FACULTAD DE INGENIERIA NAVAL Y OCEANICA

DEPARTAMENTO DE MATEMATICAS

TRABAJO FINAL DE GRADO:

Métodos numéricos aplicados a la ingeniería naval. Énfasis en los módulos de integración numérica y ecuaciones diferenciales

CARLOS TAPIA MARFIL

Profesor:
Juan Carlos Trillo Moya

AGRADECIMIENTOS

A todos aquellos que han estado a mi lado durante el transcurso de este proyecto.

Al Doctor D. Juan Carlos Trillo Moya, mi tutor en este trabajo, que me ha apoyado en todo momento y ha compartido conmigo todos los conocimientos.

Al departamento de Matemáticas Aplicada y Estadística, al departamento de Ingeniería Naval y a la Universidad Politécnica de Cartagena.

Y por último a mis padres y a mis hermanos que han estado allí siempre a pesar de la distancia.

Sinopsis

Durante los últimos años se han dedicado muchos esfuerzos al estudio y perfeccionamiento de métodos numéricos debido a la gran importancia que han adquirido en el diseño industrial además de en otros campos de investigación tanto científicos como tecnológicos.

Su gran uso se debe a que mediante algoritmos altamente sofisticados, nos permiten realizar aproximaciones y obtener resultados para casi cualquier tipo de problemas que, de otra manera nos sería muy difícil, o incluso imposible de obtener. Además, permiten un estudio elaborado del error cometido por cada método y en cada caso, para así poder tenerlo en cuenta a la hora de la toma de decisiones críticas en el proceso de diseño.

Mi compañero Jesús Monserrat Torrecillas realizó un gran trabajo enumerando cada uno de los métodos utilizados. Nuestra labor ha consistido en la ampliación de dicho trabajo en los campos de las Ecuaciones Diferenciales y la Integración Numéricos.

El trabajo está estructurado de tal manera que se explican en profundidad cada uno de los métodos. Para cada método se ha añadido su aplicación en el ámbito de la Ingeniería Naval, ya que éste es un trabajo del Grado de Arquitectura Naval y Ingeniería de Sistemas Marinos. Por otra parte hemos echo énfasis tanto casos de arquitectura naval, como en ejemplos de su aplicación en el campo de la propulsión.

Además a cada uno de estos métodos les corresponde un programa realizado con Matlab y, junto con esto, hemos desarrollado una interfaz gráfica que implementa todos estos métodos. Esto facilitará su utilización por el usuario.

En el primer apartado se realiza una introducción a los métodos numéricos, el segundo apartado corresponde a las *Reglas de Newton Cotes Cerradas*, donde se muestran tanto su forma simple como compuesta, el punto 3 se habla de las *Reglas de Newton Cotes Abiertas*, en el apartado 4 se estudia los Métodos de integración Numérica aplicadas a las integrales múltiples, el 5 esta referido al estudio de las Ecuaciones Diferenciales, en el apartado 6 se atiende a la interfaz gráfica y finalmente en el 7 hablamos de los programas auxiliares que he usado en el transcurso de mi trabajo.

Índice

Índice de figuras

1. MÉTODOS NUMÉRICOS EN INTEGRACIÓN

1.1. DEFINICIÓN DE LOS MÉTODOS

La **integración numérica** es una herramienta esencial utilizada en la ingeniería para el cálculo de integrales sin solución analítica o cuya solución sería excesivamente laboriosa y complicada mediante la aproximación de esa integral por la integral de una función más sencilla que aproxima la función original, esa función más sencilla la podemos construir por ejemplo mediante el uso del **Polinomio de Lagrange** de grado dos:

$$p_2(x) = \frac{(x-c)(x-b)}{(a-c)(a-b)}f(a) + \frac{(x-a)(x-b)}{(c-a)(c-b)}f(b) + \frac{(x-a)(x-c)}{(b-a)(b-c)}f(c). \tag{0}$$

En la ecuación (1.1) vemos la expresión del **Polinomio de Lagrange** que pasa por los puntos $(a, f(b))$,$(b, f(b))$,$(c, f(c))$. En la Figura 1 aparece representada la aproximación de una curva mediante el **Polinomio de Lagrange** a trozos:

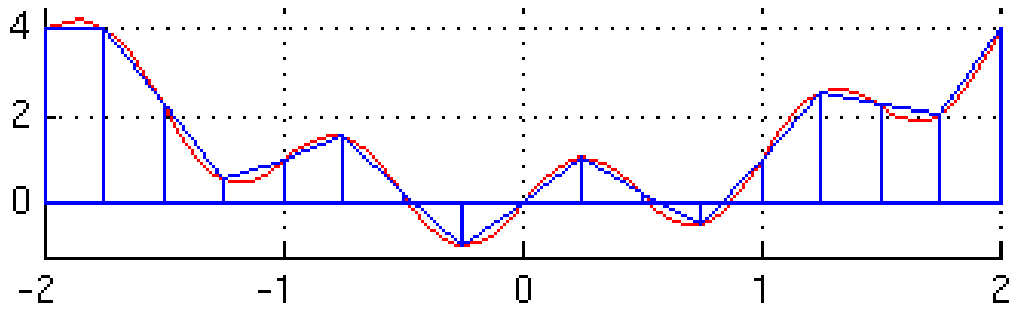

Figura 1: Ejemplo del ajuste de una curva mediante interpolación de Lagrange segmentaria

Un buen ejemplo del uso de la integración numérica aparece en el ámbito de la termodinámica para el cálculo de la capacidad calórica de un sólido mediante el modelo de Debye [1]

$$\phi(x) = \int_0^x \frac{t^3}{e^t - 1}\, dt, \tag{0}$$

donde $\phi(x)$ es la función que representa la capacidad calórica del sólido en función de la temperatura x.

Recurrimos a los métodos numéricos para obtener resultados aproximados de esta integral.

Calculamos el área delimitada por la curva y=f(t) para $0 \leq t \leq 5$, con $f(t) = \int_0^x \frac{t^3}{e^t-1} dt$. Esta situación aparece representada en la figura (2):

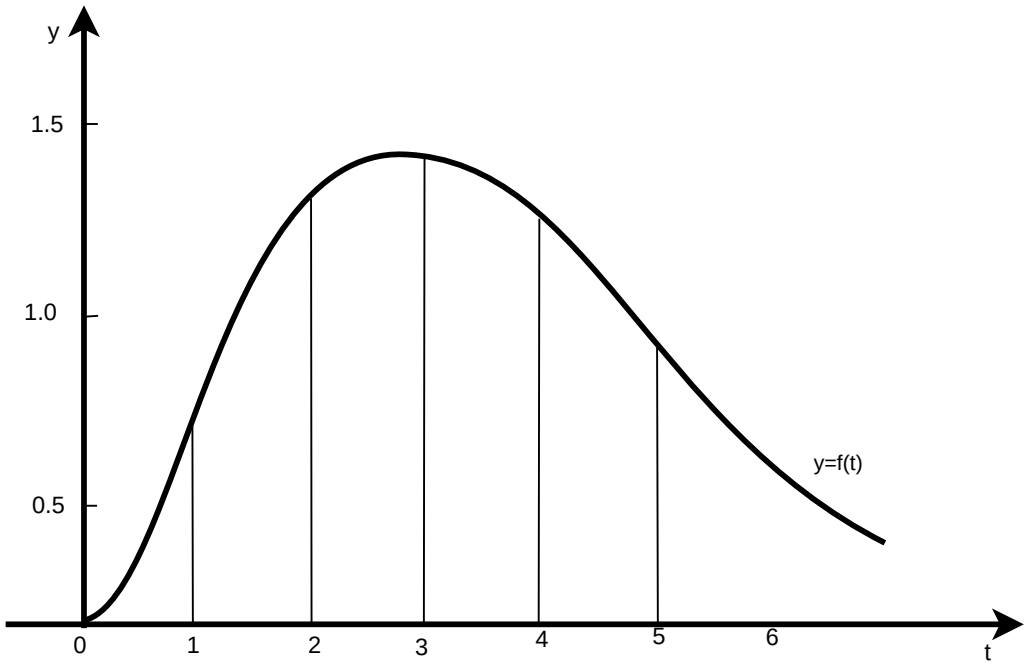

Figura 2: Ejemplo de aplicación de Debye

En general el *Polinomio de Lagrange* de grado **n** que pasa por los puntos $(x_i, f(x_i))\ \ i = 0, ...n$ toma la forma:

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i), \tag{0}$$

siendo $L_i(x)$ los polinomios de Lagrange de grado **n**

$$L_i(x) = \frac{(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)} = \prod_{j=1:n} \frac{x-x_j}{x_i-x_j},$$

$$L_i(x_j) = \begin{cases} 1 \text{ si } i = j, \\ 0 \text{ si } i \neq j. \end{cases}$$

Si denotamos $w_i := \int_a^b L_i(x)\ dx$, entonces:

$$p_n(x) = \sum_{i=0}^{n} w_i f(x_i), \tag{-2}$$

siendo ésta la fórmula de aproximación $f(x) \approx p_n(x)$ que se utiliza para conseguir las distintas *Reglas de Newton Cotes*.

Estas ecuaciones son dependientes del número de puntos para aproximar nuestra función (nosotros utilizaremos como ejemplos los casos de 1 a 6 puntos).

Resulta que el error cometido en dichas aproximaciones puede calcularse, como veremos en el apartado 1.2, y nos permitirá seleccionar el método de integración adecuado que de ellas se deriva y utilizarlo en cálculos y problemas de ingeniería, como en la Figura 3 donde aparece representado un problema para el cálculo de cargas y empujes en un buque obtenido de los apuntes de Cálculo de estructuras [2]:

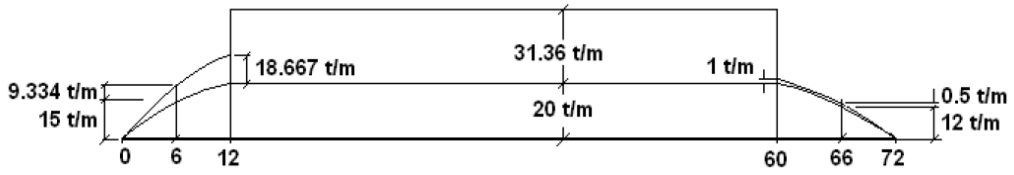

Figura 3: Curva de cargas y pesos de un buque

El primer paso para la obtención de los métodos de integración numérica es la obtención de la fórmula simple del método. Si por ejemplo estuviera basada en 5 puntos con $x_0 = a,\ x_1 = a + h,\ x_2 = a + 2h, x_3 = a + 3h,\ x_4 = b$, donde $h = \frac{b-a}{4}$ el proceso sería el siguiente:

1) Aproximar f(x) en [a,b] por $p_4(x)$, es decir $f(x) \approx p_4(x)$.

2) Integrar $p_4(x)$ en [a,b] en vez de f(x).

Gráficamente podemos ver el proceso en la Figura 4:

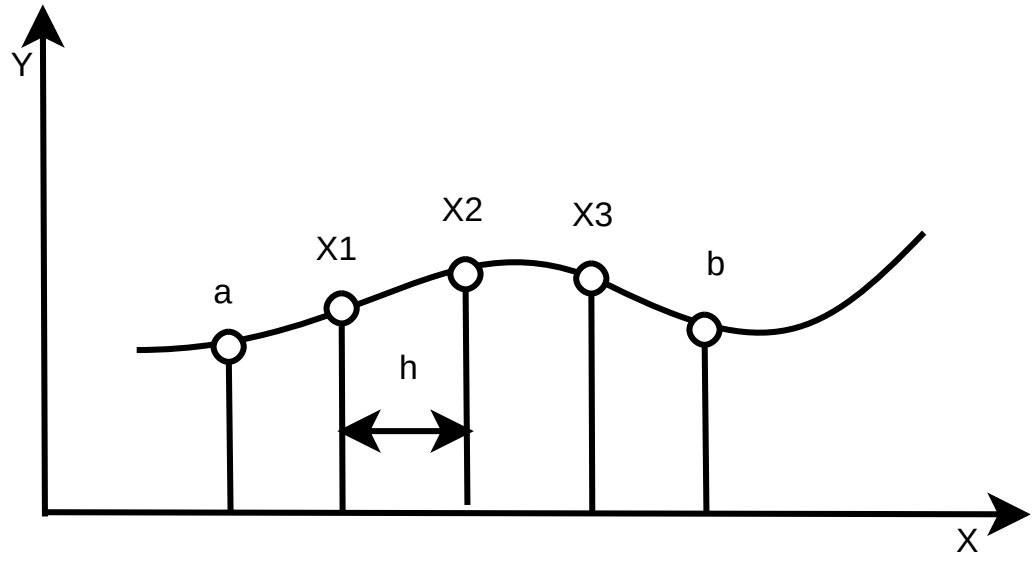

Figura 4: Visualización de los distintos elementos de la integración por Newton Cotes en 5 puntos

En el caso anteriormente explicado del modelo de *Debye* podemos utilizar la *Regla de Boole*, basada en 5 puntos, para la obtención de resultados mediante la sustitución de la función por la regla considerando el punto $a = x_0 = 0$ y $b = x_5$:

$$\phi(t) = \int_0^b \frac{t^3}{e^t - 1}\, dt \approx \frac{2}{45} h(19f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5)).$$

En el caso de $b = 5$ sustituyendo el valor correspondiente de la función en los puntos $x_0, x_1, x_2, x_3, x_4, x_5$ obtendríamos como resultado que la función en $\phi(5) = 4,8998922$.

Para una mejor comprensión aportamos la representación gráfica en la figura 5 de como seria el método de integración basado en polinomios de grado 1, es decir el *Método de los Trapecios*. El lector puede imaginarse la representación para polinomios de grado 4 a partir de este caso.

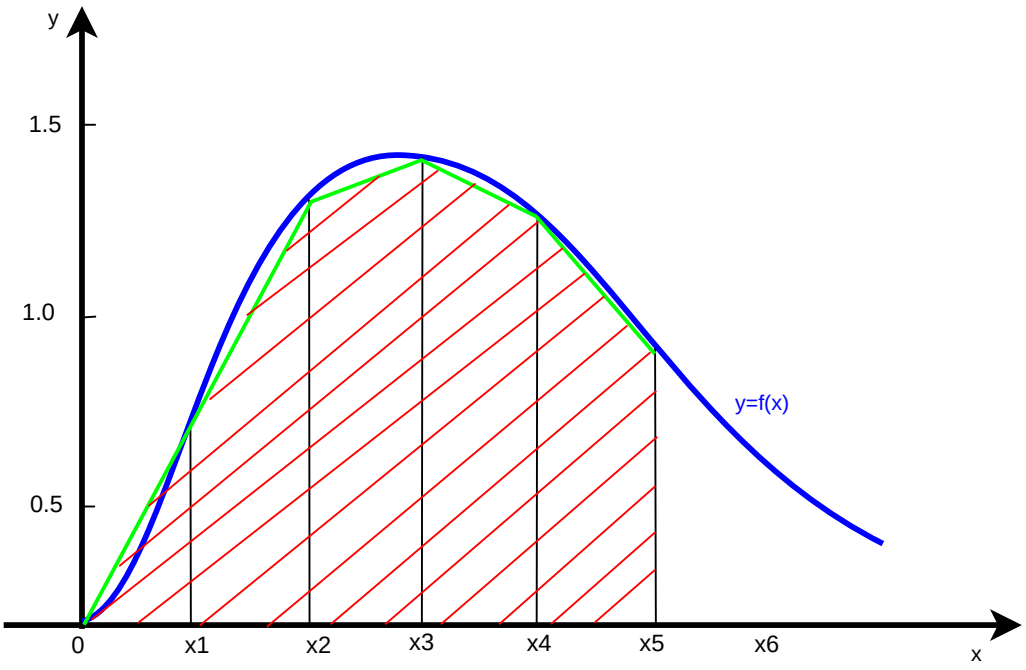

Figura 5: Visualización gráfica de la aplicación del Método de los Trapecios al problema de Debye

Dependiendo de si se toman los extremos del intervalo [a,b] o no, estos métodos de *integración de Newton Cotes* se dividen en:

- **Las Reglas de Newton Cotes Cerradas**, que sí toman los extremos del intervalo.
- **Las Reglas de Newton Cotes Abiertas**, que no toman los extremos del intervalo.

1.2. ANÁLISIS DEL ERROR

1.2.1. Método del análisis del error

A cada una de estas reglas de integración viene asociado un error propio producido por la aproximación **(Aprox)** y el resultado de la integral exacta como se ve en la Figura 1.2.1:

$$E_h = Integral\ Exacta(f(x)) - Aprox(P_n(x)). \tag{-3}$$

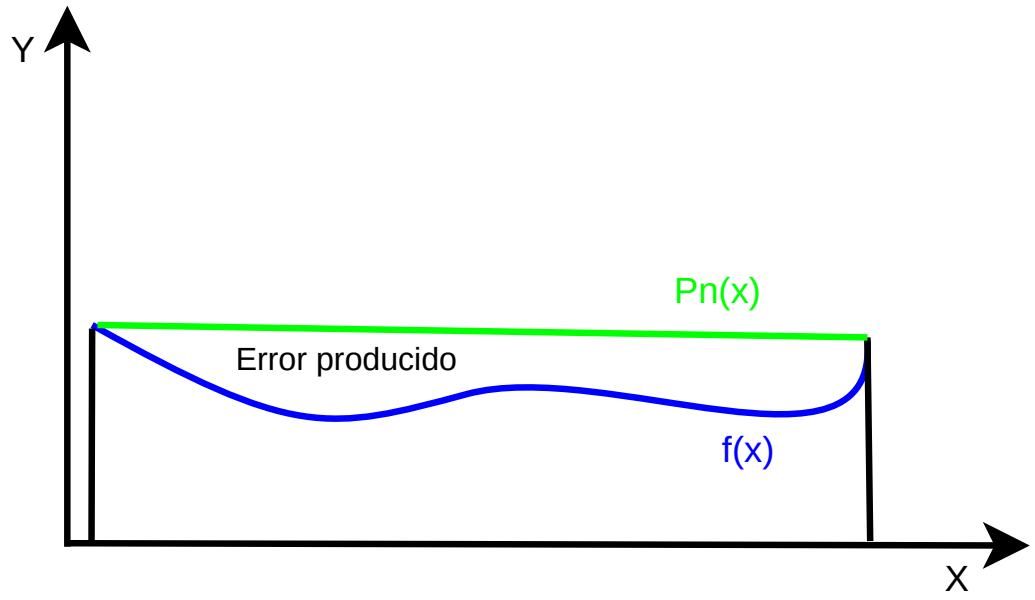

Figura 6: Explicación del error producido en *el Método de Trapecios Simple*

La exactitud de la regla que vayamos a utilizar depende de este factor (Error (E_h)) y de el orden numérico de aproximación (P). cuanto mayor sea el orden del error más exacta será la regla que utilicemos a la hora de calcular la integral y como depende de la amplitud del intervalo (h), cuanto mayor sea el número de puntos utilizados en la regla, mayor sera el orden de aproximación.

Para calcular el orden numérico del error, se recurre a problemas test donde se conoce la integral exacta y por tanto podemos calcular el valor del error exacto. Entonces aplicando las

siguientes fórmulas se puede calcular el orden numérico P.

$$E_h \longrightarrow 0(h^P) = C(h)^P,$$
$$E_{\frac{h}{2}} \longrightarrow 0(\frac{h}{2})^P = C(\frac{h}{2})^P,$$

siendo el cociente entre ambas:

$$\frac{E_h}{E_{\frac{h}{2}}} = 2^P.$$

Despejando P nos dará el orden numérico del error de dicha Regla:

$$P = log_2 \left(\frac{E_h}{E_{\frac{h}{2}}} \right). \tag{-6}$$

La fórmula del error y el orden de aproximación también los podemos obtener de manera exacta utilizando la fórmula del error de aproximación en *Interpolación de Lagrange*:

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\mu)}{(n+1)!}(x - x_0)(x - x_1) \ ... \ (x - x_n), \tag{-6}$$

donde $x_0, x_1 ... x_n$ son los puntos de división y $f^{(n+1)}$ representa la función, indicando el número romano que aparece en la derivada $(n + 1)$ el valor del grado de exactitud de la fórmula, es decir que será una regla exacta para polinomios de grado n porque su derivada de orden n+1 es idénticamente 0.

El orden de aproximación vendrá determinado por el número de términos $x - x_i$ que aparecen en la fórmula (7), veremos ejemplos mas adelante en el apartado 2.

Como se puede observar en la Figura 6 el error que se produce al aplicar trapecios en una función que presenta poca linealidad es muy considerable, teniendo que aplicar otro método para la obtención de un resultado más óptimo, o considerando la regla compuesta de los trapecios que consiste en dividir la regla en subintervalos más pequeños.

1.2.2. Problema con el método de análisis del error

Sin embargo aplicando el método del análisis del error del apartado 1.2.1 al caso de *Simpson 3/8* (Newton cotes cerradas con 4 puntos, ver apartado 2.3), vemos que existe un problema en uno de los pasos. Al aplicarlo nos quedaría la siguiente expresión:

$$\int_a^b \frac{f^{IV}(\mu)}{4!}(x-a)(x-c)(x-d)(x-b)\ dx, \tag{-6}$$

siendo a y b los extremos y c y d los puntos intermedios para el cálculo de la *Regla de Simpson 3/8*.

Si aplicamos el **Teorema del valor medio integral** que dice que dadas dos funciones f, g continua en el intervalo $[a,b]$ con $g(x) \geq 0\ \forall\ x\ \in\ [a,b]$ existe un valor $\xi\ \in\ [a,b]$ tal que:

$$\int_a^b f(x)g(x)\ dx = f(\xi)\int_a^b g(x)\ dx. \tag{-6}$$

Descomponiendo el intervalo original [a,b] en 3 subintervalos y aplicando dicho teorema llegamos a:

$$\begin{aligned}
&\frac{f^{IV}(\mu_1)}{4!}\int_a^c (x-a)(x-c)(x-d)(x-b)\ dx + \\
&\frac{f^{IV}(\mu_2)}{4!}\int_c^d (x-a)(x-c)(x-d)(x-b)\ dx + \\
&\frac{f^{IV}(\mu_3)}{4!}\int_d^b (x-a)(x-c)(x-d)(x-b)\ dx.
\end{aligned}$$

Realizando la siguiente sustitución en cada una de las integrales:

$$x - a = t(b-a),$$
$$x - c = a + t(b-a) - (2/3a + 1/3b) = (b-a)t + a/3 - b/3 = (b-a)(t-1/3),$$
$$x - d = (b-a)(t-2/3),$$
$$x - b = (b-a)(t-1).$$

nos quedaría la expresión:

$$\frac{f^{IV}(\mu_1)}{4!}(b-a)^5 \int_0^{1/3} t(t-1/3)(t-2/3)(t-1)\ dt+$$
$$\frac{f^{IV}(\mu_2)}{4!}(b-a)^5 \int_{1/3}^{2/3} t(t-1/3)(t-2/3)(t-1)\ dt+$$
$$\frac{f^{IV}(\mu_3)}{4!}(b-a)^5 \int_{2/3}^{1} t(t-1/3)(t-2/3)(t-1)\ dt.$$

Resolviendo de manera exacta las integrales se llega a:

$$E_h = \frac{(b-a)^5}{7290}\frac{1}{24}(-19f^{IV}(\mu_1) + 11f^{IV}(\mu_2) - 19f^{IV}(\mu_3)), \quad h = \frac{b-a}{3}. \tag{-16}$$

Si considerásemos iguales los μ : $\mu_1 = \mu_2 = \mu_3$ entonces podríamos simplificar la fórmula y nos quedaría precisamente el valor correspondiente al error del método de Simpson 3/8 (1.2.2):

$$E_h = -\frac{3}{80}h^5 f^{IV}(\mu). \tag{-16}$$

Sin embargo esto no sería correcto ya que no podemos igualar $\mu_1 = \mu_2 = \mu_3$ en la expresión (1.2.2) porque pertenecen a intervalos distintos. Ademas la expresión (1.2.2) no constituye una media con coeficientes positivos que permita llegar fácilmente a la expresión (1.2.2). Debemos buscar otro método que nos proporcione el valor del error.

Aun así este método lo usamos para la obtención del error tanto en el *Método de los Trapecios* como en el *Método de Simpson* ya que en ambos métodos no nos surge el problema de la negatividad en la expresión equivalente a la expresión (1.2.2) ni del cambio de signo en la integral (1.2.2). Esto es debido a que al hacer la aproximación con un numero tan pequeño de puntos nos se produce el cambio de signo.

1.2.3. Método correcto de análisis del error

Para la obtención del error nos basamos en un método en el que utilizaremos varias teorías matemáticas, siendo la principal de ellas la teoría del *Núcleo de Peano*.Para esta explicación vamos a utilizar la regla de *Simpson 3/8*.

Primero establecemos que la función del error tendrá la forma (En el caso de *Simpson 3/8*):

$$L(f) = \int_a^b f(x)\ dx - \frac{3h}{8}(f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)). \qquad (-16)$$

Debemos indicar que para aplicar este método necesitamos que $L(f)$ sea un **operador lineal**: $L : C^{k+1}[a,b] \longrightarrow \mathrm{R}$, efectivamente es lineal ya que:

$$L(\alpha f + \beta g) = \alpha L(f) + \beta L(g), \qquad (-16)$$

donde $\alpha,\ \beta\ \in\ \mathbf{R}$ y $f,\ g\ \in\ C^{k+1}[a,b]$.

Como hemos mencionado utilizamos el método conocido como la **Técnica del núcleo de Peano** para la obtención de la fórmula del error:

$$f(x) = f(a) + f'(a)(a-x) + ... + \frac{f^{(k)}(a)}{K!}(x-a)^k + \frac{1}{k!}\int_a^x (x-\theta)^k f^{k+1}(\theta)\ d\theta, \qquad \text{(-16)}$$

siendo la parte final de la ecuación (1.2.3) el **Error de Taylor** en la forma:

$$\frac{1}{k!}\int_a^x (x-\theta)^k f^{k+1}(\theta)\ d\theta \quad \theta \ \in \ (a,x).$$

Aplicando el operador L a ambos lados de la ecuación (1.2.3) llegamos a:

$$L(f) = L(p_k(x)) + L\left(\frac{1}{k!}\int_a^x (x-\theta)^k f^{(k+1)}(\theta)\ d\theta\right).$$

A partir de ahora vamos a hacer el desarrollo general con el indice k siendo en el caso particular de *Simpson 3/8* $k = 3$.

Eliminamos el término $L(p_k(x))$ ya que en el *Método de Simpson 3/8* es exacto de grado 3, por lo que queda:

$$L(f) = \frac{1}{k!}L\left(\int_a^x (x-\theta)^k f^{(k+1)}(\theta)\ d\theta\right).$$

Establecemos las condiciones para que todos los $(x-\theta)^k$ sean positivos definiendo:

$$(x-\theta)_+^k : \begin{cases} (x-\theta) & x \geq \theta, \\ 0 & x \leq \theta. \end{cases}$$

Basándonos en esta hipótesis para que nuestra función siempre nos dé positivo, escribimos que:

$$\int_a^x (x-\theta)^k f^{(k+1)}(\theta)\ d\theta = \int_a^b (x-\theta)_+^k f^{(k+1)}(\theta)\ d\theta,$$

siendo el termino $(x-\theta)_+^k \geq 0$.

Esta condición de positividad nos sirve para poder aplicar el *Teorema del valor medio integral* ya que en este caso al ser $(x-\theta)_+^k \geq 0$ no cambia de signo en todo el intervalo [a,b] y no se produce el error visto con anterioridad en el método explicado en el apartado 1.2.2 y podemos considerar que $f^{(k+1)}(\mu)$ con μ no dependiendo de θ es igual y se puede sacar como factor común:

$$\int_a^b (x-\theta)_+^k f^{(k+1)}(\theta)\ d\theta = f^{(k+1)}(\mu) \int_a^b (x-\theta)_+^k\ d\theta.$$

Ahora incorporando esto en L(f) podemos montar la ecuación:

$$L(f) = \frac{f^{(k+1)}(\mu)}{k!} L\left(\int_a^b (x-\theta)_+^k\ d\theta\right). \tag{-22}$$

Ya que:

$$L\left(\int_a^b (x-\theta)_+^k\ d\theta\right) = \int_a^b L\left((x-\theta)_+^k\right)\ d\theta.$$

Al integrando de esta última integral lo denominamos $k(\theta)$ por lo que la ecuación finalmente nos quedaría:

$$L(f) = \frac{f^{k+1}(\mu)}{k!} \int_a^b k(\theta)\ d\theta. \tag{-23}$$

Este será el método que utilizaremos para el cálculo del error a partir de *simpson 3/8*. Sólo debemos sustituir la k por el orden de la Regla utilizada y resolver la integral (1.2.3) , para las

distintas Reglas de *Newton Cotes*.

2. REGLAS DE NEWTON COTES CERRADAS

La Reglas de Newton Cotes cerradas son las que utilizamos en el caso de que la función a analizar sea continua en todo el intervalo de análisis como se muestra en la Figura 7:

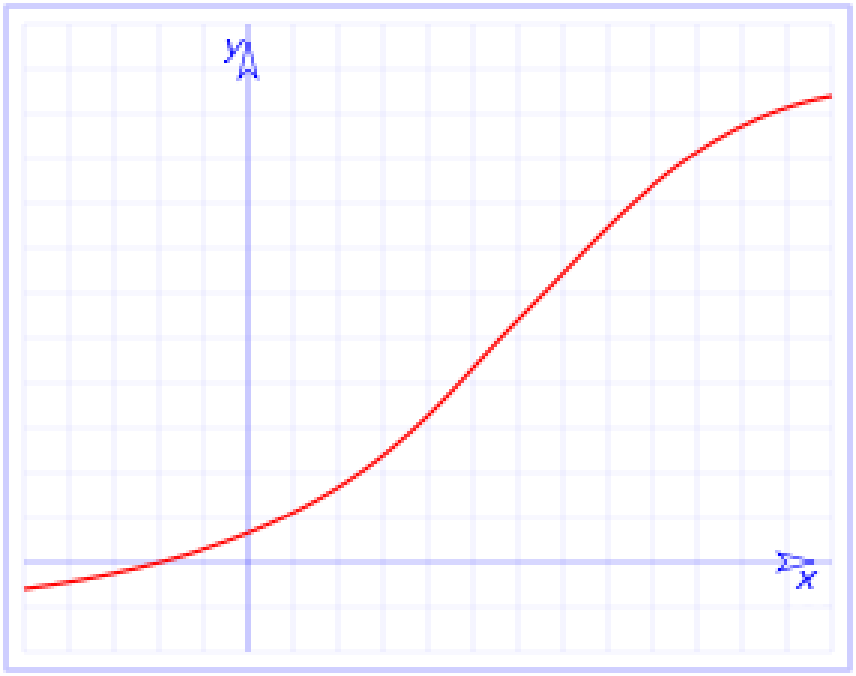

Figura 7: Representación de una función continua

A la hora de realizar la aproximación podemos optar por una de las Reglas presentadas a continuación, dependiendo del número de intervalos y por tanto de puntos en los que dividamos el dominio la función.

Nosotros hemos optado por realizar el análisis de las reglas hasta la regla de 6 intervalos, exponiendo su Regla simple, compuesta y los errores asociados a ambas, a continuación se muestra la regla simple para cada uno de los casos.

Las Reglas Simples que vamos a analizar son:

-Regla de **Trapecios**

$$\int_a^b f(x)\ dx \approx \frac{h}{2}(f(x_0) + f(x_1)),$$

con $h = b - a$.

-Regla de **Simpson**

$$\int_a^{xb} f(x)\ dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_1)),$$

con $h = \frac{b-a}{2}$.

-Regla de **Simpson 3/8**

$$\int_a^b f(x)\ dx \approx \frac{3}{8}h(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)),$$

con $h = \frac{b-a}{3}$.

-Regla de **Boole**

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)),$$

con $h = \frac{b-a}{4}$.

-Regla de **Regla de 5 intervalos**

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(19f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5)),$$

con $h = \frac{b-a}{5}$.

-Regla de **Regla de 6 intervalos**

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(41f(x_0) + 216f(x_1) + 27f(x_2) + 272f(x_3) + 27f(x_4) + 216f(x_5) + 41f(x_6)),$$

con $h = \frac{b-a}{6}$.

Estas son las ecuaciones que vamos a emplear para la resolución de problemas de ingeniería, como por ejemplo en el cálculo estructural, cálculo de elementos y características hidrostáticas, cálculo de momentos... y dependiendo de la función dada, la cantidad de datos que nos den o la aproximación que queramos aplicaremos la regla correspondiente.

La variable h denota el espaciado uniforme entre dos puntos de nuestra curva y viene dadó por la expresión:

$$h = \frac{b-a}{n}. \tag{-29}$$

Este valor de h se aplica a la Regla simple, siendo utiliza solo cuando hay un intervalo. La regla compuesta consiste en dividir el intervalo original, donde esta definida la curva, en distintos subintervalos de manera que en cada n subintervalos adyacentes se utiliza la regla simple. Para usar la regla compuesta el dominio de la función analizada debe estar dividido en un número de subintervalos que sea múltiplo de n. En este caso el espaciado h se calcula como:

$$h = \frac{b-a}{i\ n}, \tag{-29}$$

siendo i el número de intervalos correspondiente.

Por ejemplo en el caso de la *Regla de Simpson* que debe construir una parábola con 3 puntos y por tanto 2 intervalos, el valor de h será:

$$h = \frac{b-a}{2n}.$$

2.1. REGLA DE TRAPECIOS

2.1.1. Regla de Trapecios simple

Esta regla requiere de únicamente los puntos extremos de la curva, siendo la más sencilla de aplicar pero a su vez la que menor aproximación tendrá a la función original. Es utilizada en ingeniería para cálculos sencillos como en el caso de figuras básicas o por su simplicidad computacional usando la regla compuesta con muchos subintervalos, su representación gráfica se ve en la Figura 8.

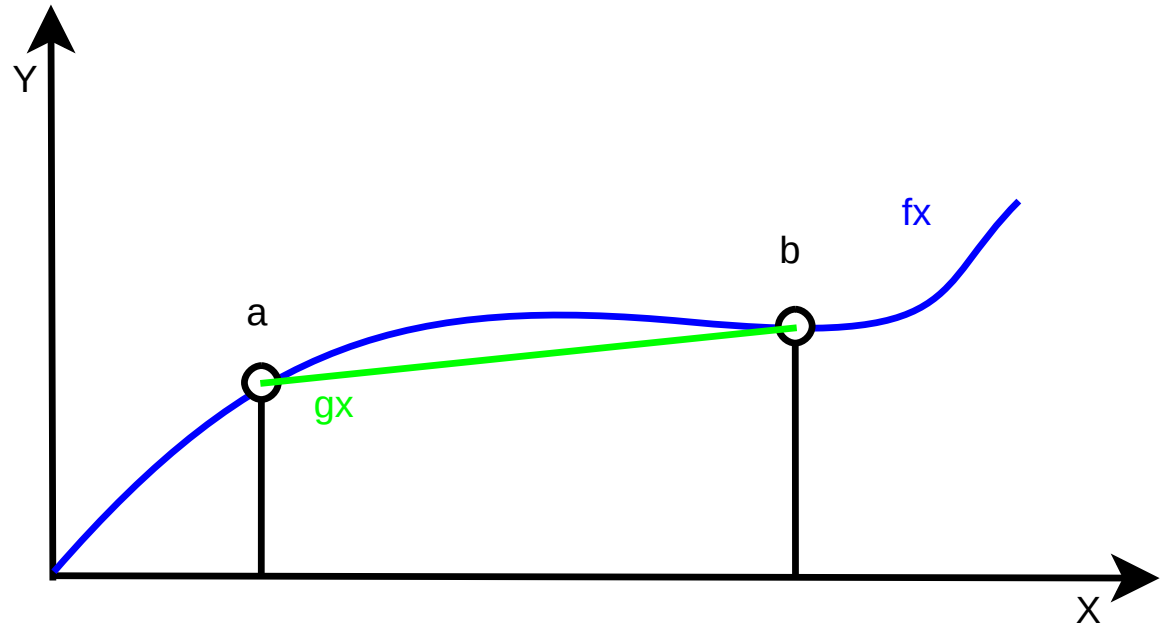

Figura 8: Aproximación de la función mediante Método de Trapecios Cerrada Simple

La fórmula de la Regla Simple obtenida a través de *Lagrange* será:

$$\int_a^b f(x)\ dx \approx \frac{h}{2}(f(x_0) + f(x_1)), \tag{-30}$$

que coincide con el área del trapecio constituido por los puntos (a,0), (a,f(a)), (b,0), (b,f(b)).

2.1.2. Error la Regla Trapecios Simple

Para el cálculo del error nos basamos en la fórmula descrita con anterioridad en el apartado 1.2.1, ya que en este caso aún no se producirá el error demostrado en el cálculo del *Método de Simpson Cerrado 3/8*:

$$E_h(f) = \int_a^b f(x)\ dx - \int_a^b P_1(x)\ dx = \int_a^b f(x) - P_1(x)\ dx = \frac{f''(\mu)}{2}\int_a^b (x-a)(x-b)\ dx.$$

Realizamos el siguiente cambio de variable para la resolución de la integral:

$$\begin{aligned} \frac{x-a}{b-a} &= t, \\ x &= a + (b-a)t, \\ dx &= (b-a)dt, \\ x - a &= (b-a)t, \\ x - b &= (b-a)(1-t). \end{aligned}$$

Quedándonos como:

$$E_h(f) = \frac{f''(\mu)}{2}(b-a)^3 \int_a^b t(t-1)\ dt = \frac{f''(\mu)}{2}(b-a)^3 \left[\frac{t^3}{3} - \frac{t^2}{2}\right]_0^1.$$

Resolviendo la integral obtenemos finalmente el valor del error de la *Regla de Trapecios Simple*:

$$E_h(f) = \frac{-f''(\mu)}{12}(b-a)^3. \qquad \text{(-37)}$$

Observamos que el valor de la derivada nos indica el Grado de exactitud de la fórmula, que en este caso es 1 y el valor de la potencia de h nos indica el orden de aproximación local, que ene este caso es 3

2.1.3. Regla de Trapecios Cerrada Compuesta

Para la Regla Compuesta consideramos el dominio de la función dividido en varios tramos de 2 puntos. Así:

$$\int_a^b f(x)\ dx \approx \frac{h}{2}(f(x_0)+f(x_1)) + \frac{h}{2}(f(x_1)+f(x_2)) + \frac{h}{2}(f(x_2)+f(x_3)) + ...$$

Sacando factor común:

$$\int_a^b f(x)\ dx \approx h(\frac{E}{2}+I), \tag{-38}$$

siendo los elementos de la expresión (2.1.3):

$$E = f(x_0)+f(x_n),$$
$$I = f(x_1)+f(x_2)...f(x_{n-1}).$$

Esta división nos permite reducir los cálculos que se deben hacer en *Matlab*.

Una de las ventajas de este método es que se puede aplicar a cualquier división que hayamos hecho del dominio de la curva ya que el subintervalo es de dos puntos.

La desventaja sin embargo es que es un método con mucha menor precision que los métodos analizados a continuación cuya aproximación irá escalando a medida que añadamos puntos al intervalo de análisis.

2.1.4. Programa de Matlab [trapecios(f,M,a,b)]

A la hora de hacer la programación en *Matlab*, esta Regla se expresara de la siguiente manera:

```
function [int]=trapecios(f,M,a,b)

% Integración numérica con el método de los trapecios
% [int]=trapecios(f,M,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% M => nº de subintervalos
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=trapecios('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo

h=(b-a)/M;

% Inicialización de variables
I=0;

% Cálculo de los nodos
x(1)=a;
x(M+1)=b;
for i=2:M
    x(i)=x(i-1)+h;
end

% Regla de trapecios

% Denotaremos por E a la suma de los ordenadas en los extremos
E=subs(f,x(1))+subs(f,x(M+1));

% Denotaremos por I a la suma de las ordenadas intermedias del método
for i=2:M
```

```
    I=I+subs(f,x(i));
end

 % Calculando el valor de la aproximación
int=h*((E/2)+I);
```

Final del programa de Matlab

2.1.5. Error de la Regla de Trapecios Compuesta

Para la regla compuesta el error es la conjunción de los errores de todos sus intervalos:

$$E_h(f) = \sum_{i=1}^{n} \frac{-1}{12} f''(\mu_i) h^3 = \frac{-h^3}{12} \sum_{i=1}^{n} f''(\mu_i)$$

Vamos a utilizar el resultado que asegura que toda función continua alcanza todos los valores entre su mínimo y su máximo en el intervalo [a,b] como se ve en la Figura 9.

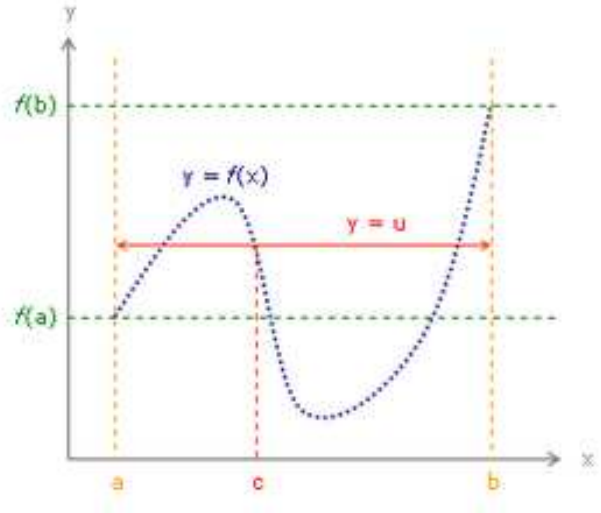

Figura 9: Teorema del valor medio

Si en el sumatorio anterior acotamos inferiormente por el mínimo (m) de $f^{''}(x)$ en [a,b] y superiormente por el máximo (M) de $f^{''}(x)$ en [a,b] tenemos:

$$nm \leq \sum_{i=1}^{n} f^{''}(\mu) \leq nM,$$

$$m \leq \sum_{i=1}^{n} \frac{f^{''}(\mu)}{n} \leq M.$$

Por tanto aplicando el resultado mencionado tenemos que existe $\mu \in (a, b)$:

$$\frac{\sum_{i=1}^{n} f^{''}(\mu)}{n} = f^{II}(\mu). \tag{-43}$$

multiplicando y dividiendo por n en la ecuación inicial, y teniendo en cuenta que $n = \frac{b-a}{h}$,

$$E_h(f) = \frac{-h^3}{12} \frac{\sum_{i=1}^{n} f^{''}(\mu_i)}{n} n = \frac{-h^3}{12} f^{''}(\mu) \frac{(b-a)}{h}$$

Quedando finalmente:

$$E_h(f) = \frac{-h^2}{12} f^{''}(\mu)(b-a), \tag{-44}$$

siendo este error (2.1.5) exacto de grado 1 y de orden de aproximación global 2, viendo que se ha reducido en 1 respecto al orden de aproximación local.

2.2. REGLA DE SIMPSON

2.2.1. Regla de Simpson Simple

En el caso de la *Regla de Simpson* el dominio de la curva debe estar dividida en dos intervalos (con 2 puntos extremos y uno central), éste es uno de los métodos más aplicado en la práctica ya que nos proporciona un resultado apropiado con un error suficientemente bajo. Un ejemplo gráfico lo vemos en la Figura 10 donde se aproxima el area bajo la curva por el área bajo la parábola que pasa por los tres puntos $(a, f(a))$, $(\frac{a+b}{2}, f(\frac{a+b}{2}))$, $(b, f(b))$:

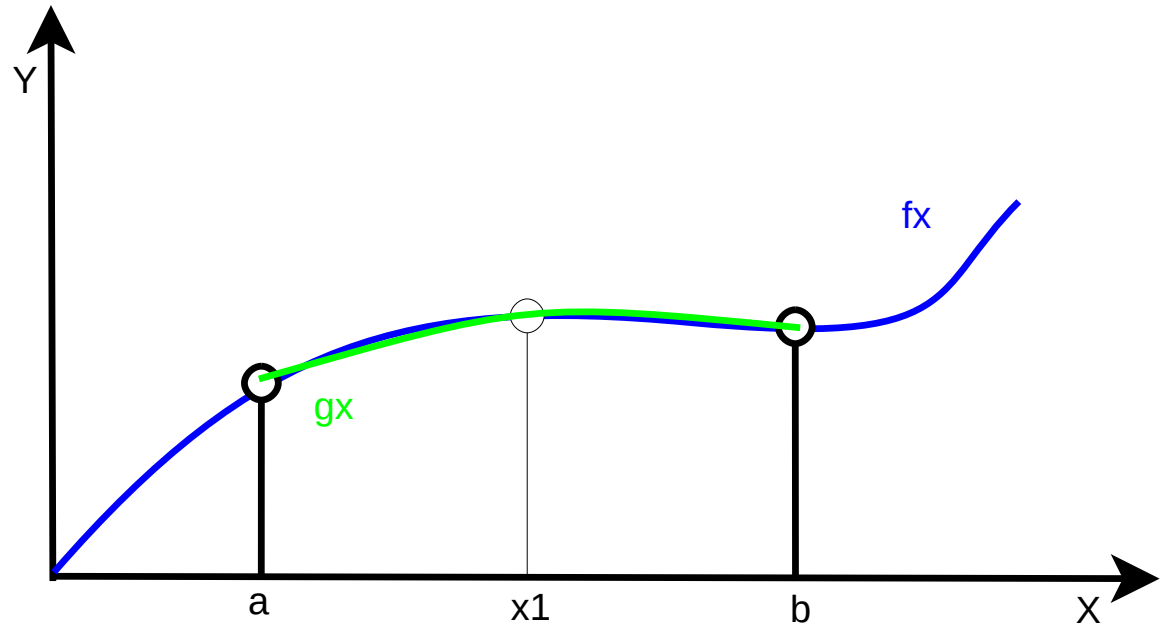

Figura 10: Aproximación de la integral mediante Método de Simpson

Para la obtención de la Regla utilizamos el *Polinomio Interpolador de Lagrange* obteniendo de ese modo:

$$p_n(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2),$$

Donde hemos llamado $x_0 = a$, $x_1 = \frac{a+b}{2}$ y $x_2 = b$.

$$\int_a^b p(x)\ dx = \int_a^b L_0(x)f(x_0)\ dx + \int_a^b L_1(x)f(x_1)\ dx + \int_a^b L_2(x)f(x_2)\ dx,$$
$$= f(x_0)\int_a^b L_0(x)\ dx + f(x_1)\int_a^b L_1(x)\ dx + f(x_2)\int_a^b L_2(x)\ dx.$$

Denotando $h = x_2 - x_0$ mediante la misma sustitución que hemos hecho en la Regla del Trapecio podemos calcular las integrales obteniendo:

$$\int_a^b f(x)\ dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)). \tag{-47}$$

2.2.2. Error de la Regla de Simpson Simple

Para la obtención del *Error de la Regla de Simpson Simple* utilizamos la fórmula del error descrita en el apartado 1.2.1, que como en el caso anterior, al no tener tantos puntos en el intervalo podemos arreglar el problema del signo y así podemos aplicarla.

$$E_h(f) = \int_a^b f(x)\ dx - \int_a^b P_2(x)\ dx$$

En primer lugar vemos que la regla de la expresión 2.2.1 es exacta para polinomios de grado menor o igual que 3, ya que:

$$\int_a^b 1\ dx = \frac{b-a}{6}(1+4+1),$$
$$\int_a^b (x-a)\ dx = \frac{b-a}{6}\left(0 + 4\left(\frac{a+b}{2} - a\right) + (b-a)\right),$$

$$\int_a^b (x-a)^2 \, dx = \frac{b-a}{6}\left(0 + 4\left(\frac{a+b}{2} - a\right)^2 + (b-a)^2\right),$$

$$\int_a^b (x-a)^3 \, dx = \frac{b-a}{6}\left(0 + 4\left(\frac{a+b}{2} - a\right)^3 + (b-a)^3\right).$$

Introducimos el polinomio auxiliar $\tilde{P}_3(x)$ de *Mermite* que satisface $f(a) = \tilde{P}_3(a)$, $f(\frac{a+b}{2}) = \tilde{P}_3(\frac{a+b}{2})$, $f'(\frac{a+b}{2}) = \tilde{P}_3'(\frac{a+b}{2})$, $f(b) = \tilde{P}_3(b)$.

Es conocido según la teoría de interpolación que:

$$f(x) - \tilde{P}_3(x) = \frac{f^{IV}(\mu_x)}{4!}(x-a)\left(x - \frac{a+b}{2}\right)^2 (x-b) \tag{-52}$$

Además como $\tilde{P}_3(x)$ es un polinomio de grado 3, y la fórmula simple es exacta de grado 3,

$$\int_a^b \tilde{P}_3(x) \, dx = \int_a^b P_2(x) \, dx.$$

Así

$$E_h(f) = \int_a^b f(x) \, dx = \int_a^b P_2(x) \, dx =$$

$$\int_a^b f(x) \, dx - \int_a^b \tilde{P}_3(x) \, dx + \int_a^b \tilde{P}_3(x) \, dx - \int_a^b P_2(x) \, dx =$$

$$\int_a^b f(x) - \tilde{P}_3(x) \, dx.$$

De aqui llegamos a que:

$$E_h(f) = \int_a^b \frac{f^{IV}(\mu_x)}{4!}(x-a)\left(x-\frac{a+b}{2}\right)^2(x-b)\ dx. \tag{-56}$$

Ahora estamos en condiciones de aplicar el teorema del valor medio integral llamado a

$$E_h(f) = \frac{f^{IV}(\mu)}{4!}\int_a^b (x-a)\left(x-\frac{a+b}{2}\right)^2(x-b)\ dx.$$

Mediante la misma sustitución $x = a+(b-a)t$ que hemos aplicado en *Trapecios* calculamos la integral:

$$E_h(f) = \frac{f^{IV}(\mu)}{4!}(b-a)^5\int_0^1 t(t-1/2)^2(t-1)\ dt$$

$$E_h(f) = \frac{-1}{24}\frac{f^{IV}(\mu)}{120}(b-a)^5 = \frac{-1}{2880}f^{IV}(\mu)(b-a)^5$$

Otra forma de escribir el error es:

$$E_h(f) = \frac{f^{IV}(\mu)}{90}\left(\frac{(b-a)}{2}\right)^5 \tag{-59}$$

Como se observa en la fórmula el error de simpson es de grado de exactitud 3 y orden de aproximación local 5 por lo que ha aumentado en 2 respecto a Trapecios, esto se puede apreciar mediante el siguiente ejemplo:

2.2.3. Comparación entre el error de Simpson y Trapecios

Aplicamos ambas Reglas sobre una misma función: $f(x) = cos(2x) + 1$, con x perteneciente al intervalo [0, 0.25] y comprobamos su error mediante la comparación con la integral exacta.

Observamos que:

$$\begin{aligned} resultado\ de\ Trapecios &= 0{,}078011838373009, \\ resultado\ de\ Simpson &= 0{,}078012000898096, \\ resultado\ integral\ exacta &= 0{,}078012000898079. \end{aligned}$$

Nota: *Todos los resultados han sido obtenidos por los programas de* Matlab *correspondientes y el resultado de la integral exacta mediante el comando* double (int (cos(2*x+1),0,0.25)).

Como se comprueba el valor obtenido en Simpson es mucho mas próximo que el de Trapecios siendo exacto hasta la catorceava cifra decimal con esta función.

2.2.4. Regla de Simpson Compuesta

Para la *Regla Compuesta de Simpson* partimos el intervalo original en subintervalos de tres puntos y aplicamos la regla simple en cada uno de los subintervalos en cada uno de los subintervalos construimos una parábola basada en los datos $[(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))]$, $[(x_2, f(x_2)), (x_3, f(x_3)), (x_4, f(x_4))]...[(x_{n-2}, f(x_{n-2})), f(x_{n-1}, f(x_{n-1})), (x_n, f(x_n))]$. Finalmente sumamos el resultado de cada regla simple para obtener el resultado buscado.

El dominio de la función debe estar obligatoriamente dividida en un numero de subintervalos múltiplo de 2. Para conseguir esto definimos $h = \frac{b-a}{2n}$ $x_i = a + ih$, $i = 0, ..., n$ donde n es el numero de veces que se aplica la regla simple.

Como podemos observar en la Figura 11 el dominio de la función f(x) esta dividido en 13 puntos componiendo de este modo 6 veces la *Regla de Simpson Cerrada Simple*.

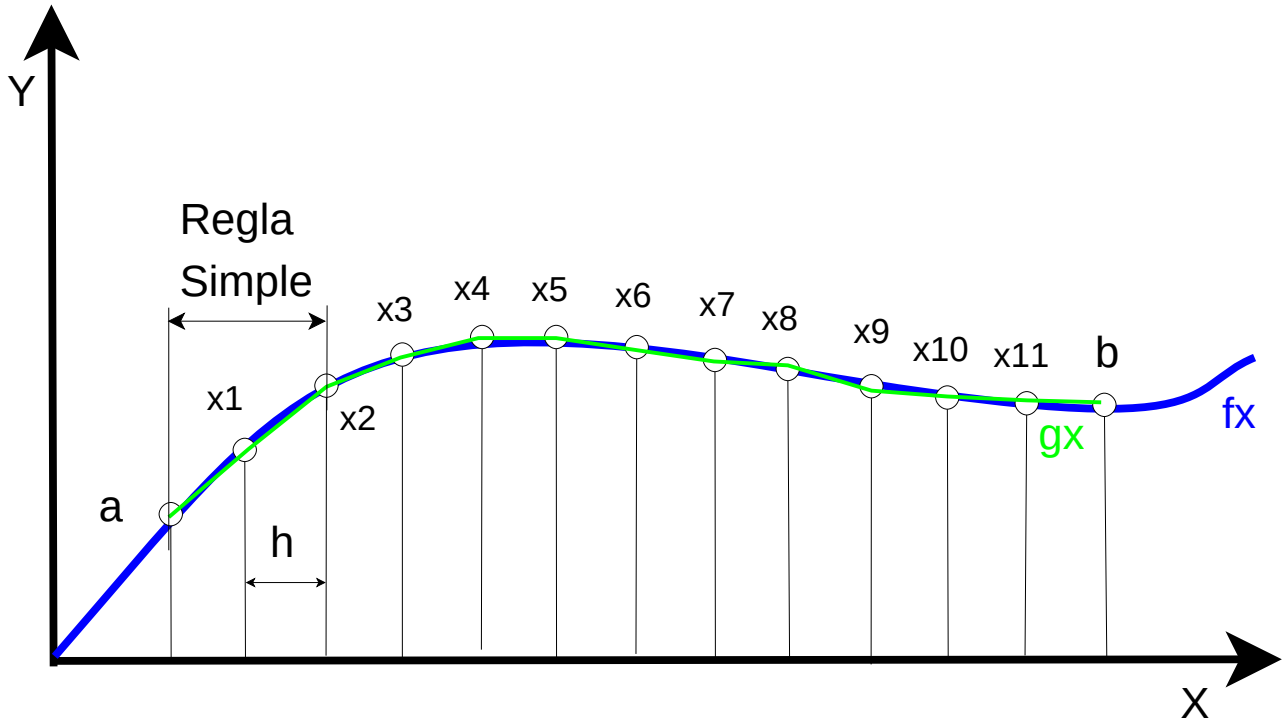

Figura 11: Regla compuesta de Simpsón

La aproximación que dará al al fórmula de la Regla Compuesta será:

$$\int_a^b f(x)\ dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)) + \frac{h}{3}(f(x_2) + 4f(x_3) + f(x_4)) + ...$$
$$... + \frac{h}{3}(f(x_{2n-2}) + 4f(x_{2n-1}) + f(x_{2n})) = \frac{h}{3}(E + 4I + 2P),$$

es decir,

$$\int_a^b f(x)\ dx \approx \frac{h}{3}(E + 4I + 2P), \tag{-64}$$

siendo el valor de cada uno de los elementos E, I, P:

$$E = f(x_0) + f(x_n),$$
$$I = f(x_1) + f(x_3) + ... + f(x_{2n-1}),$$
$$P = f(x_2) + f(x_4) + ... + f(x_{2n-2}).$$

A la hora de hacer la programación en *Matlab*, esta Regla se expresara de la siguiente manera.

2.2.5. Programa de Matlab [simpson(f,M,a,b)]

```
function [int]=simpson(f,n,a,b)

% Aplicamos la regla de Simpson para hacer integración numérica.
% [int]=simpson(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Simpson.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=simpson('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(2*n);

% Inicialización de variables
Ii=0;
Ip=0;

% Cálculo de los nodos
x(1)=a;
x(2*n+1)=b;
```

```
for i=2:2*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Simpson

% Denotaremos por E a la suma de los valores de los extremos
E=subs(f,x(1))+subs(f,x(2*n+1));

% Denotaremos por Ii a la suma de las ordenadas intermedias impares (pares en Matlab)
en el método
for i=2:2:2*n
    Ii=Ii+subs(f,x(i));
end

% Denotaremos por Ii a la suma de las ordenadas intermedias pares (impares en Matlab)
en el método
for i=3:2:2*n-1
    Ip=Ip+subs(f,x(i));
end

% Calculando el valor de la aproximación
int=(h/3)*(E+4*Ii+2*Ip);
```

Final del programa de Matlab

2.2.6. Error de la Regla de Simpson Compuesta

Para el cálculo del error en *Simpson Compuesto* debemos realizar el sumatorio de todos los errores que se cometerán al aplicar la Regla Simple un numero n de veces para crear la regla compuesta:

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{2(i-1)}}^{x_{2i}} \left(f(x) - p_{2,i}(x)\right)\, dx,$$

$$E_h(f) = \sum_{i=1}^{n} \frac{-f^{IV}(\mu_i)}{90} h^5 = \frac{-h^5}{90} \left(\frac{\sum_{i=1}^{n} f^{IV}(\mu_i)}{n} \right) n.$$

Teniendo en cuenta que existe $\mu \in (a, b)$ tal que

$$\frac{\sum_{i=1}^{n} f^{IV}(\mu_i)}{n} = f^{IV}(\mu), \tag{-69}$$

y que $h = \frac{b-a}{2n}$, llegamos a

$$E = -\frac{(b-a)}{180} h^4 - f^{IV}(\mu_i) \tag{-69}$$

Por lo que en el caso de la *Regla de Simpson Compuesto* tendremos un grado de exactitud 3 con un orden de aproximación global 4 (indicado por la potencia de h).

2.3. REGLA DE SIMPSON 3/8

2.3.1. Regla de Simpson 3/8 Simple

Esta Regla la empleamos para determinar el área bajo una función aproximandolo mediante el área bajo un polinomio de grado 3 que conecta 4 puntos sobre una curva dada: Como puede verse en la Figura 12.

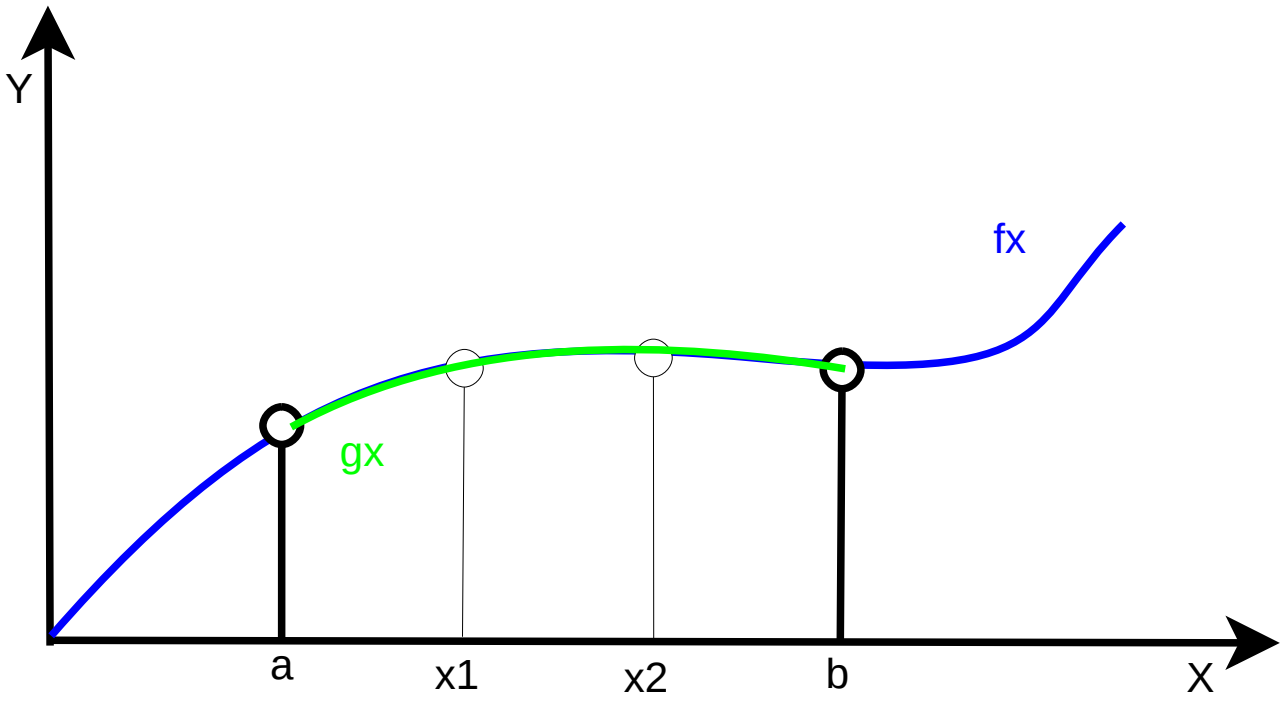

Figura 12: Regla de Simpsón 3/8

Por lo que es muy útil por ejemplo cuando tenemos una serie de puntos que no podemos utilizar en la *Regla de Simpson* porque son un numero par y no encajaría en la Regla compuesta.Lo que podemos hacer es aplicar esta regla en esos puntos y el resto aplicar la *Regla de Simpson Compuesta*.

Un ejemplo se encuentra en el cálculo de la superficie de carena de un determinado buque en la que la parte central la dividimos en intervalos de 3 pero en los y los extremos en intervalos de 4, por lo que la combinación de ambas nos proporcionará el resultado mas óptimo.

Para su obtención utilizamos, del mismo modo que para los casos anteriores, el **Polinomio de Lagrange** en este caso de grado 3:

$$p_3(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2) + L_3(x)f(x_3).$$

Sabiendo que los puntos se pueden definir como:

$$\begin{aligned} x_0 &= a, b \\ x_1 &= c = 2/3a + 1/3b, \\ x_2 &= d = 1/3a + 2/3b, \\ x_3 &= b. \end{aligned}$$

Integrando el polinomio $P_3(x)$ llegamos a la siguiente expresión:

$$\int_a^b f(x)\ dx \approx \frac{3}{8}h(f(x_0) + 3f(x_1) + 3f(x_1) + f(x_2)). \tag{-74}$$

2.3.2. Error de la Regla de Simpson 3/8 Simple

Para la obtención del error usamos el método descrito en el punto 1.3.3. Debemos calcular el *Nucelo de Peano* $k(\theta)$ correspondiente al valor de $k = 3$, lo que hacemos ahora es calcular el elemento $L(x-\theta)_+^k$,

$$L(x-\theta)_+^3 = \int_a^b (x-\theta)_+^3 \cdot d\theta - \frac{3h}{8}\left((a-\theta)_+^3 + 3\left(\left(\frac{2a+b}{3} - \theta\right)_+^3\right) + 3\left(\left(\frac{a+2b}{3} - \theta\right)_+^3\right) + (b-\theta)_+^3\right).$$

El termino $(a-\theta)_+^3$ se anula debido a que en este caso el valor de $\theta \geq a$.

Laintegral $\int_a^b (x-\theta)_+^3 \, d\theta$ la podemos resolver como:

$$\int_a^b (x-\theta)_+^3 \, d\theta = \int_\theta^b (x-\theta)^3 \, d\theta = \left[-\left(\frac{(x-\theta)^4}{4} \right) \right]_\theta^b = \frac{1}{4}(b-\theta)^4.$$

Por lo que la formula de $k(\theta) = L(k(\theta))_+^3$ tendría la siguiente forma:

$$k(\theta) = \frac{1}{4}(b-\theta)^4 - \frac{3h}{8}\left(3\left(\left(\frac{2a+b}{3} - \theta \right)_+^3 \right) + 3\left(\left(\frac{a+2b}{3} - \theta \right)_+^3 \right) + (b-\theta)_+^3 \right).$$

Descomponemos entonces cada parte de la ecuación en sus términos según el rango del valor de θ:

$$-\frac{3h}{8}\begin{cases} 3\left(\left(\frac{2a+b}{3} - \theta \right)^3 \right) + 3\left(\left(\frac{a+2b}{3} - \theta \right)^3 \right) + (b-\theta)^3 \ , & a \leq \theta \leq \frac{2a+b}{3} \\ 3\left(\left(\frac{a+2b}{3} - \theta \right)^3 \right) + (b-\theta)^3 \ , & \frac{2a+b}{3} \leq \theta \leq \frac{a+2b}{3} \\ (b-\theta)^3 \ , & \frac{a+2b}{3} \leq \theta \leq b \end{cases}$$

$$\int_a^b k(\theta) \, d\theta = \int_a^{\frac{2a+b}{3}} k(\theta) \, d\theta + \int_{\frac{2a+b}{3}}^{\frac{a+2b}{3}} k(\theta) \, d\theta + \int_{\frac{a+2b}{3}}^b k(\theta) \, d\theta = \frac{1}{4}\int_a^b (b-\theta)^4 \, d\theta$$
$$- \left(\frac{3h}{8}\int_a^b (b-\theta)^3 \, d\theta + \frac{3h}{8}\int_a^{\frac{a+2b}{3}} 3\left(\frac{a+2b}{3} - \theta \right)^3 \, d\theta + \frac{3h}{8}\int_a^{\frac{2a+b}{3}} 3\left(\frac{2a+b}{3} - \theta \right)^3 \, d\theta \right).$$

Resolviendo las integrales nos quedaría de la siguiente forma:

$$\int_a^b k(\theta) \, d\theta = \frac{1}{4}\frac{(b-a)^5}{5} - \frac{3h}{8}\left(\frac{(b-a)^4}{4} + 3\frac{\left(\frac{2}{3}(b-a) \right)^4}{4} + \frac{3\left(\frac{(b-a)}{3} \right)^4}{4} \right).$$

En el caso de la regla de *Simpson 3/8* el valor de $h = \frac{b-a}{3}$ por lo que al sustituir h por su valor correspondiente nos quedaría:

$$\int_a^b k(\theta)\ d\theta = \frac{1}{4}\frac{(b-a)^5}{5} - \frac{99h^5}{8} = -\frac{9}{40}h^5.$$

Si este valor lo introducimos dentro de la fórmula del error:

$$E_h(f) = \frac{f^{IV}(\mu)}{3!}\left(\frac{-9}{40}h^5\right).$$

Lo que nos daría finalmente la fórmula del error aplicada a *Simpson 3/8*:

$$E_(f) = -\frac{3}{80}h^5 f^{IV}(\mu), \tag{-83}$$

con $\mu\ \in (a, b)$.

Vemos que es un error con grado de exactitud 3, igual que en *Simpson* y con orden de aproximación local 5, también igual que en *Simpson*.

En general entonces es mejor aplicar Simpson, que es menos costoso computacionalmente, y posee el mismo grado de exactitud y orden de aproximación.

2.3.3. Regla de Simpson 3/8 Compuesta

La *Regla Compuesta de Simpson 3/8* se resuelve de la misma forma que las anteriores, teniendo la particularidad de que en este caso el intervalo lo componen 4 puntos, lo que conlleva que su aplicación está más restringida que para otros métodos:

$$\int_a^b f(x)\ dx \approx \frac{3}{8}h(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)) + \frac{3}{8}h(f(x_3) + 3f(x_4) + 3f(x_5) + f(x_6)) + ...$$
$$... + \frac{3}{8}h(f(x_{3n-3}) + f(x_{3n-2}) + 4f(x_{3n-1}) + f(x_{3n})).$$

Si lo englobamos todo en una ecuación más simple, facilitando el trabajo de cálculo de *Matlab* nos quedaría la expresión:

$$\int_a^b f(x)\ dx \approx \frac{3}{8}h(E + 3I + 2P), \tag{-85}$$

siendo E (Extremos del intervalo), I (Valores intermedios) y P (Extremos acumulables),

$$E = f(x_0) + f(x_3n),$$
$$I = f(x_1) + f(x_2) + f(x_4) + f(x_5)... + f(x_3n - 2) + f(x_3n - 1),$$
$$P = f(x_3) + f(x_6) + ... + f(x_3n - 3).$$

Su programación en *Matlab* se representa de la siguiente forma

2.3.4. Programa de Matlab [simpson3_8(f,M,a,b)]

```
function [int]=simpson3_8(f,n,a,b)

% Aplicamos la regla de Simpson 3/8 para hacer integración numérica.
% [int]=simpson3_8(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Simpson 3/8 simple
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=simpson3_8('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(3*n);
```

```
% Inicialización de variables (Variables no extremos)
I1=0;
I2=0;
I3=0;

% Cálculo de los nodos

x(1)=a;
x(3*n+1)=b;

for i=2:3*n
    x(i)=x(i-1)+h;
end


% Regla de Simpson 3/8

% Denotaremos por E a la suma de los valores de los extremos
E=subs(f,x(1))+subs(f,x(3*n+1));

% Denotaremos por I1 a la suma de las ordenadas intermedias múltiplos de 3 más 1

for i=2:3:3*n-1
    I1=I1+subs(f,x(i));
end


% Denotaremos por I2 a la suma de las ordenadas intermedias múltiplos de 3 más 2

for i=3:3:3*n
    I2=I2+subs(f,x(i));
end

% Denotaremos por I3 a la suma de las ordenadas intermedias múltiplos de 3

for i=4:3:3*n-2
    I3=I3+subs(f,x(i));
end

% Calculando el valor de la aproximación
```

int=(3*h/8)*(E+3*I1+3*I2+2*I3);

Final del programa de Matlab

2.3.5. Error de la Regla de Simpson 3/8 Compuesta

A través del Error de la Regla Simple podemos, mediante la suma de todos los errores de las veces que en la Regla Compuesta aplicamos la simple, obtener el error debido a su aplicación:

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{3(i-1)}}^{x_{3i}} (f(x) - p_{3,i}(x))\ dx,$$

$$E_h(f) = \sum_{i=1}^{n} -\frac{3}{80} h^5 f^{IV}(\mu_i) = \frac{-3h^5}{80} \left(\frac{\sum_{i=1}^{n} f^{IV}(\mu_i)}{n} \right) n.$$

Sustituyendo una de las h por $h = \frac{b-a}{3n}$ y aplicando que el término entre paréntisis es un valor intermedio entre el mínimo y el máximo de la derivada cuarta de la función $f(x)$ en [a,b] llegamos a:

$$E_h(f) = -\frac{(b-a)}{80} h^4 f^{IV}(\mu) \qquad \text{(-90)}$$

Con $\mu \in (a, b)$

Dicha fórmula del error tiene un grado de exactitud 3 y un orden de aproximación global 4. Hacemos notar que el orden de aproximación en la fórmula compuesta, como ocurre siempre, se reduce en una unidad respecto al obtenido en la Regla Simple, como pasa también en todos los demás métodos.

2.4. REGLA DE BOOLE

2.4.1. Regla de Boole Simple

La regla de *Regla de Boole* también denominada en algunos libros como Regla de *Milne* [4] es aquella en la que utilizamos 5 puntos de una función determinada para la aproximación del área bajo la función, tal y como se muestra en la Figura 13.

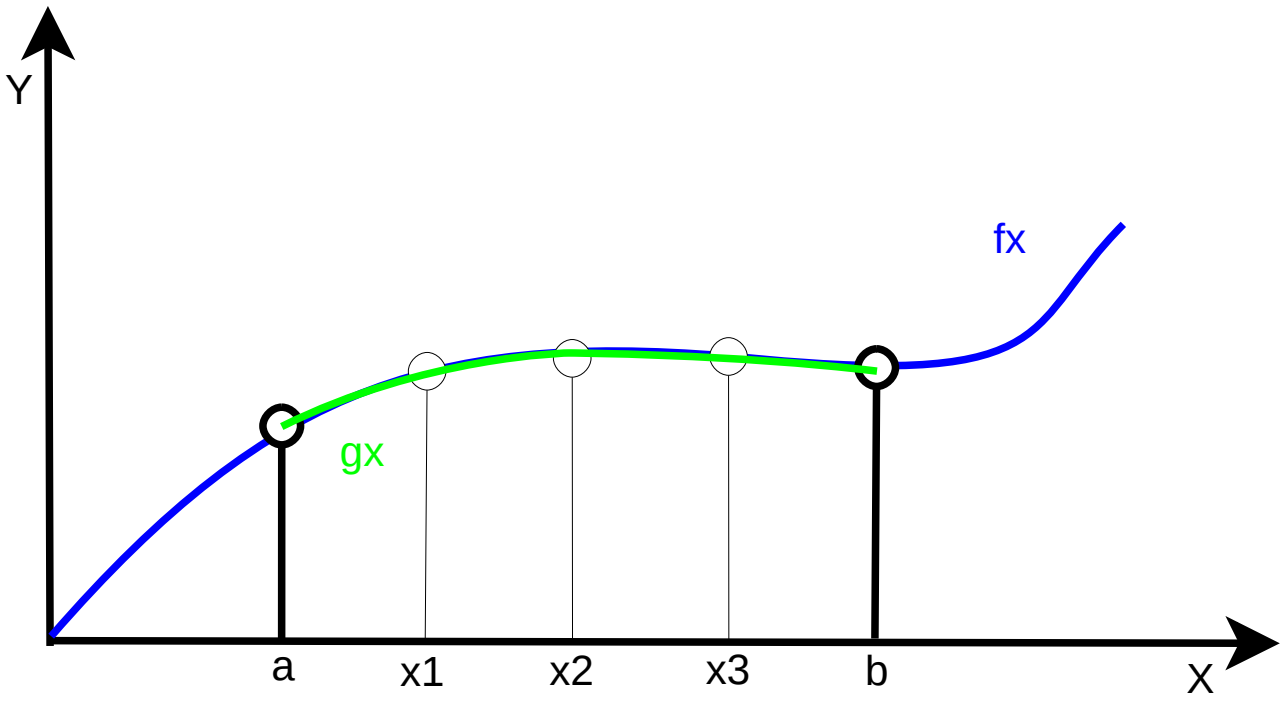

Figura 13: Regla de Boole

Volviendo a hacer uso del **Polinomio de Lagrange** para la obtención de su fórmula simple se llega a:

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)). \qquad \text{(-90)}$$

2.4.2. Error de la Regla de Boole Simple

Debido a lo explicado en el apartado 1.2.3 del análisis del error, en este caso debemos aplicar al igual que en *Simpson 3/8* la nueva forma de obtención del error ya que en *Boole* se produce el mismo fallo visto anteriormente.

En este caso $k = 5$ ya que sabemos que para polinomios de grado 5 la regla es exacta. Calculando tenemos:

$$L(x-\theta)_+^5 = \int_a^b (x-\theta)_+^5 \, d\theta - \frac{2h}{45}(7(a-\theta)_+^5 + 32\left(\left(\frac{3a+b}{5}-\theta\right)_+^5\right) + 12\left(\left(\frac{2a+2b}{5}-\theta\right)_+^5\right)$$
$$+32\left(\left(\frac{a+3b}{5}-\theta\right)_+^5\right) + (b-\theta)_+^5). \qquad \text{(-91)}$$

El valor del elemento $7(a-\theta)_+^5 = 0$ ya que en este caso $a \leq \theta$ y la integral la podemos calcular:

$$\int_a^b (x-\theta)_+^5 \, d\theta = \left[-\frac{(x-\theta)^6}{6}\right]_\theta^b = \frac{1}{6}(b-\theta)^6. \qquad \text{(-91)}$$

Volvemos a descomponer el intervalo (a,b) en subintervalos para integrar:

$$\int_a^b k(\theta)\, d\theta = \int_a^{\frac{3a+b}{5}} k(\theta)\, d\theta + \int_{\frac{3a+b}{5}}^{\frac{a+b}{2}} k(\theta)\, d\theta + \int_{\frac{a+b}{2}}^{\frac{a+3b}{5}} k(\theta)\, d\theta + \int_{\frac{a+3b}{5}}^b k(\theta)\, d\theta =$$
$$\frac{1}{6}\int_a^b (b-\theta)^6 \, d\theta - (\frac{2h}{45}\int_a^b (b-\theta)^5 \, d\theta + \frac{2h}{45}\int_a^{\frac{a+3b}{5}} 32\left(\frac{a+3b}{5}-\theta\right)^5 \, d\theta +$$
$$\frac{2h}{45}\int_a^{\frac{a+b}{2}} 12\left(\frac{a+b}{2}-\theta)^5 \, d\theta\right) + \frac{2h}{45}\int_a^{\frac{3a+b}{5}} 32\left(\frac{3a+b}{5}-\theta\right)^5 \, d\theta.$$

Resolvemos las integrales y nos queda:

$$\int_a^b k(\theta)\ d\theta = \frac{1}{6}\frac{(b-a)^7}{7} - \frac{2h}{45}\left(\frac{(b-a)^6}{6} + \frac{32\left(\frac{(3b-3a)}{4}\right)^6}{6} + \frac{12\left(\frac{(b-a)}{2}\right)^6}{6} + 32\frac{\left(\frac{(b-a)}{4}\right)^6}{6}\right).$$

En el caso de la *Regla de Boole* el valor de $h = \frac{b-a}{5}$ y simplificando obtenemos:

$$E_h(f) = -\frac{8}{945}h^7 f^{VI}(\mu) \tag{-95}$$

Con $\mu\ \in (a, b)$.

Se puede observar que en el caso del error en la *Regla de Boole* se ha incrementado el grado del error en 2 unidades, respecto al error en el caso de *Simpson 3/8*, siendo éste 5.El orden de aproximación local también se a incrementado en 2 unidades y ahora es de grado 7.

2.4.3. Regla de Boole Compuesta

La Regla Compuesta para Boole se obtiene de igual manera que las anteriores, mediante el sumatorio de las fórmulas simples de todos los conjuntos de 4 intervalos equispaciados adyacentes.

$$\begin{aligned}\int_a^b f(x)\ dx \approx \frac{2}{45}h(7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)) + \\ \frac{2}{45}h(7f(x_4) + 32f(x_5) + 12f(x_6) + 32f(x_7) + 7f(x_8)) + ... \\ ... + \frac{2}{45}h(7f(x_{4n-4}) + 32f(x_{4n-3}) + 12f(x_{4n-2}) + 32f(x_{4n-1}) + 7f(x_{4n})).\end{aligned}$$

Si lo englobamos todo en una ecuación más simple, facilitando el trabajo de cálculo de *Matlab* nos quedaría una expresión como:

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(7E + 32I_i + 12I_p + 14I_e), \tag{-98}$$

siendo el valor de cada uno de los elementos E(Extremos del intervalo), I_i (Valores intermedios impares), I_p (Valores centrales pares) y I_e (Extremos acumulables),

$$E = f(x_0) + f(x_{4n}),$$

$$I_i = f(x_1) + f(x_3) + f(x_5) + f(x_7)... + f(x_4n - 3) + f(x_{4n-1}),$$

$$I_p = f(x_2) + f(x_6) + f(x_{10})... + f(x_{4n-2}),$$

$$I_e = f(x_4) + f(x_8) + ... + f(x_{4n-4}).$$

2.4.4. Programa de Matlab [Boole(f,M,a,b)]

Su programación en matlab se representa de la siguiente forma:

```
function [int]=Boole(f,n,a,b)

% Aplicamos la regla de Boole para hacer integración numérica.
% [int]=Boole(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Boole simple.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Boole('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(4*n);
```

```
% Inicialización de variables
Ii=0;
Ip=0;
Ie=0;

% Cálculo de los nodos
x(1)=a;
x(4*n+1)=b;

for i=2:4*n
    x(i)=x(i-1)+h;
end

% Regla de Boole

% Denotaremos por E a la suma de los valores de los extremos
E=subs(f,x(1))+subs(f,x(4*n+1));

% Denotaremos por Ii a la suma de las ordenadas intermedias impares (pares en Matlab)
en el método
for i=2:2:4*n
    Ii=Ii+subs(f,x(i));
end

% Denotaremos por Ip a la suma de las ordenadas intermedias pares (impares en Matlab)
en el método
for i=3:4:4*n-1
    Ip=Ip+subs(f,x(i));
end

% Denotamos por Ie a la suma de las ordenadas donde pega un polinomio de
% grado 4 con otro

for i=5:4:4*n-3
    Ie=Ie+subs(f,x(i));
end


% Calculando el valor de la aproximación
int=(2*h/45)*(7*E+32*Ii+12*Ip+14*Ie);
```

Final del programa de Matlab

2.4.5. Error de la Regla de Boole Compuesta

El sumatorio del los errores simples cometidos en cada uno de los distintos intervalos nos dará como resultado el valor del error de la regla compuesta:

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{4(i-1)}}^{x_{4i}} (f(x) - p_{4,i}(x))\, dx,$$

$$E_h(f) = \sum_{i=1}^{n} -\frac{8}{945} h^7 f^{VI}(\mu_i) = \frac{-8h^7}{945} \left(\frac{\sum_{i=1}^{n} f^{VI}(\mu_i)}{n} \right) n.$$

Sustituyendo una de las h por $h = \frac{b-a}{4n}$llegamos a:

$$E_h(f) = -\frac{2(b-a)}{945} h^6 f^{VI}(\mu), \tag{-104}$$

Con $\mu \in (a, b)$

En este caso el error tiene grado de exactitud 5 y orden de aproximación global 6.

2.5. REGLA DE NEWTON COTES BASADA EN 5 INTERVALOS

2.5.1. Regla Simple basada en 5 intervalos

En el caso de la Regla de esta Regla, el dominio de definición de la curva debe estar dividida en cinco intervalos, con los puntos extremos y cuatro puntos centrales. La situacion gráfica la vemos en la Figura 14:

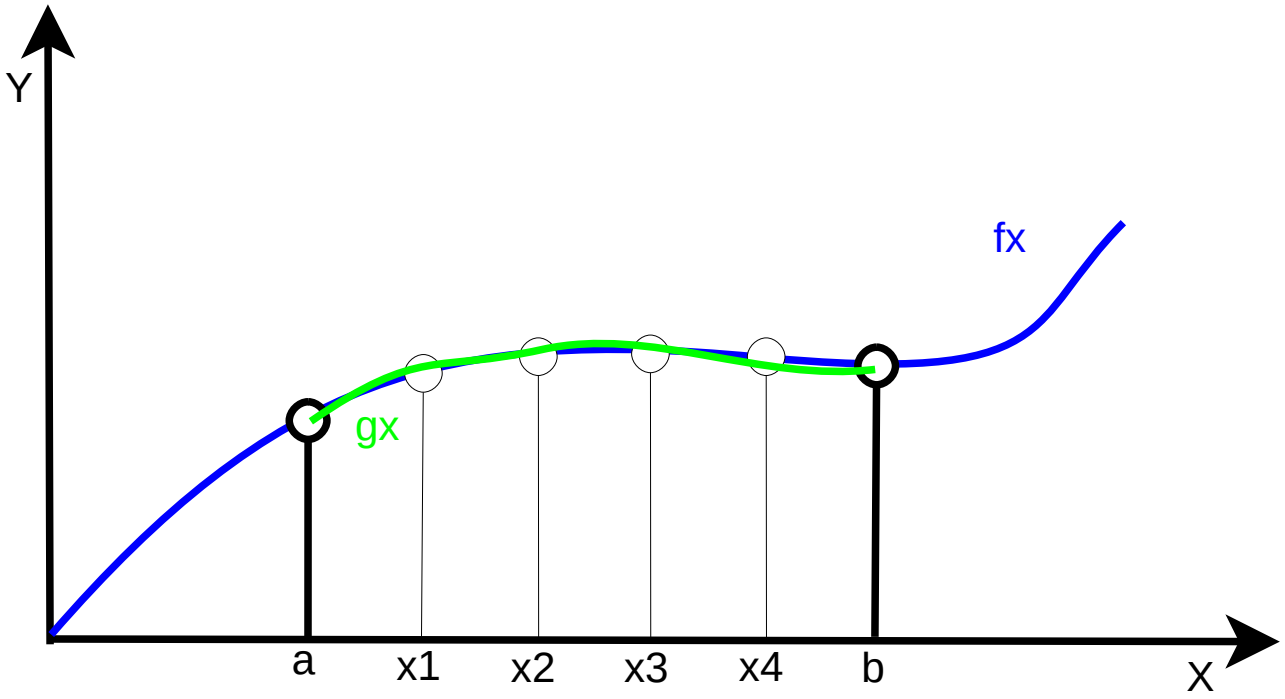

Figura 14: Regla de Quinto Orden

Para la obtención de la Regla Simple utilizamos el **Polinomio interpolador de Lagrange** obteniendo:

$$\int_a^b f(x)\,dx \approx \frac{2}{45}h(19f(x_0)+75f(x_1)+50f(x_2)+50f(x_3)+75f(x_4)+19f(x_5)). \qquad \text{(-104)}$$

2.5.2. Error de Regla Simple basada en 5 intervalos

Para el error seguimos aplicando el método del **Núcleo de Peano**. Consideramos en este caso los nodos $(a,\ (4a+b)/5,\ (3a+2b)/5,\ (2a+3b)/5,\ (a+4b)/5, b)$.

Consideramos el valor de $k=5$ por lo que la fórmula de arranque tendrá la forma siguiente:

$$L(x-\theta)_+^5 = \int_a^b (x-\theta)_+^5\ d\theta - \frac{5h}{288}[19(a-\theta)_+^5 + 75\left(\left(\frac{4a+b}{5}-\theta\right)_+^5\right) + 50\left(\left(\frac{3a+2b}{5}-\theta\right)_+^5\right) + 50\left(\left(\frac{2a+3b}{5}-\theta\right)_+^5\right) + 75\left(\left(\frac{a+4b}{5}-\theta\right)_+^5\right) + (b-\theta)_+^5]. \quad \text{(-105)}$$

Procediendo de igual forma a las Reglas obtenidas anteriormente, se llega a:

$$E_h(f) = -\frac{275}{12096}h^7 f^{VI}(\mu), \quad \text{(-105)}$$

Con $\mu\ \in\ (a,b)$

Que es un error de grado de exactitud 5y de orden de aproximación local 7 al igual que en el *Método de Boole*.

2.5.3. Regla Compuesta basada en 5 intervalos

Realizamos el sumatorio de todas las reglas simples necesarias para abarcar el intervalo de puntos analizado.

$$\int_a^b f(x)\ dx \approx \frac{5}{288}h(19f(x_0)+75f(x_1)+50f(x_2)+50f(x_3)+75f(x_4)+19f(x_5))+ \frac{5}{288}h(19f(x_5)+75f(x_6)+50f(x_7)+50f(x_8)+75f(x_9)+19f(x_{10}))+...$$

$$...+\frac{5}{288}h(19f(x_{5n-5})+75f(x_{5n-4})+50f(x_{5n-3})+50f(x_{5n-2})+75f(x_{5n-1})+19f(x_{5n})). \quad \text{(-107)}$$

Simplificando:

$$\frac{5}{288}h(19E+75M_1+50M_2+38J), \quad \text{(-107)}$$

siendo E (Extremos de intervalo), J (Extremos acumulables), M_1 (valores de los puntos centrales exteriores) y M_2 (valores de los puntos centrales interiores).

2.5.4. Programa de Matlab [Ncotes5(f,M,a,b)]

Su programación en *Matlab* se representa de la siguiente forma:

```
function [int]=Ncotes5(f,n,a,b)

% Aplicamos la regla Ncotes5 compuesta para hacer integración numérica.
% La regla simple está basada en 6 puntos
% [int]=Ncotes5(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas la regla simple.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Ncotes5('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(5*n);

% Inicialización de variables
```

```
E=0; % extremos
J=0; % donde se junta un extremo interior con otro, son múltiplos de 5
M1=0; % los de índice 1 y 4 módulo 5
M2=0; % los de índice 2 y 3 módulo 5


% Cálculo de los nodos
x(1)=a;
x(5*n+1)=b;

for i=2:5*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Quinto Orden

% Denotaremos por E a la suma de los valores de los extremos
E=double(subs(f,x(1)))+double(subs(f,x(5*n+1)));


% Denotaremos por J a la suma de las ordenadas en los extremos intermedios
for i=6:5:5*n-4
    J=J+double(subs(f,x(i)));
end

% Denotaremos por M1 a la suma de las ordenadas intermedias
% para los índices 1 y 4 módulo 5
for i=2:5:5*n-3
    M1=M1+double(subs(f,x(i)));
    M1=M1+double(subs(f,x(i+3)));
end

% Denotaremos por M2 a la suma de las ordenadas intermedias
% para los índices 2 y 3 módulo 5
for i=3:5:5*n-2
    M2=M2+double(subs(f,x(i)));
    M2=M2+double(subs(f,x(i+1)));
end
```

```
% Calculando el valor de la aproximación
int=(5*h/288)*(19*E+75*M1+50*M2+38*J);
```

Final del programa de Matlab

2.5.5. Error de la Regla Compuesta basada en 5 intervalos

El sumatorio de los errores simples cometidos en cada uno de los distintos intervalos nos dará:

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{5(i-1)}}^{x_{5i}} (f(x) - p_{5,i}(x))\ dx,$$

$$E_h(f) = \sum_{i=1}^{n} -\frac{275}{12096} h^7 f^{VI}(\mu_i) = -\frac{275h^7}{12096} \left(\frac{\sum_{i=1}^{n} f^{VI}(\mu_i)}{n} \right) n$$

Sustituyendo $h = \frac{b-a}{5n}$:

$$E_h(f) = -\frac{55(b-a)}{12096} h^6 f^{VI}(\mu), \tag{-109}$$

Con $\mu \in (a, b)$.

Siendo este un error de grado de exactitud 5 y orden de aproximación global 6.

2.6. REGLA DE NEWTON COTES BASADA EN 6 INTERVALOS

2.6.1. Regla Simple basada en 6 intervalos

En el caso de la *Regla de Newton Cotes basada en 6 intervalos* se utilizan 7 puntos para construir el polinomio que aproxima la función, la aproximación a la integral se obtiene integrando el polinomio en vez de la función, esta situación queda reflejada en la Figura 15:

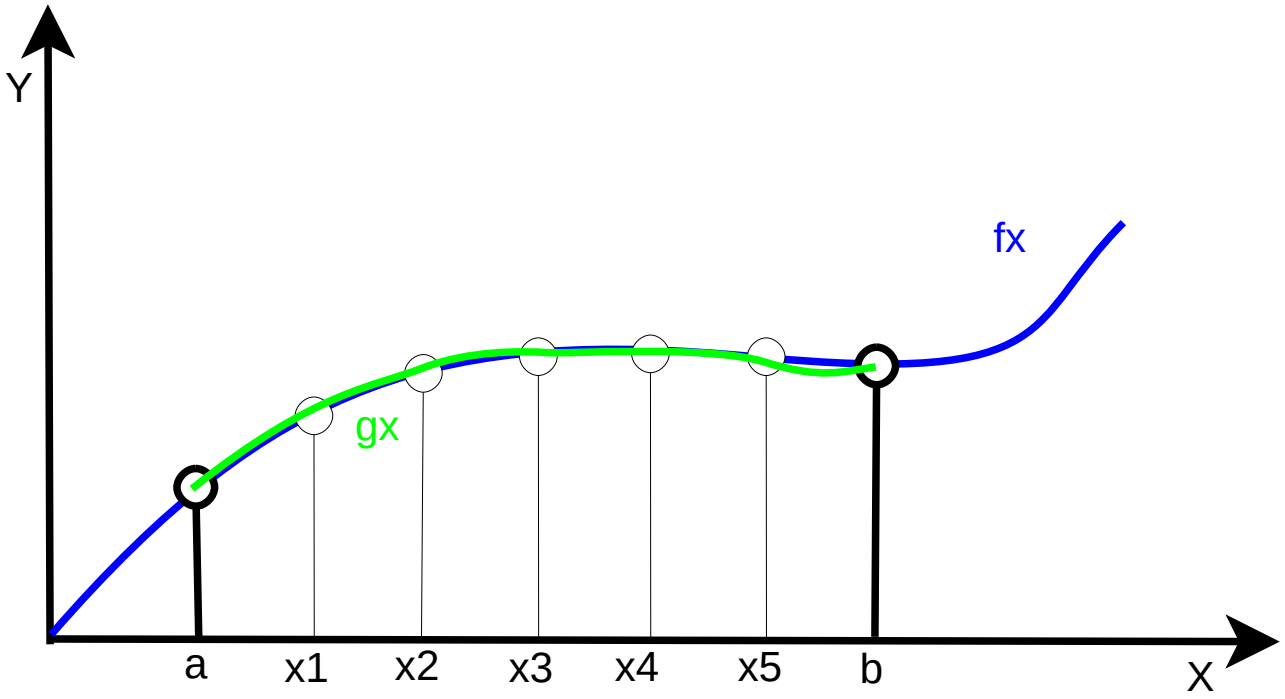

Figura 15: Regla basada en 7 puntos

Este será el último método que analizaremos en detalle debido a que en las reglas sucesivas se repite siempre el mismo procedimiento cambiando únicamente el número de puntos.

Para la obtención de la Regla utilizamos de nuevo el **Polinomio interpolador de Lagrange** obteniendo de ese modo después de integrar la expresión:

$$\int_a^b f(x)\ dx \approx \frac{2}{45}h(41f(x_0)+216f(x_1)+27f(x_2)+272f(x_3)+27f(x_4)+216f(x_5)+41f(x_6)). \tag{-109}$$

2.6.2. Error de la Regla Simple basada en intervalos

Aplicamos otra vez el método basado en el **Núcleo de Peano**.

En este caso los subintervalos en que se divide el intervalo [a,b] quedan definidos por los nodos: $(a, (5a+b)/6, (4a+2b)/6, (a+b)/2, (2a+4b)/6, (a+5b)/6, b)$.

Corresponde ahora el valor de $k=6$ por lo que la fórmula inicial tendrá la forma siguiente:

$$L(x-\theta)_+^6 = \int_a^b (x-\theta)_+^6 \cdot d\theta - \frac{h}{140}[41(a-\theta)_+^6 + 216\left(\left(\frac{5a+b}{6}-\theta\right)_+^6\right) + 27\left(\left(\frac{4a+2b}{6}-\theta\right)_+^6\right) +$$
$$272\left(\left(\frac{a+b}{2}-\theta\right)_+^6\right) + 27\left(\left(\frac{2a+4b}{6}-\theta\right)_+^6\right) + 216\left(\left(\frac{a+5b}{6}-\theta\right)_+^6\right) + (b-\theta)_+^6].$$

Al final el valor del error para la *Regla Simple basada en 6 intervalos* es:

$$E_h(f) = -\frac{9}{1400}h^9 f^{VIII}(\mu), \tag{-111}$$

Con $\mu \in (a,b)$

Siendo éste un error con grado de exactitud 7 y orden de aproximación local 9.

2.6.3. Regla Compuesta basada en 6 intervalos

Realizamos el sumatorio de todas las reglas Simples:

$$\int_a^b f(x)\ dx \approx \frac{h}{140}(41f(x_0)+216f(x_1)+27f(x_2)+272f(x_3)+27f(x_4)+216f(x_5)+41f(x_6))+$$

$$\frac{h}{140}(41f(x_6)+216f(x_7)+27f(x_8)+272f(x_9)+27f(x_{10})+216f(x_{11})+41f(x_{12}))+...$$

$$...+\frac{h}{140}(41f(x_{6n-6})+216f(x_{6n-5})+27f(x_{6n-4})+272f(x_{6n-3})+27f(x_{6n-2})+216f(x_{6n-1})+41f(x_{6n})). \tag{-113}$$

Simplificando:

$$\frac{h}{140}(41E+216M_1+27M_2+272C+82J), \tag{-113}$$

siendo E (Extremos del intervalos), J (Extremos acumulables), M_1 (valores de los puntos centrales exteriores), M_2 (valores de los puntos centrales interiores) y C (valor del punto central de cada intervalo).

2.6.4. Programa de Matlab [Ncotes6(f,M,a,b)]

Su programación en *Matlab* se representa de la siguiente forma:

```
function [int]=Ncotes6(f,n,a,b)

% Aplicamos la regla de Ncotes6 para hacer integración numérica.
% [int]=Ncotes6(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Ncotes6.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Ncotes6('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
```

```
h=(b-a)/(6*n);

% Inicialización de variables
E=0; % Extremos
J=0; % Donde se juntas los extremos, multiplos de 6
M1=0; % Indices 1 y 5 módulo 6
M2=0; % Indices 2 y 4 módulo 6
C=0; % Valor central

% Cálculo de los nodos
x(1)=a;
x(6*n+1)=b;

for i=2:6*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Ncotes6;

% Denotaremos por E a la suma de los valores de los extremos
E=double(subs(f,x(1)))+double(subs(f,x(6*n+1)));

% Denotaremos por J a la suma de las ordenadas extremos acumulables en el método
for i=7:6:6*n-5
    J=J+double(subs(f,x(i)));
end

% Denotaremos por M1 a la suma de las ordenadas intermedias de indices 1 y 5 con modulo 6 en el método
for i=2:6:6*n-4
    M1=M1+double(subs(f,x(i)));
    M1=M1+double(subs(f,x(i+4)));
end

% Denotaremos por M2 a la suma de las ordenadas intermedias extremos de indices 2 y 4 con modulo 6 en el método
for i=3:6:6*n-3
    M2=M2+double(subs(f,x(i)));
    M2=M2+double(subs(f,x(i+2)));
end
```

```
% Denotaremos por C a la suma de las ordenadas centrales en el método
for i=4:6:6*n-2
    C=C+double(subs(f,x(i)));
end

% Calculando el valor de la aproximación
int=(h/140)*(41*E+216*M1+27*M2+272*C+82*J);
```

Final del programa de Matlab

2.6.5. Error Regla Compuesta basada en 6 intervalos

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{2(i-1)}}^{x_{6i}} (f(x) - p_{6,i}(x))\, dx,$$

$$E_h(f) = \sum_{i=1}^{n} -\frac{h^9}{140} f^{VIII}(\mu_i) = -\frac{h^9}{140} \left(\frac{\sum_{i=1}^{n} f^{VIII}(\mu_i)}{n} \right) n.$$

Sustituyendo $h = \frac{b-a}{6n}$:

$$E = -\frac{(b-a)}{840} h^8 f^{VIII}(\mu) \tag{-115}$$

con $\mu \in (a, b)$

siendo este un error de grado de exactitud 7 y orden de aproximación global 8.

2.7. REGLAS DE ORDEN SUPERIOR

Existen, a parte de las analizadas con anterioridad, más Reglas conforme vayamos aumentado el número de puntos que utilizamos, pero que debido a que su obtención es prácticamente similar a lo visto anteriormente, no es necesario su desarrollo. Únicamente mencionaremos algunas Reglas Simples más a continuación junto con sus errores asociados.

-Regla de 7 intervalos

$$\int_a^b f(x)\ dx \approx \frac{7}{17280}h(751f(x_0)+3577f(x_1)+1323f(x_2)+2989f(x_3) \\ +2989f(x_4)+1323f(x_5)+3577f(x_6)+751f(x_7)), \tag{-116}$$

y su error asociado a la Regla Simple:

$$E_h(f) = -\frac{8183}{518400}h^9 f^{VIII}(\mu). \tag{-116}$$

-Regla de 8 intervalos

$$\int_a^b f(x)\ dx \approx \frac{4}{14175}h(989f(x_0)+5888f(x_1)-928f(x_2)+10496f(x_3)-4540f(x_4)+10496f(x_5) \\ -928f(x_6)+5888f(x_7)+989f(x_8)), \tag{-117}$$

y su error asociado a la Regla Simple:

$$E_h(f) = -\frac{2368}{467775}h^{11} f^{X}(\mu). \tag{-117}$$

-Regla de 9 intervalos

$$\int_a^b f(x)\,dx \approx \frac{9}{89600}h(2857f(x_0)+15741f(x_1)+1080f(x_2)+19344f(x_3)+5778f(x_4)+5778f(x_5))$$
$$+19344f(x_6)+1080f(x_7)+15741f(x_8)+2857f(x_9), \qquad \text{(-118)}$$

y su error asociado a la Regla Simple:

$$E_h(f) = -\frac{173}{14620}h^{11}f^X(\mu). \qquad \text{(-118)}$$

Esto continuaría a medida que vayamos cogiendo más puntos para el intervalo, pero debido a su tamaño son fórmulas de difícil utilización con apenas aplicación en el entorno de la ingeniería naval.

2.8. REGLA DE NEWTON COTES CERRADA GENERAL

Una vez vistas cada una de las reglas, hasta la regla basada en 6 intervalos, utilizadas para la aproximación de integrales nos planteamos la posibilidad de establecer un único programa que sea capaz de, dependiendo de nuestras preferencias, resolver nuestra integral mediante una regla u otra sin tener que estar utilizando cada programa de forma individual.

A este programa le hemos denominado NcotesN y se basa en la creación de una serie de constantes llamadas **Pesos** que variarán dependiendo de intervalos N que nosotros introduzcamos en el programa, estableciendo así el orden de la Regla. De este modo con un único programa nos podremos aproximar más a las necesidades requeridas por la función a aproximar dependiendo de sus características.

Al vector de pesos se le denomina mascara y dicha mascara se obtiene de la siguiente forma:

$$\int_a^b f(x)\ dx \approx \int_a^b p_N(x)\ dx = \sum_{i=0}^{N} \left(\int_a^b L_i(x)\ dx \right) f(x_i). \tag{-118}$$

2.8.1. PESOS

Para la obtención de la regla general primero debemos obtener lo que se denomina *función de pesos*, que no es más que las constantes que multiplican a los valores de la funcion en los nodos, tal y como aparecen en la fórmula 2.8.

La función pesos calula los pesos w_i resolviendo las siguientes integrales:

$$w_i = \int_a^b L_i(x)dx. \tag{-118}$$

Su programación en *Matlab* se representa de la siguiente forma:

```
function w = Pesos_NCotes(nod)

% This function computes the weights corresponding to Newton Cotes
% algorithm for numerical integration in uniform 1D meshes
    %
    % w = Pesos_NCotes(nod)
    %
    % Input variables:
    % nod vector with the nodes
    %
    % Output variables
    % w vector containing the weights for the simple integration rule
        %
        % Example:
        % w=Pesos_NCotes(1:5)


        % We declare x as a simbolic variable to carry out simbolic integrals
        syms x

        % Number of nodes
        n = length(nod);

        % Interval of integration
        a=nod(1);
        b=nod(n);

        % Calculus of the Lagrange polynomials

        for i = 1 : n
            L{i}=1;
            for j = 1 : n
                if(j  = i)
                L{i} = L{i}*(x - nod(j))/ (nod(i) - nod(j));
            end
        end
    end

    % Calculus of the exact integral for each Lagrange polynomial
```

```
fprintf('\n');

for i=1:n

    w{i}=int(L{i},a,b);
    fprintf('%s ',char(w{i}));

end
fprintf('\n\n');
```

Final del programa de Matlab

Por lo que la aproximación a la integral quedaría de la siguiente forma:

$$\int_a^b f(x)\ dx \approx \sum_{i=0}^{N} w_i f(x_i). \tag{-118}$$

Para este cálculo necesitamos las expresiones de los **Polinomio de Lagrange** $L_i(x)$:

$$L_i(x) = \prod_{\substack{j=0:N \\ j\neq i}} \frac{x - x_j}{x_i - x_j}. \tag{-118}$$

2.8.2. Programación del Método NcotesN en Matlab

Para su programación utilizamos un esquema similar al visto anteriormente en cada una de las Reglas y al cual añadimos una llamada a la función pesos, establecida anteriormente que nos dará las constantes necesarias para la aplicación del método y en la que debemos introducir además de los datos habituales (Función (f), Extremos (a,b)), el número de subintervalos N que vamos a utilizar en la Regla Simple y llamamos n al número de veces que aplicamos la Regla Simple para formar la Regla Compuesta.

Damos directamente la programación de la *Regla de Newton Cotes Compuesta.*

En la Figura 16 mostramos los últimos N nodos correspondientes a la aplicación de la última regla Simple en el extremo derecho.

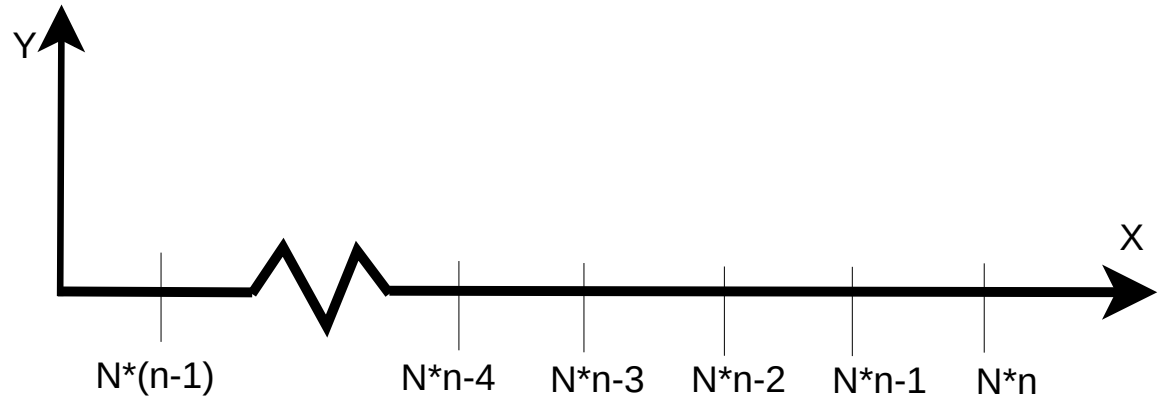

Figura 16: División del dominio de una función en puntos equidistantes en el extremo derecho

Obtenemos el sumatorio de los valores de la función en los extremos intervalo [a,b] en el elemento E mientras que el elemento J representa el sumatorio de los valores de la función en los extremos intermedios. El elemento M representa el sumatorio de los valores de la función en los puntos intermedios, que dependen del valor que hayamos dado a N. Mediante un bucle calculamos su producto por el valor del programa *Pesos* correspondiente.

Para ahorrar cálculos a *Matlab* hemos decidido, debido a la simetría que existe en todas las reglas, únicamente calcular la mitad de los puntos teniendo en cuenta que dependiendo si el número de puntos utilizado para la obtención de la regla es par o impar debemos calcular:

$$Si\ n\ es\ par \longrightarrow w_0, w_1 ... w_{\frac{n}{2}},$$
$$Si\ n\ es\ impar \longrightarrow w_0, w_1 ... w_{\frac{n+1}{2}}.$$

Nota: Hay que tener en cuenta que en Matlab *al no existir el valor 0 que usamos en estos métodos numéricos por lo que lo que nosotros consideramos par, en lenguaje de* Matlab *será considerado impar y por tanto debemos aplicar las condiciones anteriormente mencionadas de forma contraria.*

2.8.3. Programa de Matlab [NcotesN(f,M,a,b,N,n)]

Su programación en *Matlab* se representa de la siguiente forma:

```
function [int]=Ncotes(f,N,n,a,b)

% Aplicamos la regla de Ncotes compuesta para hacer integración numérica.
% La regla simple está basada en N+1 puntos
% [int]=NcotesN(f,N,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% N indica que hay N+1 nodos contando los extremos a,b
% n número de veces que aplicas la regla simple.
% a,b extremos del intervalo
% Datos de salida
% int valor de la aproximación a la integral
%
% Ejemplo:
% [int]=NcotesN('cos(2*x+1)',5,100,0,0.25)


% Generamos el vector de pesos

w=Pesos_NCotes(1:N+1);

% Calculamos la longitud del intervalo
h=(b-a)/(N*n);


% Inicialización de variables

E=0; % extremos
J=0; % donde se junta un extremo interior con otro

% Cálculo de los nodos
x(1)=a;
x(N*n+1)=b;

for i=2:N*n
    x(i)=x(1)+(i-1)*h;
end
```

```
% Regla de Ncotes con N+1 nodos

% Denotaremos por E a la suma de los valores de los extremos
E=double(subs(f,x(1)))+double(subs(f,x(N*n+1)));


% Denotaremos por J a la suma de las ordenadas en los extremos intermedios
for i=N+1:N:N*n-N+1
    J=J+double(subs(f,x(i)));
end


if mod(N,2)  %N es impar

    for j=1:(N-1)/2

        M(j)=0;


        % Denotaremos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*n+1-N+j
            M(j)=M(j)+double(subs(f,x(i+1)));
            M(j)=M(j)+double(subs(f,x(i+N+1-2*j)));
        end

    end

else  % N es par

    for j=1:N/2-1

        M(j)=0;

        % Denotamos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*n+1-N+j
```

```
            M(j)=M(j)+double(subs(f,x(i+1)));
            M(j)=M(j)+double(subs(f,x(i+N+1-2*j)));
        end
    end

    % Ahora sumamos la contribución del punto central del intervalo

    M(N/2)=0;
    for i=N/2+1:N:n*N+1-N/2
        M(N/2)=M(N/2)+double(subs(f,x(i)));
    end

end

M

% Calculando el valor de la aproximación

int=double(w{1}*E+2*w{1}*J);

if mod(N,2) %N es impar
    for j=1:(N-1)/2
        int=int+w{j+1}*M(j);
    end
else %N es par
    for j=1: N/2
        int=int+w{j+1}*M(j);
    end

end

format long
int=double(h*int);
```

Final del programa de Matlab

2.8.4. Error asociado a Newton Cotes General

El error cometido mediante este método dependerá del numero de intervalos escogido N y del número de veces que lo apliquemos en la Regla Compuesta n para la resolución de la integral de la función. Éste error, como hemos explicado anteriormente, irá disminuyendo tanto con n como con N, por lo que nos interesará utilizar el método de mayor numero de intervalos siempre que sea posible a la hora de resolver problemas para conseguir la máxima precisión.

Obtenemos los errores mediante la aplicación del método del **Teorema de Peano** que hemos explicado anteriormente en el punto 1.2.3.

2.8.5. Programa de Matlab [ErrorNcotesN(f,M,a,b)]

La programación del error asociado a la Regla obtenida con el programa *NcotesN* será:

Su programación en *Matlab* se representa de la siguiente forma:

```
function [Error_N,coef,Error_N_Compuesta,coef_Compuesta]=Error_NcotesN(N)

% Calculamos el error en la regla de Ncotes para hacer integración numérica.
% La regla simple está basada en N+1 puntos
% [Error_N,coef,Error_N_Compuesta,coef_Compuesta]=Error_NcotesN(N)
% Datos de entrada
% N indica que hay N+1 nodos contando los extremos a,b
% Datos de salida
% Error_N error teórico cometido en la fórmula de integración simple
% coef coeficiente numérico de la fórmula del error
% Error_N_Compuesta error teórico cometido en la fórmula de integración
% compuesta
% coef_Compuesta coeficiente numérico de la fórmula del error para la regla compuesta
%
% Ejemplo:
% [Error_N,coef,Error_N_Compuesta,coef_Compuesta]=Error_NcotesN(5)

% Declaramos la variables simbólicas
syms a b theta
```

```
% Generamos el vector de pesos
w=Pesos_NCotes(1:N+1);

if mod(N,2)  %N es impar
    s=N;
else
    s=N+1;
end

% Cálculamos la integral entre a y b del núcleo de peano K(theta)
La=1/(s+1)*1/(s+2)*(b-a)^(s+2);
for j=N+1:-1:2
    La=La-(b-a)/N*w{j}*int((((N+1-j)*a+(j-1)*b)/N-theta)^s,theta,a,((N+1-j)*a+(j-1)*b)/N);
end
coef=simplify(1/factorial(s)*La/(b-a)^(s+2)*N^(s+2));
Error_N=[char(coef),blanks(2),'f^(',num2str(s+1),')(mu)',blanks(2),'h^(',num2str(s+2),')'];
coef_Compuesta=factor(coef*(b-a)/N);
Error_N_Compuesta=[char(coef_Compuesta),blanks(2),'f^(',num2str(s+1),')(mu)',blanks(2),'h^(',num2
```

Final del programa de Matlab

2.9. Aplicación Naval del Método de Newton Cotes Cerrado

De entre las numerosas aplicaciones que tienen los métodos numéricos para la resolución de integrales relacionadas con el ámbito naval, debemos destacar principalmente aquellos que tienen que ver con el campo de la Hidrostática o Estabilidad del buque y con las Estructuras navales [2].

En la Figura 17 vemos representados algunos de los elementos hidrostáticos que podriamos calcular como el centro de gravedad, centro de carena...

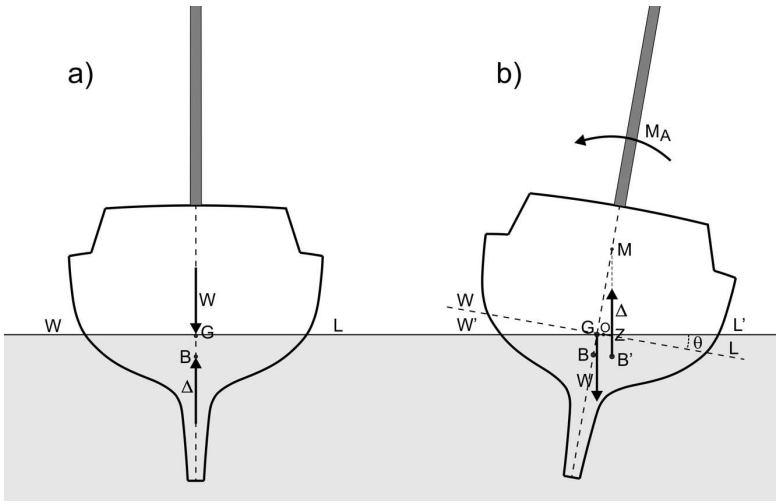

Figura 17: Fuerzas que intervienen en la hidrostática de un buque

En primer lugar podemos demostrar como se realizará el cálculo del volumen de agua desplazada por un buque a partir de los datos obtenidos de la cartilla de trazado del buque, es decir del valor de los puntos, no de la función.

Nota: *Esto nos obligará a realizar una modificación en el programa* NcotesN *que nos permita la introducción de los datos que se explicará mas adelante en el apartado 6 de interfaz gráfica.*

El cálculo del volumen de carena del buque viene dada por los siguientes datos: $L_{pp} = 120m$, $B = 22,5m$ y $D = 7,2m$ teniendo ademas los valores de la curva de áreas de secciones mostradas en la Figura 18:

Sec.	0	1	2	3	4	5	6	7	8
Área(m^2)	6,86	30,32	56,50	79,38	98,87	117,674	132,73	142,99	149,86

9	10	11	12	13	14	15	16	17	18
152,25	153,70	152,38	145,50	135,66	119,67	97,47	74,47	53,51	35,61

19	20
22,69	13,28

Figura 18: Tabla de datos de áreas de secciones

A partir de esta tabla de datos calculamos mediante nuestro programa de integración la integral correspondiente:

$$\nabla = \int_L A_s \, dx. \tag{-120}$$

Introducimos los datos en el programa siendo el método escogido el de 5 puntos y como intervalo cogemos la eslora del buque: $L = 120$, como vemos en la Figura 19

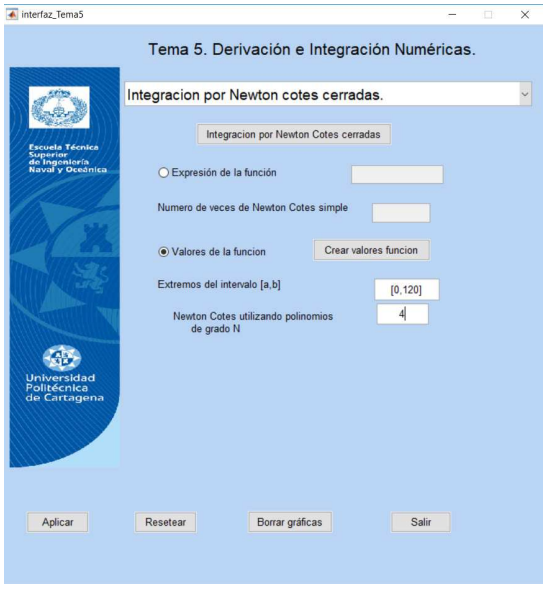

Figura 19: Muestra del programa

Y el resultado nos aparecerá en pantalla de la forma que se indica en la Figura 20:

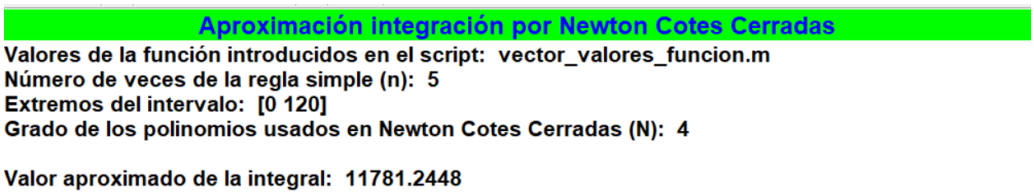

Figura 20: Muestra del resultado dado por el programa

Por lo que el valor del volumen de carena será $\nabla = 11781{,}2448 m^2$.

Otro ejemplo que vamos a mostrar es como calcular la curva de cargas sobre el casco de un buque. En este caso se nos suelen dar una serie de puntos en el que cada cual representa un dato de carga que es el que utilizaremos para la creación de la función a integrar y procederemos al cálculo de su área mediante la aproximación de la integral de dicha función.

En este caso hemos dividido la eslora de un buque de 203,5m en 20 secciones y en cada una de ellas hemos calculado el valor del carga total mediante la fórmula:

$$Carga \ = \ Peso - Empuje,$$

ya que usualmente son conocidos los datos de la curva de pesos y empujes.

De este modo obtenemos la lista de datos que mostramos a continuación en la Figura 21:

Secciones	Carga t
0-1	369,942
1-2	407,314
2-3	393,158
3-4	359,319
4-5	47,665
5-6	292,529
6-7	562,466
7-8	-749,389
8-9	-749,674
9-10	-749,218
10-11	-748,761
11-12	-748,305
12-13	-83,487
13-14	-83,031
14-15	-82,198
15-16	-111,595
16-17	-106,383
17-18	-50,625
18-19	799,479
19-20	1132,454

Figura 21: Tabla de datos de cargas de un buque

Una vez que tenemos estos datos, los introducimos en un archivo *.m* y usamos el programa de *Newton Cotes Cerradas* junto con la opción de introducir los valores de la función Además debido a que estamos trabajando con este programa podemos introducir el valor N y elegir de este modo el método.

Esto nos da gran versatilidad ya que no nos restringe la cantidad de intervalos utilizados para el cálculo de la integral y en nuestro caso usaremos un valor $N = 10$ ya que queremos un ajuste muy preciso.

Dándonos el resultado que vemos en la Figura 22:

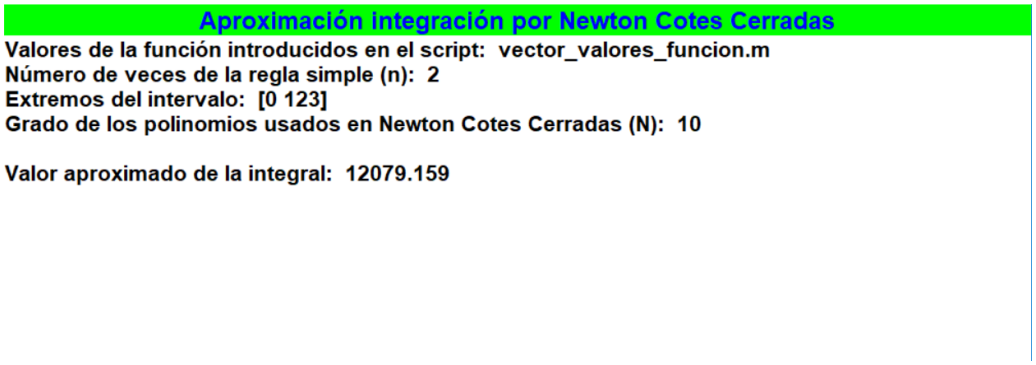

Figura 22: Resolución de cargas de un buque con el Programa

Siendo éste valor el área encerrada por la curva de cargas y por tanto el momento cortante máximo experimentado por el buque $Q(x) = 12079,159Nm$.

3. REGLAS DE NEWTON COTES ABIERTAS

A la hora de analizar una función obtenida de cualquier problema podemos encontrarnos con el hecho de que se produzca una asíntota como la que aparece en la Figura 23 u otro fenómeno que nos impida un análisis cerrado de la misma, como puede ser el caso de la aparición de una función hiperbólica dentro de una integral. En estos casos podemos aplicar otros métodos denominados como *Reglas de Newton Cotes Abiertas*.

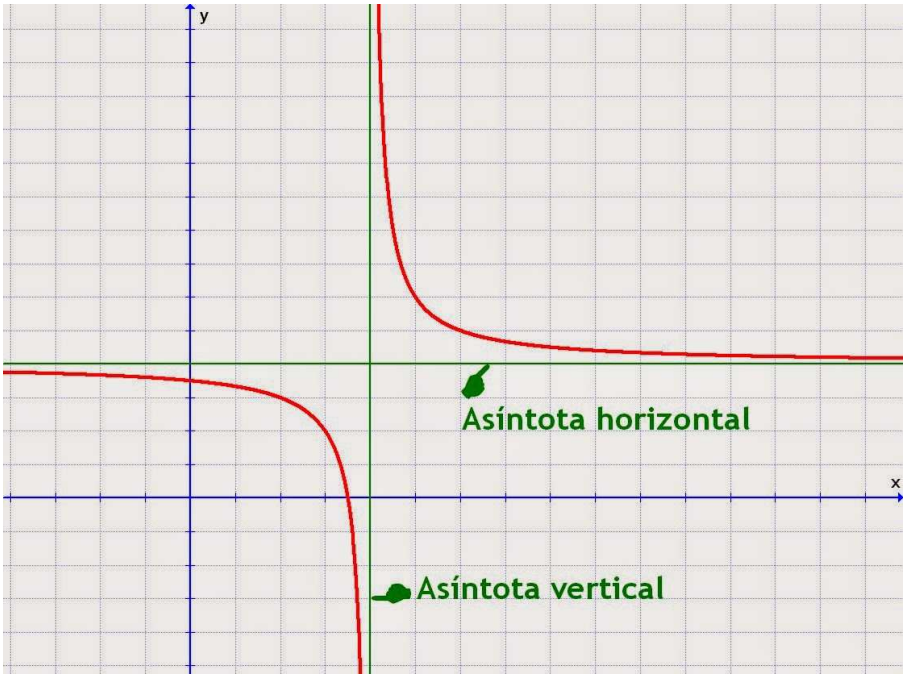

Figura 23: Asintota producida en una función

En estos métodos se realiza una sustitución de la función origen mediante el **Polinomio de Lagrange** igual que en lo Métodos Cerrados con la particularidad de que, aunque dividamos el dominio función en los mismos intervalos que en el caso de *Newton Cotes Cerrado*, sólo consideramos los puntos intermedios, obviando los puntos extremos, consiguiendo de este modo alejarnos de las asíntotas y aproximar a la integral de la función.

Al igual que ocurría en *Newton Cotes Cerrado*, denominamos h a la longitud del intervalo entre dos puntos de la partición considerada del dominio de la función, por lo que será: $h = \frac{b-a}{n}$ donde n es el número de intervalos en los que se divide el dominio de la función.

Como es de suponer, no existe una *Regla de Newton Cotes Abierta* para la *Regla de los Trapecios*, debido a que como se compone de 2 puntos no posee ningún punto intermedio sobre el cual realizar el análisis. Por este motivo la primera Regla a desarrollar será la de Simpson para intervalo abierto.

Como ya hemos dicho uno de los principales usos de estas aproximaciones, sobre todo en el ámbito naval, son aquellas integrales en las que aparecen funciones hiperbólicas como pueden ser aquellas asociadas con ondas u olas en el agua. A continuación se muestra la forma de una función hiperbólica en la Figura 24:

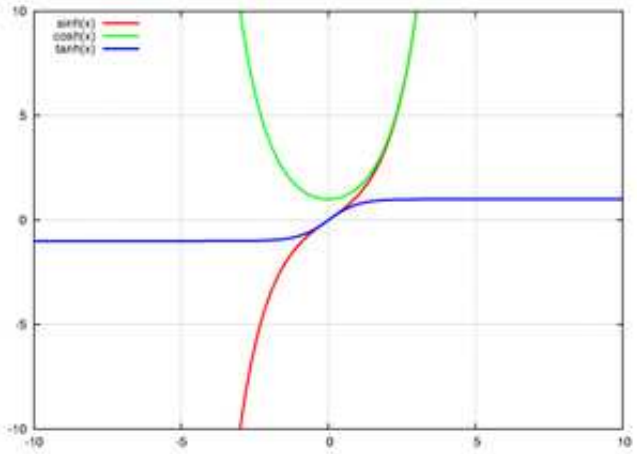

Figura 24: Funciones hiperbólicas

Un ejemplo de la aplicación de la formulas de Newton Cotes abiertas para funciones hiperbólicas se mostrará en el apartado 3.7.

3.1. REGLA DE SIMPSON ABIERTA

3.1.1. Regla de Simpson Simple Abierta

La obtención de la regla simple se realiza como en el caso anterior mediante el **Polinomio de Lagrange** pero esta vez sin embargo utilizando el único punto central, simplificándose mucho los cálculos. Al final nos queda la siguiente expresión:

$$\int_a^b f(x)\ dx \approx 2hf(x_1). \tag{-121}$$

Como se observa es mucho más simple que la cerrada, pero esto también crea una serie de inconvenientes que mostraremos a continuación.

3.1.2. Error de la Regla de Simpson Simple

Se puede demostrar que la fórmula del error resulta:

$$E_h(f) = \frac{h^3}{3} f''(\mu) \tag{-121}$$

Con $\mu \in (a, b)$

A continuación comparamos las fórmulas de el error asociado a la *Regla de Simpson Cerrada Simple* y la *Regla de Simpson Abierta Simple*:

Error en Simpson abierto:

$$E_h(f) = \frac{f''(\mu)h^3}{3}.$$

Error en simpson cerrado:

$$E_h(f) = \frac{f^{IV}(\mu)h^5}{90}.$$

Lógicamente no se aplican a una misma función ya que cada regla cumple su propósito determinado. Sin embargo observando simplemente las ecuaciones descritas del error en ambos métodos, podemos observar que en el caso de la *Regla de Newton Cotes Cerrada* tiene mayor orden de exactitud y orden de aproximación.

3.1.3. Regla de Simpson Compuesta Abierta

Para la Regla compuesta aplicamos la regla simple tantas veces sea necesaria. En este caso el valor del termino h debe ser: $h = \frac{b-a}{n}$

La aproximación a la integral queda:

$$\int_a^b f(x)\ dx \approx 2h(f(x_1) + f(x_3) + f(x_5)...f(x_{2n-1})). \tag{-123}$$

Englobamos para hacer más sencillo el cálculo en *Matlab* por lo que nos quedaría finalmente:

$$label Regla Simpson Abierta Compuesta \int_a^b f(x)\ dx \approx 2h(I_i), \tag{-123}$$

siendo I_i el sumatorio de los valores de la función en los puntos intermedios.

Su programación en *Matlab* será la siguiente:

3.1.4. Programa de Matlab [Simpson(f,N,a,b)]

```
function [int]=simpson(f,n,a,b)

% Aplicamos la regla de Simpson Abierta para hacer integración numérica.
% [int]=simpson(f,n,a,b)
% Datos de entrada
```

```
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Simpson Simple Abierta.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=simpson('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(2*n);

% Inicialización de variables
Ii=0;

% Cálculo de los nodos
x(1)=a;
x(2*n+1)=b;

for i=2:2*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Simpson Abierta


% Denotaremos por Ii a la suma de las ordenadas intermedias impares (pares en Matlab)
en el método
for i=2:2:2*n
    Ii=Ii+subs(f,x(i));
end

format long
% Calculando el valor de la aproximación
int=double(2*h*Ii);
```

Final del programa de Matlab

3.1.5. Error de la Regla Simpson Compuesta Abierta

El error de la Regla Compuesta será la suma del error cometido en cada intervalo, por lo que nos quedará:

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{2(i-1)}}^{x_{2i}} (f(x) - p_{0,1}(x))\ dx,$$

$$E_h(f) = \sum_{i=1}^{n} \frac{h^3}{3} f''(\mu_i) = -\frac{h^3}{3} \left(\frac{\sum_{i=1}^{n} f^{II}(\mu_i)}{n} \right) n.$$

Sustituimos uno de los valores h, $h = \frac{b-a}{2n}$:

$$E_h(f) = -\frac{(b-a)}{6} h^2 f''(\mu) \tag{-125}$$

con $\mu \in (a, b)$

Por lo que en el caso de la Regla de Simpson compuesto tendremos un grado de exactitud 1 con un orden de aproximación 2.

3.2. REGLA DE SIMPSON 3/8 ABIERTA

3.2.1. Regla de Simpson 3/8 Simple Abierta

Realizamos un desarrollo similar al resto de métodos presentados se llega a:

$$\int_a^b f(x)\ dx \approx \frac{3h}{2}(f(x_1) + f(x_2)). \tag{-125}$$

3.2.2. Error de la Regla de Simpson 3/8 Simple Abierta

La fórmula del error vendrá dada por la expresión:

$$E_h(f) = \frac{h^3}{4} f''(\mu), \tag{-125}$$

con $\mu \in (a, b)$

siendo este un error de grado de exactitud 1 y orden de aproximación 3.

3.2.3. Regla de Simpson 3/8 Compuesta Abierta

Aplicamos la regla simple tantas veces como sea necesario teniendo en cuenta que el valor del termino$h$ debe ser $h = \frac{b-a}{3n}$.

$$\int_a^b f(x)\ dx \approx \frac{3h}{2}(f(x_1) + f(x_2)) + \frac{3h}{2}(f(x_4) + f(x_5)) + ...\frac{3h}{2}(f(x_{3n-2}) + f(x_{3n-1})).$$

Agrupamos:

$$\int_a^b f(x)\ dx \approx \frac{3h}{2}(I_1 + I_2), \tag{-126}$$

donde I_1 representa el sumatorio de los valores de la función f(x) evaluada en los nodos con índice $(1+3k),\ k\ \in\ \mathbb{N}$ y I_2 el sumatorio de los valores de la función f(x) evaluada en los nodos con índice $(2+3k),\ k\ \in\ \mathbb{N}$.

3.2.4. Programa de Matlab [Simpson3/8(f,N,a,b)]

```
function [int]=simpson3_8(f,n,a,b)

% Aplicamos la regla de Simpson 3/8 abierta para hacer integración numérica.
% [int]=simpson3_8(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Simpson 3/8 simple Abierta.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=simpson3_8('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(3*n);

% Inicialización de variables (Variables no extremos)
I1=0;
I2=0;
I3=0;

% Cálculo de los nodos

x(1)=a;
x(3*n+1)=b;

for i=2:3*n
    x(i)=x(i-1)+h;
end
```

```
% Regla de Simpson 3/8 Abierta

% Denotaremos por I1 a la suma de las ordenadas intermedias múltiplos de 3 más 1

for i=2:3:3*n-1
    I1=I1+subs(f,x(i));
end


% Denotaremos por I2 a la suma de las ordenadas intermedias múltiplos de 3 más 2

for i=3:3:3*n
    I2=I2+subs(f,x(i));
end

format long
% Calculando el valor de la aproximación
int=double((3*h/2)*(I1+I2));
```

Final del programa de Matlab

3.2.5. Error de la Regla Simpson 3/8 Compuesta Abierta

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{3(i-1)}}^{x_{3i}} (f(x) - p_{1,i}(x))\, dx,$$

$$E_h(f) = \sum_{i=1}^{n} \frac{h^3}{4} f''(\mu_i) = \frac{h^3}{4} \left(\frac{\sum_{i=1}^{n} f^{II}(\mu_i)}{n} \right) n.$$

Sustituimos uno de los valores $h = \frac{b-a}{3n}$:

$$E_h(f) = \frac{(b-a)}{12} h^2 f''(\mu), \tag{-128}$$

con $\mu \in (a, b)$.

Por lo que tenemos un grado de exactitud 1 con un Orden de aproximación 2.

3.3. REGLA DE BOOLE ABIERTA

3.3.1. Regla de Boole Simple Abierta

$$\int_a^b f(x)\ dx \approx \frac{4h}{3}2((f(x_1)+f(x_3))-f(x_2)). \tag{-128}$$

3.3.2. Error de la Regla de Boole Simple Abierta

$$E_h(f)=\frac{28}{90}h^5 f^{IV}(\mu), \tag{-128}$$

con $\mu \in (a,b)$.

Siendo éste un error con grado de exactitud 3 y de orden de aproximación local 5

3.3.3. Regla de Boole Compuesta Abierta

$$\int_a^b f(x)\ dx = \frac{4h}{3}2((f(x_1)+f(x_3))-f(x_2)) + \frac{4h}{3}2((f(x_5)+f(x_7))-f(x_6)) + ...$$

$$\frac{4h}{3}2((f(x_{4n-3})+f(x_{4n-1}))-f(x_{4n-2})).$$

Agrupamos los términos para simplificar las operaciones en *Matlab*:

$$\frac{4h}{3}(2I_i - I_p), \tag{-130}$$

donde I_i corresponde a la suma de las ordenadas intermedias impares en el método y I_p corresponde a la suma de las ordenadas intermedias pares.

Su programación con *Matlab* será la siguiente.

3.3.4. Programa de Matlab [Boole(f,N,a,b)]

```
function [int]=Boole(f,n,a,b)

% Aplicamos la regla de Boole Abierta para hacer integración numérica.
% [int]=Boole(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Boole simple Abierta.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Boole('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(4*n);

% Inicialización de variables
Ii=0;
Ip=0;
Ie=0;

% Cálculo de los nodos
x(1)=a;
x(4*n+1)=b;

for i=2:4*n
    x(i)=x(i-1)+h;
end

% Regla de Boole Abierta


% Denotaremos por Ii a la suma de las ordenadas intermedias impares (pares en Matlab)
en el método
for i=2:2:4*n
    Ii=Ii+subs(f,x(i));
```

```
end

% Denotaremos por Ip a la suma de las ordenadas intermedias pares (impares en Matlab) en el método
for i=3:4:4*n-1
    Ip=Ip+subs(f,x(i));
end


format long
% Calculando el valor de la aproximación
int=double((4*h/3)*(2*Ii-Ip));
```

Fin del programa Matlab

3.3.5. Error de la Regla de Boole Compuesta Abierta

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{4(i-1)}}^{x_{2i}} (f(x) - p_{2,i}(x))\ dx,$$

$$E_h(f) = \sum_{i=1}^{n} \frac{28}{90} h^5 f^{IV}(\mu_i) = \frac{28}{90} h^5 \left(\frac{\sum_{i=1}^{n} f^{IV}(\mu_i)}{n} \right) n.$$

Sustituimos uno de los valores $h = \frac{b-a}{4n}$:

$$E = \frac{7(b-a)}{90} h^4 f^{IV}(\mu), \tag{-132}$$

con $\mu \in (a, b)$.

Por lo que en el caso de la Regla de Boole compuesto tendremos un grado de exactitud 3 y un orden de aproximación 4.

3.4. REGLA BASADA EN 5 INTERVALOS ABIERTA

3.4.1. Regla Simple basada en 5 intervalos Abierta

$$\int_a^b f(x)\ dx \approx \frac{5h}{24}(11(f(x_1)+f(x_4))+(f(x_2)+f(x_3)) \tag{-132}$$

3.4.2. Error de la Regla Simple basada en 5 intervalos Abierta

$$E_h(f) = \frac{95}{144}h^5 f^{IV}(\mu), \tag{-132}$$

con $\mu \in (a,b)$.

Siendo este un error de grado de exactitud 3 y orden de aproximación local 5.

3.4.3. Regla Compuesta basada en 5 intervalos Abierta

$$\int_a^b f(x)\ dx \approx \frac{5h}{24}(11(f(x_1)+f(x_4))+(f(x_2)+f(x_3))+\frac{5h}{24}(11(f(x_6)+f(x_9))+(f(x_7)+f(x_8))+...$$

$$\frac{5h}{24}(11(f(x_{5n-4})+f(x_{5n-1}))+(f(x_{5n-3})+f(x_{5n-2})).$$

Agrupamos los términos:

$$\frac{5h}{24}(11M_1 - M_2), \tag{-134}$$

donde M_1 corresponde a la suma de las ordenadas intermedias en los nodos con índices de la forma $1+5k$ o $4+5k$ con $k \in \mathbb{N}$ M_2 corresponde a la suma de las ordenadas intermedias en los nodos con índices de la forma $2+5k$ o $3+5k$ con $k \in \mathbb{N}$.

```
function [int]=Ncotes5(f,n,a,b)

% Aplicamos la regla Ncotes5 Abierta de Sexto Orden compuesta para hacer integración numérica.
% La regla simple está basada en 6 puntos
% [int]=Ncotes5(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas la regla simple Abierta.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Ncotes5('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(5*n);

% Inicialización de variables

M1=0; % los de índice 1 y 4 módulo 5
M2=0; % los de índice 2 y 3 módulo 5


% Cálculo de los nodos
x(1)=a;
x(5*n+1)=b;

for i=2:5*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Quinto Orden Abierta
```

```
% Denotaremos por M1 a la suma de las ordenadas intermedias
% para los índices 1 y 4 módulo 5
for i=2:5:5*n-3
    M1=M1+double(subs(f,x(i)));
    M1=M1+double(subs(f,x(i+3)));
end

% Denotaremos por M2 a la suma de las ordenadas intermedias
% para los índices 2 y 3 módulo 5
for i=3:5:5*n-2
    M2=M2+double(subs(f,x(i)));
    M2=M2+double(subs(f,x(i+1)));
end

format long

% Calculando el valor de la aproximación
int=(5*h/24)*(11*M1+M2);
```

Final del programa de Matlab

3.4.4. Error de la Regla Compuesta basada en 5 intervalos Abierta

$$E_h(f) = \sum_{i=1}^{n} \int_{x_{5(i-1)}}^{x_{2i}} (f(x) - p_{3,i}(x))\, dx,$$

$$E_h(f) = \sum_{i=1}^{n} \frac{95}{144} h^5 f^{IV}(\mu_i) = \frac{95}{144} h^5 \left(\frac{\sum_{i=1}^{n} f^{IV}(\mu_i)}{n} \right) n.$$

Sustituimos uno de los valores $h = \frac{b-a}{5n}$:

$$E_h(f) = \frac{19(b-a)}{144} h^4 f^{IV}(\mu), \tag{-136}$$

con $\mu \in (a, b)$.

Por lo que en este caso tendremos un grado de exactitud 3 con un orden de aproximación global 4.

3.5. REGLA BASADA EN 6 INTERVALOS ABIERTA

3.5.1. Regla Simple basada en 6 intervalos Abierta

$$\int_a^b f(x)\ dx \approx \frac{6h}{20}(11(f(x_1)+f(x_5)) - 14(f(x_2)+f(x_4)) + 26f(x_3). \tag{-136}$$

3.5.2. Error Regla Sexto Orden Abierta Simple

$$E_h(f) = \frac{41}{140}h^5 f^{VI}(\mu) \tag{-136}$$

con $\mu \in (a,b)$.

Siendo éste un error de grado de exactitud 3 y orden de aproximación local 5.

3.5.3. Regla Compuesta basada en 6 intervalos Compuesta

A continuación se muestra el sumatorio de todas las Reglas Simples:

$$\int_a^b f(x)\ dx \approx \frac{6h}{20}(11(f(x_1)+f(x_5)) - 14(f(x_2)+f(x_4))) + 26f(x_3) + \frac{6h}{20}(11(f(x_7)+f(x_{11}))$$

$$-14(f(x_8)+f(x_{10}))+26f(x_9))+\frac{6h}{20}(11(f(x_{6n-5})+f(x_{6n-1}))-14(f(x_{6n-4})+f(x_{6n-2}))+26f(x_{6n-3})),$$

agrupamos los términos:

$$\frac{5h}{24}(11M_1 - 14M_2 + 26C), \tag{-138}$$

donde M_1 corresponde a la suma de las ordenadas intermedias en los nodos con índices de la forma $1+6k$ o $5+6k$ con $k \in \mathbb{N}$, M_2 corresponde a la suma de las ordenadas intermedias en los nodos con índices de la forma $2+6k$ o $4+6k$ con $k \in \mathbb{N}$ y C corresponde a la suma de las ordenadas centrales en el método con indices $3+6k$ con $k \in \mathbb{N}$.

3.5.4. Programa de Matlab [Ncotes6(f,N,a,b)]

```
function [int]=Ncotes6(f,n,a,b)

% Aplicamos la regla de Ncotes6 Abierta para hacer integración numérica.
% [int]=Ncotes6(f,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% n => número de veces que aplicas Ncotes6 Abierta.
% a,b => extremos del intervalo
% Datos de salida
% int => valor de la aproximación a la integral
%
% Ejemplo:
% [int]=Ncotes6('cos(2*x+1)',100,0,0.25)


% Calculamos la longitud del intervalo
h=(b-a)/(6*n);

% Inicialización de variables
M1=0; % Indices 1 y 5 módulo 6
M2=0; % Indices 2 y 4 módulo 6
C=0; % Valor central

% Cálculo de los nodos
x(1)=a;
x(6*n+1)=b;

for i=2:6*n
    x(i)=x(1)+(i-1)*h;
end
```

```
% Regla de Ncotes6 Abierta;


% Denotaremos por M1 a la suma de las ordenadas intermedias de indices 1 y 5 con modulo 6 en el método
for i=2:6:6*n-4
    M1=M1+double(subs(f,x(i)));
    M1=M1+double(subs(f,x(i+4)));
end

% Denotaremos por M2 a la suma de las ordenadas intermedias extremos de indices 2 y 4 con modulo 6 en el método
for i=3:6:6*n-3
    M2=M2+double(subs(f,x(i)));
    M2=M2+double(subs(f,x(i+2)));
end


% Denotaremos por C a la suma de las ordenadas centrales en el método
for i=4:6:6*n-2
    C=C+double(subs(f,x(i)));
end

format long

% Calculando el valor de la aproximación
int=(6*h/20)*(11*M1-14*M2+26*C);
```

Por lo que en este caso será de grado de exactitud 5 con un orden de aproximación global 4.

3.6. REGLA DE NEWTON COTES ABIERTA GENERAL

Del mismo modo que hemos hecho en la serie de *Newton Cotes Cerrada* vamos a proceder a la creación de un programa que englobe todos los métodos mostrados anteriormente y nos permita, sin necesidad de cambiar de un programa a otro, realizar el cálculo con el método más apropiado a cada caso.

Al igual que en el caso de *Newton Cotes Cerradas* debemos apoyarnos en un programa auxiliar denominado **pesosNcotes** que nos permitirá obtener los pesos correspondientes a la regla abierta para cada N introducida a partir del **Polinomio de Lagrange**.

La programación de **pesosNcotes** será la siguiente:

```
function w = Pesos_NCotes(nod)

% Esta función calcula los pesos correspondientes a las fórmulas abiertas de integración
% numérica en una dimensión para mallados uniformes del método de Newton
% Cotes.
%
% w = Pesos_NCotes(nod)
%
% Variables de entrada:
% nod vector con los nodos
%
% Variables de salida:
% w vector que contiene los pesos para la regla simple
%
% Ejemplo:
% w=Pesos_NCotes(1:3)  % Fórmula del Punto Medio


% Declaramos x como variable simbólica
syms x

% Número de nodos
n = length(nod);

% Intervalo de integración
```

```
a=nod(1);
b=nod(n);

% Cálculo de los polinomios de Lagrange

for i = 2 : n-1
    L{i-1}=1;
    for j = 2 : n-1
        if(j  = i)
        L{i-1} = L{i-1}*(x - nod(j))/ (nod(i) - nod(j));
    end
end
end

% Cálculo de la integral exacta para cada polinomio de Lagrange

fprintf('\n');

for i=2:n-1

w{i-1}=int(L{i-1},x,a,b);
fprintf('%s ',char(w{i-1}));

end
fprintf('\n\n');
```

Final del programa Matlab

Una vez tenemos el programa de **pesosNcotesa** creamos la aproximación de la integral para el cálculo. Establecemos el valor de $h = \frac{b-a}{nN}$.

Una vez hecho esto establecemos una división entre si el método seleccionado engloba un número par de puntos o si éste es impar. Esto nos permite una optimización del programa y que funcione con mayor rapidez a la hora de montar la fórmula correspondiente al método.

Nota: *Hay que tener en cuenta que en* Matlab *los vactores empiezan con índice 1 por lo que debemos llevar cuidado con este echo ya que lo que se considera sobre el papel par, en* Matlab *es impar.*

3.6.1. Programa Matlab [NcotesN(f,N,n,a,b)]

Su programación se realizará de la siguiente forma:

```
function [int]=NcotesN(f,N,n,a,b)

% Aplicamos la regla de Ncotes abierta compuesta para hacer integración numérica.
% La regla simple está basada en N-1 puntos
% [int]=NcotesN(f,N,n,a,b)
% Datos de entrada
% f función integrando pasada como una cadena de caracteres
% N indica que hay N+1 nodos contando los extremos a,b
% n número de veces que aplicas la regla simple.
% a,b extremos del intervalo
% Datos de salida
% int valor de la aproximación a la integral
%
% Ejemplo:
% [int]=NcotesN('cos(2*x+1)',5,100,0,0.25)


% Generamos el vector de pesos

w=Pesos_NCotes(1:N+1);

% Calculamos la longitud del intervalo
h=(b-a)/(N*n);


% Cálculo de los nodos
x(1)=a;
x(N*n+1)=b;

for i=2:N*n
    x(i)=x(1)+(i-1)*h;
end

% Regla de Ncotes con N-1 nodos
```

```
if mod(N,2)  %N es impar

    for j=1:(N-1)/2

        M(j)=0;


        % Denotaremos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*n+1-N+j
            M(j)=M(j)+double(subs(f,x(i+1)));
            M(j)=M(j)+double(subs(f,x(i+N+1-2*j)));
        end

    end

else  % N es par

    for j=1:N/2-1

        M(j)=0;

        % Denotamos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*n+1-N+j
            M(j)=M(j)+double(subs(f,x(i+1)));
            M(j)=M(j)+double(subs(f,x(i+N+1-2*j)));
        end
    end

    % Ahora sumamos la contribución del punto central del intervalo

    M(N/2)=0;
    for i=N/2+1:N:n*N+1-N/2
        M(N/2)=M(N/2)+double(subs(f,x(i)));
    end

end
```

```
% Calculando el valor de la aproximación

int=0;

if mod(N,2)  %N es impar
    for j=1:(N-1)/2
        int=int+w{j}*M(j);
    end
else  %N es par
    for j=1: N/2
        int=int+w{j}*M(j);
    end

end

format long
int=double(h*int);
```

Final del programa Matlab

3.6.2. Obtención del error para NcotesNa

Ahora nos enfocaremos en la obtención del error producido al aplicar un método obtenido mediante nuestro programa *NcotesNa*. Para su obtención, debido a los problemas ya indicados en las fórmulas cerradas, aplicamos igualmente el método del **Núcleo de Peano**.

El programa de obtención del Error lo denominamos *Error_NcotesN* y su programación es la siguiente:

```
function [Error_N,coef,Error_N_Compuesta,coef_Compuesta]=Error_NcotesN(N)

% Calculamos el error en la regla de Ncotes abierta simple para hacer integración numéri-
ca.
% La regla simple está basada en N-1 puntos, N+1 si contamos los extremos
% [Error_N,coef,Error_N_Compuesta,coef_Compuesta]=Error_NcotesN(N)
% Datos de entrada
```

```
% N indica que hay N+1 nodos contando los extremos a,b
% Datos de salida
% Error_N error teórico cometido en la fórmula de integración simple
% coef coeficiente numérico de la fórmula del error
% Error_N_Compuesta error teórico cometido en la fórmula de integración
% compuesta
% coef_Compuesta coeficiente numérico de la fórmula del error para la regla compuesta
%
%
% Ejemplo:
% [Error_N,coef]=Error_NcotesN(5)

% Declaramos la variables simbólicas
syms a b theta


% Generamos el vector de pesos

w=Pesos_NCotes(1:N+1);



if mod(N,2)  %N es impar

    s=N-2;

else

    s=N-1;

end


% Cálculamos la integral entre a y b del núcleo de peano K(theta)

La=1/(s+1)*1/(s+2)*(b-a)^(s+2);

for j=N:-1:2

    La=La-(b-a)/N*w{j-1}*int((((N+1-j)*a+(j-1)*b)/N-theta)^s,theta,a,((N+1-j)*a+(j-1)*b)/N);
```

```
end

coef=simplify(1/factorial(s)*La/(b-a)^(s+2)*N^(s+2));

Error_N=[char(coef),blanks(2),'f^(',num2str(s+1),')(mu)',blanks(2),'h^(',num2str(s+2),')'];

coef_Compuesta=factor(coef*(b-a)/N);

Error_N_Compuesta=[char(coef_Compuesta),blanks(2),'f^(',num2str(s+1),')(mu)',blanks(2),'h^(',num2
```

Final del programa Matlab

3.7. APLICACIÓN NAVAL PARA NEWTON COTES ABIERTO

Como hemos comentado antes, este método de aproximación de integrales se utiliza cuando la función posee una discontinuidad, esto puede aparecer en la ingeniería naval en el caso de ecuaciones relacionadas con el oleaje, y el movimiento y fuerza derivado de éste [3], Ver la Figura 25

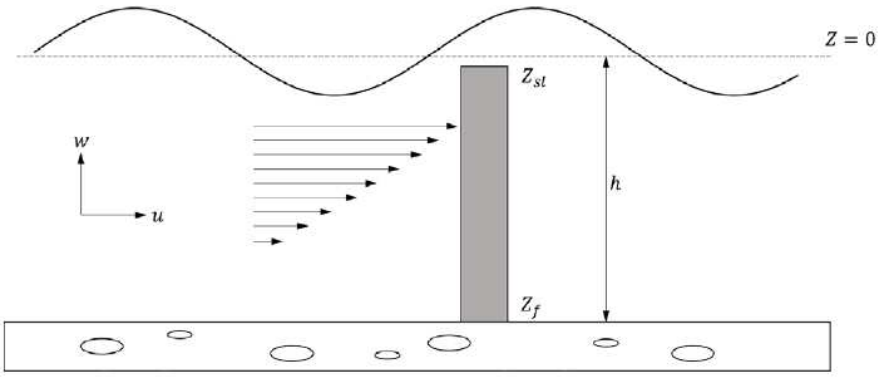

Figura 25: Movimiento de las partículas en una ola

La partícula de la ola tendrá una componente de movimiento vertical y horizontal, siendo

la ecuación de la velocidad horizontal de una partícula fluida:

$$\dot{x}(t) = A\omega \frac{cosh(k(z+h))}{senh(kh)} cos(kx - \omega t), \tag{-138}$$

donde h es la altura del poste, z es la profundidad de la partícula, A el área, ω la velocidad angular y k una constante.

La aceleración será:

$$\ddot{x}(t) = A\omega^2 \frac{cosh(k(z+h))}{senh(kh)} cos(kx - \omega t). \tag{-138}$$

La fuerza sobre el poste la obtenemos a partir de la siguiente ecuación:

$$dF(t) = \left(C_m \rho A \ddot{x} + \frac{1}{2} C_d \rho D \dot{x}^2 + C_d \rho \dot{x} D \nu + \frac{1}{2} C_d \rho D \nu^2 \right) dz,$$

siendo C_m el coeficiente de masa añadida, C_d el coeficiente de arrastre, ν la velocidad de la corriente y D la longitud del poste.

La fuerza total sobre el poste se obtiene del sumatorio de la integral de cada una de las funciones anteriores:

$$F = \int_{z_{sl}}^{z_f} dF = C_m \rho S \int_{z_{sl}}^{z_f} \ddot{x} dz + \frac{1}{2} C_d \rho D \int_{z_{sl}}^{z_f} \dot{x}^2 dz + C_d \rho D \nu \int_{z_{sl}}^{z_f} \dot{x} dz + \frac{1}{2} C_d \rho D \nu^2 \int_{z_{sl}}^{z_f} dz. \tag{-139}$$

Cada una de estas integrales las podemos aproximar mediante una regla de *Newton Cotes Abierta* ya que al ser funciones hiperbólicas, resulta un método de aproximación adecuado.

4. REGLA DE NEWTON COTES PARA VARIAS VARIABLES

Hasta ahora hemos visto la obtención de resultados óptimos mediante sustitución aplicado a integrales simples de una sola variable, sin embargo en cálculos avanzados debemos utilizar métodos más complejos como pueden ser los de *Integración Múltiple* que consiste en una modificación de forma más o menos directa del uso de la *aproximación de integrales simples* [4]. En la Figura 26 se muestra la obtención de un volumen de un cono que requiere la utilización de una integral triple.

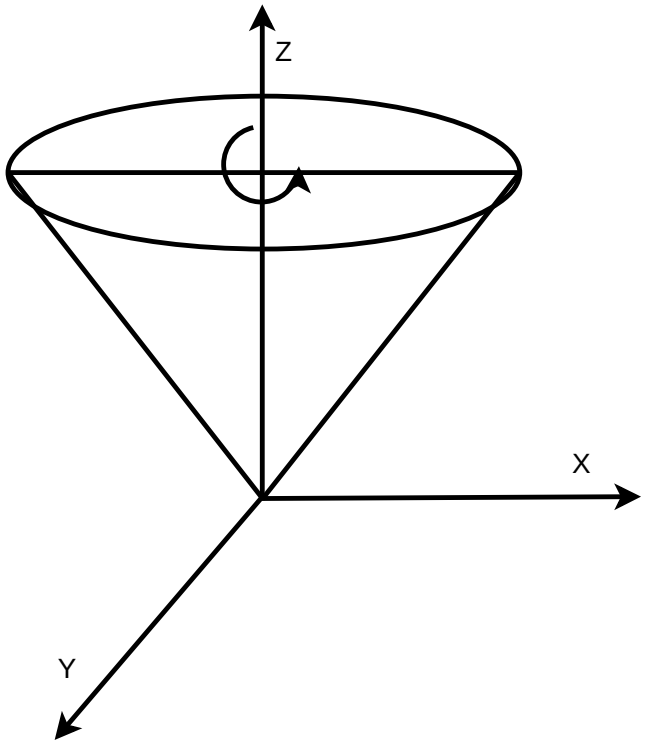

Figura 26: Muestra del cálculo de un volumen de un cono definida por una integral triple

Existen numerosas aplicaciones en el uso de las integrales múltiples, usando por ejemplo las integrales dobles para el cálculo de áreas y las integrales triples para calcular volúmenes. Además su uso se extiende al cálculo de centros de masas, momentos de inercia y otras magnitudes en el campo de la física y la ingeniería como pueden ser el flujo o la circulación de un campo vectorial, el campo eléctrico creado por una carga, el campo magnético creado por una

corriente...[4]. Estas aplicaciones aparecerán explicadas más adelante

Las integrales múltiples que vamos a estudiar se dividen en:

- **Integrales Dobles**
- **Integrales Triples**

Vamos a utilizar como variables simbólicas **x**,**y**, **z**. En la Figura 27 vemos una representación de las coordenadas cartesianas.

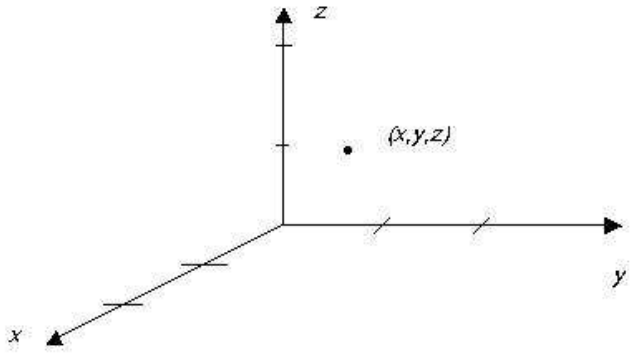

Figura 27: Coordenadas cartesianas

4.1. REGLAS DE NEWTON COTES CERRADAS PARA INTEGRALES DOBLES

Las *Integrales Dobles* se usan por ejemplo para la acotación de un área entre dos funciones $g_1(x)$ y $g_2(x)$. Tambien se usan en el cálculo del momento de un área (Usado en estructuras navales) o para el cálculo en sistemas propulsivos de la presión generada por una pala de una hélice, que se calcula como el área de las curvas entre la cara de succión y de presión. En la Figura 28 aparece representada una hélice:

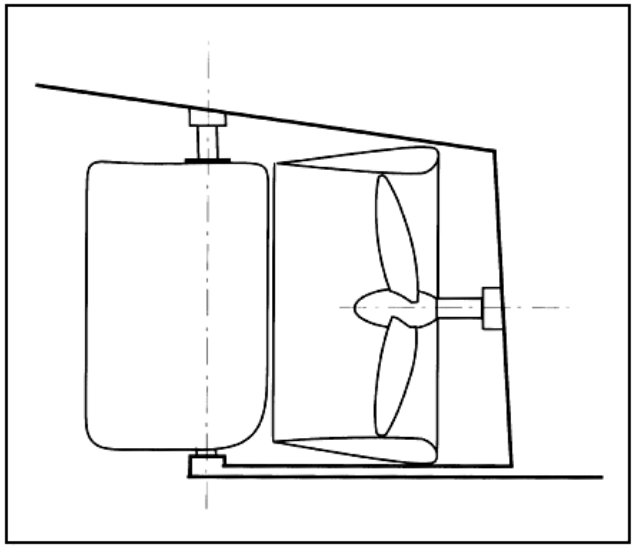

Figura 28: Esquema de hélice situada en su posición tras la carena

La forma en la que se expresa una la integral doble es la siguiente [5]:

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)\ dydx, \tag{-139}$$

donde la región considerada queda representada por los extremos del intervalo a, b en la variable x y por las funciones $c(x)$ y $d(x)$. Las variables n y m respectivamente define el número de veces que aplicamos la Regla Simple en una variable en cada dirección x o y.

Definimos:

$$h = \frac{b-a}{2n}, \quad k = \frac{d-c}{2m}, \tag{-139}$$

donde a partir de ahora vamos a prescindir en la escritura de la dependencia de las funciones $c(x)$ y $d(x)$ de la variable x, escribiendo únicamente c y d.

En lo que sigue vamos a realizar la explicación para el *Método de Simpson*, siendo las ideas igualmente válidas para cualquiera de las otras reglas.

El método de resolución consiste en evaluar la primera integral mediante la regla correspondiente considerando la variable x. Así para cada x tendremos que evaluar a su vez una integral entre c y d.

Considerando $j = 0, 1, 2...m$ y como valor de los distintos puntos $y_j = c + jk$:

$$\int_c^d f(x,y)\ dy \approx \frac{k}{3}\left[f(x,y_0) + 2\sum_{j=1}^{m-1} f(x,y_{2j}) + 4\sum_{j=1}^{m} f(x,y_{2j-1}) + f(x,y_{2m})\right].$$

A este valor habría que añadirle el valor del error perteneciente a la aproximación de la integral realizada:

$$Error = -\frac{(d-c)k^4}{180}\frac{\delta^{IV} f(x,\eta)}{\delta y^4},$$

con $\eta \in]c,d[$.

Quedándonos:

$$\int_a^b \left[\int_c^d f(x,y)\ dy\right]\ dx \approx \frac{k}{3}\int_a^b f(x,y_0)\ dx + \frac{2k}{3}\sum_{j=1}^{m-1}\int_a^b f(x,y_{2j})\ dx + \frac{4k}{3}\sum_{j=1}^{m}\int_a^b f(x,y_{2j-1})\ dx +$$

$$\frac{k}{3}\int_a^b f(x,y_{2m})\ dx. \tag{-142}$$

Ahora aplicaremos la *Regla de Simpson* en una variable a cada una de las integrales que aparecen en la fórmula 4.1, considerando los nodos $x_i = a + ih \; i = 0, 1..,2n$.

Asi para cada integral tendremos:

$$\int_a^b f(x, y_j)\, dx \approx \frac{h}{3}\left[f(x_0, y_j) + 2\sum_{i=1}^{n-1} f(x_{2i}, y_j) + 4\sum_{i=1}^{n} f(x_{2i-1}, y_j) + f(x_{2n}, y_j),\right] \qquad \text{(-142)}$$

Con un error dado por:

$$Error = -\frac{(b-a)h^4}{180}\frac{\delta^{IV} f(\mu, y_j)}{\delta x^4},$$

con $\mu \in]a, b[$

La aproximación resultante tendrá la siguiente forma:

$$\int_a^b \int_c^d f(x, y)\, dxdy \approx \frac{hk}{9}[f(x_0, y_0) + 2\sum_{i=1}^{n-1} f(x_{2i}, y_0) + 4\sum_{i=1}^{n-1} f(x_{2i-1}, y_0) +$$
$$f(x_{2n}, y_0) + 2\sum_{j=1}^{m-1} f(x_0, y_{2j}) + 4\sum_{j=1}^{m-1}\sum_{i=1}^{n-1} f(x_{2i}, y_{2j}) + 8\sum_{j=1}^{m-1}\sum_{i=1}^{n} f(x_{2i-1}, y_{2j}) +$$
$$2\sum_{j=1}^{m-1} f(x_{2n}, y_{2j}) + 4\sum_{j=1}^{m} f(x_0, y_{2j-1}) + 8\sum_{j=1}^{m}\sum_{i=1}^{n-1} f(x_{2i}, y_{2j-1}) + 16\sum_{j=1}^{m}\sum_{i=1}^{n} f(x_{2i-1}, y_{2j-1})$$
$$4\sum_{j=1}^{m} f(x_{2n}, y_{2j-1}) + f(x_0, y_{2m}) + 2\sum_{i=i}^{n-1} f(x_{2i}, y_{2m}) + 4\sum_{i=i}^{n} f(x_{2i-1}, y_{2m}) + f(x_{2n}, y_{2m})].$$

Mientras que el error lo expresaremos como:

$$E_k(f) = \frac{-(d-c)(b-a)}{180}\left[h^4\frac{\delta^{IV}f}{\delta x^4}(\mu',\eta') + k^4\frac{\delta^{IV}f}{\delta y^4}(\mu'',\eta'')\right], \qquad \text{(-147)}$$

donde (μ',η'), (μ'',η'') pertenecen al interior de la región.

En la Figura 29 vemos representada de manera muy esquemática la aproximación de una integral doble mediante Métodos Numericos.

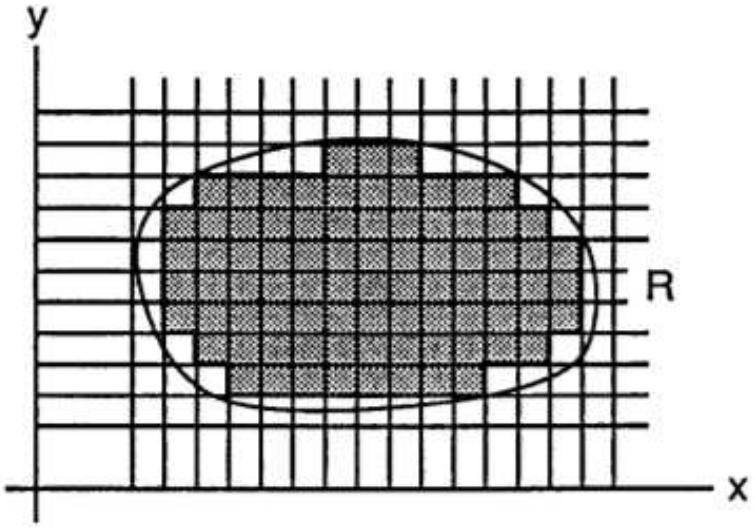

Figura 29: Muestra del cálculo de un área definida por una integral doble a través de Métodos Numéricos

4.1.1. Generación del programa NcotesN en *Matlab* para 2 variables en el caso de las fórmulas Cerradas

Hasta aquí hemos visto el planteamiento teórico de como se resolvería una integral doble a través de métodos numéricos, ahora debemos plantear como conseguir un programa en *Matlab*.

A continuación mostramos el *Método de Simpson de 2 Variables* y el *Método de los Trapecios de 2 Variables* por completitud, pudiéndose consultar mas detalles en [6], siendo la programación de ambos métodos la siguiente:

Método de los Trapecios 2 Variables:

```
function I=trapecios2var(f,c,d,a,b,m,n)

% Esta función calcula una integral doble por el método de los Trapecios
% I=trapecios2var(f,c,d,a,b,m,n)
% Variables de entrada
% f cadena de caracteres conteniendo la función integrando
% c cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% m,n número de puntos en que se parte cada intervalo según la dimensión
% Ejemplo:
% Comparamos la integral numérica con la exacta calculada con el paquete
% de simbólico de Matlab
% format long
% I=trapecios2var('x1*exp(x1+x2)','x1','2*x1',0,1,100,100)
% double(int(int('x*exp(x+y)',y,x,2*x),x,0,1))

syms x1 x2;

h=(b-a)/n;

I1=0;
I2=0;

% calculamos el valor de la integral interior para x=a

% extremos para la integral en x2 para cada x1=a
dx=subs(d,{x1},{a}); % extremo superior
cx=subs(c,{x1},{a}); % extremo inferior

% paso de integración para la integral en la segunda variable
kx=(dx-cx)/m;

% realizamos la integral en x2 por trapecios de 1 variable
K1=subs(f,{x1,x2},{a,cx})+subs(f,{x1,x2},{a,dx});
K2=0;
```

```
for j=1:m-1
    y=cx+j*kx;
    z=subs(f,{x1,x2},{a,y});
    K2=K2+z;
end

 % L guarda el resultado de la integral en x2 para cada valor de x1
L=kx*(K1/2+K2);

 % acumulamos para calcular la integral en x1
I1=I1+L;



 % bucle para la integral en la primera variable x1
for i=1:n-1

    % nodos en el eje x1
    x=a+i*h;

    % extremos para la integral en x2 para cada x1
    dx=subs(d,{x1},{x}); % extremo superior
    cx=subs(c,{x1},{x}); % extremo inferior

    % paso de integración para la integral en la segunda variable
    kx=(dx-cx)/m;

    % realizamos la integral en x2 por trapecios de 1 variable
    K1=subs(f,{x1,x2},{x,cx})+subs(f,{x1,x2},{x,dx});
    K2=0;
    for j=1:m-1
        y=cx+j*kx;
        z=subs(f,{x1,x2},{x,y});
        K2=K2+z;
    end

    % L guarda el resultado de la integral en x2 para cada valor de x1
    L=kx*(K1/2+K2);

    % acumulamos para calcular la integral en x1
```

```
    I2=I2+L;

end

% extremos para la integral en x2 para cada x1=b

dx=subs(d,{x1},{b}); % extremo superior
cx=subs(c,{x1},{b}); % extremo inferior

% paso de integración para la integral en la segunda variable
kx=(dx-cx)/m;

% realizamos la integral en x2 por simpson de 1 variable
K1=subs(f,{x1,x2},{b,cx})+subs(f,{x1,x2},{b,dx});
K2=0;
for j=1:m-1
    y=cx+j*kx;
    z=subs(f,{x1,x2},{b,y});
    K2=K2+z;
end

% L guarda el resultado de la integral en x2 para cada valor de x1
L=kx*(K1/2+K2);

% acumulamos para calcular la integral en x1
I1=I1+L;




% Valor de la integral doble

I=h*(I1/2+I2);
```

Final del programa de Matlab

Método de Simpson 2 Variables:

```
function I=simp2var(fsimp,c,d,a,b,m,n)

% Esta función calcula una integral doble por el método de Simpson
% I=simp2var(fsimp,c,d,a,b,m,n)
% Variables de entrada
% fsimp cadena de caracteres conteniendo la función integrando
% c cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% m,n número de veces que se aplica simpson simple en cada intervalo según la dimensión
% Ejemplo:
% Comparamos la integral numérica con la exacta calculada con el paquete
% de simbólico de Matlab
% format long
% I=simp2var('x1*exp(x1+x2)','x1','2*x1',0,1,20,20)
% double(int(int('x*exp(x+y)',y,x,2*x),x,0,1))

syms x1 x2;

h=(b-a)/(2*n);

I1=0;
I2=0;
I3=0;

% bucle para la integral en la primera variable x1
for i=0:(2*n)

    % nodos en el eje x1
    x=a+i*h;

    % extremos para la integral en x2 para cada x1
    dx=subs(d,{x1},{x}); % extremo superior
    cx=subs(c,{x1},{x}); % extremo inferior

    % paso de integración para la integral en la segunda variable
    kx=(dx-cx)./(2*m);
```

```
    % realizamos la integral en x2 por simpson de 1 variable
    K1=subs(fsimp,{x1,x2},{x,cx})+subs(fsimp,{x1,x2},{x,dx});
    K2=0;
    K3=0;
    for j=1:(2*m-1)
        y=cx+j*kx;
        z=subs(fsimp,{x1,x2},{x,y});
        if gcd(2,j)==2
            K2=K2+z;
        else
            K3=K3+z;
        end
    end

    % L guarda el resultado de la integral en x2 para cada valor de x1
    L=(kx/3)*(K1+2*K2+4*K3);

    % acumulamos para calcular la integral en x1
    if i==0 | i==2*n
        I1=I1+L;
    else
        if gcd(2,i)==2
            I2=I2+L;
        else
            I3=I3+L;
        end
    end

end

% Valor de la integral doble

I=(h/3).*(I1+2*I2+4*I3);
```

Final del programa de Matlab

A partir de dichos programas generaremos el nuevo programa que nos permite, además de introducir los datos habituales de los intervalos (a, b, c y d), el número de repeticiones del mé-

todo (m y n) y la función a integrar (f), introducir también N que es el número de subintervalos en los que se divide cada intervalo el la fórmula Simple y por tanto indica el tipo de método que queremos aplicar, permitiéndonos el ajuste que nosotros queramos.

El programa que hemos realizado lo describimos a grandes rasgos a continuación:

Primero realizamos una comprobación de si el resultado de la integral es cero, ya que si este es el caso podemos resolver directamente sin necesidad de que el programa haga ningún cálculo, en caso contrario comenzamos con el programa.

Para su programación recurrimos a los programas antes descritos de *Pesos_Ncotes* y *NcotesN* que nos permitirán obtener los resultados parciales necesarios para continuar con nuestro programa.

Al igual que ocurría en el programa *NcotesN* fijamos el valor del intervalo h y el valor de los extremos junto con los extremos acumulables que los denominaremos E y J respectivamente y como hemos explicado anteriormente en el punto 4.1 primero resolvemos la integral interior definiendo la función fijando a en primera instancia y mas tarde fijando b y calculando el valor de los extremos, después calculamos el valor de los extremos acumulables y para los valores de los puntos centrales realizamos la misma división que en el programa *NcotesN* y separamos si nuestro Método contiene un numero de puntos par o impar con lo que de este modo podemos ahorrar cálculos al programa, a esta función interior la denominaremos g y una vez resuelta debemos resolver la segunda integral de $x1$ del mismo modo. Una vez finalizado se repite el ciclo hasta completar todos los intervalos de nuestro Método en los que hayamos dividido las funciones.

Este proceso genera un problema al trabajar en **Matlab** y es que debido a la gran cantidad de cálculos que debemos realizar si elegimos un número alto para n, m y N el programa será lento y debemos esperar a que el programa termine de realizar cálculos.

4.1.2. Programa Matlab [NcotesN2var(f,c,d,a,b,m,n,N)]

Programa NcotesN2var:

```
function I=NcotesN2var(f,c,d,a,b,m,n,N)

% Esta función calcula una integral doble por el método de Ncotes
% I=NcotesN2var(f,c,d,a,b,m,n,N)
% Variables de entrada
% f cadena de caracteres conteniendo la función integrando
```

```
% c cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% m,n número de puntos en que se parte cada intervalo según la dimensión
% N indica que hay N+1 nodos contando los extremos a,b en la fórmula simple
% de Ncotes
% Ejemplo:
% Comparamos la integral numérica con la exacta calculada con el paquete
% de simbólico de Matlab
% format long
% I=NcotesN2var('x1*exp(x1+x2)','x1','2*x1',0,1,100,100,3)
% double(int(int('x*exp(x+y)',y,x,2*x),x,0,1))

syms x1 x2;

% Comprobamos si la integral da cero

if a==b

    I=0;

else



    % Generamos el vector de pesos

    w=Pesos_NCotes(1:N+1);


    % Calculamos la longitud del intervalo
    h=(b-a)/(N*n);

    % Cálculo de los nodos
    x(1)=a;
    x(N*n+1)=b;

    for i=2:N*n
        x(i)=x(1)+(i-1)*h;
```

```
end


% Inicialización de variables

E=0; % extremos
J=0; % donde se junta un extremo interior con otro



% calculamos el valor de la integral interior para x=a

% extremos para la integral en x2 para cada x1=a
dx=double(subs(d,{x1},{a})); % extremo superior
cx=double(subs(c,{x1},{a})); % extremo inferior


% Definición de la función fijado a

g=subs(f,{x1},{a});
[L]=NcotesN(g,N,n,cx,dx);

% acumulamos para calcular la integral en x1
E=E+L;

display('Fijado a correcto')


% extremos para la integral en x2 para cada x1=b

dx=double(subs(d,{x1},{b})); % extremo superior
cx=double(subs(c,{x1},{b})); % extremo inferior


% Definición de la función fijado b

g=subs(f,{x1},{b});
```

```
% L guarda el resultado de la integral en x2 para cada valor de x1
[L]=NcotesN(g,N,n,cx,dx);


% acumulamos para calcular la integral en x1

E=E+L;


% Denotaremos por J a la suma de las ordenadas en los extremos intermedios
for i=N+1:N:N*n-N+1

    % extremos para la integral en x2 para cada x1=x(i)

    dx=double(subs(d,{x1},{x(i)})); % extremo superior
    cx=double(subs(c,{x1},{x(i)})); % extremo inferior


    % Definición de la función fijado x(i)

    g=subs(f,{x1},{x(i)});

    % L guarda el resultado de la integral en x2 para cada valor de x1
    [L]=NcotesN(g,N,n,cx,dx);

    J=J+L;
end



if mod(N,2) %N es impar

    for j=1:(N-1)/2

        M(j)=0;


        % Denotaremos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N
```

```
for i=j:N:N*n+1-N+j

    % extremos para la integral en x2 para cada x1=x(i+1)

    dx=double(subs(d,{x1},{x(i+1)})); % extremo superior
    cx=double(subs(c,{x1},{x(i+1)})); % extremo inferior


    % Definición de la función fijado x(i+1)

    g=subs(f,{x1},{x(i+1)});

    % L guarda el resultado de la integral en x2 para cada valor de x1
    [L]=NcotesN(g,N,n,cx,dx);

    % Actualiza el valor del acumulador

    M(j)=M(j)+double(L);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % extremos para la integral en x2 para cada x1=x(i+N+1-2*j)

    dx=double(subs(d,{x1},{x(i+N+1-2*j)})); % extremo superior
    cx=double(subs(c,{x1},{x(i+N+1-2*j)})); % extremo inferior


    % Definición de la función fijado x(i+N+1-2*j)

    g=subs(f,{x1},{x(i+N+1-2*j)});

    % L guarda el resultado de la integral en x2 para cada valor de x1
    [L]=NcotesN(g,N,n,cx,dx);

    % Actualiza el valor del acumulador

    M(j)=M(j)+double(L);
```

```
        end

    end

else % N es par

    for j=1:N/2-1

        M(j)=0;

        % Denotamos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*n+1-N+j

            % extremos para la integral en x2 para cada x1=x(i+1)

            dx=double(subs(d,{x1},{x(i+1)})); % extremo superior
            cx=double(subs(c,{x1},{x(i+1)})); % extremo inferior


            % Definición de la función fijado x(i+1)

            g=subs(f,{x1},{x(i+1)});

            % L guarda el resultado de la integral en x2 para cada valor de x1
            [L]=NcotesN(g,N,n,cx,dx);

            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            % extremos para la integral en x2 para cada x1=x(i+N+1-2*j)

            dx=double(subs(d,{x1},{x(i+N+1-2*j)})); % extremo superior
            cx=double(subs(c,{x1},{x(i+N+1-2*j)})); % extremo inferior
```

```
            % Definición de la función fijado x(i+N+1-2*j)

            g=subs(f,{x1},{x(i+N+1-2*j)});

            % L guarda el resultado de la integral en x2 para cada valor de x1
            [L]=NcotesN(g,N,n,cx,dx);

            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);

        end
    end

    % Ahora sumamos la contribución del punto central del intervalo

    M(N/2)=0;
    for i=N/2+1:N:n*N+1-N/2

        % extremos para la integral en x2 para cada x1=x(i)

        dx=double(subs(d,{x1},{x(i)})); % extremo superior
        cx=double(subs(c,{x1},{x(i)})); % extremo inferior


        % Definición de la función fijado x(i)

        g=subs(f,{x1},{x(i)});

        % L guarda el resultado de la integral en x2 para cada valor de x1
        [L]=NcotesN(g,N,n,cx,dx);


        M(N/2)=M(N/2)+double(L);
    end

end
```

```
    % Calculando el valor de la aproximación

    I=double(w{1}*E+2*w{1}*J);

    if mod(N,2)  %N es impar
        for j=1:(N-1)/2
            I=I+w{j+1}*M(j);
        end
    else  %N es par
        for j=1: N/2
            I=I+w{j+1}*M(j);
        end

    end

    format long
    I=double(h*I);

end
```

Final del programa de Matlab

Ejecutando el ejemplo test que se indica en el código del programa obtenemos un error dado por:

$$E = (Integral\_Exacta) - (Newton\_Cotes) = -1{,}424643514269519e - 09.$$

Este resultado, que he obtenido mediante el cálculo con **Matlab** nos indica lo pequeño que es el error producido cuando aplicamos el método usando un número considerable de puntos, determinado por los valores de n y m. Podemos asumir este error como permisible y despreciable a la hora de realizar cálculos por este procedimiento.

4.1.3. Aplicación Naval de Newton Cotes Cerradas para 2 variables

El caso práctico principal que vamos a analizar es el cálculo de las presiones generadas por una pala de una hélice que se mueve en un fluido. En nuestro caso nos estamos refiriendo a la hélice que propulsa el buque, ver Figura 30

Figura 30: Hélice de un buque

La hélice genera una fuerza a través de la suma de la presión generada por la cara de presión y la depresión generada en la cara de succión comom se ve en la Figura 31. Ambas vienen dadas por una curva de presiones cuyas características dependen de las formas de la pala, la relación área expandida área disco y demás características de cada pala.

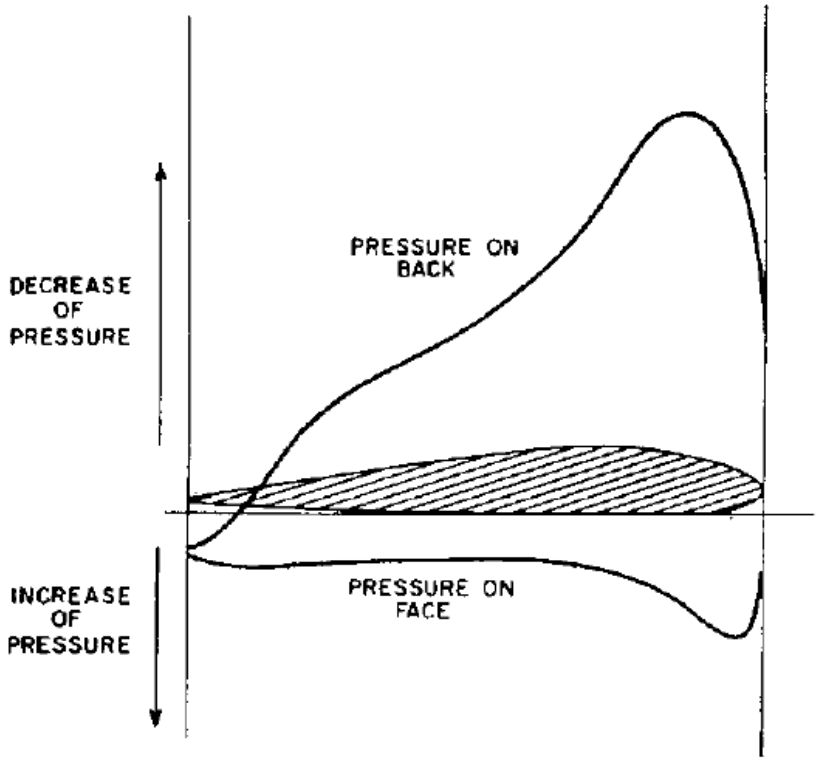

Figura 31: Curvas de presiones en una pala entre la cara de presión y succión

El área entre estas curvas se pueden integrar para obtener la presión generada y por tanto la presión total experimentada por cada pala.

Las curvas nos las pueden dar mediante la función que representa cada una de ellas. Otra opción es que los datos de entrada sean los puntos de un mallado que cubre la región entre las dos curvas.

4.2. REGLAS DE NEWTON COTES ABIERTAS PARA INTEGRALES DOBLES

Una vez establecido el programa para integrales cerradas de dos variables nos presentamos en el caso de que ambas integrales sean abiertas por lo que presentan una asíntota o cualquier perturbación que no se pueda calcular a partir del método cerrado.

Utilizado principalmente para el caso en que nos encontremos con funciones hiperbólicas como en ecuaciones de ondas del agua o vibraciones.

A la hora de realizar el programa nos hemos basado principalmente en el programa de *NcotesN2var* al cual le hemos realizado los cambios necesarios al igual que hicimos cuando modificamos el programa *NcotesN* para transformarlo en su versión abierta.

4.2.1. Programa Matlab [NcotesN2varA(f,c,d,a,b,m,n,N)]

```
function int=NcotesN2varA(f,c,d,a,b,m,n,N)

% Tengo que modificarlo basandome en el programa de NcotesN abierta
% Esta función calcula una integral doble por el método de Ncotes
% I=NcotesN2varA(f,c,d,a,b,m,n,N)
% Variables de entrada
% f cadena de caracteres conteniendo la función integrando
% c cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% m,n número de puntos en que se parte cada intervalo según la dimensión
% N indica que hay N+1 nodos contando los extremos a,b en la fórmula simple
% de Ncotes
% Ejemplo:
% Comparamos la integral numérica con la exacta calculada con el paquete
% de simbólico de Matlab
% format long
% int=NcotesN2varA('x1*exp(x1+x2)','x1','2*x1',0,1,100,100,4)
% double(int(int('x*exp(x+y)',y,x,2*x),x,0,1))

syms x1 x2;
```

```
if a==b
    int=0
else


    % Generamos el vector de pesos

    w=Pesos_NCotes(1:N+1);


    % Calculamos la longitud del intervalo
    h=(b-a)/(N*n);

    % Cálculo de los nodos
    x(1)=a;
    x(N*n+1)=b;

    for i=2:N*n
        x(i)=x(1)+(i-1)*h;
    end

    if mod(N,2)  %N es impar

        for j=1:(N-1)/2

            M(j)=0;

            % Denotaremos por M(j) a la suma de las ordenadas intermedias
            % para los índices j y i+N+1-2*j módulo N

            for i=j:N:N*n+1-N+j

                % extremos para la integral en x2 para cada x1=x(i+1)

                dx=double(subs(d,{x1},{x(i+1)}));  % extremo superior
                cx=double(subs(c,{x1},{x(i+1)}));  % extremo inferior


                % Definición de la función fijado x(i+1)
```

```
            g=double(subs(f,{x1},{x(i+1)}));

            % L guarda el resultado de la integral en x2 para cada valor de x1
            [L]=NcotesN(g,N,n,cx,dx);

            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            % extremos para la integral en x2 para cada x1=x(i+N+1-2*j)

            dx=double(subs(d,{x1},{x(i+N+1-2*j)})); % extremo superior
            cx=double(subs(c,{x1},{x(i+N+1-2*j)})); % extremo inferior


            % Definición de la función fijado x(i+N+1-2*j)

            g=subs(f,{x1},{x(i+N+1-2*j)});

            % L guarda el resultado de la integral en x2 para cada valor de x1
            [L]=NcotesN(g,N,n,cx,dx);

            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);


        end

    end

else % N es par

    for j=1:N/2-1

        M(j)=0;
```

```
 % Denotamos por M(j) a la suma de las ordenadas intermedias
 % para los índices j y i+N+1-2*j módulo N

for i=j:N:N*n+1-N+j

    % extremos para la integral en x2 para cada x1=x(i+1)

    dx=double(subs(d,{x1},{x(i+1)})); % extremo superior
    cx=double(subs(c,{x1},{x(i+1)})); % extremo inferior


    % Definición de la función fijado x(i+1)

    g=subs(f,{x1},{x(i+1)});

    % L guarda el resultado de la integral en x2 para cada valor de x1
    [L]=NcotesN(g,N,n,cx,dx);

    % Actualiza el valor del acumulador

    M(j)=M(j)+double(L);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % extremos para la integral en x2 para cada x1=x(i+N+1-2*j)

    dx=double(subs(d,{x1},{x(i+N+1-2*j)})); % extremo superior
    cx=double(subs(c,{x1},{x(i+N+1-2*j)})); % extremo inferior


    % Definición de la función fijado x(i+N+1-2*j)

    g=subs(f,{x1},{x(i+N+1-2*j)});

    % L guarda el resultado de la integral en x2 para cada valor de x1
    [L]=NcotesN(g,N,n,cx,dx);

    % Actualiza el valor del acumulador
```

```
            M(j)=M(j)+double(L);

        end
    end

    % Ahora sumamos la contribución del punto central del intervalo

    M(N/2)=0;
    for i=N/2+1:N:n*N+1-N/2

        % extremos para la integral en x2 para cada x1=x(i)

        dx=double(subs(d,{x1},{x(i)})); % extremo superior
        cx=double(subs(c,{x1},{x(i)})); % extremo inferior


        % Definición de la función fijado x(i)

        g=subs(f,{x1},{x(i)});

        % L guarda el resultado de la integral en x2 para cada valor de x1
        [L]=NcotesN(g,N,n,cx,dx);


        M(N/2)=M(N/2)+double(L);
    end

end


% Calculando el valor de la aproximación

int=0;

if mod(N,2) %N es impar
    for j=1:(N-1)/2
        int=int+w{j}*M(j);
    end
else %N es par
    for j=1: N/2
```

```
            int=int+w{j}*M(j);
        end

    end

    format long
    int=double(h*int);
end
```

Final del programa de Matlab

4.2.2. Aplicación Naval de Newton Cotes Abiertas para 2 variables

Una de las aplicaciones que podemos dar a este programa, igual que en el caso visto en el apartado 3.7 de aplicación naval para *Newton Cotes Abierto en una variable*, es en el ámbito de las estructuras marinas [3] sometidas a la acción de un campo de olas con profundidad finita.

En el caso de un cilindro sumergido en un campo de olas, lo que nos interesa es conocer la fuerza por unidad de longitud sobre el cilindro. Una forma de obtener esta fuerza es mediante la integración de las fuerzas de *Froude - Krylov*:

$$f_{fk} = -\int_{-h}^{0}\int_{0}^{2\pi} \frac{\delta\phi}{\delta t} r cos(\theta)\ d\theta\ dz, \tag{-148}$$

Siendo el elemento ϕ el potencial del oleaje que se descompone en:

Potencial de radiación ϕ_r:

$$\phi_r = \frac{gA}{\omega}\frac{cosh(k(z+d)))}{cosh(kh)}\left[A_0 H_0^{(1)}(kr) + \sum_{m=1}^{\infty} 2i^m A_m H_m^{(1)}(kr) cosm\theta\right] e^{-i\omega t}. \tag{-148}$$

Potencial de la ola ϕ_o:

$$\phi_o = \frac{gA}{\omega}\frac{cosh(k(z+d)))}{cosh(kh)}e^{-i(kx-\omega t)}. \tag{-148}$$

Como se observa debido a que en ambas ecuaciones aparece un *cosh*, parece apropiado aplicar en la expresión (4.2.2) la aproximación mediante *Newton Cotes Abiertas para 2 variables.*

4.3. REGLAS DE NEWTON COTES CERRADAS PARA INTEGRALES TRIPLES

Los métodos que hemos mostrado anteriormente se pueden generalizar para la aproximación de funciones de más variables. Las integrales de 3 variables son útiles por ejemplo para el cálculo de volúmenes:

$$I = \int_a^b \int_c^d \int_e^f g(x, y, z)\, dz\, dy\, dx. \tag{-148}$$

En la Figura 32 aparece el caso típico de cálculo de volumen de una esfera mediante usando coordenadas esféricas.

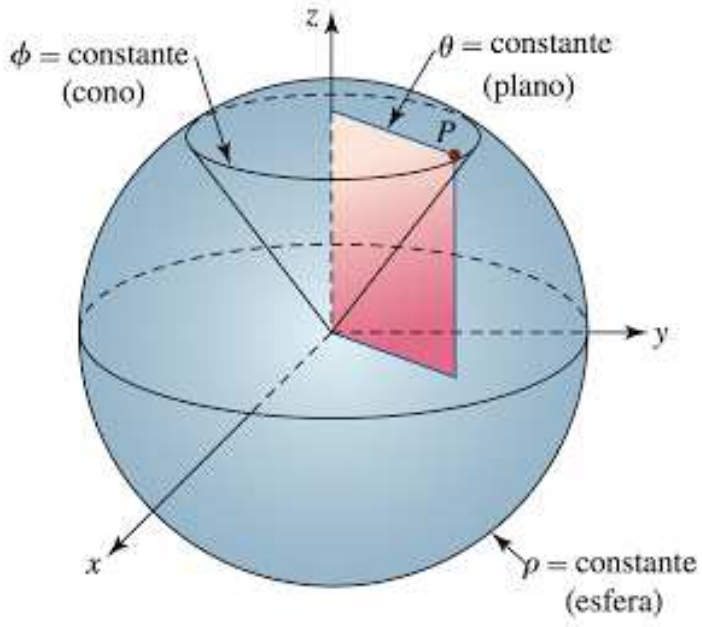

Figura 32: Obtención del volumen de una esfera mediante una integral triple

La aproximación de las *Integrales Triples* se realiza del mismo modo que hemos visto en las integrales dobles, pero con la diferencia de que al haber tres incógnitas deberemos repetir el proceso de integración secuencialmente para cada integral de la más interior a la más

exterior, con lo que se alarga el cálculo, la aproximación se realiza mediante la aplicación del método correspondiente a cada una de las integrales considerando la superior como constante y resolviendo el método con esta integral superior una vez finalizado.

Es decir, la aproximación de esta integral mediante métodos numéricos es muy similar a la realizada con la integral de 2 variables con la diferencia de que al ser triple debemos repetir del ciclo de integración una vez más. La formulación matemática no alberga ideas novedosas respecto de lo expuesto anteriormente, si bien las ecuaciones resultantes son laboriosas de escribir.La represantación gráfica de la Figura 33 muestra el mallado realizado sobre el tetraedro para acceder a los valores necesarios para computar la *Integral Triple* que nos proporciona el volumen de la figura.

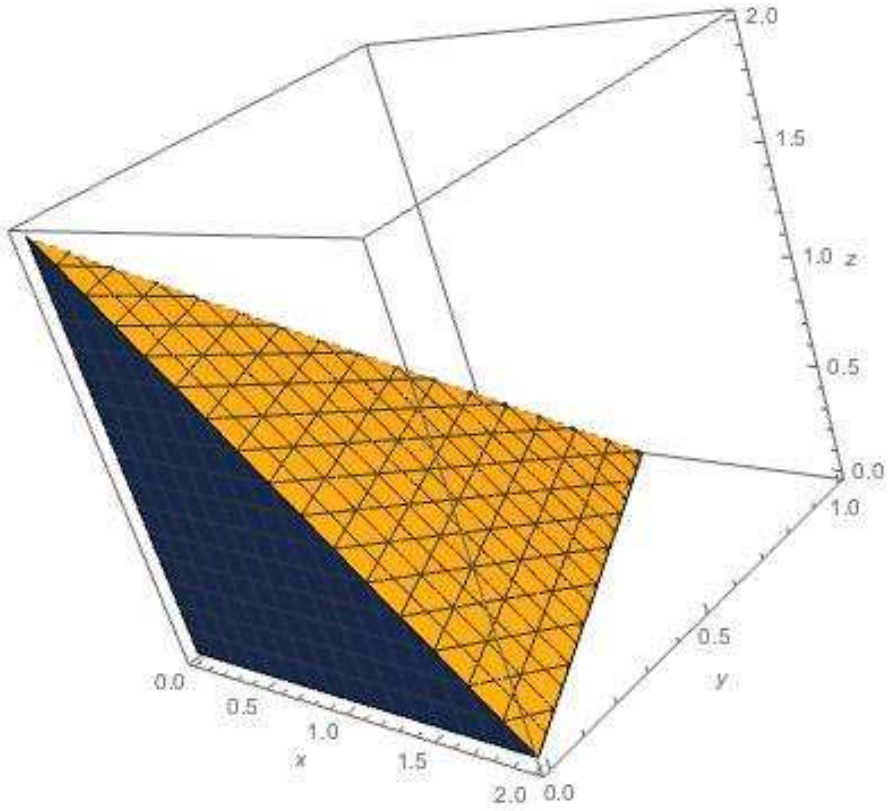

Figura 33: Representación de un mallado tridimensional de un tetraedro

4.3.1. Generación del programa NcotesN Cerradas en *Matlab* para 3 variables

Los programas en los que me he basado para la creación de un programa general al que he denominado **NcotesN3var** son los de **trapecios3var**, **simpson3var** [6] que expondré a

continuación:

Programa trapecios3var:

```
function I=trapecios3var(fint,e,f,c,d,a,b,p,m,n)

% Esta función calcula una integral triple por el método de los Trapecios
% I=trapecios3var(fint,e,f,c,d,a,b,m,n,p)
% Variables de entrada
% fint cadena de caracteres conteniendo la función integrando
% e cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% f cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% c cadena de caracteres conteniendo el extremo inferior de la segunda integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la segunda integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% p,m,n número de puntos en que se parte cada intervalo según la dimensión
% Ejemplo:
% I=trapecios3var('x1^2+x2^2+x3^2','0','sqrt(1-x1^2-x2^2)','0','sqrt(1-x1^2)',0,1,20,20,20)


syms x1 x2 x3;

h=(b-a)/n;

I1=0;
I2=0;

% bucle para la integral en la primera variable x1
for i=0:n

    % nodos en el eje x1
    x=a+i*h;

    % extremos para la integral doble resultante

    % segunda integral a calcular
    dx=subs(d,{x1},{x}); % extremo superior
    cx=subs(c,{x1},{x}); % extremo inferior
```

```
% primera integral a calcular
fx=subs(f,{x1},{x}); % extremo superior
if isnumeric(fx)
    fx=num2str(fx);
else
    fx=char(fx);
end
ex=subs(e,{x1},{x}); % extremo inferior
if isnumeric(ex)
    ex=num2str(ex);
else
    ex=char(ex);
end

% calculamos la integral doble
fint2=subs(fint,{x1},{x});
if isnumeric(fint2)
    fint2=num2str(fint2);
else
    fint2=char(fint2);
end

% cambiamos primero el nombre de las variables
end
end
end
end


L=trapecios2var(fint2,ex,fx,cx,dx,p,m);


% acumulamos para calcular la integral en x1
if i==0 | i==n
    I1=I1+L;
else
    I2=I2+L;
end
```

```
end

 % Valor de la integral doble

I=h*(I1/2+I2);
```

Final del programa de Matlab

Programa simpson3var:

```
 function I=simp3var(fsimp,e,f,c,d,a,b,p,m,n)

% Esta función calcula una integral triple por el método de Simpson
% I=simp3var(fsimp,e,f,c,d,a,b,p,m,n)
% Variables de entrada
% fsimp cadena de caracteres conteniendo la función integrando
% e cadena de caracteres conteniendo el extremo inferior de la primera integral a calcular
% f cadena de caracteres conteniendo el extremo superior de la primera integral a calcular
% c cadena de caracteres conteniendo el extremo inferior de la segunda integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la segunda integral a calcular
% a extremo inferior de la segunda integral a calcular
% b extremo superior de la segunda integral a calcular
% p,m,n número de puntos en que se parte cada intervalo según la dimensión
% Ejemplo:
% I=simp3var('x1^2+x2^2+x3^2','sqrt(1-x1^2-x2^2)','0','sqrt(1-x1^2)','0',0,1,20,20,20)

syms x1 x2 x3;

h=(b-a)/(2*n);

I1=0;
I2=0;
I3=0;

 % bucle para la integral en la primera variable x1
for i=0:(2*n)

     % nodos en el eje x1
```

```
    x=a+i*h;

    % extremos para la integral doble resultante

    % segunda integral a calcular
    dx=subs(d,{x1},{x}); % extremo superior
    cx=subs(c,{x1},{x}); % extremo inferior

    % primera integral a calcular
    ex=subs(e,{x1},{x}); % extremo superior
    fx=subs(f,{x1},{x}); % extremo inferior

    % calculamos la integral doble
    fint2=subs(fint,{x1},{x});
    % cambiamos primero el nombre de las variables
    ind=strfind(f,'x2'); fint2(ind+1)='1';
    ind=strfind(f,'x3'); fint2(ind+1)='2';

    L=simp2var(fint2,fx,ex,cx,dx,p,m);


    % acumulamos para calcular la integral en x1
    if i==0 | i==2*n
        I1=I1+L;
    else
        if gcd(2,i)==2
            I2=I2+L;
        else
            I3=I3+L;
        end
    end

end

% Valor de la integral doble

I=(h/3)*(I1+2*I2+4*I3);
```

Final del programa de Matlab

A partir de estos dos programas definimos el nuestro que nos permitirá introducir el número de puntos en los que dividimos cada intervalo, el método que queremos utilizar mediante la introducción del parámetro N, la función integrando y los extremos de integración.

Ya que la estructura del programa es muy similar al visto en $NcotesN2var$, solo nos queda mencionar que en este caso hay que añadir los valores extremos e y f para la tercera variable y el parámetro p que es el número de veces que se aplica la regla simple en la tercera variable. A partir de p se define:

$$l = \frac{f-e}{Np}. \tag{-148}$$

De igual manera utilizamos los programas *Pesos_Ncotes*. Pero, en lugar de usar el programa *NcotesN*, usamos el programa antes definido *NcotesN2var*debido a que así ahorramos en código de programación y realizamos una estructura que se basa en el cálculo de la integral entre los extremos a y b de la función de 2 variables:

$$I = \int_{c(x)}^{d(x)} \int_{e(x,y)}^{f(x,y)} g(x,y,z)\ dz\ dy. \tag{-148}$$

4.3.2. Programa Matlab [NcotesN3var(T,e,f,c,d,a,b,p,m,n,N)]

```
function I=NcotesN3var(T,e,f,c,d,a,b,p,m,n,N)

% Esta función calcula una integral triple por el método de Ncotes
% I=NcotesN3var(T,e,f,c,d,a,b,m,n,N)
% Variables de entrada
% T cadena de caracteres conteniendo la función integrando
% e cadena de caracteres conteniendo el extremo inferior de la primera
% integral a calcular
% f cadena de caracteres conteniendo el extremo superior de la primera
% integral a calcular
% c cadena de caracteres conteniendo el extremo inferior de la segunda integral a calcular
% d cadena de caracteres conteniendo el extremo superior de la segunda integral a calcular
% a extremo inferior de la tercera integral a calcular
% b extremo superior de la tercera integral a calcular
```

```
% m,n número de puntos en que se parte cada intervalo según la dimensión
% N indica que hay N+1 nodos contando los extremos a,b en la fórmula simple
% de Ncotes
% Ejemplo:
% Comparamos la integral numérica con la exacta calculada con el paquete
% de simbólico de Matlab
% format long
% I=NcotesN3var('x1*x2*exp(x1+x2+x3)','0','x1+x3','x3','2*x3',0,0.1,20,20,20,3)
% double(int(int(int('x*y*exp(x+y+z)',z,0,x+y),y,x,2*x),x,0,0.1))

syms x1 x2 x3;

if a==b

    I=0;

else


    % Generamos el vector de pesos

    w=Pesos_NCotes(1:N+1);


    % Calculamos la longitud del intervalo
    h=(b-a)/(N*p);

    % Cálculo de los nodos
    x(1)=a;
    x(N*p+1)=b;

    for i=2:N*p
        x(i)=x(1)+(i-1)*h;
    end



    % Inicialización de variables

    E=0; % extremos
```

```
J=0; % donde se junta un extremo interior con otro



% calculamos el valor de la integral interior para x3=a

% extremos para la integral en x1 y x2 para x3=a
dx=double(subs(d,{x3},{a})); % extremo superior
cx=double(subs(c,{x3},{a})); % extremo inferior

fx=subs(f,{x3},{a}); % extremo superior
ex=subs(e,{x3},{a}); % extremo inferior


% Definición de la función fijado a

g=subs(T,{x3},{a})
ex
fx
cx
dx
whos


[L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)


% acumulamos para calcular la integral en x3
E=E+L;

display('Fijado a correcto')


% extremos para la integral en x1 y x2 para x3=b
dx=subs(d,{x3},{b}); % extremo superior
cx=subs(c,{x3},{b}); % extremo inferior

fx=subs(f,{x3},{b}); % extremo superior
ex=subs(e,{x3},{b}); % extremo inferior
```

```
% Definición de la función fijado b

g=subs(T,{x3},{b})
ex
fx
cx
dx
whos

[L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)



% acumulamos para calcular la integral en x3

E=E+L;

display('Fijado b correcto')

% Denotaremos por J a la suma de las ordenadas en los extremos intermedios
for i=N+1:N:N*p-N+1

    % extremos para la integral en x1 y x2 para cada x3=x(i)
    dx=subs(d,{x3},{x(i)}); % extremo superior
    cx=subs(c,{x3},{x(i)}); % extremo inferior

    fx=subs(f,{x3},{x(i)}); % extremo superior
    ex=subs(e,{x3},{x(i)}); % extremo inferior


    % Definición de la función fijado x(i)

    g=subs(T,{x3},{x(i)})
    ex
    fx
    cx
    dx
    whos
```

```
    [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)


    J=J+L;
end

display('J correcto')


if mod(N,2)  %N es impar

    for j=1:(N-1)/2

        M(j)=0;


        % Denotaremos por M(j) a la suma de las ordenadas intermedias
        % para los índices j y i+N+1-2*j módulo N

        for i=j:N:N*p+1-N+j

            % extremos para la integral en x1 y x2 para cada x3=x(i+1)
            dx=subs(d,{x3},{x(i+1)}); % extremo superior
            cx=subs(c,{x3},{x(i+1)}); % extremo inferior

            fx=subs(f,{x3},{x(i+1)}); % extremo superior
            ex=subs(e,{x3},{x(i+1)}); % extremo inferior


            % Definición de la función fijado x(i+1)

            g=subs(T,{x3},{x(i+1)})
            ex
            fx
            cx
            dx
            whos

            [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)
```

```
            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            % extremos para la integral en x1 y x2 para cada x3=x(i+N+1-2*j)

            dx=subs(d,{x3},{x(i+N+1-2*j)}); % extremo superior
            cx=subs(c,{x3},{x(i+N+1-2*j)}); % extremo inferior

            fx=subs(f,{x3},{x(i+N+1-2*j)}); % extremo superior
            ex=subs(e,{x3},{x(i+N+1-2*j)}); % extremo inferior


            % Definición de la función fijado i+N+1-2*j

            g=subs(T,{x3},{x(i+N+1-2*j)})
            ex
            fx
            cx
            dx
            whos

            [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)


            % Actualiza el valor del acumulador

            M(j)=M(j)+double(L);


        end

    end

else % N es par
```

```
for j=1:N/2-1

    M(j)=0;

    % Denotamos por M(j) a la suma de las ordenadas intermedias
    % para los índices j y i+N+1-2*j módulo N

    for i=j:N:N*p+1-N+j

        % extremos para la integral en x1 y x2 para cada x3=x(i+1)
        dx=subs(d,{x3},{x(i+1)}); % extremo superior
        cx=subs(c,{x3},{x(i+1)}); % extremo inferior

        fx=subs(f,{x3},{x(i+1)}); % extremo superior
        ex=subs(e,{x3},{x(i+1)}); % extremo inferior


        % Definición de la función fijado x(i+1)

        g=subs(T,{x3},{x(i+1)})
        ex
        fx
        cx
        dx
        whos

        [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)



        % Actualiza el valor del acumulador

        M(j)=M(j)+double(L);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        % extremos para la integral en x1 y x2 para cada x3=x(i+N+1-2*j)

        dx=subs(d,{x3},{x(i+N+1-2*j)}); % extremo superior
```

```
        cx=subs(c,{x3},{x(i+N+1-2*j)}); % extremo inferior

        fx=subs(f,{x3},{x(i+N+1-2*j)}); % extremo superior
        ex=subs(e,{x3},{x(i+N+1-2*j)}); % extremo inferior


        % Definición de la función fijado i+N+1-2*j

        g=subs(T,{x3},{x(i+N+1-2*j)})
        ex
        fx
        cx
        dx
        whos

        [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)

        % Actualiza el valor del acumulador

        M(j)=M(j)+double(L);

    end
end

% Ahora sumamos la contribución del punto central del intervalo

M(N/2)=0;
for i=N/2+1:N:p*N+1-N/2

    % extremos para la integral en x1 y x2 para cada x3=x(i)
    dx=subs(d,{x3},{x(i)}); % extremo superior
    cx=subs(c,{x3},{x(i)}); % extremo inferior

    fx=subs(f,{x3},{x(i)}); % extremo superior
    ex=subs(e,{x3},{x(i)}); % extremo inferior


    % Definición de la función fijado x(i)

    g=subs(T,{x3},{x(i)})
```

```
            ex
            fx
            cx
            dx
            whos

            [L]=NcotesN2var(g,ex,fx,cx,dx,m,n,N)


            M(N/2)=M(N/2)+double(L);
        end

    end







    % Calculando el valor de la aproximación

    I=double(w{1}*E+2*w{1}*J);

    if mod(N,2) %N es impar
        for j=1:(N-1)/2
            I=I+w{j+1}*M(j);
        end
    else %N es par
        for j=1: N/2
            I=I+w{j+1}*M(j);
        end

    end

    format long
    I=double(h*I);

end
```

Final del programa de Matlab

Del mismo modo que ocurría en el programa *NcotesN2var* el proceso de cálculo es muy lento, más que el caso anterior, ya que hay una integral anidada a calcular mas .

Nota: Un detalle interesante a tener en cuenta es que he programado tanto *NcotesN2var* como *NcotesN3var* para que muestre por pantalla la aproximación calculada en cada bucle, de este modo sabemos que el programa está en funcionamiento y en que parte del cálculo se encuentra.

4.3.3. Aplicación Naval de Newton Cotes Cerradas para 3 variables

De entre las reglas utilizadas este es el método en el que menos aplicaciones podemos describir, sin embargo estas reglas pueden ser útiles tanpo para ecuaciones de fuerza de ondas en 3 dimensiones, o en el caso de hacer un estudio de presupuestos en los que tengamos que analizar 3 variables distintas.

Con esto quedaría cerrado el apartado de **Métodos Numéricos para la aproximación de integrales**.

5. ECUACIONES DIFERENCIALES

Los métodos numéricos se usan para la modelización de la evolución en el tiempo de los sistemas físicos, mecánicos o biológicos. De este modo se pueden predecir su comportamiento en un futuro o bien conociendo su comportamiento pasado, tratar de comprender mejor los mecanismos que hacen aparición en dicha evolución, mejorando de este modo los modelos [4].

Puede ocurrir que en ciertos casos no exista únicamente una sola solución analítica, o que la solución sea muy difícil de calcular de manera exacta, por lo que debemos recurrir a las aproximaciones numéricas para su resolución. Normalmente medimos la variación de una variable respecto a otra, lo que que traducido al modelo matemático es una ecuación diferencial en donde la velocidad de cambio de una función determinada se relaciona con variables dependientes e independientes.

Un buen ejemplo de su aplicación [6] es el caso de un cuerpo que se enfría desde una temperatura $y(t)$, conjeturando que el cambio de temperatura está relacionado con la diferencia entre la temperatura del cuerpo y del medio que lo rodea y apoyándonos en la ley de Newton que establece que «*Cuando la diferencia de temperaturas entre un cuerpo y su medio ambiente no es demasiado grande, el calor transferido en la unidad de tiempo hacia el cuerpo o desde el cuerpo por conducción, convicción y radiación es aproximadamente proporcional a la diferencia de temperatura entre el cuerpo y el medio externo.*». En la Figura 34 se puede observar el enfriamiento de un sólido.

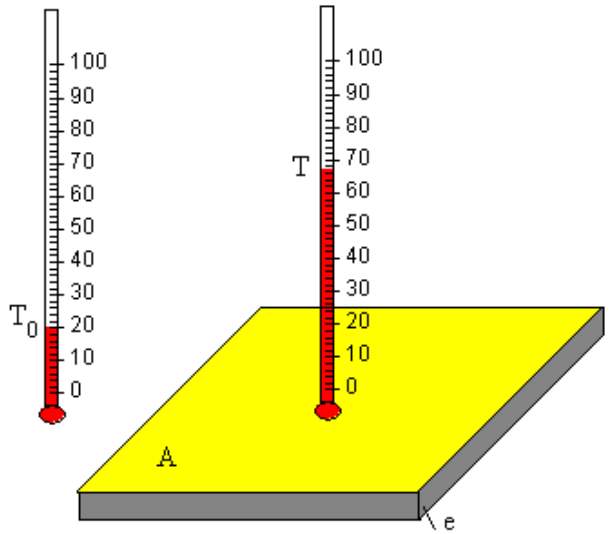

Figura 34: Variación de temperatura de un cuerpo

Si llamamos A a la temperatura ambiente que rodea al cuerpo y $y(t)$ a la temperatura del cuerpo en un instante t y siendo k una constante positiva que define el intercambio de calor:

$$\frac{dy}{dt} = -k(y - A).$$

Nota: *El signo negativo es debido a que el cuerpo se enfría si está a una temperatura mayor que la del medio.*

Si conocemos la temperatura inicial del cuerpo y_0 *con* $t = 0$ (que se denomina condición inicial), la incluimos en la ecuación. La solución la obtenemos mediante el método de la separación de variables:

$$y = A + (y_0 - A)e^{-kt}.$$

Mediante esta ecuación a partir de los datos de la condición inicial obtendremos distintos resultados, que se representarán en forma de curva como se ve en la Figura (35).

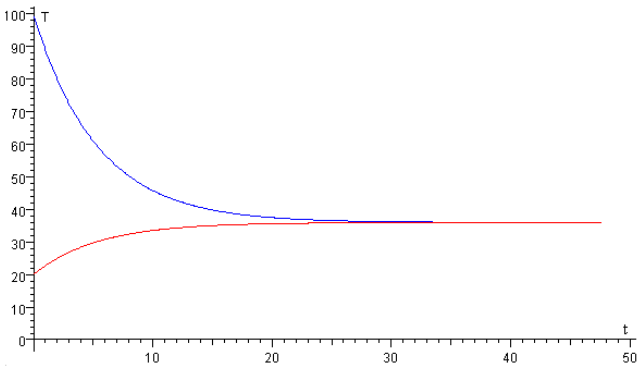

Figura 35: Representación grafica de la variación de temperatura
Rojo: Temperatura ambiente Azul: Temperatura del solido

5.1. PROBLEMAS DE VALOR INICIAL

El problema de valor inicial (PVI) consiste en obtener una solución $y(t)$ de una ecuación diferencial ordinaria de primer orden:

$$y'(t) = f(t, y(t)), \quad a \leq t \leq b,$$

que en el instante t=a toma un valor determinado y_a y que bajo ciertas condiciones se demuestra que tiene solución única. Dado el rectangulo $\mathrm{R} = (t, y) : a \leq t \leq b, c \leq y \leq d$ suponiendo la función continua en R se dice que la función f verifica una *Condición de Lipschitz* con respecto a su variable y en R si existe una constante L>0 tal que:

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|.$$

Para cualquier $(t, y_1), (t, y_2) \in$ R la constante **L** se llama **Constante de Lipschitz**, a continuación damos una condición para que $f(t, y)$ verifique una *Condición de Lipschitz* del tipo anterior:

Si $f(t, y)$ y $\frac{df(t,y)}{dy}$ son continuas para todo (t,y) $\in$ R, por el *Teorema del valor medio* se obtiene:

$$f(t, y_1) - f(t, y_2) = \frac{df(t, \hat{y})}{dy}(y_1 - y_2), \tag{-152}$$

con $y_1 < \hat{y} < y_2$. Entonces si existe $L = sup_{(t,y)\in D}|\frac{df(t,y)}{dy}|$ y además se cumple que $L > 0$ tal que $|f_y(t, y)| \leq L$ *para todo* $(t, y) \in$ R entonces f verifica una *Condición de Lipschitz* con respecto a la variable (y) en R siendo L su *Constante de Lipschitz*.

5.2. MÉTODOS DE UN PASO

La resolución de ecuaciones diferenciales mediante métodos numéricos se realiza por los llamados métodos de discretización que no es mas que los valores aproximados de la ecuación

diferencial en puntos t_k en el intervalo $[t_0, t_0 + \alpha]$, siendo $y(t_k)$ el valor exacto de la solución en el punto t_k e y_k será el valor aproximado de la solución que obtenemos cuando aplicamos un método numérico.

El esquema general es el siguiente:

1. Se inicia realizando una partición del intervalo $[t_0, t_0 + \alpha]$ en **N** partes iguales.

$$t_j = t_0 + jh,\ j = 0, 1, ..., N,\ h = \frac{\alpha}{N}.$$

2. Conocemos el valor inicial o condición inicial: $y(t_0) = y_0$.

3. El resto de $y_k, y_{k+1}...$ se calculan de forma progresiva en función de los k anteriores.

Llamaremos método de un paso a todo aquel algoritmo en el que a partir de t_k e y_k y de paso h calculamos el valor y_{k+1} que aproxima la solución del problema de valor inicial en el punto $t_k + h$. Esta solución se aproxima a partir de nodos (conjunto finito de puntos).

Se denomina método de paso simple, o de un solo paso, a los métodos de la forma $y_{k+1} = y_k + h\phi(t_k, y_k)$ siendo ϕ la función incremental interviniendo en el cálculo unicamente el punto inmediatamente anterior.
A la hora de resolver un problema de valor inicial mediante la aproximación por un método de variable discreta pueden ocurrir dos errores:

-**(Discretización global**: Si suponemos $[(t_k, y_k)]_{k=0}^{M}$ como un conjunto finito de aproximaciones a la solución única $y = y(t)$, definimos *el error de discretización global* (e_k) como:

$$e_k = y(t_k) - y_k \quad parak = 0, 1, ..., M. \tag{-153}$$

que es la diferencia entre la solución real exacta y la calculada en el nodo.

-**Discretización local**: también denominado de truncamiento local en el que ϵ_{k+1} queda definido como:

$$\epsilon_{k+1} = y(t_{k+1}) - y_k - h\phi(t_k, y_k) \; para \;\; k = 0, 1, ...M - 1, \tag{-153}$$

este error es el que se comete en un solo paso (El que nos lleva desde t_k hasta el nodo t_{k+1}).

El error lo consideramos el máximo entre los errores locales.

Por otro lado se dice que el método es *convergente* si el error tiende a 0 cuando reducimos el paso y *divergente* en caso contrario.

5.3. MÉTODO DE EULER

Debido a que no todos los problemas se pueden resolver directamente, debemos hallar fórmulas que representen la solución $y(t)$ en ciertos problemas de ingeniería y ciencia.

El *Método de Euler* aproxima la solución de forma bastante rudimentaria. No es muy utilizado en la práctica debido a la gran acumulación de errores que conlleva el proceso, pero sin embargo es útil estudiarlo ya que es muy sencillo analizar el error producido con este método.

Obtenemos el valor y_k de el valor anterior y_{k-1}. Siendo por tanto un método de un solo paso.

El proceso llevado a cabo en el *Método de Euler* consiste en dividir el intervalo $t_0, t_0 + \alpha$ en un número N de partes iguales:

$$t_1 = t_0 + h, \; t_2 = t_0 + 2h, ..., \;\; t_N = t_0 + Nh = t_0 + \alpha, \; h = \frac{\alpha}{N}.$$

Aplicando la definición de derivada:

$$y'(t_k) = \lim_{h \to \infty} \frac{y(t_k + h) - y(t_k)}{h}.$$

Y para un h lo suficientemente pequeño:

$$y'(t_k) = f(t_k, y(t_k)) \approx \frac{y(t_k + h) - y(t_k)}{h}.$$

Por lo que:

$$y(t_{k+1}) \approx y(t_k) + hf(t_k, y(t_k)), \quad k = 0, 1, ...M - 1.$$

Mediante la igualdad anterior, que nos sugiere el cálculo de los y_k mediante la ley de recurrencia, partiendo de la condición inicial $y_0 = y(0)$ y se obtiene el *Método de Euler*:

$$y_{k+1} = y_k + hf(t_k, y_k), \quad k = 0, 1, ...M - 1. \tag{-157}$$

5.4. PRECISIÓN DEL MÉTODO DE EULER

Si consideramos $y(t)$ la solución del problema de valor inicial, si $y(t) \in C^2[t_0, b]$ y $[(t_k, y_k)]_{k=0}^M$ es la sucesión de aproximaciones generadas por el *Método de Euler*:

$$|e_k| = |y(t_k) - y_k| = 0(h),$$

$$|\epsilon_{k+1}| = |y(t_{k+1}) - y_k - hf(t_k, y_k)| = 0(h^2).$$

Lo que consideramos como *Error Global Final* o el error final del intervalo viene dado por la siguiente expresión:

$$E(y(b), h) = |y(b) - y_N| = 0(h). \tag{-159}$$

Si las aproximaciones se calculan usando como pasos h y $h/2$:

$$E(y(b), h) \approx Ch,$$
$$E\left(y(b), \frac{h}{2}\right) \approx C\frac{h}{2} = \frac{1}{2}Ch \approx \frac{1}{2}E(y(b), h).$$

Lo que nos muestra que si reducimos a la mitad el tamaño del paso h en el *Método de Euler*, es de esperar que el error final se reduzca del mismo modo a la mitad.

5.4.1. Programación del método: *euler_explicito*(f, g, a, b, xa, ya, h)

El programa está diseñado para además de darnos los resultados de la aproximación, crear una gráfica en la que se ve la representación del valor real y de la aproximación. Partiendo de un punto (t_0, y_0) calculamos el valor de la pendiente $m_0 = f(t_0, y_0)$, y nos movemos una distancia $hf(t_0, y_0)$. La recta es tangente a la curva $y(t)$ concluyendo en el punto (t_1, y_1), que es una nueva aproximación, no un punto de la curva. Calculando la pendiente $m_1 = f(t_1, y_1)$, y obteniendo el desplazamiento $hf(t_1, y_1)$, esto nos llevará al punto (t_2, y_2) y así de forma sucesiva.

La programación del *Método de Euler* será:

```
function [T,Y,Ye,e]=euler_explicito(f,a,b,ya,h)

% Metodo de Euler explícito para el problema de valores iniciales
% y'=f(t,y), y(a)=y0
% [T,Y,Ye]=euler_explicito(f,a,b,ya,M);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente
% a y b son los extremos del intervalo
% ya es la condición inicial
% h es el valor de paso en cada iteración
% Datos de salida
% T es el vector de las abscisas de la solución
% Y es el vector de las ordenadas
% Ye valor de la solución exacta
% e diferencia entre la solución exacta y la aproximada en cada una de las
% iteraciones
% [T,Y,Ye,e]=euler_explicito('(t-y)/2',0,3,0,1/2)
```

```
% [T,Y,Ye,e]=euler_explicito('(exp(t^3/(1+y))-y^2)/2',0,3,0,1/2)



syms t y;

M=(b-a)/h;
T=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
Y(1)=ya;
for j=1:M
    Y(j+1)=Y(j)+h*subs(f,{t,y},{T(j),Y(j)});
end

% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(ya)];
S=dsolve(ecuacion,condicion_inicial);

if isempty(S)==0

    Ye(1)=ya;
    for j=2:M+1
        Ye(j)=subs(S,T(j));
    end


    e=(Ye-Y);

    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes

    h1=plot(T,Y,'o');
    hold on
    h2=ezplot(S,[a,b]);
    h3=plot(T,Ye,'ro');
    legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor exacto en los pun-
```

```
tos');

else
    e=[];
    Ye=[];
    fprintf('No existe una expresión exacta para la solucón');
    h1=plot(T,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

El resultado vendrá acompañado de una gráfica que muestra la aproximación entre el valor real y el obtenido mediante el método, apareciendo arriba la expresión de la función correspondiente, tal y como se ve en la Figura 36.

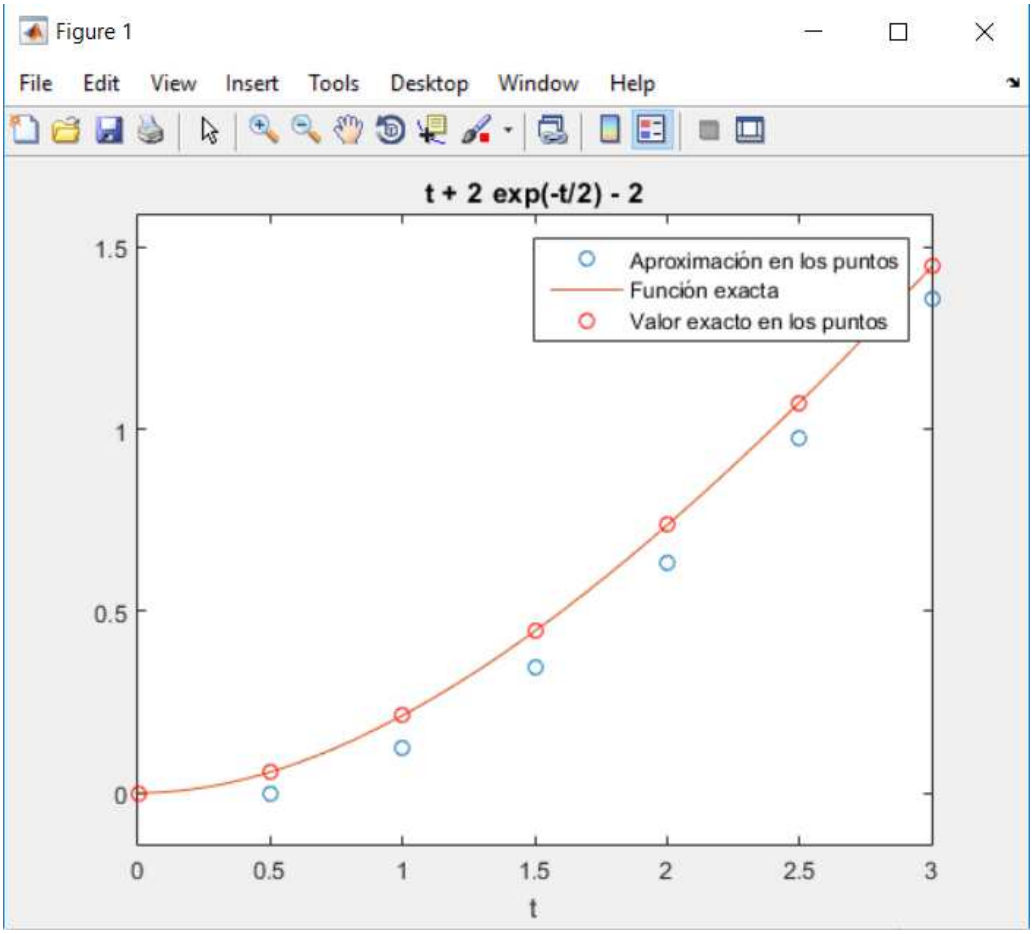

Figura 36: Curva de aproximación entre la solución exacta y las aproximaciones

5.5. MÉTODO DE TAYLOR

Consiste en un desarrollo de la función $y(t)$ en el punto t_k. Este método nos permite mejorar la aproximación de la solución del problema a medida que aumentamos el numero de terminos,.

$$y(t_{k+1}) = y(t_k) + hy'(t_k) + \frac{h^2}{2}y''(t_k) + ... + \frac{h^n}{n!}y^n(t_k) + \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(\xi_k),$$

con $\xi_k \in (t_k, t_k + h)$.

Por lo que el procedimiento básico consiste en la obtención de las derivadas sucesivas de y(t):

$$y'(t) = f(t, y(t)) = f^{(0)}(t, y),$$
$$y''(t) = \frac{dy'}{dt} = \frac{df}{dt} + \frac{df}{dy}\frac{dy}{dt} = \frac{df}{dt} + f\frac{df}{dy} = f^{(1)}(t, y),$$
$$y'''(t) = \frac{dy''}{dt} = \frac{df^{(1)}}{dt} + \frac{df^{(1)}}{dy}\frac{dy}{dt} = \frac{df^{(1)}}{dt} + f\frac{df^{(1)}}{dy} = f^{(2)}(t, y),$$
$$y^n(t) = \frac{dy^{(n-1)}}{dt} = \frac{df^{(n-2)}}{dt} + \frac{df^{(n-2)}}{dy}\frac{dy}{dt} = \frac{df^{(n-2)}}{dt} + f\frac{df^{(n-2)}}{dy} = f^{(n-1)}(t, y).$$

Cabe aclarar que el *Método de Euler* consiste en un *Método de Taylor* con $n = 1$.

Más adelante en el apartado [5.8],veremos como hemos realizado el programa de *Taylor* para orden n en el apartado .

5.5.1. Programación del método:$Taylor\_orden2(f, a, b, ya, h)$

Para el *Método de Taylor* primero hemos programado su caso básico que consiste en el método de orden $n = 2$, siendo su programación:

```
function [T,Y,Ye,e]=Taylor_orden2(f,a,b,ya,h)

% Metodo de Taylor de orden 2 para el problema de valores iniciales
% y'=f(t,y), y(a)=y0
% [T,Y,Ye]=Taylor_orden2(f,a,b,ya,M);
```

```
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente
% a y b son los extremos del intervalo
% ya es la condición inicial
% h es el valor de paso en cada iteración
% Datos de salida
% T es el vector de las abscisas de la solución
% Y es el vector de las ordenadas
% Ye valor de la solución exacta
% e diferencia entre la solución exacta y la aproximada en cada una de las
% iteraciones
% [T,Y,Ye,e]=Taylor_orden2('(t-y)/2',0,3,0,1/2)
% [T,Y,Ye,e]=Taylor_orden2('(exp(t^3/(1+y))-y^2)/2',0,3,0,1/2)



syms t y;

M=(b-a)/h;
T=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
Y(1)=ya;

% Segunda derivada de la solución y
y2=diff(f,t)+diff(f,y)*f;

for j=1:M
    Y(j+1)=Y(j)+h*subs(f,{t,y},{T(j),Y(j)})+h^2/2*subs(y2,{t,y},{T(j),Y(j)});
end


% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(ya)];
S=dsolve(ecuacion,condicion_inicial);

if isempty(S)==0
```

```
    Ye(1)=ya;
    for j=2:M+1
        Ye(j)=subs(S,T(j));
    end


    e=(Ye-Y);

    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes

    h1=plot(T,Y,'o');
    hold on
    h2=ezplot(S,[a,b]);
    h3=plot(T,Ye,'ro');
    legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor exacto en los puntos');

else
    e=[];
    Ye=[];
    fprintf('No existe una expresión exacta para la solucón');
    h1=plot(T,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

Del mismo modo que en *Euler* , hemos añadido al programa un código que imprima el resultado en forma de gráfica en la pantalla de modo que podamos ver la aproximación entre ambas.

5.6. MÉTODO DE TAYLOR ORDEN N

Nuestro proyecto en este campo se ha basado en la ampliación del *Método de Taylor* y su programación. Para ello hemos modificado el programa de *Taylor_ orden2* para adaptarlo y poder introducir como variable el numero de orden $n$de derivadas que queremos en la aproximación consiguiendo de este modo unos resultados más precisos, si bien también añadiendo mucho gasto computacional.

En realidad estos métodos no son muy utilizados para n mayor que dos.

En el caso de aumentar la n tenemos que tener en cuenta que se deben calcular más derivadas. Esto nos lleva a la creación de un bucle en el que a partir del valor de n escogido el programa te calcule las correspondientes derivadas:

$$y'(t) = f(t, y(t)) = f^{(0)}(t, y),$$
$$y''(t) = \frac{dy'}{dt} = \frac{df}{dt} + \frac{df}{dy}\frac{dy}{dt} = \frac{df}{dt} + f\frac{df}{dy} = f^{(1)}(t, y),$$
$$y'''(t) = \frac{dy''}{dt} = \frac{df^{(1)}}{dt} + \frac{df^{(1)}}{dy}\frac{dy}{dt} = \frac{df^{(1)}}{dt} + f\frac{df^{(1)}}{dy} = f^{(2)}(t, y),$$
$$y^n(t) = \frac{dy^{(n-1)}}{dt} = \frac{df^{(n-2)}}{dt} + \frac{df^{(n-2)}}{dy}\frac{dy}{dt} = \frac{df^{(n-2)}}{dt} + f\frac{df^{(n-2)}}{dy} = f^{(n-1)}(t, y).$$

El resto del programa sigue una estructura similar al de Taylor de orden 2.

5.6.1. Programación del Método: $Taylor\_ordenN(f, a, b, ya, h, n)$

```
function [T,Y,Ye,e]=Taylor_ordenN(f,a,b,ya,h,n)

% Metodo de Taylor de orden N para el problema de valores iniciales
% y'=f(t,y), y(a)=y0
% [T,Y,Ye]=Taylor_orden2(f,a,b,ya,M,n);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente
% a y b son los extremos del intervalo
% ya es la condición inicial
% h es el valor de paso en cada iteración
% n número de términos y orden global del método de Taylor de Orden N
% Datos de salida
% T es el vector de las abscisas de la solución
% Y es el vector de las ordenadas
% Ye valor de la solución exacta
% e diferencia entre la solución exacta y la aproximada en cada una de las
```

```
% iteraciones
% [T,Y,Ye,e]=Taylor_ordenN('(t-y)/2',0,3,0,1/2,4)
% [T,Y,Ye,e]=Taylor_ordenN('(exp(t^3/(1+y))-y^2)/2',0,3,0,1/2,4)



syms t y;

M=(b-a)/h;
T=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
Y(1)=ya;

% Calculamos las derivadas necesarias

d{1}=f;

for i=2:n
    d{i}=diff(d{i-1},t)+diff(d{i-1},y)*f;
end



for j=1:M
    Y(j+1)=Y(j)+h*subs(f,{t,y},{T(j),Y(j)});

    for i=2:n
        Y(j+1)=Y(j+1)+h^i/factorial(i)*subs(d{i},{t,y},{T(j),Y(j)});
    end
end


% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(ya)];
S=dsolve(ecuacion,condicion_inicial);

if isempty(S)==0
```

```
    Ye(1)=ya;
    for j=2:M+1
        Ye(j)=subs(S,T(j));
    end


    e=(Ye-Y);

    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes

    h1=plot(T,Y,'o');
    hold on
    h2=ezplot(S,[a,b]);
    h3=plot(T,Ye,'ro');
    legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor exacto en los puntos');

else
    e=[];
    Ye=[];
    fprintf('No existe una expresión exacta para la solucón');
    h1=plot(T,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

5.7. MÉTODO DE RUNGE-KUTTA

Construidos a partir del *Método de Taylor* de orden n, explicado previamente, y diseñado de tal manera que el error global final sea de orden $0(h^n)$, pero se evite la evaluación de derivadas parciales, evaluando la función en varios puntos en su lugar.

Partimos de la ecuación diferencial:

$$\begin{cases} y' = f(t,y), \\ y(t_0) = y_0. \end{cases}$$

La forma general de expresar el *Método de Runge-Kutta* explícito de m etapas es:

$$y_k = y_{k-1} + \sum_{j=1}^{m} b_j g_j,$$

donde:

$$\begin{cases} g_1 = hf(t_{k-1}, yk-1), \\ g_2 = hf(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1), \\ g_3 = hf(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} g_2), \\ ... \\ g_m = hf(t_{k-1} + c_m h, y_{k-1} + a_{m1} g_1 + a_{m2} g_2 + ... + a_{m1} g_{m-1}), \end{cases}$$

siendo $c_j,\ j = 2, ..., m, b_j = 1, ..., m\ y\ a_{jk}, j = 1..., m, k = 1, ..., j-1$ los coeficientes del método.

Normalmente estos coeficientes los agruparemos en una tabla:

$$\begin{array}{c|ccccc} 0 & & & & & \\ C_2 & a_{21} & & & & \\ C_3 & a_{31} & a_{32} & & & \\ \cdots & \cdots & \cdots & \cdots & & \\ C_m & a_{m1} & a_{m2} & \cdots & a_{mm-1} & \\ \hline & b_1 & b_2 & \cdots & b_{m-1} & b_m \end{array}$$

Esta tabla se denomina la *tabla de Butcher*, y en su forma matricial se representa así:

$$\begin{array}{c|c} c^t & A \\ \hline & b \end{array}$$

Donde $c = (0, c_2, ..., c_m),\ b = (b_1, b_2, ..., b_m)$ y $A = (a_{jk}),\ a_{jk} = 0\ si\ k > j$. Satisfaciendo en general las condiciones de simplificación.

$$c_j = \sum_{k=1}^{j-1} a_{jk}, \quad j = 2, ..., m. \tag{-175}$$

Dependiendo de la cantidad de filas N que realicemos en la *tabla de Butcher* el *Método de Runge-Kutta* sera de N etapas, siendo las que analizaremos las de 3 y 4 etapas.

5.7.1. Método de 3 etapas

En el Método de 3 etapas el valor del error de truncamiento será de $0(h^4)$ y su *Tabla de Butcher* será:

$$\begin{array}{c|ccc} 0 & & & \\ C_2 & a_{21} & & \\ C_3 & a_{31} & a_{32} & \\ \hline & b_1 & b_2 & b3 \end{array}$$

Y el método será de la siguiente forma:

$$\begin{cases} g_1 = hf(t_{k-1}, yk - 1), \\ g_2 = hf(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1), \\ g_3 = hf(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} g_2). \end{cases}$$

Y por tanto y_k:

$$y_k = y_{k-1} + b_1 g_1 + b_2 g_2 + b_3 g_3.$$

La aproximación de *Taylor de orden 3* de $y(t; t_{k-1}, y_{k-1})$ es:

$$y_k = y_{k-1} + f(t_{k-1}, y_{k-1})h + \frac{1}{2}(\frac{d}{dt}f(t_{k-1}, y_{k-1})\frac{d}{dy}f(t_{k-1}, y_{k-1})h^2$$
$$+\frac{1}{6}\left(\frac{d^2}{dt^2}f(t_{k-1}, y_{k-1}) + 2f(t_{k-1}, y_{k-1})\frac{d^2}{dydt}f(t_{k-1}, y_{k-1})\right.$$
$$+\frac{d}{dt}f(t_{k-1}, y_{k-1})\frac{d}{dy}f(t_{k-1}, y_{k-1}) + f(t_{k-1}, y_{k-1})\left(\frac{d}{dy}ft(t_{k-1}, y_{k-1}\right)^2$$
$$\left.f(t_{k-1}, y_{k-1})^2\frac{d^2}{dy^2}f(t_{k-1}, y_{k-1})\right)h^3 + 0(h^4), \qquad \text{(-181)}$$

siendo el último valor el correspondiente al error cometido por el método.

Tomamos la función:

$$G_2(h) = f(t_{k-1} + c_2h, y_{k-1} + a_{21}g_1), \qquad \text{(-181)}$$

derivamos dos veces respecto de h:

$$G_2'(h) = c_2\frac{d}{dt}f(t_{k-1} + c_2h, y_{k-1} + a_{21}g1)+$$
$$\frac{d}{dy}f(t_{k-1} + c_2h, y_{k-1} + a_{21}g1)a_{21}f(t_{k-1}, y_{k-1})$$

y la segunda derivada:

$$G_2''(h) = c_2^2\frac{d^2}{dt^2}f(t_{k-1} + c_2h, y_{k-1} + a_{21}g1)+$$
$$2c_2\frac{d^2}{dt^2}f(t_{k-1} + c_2h, y_{k-1} + a_{21}g1)a_{21}f(t_{k-1}, y_{k-1})$$
$$\frac{d^2}{dt^2}f(t_{k-1} + c_2h, y_{k-1} + a_{21}g1)a_{21}^2f(t_{k-1}, y_{k-1})^2.$$

Mientras que el desarrollo en Taylor sería:

$$G_2(h) = G_2(0) + G_2'(0)h + \frac{1}{2}G_2''(0)h^2 + 0(h^3) =$$

$$f(t_{k-1}, y_{k-1}) + \left(c_2 \frac{d}{dt} f(t_{k-1}, y_{k-1}) \frac{d}{dy} f(t_{k-1}, y_{k-1}) \right) h$$

$$+\frac{1}{2} \left(c_2^2 \frac{d^2}{dt^2} f(t_{k-1}, y_{k-1}) 2c_2 a_{21} f(t_{k-1}, y_{k-1}) \frac{d^2}{dtdy} f(t_{k-1}, y_{k-1}) + \right.$$

$$\left. a_{21}^2 f(t_{k-1}, y_{k-1})^2 \frac{d^2}{dy^2} f(t_{k-1}, y_{k-1}) \right) h^2 + 0(h^3).$$

Y por otro lado:

$$G_3(h) = f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)). \tag{-190}$$

Derivando 2 veces:

$$G_3'(h) = c_3 \frac{d}{dt} f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g1 + a_{32} h G_2(h)) +$$

$$\frac{d}{dy} f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g1 + a_{32} h G_2(h))(a_{31} f(t_{k-1}, y_{k-1}) + a_{32}(G_2(h) + h G_2'(h))).$$

Y la segunda derivada:

$$G_3''(h) = c_3^2 \frac{d^2}{dt^2} f(t_{k-1} + c_2 h, y_{k-1} + a_{31} g1 + a_{32} h G_2(h)) +$$

$$2c_3 \frac{d^2}{dtdy} f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g1 + a_{32} h G_2(h))(a_{21} f(t_{k-1}, y_{k-1}) + a_{32}(G_2(h) + h G_2'(h)))$$

$$\frac{d^2}{dt^2} f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g1 + a_{32} G_2'(h))(a_{31} f(t_{k-1}, y_{k-1})^2 + a_{32}(G_2'(h)))^2$$

$$+\frac{d^2}{dt^2} f(t_{k-1} + c_3 h, y_{k-1} + a_{31} g1 + a_{32} G_2'(h)) \left(a_{32}(2G_2'(h) + h G_2''(h)) \right).$$

De donde:

$$G_3'(0) = c_3 \frac{d}{dt} f(t_{k-1}, y_{k-1}) + (a_{31} + a_{32}) \frac{d}{dy} f(t_{k-1}, y_{k-1}) f(t_{k-1}, yk - 1).$$

y la segunda derivada:

$$G_3''(0) = c_3^2 \frac{d^2}{dt^2} f(t_{k-1}+, y_{k-1} + 2c_3(a_{31} + a_{32}) \frac{d^2}{dtdy} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})$$

$$+(a_{31} + a_{32})^2 \frac{d^2}{dy^2} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})^2$$

$$2a_{32} \frac{d}{dy} f(t_{k-1}, y_{k-1}) \left(c_2 \frac{d}{dt} f(t_{k-1}, yk - 1) + a_{21} f(t_{k-1}, y_{k-1}) \frac{d}{dy} f(t_{k-1}, y_{k-1}) \right). \qquad \text{(-199)}$$

Por lo que el desarrollo de *Taylor* en cero es:

$$G_3(h) = G_3(0) + G_3'(0)h + \frac{1}{2} G_3''(0)h^2 + 0(h^3)$$

$$= f(t_{k-1}, yk - 1) + \left(c_3 \frac{d}{dt} f(t_{k-1}, y_{k-1}) + 2c_3(a_{31} + a_{32}) \frac{d^2}{dtdy} f(t_{k-1}, yk - 1) f(t_{k-1}, y_{k-1}) \right.$$

$$\left. (c_2 \frac{d}{dy} f(t_{k-1}, y_{k-1}) + a_{21} f(t_{k-1}, y_{k-1}) \frac{d}{dy} f(t_{k-1}, y_{k-1}) \right) h^2. \qquad \text{(-201)}$$

Si introducimos los desarrollos de *Taylor* G_2 y G_3 en la igualdad (5.7.1) y realizando la comparación con el desarrollo de *Taylor de orden 2* expresado en la fórmula (5.7.1), obtenemos las siguientes ecuaciones:

$$
\begin{aligned}
b_1 + b_2 + b_3 &= 1, \\
b_2 c_2 + b_3 c_3 &= \frac{1}{2}, \\
b_2 a_{21} + b_3 (a_{31} + a_{32}) &= \frac{1}{2}, \\
b_2 c_2^2 + b_3 c_3^2 &= \frac{1}{3}, \\
b_2 c_2 a_{21} + b_3 (a_{31} + a_{32})^2 &= \frac{1}{3}, \\
b_3 a_{32} c_2 &= \frac{1}{6}, \\
b_3 a_{32} a_{21} &= \frac{1}{6}.
\end{aligned}
$$

A través de estas expresiones deducimos que $c_2 = a_{21}$ y que $c_3 = a_{31} + a_{32}$, simplificando las ecuaciones 5^{a} y 6^{a}:

$$
b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}.
$$

Reduciendo el sistema con estas sustituciones nos quedaría:

$$
\begin{aligned}
b_1 + b_2 + b_3 &= 1, \\
c_2 &= a_{21}, \\
c_3 &= a_{31} + a_{32}, \\
b_2 c_2 + b_3 c_3 &= \frac{1}{2}, \\
b_2 c_2^2 + b_3 c_3^2 &= \frac{1}{3}, \\
b_3 a_{32} c_2 &= \frac{1}{6}.
\end{aligned}
$$

Lo cual nos da los *Métodos de Runge-Kutta* de orden 3, que forman una familia biparamétrica de métodos numéricos. Si tomamos c_2 y c_3 como parámetros, tenemos las siguientes ecuaciones;

$$b_2 c_2 + b_3 c_3 = \frac{1}{2},$$

$$b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}.$$

Que da pie a:

$$b_2 = \frac{c_3 \left(\frac{c_3}{2} - \frac{1}{3}\right)}{c_2 c_3 (c_3 - c_2)},$$

$$b_3 = \frac{c_2 \left(\frac{1}{3} - \frac{c_2}{2}\right)}{c_2 c_3 (c_3 - c_2)}.$$

Y de la última:

$$a_{32} = \frac{1}{6 b_3 c_2} = \frac{c_2 c_3 (c_3 - c_2)}{6 \left(\frac{1}{3} - \frac{c_2}{2}\right)}.$$

Y como $b_1 + b_2 + b_3 = 1$ concluimos que:

$$b_1 = 1 - b_2 - b_3 = 1 - \frac{c_3 \left(\frac{c_3}{2} - \frac{1}{3}\right)}{c_2 c_3 (c_3 - c_2)} - b_3 = \frac{c_2 \left(\frac{1}{3} - \frac{c_2}{2}\right)}{c_2 c_3 (c_3 - c_2)} = \frac{\left(c_2 c_3 + \frac{1}{3}\right)(c_3 - c_2) - \frac{1}{2}(c_3^2 - c_3^2)}{c_2 c_3 (c_3 - c_2)}.$$

De este modo obtenemos Runge-Kutta de orden 3:

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 3/4 & 0 & 3/4 & \\ \hline & 2/9 & 1/3 & 4/9 \end{array}$$

O bien:

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array} \tag{-222}$$

Estos soluciones serán validas siempre y cuando se cumpla que c_2 y c_3 sean no nulos y diferentes. Si se da el caso deberíamos utilizar otros métodos distintos, como observamos, el error local del *Método de Runge-Kutta de 3 etapas* es de orden $0(h^4)$ y por lo tanto el error global es 3.

5.7.2. Método de 4 etapas

En el caso del Método para 4 etapas seguiremos el mismo procedimiento que en el caso anterior de 3 etapas, solo que en este caso debemos aumentar el orden de los desarrollos del *Método de Taylor* de las funciones implicadas, siendo la *tabla de Butcher* para este caso:

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ c_4 & a_{41} & a_{42} & a_{43} & \\ \hline & b_1 & b_2 & b_3 & b_4 \end{array}$$

Satisfaciendo la simplicación:

$$c_j = \sum_{k=1}^{j-1} a_{jk},\ j = 2, 3, 4. \tag{-223}$$

Y con las condiciones:

$$
\begin{aligned}
b_1 + b_2 + b_3 + b_4 &= 1, \\
b_2c_2 + b_3c_3 + b_4c_4 &= \frac{1}{2}, \\
b_2c_2^2 + b_3c_3^2 + b_4c_4^2 &= \frac{1}{3}, \\
b_3a_{32}c_2 + b_4(a_{42}c_2 + a_{43}c_3) &= \frac{1}{6}, \\
b_2c_2^3 + b_3c_3^3 + b_4c^3 &= \frac{1}{4}, \\
b_3c_3a_{32}c_2 + b_4c_4(a_{42}c_2 + a_{43}c_3)^2 &= \frac{1}{8}, \\
b_3a_{32}c_2^2 + b_4(a_{42}c_2^2 + a_{43}c_3^2) &= \frac{1}{12}, \\
b_4a_{43}a_{32}c_2 &= \frac{1}{24}.
\end{aligned}
$$

A través de la simplificación de *Butcher* podemos expresar la primera condición como:

$$\sum_{i=1}^{4} b_i a_{ij} = b_j(1 - c_j),\ j = 2, 3, 4. \tag{-231}$$

Siendo los elementos de la ecuación (**??**) $a_{ij} = 0\ si\ i \geq j$, de la cual se tiene para $j = 4$ que $b_4(1 - c_4) = 0$ donde $c_4 = 1$ ya que $b \neq 0$ y como las ecuaciones 2^{a}, 3^{a} y 5^{a} son lineales en $b_2, b_3\ y\ b_4$ tenemos que:

$$b_2 = \frac{1 - 2c_3}{12c_2(1 - c_2)(c_2 - c_3)},$$

$$b_3 = \frac{1 - 2c_2}{12c_3(1 - c_3)(c_3 - c_2)},$$

$$b_4 = \frac{6c_2c_3 - 4(c_3 + c_2) + 3}{12c_2(1 - c_2)(1 - c_3)}.$$

Tomando ahora $j = 3$ en (5.7.2)

$$a_{42} = \frac{(1-c_2)[2(1-c_3)(1-2c_3)(1-2c_3)-(c_2-c_3)]}{2c_2(c_2-c_3)[6c_2c_3-4(c_2+c_3)+3]}.$$

Debemos establecer la condición de que $c_2 \not\in \left[0, \frac{1}{2}, 1\right]$, $c_3 \not\in [0,1]$ y $c_2 \neq c_3$. En el caso de que no se satisfagan estas condiciones existen otros métodos alternativos para el cálculo de los coeficientes.

Estos métodos son de orden 4 (orden 5 tiene el error local de truncamiento) y representa una familia de métodos de 5 parámetros.

Por ejemplo:

0				
1/3	1/3			
2/3	−1/3	1		
1	1	−1	1	
	1/8	3/8	3/8	1/8

Y cumple $c_2 = c_3$

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Siendo este el *Método de Runge Kutta*.

Para conseguir las sucesivas aproximaciones realizamos la siguiente sustitución:

$$y_{k+1} = y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{-237}$$

donde:

$$
\begin{aligned}
k_1 &= f(t_k, y_k), \\
k_2 &= f(t_k + \frac{h}{2}, y_k + k_1\frac{h}{2}), \\
k_3 &= f(t_k + \frac{h}{2}, y_k + k_2\frac{h}{2}), \\
k_4 &= f(t_k + h, y_k + k_3 h).
\end{aligned}
$$

La ventaja de este método es que posibilita obtener resultados con precisión alta sin tener que tomar el paso h muy pequeño.

5.7.3. Programación del Metodo: *RKCuatro(f, a, y0, h, n)*

La programación de este método la hemos basado en $Runge_Kutta$ 4 y su programación en *Matlab* será:

```
function [Y,Ye,e]=RKCuatro(f,a,y0,h,n)

% Programa para la resolución numérica de EDOs
% mediante el método de Runge-Kutta Cuatro
% y(n+1)=y(n)+1/6*(k1+2k2+2k3+k4)
% k1=h*f(t(n),y(n))
% k2=h*f(t(n)+1/2h,y(n)+1/2k1)
% k3=h*f(t(n)+1/2h,y(n)+1/2k2)
% k4=h*f(t(n)+h,y(n)+k3)
% El problema a estudio es
% y'(t)= f(t,y)
% y(a)=y0
% [Y,Ye]=RKCuatro(f,a,y0,h,n)
% Datos de Entrada
% f cadena de caracteres identificando la función
% a tiempo inicial
% y0 condición inicial
% h paso temporal
% n número de pasos temporales a realizar
% Datos de Salida
```

```
% Y aproximación a la solución para los tiempos desde a hasta a+nh con paso
% h
% Ye solución discretizada para los tiempos desde a hasta a+nh con paso
% h
% e error en la aproximación para los tiempos desde a hasta a+nh con paso h
% Ejemplo:
% Resolver y'(t)= exp(-2*t)-2*y, y(0)=1/10 con paso h=0.8, y haced 2 pasos de tiempo
% [Y,Ye,e]=RKCuatro('exp(-2*t)-2*y',0,1/10,0.8,2)
% En esta ecuación podemos apreciar el error cometido



syms t y

tiempo(1)=a; % tiempo inicial

% Inicialización y=y0
Y(1)=y0;

% bucle que realiza los sucesivos pasos temporales

for i=1:n
    % siguiente paso temporal
    tiempo(i+1)=a+i*h;
    % cálculo de k1, k2, k3, k4
    k1=h*subs(f,{t,y},{tiempo(i),Y(i)});
    k2=h*subs(f,{t,y},{tiempo(i)+h/2,Y(i)+k1/2});
    k3=h*subs(f,{t,y},{tiempo(i)+h/2,Y(i)+k2/2});
    k4=h*subs(f,{t,y},{tiempo(i)+h,Y(i)+k3});
    % aproximación de Runge-Kutta 4
    Y(i+1)=Y(i)+1/6*(k1+2*k2+2*k3+k4);

end


% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(y0)];
S=dsolve(ecuacion,condicion_inicial);
```

```
if isempty(S)==0
    Ye(1)=y0;
    for j=2:n+1
        Ye(j)=subs(S,tiempo(j));
    end



    % Imprimimos en pantalla los diferentes tiempos, la solución exacta, la
    % aproximada, y el error
    e=Ye'-Y';


    % Dibujamos en pantalla la solución aproximada y la real con
    % colores diferentes

    h1=plot(tiempo,Y,'o');
    hold on
    h2=ezplot(S,[a,a+h*n]);
    h3=plot(tiempo,Ye,'ro');
    legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor exacto en los puntos');
else
    Ye=[];
    frpintf('No existe una expresión exacta para la solución');
    h1=plot(tiempo,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

Mostrando en una gráfica auxiliar la aproximación de nuestro valor y el valor real.

5.8. SISTEMAS DE ECUACIONES DIFERENCIALES

Aunque los apartados que hemos visto anteriormente son referidos a problemas iniciales de primer orden con una única variable independiente, muchos casos de la vida real utilizan ecuaciones diferenciales ordinarias de orden superior y sistemas de ecuaciones diferenciales. Por ejemplo en muchas ocasiones aparece la ecuación de segundo orden:

$$\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right). \tag{-241}$$

El proceso que se sigue en la mayoría de los casos es la transformación de ecuaciones de mayor orden en sistemas de ecuaciones diferenciales de primer orden.

La ecuación (5.8) se transforma en un sistema de primer orden mediante la sustitución:

$$y' = x, \quad x' = f(t, x, y).$$

El objetivo de los programas de sistemas es aproximar el resultado de ecuaciones del tipo:

$$\begin{cases} x' = f(t, x, y), \\ y' = g(t, x, y). \end{cases}$$

Estableciendo como condiciones iniciales $x(t_0) = x_0$ e $y(t_0) = y_0$ y compartiendo ambas el mismo intervalo [a,b].

Un problema que se nos puede presentar es la regularidad de las funciones f y g. Nosotros las consideramos lo suficientemente regulares para que el problema tenga una única solución. Las variables independientes en este caso son x e y, y la variablle independiente es t. Mediante los métodos numéricos que vamos a presentar se van a hallar aproximaciones a $x(t)$ $y(t)$ en los puntos $t = t_0,\ t = t_1 = t_0 + h, ..., t_N = t_0 + Nh$.

Además el programa nos dará la solución de forma gráfica de ambas variables, y muestran la diferencia producida con la solución exacta, de existir está.

La programación de los métodos para sistemas será la siguiente.

5.8.1. Programación del Método: $euler\_explicito\_sistemas(f, g, a, b, xa, ya, h)$

```
function [T,X,Y,Xe,Ye]=euler_explicito_sistemas(f,g,a,b,xa,ya,h)
```

```
%function [T,X,Xe,Y,Ye]=euler_explicito_general_sistemas(f,g,a,b,xa,ya,h)
% Metodo de Euler explícito para sistemas. El problema de valores iniciales
% x'=f(t,x,y), x(a)=xa
% y'=g(t,x,y), y(a)=ya
% [T,X,Y,Xe,Ye]=euler_explicito_general_sistemas(f,g,a,b,xa,ya,h);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente de la primera
% ecuacion
% g cadena de caracteres identificando el termino fuente de la segunda
% ecuacion
% a y b son los extremos del intervalo
% xa condicion inicial en la variable x
% ya es la condición inicial en la variable y
% h es el tamaño de paso de cada aproximación
% Datos de salida
% T es el vector de las abscisas de la solución
% X es el vector de las ordenadas en la variable x
% Xe valor de la solución exacta para la variable x
% Y es el vector de las ordenadas en la variable y
% Ye valor de la solución exacta para la variable y
% Ejemplo1:
% [T,X,Y,Xe,Ye]=euler_explicito_sistemas('x+y-7','3*x+2*y+5',0,1,2,3,0.2)
% Ejemplo2, sin solución explícita.
% [T,X,Y,Xe,Ye]=euler_explicito_sistemas('x*y+t','x-t',0,1,0,1,0.2)


syms t x y;



M=(b-a)/h;
T=zeros(1,M+1);
X=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
X(1)=xa;
Y(1)=ya;
for j=1:M
    X(j+1)=X(j)+h*subs(f,{t,x,y},{T(j),X(j),Y(j)});
```

```
    Y(j+1)=Y(j)+h*subs(g,{t,x,y},{T(j),X(j),Y(j)});
end

 % Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(xa)];
condicion_inicial2=['y(',num2str(a),')=',num2str(ya)];



S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);
if isempty(S)==0

    Xe(1)=xa;
    Ye(1)=ya;
    for j=2:M+1
        Xe(j)=subs(S(1).x,{t},{T(j)});
        Ye(j)=subs(S(1).y,{t},{T(j)});
    end

    % Calculamos los errores cometidos en cada aproximación.
    ex=(Xe-X);
    ey=(Ye-Y);


    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes

    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    hold on

    h2=plot(T,Xe,'ro');
    legend([h1,h2],'Aproximación en los puntos','Valor exacto en los puntos');


    figure(2)
```

```
    h4=plot(T,Y,'o');
    title('Variable y')
    hold on
    h5=plot(T,Ye,'ro');
    legend([h4,h5],'Aproximación en los puntos','Valor exacto en los puntos');

else
    Xe=[];
    Ye=[];
    fprintf('No existe solución exacta para la ecuación diferencial');
    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    legend([h1],'Aproximación de los puntos');

    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    legend([h4],'Aproximación de los puntos');
end
```

Final del programa Matlab

5.8.2. Programación del Método: $RKCuatro\_sistemas(f, g, a, x0, y0, h, n)$

```
    function [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas(f,g,a,x0,y0,h,n)

% Programa para la resolución numérica de sistemas de EDOs
% mediante el método de Runge-Kutta Cuatro
% x(n+1)=x(n)+1/6*(xk1+2xk2+2xk3+xk4)
%  % xk1=h*f(t(n),x(n))
%  % xk2=h*f(t(n)+1/2h,x(n)+1/2xk1)
%  % xk3=h*f(t(n)+1/2h,x(n)+1/2xk2)
%  % xk4=h*f(t(n)+h,x(n)+xk3)
% y(n+1)=y(n)+1/6*(k1+2k2+2k3+k4)
%  % k1=h*f(t(n),y(n))
%  % k2=h*f(t(n)+1/2h,y(n)+1/2k1)
%  % k3=h*f(t(n)+1/2h,y(n)+1/2k2)
```

```
% % k4=h*f(t(n)+h,y(n)+k3)
% El problema a estudio es
% x'(t)= f(t,y)
% y'(t)= g(t,y)
% x(a)=x0, y(a)=y0
% [X,Xe,Y,Ye,ex,ey]=RKCuatroSistemas(f,g,a,x0,y0,h,n)
% Datos de Entrada
% f cadena de caracteres identificando la función (x)
% g cadena de caracteres identificando la función (y)
% a tiempo inicial
% x0 condición inicial para x
% y0 condicion inicial para y
% h paso temporal
% n número de pasos temporales a realizar
% Datos de Salida
% X aproximación a la solución para tiempos desde a hasta a+nh con paso h
% Y aproximación a la solución para tiempos desde a hasta a+nh con paso h
% Xe solución discretizada para tiempos desde a hasta a+nh con paso h
% Ye solución discretizada para tiempos desde a hasta a+nh con paso h
% ex error en la aproximación para los tiempos desde a hasta a+nh con paso h
% ey error en la aproximación para los tiempos desde a hasta a+nh con paso h
% Ejemplo 1. Sistema sin solución explícita.
% Resolver x'=x*y+t, y'(t)=x-t, x(0)=0, y(0)=1 con paso h=0.1, y haced 2 pasos de tiempo
% [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas('x*y+t','x-t',0,0,1,0.1,2)
% Ejemplo 2. Sistema con solución explicita.
% [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas('x+y-7','3*x+2*y+5',0,2,3,0.1,10)



syms t y x


tiempo(1)=a; % tiempo inicial

% Inicialización
Y(1)=y0;
X(1)=x0;

% bucle que realiza los sucesivos pasos temporales
```

```
for i=1:n
    % siguiente paso temporal
    tiempo(i+1)=a+i*h;

    % cálculo de k1, k2, k3, k4
    k1x=h*subs(f,{t,x,y},{tiempo(i),X(i),Y(i)});
    k1y=h*subs(g,{t,x,y},{tiempo(i),X(i),Y(i)});
    k2x=h*subs(f,{t,x,y},{tiempo(i)+h/2,X(i)+k1x/2,Y(i)+k1y/2});
    k2y=h*subs(g,{t,x,y},{tiempo(i)+h/2,X(i)+k1x/2,Y(i)+k1y/2});
    k3x=h*subs(f,{t,x,y},{tiempo(i)+h/2,X(i)+k2x/2,Y(i)+k2y/2});
    k3y=h*subs(g,{t,x,y},{tiempo(i)+h/2,X(i)+k2x/2,Y(i)+k2y/2});
    k4x=h*subs(f,{t,x,y},{tiempo(i)+h,X(i)+k3x,Y(i)+k3y});
    k4y=h*subs(g,{t,x,y},{tiempo(i)+h,X(i)+k3x,Y(i)+k3y});

    % aproximación de Runge-Kutta 4
    X(i+1)=X(i)+1/6*(k1x+2*k2x+2*k3x+k4x);
    Y(i+1)=Y(i)+1/6*(k1y+2*k2y+2*k3y+k4y);
end


% Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(x0)];
condicion_inicial2=['y(',num2str(a),')=',num2str(y0)];

S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);
if isempty(S)==0

    Xe(1)=x0;
    Ye(1)=y0;
    for i=2:n+1
        Xe(i)=subs(S(1).x,{t},{tiempo(i)});
        Ye(i)=subs(S(1).y,{t},{tiempo(i)});
    end



    % Error cometido
```

```
    ex=abs(Xe-X);
    ey=abs(Ye-Y);



    % Dibujamos en pantalla la solución aproximada y la real con
    % colores diferentes
    figure(1)
    h1=plot(tiempo,X,'o');
    title('Variable x')
    hold on
    h2=plot(tiempo,Xe,'ro');
    legend([h1,h2],'Aproximación obtenida','Valor exacto');
    hold off

    figure(2)
    h3=plot(tiempo,Y,'o');
    title('Variable y')
    hold on
    h4=plot(tiempo,Ye,'ro');
    legend([h3,h4],'Aproximación obtenida','Valor exacto');
    hold off
else
    Xe=[];
    Ye=[];
    ex=[];
    ey=[];
    fprintf('No existe solución exacta del problema')
    figure(1)
    h1=plot(tiempo,X,'o');
    title('Variable x')
    legend([h1],'Aproximación obtenida');

    figure(2)
    h3=plot(tiempo,Y,'o');
    title('Variable y')
    legend([h3],'Aproximación obtenida');
end
```

Final del programa Matlab

5.8.3. Programación del Método: $Taylor\_orden2\_sistemas(f, g, a, b, xa, ya, h)$

```
function [T,X,Y,Xe,Ye]=Taylor_orden2_sistemas(f,g,a,b,xa,ya,h)

% Metodo de Taylor de orden 2 para sistemas. El problema de valores
% iniciales
% x'=f(t,x,y), x(a)=xa
% y'=g(t,x,y), y(a)=ya
% [T,Y,Ye]=Taylor_orden2_sistemas(f,g,a,b,xa,ya,h);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente de la primera
% ecuacion
% g cadena de caracteres identificando el termino fuente de la segunda
% ecuacion
% a y b son los extremos del intervalo
% xa condicion inicial en la variable x
% ya es la condición inicial en la variable y
% h es el tamaño de paso de cada aproximación
% Datos de salida
% T es el vector de las abscisas de la solución
% X es el vector de las ordenadas en la variable x
% Xe valor de la solución exacta para la variable x
% Y es el vector de las ordenadas en la variable y
% Ye valor de la solución exacta para la variable y
% Ejemplo1:
% [T,X,Y,Xe,Ye]=Taylor_orden2_sistemas('x+y-7','3*x+2*y+5',0,1,2,3,0.2)
% Ejemplo2, sin solución explícita.
% [T,X,Y,Xe,Ye]=Taylor_orden2_sistemas('x*y+t','x-t',0,1,0,1,0.2))


syms t x y;

M=(b-a)/h;
T=zeros(1,M+1);
X=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
```

```
X(1)=xa;
Y(1)=ya;

% Segunda derivada de la solución y
x2=diff(f,t)+diff(f,x)*f+diff(f,y)*g;
y2=diff(g,t)+diff(g,x)*f+diff(g,y)*g;

for j=1:M
    X(j+1)=X(j)+h*subs(f,{t,x,y},{T(j),X(j),Y(j)})+h^2/2*subs(x2,{t,x,y},{T(j),X(j),Y(j)});
    Y(j+1)=Y(j)+h*subs(g,{t,x,y},{T(j),X(j),Y(j)})+h^2/2*subs(y2,{t,x,y},{T(j),X(j),Y(j)});
end


% Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(xa)];
condicion_inicial2=['y(',num2str(a),')=',num2str(ya)];


S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);

if isempty(S)==0

    Xe(1)=xa;
    Ye(1)=ya;
    for j=2:M+1
        Xe(j)=subs(S(1).x,{t},{T(j)});
        Ye(j)=subs(S(1).y,{t},{T(j)});
    end

    % Calculamos los errores cometidos en cada aproximación.
    ex=(Xe-X);
    ey=(Ye-Y);


    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes
```

```
    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    hold on

    h2=plot(T,Xe,'ro');
    legend([h1,h2],'Aproximación en los puntos','Valor exacto en los puntos');


    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    hold on
    h5=plot(T,Ye,'ro');
    legend([h4,h5],'Aproximación en los puntos','Valor exacto en los puntos');

else
    Xe=[];
    Ye=[];
    fprintf('No existe solución exacta para la ecuación diferencial');
    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    legend([h1],'Aproximación de los puntos');

    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    legend([h4],'Aproximación de los puntos');
end
```

Final del programa Matlab

5.8.4. Programación del Método: $Taylor\_ordenN\_sistemas(f, a, b, ya, h, n)$

```
    function [T,X,Y,Xe,Ye]=Taylor_ordenN_sistemas(f,g,a,b,xa,ya,h,n)

% Metodo de Taylor de orden N para sistemas. El problema de valores
% iniciales
% x'=f(t,x,y), x(a)=xa
```

```
% y'=g(t,x,y), y(a)=ya
% [T,X,Y,Xe,Ye]=Taylor_ordenN_sistemas(f,g,a,b,xa,ya,h,n)
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente de la primera
% ecuacion
% g cadena de caracteres identificando el termino fuente de la segunda
% ecuacion
% a y b son los extremos del intervalo
% xa condicion inicial en la variable x
% ya es la condición inicial en la variable y
% h es el tamaño de paso de cada aproximación
% Datos de salida
% T es el vector de las abscisas de la solución
% X es el vector de las ordenadas en la variable x
% Xe valor de la solución exacta para la variable x
% Y es el vector de las ordenadas en la variable y
% Ye valor de la solución exacta para la variable y
% Ejemplo1:
% [T,X,Y,Xe,Ye]=Taylor_ordenN_sistemas('x+y-7','3*x+2*y+5',0,1,2,3,0.2,4)
% Ejemplo2, sin solución explícita.
% [T,X,Y,Xe,Ye]=Taylor_ordenN_sistemas('x*y+t','x-t',0,1,0,1,0.2,4 ))


syms t x y;

M=(b-a)/h;
T=zeros(1,M+1);
X=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
X(1)=xa;
Y(1)=ya;

%Calculamos las derivadas necerarias

d{1}=f;
m{1}=g;

for i=2:n
```

```
    d{i}=diff(d{i-1},t)+diff(d{i-1},x)*f+diff(d{i-1},y)*g;
end

for i=2:n
    m{i}=diff(m{i-1},t)+diff(m{i-1},x)*f+diff(m{i-1},y)*g;
end

% Desarrollo de Taylor

for j=1:M
    X(j+1)=X(j)+h*subs(f,{t,x,y},{T(j),X(j),Y(j)});
    Y(j+1)=Y(j)+h*subs(g,{t,x,y},{T(j),X(j),Y(j)});

    for i=2:n
        X(j+1)=X(j+1)+h^i/factorial(i)*subs(d{i},{t,x,y},{T(j),X(j),Y(j)});
        Y(j+1)=Y(j+1)+h^i/factorial(i)*subs(m{i},{t,x,y},{T(j),X(j),Y(j)});
    end
end


% Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(xa)];
condicion_inicial2=['y(',num2str(a),')=',num2str(ya)];

S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);

if isempty(S)==0

    Xe(1)=xa;
    Ye(1)=ya;
    for j=2:M+1
        Xe(j)=subs(S(1).x,{t},{T(j)});
        Ye(j)=subs(S(1).y,{t},{T(j)});
    end

    % Calculamos los errores cometidos en cada aproximación.
    ex=(Xe-X);
```

```
    ey=(Ye-Y);


    % Código para pintar las soluciones aproximadas y reales con
    % colores diferentes

    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    hold on

    h2=plot(T,Xe,'ro');
    legend([h1,h2],'Aproximación en los puntos','Valor exacto en los puntos');


    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    hold on
    h5=plot(T,Ye,'ro');
    legend([h4,h5],'Aproximación en los puntos','Valor exacto en los puntos');

else
    Xe=[];
    Ye=[];
    fprintf('No existe solución exacta para la ecuación diferencial');
    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    legend([h1],'Aproximación de los puntos');

    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    legend([h4],'Aproximación de los puntos');
end
```

5.9. APLICACION NAVAL PARA ECUACIONES DIFERENCIALES

Una de las aplicaciones que podemos asociar a las ecuaciones diferenciales es el caso de un vacío de un tanque de lastre o un deposito en un buque.

El caso que expondré a continuación es el de un tanque de lastre de 100L de volumen en los que se encuentra disuelta 20kg de sal A. Disponemos en otra zona de un tanque que contiene 1kg de sal por litro ($a = 1kg/L$) que bombea su contenido al primer tanque a razón de 7L/min (ν_1). La mezcla se expulsa al exterior a 8L/min (ν_2). Se trata de determinar la función que da la cantidad de sal en cada instante y el tiempo en el que se vaciará completamente el tanque.

Como datos tenemos $A = 20kg$, $a = 1kg/L$, $V_0 = 100L$, $\nu_1 = 7L/min$ $\nu_2 = 8L/min$.
Como condición inicial tenemos que $y(0) = A = 20$ en el instante $t = 0$.
La ecuación que mide la cantidad de sal en cada instante es:

$$y'(t) + \frac{8}{100 + (7-8)t} y(t) = 7 \cdot 1.$$

Que se puede expresar:

$$y'(t) + \frac{8}{100 + t} y(t) = 7.$$

Siendo la ecuación que introducimos en el programa:

$$y'(t) = 7 - \frac{8y}{100 + t}.$$

Para su resolución aplicamos el *Método de Taylor* de orden N a través del programa *TaylorN*, que se puede observar en la Figura 37:

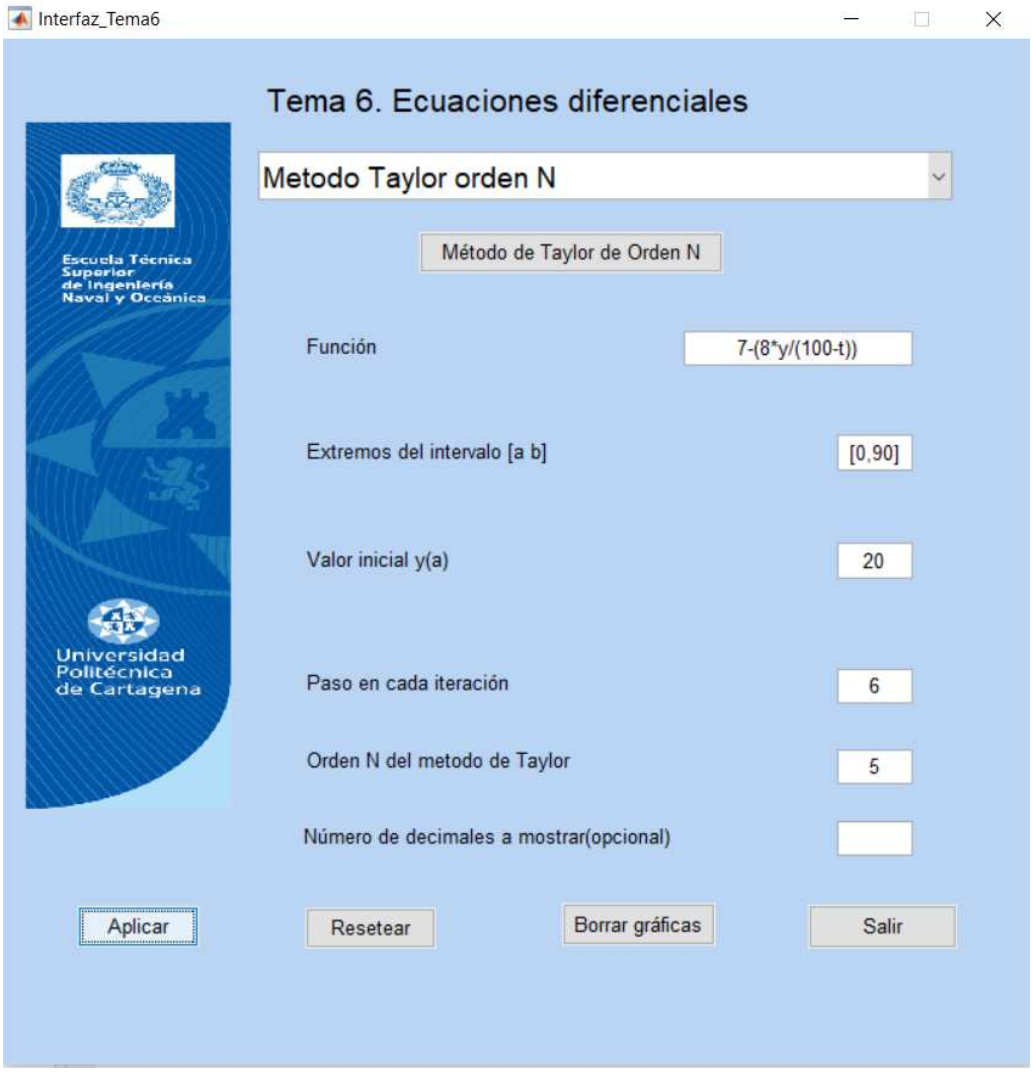

Figura 37: Pantallazo del programa con los datos introducidos

Se puede apreciar que el orden de *Taylor* utilizado es 5, lo que nos da un resultado muy aproximado. Pulsamos aplicar y obtenemos una ventana con los resultados, como se muestra en la Figura 38:

```
Expresión de la ecuación diferencial :
7-(8*y/(100-t))
********************************************************************************
Intervalo de aproximación:
[0.00000000  90.00000000]
********************************************************************************
Valor inicial:
y(0.00000000)=20.00000000
********************************************************************************
Paso en cada iteración:
6.00000000
********************************************************************************
Orden de Taylor:
5
********************************************************************************
Resultados
********************************************************************************
|        Abscisas        |  |          Y          |  |       Yexacto       |  |        error        |
|   0.00000000e+00   |  |   2.00000000e+01   |  |   2.00000000e+01   |  |   0.00000000e+00   |
|   6.00000000e+00   |  |   4.52345876e+01   |  |   4.52344849e+01   |  |   -1.02731287e-04   |
|   1.20000000e+01   |  |   5.92293893e+01   |  |   5.92292380e+01   |  |   -1.51283272e-04   |
|   1.80000000e+01   |  |   6.56470385e+01   |  |   6.56468731e+01   |  |   -1.65357026e-04   |
|   2.40000000e+01   |  |   6.70958806e+01   |  |   6.70957217e+01   |  |   -1.58852692e-04   |
|   3.00000000e+01   |  |   6.53883005e+01   |  |   6.53881592e+01   |  |   -1.41290858e-04   |
|   3.60000000e+01   |  |   6.17483191e+01   |  |   6.17482002e+01   |  |   -1.18954780e-04   |
|   4.20000000e+01   |  |   5.69755911e+01   |  |   5.69754953e+01   |  |   -9.57927143e-05   |
|   4.80000000e+01   |  |   5.15723963e+01   |  |   5.15723222e+01   |  |   -7.41166141e-05   |
|   5.40000000e+01   |  |   4.58396742e+01   |  |   4.58396191e+01   |  |   -5.51304513e-05   |
|   6.00000000e+01   |  |   3.99476105e+01   |  |   3.99475712e+01   |  |   -3.93184222e-05   |
|   6.60000000e+01   |  |   3.39857404e+01   |  |   3.39857136e+01   |  |   -2.67201959e-05   |
|   7.20000000e+01   |  |   2.79969947e+01   |  |   2.79969776e+01   |  |   -1.71170219e-05   |
|   7.80000000e+01   |  |   2.19995711e+01   |  |   2.19995610e+01   |  |   -1.01479471e-05   |
|   8.40000000e+01   |  |   1.59999710e+01   |  |   1.59999656e+01   |  |   -5.36366819e-06   |
|   9.00000000e+01   |  |   1.00000014e+01   |  |   9.99999920e+00   |  |   -2.15211167e-06   |
```

Figura 38: Resultados obtenidos en Matlab

Mostramos a continuación la gráfica en la que se muestra la solución de la ecuación diferencial en el tiempo de manera exacta (Rojo) y la aproximación realizada por *Taylor* de orden 5 (Azul), Figura 39

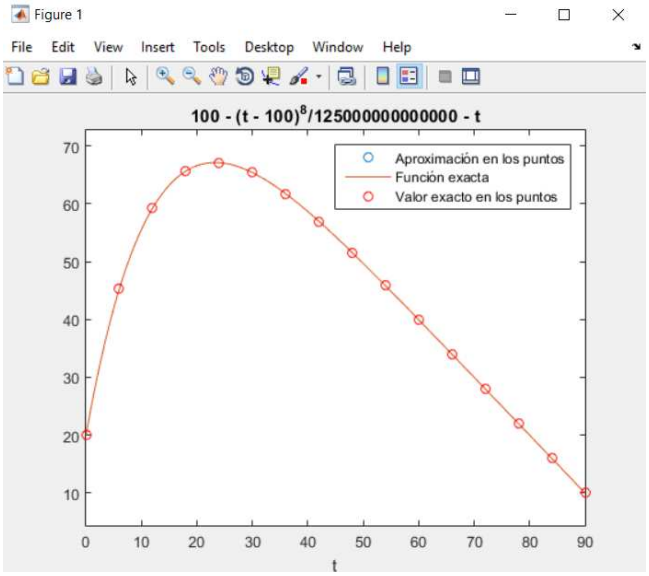

Figura 39: Grafica de los resultados obtenida con Matlab

Como se puede observar en la Figura 39 la aproximación obtenida mediante Taylor de orden 5 es tan alta que apenas se aprecia la diferencia entre el valor real y la aproximación ya que el error, como se muestra en la Figura 38, es mínimo. Cuando el valor de $t = 100$ el tanque quedará vacío.

6. INTERFAZ GRÁFICA

Para un uso más sencillo y accesible de todos los programas que hemos mostrado anteriormente, hemos decidido implementarlos todos en una interfaz gráfica realizada mediante el programa *Matlab*.

Para la realización del la interfaz, ya que mi proyecto se basa en la realización de una ampliación del proyecto de *Jesus Monserrat Torrecillas* [6] he decidido realizar las modificaciones sobre la interfaz gráfica que el creó, añadiéndole todos los elementos en los que he trabajado con el objetivo de mejorar y expandir las funciones de este programa. A continuación mostraré la interfaz gráfica tal y como mi compañero la presentó en su trabajo en sus temas 5 y 6 en las Figuras 40, 41 [6]:

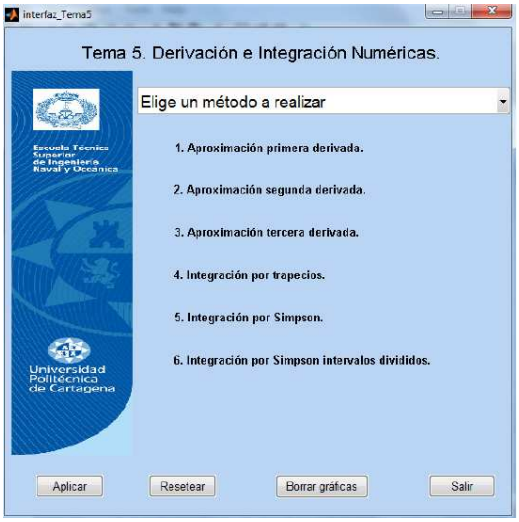

Figura 40: Interfaz Tema 5

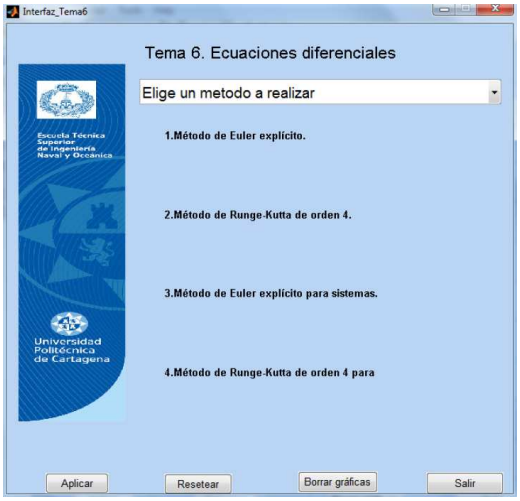

Figura 41: Interfaz Tema 6

En el primer caso, relacionado con el programa de *Integración Numéricos* (Tema 5) hemos añadido los programas correspondientes a las reglas de *Newton cotes* en una variable cerradas y abiertas, *NcotesN* y *NcotesNa* respectivamente. Y en dos variables *NcotesN2var* y *NcotesN2vara*. Los programas para integración de 3 variables se han implementado pero hemos decidido dejarlos fuera de la interfaz gráfica, debido a su excesivo tiempo de cálculo en la forma que están programados.

En la Figura 42 vemos la ampliación de la interfaz gráfica realizada.

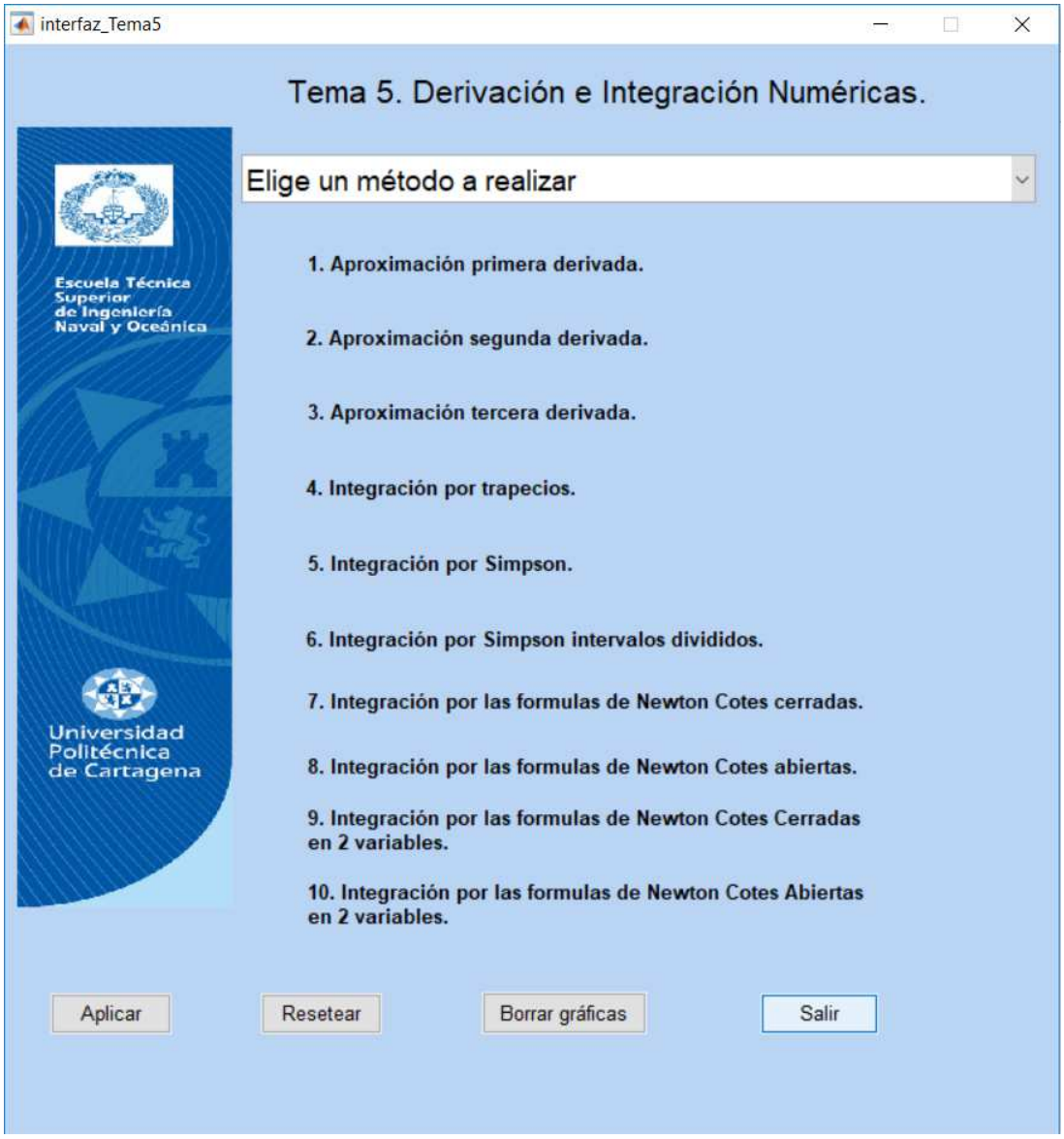

Figura 42: Interfaz Tema 5 ampliada

En el segundo caso, la interfaz de *Ecuaciones Diferenciales*, hemos añadido el programa de *Taylor de orden N* tanto para ecuaciones diferenciales ordinarias *TaylorN* como el para sistemas $TaylorN\_sistemas$

En la Figura 43 vemos la ampliación en la interfaz gráfica, añadiendo los dos programas correspondientes.

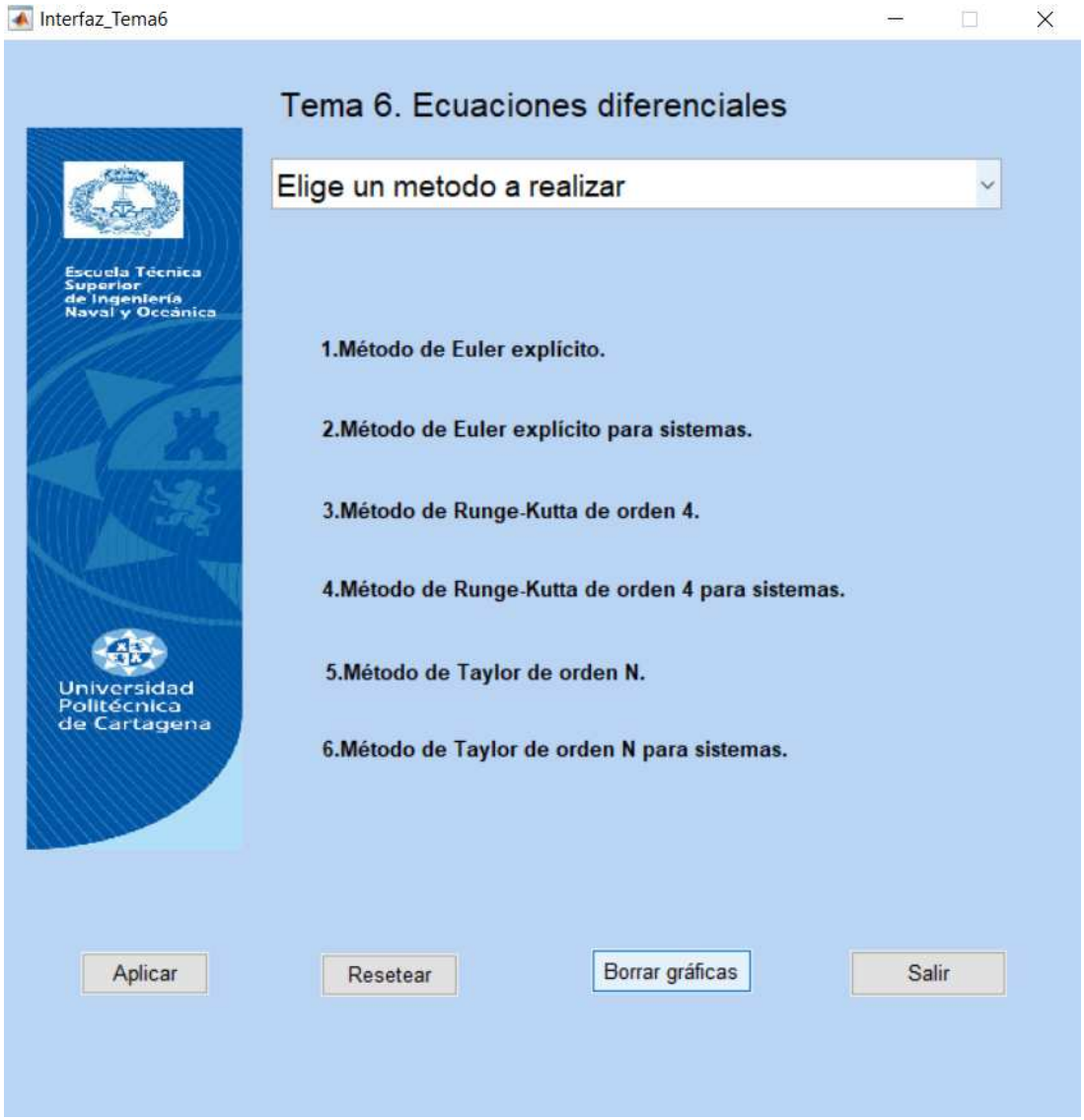

Figura 43: Interfaz Tema 6 ampliada

6.1. Interfaz Tema 5

A continuación explicaré como hemos realizado la modificación del programa *Interfaz Tema 5*:

Para ello introducimos el programa $interfaz\_Tema5.fig$ en *Matlab* e introducimos el comando *guide* seguido del nombre del archivo. Esto generará la ventana de editor de la interfaz gráfica que puede verse en la Figura fig:Explicación del editor:

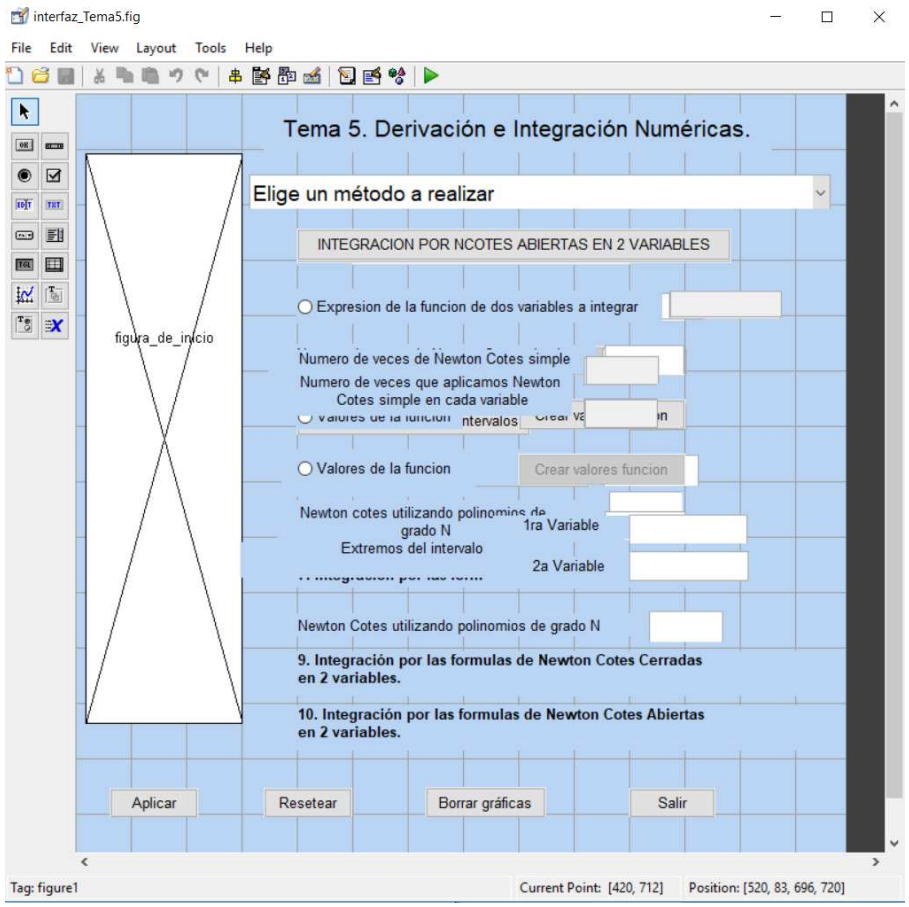

Figura 44: Editor de interfaz gráfica

Lo primero que debemos realizar es un esquema en el cual selecciono los nuevos programas que quiero implementar y la estructura gráfica que tendrá cada uno de ellos, eso incluye variables a introducir, elementos representativos, colocación de las pestañas...

El método de cómo realizar la interfaz gráfica lo podemos dividir en los siguientes puntos:

1. Crear la opción, en el menú desplegable y en la pantalla inicial

2. Crear los objetos (pushbutton, text, radiobutton...) prestando atención a los **Tag**

3. Programar cada uno de los objetos si es necesario

4. Programar la lista desplegable

5. Añadir las opciones en el botón *Aplicar* y en el botón *Reset*

6. Modificar el programa específico **.m** para que conecte bien con la interfaz

En nuestro caso debemos modificar la interfaz para la introducción de los programas:

- Programa general de Newton Cotes cerrado NcotesN

- Programa general de Newton Cotes abierto NcotesNa

- Programa general de Newton Cotes cerrado para 2 variables NcotesN2var

- Programa general de Newton Cotes abierto para 2 variables NcotesN2varA

Tambien introducirlos dentro de la carpeta correspondiente a la interfaz, incluyendo como es obvio, los programas auxiliares que necesitan para su correcto funcionamiento (por ejemplo el programa *PesosNcotes*).

La principal variación que debemos incluir a cada uno de los programas es la opción de introducir los datos a través de tabla de valores en vez de a partir de una función. Esto es debido a que los datos dados en algunos problemas de ingeniería nos los dan en forma de tabla,como en el caso del apartado 2.9 en el cálculo de la curva de áreas de secciones.

Esto lo realizamos mediante un comando «if» que dependiendo de las características de los datos introducidos, realizará el programa de forma distinta.

En el caso del programa *NcotesN2var* y *NcotesN2vara* es más laboriosa, ya que la cadena de caracteres que representa la función debe ser sustituida ahora por una matriz de datos.

Además incluimos dos programas llamados *vector_valores_funcion* y *generar_datos2D* que nos permiten introducir los datos directamente en la interfaz a través del *pushbutton* correspondiente. A continuación se muestran ambos programas:

6.1.1. Programa de Matlab *vector_valores_funcion*

```
function f=vector_valores_funcion

% Esta función debe completarse con los datos particulares de los valores
% que toma la función en los diversos puntos en los que se quiere hallar su
% integral.
% A continuación aparecen indicaciones para ir completándolo.
% En general un vector f de una una fila y n columnas se introduce escribiendo
% entre corchetes cada dato de las ordenadas de los puntos, separando cada
% una de ellas por un espacio o una coma.
% A continuación vemos un ejemplo.
% f=[0 5.33 6.08 6.251 6.724]


% Modificaz la definición del vector f apropiadamente, incluyendo las
% columnas que sean necesarias

f=[6.86 30.32 56.50 79.38 98.87 117.674 132.73 142.99 149.86 152.25 153.70 152.38 145.50 135.66
119.67 97.47 74.47 53.51 35.61 22.69 13.28];
```

6.1.2. Programa de Matlab *generar_datos2D*

```
function f=valores_funcion2D

% Esta función debe completarse con los datos particulares de los valores
% que toma la función en dos variables en los diversos puntos de un mallado
% bidimensional en los que se quiere hallar su
% integral.
% A continuación aparecen indicaciones para ir completándolo.
% En general una matriz f de n filas y m columnas se introduce escribiendo
% entre corchetes los datos por filas, separándolas por puntos y coma
% A continuación vemos un ejemplo.
```

```
% f=[ 0 5.33 6.08 6.251 6.724;0 5.33 6.08 6.251 6.724;0 5.33 6.08 6.251 ...
% 6.724;0 5.33 6.08 6.251 6.724;0 5.33 6.08 6.251 6.724;0 5.33 6.08 6.251 6.724]


% Modificaz la definición de la matriz f apropiadamente, incluyendo las
% filas y columnas que sean necesarias o leyendo los datos de un fichero

load('datos2D.mat');
```

Otra variación que incluimos en el programa es la capacidad del mismo de generar una ventana en la que se muestren los resultados obtenidos, y que además incluya los datos iniciales dados, como se muestra a continuación en la figura 45:

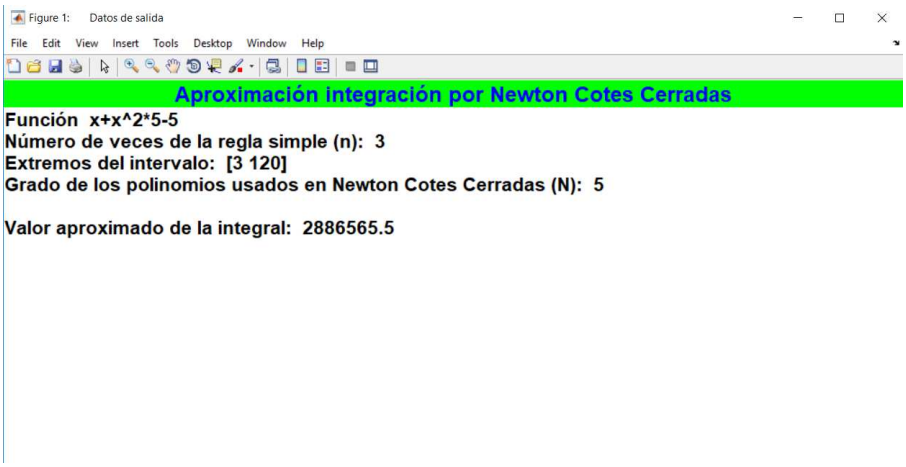

Figura 45: Ventana donde se muestran los resultados de la integración de la función $x+5x^2-5$ entre 3 y 120 con un valor N=5 y n=3

Una vez finalizado los cambios, las interfaces quedarán de la forma que se muestra en las Figuras 46, 47, 48, 49:

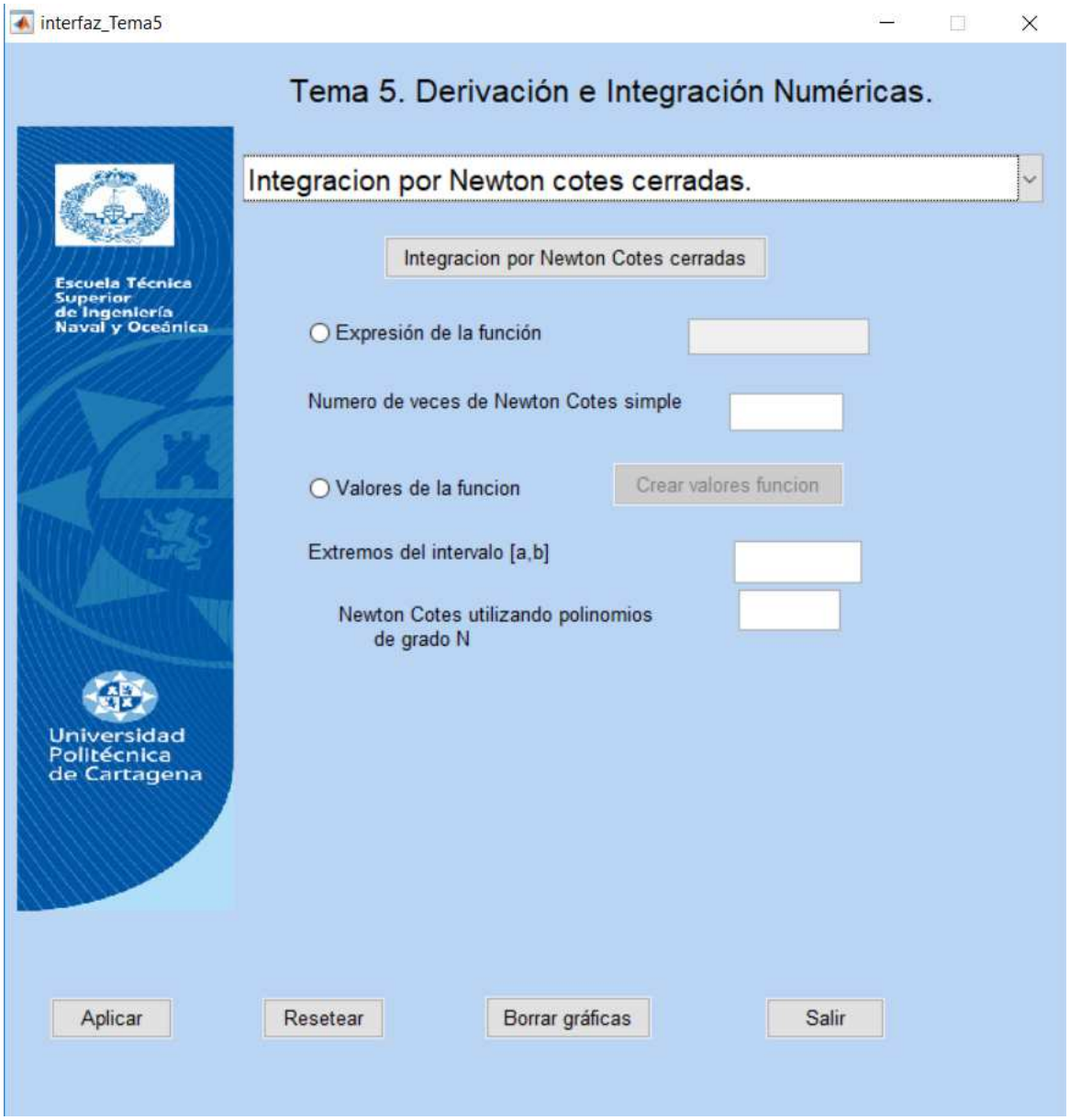

Figura 46: Interfaz del programa de Newton Cotes Cerradas

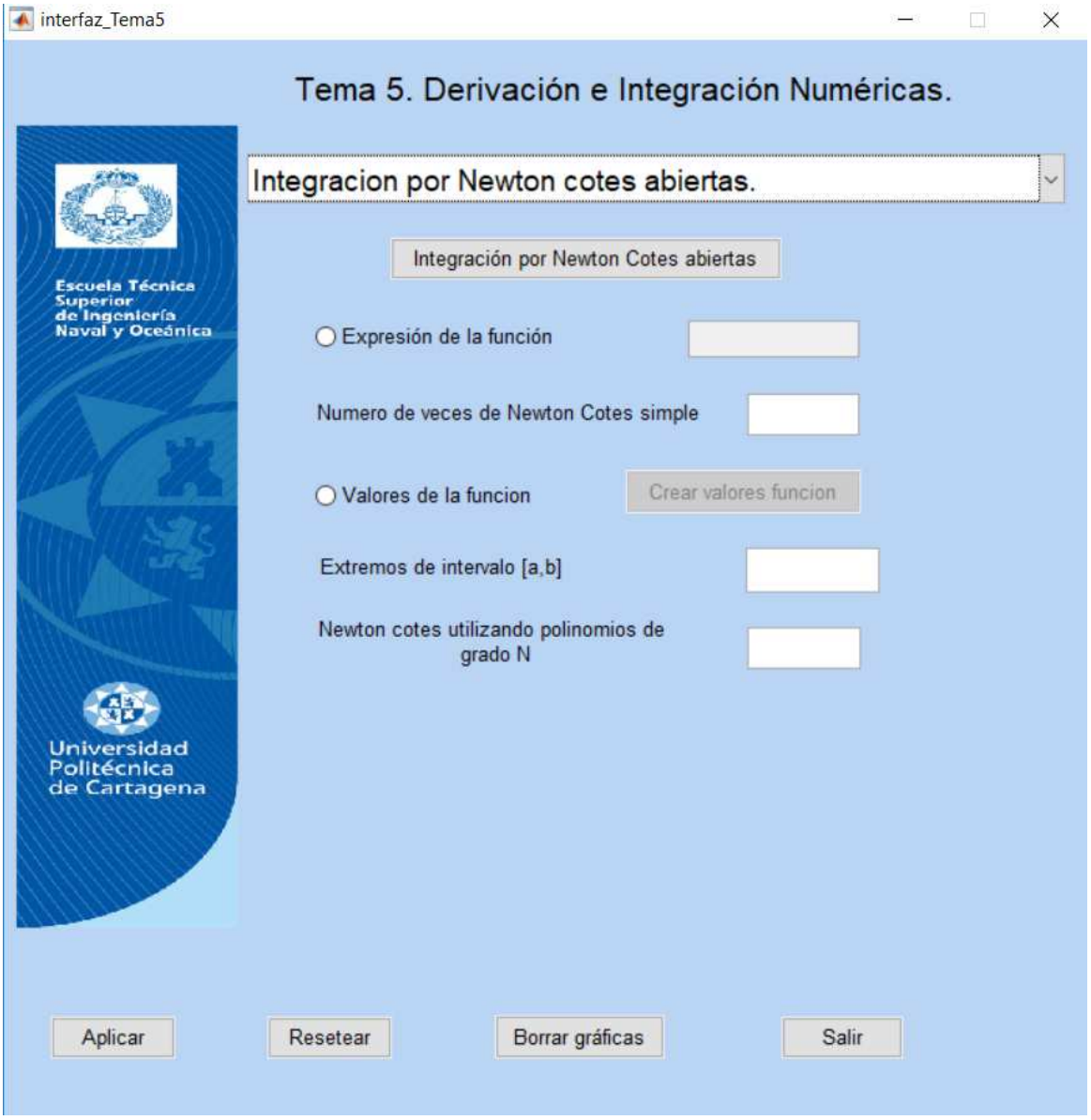

Figura 47: Interfaz del programa de Newton Cotes Abiertas

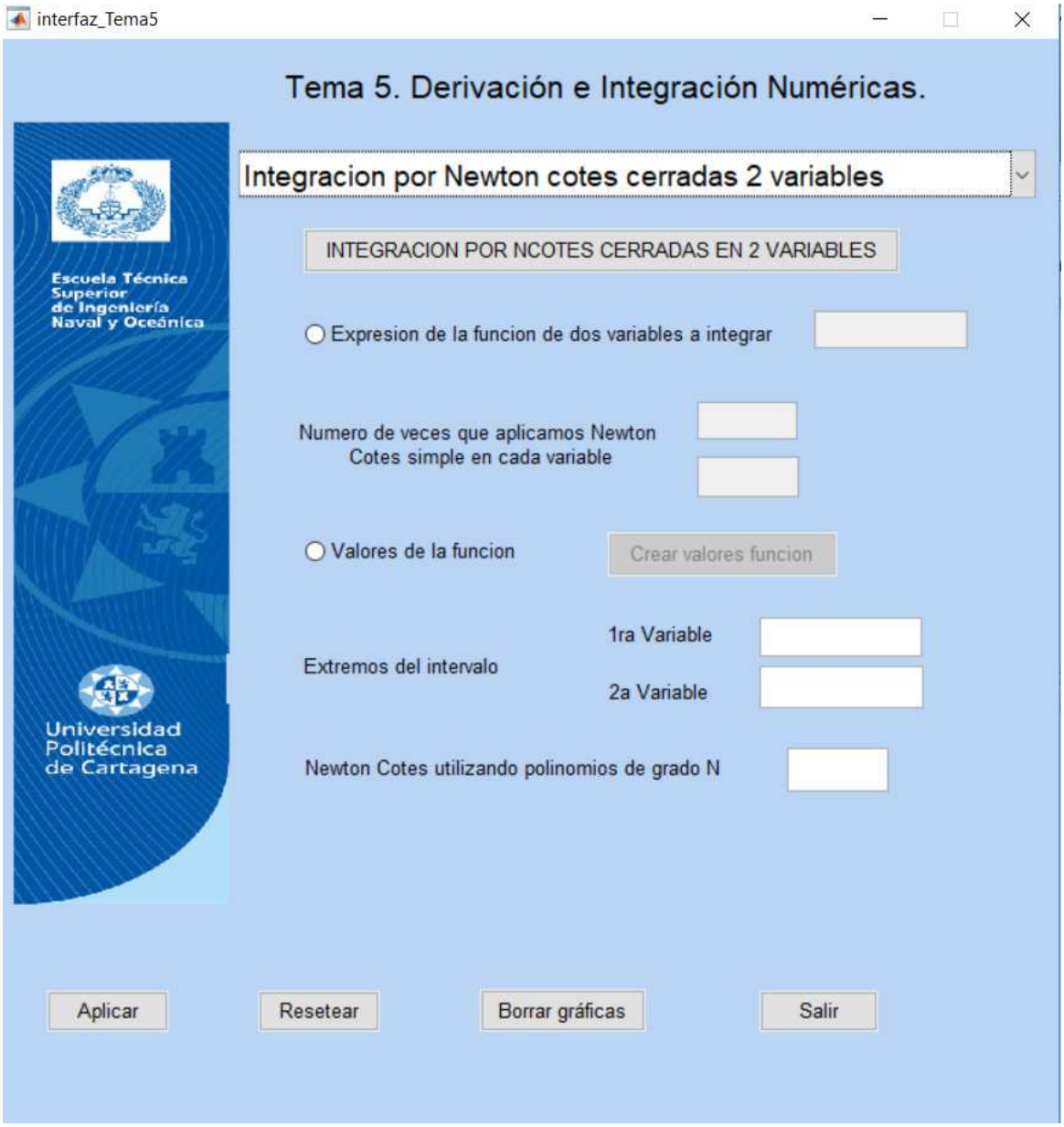

Figura 48: Interfaz del programa de Newton Cotes en 2 variables Cerradas

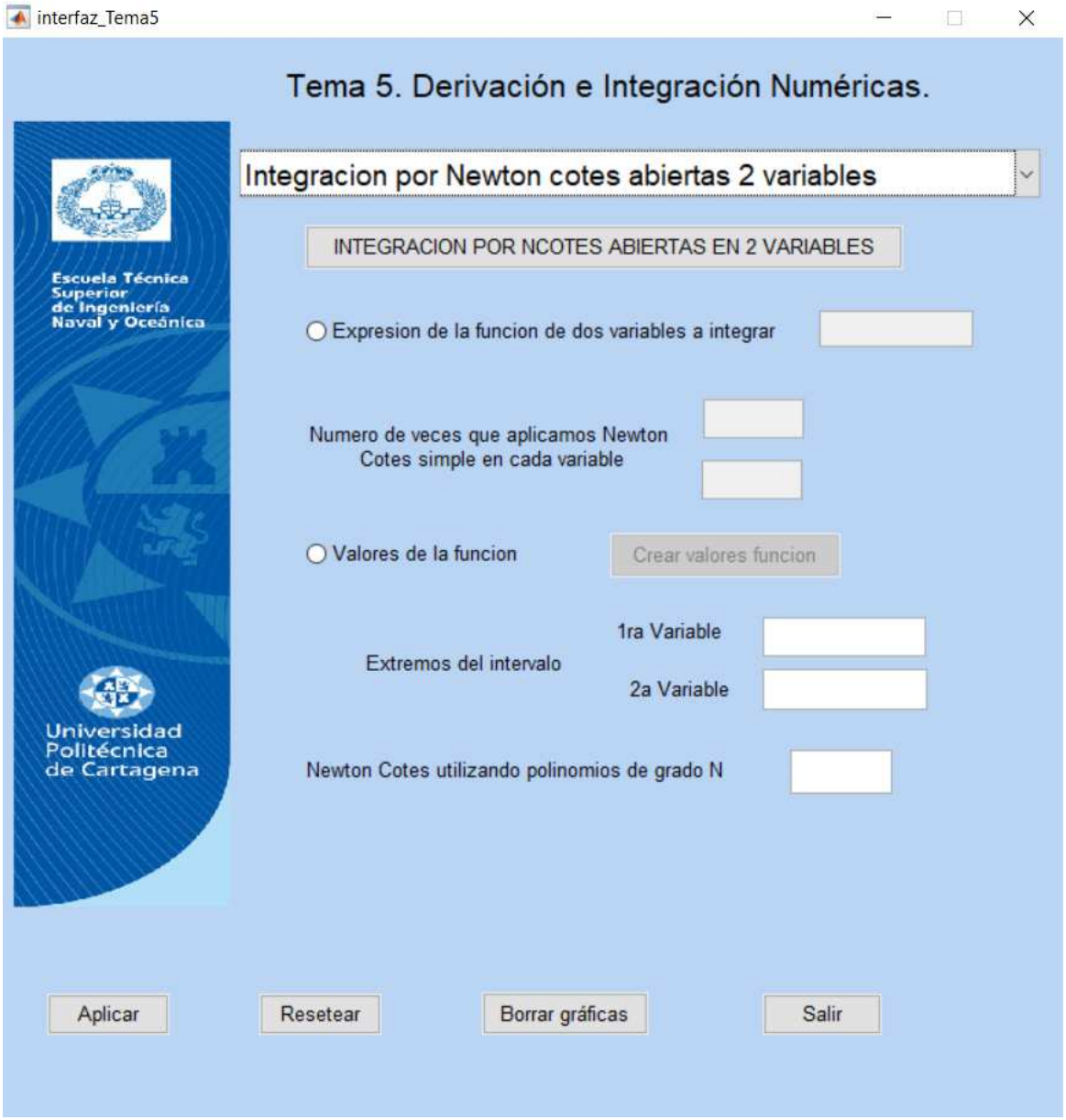

Figura 49: Interfaz del programa de Newton Cotes en 2 variables Abiertas

Como añadido hemos modificado el programa para que en caso de que selecciones el *radio-button* de la opción que elijas, el correspondiente a la otra opción se desactive, sin necesidad de

desactivarlo manualmente para seleccionar el otro.

6.2. Interfaz Tema 6

Este caso ha sido mucho más sencillo, ya que únicamente hemos tenido que introducir los programas correspondientes al *Método de Taylor* para ecuaciones diferenciales ordinarias y para sistemas: $Taylor\_OrdenN$, $Taylor\_OrdenN\_sistemas$.

Ambos programas han sido modificados para que al ejecutarlos a traves de la interfaz nos de una ventana con los resultados obtenidos. Su resultado en un ejemplo concreto se muestra a continuación en la Figura 50:

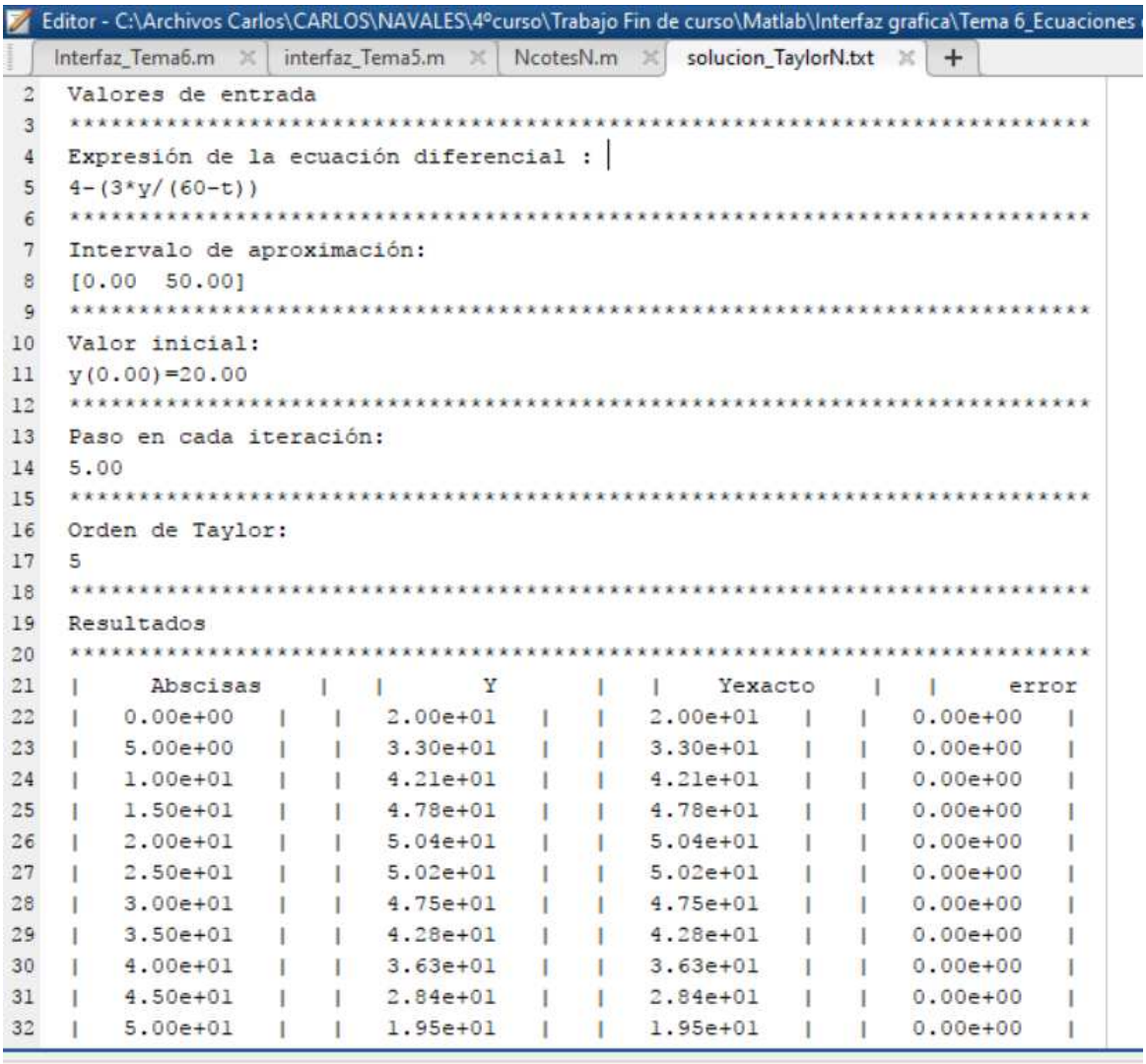

Figura 50: Imagen de la presentación de los resultados obtenidos con Taylor N

La interfaz de los programas se muestra a continuación:

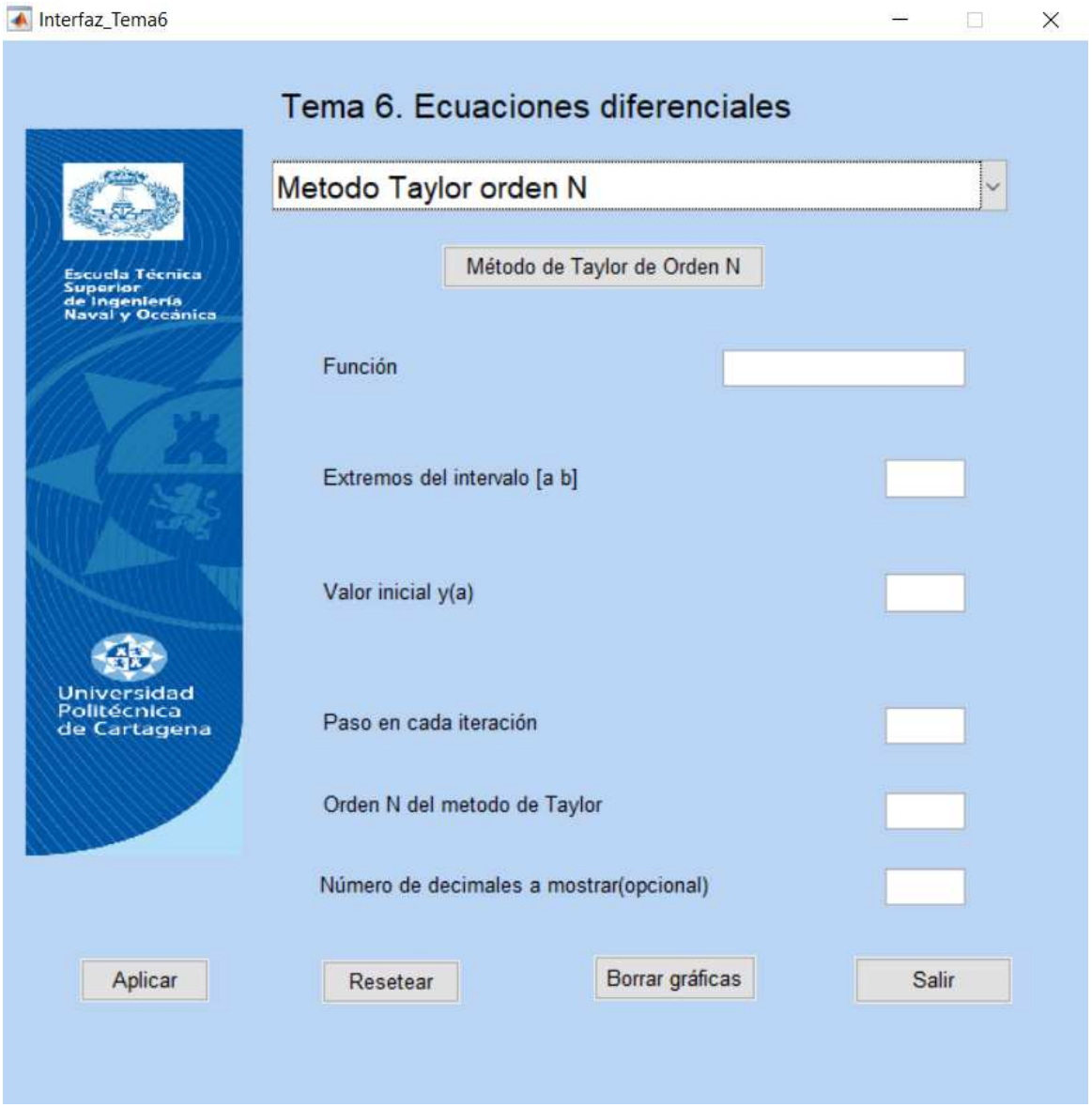

Figura 51: Interfaz del programa Taylor de orden N

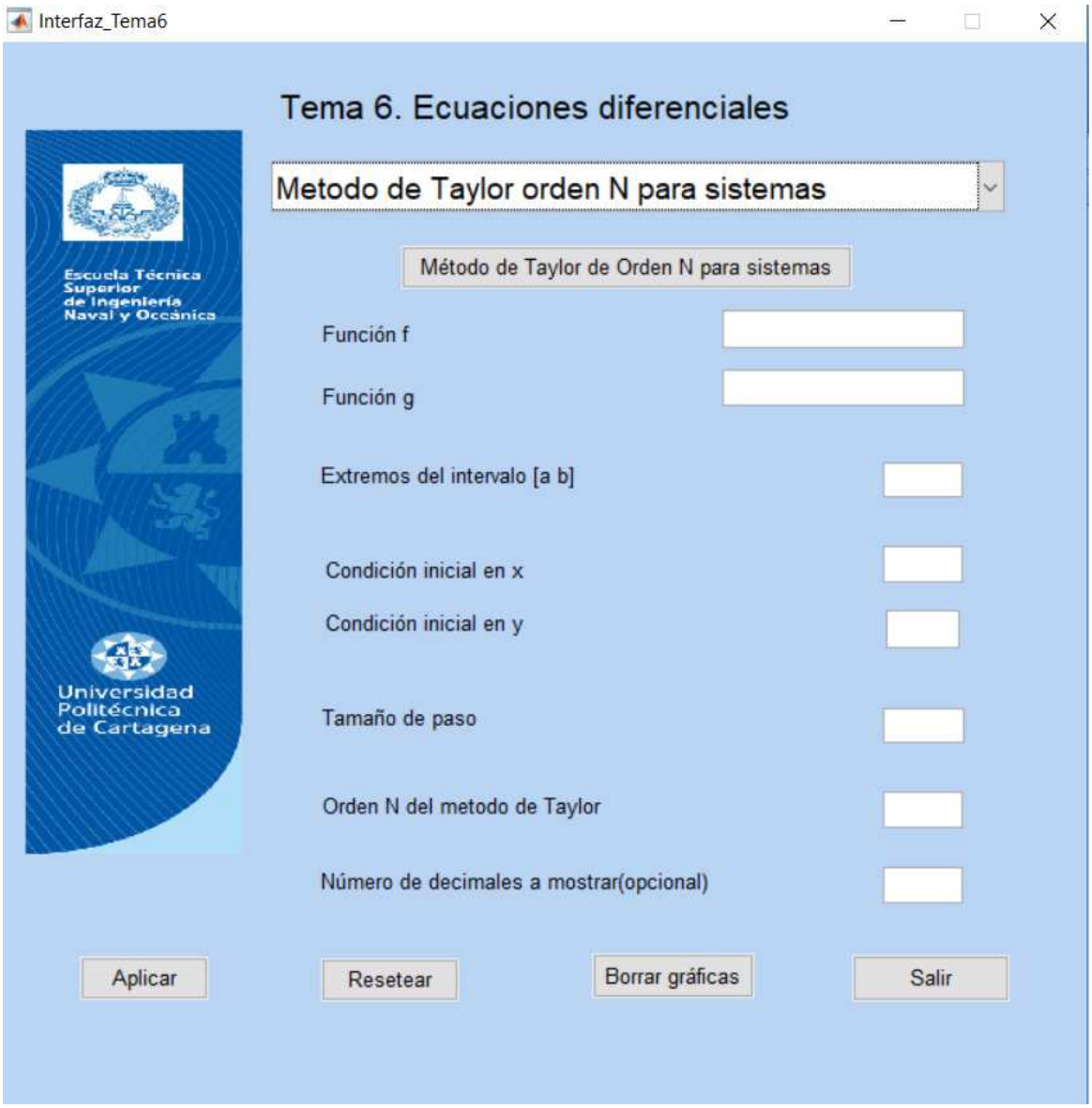

Figura 52: Interfaz del programa Taylor de orden N para sistemas

7. PROGRAMAS Y ELEMENTOS AUXILIARES USADOS

7.1. Matlab

Figura 53: Icono de Programa Matlab

El programa base que he usado para la programación de los distintos Métodos de cálculo numérico ha sido *Matlab*, un programa de programación matemática que nos permite la generación de bucles y condicionales y nos da las herramientas necesarias para conseguir programarlos.

Matlab consiste en una barra de herramientas donde aparecen las distintas opciones del programa **1**, una lista de archivos donde aparece la carpeta que estamos usando a la hora de trabajar **2**, un editor en el que podemos escribir el código del programa correspondiente, añadir comentarios que nos permitan definir cada uno de los elementos que componen el programa y estructurarlo de forma ordenada **3**, una línea de comandos donde ejecutar el código y obtener los resultados **4** y finalmente una ventana donde aparecen todas las variables establecidas en el programa **5**. Esto se muestra en la Figura 54.

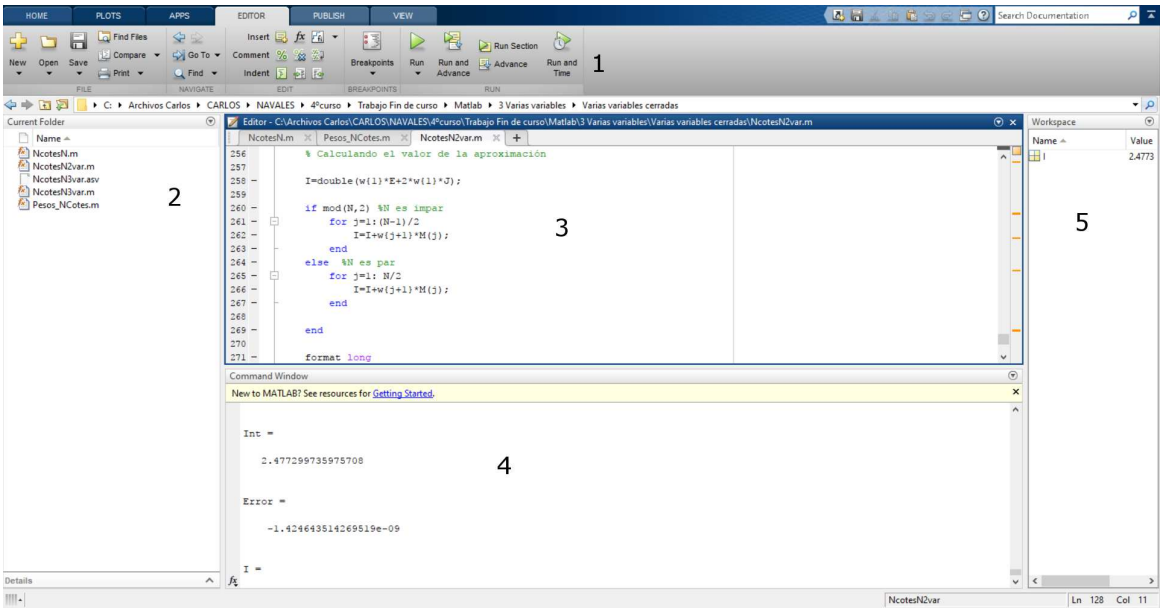

Figura 54: Pantalla del programa Matlab

De este modo tenemos idea general del programa utilizado, siendo la zona donde más nos centraremos la línea de comandos **2**.

Para montar el programa *Matlab* utiliza un código propio para la realización de los distintos bucles y condicionales. Toda la información para el trabajo la he obtenido de la asignatura *Informática* de 1º de carrera del *Grado de Arquitectura Naval e Ingeniería de Sistemas Marinos* y de un documento llamado *Aprenda Matlab 7.0 como si estuviera en primero* [8] además de las anotaciones y ayuda recibida por el profesor encargado de dirigir mi trabajo fin de grado.

Todos los programas vistos anteriormente han sido realizados con *Matlab* y para realizar su conversión del formato *Matlab* (.m) a un .tex (necesario para introducirlo en *Latex*) hemos utilizado un programa realizado en *Matlab* con el nombre de *m2tex* que permite mediante la introducción de un archivo su transformación en lenguaje de *Latex*.

A este programa va asociado otro denominado *guion_matlab2tex* el cual nos permite introducir la carpeta origen donde se encuentran los archivos .m a transformar. Sin embargo esto nos obliga a introducir los datos de todas las carpetas y todos los archivos por separado uno por uno en un listado.

Nuestra solución fue la modificación de este programa complementario, y la generación de una nueva subrutina que hemos llamado *matlab2tex*, con el fin de que únicamente introduciendo

do la carpeta origen el programa detecte aquellos archivos con extension .m y los transforme en .tex. Estos archivos transformados se situarán en una carpeta paralela con el mismo nombre que la original pero añadiendo la palabra «tex» al inicio.

Dicha carpeta guarda referencia de la localización del archivo dentro de la carpeta origen.

7.2. LaTex

Figura 55: Icono de Programa Latex

Latex es un programa de composición de textos que he usado para la redacción de mi Trabajo Fin de Grado. Consiste en una consola, en la que mediante la escritura de un código propio de programación, permite la obtención de textos científicos sin la necesidad de preocuparse de los distintos aspectos estructurales del texto ya que *Latex* lo realiza de manera automática y al principio del programa nos permite la introducción de manera automatica de las características que tendrá el documento ($documentclass[a4paper, openright, 12pt]\{article\}$). También debemos mencionar que *Latex* proporcionar muchas facilidades para representar ecuaciones matemáticas.

Latex como programa se divide en tres elementos fundamentales:

- **Una distribución de Latex**: Es el motor de Latex siendo **MiKTeX** el programa utilizado.
- **Un editor de Texto**: Nos permite la escritura del texto y la programación de los distintos elementos que queramos introducir en él. **TeXnicCenter** es el programa que usamos para este propósito.
- **Un visor de documentos**:Nos permite la visualización del documento. Programas como **Adobe Reader** nos permiten su visualización.

Latex como programa tiene varios elementos en su interfaz, la barra de herramientas **1** que nos permite realizar las distintas funciones del programa y dependiendo de la versión incluso

tiene guardadas distintos códigos en forma de botones con símbolo que representan su función y nos permiten la redacción del texto de manera más sencilla y cómoda, la ventana de trabajo **2** donde introducimos el código que corresponde al documento que estamos redactando y es el lugar donde desarrollamos el trabajo con este programa, la ventana de salida de código **3** (Build Output) que nos muestra información sobre los resultados obtenidos al compilar el código escrito y nos informa si existe algún error en el código e incluso donde se encuentra este error y su descripción y finalmente una ventana lateral (Outline) **4** donde se muestran las distintas partes en la que está dividido el documento y lleva un registro de todas las ecuaciones, imágenes y elementos etiquetados como tal. Esto se muestra en la Figura 56.

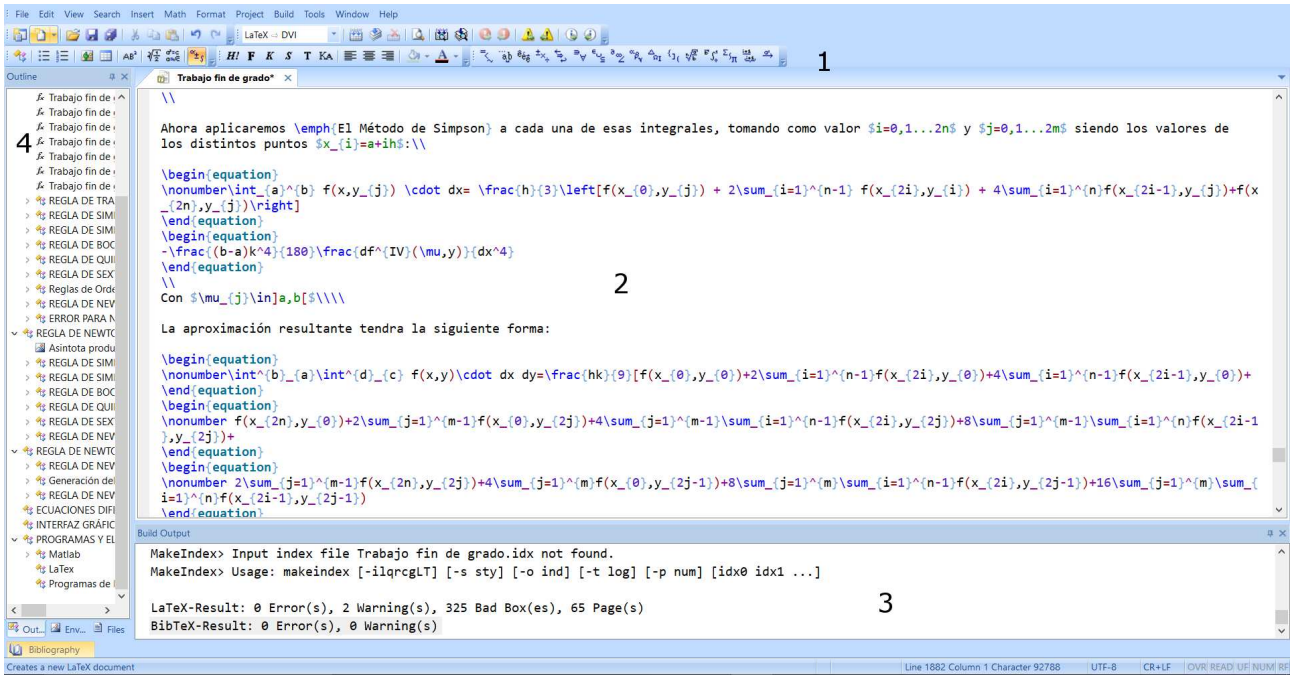

Figura 56: Icono de Programa Latex

Cabe destacar que *Latex* utiliza distintos paquetes que contienen información de códigos que modifican o nos permiten la escritura y la introducción de elementos necesarios para la creación del documento. Algunos son básicos como los que nos permiten la escritura en español o nos permiten el uso de acentos ([spanish]babel) y otros sirven para la introducción de imágenes o ecuaciones y símbolos matemáticos (upgreek, amsmath, graphicx...) o la modificación de algunos elementos del texto (usenames), como vemos en la Figura 57.

Figura 57: Muestra de varios paquetes cargados para la realización de este documento

Para la visualización del resultado producido por nuestro código debemos primero construir el documento mediante un icono denominado (Built current file) que compila el código. En caso de que se produzca algún error, éste aparecerá en la consola de salida (Build Output) donde se nos muestra su posición exacta para que lo resolvamos, además de aparecernos lo que denomina «Warnings» que si bien no impiden la buena compilación del texto, son elementos que el programa o bien no reconoce o que deben ser revisados.

Esto se muestra en la Figura 58

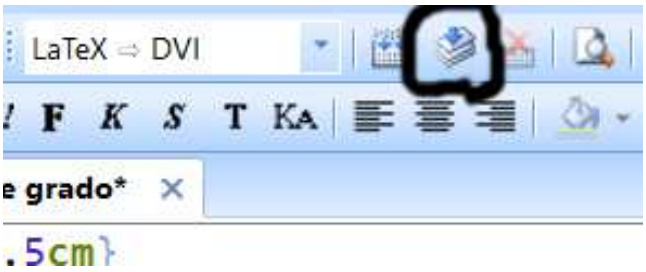

Figura 58: Icono Built current file

Una vez realizado esto, la visualización del documento se suele realizar mediante un programa que permite abrir documentos *DVI* ya que es la manera más rápida y sencilla de ofrecer una visualización del documento tal y como se vería una vez impresa, ya que transformar el documento directamente en el formato **PDF** es más complicado y le lleva más tiempo al programa.

Por último cabe destacar que para la introducción de imágenes en este programa es esencial saber que sólo admite archivos .eps (postScrip Encapsulado) ya que de otra manera no los reconoce y para ello requiere una serie de programas para la realización de gráficas y para trasformar archivos en este formato, los cuales son explicados a continuación.

Cabe destacar que uno de los principales problemas que he encontrado a la hora de trabajar con Latex es su lenguaje de programación ya que cualquier elemento que queramos introducir tiene su código y debemos aprendérnoslo. Para ello una gran ayuda ha sido la página web (http://minisconlatex.blogspot.com.es/) [7] llamada *miniejercicios con Latex* mostrado en la Figura 59, un blog en el que aparecen de forma ordenada y clara muchas de las funciones de las que dispone *Latex* ademas de un tutorial del código para representar todo lo necesario para nuestro documento.

Figura 59: Muestra de la pagina principal del Blog sobre Latex

Como se ve en la imagen, el blog esta ordenado por distintos apartados cada uno abarcando distintos elementos dentro del documento.

7.3. Programas de Imágenes y figuras

Como hemos comentado antes, *Latex* no es capaz de abrir y trabajar con archivos del tipo *.JPG* o *.PNG* ya que solo admite el formato .eps, eso quiere decir que para trabajar con imágenes es necesario el uso de programas auxiliares que nos permitan trabajar con este formato. Estos son:

7.3.1. Dia

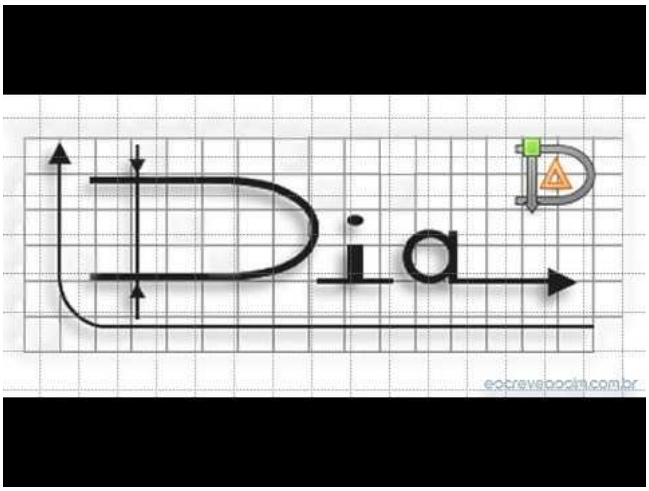

Figura 60: Icono de Programa Dia

Dia es un programa que nos permite la creación de imágenes y la modificación de éstas mediante una sencilla interfaz gráfica muy parecida a programas como *paint* aunque más profesional y enfocado a gráficas. Nos permite la obtención de archivos .eps que podemos utilizar en *Latex*. Consiste en distintas ventanas que describiremos a continuación: La ventana de trabajo **1** donde realizar la imagen que queremos a partir de las herramientas que dispone el programa y la barra de herramientas **2** donde mediante iconos aparecen representadas todas las funciones que podemos realizar con el programa, vemos su interfaz en la Figura 61.

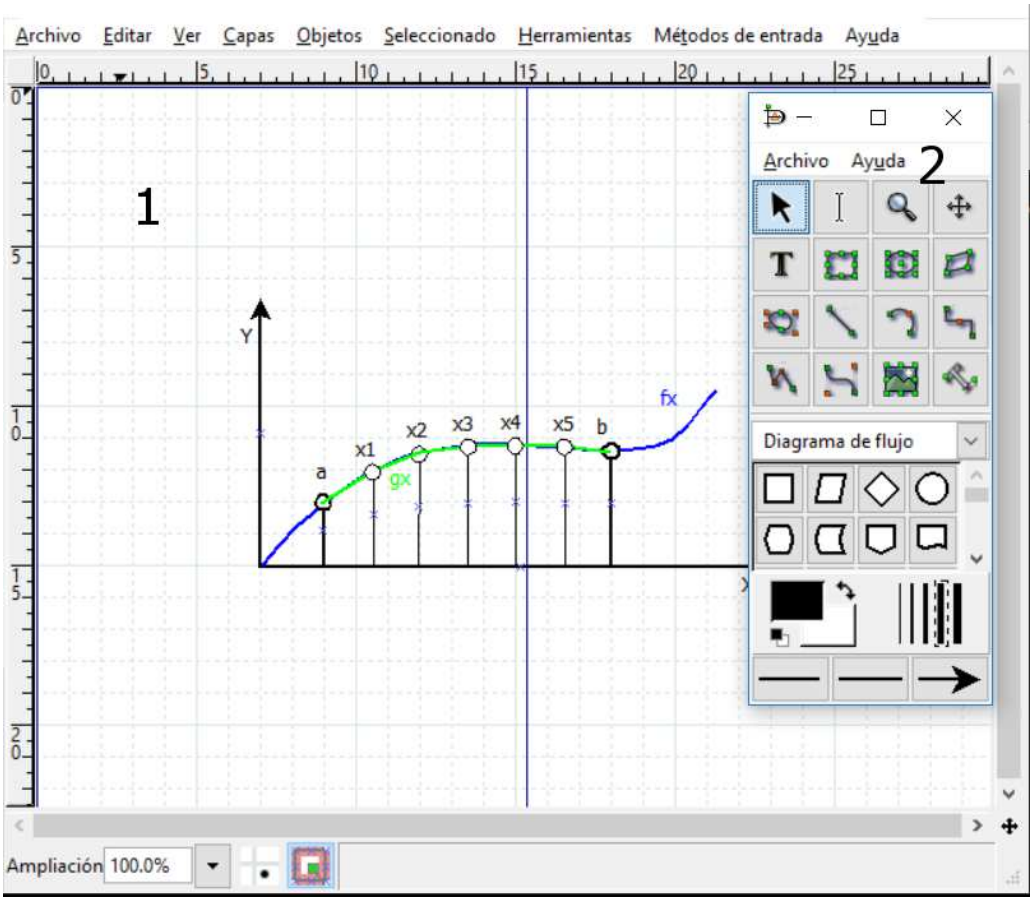

Figura 61: Icono de Programa Dia

Este programa aunque básico es una herramienta muy útil a la hora de representar funciones y distintos elementos que de otra forma serían mas complicados de representar para exportar a *Latex*.

7.3.2. Gimp 2

Figura 62: Icono de Programa Gimp2

El programa *Gimp* es un editor de imágenes de gran versatilidad ya que permite trabajar con un amplio abanico de formatos y nos permite transformar o mas bien exportar nuestras imágenes de un formato a otro. La versión que he usado es la **Gimp2** que incluye mas formatos, entre los cuales se incluye el .eps que es el que realmente me interesa y con el que voy a trabajar más tarde en *Latex*.

El programa en si esta compuesto por varias ventanas separadas, cada una con su función. Estas son: La ventana principal **1** donde aparece la figura con la que estamos trabajando y donde mediante las herramientas de las que dispone el programa podemos modificar la imagen y adaptarla a nuestras necesidades. La caja de herramientas **2** donde aparecen representadas con iconos las distintas funciones gráficas de las que disponemos en el programa, aparte de las distintas opciones dentro de cada una como pueden ser el tamaño, tipología, color... y por último una ventana donde aparecen las características del elemento introducido en la imagen, como su capa correspondiente, visibilidad... Esto se ve en la Figura 63

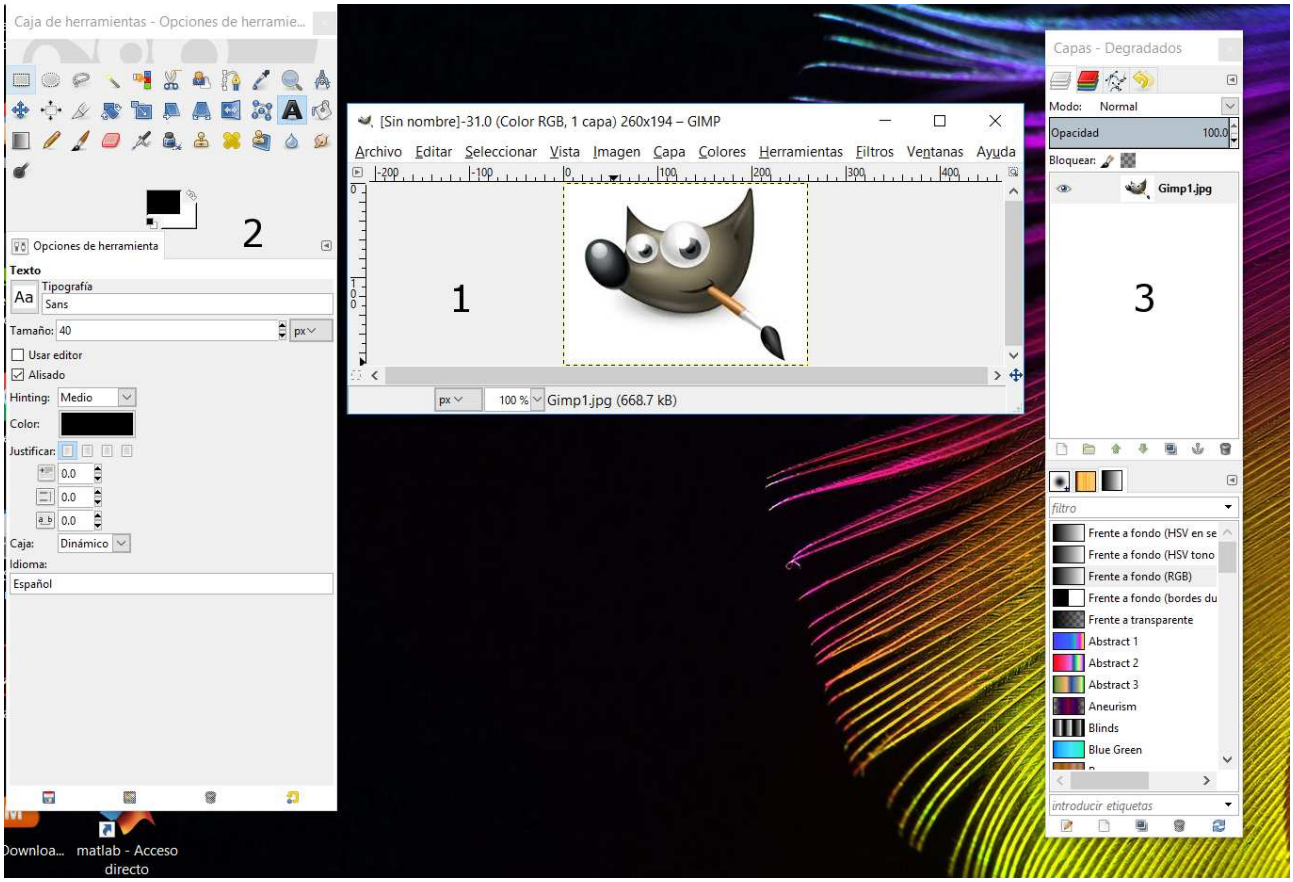

Figura 63: Ventanas correspondientes al programa Gimp2

Este programa lo he usado principalmente con el fin de trasformar imágenes de su formato correspondiente a formato .eps para poder ser utilizadas por *Latex*. El funcionamiento es sencillo ya que sólo debemos abrir la imagen con el programa, el cual admite una gran cantidad de formatos, y una vez abierto seleccionamos la opción *exportar archivo como* y elegimos el formato .eps dentro de todas las opciones que nos ofrece. Si antes de exportar la imagen queremos hacer alguna modificación en ella, mediante las herramienta que dispone *Gimp2* podemos hacerlo. .

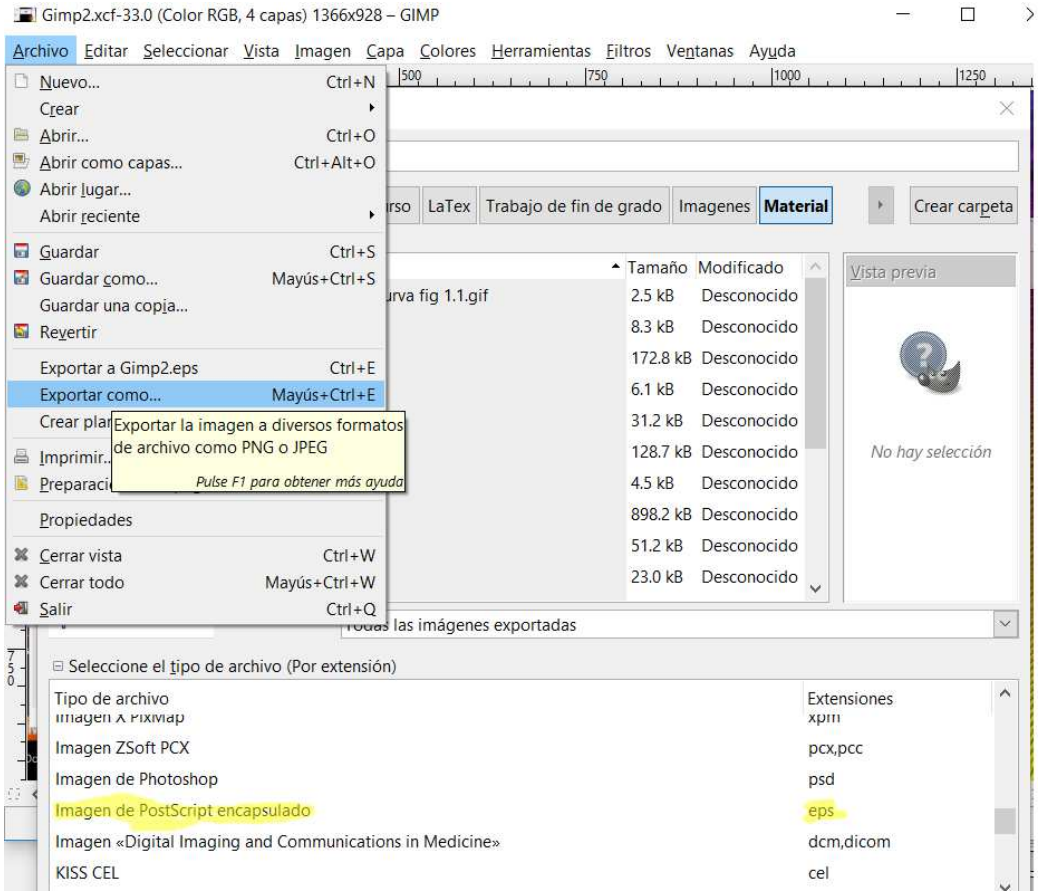

Figura 64: Exportación de un archivo para transformal en formato .eps

En la Figura 64 el elemento marcado es el correspondiente al formato deseado (.eps) con lo que nos crearía un nuevo archivo en la carpeta que decidamos y con el ya podríamos trabajar con la imagen en *Latex*.

8. CONCLUSIÓN

Para concluir este trabajo es necesario destacar la importancia que estos métodos pueden tener en el ámbito, no sólo de la ingeniería, sino en muchos más campos. Cualquier resolución de integrales o ecuaciones diferenciales de gran complejidad puede ser aproximada y resuelta de este modo.

Su aplicación en el ámbito naval, así como en otras ingenierías, es indiscutible teniendo en cuenta la cantidad de veces que aparecen expresiones de difícil resolución, por lo que la explicación de estos métodos de forma exhaustiva, como se ha realizado en este trabajo, está justificada. A su vez, su implementación en una interfaz global para que su uso resulte más sencillo se debe a que no todo el mundo comprende el programa *Matlab* y es una buena forma de acercar estos métodos a este tipo de usuarios.

Para terminar quiero destacar que gracias a haber realizado el trabajo en *Latex*, he conocido mejor esta herramienta de creación de textos y de seguro, en un futuro, me sea de gran ayuda todo lo aprendido gracias a este proyecto.

9. BIBLIOGRAFÍA

Referencias

[1] **Métodos Numéricos con Matlab** *Tercera edición* (*John H. Mathews and Kurtis D Fink*)

[2] **Apuntes de Estructuras** *Universidad Politécnica de Cartagena 3º curso José Alfonso Martínez García*

[3] **Apuntes de Proyecto de Construcción de Plataformas y Artefactos** *Universidad Politécnica de Cartagena José Enrique Gutiérrez Romero*

[4] **Cálculo Numérico** *Teoría y problemas* (*Alicia Cordero Barbero, Jose Luis Hueso Pagoaga and Juan Ramón Torregrosa Sánchez*)

[5] **Métodos Numéricos con Matlab** (*A. Cordero Barbero, E Martínez Molada, J. L. Hueso Pagoaga and J. R Torregrosa Sánchez*)

[6] **Métodos Numéricos en el ámbito Naval** *Trabajo de fin de grado Jesus Monserrat Torrecillas*

[7] **Miniejercicios con Latex** http://minisconlatex.blogspot.com.es.html,

[8] **Aprenda Matlab 7.0 como si estuviera en primero** *Universidad Politécnica de Madrid Javier García de Jalón, José Ignacio Rodríguez y Jesús Vidal*

[9] **Apuntes de Hidrodinámica y propulsión** *Universidad Politécnica de Cartagena 4º curso Domingo García López*