuestos de una parte real y una imaginaria
tipos derivados tipos especificados por usuario

Los objetos de clase `real` poseen ciertos parámetros que describen sus características. Un paramétro relevante a estudiar es la precisión que describe a un objeto declarado como `real`

Entero	$-2.147.483.648 \leq i \leq 2.147.483.647$	–
Real Simple Precisión	$1.2 \times 10^{-38} \leq \lvert x \rvert \leq 3.4 \times 10^{38}$	7 cifras significativas
Real Doble Precision	$2.2 \times 10^{-308} \leq \lvert x \rvert \leq 1.8 \times 10^{308}$	16 cifras significativas

Tabla 1. Características de precisión de reales en Fortran

La especificación del parámetro precisión especificará el tamaño de memoria asignada al objeto. Dependiendo de la naturaleza del cálculo empleado será más conveniente utilizar una u otra precisión.

2.2 Método de Diferencias Finitas

Una manera de aproximar numéricamente derivadas presentes en una ecuación diferencial ordinaria o parcial es mediante el método de Diferencias Finitas, que consiste representar las razones de cambio como una diferencia de valores nodales discretos. Se desprende del desarrollo de series de Taylor de la función incógnita: Sea $\phi(x)$ una función diferenciales en una dimensión en un dominio de interés, entonces el valor de $\phi(x + \Delta x)$ se puede expresar mediante el desarrollo en series de Taylor:

$$\phi(x + \Delta x) = \sum_{n=0}^{\infty} \frac{\Delta x^n}{n!} \frac{\partial \phi^{(n)}}{\partial^n}\Big|_x \tag{1}$$

Para una diferencia $+\Delta x$ se tiene,

$$\phi(x + \Delta x) = \phi(x) + \Delta x \frac{\partial \phi}{\partial x}\Big|_x + \frac{(\Delta x)^2}{2} \frac{\partial^2 \phi}{\partial x^2}\Big|_x + \frac{(\Delta x)^3}{6} \frac{\partial^3 \phi}{\partial x^3}\Big|_x + \cdots \tag{2}$$

Para una diferencia $-\Delta x$ se tiene

$$\phi(x - \Delta x) = \phi(x) - \Delta x \frac{\partial \phi}{\partial x}\Big|_x + \frac{(\Delta x)^2}{2} \frac{\partial^2 \phi}{\partial x^2}\Big|_x - \frac{(\Delta x)^3}{6} \frac{\partial^3 \phi}{\partial x^3}\Big|_x + \cdots \tag{3}$$

Distintas combinaciones de las ecuaciones (2) y (3) permiten obtener las aproximaciones de distintos ordendes de derivada.

Truncando el desarrollo de la serie se obtiene la aproximación. Al conocer la expresión analitica en (1) se puede determinar el orden el error obtenido. El desarrollo en una dimensión se extiende a dimensiones superiores. A continuación se exponen los tipos de aproximaciones utilizados en este trabajo:

Diferencia hacia atras (Backward) Aproximando la primera derivada de $\phi(x)$ mediante diferencias hacia atras resulta:

$$\frac{\partial \phi}{\partial x}\Big|_{x=x_i} = \frac{\phi(x_i) - \phi(x_{i-1})}{\Delta x} + o(\Delta x^1) \tag{4}$$

$o(\Delta x)$ agrupa el términos truncados de la serie y representa el error de la aproximación, de tal manera que,

$$\frac{\partial \phi_i}{\partial x} \approx \frac{\phi_i - \phi_{i-1}}{\Delta x} \tag{5}$$

Donde ϕ_i denota el valor nodal que discretiza a la función en el dominio ($\phi(x_i) = \phi_i$) , En este caso se tiene un error de orden 1

Diferencias centradas Aproximando la segunda derivada de $\phi(x)$ utilizando diferencias finitas centradas resulta:

$$\frac{\partial^2 \phi}{\partial x^2}\Big|_{x_i} = \frac{\phi(x_{i+1}) - 2\phi(x_i) + \phi(x_{i-1})}{\Delta x^2} + o(\Delta x^2) \tag{6}$$

Este esquema de aproximación posee un error de orden 2. Analogo al caso anterior la aproximación se plantea como,

$$\frac{\partial^2 \phi}{\partial x^2}\Big|_{x_i} \approx \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \tag{7}$$

El método de diferencias finitas se aplica para discretizar derivadas espaciales y temporales. Estas últimas determinan los esquemas de integración temporales.

2.3 Esquema de integración temporal

Sea una ecuación diferencial de $\phi(t)$ tal que,

$$\begin{cases} \dfrac{d\phi(t)}{dt} = f(t, \phi(t)) \\ \phi(t=0) = \phi_0 \end{cases} \tag{8}$$

Se tiene un problema de Cauchy o de valor inicial. La solución de ϕ está dada por,

$$\int_{t_n}^{t_{n+1}} \frac{d\phi(t)}{dt} = \int_{t_n}^{t_{n+1}} f(t, \phi(t)) dt \tag{9}$$

Del teorema fundamental del cálculo se tiene,

$$\phi(t_{n+1}) - \phi(t_n) = \int_{t_n}^{t_{n+1}} f(t, \phi(t)) dt \tag{10}$$

Según como se integre el término de la derecha de la ecuación en (10) se obtienen los distintos esquemas de integración.

2.3.1 Esquema Euler Implicito

El esquema de Euler Implicito (Backward) se define como,

$$\phi^{n+1} = \phi^n + \Delta t f(t_{n+1}, \phi(t_{n+1})) \tag{11}$$

Este esquema requiere conocer el valor del pasos t_n y t_{n+1}. Este tipo de esquemas son conocidos como esquemas de dos pasos (*two level scheme*)

2.3.2 Esquema integración de Θ

La familia de esquemas Θ se describen como,

$$\phi^{n+1} = \phi^n + \Theta f(t_n, \phi(t_n)) + (1-\Theta) f(t_{n+1}, \phi(t_{n+1}) \tag{12}$$

Para el valor de $\Theta = \frac{1}{2}$ se tiene el esquema de Crank Nicolson:

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{2} \left(f(t_n, \phi(t_n) + f(t_{n+1}, \phi(t_{n+1})) \right) \tag{13}$$

2.3.3 Esquema Leap-Frog

Los esquemas Leap-Frog corresponden a una discretizacion central de la derivada temporal.

$$\phi^{n+1} = \phi^{n-1} + 2\Delta t f(t_n, \phi(t_n)) \tag{14}$$

Se calcula el valor de ϕ en el tiempo t_{n+1} a partir de dos valores anteriores t_n y t_{n-1}. Estos tipos de esquema son llamadas de tres pasos (*three level scheme*)

2.3.4 Esquema Newmark

Consiste en un esquema de dos pasos para el cálculo de ϕ y su derivada $\partial\phi(t)/\partial t$. Sea $\psi(t) = \partial\phi(t)/\partial t$, se integra ψ utilizando un esquema Θ,

$$\psi_{n+1} = \psi_n + \Delta t \left[\Theta \frac{\partial \psi}{\partial t}^n \Big|_t + (1-\Theta) \frac{\partial \psi}{\partial t}^{n+1} \Big|_t \right] \tag{15}$$

Para integrar ϕ se utiliza un esquema explícito donde el último término utiliza un esquema Θ

$$\phi_{n+1} = \phi_n + \Delta t \psi + (\Delta t)^2 \left[\beta \frac{\partial \psi}{\partial t}^n \Big|_t + (1-\beta) \frac{\partial \psi}{\partial t}^{n+1} \Big|_t \right] \tag{16}$$

donde β es un parámetro que reemplaza a Θ e integra al factor 2 del desarrollo de la serie de Taylor.

2.3.5 Método Runge Kutta de orden 4

Los métodos de Runge Kutta conocidos como métodos *Predictor-Corrector*: Se calcula uno o varios valores intermedios de la función incógnita ϕ^*, llamados predictores, para finalmente calcular el resultado final $\phi(t + \Delta t)$ (corrector).

El método Runge Kutta de orden 4 consiste en calcular tres pasos de predicción y el último paso de corrección:

- 1^{ra} predicción: Euler Explícito
- 2^{da} predicción: Euler Implícito
- 3^{ra} predicción: Leap-Frog
- Corrección: Método de integración de Simpson

Luego, se puede expresar como:

$$\phi^*_{n+1/2} = \phi_n + \frac{\Delta t}{2} f(t_n, \phi_n) \tag{17}$$

$$\phi^{**}_{n+1/2} = \phi_n + \frac{\Delta t}{2} f(t_{n+1/2}, \phi^*_{n+1/2}) \tag{18}$$

$$\phi^*_{n+1} = \phi_n + \Delta t f(t_{n+1/2}, \phi^{**}_{n+1/2}) \tag{19}$$

$$\phi_{n+1} = \phi_n + \frac{\Delta t}{6} \left[f(t_n, \phi_n) + 2f(t_{n+1/2}, \phi^*_{n+1/2}) + 2f(t_{n+1/2}, \phi^{**}_{n+1/2}) + f(t_{n+1}, \phi^*_{n+1}) \right] \tag{20}$$

2.4 Analisis Espectral

La discretización de una ecuación diferencial de $u = u(\vec{x}, t)$ se puede expresar en una notación matricial: Sea $\vec{U}$ el vector que contiene los valores u_i $(i = 1, \ldots, n)$.

$$\frac{d\vec{U}}{dt} = \boldsymbol{S}\vec{U} + \vec{Q} \tag{21}$$

Donde $\boldsymbol{S}$ es la matriz asociada a la discretización espacial y $\vec{Q}$ es el vector que contiene los componentes del término fuente. Esta ecuación se descompone a partir de sus valores propios (Descomposición modal), en ella se desacopla la incognita en espacio y tiempo. Para cada componente del vector $\vec{U}$ se tiene,

$$\frac{d\overline{U}_j}{dt} = \Omega_j \overline{U}_j + Q_j \tag{22}$$

donde,

$$\vec{\overline{U}}(\vec{x}, t) = \sum_{j=1}^{N} \vec{\overline{U}}_j(t) V^{(j)}(\vec{x}) \qquad \text{y} \qquad Q = \sum_{j=1}^{N} Q_j V^{(j)} \tag{23}$$

Ω_j son los autovalores asociados a la dirección del vector propio $V^{(j)}$ de la matriz $\boldsymbol{S}$; N es el número de dimensiones de la ecuación (21). Luego, su solución analítica en función de sus valores propios viene dado por,

$$\overline{U}_j(t) = \left(U^0 + \frac{Q_j}{\Omega_j} \right) e^{\Omega_j t} - \frac{Q_j}{\Omega_j} \tag{24}$$

Donde U^0 es la condición inicial del problema de Cauchy.

2.4.1 Factor de Amplificación

Es de interés conocer el comportamiento de la solución transiente de $\vec{U}$. Para ello se supone $Q = 0$ (solución homogenea) y se denota como U^T la solución transiente,

$$U_j^T(t) = U^0 e^{\Omega_j t} \tag{25}$$

Se define el factor de amplificación $G(\Omega_j)$,

$$\overline{U}_j^T(n\Delta t) = G(\Omega_j)\overline{U}_j^T\left([n-1]\,\Delta t\right) \tag{26}$$

Notar que $G = G(\Omega_j)$ es función de la discretización espacial. Reemplazando (25) en (26),

$$\overline{U}_j^0 e^{\Omega_j n \Delta t} = G(\Omega_j)\overline{U}_j^0 e^{\Omega_j (n-1)\Delta t} \tag{27}$$

Despejando $G(\Omega_j)$ se obtiene el factor de amplicación para un paso de tiempo (de $(n-1)\Delta t$ a $n\Delta t$)

$$G = e^{\Omega_j \Delta t} \tag{28}$$

Entonces, el factor de amplificación G calculado desde la solución inicial U^0 hasta el paso de tiempo $n\Delta t$ se obtiene,

$$G = e^{\Omega_j n \Delta t} \tag{29}$$

Para garantizar que la solución transiente sea estable se debe cumplir que,

$$|G(\Omega_j)| = |e^{[\Re(\Omega_j)] + [\Im(\Omega_j) n \Delta t] i}| < 1 \tag{30}$$

lo que implica que

$$\Re(\Omega_j) \leq 0 \tag{31}$$

Se desprende la ecuación anterior que $\Re(\Omega_j)$ esta asociado al error de disipación (exponencial), mientras que $\Im(\Omega_j)$ al error de dispersión (oscilación)

3 Desarrollo y Análisis

3.1 Ejercicios en Fortran

3.1.1 Ejercicio 1

Sea $A(n)$ un número real tal que,

$$A(n) = \sum_{n=1}^{\infty} \frac{1}{n} \tag{32}$$

Se implementa una programa en Fortran que permite calcular y graficar $A(n)$ para ciertos valores de n. Esta serie geométrica es divergente, es decir, $A(n) \to \infty$ cuando $n \to \infty$.

En la Figura 1 se grafica la función A en simple precisión y doble precisión (A_{sp} y A_{dp}, respectivamente). El desarrollo de A es práctimente el mismo, salvo una diferencia decimal, hasta aproximadamente $n = 100000$. Como se observa A_{sp} y A_{dp} empiezan a mostrar diferencias que se vuelven más prominentes en la medida que incrementa n. Para $n_{crit} = 2097151$, el valor de A_{sp} se estanca, ya que,

$$\frac{1}{n} = \frac{1}{2097151} \approx 0,000000477...$$

Los reales de simple precisión con los que trabaja Fortran poseen 7 cifras significativas, luego por redondeo $1/n_{crit}$ no contribuye a la sumatoria, resultando en $A_{sp}|_{n=k} = 15.4036827$ para $k \geq n_{crit}$. La diferencia entre los valores obtenidos con la simple y doble precision se muestran en la Figura 2, donde se grafica un error porcentual respecto a A_{dp}:

$$E(\%) = \frac{A_{dp}(n) - A_{sp}(n)}{A_{dp}(n)} \tag{33}$$

donde se asume que el valor con doble precisión es el valor más exacto.

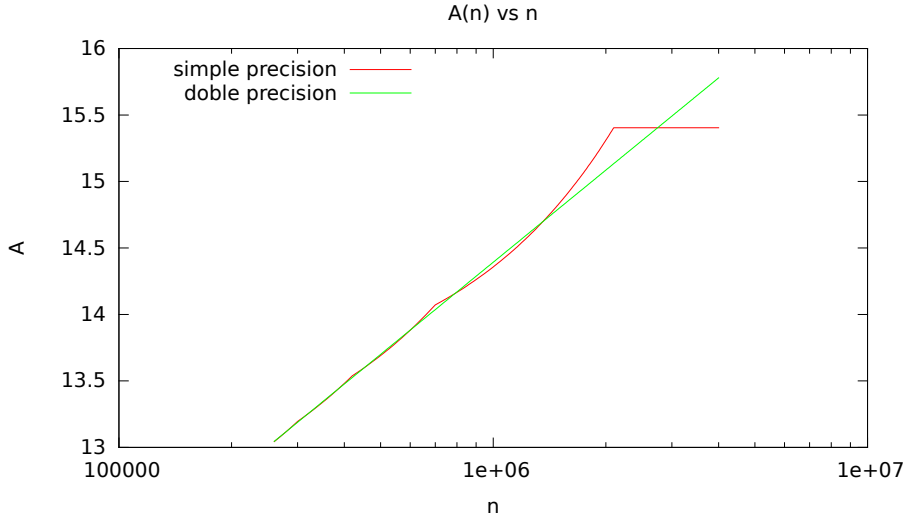


Figura 1. Gráfica de A_{sp} y A_{dp} para algunos valores de n. Abscisa en escala logarítmica

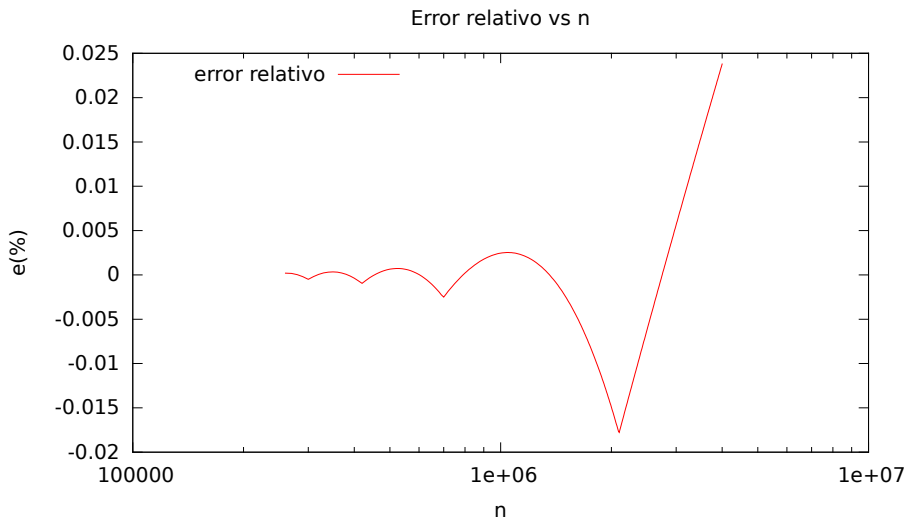


Figura 2. Error relativo $E(\%)$ de A_{sp} respecto a A_{dp}. Abscisa en escala logarítmica

3.1.2 Ejercicio 2

Se implementa una rutina en Fortran que permite calcular los $n+1$ valores de la serie Fibonacci

$$u_{n+1} = u_n + u_{n-1} \qquad \text{tal que} \qquad u_0 = 0\ ;\ u_1 = 1 \tag{34}$$

En las Figuras 3 y 4 se representa los elementos de la serie. Se observa en la primera gráfica que la serie presenta un crecimiento exponencial. Para $n = 46$ se presenta una inestabilidad numérica, lo cual se explica por la memoria asignada a un objeto real (Tabla 1).

Se utiliza un real de doble precision en vez de un entero y se grafica la serie (Figura 5). Se vuelve a apreciar el comportamiento exponencial. La serie se grafica hasta que el se alcanza el tope de memoria, arrojando *inf*

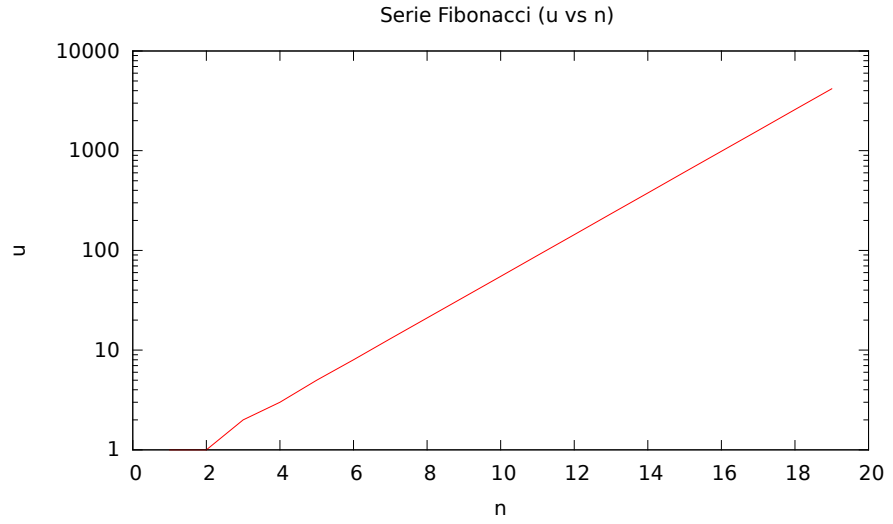


Figura 3. Gráfica de la serie Fibonacci para $n \in [0, \ldots, 20]$. Ordenanda en escala logarítmica

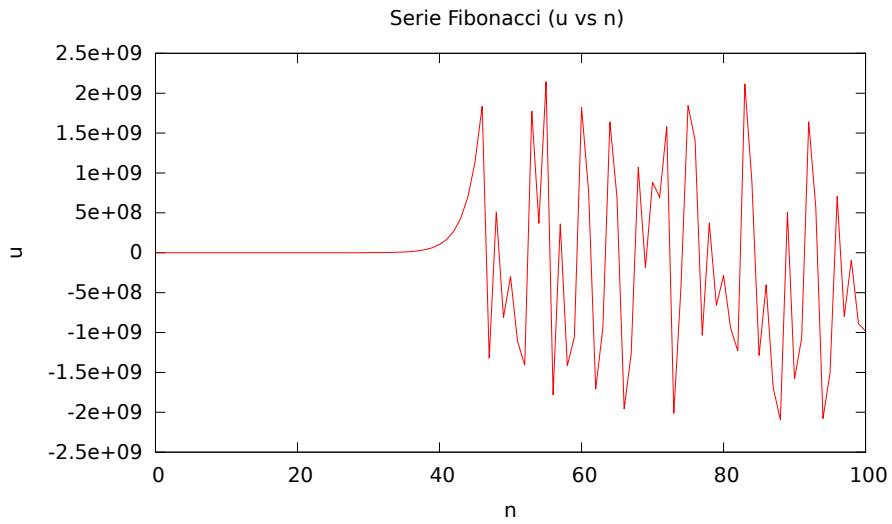


Figura 4. Gráfica de la serie Fibonacci para $n \in [0, \ldots, 100]$ definiendo u como un objeto `integer`. Para $n = 46$ se presenta una inestabilidad numérica: se obtiene números negativos en una serie estrictamente creciente. Ordenanda en escala logarítmica

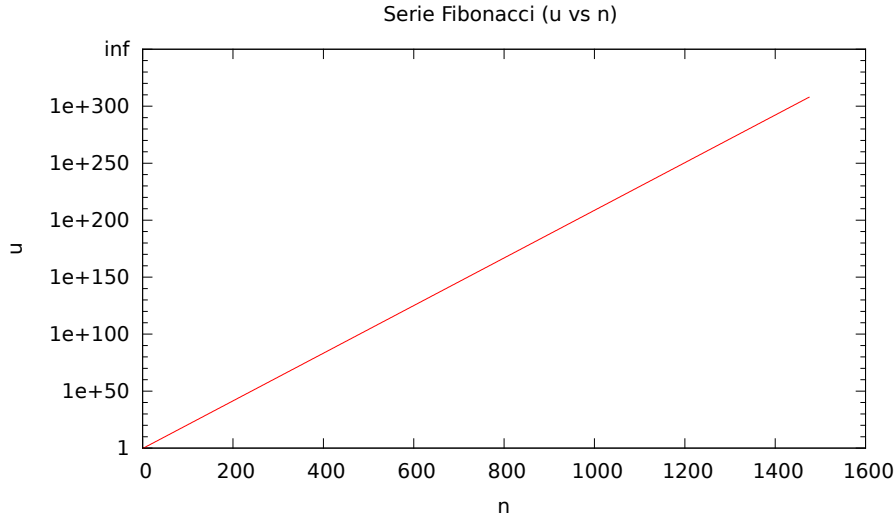


Figura 5. Gráfica de la serie Fibonacci para $n \in [0, \ldots, 1600]$ definiendo u como un objeto `real` de doble precisión. El resultado crece exponencialmente hasta alcanzar el máximo de memoría permitido, arrojando *inf*. Ordenanda en escala logarítmica

3.2 Ejercicio 3

Se escribe una rutina *matrix-mult* que realiza el producto entre dos matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ de dimensiones (m_a, n_a) y (m_b, n_b) respectivamente. El algoritmo implementado es el siguiente:

1. Se ingresan los valores de las matrices $\boldsymbol{A}_{n_a,m_a}$ y $\boldsymbol{B}_{n_b,m_b}$.
2. Se verifica la dimensión entre $\boldsymbol{A}$ y $\boldsymbol{B}$. Si $m_a \neq n_b$ entonces $l = 1$ y se sale de la subrutina.

En el caso contrario, las dimensiones son consistente con la multiplicación matricial, $l = 0$, y se pasa al siguiente paso (l: indicador del error)

3. Se define el tamaño y se asigna la memoria para arreglo C_{n_a,m_b}
4. Se realiza la multiplicación matricial entre $\boldsymbol{A}$ y $\boldsymbol{B}$. Se asigna a la matriz $\boldsymbol{C}$
5. Salida de la subrutina $\boldsymbol{C}_{n_a,m_b}$ y l

En la Figura 6 se expone un diagrama de flujo que explica la subrutina

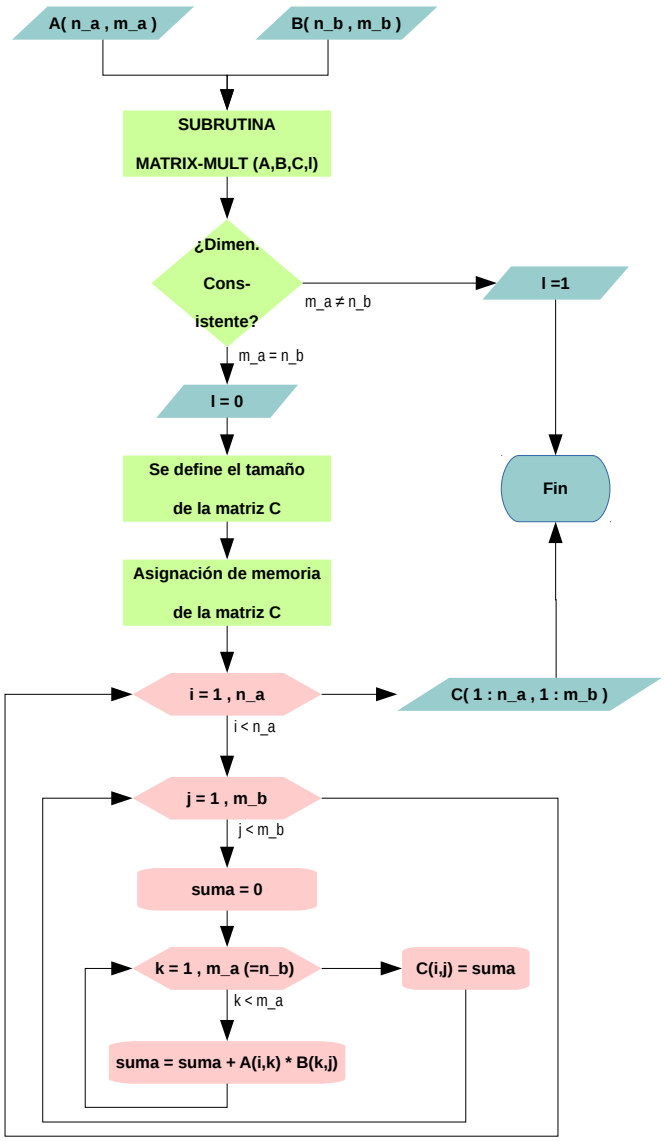


Figura 6. Diagrama de flujo de la subrutina *matrix-mult*. Variables de entrada: Matrices $\boldsymbol{A}_{n_a,m_a}$ y $\boldsymbol{B}_{n_b,m_b}$. Variables de salida: Matriz $\boldsymbol{C}_{n_a,m_b}$ y el indicador de error l

3.3 Estudio del comportamiento mecánico de una arteria

3.4 Parte 1: Movimiento de una pared arterial

Una arteria puede modelarse por un cilindro flexible de base circular, longitud L, radio R_0, cuyas paredes poseen un espesor H. Se supone que está constituido de un material elástico, incompresible, homogéneo e isotrópico.

Un modelo simplificado que describe el comportamiento mecánico de la pared arterial en interacción con el flujo sanguíneo se obtiene considerando que el cilindro es constituido por un conjunto de anillos independientes uno de otros. De esta manera se puede despreciar las interacciones longitudinales y axiales a lo largo de la arteria. Luego, se supone que la arteria se deforma solamente en la dirección radial.

El radio de la arteria está dado por,

$$R(t) = R_0 + y(t) \tag{35}$$

donde $y(t)$ es la deformación radial en función del tiempo t. Al aplicar la ley de Newton en el sistema de anillo independientes conduce a una ecuación que permite modelar el comportamiento mecánico de la pared de la arteria en función del tiempo,

$$\frac{d^2y(t)}{dt^2} + \beta\frac{dy(t)}{dt} + \alpha y(t) = \gamma(p(t) - p_0) \tag{36}$$

donde,

$$\alpha = \frac{E}{\rho_w R_0^2} \qquad \gamma = \frac{1}{\rho_w H} \qquad \beta = \text{constante} > 0 \tag{37}$$

Particularmente se modela la variación de la presión a lo largo de la arteria como una función sinusoidal que depende de la posición x y el instante de tiempo t,

$$(p - p_0) = x\Delta p\,(a + bcos(\omega_0 t)) \tag{38}$$

Se calcula numericamente la ecuación (36) con el término fuente (38). Se uyilizan los siguientes valores realistas para los parámetros físicos:

L	$= 5 \times 10^{-2}$ m	b	$= 133.32$ N m $^{-2}$
R_0	$= 5 \times 10^{-3}$ m	a	$= 1333.2$ N m $^{-2}$
ρ_w	$= 1 \times 10^{3}$ kg m $^{-3}$	Δp	$= 33.33$ N m $^{-2}$
H	$= 3 \times 10^{-4}$ m	w_0	$= 2\pi/0.8$
E	$= 9 \times 10^{5}$ N m $^{-2}$		

Tabla 2. Parametros utilizados para la simulación 1

Y considerando a su vez dos parametros de β:

(a) $\beta = \sqrt{\alpha}$

(b) $\beta = \alpha$

Se reescribe la ecuación (36) como un sustema de ecuaciones lineales. En forma matricial,

$$\boldsymbol{A}\vec{y} + \vec{b} \tag{39}$$

donde $\vec{y} = \begin{vmatrix} y & y' \end{vmatrix}^T$ (T significa transpuesta), y $\vec{b}(t)$ es un vector fuente dependiente del tiempo t. La matriz $\boldsymbol{A}$ resultante es,

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ -\alpha & -\beta \end{pmatrix} \tag{40}$$

Los valores propios de $\boldsymbol{A}$ se obtienen del desarrollo del polinomio característico,

$$det(\boldsymbol{A} - \Omega \boldsymbol{I}) = \begin{vmatrix} -\Omega & 1 \\ -\alpha & -\Omega - \beta \end{vmatrix} \rightarrow \alpha\Omega^2 + \beta\Omega + 1 = 0 \tag{41}$$

Luego, los valores propios se calculan como la raíz del polinomio,

$$\Omega_{1,2} = \frac{(-\beta \pm \sqrt{\beta^2 - 4\alpha})}{2} \tag{42}$$

Notar que para valores de $\beta \geq 2\sqrt{\alpha}$ ambos valores, Ω_1 y Ω_2, resultan reales y negativos, mientras que para valores de $\beta < 2\sqrt{\alpha}$ ambos autovalores resultan números complejos con su componente real negativa.

Se implementa una subrutina que permite calcular los valores propios de la matriz $\boldsymbol{A}$. Utilizando los valores de la Tabla 2 se obtiene:

(a) $\beta = \sqrt{\alpha} = 6.0 \times 10^3$

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 36.0 \times 10^6 & 6.0 \times 10^3 \end{pmatrix} \rightarrow \begin{matrix} \Omega_1 = & -3000.00 + 5196.15i \\ \Omega_2 = & -3000.00 - 5196.15i \end{matrix} \tag{43}$$

(b) $\beta = \alpha = 36.0 \times 10^6$

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 36.0 \times 10^6 & 36.0 \times 10^6 \end{pmatrix} \rightarrow \begin{matrix} \Omega_1 = & -1.0 \\ \Omega_2 = & -36.0 \times 10^6 \end{matrix} \tag{44}$$

3.4.1 Euler Implicito

Discretización de la ecuación diferencial Se implementa una subrutina que permite calcular la ecuacion (36) usando el método de Euler Implícito para dos valores de β. Sea $y(x,t) = y_j^n$ y $\partial y/\partial t(x,t) = z_j^n$, recurriendo a la expresión (**??**) e implementando un esquema de integración implícito se tiene que,

$$\frac{y^n - y^{n-1}}{\Delta t} = z^n \tag{45}$$

$$\frac{z^n - z^{n-1}}{\Delta t} = -\alpha y^n - \beta z^n + \gamma(p_n - p_0) \tag{46}$$

Reordenando los valores en los pasos de tiempo n y $n-1$ en los lados izquierdo y derecho respectivamente, se expresa la relación anterior en forma matricial como,

$$\boldsymbol{A} \cdot \begin{Bmatrix} y^n \\ z^n \end{Bmatrix} = \begin{Bmatrix} y^{n-1} \\ z^{n-1} \end{Bmatrix} + \Delta t \begin{Bmatrix} 0 \\ \gamma(p_n - p_0) \end{Bmatrix} \tag{47}$$

donde,

$$\boldsymbol{A} = \begin{pmatrix} 1 & -\Delta t \\ \Delta t\alpha & 1 + \Delta t\beta \end{pmatrix} \tag{48}$$

Despenjando las incognitas $\begin{Bmatrix} y^n & z^n \end{Bmatrix}^T$ se obtiene,

$$\begin{Bmatrix} y^n \\ z^n \end{Bmatrix} = \boldsymbol{A}^{-1} \cdot \begin{Bmatrix} y^{n-1} \\ z^{n-1} \end{Bmatrix} + \Delta t \boldsymbol{A}^{-1} \cdot \begin{Bmatrix} 0 \\ \gamma(p_n - p_0) \end{Bmatrix} \tag{49}$$

donde

$$\boldsymbol{A}^{-1} = \frac{1}{1 + \beta\Delta t + \alpha(\Delta t)^2} \begin{pmatrix} 1 + \beta\Delta t & \Delta t \\ -\Delta t\alpha & 1 \end{pmatrix} \tag{50}$$

Estabilidad de la solución Se quiere estudiar la estabilidad de la solución transiente de (**??**). Para ello se recurre a las expresiones (21) y (22). Luego,

$$\frac{d\vec{U}}{dt} = \boldsymbol{S}\vec{U} + \vec{Q} \rightarrow \begin{cases} (y^{n+1} - y^n)/\Delta t = \Omega_1 y^{n+1} \\ (z^{n+1} - z^n)/\Delta t = \Omega_2 z^{n+1} \end{cases} \tag{51}$$

Despejando los terminos evaluados en t_{n+1} en la izquierda de la ecuación

$$y^{n+1} = \frac{1}{1 - \Delta t \Omega_1} y^n \tag{52}$$

$$z^{n+1} = \frac{1}{1 - \Delta t \Omega_2} z^n \tag{53}$$

Se define z_{pj} como una aproximación de $G(\Omega_j)$ descrito en la Sección 2.4.

$$\begin{aligned} G(\Omega_1) \approx z_{py} = \frac{1}{1 - \Delta t \Omega_1} \\ G(\Omega_2) \approx z_{pz} = \frac{1}{1 - \Delta t \Omega_2} \end{aligned} \tag{54}$$

Es de interés conocer el módulo de z_p. La condición de estabilidad se garantiza por el cumplimiento de las restricciones expuestas en las ecuaciones (30) y (31)

$$\begin{aligned} z_p &= \frac{1}{1 - \Omega_j \Delta t} \\ &= \frac{1}{[1 - \Re(\Omega_j)\Delta t] - [\Im(\Omega_j)\Delta t]\, i} \\ &= \frac{[1 - \Re(\Omega_j)\Delta t] + [\Im(\Omega_j)\Delta t]\, i}{[1 - \Re(\Omega_j)\Delta t]^2 + [\Im(\Omega_j)\Delta t]^2} \end{aligned} \tag{55}$$

Entonces,

$$||z_p|| = \frac{\sqrt{[1 - \Re(\Omega_j)\Delta t]^2 + [\Im(\Omega_j)\Delta t]^2}}{[1 - \Re(\Omega_j)\Delta t]^2 + [\Im(\Omega_j)\Delta t]^2} \tag{56}$$

Se observa que $||z_p|| \leq 1, \ \forall \Omega_j \in \mathbb{C} : \Re(\Omega_j) < 0$. Es decir, la solución es incondicionalmente convergente si la parte real de los j-ésimos valores propios Ω_j son menores a cero. A continuación me muestra el cálculo de Ω para $\beta = \sqrt{\alpha}$ y $\beta = \alpha$ para pasos de tiempo $\Delta t = 10^{-4}$ y $\Delta t = 0.1$. En las Figura 7 y 8 se grafica la solución obtenida por la subrutina para $\beta = \sqrt{\alpha}$ y $\beta = \alpha$, respectivamente.

3.4.2 Crank Nicolson

Discretización de la ecuación diferencial Se utiliza una subrutina que implementa el esquema de Crank Nicolson para la resolución del problema (36). Al igual que en el caso anterior se resuelve el problema equivalente (**??**)

$$\frac{y^{n+1} - y^n}{\Delta t} = \frac{1}{2}\left(z^{n+1} + z^n\right) \tag{57}$$

$$\frac{z^{n+1} - z^n}{\Delta t} = \frac{1}{2}\left(-\alpha y^{n+1} - \beta y^{n+1} + \gamma(p_{n+1} - p_0)\right) + \frac{1}{2}\left(-\alpha y^n - \beta y^n + \gamma(p_n - p_0)\right) \tag{58}$$

La relación anterior se escribe en forma matricial,

$$\boldsymbol{A} \cdot \begin{Bmatrix} y^{n+1} \\ z^{n+1} \end{Bmatrix} = \boldsymbol{B} \cdot \begin{Bmatrix} y^n \\ z^n \end{Bmatrix} + \frac{\Delta t}{2} \begin{Bmatrix} 0 \\ (\gamma(p_{n+1} - p_0) + \gamma(p_n - p_0)) \end{Bmatrix} \tag{59}$$

donde,

$$\boldsymbol{A} = \begin{pmatrix} 1 & -\Delta t/2 \\ \alpha\Delta t/2 & 1+\beta\Delta t/2 \end{pmatrix} \tag{60}$$

$$\boldsymbol{B} = \begin{pmatrix} 1 & \Delta t/2 \\ -\alpha\Delta t/2 & 1-\beta\Delta t/2 \end{pmatrix} \tag{61}$$

Despejando las variables incognitas y^{n+1} y z^{n+1} se obtiene,

$$\begin{Bmatrix} y^{n+1} \\ z^{n+1} \end{Bmatrix} = \boldsymbol{A}^{n-1} \cdot \boldsymbol{B} \cdot \begin{Bmatrix} y^{n} \\ z^{n} \end{Bmatrix} + \frac{\Delta t}{2} \boldsymbol{A}^{-1} \cdot \begin{Bmatrix} 0 \\ (\gamma(p_{n+1} - p_0) + \gamma(p_n - p_0)) \end{Bmatrix} \tag{62}$$

donde,

$$\boldsymbol{A}^{-1} = \frac{1}{1 + \beta\dfrac{\Delta t}{2} + \alpha\left(\dfrac{\Delta t}{2}\right)^2} \begin{pmatrix} 1+\beta\Delta t/2 & \Delta t/2 \\ -\alpha\Delta t/2 & 1 \end{pmatrix} \tag{63}$$

Estabilidad de la solución Se quiere estudiar la estabilidad de la solución transiente. Al igual que para el caso de resolción mediante Euler Implícito,

$$\frac{d\vec{U}}{dt} = \boldsymbol{S}\vec{U} + \vec{Q} \rightarrow \begin{cases} (y^{n+1} - y^n)/\Delta t = (\Omega_1^{n+1}/2)y^{n+1} + (\Omega_1^n/2)y^n \\ (z^{n+1} - z^n)/\Delta t = (\Omega_1^{n+1}/2)z^{n+1} + (\Omega_1^n/2)z^n \end{cases} \tag{64}$$

Como $\Omega_j^n = \Omega_j^{n+1} = \Omega_j'$

$$\frac{y^{n+1} - y^n}{\Delta t} = \Omega_1' \frac{1}{2}\left(y^{n+1} + y^n\right) \tag{65}$$

$$\frac{z^{n+1} - z^n}{\Delta t} = \Omega_2' \frac{1}{2}\left(z^{n+1} + z^n\right) \tag{66}$$

Despejando los terminos evaluados en t_{n+1} en la izquierda de la ecuación

$$y^{n+1} = \left(\frac{1 + \Omega_1\Delta t/2}{1 - \Omega_1\Delta t/2}\right) y^n \tag{67}$$

$$z^{n+1} = \left(\frac{1 + \Omega_2\Delta t/2}{1 - \Omega_2\Delta t/2}\right) z^n \tag{68}$$

Se reconocen los términos $\vec{z}_{pj}$ que aproximan a $G(\Omega_j)$

$$z_p = \frac{1 + \Omega_j\Delta t/2}{1 - \Omega_j\Delta t/2} \tag{69}$$

Se observa que para $\Re(\Omega_j) < 0$ el modulo del numerador de (69) es menor que el denominador, es decir,

$$||1 + \frac{\Omega_j\Delta t}{2}|| < ||1 - \frac{\Omega_j\Delta t}{2}|| \tag{70}$$

Luego, se deduce que $||z_p|| < 1$ por lo tanto el esquema es incondicionalmente estable para $\Re(\Omega_j) < 0$.

En las Figura 9 y 10 se grafica la solución obtenida por la subrutina para $\beta = \sqrt{\alpha}$ y $\beta = \alpha$, respectivamente.

3.4.3 Resultados

La ecuación (36) es una ecuación diferencial ordinaria lineal de segundo orden con coeficientes constantes. Se resuelve el problema equivalente (**??**). Se obtiene un sistema de ecuaciones con dos incognitas $y(t)$ y $z(t) = dt/dt$, se pueden interpretar como un problema de 2 grados de libertad (Se obtiene una matriz $\boldsymbol{A}_{2\times 2}$). En las Tablas 3 y 4 se muestran los valores $||z_p||$ para los valores propios complejos ($\Im(\Omega_j) \neq 0$) y los valores de zp para los valores propios reales ($\Im(\Omega_j) = 0$)

Se puede ver que los esquemas poseen son estables en el tiempo para ambos esquemas de discretización ya que $||zp\,(\approx G(\Omega_j))|| < 1$. Notar que los valores propios tienen una significancia asociada a las características de la ecuación diferencial y no del esquema de discretización espacial. Ω_1 está asociado a y y Ω_2 a su derivada dy/dt.

Para los casos (a) se tiene que los valores propios son números con parte imaginaria distinta de cero. La parte real es negativa, por lo que la solución converge a la solución permanente, pero su componente imaginaria produce perturbaciones de forma oscilatoria. Luego, la solución transiente es una oscilación amortiguada en el tiempo ; Para los casos (b) se obtienen valores propios reales negativos, por lo que la solución transiente es una exponencial que decae a la solución estacionaria.

En el esquema de Euler Implícito el modulo de z_p es mayor para pasos de tiempo más reducidos. Esto se debe a que al aumentar el tamaño del paso de tiempo se reducen la cantidad de pasos de tiempos, por lo que la solución debe debe converger con mayor rapidez, dicho de otra manera, si $z_{pa} < z_{pb}$ entonces z_{pa}^k converge más rápido que z_{pb}^k para un mismo k. Euler implícito es un esquema de orden 1, por lo que se caracteriza por ser un esquema disipativo

El esquema de Crank Nicolson ocurre lo contrario: el módulo de z_p es menor para pasos de tiempo más reducidos. Para el caso (b) donde se obtiene valores propios reales se observa que z_p asociado $z = dy/dt$ es negativo, entonces el comportamiento de z_p^k es oscilatorio (para k pasos de tiempo). Como se ve en las Figuras, el esquema presenta una estabilidad marginal. Esto se condice con el hecho que Crank Nicolson es un esquema de orden 2, por lo tanto se caracteriza por ser un esquema dispersivo

Euler Implícito

(a) $\beta = \sqrt{\alpha}$

- Para $\Delta t = 10^{-4}$ (Simulación 1)

$$\begin{array}{lcl} \Omega_1 = -3000.00 + 5196.15i & \rightarrow & ||z_{p1}|| = 0.152501 \\ \Omega_2 = -3000.00 - 5196.15i & \rightarrow & ||z_{p2}|| = ||z_{p1}|| \end{array}$$

- Para $\Delta t = 0.1$ (Simulación 2)

$$\begin{array}{lcl} \Omega_1 = -3000.00 + 5196.15i & \rightarrow & ||z_{p1}|| = 0.00166531 \\ \Omega_2 = -3000.00 - 5196.15i & \rightarrow & ||z_{p2}|| = ||z_{p1}|| \end{array}$$

(b) $\beta = \alpha$

- Para $\Delta t = 10^{-4}$ (Simulación 1)

$$\begin{array}{lcl} \Omega_1 = -1.0 & \rightarrow & z_{p1} = 0.9999 \\ \Omega_2 = -36.0 \times 10^6 & \rightarrow & z_{p2} = 2.777 \times 10^{-4} \end{array}$$

- Para $\Delta t = 0.1$ (Simulación 2)

$$\begin{array}{lcl} \Omega_1 = -1.0 & \rightarrow & z_{p1} = 0.9090 \\ \Omega_2 = -36.0 \times 10^6 & \rightarrow & z_{p2} = 2.780 \times 10^{-7} \end{array}$$

Tabla 3.

Crank Nicolson

(a) $\beta = \sqrt{\alpha}$

- Para $\Delta t = 10^{-4}$ (Simulación 1)

$$\begin{array}{lll} \Omega_1 = -3000.00 + 5196.15i & \rightarrow & ||z_{p1}|| = 0.84907 \\ \Omega_2 = -3000.00 - 5196.15i & \rightarrow & ||z_{p2}|| = ||z_{p1}|| \end{array}$$

- Para $\Delta t = 0.1$ (Simulación 2)

$$\begin{array}{lll} \Omega_1 = -3000.00 + 5196.15i & \rightarrow & ||z_{p1}|| = 0.9983 \\ \Omega_2 = -3000.00 - 5196.15i & \rightarrow & ||z_{p2}|| = ||z_{p1}|| \end{array}$$

(b) $\beta = \alpha$

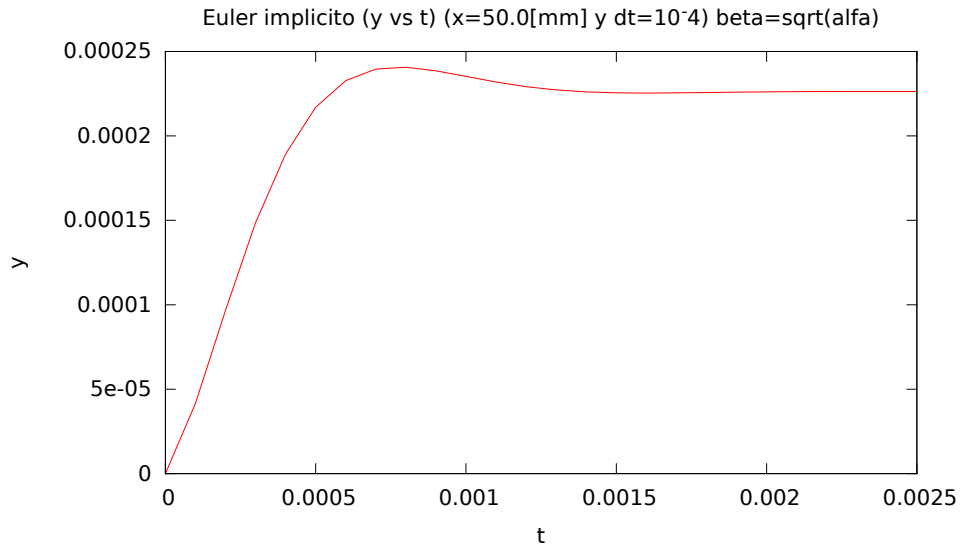
- Para $\Delta t = 10^{-4}$ (Simulación 1)

$$\begin{array}{lll} \Omega_1 = -1.0 & \rightarrow & z_{p1} = 0.9999 \\ \Omega_2 = -36.0 \times 10^6 & \rightarrow & z_{p2} = -0.9989 \end{array}$$

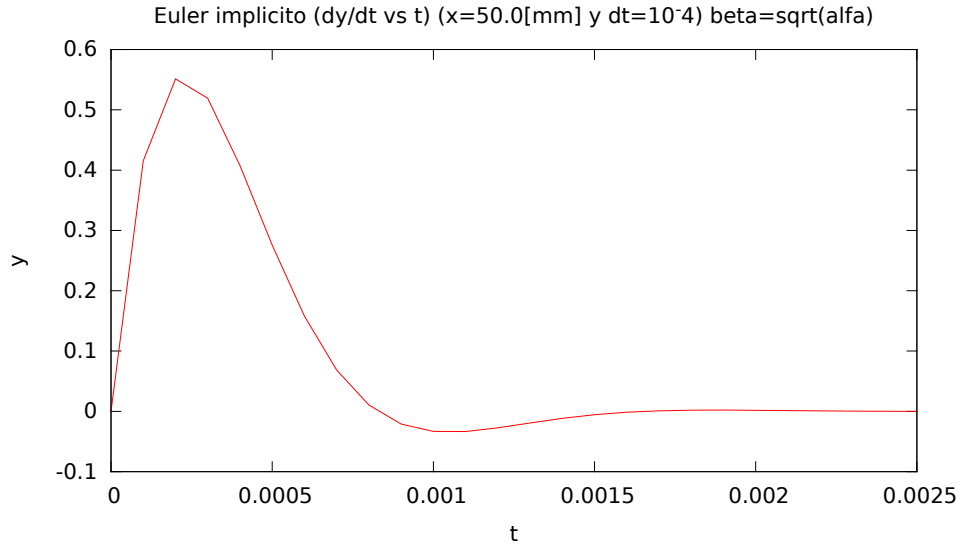
- Para $\Delta t = 0.1$ (Simulación 2)

$$\begin{array}{lll} \Omega_1 = -1.0 & \rightarrow & z_{p1} = 0.9048 \\ \Omega_2 = -36.0 \times 10^6 & \rightarrow & z_{p2} = -0.9999 \end{array}$$

Tabla 4.

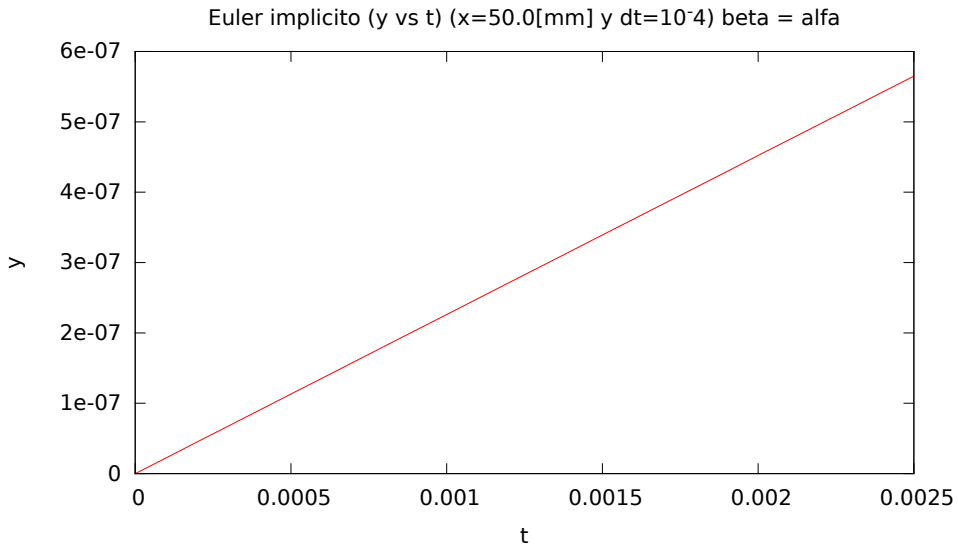


(a) Grafico de y vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

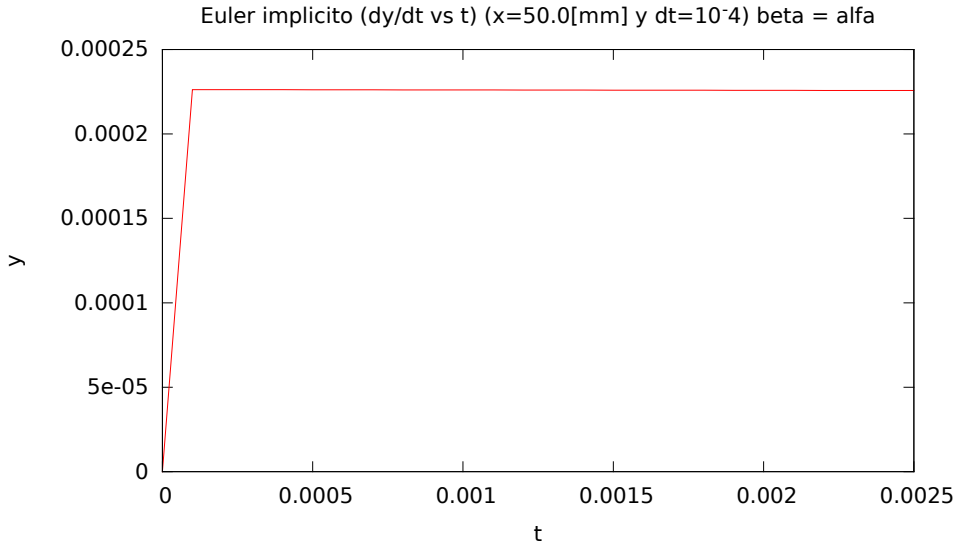


(b) Grafico de dy/dt vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

Figura 7. Simulación 1: Solución numérica empleando Euler Implícito para $\beta = \sqrt{\alpha}$

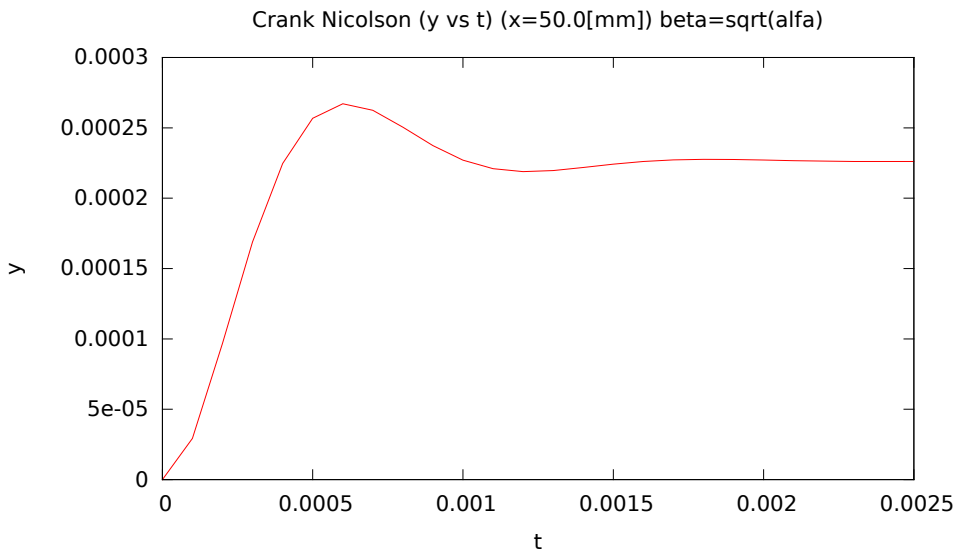


(a) Grafico de y vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

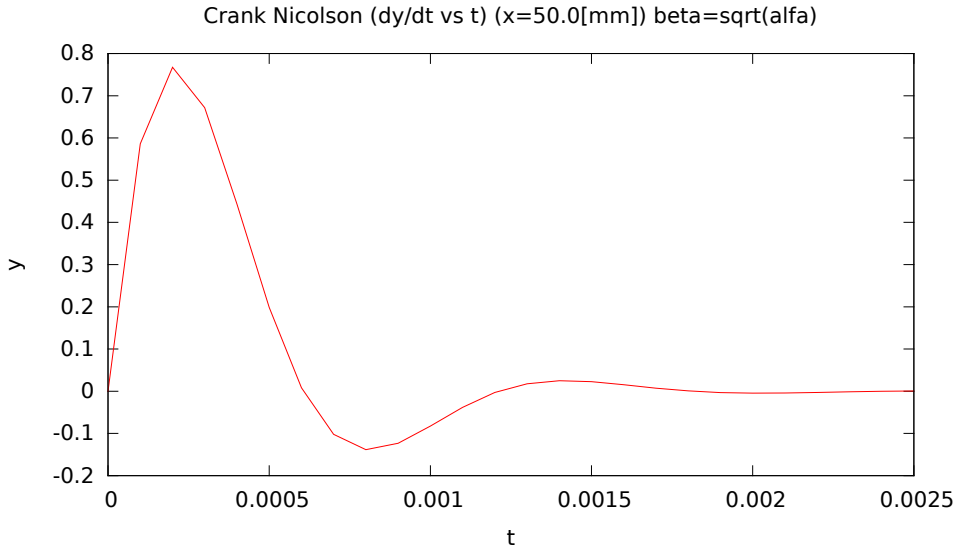


(b) Grafico de dy/dt vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

Figura 8. Simulación 1: Solución numérica empleando Euler Implícito para $\beta = \alpha$

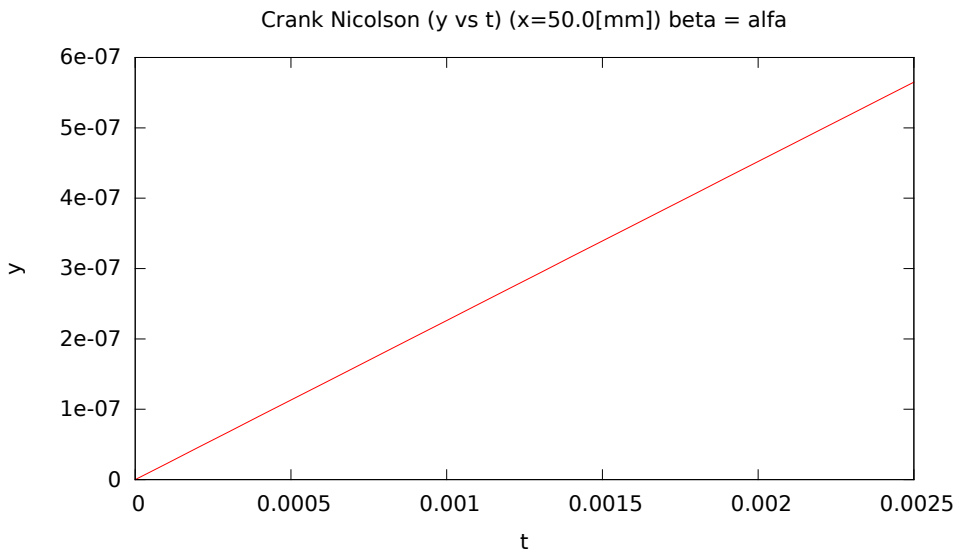


(a) Grafico de y vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

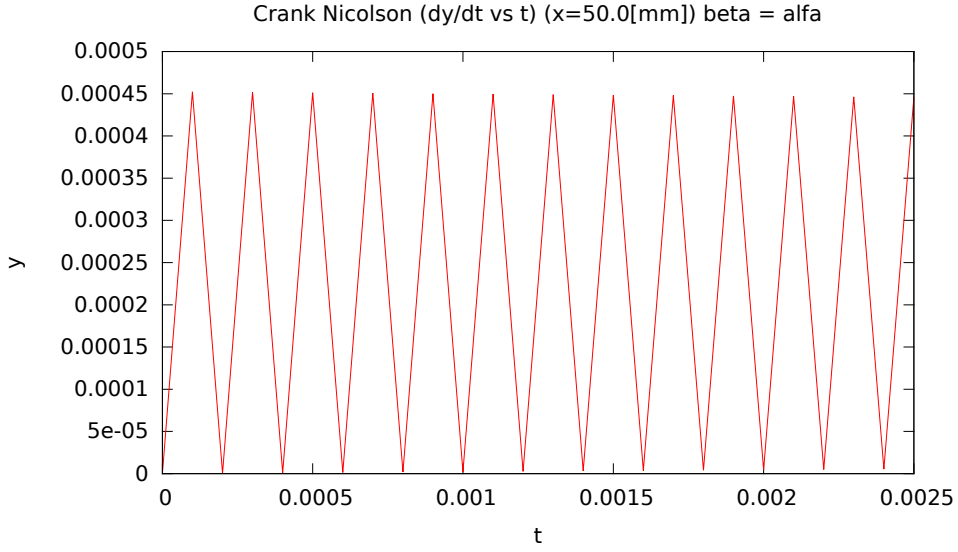


(b) Grafico de dy/dt vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

Figura 9. Simulación 1: Solución numérica empleando Crank Nicolson para $\beta = \sqrt{\alpha}$

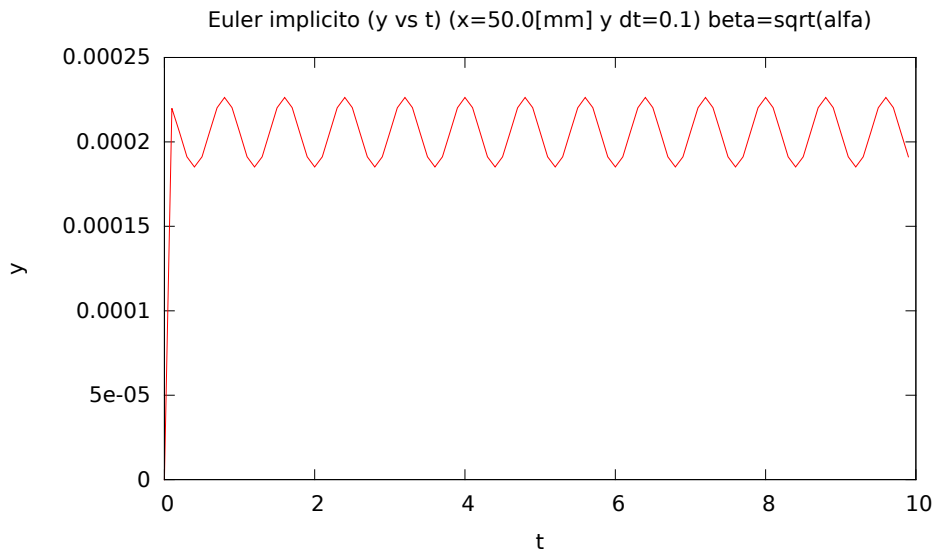


(a) Grafico de y vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

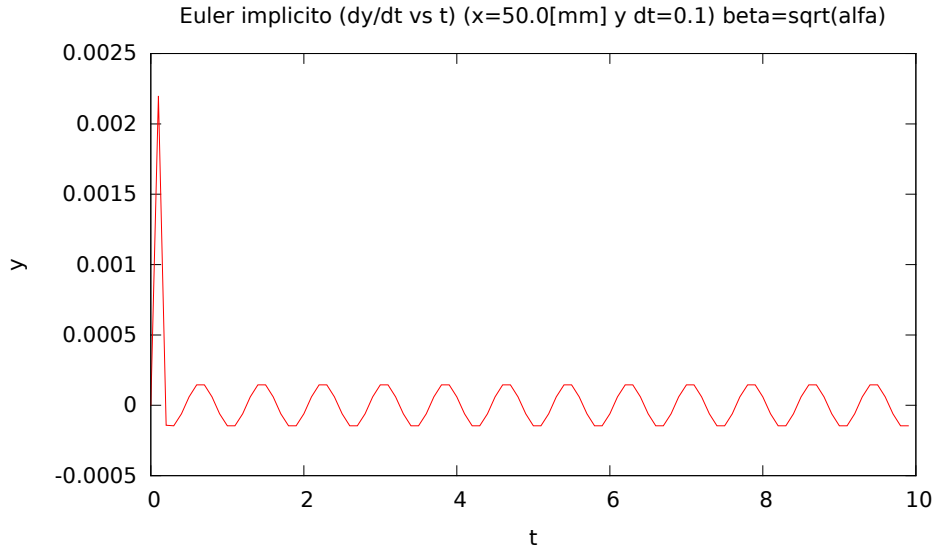


(b) Grafico de dy/dt vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 10^{-4}$

Figura 10. Simulación 1: Solución numérica empleando Crank Nicolson para $\beta = \sqrt{\alpha}$

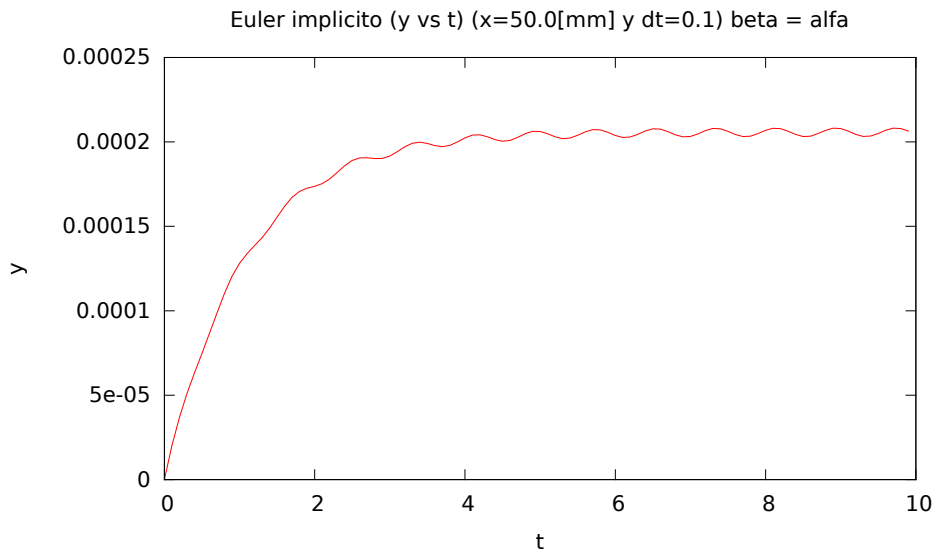


(a) Grafico de y vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 0.1$

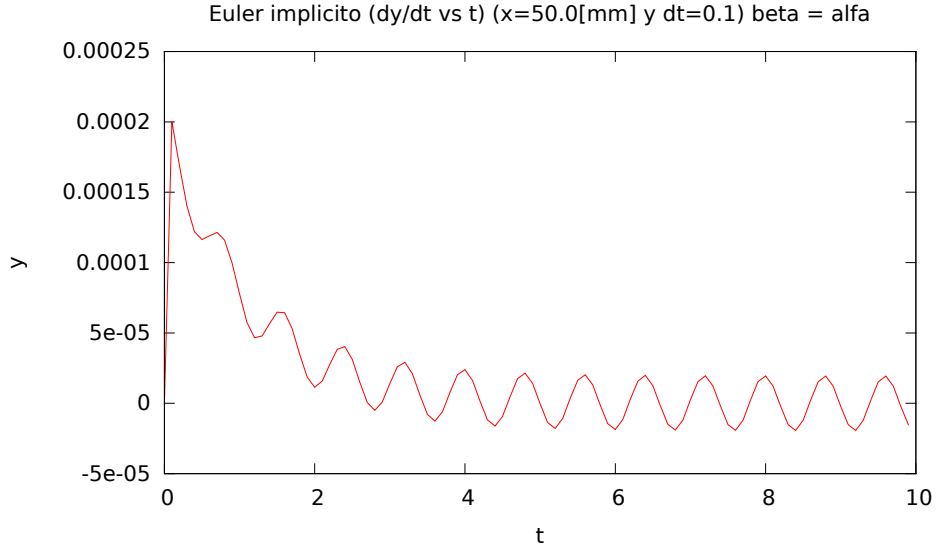


(b) Grafico de dy/dt vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 0.1$

Figura 11. Simulación 2: Solución numérica empleando Euler Implícito para $\beta = \alpha$

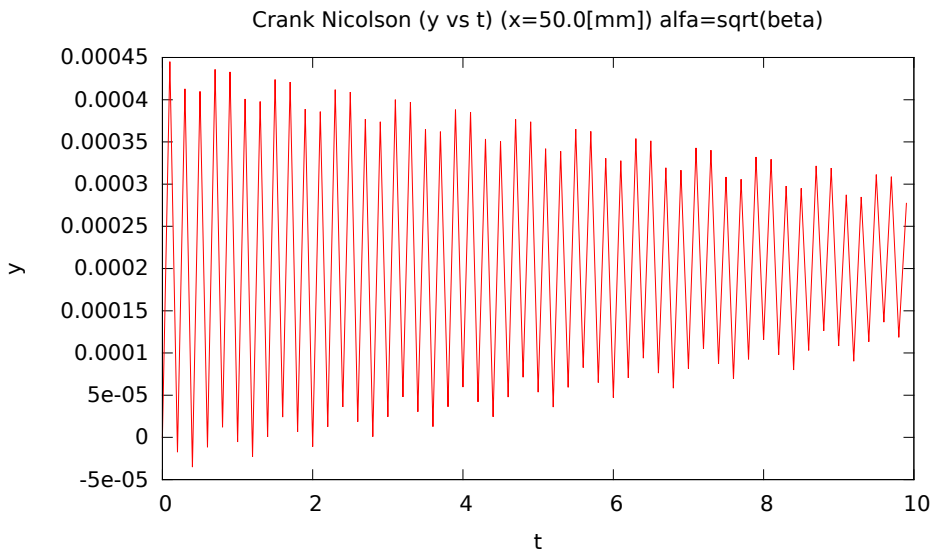


(a) Grafico de y vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 0.1$

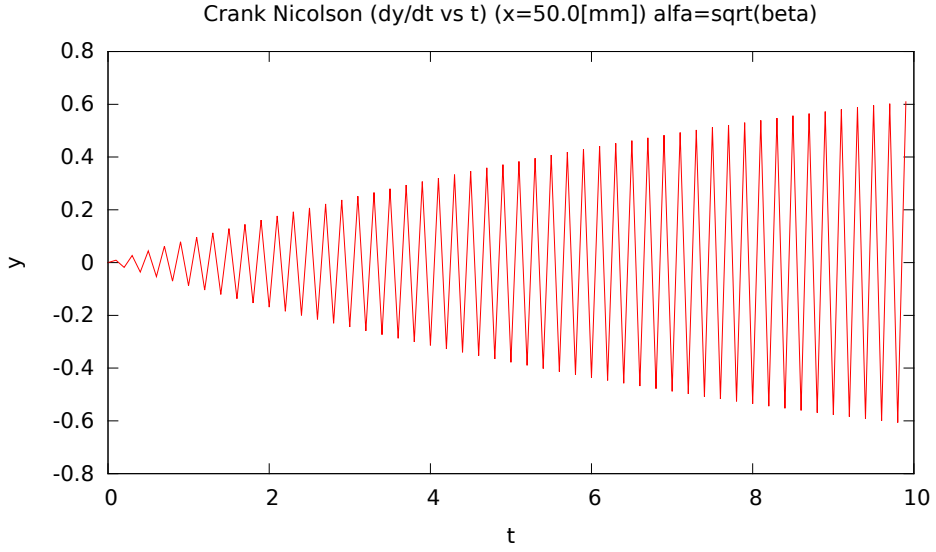


(b) Grafico de dy/dt vs t empleando un esquema de integración de Euler Implícito. $x = 0.5[mm]$ y $\Delta t = 0.1$

Figura 12. Simulación 2: Solución numérica empleando Euler Implícito para $\beta = \sqrt{\alpha}$

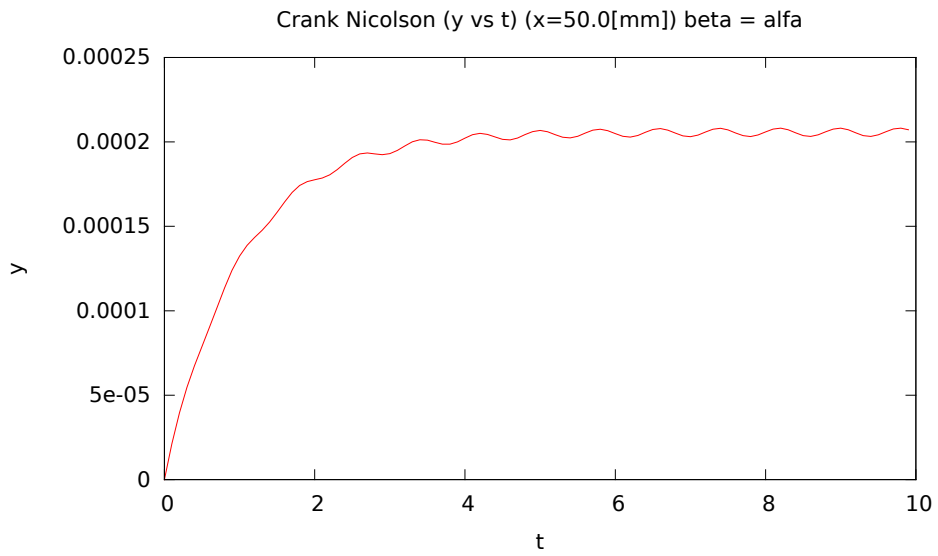


(a) Grafico de y vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 0.1$

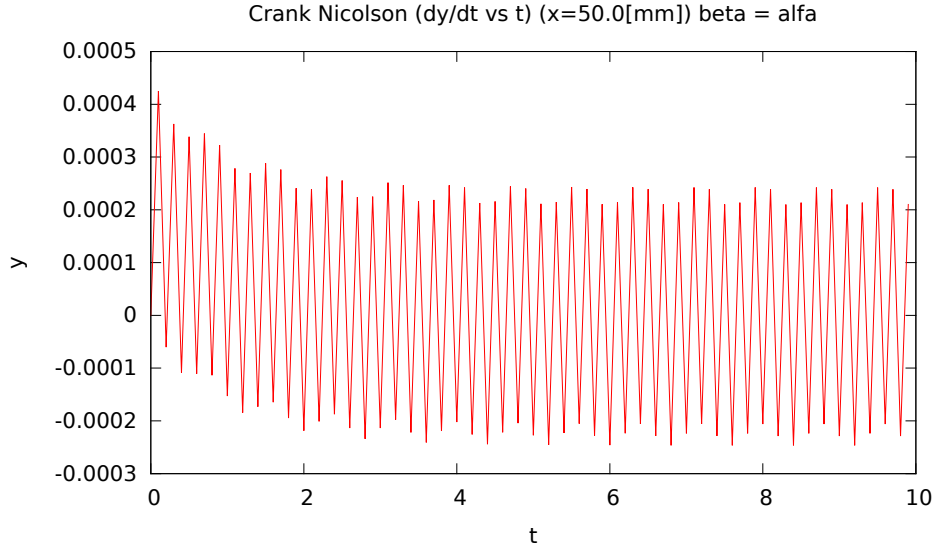


(b) Grafico de dy/dt vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 0.1$

Figura 13. Simulación 2: Solución numérica empleando Crank Nicolson para $\beta = \alpha$



(a) Grafico de y vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 0.1$



(b) Grafico de dy/dt vs t empleando un esquema de integración de Crank Nicolson. $x = 0.5[mm]$ y $\Delta t = 0.1$

Figura 14. Simulación 2: Solución numérica empleando Crank Nicolson para $\beta = \sqrt{\alpha}$

3.5 Parte 2: Un modelo hiperbólico para la interacción de la sangre con la pared

Si para el mismo fenomeno descrito en la Parte 1 no se desprecia la interacción axial entre los anillos, entonces la ecuación (36) se modifica resultado en,

$$\rho_\omega H \frac{\partial^2 y}{\partial t^2} - \sigma_x \frac{\partial^2 y}{\partial x^2} + \frac{H\,E}{R_0^2} y = p - p_0 \qquad t > 0 \qquad 0 < x < L \tag{71}$$

Se denota la coordenada longitudinal x. σ_x es la componente radial del esfuerzo axial y L es el largo del cilindro considerado. Despreciando el factor de y de la ecuación (71) y considerando $p - p_0 = f$ entonces se obtiene la ecuación de onda en una dimensión

$$\frac{\partial^2 u}{\partial t^2} - \gamma \frac{\partial^2 u}{\partial x^2} = f \qquad x \in]\alpha, \beta[\qquad t > 0 \tag{72}$$

Se emplean los esquemas de Leap-Frog y Newmark para discretizar la ecuación anterior.

3.5.1 Leap-Frog

El termino fuente utilizado para la simulación es $f = (1 + \pi^2\gamma^2)e^{-t}sin(\pi x)$. Empleando un esquema de diferencias centradas en el espacio,

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} + (1 + \pi^2\gamma^2)e^{-t}sin(\pi x_j) \tag{73}$$

Se utiliza el esquema de integración temporal Leap-Frog,

$$u_j^{n+1} - 2u_j^n + u^{n-1} = (\gamma\lambda)^2(u_{j-1}^n - 2u_j^n + u_{j+1}^n) + f_j^n \tag{74}$$

donde $\lambda = \Delta t/\Delta x$

Discretización espacial Se realiza una descomposición modal como se expuso en la Sección 2.4,

$$\begin{aligned} Se^{ik_j p\Delta x} &= \frac{\gamma^2}{\Delta x^2}\left(e^{ik_j(p+1)\Delta x} - 2e^{ik_j p\Delta x} + e^{ik_j(p-1)\Delta x}\right) \\ &= \underbrace{\frac{2\gamma^2}{\Delta x^2}\left(cos(k_j\Delta x) - 1\right)}_{\Omega_j} e^{ik_j p\Delta x} \end{aligned}$$

Entonces, los valores propios Ω_j son,

$$\Omega_j = \frac{2\gamma^2}{\Delta x^2}\left(cos(k_j\Delta x) - 1\right) \tag{75}$$

Notar que $\Omega_j = \Re(\Omega_j)$. Se debe cumplir que:

$$\Re(\Omega_j) \leq 0$$

$$\frac{2\gamma^2}{\Delta x^2}\left[cos(k_j\Delta x) - 1\right] \leq 0 \tag{76}$$

Finalmente se debe cumplir que,

$$\frac{-4\gamma^2}{\Delta x^2} \leq \Re(\Omega_j) \leq 0 \tag{77}$$

Discretización temporal Estudia la estabilidad en el tiempo: se reemplaza $u_j = \omega$

$$\frac{\omega^{n+1} - 2\omega^n + \omega^{n+1}}{\Delta t^2} = \boldsymbol{S} = \Omega_j \omega^n \tag{78}$$

Reeordenando,

$$\omega^{n+1} - (\Omega_j \Delta t^2 + 2)\omega^n + \omega^{n-1} = 0 \tag{79}$$

Sea $z_p \approx G(\Omega_j)$ una aproximación del factor de amplificación, se puede escribir la ecuación anterior como,

$$z_p^2 \omega^n - z_p(\Omega_j \Delta t^2 + 2)\omega^n + \omega^{n-1} = 0 \tag{80}$$

Se deduce entonces,

$$z_p^2 - \left(\Omega_j \Delta t^2 + 2\right) z_p + 1 = 0 \tag{81}$$

Diviendo por z_p ($z_p \neq 0$ ya que se está trabajando en estado transiente)

$$\left(\Omega_j \Delta t^2 + 2\right) = \frac{1}{z_p} + z_p \tag{82}$$

Se impone como solución $z_p = e^{i\theta}$ (notar que $|z_p| \leq 1$) . Reemplazando en la ecuación anterior,

$$\left(\Omega_j \Delta t^2 + 2\right) = e^{i\theta} + e^{-i\theta} = 2cos(\theta) \tag{83}$$

La estabilidad se consigue acotando el lado izquierdo de la ecuación, obteniendo

$$-4 \leq \Re(\Omega_j \Delta t^2) \leq 0 \tag{84}$$

Reemplazando Ω_j obtenido de la ecuación (75) y considerando $\Omega_j = \Re(\Omega_j)$ resulta,

$$-4 \leq \frac{\gamma^2 \Delta t^2}{\Delta x^2} \left(cos(k_j \Delta x) - 1\right) \leq 0 \tag{85}$$

El criterio de estabilidad es

$$\frac{\gamma^2 \Delta t^2}{\Delta x^2} \leq 2 \tag{86}$$

$\gamma^2 \Delta t^2 / \Delta x^2$ se puede interpretar como $(CFL)^2$ donde CFL es el número de Courant-Friedrichs-Lewy

3.5.2 Newmark

De la ecuacion (15) (Sección 2.3.4) se tiene $\partial u / \partial t = v$

$$\frac{\partial v}{\partial t} = \gamma^2 \left[\Theta \left(\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2}\right) - (1 - \Theta) \left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}\right)\right] \tag{87}$$

Discretización espacial Se utiliza $\Theta = 0.5$ (Esquema Crank Nicolson)

$$\frac{\partial v}{\partial t} = \left[\left(\boldsymbol{S} e^{ik_j p \Delta x}\right)^{n+1} + \left(\boldsymbol{S} e^{ik_j p \Delta x}\right)^n\right] \tag{88}$$

Se resulve $(\boldsymbol{S})^n$

$$\begin{aligned} S e^{ik_j p \Delta x} &= \frac{\gamma^2}{2\Delta x^2} \left(e^{ik_j (p+1) \Delta x} - 2e^{ik_j p \Delta x} + e^{ik_j (p-1) \Delta x}\right) \\ &= \underbrace{\frac{\gamma^2}{\Delta x^2} \left(cos(k_j \Delta x) - 1\right)}_{\Omega_j} e^{ik_j p \Delta x} \end{aligned}$$

Es decir,

$$\Omega_j = \frac{\gamma^2}{\Delta x^2}\left(cos(k_j \Delta x) - 1\right) \tag{89}$$

$$\frac{-2\gamma^2}{\Delta x^2} \leq \Re(\Omega_j) \leq 0 \tag{90}$$

Se obtiene el mismo resultado para $(\boldsymbol{S})^{n+1}$, Sumando $(\boldsymbol{S})^{n+1} + (\boldsymbol{S})^n$ luego obtenemos la mismas ecuación obtenida en la discretización Leap-Frog. Por lo tanto, la condición de $\Re(\Omega'_j)$ ($\Omega'_j = \Omega_j^{n+1} + \Omega_j^n$)

$$\frac{-4\gamma^2}{\Delta x^2} \leq \Re(\Omega'_j) \leq 0 \tag{91}$$

Notar que u_j y v_j emplean diferencias finitas centradas para discretizar el dominio, por lo tanto se obtienen los mismos valores propios de $\boldsymbol{S}$

Discretización temporal Se integra la ecuación $\partial v/\partial t$. Se utiliza la notación $v_j = \omega$

$$\frac{\omega^{n+1} - \omega^n}{\Delta t} = \Omega_j^{n+1}\omega^{n+1} + \Omega_j^n \omega^n \tag{92}$$

agrupando términos

$$\omega^{n+1} = \underbrace{\frac{1 + \Omega_j \Delta t}{1 - \Omega_j \Delta t}}_{z_p} \omega^n \tag{93}$$

como $\Omega_j = \Re(\Omega_j)$, entonces

$$z_p = \frac{1 + \Omega_j}{1 - \Omega_j} \tag{94}$$

tomando en cuenta la condición (90) se verifica que

$$-1 \leq z_p \leq 1 \tag{95}$$

3.5.3 Resultados

A continuación se muestra la convergencia de los métodos Leap-Frog y Newmark

$t_j^{(0)}$	$p_{LF}^{(1)}$	$p_{LF}^{(2)}$	$p_{LF}^{(3)}$	$t_j^{(0)}$	$p_{NW}^{(1)}$	$p_{NW}^{(2)}$	$p_{NW}^{(3)}$
0.1000	-1.8233	-1.7515	-1.2447	0.1000	2.5229	2.0236	2.3471
0.2000	-1.3451	-1.1519	-0.5895	0.2000	1.6501	1.7153	2.2387
0.3000	-0.7807	-0.5283	0.0635	0.3000	1.3800	1.6463	2.2604
0.4000	0.1544	0.4775	1.1064	0.4000	1.2057	1.6738	2.3914
0.5000	2.6457	3.7409	5.3562	0.5000	0.8952	1.9875	3.2940
0.6000	1.8799	2.1175	2.7069	0.6000	1.6377	1.5219	2.0791
0.7000	4.5569	4.8068	4.6839	0.7000	1.3065	2.4637	4.5019
0.8000	1.5724	1.7390	2.2901	0.8000	1.3260	0.9585	1.3992
0.9000	1.2640	1.5152	2.1067	0.9000	1.3014	1.4944	2.0995
1.0000	1.7824	2.1298	2.7728	1.0000	1.3536	1.8813	2.6445

Tabla 5.

3.6 Atractor de Lorenz

El sistema de ecuaciones de Lorenz es un ejemplo de un sistema de ecuaciones diferenciales de orden 1, tridimensional, no lineal, que tiene un comportamiento caótico para algunos valores de sus parámetros. Este sistema de ecuaciones permite modelar los rollos de convección que se producen en la atmosfera terrestre. Es un modelo simplificado de la convección de Rayleigh-Benard (ecuaciones de Navier-Stokes con hipótesis de Boussinesq).

$$\begin{cases} dx/dt = & Pr(y(t) - x(yt)) \\ dy/dt = & Rax(t) - y(t) - x(t)z(t) \\ dz/dt = & x(t)y(t) - \beta z(t) \end{cases} \tag{96}$$

Donde Pr es el número de Prandtl y Ra es el número de Rayleigh. Las variables dinámicas x, y y z represetan el estado del sistema a cada instante t:

- x(t) es proporcional a la intensidad del movimiento de convección
- y(t) es proporcional a la diferencia de temperatura entre las corrientes ascendentes y descendentes
- z(t) es proporcional a la diferencia entre el perfil vertical de temperatura y un perfil vertical de temperatura lineal

Los sistemas dinámicos son sistemas que son función del tiempo. Que sea caótico significa que varia de manera no lineal y a su vez que el sistema presenta sensibilidad a frente a los parámetros entrada. Además presentan un comportamiento oscilante, pudiendo ser periodico o no periodico.

3.6.1 Parte 1

Las Figuras 15 y 16 reproducen los gráficos del articulo *Deterministic Nonperiodic Flow* de Lorenz

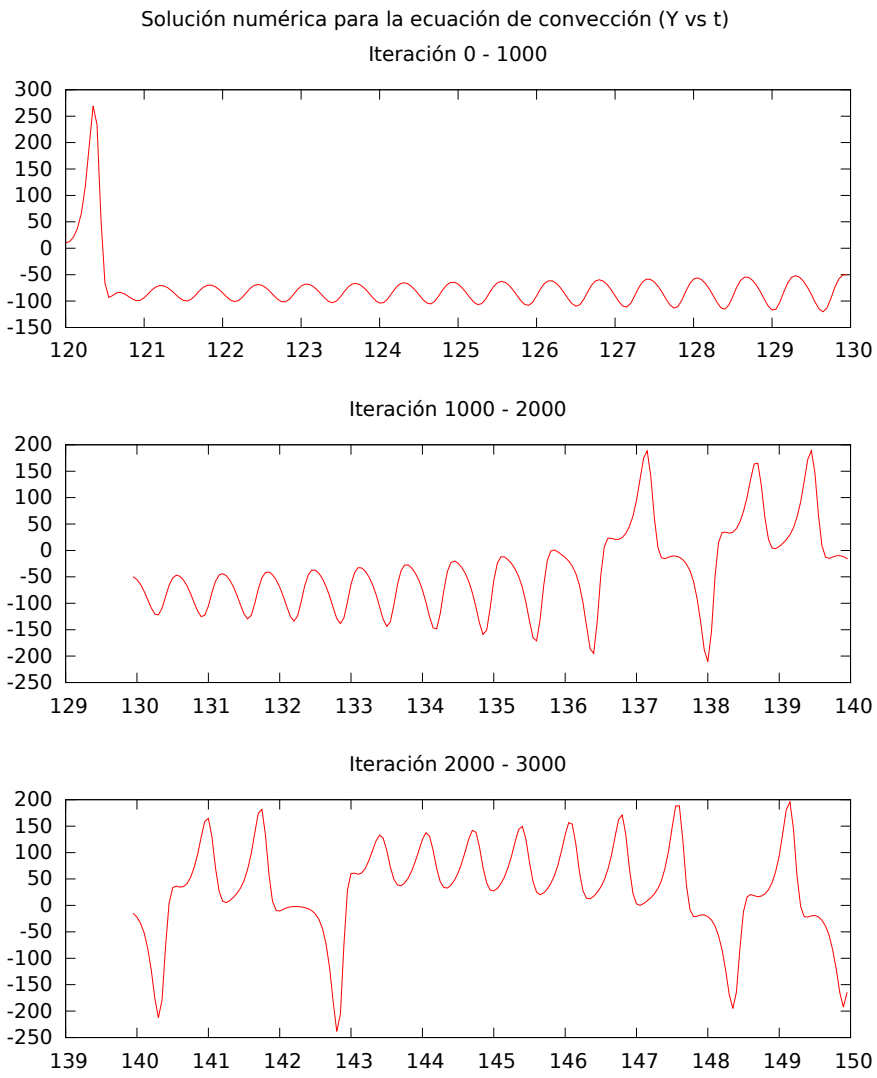

Solución numérica para la ecuación de convección (Y vs t)
Iteración 0 - 1000
300
250
200
150
100
50
0
-50
-100
-150
120
121
122
123
124
125
126
127
128
129
130
Iteración 1000 - 2000
200
150
100
50
0
-50
-100
-150
-200
-250
129
130
131
132
133
134
135
136
137
138
139
140
Iteración 2000 - 3000
200
150
100
50
0
-50
-100
-150
-200
-250
139
140
141
142
143
144
145
146
147
148
149
150

Figura 15.

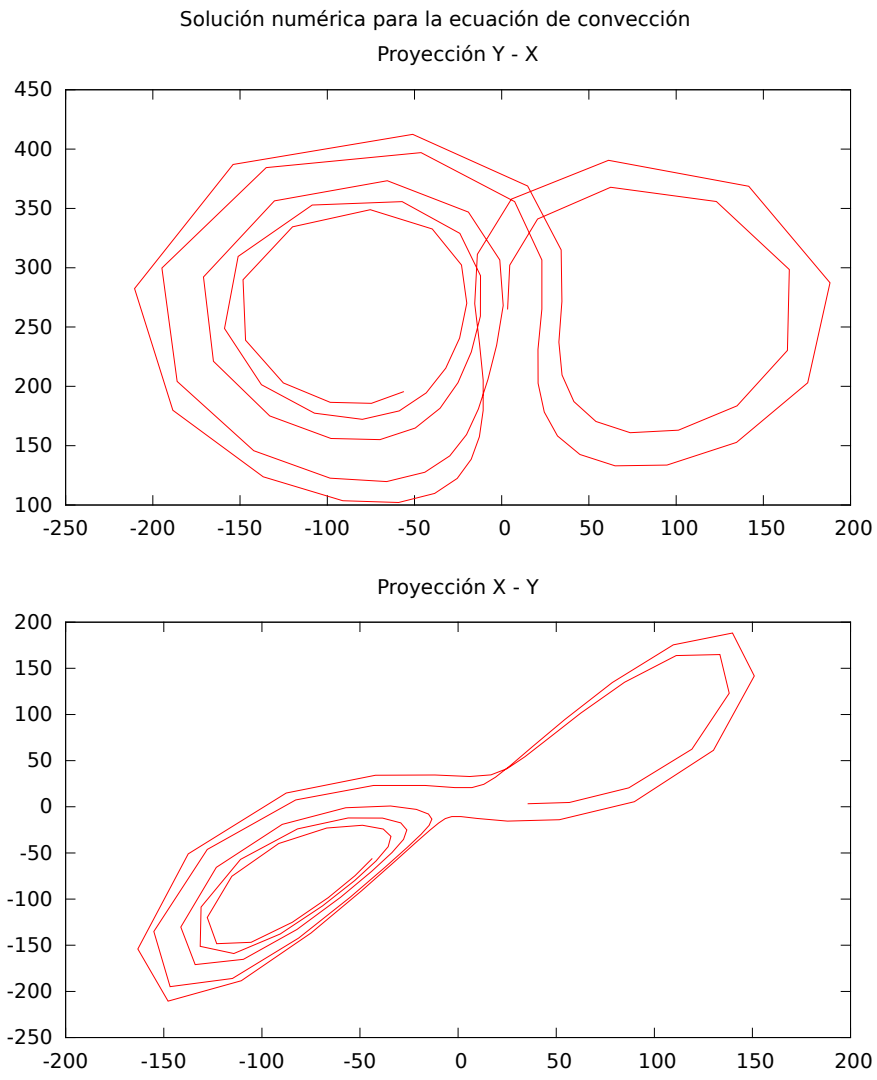


Figura 16.

3.6.2 Parte 2

Se grafica la solución a la ecuación de convección utilizando los siguientes valores: $Pr = 10$, $\beta = 8/3$ y $Ra = 0.5\,,\ 10,\ 28$

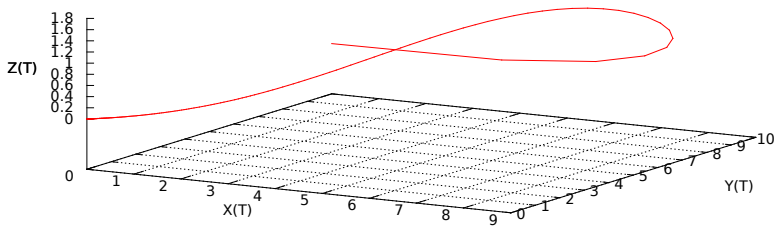


Figura 17.

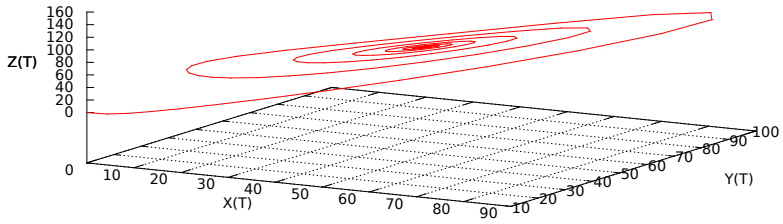


Figura 18.

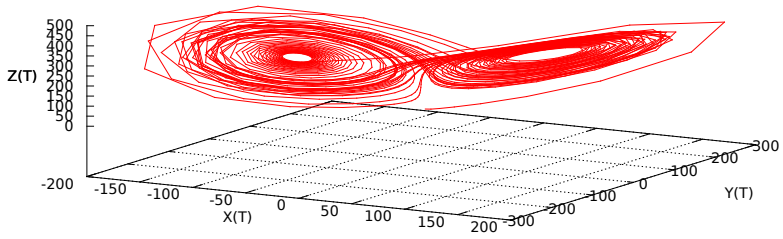
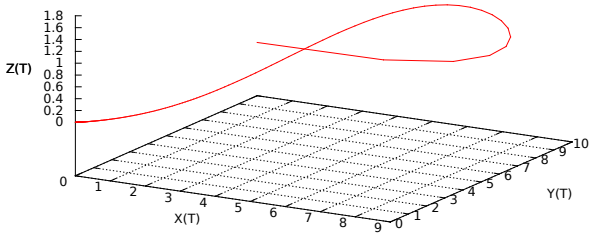


Figura 19.

3.6.3 Parte 3

Haciendo variar paulatinamente Ra entre 0 y 30

Solución de la ecuación de convección (Ra=00)



Solución de la ecuación de convección (Ra=03)

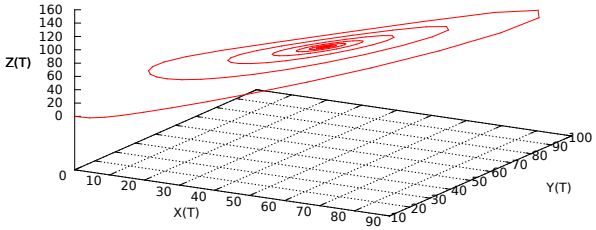
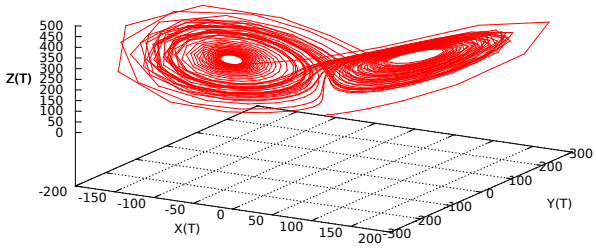


Figura 20.

Solución de la ecuación de convección (Ra=06)



Solución de la ecuación de convección (Ra=09)

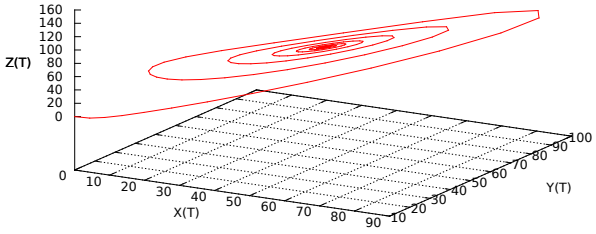
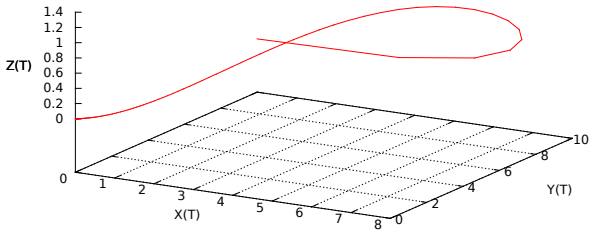


Figura 21.

Solución de la ecuación de convección (Ra=12)



Solución de la ecuación de convección (Ra=15)

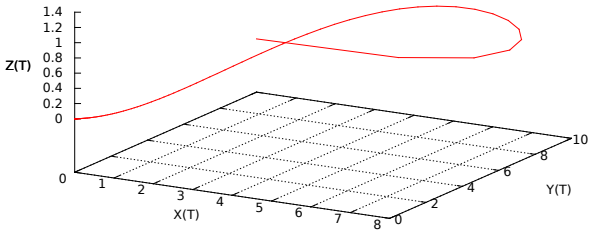
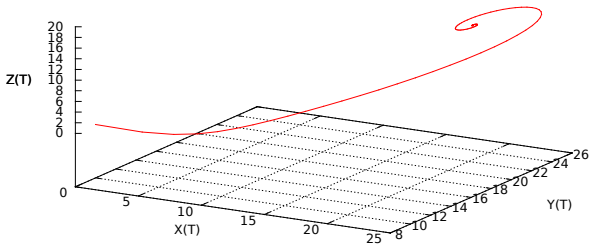


Figura 22.

Solución de la ecuación de convección (Ra=18)



Solución de la ecuación de convección (Ra=18)

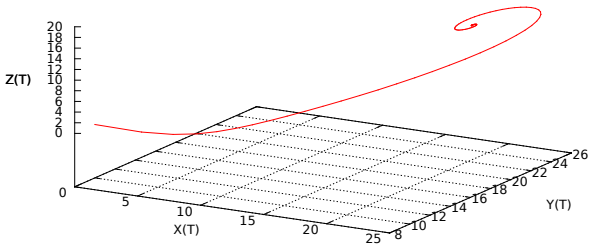
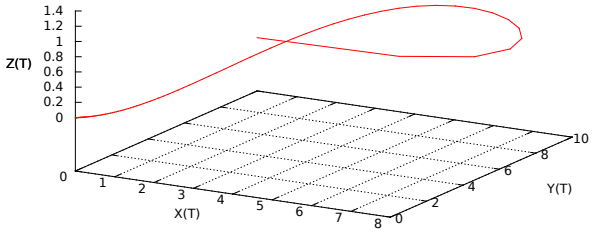


Figura 23.

Solución de la ecuación de convección (Ra=21)



Solución de la ecuación de convección (Ra=24)

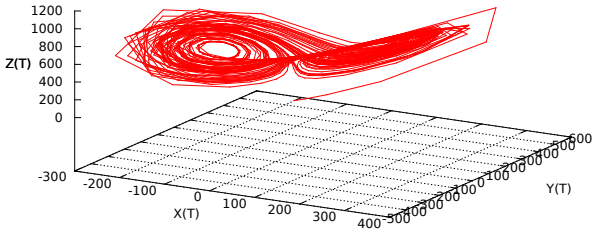
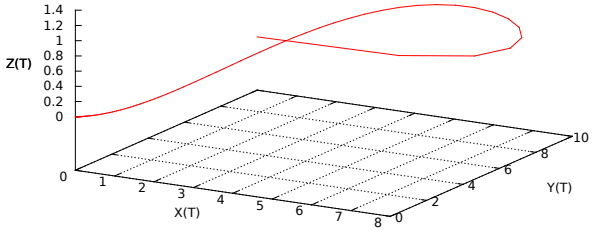


Figura 24.

Solución de la ecuación de convección (Ra=27)



Solución de la ecuación de convección (Ra=30)

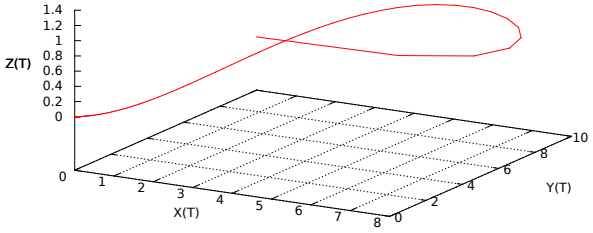


Figura 25.

4 Conclusiones y Observaciones

Ejercicios en Fortran Se estudió la relevancia del parámetro `precision` en la implementación de las rutinas. En el Ejercicio 1 se observa que para un número determinado de operaciones la precisión representa un freno debido a la debil resolución numérica, es decir, para una cantidad de decimales que se supera la cantidad de cifras significativas no alteran al objeto, por que la operación virtualmente no ocurre.

El caso contrario ocurre en el Ejercicio 2. Se trabajó con número enteros y reales y se observó el comportamiento de ambos al representar la serie Fibonacci. La inestabilidad se presenta cuando la operación numérica produce un número de mayor tamaño del que la memoria asignada puede contemplar. Cuando sucede esto ocurre una asignación de memoria que no representa ninguna operación, es decir, se obtiene un resultado aleatorio sin significancia física ni matematica.

El Ejercicio 3 se implementa una subrutina que permite realizar la operación de multiplicación matricial. Se implementa una rutina que realiza la multiplicación cuando las dimensiones de los dos factores son consistentes con los de la operación. La rutina implementada trabaja con un parametro l de tal manera que $l = 0$ cuando la operación en satisfactoria, y $l = 1$ cuando ocurre un error. Este tipo de consideración son relevantes al momento de implementar algoritmos: Una práctica aconsejable al momento de programar es establercer detenciones o *flags* que permitan identificar cuando ocurre una operación indeseado o se obtienen resultado malogrados. De esta manera se puede identificar facilmente el problema, con los tiempo de revisión y tiempo de cómputo malgastado.

Estudio del comportamiento de una arteria. Parte 1 Se implementó una rutina de integración de Euler Implícito y una de Crank Nicolson para la resolución de la ecuación (**??**).

$$y' = \boldsymbol{A}\vec{y} + b(x,t)$$

Se observa que es una ecuación diferencial ordinaria, donde el término fuente es el único término que es función de la variable espacial x. En este caso la descomposición modal $\boldsymbol{A}V^{(j)} = \Omega_j V^{(j)}$ tiene una significancia distinta a la descomposición obtenida de un esquema de diferencias finitas: En el primer caso los valores propios están asocidos a la naturaleza del fenomeno (o bien, a la clasifica de la EDO), mientras que en el segundo caso está asociado a la estabilidad de la discretización.

Euler representa un esquema disipativo de orden 1, mientras que Crank Nicolson es un esquema dispersivo de orden 2. Al comparar las solución se observa que ambos métodos arrojan resultados distintos para un mismo problema. Además para un el caso $\beta = \sqrt{\alpha}$ se obtiene una solución con un término transiente oscilatorio. Se obtuvo numericamente que el método de Crank Nicolson es un esquema dispersivo. En este caso una solución que presenta oscilaciones se resuelve de manera desfavorable con ciertos métodos, particularmente, el método de Crank Nicolson, al ser un esquema dispersivo, agrega errores claramente visibles.

Estudio del comportamiento de una arteria. Parte 2 Se resolvió la ecuación de onda mediante los métodos de Leap-Frog y Newmark. Ambos métodos son *three level scheme* ya que la discretización de la derivada temporal requiere de 3 valores n, $n+1$ y $n+2$. Ambos esquemas son de orden 2, luego, se espera que la tabla de convergencia 5 se obtuviesen valores cercanos a 2. Discrepancias en los resultados pueden tener algunas posibles explicaciones:

- Iniciación (condiciones iniciales), Para obtener esta tabla se fue necesario conocer la función analítica para calcular los errores. Sin embargo se utilizó la condición inicial u_j^0 y se aproximó u_j^1 mediante diferencias finitas hacia adelante.
- Para la resolución de Newmark se implementó una subrutina *solver* que permite invertir la matriz que acompaña las incognitas u^{n+1} mediante factorización pivoteada LU. Este método

no es el más efectivo y puede incluir errores numéricos. Ya que se trata de un sistema tridiagonal (matriz sparse o con varios elementos nulos) lo mejor es utilizar un subrutina que trabaje que estos valores no nulos. Metodos más eficientes pueden verse en [5], o utilizando librerias especializadas como LAPACK-BLAS.

5 Codigos implementados

5.1 Ejercicios en Fortran

5.1.1 Ejercicio 1

```
program EJERCICIOS_EN_FORTRAN_1
        !-----------------------------------------------
        !PREAMBULO
        integer::i,inf,contador
        real(kind=4)::A !simple precision
        real(kind=8)::B,error !doble precision
        character(len=22)::datos_p1='./datos/datos_p1.dat'
        character(len=22)::datos_pe='./datos/datos_p1_error.dat'
        !-----------------------------------------------

        !simple y doble precisión
        A=1._4
        B=1._8
        inf=4000000
        contador=1

        ! los datos generados se exportan al directorio
        ! ./datos/datos_p1.dat
        open(unit=10,file=datos_p1,action='write')
        do i=2,inf
                !contador
                contador = contador + 1
                !desarrollo de la serie
                A = A + ( 1._4 / real(i,kind=4) )
                B = B + ( 1._8 / real(i,kind=8) )
                error = (B-real(A,kind=8))/B
                !cada 1000 operaciones exportar datos
                if ( contador .eq. 10000 ) then
                        if ( i .gt. 250000 ) then
                                write(10,*) i, A, B, error
                        end if
                        contador=0
                end if
        end do
        close(unit=10)

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

        ! se grafica la curva en el directorio
        ! ./graficos/graficos_p1.dat
        call system ( ' cd ./gnuplot && gnuplot plot_p1_1.txt ' )

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

end program
```

5.1.2 Ejercicio 2

```
program EJERCICIOS_EN_FORTRAN_2
        !-----------------------------------------------
        !PREAMBULO
        integer::n,u0,u1,u2,j
        real(kind=8)::u0_r,u1_r,u2_r
        character(len=22)::datos_p2_1='./datos/datos_p2_1.dat'
        character(len=22)::datos_p2_2='./datos/datos_p2_2.dat'
        !-----------------------------------------------

        !se calcula la serie para número enteros
        n=100
        u0=0 ; u1=1
        open(unit=10,file=datos_p2_1,action='write')
                write(10,*)0,u0
                write(10,*)1,u1
                do j=2,n
                        u2=u0+u1
                        write(10,*)j,u2
                        u0=u1
                        u1=u2
                end do
        close(unit=10)

        !se calcula la serie para número reales d. precision
        n=1500
        u0_r = 0._8 ; u1_r = 1._8
        open(unit=10,file=datos_p2_2,action='write')
                write(10,*)0,u0_r
                write(10,*)1,u1_r
                do j=2,n
                        u2_r = u0_r + u1_r
                        write(10,*) j , u2_r
                        u0_r = u1_r
                        u1_r = u2_r
                end do
        close(unit=10)

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

        ! se grafica la curva en el directorio
        ! ./graficos/graficos_p1.dat
        call system ( ' cd ./gnuplot && gnuplot plot_p1_2.txt ' )
```

5.1.3 Ejercicio 3

```
!EL PROGRAMA EJECUTA DOS OPERACIONES DE EJEMPLO
!COMO SE MUESTRA A CONTINUACION

!PRIMER CASO AxB=C (DIMENSIONES CONSISTENTES)

!       A =     1       2       3       4
!               5       6       7       8
!               9       10      11      12

!       B =     1       2
!               3       4
!               5       6
!               7       8

!SEGUNDO CASO AxD (DIMENSIONES INCOSITENTES)

!       A =     1       2       3       4
!               5       6       7       8
!               9       10      11      12

!       D =     1       2
!               3       4
!               5       6

program EJERCICIOS_EN_FORTRAN_3
        !-----------------------------------------------
        !INTERFACE
        implicit none
        interface
                subroutine matrix_mult(A,B,C,i)
                real(kind=4),dimension(:,:),intent(in)::A,B
                real(kind=4),dimension(:,:),intent(out)::C
                integer,intent(out)::i
                end subroutine
        end interface
        !-----------------------------------------------
        !PREAMBULO

        !PREGUNTA 3
        integer::k,i,tmp,tmp_v(2)
        real(kind=4) :: A(3,4),B(4,2),D(3,2)
        real(kind=4),allocatable :: C(:,:)

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

        !presentacion en pantalla
        write(*,*) '----------------------'
        write(*,*) 'EJERCICIOS_EN_FORTRAN_3'
        write(*,*) 'codigo              : P1_3.f90'
        write(*,*) 'ejecutable  : P1_3'
        write(*,*)
        write(*,*) 'Este programa implementan la'
```

```
write(*,*) "subrutina 'matrix_mult' en"
write(*,*) 'dos casos posibles:'
write(*,*)
write(*,*) '1. multiplicacion de matrices'
write(*,*) 'con dimensiones consistentes'
write(*,*)
write(*,*) '2. multiplicacion de matrices'
write(*,*) 'con dimensions inconsistentes'
write(*,*) '----------------------'

!PRIMER CASO AxB=C (DIMENSIONES CONSISTENTES)
write(*,*) '----------------------'
write(*,*) 'PRIMER CASO AxB=C '//&
&'(DIMENSIONES CONSISTENTES)'
write(*,*)

!se crean las matrices A, B y D
A=reshape((/(1._8*k,k=1,12)/),(/3,4/))
B=reshape((/(1._8*k,k=1,8)/),(/4,2/))
D=reshape((/(1._8*k,k=1,6)/),(/3,2/))


!se muestran las matrices A y B matrices
write(*,*) 'matriz A'
do k=1,3
        write(*,*) A(k,:)
end do
write(*,*)
write(*,*) 'matriz B'
do k=1,4
        write(*,*) B(k,:)
end do
write(*,*)

!se ejecuta la subrutina matrix_mult
call matrix_mult(A,B,C,i)

!se muestra el resultado C en pantalla
if (i.eq.0) then
        tmp_v=shape(C)
        tmp=tmp_v(1)
        do k=1,tmp
                write(*,*) C(k,:)
        end do
else
        write(*,*) 'ERROR!'
        write(*,*) 'DIMENSIONES INCONSISTENTES'
end if

deallocate(C)
!-----------------------------------------------

!SEGUNDO CASO AxD (DIMENSIONES INCOSITENTES)
write(*,*) '----------------------'
```

```
        write(*,*) 'SEGUNDO CASO AxD '//&
        &'(DIMENSIONES INCONSISTENTES)'
        write(*,*)

        !se muestran las matrices A y D matrices
        write(*,*) 'matriz A'
        do k=1,3
                write(*,*) A(k,:)
        end do
        write(*,*)
        write(*,*) 'matriz D'
        do k=1,3
                write(*,*) D(k,:)
        end do
        write(*,*)

        !se ejecuta la subrutina matrix_mult
        call matrix_mult(A,D,C,i)

        !se muestra el resultado C en pantalla
        if (i.eq.0) then
                tmp_v=shape(C)
                tmp=tmp_v(1)
                do k=1,tmp
                        write(*,*) C(k,:)
                end do
        else
                write(*,*) 'ERROR!'
                write(*,*) 'DIMENSIONES INCONSISTENTES'
        end if

end program
```

Subrutina: matrix_mult()

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! la subrutina matrix_mult realiza la multiplicacion
! de las matrices A(na,ma) y B(nb,mb). Si ma=nb ento-
! nces se obtiene como resultado una matriz C(na,mb)
! y l=0 . Si ma=/nb entonces l=1
!
! entrada
! A: matriz lado izquierdo
! B: matriz lado derecho
!
! salida
! C: matriz resultante
! l: indicador de error
!       l=0 operacion satisfactoria (m. consistentes)
!       l=1 fallo (matrices inconsistentes)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine matrix_mult(A,B,C,l)
        !entrada
```

```
        real(kind=4),dimension(:,:),intent(in)::A,B
        !salida
        integer,intent(out)::l
        real(kind=4),dimension(:,:),allocatable,intent(out)::C
        !variables locales
        integer::i,j
        integer,dimension(2)::dim_a,dim_b,dim_c
        real(kind=4)::tmp
        !-----------------------------------------------
        l=0
        dim_a=shape(A)
        dim_b=shape(B)
        if ( dim_a(2) .ne. dim_b(1) ) then
                l=1
                return
        end if
        allocate(C(dim_a(1),dim_b(2)))
        dim_c=(/dim_a(1),dim_b(2)/)
        do i=1,dim_c(1)
                do j=1,dim_c(2)
                        tmp=0._8
                        do k=1,dim_a(2) !do k=1,dim_b(1)
                                tmp=tmp+A(i,k)*b(k,j)
                        end do
                        C(i,j)=tmp
                end do
        end do
        !-----------------------------------------------
end subroutine
```

5.2 Estudio del comportamiento mécanico de una arteria

5.2.1 Parte 1

```
program simulacion_1

        use modulo_pregunta2
        implicit none

        integer::i,j,k,nt
        integer,parameter::nx=11
        real(kind=np)::alfa,gama,dt,dx
        real(kind=np),dimension(2)::beta
        complex(kind=np),dimension(2,2)::lambda
        real(kind=np),dimension(nx)::x
        real(kind=np),dimension(:),allocatable::t
        real(kind=np),dimension(:,:,:,:),allocatable::y_euler,y_cn
        character(len=11)::data_file1='./datos/S1/'
        character(len=11)::data_file2='./datos/S2/'
        character(len=1),dimension(2)::data_beta

        !-------------------------------------------------------
        !                                   SIMULACION 1
        !-------------------------------------------------------

        nt = 26
        allocate(t(nt),y_euler(nt,nx,2,2),y_cn(nt,nx,2,2))

        !parametros del problema
        alfa=E/(rho_w*R_0**2._8)
        gama=1._8/(rho_w*H)
        beta(1) = sqrt(alfa)
        beta(2) = alfa

        !discretizacion espacio-tiempo (t=0)
        dt=0.0001_8
        dx=0.005_8
        t(:)=(/ ((i-1)*dt,i=1,nt) /)
        x(:)=(/ ((i-1)*dx,i=1,nx) /)

        !condiciones iniciales
        do i=1,nx
                y_euler(1,i,:,1) = (/ 0._8 , 0._8 /)
                y_euler(1,i,:,2) = (/ 0._8 , 0._8 /)
                y_cn(1,i,:,1) = (/ 0._8 , 0._8 /)
                y_cn(1,i,:,2) = (/ 0._8 , 0._8 /)
        end do

!       !calculo valores propios de A
        call eigenval(alfa,beta(1),lambda(1,1),lambda(1,2))
        call eigenval(alfa,beta(2),lambda(2,1),lambda(2,2))

        !euler implicito
        do k=1,2                        ! beta=sqrt(alfa) y beta=alfa
                do j=2,nt               ! discretizacion tiempo
```

```
                do i=1,nx       ! discretizacion espacio
                        call euler_implicito(alfa,beta(k),gama,t(j),dt,x(i),&
                                &y_euler((j-1),i,:,k),y_euler(j,i,:,k))
                end do
        end do
end do


!crank nicholson
do k=1,2                        ! beta=sqrt(alfa) y beta=alfa
        do j=2,nt               ! discretizacion tiempo
                do i=1,nx       ! discretizacion espacio
                        call crank_nicolson(alfa,beta(k),gama,t(j-1:j),x(i),&
                                &y_cn((j-1),i,:,k),y_cn(j,i,:,k))
                end do
        end do
end do

!---------------- exportar datos----------------------

data_beta(1) = '1' ; data_beta(2) = '2'
do k=1,2
        !grafica para y_euler(x=0.005,t) vs t
        open(unit=10, action="write", &
        &file=data_file1//'datos_euler_S1_y_b'//data_beta(k)//'.txt')
                do i=1,nt
                        write(10,*) t(i) , y_euler(i,nx,1,k)
                end do
        close(10)
        !grafica para dy_euler(x=0.005,t) vs t
        open(unit=10, action="write", &
        &file=data_file1//'datos_euler_S1_dy_b'//data_beta(k)//'.txt')
                do i=1,nt
                        write(10,*) t(i) , y_euler(i,nx,2,k)
                end do
        close(10)
        !grafica para y_cn(x=0.005,t) vs t
        open(unit=10, action="write",&
        &file=data_file1//'datos_cn_S1_y_b'//data_beta(k)//'.txt')
                do i=1,nt
                        write(10,*) t(i) , y_cn(i,nx,1,k)
                end do
        close(10)
        !grafica para dy_cn(x=0.005,t) vs t
        open(unit=10, action="write",&
        &file=data_file1//'datos_cn_S1_dy_b'//data_beta(k)//'.txt')
                do i=1,nt
                        write(10,*) t(i) , y_cn(i,nx,2,k)
                end do
        close(10)
end do

deallocate(t,y_euler,y_cn)
```

```
!------------------ graficar ---------------------------

call system('cd gnuplot/ && gnuplot plot_p2_1.txt')


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!-------------------------------------------------------------
!                              SIMULACION 2
!-------------------------------------------------------------

!se conservan los parámetros, cambia el paso y el
!intervalo de tiempo
nt = 100
allocate(t(nt),y_euler(nt,nx,2,2),y_cn(nt,nx,2,2))
dt = 0.1_8
t(:)=(/ ((i-1)*dt,i=1,nt) /)

!condiciones iniciales (t=0)
do i=1,nx
        y_euler(1,i,:,1) = (/ 0._8 , 0._8 /)
        y_euler(1,i,:,2) = (/ 0._8 , 0._8 /)
        y_cn(1,i,:,1) = (/ 0._8 , 0._8 /)
        y_cn(1,i,:,2) = (/ 0._8 , 0._8 /)
end do

!euler implicito
do k=1,2                        ! beta=sqrt(alfa) y beta=alfa
        do j=2,nt               ! discretizacion tiempo
                do i=1,nx       ! discretizacion espacio
                        call euler_implicito(alfa,beta(k),gama,t(j),dt,x(i),&
                                y_euler((j-1),i,:,k),y_euler(j,i,:,k))
                end do
        end do
end do

!crank nicholson
do k=1,2                        ! beta=sqrt(alfa) y beta=alfa
        do j=2,nt               ! discretizacion tiempo
                do i=1,nx       ! discretizacion espacio
                        call crank_nicolson(alfa,beta(k),gama,t(j-1:j),x(i),&
                                &y_cn((j-1),i,:,k),y_cn(j,i,:,k))
                end do
        end do
end do

!---------------- exportar datos-----------------------

data_beta(1) = '1' ; data_beta(2) = '2'
```

```
        do k=1,2
                !grafica para y_euler(x=0.005,t) vs t
                open(unit=10, action="write", &
                &file=data_file2//'datos_euler_S2_y_b'//data_beta(k)//'.txt')
                        do i=1,nt
                                write(10,*) t(i) , y_euler(i,nx,1,k)
                        end do
                close(10)
                !grafica para dy_euler(x=0.005,t) vs t
                open(unit=10, action="write", &
                &file=data_file2//'datos_euler_S2_dy_b'//data_beta(k)//'.txt')
                        do i=1,nt
                                write(10,*) t(i) , y_euler(i,nx,2,k)
                        end do
                close(10)
                !grafica para y_cn(x=0.005,t) vs t
                open(unit=10, action="write",&
                &file=data_file2//'datos_cn_S2_y_b'//data_beta(k)//'.txt')
                        do i=1,nt
                                write(10,*) t(i) , y_cn(i,nx,1,k)
                        end do
                close(10)
                !grafica para dy_cn(x=0.005,t) vs t
                open(unit=10, action="write",&
                &file=data_file2//'datos_cn_S2_dy_b'//data_beta(k)//'.txt')
                        do i=1,nt
                                write(10,*) t(i) , y_cn(i,nx,2,k)
                        end do
                close(10)
        end do

        deallocate(t,y_euler,y_cn)

        !---------------- graficar ------------------------

        call system('cd gnuplot/ && gnuplot plot_p2_1.txt')

end program
```

Subrutina: eigenval()

```
subroutine eigenval(alfa,beta,lamb1,lamb2)
!-------------------------------------------
        use modulo_pregunta2
        implicit none
        !variables entrada
        real(kind=np),intent(in)        :: alfa,beta
        !variables salida
        complex(kind=np),intent(out)    :: lamb1,lamb2
!-------------------------------------------
        if ( beta .ge. 2._8*sqrt(alfa) ) then
                lamb1 = complex( 0.5_8*(-beta+sqrt(beta**2d0-4d0*alfa)) , 0._8 )
                lamb2 = complex( 0.5_8*(-beta-sqrt(beta**2d0-4d0*alfa)) , 0._8 )
        else
                lamb1 = complex( -0.5_8*beta , 0.5_8*sqrt(-beta**2d0+4d0*alfa) )
```

```
                lamb2 = complex( -0.5_8*beta , -0.5_8*sqrt(-beta**2d0+4d0*alfa) )
        end if
!-------------------------------------------------
end subroutine
```

Subrutina: euler_implicito

```
subroutine euler_implicito(alfa,beta,gama,t,dt,x,y_0,y_1)
!-------------------------------------------------
        use modulo_pregunta2
        implicit none
        !variables entrada
        real(kind=np),intent(in) :: alfa,beta,gama,t,dt,x
        !variables entrada-salida
        real(kind=np),dimension(2),intent(inout)          :: y_0,y_1
        !variables locales
        integer                                                  :: i,j
        real(kind=np)                                     :: suma
        real(kind=np),dimension(2)                  :: yn0,yn1,f_0
        real(kind=np),dimension(2,2)      :: Matrix_A
!-------------------------------------------------
        Matrix_A(1,:) = (/ 1._8 + beta*dt , dt /)
        Matrix_A(2,:) = (/ -alfa*dt , 1._8 /)
        Matrix_A = Matrix_A / ( 1._8 + beta*dt + alfa*dt**2._8 )
        f_0(:) = (/ 0._8 , x*dt*gama*dp*(a+b*cos(w*(t))) /)
        do i=1,2
                suma=0d0
                do j=1,2
                        suma = suma + Matrix_A(i,j)*(y_0(j)+f_0(j))
                enddo
                y_1(i)=suma
        end do
end subroutine
```

Subrutina: crank_nicolson()

```
subroutine crank_nicolson(alfa,beta,gama,t,x,y_0,y_1)
!-------------------------------------------------
        use modulo_pregunta2
        implicit none
        !variables entrada
        real(kind=np),intent(in) :: alfa,beta,gama,x
        real(kind=np),dimension(2),intent(in)::t
        !variables entrada-salida
        real(kind=np),dimension(2),intent(inout)          :: y_0,y_1
        !variables locales
        integer                                                  :: i,j
        real(kind=np)                                     :: suma,dt
        real(kind=np),dimension(2)                  :: yn0,yn1,f_0
        real(kind=np),dimension(2,2)      :: Matrix_A,Matrix_B
!-------------------------------------------------
        dt = t(2) - t(1)

        !matriz A
        Matrix_A(1,:) = (/ 1._8 + beta*dt*0.5_8 - alfa*dt**2._8*0.25_8 ,
dt /)
```

```
        Matrix_A(2,:) = (/ -alfa*dt , 1._8 - beta*dt*0.5_8 - alfa*0.25_8*dt**2._8 /)
        Matrix_A = Matrix_A/(1._8 + beta*dt*0.5_8 + alfa*dt**2._8*0.25_8)

        !matriz B
        Matrix_B(1,:) = (/ 1._8 + beta*dt*0.5_8 , dt*0.5_8 /)
        Matrix_B(2,:) = (/ -alfa*dt*0.5_8 , 1._8 /)
        Matrix_B = Matrix_B/(1._8 + beta*dt*0.5_8 + alfa*dt**2._8*0.25_8)

        !vector f_0
        f_0(:) = (/ 0._8 , x*dt*gama*dp*(a+b*cos(w*t(1)))*0.5_8 + &
                &x*dt*gama*dp*(a+b*cos(w*t(2)))*0.5_8  /)

        !calcular y_(n+1) y dy_(n+1)
        do i=1,2
                suma=0d0
                do j=1,2
                        suma = suma + Matrix_A(i,j)*y_0(j) + Matrix_B(i,j)*f_0(j)
                enddo
                y_1(i)=suma
        end do

end subroutine
```

Parte 2

```
program simulacion_3
!-------------------------------------------------------
        use modulo_pregunta2
        implicit none

        interface
                subroutine leap_frog(gama,lambda,t,dt,x,y_0,y_1,y_2)
                        implicit none
                        real(kind=8),intent(in) :: gama,lambda,t,dt
                        real(kind=8),dimension(:),intent(in) :: x
                        real(kind=8),dimension(:),intent(inout) :: y_0,y_1,y_2
                end subroutine
                subroutine newmark(gama,lambda,t,dt,x,theta,beta,u_0,u_1,v_0,v_1)
                        implicit none
                        real(kind=8),intent(in) :: gama,lambda,t,dt,theta,beta
                        real(kind=8),dimension(:),intent(in) :: x
                        real(kind=8),dimension(:),intent(inout) :: u_0,u_1,v_0,v_1
                end subroutine
        end interface

        type y_data
                real(kind=8),dimension(:,:),allocatable::kk
        end type
        type xt_data
                real(kind=8),dimension(:),allocatable::kk_
        end type

        integer::nx,nt_lf,nt_nm,k,i,j,q1,q2,contador1,contador2,ii,jj
        integer,dimension(4)::datos_nx
        integer,dimension(2,4)::datos_nt
```

```
real(kind=8),dimension(4)::datos_dx
real(kind=8),dimension(2,4)::datos_dt
real(kind=8)::dx,dt_lf,dt_nm,lambda_lf,lambda_nm,gama,theta,beta
real(kind=8),dimension(4,10)::e_lf,e_nm,p_lf,p_nm
real(kind=8),dimension(2,10)::tiempo
type(y_data),dimension(4)::y_lf,y_nm,dy_nm,y_exacta_lf,y_exacta_nm
type(xt_data),dimension(4)::x,t_lf,t_nm
character(len=11)::data_folder='./datos/S3/'
character(len=20)::data_folder_informe='./../INFORME/parte3/'

!para la simulación 2
integer::nt
real(kind=8)::y_lf1(101,11),y_lf2(401,11),y_nm1(101,11),dy_nm1(101,11),&
        &y_nm2(401,11),dy_nm2(401,11),dt,TT,xx(11),vt1(101),vt2(401),lambda

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

!SIMULACIÓN 1

do k=1,4

        !discretizacion
        call discretizacion(k-1,nx,dx,nt_lf,dt_lf,nt_nm,dt_nm)

        !espacial
        datos_dx(k)=dx
        datos_nx(k)=nx

        !leapfrog
        datos_dt(1,k)=dt_lf
        datos_nt(1,k)=nt_lf

        !newman
        datos_dt(2,k)=dt_nm
        datos_nt(2,k)=nt_nm

        !parametros
        lambda_lf       = dt_lf/dx
        lambda_nm       = dt_nm/dx
        gama    = sigma/(rho_w*H)

        allocate(x(k)%kk_(nx+1),t_lf(k)%kk_(nt_lf+1),&
                &t_nm(k)%kk_(nt_nm+1),y_lf(k)%kk(nt_lf+1,nx+1),&
                &y_nm(k)%kk(nt_nm+1,nx+1),dy_nm(k)%kk(nt_nm+1,nx+1),&
                &y_exacta_lf(k)%kk(nt_lf+1,nx+1),&
                &y_exacta_nm(k)%kk(nt_nm+1,nx+1))

        !discretizacion espacio-tiempo
        x(k)%kk_(:) = (/ (dx*(i-1),i=1,nx+1) /)
        t_lf(k)%kk_(:) = (/ (dt_lf*(i-1),i=1,nt_lf+1) /)
        t_nm(k)%kk_(:) = (/ (dt_nm*(i-1),i=1,nt_nm+1) /)

        !condicion inicial esquema newmark
        y_nm(k)%kk(1,:) = (/ (sin(pi*x(k)%kk_(i)),i=1,nx+1) /)
```

```
dy_nm(k)%kk(1,:)= (/ (-sin(pi*x(k)%kk_(i)),i=1,nx+1) /)

!condiciones iniciales esquema leapfrog
y_lf(k)%kk(1,:) = (/ (sin(pi*x(k)%kk_(i)),i=1,nx+1) /)
y_lf(k)%kk(2,:) = y_lf(k)%kk(1,:) + &
        &dt_lf*(/ (-sin(pi*x(k)%kk_(i)),i=1,nx+1) /)

!integracion temporal LEAP-FROG
do i=3,nt_lf+1
        call leap_frog(gama,lambda_lf,t_lf(k)%kk_(i),&
                &dt_lf,x(k)%kk_(:),&
                &y_lf(k)%kk(i-2,:),y_lf(k)%kk(i-1,:),&
                        &y_lf(k)%kk(i,:))
end do

!integracion temporal NEWMARK
theta=0.5_8
beta=0.25_8
do i=2,nt_nm+1
        call newmark(gama,lambda_nm,t_nm(k)%kk_(i),&
                &dt_nm,x(k)%kk_(:),theta,beta,&
                &y_nm(k)%kk(i-1,:),y_nm(k)%kk(i,:),&
                &dy_nm(k)%kk(i-1,:),dy_nm(k)%kk(i,:))
end do

!solucion analitica
do i=1,nt_lf+1
        y_exacta_lf(k)%kk(i,:) = (/ (exp(-t_lf(k)%kk_(i))*&
        &sin(pi*x(k)%kk_(j)),j=1,nx+1) /)
end do
do i=1,nt_nm+1
        y_exacta_nm(k)%kk(i,:) = (/ (exp(-t_nm(k)%kk_(i))*&
                &sin(pi*x(k)%kk_(j)),j=1,nx+1) /)
end do

if (k.eq.4) then
        !exportar datos
        open(unit=10, status="unknown", action="write",&
                & file=data_folder//'datos_S3_1.txt')
                write(10,*) 'x',t_nm(k)%kk_(:)
                do i=1,nx+1
                        write(10,*) x(k)%kk_(i),y_exacta_nm(k)%kk(:,i)
                end do
        close(10)
        open(unit=10, status="unknown", action="write",&
                & file=data_folder//'datos_S3_2.txt')
                write(10,*) 'x',t_lf(k)%kk_(:)
                do i=1,nx+1
                        write(10,*) x(k)%kk_(i),y_lf(k)%kk(:,i)
                end do
        close(10)
        open(unit=10, status="unknown", action="write",&
                & file=data_folder//'datos_S3_3.txt')
                write(10,*) 'x',t_nm(k)%kk_(:)
```

```
                        do i=1,nx+1
                                write(10,*) x(k)%kk_(i),y_nm(k)%kk(:,i)
                        end do
                close(10)
        end if
end do

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

!calculo del error
!p(1,:) => error e_0
do i=1,10
        !p_lf(1,i) = maxval( abs( y_lf(1)%kk(i+1,:) - &
                &y_exacta(1)%kk(i+1,:) ) )
        !p_nm(1,i) = maxval( abs( y_nm(1)%kk(i+1,:) - &
                &y_exacta(1)%kk(i+1,:) ) )
        p_lf(1,i) = sqrt(sum( ( y_lf(1)%kk(i+1,:) - &
                &y_exacta_lf(1)%kk(i+1,:) )**2._8 ))/datos_nx(1)
        p_nm(1,i) = sqrt(sum( ( y_nm(1)%kk(i+1,:) - &
                &y_exacta_nm(1)%kk(i+1,:) )**2._8 ))/datos_nx(1)
end do

!p(k,j) convergencia en cada paso de tiempo j para cada malla k
do k=2,4

        q1=datos_nt(1,k)/10
        q2=datos_nt(2,k)/10
        contador1 = 1
        contador2 = 1
        j=1
        jj=1

!LEAP-FROG
do i=1,datos_nt(1,k)
        if (j.eq.q1) then
                !p_lf(k,contador1) = maxval( abs( y_lf(k)%kk(i,:)&
                        & - y_exacta_lf(k)%kk(i,:) ) )
                p_lf(k,contador1) = sqrt(sum( ( y_lf(k)%kk(i,:) -&
                        & y_exacta_lf(k)%kk(i,:) )**2._8 ))/datos_nx(k)
                p_lf(k,contador1) = log(p_lf(1,contador1)/&
                        &p_lf(k,contador1)/log(2._8**dfloat(k-1)))
                tiempo(1,contador1) = t_lf(k)%kk_(i+1)
                contador1 = contador1 + 1
                j = 0
        end if
        j = j + 1
end do

!NEWMARK
do i=1,datos_nt(2,k)
        if (jj.eq.q2) then
                !p_nm(k,contador2) = maxval( abs( &
                        &y_nm(k)%kk(i,:) - y_exacta_nm(k)%kk(i,:) ) )
                p_nm(k,contador2) = sqrt(sum( ( y_nm(k)%kk(i,:) -&
```

```
                                & y_exacta_nm(k)%kk(i,:) )**2._8 ))/datos_nx(k)
                        p_nm(k,contador2) = log(p_nm(1,contador2)/&
                                &p_nm(k,contador2)/log(2._8**dfloat(k-1)))
                        tiempo(2,contador2) = t_nm(k)%kk_(i+1)
                        contador2 = contador2 + 1
                        jj = 0
                end if
                jj= jj + 1
        end do

        end do

        !genera tabla LF
        open(unit=10, status="unknown", action="write",&
        & file=data_folder//'TABLA_CONVERGENCIA_LEAPFROG.txt')
        !'t_j(0)', 'p_LF(1)' , 'p_LF(2)' , 'p_LF(3)'
        do i=1,10
                write(10,'(4F8.4)') tiempo(1,i) , p_lf(2,i) ,&
                        & p_lf(3,i) , p_lf(4,i)
        end do
        close(10)

        !genera tabla NW
        open(unit=10, status="unknown", action="write",&
        & file=data_folder//'TABLA_CONVERGENCIA_NEWMARK.txt')
        ! 't_j(0)', 'p_NM(1)' , 'p_NM(2)' , 'p_NM(3)'
        do i=1,10
                write(10,'(4F8.4)') tiempo(2,i) , p_nm(2,i) ,&
                        & p_nm(3,i) , p_nm(4,i)
        end do
        close(10)

        !exportar tabla al informe
100 format(' ',2(F8.4,A3,F8.4,A3,F8.4,A3,F8.4,A4))
        open(unit=10, status="unknown", action="write",&
        & file=data_folder_informe//'TABLA_CONVERGENCIA.tex')
        !'t_j(0)', 'p_LF(1)' , 'p_LF(2)' , 'p_LF(3)'
        do i=1,10
                write(10,100) tiempo(1,i),'&',p_lf(2,i),'&',&
                        &p_lf(3,i),'&',p_lf(4,i),&
                        &'&', tiempo(2,i),'&',p_nm(2,i),'&',&
                                &p_nm(3,i),'&',p_nm(4,i),'\\'
        end do
        close(10)

        !generar animaciones
        call system('cd gnuplot/ && gnuplot plot_p3s1.txt')

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

end program
```

Subrutina: discretizacion()

```
subroutine discretizacion(k,nx,dx,nt_lf,dt_lf,nt_nm,dt_nm)
```

```
        implicit none
        !entrada
        integer,intent(in)::k
        !salida
        integer,intent(out)::nx,nt_lf,nt_nm
        real(kind=8),intent(out)::dx,dt_lf,dt_nm
!-------------------------------------
        ! espacial
        dx = 1._8 / (2._8**k*10)
        nx = nint(1._8/dx)
        ! tiempo lf
        dt_lf = 0.25_8*dx
        nt_lf = nint(1._8/dt_lf)
        ! tiempo nm
        dt_nm = dx
        nt_nm = nint(1._8/dt_nm)
end subroutine
```

Subrutina: leap_frog()

```
subroutine leap_frog(gama,lambda,t,dt,x,y_0,y_1,y_2)
!---------------------------
        use modulo_pregunta2
        implicit none
        !entrada
        real(kind=8),intent(in) :: gama,lambda,t,dt
        real(kind=8),dimension(:),intent(in) :: x
        !entrada-salida
        real(kind=8),dimension(:),intent(inout) :: y_0,y_1,y_2
        !variables locales
        integer::n,i
        real(kind=8)::alfa
        real(kind=8),dimension(:),allocatable::f
!---------------------------
        n=size(x)
        alfa = (lambda*gama)**2._8
        allocate(f(n))
        f(:) = (/ ( (1._8+pi**2._8*gama**2._8)*exp(-t)*&
                &sin(pi*x(i)) , i=1,n) /)  * dt**2._8
        y_2(1) = 0._8
        y_2(2) = (2._8-2._8*alfa)*y_1(2) + alfa*y_1(3) -&
                & y_0(2) + f(2)
        do i=3,n-2
                y_2(i) = alfa*y_1(i-1) + (2._8-2._8*alfa)*y_1(i)&
                        & + alfa*y_1(i+1) - y_0(i) + f(i)
        end do
        y_2(n-1) = (2._8-2._8*alfa)*y_1(n-1) + alfa*y_1(n-2)&
                & - y_0(n-1) + f(n-1)
        y_2(n) = 0._8
end subroutine
```

Subrutina: newmark()

```
subroutine newmark(gama,lambda,t,dt,x,theta,beta,u_0,u_1,v_0,v_1)
!---------------------------
        use modulo_pregunta2
```

```
        implicit none
        !entrada
        real(kind=8),intent(in) :: gama,lambda,t,dt,theta,beta
        real(kind=8),dimension(:),intent(in) :: x
        !entrada-salida
        real(kind=8),dimension(:),intent(inout) :: u_0,u_1,v_0,v_1
        !variables locales
        integer::n,nn,i,j,k
        integer,dimension(:),allocatable::indx
        real(kind=8)::alfa,tmp,d
        real(kind=8),dimension(:),allocatable::f_u,f_v
        real(kind=8),dimension(:,:),allocatable::M,M_inv,AA,tmp_m
!-------------------------------
        n=size(x)
        nn=n-2
        alfa = lambda*gama**2._8

        allocate(f_u(n),f_v(n))
        f_u(:) = (/ (   beta*(1._8+pi**2._8*gama**2._8)&
                &*exp(-t-dt)*sin(pi*x(i)) &
                &+ (0.5_8-beta)*(1._8+pi**2._8*gama**2._8)&
                        &*exp( -t   )*sin(pi*x(i)) , i=1,n          ) /)
        f_v(:) = (/ (      theta*(1._8+pi**2._8*gama**2._8)
                &*exp(-t-dt)*sin(pi*x(i)) &
                        &+ (1._8-theta)*(1._8+pi**2._8*gama**2._8)&
                                &*exp( -t   )*sin(pi*x(i)) , i=1,n
) /)

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

        !crear matriz M (matriz del lado izquierdo de la ecuacion)
        allocate(M(nn,nn))
        M(:,:) = 0d0
        M(1,1) = 1._8 + 2._8*alfa*beta ; M(1,2) = - beta*alfa
        do i=2,nn-1
                M(i,i-1:i+1) = (/ - beta*alfa , 1._8 + &
                        &2._8*alfa*beta , - beta*alfa /)
        end do
        M(nn,nn-1) = - beta*alfa ; M(nn,nn) = 1._8 +&
                & 2._8*alfa*beta

        !invertir matriz M
        allocate(indx(nn))
        allocate(M_inv(nn,nn))
        call invertir_matriz(M,nn,1d-10,M_inv)

!       tmp_m
        allocate(tmp_m(nn,nn))
        do i=1,nn
                tmp_m(i,:) = (/ (0._8,j=1,nn) /)
        end do
        tmp_m(1,1) = 1._8 - alfa + 2._8*alfa*beta ; &
                &tmp_m(1,2) = 0.5_8*alfa - alfa*beta
        do i=2,nn-1
```

```
                tmp_m(i,i-1:i+1) = (/ 0.5_8*alfa - alfa*beta &
                        &, 1._8 - alfa + 2._8*alfa*beta , 0.5_8*alfa &
                                &- alfa*beta /)
        end do
        tmp_m(nn,nn-1) = 0.5_8*alfa - alfa*beta ;&
                & tmp_m(nn,nn) = 1._8 - alfa + 2._8*alfa*beta

!       AA = M_inv * tmp_m
        allocate(AA(nn,nn))
        do i=1,nn
                do j=1,nn
                        tmp=0._8
                        do k=1,nn
                                tmp = tmp + M_inv(i,k)*tmp_m(k,j)
                        end do
                        AA(i,j) = tmp
                end do
        end do

!       u_(n+1) = AA * u_(n) + M_inv * v(n) + dt * M_inv * f_u
        u_1(1) = 0._8
        do i=1,nn
                tmp = 0._8
                do j=1,nn
                        tmp = tmp + AA(i,j)*u_0(j+1) + M_inv(i,j)*&
                                &v_0(j+1)*dt + M_inv(i,j)*f_u(j+1)*dt**2._8
                end do
                u_1(i+1) = tmp
        end do
        u_1(n) = 0._8

!       v_(n+1) es explicito
!       v_1(1) = v_0(1) + dt*alfa*theta*(-2._8*u_1(1)+u_1(2)) &
!                       &+ dt*alfa*(1._8-theta)*(-2._8*u_0(1)+u_0(2)) &
!                       &+ dt*f_v(1)*dt

!       v_1(1) = (3._8*u_1(1)-4._8*u_1(2)+1._8*u_1(3))/(2._8*dt)

        v_1(1)=0d0
        do i=2,n-1
                v_1(i) = v_0(i) + alfa*theta*(1._8/dt)*&
                        &(u_1(i-1)-2._8*u_1(i)+u_1(i+1)) &
                                &+ (1._8/dt)*alfa*(1._8-theta)*&
                                        &(u_0(i-1)-2._8*u_0(i)+u_0(i+1)) &
                                &+ dt*f_v(i)
        end do
!       v_1(n) = v_0(n) + dt*alfa*theta*(-2._8*u_1(n)+u_1(n-1)) &
!                       &+ dt*alfa*(1._8-theta)*(-2._8*u_0(n)+u_0(n-1)) &
!                       &+ f_v(n)*dt

!       v_1(n) = (3._8*u_1(n)-4._8*u_1(n-1)+1._8*u_1(n-2))/(2._8*dt)

        v_1(n) = 0d0
```

```
end subroutine

subroutine Ludecomp(a,b,n,tol,x)

        integer::n,er
        integer::o(n)
        double precision::a(n,n),b(n),x(n),tol,s(n)

        er=0
        call Decompose(a,n,tol,o,s,er)
        if(er/=-1)then
                call Substitute(a,o,n,b,x)
        endif

end subroutine

subroutine Decompose(a,n,tol,o,s,er)

        integer::n,er
        integer::o(n)
        double precision::a(n,n),tol,s(n)

        do i=1,n
                o(i)=i
                s(i)=abs(a(i,1))
                do j=2,n
                        if(abs(a(i,j))>s(i))then
                                s(i)=abs(a(i,j))
                        endif
                enddo
        enddo
        do k=1,n-1
                call Pivot(a,o,s,n,k)
                if(abs(a(o(k),k)/s(o(k)))<tol)then
                        er=-1
!                       write(*,*)a(o(k),k)/s(o(k))
                        exit
                endif
                do i=k+1,n
                        factor=a(o(i),k)/a(o(k),k)
                        a(o(i),k)=factor
                        do j=k+1,n
                                a(o(i),j)=a(o(i),j)-factor*a(o(k),j)
                        enddo
                enddo
        enddo
        if(abs(a(o(k),k)/s(o(k)))<tol)then
                er=-1
!               write(*,*)a(o(k),k)/s(o(k))
        endif

end subroutine

subroutine Pivot(a,o,s,n,k)
```

```
        integer::n,k,p
        integer::o(n),dummy1
        double precision::a(n,n),s(n),big,dummy2

        p=k
        big=abs(a(o(k),k)/s(o(k)))
        do ii=k+1,n
                dummy2=abs(a(o(ii),k)/s(o(ii)))
                if(dummy>big)then
                        big=dummy2
                        p=ii
                endif
        enddo
        dummy1=o(p)
        o(p)=o(k)
        o(k)=dummy1

end subroutine

subroutine Substitute(a,o,n,b,x)

        integer::n
        integer::o(n)
        double precision::a(n,n),b(n),x(n),suma

        do i=2,n
                suma=b(o(i))
                do j=1,i-1
                        suma=suma-a(o(i),j)*b(o(j))
                enddo
                b(o(i))=suma
        enddo
        x(n)=b(o(n))/a(o(n),n)
        do i=n-1,1,-1
                suma=0d0
                do j=i+1,n
                        suma=suma+a(o(i),j)*x(j)
                enddo
                x(i)=(b(o(i))-suma)/a(o(i),i)
        enddo

end subroutine

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine invertir_matriz(a,n,tol,ai)

integer::n,er=0
integer::o(n)
double precision::a(n,n),s(n),tol,b(n),x(n),ai(n,n)

        call Decompose(a,n,tol,o,s,er)
        if(er==0)then
```

```
            do i=1,n
                    do j=1,n
                            if(i==j)then
                                    b(j)=1.
                            else
                                    b(j)=0.
                            endif
                    enddo
                    call Substitute(a,o,n,b,x)
                    do j=1,n
                            ai(j,i)=x(j)
                    enddo
            enddo
        !write(*,*)a
        else
        !write(*,*)'sistema mal condicionado'
        endif

end subroutine
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Atractor de Lorenz

```
program Pregunta_3
        !-------------------------------
        implicit none
        integer::n,m,i,j
        real(kind=8) :: tiempo_total,dt,Pr,beta,t
        real(kind=8),dimension(3) :: z0,z1,Ra
        real(kind=8),dimension(11)::Ra_2
        character(len=8)::folder='./datos/'
        character(len=2),dimension(3) :: ra_n
        character(len=2),dimension(11) :: ra_n2
        !-------------------------------

        !discretizacion tiempo
        tiempo_total = 6d1
        t       = 0._8
        dt      = 0.01_8
        m       = nint(tiempo_total/dt)

        !parametros
        Pr      = 10._8
        Ra      = (/ 0.5_8 , 10._8 , 28._8 /)
        beta = 8._8/3._8

        !resolucion atractor de lorenz
        ra_n(1) = '05' ; ra_n(2) = '10' ; ra_n(3) = '28'
        do j=1,3
                open(unit=10, file=folder//'datos_P3_'//&
                        &ra_n(j)//'.dat', action='write')
                !inicializacion
                n = 0
                z0(:) = (/ 0._8 , 1._8 , 0._8 /)
                write(10,*) n , t , 10._8*z0(1) , 10._8*z0(2) , 10._8*z0(3)
```

```
            !subrutina RK4 -> Ra = 0.5
            do i=1,m
                    call rk4(Pr,Ra(j),beta,t,dt,z0,z1)
                    z0 = z1
                    n = n+1
                    t = t+dt
                    if ( mod(n,5) .eq. 0 ) then
                            write(10,*) n , t , 10._8*z0(1) , &
                                    &10._8*z0(2) , 10._8*z0(3)
                    end if
            end do
            close(unit=10)
    end do

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

    !variar lentamente Ra
    Ra_2(:) = (/ (3.0*(j-1) , j=1,11) /)

    ra_n2(1) = '00'
    ra_n2(2) = '03'
    ra_n2(3) = '06'
    ra_n2(4) = '09'
    ra_n2(5) = '12'
    ra_n2(6) = '15'
    ra_n2(7) = '18'
    ra_n2(8) = '21'
    ra_n2(9) = '24'
    ra_n2(10) = '27'
    ra_n2(11) = '30'

    do j=1,11
            open(unit=10, file=folder//'datos_P3_'//&
                    &ra_n2(j)//'.dat', action='write')
            !inicializacion
            n = 0
            z0(:) = (/ 0._8 , 1._8 , 0._8 /)
            write(10,*) n , t , 10._8*z0(1) , 10._8*z0(2)&
                    & , 10._8*z0(3)
            !subrutina RK4 -> Ra = 0.5
            do i=1,m
                    call rk4(Pr,Ra(j),beta,t,dt,z0,z1)
                    z0 = z1
                    n = n+1
                    t = t+dt
                    if ( mod(n,5) .eq. 0 ) then
                            write(10,*) n , t , 10._8*z0(1) ,&
                                    & 10._8*z0(2) , 10._8*z0(3)
                    end if
            end do
            close(unit=10)
    end do

    call system('cd gnuplot/ && gnuplot plot_P3.txt')
```

```
end program
```

Subrutina: rk4()

```
subroutine rk4(Pr,Ra,beta,t0,dt,y0,y4)
        !-------------------------------
        ! esta subrutina implementa el método de
        ! integracion RK4 vectorial (dimension 3)
        ! de la forma dy/dt = g(t,y)
        !-------------------------------
        implicit none
        !entrada
        real(kind=8),intent(in) :: t0,dt,Pr,Ra,beta
        real(kind=8),dimension(3),intent(in)    :: y0
        !salida
        real(kind=8),dimension(3),intent(out)   :: y4
        !local
        real(kind=8)                            :: t1,t2
        real(kind=8),dimension(3)       :: y1,y2,y3
        !-------------------------------
        !tiempo
        t1 = t0 + 0.5_8*dt      !medio paso
        t2 = t0 + dt            !un paso
        !pasos rk4
        y1 = y0 + 0.5_8*dt * g(Pr,Ra,beta,t0,y0)
        y2 = y0 + 0.5_8*dt * g(Pr,Ra,beta,t1,y1)
        y3 = y0 + dt * g(Pr,Ra,beta,t1,y2)
        y4 = y0 +  (dt/6._8) * &
                & ( g(Pr,Ra,beta,t0,y0) + &
                & 2._8*g(Pr,Ra,beta,t1,y1) + &
                & 2._8*g(Pr,Ra,beta,t1,y2) + &
                & g(Pr,Ra,beta,t2,y3) )
        !-------------------------------
        contains
                function g(Pr,Ra,beta,t,z) result(v)
                ! la funcion g representa el lado derecho
                ! de la ecuacion (rhs) de lorenz
                        real(kind=8),intent(in)::Pr,Ra,beta,t,z(3)
                        real(kind=8)::v(3)
                        v(1) = Pr*(z(2)-z(1))
                        v(2) = Ra*z(1)-z(2)-z(1)*z(3)
                        v(3) = z(1)*z(2)-beta*z(3)
                end function g
        !-------------------------------
end subroutine
```

Referencias

[1] Lorenz, E. , *Deterministic Nonperiodic Flow*, Journal of the Atmospheric Science, vol. 20, pag. 130-141, 1963

[2] Chapman, S. , *Fortran 90/95 for Scientits and Engineers*, Ediforial Mcgraw-Hill, 2003, ISBN-13: 978-0072922387

[3] Quarteroni, A. , Sacco, R. y Saleri, F. , *Numerical Mathematics*, Editorial Springer, 2000, ISBN 0-387-98959-5

[4] Chapra, S. y Canale, R. , *Métodos numéricos para ingenieros*, 5ta edición, Editorial McGraw-Hill, 2007, ISBN-13: 978-970-10-6114-5

[5] Moawwad, E. , Abdelrahman, K. , *Inversion of general tridiagonal matrices*, Applied Mathematics Letters 19, pag. 712-720. 2006