Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

Métodos de extrapolación en la resolución numérica de ecuaciones diferenciales

Autor/a: Andrés Escorial Pereña
Tutor/es/as: Begoña Cano Urdiales
2024

Resumen:

A partir del estudio de los desarrollos asintóticos del error global cuando se integran ecuaciones diferenciales ordinarias con métodos de un paso, se propondrán y se justificarán técnicas de extrapolación para incrementar el orden de cualquier método. En particular, se considerará el algoritmo de Aitken-Neville para realizar efectivamente dicha extrapolación, que se aplicará tomando como base varios métodos de orden bajo, y haciendo énfasis en el método de Gragg.

Palabras clave:

Algoritmo de Aitken-Neville, error global, error local, extrapolación, método de Gragg, método simétrico.

Abstract:

Based on the study of asymptotic expansions of the global error when integrating ordinary differential equations with one-step methods, extrapolation techniques will be proposed and justified to increase the order of any method. In particular, the Aitken-Neville algorithm will be considered to effectively perform such extrapolation, which will be applied based on several low-order methods, with emphasis on Gragg's method.

Keywords:

Aitken-Neville algorithm, global error, local error, extrapolation, Gragg's method, symmetric method.

Introducción

Este Trabajo de Fin de Grado tiene como principal objetivo el estudio de los desarrollos asintóticos del error global al integrar ecuaciones diferenciales ordinarias con métodos de un paso para justificar técnicas de extrapolación y obtener así aproximaciones de diferentes órdenes en cada paso que permitan hacer estimaciones del error local e implementaciones con paso y orden variable. El libro de Ernst Hairer, Gerhard Wanner y Syvert P. Nørsett [4] ha sido nuestra principal fuente bibliográfica.

En el primer capítulo, tras una introducción a los métodos de un paso, que incluyen en particular a los métodos Runge-Kutta, justificamos los desarrollos asintóticos en la longitud de paso de sus errores locales y globales. Definimos también los métodos adjuntos y vemos qué propiedades cumplen. A continuación, definimos y trabajamos con los métodos simétricos, que son adecuados para el diseño de los métodos de extrapolación más eficientes.

En el segundo capítulo explicamos detalladamente cómo se construyen los métodos de extrapolación. En primer lugar, cuando el método base tiene orden 1, se justifica que el algoritmo de Aitken-Neville proporciona las aproximaciones de diferentes órdenes de manera muy económica. Se consideran además distintas sucesiones de pasos para conseguir dichas aproximaciones, como son la sucesión de Romberg, Bulirsch y la armónica [2, 7]. Tomando como base un método de orden 2 simétrico, se justifica que el algoritmo de Aitken-Neville también puede aplicarse y se recomienda en particular el método de Gragg (también llamado método de Gragg-Bulirsch-Stoer [1, 3]) por ser completamente explícito.

En el tercer capítulo proponemos y justificamos un algoritmo para ajustar el orden y tamaño de paso del método de extrapolación utilizado en cada paso para integrar ecuaciones diferenciales ordinarias con el método de Gragg o GBS. Los primeros códigos de este tipo fueron desarrollados por Bulirsch, Stoer y sus estudiantes [1]. Más adelante se perfeccionaron otros enormemente utilizados [2], denominados DIFEX1. El propuesto en este trabajo se ha realizado de manera independiente atendiendo a las cotas de error en [4], pero con propuestas ligeramente diferentes.

A lo largo del trabajo se han ido resolviendo ejemplos de uso de los diferentes algoritmos para mostrar que, efectivamente, los resultados teóricos se reproducen en los ejemplos prácticos. Para ello han sido utilizados varios programas de Matlab cuyo código está incluido en el apéndice A.

Índice general

Capítulo 1

Desarrollo asintótico del error global para métodos de un paso

Queremos resolver numéricamente un problema de valores iniciales de la forma

$$\begin{aligned} y'(x) &= f(x, y(x)), \quad x \in [x_0, x_F], \\ y(x_0) &= y_0 \in \mathbb{R}^m, \end{aligned} \tag{1.1}$$

donde $f : \Omega \subset I \times \mathbb{R}^m$ es una función de clase $\mathcal{C}^1(\Omega)$. Supongamos que tenemos un método de un paso,

$$y_{n+1} = y_n + h\Phi(x_n, y_n, h), \tag{1.2}$$

con $x_n = x_0 + nh$, siendo h un parámetro llamado longitud de paso y siendo $\Phi(x_n, y_n, h)$ una función que depende de f y se llama función incremento del método. De esta manera, y_n aproxima a $y(x_n)$. Para aclarar esto, se puede pensar en esta función como la aproximación del incremento por unidad de paso. Por ejemplificar, vamos a ver cúal es esta función en los métodos de Euler:

Ejemplo 1.1. *Para el método de Euler explícito, la función incremento es*

$$\Phi(x_n, y_n, h) := f(x_n, y_n). \tag{1.3}$$

Ejemplo 1.2. *Para el método de Euler implícito, hay que definir la función de forma implícita,*

$$\Phi(x_n, y_n, h) := f(x_n + h, y_n + h\Phi(x_n, y_n, h)).$$

La solución exacta de la ecuación diferencial tras dar un paso de longitud h es $y(x + h)$. Vamos a introducir ahora la definición de error local.

Definición 1.1. *Se denomina error local al residuo existente cuando en la ecuación que define el método se sustituye la solución numérica por la exacta. Si el método tiene orden p, la solución considerada es suficientemente regular y Φ también, dicha expresión tiene un desarrollo asintótico en $h = 0$ de la forma:*

$$y(x+h) - y(x) - h\Phi(x, y(x), h) = d_{p+1}(x)h^{p+1} + \cdots + d_{N+1}(x)h^{N+1} + \mathcal{O}(h^{N+2}). \quad (1.4)$$

1.1. El error global

Es bien conocido que, cuando el error local se comporta como $\mathcal{O}(h^{p+1})$, el error global se comporta como $\mathcal{O}(h^p)$. Nuestro objetivo ahora es encontrar una función $e_p(x)$ tal que

$$y(x_n) - y_n = e_p(x_n)h^p + \mathcal{O}(h^{p+1}). \quad (1.5)$$

Para ello consideramos

$$\widehat{y}_n := y_n + e_p(x_n)h^p, \quad (1.6a)$$

como la solución numérica de un nuevo método, que comparándolo con (1.2), podemos escribir de la forma

$$\widehat{y}_{n+1} = \widehat{y}_n + h\widehat{\Phi}(x_n, \widehat{y}_n, h). \quad (1.6b)$$

Observamos que entonces

$$\begin{aligned}\widehat{\Phi}(x_n, \widehat{y}_n, h) &= \frac{\widehat{y}_{n+1} - \widehat{y}_n}{h} = \frac{y_{n+1} + e_p(x_{n+1})h^p - y_n - e_p(x_n)h^p}{h} \\ &= \Phi(x_n, y_n, h) + (e_p(x_{n+1}) - e_p(x_n))h^{p-1} \\ &= \Phi(x_n, \widehat{y}_n - e_p(x_n)h^p, h) + (e_p(x_{n+1}) - e_p(x_n))h^{p-1},\end{aligned}$$

por lo que $\widehat{\Phi}(x, y, h)$ se puede expresar como

$$\widehat{\Phi}(x, y, h) = \Phi(x, y - e_p(x)h^p, h) + (e_p(x+h) - e_p(x))h^{p-1}. \quad (1.7)$$

La idea clave es ver cómo debe ser e_p para que el nuevo método tenga orden $p+1$. Notamos en primer lugar que debe ocurrir que $e_p(x_0) = 0$, puesto que, según (1.1), $y(x_0) = y_0$, y es necesario que esto ocurra para que (1.5) sea cierto para $n = 0$ independientemente del h escogido.

Para ver qué ecuación diferencial debe cumplir la función e_p, notemos en primer lugar lo siguiente:

Nota 1.1. *Si el método* (1.2) *tiene al menos orden 1 de consistencia, ocurre que*

$$f(x, y) = \Phi(x, y, 0), \quad (1.8)$$

y, por tanto,

$$\frac{\partial \Phi}{\partial y}(x, y, 0) = \frac{\partial f}{\partial y}(x, y). \quad (1.9)$$

Demostración. Recordemos en primer lugar que la consistencia es equivalente a que

$$y(x+h) - y(x) - h\Phi(x, y(x), h) = \mathcal{O}(h^2),$$

para cualquier función $y(x)$ solución de un problema de la forma (1.1). Haciendo desarrollos de Taylor en torno a $h = 0$, se tiene entonces que

$$hy'(x) + \mathcal{O}(h^2) - h\left(\Phi(x, y(x), 0) + h\frac{\partial \Phi}{\partial h}(x, y(x), 0) + \mathcal{O}(h^2)\right) = \mathcal{O}(h^2),$$

de donde anulando el coeficiente de h se obtiene

$$f(x, y(x)) = \Phi(x, y(x), 0).$$

Como y_0 en (1.1) puede ser cualquiera, $y(x)$ también, de donde se deduce (1.8) y, derivando respecto de y, (1.9). □

Desarrollando el error local del nuevo método (1.6) en potencias de h y utilizando (1.8) obtenemos:

$$\begin{aligned}
&y(x+h) - y(x) - h\widehat{\Phi}(x, y(x), h) \\
&= y(x+h) - y(x) - h(\Phi(x, y(x) - e_p(x)h^p, h) + (e_p(x+h) - e_p(x))h^{p-1}) \\
&= y(x+h) - y(x) - h\Phi(x, y(x), h) + h\left(e_p(x)h^p\frac{\partial \Phi}{\partial y}(x, y(x), h) + \mathcal{O}(h^{2p})\right. \\
&\qquad\qquad \left. -h^{p-1}(he_p'(x) + \mathcal{O}(h^2))\right) \\
&= d_{p+1}(x)h^{p+1} + e_p(x)h^{p+1}\frac{\partial \Phi}{\partial y}(x, y(x), 0) - e_p'(x)h^{p+1} + \mathcal{O}(h^{p+2}) \\
&= h^{p+1}\left(d_{p+1}(x) + e_p(x)\frac{\partial f}{\partial y}(x, y(x)) - e_p'(x)\right) + \mathcal{O}(h^{p+2}).
\end{aligned}$$

Observamos, por tanto, que el término en h^{p+1} desaparece si lo hace el paréntesis, por lo que para lograr esto $e_p(x)$ debe ser solución de la ecuación diferencial

$$\begin{aligned}
e_p'(x) &= \frac{\partial f}{\partial y}(x, y(x))e_p(x) + d_{p+1}(x), \\
e_p(x_0) &= 0.
\end{aligned} \tag{1.10}$$

Este proceso puede repetirse con el método con función incremento $\widehat{\Phi}(x, y(x), h)$, puesto que es de orden $p+1$ y satisface (1.8). Procediendo recursivamente, queda demostrado el siguiente teorema [3]:

Teorema 1.1. *Consideremos un método de la forma* (1.2) *con función incremento* Φ *suficientemente regular y consistente y cuyo error local admite un desarrollo de la forma* (1.4) *con* $p \geq 1$. *Entonces el error global tiene un desarrollo asintótico de la forma*

$$y(x) - y_n = e_p(x)h^p + ... + e_N(x)h^N + E_h(x)h^{N+1}, \tag{1.11}$$

donde las funciones e_j son soluciones de ecuaciones diferenciales no homogéneas del tipo (1.10) *y el residuo $E_h(x)$ queda acotado para $x_0 \le x \le x_F$ y $0 \le h \le h_0$, para cierto $h_0 > 0$.*

Las propiedades de diferenciabilidad de $e_j(x)$ dependen, como hemos visto antes, de las de f y Φ. El desarrollo (1.11) *será la base teórica para los métodos de extrapolación.*

Nota 1.2. *Cuando el método de partida* (1.2) *tiene orden $p \ge 2$ y Φ es lo suficientemente regular,*

$$\frac{\partial \Phi}{\partial h}(x,y,0) = \frac{1}{2}\left(\frac{\partial f}{\partial x}(x,y) + \frac{\partial f}{\partial y}(x,y) f(x,y)\right). \tag{1.12}$$

Demostración. Puesto que

$$y(x+h) - y(x) - h\Phi(x,y(x),h) = \mathcal{O}(h^3),$$

haciendo desarrollos de Taylor entorno a $h = 0$, se tiene que

$$hy'(x) + \frac{h^2}{2}y''(x) - h\Phi(x,y(x),0) - h^2\frac{\partial \Phi}{\partial h}(x,y(x),0) = \mathcal{O}(h^3),$$

que usando (1.8) puede escribirse como

$$\frac{h^2}{2}\left(\frac{\partial f}{\partial x}(x,y(x)) + \frac{\partial f}{\partial y}(x,y(x)) f(x,y(x))\right) - h^2\frac{\partial \Phi}{\partial h}(x,y(x),0) = 0,$$

de donde se obtiene el resultado teniendo en cuenta que el coeficiente de h^2 debe anularse. □

Nota 1.3. *Cuando el método de partida* (1.2) *tiene orden $p \ge 2$ y $\Phi(x,y(x),h)$ es lo suficientemente regular, el término principal del error local del método* (1.6) *viene dado por:*

$$\widehat{d}_{p+2}(x) = d_{p+2}(x) - \frac{1}{2}\frac{\partial f}{\partial y}(x,y(x)) d_{p+1}(x) - \frac{1}{2}d'_{p+1}(x),$$

con lo que el coeficiente $e_{p+1}(x)$ debe satisfacer

$$\begin{aligned} e'_{p+1}(x) &= \frac{\partial f}{\partial y}(x,y(x)) e_{p+1}(x) + d_{p+2}(x) - \frac{1}{2}\frac{\partial f}{\partial y}(x,y(x)) d_{p+1}(x) - \frac{1}{2}d'_{p+1}(x), \\ e_{p+1}(x_0) &= 0. \end{aligned} \tag{1.13}$$

Demostración. Si avanzamos un poco más en el desarrollo asintótico del error local del método (1.6) tenemos que

$$\begin{aligned} &y(x+h) - y(x) - h\widehat{\Phi}(x,y(x),h) \\ &= y(x+h) - y(x) - h(\Phi(x,y(x) - e_p(x)h^p, h) + (e_p(x+h) - e_p(x))h^{p-1}) \\ &= y(x+h) - y(x) - h\Phi(x,y(x),h) + h\left(e_p(x)h^p\frac{\partial \Phi}{\partial y}(x,y(x),h) + \mathcal{O}(h^{2p})\right. \\ &\quad \left. -h^{p-1}(he'_p(x) + \frac{h^2}{2}e''_p(x) + \mathcal{O}(h^3))\right) \\ &= d_{p+1}(x)h^{p+1} + d_{p+2}(x)h^{p+2} + e_p(x)h^{p+1}\left(\frac{\partial \Phi}{\partial y}(x,y(x),0) + \frac{\partial^2 \Phi}{\partial y \partial h}(x,y(x),0)h\right) \\ &\quad - e'_p(x)h^{p+1} - \frac{h^{p+2}}{2}e''_p(x) + \mathcal{O}(h^{p+3}). \end{aligned}$$

Como el método (1.6) tiene orden $p+1$, el coeficiente de h^{p+1} se va a anular. Así pues,

$$\begin{aligned} &y(x+h) - y(x) - h\widehat{\Phi}(x, y(x), h) \\ &= \left(d_{p+2}(x) + e_p(x)\frac{\partial^2 \Phi}{\partial y \partial h}(x, y(x), 0) - \frac{1}{2}e_p''(x) \right) h^{p+2} + \mathcal{O}(h^{p+3}). \end{aligned}$$

Desarrollamos ahora $e_p''(x)$ y $\frac{\partial^2 \Phi}{\partial y \partial h}(x, y(x), 0)$, teniendo en cuenta (1.10) y (1.12)

$$\begin{aligned} e_p''(x) &= \left(\frac{\partial^2 f}{\partial y \partial x}(x, y(x)) + \frac{\partial^2 f}{\partial y^2}(x, y(x))y'(x) \right) e_p(x) + \frac{\partial f}{\partial y}(x, y(x))e_p'(x) + d_{p+1}'(x) \\ &= \left(\frac{\partial^2 f}{\partial y \partial x}(x, y(x)) + \frac{\partial^2 f}{\partial y^2}(x, y(x))f(x, y(x)) + \left(\frac{\partial f}{\partial y}(x, y(x)) \right)^2 \right) e_p(x) \\ &\quad + \frac{\partial f}{\partial y}(x, y(x))d_{p+1}(x) + d_{p+1}'(x), \end{aligned}$$

$$\frac{\partial^2 \Phi}{\partial y \partial h}(x, y(x), 0) = \frac{1}{2}\left(\frac{\partial^2 f}{\partial y \partial x}(x, y) + \frac{\partial^2 f}{\partial y^2}(x, y)f(x, y) + \left(\frac{\partial f}{\partial y}(x, y(x)) \right)^2 \right).$$

Por lo tanto llegamos a que

$$\begin{aligned} &y(x+h) - y(x) - h\widehat{\Phi}(x, y(x), h) \\ &= \left(d_{p+2}(x) - \frac{1}{2}\frac{\partial f}{\partial y}(x, y(x))d_{p+1}(x) - \frac{1}{2}d_{p+1}'(x) \right) h^{p+2} + \mathcal{O}(h^{p+3}), \end{aligned}$$

que es lo que queríamos probar. □

1.2. Método adjunto y propiedades

Los algoritmos de extrapolación más importantes utilizan expresiones asintóticas con potencias pares de h. Para dar una base teórica para estos métodos, necesitamos explicar el significado de y_n para h negativo. Vamos a escribir (1.2) pero cambiando h por $-h$,

$$y_{n-1} = y_n - h\Phi(x_n, y_n, -h),$$

y ahora cambiamos x_n por $x_n + h$,

$$y_n = y_{n+1} - h\Phi(x_n + h, y_{n+1}, -h).$$

Hemos llegado a una ecuación implícita para y_{n+1} que, por el teorema de las funciones implícitas, posee una única solución para h suficientemente pequeño, expresado como,

$$y_{n+1} = y_n + h\Phi^*(x_n, y_n, h). \tag{1.14}$$

Llamando A a y_{n+1} y B a y_n estamos en condiciones de presentar la siguiente definición.

Definición 1.2. *Sea $\Phi(x,y,h)$ la función incremento de un método. Definimos la función incremento del método adjunto, $\Phi^*(x,y,h)$ a traves del par de fórmulas*

$$\begin{aligned} B &= A - h\Phi(x+h, A, -h), \\ A &= B + h\Phi^*(x, B, h). \end{aligned} \tag{1.15}$$

Ejemplo 1.3. *El método adjunto del método explícito de Euler es el método implícito de Euler, ya que de $B = A - hf(x+h, A)$, se obtiene $A = B + hf(x+h, A)$.*

Definición 1.3. *Sean b_i, a_{ij} y c_i $(i,j = 1,\ldots,s)$ números reales, el método*

$$\begin{aligned} k_i &= f(x_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_j), \quad i = 1,\cdots,s, \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{1.16}$$

se llama método Runge-Kutta de s etapas.

Teorema 1.2. *Sea un método Runge-Kutta con coeficientes a_{ij}, b_j, c_i $(i,j = 1,\ldots,s)$. Entonces el método adjunto es equivalente a un método Runge-Kutta con s etapas y con coeficientes,*

$$\begin{aligned} c_i^* &= 1 - c_{s+1-i}, \\ a_{ij}^* &= b_{s+1-j} - a_{s+1-i,s+1-j}, \\ b_j^* &= b_{s+1-j}. \end{aligned}$$

Demostración. Partiendo de (1.16) y sustituyendo h por $-h$,

$$\begin{aligned} k_i &= f(x_n - c_i h, y_n - h\sum_{j=1}^{s} a_{ij}k_j), \\ y_{n-1} &= y_n - h\sum_{i=1}^{s} b_i k_i. \end{aligned}$$

Cambiando x_n por $x_n + h$

$$\begin{aligned} k_i &= f(x_n + (1-c_i)h, y_n + h\sum_{j=1}^{s}(b_j - a_{ij})k_j), \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i. \end{aligned}$$

Para preservar el orden natural de $c_1,\cdots,c_s$ vamos a permutar los valores k_i y a cambiar los índices i y j por $s+1-i$ y $s+1-j$. Así tenemos que

$$\begin{aligned} k_{s+1-i} = k_{i^*} &= f(x_n + (1-c_{s+1-i})h, y_n + h\sum_{j=1}^{s}(b_{s+1-j} - a_{s+1-i,s+1-j})k_{j^*}), \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_{s+1-i}k_{i^*}, \end{aligned}$$

que es lo que queríamos probar. □

Otras propiedades del método adjunto son las siguientes:

Teorema 1.3. $\Phi^{**} = \Phi$.

Demostración. Si sustituimos h por $-h$, x por $x+h$, B por A y A por B en (1.15) llegamos al método original. □

Teorema 1.4. *El adjunto de un método tiene el mismo orden que el método original y el término principal del error local es el del método inicial multiplicado por* $(-1)^p$.

Demostración. Si cambiamos h por $-h$ en (1.4),

$$y(x-h) - y(x) + h\Phi(x, y(x), -h) = d_{p+1}(x)h^{p+1}(-1)^p + \mathcal{O}(h^{p+2}).$$

Sustituyendo x por $x+h$ tenemos

$$y(x) - y(x+h) + h\Phi(x+h, y(x+h), -h) = d_{p+1}(x+h)h^{p+1}(-1)^{p+1} + \mathcal{O}(h^{p+2}),$$

y teniendo en cuenta que $d_{p+1}(x+h) = d_{p+1}(x) + \mathcal{O}(h)$,

$$y(x) + d_{p+1}(x)h^{p+1}(-1)^p + \mathcal{O}(h^{p+2}) = y(x+h) - h\Phi(x+h, y(x+h), -h).$$

Tomando $A = y(x+h)$ y $B = y(x) + d_{p+1}(x)h^{p+1}(-1)^p + \mathcal{O}(h^{p+2})$, llegamos a que

$$B = A - h\Phi(x+h, A, -h),$$

y utilizando (1.15) tenemos

$$y(x+h) = y(x) + d_{p+1}(x)h^{p+1}(-1)^p + h\Phi^*(x, y(x), h) + \mathcal{O}(h^{p+2}),$$

que es lo buscado. □

Teorema 1.5. *El método adjunto tiene el mismo desarrollo asintótico del error global* (1.11) *que el original, pero con* $-h$ *en lugar de* h.

Demostración. Teniendo en cuenta el teorema anterior sobre el error local del método adjunto,

$$y(x_n) - y_n = e_p(x_n)(-h)^p + \mathcal{O}(h^{p+1}), \tag{1.17}$$

donde y_n es la solución numérica que proporciona el método adjunto. Esto es cierto porque el cambio de signo de $e_p(x)$, es decir de la solución de (1.10), cambia con d_{p+1}, teniendo en cuenta que $e_p(x_0) = 0$ y la fórmula de variación de las constantes. Más concretamente,

$$e_p(x) = \int_0^x M(x,s)d_{p+1}(s)ds,$$

siendo $M(x,s)$ la matriz de transición asociada al problema variacional

$$\begin{aligned}\delta'(x) &= \frac{\partial f}{\partial y}(x, y(x))\delta(x),\\ \delta(0) &= I.\end{aligned}$$

Así pues, tenemos el primer término que queríamos. Para finalizar, habría que repetir el procedimiento iterativo de la sección 1.1 teniendo en cuenta que la transformación de la función incremento de un método presentada en (1.6b) conmuta con la transformación del adjunto presentada en (1.15), es decir,

$$\widehat{\Phi}^* = \widehat{\Phi^*}. \tag{1.18}$$

Probémoslo. Partiendo de (1.6a) y (1.6b), para el método de incremento Φ,

$$\begin{aligned}y_{n+1} + e_p(x_n + h)h^p = \widehat{y}_{n+1} &= \widehat{y}_n + h\widehat{\Phi}(x_n, \widehat{y}_n, h)\\ &= y_n + e_p(x_n)h^p + h\widehat{\Phi}(x_n, y_n + e_p(x_n)h^p, h).\end{aligned}$$

Sustituyendo h por $-h$

$$y_{n-1} + e_p(x_n - h)(-h)^p = y_n + e_p(x_n)(-h)^p - h\widehat{\Phi}(x_n, y_n + e_p(x_n)(-h)^p, -h),$$

y sustituyendo x_n por $x_n + h$

$$y_n + e_p(x_n)(-h)^p = y_{n+1} + e_p(x_n + h)(-h)^p - h\widehat{\Phi}(x_n + h, y_{n+1} + e_p(x_n + h)(-h)^p, -h).$$

Tomando $A = y_{n+1} + e_p(x_n + h)(-h)^p$ y $B = y_n + e_p(x_n)(-h)^p$ y aplicando (1.15) tenemos,

$$y_{n+1} + e_p(x_n + h)(-h)^p = y_n + e_p(x_n)(-h)^p + h\widehat{\Phi}^*(x_n, y_n + e_p(x_n)(-h)^p, h). \tag{1.19}$$

Ahora bien, si utilizamos (1.6), teniendo en cuenta el error global del método adjunto probado en (1.17), obtenemos (1.19) de nuevo pero con $\widehat{\Phi^*}$ en lugar de $\widehat{\Phi}^*$, por lo que el resultado queda demostrado. Vamos a verlo de otra manera. El teorema 1.5 nos dice que si

$$y_{n+1} = y_n + h\Phi^*(x_n, y_n, h),$$

entonces

$$\widehat{y}_n = y_n + e_p(x_n)(-h)^p,$$

por lo que

$$\begin{aligned}\widehat{\Phi^*}(x_n, \widehat{y}_n, h) = \frac{\widehat{y}_{n+1} - \widehat{y}_n}{h} &= \frac{y_{n+1} + e_p(x_{n+1})(-h)^p - y_n - e_p(x_n)(-h)^p}{h}\\ &= \Phi^*(x_n, y_n, h) + (e_p(x_{n+1}) - e_p(x_n))\frac{(-h)^p}{h}\\ &= \Phi^*(x_n, \widehat{y}_n - e_p(x_n)(-h)^p, h) + (e_p(x_{n+1}) - e_p(x_n))\frac{(-h)^p}{h}.\end{aligned}$$

Luego

$$y_{n+1} = y_n + h\Phi^*(x_n, \widehat{y}_n - e_p(x_n)(-h)^p, h) = y_n - (e_p(x_{n+1}) - e_p(x_n))(-h)^p + h\widehat{\Phi^*}(x_n, \widehat{y}_n, h).$$

Así llegamos a

$$y_{n+1} + e_p(x_n + h)(-h)^p = y_n + e_p(x_n)(-h)^p + h(\widehat{\Phi^*})(x, y_n + e_p(x_n)(-h)^p, h),$$

y por comparación con (1.19), (1.18) queda probado. □

1.3. Métodos simétricos

Definición 1.4. *Un método se llama simétrico si* $\Phi = \Phi^*$

Definición 1.5. *El método númerico definido por*

$$y_{n+1} = y_n + \frac{h}{2}\left(f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right), \tag{1.20}$$

se llama regla de los trapecios.

Definición 1.6. *Al método*

$$y_{n+1} = y_n + hf\left(x_n + \frac{h}{2}, \frac{1}{2}(y_n + y_{n+1})\right), \tag{1.21}$$

se le denomina regla del punto medio implícita.

Ejemplo 1.4. *Los métodos* (1.20) *y* (1.21) *son simétricos, pues haciendo sus adjuntos (es decir, cambiando* y_{n+1} *por* y_n*,* h *por* $-h$ *y* x_n *por* $x_n + h$*) los métodos quedan invariantes.*

Ahora vamos a ver un par de teoremas para caracterizar los métodos simétricos Runge-Kutta. Los métodos Runge-Kutta que se utilizan son todos ellos consistentes y satisfacen la condición de simplificación

$$\sum_{j=1}^{s} a_{ij} = c_i, \tag{1.22}$$

por lo que supondremos a partir de ahora ambas propiedades en el método Runge-Kutta.

Teorema 1.6. *Si*

$$a_{s+1-i,s+1-j} + a_{ij} = b_{s+1-j} = b_j, \quad i, j = 1, \cdots, s, \tag{1.23}$$

entonces el método Runge-Kutta correspondiente es simétrico. Además, si $b_i \neq 0$ *para todo* i *y los* c_i *son distintos y están ordenados de la forma* $c_1 < c_2 < \cdots < c_s$*, entonces* (1.23) *es también una condición necesaria para la simetría.*

Demostración. La demostración de que (1.23) es condición suficiente es trivial a partir del teorema 1.2, puesto que la condición $c_i = 1 - c_{s+1-i}$ se cumple sumando (1.23) para $j = 1, \cdots, s$, teniendo en cuenta que los métodos Runge-Kutta se construyen con la condición de simplificación (1.22) y con la condición de consistencia $\sum_{i=1}^{s} b_i = 1$. Que el método sea simétrico implica que tanto el método original (de coeficientes c_i, a_{ij}, b_j) como el adjunto (c_i^*, a_{ij}^*, b_j^*) dan los mismos resultados numéricos. Si aplicamos ambos métodos a $y'(x) = f(x)$ con $h = 1$ e $y_n = x_n = 0$, entonces obtenemos

$$\sum_{i=1}^{s} b_i f(c_i) = \sum_{i=1}^{s} b_i^* f(c_i^*),$$

para cualquier f(x). Tomando, para cada i, f tal que valga 1 en c_i y 0 en el resto de valores c_j y c_j^* distintos de c_i y ordenando los índices de $\{c_j^*\}$ para que c_i^* coincida con c_i (si ningún c_i^* coincide con c_i, se llegaría a $b_i = 0$, que es absurdo) se llega a que

$$b_i^* = b_i, \quad c_i^* = c_i \quad \text{para todo } i. \tag{1.24}$$

Ahora aplicamos ambos métodos a $y_1'(x) = f(x)$, $y_2'(x) = x^q y_1$, con $h = 1$ e $y_{n,1} = y_{n,2} = x_{n,1} = x_{n,2} = 0$ y obtenemos que

$$\sum_{i,j=1}^{s} b_i c_i^q a_{ij} f(c_j) = \sum_{i,j=1}^{s} b_i^* c_i^{*q} a_{ij}^* f(c_j^*),$$

puesto que en este caso

$$\begin{aligned} y_{n+1,1} &= y_{n,1} + \sum_{i=1}^{s} b_i k_{i,1}, \quad k_{i,1} = f(c_i), \\ y_{n+1,2} &= y_{n,2} + \sum_{i=1}^{s} b_i k_{i,2}, \quad k_{i,2} = (c_i)^q \sum_{j=1}^{s} a_{ij} k_{j,1}. \end{aligned}$$

Esto implica, teniendo en cuenta (1.24) y el mismo razonamiento que antes, que

$$\sum_{i=1}^{s} b_i c_i^q a_{ij} = \sum_{i=1}^{s} b_i c_i^q a_{ij}^*, \quad \text{para todo } j.$$

De aquí, teniendo en cuenta que podemos tomar $q = 0, 1, \cdots, s-1$, tenemos para cada j un sistema homogéneo de Vandermonde con las incógnitas $b_i(a_{ij} - a_{ij}^*)$ para cada $i = 1, .., s$. Teniendo en cuenta que $b_i \neq 0$ se llega a que $a_{ij} = a_{ij}^*$. □

La siguiente propiedad de los métodos simétricos se desprende de los resultados previos.

Teorema 1.7. *Si además de cumplir las hipótesis del teorema 1.1, el método de un paso es simétrico entonces el desarrollo asintótico del error global* (1.11) *contiene sólo potencias pares de h:*

$$y(x) - y_n = e_{2q}(x)h^{2q} + e_{2q+2}(x)h^{2q+2} + \cdots, \tag{1.25}$$

con $e_{2j}(x_0) = 0$.

Demostración. Por ser $\Phi = \Phi^*$, el desarrollo asintótico del error global de ambos métodos es el mismo, por lo que utilizando el teorema 1.5,

$$y(x) - y_n = e_p(x)h^p + e_{p+1}(x)h^{p+1} + \cdots = e_p(x)(-h)^p + e_{p+1}(x)(-h)^{p+1} + \cdots.$$

Necesariamente los $e_j(x)$ tales que j es impar son 0, por lo que el resultado queda demostrado. □

1.4. Métodos de colocación

Vamos a presentar ahora los métodos de colocación y la propiedad que permite garantizar que son simétricos.

Definición 1.7. *Para s entero positivo y $c_1, \cdots, c_s$ números reales distintos (usualmente entre 0 y 1), el correspondiente polinomio de colocación $u(x)$ de grado s se define, imponiendo*

$$\begin{aligned} u(x_n) &= y_n \quad \textit{(valor inicial)} \\ u'(x_n + c_i h) &= f(x_n + c_i h, u(x_n + c_i h)), \quad i = 1, \cdots, s, \end{aligned} \tag{1.26}$$

que determina un método de la forma

$$y_{n+1} = u(x_n + h). \tag{1.27}$$

Teorema 1.8. *El método de colocación* (1.27) *es equivalente al método Runge-Kutta* (1.16) *de s etapas con coeficientes*

$$a_{ij} = \int_0^{c_i} \ell_j(t)dt, \qquad b_j = \int_0^1 \ell_j(t)dt, \qquad i, j = 1, \cdots, s,$$

donde las funciones $\ell_j(t)$ son los polinomios de Lagrange,

$$\ell_j(t) = \prod_{k \neq j} \frac{(t - c_k)}{(c_j - c_k)}.$$

Demostración. Tomamos $k_i = u'(x_n + c_i h)$ y como $u'(x_n + th)$ será un polinomio de grado $s - 1$, utilizando la forma de Lagrange,

$$u'(x_n + th) = \sum_{j=1}^{s} k_j \ell_j(t).$$

Integramos entonces $u'(x_n + th)$, entre $t = 0$ y $t = c_i$ y obtenemos

$$\begin{aligned} u(x_n + c_i h) &= y_n + h \int_0^{c_i} u'(x_n + th)dt = y_n + h \int_0^{c_i} \sum_{j=1}^{s} k_j \ell_j(t)dt \\ &= y_n + h \sum_{j=1}^{s} k_j \int_0^{c_i} \ell_j(t)dt = y_n + h \sum_{j=1}^{s} \left(\int_0^{c_i} \ell_j(t)dt \right) k_j. \end{aligned}$$

Ahora bien, usando (1.26),

$$k_i = f(x_n + c_i h, u(x_n + c_i h)) = f\left(x_n + c_i h, y_n + h \sum_{j=1}^{s} \left(\int_0^{c_i} \ell_j(t)dt \right) k_j \right),$$

con lo que hemos demostrado que $a_{ij} = \int_0^{c_i} \ell_j(t)dt$. Utilizando (1.27),

$$\begin{aligned} y_{n+1} &= u(x_n + h) = y_n + h\int_0^1 u'(x_n + th)dt = y_n + h\int_0^1 \sum_{j=1}^{s} k_j \ell_j(t)dt \\ &= y_n + h\sum_{j=1}^{s}\left(\int_0^1 \ell_j(t)dt\right) k_j, \end{aligned}$$

demostrando así que $b_j = \int_0^1 \ell_j(t)dt$, por lo que hemos llegado al método buscado. □

Teorema 1.9. *Un método de colocación basado en puntos de colocación distribuidos simétricamente es simétrico.*

Demostración. Que los puntos de colocación estén distribuidos simétricamente significa que $c_i = 1 - c_{s+1-i}$. Por este motivo, los polinomios de Lagrange cumplen que $\ell_i(t) = \ell_{s+1-i}(1-t)$, ya que en ese caso $\ell_{s+1-i}(1 - c_i) = \delta_{ij}$, con

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{si } i \neq j, \end{cases}$$

y $\ell_{s+1-i}(1 - t)$ sigue siendo un polinomio de grado $\leq s - 1$.

Tenemos ahora que ver que se cumple la condición (1.23), que se verifica fácilmente con lo anterior y un cambio de variable $u = 1 - t$:

$$\begin{aligned} a_{s+1-i,s+1-j} + a_{ij} &= \int_0^{c_{s+1-i}} \ell_{s+1-j}(t)dt + \int_0^{c_i} \ell_j(t)dt \\ &= \int_0^{c_{s+1-i}} \ell_{s+1-j}(t)dt + \int_0^{c_i} \ell_{s+1-j}(1-t)dt \\ &= \int_0^{c_{s+1-i}} \ell_{s+1-j}(t)dt + \int_{1-c_i}^{1} \ell_{s+1-j}(u)du \\ &= \int_0^{c_{s+1-i}} \ell_{s+1-j}(t)dt + \int_{c_{s+1-i}}^{1} \ell_{s+1-j}(u)du \\ &= \int_0^1 \ell_{s+1-j}(u)du = b_{s+1-j}. \end{aligned}$$

□

Capítulo 2

Descripción de los métodos de extrapolación

2.1. Procedimiento general de extrapolación

Vamos ahora a presentar un nuevo método de aproximación. Tenemos otro problema de la forma (1.1) y sea $H > 0$ una longitud de paso base. Elegimos una sucesión de enteros positivos tales que

$$n_1 < n_2 < n_3 < \cdots ,$$

y definimos sus correspondientes longitudes de paso $h_1 > h_2 > h_3 > \cdots$, como $h_i = H/n_i$. Elegimos ahora un método de orden p y calculamos el resultado numérico con n_i pasos de longitud h_i, obteniendo

$$T_{i,1} := y_{n_i,h_i},$$

donde y_{n_i,h_i} aproxima $y(x_0 + H)$.

La idea consiste en eliminar tantos términos como podamos del desarrollo asintótico del error global, calculando el polinomio interpolador

$$p(h) = \widehat{y} - e_p h^p - e_{p+1} h^{p+1} - \cdots - e_{p+k-2} h^{p+k-2}, \tag{2.1}$$

de forma que

$$p(h_i) = T_{i,1}, \quad i = j, j-1, \cdots , j-k+1. \tag{2.2}$$

Notemos que $\widehat{y}$ hará las veces de la solución exacta en el desarrollo asintótico despreciando los términos de orden mayor. Por ello, consideramos como mejor aproximación a la misma,

$$T_{j,k} := p(0) = \widehat{y}.$$

Las condiciones (2.2) corresponden a k ecuaciones lineales para las k incógnitas $\widehat{y}$, e_p, $\cdots$, e_{p+k-2}. Veamos un ejemplo.

Ejemplo 2.1. *Para $k = 2$, $n_1 = 1$, $n_2 = 2$, aparece la llamada extrapolación de Richardson, que consiste en lo siguiente:*

$$\begin{aligned} p(h_1) = T_{1,1} &= \widehat{y} - e_p H^p \\ p(h_2) = T_{2,1} &= \widehat{y} - e_p (\frac{H}{2})^p, \end{aligned}$$

por lo que

$$\begin{aligned} e_p H^p &= \widehat{y} - T_{1,1} \\ e_p H^p &= 2^p(\widehat{y} - T_{2,1}). \end{aligned}$$

Así pues tenemos que

$$\widehat{y} - T_{1,1} = 2^p(\widehat{y} - T_{2,1}),$$

y, despejando $\widehat{y}$, llegamos a la extrapolación de Richardson:

$$T_{2,2} = \widehat{y} = \frac{2^p T_{2,1} - T_{1,1}}{2^p - 1} = T_{2,1} + \frac{T_{2,1} - T_{1,1}}{2^p - 1}.$$

Vamos a ver ahora un lema previo que necesitaremos más adelante.

Lema 2.1. *Si $a_1, a_2, \cdots, a_n$ son distintos números reales positivos y $r_1, r_2, \cdots, r_n$ son números reales distintos, entonces*

$$A = \begin{pmatrix} a_1^{r_1} & a_1^{r_2} & \cdots & a_1^{r_n} \\ a_2^{r_1} & a_2^{r_2} & \cdots & a_2^{r_n} \\ \vdots & \vdots & \vdots & \vdots \\ a_n^{r_1} & a_n^{r_2} & \cdots & a_n^{r_n} \end{pmatrix}, \tag{2.3}$$

es invertible.

Demostración. Como A es invertible si y sólo si $\ker A = \vec{0}$, el resultado es equivalente a probar que si

$$g(t) = \sum_{i=1}^{n} \alpha_i t^{r_i}, \tag{2.4}$$

tiene n ceros positivos distintos, entonces $g(t) \equiv 0$. Para probar esto vamos a razonar por inducción sobre n. Si $n = 1$, el hecho de que $g(t) = \alpha_1 t^{r_1}$ tenga un cero positivo, por ejemplo $t = c$, implica $\alpha_1 c^{r_1} = 0$ con $c^{r_1} \neq 0$, por lo que $\alpha_1 = 0$. En otras palabras, $g(t) \equiv 0$. Supongamos cierto para $n - 1$ el resultado y veamos ahora si es cierto para n. Suponiendo que $g(t)$ en (2.4) tiene n ceros positivos distintos, lo mismo será cierto para $t^{-r_1} g(t)$. Por el teorema de Rolle, entonces

$$\frac{d}{dt}(t^{-r_1} g(t)) = \sum_{i=2}^{n} \alpha_i (r_i - r_1) t^{r_i - r_1 - 1},$$

tendrá $n - 1$ ceros positivos ditintos. Podemos por tanto aplicar la hipótesis de inducción pues hay $n - 1$ sumandos. Llegamos a que $\frac{d}{dt}(t^{-r_1} g(t)) \equiv 0$, en otras palabras, $t^{-r_1} g(t) = C$ con C una constante. Sea x un cero positivo, de g, se tiene $g(x) = C x^{r_1} = 0$. Por lo tanto, $C = 0$ con lo que $g(t) \equiv 0$, finalizando así nuestro razonamiento de inducción. □

Teorema 2.1. *El valor $T_{j,k}$ representa un método numérico de orden $p+k-1$.*

Demostración. Comparando (2.1) y (2.2) con (1.11), tomando $N = p+k-1$,

$$T_{i,1} = y(x_0+H) - e_p(x_0+H)h_i^p - \cdots - e_{p+k-2}(x_0+H)h_i^{p+k-2} - \Delta_i, \tag{2.5}$$

donde

$$\Delta_i = e_{p+k-1}(x_0+H)h_i^{p+k-1} + E_{h_i}(x_0+H)h_i^{p+k} = \mathcal{O}(H^{p+k}),$$

porque $e_{p+k-1}(x_0) = 0$ y $h_i \leq H$.

Notemos que (2.5) puede verse como un sistema de ecuaciones lineales, siendo las incógnitas $y(x_0+H)$, $H^p e_p(x_0+H)$, $\cdots$, $H^{p+k-2}e_{p+k-2}(x_0+H)$, donde la matriz de coeficientes es

$$A = \begin{pmatrix} -1 & \frac{1}{n_j^p} & \cdots & \frac{1}{n_j^{p+k-2}} \\ \vdots & \vdots & \vdots & \vdots \\ -1 & \frac{1}{n_{j-k+1}^p} & \cdots & \frac{1}{n_{j-k+1}^{p+k-2}} \end{pmatrix},$$

que, por el lema 2.1 es regular, ya que coincide salvo el signo de la primera columna con una matriz de la forma (2.3), con $n = k$, $a_1 = 1/n_j, \cdots, a_k = 1/n_{j-k+1}$, $r_1 = 0$ y $r_2 = p, \cdots, r_k = p+k-2$. Dicho sistema coincide con (2.2) pero en el lado derecho perturbado por $\Delta_i = \mathcal{O}(H^{p+k})$. Como la matriz de coeficientes es invertible, restando de (2.5) (2.2) obtenemos

$$\begin{pmatrix} y(x_0+h) - \widehat{y} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = A^{-1} \begin{pmatrix} \Delta_j \\ \vdots \\ \Delta_{j-k+1} \end{pmatrix}.$$

Por lo que,

$$|y(x_0+H) - \widehat{y}| \leq \|A^{-1}\|_\infty \cdot \max|\Delta_i| = \mathcal{O}(H^{p+k}),$$

que es una cota para el error local, que se comporta como $\mathcal{O}(H^{p+k})$. □

Una gran ventaja de este método es que nos proporciona una tabla de resultados numéricos de la forma,

$$\begin{array}{lllll} T_{11} & & & & \\ T_{21} & T_{22} & & & \\ T_{31} & T_{32} & T_{33} & & \\ T_{41} & T_{42} & T_{43} & T_{44} & \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{array} \tag{2.6}$$

que forma una sucesión de métodos encajados que permite estimar el error local fácilmente y desarrollar estrategias de orden variable. A modo de ejemplo vamos a presentar varias sucesiones de números de pasos $\{n_i\}$ utilizadas comúnmente:

Ejemplo 2.2. *La sucesión de Romberg: (Romberg 1955 [7])*

$$1, 2, 4, 8, 16, 32, 64, 128, 256, 512, \cdots \tag{2.7}$$

Esta sucesión está formada por potencias de 2.

Ejemplo 2.3. *La sucesión de Bulirsch: (Romberg 1955 [7])*

$$1, 2, 3, 4, 6, 8, 12, 16, 24, 32, \cdots \tag{2.8}$$

Para construir esta sucesión hay que alternar potencias de 2 *junto con* $1{,}5 \cdot 2^k$*. Una ventaja respecto a la anterior es que, para un método base dado, realiza menos pasos y, por tanto, necesita menos evaluaciones de función para llegar al mismo orden.*

Ejemplo 2.4. *La sucesión armónica: (Deuflhard 1983 [2])*

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \cdots \tag{2.9}$$

Es la sucesión más "económica" en el sentido de que es la que menos pasos debe dar para llegar a un orden dado.

Cuando el método es simétrico una demostración análoga a la del teorema 2.1 permite probar el siguiente resultado.

Teorema 2.2. *Cuando el método base es un método simétrico de orden* p*,* $T_{j,k}$ *representa un método numérico de orden* $p + 2k - 2$*.*

Demostración. Igual que en el teorema 2.1, teniendo en cuenta el desarrollo en potencias pares del error y tomando $N = p + 2(k-1)$, con lo que $\Delta_i = \mathcal{O}(H^{p+2k-1})$, que indica el comportamiento del error. Como antes, puede volver a aplicarse el lema 2.1. De aquí, el error global se comporta como $\mathcal{O}(H^{p+2k-2})$. □

2.2. El algoritmo de Aitken-Neville

En el caso $p = 1$, (2.1) y (2.2) se convierten en un problema de interpolación clásico y podemos calcular los valores de $T_{j,k}$ económicamente mediante el uso de métodos clásicos. Como sólo necesitamos los valores de los polinomios de interpolación en el punto $h = 0$, el algoritmo más económico es el de "Aitken-Neville". Vamos a presentar antes de ver este algoritmo un lema previo.

Lema 2.2. *En el problema de interpolación polinómica de Lagrange, denotemos por* W *al conjunto de los nodos y, para cada subconjunto* S *no vacío de* W*, llamemos* P_S *al polinomio interpolador de Lagrange de* f *en* S*. Si* S *y* T *son dos subconjuntos no vacíos de* W *que tienen todos sus puntos en común salvo el* x_i*, que está en* S *y no en* T*, y el* x_j *que está en* T *y no en* S*, entonces*

$$P_{S\cup T}(x) = \frac{(x_i - x)P_T(x) - (x_j - x)P_S(x)}{x_i - x_j}. \tag{2.10}$$

Demostración. Para demostrar que (2.10) es el polinomio interpolador de Lagrange de f en $S \cup T$ tenemos que ver que es un polinomio de grado $\leq m$, siendo $m+1$ el número de nodos de $S \cup T$, y que coincide con f en los nodos pues, de existir, es único. Esto último es trivial

para x_i y para x_j, pues lo cumplen P_S y P_T respectivamente. Para $x_k \in S$ distinto de x_i tenemos que $P_S(x_k) = P_T(x_k) = f(x_k)$, por lo que

$$P_{S\cup T}(x_k) = \frac{(x_i - x_k)f(x_k) - (x_j - x_k)f(x_k)}{x_i - x_j} = \frac{x_i f(x_k) - x_j f(x_k)}{x_i - x_j} = f(x_k).$$

Ahora bien, tanto P_S como P_T tienen grado $\leq m-1$ y, como están multiplicados por polinomios de primer grado, $P_{S\cup T}$ tiene grado $\leq m$. □

Basándonos en el lema 2.2 podemos construir $P = P_W$ a partir de dos polinomios que interpolen en N puntos. Cada uno de estos se puede construir a su vez a partir de dos que interpolen en $N-1$ puntos y así hasta llegar a polinomios constantes. Veámoslo.

Teorema 2.3. *Sea para $j = 0, 1, \cdots, N$,*

$$P_{j,0} = f(x_j),$$

y para $g = 0, 1, \cdots, N-1$, $j = g+1, \cdots, N$,

$$P_{j,g+1}(x) = \frac{(x_j - x)P_{j-1,g}(x) - (x_{j-g-1} - x)P_{j,g}(x)}{x_j - x_{j-g-1}}, \tag{2.11}$$

Entonces $P_{j,g}$ es el polinomio de grado $\leq g$ que interpola en los $g+1$ nodos $x_{j-g}, \cdots, x_j$.

Demostración. Sean $P_{j,g}(x)$ el polinomio de interpolación de Lagrange de f en los puntos $x_{j-g}, \cdots, x_j$, $P_{j-1,g}(x)$ el polinomio de interpolación en los puntos $x_{j-1-g}, \cdots, x_{j-1}$ y $P_{j,g+1}(x)$ el polinomio de interpolación en los puntos $x_{j-g-1}, \cdots, x_j$. Entonces los polinomios $P_{j,g+1}(x)$ y $P_{j,g}(x)$ coinciden en los puntos $x_{j-g}, \cdots, x_j$. Por tanto, las raíces de $P_{j,g+1}(x) - P_{j,g}(x)$, que es un polinomio de grado $\leq g+1$, son $x_{j-g}, \cdots, x_j$, es decir,

$$P_{j,g+1}(x) - P_{j,g}(x) = f_{j,g+1}(x - x_{j-g}) \cdots (x - x_j),$$

siendo $f_{j,g+1}$ el coeficiente del término de grado $g+1$. De manera análoga también tenemos que

$$P_{j,g+1}(x) - P_{j-1,g}(x) = f_{j,g+1}(x - x_{j-g-1}) \cdots (x - x_{j-1}).$$

Dividiendo ambas expresiones tenemos

$$\frac{P_{j,g+1}(x) - P_{j,g}(x)}{P_{j,g+1}(x) - P_{j-1,g}(x)} = \frac{x - x_j}{x - x_{j-g-1}}.$$

Despejando $P_{j,g+1}(x)$ llegamos a

$$P_{j,g+1}(x) = \frac{(x_j - x)P_{j-1,g}(x) - (x_{j-g-1} - x)P_{j,g}(x)}{x_j - x_{j-g-1}}.$$

□

Vamos ahora a simplificar (2.11) para que nos sea más cómodo utilizarlo:

$$\begin{aligned} P_{j,g+1}(x) &= P_{j,g}(x) + \frac{(x_j - x)P_{j-1,g}(x) - (x_{j-g-1} - x + x_j - x_{j-g-1})P_{j,g}(x)}{x_j - x_{j-g-1}} \\ &= P_{j,g}(x) + \frac{x - x_j}{x_j - x_{j-g-1}}(P_{j,g}(x) - P_{j-1,g}(x)). \end{aligned}$$

Si aplicamos el resultado previo con $x = 0$ y $x_j = \frac{H}{n_{j+1}}$, que son los valores que nos interesan, llegamos a

$$\begin{aligned} P_{j,g+1}(0) &= P_{j,g}(0) - \frac{\frac{H}{n_{j+1}}}{\frac{H}{n_{j+1}} - \frac{H}{n_{j-g}}}(P_{j,g}(0) - P_{j-1,g}(0)) \\ &= P_{j,g}(0) - \frac{1}{1 - \frac{n_{j+1}}{n_{j-g}}}(P_{j,g}(0) - P_{j-1,g}(0)) \\ &= P_{j,g}(0) + \frac{1}{\frac{n_{j+1}}{n_{j-g}} - 1}(P_{j,g}(0) - P_{j-1,g}(0)). \end{aligned}$$

Utilizaremos ahora la notación presentada en este capítulo, ya que al ser $p = 1$, $T_{j,k}$ es la evaluación en 0 de un polinomio de grado $g = k - 1$. Más concretamente, $T_{j,k} = P_{j-1,k-1}(0)$, donde los nodos de interpolación son $x_{j-1}, \cdots, x_{j-k}$. Por tanto, tomando en la fórmula de arriba $g = k - 1$ y cambiando j por $j - 1$ tenemos

$$T_{j,k+1} = T_{j,k} + \frac{T_{j,k} - T_{j-1,k}}{(n_j/n_{j-k}) - 1}. \tag{2.12}$$

Si el método utilizado es simétrico sabemos, según comentamos en la demostración del teorema 2.2, que cada extrapolación elimina dos potencias de h. Por lo tanto, para $p = 2$ (es decir, cuando $q = 1$ en (1.25)) también utilizamos el algoritmo de Aitken-Neville (con $x_k = (H/n_{k+1})^2$) llegando así a

$$T_{j,k+1} = T_{j,k} + \frac{T_{j,k} - T_{j-1,k}}{(n_j/n_{j-k})^2 - 1}, \tag{2.13}$$

en lugar de (2.12). Gracias a esto podemos calcular los elementos de (2.6) a partir de los 2 elementos a la izquierda y arriba, como en una tabla de diferencias divididas, lo cual es muy eficiente.

Vamos a presentar ahora un ejemplo numérico para esclarecer esto.

Ejemplo 2.5. *Vamos a resolver el problema*

$$\begin{aligned} y'(x) &= (-y(x)\sin x + 2\tan x)y(x) \\ y(\pi/6) &= 2/\sqrt{3}, \end{aligned}$$

con la solución $y(x) = 1/\cos x$ *y longitud de paso base* $H = 0{,}2$ *con el método de Euler explícito* (1.3)*, para lo que utilizaremos* (2.12)*, y con la regla del punto medio implícita* (1.21)*, que es un método simétrico, para lo que utilizaremos* (2.13)*. Las figuras 2.1, 2.2 y*

2.3 representan, para cada uno de los $T_{j,k}$ de la tabla de extrapolación, el trabajo numérico $1+n_j-1+n_{j-1}-1+\cdots+n_{j-k+1}-1$ (entendido como el número de evaluaciones de función realizadas), comparado con la precisión $|T_{j,k}-y(x_0+H)|$ en escala doblemente logarítmica y tomando como base el método de Euler explícito. Notemos que la evaluación de la función en la condición inicial se hace una vez por todas para todas las longitudes de paso necesarias. La primera gráfica corresponde a la sucesión (2.7)*, la segunda a* (2.8) *y la última a* (2.9)*.*

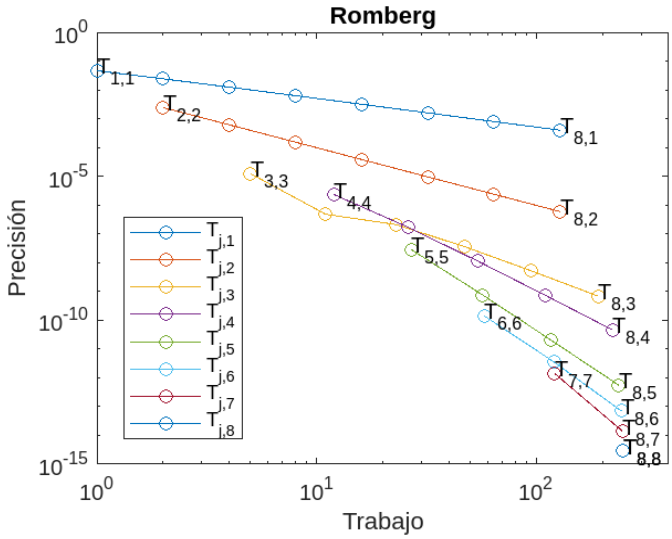

Figura 2.1: $T_{j,k}$ para el método de Euler explícito con la sucesión de Romberg.

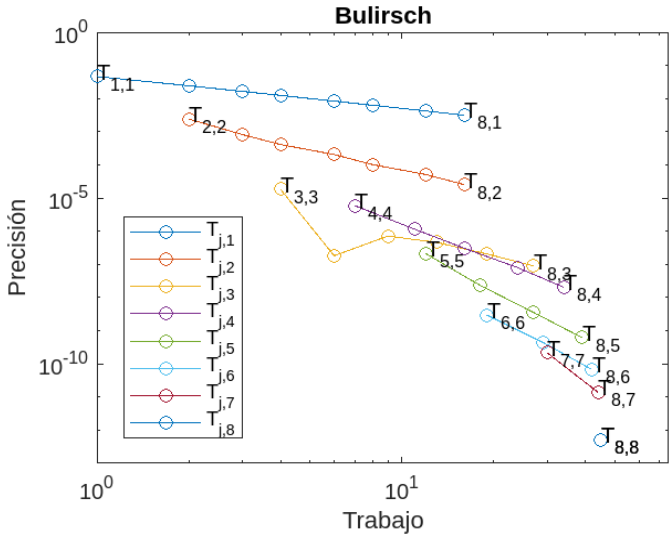

Figura 2.2: $T_{j,k}$ para el método de Euler explícito con la sucesión de Bulirsch.

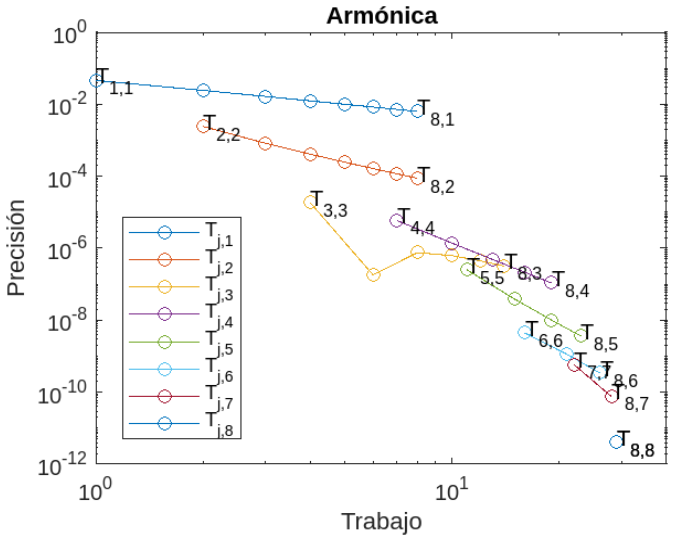

Figura 2.3: $T_{j,k}$ para el método de Euler explícito con la sucesión Armónica.

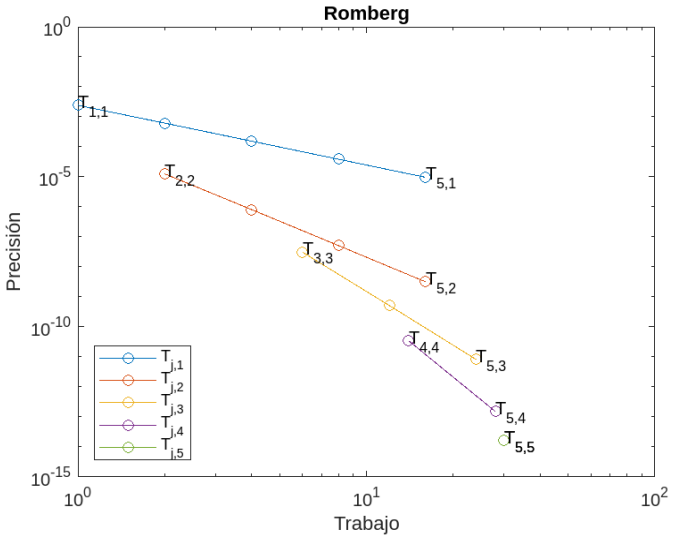

Figura 2.4: $T_{j,k}$ para la regla del punto medio implícita con la sucesión de Romberg.

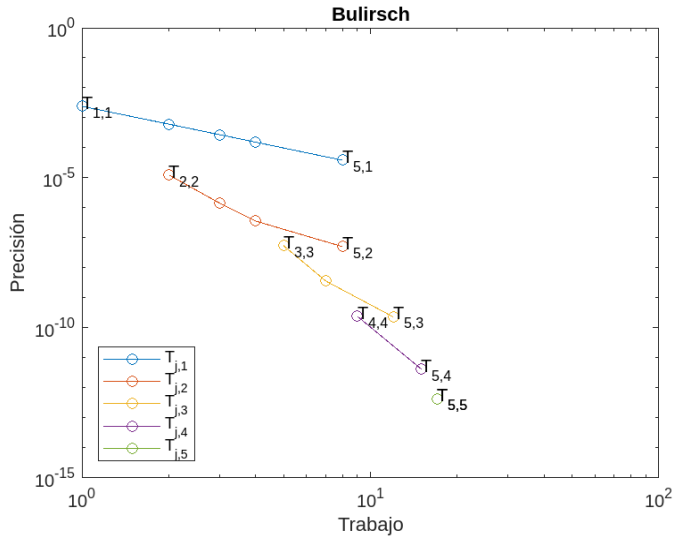

Figura 2.5: $T_{j,k}$ para la regla del punto medio implícita con la sucesión de Bulirsch.

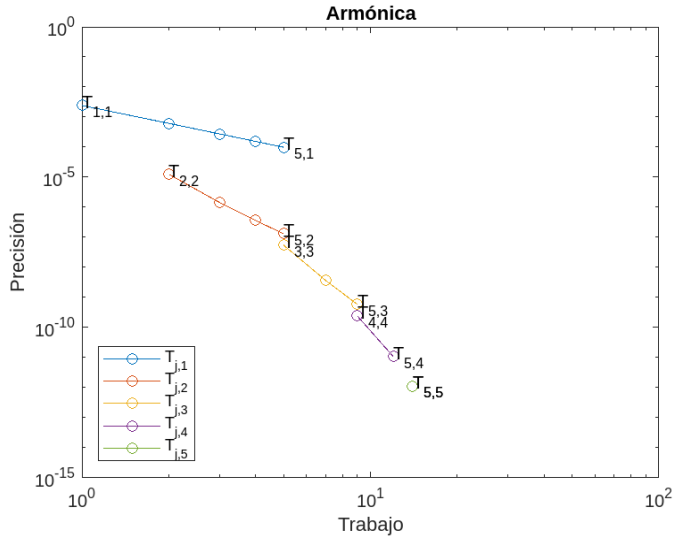

Figura 2.6: $T_{j,k}$ para la regla del punto medio implícita con la sucesión Armónica.

Observamos que el mejor método es el que viene dado por la sucesión armónica (2.9)*, en el sentido de que para obtener un cierto error es el que menos coste computacional requiere.*

Las figuras 2.4, 2.5 y 2.6 representan de nuevo el error cometido frente al coste computacional, pero tomando ahora como base la regla del punto medio implícita. Además ahora medimos el coste computacional como el número de etapas implícitas que hay que calcular,

que son $n_j + n_{j-1} + \cdots + n_{j-k+1}$. Tal y como se mide el trabajo computacional no son comparables las gráficas correspondientes al método de Euler explícito y a la regla del punto medio implícita, porque cada paso de la regla del punto medio implícita conlleva mucho más coste computacional al tener que resolver implícitamente una ecuación no lineal. No obstante, sí se obtiene que para unos mismos valores de j y k, $T_{j,k}$ es bastante más precisa. Esto es esperable al tratarse de un método de orden $p = 2$, frente a $p = 1$ en el método de Euler explícito.

Procedemos ahora a corroborar los teoremas 2.1 y 2.2. Más concretamente, el orden de los métodos $T_{k,k}$ cuando se toma como base el método de Euler explícito o la regla del punto medio implícita. Notemos que, en el primer caso, como $p = 1$, el orden de $T_{k,k}$ debe ser k. En el segundo caso, como $p = 2$, el orden de $T_{k,k}$ debe ser $2k$. Esto puede comprobarse en las figuras 2.7, 2.8, 2.9 y en las figuras 2.10, 2.11, 2.12, donde se representa el error en tiempo $t = 1$ frente a la longitud de paso H. Puede observarse que la pendiente de las rectas obtenidas para cada k uniendo los puntos correspondientes a las diferentes H's coincide de manera bastante aproximada con el orden esperado de todos los métodos, para tanto la sucesión de Romberg, como la de Bulirsch y como la armónica.

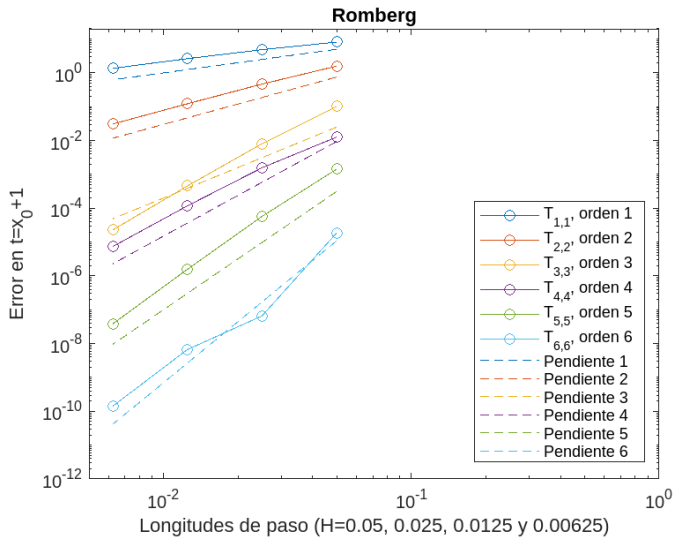

Figura 2.7: Orden de los métodos $T_{k,k}$ para el método de Euler explícito con la sucesión de Romberg.

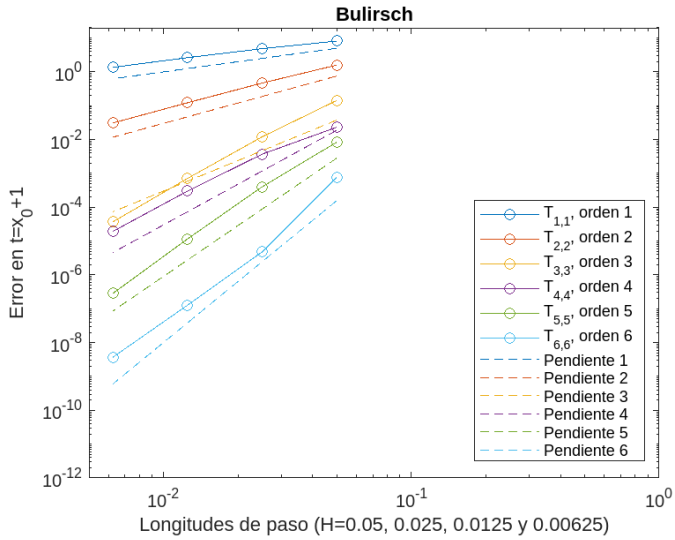

Figura 2.8: Orden de los métodos $T_{k,k}$ para el método de Euler explícito con la sucesión de Bulirsch.

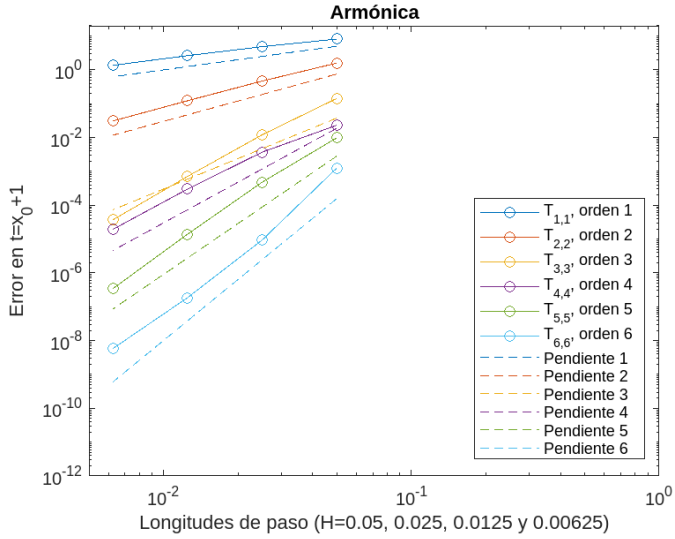

Figura 2.9: Orden de los métodos $T_{k,k}$ para el método de Euler explícito con la sucesión Armónica.

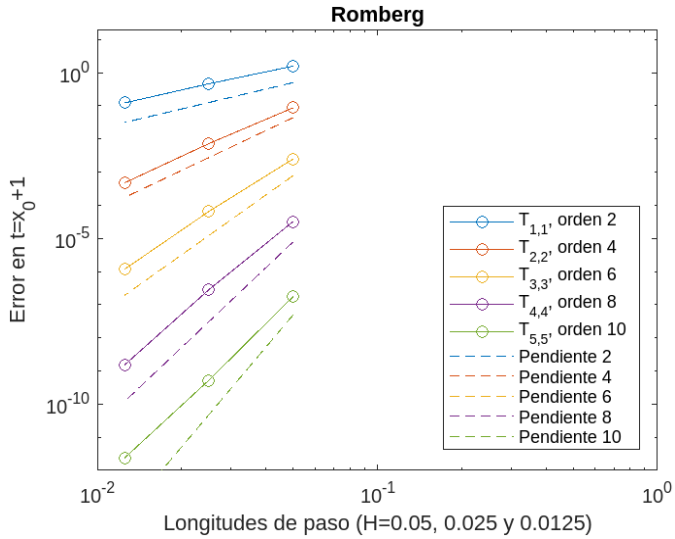

Figura 2.10: Orden de los métodos $T_{k,k}$ para la regla del punto medio implícita con la sucesión de Romberg.

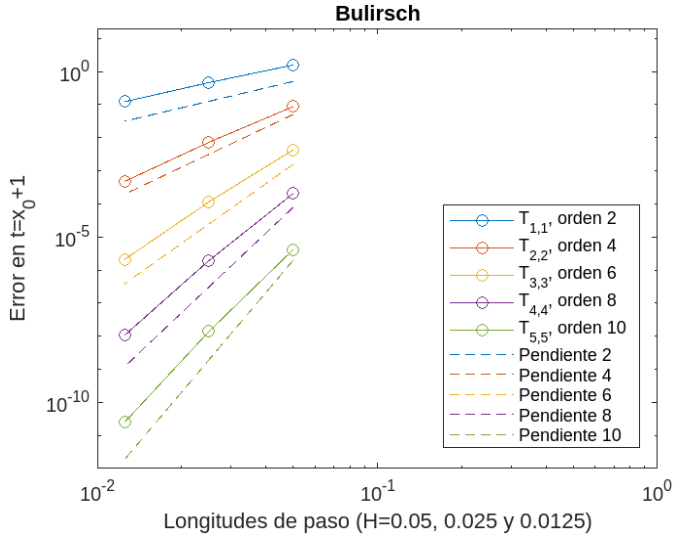

Figura 2.11: Orden de los métodos $T_{k,k}$ para la regla del punto medio implícita con la sucesión de Bulirsch.

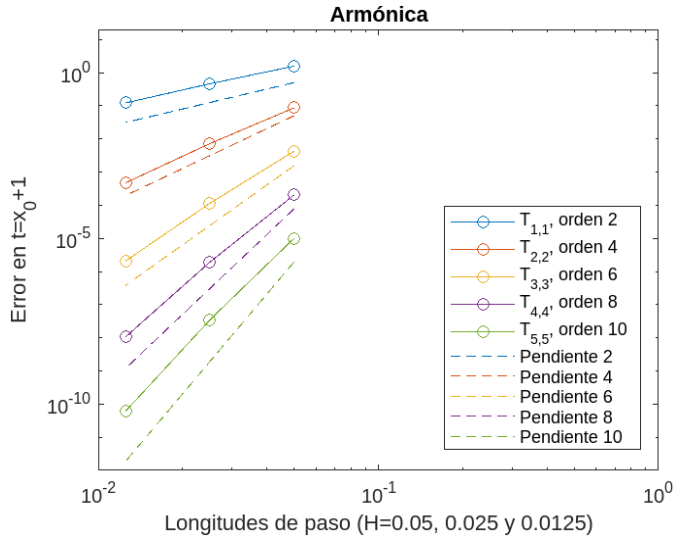

Figura 2.12: Orden de los métodos $T_{k,k}$ para la regla del punto medio implícita con la sucesión Armónica.

2.3. El método de Gragg o GBS

El desarrollo asintótico en potencias de h^2 es muy importante para una aplicación eficiente de la extrapolación de Richardson. Gragg demostró [3] que $S_h(x)$, el método descrito mediante el algoritmo ($x = x_0 + 2nh$, $x_i = x_0 + ih$):

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) \\ y_{i+1} &= y_{i-1} + 2hf(x_i, y_i) \quad i = 1, 2, \cdots, 2n \\ S_h(x) &= \frac{1}{4}(y_{2n-1} + 2y_{2n} + y_{2n+1}), \end{aligned} \tag{2.14}$$

posee un desarrollo asintótico en potencias pares de h y tiene buenas propiedades de estabilidad. Esto condujo a la construcción del poderoso algoritmo de extrapolación de G(ragg)-B(ulirsh)-S(toer).

La prueba de Gragg es muy larga y complicada, pero Stetter [8] tuvo la elegante idea de interpretar (2.14) como un método de un paso, reescribiéndolo en términos de índices pares e impares. Para ver esto vamos a definir

$$\begin{aligned} h^* &= 2h, \quad x_k^* = x_0 + kh^*, \quad u_0 = v_0 = y_0, \\ u_k &= y_{2k}, \quad v_k = y_{2k+1} - hf(x_{2k}, y_{2k}) = \frac{1}{2}(y_{2k+1} + y_{2k-1}). \end{aligned} \tag{2.15}$$

Entonces, utilizando lo anterior, el método (2.14) se puede reescribir de la siguiente manera:

$$\begin{aligned} u_{k+1} &= y_{2k+2} = y_{2k} + 2hf(x_{2k+1}, y_{2k+1}) = u_k + h^* f(x_0 + (2k+1)h, v_k + hf(x_{2k}, y_{2k})) \\ &= u_k + h^* f(x_0 + kh^* + \frac{h^*}{2}, v_k + \frac{h^*}{2} f(x_0 + kh^*, u_k)) \\ &= u_k + h^* f(x_k^* + \frac{h^*}{2}, v_k + \frac{h^*}{2} f(x_k^*, u_k)). \end{aligned}$$

Ahora,

$$\begin{aligned} v_{k+1} &= y_{2k+3} - hf(x_{2k+2}, y_{2k+2}) = y_{2k+1} + hf(x_{2k+2}, y_{2k+2}) \\ &= y_{2k+1} + hf(x_{2k+2}, y_{2k+2}) + (hf(x_{2k}, y_{2k}) - hf(x_{2k}, y_{2k})) \\ &= v_k + \frac{h^*}{2}(f(x_0 + kh^*, u_k) + f(x_0 + (k+1)h^* + h^*, u_{k+1})) \\ &= v_k + \frac{h^*}{2}(f(x_k^*, u_k) + f(x_k^* + h^*, u_{k+1})). \end{aligned}$$

Juntando ambas cosas tenemos que

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ v_k \end{pmatrix} + h^* \begin{pmatrix} f(x_k^* + \frac{h^*}{2}, v_k + \frac{h^*}{2} f(x_k^*, u_k)) \\ \frac{1}{2}(f(x_k^*, u_k) + f(x_k^* + h^*, u_{k+1})) \end{pmatrix}. \tag{2.16}$$

Teorema 2.4. *El método* (2.16) *es un método simétrico.*

Demostración. Hay que ver que $\Phi = \Phi^*$, es decir, que intercambiando $u_{k+1} \leftrightarrow u_k$, $v_{k+1} \leftrightarrow v_k$, $h^* \leftrightarrow -h^*$ y $x_k^* \leftrightarrow x_k^* + h^*$ en (2.16) queda igual. Así pues, haciéndolo,

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = \begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} - h^* \begin{pmatrix} f(x_k^* + h^* - \frac{h^*}{2}, v_{k+1} - \frac{h^*}{2} f(x_k^* + h^*, u_{k+1})) \\ \frac{1}{2}(f(x_k^* + h^*, u_{k+1}) + f(x_k^*, u_k)) \end{pmatrix},$$

y, despejando,

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ v_k \end{pmatrix} + h^* \begin{pmatrix} f(x_k^* + \frac{h^*}{2}, v_{k+1} - \frac{h^*}{2} f(x_k^* + h^*, u_{k+1})) \\ \frac{1}{2}(f(x_k^* + h^*, u_{k+1}) + f(x_k^*, u_k)) \end{pmatrix}.$$

Vemos que la única diferencia con (2.16) está en la primera entrada de la matriz, pero

$$\begin{aligned} v_{k+1} - \frac{h^*}{2} f(x_k^* + h^*, u_{k+1}) &= v_k + \frac{h^*}{2}(f(x_k^*, u_k) + f(x_k^* + h^*, u_{k+1})) \\ &\quad - \frac{h^*}{2} f(x_k^* + h^*, u_{k+1}) \\ &= v_k + \frac{h^*}{2} f(x_k^*, u_k), \end{aligned}$$

quedando así demostrado el resultado. □

El método (2.16) es también consistente para la ecuación diferencial

$$\begin{aligned} u' &= f(x,v) \quad u(x_0) = y_0 \\ v' &= f(x,u) \quad v(x_0) = y_0, \end{aligned}$$

cuya solución es $u(x) = v(x) = y(x)$. Por ser un método simétrico podemos aplicar el teorema 1.7 y, entonces, el desarrollo asintótico del error global viene dado por

$$y(x) - u_n = \sum_{j=1}^{l} a_{2j}(x)(h^*)^{2j} + (h^*)^{2l+2} A(x, h^*) \tag{2.17}$$

$$y(x) - v_n = \sum_{j=1}^{l} b_{2j}(x)(h^*)^{2j} + (h^*)^{2l+2} B(x, h^*), \tag{2.18}$$

donde $a_{2j}(x_0) = b_{2j}(x_0) = 0$. De (2.15) y de (2.17) vemos que

$$y(x) - y_{2n} = \sum_{j=1}^{l} a_{2j}(x)(h^*)^{2j} + (h^*)^{2l+2} A(x, h^*),$$

por lo cual siempre que el número de pasos sea par, esto es $x = x_0 + 2nh$, y_n tendrá un desarrollo asintótico del error global en potencias pares de h. Es decir,

$$y(x) - y_n = \sum_{j=1}^{l} \widehat{a}_{2j}(x)(h)^{2j} + (h)^{2l+2} \widehat{A}(x, h),$$

donde $\widehat{a}_{2j}(x) = 2^{2j} a_{2j}(x)$ y $\widehat{A}(x,h) = 2^{2l+2} A(x, 2h)$.

De la misma manera tenemos que

$$S_h(x_0 + 2nh) = \frac{1}{4}(y_{2n-1} + 2y_{2n} + y_{2n+1}) = \frac{1}{2}(u_n + v_n), \tag{2.19}$$

que es el llamado paso suavizador, que tiene en cuenta la evaluación de f al final del último paso. Al incluir el paso suavizador la región de estabilidad del método de extrapolación aumenta (ver figura IV.2.3 de [5]). Sin embargo, la ganancia que se obtiene gracias a ello se equilibra con el aumento del coste del método, por lo que este paso no es de mucha importancia. En cualquier caso el desarrollo asintótico del error global que se obtiene es el siguiente.

Teorema 2.5. *Sea $f \in \mathcal{C}^{2l+2}$, entonces la solución numérica definida por* (2.14) *posee para $x = x_0 + 2nh$ un desarrollo asintótico del error global de la forma*

$$y(x) - S_h(x) = \sum_{j=1}^{l} e_{2j}(x) h^{2j} + h^{2l+2} C(x, h), \tag{2.20}$$

donde $e_{2j}(x_0) = 0$ y $C(x,h)$ está acotada para $x_0 \le x \le \overline{x}$ y $0 \le h \le h_0$.

Demostración. Basta sumar las expresiones (2.17) y (2.18),

$$2y(x) - u_n - v_n = \sum_{j=1}^{l}(a_j(x) + b_j(x))(h^*)^{2j} + (h^*)^{2l+2}(A(x,h^*) + B(x,h^*)),$$

que es lo mismo que decir que

$$y(x) - \frac{1}{2}(u_n + v_n) = \sum_{j=1}^{l}(a_j(x) + b_j(x))2^{2j-1}h^{2j} + 2^{2l+1}h^{2l+2}(A(x,2h) + B(x,2h)).$$

Usando (2.19) queda probado el teorema tomando $e_{2j}(x) = (a_j(x) + b_j(x))2^{2j-1}$ y $C(x,h) = 2^{2l+1}(A(x,2h) + B(x,2h))$, ya que $(a_j(x_0) + b_j(x_0))2^{2j-1} = 0$ y $2^{2l+1}(A(x,2h) + B(x,2h))$ está acotada para $x_0 \leq x \leq \overline{x}$ y $0 \leq h \leq h_0$. □

El método de Gragg puede utilizarse como base de un método de extrapolación, igual que los métodos del ejemplo 2.5, con la condición de que la sucesión de número de pasos $\{n_i\}$ elegida tenga los n_i pares. Así pues, las sucesiones (2.7), (2.8) y (2.9) quedan como

$$2, 4, 8, 16, 32, 64, 128, 256, \cdots \tag{2.21}$$

$$2, 4, 6, 8, 12, 16, 24, 32, 48, \cdots \tag{2.22}$$

$$2, 4, 6, 8, 10, 12, 14, 16, 18, \cdots \tag{2.23}$$

respectivamente, con

$$T_{i,1} := S_{h_i}(x_0 + n_i h_i), \quad h_i = \frac{H}{n_i}.$$

Se puede por tanto calcular las expresiones de extrapolación $T_{i,j}$ con la expresión del algoritmo de Aitken-Neville para métodos simétricos (2.13). Vamos a ver un ejemplo de ello.

Ejemplo 2.6. *Vamos a aplicar el algoritmo* (2.14) *al problema del ejemplo 2.5, con las nuevas sucesiones* (2.21), (2.22) *y* (2.23). *Las tres primeras gráficas corresponden al algoritmo sin paso suavizador* (2.19) *(cuyo trabajo numérico en términos de evaluaciones de función vendrá dado, igual que con el método de Euler explícito, por* $1 + n_j - 1 + n_{j-1} - 1 + \cdots + n_{j-k+1} - 1$*) y, las tres últimas, al algoritmo con paso suavizador (cuyo trabajo numérico vendrá dado por* $1 + n_j + n_{j-1} + \cdots + n_{j-k+1}$*).*

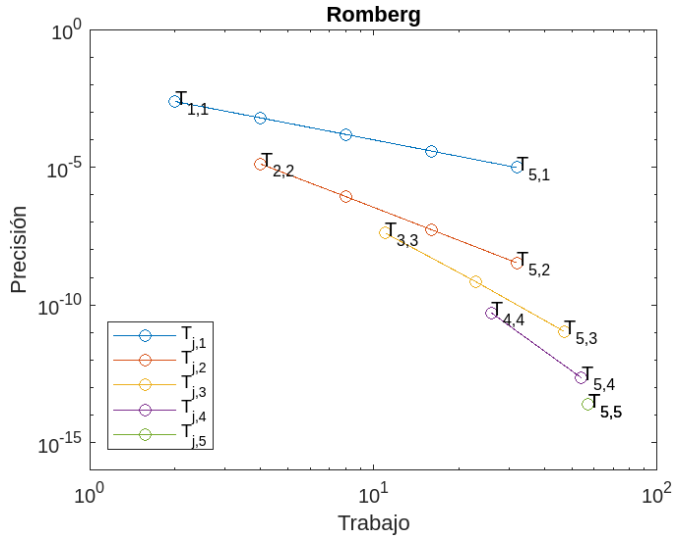

Figura 2.13: $T_{j,k}$ para el método de Gragg sin paso suavizador con la sucesión de Romberg.

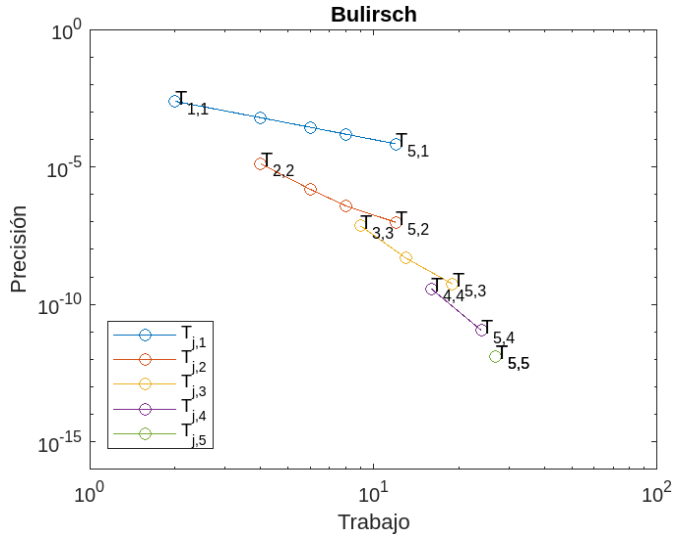

Figura 2.14: $T_{j,k}$ para el método de Gragg sin paso suavizador con la sucesión de Bulirsch.

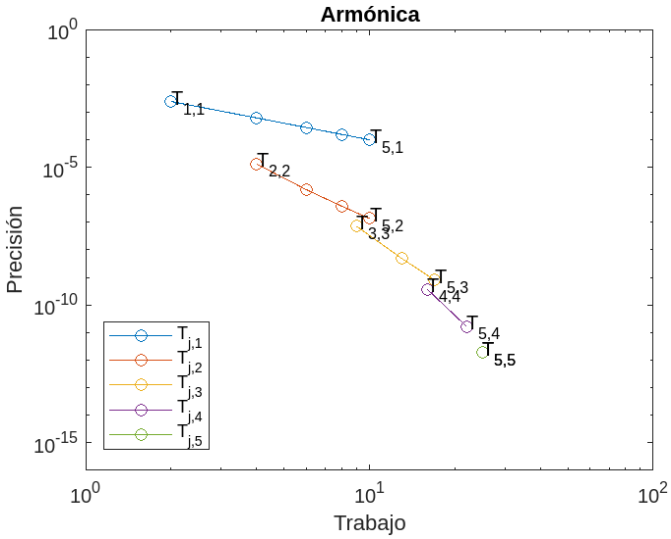

Figura 2.15: $T_{j,k}$ para el método de Gragg sin paso suavizador con la sucesión Armónica.

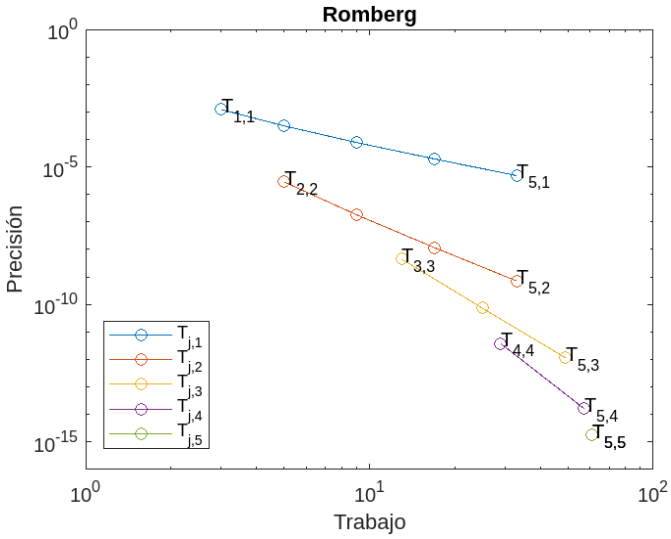

Figura 2.16: $T_{j,k}$ para el método de Gragg con paso suavizador con la sucesión de Romberg.

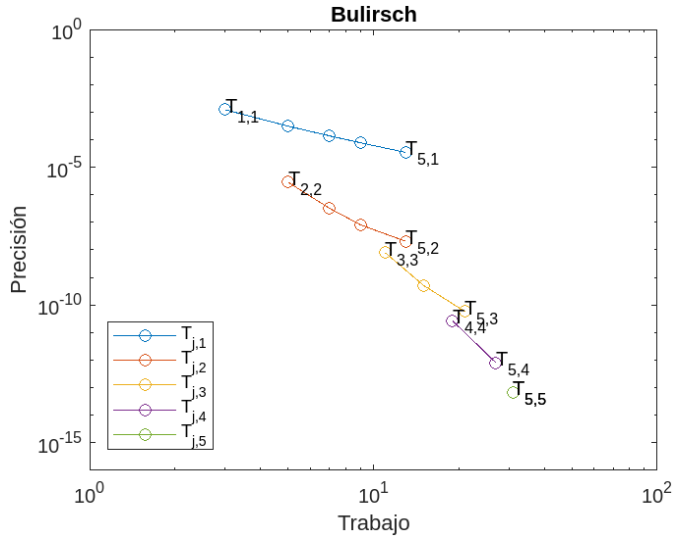

Figura 2.17: $T_{j,k}$ para el método de Gragg con paso suavizador con la sucesión de Bulirsch.

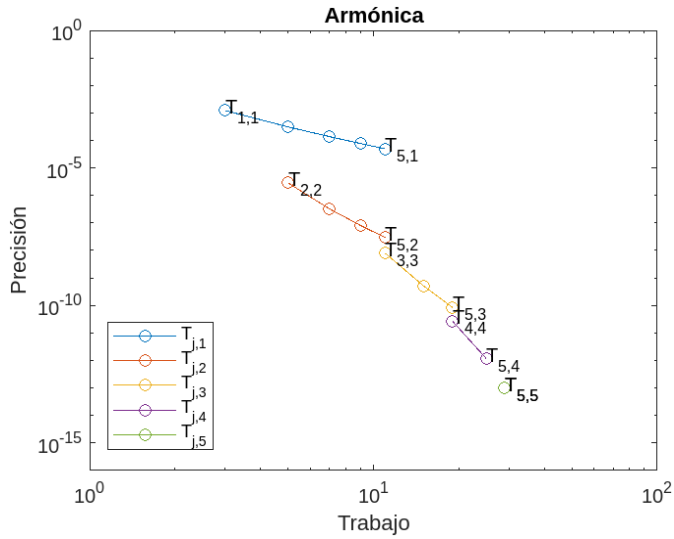

Figura 2.18: $T_{j,k}$ para el método de Gragg con paso suavizador con la sucesión Armónica.

Los errores obtenidos son del mismo orden de los de las gráficas de la regla implícita del punto medio; pero, con la gran ventaja de que este método es explícito, con lo que el coste computacional por paso es mucho menor. Observamos también que los resultados con paso suavizador son un poco más precisos que los resultados sin paso suavizador, pero también tienen un coste un poco mayor.

Al igual que antes procedemos a corroborar el teorema 2.2 con las figuras 2.19, 2.20 y 2.21, donde se representa el error en $t = 1$ frente a la longitud de paso H. Y esto para las nuevas sucesiones (2.21), (2.22) *y* (2.23).

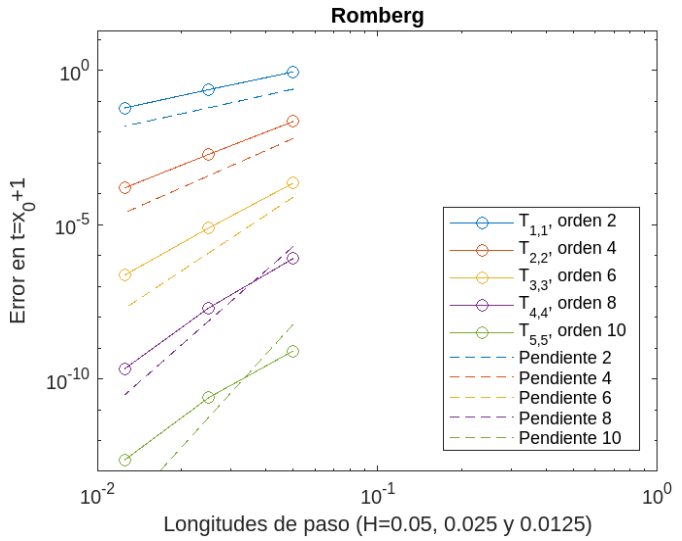

Figura 2.19: Orden de los métodos $T_{k,k}$ para el método de Gragg con paso suavizador con la sucesión de Romberg.

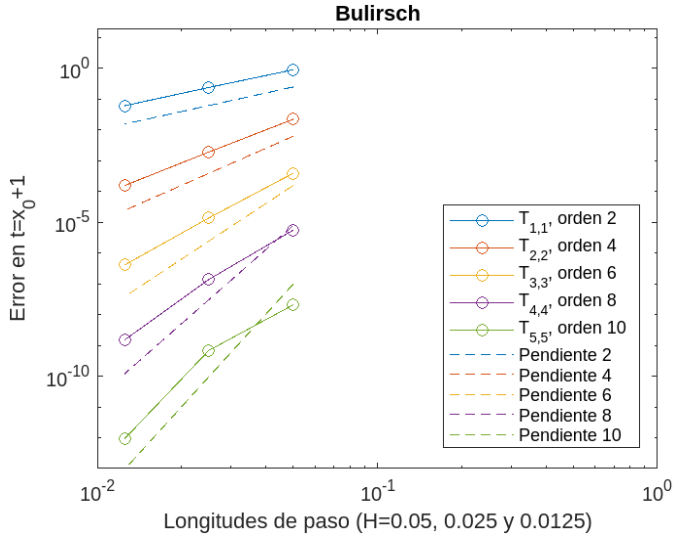

Figura 2.20: Orden de los métodos $T_{k,k}$ para el método de Gragg con paso suavizador con la sucesión de Bulirsch.

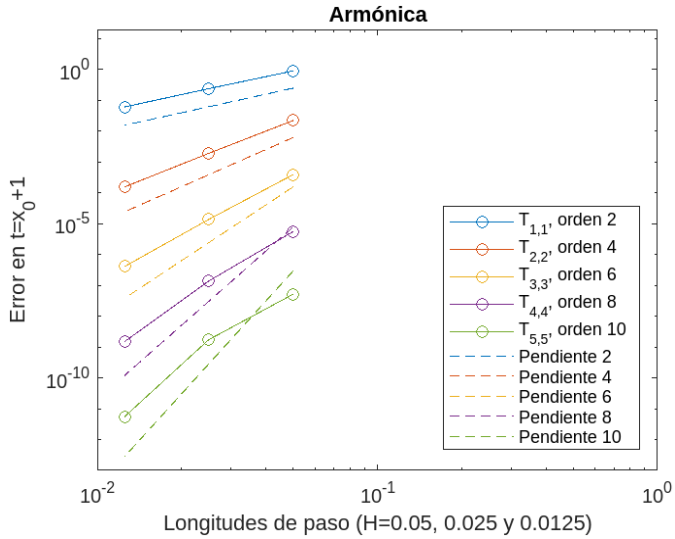

Figura 2.21: Orden de los métodos $T_{k,k}$ para el método de Gragg con paso suavizador con la sucesión Armónica.

Capítulo 3

Control del orden y del tamaño de paso con el método GBS

Los métodos de extrapolación permiten la libertad no sólo de cambiar el tamaño de paso sino también de cambiar el orden en cada paso, es decir, el número de columnas de (2.6). Tomando como método de base el método de Gragg, si calculamos las primeras k filas de (2.6), tenemos (por el teorema 2.2) $T_{k,k}$ como la aproximación de mayor orden, $2k$, y $T_{k,k-1}$ como una aproximación de orden $2k-2$. Puesto que los errores locales serían respectivamente $\mathcal{O}(H^{2k+1})$ y $\mathcal{O}(H^{2k-1})$, resulta natural utilizar

$$\begin{aligned} err_k &= \|T_{k,k-1} - T_{k,k}\| = \|(T_{k,k-1} - y(x_0+H)) + (y(x_0+H) - T_{k,k})\| \\ &\leq \mathcal{O}(H^{2k-1}) + \mathcal{O}(H^{2k+1}) \approx C \cdot H^{2k-1}, \end{aligned}$$

como estimación del error local cometido con $T_{k,k-1}$ para controlar el tamaño del paso. La norma elegida no es de mucha importancia. Si queremos que el error local en cada paso se mantenga por debajo de una tolerancia fija, TOL, tendríamos que elegir la longitud de paso de forma que $C \cdot H_k^{2k-1} = TOL$, por lo que

$$H_k = H \cdot 0{,}94 \cdot \left(\frac{TOL}{err_k}\right)^{1/(2k-1)}, \tag{3.1}$$

puede ser una opción, siendo 0,94 un factor de seguridad que hemos añadido.

En cuanto al orden óptimo, tenemos que medir el trabajo realizado con diferentes órdenes para intentar minimizar el coste computacional. El trabajo de $T_{k,k}$, al que denotaremos por A_k, se puede medir como hemos hecho antes, es decir, midiendo el número de evaluaciones de función. Para el algoritmo GBS con paso suavizador tenemos una forma recursiva de calcularlo, que es

$$\begin{aligned} A_1 &= n_1 + 1 \\ A_k &= A_{k-1} + n_k. \end{aligned}$$

Sin embargo, un gran trabajo A_k se puede compensar con un gran tamaño de paso H_k, por lo que vamos a considerar como una mejor medida del trabajo al trabajo por unidad de longitud de paso dada, es decir,

$$W_k = \frac{A_k}{H_k}.$$

El objetivo es encontrar un orden que minimice W_k.

Vamos a describir el algoritmo para encontrar el tamaño de paso y el orden óptimos. Para ello, en primer lugar, veamos el siguiente resultado.

Teorema 3.1.

$$y(x_0 + H) - T_{j,k} = \frac{(-1)^{k+1}}{n_j^2 \cdot \ldots \cdot n_{j-k+1}^2} e'_{2k}(x_0) H^{2k+1} + \mathcal{O}(H^{2k+2}), \tag{3.2}$$

donde e_{2k} son las funciones en (2.20).

Demostración. Basta tener en cuenta que, por la clásica fórmula sobre el error de interpolación, si p interpola a f en los nodos $h_j^2, \cdots, h_{j-k+1}^2$ y f es de clase $\mathcal{C}^k$,

$$f(t) - p(t) = \frac{(t - h_j^2) \cdots (t - h_{j-k+1}^2) f^{(k)}(\xi)}{k!},$$

para cierto valor ξ entre $t, h_j^2, \cdots, h_{j-k-1}^2$. Tomando $t = 0$ y teniendo en cuenta que $p(0) = T_{j,k}$ para el polinomio p tal que $p(h_i^2) = T_{i,1} = S_{h_i}(x + H)$ para $i = j, \cdots, j - k + 1$, se tiene que

$$f(0) - T_{j,k} = \frac{(-1)^k H^{2k}}{n_j^2 \cdot \ldots \cdot n_{j-k+1}^2} f^{(k)}(\xi),$$

para toda función f suficientemente regular tal que

$$f(h_i^2) = S_{h_i}(x + H).$$

Por tanto basta tomar $f(u) = S_{\sqrt{u}}(x + H)$, que teniendo en cuenta (2.20) puedo escribir como

$$f(u) = y(x + H) - \sum_{j=1}^{k} e_{2j}(x + H) u^j + \mathcal{O}(u^{k+1}).$$

Por tanto, $f(0) = y(x + H)$ y $f^{(k)}(\xi) = -k! e_{2k}(x + H) + \mathcal{O}(H^2)$. Usando que $e_{2k}(x) = 0$, se obtiene que $f^{(k)}(\xi) = -k! e'_{2k}(x) H + \mathcal{O}(H^2)$, de donde se obtiene el resultado. □

Partimos de un tamaño de paso H y un índice $k > 2$ dados. Ahora calculamos las $k - 1$ primeras filas de (2.6) y también los valores err_{k-2}, H_{k-2}, W_{k-2} (si $k > 3$), err_{k-1}, H_{k-1} y W_{k-1}. Veamos los distintos pasos con detalle:

Paso 1. *Convergencia en la fila $k-1$. Si $err_{k-1} \leq TOL$ aceptamos $T_{k-1,k-1}$ como la solución numérica y continuamos con los nuevos valores de la siguiente forma (nos quedamos con el mismo orden si la tendencia sobre el k anterior ha sido a mejorar el coste. Sino, lo bajamos):*

$$\begin{aligned} k_{new} &= \begin{cases} k & si\ W_{k-1} < 0{,}94 \cdot W_{k-2}, \\ k-1 & si\ W_{k-1} \geq 0{,}94 \cdot W_{k-2}, \end{cases} \\ H_{new} &= \begin{cases} H_{k_{new}} & si\ k_{new} = k-1, \\ H_{k-1}(A_k/A_{k-1}) & si\ k_{new} = k. \end{cases} \end{aligned} \tag{3.3}$$

En (3.3) *la elección de H_{new} no es trivial si $k_{new} = k$. Nos interesa evitar el cálculo de err_k. De esta manera, H_k y W_k permanecen desconocidos. Sin embargo, como se supone que k está cerca del valor óptimo tenemos que $W_k \approx W_{k-1}$, con lo que de la fórmula $\frac{A_k}{H_k} = \frac{A_{k-1}}{H_{k-1}}$ despejamos H_k, que es menos costoso que calcularlo a través de* (3.1).

Paso 2. *Monitor de convergencia. Si $err_{k-1} > TOL$ tenemos que ver si podemos esperar convergencia en la fila k. Observamos por el teorema 3.1, que*

$$\begin{aligned} \|T_{k,k-2} - T_{k,k-1}\| &\approx \frac{e'_{2k-4}(x_0)H^{2k-3}}{n_k^2 \cdot \ldots \cdot n_3^2} + \mathcal{O}(H^{2k-2}) \\ &\approx \left(\frac{n_2}{n_k}\right)^2 \frac{e'_{2k-4}(x_0)H^{2k-3}}{n_{k-1}^2 \cdot \ldots \cdot n_2^2} + \mathcal{O}(H^{2k-2}) \approx \left(\frac{n_2}{n_k}\right)^2 err_{k-1}. \end{aligned} \tag{3.4}$$

Desgraciadamente de (3.2) *vemos que err_k no se puede comparar fácilmente con* (3.4) *debido a que en lugar de $e'_{2k-4}(x_0)$ tendríamos $e'_{2k-2}(x_0)$, pero si asumimos que err_k es $(H/n_2)^2$ veces* (3.4)*, entonces $err_k \approx (H/n_k)^2 err_{k-1}$. Si*

$$err_{k-1} > \left(\frac{n_k}{H}\right)^2 TOL,$$

reiniciamos el proceso con $k_{new} = k-1$ y calculamos $H_{k_{new}}$ según (3.1)*. Sino continuamos.*

Paso 3. *Convergencia en la fila k. Si $err_k \leq TOL$, aceptamos $T_{k,k}$ y procedemos como sigue*

$$\begin{aligned} k_{new} &= \begin{cases} k-1 & si\ W_{k-1} < 0{,}94 \cdot W_k, \\ k+1 & si\ W_k < 0{,}94 \cdot W_{k-1}, \\ k & en\ el\ resto\ de\ casos, \end{cases} \\ H_{new} &= \begin{cases} H_{k_{new}} & si\ k_{new} \leq k, \\ H_k(A_{k+1}/A_k) & si\ k_{new} = k+1. \end{cases} \end{aligned} \tag{3.5}$$

Paso 4. *Segundo monitor de convergencia. Si $err_k > TOL$ y*

$$err_k > \left(\frac{n_{k+1}}{H}\right)^2 TOL,$$

rechazamos el paso y reiniciamos el proceso con $k_{new} = k-1$ y calculamos $H_{k_{new}}$ según (3.1)*. Sino continuamos.*

Paso 5. *Esperanza de convergencia en la fila $k+1$. Calculamos $T_{k+1,k+1}$, err_{k+1}, H_{k+1} y W_{k+1}. Si $err_{k+1} \leq TOL$ aceptamos $T_{k+1,k+1}$ y hacemos la siguiente propuesta para el paso siguiente.*

$$\begin{aligned} k_{new} &= \begin{cases} k-1 & si\ W_k < 0{,}94 \cdot W_{k+1}, \\ k+1 & si\ W_{k+1} < 0{,}94 \cdot W_k, \\ k & en\ el\ resto\ de\ casos, \end{cases} \\ H_{new} &= H_{k_{new}} \end{aligned} \tag{3.6}$$

Paso 6. *Tercer monitor de convergencia. Si $err_{k+1} > TOL$ el paso es rechazado y repetimos con $k_{new} = k$ y el H_{new} de* (3.1).

Para utilizar correctamente el algoritmo hay que tener en cuenta primero que, por razones de capacidad, es interesante fijar un k_{max} para limitar el número de columnas de la tabla de extrapolación que se calcularán como máximo, de forma que $2 \leq k_{new} \leq k_{max} - 1$.

Veamos ahora en un par de ejemplos cómo es el estudio numérico para el algoritmo previo.

Ejemplo 3.1. *Consideramos el modelo matemático presentado por Lefever y Nicolis [6] para estudiar e interpretar fenómenos biológicos producidos por reacciones multimoleculares. Este viene dado por:*

$$\begin{aligned} y_1' &= A + y_1^2 y_2 - (B+1) y_1, \\ y_2' &= B y_1 + y_1^2 y_2, \end{aligned}$$

donde A y B son constantes positivas referidas a las concentraciones de dos sustancias. Si $B > A^2 + 1$ (como es el caso tomado posteriormente) se sabe que existe un ciclo límite. Mostraremos la solución, el orden y el tamaño de paso según el algoritmo de paso y orden variable descrito para valores de los parámetros $A = 1$ y $B = 3$. Más concretamente:

$$\begin{aligned} y_1' &= 1 + y_1^2 y_2 - 4 y_1, \\ y_2' &= 3 y_1 + y_1^2 y_2, \\ y_1(0) &= 3/2, \quad 0 \leq x \leq 20, \\ y_2(0) &= 3, \quad 0 \leq x \leq 20. \end{aligned}$$

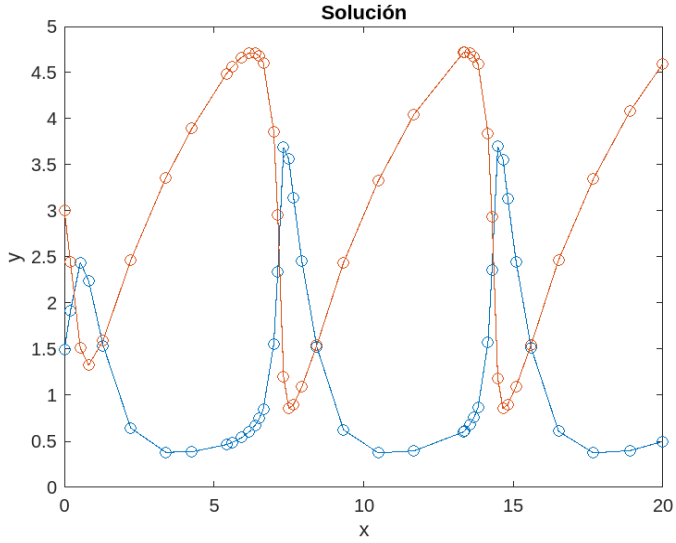

Figura 3.1: Solución aproximada del ejemplo 3.1, con TOL=10^{-5}.

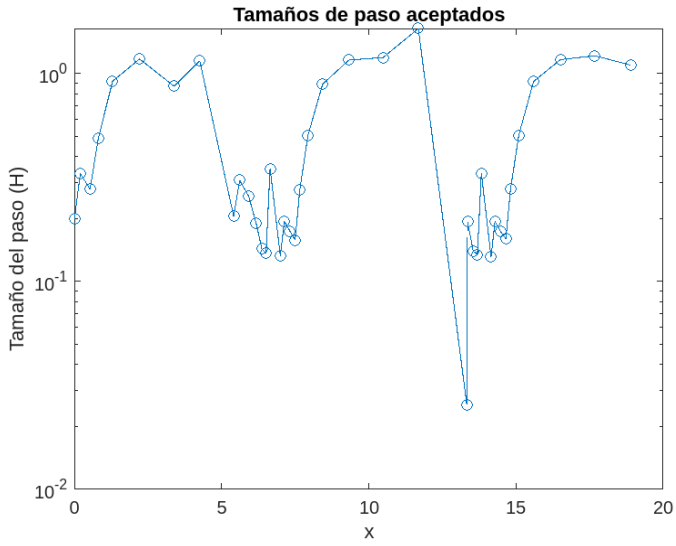

Figura 3.2: Longitudes de paso utilizadas para el ejemplo 3.1, con TOL=10^{-5}.

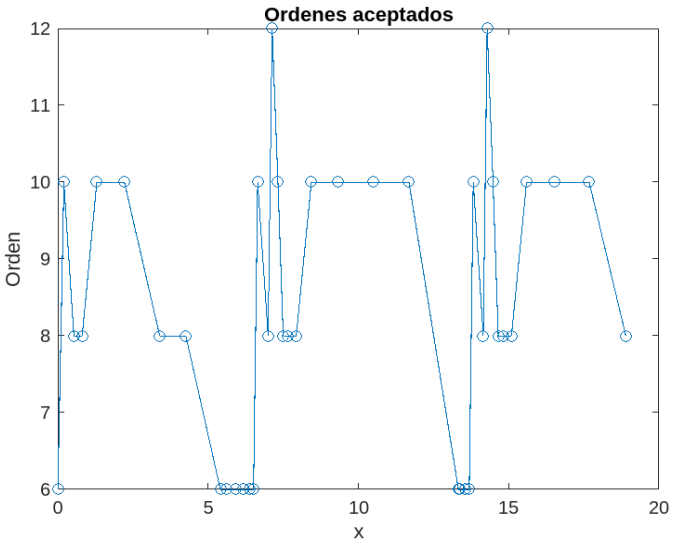

Figura 3.3: Órdenes utilizados para el ejemplo 3.1, con TOL=10^{-5}.

La imagen 3.1 muestra las dos componentes de la solución (obtenida con $TOL = 10^{-5}$). Las imágenes 3.2 y 3.3 muestran los tamaños de paso y los órdenes aceptados para $TOL = 10^{-5}$. Observamos que nuestro algoritmo elige automáticamente un tamaño de paso y un orden adecuado según la situación. En particular, en los puntos de mayor variabilidad de la solución, las longitudes de paso tomadas son más pequeñas y los órdenes más grandes.

Ejemplo 3.2. *También hemos considerado el siguiente problema, con una discontinuidad de la primera derivada en $x = 0$ y dos de la segunda derivada en $x = \pm 1$.*

$$\begin{aligned} y' &= -sign(x)|1 - |x||y^2, \\ y(-2) &= 2/3, \quad -2 \le x \le 2. \end{aligned}$$

En este caso podemos calcular fácilmente la solución exacta. Si $-2 < x < -1$,

$$y' = -(1+x)y^2, \quad \frac{y'}{y^2} = -(1+x), \quad \frac{-1}{y} = \frac{-(1+x)^2}{2} + C.$$

Como $y(-2) = 2/3$, entonces

$$y(x) = \frac{2}{(1+x)^2 + 2}.$$

Si $-1 < x < 0$,

$$y' = (1+x)y^2, \quad \frac{y'}{y^2} = 1 + x, \quad \frac{-1}{y} = \frac{(1+x)^2}{2} + C.$$

Como $y(-1) = 1$ (por continuidad), entonces

$$y(x) = \frac{-2}{(1+x)^2 - 2}.$$

Si $0 < x < 1$,

$$y' = -(1-x)y^2, \quad \frac{y'}{y^2} = -1 + x, \quad \frac{-1}{y} = \frac{(x-1)^2}{2} + C.$$

Como $y(0) = 2$ *(por continuidad), entonces*

$$y(x) = \frac{-2}{(x-1)^2 - 2}.$$

Si $1 < x < 2$,

$$y' = -(x-1)y^2, \quad \frac{y'}{y^2} = 1 - x, \quad \frac{-1}{y} = \frac{-(1-x)^2}{2} + C.$$

Como $y(1) = 1$ *(por continuidad), entonces*

$$y(x) = \frac{2}{(1-x)^2 + 2}.$$

Con esta información podemos calcular los errores cometidos.

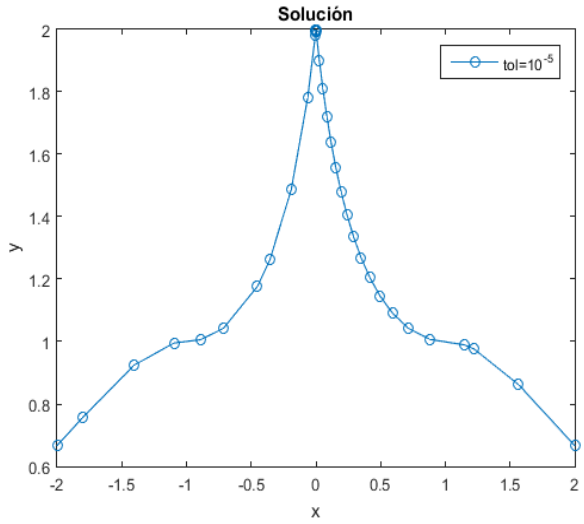

Figura 3.4: Solución aproximada del ejemplo 3.2, con TOL=10^{-5}.

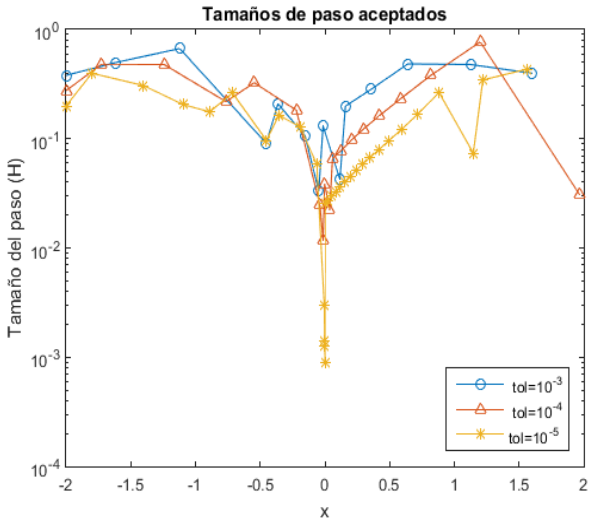

Figura 3.5: Longitudes de paso utilizadas para el ejemplo 3.2, con TOL=$10^{-3}, 10^{-4}, 10^{-5}$.

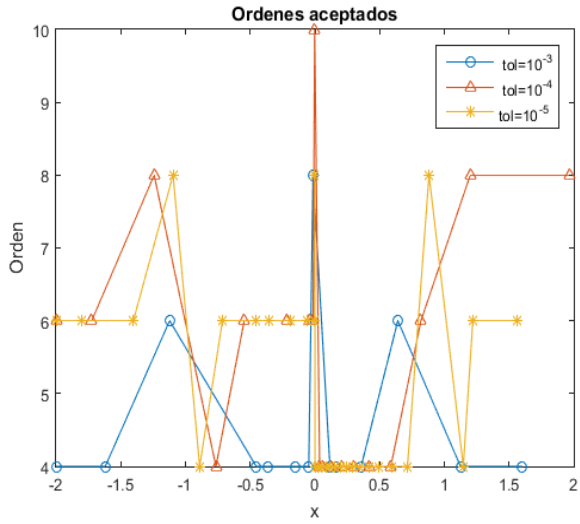

Figura 3.6: Órdenes utilizados para el ejemplo 3.2, con TOL=$10^{-3}, 10^{-4}, 10^{-5}$.

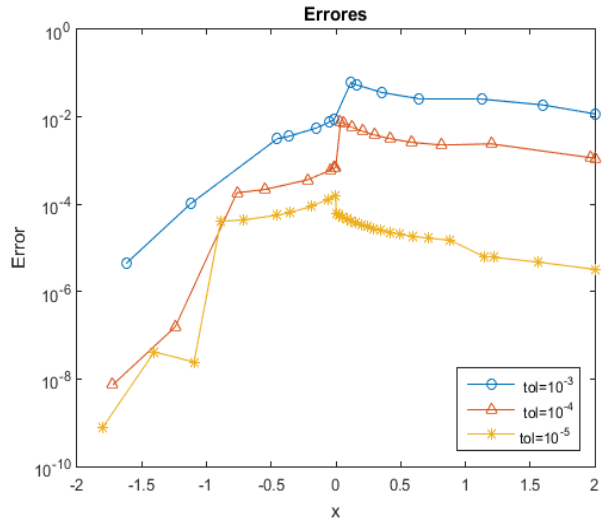

Figura 3.7: Errores producidos en el ejemplo 3.2, con TOL=$10^{-3}, 10^{-4}, 10^{-5}$.

En las figuras 3.4, 3.5, 3.6 y 3.7 mostramos la solución, el orden, el tamaño de paso y los errores según el algoritmo previo.

Vemos cómo se comporta el algoritmo cerca de las discontinuidades. Cuando el algoritmo detecta las discontinuidades, el orden aumenta y el tamaño de paso disminuye, de modo que la integración salva esos puntos de manera bastante eficiente. Observamos también como se adapta el algoritmo según la tolerancia, disminuyendo los errores conforme disminuye esta.

Apéndice A

Programas de Matlab

En este apéndice se incluyen los códigos de los distintos programas de Matlab que hemos ido utilizando a lo largo del trabajo para generar las figuras incluidas en este.

A.1. Primer programa

He utilizado el siguiente programa para generar las figuras 2.1, 2.2 2.3.

```
% Datos iniciales
x0 = pi/6;
y0 = 2/sqrt(3);
H = 0.2;
N = 8; % 6
n=[1,2,4,8,16,32,64,128]; %Romberg
m=[1,2,3,4,6,8,12,16]; %Bulirsch
w=[1,2,3,4,5,6,7,8]; %Arm nica

% Inicializaci n de variables
Tn = zeros(N, N);
trabajon = zeros(N, N);
errorn = zeros(N, N);

Tm = zeros(N, N);
trabajom = zeros(N, N);
errorm = zeros(N, N);

Tw = zeros(N, N);
trabajow = zeros(N, N);
errorw = zeros(N, N);
```

```
22
23 %Euler para T(j,1)
24 for i=1:N
25     h=H/n(i);
26     x=x0;
27     y=y0;
28     for j=1:n(i)
29     y=y+h*((-y*sin(x)+2*tan(x))*y);
30     x=x+h;
31     end
32     Tn(i,1)=y;
33
34 end
35
36 for i=1:N
37     h=H/m(i);
38     x=x0;
39     y=y0;
40     for j=1:m(i)
41     y=y+h*((-y*sin(x)+2*tan(x))*y);
42     x=x+h;
43     end
44     Tm(i,1)=y;
45
46 end
47
48 for i=1:N
49     h=H/w(i);
50     x=x0;
51     y=y0;
52     for j=1:w(i)
53     y=y+h*((-y*sin(x)+2*tan(x))*y);
54     x=x+h;
55     end
56     Tw(i,1)=y;
57
58 end
59
60 % Inicializa la primera columna con los valores num ricos
61 errorn(:,1) = abs(Tn(:, 1) - 1./cos(x0+H));
62 trabajon(:,1) = n(:);
63
64 errorm(:,1) = abs(Tm(:, 1) - 1./cos(x0+H));
65 trabajom(:,1) = m(:);
66
67 errorw(:,1) = abs(Tw(:, 1) - 1./cos(x0+H));
68 trabajow(:,1) = w(:);
```

```
69
70  % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
71  % calculo el error y el trabajo
72  for k = 1:N-1
73      for j = k+1:N
74          Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / (n(j)
                / n(j-k) - 1);
75          errorn(j,k+1) = abs(Tn(j, k+1) - 1./cos(x0+H));
76          trabajon(j,k+1)=sum(n(j-k+1:j))-(k-1);
77
78      end
79  end
80
81  for k = 1:N-1
82      for j = k+1:N
83          Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / (m(j)
                / m(j-k) - 1);
84          errorm(j,k+1) = abs(Tm(j, k+1) - 1./cos(x0+H));
85          trabajom(j,k+1)=sum(m(j-k+1:j))-(k-1);
86
87      end
88  end
89
90  for k = 1:N-1
91      for j = k+1:N
92          Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / (w(j)
                / w(j-k) - 1);
93          errorw(j,k+1) = abs(Tw(j, k+1) - 1./cos(x0+H));
94          trabajow(j,k+1)=sum(w(j-k+1:j))-(k-1);
95
96      end
97  end
98
99  % Gr fica en escala doble logar tmica
100
101 figure(1)
102 clf
103 loglog(trabajon, errorn, '-o');
104 xlabel('Trabajo');
105 ylabel('Precisi n');
106 title('Romberg');
107 for i=1:N
108     txt=['T_{',num2str(i) ',' num2str(i),'}'];
109     text(trabajon(i,i),errorn(i,i),txt)
110     txt=['T_{',num2str(N) ',' num2str(i),'}'];
111     text(trabajon(N,i),errorn(N,i),txt)
```

```
112 end
113 axis([1 400 10.^-15 1]);
114 legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}','T_{j
    ,6}','T_{j,7}','T_{j,8}'},'Location','southwest','
    Orientation','vertical')
115
116 figure(2)
117 clf
118 loglog(trabajom, errorm, '-o');
119 xlabel('Trabajo');
120 ylabel('Precisi n');
121 title('Bulirsch');
122 for i=1:N
123     txt=['T_{',num2str(i) ',' num2str(i),'}'];
124     text(trabajom(i,i),errorm(i,i),txt)
125     txt=['T_{',num2str(N) ',' num2str(i),'}'];
126     text(trabajom(N,i),errorm(N,i),txt)
127 end
128 axis([1 75 10.^-13 1]);
129 legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}','T_{j
    ,6}','T_{j,7}','T_{j,8}'},'Location','southwest','
    Orientation','vertical')
130
131
132 figure(3)
133 clf
134 loglog(trabajow, errorw, '-o');
135 xlabel('Trabajo');
136 ylabel('Precisi n');
137 title('Arm nica');
138 for i=1:N
139     txt=['T_{',num2str(i) ',' num2str(i),'}'];
140     text(trabajow(i,i),errorw(i,i),txt)
141     txt=['T_{',num2str(N) ',' num2str(i),'}'];
142     text(trabajow(N,i),errorw(N,i),txt)
143 end
144 axis([1 40 10.^-12 1]);
145 legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}','T_{j
    ,6}','T_{j,7}','T_{j,8}'},'Location','southwest','
    Orientation','vertical')
```

A.2. Segundo programa

He utilizado el siguiente programa para generar las figuras 2.4, 2.5 2.3.

```
%Empiezo cambiando la tolerancia de fsolve para no tener
    problemas
options = optimoptions('fsolve', 'OptimalityTolerance', 1e-100,
     'StepTolerance', 1e-100, 'FunctionTolerance', 1e-100, '
    MaxIterations', 1e100, 'MaxFunctionEvaluations', 1e100);
% Datos iniciales
f = @(x, y) (-y*sin(x) + 2*tan(x))*y;
x0 = pi/6;
y0 = 2/sqrt(3);
H = 0.2;
N = 5; % 5
n=[1,2,4,8,16]; %Romberg
m=[1,2,3,4,8]; %Bulirsch
w=[1,2,3,4,5]; %Arm nica

% Inicializaci n de variables
Tn = zeros(N, N);
trabajon = zeros(N, N);
errorn = zeros(N, N);

Tm = zeros(N, N);
trabajom = zeros(N, N);
errorm = zeros(N, N);

Tw = zeros(N, N);
trabajow = zeros(N, N);
errorw = zeros(N, N);

%Regla punto medio impl cita para T(j,1)
for i=1:N
    h=H/n(i);
    x=x0;
    y=y0;
    for j=1:n(i)
    yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
        0, options);
    y=yfinal;
    x=x+h;
    end
    Tn(i,1)=y;

end

for i=1:N
    h=H/m(i);
    x=x0;
```

```
43      y=y0;
44      for j=1:m(i)
45      yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
            0, options);
46      y=yfinal;
47      x=x+h;
48      end
49      Tm(i,1)=y;
50
51  end
52
53  for i=1:N
54      h=H/w(i);
55      x=x0;
56      y=y0;
57      for j=1:w(i)
58      yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
            0, options);
59      y=yfinal;
60      x=x+h;
61      end
62      Tw(i,1)=y;
63
64  end
65
66  % Inicializa la primera columna con los valores num ricos
67  errorn(:,1) = abs(Tn(:, 1) - 1./cos(x0+H));
68  trabajon(:,1) = n(:);
69
70  errorm(:,1) = abs(Tm(:, 1) - 1./cos(x0+H));
71  trabajom(:,1) = m(:);
72
73  errorw(:,1) = abs(Tw(:, 1) - 1./cos(x0+H));
74  trabajow(:,1) = w(:);
75
76  % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
77  % calculo el error y el trabajo
78  for k = 1:N-1
79      for j = k+1:N
80          Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / ((n(j
                ) / n(j-k))^2 - 1);
81          errorn(j,k+1) = abs(Tn(j, k+1) - 1./cos(x0+H));
82          trabajon(j,k+1)=sum(n(j-k+1:j));
83
84      end
85  end
```

```
86
87  for k = 1:N-1
88      for j = k+1:N
89          Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / ((m(j
                ) / m(j-k))^2 - 1);
90          errorm(j,k+1) = abs(Tm(j, k+1) - 1./cos(x0+H));
91          trabajom(j,k+1)=sum(m(j-k+1:j));
92
93      end
94  end
95
96  for k = 1:N-1
97      for j = k+1:N
98          Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
                ) / w(j-k))^2 - 1);
99          errorw(j,k+1) = abs(Tw(j, k+1) - 1./cos(x0+H));
100         trabajow(j,k+1)=sum(w(j-k+1:j));
101
102     end
103 end
104
105 % Gr fica en escala doble logar tmica
106
107 figure(1)
108 clf
109 loglog(trabajon, errorn, '-o');
110 xlabel('Trabajo');
111 ylabel('Precisi n');
112 title('Romberg');
113 for i=1:N
114     txt=['T_{',num2str(i) ',' num2str(i),'}'];
115     text(trabajon(i,i),errorn(i,i),txt)
116     txt=['T_{',num2str(N) ',' num2str(i),'}'];
117     text(trabajon(N,i),errorn(N,i),txt)
118 end
119 axis([1 100 10.^-15 1]);
120 legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
        Location','southwest','Orientation','vertical')
121
122 figure(2)
123 clf
124 loglog(trabajom, errorm, '-o');
125 xlabel('Trabajo');
126 ylabel('Precisi n');
127 title('Bulirsch');
128 for i=1:N
129     txt=['T_{',num2str(i) ',' num2str(i),'}'];
```

```
    text(trabajom(i,i),errorm(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajom(N,i),errorm(N,i),txt)
end
axis([1 100 10.^-15 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
   Location','southwest','Orientation','vertical')

figure(3)
clf
loglog(trabajow, errorw, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Arm nica');
for i=1:N
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajow(i,i),errorw(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajow(N,i),errorw(N,i),txt)
end
axis([1 100 10.^-15 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
   Location','southwest','Orientation','vertical')
```

A.3. Tercer programa

He utilizado el siguiente programa para generar las figuras 2.7, 2.8 2.9.

```
% Datos iniciales
H = zeros(1, 4);
H(1) = 0.05;
H(2) = 0.025;
H(3) = 0.0125;
H(4) = 0.00625;
n=[1,2,4,8,16,32]; %Romberg
m=[1,2,3,4,6,8]; %Bulirsch
w=[1,2,3,4,5,6]; %Arm nica
errorn = zeros(4, 6);
errorm = zeros(4, 6);
errorw = zeros(4, 6);


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:4
df=1/H(l); % calculo el n mero de pasos
```

```
for N=1:6 % calculo el error global de T_{N,N}
Tn = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/n(i);
    x=x0;
    y=y0;
    for j=1:n(i)
    y=y+h*((-y*sin(x)+2*tan(x))*y);
    x=x+h;
    end
    Tn(i,1)=y;

end
% Construyo la tabla de extrapolaci n con el algortimo de
    Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / (n(j)
            / n(j-k) - 1);
    end
end
y0=Tn(N,N);
x0=x0+H(l);

end
errorn(l,N) = abs(Tn(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(1)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30
    ";"#4DBEEE"])
loglog(H, errorn, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025, 0.0125 y 0.00625)');
ylabel('Error en t=x_{0}+1');
title('Romberg');
axis([10.^-2.3 1 10.^-12 20]);
hold on
loglog(H,100*H,'--')
loglog(H,300*H.^2,'--')
```

```
loglog(H,200*H.^3,'--')
loglog(H,1500*H.^4,'--')
loglog(H,1000*H.^5,'--')
loglog(H,700*H.^6,'--')
legend({'T_{1,1}, orden 1', 'T_{2,2}, orden 2', 'T_{3,3}, orden
    3', 'T_{4,4}, orden 4', 'T_{5,5}, orden 5', 'T_{6,6}, orden
    6', 'Pendiente 1', 'Pendiente 2', 'Pendiente 3', 'Pendiente
    4', 'Pendiente 5', 'Pendiente 6'},'Location','southeast','
   Orientation','vertical')



%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:4
df=1/H(l); % calculo el n mero de pasos
for N=1:6 % calculo el error global de T_{N,N}
Tm = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/m(i);
    x=x0;
    y=y0;
    for j=1:m(i)
    y=y+h*((-y*sin(x)+2*tan(x))*y);
    x=x+h;
    end
    Tm(i,1)=y;

end
% Construyo la tabla de extrapolaci n con el algortimo de
   Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / (m(j)
            / m(j-k) - 1);
    end
end
y0=Tm(N,N);
x0=x0+H(l);

end
errorm(l,N) = abs(Tm(N,N) - 1./cos((pi/6)+1));
end
```

```
end

% Gr fica en escala doble logar tmica
figure(2)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30
    ";"#4DBEEE"])
loglog(H, errorm, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025, 0.0125 y 0.00625)');
ylabel('Error en t=x_{0}+1');
title('Bulirsch');
axis([10.^-2.3 1 10.^-12 20]);
hold on
loglog(H,100*H,'--')
loglog(H,300*H.^2,'--')
loglog(H,300*H.^3,'--')
loglog(H,3000*H.^4,'--')
loglog(H,9000*H.^5,'--')
loglog(H,10000*H.^6,'--')
legend({'T_{1,1}, orden 1', 'T_{2,2}, orden 2', 'T_{3,3}, orden
    3', 'T_{4,4}, orden 4', 'T_{5,5}, orden 5', 'T_{6,6}, orden
    6', 'Pendiente 1', 'Pendiente 2', 'Pendiente 3', 'Pendiente
    4', 'Pendiente 5', 'Pendiente 6'},'Location','southeast','
   Orientation','vertical')


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:4
df=1/H(l); % calculo el n mero de pasos
for N=1:6 % calculo el error global de T_{N,N}
Tw = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/w(i);
    x=x0;
    y=y0;
    for j=1:w(i)
    y=y+h*((-y*sin(x)+2*tan(x))*y);
    x=x+h;
    end
    Tw(i,1)=y;

end
% Construyo la tabla de extrapolaci n con el algortimo de
```

```
    Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / (w(j)
            / w(j-k) - 1);
    end
end
y0=Tw(N,N);
x0=x0+H(l);

end
errorw(l,N) = abs(Tw(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(3)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30
    ";"#4DBEEE"])
loglog(H, errorw, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025, 0.0125 y 0.00625)');
ylabel('Error en t=x_{0}+1');
title('Arm nica');
axis([10.^-2.3 1 10.^-12 20]);
hold on
loglog(H,100*H,'--')
loglog(H,300*H.^2,'--')
loglog(H,300*H.^3,'--')
loglog(H,3000*H.^4,'--')
loglog(H,9000*H.^5,'--')
loglog(H,10000*H.^6,'--')
legend({'T_{1,1}, orden 1', 'T_{2,2}, orden 2', 'T_{3,3}, orden
    3', 'T_{4,4}, orden 4', 'T_{5,5}, orden 5', 'T_{6,6}, orden
    6', 'Pendiente 1', 'Pendiente 2', 'Pendiente 3', 'Pendiente
    4', 'Pendiente 5', 'Pendiente 6'},'Location','southeast','
   Orientation','vertical')
```

A.4. Cuarto programa

He utilizado el siguiente programa para generar las figuras 2.10, 2.11 2.12.

```
%Empiezo cambiando la tolerancia de fsolve para no tener
    problemas
```

```
options = optimoptions('fsolve', 'OptimalityTolerance', 1e-100,
    'StepTolerance', 1e-100, 'FunctionTolerance', 1e-100, '
   MaxIterations', 1e100, 'MaxFunctionEvaluations', 1e100);
% Datos iniciales
f = @(x, y) (-y*sin(x) + 2*tan(x))*y;
H = zeros(1, 3);
H(1) = 0.05;
H(2) = 0.025;
H(3) = 0.0125;
n=[1,2,4,8,16]; %Romberg
m=[1,2,3,4,8]; %Bulirsch
w=[1,2,3,4,5]; %Arm nica
errorn = zeros(3, 5);
errorm = zeros(3, 5);
errorw = zeros(3, 5);


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:3
df=1/H(l); % calculo el n mero de pasos
for N=1:5 % calculo el error global de T_{N,N}
Tn = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/n(i);
    x=x0;
    y=y0;
    for j=1:n(i)
    yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
       0, options);
    y=yfinal;
    x=x+h;
    end
    Tn(i,1)=y;

end
% Construyo la tabla de extrapolaci n con el algortimo de
   Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / ((n(j
           ) / n(j-k))^2 - 1);
    end
```

```
end
y0=Tn(N,N);
x0=x0+H(l);

end
errorn(l,N) = abs(Tn(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(1)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
loglog(H, errorn, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
ylabel('Error en t=x_{0}+1');
title('Romberg');
axis([10.^-2 1 10.^-12 20]);
hold on
loglog(H,200*H.^2,'--')
loglog(H,7000*H.^4,'--')
loglog(H,50000*H.^6,'--')
loglog(H,200000*H.^8,'--')
loglog(H,500000*H.^10,'--')
legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
    6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
    'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
   },'Location','southeast','Orientation','vertical')


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:3
df=1/H(l); % calculo el n mero de pasos
for N=1:5 % calculo el error global de T_{N,N}
Tm = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/m(i);
    x=x0;
    y=y0;
    for j=1:m(i)
    yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
       0, options);
    y=yfinal;
```

```
87      x=x+h;
88      end
89      Tm(i,1)=y;
90
91  end
92  % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
93  % calculo la soluci n
94  for k = 1:N-1
95      for j = k+1:N
96          Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / ((m(j
                ) / m(j-k))^2 - 1);
97      end
98  end
99  y0=Tm(N,N);
100 x0=x0+H(l);
101
102 end
103 errorm(l,N) = abs(Tm(N,N) - 1./cos((pi/6)+1));
104 end
105 end
106
107 % Gr fica en escala doble logar tmica
108 figure(2)
109 clf
110 colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
111 loglog(H, errorm, '-o');
112 xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
113 ylabel('Error en t=x_{0}+1');
114 title('Bulirsch');
115 axis([10.^-2 1 10.^-12 20]);
116 hold on
117 loglog(H,200*H.^2,'--')
118 loglog(H,8000*H.^4,'--')
119 loglog(H,100000*H.^6,'--')
120 loglog(H,2000000*H.^8,'--')
121 loglog(H,20000000*H.^10,'--')
122 legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
        6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
        'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
       },'Location','southeast','Orientation','vertical')
123
124
125 %%%%%%%%%%%%%%%%%%%%%%%%
126 for l=1:3
127 df=1/H(l); % calculo el n mero de pasos
128 for N=1:5 % calculo el error global de T_{N,N}
```

```
Tw = zeros(N, N);
x0 = pi/6;
y0 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%Euler para T(j,1)
for i=1:N
    h=H(l)/w(i);
    x=x0;
    y=y0;
    for j=1:w(i)
    yfinal=fsolve(@(yfinal) yfinal-y-h*f(x+h/2, (y+yfinal)/2),
       0, options);
    y=yfinal;
    x=x+h;
    end
    Tw(i,1)=y;

end
% Construyo la tabla de extrapolaci n con el algortimo de
   Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
           ) / w(j-k))^2 - 1);
    end
end
y0=Tw(N,N);
x0=x0+H(l);

end
errorw(l,N) = abs(Tw(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(3)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
loglog(H, errorw, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
ylabel('Error en t=x_{0}+1');
title('Arm nica');
axis([10.^-2 1 10.^-12 20]);
hold on
loglog(H,200*H.^2,'--')
loglog(H,8000*H.^4,'--')
```

```
loglog(H,100000*H.^6,'--')
loglog(H,2000000*H.^8,'--')
loglog(H,20000000*H.^10,'--')
legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
    6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
    'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
   },'Location','southeast','Orientation','vertical')
```

A.5. Quinto programa

He utilizado el siguiente programa para generar las figuras 2.13, 2.14 2.15.

```
% Datos iniciales
x0 = pi/6;
y00 = 2/sqrt(3);
H = 0.2;
N = 5; % 5
n=[2,4,8,16,32]; %Romberg
m=[2,4,6,8,12]; %Bulirsch
w=[2,4,6,8,10]; %Arm nica

% Inicializaci n de variables
Tn = zeros(N, N);
trabajon = zeros(N, N);
errorn = zeros(N, N);

Tm = zeros(N, N);
trabajom = zeros(N, N);
errorm = zeros(N, N);

Tw = zeros(N, N);
trabajow = zeros(N, N);
errorw = zeros(N, N);

%Gragg sin paso suavizador para T(j,1)
for i=1:N
    h=H/n(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:n(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    y0=y1;
```

```
    y1=y2;
    end
    Tn(i,1)=y0;

end

for i=1:N
    h=H/m(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:m(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    y0=y1;
    y1=y2;
    end
    Tm(i,1)=y0;

end

for i=1:N
    h=H/w(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:w(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    y0=y1;
    y1=y2;
    end
    Tw(i,1)=y0;

end

% Inicializa la primera columna con los valores num ricos
errorn(:,1) = abs(Tn(:, 1) - 1./cos(x0+H));
trabajon(:,1) = n(:);

errorm(:,1) = abs(Tm(:, 1) - 1./cos(x0+H));
trabajom(:,1) = m(:);

errorw(:,1) = abs(Tw(:, 1) - 1./cos(x0+H));
trabajow(:,1) = w(:);
```

```
81
82  % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
83  % calculo el error y el trabajo
84  for k = 1:N-1
85      for j = k+1:N
86          Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / ((n(j
                ) / n(j-k))^2 - 1);
87          errorn(j,k+1) = abs(Tn(j, k+1) - 1./cos(x0+H));
88          trabajon(j,k+1)=sum(n(j-k+1:j))-(k-1);
89
90      end
91  end
92
93  for k = 1:N-1
94      for j = k+1:N
95          Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / ((m(j
                ) / m(j-k))^2 - 1);
96          errorm(j,k+1) = abs(Tm(j, k+1) - 1./cos(x0+H));
97          trabajom(j,k+1)=sum(m(j-k+1:j))-(k-1);
98
99      end
100 end
101
102 for k = 1:N-1
103     for j = k+1:N
104         Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
                ) / w(j-k))^2 - 1);
105         errorw(j,k+1) = abs(Tw(j, k+1) - 1./cos(x0+H));
106         trabajow(j,k+1)=sum(w(j-k+1:j))-(k-1);
107
108     end
109 end
110
111 % Gr fica en escala doble logar tmica
112
113 figure(1)
114 clf
115 loglog(trabajon, errorn, '-o');
116 xlabel('Trabajo');
117 ylabel('Precisi n');
118 title('Romberg');
119 for i=1:N
120     txt=['T_{',num2str(i) ',' num2str(i),'}'];
121     text(trabajon(i,i),errorn(i,i),txt)
122     txt=['T_{',num2str(N) ',' num2str(i),'}'];
123     text(trabajon(N,i),errorn(N,i),txt)
```

```
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
    Location','southwest','Orientation','vertical')

figure(2)
clf
loglog(trabajom, errorm, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Bulirsch');
for i=1:N
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajom(i,i),errorm(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajom(N,i),errorm(N,i),txt)
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
    Location','southwest','Orientation','vertical')

figure(3)
clf
loglog(trabajow, errorw, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Arm nica');
for i=1:N
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajow(i,i),errorw(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajow(N,i),errorw(N,i),txt)
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
    Location','southwest','Orientation','vertical')
```

A.6. Sexto programa

He utilizado el siguiente programa para generar las figuras 2.16, 2.17 2.18.

```
% Datos iniciales
x0 = pi/6;
y00 = 2/sqrt(3);
H = 0.2;
```

```
N = 5; % 5
n=[2,4,8,16,32]; %Romberg
m=[2,4,6,8,12]; %Bulirsch
w=[2,4,6,8,10]; %Arm nica

% Inicializaci n de variables
Tn = zeros(N, N);
trabajon = zeros(N, N);
errorn = zeros(N, N);

Tm = zeros(N, N);
trabajom = zeros(N, N);
errorm = zeros(N, N);

Tw = zeros(N, N);
trabajow = zeros(N, N);
errorw = zeros(N, N);

%Gragg con paso suavizador para T(j,1)
for i=1:N
    h=H/n(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:n(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    aux=y0;
    y0=y1;
    y1=y2;
    end
    S=(aux+2*y0+y1)/4;
    Tn(i,1)=S;

end

for i=1:N
    h=H/m(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:m(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    aux=y0;
```

```
52      y0=y1;
53      y1=y2;
54      end
55      S=(aux+2*y0+y1)/4;
56      Tm(i,1)=S;
57
58  end
59
60  for i=1:N
61      h=H/w(i);
62      x=x0;
63      y0=y00;
64      y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
65      h2=2*h;
66      for j=1:w(i)
67      x=x+h;
68      y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
69      aux=y0;
70      y0=y1;
71      y1=y2;
72      end
73      S=(aux+2*y0+y1)/4;
74      Tw(i,1)=S;
75
76  end
77
78  % Inicializa la primera columna con los valores num ricos
79  errorn(:,1) = abs(Tn(:, 1) - 1./cos(x0+H));
80  trabajon(:,1) = n(:)+1;
81
82  errorm(:,1) = abs(Tm(:, 1) - 1./cos(x0+H));
83  trabajom(:,1) = m(:)+1;
84
85  errorw(:,1) = abs(Tw(:, 1) - 1./cos(x0+H));
86  trabajow(:,1) = w(:)+1;
87
88  % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
89  % calculo el error y el trabajo
90  for k = 1:N-1
91      for j = k+1:N
92          Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / ((n(j
                ) / n(j-k))^2 - 1);
93          errorn(j,k+1) = abs(Tn(j, k+1) - 1./cos(x0+H));
94          trabajon(j,k+1)=sum(n(j-k+1:j))+1;
95
96      end
```

```
end

for k = 1:N-1
    for j = k+1:N
        Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / ((m(j
            ) / m(j-k))^2 - 1);
        errorm(j,k+1) = abs(Tm(j, k+1) - 1./cos(x0+H));
        trabajom(j,k+1)=sum(m(j-k+1:j))+1;

    end
end

for k = 1:N-1
    for j = k+1:N
        Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
            ) / w(j-k))^2 - 1);
        errorw(j,k+1) = abs(Tw(j, k+1) - 1./cos(x0+H));
        trabajow(j,k+1)=sum(w(j-k+1:j))+1;

    end
end

% Gr fica en escala doble logar tmica

figure(1)
clf
loglog(trabajon, errorn, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Romberg');
for i=1:N
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajon(i,i),errorn(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajon(N,i),errorn(N,i),txt)
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
   Location','southwest','Orientation','vertical')

figure(2)
clf
loglog(trabajom, errorm, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Bulirsch');
for i=1:N
```

```
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajom(i,i),errorm(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajom(N,i),errorm(N,i),txt)
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
   Location','southwest','Orientation','vertical')

figure(3)
clf
loglog(trabajow, errorw, '-o');
xlabel('Trabajo');
ylabel('Precisi n');
title('Arm nica');
for i=1:N
    txt=['T_{',num2str(i) ',' num2str(i),'}'];
    text(trabajow(i,i),errorw(i,i),txt)
    txt=['T_{',num2str(N) ',' num2str(i),'}'];
    text(trabajow(N,i),errorw(N,i),txt)
end
axis([1 100 10.^-16 1]);
legend({'T_{j,1}','T_{j,2}','T_{j,3}','T_{j,4}','T_{j,5}'},'
   Location','southwest','Orientation','vertical')
```

A.7. Séptimo programa

He utilizado el siguiente programa para generar las figuras 2.19, 2.20 2.21.

```
% Datos iniciales
H = zeros(1, 3);
H(1) = 0.05;
H(2) = 0.025;
H(3) = 0.0125;
n=[2,4,8,16,32]; %Romberg
m=[2,4,6,8,16]; %Bulirsch
w=[2,4,6,8,10]; %Arm nica
errorn = zeros(3, 5);
errorm = zeros(3, 5);
errorw = zeros(3, 5);


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:3
df=1/H(l); % calculo el n mero de pasos
```

```
for N=1:5 % calculo el error global de T_{N,N}
Tn = zeros(N, N);
x0 = pi/6;
y00 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%GBS para T(j,1)
for i=1:N
    h=H(l)/n(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
    for j=1:n(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    aux=y0;
    y0=y1;
    y1=y2;
    end
    S=(aux+2*y0+y1)/4;
    Tn(i,1)=S;

end
% Construyo la tabla de extrapolaci n con el algortimo de
   Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tn(j, k+1) = Tn(j, k) + (Tn(j, k) - Tn(j-1, k)) / ((n(j
           ) / n(j-k))^2 - 1);
    end
end
y00=Tn(N,N);
x0=x0+H(l);

end
errorn(l,N) = abs(Tn(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(1)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
loglog(H, errorn, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
ylabel('Error en t=x_{0}+1');
```

```
62  title('Romberg');
63  axis([10.^-2 1 10.^-13 20]);
64  hold on
65  loglog(H,100*H.^2,'--')
66  loglog(H,1000*H.^4,'--')
67  loglog(H,5000*H.^6,'--')
68  loglog(H,50000*H.^8,'--')
69  loglog(H,60000*H.^10,'--')
70  legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
        6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
        'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
       },'Location','southeast','Orientation','vertical')
71
72
73
74  %%%%%%%%%%%%%%%%%%%%%%%%
75  for l=1:3
76  df=1/H(l); % calculo el n mero de pasos
77  for N=1:5 % calculo el error global de T_{N,N}
78  Tm = zeros(N, N);
79  x0 = pi/6;
80  y00 = 2/sqrt(3);
81  for d=1:df % avanzo hasta x=x_0+1
82  %GBS para T(j,1)
83  for i=1:N
84      h=H(l)/m(i);
85      x=x0;
86      y0=y00;
87      y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
88      h2=2*h;
89      for j=1:m(i)
90      x=x+h;
91      y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
92      aux=y0;
93      y0=y1;
94      y1=y2;
95      end
96      S=(aux+2*y0+y1)/4;
97      Tm(i,1)=S;
98
99  end
100 % Construyo la tabla de extrapolaci n con el algortimo de
        Aitken-Neville,
101 % calculo la soluci n
102 for k = 1:N-1
103     for j = k+1:N
104         Tm(j, k+1) = Tm(j, k) + (Tm(j, k) - Tm(j-1, k)) / ((m(j
```

```
            ) / m(j-k))^2 - 1);
    end
end
y00=Tm(N,N);
x0=x0+H(l);

end
errorm(l,N) = abs(Tm(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(2)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
loglog(H, errorm, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
ylabel('Error en t=x_{0}+1');
title('Bulirsch');
axis([10.^-2 1 10.^-13 20]);
hold on
loglog(H,100*H.^2,'--')
loglog(H,1000*H.^4,'--')
loglog(H,10000*H.^6,'--')
loglog(H,200000*H.^8,'--')
loglog(H,1000000*H.^10,'--')
legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
    6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
    'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
   },'Location','southeast','Orientation','vertical')


%%%%%%%%%%%%%%%%%%%%%%%%
for l=1:3
df=1/H(l); % calculo el n mero de pasos
for N=1:5 % calculo el error global de T_{N,N}
Tw = zeros(N, N);
x0 = pi/6;
y00 = 2/sqrt(3);
for d=1:df % avanzo hasta x=x_0+1
%GBS para T(j,1)
for i=1:N
    h=H(l)/w(i);
    x=x0;
    y0=y00;
    y1=y0+h*((-y0*sin(x)+2*tan(x))*y0);
    h2=2*h;
```

```
    for j=1:w(i)
    x=x+h;
    y2=y0+h2*((-y1*sin(x)+2*tan(x))*y1);
    aux=y0;
    y0=y1;
    y1=y2;
    end
    S=(aux+2*y0+y1)/4;
    Tw(i,1)=S;

end
% Construyo la tabla de extrapolaci n con el algortimo de
   Aitken-Neville,
% calculo la soluci n
for k = 1:N-1
    for j = k+1:N
        Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
           ) / w(j-k))^2 - 1);
    end
end
y00=Tw(N,N);
x0=x0+H(l);

end
errorw(l,N) = abs(Tw(N,N) - 1./cos((pi/6)+1));
end
end

% Gr fica en escala doble logar tmica
figure(3)
clf
colororder(["#0072BD";"#D95319";"#EDB120";"#7E2F8E";"#77AC30"])
loglog(H, errorw, '-o');
xlabel('Longitudes de paso (H=0.05, 0.025 y 0.0125)');
ylabel('Error en t=x_{0}+1');
title('Arm nica');
axis([10.^-2 1 10.^-13 20]);
hold on
loglog(H,100*H.^2,'--')
loglog(H,1000*H.^4,'--')
loglog(H,10000*H.^6,'--')
loglog(H,200000*H.^8,'--')
loglog(H,3000000*H.^10,'--')
legend({'T_{1,1}, orden 2', 'T_{2,2}, orden 4', 'T_{3,3}, orden
    6', 'T_{4,4}, orden 8', 'T_{5,5}, orden 10', 'Pendiente 2',
    'Pendiente 4', 'Pendiente 6', 'Pendiente 8', 'Pendiente 10'
   },'Location','southeast','Orientation','vertical')
```

A.8. Octavo programa

He utilizado el siguiente programa para generar las figuras 3.1, 3.2 3.3.

```
% Datos iniciales
x0 = 0; %inicial
xf=20; %final
y00 = 3/2;
z00 = 3;
w=[2,4,6,8,10,12,14,16,18]; %Arm nica
TOL=10^-5;
Hv = TOL^0.14; %tama o de paso inicial
% Inicializaci n de variables
yy(1) = y00;
zz(1) = z00;
xx(1) = x0;
l=3; %el orden inicial
s=1; %los pasos aceptados +1
A(1)=w(1)+1;
W(1)=A(1)/Hv;

while xx(end) < xf-1e-13 %% si ocurre paramos

if l<9 %% sino bajamos el orden
if l>2 %% sino significa que hemos bajado mucho el orden
for i=1:l-1 %computo las l-1 primeras lineas de la tabla y los
    datos necesarios
     h=min([Hv, (xf-xx(s))])/w(i);
     x=xx(end);
     y0=yy(end);
     z0=zz(end);
     y1=y0+h*(1+y0^2*z0-4*y0);
     z1=z0+h*(3*y0-y0^2*z0);
     h2=2*h;
     for j=1:w(i)
     x=x+h;
     y2=y0+h2*(1+y1^2*z1-4*y1);
     z2=z0+h2*(3*y1-y1^2*z1);
     auxy=y0;
     auxz=z0;
     y0=y1;
     y1=y2;
     z0=z1;
     z1=z2;
     end
     Sy=(auxy+2*y0+y1)/4;
     Sz=(auxz+2*z0+z1)/4;
```

```
43      Twy(i,1)=Sy;
44      Twz(i,1)=Sz;
45
46
47  end
48  for k = 1:l-2
49      for j = k+1:l-1
50          Twy(j, k+1) = Twy(j, k) + (Twy(j, k) - Twy(j-1, k)) /
                ((w(j) / w(j-k))^2 - 1);
51          Twz(j, k+1) = Twz(j, k) + (Twz(j, k) - Twz(j-1, k)) /
                ((w(j) / w(j-k))^2 - 1);
52
53      end
54  end
55
56  errorw(l-1)=norm(Twy(l-1,l-2)-Twy(l-1,l-1))+norm(Twz(l-1,l-2)-
        Twz(l-1,l-1));
57  H(l-1)=0.94*Hv*(TOL/errorw(l-1)).^(1/(2*l-3));
58  for t=2:l-1
59  A(t)=A(t-1)+w(t);
60  end
61  W(l-1)=A(l-1)/H(l-1);
62  A(l)=A(l-1)+w(l);
63
64  if l>3
65  errorw(l-2)=norm(Twy(l-2,l-3)-Twy(l-2,l-2))+norm(Twz(l-2,l-3)-
        Twz(l-2,l-2));
66  H(l-2)=0.94*Hv*(TOL/errorw(l-2)).^(1/(2*l-5));
67  W(l-2)=A(l-2)/H(l-2);
68  end
69
70
71  if errorw(l-1) <=TOL, %aceptamos y hacemos cambios
72  Hdef(s)=min([Hv, (xf-xx(s))]);
73  L(s)=l-1;
74  s=s+1;
75  yy(s)=Twy(l-1,l-1);
76  zz(s)=Twz(l-1,l-1);
77  xx(s)=xx(s-1)+Hdef(s-1);
78  if W(l-1) < 0.94*W(l-2),
79  Hv=H(l-1)*A(l)/A(l-1);
80  else
81  l=l-1;
82  Hv=H(l);
83  end
84
85  else %rechazamos, seguimos buscando
```

```
86  if errorw(l-1) > (w(l)/Hv)^2*TOL, %bajamos el orden y hacemos
        el paso m s peque o
87  l=l-1;
88  Hv=H(l);
89  else %continuamos, calculando la fila l y los siguientes datos
90  h=min([Hv, (xf-xx(s))])/w(l);
91  x=xx(end);
92  y0=yy(end);
93  z0=zz(end);
94  y1=y0+h*(1+y0^2*z0-4*y0);
95  z1=z0+h*(3*y0-y0^2*z0);
96  h2=2*h;
97  for j=1:w(l)
98  x=x+h;
99  y2=y0+h2*(1+y1^2*z1-4*y1);
100 z2=z0+h2*(3*y1-y1^2*z1);
101 auxy=y0;
102 auxz=z0;
103 y0=y1;
104 y1=y2;
105 z0=z1;
106 z1=z2;
107 end
108 Sy=(auxy+2*y0+y1)/4;
109 Sz=(auxz+2*z0+z1)/4;
110 Twy(l,1)=Sy;
111 Twz(l,1)=Sz;
112
113 for k = 1:l-1
114 Twy(l, k+1) = Twy(l, k) + (Twy(l, k) - Twy(l-1, k)) / ((w(l) /
        w(l-k))^2 - 1);
115 Twz(l, k+1) = Twz(l, k) + (Twz(l, k) - Twz(l-1, k)) / ((w(l) /
        w(l-k))^2 - 1);
116 end
117
118 errorw(l)=norm(Twy(l,l-1)-Twy(l,l))+norm(Twz(l,l-1)-Twz(l,l));
119 H(l)=0.94*Hv*(TOL/errorw(l)).^(1/(2*l-1));
120 A(l)=A(l-1)+w(l);
121 A(l+1)=A(l)+w(l+1);
122 W(l)=A(l)/H(l);
123 if errorw(l) <=TOL, %aceptamos y hacemos cambios
124 Hdef(s)=min([Hv, (xf-xx(s))]);
125 L(s)=l;
126 s=s+1;
127 yy(s)=Twy(l,l);
128 zz(s)=Twz(l,l);
129 xx(s)=xx(s-1)+Hdef(s-1);
```

```
130 if W(l-1) < 0.94*W(l),
131 l=l-1;
132 Hv=H(l);
133 elseif W(l) < 0.94*W(l-1),
134 l=l+1;
135 Hv=H(l-1)*A(l)/A(l-1);
136 else
137 Hv=H(l);
138 end
139 else %veamos si rechazamos o seguimos
140 if errorw(l) > (w(l+1)/Hv)^2*TOL, %rechazamos y bajo el orden y
        el paso
141 l=l-1;
142 Hv=H(l);
143 else % continuamos, calculando T_{l+1,l+1} y el resto de datos
144 h=min([Hv, (xf-xx(s))])/w(l+1);
145 x=xx(end);
146 y0=yy(end);
147 z0=zz(end);
148 y1=y0+h*(1+y0^2*z0-4*y0);
149 z1=z0+h*(3*y0-y0^2*z0);
150 h2=2*h;
151 for j=1:w(l+1)
152 x=x+h;
153 y2=y0+h2*(1+y1^2*z1-4*y1);
154 z2=z0+h2*(3*y1-y1^2*z1);
155 auxy=y0;
156 auxz=z0;
157 y0=y1;
158 y1=y2;
159 z0=z1;
160 z1=z2;
161 end
162 Sy=(auxy+2*y0+y1)/4;
163 Sz=(auxz+2*z0+z1)/4;
164 Twy(l+1,1)=Sy;
165 Twz(l+1,1)=Sz;
166
167 for k = 1:l
168 Twy(l+1, k+1) = Twy(l+1, k) + (Twy(l+1, k) - Twy(l, k)) / ((w(l
        +1) / w(l+1-k))^2 - 1);
169 Twz(l+1, k+1) = Twz(l+1, k) + (Twz(l+1, k) - Twz(l, k)) / ((w(l
        +1) / w(l+1-k))^2 - 1);
170 end
171
172 errorw(l+1)=norm(Twy(l+1,l)-Twy(l+1,l+1))+norm(Twz(l+1,l)-Twz(l
        +1,l+1));
```

```
173
174 if errorw(l+1) <=TOL, %aceptamos y hacemos cambios
175 H(l+1)=0.94*Hv*(TOL/errorw(l+1)).^(1/(2*l+1));
176 A(l+1)=A(l)+w(l+1);
177 W(l+1)=A(l+1)/H(l+1);
178
179 Hdef(s)=min([Hv, (xf-xx(s))]);
180 L(s)=l+1;
181 s=s+1;
182 yy(s)=Twy(l+1,l+1);
183 zz(s)=Twz(l+1,l+1);
184 xx(s)=xx(s-1)+Hdef(s-1);
185 if W(l) < 0.94*W(l+1),
186 l=l-1;
187 Hv=H(l);
188 elseif W(l+1) < 0.94*W(l),
189 l=l+1;
190 Hv=H(l);
191 else
192 Hv=H(l);
193 end
194 else %rechazamos y cambiamos el tama o del paso y volvemos a
      empezar el bucle
195 Hv=H(l);
196 end
197
198 end
199
200 end
201
202 end
203 end
204
205 else
206     l=3;
207 end
208
209 else
210     l=l-1;
211 end
212
213 end
214
215
216 figure(1)
217 clf
218 plot(xx,yy,'o-')
```

```
xlabel('x')
ylabel('y')
title('Soluci n')
hold on
plot(xx,zz,'o-')


figure(2)
clf
semilogy(xx(1:end-1),Hdef,'o-')
xlabel('x')
ylabel('Tama o del paso (H)')
title('Tama os de paso aceptados')

figure(3)
clf
plot(xx(1:end-1),2*L,'o-') %El orden siempre es par y el doble
    del L
xlabel('x')
ylabel('Orden')
title('Ordenes aceptados')
```

A.9. Noveno programa

He utilizado el siguiente programa (cambiando la tolerancia y guardando los datos para hacer las gráficas) para generar las figuras 3.4, 3.5 3.6.

```
% Datos iniciales
x0 = -2; %inicial
xf=2; %final
y00 = 2/3;
w=[2,4,6,8,10,12,14,16,18]; %Arm nica
TOL=1e-5;
Hv = TOL^0.14; %tama o de paso inicial
% Inicializaci n de variables
yy(1) = y00;
xx(1) = x0;
l=3; %el orden inicial
s=1; %los pasos aceptados +1
A(1)=w(1)+1;
W(1)=A(1)/Hv;


while xx(end) < xf-1e-13 %% si ocurre paramos

```

```
if l<9 %% sino bajamos el orden
if l>2 %% sino significa que hemos bajado mucho el orden
for i=1:l-1 %computo las l-1 primeras lineas de la tabla y los
    datos necesarios
    h=min([Hv, (xf-1e-13-xx(s))])/w(i);
    x=xx(end);
    y0=yy(end);
    y1=y0+h*((-sign(x)*abs(1-abs(x)))*y0^2);
    h2=2*h;
    for j=1:w(i)
    x=x+h;
    y2=y0+h2*((-sign(x)*abs(1-abs(x)))*y1^2);
    aux=y0;
    y0=y1;
    y1=y2;
    end
    S=(aux+2*y0+y1)/4;
    Tw(i,1)=S;


end
for k = 1:l-2
    for j = k+1:l-1
        Tw(j, k+1) = Tw(j, k) + (Tw(j, k) - Tw(j-1, k)) / ((w(j
            ) / w(j-k))^2 - 1);

    end
end

errorw(l-1)=norm(Tw(l-1,l-2)-Tw(l-1,l-1));
H(l-1)=0.94*Hv*(TOL/errorw(l-1)).^(1/(2*l-3));
for t=2:l-1
A(t)=A(t-1)+w(t);
end
W(l-1)=A(l-1)/H(l-1);
A(l)=A(l-1)+w(l);

if l>3
errorw(l-2)=norm(Tw(l-2,l-3)-Tw(l-2,l-2));
H(l-2)=0.94*Hv*(TOL/errorw(l-2)).^(1/(2*l-5));
W(l-2)=A(l-2)/H(l-2);
end


if errorw(l-1) <=TOL, %aceptamos y hacemos cambios
Hdef(s)=min([Hv, (xf-1e-13-xx(s))]);
L(s)=l-1;
```

```
64  s=s+1;
65  yy(s)=Tw(l-1,l-1);
66  xx(s)=xx(s-1)+Hdef(s-1);
67
68  if xx(s) > -2 && xx(s) <= -1,
69  err(s)= abs(yy(s) - 2./((1+xx(s))^2+2));
70  elseif  xx(s) > -1 && xx(s) <= 0,
71  err(s)= abs(yy(s) + 2./((1+xx(s))^2-2));
72  elseif  xx(s) > 0 && xx(s) <= 1,
73  err(s)= abs(yy(s) + 2./((-1+xx(s))^2-2));
74  elseif  xx(s) > 1 && xx(s) < 2,
75  err(s)= abs(yy(s) - 2./((1-xx(s))^2+2));
76  end
77
78  if W(l-1) < 0.94*W(l-2),
79  Hv=H(l-1)*A(l)/A(l-1);
80  else
81  l=l-1;
82  Hv=H(l);
83  end
84
85  else %rechazamos, seguimos buscando
86  if errorw(l-1) > (w(l)/Hv)^2*TOL, %bajamos el orden y hacemos
        el paso m s peque o
87  l=l-1;
88  Hv=H(l);
89  else %continuamos, calculando la fila l y los siguientes datos
90  h=min([Hv, (xf-1e-13-xx(s))])/w(l);
91  x=xx(end);
92  y0=yy(end);
93  y1=y0+h*((-sign(x)*abs(1-abs(x)))*y0^2);
94  h2=2*h;
95  for j=1:w(l)
96  x=x+h;
97  y2=y0+h2*((-sign(x)*abs(1-abs(x)))*y1^2);
98  aux=y0;
99  y0=y1;
100 y1=y2;
101 end
102 S=(aux+2*y0+y1)/4;
103 Tw(l,1)=S;
104
105 for k = 1:l-1
106 Tw(l, k+1) = Tw(l, k) + (Tw(l, k) - Tw(l-1, k)) / ((w(l) / w(l-
        k))^2 - 1);
107 end
108
```

```
errorw(l)=norm(Tw(l,l-1)-Tw(l,l));
H(l)=0.94*Hv*(TOL/errorw(l)).^(1/(2*l-1));
A(l)=A(l-1)+w(l);
A(l+1)=A(l)+w(l+1);
W(l)=A(l)/H(l);
if errorw(l) <=TOL, %aceptamos y hacemos cambios
Hdef(s)=min([Hv, (xf-1e-13-xx(s))]);
L(s)=l;
s=s+1;
yy(s)=Tw(l,l);
xx(s)=xx(s-1)+Hdef(s-1);

if xx(s) > -2 && xx(s) <= -1,
err(s)= abs(yy(s) - 2./((1+xx(s))^2+2));
elseif  xx(s) > -1 && xx(s) <= 0,
err(s)= abs(yy(s) + 2./((1+xx(s))^2-2));
elseif  xx(s) > 0 && xx(s) <= 1,
err(s)= abs(yy(s) + 2./((-1+xx(s))^2-2));
elseif  xx(s) > 1 && xx(s) < 2,
err(s)= abs(yy(s) - 2./((1-xx(s))^2+2));
end

if W(l-1) < 0.94*W(l),
l=l-1;
Hv=H(l);
elseif W(l) < 0.94*W(l-1),
l=l+1;
Hv=H(l-1)*A(l)/A(l-1);
else
Hv=H(l);
end
else %veamos si rechazamos o seguimos
if errorw(l) > (w(l+1)/Hv)^2*TOL, %rechazamos y bajo el orden y
    el paso
l=l-1;
Hv=H(l);
else % continuamos, calculando T_{l+1,l+1} y el resto de datos
h=min([Hv, (xf-1e-13-xx(s))])/w(l+1);
x=xx(end);
y0=yy(end);
y1=y0+h*((-sign(x)*abs(1-abs(x)))*y0^2);
h2=2*h;
for j=1:w(l+1)
x=x+h;
y2=y0+h2*((-sign(x)*abs(1-abs(x)))*y1^2);
aux=y0;
y0=y1;
```

```
y1=y2;
end
S=(aux+2*y0+y1)/4;
Tw(l+1,1)=S;

for k = 1:l
Tw(l+1, k+1) = Tw(l+1, k) + (Tw(l+1, k) - Tw(l, k)) / ((w(l+1)
    / w(l+1-k))^2 - 1);
end

errorw(l+1)=norm(Tw(l+1,l)-Tw(l+1,l+1));


if errorw(l+1) <=TOL, %aceptamos y hacemos cambios
Hdef(s)=min([Hv, (xf-1e-13-xx(s))]);
L(s)=l+1;
s=s+1;
yy(s)=Tw(l+1,l+1);
xx(s)=xx(s-1)+Hdef(s-1);

if xx(s) > -2 && xx(s) <= -1,
err(s)= abs(yy(s) - 2./((1+xx(s))^2+2));
elseif  xx(s) > -1 && xx(s) <= 0,
err(s)= abs(yy(s) + 2./((1+xx(s))^2-2));
elseif  xx(s) > 0 && xx(s) <= 1,
err(s)= abs(yy(s) + 2./((-1+xx(s))^2-2));
elseif  xx(s) > 1 && xx(s) < 2,
err(s)= abs(yy(s) - 2./((1-xx(s))^2+2));
end

H(l+1)=0.94*Hv*(TOL/errorw(l+1)).^(1/(2*l+1));
A(l+1)=A(l)+w(l+1);
W(l+1)=A(l+1)/H(l+1);


if W(l) < 0.94*W(l+1),
l=l-1;
Hv=H(l);
elseif W(l+1) < 0.94*W(l),
l=l+1;
Hv=H(l);
else
Hv=H(l);
end
else %rechazamos y cambiamos el tama o del paso y volvemos a
    empezar el bucle
Hv=H(l);
```

```
end

end

end

end
end

else
    l=3;
end

else
    l=l-1;
end

end


figure(1)
clf
plot(xx,yy,'o-')
xlabel('x')
ylabel('y')
title('Soluci n')

figure(2)
clf
semilogy(xx(1:end-1),Hdef,'o-')
xlabel('x')
ylabel('Tama o del paso (H)')
title('Tama os de paso aceptados')

figure(3)
clf
plot(xx(1:end-1),2*L,'o-') %El orden siempre es par y el doble
    del L
xlabel('x')
ylabel('Orden')
title('Ordenes aceptados')

figure(4)
clf
semilogy(xx,err,'o-')
xlabel('x')
ylabel('Error')
```

```
246 title('Errores')
```

Bibliografía

[1] R. Bulirsch and J. Stoer (1966): Numerical treatment of ordinary differential equations by extrapolation methods. Num. Math., Vol. 8, p. 1-13.

[2] P. Deuflhard (1983): Order and stepsize control in extrapolation methods. Num. Math., Vol. 41, p. 399-422.

[3] W. B. Gragg (1964): Repeated extrapolation to the limit in the numerical solution of ordinary differential equations. Thesis, Univ. of California.

[4] E. Hairer, S. P. Nørsett and G. Wanner (1993): Solving Ordinary Differential Equations I. Nonstiff Problems: 8 (Springer Series in Computational Mathematics). Second revised edition. Springer-Verlag Berlin Heidelberg.

[5] E. Hairer and G. Wanner (2004): Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems: 14 (Springer Series in Computational Mathematics). Second revised edition. Springer-Verlag Berlin Heidelberg.

[6] R. Lefever and G. Nicolis (1970): Chemical Instabilities and Sustained Oscillations. J. theor. Biol., Vol. 30, p. 267-284.

[7] W. Romberg (1955): Vereinfachte numerische Integration. Norske Vid. Selsk. Forhdl, Vol. 28, p. 30-36.

[8] H.J. Stetter (1970): Symmetric two-step algorithms for ordinary differential equations. Computing, Vol. 5, p. 267-280.