Introducción:

Esta monografía se ciñe a la materia estudiada en la asignatura de Análisis Numérico II del grado de Matemáticas. Debido a la ausencia de material en español (lo cuál ha llegado a ser motivo de queja para los alumnos) que sirva a los estudiantes de guía para el estudio, hemos considerado adecuado poner por escrito algunas ideas que les puedan servir en el futuro de la asignatura.

Los presentes apuntes se dedican al estudio de los métodos más clásicos de resolución de ecuaciones diferenciales.

Hemos dividido el trabajo en tres:

i) Un primer capítulo que les sirva de introducción a los métodos, con diferentes definiciones y ejemplos, que sirven al alumno de antesala ante lo que se avecina.

ii) El segundo capítulo está dedicado a los Métodos Runge-Kutta, especialmente a los explícitos.

iii) Por último, el capítulo tres se dedica al estudio de los métodos multipaso: Adams-Bashforth, Adams-Moulton, BDF, Stormer y Cowell.

Puesto que la asignatura tiene tanto parte teórica como práctica, en cada capítulo hay una serie de problemas que, esperemos, sirvan al alumno a comprender mejor los términos más complejos de la asignatura.

Firmado: Jesus Vigo Aguiar y resto de miembros del equipo

Capítulo 1

Introducción a los métodos numéricos

1.1. Métodos numéricos: definiciones, nomenclatura y ejemplos

En otras asignaturas hemos obtenido una serie de herramientas para obtener soluciones analíticas en diferentes clases de ecuaciones diferenciales. Desgraciadamente hay un gran número de ecuaciones diferenciales no resolubles por ninguno de estos métodos. En los capítulos que proponemos a continuación se estudiarán diferentes métodos numéricos que aproximan la curva cuya condición inicial y derivada son los datos dados por el problema de valor inicial. Esto es, se logra una tabla de números con dos coordenadas (x_i,y_i), donde $y_i \approx y(x_i)$.

Los métodos numéricos pueden tener muchas forma, aquí se propondrán algunos de los métodos Runge-Kutta y multipaso más conocidos.

El problema standard que procuraremos resolver es

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \tag{1.1.1}$$

donde $y = (y^1, \ldots, y^m)$, $f = (f^1, \ldots, f^m)$ e $y_0 = (y_0^1, \ldots, y_0^m)$, son vectores de dimensión m, mientras que t y t_0 son escalares.

Sólo consideraremos el problema (1.1.1) cuya solución exista y sea única. Por ejemplo cuando $f : R \times R^m \to R^m$ **sea continua** $\forall (x, y) \in D$**, donde** $D = \{(t_0, t_f) \times R^m\}$**, y de forma que exista una constante finita L, tal**

que

$$\| f(x,y) - f(x,y^*) \| \leq L \| y - y^* \| \tag{1.1.2}$$

para cualquier (x,y)**,** (x,y^*) **de D. Cumpliéndose estas hipótesis, diversos resultados que no se estudiarán en este manual, nos darán garantías de que exista solución única de nuestro problema, y de que** $y(t)$ **sea continua y diferenciable en D**.

Habrá veces en las que estas hipótesis no se verifiquen, en esos casos el lector habrá de tener cuidado, ya que los métodos que se estudiarán a conti-nuación son bastante generales, y puede que no sirvan para el caso específico. A continuación se dan ejemplos muy breves, que muestran como antes de intentar resolver el problema, es preciso estudiar la casuística del mismo, de esta forma se evitará la perdida de tiempo y de esfuerzos.

Ejemplo 1: consideremos

$$y'(x) = -\sqrt{|y|}, \quad y(3) = 0. \tag{1.1.3}$$

Aquí no se cumplen las condiciones, el resultado es que existen infinitas soluciones, por ejemplo son válidas todas las que sean del tipo

$$y(x) = \begin{cases} \frac{(x-c_1)^2}{4} \text{ si } x \leq c_1 \leq 3 \\ 0 \text{ si } x > c_1 \end{cases}$$

Ejemplo 2: sea nuestro problema

$$y'(x) = 1 + y^2, \quad y(0) = 1, \quad \text{con } x \in [0, \frac{\pi}{4}]. \tag{1.1.4}$$

En este caso no existe dicha constante, razón por la que la solución del problema $y(t) = \tan(x + \frac{\pi}{4})$ no está acotada en $\frac{\pi}{4}$, donde presenta una singularidad.

Ejemplo 3: supongamos que nuestro problema es ahora

$$y'(x) = \lambda(y - E(x)) + E'(x), \quad y(0) = 10. \tag{1.1.5}$$

Donde $E(x) = 10 - (10 - x)e^{-x}$ y $\lambda = -2000$, por ejemplo.

En este caso sí se verifican las condiciones pedidas, pero el hecho de que la constante Lipschitziana sea tan alta hace que los métodos numéricos tengan problemas.

Este ejemplo, considerado de la familia de los Prothero-Robinson es típico entre los problemas stiff que veremos más adelante.

Todos los métodos que se proponen aquí parten de la idea de la discretización del continuo $[t_0, t_f]$ donde se busca la solución $y(t)$ del problema (1.1.1). Consideramos un conjunto de puntos $t_0 < t_1 < \ldots < t_f$, y llamaremos longitud de paso en la iteración n a $h_n = t_n - t_{n-1}$.

De esta forma un método numérico es una ecuación en diferencias que nos permite obtener de forma recursiva una serie de aproximaciones y_n de $y(x_n)$, utilizando valores obtenidos anteriormente $y_{n-1}, \ldots, y_{n-k}$ y evaluaciones de la función $f(t, y(t))$.

La primera gran familia de métodos que se proponen son los Runge-Kutta. Un Runge-Kutta de s pasos se escribe de forma general como

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{1.1.6}$$

donde

$$k_i = f(x_n + c_i h, y_n + \sum_{j=1}^{s} a_{ij} k_i)$$

Mientras que llamaremos métodos multipaso a aquellos métodos de integración numérica que se pueden escribir como

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} = h(\beta_0 f_n + \beta_1 f_{n-1} + ... + \beta_k f_{n-k}) \tag{1.1.7}$$

donde $k > 1$ es un entero fijo que define el número de pasos del método, $f_i = f(x_i, y_i)$, y las α_i, β_i, son constantes que no dependen de n; y_n es el valor que se quiere calcular en cada iteración, mientras que los y_{n-i}, con $0 < i$ son valores que se han obtenido previamente.

1.2. Consistencia

Si consideramos que el método se obtiene de la fórmula general

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} = h\phi_f(y_n, \ldots, y_{n-k}, x_n; h), \tag{1.2.1}$$

donde ϕ_f denota que ϕ depende de y a través de f, y definimos el residuo como

$$R_n := \sum_{j=0}^{k} \alpha_j y(x_{n-j}) - h\phi_f(y_n, \ldots, y_{n-k}, x_n; h) \tag{1.2.2}$$

es fácil observar que para que un método sea "válido "para encontrar una solución apropiada a nuestro problema, necesitaremos no sólo que

$$\lim_{h\to 0} R_n = 0, \tag{1.2.3}$$

también será preciso que

$$\lim_{h\to 0} \frac{R_n}{h} = 0. \tag{1.2.4}$$

Si esto sucede se dice que el método es **consistente**. Para más detalles de esta y otras futuras definiciones, un interesante libro es el de Lambert ([Lam91]).

1.3. Estabilidad

Supongamos que sea nuestro objetivo encontrar una solución numérica del problema (1.1.1), y supongamos ahora que perturbamos levemente tanto la función (con $\delta(t)$), como el valor inicial (en δ), siendo ahora nuestro problema

$$z'(t) = f(t, z(t)) + \delta(t), \quad z(t_0) = z_0 + \delta, \tag{1.3.1}$$

Entonces, se dice que el problema de valor inicial (1.1.1) es **totalmente estable**, si dadas dos perturbaciones $(\delta(t), \delta)$ y $(\delta(t)^*, \delta(t)^*)$ de (1.1.1) y siendo $z(t)$ y $z(t)^*$ las soluciones de los problemas problemas perturbados, existe una constante S tal que $\forall t \in [t_0, t_f]$

$$\| z(t) - z(t)^* \| \leq S\varepsilon$$

siempre que

$$\| \delta(t) - \delta(t)^* \| \leq \varepsilon \text{ y } \| \delta - \delta^* \| \leq \varepsilon. \tag{1.3.2}$$

Aunque esta definición es muy clara, es preciso ser muy cuidadoso con la misma, ya que, por ejemplo, así la función exponencial es estable. Sin embargo, más adelante veremos que este no es un detalle nada trivial, pues pocos métodos integran numéricamente bien dicha función.

Sucede que algunos métodos numéricos aplicados a algunos problemas de valor inicial, introducen errores que deterioran la solución del problema, pudiendo suceder que ante determinadas longitudes de paso, las soluciones oscilen de gran manera llegando a divergir las soluciones, lo cual no sería aceptable en absoluto.

Por eso decimos que un método es **cero-estable** si dadas dos perturbaciones $\{\delta_n, n = 0, \ldots, N\}$ y $\{\delta_n^*, n = 0, \ldots, N\}$ del método

$$\sum_{j=0}^{k} \alpha_j y_{n-j} = h\phi_f(y_n, \ldots, y_{n-k}, x_n; h), \quad y_i = a_i(h), \quad i = -1, \ldots, -k \tag{1.3.3}$$

y siendo $\{z_n, n = 0, \ldots, N\}$ y $\{z_n^*, n = 0, \ldots, N\}$ sus respectivas soluciones, entonces existen constantes S y h_0 tales que $\forall\ h \in (0, h_0]$, se cumple que

$$\| z_n - z_n^* \| \leq S\varepsilon, \qquad 0 \leq n \leq N$$

siempre que

$$\| \delta_n - \delta_n^* \| \leq \varepsilon \qquad 0 \leq n \leq N. \tag{1.3.4}$$

Ahora si introducimos los siguientes polinomios:

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} \longrightarrow \pi(x) = \alpha_0 x^k + \alpha_1 x^{k-1} + ... + \alpha_k, \tag{1.3.5}$$

se dice que el método (1.3.3) verifica la **condición de la raíz** si todas las raíces de $\pi(x)$ tienen módulo menor que uno y las de módulo uno tienen multiplicidad menor o igual que la del orden del problema a resolver.

Dadas estas nociones es posible enunciar el siguiente teorema:

Teorema: la condición necesaria y suficiente para que un método sea estable, es que se cumpla la condición de la raíz.

1.4. Convergencia:

La idea principal tanto de los métodos Runge-Kutta como de los multipaso es que dado el problema de valor inicial

$$y'(x) = f(x, y), \quad y(x_0) = y_0 \tag{1.4.1}$$

entonces cuando $h \to 0$ y para $x_n = x$, se cumpla que

$$\lim_{h \to 0} y_n = y(x). \tag{1.4.2}$$

Pero además es importante cómo se comporta este límite, ya que si para valores de $h \ll 1$, se sigue dando que la diferencia entre el valor exacto $y(x)$ y el valor estimado y_n, es todavía bastante elevado, entonces se está dando

el caso de que el método numérico es muy costoso. Para explicarlo de otra forma, es necesario hacer un número muy elevado de cuentas para hallar una buena aproximación de la solución verdadera.

Con ese fin definimos que un método definido como (1.3.3) es **convergente** si, para todo problema de valor inicial del tipo (1.1.1), con $f : R \times R^m \to R^m$ continua $\forall (x, y) \in D$, donde $D = \{(t_0, t_f) \times R^m\}$, de forma que exista una constante finita L, tal que

$$\parallel f(x, y) - f(x, y^*) \parallel \leq L \parallel y - y^* \parallel \tag{1.4.3}$$

para cualquier (x, y), (x, y^*) de D; entonces tenemos que

$$\max_{0 \leq n \leq N} \parallel y(x_n) - y_n \parallel \to 0 \quad \text{cuando } h \to 0. \tag{1.4.4}$$

Además, estudios más elevados que el que proponemos aquí, provaron que **la condición necesaria y suficiente para que el método sea convergente es que sea a la vez consistente y cero-estable**.

Como ya hemos visto, en general todos los métodos multipaso se pueden escribir como

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} = h(\beta_0 f_n + \beta_1 f_{n-1} + ... + \beta_k f_{n-k}) \tag{1.4.5}$$

donde k es un entero fijo mayor que 1, $f_i = f(x_i, y_i)$, y las α_i, β_i, son constantes que no dependen de n; y_n es el valor que se quiere calcular en cada iteración, mientras que los y_{n-i}, con $0 < i$ son valores que se han obtenido previamente.

Supongamos que el método es cero-estable, como lo queremos hallar es $y(x-ih)-y_{n-i}$, en la fórmula (1.4.5) reemplazamos las y_{n-i}, por las $y(x-ih)$, y las $hf_{n-i} = hy'_{n-i}$, por $hy'(x_{n-i})$, de esta forma obtenemos

$$\alpha_0 y(x_n)+\alpha_1 y(x_{n-1})+...+\alpha_k y(x_{n-k})-h(\beta_0 y'(x_n)+\beta_1 y'(x_{n-1})+...+\beta_k y'(x_{n-k}) \tag{1.4.6}$$

Utilizando el desarrollo de Taylor de las funciones $y(x - ih)$, e $y'(x - ih)$, en un entorno de $x = x_n$ vamos a obtener una expresión del tipo

$$C_0 y(x) + C_1 h y'(x) + ... + C_k h^k y^{(k)}(x) + ... \tag{1.4.7}$$

en ese caso se dice que el método es de orden k-1 cuando se obtiene que $C_0 = C_1 = ... = C_{k-1} = 0$ y que $C_k \neq 0$.

En ese caso el error producido con dicho método es $C_k h^k y^{(k)}(x)+O(h^{k+1})$, que para $0 < h \ll 1$, es aproximadamente $C_k h^k y^{(k)}(x)$.

De manera equivalente, se puede trabajar con los Runge-Kutta hasta hallar el orden del método.

1.5. Problemas stiff:

Los problemas de valor inicial presentan a menudo dificultades que no pueden ser resueltas de forma adecuada por un gran número de métodos numéricos.

Un claro ejemplo puede ser

$$y_1'(x) = 5y_1(x), \quad y_1(0) = 1, \tag{1.5.1}$$

cuya solución es $y_1(x) = e^{5x}$, sin embargo, si modificamos el problema, tomando

$$y_2'(x) = 5y_2(x), \quad y_2(0) = 1 + \epsilon, \tag{1.5.2}$$

resulta que la solución del problema cambia sustancialmente a $y_2(x) = (1 + \epsilon)e^{5x}$, de forma que, por ejemplo si $\epsilon = 10^{-10}$ y $x = 10$, la diferencia $y_2(10) - y_1(10) \approx 5{,}18471 \times 10^{11}$, esto lo que nos indica es que si en la primera iteración el error es de 10^{-10}, ese error se propaga de forma exponencial de forma que en el punto $x = 10$, el error es abismal.

Otro caso particular de las ecuaciones diferenciales es el de los problemas stiff (en algunos libros en español se llaman problemas rígidos).

Este tipo de ecuaciones diferenciales aparecen en aquellos problemas $y'(x) = f(x, y(x))$, donde el jacobiano de $f(x, y(x))$ tiene uno de sus autovalores muy grande en valor absoluto y negativo (si fuera positivo ya hemos visto que tiene difícil solución). En este caso sí es posible encontrar métodos numéricos que tengan un buen comportamiento.

Pongamos un ejemplo de lo que es un problema stiff (ver [Lam91]), para explicar lo que sucede en estos casos:

$$\begin{aligned} y_1'(x) &= -2y_1(x) + y_2(x) + 2\sin(x) \\ y_2'(x) &= 998y_1(x) - 999y_2(x) + 999(\cos(x) - \sin(x)) \\ y_1(0) &= 2 \qquad y_2(0) = 3 \end{aligned} \tag{1.5.3}$$

cuya solución exacta se obtiene fácilmente:

$$y_1(x) = 2e^{-x} + \sin(x), \qquad y_2(x) = 2e^{-x} + \cos(x) \tag{1.5.4}$$

Sin embargo, supongamos que no es conocida, y que no observamos como poder obtenerla. Entonces buscamos un método numérico para poderlo aplicar a este problema.

Por ejemplo, Lambert para explicar lo que sucede en este caso utiliza el método conocido como Runge-Kutta-Fehlberg 45 (es un método de 6 pasos), que es un embedding, esto es, un método que utiliza un par de métodos Runge-Kutta, en este caso los de órdenes 4 y 5 explícitos (el resultado lo da el de orden 5 y el de orden 4 se utiliza para predecir el error y con ello variar el paso, para que de esta forma el error se mantenga por debajo de una cota establecida), para dar una aproximación numérica sin exceder un error que el propio programador establece en cada ejemplo. Este es un método que se estudiará con más detenimiento en el próximo capítulo.

Nuestro problema es aproximar la curva solución de la ecuación entre 0 y 10, e imponemos una modesta cota de 0.01, así el RKF 45 necesita la enorme cifra de 3375 pasos. Y si utilizamos uno del resto de métodos explícitos habituales llegamos a resultados parecidos. Esto sucede porque en el problema que hemos establecido los valores propios del jacobiano son -1 y -1000. Precisamente es este -1000 el que engaña a los métodos que normalmente se utilizan. Para que estos funcionen bien, es necesario que las longitudes de paso de los métodos sean muy pequeñas, por lo que es necesario hacer muchas iteraciones para poder observar cuando se estabiliza la ecuación (estabilidad que viene dada por el otro valor propio que es el más pequeño en valor absoluto). El problema se puede complicar fácilmente cambiando los datos del problema (1.5.4) de tal forma que los valores propios del jacobiano pasen a ser -1 y -1000000, ó -0.001 y -10^8.

Sin embargo, los problemas stiff están considerados más interesantes que los no-stiff en la investigación moderna, ya que al ser más costosos (en cuentas) hay más margen de mejora, por lo que es un tema en el que se estudia bastante en la actualidad. Además tiene importantes aplicaciones, sobre todo en problemas de mecánica celeste o procesos estocásticos, aunque los ejemplos más notables suelen ser reacciones químicas en las que se den variaciones muy bruscas en las concentraciones de las sustancias (en [HaWa96] o [Sha94] es posible encontrar bastantes ejemplos de problemas stiff).

El tema de cúal es el mejor método numérico a utilizar en absoluto es banal. Para entenderlo, nada mejor que poner un ejemplo: nuestro "tremendo"problema stiff que es "tan complicado "de integrar (nada menos que 3375 iteraciones con el RKF45), sin embargo con el método de Gauss de 2 pasos (aunque sea implícito), sólo son necesarias 24 iteraciones para aproximar la curva con la misma cota de error.

En futuros capítulos se presentarán más ampliamente los problemas stiff. Si bien los métodos numéricos que se mostrarán son habituales para resolver

problemas no-stiff, sin embargo como ya veremos pueden tener dificultades al integrar P.V.I. stiff.

Entre los métodos numéricos más generalizados para estos problemas stiff se encuentran el radau5 (un Runge-Kutta implícito de orden 5) y el MEBDF (un BDF implícito de orden variable de 1 a 7).

Los problemas stiff suelen presentar una zona donde la solución varía considerablemente, en esa zona es necesario muchas veces un paso relativamente pequeño, mientras que en otras zonas los mismos problemas no presentan grandes dificultades. Por ello, es bastante habitual que los integradores tengan un cambiador de paso muy estudiado. Además, como se ha visto ya los integradores más generalizados para estos problemas suelen ser implícitos. Los coeficientes de estos métodos son muy precisos, ya que es necesario que presenten propiedades más complejas de estabilidad.

Por todo ello no profundizaremos en este tipo de problemas que, sin embargo, son muy interesantes.

1.6. Selección de problemas.

1. Considérese el siguiente problema de valor inicial:

$$y'(t) = z(t), \quad z'(t) = \frac{z(t)(z(t)-1)}{y(t)}, \quad y(0) = \frac{1}{2}, \quad z(0) = -3. \tag{1.6.1}$$

Es un ejemplo que cumple todas las condiciones para que tenga una única solución y su constante Lipschiztiana no es muy alta, su única solución es

$$y(t) = \frac{1+3e^{-8t}}{8}, \quad z(t) = -3e^{-8t}. \tag{1.6.2}$$

1) Calcúlese el error cometido al utilizar el siguiente método numérico:

$$y_{n+2} + y_{n+1} - 2y_n = \frac{h}{4}(8f(t_{n+1}, y_{n+1}) + 4f(t_n, y_n)) \tag{1.6.3}$$

para $t = 0{,}2$, $t = 0{,}4$, $t = 0{,}6$, $t = 0{,}8$, $t = 1{,}0$; después de utilizar como longitudes de paso $h = 0{,}1$, $h = 0{,}01$ y $h = 0{,}001$.

Posteriomente, explíquese el comportamiento de dicho método.

2) Calcúlese el error cometido al utilizar el siguiente método numérico:

$$y_{n+2} - y_{n+1} = \frac{h}{3}(3f(t_{n+1}, y_{n+1}) - 2f(t_n, y_n)) \tag{1.6.4}$$

para $t = 0{,}2$, $t = 0{,}4$, $t = 0{,}6$, $t = 0{,}8$, $t = 1{,}0$; después de utilizar como longitudes de paso $h = 0{,}1$, $h = 0{,}01$ y $h = 0{,}001$.

Posteriomente, explíquese el comportamiento de dicho método.

3) Calcúlese el error cometido al utilizar el siguiente método numérico:

$$y_{n+2} - y_n = 2h(f(t_{n+1}, y_{n+1})) \tag{1.6.5}$$

para $t = 0{,}2$, $t = 0{,}4$, $t = 0{,}6$, $t = 0{,}8$, $t = 1{,}0$; después de utilizar como longitudes de paso $h = 0{,}1$, $h = 0{,}01$ y $h = 0{,}001$.

Posteriomente, explíquese el comportamiento de dicho método.

Solución :

1) Calcularemos el error cometido en t_n como $\sqrt{(y_n - y(t_n)^2 + (z_n - z(t_n))^2}$, según ello los resultados obtenidos se muestran en la siguiente tabla:

t	h=0.1	h=0.01	h=0.001
0.2	0.51443	2323.51	$3{,}81574 \times 10^{55}$
0.4	3.59772	$5{,}44887 \times 10^{9}$	$1{,}36535 \times 10^{116}$
0.6	31.9894	$1{,}27782 \times 10^{16}$	$4{,}88548 \times 10^{176}$
0.8	287.797	$2{,}99665 \times 10^{22}$	$9{,}31632 \times 10^{236}$
1.0	2590.14	$7{,}02749 \times 10^{28}$	$6{,}25513 \times 10^{297}$

Aunque el método es consistente, sin embargo es cero-inestable, ya que una de las raíces de $x^2 + x - 2 = 0$ es -2 que tiene módulo mayor que uno. Por eso el método es divergente como se puede observar en la tabla.

2) Los resultados que se pueden obtener son:

t	h=0.1	h=0.01	h=0.001
0.2	1.27374	1.11037	1.15759
0.4	1.10191	0.90784	0.91616
0.6	0.79501	0.59294	0.58618
0.8	0.55384	0.36517	0.35422
1.0	0.38425	0.22080	0.21018

Aunque, a juzgar por los resultados, en este ejemplo resulta más complicado darse cuenta, este método tampoco es convergente.

En este caso, sí es cero-estable. Sin embargo, no es consistente:
$y(x_{n+2}) - y(x_{n+1}) - \frac{h}{3}(3f(t_{n+1}, y(x_{n+1})) - 2f(t_n, y(x_n))) =$

$y(x_n)+2hy'(x_n)-y(x_n)-hy'(x_n)-\frac{h}{3}(3f(t_n,y(x_n))-2f(t_n,y(x_n)))+O(h^2) =$
$hy'(x_n) - \frac{h}{3}(f(t_n,y(x_n))) + O(h^2) = \frac{2h}{3}y'(x_n) + O(h^2)$.

Por eso el error se mantiene bastante estable, aún cuando el paso se reduce considerablemente.

3) Este método también es conocido como la regla del punto medio, y éste sí es convergente como se puede suponer por los datos obtenidos:

t	h=0.1	h=0.01	h=0.001
0.2	0.23938	0.0016010	0.0000110253
0.4	0.728426	0.0033092	$7{,}34036 \times 10^{-6}$
0.6	3.00892	0.0144356	0.0000168206
0.8	12.9775	0.0707874	0.0000773499
1.0	56.1653	0.349856	0.000381499

Aunque al principio el error es bastante alto, sin embargo, pronto disminuye rápidamente.

Es claro que es cero-estable, pues las raíces de $x^2 - 1 = 0$ son simples. Y también es consistente, ya que
$y(x_{n+2}) - y(x_n) - 2hf(t_{n+1}, y(x_{n+1})) =$
$y(x_n) + 2hy'(x_n) - y(x_n) - 2hf(t_{n+1}, y(x_{n+1})) + O(h^2) =$
$2hy'(x_n) - 2hy'(x_n) + O(h^2) = O(h^2)$.

2. Demuéstrese que el siguiente método, conocido como la fórmula de Milne (aunque en la teoría de integraación numérica es más conodido como la fórmula de Simpson simple)

$$y_{n+2} - y_n = \frac{h}{3}(f(t_{n+2}, y_{n+2}) + 4f(t_{n+1}, y_{n+1}) - f(t_n, y_n))$$

es convergente.

Solución :

De nuevo es fácil observar que es cero-estable, ya que al igual que en el caso tres del ejercicio anterior, el polinomio correspondiente de estabilidad es $x^2 - 1$, cuyas raíces ± 1 son simples.

También es consistente, ya que
$y(x_{n+2}) - y(x_n)\frac{h}{3}(f(t_{n+2}, y(x_{n+2})) + 4f(t_{n+1}, y(x_{n+1})) - f(t_n, y(x_n))) =$
$y(x_n) + 2hy'(x_n) - y(x_n) - 2hf(t_{n+1}, y(x_{n+1})) + O(h^2) =$
$2hy'(x_n) - 2hy'(x_n) + O(h^2) = O(h^2)$.

De hecho sin más que hacer el desarrollo de Taylor de orden 4 de $y(x_{n+2})$, y los desarrollo de orden 3 de $f(t_{n+2}, y(x_{n+2}))$ y de $f(t_{n+1}, y(x_{n+1}))$, se llega a que
$y(x_{n+2}) - y(x_n) \frac{h}{3}(f(t_{n+2}, y(x_{n+2})) + 4f(t_{n+1}, y(x_{n+1})) - f(t_n, y(x_n))) = O(h^5)$, esto es, el método es de orden 4.

3. Escríbase la condición necesaria para que un Runge-Kutta sea convergente.

Solución :

Sabemos que para que sea convergente será necesario que sea a la vez consistente y cero-estable.

Pero fácilmente podemos observar que es cero-estable, ya que de la fórmula (1.1.6) obtenemos que su polinomio característico es x-1, con raíz de módulo 1 simple.

Luego bastará con demostrar que es consistente. Para ello estudiaremos el residuo del método utilizando el desarrollo de Taylor tal como se explica en la sección de Convergencia de este capítulo.

Observando que cuando $h \to 0$, el residuo de un método general como (1.2.1) verifica que

$$\lim_{h \to 0} \frac{R_n}{h} = \frac{1}{h} \left(\sum_{j=0}^{k} \alpha_j \right) y(x_n) + \sum_{j=0}^{k} j \alpha_j y'(\xi_j) - \phi_f(y(x_{n+k}), \ldots, y(x_n), x_n; h)$$

donde $\xi_j \in [x_n, x_{n+j}]$, sin embargo cuando $h \to 0$

$$y'(\xi_j) = y'(x_n), \quad j = 0, \ldots, k$$

y

$$\phi_f(y(x_{n+k}), \ldots, y(x_n), x_n; h) = \phi_f(y(x_n), \ldots, y(x_n), x_n; 0).$$

Con lo que el residuo vale 0 si se verifican a la vez

$$\left(\sum_{j=0}^{k} \alpha_j \right) = 0, \tag{1.6.6}$$

que se verifica en el caso de un Runge-Kutta, ya que $(\sum_{j=0}^{k} \alpha_j) = 1 - 1 = 0$, y

$$\left(\sum_{j=0}^{k} j\alpha_j \right) y'(x_n) = \phi_f(y(x_n), \ldots, y(x_n), x_n; 0) \tag{1.6.7}$$

o, de manera equivalente

$$y'(x_n) = \frac{\phi_f(y(x_n), \ldots, y(x_n), x_n; 0)}{(\sum_{j=0}^{k} j\alpha_j)} = f(x_n, y(x_n)). \tag{1.6.8}$$

Aplicando dicha condición en un Runge-Kutta de s pasos como (1.1.6)

$$\phi_f(y(x_n), x_n; 0) = f(x_n, y(x_n)) \Leftrightarrow \sum_{i=1}^{s} b_i = 1.$$

4. Calcúlese el orden del siguiente Runge-Kutta, denominado Gauss de 2 pasos

$$y_{n+1} = y_n + h\left(\frac{k_1}{2} + \frac{k_2}{2} \right),$$

donde

$$k_1 = f\left(x_n + \left(\frac{1}{2} + \frac{\sqrt{3}}{6} \right) h, y_n + h\left(\frac{k_1}{4} + k_2\left(\frac{1}{4} + \frac{\sqrt{3}}{6} \right) \right) \right),$$

$$k_2 = f\left(x_n + \left(\frac{1}{2} - \frac{\sqrt{3}}{6} \right) h, y_n + h\left(k_1\left(\frac{1}{4} - \frac{\sqrt{3}}{6} \right) \right) + \frac{k_2}{4} \right).$$

Solución :

El anterior ejercicio nos da una idea de cómo debemos de proceder, dado que el Runge-Kutta es cero-estable, para que sea convergente, lo primero que se debe comprobar es que se verifica que $\sum_{i=1}^{s} b_i = 1$, condición que marca que un Runge-Kutta tiene orden igual o superior a 1.

Para que el orden sea al menos 2 se debe cumplir además que

$$\lim_{h \to 0} \frac{R_n}{h^2} = 0, \tag{1.6.9}$$

que procediendo tal y como hemos hecho en el ejercicio anterior y que detallaremos más adelante en el siguiente capítulo, se transforma en la condición

$$\sum_{i=1}^{s} b_i c_i = \frac{1}{2}. \tag{1.6.10}$$

En general para que un Runge-Kutta tenga orden igual o superior a $n \geq 1$ es necesario que

$$\lim_{h \to 0} \frac{R_n}{h^n} = 0, \tag{1.6.11}$$

que utilizando el desarrollo de Taylor de orden n se convierte en las siguientes condiciones

Orden	Condiciones
1	$\sum_{i=1}^{s} b_i = 1$
2	$\sum_{i=1}^{s} b_i c_i = \frac{1}{2}$
3	$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3} \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j = \frac{1}{6}$
4	$\sum_{i=1}^{s} b_i c_i^3 = \frac{1}{4} \quad \sum_{i,j=1}^{s} b_i c_i a_{ij} c_j = \frac{1}{8}$ $\sum_{i,j=1}^{s} b_i a_{ij} c_j^2 = \frac{1}{12} \quad \sum_{i,j,k=1}^{s} b_i a_{ij} a_{jk} c_k = \frac{1}{24}$
5	$\sum_{i=1}^{s} b_i c_i^4 = \frac{1}{5} \quad \sum_{i,j=1}^{s} b_i c_i^2 a_{ij} c_j = \frac{1}{10}$ $\sum_{i,j=1}^{s} b_i c_i a_{ij} c_j^2 = \frac{1}{15} \quad \sum_{i,j,k=1}^{s} b_i c_i a_{ij} a_{jk} c_k = \frac{1}{30}$ $\sum_{i,j=1}^{s} b_i (\sum_{i,j=1}^{s} a_{ij} c_j)^2 = \frac{1}{20} \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j^3 = \frac{1}{20}$ $\sum_{i,j,k=1}^{s} b_i a_{ij} c_j a_{jk} c_k = \frac{1}{40} \quad \sum_{i,j,k=1}^{s} b_i a_{ij} a_{jk} c_k^2 = \frac{1}{60}$ $\sum_{i,j,k,l=1}^{s} b_i a_{ij} a_{jk} a_{kl} c_l = \frac{1}{120}$

El Gauss de dos pasos cumple claramente:

$$\sum_{i=1}^{s} b_i = \frac{1}{2} + \frac{1}{2} = 1,$$

con lo que al menos tiene orden 1

$$\sum_{i=1}^{s} b_i c_i = \frac{1}{2} \times \left(\frac{1}{2} + \frac{\sqrt{3}}{6} \right) + \frac{1}{2} \times \left(\frac{1}{2} - \frac{\sqrt{3}}{6} \right) = \frac{1}{2},$$

de esta forma podemos asegurar que como mínimo tiene orden 2,

$$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3} \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j = \frac{1}{6},$$

y podemos asegurar que tiene orden mayor o igual a tres. También se verifica que tiene orden 4:

$$\sum_{i=1}^{s} b_i c_i^3 = \frac{1}{4} \quad \sum_{i,j=1}^{s} b_i c_i a_{ij} c_j = \frac{1}{8} \quad \sum_{i,j=1}^{s} b_i a_{ij} c_j^2 = \frac{1}{12} \quad \sum_{i,j,k=1}^{s} b_i a_{ij} a_{jk} c_k = \frac{1}{24},$$

sin embargo, no es cierto que tenga orden 5, ya que por ejemplo:

$$\sum_{i=1}^{s} b_i c_i^4 = \frac{1}{2} \times \left(\frac{7 + 4\sqrt{3}}{36} + \frac{7 - 4\sqrt{3}}{36} \right) = \frac{7}{36} \neq \frac{1}{5}.$$

Por lo que el método de Gauss de dos pasos tiene orden 4.

Nota:

En el ejercicio anterior, se mostraba un método numérico más complejo de lo que en un primer momento podía parecer, ya que para calcular k_1, utilizábamos k_1 y k_2, es decir, previsiblemente vamos a tener que hacer una primera inicialización de dichos valores, lo mismo sucede con k_2, que también depende de k_1 y de él mismo.

De ahora en adelante vamos a representar el esquema de los Runge-Kutta con la siguiente tabla:

c_1	a_{11}	$\dots$	a_{1s}
c_2	a_{21}	$\dots$	a_{2s}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	$\dots$	a_{ss}
	b_1	$\dots$	b_s

y diremos que un Runge-Kutta es implícito cuando haya algún $a_{ij} \neq 0$ con $i > j$. Diremos que es semiimplícito si $a_{ij} = 0$ si $i > j$, pero algún $a_{ii} \neq 0$. Esto es, si

c_1	a_{11}	0	0	$\dots$	0
c_2	a_{21}	a_{22}	0	$\dots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_{s-1}	$a_{s-1,1}$	$a_{s-1,2}$	$\dots$	$a_{s-1,s}$	0
c_s	a_{s1}	a_{s2}	$\dots$	$a_{s,s-1}$	a_{ss}
	b_1	$\dots$	$\dots$	b_s	

Por último, un Runge-Kutta es explícito si $a_{ij} = 0$ si $i \geq j$. Esto es, si

c_1	0	0	$\dots$	0
c_2	a_{21}	0	$\dots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	$\dots$	$a_{s,s-1}$	0
	b_1	$\dots$	$\dots$	b_s

Está claro que este tipo de métodos son más sencillos y también menos costosos en cuentas. Por el contrario, ya hemos visto que un Runge-Kutta implícito de dos pasos puede llegar a tener orden 4, sin embargo, esto no es posible con los explícitos como vamos a ver a continuación. En general un implícito puede llegar a tener mayor orden que uno explícito.

5. 1) Pruébese que un Runge-Kutta explícito de dos pasos como mucho tiene orden dos.
2) Pruébese que un Runge-Kutta explícito de tres pasos tiene a lo sumo orden tres.

Solución :

1) Un Runge-Kutta explícito de dos pasos tiene el siguiente esquema:

0		
c_2	a_{21}	
	b_1	b_2

y además $a_{21} = c_2$. Como se tiene que verificar, para que al menos tenga orden 2, que $\sum_{i=1}^{2} b_i = 1$ y $\sum_{i=1}^{2} b_i c_i = \frac{1}{2}$, de esta manera $b_2 c_2 = \frac{1}{2}$, con lo que en general el método viene dado por $c_2 = \lambda \neq 0$, entonces $b_2 = \frac{1}{2\lambda}$, con lo que $b_1 = 1 - \frac{1}{2\lambda}$.

Por lo que hemos visto es posible obtener un Runge-Kutta explícito de orden dos, sin embargo esto es lo máximo, ya que para que tuviera orden 3,

se tendría que dar además $\sum_{i=1}^{2} b_i c_i^2 = \frac{1}{3}$ y $\sum_{i,j=1}^{2} b_i a_{ij} c_j = \frac{1}{6}$.

Pero $\sum_{i,j=1}^{2} b_i a_{ij} c_j = b_2 a_{21} c_1 = 0 \neq \frac{1}{6}$, (ya que a_{21} es el único $a_{ij} \neq 0$).

2) Un Runge-Kutta explícito de tres pasos tiene el siguiente esquema:

0			
c_2	a_{21}		
c_3	a_{31}	a_{32}	
	b_1	b_2	b_3

donde además $a_{21} = c_2$ y $a_{31} + a_{32} = c_3$. De resolver las condiciones de orden 3 se pueden sacar distintas familias de tres pasos:

i) Si tomamos $a_{21} = c_2 = \lambda \neq 0$ y $c_3 = \mu \neq 0$, entonces se obtienen

$$b_1 = \frac{6\lambda\mu - 3(\lambda+\mu) + 2}{6\lambda\mu}, \quad b_2 = \frac{2 - 3\mu}{6\lambda(\lambda - \mu)},$$

$$b_3 = \frac{3\lambda - 2}{6\mu(\lambda - \mu}, \quad a_{31} = \frac{\mu(3\lambda(\lambda-1)+\mu)}{\lambda(3\lambda-2)},$$

$$a_{32} = \frac{\mu(\lambda-\mu)}{\lambda(3\lambda-2)}.$$

Sin embargo, para que esta familia tuviera orden 4, se tendría que cumplir por ejemplo $\sum_{i,j,k=1}^{3} b_i a_{ij} a_{jk} c_k = \frac{1}{24}$, y dado que $c_1 = 0$, y que sólo a_{21}, a_{31} y a_{32} son distintas de cero entre las a_{ij}, entonces
$\sum_{i,j,k=1}^{3} b_i a_{ij} a_{jk} c_k = \sum_{i,j=1}^{3} b_i a_{ij} a_{j2} c_2 = \sum_{i,j=1}^{3} b_i a_{i3} a_{32} c_2 = 0 \neq \frac{1}{24}$.

De igual forma se llega al mismo resultado en las otras posiblidades.

ii) $c_2 = \frac{2}{3}$, $c_3 = 0$, $b_2 = \frac{3}{4}$, $b_3 = \lambda \neq 0$, $b_1 = \frac{1}{4} - \lambda$ y $a_{31} = -a_{32} = \frac{-1}{4\lambda}$.

iii) $c_2 = c_3 = \frac{2}{3}$, $b_1 = \frac{1}{4}$, $b_3 = \lambda \neq 0$, $b_2 = \frac{3}{4} - \lambda$, $a_{31} = \frac{2}{3} - \frac{1}{4\lambda}$ y $a_{32} = \frac{1}{4\lambda}$.

De forma similar se puede demostrar el siguiente teorema:

Teorema: Un método Runge-Kutta explícito de s etapas no puede tener orden mayor que s.

6. Pruébese que, por el contrario, hay Runge-Kutta semiimplícitos de dos pasos y orden 3.

Solución :

Un Runge-Kutta semiimplícito de dos pasos tiene el siguiente esquema:

$$\begin{array}{c|cc} c_1 & a_{11} & \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array}$$

y además $a_{11} = c_1$ y $a_{21} + a_{22} = c_2$

Para que tenga orden 3, ya hemos visto que basta con que se verifique $\sum_{i=1}^{2} b_i = 1$, $\sum_{i=1}^{2} b_i c_i = \frac{1}{2}$, $\sum_{i=1}^{2} b_i c_i^2 = \frac{1}{3}$, y $\sum_{i,j=1}^{2} b_i a_{ij} c_j = \frac{1}{6}$.

Tomemos $a_{21} = \lambda$, basta con tomar

$$a_{11} = c_1 = \frac{3\lambda - 1}{6\lambda}, \quad c_2 = \frac{1+\lambda}{2},$$

$$a_{22} = \frac{1-\lambda}{2}, b_1 = \frac{3\lambda^2}{3\lambda^2+1} \text{ y } b_2 = \frac{1}{3\lambda^2+1}.$$

Sin embargo, ninguno de estos métodos tiene orden mayor que 3, como hemos visto que sucede con los implícitos, ya que para que tuvieran orden 4, se tendría que cumplir, por ejemplo que $\sum_{i=1}^{2} b_i c_i^3 = \frac{3\lambda^2+18\lambda-1}{72\lambda} = \frac{1}{4}$, lo que sucede sólo si $\lambda = \pm\frac{1}{\sqrt{3}}$.

Pero si además exigimos $\sum_{i,j,k=1}^{2} b_i a_{ij} a_{jk} c_k = \frac{1}{24}$, nos encontramos con

$$\sum_{i,j,k=1}^{2} b_i a_{ij} a_{jk} c_k = b_1 a_{11}^2 c_1 + b_2(a_{21}a_{11}c_1 + a_{22}(a_{21}c_1 + a_{22}c_2)) = \frac{\pm 1}{12\sqrt{3}} \neq \frac{1}{24}.$$

Nota

Es de justicia explicar que la mayoría de los lenguajes matemáticos, o que tengan varias librerías dedicadas exclusivamente al cálculo numérico, presentan comandos con los que resolver ecuaciones diferenciales ordinarias de modo tanto analítico como numérico, y en estas, tanto ecuaciones no-stiff como stiff.

Así en Maple encontramos la función integrada *dsolve* que resuelve ecuaciones diferenciales y también sistemas de ecuaciones, tanto analíticamente como de forma numérica. Si nos centramos en esta última opción, observamos que los métodos que podemos utilizar son muy variados (se invocan al escribir *method*= ' nombre', y algunas de las opciones son *gear, lsode o rkf45*). MATLAB es muy similar, ya que proporciona varias opciones para resolver problemas de valor inicial. Además, tanto uno como el otro, presentan buenas rutinas gráficas para dibujar la solución tanto en 2 como en 3 dimensiones.

El Mathematica, también presenta las funciones *DSolve y NDSolve*. La primera para resolver ODE's de forma analítica, la segunda sirve para resolverlas con métodos numéricos, en este caso el Mathematica distingue entre ecuaciones stiff y no-stiff, y dependiendo de esto elige el método a utilizar.

Capítulo 2

Métodos Runge-Kutta

2.1. Introducción :

El conjunto de problemas relacionados con las ecuaciones diferenciales es muy amplio y diverso, como ya se ha visto en otras asignaturas. Ya se ha estudiado como resolver ecuaciones y sistemas diferenciales de forma analítica. Así, dada una ecuación diferencial como por ejemplo

$$y'(t) = 2ty(t) \tag{2.1.1}$$

primero se obtiene una solución general

$$y(t) = ce^{t^2} \tag{2.1.2}$$

en este caso. Dicha solución depende de una constante arbitraria c. Si se presenta un problema de valor inicial (P.V.I.), en ciertas condiciones, esa constante quedará fijada, dando lugar a lo que se conoce como la solución particular del P.V.I., por ejemplo

$$y'(t) = 2ty(t), \quad y(0) = 3 \tag{2.1.3}$$

tiene como solución del P.V.I. a

$$y(t) = ce^{t^2}, \tag{2.1.4}$$

pero como $y(0) = 3$, c sólo puede valer 3.

Sin embargo, en numerosas ocasiones, no será posible encontrar una solución analítica del problema. Así, es bien conocido que la función $exp(-x^2)$

no tiene una función elemental simple cuya derivada sea dicha función. De la misma forma ecuaciones con expresiones tan "inocentes "como

$$y'(x) = x^2 + y^2, \quad \text{ó} \quad y''(x) = 6y^2 + x \tag{2.1.5}$$

no pueden resolverse en términos de funciones elementales.

En estos casos, sólo es posible conocer una solución numérica que aproxime la curva cuya condición inicial y derivada sean los datos dados por el P.V.I. Esto es, se logra una tabla de números con dos coordenadas (x_i,y_i), siendo por ejemplo $x_i = x_0 + ih$, e $y_i \approx y(x_i)$.

2.2. Idea geométrica de los métodos Runge-Kutta

Este conjunto de métodos nos permiten calcular en ciertos puntos el valor de la solución de la ecuación diferencial de forma numérica. Cuando necesitemos el valor de la solución en un punto distinto a los que nos ha proporcionado el método Runge-Kutta usaremos una interpolación. Siendo los Runge-Kutta especialmente valiosos por el orden de la aproximación.

Supongamos que queremos encontrar una curva que pase por el punto $P = (x_0, y_0)$, y que sea solución de la ecuación $y' = f(x, y)$. Ahora tratamos obtener y_1, el valor en $x_1 = x_0 + h$, esto es Q=(x_1, y_1), pero cuando h sea pequeña, ya hemos visto que la recta PQ se acercará mucho a la tangente de

la curva solución en P, esto es, $y_1 \approx y_0 + hk_1 = y_0 + hy'(0) = y_0 + hf(x_0, y_0)$, lo que se conoce como la Fórmula de Euler explícita. Sin embargo, es fácil observar que es posible lograr una mayor aproximación mediante $k_2 = y'(\xi)$ con ξ un valor apropiado intermedio entre x_0 y x_1, es decir $k_2 = f(x_0 + ah, y_0 + bf(x_0, y_0))$. Con a, b dos valores adecuados entre 0 y 1. El problema está en averiguar qué valores de a y b son los adecuados. Para ello también sabemos que $y_1 = y(x_0 + h) \approx y_0 + hy'_0 + 1/2h^2y''_0$ es mejor aproximación.

2.3. Métodos de segundo orden.

Por lo que hemos visto, si tomamos $y_1 = y_0 + k_3$, donde $k_3 = hy'_0 + 1/2h^2y''_0$, entonces obtendremos un método más efectivo, de hecho ahora tiene orden 2, esto es el error cometido ahora es de $O(h^3)$:
sabemos que $y' = f(t, y)$, y por tanto $y'' = f_t + f_y y' = f_t + f_y f$.
Por tanto

$$y_1 = y_0 + hf + 1/2h^2(f_t + f_y f) + O(h^3) = y_0 + 1/2hf + 1/2h(f + hf_t + hf_y f) + O(h^3)$$

Pero $f(t + h, x + hf) = f + hf_t + hf_y f + O(h^2)$, luego $y_{n+1} = y_n + 1/2hf + 1/2hf(t + h, x + hf) + O(h^3)$,
con lo que la implementación estará dada por tres pasos

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h, y_n + k_1) \\ y_{n+1} &= y_n + 1/2(k_1 + k_2) + O(h^3) \end{aligned} \tag{2.3.1}$$

Esta fórmula se conoce también como método de Heun, y es sólo uno de los muchos Runge-Kutta de orden 2 explícitos. En general las fórmulas de Runge-Kutta explícitas de orden 2 tienen la siguiente forma :
$y_{n+1} = y_n + w_1 hf + w_2 hf(t + ah, x + bhf) + O(h^3)$,
que usando la serie de Taylor de dos variables queda como
$y_{n+1} = y_n + w_1 hf + w_2 hf(t + ah, x + bhf) + O(h^3)$,
con las condiciones
$w_1 + w_2 = 1$,
$w_2 a = 1/2$,
$w_2 b = 1/2$
con lo que $a = b$, $w_1 = 1 - 1/2a$, $w_2 = 1/2a$.

2.4. Métodos Runge-Kutta explícitos de orden superior.

Para obtener métodos de orden superior el mecanismo es muy similar y en general repetitivo, en la selección de problemas que se propone a continuación se hace un estudio más detallado, ahora daremos un ejemplo de orden 3 y otro de orden 4, pero en general los procesos no son únicos, sino que al igual que los Runge-Kutta de orden 2, hay muchas posibilidades.

Un ejemplo de un método de orden tres sería :

$k_1 = hf(x_0, y_0)$
$k_2 = hf(x_0 + 1/2h, y_0 + 1/2k1)$
$k_3 = hf(x_0 + h, y_0 - k1 + 2k2)$
$y_{n+1} = y_n + 1/6(k_1 + 4k_2 + k_3) + O(h^4)$,

mientras que uno de orden 4 podría ser

$k_1 = hf(x_0, y_0)$,
$k_2 = hf(x_0 + 1/2h, y_0 + 1/2k1)$,
$k_3 = hf(x_0 + 1/2h, y_0 + 1/2k2)$,
$k_4 = hf(x_0 + h, y_0 + k3)$,
$y_{n+1} = y_n + 1/6(k1 + 2k2 + 2k3 + k4) + O(h^5)$.

Para cada orden hay una familia de métodos en función de ciertos paráme-

tros, según se tomen distintos valores para cada letra se obtendrán distintos métodos, y cada uno funciona mejor con determinadas funciones. Sin embargo, las fórmulas dadas anteriormente suelen tener muy buen comportamiento.

Habitualmente se trabaja con métodos Runge-Kutta hasta orden 4, ya que como muestra la siguiente tabla, a partir de aquí el número de evaluaciones requeridas aumenta más rápidamente que el orden de los métodos.

Número de evaluaciones de la función	1	2	3	4	5	6	7	8
Orden máximo de los Runge-Kutta explícitos	1	2	3	4	4	5	6	6

2.5. Método Runge-Kutta-Fehlberg adaptativo.

En ocasiones, por el contrario, es preferible que el paso $h_i = x_{i+1} - x_i$, sea una cantidad variable, ya que hay numerosas curvas que centran su interés en intervalos muy concretos. Por ello entre los Runge-Kutta son destacables un método de cuarto orden con 5 evaluaciones, y otro de orden 5 con 6 evaluaciones. A priori no parecen que sean beneficiosos, pero Fehlberg (1969) fue capaz en ambos métodos de lograr una estimación del error de truncamiento local, que sirve para adaptar la longitud de paso h_i en cada iteración, de forma que el error se mantenga por debajo de una cota que es fijada previamente.

A continuación daremos el método Runge-Kutta-Fehlberg de orden 5, que utiliza dos fórmulas, de órdenes 5 y 4. Estos métodos nos darán valores aproximados en cada iteración, que se denotarán y_{i+1}, $\tilde{y}_{i+1}$ respectivamente:

$$y_{i+1} = y_i + \sum_{i=1}^{6} a_i F_i \tag{2.5.1}$$

$$\tilde{y}_{i+1} = y_i + \sum_{i=1}^{6} b_i F_i \tag{2.5.2}$$

donde F_i se calcularán como venimos haciendo :

$$F_i = hf(t + c_i h, y + \sum_{j=1}^{i-1} d_{ij} F_j)(i = 1, ..., 6) \tag{2.5.3}$$

si consideramos que la fórmula (2.5.1) es la de orden 5 y la fórmula (2.5.2) es la de orden 4, entonces es (2.5.1) la que nos da y_i, por tanto es la que

da el valor de salida del algoritmo. A continuación se da una tabla con los valores de los coeficientes para dicho Runge-Kutta-Fehlberg adaptativo.

i	a_i	$a_i - b_i$	c_i	d_{ij}
1	$\frac{16}{135}$	$\frac{1}{360}$	0	0
2	0	0	$\frac{1}{4}$	$\frac{1}{4}$
3	$\frac{6656}{12825}$	$-\frac{128}{4275}$	$\frac{3}{8}$	$\frac{3}{32}, \frac{9}{32}$
4	$\frac{28561}{56430}$	$-\frac{2197}{75240}$	$\frac{12}{13}$	$\frac{1932}{2197}, -\frac{7200}{2197}, \frac{7296}{2197}$
5	$-\frac{9}{50}$	$\frac{1}{50}$	1	$\frac{439}{216}, -8, \frac{3680}{513}, -\frac{845}{4104}$
6	$\frac{2}{55}$	$\frac{2}{55}$	$\frac{1}{2}$	$-\frac{8}{27}, 2, -\frac{3544}{2565}, \frac{1859}{4104}, -\frac{11}{40}$

La diferencia $e_i = y_i - \tilde{y}_i$ nos dará una estimación del error. Cuando e_i es mayor que la tolerancia δ elegida previamente, entonces h se reduce.

Por otra parte como el error de truncamiento local es Ch^5 cuando se calcula con $e_i = y_i - \tilde{y}_i$, parece sensato duplicar el paso en caso de que $C(2h)^5 < \delta/4$, esto es de que $e_i < \delta/128$.

Otra forma habitual de controlar el error por paso es la de tomar $h = c_1 h(d/e)^{1/(1+p)}$, donde $c_1 < 1$, por ejemplo 0.9, y p es el orden del primer método. Si $e > d$ se vuelve a establecer la longitud de paso y se repite la iteración, mientras que en caso contrario se da el valor de salida del algoritmo y se calcula una nueva longitud de paso.

2.6. Estabilidad absoluta de los métodos Runge-Kutta.

Hasta ahora hemos hablado de la cero-estabilidad de un método en el análisis de la convergencia, puede suceder, sin embargo, que un método convergente no trabaje bien para ciertos tamaños de paso en ciertos problemas. Pongamos un ejemplo :

$$y' = -\lambda y, \quad y(0) = 1, \tag{2.6.1}$$

cuya solución es $y(x) = exp(-\lambda x)$. Claramente cuando $x \to \infty$, entonces $y \to 0$ (y siempre tiene valores positivos). Pero vamos a estudiar cómo se comporta el método de Euler explícito con el problema $f(x, y) = -20y$, esto es, $\lambda = 20$, en este caso, por tanto si $h = 0{,}2$, entonces

$y_1 = y_0 + hf(x_0, y_0) = 1 + 0{,}2 \cdot (-20) = -3$
$y_2 = y_1 + hf(x_1, y_1) = -3 + 0{,}2 \cdot 60 = 9$
$y_3 = y_2 + hf(x_2, y_2) = 9 + 0{,}2 \cdot (-180) = -25$, ...,
y el método parece que diverge. Por el contrario, si tomamos $h = 0{,}01$, entonces :
$y_1 = y_0 + hf(x_0, y_0) = 1 + 0{,}01 \cdot (-20) = 0{,}8$
$y_2 = y_1 + hf(x_1, y_1) = 0{,}8 + 0{,}01 \cdot (-16) = 0{,}64$
$y_3 = y_2 + hf(x_2, y_2) = 0{,}64 + 0{,}01 \cdot (-12{,}8) = 0{,}512$, ...,
$y_{20} = 0{,}011529$, $y_{40} = 0{,}00013292$, $y_{60} = 1{,}5325 \cdot 10^{-6}$,$y_{100} = 2{,}03704 \cdot 10^{-10}$, con $y_{100} - y(1) = -1{,}8574510^{-9}$, y el método de Euler converge con este paso.

Si nos damos cuenta, en general al aplicar un método Runge-Kutta a la ecuación (2.6.1), tendremos una fórmula tal que $y_{n+1} = R(\hat{h})y_n$, con lo que $y_n \to 0$ si y sólo si $|R(\hat{h})| < 1$ (tomando $\hat{h} = \lambda h$).

Para la fórmula de Euler explícita, $R(\hat{h}) = 1 - \lambda h = 1 - \hat{h}$, y por tanto, el método funcionará bien siempre que $\hat{h}$ esté en el círculo cuya circunferencia esta centradaen el punto (-1,0) y tiene como radio 1.

Todos los métodos Runge-Kutta explícitos de orden s y de s pasos (con s=1, 2, 3 y 4) tienen la misma función de estabilidad, y por tanto las mismas regiones de estabilidad. En las figuras (2.1) y (2.2) se dan las regiones de estabilidad absoluta para dichos métodos.

2.7. Selección de problemas.

1. Consideremos el siguiente problema de valor inicial :

$$y'(x) = 2x, \quad y(0) = 3, \tag{2.7.1}$$

1) Encuéntrese la solución analítica.
2) Escríbase un programa del Runge-Kutta de orden 2 ya dado (método de Heun), para resolver dicho P.V.I. con x variando entre 0 y 2.

Solución :

1) Integrando ambos miembros de la igualdad, llegamos a que la solución general es

$$y(x) = x^2 + c \tag{2.7.2}$$

(a) Orden 1 (region de estabilidad en gris). *(b) Orden 2 (en gris).*

Figura 2.1: *Región de Estabilidad absoluta de los métodos explícitos de 1 y 2 pasos.*

(c) Orden 3 (en gris). *(d) Orden 4 (en gris).*

Figura 2.2: *Región de Estabilidad absoluta de los métodos explícitos de 3 y 4 pasos.*

Como $y = 3$ cuando $x = 0$, la solución particular de nuestro problema es

$$y(x) = x^2 + 3 \tag{2.7.3}$$

2) Existen muchos lenguajes diferentes para programar, en todos ellos el pseudocódigo es similar, y tan sólo varían unas pocas líneas. Aquí vamos a programar en C++, Fortran y Mathematica.

El siguiente programa está hecho con el Mathematica:

$x0 = 0;$
$xf = 2;$
$h = 0{,}01;$
$n = (xf - x0)/h;$
$y0 = 2;$
$f[x\_, y\_] := 2 * x;$
$sol[x\_] := x^2 + 3;$
$For[j = 1, j <= n, j++,$
$k1 = h * f[x0, y0];$
$k2 = h * f[x0 + h, y0 + k1];$
$y0 = y0 + 1/2(k1 + k2);$
$x0 = x0 + h;$
$Print[x0];$
$Print[N[y0, 15]];$
$Print[N[y0 - sol[x0]]]];$

x_n	0.01	0.02	1.	2.
y_n	3.0001	3.0004	4.	7.
error	0.	$4{,}44089 \times 10^{-16}$	$-8{,}88178 \times 10^{-16}$	$-1{,}77636 \times 10^{-16}$

Utilizando el desarrollo de Taylor para esta función se puede deducir que en este ejemplo debería darse que $y_N = y(2)$. Si se comprueba realizando los cálculos ayudándose del programa Mathematica, podemos comprobar que $|y_N - y(2)| \approx 1{,}77636 \times 10^{-16}$. Lo cual no significa que nos hayamos equivocado. Este error, llamado error de la máquina es propio del programa. En el caso del Mathematica está comprobado que ronda 10^{-15}.

2. Consideremos el siguiente problema de valor inicial :

$$y'(x) = \sin(x), \quad y(0) = 5, \tag{2.7.4}$$

1) Utilizando el programa del ejercicio anterior, calcúlese el valor aproximado para $x = 2$ con paso constante $h = 0{,}1$.
2) Hágase el mismo cálculo pero con $h = 0{,}01$.
3) Compruébese el error ahora con $h = 0{,}001$.

Solución :

1) Utilizando el programa del ejercicio anterior, se puede obtener una tabla semejante a la siguiente:

x_n	0.1	0.2	1.	2.
y_n	5.0043916708	5.0199168082	5.493145488	6.4149665174
error	$-4{,}16389 \times 10^{-6}$	$-0{,}0000166$	$-0{,}0003831$	$-0{,}0011803$

El valor aproximado con $h = 0{,}1$ es $y_N = 6{,}4149665174$. Como la solución exacta del problema de valor inicial es $y(x) = -cos(x) + 6$, $y(2) = 6{,}4161468365$, la aproximación es buena en las dos primeras cifras decimales.

2) Obtenemos una tabla como la siguiente:

x_n	0.01	0.02	1.	2.
y_n	5.0000499991	5.0001999916	5.4596938633	6.4161350353
error	$-4{,}1666 \times 10^{-10}$	$-1{,}6666 \times 10^{-9}$	$-3{,}8308 \times 10^{-6}$	$-0{,}0000588$

El valor aproximado es $y_N = 6{,}4161350353$. Ahora son correctas las 4 primeras cifras decimales.

3) Obtenemos una tabla como la siguiente:

x_n	0.001	0.002	1.	2.
y_n	5.0000004999	5.0000199999	5.4596976558	6.4161467185
error	$-4{,}0856 \times 10^{-14}$	$-1{,}6608 \times 10^{-13}$	$-3{,}8308 \times 10^{-8}$	$-1{,}1804 \times 10^{-7}$

$y_N = 6{,}4161467185$, luego las 6 primeras cifras decimales son correctas. De aquí se podría llegar a la conclusión de que si tomamos un paso $h = 10^{-4}$, el error será inferior a 10^{-8} (4×2, donde 2 es el orden del método y 4 viene de la longitud de paso).

Aunque es un razonamiento correcto en este caso, en general no es cierto, pues hay que tener en cuenta el valor de las derivadas. En ejercicios posteriores se expondrán otras formas de llegar a expresiones del error.

3. Modifiquemos levemente el anterior problema de valor inicial :

$$y'(x) = \sin(\lambda x), \quad y(0) = y_0, \tag{2.7.5}$$

1) Utilizando el programa del ejercicio 1, calcúlese el valor aproximado para

$x = 2$ con pasos $h = 0{,}1$, $h = 0{,}01$ y $h = 0{,}001$, con $\lambda = 10$, $y_0 = 5{,}9$.
2) Hágase los mismos cálculos pero con $\lambda = 100$, $y_0 = 5{,}99$.

Solución :

1) Los resultados son los siguientes:

x_n	0.1	0.2	1.	2.
y_n	5.9420735492	6.0296119698	6.0683198926	5.9541749259
error	$-0{,}0038896$	$-0{,}012002$	$-0{,}01558$	$-0{,}005016$

x_n	0.01	0.02	1.	2.
y_n	5.9004991670	5.9019916808	6.06837538713	5.9591424591
error	$-4{,}1639 \times 10^{-7}$	$-1{,}6614 \times 10^{-6}$	$-0{,}0001532$	$-0{,}00004933$

x_n	0.001	0.002	1.	2.
y_n	5.9000049999	5.9000199991	6.06839056203	5.959192287
error	$-4{,}1665 \times 10^{-11}$	$-1{,}6666 \times 10^{-10}$	$-1{,}5325 \times 10^{-6}$	$-4{,}9326 \times 10^{-7}$

La solución exacta es $-\cos(10x)/10 + 6$ y ahora podemos observar que los errores son mayores a los del ejercicio anterior: con $h = 0{,}1$ el error es de 0.0050168, mientras que con $h = 0{,}01$ es de 0.00004933 y si $h = 0{,}001$, el error cuando $x = 2$ es de $4{,}93266 \times 10^{-7}$. Esto se debe a que las derivadas sucesivas $f'(x) = 10\cos(10x)$, $f''(x) = -100\text{sen}(10x)$, ..., crecen en valor absoluto, y por tanto el error también.

2) Los resultados son los siguientes:

x_n	0.1	0.2	1.	2.
y_n	5.96279844455	5.9812345144	5.9879636072	5.9824252745
error	$-0{,}04559$	$-0{,}01467$	$-0{,}003413$	$-0{,}01271$

x_n	0.01	0.02	1.	2.
y_n	5.9942073549	6.0029611969	5.9912601180	5.9946934833
error	$-0{,}0003896$	$-0{,}001200$	$-0{,}0001167$	$-0{,}0004364$

x_n	0.001	0.002	1.	2.
y_n	5.9900499167	5.9901991680	5.9913756637	5.995132397
error	$-4{,}1639 \times 10^{-8}$	$-1{,}6614 \times 10^{-7}$	$-1{,}1475 \times 10^{-6}$	$-4{,}274185 \times 10^{-6}$

Como era de esperar tras la explicación del apartado anterior, los errores vuelven a crecer, ahora son de 0.01271294 (para $h = 0{,}1$), 0.00043463 (si $h = 0{,}01$) y $4{,}274185 \times 10^{-6}$ ($h = 0{,}001$).

4. Consideremos el siguiente problema de valor inicial :

$$y'(x) = 2x\sqrt{y-1}, \quad y(0) = 2, \tag{2.7.6}$$

1) Calcúlese la solución analítica.
2) Escríbase un programa del método Runge-Kutta de orden tres ya propuesto para resolver la ecuación entre $0 \leq x \leq 2$.
3) Calcúlese el error del algoritmo para $h = 0{,}01$ después de 2 pasos.
4) Calcúlese el error del algoritmo para $h = 0{,}1$ después de 2 pasos.
5) Estímese el error de truncamiento usando $h = 0{,}01$ y $h = 0{,}02$.

Solución :

1) Es una ecuación de variables separables, por lo que procedemos de la siguiente forma
(a) $\sqrt{y-1} = 0$ no puede darse ya que $y(0) = 2$.
(b) si $\sqrt{y-1} \neq 0$ entonces, dividiendo por dicha función e integrando resulta

$$\int \frac{dy}{\sqrt{y-1}} = \int 2x dx \tag{2.7.7}$$

de donde obtenemos la solución implícita $2\sqrt{y-1} = x^2 + c$. La solución general por tanto es $y(x) = 1 + (x^2 + c)^2/4$. De donde obtenemos la solución particular de nuestro problema $y(x) = 1 + (x^2 + 2)^2/4$.

2) Esta vez hemos elegido C++ para escribir nuestro programa:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

void main(void)
 {
 double func (float x0, float y);
 double sol (float x0);
   int i,a,b,nstep;
   double h,x0,y0,k1,k2,k3;
   double f,g,t,s,er;
   clrscr ();
   printf ("\n Introduzca el paso h: ");
```

```
    scanf (" %lf ", &h);
    printf ("Introduzca valor inicial de x: ");
    scanf (" %d ", &a);
    x0=a;
    printf (" Introduzca valor final de x: ");
    scanf (" %d ",  &b);
    nstep=(b-x0)/h+1;
    printf (" Introduzca valor inicial de y: ");
    scanf (" %lf ", &y0);
    printf (" \n %d ", nstep);

 for ( i=0;i<2;i++)
  {
    f=func(x0,y0);
    k1=h*f;
    g=func(x0+h/2,y0+k1/2);
    k2=h*g;
    t=func(x0+h,y0-k1+2*k2);
    k3=h*t;
    y0=y0+(k1+4*k2+k3)/6.0;
    s=sol(x0+h);
    er=y0-s;
    x0=x0+h;
    printf("\n Valor de x: x= %lf",x0);
    printf("\n Valor del y: y= \%G",y0);
    printf("\n Valor del error: error=  %G",er);
  }
    getch ();
    return;
}


 double func (float x0, float y0)
{
   double g;
   g=2.0*x0*sqrt(y0-1.0);
   return g;
}
```

```
double sol (float x0)
{
   double g;
   g=1.0+(x0*x0+2.0)*(x0*x0+2.0)/4.0;
   return g;
}
```

Para esta longitud de paso, el error en la última iteración es de $-1{,}52195 \times 10^{-4}$. Si utilizamos un paso menor, por ejemplo $h = 0{,}01$, observamos que el error para $x = 2$ es del orden de $-1{,}32025 \times 10^{-7}$. En este caso resulta más complicado establecer una relación entre el número de cifras decimales correctas y la longitud de paso.

3) El error del algoritmo en los dos primeros pasos es de $8{,}35955 \times 10^{-10}$ para $x = 0{,}01$, y de $1{,}67327 \times 10^{-9}$ en el segundo paso.

4) Con h mayor, en este caso $h = 0{,}1$, el error aproximado crece y es de $8{,}16813 \times 10^{-6}$ para $x = 0{,}1$, y de $1{,}6048 \times 10^{-5}$ para $x = 0{,}2$.

En ambos casos, podemos observar que el error tras la segunda iteración es mayor que tras la primera, ya que y_2 se calcula a través de y_1, con lo que si $y(x_1)$ es aproximadamente $y_1 - E$, $y(x_2)$ suele estar próximo a $y_2 - 2E$, cuando $h \ll 1$.

5) Esta idea nos permite hallar una interesante manera de calcular aproximadamente el error de truncamiento : si tomamos $h_1 = 0{,}01$,
$y_1 \approx 2{,}00010000333 \approx y(x_1) + E$,
$y_2 \approx 2{,}00040004166 \approx y(2h_1) + 2E$;
pero si tomamos ahora $h_2 = 2h_1 = 0{,}02$, como el error de truncamiento tiene orden h^3, entonces ahora
$y_1 = 2{,}00040005332 \approx y(2h_1) + 8E$.
Por lo que
$6E \approx 2{,}00040005332 - 2{,}00040004166 \approx 1{,}165663 \times 10^{-8}$,
con lo que
$2E \approx 3{,}88554 \times 10^{-9}$.

Esto es, hemos valorado que el error de truncamiento en y(0.02), después de utilizar el Runge-Kutta de orden 3 con paso $h = 0{,}01$ y tras dos iteraciones es de $3{,}88554 \times 10^{-9}$. Aunque no es exacto, pues ya hemos visto que es del orden de $1{,}67327 \times 10^{-9}$, esta forma de estimación del error es muy realista en cuanto al número de cifras decimales correctas, que es lo que debe pedirse

a una estimación del error.

5. Encuéntrese todos los Runge-Kutta explícitos de orden menor o igual que 3 para el problema escalar.

Solución :

Ya hemos visto cómo se hace para hallar los Runge-Kutta de orden 2, nos disponemos a hacer lo mismo con los de 3 pasos, por tanto los podremos escribir como :
$k_1 = f(x_n, y_n)$
$k_2 = f(x_n + hc_2, y_n + hc_2k_1)$
$k_3 = f(x_n + hc_3, y_n + h(c_3 - a_{32})k_1 + ha_{32}k_2)$, pues $\sum_{i=1}^{3} a_{3i} = c_3$,
$y_{n+1} = y_n + h(b_1k_1 + b_2k_2 + b_3k_3)$.

Para hacer más sencillos los cálculos utilizaremos la siguiente notación
$f := f(x, y)$, $f_x := \frac{df}{dx}$, $f_{xx} := \frac{d^2f}{dx^2}$,$f_{xy} := \frac{d^2f}{dxdy}$

$\{f\} := f_x + ff_y$; $\{\{f\}\} := f_xf_y + ff_y^2$; $\{ff\} := f_{xx} + 2ff_{xy} + f^2f_{yy}$;

Entonces utilizando el desarrollo de Taylor en x_n, sabemos que
$y(x_{n+1}) = y(x_n) + hy'(x_n) + 1/2h^2y''(x_n) + 1/6h^3y'''(x_n) + O(h^4)$.

Además
$y' = f$,
$y'' = f_x + f_yy = f_x + ff_y$,
$y''' = f_{xx} + 2ff_{xy} + f^2f_{yy} + f_y(f_x + ff_y)$,
entonces nos queda que
$y(x_{n+1}) = y(x_n) + hf + 1/2h^2\{f\} + 1/6h^3(\{\{f\}\} + \{ff\}) + O(h^4)$

Como $k_1 = f$, y
$k_2 = f + hc_2(f_x + k1f_y) + 1/2h^2c_2^2(f_{xx} + 2k_1f_{xy} + k_1^2f_{yy}) + O(h^3) = f + hc_2\{f\} + 1/2h^2c_2^2\{ff\} + O(h^3)$,
$k_3 = f + hc_3\{f\} + h2(c_2a_{32}\{\{f\}\} + 1/2c_3^2\{ff\}) + O(h^3)$,
sustituyendo nos queda
$y_{n+1} = y(x_n) + h(b_1 + b_2 + b_3)f + h^2(b_2c_2 + b_3c_3)\{f\} + 1/2h^3[2b_3c_2a_{32}\{\{f\}\} + (b_2c_2^2 + b_3c_3^2)\{ff\}] + O(h^4)$.
De esta forma se puede conseguir un método de orden 3 satisfaciendo las siguientes condiciones :

$$\begin{cases} b_1 + b_2 + b_3 = 1 \\ b_2 c_2 + b_3 c_3 = 1/2 \\ b_2 c_2^2 + b_3 c_3^2 = 1/3 \\ b_3 c_2 a_{32} = 1/6 \end{cases} \tag{2.7.8}$$

Con lo que hemos obtenido un sistema de 4 ecuaciones con 6 incógnitas, de forma que las soluciones dependen de dos parámetros.

6. 1) Considerando el P.V.I.

$$y'(x) = e^{-x^2}, \quad y(0) = 1, \tag{2.7.9}$$

estímese y(2) utilizando el método de Heun y $h = 0{,}1$.
2) Calcúlese una cota máxima del error utilizando la solución del problema 4.
3) Estímese y(2), esta vez con $h = 0{,}001$.
4) Calcúlese ahora el intervalo donde estará y(2).
5) Estímese ahora el error que hubo con paso $h = 0{,}1$.

Solución :

1) Utilizando el programa dado en el ejercicio 1, sin más que cambiar que ahora no hay solución exacta, y por tanto no podemos evaluar el error, y modificando el valor inicial de la y, y el valor de la f(x,y); obtenemos el programa que nos permite aproximar la curva solución de nuestro problema de valor inicial.

En este caso $y_N = 1{,}88202044039556$.

2) Como ya hemos visto los métodos de dos pasos son los que tienen $b_3 = 0$, entonces

$y_{n+1} = y(x_n) + h(b_1 + b_2)f + h^2 b_2 c_2 \{f\} + 1/2h^3 b_2 c_2^2 \{ff\} + O(h^4)$,
con lo que el método de Heun ($b_1 = b_2 = 1/2$, $c_2 = 1$), tendrá
$y_{n+1} = y(x_n) + hf + 1/2h^2\{f\} + 1/4h^3\{ff\} + O(h^4)$.
Sin embargo ya hemos visto que
$y(x_{n+1}) = y(x_n) + hf + 1/2h^2\{f\} + 1/6h^3(\{\{f\}\} + \{ff\}) + O(h^4)$,
con lo cual
$y(x_{n+1}) - y_{n+1} =$
$= 1/6h^3(\{\{f\}\} + \{ff\}) - 1/4h^3\{ff\} =$
$= 1/6h^3\{\{f\}\} - 1/12h^3\{ff\} + O(h^4) =$

$= 1/6h^3 f_y(f_x + ff_y) - 1/12h^3(f_{xx} + 2ff_{xy} + f^2 f_{yy}) + O(h^4)$.

Como $f(x,y) = e^{-x^2}$, entonces
$|y(x_{n+1}) - y_{n+1}| \approx 1/12h^3|(4x^2-2)/e^{x^2}| \leq 1/12h^3 \times 2 = 1/6 \times 10^{-3}$,
ya que $\|(4x^2-2)/e^{x^2}\| \leq 2$ en el intervalo $0 \leq x \leq 2$. Con lo que podemos afirmar que las 3 primeras cifras decimales son correctas.

3) Ahora obtenemos que $y(2) \approx$ 1,88208138465722.

4) Además repitiendo los cálculos del apartado 3) podemos asegurar que $|y(2) - y_N| <= 1/6 \times 10^{-9}$, con lo que y(2) estará en el intervalo (1,882081384, 1,882081385).

5) Con $h =$ 0,1 obtuvimos un valor aproximado $y_N =$ 1,88202044039556, por lo que el error cometido fue aproximadamente 0.000060943, esto es las 4 primeras cifras decimales (y no tres) eran correctas.

De todas formas este método suele ser el elegido a la hora de estimar el error de truncamiento cometido, siempre y cuando h sea sensiblemente inferior a 1.

7. 1) Escríbase un programa del Runge-Kutta clásico de orden 4.
2) Resuélvase ahora el ejercicio 5.- 1) utilizando este último método.

Solución :

1) El problema que viene a continuación está hecho en Fortran:

```
c Entrada:la función f(x,y), la condición inicial y(x0)=y,el paso h
c y la solución de la ecuación y(x)=sol
c Salida: Una sucesión de puntos que son la aproximación de la
c solución de la ecuación diferencial, y el numero
c de puntos solución

      implicit double precision (a-h,o-z)
      h=0.1d0
      y=1.0d0
      x0=0.0d0
      xf=2.0d0
      nstep=dint((xf-x0)/h)+1
c Numero de pasos que debemos tomar
      do 10 i=1,nstep
      call rk4(x0,h,y)
```

```
      write (*,20) x0+h,y
20    format('x=',D12.7,' y=',D12.7)
      x0=x0+h
10    continue
      write(*,*) ' '
      write(*,*)'El numero de puntos es:',nstep+1
      stop
      end

      double precision function f(x,y)
      implicit double precision (a-h,o-z)
      f=dexp(-x*x) +0*y
      return
      end

      subroutine rk4(x0,h,y)
      implicit double precision (a-h,o-z)
      external f
      dimension pk(4)
      pk(1)=h*f(x0,y)
      pk(2)=h*f(x0+0.5d0*h,y+0.5d0*pk(1))
      pk(3)=h*f(x0+0.5d0*h,y+0.5d0*pk(2))
      pk(4)=h*f(x0+h,y+pk(3))
      y=y+(pk(1)+2d0*pk(2)+2d0*pk(3)+pk(4))/6.0d0
      return
      end
```

2) El resultado que se obtiene ahora es que $y_N \approx 1{,}88208136$, como ya sabemos que $y(2) \approx 1{,}882081384$, resulta que coinciden las 7 primeras cifras decimales, cuando $h = 0{,}1$ tan sólo.

8. 1) Consideremos el siguiente problema:

$$\begin{aligned} y'(x) &= -2xy + z, \quad y(0) = 0 \\ z'(x) &= -2xz - y, \quad z(0) = 1 \end{aligned} \tag{2.7.10}$$

1) Encuéntrese la solución analítica y aproxímense los valores y(1), z(1).
2) Escríbase un programa del Runge-Kutta de orden 4 para resolver este

sistema para x variando entre 0 y 1, y con $h = 0{,}1$.

Solución :

1) La solución analítica del sistema es

$$y(x) = e^{-x^2}\sin(x), z(x) = e^{-x^2}\cos(x) \tag{2.7.11}$$

Si calculamos la solución en $x = 1$, obtenemos $y(1) = 0{,}30955987$, $z(1) = 0{,}19876611$.

2) Volvemos a utilizar el programa Mathematica:

```
x0=0;
xf=1;
h=0.02;
n=(xf-x0)/h;
y0=0;
z0=1;
f[x_,y_,z_]:=-2*x*y+z;
g[x_,y_,z_]:=-2*x*z-y;
sol1[x_] := E^-x^2 * Sin[x];
sol2[x_] := E^-x^2 * Cos[x];
For[j = 1, j <= n, j + +,
k1=h*f[x0,y0,z0];
l1=h*g[x0,y0,z0];
k2=h*f[x0+h/2,y0+k1/2,z0+l1/2];
l2=h*g[x0+h/2,y0+k1/2,z0+l1/2];
k3=h*f[x0+h/2,y0+k2/2,z0+l2/2];
l3=h*g[x0+h/2,y0+k2/2,z0+l2/2];
k4=h*f[x0+h,y0+k3,z0+l3];
l4=h*g[x0+h,y0+k3,z0+l3];
y0=y0+1/6*(k1+2*k2+2*k3+k4);
z0=z0+1/6*(l1+2*l2+2*l3+l4);
x0=x0+h;
Print[x0];
Print[N[Abs[y0-sol1[x0]],15]];
Print[N[Abs[z0-sol2[x0]],15]]]
```

Con $h = 0{,}1$ el programa nos permite obtener ya una buena aproximación: el error en la primera componente es cercana a $3{,}32006 \times 10^{-6}$ y en la segunda componente es de $1{,}35185 \times 10^{-6}$.

Y si calculamos el error con $h = 0{,}001$, $3{,}14193 \times 10^{-14}$ (en la primera componente) y $6{,}60583 \times 10^{-15}$ (en la z), vemos que ya tenemos error de máquina con este problema.

9. 1) Escríbase un programa del Runge-Kutta-Fehlberg de orden 5.
2) Consideremos ahora el problema:

$$y'(x) = -100y, \quad y(0) = 10, \tag{2.7.12}$$

y queremos obtener una solución aproximada para x variando entre 0 y 5. Si colocamos cotas de $\delta = 10^{-8}$, $\delta = 10^{-5}$, ¿cómo evoluciona el paso h?

Solución :

1) Optamos por programar en Fortran esta vez:

```
      program Embedding
      implicit real*8 (a-h,o-z)
      dimension pk(6)
c El contador "ncont" nos da el numero final de pasos.
      ncont=0
c h=paso inicial que debe estar entre hmin y hmax.
      hmax=0.2d0
      hmin=1.0d-8
      h=hmax
c (x0,y) es la condicion inicial. (x0,xf) es el intervalo donde
c queremos hallar y(x).
      y=10.0d0
      x0=0.0d0
      xf=5.0d0
c Antes poniamos "nstep=dint((xf-x0)/h)" pero no se puede utilizar
c el numero de pasos porque al variar la h no se puede predecir
c nstep.

      write (*,*) 'x=',x0,'y=',y
c     Tolerancia=tol
      tol=1.0d-5
      flag=1.
      do while (flag.eq.1.)
```

```
      pk(1)=f(x0,y)
      pk(2)=f(x0+0.25d0+h,y+0.25*h*pk(1))
      pk(3)=f(x0+3d0/8d0*h,y+h/32d0*(3d0*pk(1)+9d0*pk(2)))
      pk(4)=f(x0+12d0/13d0*h,y+h/2197*(1932d0*pk(1)-7200d0*pk(2)
     1      +7296d0*pk(3)))
      pk(5)=f(x0+h,y+h*(439d0/216d0*pk(1)-8d0*pk(2)+
     1      3680d0/513d0*pk(3)-845d0/4104d0*pk(4)))
      pk(6)=f(x0+0.5d0*h,y+h*(-8d0/27d0*pk(1)+2d0*pk(2)
     1    -3544d0/2565d0*pk(3)+1859d0/4104d0*pk(4)-11d0/40d0*pk(5)))

      E=dabs(1./360*pk(1)-128./4275*pk(3)-2197./75240*pk(4)+
     1       1./50*pk(5)+2./55*pk(6) )

      write(*,*) 'E=',E

      if (E.eq.0) then
        q=1.
c  Para no dividir por cero la h=qh no se modifica.
        write (*,*) 'Error=0'
        pause
      end if
      q=0.84*(tol/E)**0.2
      if (E.le.tol) then
         x0=x0+h
         y=y+h*(25d0/216d0*pk(1)+0d0*pk(2)+1408d0/2565d0*pk(3)+
     1        2197d0/4104d0*pk(4)-1d0/5d0*pk(5)+0d0*pk(6))
         ncont=ncont+1
         write (*,*) 'x=',x0
         write(*,*) '           y=',y
         write(*,*) '                          err=',y-sol(x0)
         write(*,*) '                                      h=',h
         pause
      end if

      if (q.le.0.1) then
         h=0.1*h
      else if(q.ge.4) then
          h=4*h
```

```
      else
          h=q*h
      endif

      if (h.gt.hmax) then
          h=hmax
      endif

      if (x0.ge.xf) then
         flag=0.
      else if (x0+h.gt.xf) then
         h=xf-x0
      else if (h.lt.hmin) then
         flag=0.
         write (*,*) '!‘ERROR! Hemos excedido el hmin.'
      endif
   end do
   write (*,*) ncont
   stop
   end

   real*8 function f(x,y)
   implicit real*8 (a-h,o-z)
   f= -100*y+0*x
   return
   end

   real*8 function sol(x)
   implicit real*8 (a-h,o-z)
   sol= 10*dexp(-100*x)
   return
   end
```

2) Con $\delta = 10^{-8}$, se excede el paso mínimo en la primera iteración, luego es necesario reducir el valor de *hmin* o bien el δ.

Optamos por lo segundo, si ahora $\delta = 10^{-5}$, obtenemos la siguiente tabla:

número de iteraciones	1	2
paso h	$h = 4{,}39567 \times 10^{-4}$	$4{,}03338 \times 10^{-4}$
x	$h = 4{,}39567 \times 10^{-4}$	$8{,}42905 \times 10^{-4}$
error estimado	$6{,}42941 \times 10^{-6}$	$4{,}79819 \times 10^{-6}$
error verdadero	$-2{,}20348 \times 10^{-9}$	$-3{,}48295 \times 10^{-9}$

3	4	5
$3{,}92404 \times 10^{-4}$	$h = 3{,}90606 \times 10^{-4}$	$h = 3{,}92831 \times 10^{-4}$
$1{,}23531 \times 10^{-3}$	$1{,}23531 \times 10^{-3}$	$1{,}62591 \times 10^{-3}$
$4{,}27924 \times 10^{-6}$	$4{,}06502 \times 10^{-6}$	$3{,}96838 \times 10^{-6}$
$-4{,}49166 \times 10^{-9}$	$-5{,}39322 \times 10^{-9}$	$-6{,}24795 \times 10^{-9}$

iteraciones	260	261
paso h	$h = 2{,}78213 \times 10^{-2}$	$8{,}44478 \times 10^{-3}$
x	$h = 4{,}9915552$	5
error estimado	$5{,}18533 \times 10^{-6}$	$1{,}72301 \times 10^{-8}$
error verdadero	$-2{,}00702 \times 10^{-7}$	$-8{,}60573 \times 10^{-8}$
y_n	$-2{,}00702 \times 10^{-7}$	$-8{,}60573 \times 10^{-8}$

en la que observamos que en la primera iteración $h = 4{,}39567 \times 10^{-4}$, el error estimado E es de $6{,}42941 \times 10^{-6}$, y el error verdadero es de $-2{,}20348 \times 10^{-9}$. En la segunda iteración h se reduce ($4{,}03338 \times 10^{-4}$), y sigue disminuyendo hasta la cuarta iteración, donde $h = 3{,}90606 \times 10^{-4}$, aquí el error estimado $E = 4{,}06502 \times 10^{-6}$ y el error verdadero es $-5{,}39322 \times 10^{-9}$.

De hay en adelante el paso comienza a crecer hasta que en la última iteración $h = 8{,}44478 \times 10^{-3}$, ahí $y(5) = -8{,}60573 \times 10^{-8}$. El total de pasos efectuados es de 261, y el error verdadero en todos ellos es menor a 4×10^{-7}.

Toda esta información nos permite observar que los Runge-Kutta no respetan el principio del máximo, esto es, aunque la solución correcta $y(x) = 10e^{-100x}$ es positiva, es posible que en algún punto el valor de la aproximación sea negativa. La segunda conclusión que hemos obtenido es que los Runge-Kutta en realidad sólo tienen dificultades en estos problemas en las primeras iteraciones, donde las pendientes son muy elevadas, ahí es preciso reducir la h, después ya no es tan necesario.

10. Utilícese el programa del Runge-Kutta de orden 4 del ejercicio 7 para

aproximar y(0.001) en el problema

$$y'(x) = \frac{1}{2\sqrt{|x|}}, \quad y(9) = 3, \tag{2.7.13}$$

utilizando paso constante $h = -0{,}001$.

Solución :

La dificultad que aquí se nos presenta no es la utilización de un paso negativo, como podría parecer, sino el hecho de que la función $y(x) = \sqrt{|x|}$, tiene una singularidad en el punto $x = 0$, y las aproximaciones pierden precisión. De hecho el error que se comete para este problema es aproximadamente de $6{,}977 \times 10^{-6}$, que podría parecer pequeño, pero no lo es tanto pues utilizamos un Runge-Kutta de orden 4 con paso pequeño.

Para observarlo mejor tomemos $h = -10^{-4}$, y aproximamos $y(10^{-4})$ para el anterior problema (2.7.13), ahora el error es de $2{,}364 \times 10^{-6}$, apenas un tercio del anterior, aunque el paso sea 10 veces menor.

11. 1) Calcúlese la solución analítica del siguiente P.V.I.

$$y'(x) = \sin(x) + 10e^{-10x}, \quad y(0) = 0, \tag{2.7.14}$$

2) Utilícese el programa del Runge-Kutta-Fehlberg de orden 5 del ejercicio 9 para aproximar y(1) en dicho problema, utilizando paso variable con $10^{-4} \leq h \leq 0{,}1$, y tolerancia máxima permitida $\delta = 10^{-8}$.
3) Utilícese ahora el programa del Runge-Kutta-Fehlberg de orden 5 para aproximar y(1) en

$$y'(x) = \sin(x) + 1000e^{-1000x}, \quad y(0) = 0, \tag{2.7.15}$$

utilizando paso variable con $10^{-4} \leq h \leq 0{,}1$, y tolerancia máxima permitida $\delta = 10^{-8}$.

Solución :

1) Nos basta integrar ambos miembros de la igualdad para saber que $y(x) = -\cos(x) - e^{-10x} + c$, y como $y(0) = 0$, entonces $y(x) = -\cos(x) - e^{-10x} + 2$.

2) En este caso el programa requiere 50 pasos de tamaños que varían entre 10^{-2} y $6{,}3 \times 10^{-2}$, para conseguir errores que rondan 10^{-10} y 4×10^{-9}.

3) Aquí se excede el paso mínimo en la primera iteracíón, necesitamos reducir δ hasta 10^{-3} para obtener resultados, el primer $h \approx 4{,}8 \times 10^{-4}$, aunque pronto h alcanza hmax y de hecho se halla la aproximacíón en 25 pasos. En este problema, si tomamos $\lambda e^{-\lambda x}$, y λ lo cambiamos por valores cada vez más altos, observamos que las aproximaciones son cada vez más erróneas, se debe a lo que se denomina rigidez del problema. En general hay pocos métodos que funcionen bien con problemas muy rígidos, aunque cada vez hay más estudios en este campo.

Capítulo 3

Métodos Multipaso

3.1. Preliminares:

En general llamaremos métodos multipaso a aquellos métodos de integración numérica que se pueden escribir como

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} = h(\beta_0 f_n + \beta_1 f_{n-1} + ... + \beta_k f_{n-k}) \quad (3.1.1)$$

donde k es un entero fijo que representa el número de pasos del método, $f_i = f(x_i, y_i)$, y α_i, β_i, son constantes que no dependen de n; y_n es la aproximación a $y(x_n)$ que se quiere calcular en cada iteración, mientras que los y_{n-i}, con $0 < i$ son valores que se han obtenido previamente.

Como la mayoría de los métodos multipaso están basados en la interpolación, daremos unas nociones previas de los polinomios de interpolación de Lagrange y de Newton y de sus propiedades.

Sea f una función de variable real de clase C^{n+1} en un intervalo [a,b], y $\{x_0 = a, x_1, ..., x_n = b\}$ un conjunto de puntos de dicho intervalo donde conocemos el valor de la f.

Es fácil demostrar que existe un único polinomio P(x) de grado menor o igual que n tal que

$$P(x_i) = f(x_i) = y_i, \qquad i = 1, ..., n \quad (3.1.2)$$

ya que si existiera otro polinomio Q(x), entonces P(x)-Q(x) es un polinomio de grado menor o igual que n, pero con n+1 raíces, llegando a contradicción. Además no es difícil encontrar dicho polinomio:

$$P(x) = L(x) \sum_{i=0}^{n} \frac{y_i}{L'(x_i)(x - x_i)} \quad (3.1.3)$$

donde

$$L(x) = \prod_{i=0}^{n}(x - x_i). \tag{3.1.4}$$

Dicho polinomio es definido como el polinomio interpolador de grado n de f en los puntos $\{x_0, ..., x_n\}$

Además sin más que utilizar el teorema de Rolle se llega rápidamente al siguiente resultado:

Teorema: Sea f una función de clase C^{n+1} en el intervalo [a,b], P(x) el polinomio interpolador de f en los puntos $\{x_0, ..., x_n\}$. Entonces existe un $c \in [a, b]$, tal que

$$f(x) - P(x) = \frac{f^{(n+1)}(c)}{(n+1)!}L(x), \forall x \in [a, b] \tag{3.1.5}$$

Sin embargo, este método para calcular el polinomio de interpolación adolece de ineficiencia, ya que el polinomio de n puntos no guarda relación con los de n+i puntos.

Mucho más práctico es el método de Newton: sea $f(x)$ cuyos valores son conocidos en $x_0, \ldots, x_n$. Se define la diferencia dividida en dos puntos x_0, x_1 como

$$f[x_0, x_1] = \frac{f(x_0) - f(x_1)}{x_0 - x_1},$$

y la diferencia dividida en n+1 puntos $x_0, \ldots, x_n$ como

$$f[x_0, \ldots, x_n] = \frac{f[x_0, \ldots, x_{n-1}] - f[x_1, \ldots, x_n]}{x_0 - x_n}.$$

Si ahora escribimos

$$f[x, x_0, \ldots, x_n] = \frac{f[x, \ldots, x_{n-1}]}{x - x_n} - \frac{f[x_0, \ldots, x_n]}{x - x_n}$$

y seguimos desarrollando

$$f[x, \ldots, x_{n-1}] = \frac{f[x, \ldots, x_{n-2}]}{x - x_{n-1}} - \frac{f[x_0, \ldots, x_{n-1}]}{x - x_{n-1}}, \ldots$$

$$\ldots, f[x, x_0, x_1] = \frac{f[x, x_0]}{x - x_1} - \frac{f[x_0, x_1]}{x - x_1},$$

$$f[x, x_0] = \frac{f(x)}{x - x_0} - \frac{f(x_0)}{x - x_0}.$$

Y ahora sustituyendo obtenemos

$$f[x, x_0, \ldots, x_n] = -\frac{f[x_0, \ldots, x_n]}{x - x_n} - \frac{f[x_0, \ldots, x_{n-1}]}{(x - x_{n-1})(x - x_{n-2})} - \ldots$$

$$\ldots - \frac{f(x_0)}{(x - x_n)\ldots(x - x_0)} + \frac{f(x)}{(x - x_n)\ldots(x - x_0)}.$$

Llegando así a la siguiente igualdad:

$$f(x) = f(x_0) + f[x_0, x_1](x - x_0) + \ldots \quad (3.1.6)$$
$$\ldots + f[x_0, \ldots, x_n](x - x_0)\ldots(x - x_{n-1}) + f[x, x_0, \ldots, x_n]L(x),$$

y si llamamos

$$P_n(x) := f(x_0) + f[x_0, x_1](x - x_0) + \ldots + f[x_0, \ldots, x_n](x - x_0)\ldots(x - x_{n-1}), \quad (3.1.7)$$

se verifica que

$$f(x_i) = f(x_0) + f[x0, x1](x_i - x_0) + \ldots \quad (3.1.8)$$
$$\ldots + f[x_0, \ldots, x_n](x_i - x_0)\ldots(x_i - x_{n-1}), \forall i = 0, \ldots, n$$

y además $P_n(x)$ es un polinomio de grado n, por lo que $P_n(x)$ es el polinomio interpolador de $f(x)$ en los puntos $x_0, \ldots, x_n$.

Sin embargo, el conjunto de puntos $\{x_0, ..., x_n\}$ puede tomarse idénticamente espaciados, ya que los métodos obtenidos son menos costosos de calcular, y los errores suelen ser más pequeños. Esto es, consideraremos $x_i = x_0 + ih$.

Ahora si definimos $\nabla^0 y_p = y_p$, $\nabla^1 y_p = y_p - y_{p-1}$, ... $\nabla^q y_p = \nabla(\nabla^{q-1} y_p)$, podemos observar que
$y_p = \nabla^0 y_p$,
$y_{p-1} = y_p - (y_p - y_{p-1}) = \nabla^0 y_p - \nabla^1 y_p$
y en general

$$y_{p-j} = \sum_{i=0}^{j} (-1)^i \binom{j}{i} \nabla^i y_p \quad (3.1.9)$$

y así podemos probar que

$$P(x) = \sum_{i=0}^{j} (-1)^i \binom{-s}{i} \nabla^i y_n \quad (3.1.10)$$

donde $s = \frac{x-x_n}{h}$, y P(x) es el polinomio interpolador de f(x) en $\{x_0, ..., x_n\}$, pues $\binom{-s}{i}$ es un polinomio de grado menor o igual que n, y que para $m = 0, ..., n$, cumple que

$$P(x_{n-m}) = \sum_{i=0}^{j} (-1)^i \binom{m}{i} \nabla^i y_n = y_{n-m} \tag{3.1.11}$$

considerando siempre que $\binom{m}{i} = 0$, si $m < i$.

3.2. Métodos Adams-Bashforth :

Este tipo de métodos se deducen inmediatamente de la identidad

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(\xi, y(\xi)) d\xi, \tag{3.2.1}$$

donde el valor f(x,y(x)) es desconocido, y por tanto deberá ser aproximado, para lo cual utilizaremos la teoría del anterior capítulo.

De esta forma

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(\xi, y(\xi)) d\xi \approx \int_{x_n}^{x_{n+1}} P(x) dx \tag{3.2.2}$$

donde P(x) será el polinomio interpolador de $f(x, y(x))$ en los puntos $\{x_{n-q}, ..., x_n\}$, así

$$y(x_{n+1}) - y(x_n) \approx \int_{x_n}^{x_{n+1}} P(x) dx = h \sum_{i=0}^{q} \gamma_i \nabla^i f_n \tag{3.2.3}$$

donde las constantes γ_i salen de calcular

$$\gamma_i = (-1)^i \frac{1}{h} \int_{x_n}^{x_{n+1}} \binom{-s}{i} dx = (-1)^i \int_0^1 \binom{-s}{i} ds. \tag{3.2.4}$$

De esta manera dadas $y_{n-q}, ..., y_n$, se puede calcular y_{n+1}, y después sucesivamente y_{n+2}, y_{n+3}, ...

Para facilitar los cálculos de los coeficientes γ_i, se suele utilizar la función generatriz de dichos coeficientes $G(t) = -\frac{t}{(1-t)\log(1-t)}$, pues

$$G(t) = \sum_{n=0}^{\infty} \gamma_n t^n = \sum_{n=0}^{\infty} (-t)^n \int_0^1 \binom{-s}{n} ds = \tag{3.2.5}$$

$$\int_0^1 \sum_{n=0}^{\infty} (-t)^n \binom{-s}{n} ds = \int_0^1 (1-t)^{-s} ds = \frac{-t}{(1-t)\log(1-t)}$$

(ya que podemos considerar que $|t| < 1$, pues vamos a calcular los coeficientes de G(t) a partir del desarrollo de G(t) en un entorno del 0).

De esta forma se hallan de forma recursiva los γ_i, que resultan ser

i	0	1	2	3	4	5	6
γ_i	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$
i	7	8	9	10	11	12	
γ_i	$\frac{5257}{17280}$	$\frac{1070017}{3628800}$	$\frac{25713}{89600}$	$\frac{26842253}{95800320}$	$\frac{4777223}{17418240}$	$\frac{703604254357}{703604254357}$	

Sobre la fórmula (3.2.3) se pueden intoducir mejoras, sin más que desarrollar $\nabla^i f_n$, de tal forma que nos quede

$$y_{n+1} - y_n = h \sum_{i=0}^{q} \gamma_{q,i} f_{n-i}, \tag{3.2.6}$$

donde ahora el subíndice de $\gamma_{q,i}$ se explica porque los coeficientes dependen tanto de q como de i.

La siguiente tabla nos presenta los valores de dichos coeficientes con q variando entre 0 y 5

i	0	1	2	3	4	5
$\gamma_{0,i}$	1	0	0	0	0	0
$\gamma_{1,i}$	$\frac{3}{2}$	$\frac{-1}{2}$	0	0	0	0
$\gamma_{2,i}$	$\frac{23}{12}$	$\frac{-16}{12}$	$\frac{5}{12}$	0	0	0
$\gamma_{3,i}$	$\frac{55}{24}$	$\frac{-59}{24}$	$\frac{37}{24}$	$\frac{-9}{24}$	0	0
$\gamma_{4,i}$	$\frac{1901}{720}$	$\frac{-2774}{720}$	$\frac{2616}{720}$	$\frac{-1274}{720}$	$\frac{251}{720}$	0
$\gamma_{5,i}$	$\frac{4227}{1440}$	$\frac{-7673}{1440}$	$\frac{9482}{1440}$	$\frac{-6798}{1440}$	$\frac{2627}{1440}$	$\frac{-425}{1440}$

Otro aspecto importante en la elaboración de los métodos multipaso es la inicialización: en cada iteración de un multipaso, para calcular y_{n+1} es necesario tener los valores de $y_n, \ldots, y_{n-k}$, pero en la mayoría de los casos en la primera iteración del método sólo disponemos del valor de y_0, por tanto habrá que calcular $y_1, \ldots, y_k$. Uno de los métodos más habituales en todos ellos es utilizar el Runge-Kutta de orden el mismo que el número de pasos del multipaso.

3.3. Métodos Adams-Moulton :

Este tipo de métodos se deducen de forma parecida a los Adams-Basforth, sólo que ahora se utiliza el polinomio interpolador en los puntos $\{x_{n-q}, ..., x_n\}$,

pero se integra en el intervalo $[x_{n-1}, x_n]$

$$y(x_n) - y(x_{n-1}) = \int_{x_{n-1}}^{x_n} f(\xi, y(\xi))d\xi, \tag{3.3.1}$$

y por tanto ahora

$$y_n - y_{n-1} = \int_{x_{n-1}}^{x_n} P(x)dx = h\sum_{i=0}^{q} \gamma_i^* \nabla^i f_n \tag{3.3.2}$$

donde ahora

$$\gamma_i^* = (-1)^i \frac{1}{h} \int_{x_{n-1}}^{x_n} \binom{-s}{i} dx = (-1)^i \int_{-1}^{0} \binom{-s}{i} ds \tag{3.3.3}$$

Y al igual que con los métodos Adams-Bashforth, se calcula la función generatriz cuyo desarrollo de Taylor en el origen da lugar a los γ_i^*, en este caso:

$$G(t) = \sum_{n=0}^{\infty} \gamma_n^* t^n = \sum_{n=0}^{\infty} (-t)^n \int_{-1}^{0} \binom{-s}{n} ds = \tag{3.3.4}$$

$$\int_{-1}^{0} \sum_{n=0}^{\infty} (-t)^n \binom{-s}{n} ds = \int_{-1}^{0} (1-t)^{-s} ds = \frac{-t}{\log(1-t)}.$$

Y se pueden hallar de forma recursiva los γ_i^*, que resultan ser

i	0	1	2	3	4	5	6
γ_i^*	1	$\frac{-1}{2}$	$\frac{-1}{12}$	$\frac{-1}{24}$	$\frac{-19}{720}$	$\frac{-3}{160}$	$\frac{-863}{60480}$
i	7	8	9	10	11	12	
γ_i^*	$\frac{-275}{24192}$	$\frac{-33953}{3628800}$	$\frac{-8183}{1036800}$	$\frac{-3250433}{479001600}$	$\frac{-4671}{788480}$	$\frac{-13695779093}{2615348736000}$	

Hasta aquí ambos métodos son muy parecidos, sin embargo existe una gran diferencia entre los Adams-Basforth y los Adams-Moulton, y es que de (3.3.2) no es fácil determinar y_n, ya que lo determinamos por $f_n = f(x_n, y_n)$, con lo que el método resulta implícito.

Un mecanismo muy común en estos casos es utilizar la fórmula llamada Predictor-Corrector con los métodos Adams-Basforth y los Adams-Moulton respectivamente.

Esto es, en cada iteración se realizan los siguientes pasos:

1) necesitamos estimar una primera vez y_n, que denotaremos como $y_n^{(0)}$, para ello utilizamos el Adams-Basforth (P),

2) a continuación estimamos por primera vez el valor de f_n, que denotaremos como $f_n^{(0)} = f(x_n, y_n^{(0)})$ (E),

3) ahora usamos el Corrector, cuyo error es menor (con h suficiéntemente pequeña), para estimar $y_n^{(1)}$ (C)

4) por último se estima $f_n^{(1)} = f(x_n, y_n^{(1)})$, ya que este valor es necesario en la siguiente iteración (E)

Este esquema, es denominado PECE. En general todos ellos tienen la forma $P(EC)^n E^s$, con $s = 0$, ó bien $s = 1$.

Lo más habitual es utilizar el Adams-Bashforth del mismo número de pasos como Corrector, pero también se puede utilizar un Runge-Kutta del mismo orden que el número de pasos del Adams-Moulton.

Al igual que ya explicamos con los Adams-Bashforth, en los Adams-Moulton también es fundamental la inicialización, ya que muchas veces es origen de gran parte del error que se origina. También lo más habitual en este caso es utilizar el Runge-Kutta correspondiente.

Aparéntemente este método es mucho más costoso, y pudiera parecer que el error será mayor, ya que en cada iteración se producen varias estimaciones, y pudiera parecer que el error en cada iteración va ser la suma de todos ellos. Por el contrario el error con este tipo de esquemas suele ser bastante más pequeño, ya que el orden del error del corrector es en una unidad mayor que la del predictor (considerando los Adams-Bashforth y Adams-Moulton con la misma q). Y aunque en un esquema $PECE$ el orden del error no es exactamente el del corrector, con un esquema $P(EC)^n E$, con $n \geq 2$, ya se tiene que el orden del esquema es el del Adams-Moulton.

Al igual que con los métodos Adams-Bashforth, se puede desarrollar $\nabla^i f_n$ de (3.2.3), de tal forma que nos quede

$$y_n - y_{n-1} = h \sum_{i=0}^{q} \gamma_{q,i}^* f_{n-i}, \tag{3.3.5}$$

la siguiente tabla nos presenta los valores de dichos coeficientes con q variando entre 0 y 5

i	0	1	2	3	4	5
$\gamma^*_{0,i}$	1	0	0	0	0	0
$\gamma^*_{1,i}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
$\gamma^*_{2,i}$	$\frac{5}{12}$	$\frac{8}{12}$	$\frac{-1}{12}$	0	0	0
$\gamma^*_{3,i}$	$\frac{9}{24}$	$\frac{19}{24}$	$\frac{-5}{24}$	$\frac{1}{24}$	0	0
$\gamma^*_{4,i}$	$\frac{251}{720}$	$\frac{646}{720}$	$\frac{-264}{720}$	$\frac{106}{720}$	$\frac{-19}{720}$	0
$\gamma^*_{5,i}$	$\frac{475}{1440}$	$\frac{1427}{1440}$	$\frac{-798}{1440}$	$\frac{482}{1440}$	$\frac{-173}{1440}$	$\frac{27}{1440}$

3.4. Fórmulas en diferencias regresivas (BDF):

Para deducir los métodos BDF lo primero que consideramos es el polinomio interpolador de la función $y(x)$ en el conjunto de puntos $\{x_{n-q}, ..., x_n\}$, que será

$$P(x) = \sum_{i=0}^{q} (-1)^i \binom{-s}{i} \nabla^i y_n \tag{3.4.1}$$

donde s sigue siendo $\frac{x-x_n}{h}$. Como nosotros queremos interpolar $y'(x)$, vamos a derivar y luego evaluar en x_{n-1} , de forma que quede un método explícito:

$$f_{n-1} = P'(x_{n-1}) = \frac{1}{h} \sum_{i=0}^{q} (-1)^i h \frac{d}{dx} \binom{-s}{i}_{x=x_{n-1}} \nabla^i y_n \tag{3.4.2}$$

Con lo que obtenemos el siguiente método:

$$\sum_{i=0}^{q} \delta_{1,i} \nabla^i y_n = h f_{n-1} \tag{3.4.3}$$

donde

$$\delta_{1,i} = (-1)^i h \frac{d}{dx} \binom{-s}{i}_{x=x_{n-1}} = (-1)^i \frac{d}{ds} \binom{-s}{i}_{s=-1} \tag{3.4.4}$$

Otra posibilidad es calcular el método implícito, derivando en x_n, es decir, considerar

$$f_n = P'(x_n) = \frac{1}{h} \sum_{i=0}^{q} (-1)^i h \frac{d}{dx} \binom{-s}{i}_{x=x_n} \nabla^i y_n, \tag{3.4.5}$$

obteniendo el método:

$$\sum_{i=0}^{q} \delta_{0,i} \nabla^i y_n = h f_n \tag{3.4.6}$$

donde

$$\delta_{0,i} = (-1)^i h \frac{d}{dx}\binom{-s}{i}_{x=x_n} = (-1)^i \frac{d}{ds}\binom{-s}{i}_{s=-1} \tag{3.4.7}$$

Sin embargo, los métodos explícitos tienen grandes problemas de estabilidad y no se suelen utilizar.

Las fórmulas implícitas suelen funcionar mucho mejor, de hecho los métodos son estables hasta $q = 6$, lo que nos proporciona una convergencia bastante buena, sin embargo al igual que con los métodos Adams-Moulton, de nuevo necesitamos un esquema que en cada iteración nos proporcione la primera iteración de f_n.

Para hallar el valor de los coeficientes, podemos considerar que esta vez conocemos $\delta_{r,i}$, cuando $i > r$, pues

$$\delta_{r,i} = \frac{d}{ds}\binom{s+i-1}{i}_{s=-r} = \tag{3.4.8}$$

$$\frac{1}{i!}\lim_{s\to -r} s...(s+r-1)(s+r+1)...(s+i-1) = (-1)^r \frac{r!(i-r-1)!}{m!}$$

Mientras que cuando $i < r$, es mejor el método que venimos realizando, considerar la función generatriz de los coeficientes, y obtener a partir de ella una fórmula de recurrencia, así:

$$D(t) = \sum_{n=0}^{\infty} \delta_{r,n} t^n = \sum_{n=0}^{\infty} (-t)^n \frac{d}{ds}\binom{-s}{n}_{s=-r} =$$

$$\frac{d}{ds}\left(\sum_{n=0}^{\infty}(-t)^n\binom{-s}{n}\right)_{s=-r} = \frac{d}{ds}(1-t)^{-s}|_{s=-r} = \tag{3.4.9}$$

$$\frac{d}{ds}e^{-s\log(1-t)}|_{s=-r} = -\log(1-t)(1-t)^r$$

Y fácilmente se obtiene la siguiente tabla con los valores de los coeficientes:

n	0	1	2	3	4	5	6
r=0	0	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$
r=1	0	1	$-\frac{1}{2}$	$-\frac{1}{6}$	$-\frac{1}{12}$	$-\frac{1}{20}$	$-\frac{1}{30}$

3.5. Convergencia de los métodos multipaso:

Como ya hemos visto, la idea principal tanto de los métodos Runge-Kutta como de los multipaso es que dado el problema de valor inicial

$$y'(x) = f(x, y), \quad y(x_0) = y_0 \tag{3.5.1}$$

entonces cuando $h \to 0$ y para $x_n = x$, se cumpla que

$$\lim_{h\to 0} y_n = y(x), \tag{3.5.2}$$

esto es lo que exigimos para decir que un método sea convergente.

Pero además es importante cómo se produce esta convergencia, tanto por el orden, como por las constantes del error.

En general, todos los métodos multipaso se pueden escribir como

$$\alpha_0 y_n + \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} = h(\beta_0 f_n + \beta_1 f_{n-1} + ... + \beta_k f_{n-k}) \tag{3.5.3}$$

donde k es un entero fijo, $f_i = f(x_i, y_i)$, y las α_i, β_i, son constantes que no dependen de n; y_n es el valor que se quiere calcular en cada iteración, mientras que los y_{n-i}, con $0 < i$ son valores que se han obtenido previamente, y como queremos queremos hallar $y(x-ih)-y_{n-i}$, en la fórmula (3.5.3) reemplazamos las y_{n-i}, por las $y(x - ih)$, y las $hf_{n-i} = hy'_{n-i}$, por $hy'(x_{n-i})$. De esta forma obtenemos

$$\alpha_0 y(x_n)+\alpha_1 y(x_{n-1})+...+\alpha_k y(x_{n-k})-h(\beta_0 y'(x_n)+\beta_1 y'(x_{n-1})+...+\beta_k y'(x_{n-k})) \tag{3.5.4}$$

Utilizando el desarrollo de Taylor de las funciones $y(x - ih)$, e $y'(x - ih)$, en un entorno de $x = x_n$ vamos a obtener una expresión del tipo

$$C_0 y(x) + C_1 hy'(x) + ... + C_k h^k y^{(k)}(x) + ... \tag{3.5.5}$$

suponiendo que $C_0 = C_1 = ... = C_{k-1} = 0$ y que $C_k \neq 0$, en ese caso el error producido con dicho método es $C_k h^k y^{(k)}(x) + O(h^{k+1})$, que para $0 < h \ll 1$, es aproximadamente $C_k h^k y^{(k)}(x)$.

Si ahora consideramos la fórmula (3.5.3), con las constantes descritas para los métodos Adams-Bashforth, Adams-Moulton y BDF y hacemos el desarrollo hasta aquí descrito llegamos en general a la siguiente tabla de errores

Método	Orden del método	Cte. del error	Número de pasos
Adams-Bashforth	k	γ_k	k
Adams-Moulton	k+1	γ^*_{k+1}	k
BDF explícito	k	$\delta_{1,k+1}$	k
BDF implícito	k	$\delta_{0,k+1}$	k

donde la constante del error es la primera C_i no nula en la expresión $C_0y(x)+C_1hy'(x)+...+C_kh^ky^{(k)}(x)+...$, (en este tipo de métodos las C_k son constantes debido a que el desarrollo de Taylor tanto de $y(x-ih)$, como de $y'(x-ih)$, en un entorno de $x=x_n$; no depende más que de h^j, de $y^{(k)}(x)$ y de una serie de constantes).

3.6. Métodos de Störmer :

Hasta ahora hemos tratado con métodos para tratar las ecuaciones diferenciales $y'(x)=f(x,y(x))$. Los métodos de Störmer y de Cowell, que se presentan a continuación son muy útiles para integrar las ecuaciones $y''=f(x,y(x))$, un caso particular de $y''(x)=f(x,y(x),y'(x))$ muy interesante en astronomía por ser frecuente en problemas mecánicos sin disipación, o también en mecánica cuántica para calcular energias.

Entonces si consideramos

$$y''(x)=f(x,y(x)) \tag{3.6.1}$$

integrando dos veces tenemos

$$y(x+h)-y(x)=hy'(x)+\int_x^{x+h}(x+h-t)f(t,y(t))dt \tag{3.6.2}$$

Si repetimos el proceso pero ahora con $-h$ en vez de h, y sumando ambas expresiones observamos que ahora no hay término con primera derivada:

$$\begin{aligned} y(x+h)-2y(x)+y(x-h) &= \int_x^{x+h}(x+h-t)f(t,y)dt+ \\ \int_x^{x-h}(x-h-t)f(t,y)dt &= \int_x^{x+h}(x+h-t)\{f(t,y)+f(2x-t,y)\}dt \end{aligned} \tag{3.6.3}$$

Ahora utilizamos los polinomios de interpolación de f(t,y(t)) en los puntos $\{\, x_{n-q},...,x_n \,\}$, de esta forma integrando (3.6.3) entre $x=x_n$ y $x+h=x_{n+1}$,

se obtiene

$$y_{n+1} - 2y_n + y_{n-1} = h^2 \sum_{i=0}^{q} \rho_i \nabla^i f_n \tag{3.6.4}$$

donde ahora

$$\rho_i = \frac{(-1)^i}{h^2} \int_{x_n}^{x_{n+1}} (x_{n+1} - x) \left[\binom{-s}{i} + \binom{s}{i} \right] dx = (-1)^i \int_0^1 (1-s) \left[\binom{-s}{i} + \binom{s}{i} \right] dx \tag{3.6.5}$$

donde s sigue siendo $\frac{x-x_n}{h}$.

Igual que hasta ahora los coeficientes se calculan por su función generatriz, la cual se calcula de forma similar a como se viene haciendo, sin más que intercambiar el sumatorio por la integral tenemos que

$$G(t) = \sum_{n=0}^{\infty} \rho_n t^n = \left[\frac{t}{\log(1-t)} \right]^2 \frac{1}{1-t} \tag{3.6.6}$$

De donde se obtiene la siguiente tabla de coeficientes:

i	0	1	2	3	4	5	6
ρ_i	1	0	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{19}{240}$	$\frac{3}{40}$	$\frac{863}{12096}$
i	7	8	9	10	11	12	
ρ_i	$\frac{275}{4032}$	$\frac{33953}{518400}$	$\frac{8183}{129600}$	$\frac{3250433}{53222400}$	$\frac{4671}{78848}$	$\frac{13695779093}{237758976000}$	

Por último se puede observar que para q=0,1; se obtiene la fórmula ya conocida de

$$y_{n+1} - 2y_n + y_{n-1} = h^2 f_n \Leftrightarrow \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} \approx f(x, y(x)). \tag{3.6.7}$$

3.7. Métodos de Cowell :

Los métodos de Cowell también parten de la igualdad

$$\begin{aligned} y(x+h) - 2y(x) + y(x-h) &= \int_x^{x+h} (x+h-t) f(t,y) dt + \\ \int_x^{x-h} (x-h-t) f(t,y) dt &= \int_x^{x+h} (x+h-t) \{ f(t,y) + f(2x-t,y) \} dt \end{aligned} \tag{3.7.1}$$

y al igual que antes se utilizan los polinomios de interpolación de f(t,y(t)) en los puntos $\{ x_{n-q}, ..., x_n \}$, la única diferencia es que esta vez se integra entre $x = x_{n-1}$ y $x + h = x_n$, con lo que ahora se obtiene

$$y_n - 2y_{n-1} + y_{n-2} = h^2 \sum_{i=0}^{q} \rho_i^* \nabla^i f_n \tag{3.7.2}$$

donde ahora

$$\rho_i^* = \frac{(-1)^i}{h^2} \int_{x_{n-1}}^{x_n} (x_n - x) \left[\binom{-s}{i} + \binom{s+2}{i} \right] dx = \tag{3.7.3}$$
$$(-1)^i \int_{-1}^{0} (-s) \left[\binom{-s}{i} + \binom{s+2}{i} \right] dx$$

donde s sigue siendo $\frac{x-x_n}{h}$.

Se determina la función generatriz de los coeficientes y nos queda que

$$G^*(t) = \sum_{n=0}^{\infty} \rho_n^* t^n = \left[\frac{t}{\log(1-t)} \right]^2 \tag{3.7.4}$$

además de la relación $G^*(t) = (1-t)G(t)$, es muy fácil obtener los coeficientes. Obtenemos la siguiente tabla de valores

i	0	1	2	3	4	5	6
ρ_i^*	1	-1	$\frac{1}{12}$	0	$\frac{-1}{240}$	$\frac{-1}{240}$	$\frac{-221}{60480}$
i	7	8	9	10	11	12	
ρ_i^*	$\frac{-19}{6048}$	$\frac{-9829}{3628800}$	$\frac{-407}{172800}$	$\frac{-330157}{159667200}$	$\frac{-24377}{13305600}$	$\frac{-4281164477}{2615348736000}$	

si bien con una serie de particularidades: la fórmula con q=0 no se usa, mientras que con q=1, se puede reducir a la ya presentada en (3.6.7).

Por otra parte las fórmulas de Cowell son implícitas para $q \geq 2$, lo normal en estos casos es utilizar una fórmula Predictor-Corrector de la misma forma que ya hicimos con los Adams-Moulton, pero esta vez el Predictor más habitual suele ser el correspondiente método de Störmer. Otra vez el cálculo puede ser más costoso en cuentas, pero vale la pena puesto que el error en cada iteración suele ser más pequeño, sobre todo cuando $0 < h \ll 1$.

3.8. Selección de problemas.

1. Ya conocemos cuál es la solución del problema de valor inicial

$$y'(x) = \sin(x), \quad y(0) = 5, \tag{3.8.1}$$

y ya hemos visto cuál es la aproximación numérica que realiza un Runge-Kutta de orden 2 con este problema.

Ahora, escríbase un programa del Adams-Bashforth de orden 2 ya dado, para resolver dicho P.V.I. con x variando entre 0 y 2 y con longitud de paso h=0.1 primero, luego h=0.01 y después h=0.001 .

Solución :

Los programas que proponemos en este capítulo son bastante similares para todos los lenguajes de programación. Puesto que el programa del Runge-Kutta de orden 2 estaba escrito con el Mathematica, a partir de ahora los realizaremos con el Mathematica, salvo los que digamos expresamente que están escritos en otros lenguajes.

```
h=0.1;
x1=0;
xf=2;
niter=(xf-x1)/h;
x0=x1-h;
y1=5;
y0=6-Cos[x0];
f[x_,y_]:=Sin[x];
g[x_]:=-Cos[x]+6;
For[j=1,j<=niter,j++,
  y2=y1+h*(3/2*f[x1,y1]-1/2*f[x0,y0]);
  y0=y1;
  y1=y2;
  x0=x1;
  x1=x1+h;
  Print[x1];
  Print[N[y1,15]];
  Print[N[y1-g[x1]]];
]
```

Utilizando el programa del ejercicio anterior, se puede obtener una tabla semejante a la siguiente:

x_n	0.1	0.2	1.	2.
y_n	5.0049916708	5.0199666833	5.4613990159	6.4218083213
error	$-4{,}16389 \times 10^{-6}$	0,0000333	0,0017013	0,00566148

En este caso el Runge-Kutta tiene un comportamiento bastante similar al del Adams-Basforth, ya que el error para $x = 2$, con el primero era de -0.0011803, mientras que ahora es de 0.00566148.

Cambiando la primera línea del código para hacer h=0.01, ahora conseguimos una tabla como la siguiente:

x_n	0.01	0.02	1.	2.
y_n	5.0000499991	5.0001999996	5.4597166374	6.4162056141
error	$-4{,}1666 \times 10^{-10}$	$3{,}33326 \times 10^{-9}$	0,00001894	0,0000588

por último, si tomamos h=0.001, ahora obtenemos una tabla como la siguiente:

x_n	0.001	0.002	1.	2.
y_n	5.0000004999	5.0000019999	5.4596978854	6.4161474264
error	$-4{,}08562 \times 10^{-14}$	$3{,}33955 \times 10^{-13}$	$1{,}9133 \times 10^{-7}$	$5{,}89834 \times 10^{-7}$

en definitiva, si comporamos ambos métodos, observamos que el Runge-Kutta se adapta algo mejor a este problema concreto.

2. Escríbase un programa del Adams-Moulton de 2 pasos, con el Adams-Bashforth del mismo número de pasos como predictor, para resolver el problema anterior,de nuevo con x variando entre 0 y 2 y con longitud de paso h=0.1 primero, luego h=0.01 y por último h=0.001 .

Solución :

El siguiente programa:

```
h=0.1;
x1=0;
xf=2;
niter=(xf-x1)/h;
x0=x1-h;
y1=5;
y0=6-Cos[x0];
f[x_,y_]:=Sin[x];
g[x_]:=-Cos[x]+6;
For[j=1,j<=20,j++,
y3=y1+h*(3/2*f[x1,y1]-1/2*f[x0,y0]);
y2=y1+h*(f[x1+h,y3]+f[x1,y1])/2;
y0=y1;
y1=y2;
```

```
x0=x1;
x1=x1+h;
Print[x1];
Print[N[y1,15]];
Print[N[y1-g[x1]]];
]
```

nos da la siguiente tabla:

x_n	0.1	0.2	1.	2.
y_n	5.0049916708	5.0199168082	5.4593145488	6.4149665174
error	$-4{,}16389 \times 10^{-6}$	$-0{,}000383145$	0,0017013	$-0{,}00118032$

podemos comprobar que el error en este caso es sensiblemente inferior al dado por el Adams-Bashforth, tal como asegura la teoría. Sin embargo el error es similar al del Runge-Kutta de orden 2 que dábamos en el capítulo anterior, aún cuando el Adams-Moulton es de orden 3.

Si variamos el paso, ahora h=0.01, obtenemos nuevos resultados:

x_n	0.01	0.02	1.	2.
y_n	5.0000499991	5.0001999916	5.4596938633	6.4161350353
error	$-4{,}1666 \times 10^{-10}$	$-1{,}6666 \times 10^{-9}$	$-3{,}8308 \times 10^{-6}$	0,0000588

y si tomamos h=0.001

x_n	0.001	0.002	1.	2.
y_n	5.0000004999	5.0000019999	5.4596976558	6.4161467185
error	$-4{,}0856 \times 10^{-14}$	$3{,}3395 \times 10^{-13}$	$-4{,}9597 \times 10^{-8}$	$-1{,}18012 \times 10^{-7}$

resultados prácticamente idénticos a los del Runge-Kutta de orden 2, aún cuando el Adams-Moulton tiene en este caso orden superior. Esto sucede porque en el esquema conjunto el orden que prevalece es el del Adams-Basforth, que es 2 (aunque las constantes del error del esquema PECE no coinciden con las de los Adams-Basforth, de hecho estas son mayores), más adelante pondremos un ejemplo de un esquema de orden superior, con los mismos métodos.

3. 1) Escríbase un programa del Adams-Basforth de orden 2, para resolver el problema

$$y'(x) = \sin(\lambda x), \quad y(0) = y_0, \tag{3.8.2}$$

2) Utilizando el programa del apartado 1, calcúlese el valor aproximado para $x = 2$ con pasos $h = 0{,}1$, $h = 0{,}01$ y $h = 0{,}001$, con $\lambda = 10$, $y_0 = 5{,}9$.

3) Hágase los mismos cálculos pero con $\lambda = 100$, $y_0 = 5{,}99$.

Solución :

1) Podría valer un programa parecido al del ejercicio 1 de este capítulo, sin más que introducir dos líneas con los valores de λ e y_0, y cambiando las líneas donde se introducen los datos del problema de Cauchy.

La solución exacta de este problema ya se dió en el capítulo anterior, y la explicación de la utilidad de este ejercicio está clara: se trata de observar como se comportan los Adams-Basforth ante problemas cada vez más oscilatorios.

2) Algunos de los resultados son los siguientes:

x_n	0.1	0.2	1.	2.
y_n	5.9420735492	6.0682941969	6.1582004217	5.9580950730
error	$-0{,}00389622$	0,0266795	0,0742933	$-0{,}00109672$

x_n	0.01	0.02	1.	2.
y_n	5.9004991671	5.9019966683	6.0846854646	5.9594151204
error	$-4{,}1639 \times 10^{-7}$	$3{,}3261 \times 10^{-6}$	0,0007783	0,00022332

x_n	0.001	0.002	1.	2.
y_n	5.9000049999	5.9000199996	6.0839056203	5.9591942372
error	$-4{,}1665 \times 10^{-11}$	$3{,}3333 \times 10^{-10}$	$7{,}6985 \times 10^{-6}$	$2{,}4435 \times 10^{-6}$

Podemos observar que los errores son mayores a los del ejercicio anterior y también a los obtenidos con el Runge-Kutta: con $h = 0{,}1$ el error con Runge-Kutta era de 0.0050168, con $h = 0{,}01$ de 0.00004933 y si $h = 0{,}001$, el error cuando $x = 2$ era de $4{,}93266 \times 10^{-7}$.

Con el Adams-Basforth ahora son de $-0{,}00109672$, 0,00022332 y $2{,}4435 \times 10^{-6}$, aproximadamente 5 veces mayor

3) Los resultados son los siguientes:

x_n	0.1	0.2	1.	2.
y_n	5.9627989444	5.9811957778	6.0307806669	6.0487689478
error	$-0{,}04559$	$-0{,}11472$	0,0394039	0,0536408

x_n	0.01	0.02	1.	2.
y_n	5.9942073549	6.0068294197	5.9930032670	5.9988583305
error	$-0{,}0003896$	0,0026679	0,00162646	0,0037302

x_n	0.001	0.002	1.	2.
y_n	5.9900499167	5.9901996668	5.9913838012	5.9951516284
error	$-4{,}1639 \times 10^{-8}$	$3{,}32612 \times 10^{-7}$	$6{,}9899 \times 10^{-6}$	0,00002350

Ahora con el Adams-Basforth los errores en $x = 2$ son 0,0037302, 0,0037302 y 0,00002350, mientras que con los Runge-Kutta eran 0.01271294, 0.00043463 y $4{,}274185 \times 10^{-6}$.

Esto es si dividimos 0,00002350 entre $4{,}274185 \times 10^{-6}$ (los errores con paso h=0.001 con Adams-Basforth y Runge-Kutta respectivamente) tenemos un coeficiente de 5,4993, mientras que si hacemos la respectiva operación con los valores obtenidos del problema del primer ejercicio de este capítulo ($5{,}89834 \times 10^{-7}$ con el Adams-Basforth y $-1{,}1804 \times 10^{-7}$ con el Runge-Kutta), ahora el coeficiente es $-4{,}9969$, es decir aunque poco a poco la diferencia entre el Runge-Kutta y el Adams-Basforth aumenta cuanto más oscilatorio es el problema, esto es porque el Runge-Kutta aproxima la función en coordenadas intermedias entre una iteración y la siguiente, por el contrario los métodos multipaso sólo se fijan en lo que sucedió en iteraciones anteriores.

Por eso muchos investigadores prefieren trabajar con métodos multipaso antes que con Runge-Kutta al enfrentarse a ciertos problemas, y sin embargo en los ejercicios de valor inicial dados anteriormente parecía que los Runge-Kutta en general tienen un comportamiento mejor que los Adams-Basforth.

4. Escríbase un programa del Adams-Moulton de 2 pasos, con el respectivo Adams-Bashforth de predictor, para resolver el problema dado en el ejercicio anterior

$$y'(x) = \sin(100x), \quad y(0) = 5{,}99, \tag{3.8.3}$$

de nuevo con x variando entre 0 y 2 y con longitud de paso h=0.1 primero, luego h=0.01 y por último h=0.001 .

Solución :

Podría valer un programa parecido al del segundo ejercicio de este capítulo, sin más que introducir dos líneas con los valores de λ e y_0, y cambiando las líneas donde se introducen los datos del problema de Cauchy. Además la solución exacta de este problema ya se ha dado, por lo que aquí se procede a dar los resultados y su explicación teórica.

x_n	0.1	0.2	1.	2.
y_n	5.9627989444	5.9812451514	5.9879636072	5.9824151745
error	$-0{,}04559$	$-0{,}01467$	$-0{,}003413$	$-0{,}0127129$

x_n	0.01	0.02	1.	2.
y_n	5.9942073549	6.0029611970	5.9912601181	5.9946934833
error	$-0{,}0003896$	$-0{,}001200$	$-0{,}0001167$	$-0{,}0004346$

x_n	0.001	0.002	1.	2.
y_n	5.9900499167	5.9901991681	5.9913756637	5.9951238491
error	$-4{,}1639 \times 10^{-8}$	$-1{,}6614 \times 10^{-7}$	$-1{,}1475 \times 10^{-6}$	$-4{,}2741 \times 10^{-6}$

Si comparamos los resultados aquí obtenidos con los del ejercicio anterior y con los del correspondiente ejercicio del capítulo anterior (ejercicio 3), observamos que este esquema funciona de modo similar al Runge-Kutta, los errores son inferiores a los del Adams-Bashforth de orden 2, pero el orden del esquema es 2, coincidiendo con lo dicho en el ejercicio 2 de este tema, pero contradiciendo (en principio con lo que hemos explicado en la sección teórica de los Adams-Moulton).

5. Escríbase un programa para resolver el problema

$$y'(x) = 1 - xy(x)^2, \quad y(0) = 0, \tag{3.8.4}$$

con h=0.1 h=0.01 y 0.001 y con x variando entre 0 y 5, y con un esquema $P(EC)^2E$, el corrector esta vez será el Adams-Moulton de 3 pasos y el predictor:
1) El Runge-Kutta de orden 3 y calculando los valores iniciales también con el Runge-Kutta de orden 3.
2) El Adams-Bashforth de 3 pasos y calculando los valores iniciales con el Runge-Kutta de orden 3.

Solución :

1) Un posible resultado sería escribir el programa con h=0.1 de la siguiente forma:

```
h=0.1;
x2=0;
x1=x2-h;
x0=x2-2*h;
y2=0;

f[x_,y_]:=1-x*y^2;
```

```
k1=-h*f[x2,y2];
k2=-h*f[x2-1/2 h,y2+1/2 k1];
k3=-h*f[x2-h,y2-k1+2*k2];

y1=y2+1/6*(k1+4 k2+k3);

k1=-2*h*f[x2,y2];
k2=-2*h*f[x2-h,y2+1/2 k1];
k3=-2*h*f[x2-2*h,y2-k1+2 k2];
y0=y2+1/6*(k1+4 k2+k3);

For[j=1,j<=50,j++,
k1=h*f[x2,y2];
k2=h*f[x2+1/2 h,y2+1/2 k1];
k3=h*f[x2+h,y2-k1+2 k2];
yp=y2+1/6*(k1+4 k2+k3);
yc=y2+h*(5*f[x2+h,yp]+8*f[x2,y2]-f[x1,y1])/12;
yc=y2+h*(5*f[x2+h,yc]+8*f[x2,y2]-f[x1,y1])/12;

y0=y1;
y1=y2;
y2=yc;

x0=x1;
x1=x2;
x2=x2+h;
Print[x2];
Print[N[y2,15]];
]
```

Y se obtiene la siguiente tabla de resultados:

x_n	0.1	0.2	1.	5.
y_n	0.0999500375	0.1995515973	0.8055072480	0.4581298473

Si ahora quisiéramos controlar el error que hemos cometido podemos utilizar los conocimientos adquiridos en el tema anterior, uno de los posibles métodos para aproximar el error es tomar pasos más pequeños, hagamos pues h=0.01 y h=0.001.

x_n	0.1	0.2	1.	5.
y_n con h=0.01	0.09997498221	0.1996008633	0.8054800607	0.4581299060
y_n con h=0.001	0.09997500712	0.1996008633	0.8054800224	0.4581299060

Podemos comprobar que el error con h=0.1 era de -0.00002497 en la primera iteración, de -0.0000493148 en la segunda iteración, 0.0000272256 en el punto x=1, $-5{,}87 \times 10^{-8}$, cuando x=5.

Mientras que con paso h=0.01, ahora es de $-2{,}491 \times 10^{-8}$, $-4{,}88 \times 10^{-8}$, $3{,}83 \times 10^{-8}$ y $6{,}1 \times 10^{-11}$. Esto es, al dividir por 10 el paso, el error disminuye en 1000 veces aproximadamente. Ocurre que el esquema $P(EC)^2E$ tiene como orden el del Adams-Moulton, mientras que el esquema $PECE$ tiene el mismo orden que el del Adams-Bashforth correspondiente, como hemos podido observar con los ejercicios 2 y 4.

2) Escribimos el programa con paso constante h=0.1:

```
h=0.1;
x2=0;
x1=x2-h;
x0=x2-2*h;
y2=0;

f[x_,y_]:=1-x*y^2;

k1=-h*f[x2,y2];
k2=-h*f[x2-1/2 h,y2+1/2 k1];
k3=-h*f[x2-h,y2-k1+2*k2];
y1=y2+1/6(k1+4 k2+k3);

k1=-2*h*f[x2,y2];
k2=-2*h*f[x2-h,y2+1/2 k1];
k3=-2*h*f[x2-2*h,y2-k1+2 k2];
y0=y2+1/6(k1+4 k2+k3);

For[j=1,j<=50,j++,

yb=y2+h*(23/12*f[x2,y2]-16/12*f[x1,y1]+5/12*f[x0,y0]);
ym=y2+h*(5*f[x2+h,yb]+8*f[x2,y2]-f[x1,y1])/12;
ym=y2+h*(5*f[x2+h,ym]+8*f[x2,y2]-f[x1,y1])/12;
```

```
y0=y1;
y1=y2;
y2=ym;

x0=x1;
x1=x2;
x2=x2+h;

Print[x2]; Print[N[y2,15]];

]
```

Obteniendo los siguientes resultados:

x_n	0.1	0.2	1.	5.
y_n con h=0.1	0.0999500376	0.1995515000	0.8055048615	0.4581299621
y_n con h=0.01	0.0999749822	0.1996008633	0.8054800607	0.4581299061
y_n con h=0.001	0.09997500712	0.1996009121	0.8054800224	0.4581299060

Prácticamente son idénticos a los conseguidos en el apartado anterior del mismo ejercicio, sólo que ahora el método es menos costoso, es necesario hacer menos cuentas para obtener resultados muy similares. Por eso al Adams-Moulton le suele acompañar el Adams-Bashforth. Por supuesto este esquema tiene el mismo orden del Adams-Moulton.

6. Escríbase un programa para resolver el problema dado en el ejercicio anterior, pero ahora utilizando el BDF implícito de orden 4.

Solución :

Si utilizamos el correspondiente Adams-Bashforth de predictor, nos queda un programa como el que sigue:

```
h=0.1;
x3=0;
x2=x3-h;
x2=x3-2*h;
x0=x3-3*h;

y3=0;
```

```
f[x_,y_]:=1-x*y^2;

k1=-h*f[x3,y3];
k2=-h*f[x3-1/2 h,y3+1/2 k1];
k3=-h*f[x3-h,y3-k1+2*k2];
y2=y3+1/6*(k1+4 k2+k3);

k1=-2*h*f[x3,y3];
k2=-2*h*f[x3-h,y3+1/2 k1];
k3=-2*h*f[x3-2*h,y3-k1+2 k2];
y1=y3+1/6*(k1+4 k2+k3);

k1=-3*h*f[x3,y3];
k2=-3*h*f[x3-3/2*h,y3+1/2 k1];
k3=-3*h*f[x3-3*h,y3-k1+2 k2];
y0=y3+1/6*(k1+4 k2+k3);

For[j=1,j<=50,j++,
  yp=y3+h*(55*f[x3,y3]-59*f[x2,y2]+37*f[x1,y1]-9*f[x0,y0])/24;
  ybdf=(48*y3-36*y2+16*y1-3*y0+12*h*f[x3+h,yp])/25;

  y0=y1;
  y1=y2;
  y2=y3;
  y3=ybdf;

  x0=x1;
  x1=x2;
  x2=x3;
  x3=x3+h;

  Print[x3];
  Print[N[y3,15]];

]
```

Pudiendo obtener una tabla como la siguiente:

x_n	0.1	0.2	1.	5.
y_n con h=0.1	0.0999767914	0.1996045025	0.8057284363	0.4581296999
y_n con h=0.05	0.0999751291	0.1996020393	0.8054924548	0.4581298997
y_n con h=0.025	0.0999750083	0.1996009317	0.8054807038	0.4581299057

Vamos a tomar como buena aproximación los valores obtenidos del esquema $P(EC)^2E$, con h=0.001 del ejercicio 5 apartado 1: esto es, cuando x=0.1, y=0.09997500712, si x=0.2, y=0.1996009121, cuando x=1 y vale 0.8054800224 y si x=5, consideramos y=0.4581299060.

En ese caso obtenemos la siguiente tabla de errores:

x_n	0.1	0.2	1.	5.
y_n con h=0.1	$-1{,}8 \times 10^{-6}$	$-3{,}6 \times 10^{-6}$	-0.0002484	$2{,}061 \times 10^{-7}$
y_n con h=0.05	$-1{,}2 \times 10^{-7}$	$-2{,}2 \times 10^{-7}$	-0.00001243	$6{,}3 \times 10^{-9}$
y_n con h=0.025	$-1{,}18 \times 10^{-8}$	$-1{,}9 \times 10^{-8}$	$-6{,}8 \times 10^{-7}$	$3{,}0 \times 10^{-10}$

Si dividimos el error en un punto con paso 0.1 entre el error en ese mismo punto pero esta vez con paso 0.05, y después dividimos éste entre el error con paso 0.025, observamos que obtenemos resultados bastante variables, pero la mayoría de ellos entre 10 y 21, algo bastante lógico pues dicho BDF habíamos dicho que tendría orden 4, por tanto al dividir el paso a la mitad, el error se debería reducir a la decimosexta parte ($2^4 = 16$). Ya que en este caso el esquema conserva el orden del BDF, no como con el Adams-Moulton, que veíamos que necesitaba una corrección más.

7. Escríbase un programa para resolver el siguiente problema de valor inicial de segundo orden

$$y''(t) = -\frac{4t^2}{(1+t^2)y(t)}, \quad y(0) = 1, \quad y'(0) = 1, \tag{3.8.5}$$

utilizando el método de Stormer de 3 pasos, con longitud de paso $h = 0{,}1$ y $h = 0{,}02$, hasta el punto $t = 2$.

Solución :

En primer lugar es preciso inicializar, para ello, tenemos varias opciones:

1) Se podría inicializar con un Runge-Kutta o con alguno de los métodos multipaso descritos anteriormente para problemas de valor inicial de primer orden, para ello se procede del siguiente modo:

i) En primer lugar nuestra problema se transforma en un sistema de ecuaciones de primer orden. Para ello introducimos una nueva variable $z(t) =$

$y'(t)$, entonces nuestro problema se convierte en

$$\begin{array}{l} y'(t) = z(t) \\ z'(t) = -\frac{4t^2}{(1+t^2)y(t)} \\ y(0) = 1 \qquad z(0) = 1. \end{array} \tag{3.8.6}$$

ii) Ahora ya se puede inicializar con uno de los métodos dados.

2) La segunda opción es utilizar los comandos que dispone el propio paquete Mathematica.

Para resolver ecuaciones diferenciales ordinarias de forma numérica, el programa Mathematica dispone del comando "NDSolve ", en el que hay que introducir además de las propias ecuaciones con las condiciones iniciales, hay que especificar cúal es la variable incognita y la variable temporal. Es decir, en nuestro ejemplo particular procederíamos de la siguiente forma:

```
NDSolve[{y''[t]==-4*t^2/(1+t^2)*y[t],y[0]==1,y'[0]==1},
  y,{t,0,2}]
```

Con este comando obtendríamos como respuesta un número, y la indicación de que tenemos la curva solución del problema:

```
Out[1]= {{y->InterpolatingFunction[{{0.,2.}},"<>"]}}
```

Esto nos indica que en la salida 1 tendremos nuestra curva. Así, si quisiéramos ver el dibujo de la solución que da el Mathematica disponemos del comando Plot:

```
Plot[y[t]/.%1,{t,0,2}]
```

Con esta sentencia hemos pedido que nos dibuje la curva almacenada en la salida 1, con la variable temporal variando entre $t = 0$ $t = 2$, y obtendríamos la figura 3.1. Ahora le pedimos la solución que obtiene el Mathematica en los puntos $t = 0{,}1$, $t = 0{,}2$, $t = 0{,}02$, $t = 0{,}04$ y $t = 2$ con

```
y[0.1]/.%1
....
y[2]/.%1
```

De esta forma obtenemos los valores para inicializar y el valor en el punto $t = 2$, para comparar los resultamos que nosotros obtenemos con los que ha obtenido el Mathematica.

Ahora ya podemos escribir nuestro programa del método de Stormer:

Figura 3.1: Solución que da el Mathematica al problema .

```
t0=0;
tf=2;
y[0]=1;
h=0.1;
pasos=(tf-t0)/h;
f[t_,y_]:=-4*t^2/(1+t^2)*y;

(*  Introducimos los valores iniciales *)

y[1]=1.0999648057686668;

(* Valor que hemos obtenido con el NDSolve *)

y[2]=1.1994122781745542;

t0=t0+2*h;

For[j=3,j<=pasos,j++,
  y[j]= N[2*y[j-1]-y[j-2]+h^2*
  (13/12*f[t0,y[j-1]]-1/6*f[t0-h,y[j-2] ]+1/12*f[t0-2*h,y[j-3]]),
  15];
  t0=t0+h;
```

```
  Print[t0];
  Print[y[j]];
];
Print[y[j-1]+0.2162002026731293 ];

(* El valor obtenido con el Mathematica para t=2 es
y(2)=-0.2162002026731293  *)
```

El programa para paso constante $h = 0{,}02$, se escribiría de forma similar, cambiando únicamente los valores iniciales que ahora serán $y(0{,}02) = 1{,}0199999445487868$ y $y(0{,}04) = 1{,}039999123323856$.

Con estos programas obtenemos que el valor para $t = 2$ con paso $h = 0{,}1$ con el método de Stormer de 3 pasos es $y(2) = -0{,}21712591345924553$, siendo el error en relación a la solución que da el programa Mathematica $error = -0{,}0009257107861162239$. En cambio si utilizamos $h = 0{,}02$, obtenemos que $y(2) = -0{,}21620686767685685$ y el error es de $error = -6{,}66500372753509 \times 10^{-6}$.

8. Escríbase un programa para resolver el problema de valor inicial de segundo orden descrito en el problema anterior, utilizando ahora el método de Cowell de 2 pasos, con longitud de paso $h = 0{,}1$ y $h = 0{,}02$.

Solución : Ya hemos comentado que uno de los inconvenientes de los métodos de Cowell es el hecho de que sean implícitos.

En otros ejercicios anteriores donde los métodos eran implícitos, hemos utilizado métodos predictores para solucionar esa dificultad. Sin embargo, en este caso vamos a aprovechar el hecho de que el programa estará escrito en Mathematica para para solucionar ese inconveniente.

La dificultad añadida con los métodos implícitos es que en cada iteración tenemos que resolver una ecuación, muchas veces no lineal. El método más usual para resolver ecuaciones no lineales es el método de Newton. El Mathematica dispone de una función para resolver ecuaciones no lineales, dicha función es el "FindRoot ", a la cual hay que darle como parámetros la ecuación, la incognita de la ecuación y un valor inicial sobre el cual empieza a buscar.

Por ejemplo: supongamos que tenemos que resolver $f(x) = x - \sin(x) + e^x = 0$, que tiene claramente una sola raíz, ya que $f'(x) = 1 - \cos(x) + e^x > e^x > 0$ por lo que es siempre creciente, y como $f(-2) < 0$ y $f(0) > 0$, entonces tiene una raíz en el intervalo $[-2, 0]$.

Entonces utilizamos el comando “FindRoot ”como sigue:

```
FindRoot[x - Sin[x] + E^(x) == 0, {x, 0}]
```

Obteniendo como resultado $\{x- > -1{,}23498\}$.

De esta forma un posible programa para resolver el problema de valor inicial dado con $h = 0{,}1$, sería:

```
t0 = 0;
tf = 2;
y[0] = 1;
h = 0.1;
pasos = (tf - t0)/h;

f[t_,y_]:= -4*t^2/(1 + t^2)*y;

(*  Hab\'{\i}amos obtenido el valor inicial  *)

y[1] = 1.0999648057686668; t0 = t0 + h;

For[j = 2, j <= pasos, j++,
  a = FindRoot[
   l == 2*y[j - 1] - y[j - 2] +
   h^2/12*(f[t0 + h, l] + 10*f[t0, y[j - 1] ] +
   f[t0 - h, y[j - 2]] ), {l, y[j - 1]}, MaxIterations -> 20];
  y[j] = l /. a;
  t0 = t0 + h;
  Print[t0];
  Print[y[j] ];

];
Print[y[j - 1] + 0.2162002026731293 ];
```

El método de Newton es un método iterativo, el comando “MaxIterations”da un tope de iteraciones que nosotros hemos colocado en 20, en caso de que después de 20 iteraciones, no se haya logrado éxito, el Mathematica da un mensaje de error y nos da como solución del problema el valor de la x en la última iteración.

Con este programa los resultados obtenidos fueron:

	y_n	error con resultado del Mathematica
h=0.1	-0.21620535349783526	$-5{,}15082470595174 \times 10^{-6}$
h=0.02	-0.2162001718485963	$3{,}0824533009399246 \times 10^{-6}$

De todas formas combiene señalar que el resultado dado por el programa Mathematica no tiene porque haber sido el correcto. Es más dando un valor más pequeño al paso h en los métodos de Stormer o de Cowell, o exigiendo mayor precisión al Mathematica, se observa que en realidad tampoco era exacto el valor que habíamos dado por bueno.

9. Escríbase un programa en C++ para resolver el sistema de ecuaciones diferenciales

$$\begin{aligned} x'(t) &= x(t)y(t) + t \\ y'(t) &= y(t)t + x(t) \\ x(0) &= 1 \qquad y(0) = -1 \end{aligned} \tag{3.8.7}$$

utilizando el Adams-Moulton de 4 pasos como corrector y el correspondiente Adams-Bashforth como predictor, con h=0.1 y x variando entre 0 y 2.0.

Para inicializar se utilizará el Runge-Kutta de orden 4 que se dió en el anterior capítulo (para hacer más sencillo el programa se podrían introducir los siguientes valores $x(0{,}1) = 0{,}913936$, $y(0{,}1) = -0{,}909217$, $x(0{,}2) = 0{,}852186$, $y(0{,}2) = -0{,}834089$, $x(0{,}3) = 0{,}810633$ y $y(0{,}3) = -0{,}771093$, aunque el resultado obtenido sería ligeramente diferente dado que estos valores están truncados).

Solución :

```
/* Programa para resolver el par de ecuaciones x'=xy+t, y'=ty+x
x(0)=1, y(0)=-1 mediante el Adams-Moulton */

       #include <stdio.h>
       #include <conio.h>
       #include <stdlib.h>
       #include <math.h>

       float x[6], y[6], t, tfinal, h, fx[6], fy[6], ffx, ffy;
       int i, j, niter;

       float k1,k2,k3,k4,l1,l2,l3,l4;
```

```
        float derivx(x,y,t)
        float x, y, t;
        {
       return (x*y +t);
        }

        float derivy (x,y,t)
        float x, y, t;
        {
       return (y*t+x);
        }

/* Hasta ahora hemos calculado las derivadas ahora empezamos con
la rutina principal */

        void main ()
        {

        clrscr();

        t=0.0;
        h=0.1;

        x[1]=1.0;
        y[1]=-1.0;

/* Utilizamos el Runge-Kutta de orden 4 para inicializar los datos
*/

        for (i=1;i<=3;i++)
        {
       k1=h*derivx(x[i],y[i],t);
       l1=h*derivy(x[i],y[i],t);
       k2=h*derivx(x[i]+k1/2,y[i]+l1/2,t+h/2);
       l2=h*derivy(x[i]+k1/2,y[i]+l1/2,t+h/2);
       k3=h*derivx(x[i]+k2/2,y[i]+l2/2,t+h/2);
       l3=h*derivy(x[i]+k2/2,y[i]+l2/2,t+h/2);
```

```
       k4=h*derivx(x[i]+k3,y[i]+l3,t+h);
       l4=h*derivy(x[i]+k3,y[i]+l3,t+h);

      x[i+1]=x[i]+1.0/6.0*(k1+2*k2+2*k3+k4);
      y[i+1]=y[i]+1.0/6.0*(l1+2*l2+2*l3+l4);

       t=t+h;

printf("\nLa inicializaci\'{o}n en t=%lf es: %lf %lf",t,x[i+1],y[i+1]);

        }

        t=t-3*h;

/* Calculamos los valores de las derivadas */

    for (i=1;i<=4;i++)
    {
        fx[i]=derivx(x[i],y[i],t+(i-1)*h);
        fy[i]=derivy(x[i],y[i],t+(i-1)*h);
    }

       t=t+4*h;

       tfinal=2.0;
       niter=(tfinal-3*h)/h+1;

       for(i=1;i<=niter;i++)
       {
        x[5]=x[4]+h*(55.0*fx[4]-59.0*fx[3]+37.0*fx[2]-9.0*fx[1])/24.0;
        y[5]=y[4]+h*(55.0*fy[4]-59.0*fy[3]+37.0*fy[2]-9.0*fy[1])/24.0;

printf("\n Con el predictor para t= %lf tenemos: %lf %lf ", t,x[5],y[5]);

        fx[5]=derivx(x[5],y[5],t);
        fy[5]=derivy(x[5],y[5],t);

        x[5]=x[4]+h*(9.0*fx[5]+19.0*fx[4]-5.0*fx[3]+fx[2])/24.0;
```

```
        y[5]=y[4]+h*(9.0*fy[5]+19.0*fy[4]-5.0*fy[3]+fy[2])/24.0;
printf("\n Con el corrector para t= %lf son: %lf %lf ", t,x[5],y[5]);

       fx[5]=derivx(x[5],y[5],t);
       fy[5]=derivy(x[5],y[5],t);
           getch();


           for(j=2;j<=5;j++)
           {
         x[j-1]=x[j];
         y[j-1]=y[j];
         fx[j-1]=fx[j];
         fy[j-1]=fy[j];
           }

       t=t+h;

           }

       getch();

       }
```

Obteniendo como resultados

t	x	y
0.5	0.777193	-0.670450
1	0.918813	-0.468043
1.5	1.374956	-0.100122
2	2.707228	1.229970

Bibliografía:

[HaWa96]- Hairer, E. y Wanner, G. (1996), *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.*
Springer-Verlag, Berlin.

[Hen62]- Henrici, P. (1962), *Discrete variable methods in ordinary differential equations.*
John Wiley and sons, Inc., New York.

[IxVaMe03]-Ixaru, L. Gr., Vanden Berghe, G. y De Meyer, H. (2003), *Exponentially fitted variable two-step BDF algorithm for first order ODEs.*
Computer Physics Communications 150.

[KiCh94]-Kincaid, D. y Cheney W. (1994), *Análisis numérico. Las matemáticas del cálculo científico.*
Addison-Wesley Iberoamericana, S.A.

[Lam91]-Lambert, J.D. (1991), *Numerical Methods for Ordinary Differential Systems. The initial Value Problem.*
Wiley, Chichester.

[Scr87]- Scraton, R.E. (1987), *Métodos numéricos básicos.*
Mc Graw-Hill.

[Sha94]-Shampine, L.F. (1994), *Numerical solution of Ordinary Differential Equations.*
Chapman and Hall, Nueva York .