Integración Numérica de Ecuaciones Diferenciales **Código:A_IntNum**

1. **Introducción:** [1]

Los métodos numéricos para el estudio del comportamiento de sistemas dinámicos han cobrado fuerza en los últimos años por varias razones. Probablemente la más importante, es que puede accederse a computadoras altamente eficientes a un costo cada vez más bajo, lo que permite su uso para la resolución de problemas áltamente complejos.

Una segunda razón es que los métodos numéricos son en muchos casos la única alternativa posible para la resolución de los frecuentes problemas no-lineales muchas veces intratables analíticamente. Por otra parte los problemas lineales continúan creciendo en magnitud, requiriendo un mayor esfuerzo para su solución.

Discutiremos aquí la solución de ecuaciones diferenciales ordinarias (EDO's) o ecuaciones de estado (EE) a través de la aplicación de métodos numéricos.

1.1. **Formulación del Problema - Existencia y Unicidad de la Solución** [2]

Estamos interesados en resolver problemas de la forma:

$$\frac{dX(t)}{dt} = \dot{X}(t) = f^1[X(t), u(t), t] \qquad (1)$$
$$X(t_0) = X_0$$

donde $X(t) \in \Re^n$ (vector de estado)
$f^1 = [f^1_1, f^1_2, ..., f^1_n]^T$ (función vectorial de n componentes)
t: variable absoluta tiempo
X_0 : estado inicial
$u(t) \in \Re^m$ (vector de entrada)
t_0 : tiempo inicial

Dado que para la resolución del problema (1) $u(t)$ es un dato, podemos escribir directamente:

$$f^1[X(t), u(t), t] = f[X(t), t]$$

Observar que en esta forma se incluyen también sistemas no estacionarios (sistemas cuyos parámetros son variables en el tiempo).

El problema (1), en el cual se tiene una ecuación diferencial y se conoce el valor inicial de la solución, se denomina "problema de Cauchy" o "problema del valor inicial". Lo que se busca entonces es una solución $f=f(t)$, que satisfaga la condición inicial $f(t_0)=X_0$.

La existencia y unicidad de soluciones locales, es decir en un entorno U de X_0 y t_0

$$U = \{t, x_1, ..., x_n / |t - t_0| < a, |x_1 - x_{10}| < b_1, ..., |x_n - x_{n0}| < b_n\},$$

es garantizada por las siguientes hipótesis:

H1) Las funciones $f_i(x_1, ..., x_n, t)$ (componentes de $f[X(t), t]$)son funciones continuas y definidas en U.
H2) Las funciones $f_i(x_1, ..., x_n, t)$ satisfacen en U las condiciones de Lipschitz:

$$| f_i(\tilde{x}_1,...,\tilde{x}_n,t) - f_i(x_1,...,x_n,t) | \leq \sum_{j=1}^{n} M \mid \tilde{x}_j - x_j \mid$$

con $(x_1, x_2,.....,x_n,t) \in U$ y $(\tilde{x}_1, \tilde{x}_2,.....,\tilde{x}_n,t) \in U$ y M es alguna constante (constante de Lipschitz).

Bajo estas condiciones se demuestra que existe solución única en un intervalo $|t - t_0| < a$, y está dada por las componentes

$f_1 = f_1(t),....,f_n = f_n(t)$, que satisfacen las condiciones iniciales:

$f_1(t_0) = x_{10},....,f_n(t_0) = x_{n0}$

Una condición suficiente para que se cumplan las condiciones de Lipschitz es que las funciones f_i tengan en el intervalo U derivadas parciales acotadas:

$\frac{\partial f_i}{\partial x_j}$, con $i, j=1,.....,n$

ya que en ese caso puede tomarse:

$M = \max_{i,j} \left| \frac{\partial f_i}{\partial x_j} \right|$ en U.

En los problemas lineales y estacionarios siempre es posible encontrar una expresión analítica para la solución de (1). Desafortunadamente no ocurre lo mismo en el caso no-lineal, donde la mayor parte de las veces es imposible hallar la solución de (1) por métodos analíticos y debe recurrirse a métodos numéricos.

Sin embargo, aún en el caso lineal, es de interés disponer de métodos que permitan computar de manera rápida y eficiente la solución de (1), principalmente en problemas de gran magnitud.

2. **Representación numérica** [9]

Consideremos por simplicidad el caso escalar del problema (1). Tenemos entonces:

$$\frac{dx(t)}{dt} = f[x(t),t] \qquad (2)$$
$$x(t_0) = x_0$$

Bajo un enfoque "numérico" para la solución del problema (2), acordamos representar la variable continua tiempo t a través de una secuencia de puntos discretos $t_k, k = 1,2,....,n$. Estos puntos se encuentran usualmente espaciados a intervalos iguales *h*, con lo cual se tiene:

$$t_k = t_0 + k \cdot h$$

Notamos además:

$$f(t_0 + k \cdot h) = f(t_k) = f_k$$

Un segundo elemento en la representación numérica consistirá en caracterizar la solución $f(t)$ del problema (2) a través de un conjunto de puntos (f_k, t_k).

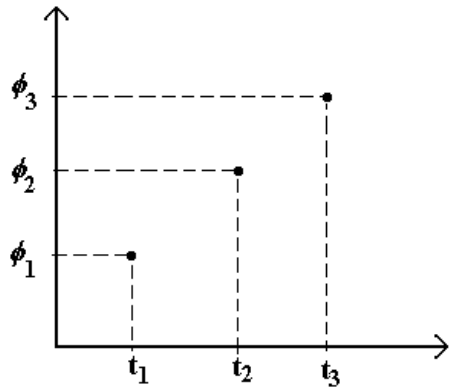

Fig. 1: Representación numérica de la solución.

Lo que se busca es obtener una sucesión de valores x_k como una buena aproximación a la solución del problema continuo, es decir al valor f_k. (Observar que denominamos f_k al valor que la solución del problema contínuo toma en el instante t_k mientras que x_k es el valor estimado numéricamente).

Debe observarse también que aún cuando se haya encontrado exactamente f_k, los valores intermedios de $f(t)$ continúan siendo desconocidos. De todos modos si se eligió h para obtener valores "razonablemente exactos" de x_k, probablemente se tendrán suficientes puntos para estimar $f(t)$ muy fácilmente (interpolando entre los x_k calculados).

2.1. **Método de Euler** [2], [7], [8]

Existe una gran cantidad de métodos numéricos para la solución aproximada del problema (1). La elección de un algoritmo en particular está determinada por varios factores, entre ellos los requerimientos de precisión, de complejidad informática, de velocidad de procesamiento, etc.

El método conocido como "Método de integración de Euler" es sin dudas el más simple de todos los existentes.

Recordando el problema (1), se tiene:

$$\dot{X} = f(X,t)$$
$$X(t_0) = X_0$$

donde X es un vector de *n* componentes, y X_0 es el vector de condiciones iniciales.

Se plantea símplemente aproximar la derivada por el cociente incremental, luego

$$X(t_k + \Delta t) = X(t_k) + f(X(t_k), t_k) \cdot \Delta t \Rightarrow$$
$$X_{k+1} = X_k + f_k \cdot h$$

donde $\Delta t = h$ recibe el nombre de paso de integración.

Así, conociendo la condición inicial X_0 pueden calcularse de manera sucesiva $X_1, X_2, ...$

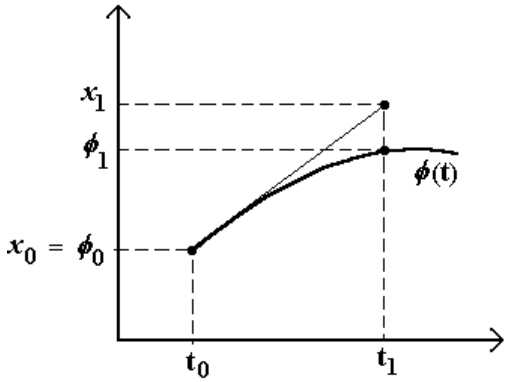

Fig. 2: Interpretación gráfica del método de Euler (caso escalar).

Como veremos luego, del desarrollo en serie de Taylor se deduce que el error por truncamiento (ver sección 3) es del orden de Δt^2, lo cual es relativamente grande y obliga generalmente a utilizar un paso de integración muy pequeño en detrimento de la velocidad y, como se verá mas adelante, aumentando los errores por redondeo. No obstante estas desventajas el método es ampliamente utilizado debido a su escasa complejidad computacional.

Ejemplo:

Consideremos el sistema de primer orden:

$$\dot{x} = -3x^3 + t$$
$$x(0) = 1$$

Al aplicar Euler tenemos la siguiente fórmula:

$x_{k+1} = x_k + (-3x_k^3 + t_k) \cdot h$, de donde

$x_1 = 1 - 3h \quad x_2 = x_1 + (-3x_1^3 + h) \cdot h \quad x_3 = x_2 + (-3x_2^3 + 2h) \cdot h$, y así sucesivamente.

2.2. **Series de Taylor** [9]

Sea $f(t)$ la solución del problema (2) (caso escalar). Consideremos entonces el desarrollo en serie de Taylor de la misma en torno a t_k :

$$f(t_k + h) = f_{k+1} = f_k + h\frac{df}{dt}\bigg|_{t_k} + \frac{h^2}{2!}\frac{d^2f}{dt^2}\bigg|_{t_k} + ...+ \frac{h^p}{p!}\frac{d^pf}{dt^p}\bigg|_{t_k} + ...$$

Al ser $f(t)$ solución del problema, resulta $f[f(t),t] = \frac{df(t)}{dt}$, de donde podemos plantear la siguiente aproximación numérica:

$$x_{k+1} = x_k + hf(x_k,t_k) + \frac{h^2}{2!}\frac{df}{dt}\bigg|_{x_k,t_k} + ...+ \frac{h^p}{p!}\frac{d^{p-1}f}{dt^{p-1}}\bigg|_{x_k,t_k} \quad (3)$$

que coincide con la serie anterior hasta el término de h^p. El error por lo tanto puede escribirse como una serie, cuyo primer término tiene un factor h^{p+1}. Este error es el denominado "error por truncamiento", y en el caso presentado se dice que es del orden $p+1$ (o sea, del exponente al cual se encuentra elevado h en el primer término de la serie del error).

Esta denominación se debe al hecho que según el teorema del resto [6], bajo ciertas condiciones, el error al considerar solamente los términos de la serie hasta el orden de h^p está acotado por un factor del orden de h^{p+1}.

Como vemos, el método de Euler es un caso particular de la aproximación presentada en (3), haciendo $p=1$, por lo que el error, como fue mencionado anteriormente, es del orden de h^2.

Lamentablemente la fórmula dada por (3) no es muy útil para plantear métodos de mayor orden, ya que requiere del conocimiento de las derivadas de f, que en un caso general a implementar sobre un programa no se tiene; y por lo tanto implicará o bien algún tipo de cálculo simbólico (con la consiguiente complejidad computacional) o alguna aproximación numérica (con la introducción de un nuevo error).

2.3. **Método de Runge-Kutta** [2]

Otro método de gran uso en la integración numérica de ecuaciones diferenciales es el denominado método de Runge-Kutta. Éste conserva la importante ventaja de requerir sólo un valor previo en el proceso de integración, dando una mejora sustancial en el error cometido; dependiendo ello del orden específico del procedimiento de Runge-Kutta elegido.

Desarrollaremos aquí el procedimiento de cuarto orden, en el cual el error cometido resultará del orden de h^5.

Se calcula:

$$X_{k+1} = X_k + a \cdot S_1^{(k)} + b \cdot S_2^{(k)} + c \cdot S_3^{(k)} + d \cdot S_4^{(k)} \quad (4)$$

Donde:

$$S_1^{(k)} = h \cdot f(X_k, t_k)$$
$$S_2^{(k)} = h \cdot f(X_k + \frac{S_1^{(k)}}{2}, t_k + \frac{h}{2})$$
$$S_3^{(k)} = h \cdot f(X_k + \frac{S_2^{(k)}}{2}, t_k + \frac{h}{2})$$
$$S_4^{(k)} = h \cdot f(X_k + S_3^{(k)}, t_k + h)$$

Si se desarrolla en serie de Taylor la expresión (4), y se la compara con la expresión exacta en series de Taylor de f_{k+1} es posible ajustar los 4 parámetros (α, β, χ, δ) de tal forma que coincidan los cuatro primeros términos de ambas series.

A cálculo hecho resulta:

α=1/6, β=1/3, χ=1/3, δ=1/6.

Como mencionamos antes, este es el método para orden 4, que es el más utilizado en la práctica (dentro de los métodos de paso fijo). Sin embargo, el procedimiento mostrado puede aplicarse para obtener algoritmos de mayor orden.

Ejemplo:

Para el mismo ejemplo que antes:

$$\dot{x} = -3x^3 + t$$
$$x(0) = 1$$

tomando *h*=0,1, resulta:

$$S_1^{(0)} = 0{,}1 \cdot (-3) = -0{,}3$$
$$S_2^{(0)} = 0{,}1 \cdot [-3(1-0{,}15)^3 + 0{,}05] = -0{,}1792$$
$$S_3^{(0)} = 0{,}1 \cdot [-3(1-0{,}0896)^3 + 0{,}05] = -0{,}2214$$
$$S_4^{(0)} = 0{,}1 \cdot [-3(1-0{,}2214)^3 + 0{,}1] = -0{,}1316$$

Luego resulta:

$$x_1 = 1 + \frac{1}{6}(-0{,}3 - 2 \cdot 0{,}1792 - 2 \cdot 0{,}2214 - 0{,}1316) = 0{,}7945$$

y de idéntica forma se calculan los siguientes valores.

2.4. **Método de Adams-Bashforth-Moulton** [7]

Los métodos vistos hasta acá son conocidos como métodos de un solo paso, ya que requieren de un único juego de condiciones iniciales (esto es, las condiciones iniciales en un solo instante). El método de Adams es un método de múltiples pasos ya que requiere tantos juegos de condiciones iniciales como orden tenga el método. Por lo tanto, para implementarlo se necesita primero realizar algunos pasos con otro método (por ejemplo el de Euler o el de Runge-Kutta) de manera de tener el número sufuciente de juegos de condiciones iniciales. La fundamentación del método de Adams se basa en la segunda fórmula de interpolación de Newton [2].

El más utilizado de los métodos de Adams es el de Adams-Bashforth-Moulton de cuarto orden, que constituye también el más popular de los métodos de "Predictor-Corrector". En este método, se calcula:

$$X(t_k + h) = X_{k+1} = X_k + \frac{h}{24}(55 f_k - 59 f_{k-1} + 37 f_{k-2} - 9 f_{k-3}) \quad \text{(predictor)}$$

Luego se recalcula:

$$X_{k+1} = X_k + \frac{h}{24}(9 f_{k+1} + 19 f_k - 5 f_{k-1} + f_{k-2})$$ (corrector)

Observar que para este último cálculo es necesario utilizar el valor calculado por el predictor, ya que f_{k+1} depende de X_{k+1} y si no fuera por el primer paso (predictor), el cálculo del corrector hubiera resultado en una ecuación implícita.

Esta característica "implícita" le brinda al método ventajas en lo concerniente a estabilidad (como veremos más adelante). Se puede (aunque no es conveniente desde el punto de vista de optimización del tiempo de cómputo) aplicar en forma iterativa el paso "corrector" hasta que X_{k+1} converja a un valor. Esta convergencia implicaría que se ha resuelto la ecuación implícita y por lo tanto, el método se convierte verdaderamente en un método implícito. Sin embargo, el tiempo utilizado en conseguir esta convergencia es más útil emplearlo en realizar una simulación con un paso de integración más pequeño.

En muchos casos la diferencia entre el resultado del corrector y del predictor es utilizada como medida del error cometido con el fin de ajustar el paso de integración.

3. Errores en las aproximaciones numéricas [3]

3.1. Errores de método (truncamiento)

Se empleará el término "error por truncamiento" para indicar errores causados por las limitaciones inherentes a los algoritmos de integración (la denominación se debe a que la causa del error puede interpretarse como debida al truncamiento de la serie de Taylor utilizada para el cálculo).

Estos errores podrían surgir independientemente de la precisión del sistema de cómputo empleado en la implementación del algoritmo.

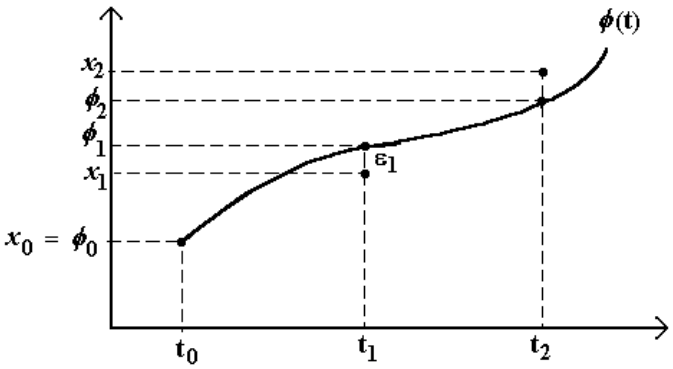

Fig.3: Error por truncamiento

Un dado método de integración producirá un error por truncamiento en cada paso de integración (considerando precisión infinita del sistema de cómputo) $e_k = x_k - f_k$, para una integración que comenzó en t_{k-1}. Es importante notar que este es el error cometido en un solo paso de integración, es decir bajo el supuesto que se conoce f_{k-1} con exactitud.

Naturalmente, la solución del problema continuo es desconocida y en general el error puede acotarse pero no conocerse con exactitud (conocer el error con exactitud implicaría conocer la solución exacta!)

Una medida más importante del error cometido la constituye el denominado error total e_{kT}, evaluado desde el comienzo de la simulación hasta el instante t_k (es decir, suponiendo que se conoce exactamente $f_0 = x_0$ pero no se conoce con exactitud f_{k-1}, que es lo que ocurre en la realidad). Normalmente resulta $|e_{kT}| > |e_k|$ ya que se produce una acumulación de error en los pasos precedentes. Mas aún, en general $|e_{kT}|$ no puede estimarse y grandes esfuerzos por mantener $|e_k|$ dentro de límites aceptables podrían resultar en valores inadmisibles de $|e_{kT}|$.

Usualmente el error por truncamiento en cada paso de integración puede reducirse disminuyendo el valor del paso de integración h y se establece generalmente que $|e_k| \sim h^m$ donde m es un exponente que depende del método específico de integración que se está empleando. Métodos de integración de orden elevado poseen valores altos de m permitiendo así aumentar el valor de h.

3.2. Errores inherentes al sistema de cómputo (redondeo) [4]

3.2.1. Representación de cantidades en Punto Flotante (PF)

Los números son usualmente representados en las computadoras bajo la notación de punto flotante, la cual puede ser generalizada de la siguiente manera:

$x \cong db^e$, donde

x : cantidad representada

b : base de la representación en PF.

e : exponente entero; $e = e_q, e_{q-1}, ..., e_1$

$d : -1 < d < 1, \quad d = d_1, d_2, ..., d_t$, donde t indica el número de caracteres disponibles para representar la parte fraccional de x (mantisa).

Así, una representación típica de un número en PF en una computadora es de la forma:

$s, e_q .....e_1, d_1 .....d_t$

donde s es el signo del exponente del número que se está representando. Debe observarse que la cantidad de dígitos disponibles para representar la parte fraccional de x (d), como así también el exponente (e) son finitas. Por esta razón, la aritmética implementada por la computadora (es decir las operaciones de suma, resta, multiplicación y división) se efectúa en general en presencia de redondeo.

Debido a las características de este tipo de representación, el error de redondeo introducido al realizar una operación depende de las magnitudes de los números involucrados en dicha operación; siendo particularmente grande en el caso de sumar números de órdenes de magnitud muy disímiles, o bien al restar números muy parecidos.

3.2.2. **Integración numérica: errores por redondeo**

Supongamos que queremos implementar sobre un programa que trabaja en punto flotante el algoritmo de Euler. La fórmula con que se calcula cada paso es la siguiente:

$$x_{k+1} = x_k + h \cdot f(x_k, t_k) .$$

Si se quiere hacer hacer h muy pequeño (lo que sabemos, mejoraría la precisión del algoritmo), aumentará el error de redondeo, ya que estaremos sumando dos números de magnitudes muy disímiles y el redondeo hará que el resultado sea prácticamente igual al primer número (grande) ya que utilizará en la operación sólo los dígitos mas significativos del segundo.

3.3. **Errores por truncamiento vs. errores por redondeo**

Vimos que los errores de truncamiento pueden en general reducirse con la reducción del paso de integración, mientras que, por otro lado, la reducción excesiva del paso producirá un error de redondeo inadmisible. Esto nos muestra que en general habrá que encontrar una solución de compromiso entre ambos errores. Para ilustrar esta idea, presentaremos un nuevo ejemplo, tomado de [3] con los correspondientes resultados de simulación:

Consideremos la ecuación diferencial lineal:

$$\ddot{x} + 2x \cdot \dot{x} + x = 0$$
$$x(0) = 100, \quad \dot{x}(0) = 0$$

Utilizaremos el método de Runge-Kutta de 2° orden en un caso y el de 4° orden en el otro. Se resolvieron las ecuaciones para varios valores del paso de integración h, desde 0,001 a 1 seg y para dos valores de la constante de amortiguamiento, ξ=0 y ξ=0.5. Todas las corridas se realizaron en el rango de $0 \le t \le 13$ seg. El error cometido en la estimación de f se encontró por comparación con la solución analítica de la EDO:

$$f(t) = A \cdot e^{-\xi w_n t} \cos(w_d t + q)$$

$$A = \frac{100}{\cos(q)}, \quad q = \text{arctg}\left(\frac{x}{w_d}\right), \quad w_d = w_n\sqrt{1-x^2}$$

La tabla siguiente resume los resultados obtenidos. La integración se implementó utilizando una aritmética de punto flotante de 8 decimales con redondeo.

	h	Runge-Kutta de 2° orden		Runge-Kutta de 4° orden	
	Paso de integración (seg)	Max. Error	Error en $t = 12.6$	Max. Error	Error en $t = 12.6$
ξ=0	0.001	0.00798	0.00797	0.00797	0.00729
	0.01	0.01832	0.00134	0.00078	0.00078
	0.02	0.07364	0.00197	0.00038	0.00037
	0.05	0.46138	0.00050	0.00021	0.00015
	0.10	1.84962	0.06498	0.00104	0.00013
	0.20	7.46967	0.63649	0.01501	0.00225
	0.50	48.52581	11.43471	0.59470	0.30296
	1.00	315.42655	-	9.48049	-
ξ=0.5	0.001	0.00105	0.00003	0.00104	0.00003
	0.01	0.00115	0.00003	0.00010	0.00000
	0.02	0.00451	0.00011	0.00005	0.00000
	0.05	0.02869	0.00068	0.00001	0.00000
	0.10	0.11838	0.00285	0.00003	0.00000
	0.20	0.50435	0.01228	0.00107	0.00001
	0.50	3.59918	0.08533	0.01645	0.00001
	1.00	15.97000	0.28289	0.79498	0.03274

Puede observarse que existe un valor óptimo del paso de integración que minimiza los efectos combinados de los errores por truncamiento y por redondeo. Este valor depende del método específico utilizado y de la representación numérica utilizada para la implementación.

4. **Estabilidad de las soluciones numéricas**

La aplicación de un método numérico transforma el sistema de ecuaciones diferenciales en un sistema de ecuaciones en diferencia (sistema discreto). La solución de este nuevo sistema puede ser inestable, aún cuando el sistema de ecuaciones diferenciales original fuese totalmente estable.

Consideremos un sistema lineal, de primer orden, al que queremos aproximar utilizando el método de Euler:
$\dot{x} = -lx$
Como sabemos, para valores de λ positivos, este sistema es estable.

Usando el algoritmo de Euler, resulta:

$x_{k+1} = x_k - hlx_k = (1-hl)x_k$, resultando de la aplicación recursiva de esta fórmula:

$x_k = (1-hl)^k x_0$.

Si $hl > 2$ es muy simple ver que la solución encontrada diverge, ya que no existe $\underset{k\to\infty}{Lím}\, x_k$.

Esto nos dice que la condición de estabilidad del método es $h < \frac{2}{l}$, es decir, hay una cota superior del valor de h para que el método sea estable.

En caso de que el sistema sea de mayor orden (pero aún lineal), la condición para que la discretización por el método de Euler sea estable es la misma, sólo que utilizando la menor de las constantes de tiempo.

Esto tiene como consecuencia que en sistemas con constantes de tiempo muy diferentes (sistemas Stiff) el número total de pasos al utilizar este método sea desmesurado, ya que al tener una constante de tiempo muy pequeña el paso de integración debe ser muy pequeño; y al querer simular toda la respuesta en presencia de una constante de tiempo muy grande, el tiempo total es muy grande.

La utilización de algoritmos de mayor orden (como por ejemplo Runge-Kutta de 4° orden) permite la utilización de un paso de integración algo mayor sin que el sistema discreto se torne inestable [3]. Sin embargo, esta mejora no es realmente significativa en casos como el mencionado en el párrafo anterior.

Una posibilidad para salvar estos inconvenientes es la utilización de métodos de paso variable, tales como el método Runge Kutta 4-5. Otra posibilidad es la utilización de los llamados métodos implícitos. Más adelante veremos en detalle estos métodos.

5. **Sistemas Rígidos (Stiff Systems)**

Diremos que un sistema es rígido cuando en el mismo se encuentran presentes dinámicas rápidas y lentas. Puede darse el caso que ambas dinámicas estén presentes en forma simultánea o bien que en ciertos instantes de tiempo actúe la parte rápida y en ciertos instantes de tiempo la parte lenta. El motivo por el que se estudia en particular este tipo de sistemas es por los inconvenientes que presentan a la hora de la simulación, además de la frecuencia de aparición que tienen en los problemas relacionados con la técnica. Ejemplos comunes de este tipo de sistemas se encuentran muy frecuentemente en los sistemas electromecánicos, donde coexisten una dinámica rápida asociada a la parte eléctrica y una dinámica lenta asociada a la parte mecánica.

Consideremos el sistema descripto por la siguiente ecuación diferencial:

$$\ddot{x} + 1001 \cdot \dot{x} + 1000 \cdot x = 0$$
$$x(0) = 1000 \quad \dot{x}(0) = 0$$

Utilizando algún método analítico, es simple ver que la solución para $t>0$ es de la forma:

$$k_1 \cdot e^{-t} + k_2 \cdot e^{-1000t}$$

La gráfica de la respuesta es la siguiente:

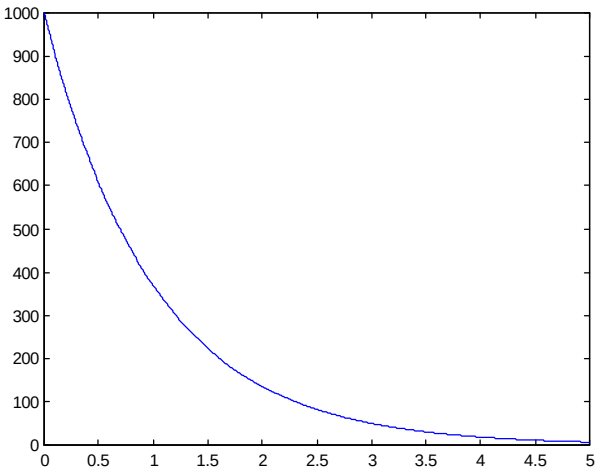

Fig. 4: Respuesta a condiciones iniciales de un sistema rígido de 2do orden

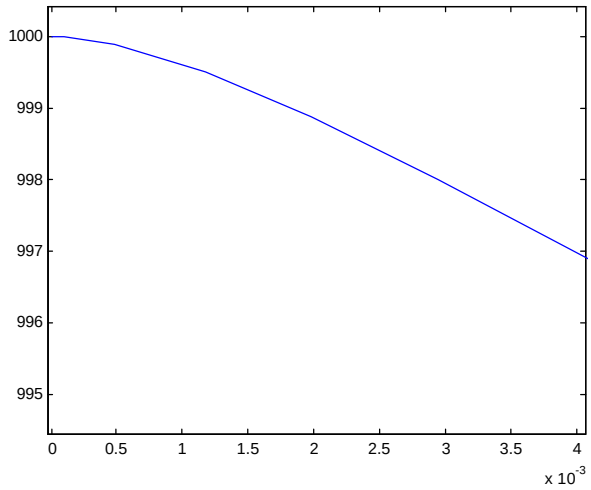

Fig. 5: Ampliación de los primeros 4 ms de la respuesta.

Recordando la condición de estabilidad del algoritmo de Euler, es necesario que el paso de integración sea del orden de 1ms. (para tener cierta precisión, es aconsejable que sea aún menor, digamos 0,1ms) Además, el tiempo total de simulación, si queremos ver la respuesta hasta que el sistema alcanza su valor final, debe ser mayor a 5 segundos.

De esta forma, el número total de pasos es al menos 50000, lo que sin dudas es algo excesivo para un sistema tan simple. Como ya mencionamos antes, la solución a este inconveniente es la utilización de algoritmos de paso variables y/o métodos implícitos, que veremos más adelante.

Consideremos ahora el siguiente sistema:

$$\ddot{x} + \dot{x} + 10001 \cdot x = 0$$
$$x(0) = 1 \quad \dot{x}(0) = 0$$

Nuevamente, calculando de manera analítica, resulta:

$$x(t) = e^{-t} \cos(100 \cdot t) + 0{,}01 \cdot e^{-t} \operatorname{sen}(100 \cdot t)$$

donde la presencia de senoidales se debe a la presencia de raíces complejas en la ecuación característica.

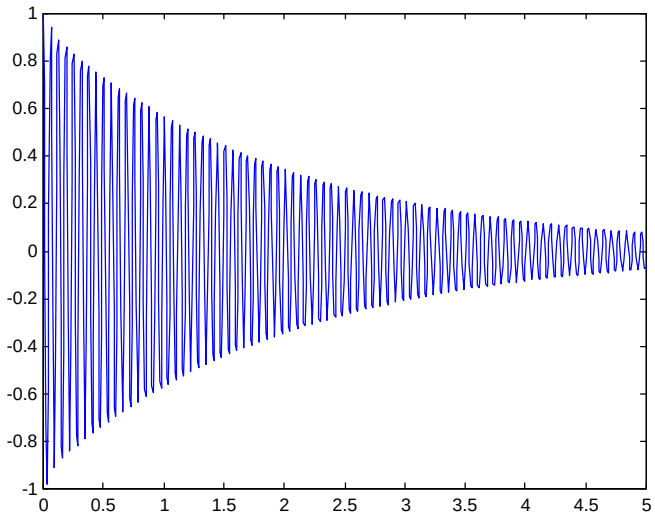

Fig. 6: Respuesta a condiciones iniciales de un sistema rígido con oscilaciones de 2do orden

Nuevamente, si se quiere simular este sistema, nos encontramos con que el tiempo final de simulación es de al menos 5 segundos, mientras que si se quiere reproducir de forma aceptable las senoidales presentes el paso de integración debe ser del orden de 1 ms, lo que nos da un número bastante grande de cálculos. Este número podría ser mucho mayor si la frecuencia de la senoide fuera mayor. En este caso, no hay posibilidad de gran mejora utilizando algoritmos de paso variable.

Es posible también encontrar otros casos de sistemas rígidos debidos a la presencia de alguna alinealidad que produzca que en cierta zona el sistema evolucione de manera rápida mientras que en otra zona la evolución es lenta (esto puede ser bien resuelto por los algoritmos de paso variable).

Finalmente, mencionaremos que otro caso donde aparecen dificultades para la simulación análogas a las de las raíces complejas es cuando se tiene un sistema lento con una entrada que varía muy rápidamente.

6. **Algoritmos de paso variable**

Como ya mencionamos, una de las alternativas para resolver los problemas que conlleva la simulación de sistemas rígidos es la utilización de algoritmos de paso de integración variable o adaptativo. La idea de todos estos algoritmos es tratar de utilizar un paso pequeño cuando se está en presencia de dinámica rápida y un paso grande cuando esta dinámica desaparece. La forma de lograr esto es estimando de alguna manera el error cometido al realizar un paso y si este error fuera muy grande, reducir el paso. Si en cambio el error fuera muy pequeño (mas pequeño que el error aceptable), será posible incrementar el paso de integración. Generalmente, estos algoritmos también tienen algún método para calcular en cuanto se puede incrementar (o en cuanto se debe reducir) el paso de integración para mantener el error dentro de márgenes aceptables.

Presentaremos entonces el método de Runge-Kutta 4-5, basado en el método de Runge-Kutta de 4° orden (RK4) como ejemplo de este tipo de métodos, ya que es uno de los métodos para casos generales más utilizados en la práctica (es el método que Simulink utiliza por defecto), si bien cabe mencionar que no es muy apropiado para simular sistemas muy rígidos o con presencia de discontinuidades.

6.1. **Algoritmo de Runge Kutta 4-5** [7]

La idea del método es, partiendo de una condición inicial, calcular primero el próximo valor en un solo paso "grande" (de valor $2h$), utilizando RK4, según

$$\overline{X}(t_k + 2h) = X(t_k) + \frac{1}{6}(S_1^{(k)} + 2S_2^{(k)} + 2S_3^{(k)} + S_4^{(k)})$$

donde, como vimos en Runge-Kutta de 4° orden, es:

$$S_1^{(k)} = 2h \cdot f(X_k, t_k)$$

$$S_2^{(k)} = 2h \cdot f(X_k + \frac{S_1^{(k)}}{2}, t_k + \frac{2h}{2})$$

$$S_3^{(k)} = 2h \cdot f(X_k + \frac{S_2^{(k)}}{2}, t_k + \frac{2h}{2})$$

$$S_4^{(k)} = 2h \cdot f(X_k + S_3^{(k)}, t_k + 2h)$$

y luego se vuelve a calcular el valor de X en el mismo instante pero haciendo dos pasos "chicos" (de valor h cada uno) , también mediante RK4.

$$\hat{X}(t_k + h) = X(t_k) + \frac{1}{6}(\tilde{S}_1^{(k)} + 2\tilde{S}_2^{(k)} + 2\tilde{S}_3^{(k)} + \tilde{S}_4^{(k)})$$

y luego, a partir de éste, se calcula:

$$\hat{X}(t_k + 2h) = \hat{X}(t_k + h) + \frac{1}{6}(\hat{S}_1 + 2\hat{S}_2 + 2\hat{S}_3 + \hat{S}_4)$$

Se puede demostrar que el error cometido en $\overline{X}(t_k + 2h)$ (con un solo paso "grande") es:

$$X(t_k + 2h) - \overline{X}(t_k + 2h) = k \cdot (2h)^5 + O(h^6)$$

en tanto que el error cometido en $\hat{X}(t_k + 2h)$ (con dos pasos "chicos") vale:

$$X(t_k + 2h) - \hat{X}(t_k + 2h) = 2k \cdot h^5 + O(h^6)$$

siendo k una constante (la misma para ambos casos).

Luego, resulta que la diferencia entre ambos valores calculados es:

$$\Delta = \overline{X}(t_k + 2h) - \hat{X}(t_k + 2h) = 2k \cdot 15 \cdot h^5 + O(h^6)$$

y luego se puede despejar:

$$2k \cdot h^5 = \frac{\Delta}{15} + O(h^6)$$

resultando:

$$X(t_k + 2h) = \hat{X}(t_k + 2h) + \frac{\Delta}{15} + O(h^6)$$

lo que nos dice que podríamos utilizar la suma $\hat{X}(t_k + 2h) + \frac{\Delta}{15}$ para obtener un error del orden de h^6. Sin embargo lo que en realidad estamos buscando es estimar el error cometido, entonces podemos ver que en realidad Δ es aproximadamente el error cometido al calcular $\overline{X}$, y por lo tanto se lo puede utilizar como medida del error.

Supongamos conocido Δ_0 (cota deseada de error). Además, sabemos que:

$$\Delta \approx 30k \cdot h^5$$

y por lo tanto,

$$\Delta_0 \approx 30k \cdot h_0^{\,5}$$

donde h_0 es el "paso apropiado". Podemos entonces independizarnos de k (que es una constante desconocida) dividiendo miembro a miembro, resultando:

$$\frac{\Delta}{\Delta_0} = \left(\frac{h}{h_0}\right)^5 \Rightarrow h_0 = h\left(\frac{\Delta}{\Delta_0}\right)^{1/5}$$

lo que nos da una fórmula para calcular el nuevo paso de integración.
Esto nos dice lo siguiente: si el valor Δ obtenido es mayor que el desesado, entonces debe recalcularse el último valor, utilizando un paso de integración menor, (o sea, h_0).Si en cambio el error obtenido es menor que el deseado, debe utilizarse un paso de integración mayor (de nuevo, h_0), para el próximo cálculo (obviamente, no se debe recalcular!).

A esta fórmula se la suele modificar para tener en cuenta que al disminuir el paso de integración hay además un aumento lineal del error total acumulado. Por lo que en lugar de utilizar una raíz quinta se suele utilizar una raíz cuarta.

7. Algoritmos implícitos

Los algoritmos implícitos tienen como principal característica el hecho de garantizar la estabilidad de las soluciones aproximadas cuando las soluciones exactas son estables. Por este motivo son muy utilizados para simular especialmente sistemas "stiff" o sistemas con discontinuidades. Como contrapartida cabe mencionar que estos algoritmos tienen en general mucha complejidad para la implementación computacional.

Desarrollaremos a continuación el más simple de los métodos implícitos (equivalente al método de Euler en los métodos explícitos).

7.1. Algoritmo de Euler Implícito (Backward Euler) [7]

Se plantea:

$$X_{k+1} = X_k + f_{k+1} \cdot h$$

lo que evidentemente nos lleva a una ecuación implícita, dado que f_{k+1} depende de X_{k+1}. Antes de avanzar en la forma de resolver esta ecuación, veremos que pasa con la estabilidad.

Supongamos en primer lugar que estamos en presencia de un sistema lineal de primer orden.

$$\dot{x} = -l\,x$$

Luego, aplicando la fórmula de Euler implícito queda:

$$x_{k+1} = x_k - l \cdot x_{k+1} \cdot h \Rightarrow x_{k+1} = \frac{x_k}{1 + l\,h}$$

de donde se desprende que $x_k = \frac{x_0}{(1 + l\,h)^k}$

que es estable siempre para cualquier valor de l positivo, independientemente de h.

Esta conclusión, nuevamente, se puede generalizar para el caso de sistemas lineales de mayor orden (puede demostrarse fácilmente que la aplicación de este algoritmo a un sistema lineal y estacionario de orden *n* estable tiene como resultado un sistema discreto estable), y puede también extenderse a sistemas no lineales.

En el caso de un sistema lineal y estacionario de orden *n*, la implementación computacional del algoritmo implica contar con herramientas de cálculo capaces de invertir matrices, ya que la solución de la ecuación implícita presente en el método es:

$$X_{k+1} = X_k + f_{k+1} \cdot h = X_k + A \cdot X_{k+1} \cdot h + B \cdot u_{k+1} \cdot h \Rightarrow$$
$$X_{k+1} = (I - Ah)^{-1} \cdot [X_k + B \cdot u_{k+1} \cdot h]$$

Lo que implica invertir la matriz $(I - Ah)$ (si el paso de integración es además variable, esta inversión debe realizarse en cada paso).

En el caso de sistemas no lineales, no es tan simple "despejar" el valor de X_{k+1}, y suele recurrirse a algún método numérico (por ej. el de Newton-Raphson) o al cálculo e inversión de la matriz jacobiana, resultando en este último caso un método semi-implícito (que no garantiza siempre la estabilidad) [7].

Estas ideas, no sólo se aplican con métodos de primer orden. Métodos análogos al de Runge Kutta también han sido desarrollados como métodos implícitos, combinados con un control del paso de integración [10].

8. **Lazos algebraicos**

Introduciremos con un ejemplo sencillo el problema de los lazos algebraicos en la simulación de sistemas dinámicos.

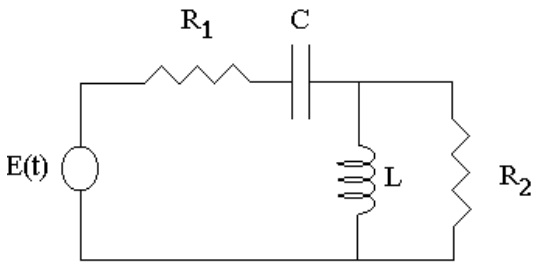

Figura 7: Circuito eléctrico

El Diagrama de Bloques correspondiente es:

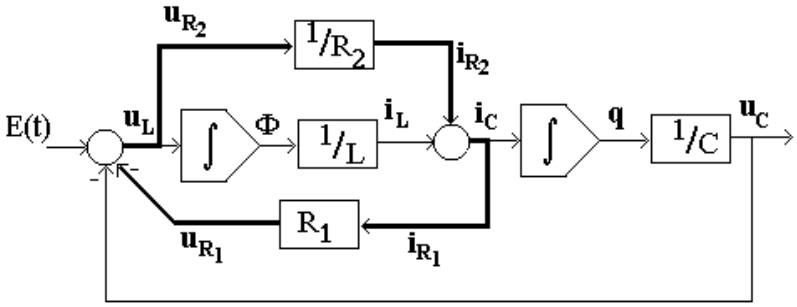

Figura 8: DB correspondiente al circuito de la Fig.7

Si planteamos la aproximación de Euler por ejemplo, obtenemos:

$$\Phi_{k+1} = \Phi_k + h \cdot u_{Lk}$$
$$q_{k+1} = q_k + h \cdot i_{Ck}$$

donde Φ_k y q_k son conocidos (o son condiciones iniciales o bien fueron calculados en el paso anterior). Por otro lado, i_{Ck} y u_{Lk} pueden ser calculados en función de Φ_k y q_k ya que estas últimas son las variables de estado. Para esto, se propagan las variables conocidas a través del DB hasta calcular i_{Ck} y u_{Lk}. El proceso comenzaría así:

$$i_{Lk} = \frac{\Phi_k}{L}$$
$$u_{Ck} = \frac{q_k}{C}$$

pero de aquí en más ya no es posible avanzar, debido a que no se conocen ni i_{R2k} ni u_{R1k}. Lo único que se tiene es un sistema de ecuaciones entre ellas:

$$i_{R2k} = \frac{1}{R_2} u_{Lk} = \frac{1}{R_2}\left[E(t_k) - u_{Ck} - u_{R1k}\right]$$
$$u_{R1k} = R_1 \cdot i_{Ck} = R_1\left[i_{Lk} + i_{R2k}\right]$$

Para poder continuar la simulación, es entonces necesario resolver este sistema de ecuaciones. Un programa capaz de simular un sistema como éste, necesita tener entonces herramientas de cálculo simbólico (que en caso de sistemas no lineales podría no funcionar, ya que el sistema podría tener de por medio alguna ecuación trascendente) o de resolución numérica de ecuaciones algebraicas (por ejemplo, Newton-Raphson).

La aparición de este inconveniente puede explicarse debido a la existencia de un "lazo algebraico", que en el Diagrama de Bloques se traduce en la existencia de un camino cerrado que pasa solamente por bloques estáticos. En el ejemplo, este camino puede verse siguiendo las líneas de trazo grueso en el DB.

Los sistemas de ecuaciones producidos por modelos que contienen lazos algebraicos se denominan "Sistemas de Ecuaciones Algebraico Diferenciales" (DAE, por Differential Algebraic Equations), y su planteo general es de la forma:

$$\dot{X} = f(X, Z, t)$$
$$g(X, Z, t) = 0$$

donde X es el vector de estados y Z es un vector de variables que deben obtenerse de la segunda ecuación, generalmente mediante la utilización de algún método numérico.

La resolución de este tipo de problemas mediante algoritmos de integración implícitos, tiene la ventaja de que se pueden colocar dentro del mismo "paquete" tanto las ecuaciones algebraicas propias del método como las del sistema (causadas por el lazo algebraico), resultando todo en un único sistema de ecuaciones algebraicas. Existen diversos programas de simulación que realizan esto para optimizar el código de resolución del sistema de ecuaciones buscando entre otras cosas reducir al mínimo el número de variables involucradas en el sistema de ecuaciones mencionado para luego aplicar o bien cálculo simbólico (si es que se trata de sistemas lineales no muy grandes generalmente) o bien algún método numérico de cálculo.

Esta política de simulación en la que se combinan las ecuaciones propias del método numérico y las del sistema para optimizar la velocidad de cálculo se conoce como integración "Inline". [10]

9. **Referencias**

[1] Karnopp,D., Rosenberg, R, *Introduction to Physical System Dynamics,* Mc Graw Hill, 1983.

[2] Demidowitsch B.P., Maron I.A., Schuwalowa E.S., *Métodos numéricos de Análisis*, Paraninfo, 1980.

[3] Korn, G.A., Wait, J.V., *Digital Continuous-System Simulation*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978

[4] Laub,A., *Numerical Linear Algebra Aspects of Control Design Computation*, IEEE Tr. on Automatic Control, 1985.

[5] Badano, F., *Integración de Ecuaciones Diferenciales Ordinarias Rígidas*, Proyecto de Ingeniería, FCEIyA, UNR.

[6] Apostol, T., *Calculus*, Vol I, Reverté, 1965

[7] Press W.H., Flannery B.P., Teukolsky S.A., Vetterling W.T. *Numerical Recipes*, Cambridge University Press, 1986.

[8] Kelly L.G., *Handbook of Numerical Method and Applications*, Addison-Wesley Publishing Company, 1967.

[9] Junco, S y Serra S., *Métodos numéricos en la simulación digital de Sistemas Dinámicos*, Cátedra de Dinámica de los Sistemas Físicos.

[10] Elmqvist, H., Otter, M., Cellier, F., *Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems*, European Simulation MultiConference, 1995.