MÉTODOS NUMÉRICOS Y PROGRAMACIÓN 2023/2024

Errores (PRÁCTICA 2)

1.— Se desea evaluar con un error absoluto inferior a 1 cm^2 el área de un recinto cuadrado. Para ello se mide un lado del recinto y se calcula el área elevándolo al cuadrado. Se sabe *"a priori"* que el recinto tiene 1 m^2 de superficie aproximadamente. ¿Qué precisión debe tener la regla que se utilice en la medición? ¿Sería posible efectuar correctamente este cálculo en un ordenador utilizando simple precisión?

Sol. 1. Dado el lado del recinto x, el cálculo del área se realiza como $A = x\,x$. Únicamente se realiza una operación, un producto, por tanto el error relativo será:

$$r_A = 1\,r_x + 1\,r_x + r_A^A$$

Sabemos que el error relativo en x lo podemos descomponer como el error inherente más el de almacenamiento, es decir $r_x = r_x^I + r_x^A$

Por tanto:

$$r_A = 2(r_x^I + r_x^A) + r_A^A = 2r_x^I + (2r_x^A + r_A^A)$$

Para acotar el error relativo sabemos también que el error de almacenamiento está acotado por el error de máquina:

$$\left|r_x^A\right| \leq r_M \qquad ; \qquad \left|r_A^A\right| \leq r_M$$

De modo que:

$$|r_A| \;\leq\; 2\left|r_x^I\right| + \left(2\left|r_x^A\right| + \left|r_A^A\right|\right) \;\leq\; 2\left|r_x^I\right| + 3\,r_M$$

Deseamos evaluar el área con un error absoluto inferior a 1 cm^2, por tanto:

$$|E_A| \leq 1\ [\text{cm}^2] \qquad ; \qquad \text{con } r_A = \frac{E_A}{A} \qquad \Rightarrow \qquad |r_A| \leq \frac{1\ [\text{cm}^2]}{1\ [\text{m}^2]} = 10^{-4}$$

La precisión de la regla p nos determina el error en la medición de x o lo que es lo mismo, su error inherente:

$$\left|r_x^I\right| \leq \frac{p}{1\ [\text{m}]}$$

Si el cálculo se realiza con infinita precisión podemos suponer $r_M = 0$ y por tanto la fuente de error en el cálculo del área provendrá únicamente del error inherente, es decir:

$$|r_A| \leq \frac{2p}{1\ [\text{m}]}$$

El cual queremos que sea inferior a 10^{-4}, por tanto:

$$\frac{2p}{1\ [\text{m}]} \leq 10^{-4} \ \Leftrightarrow \ p \leq (1/2)\ 10^{-4}\ [\text{m}] \ = \ 0{,}05\ [\text{mm}]$$

No obstante, si el cálculo se realiza en simple precisión, tenemos que $r_M = (1/2)\ 2^{-24+2} = 2^{-23}$

$$\Rightarrow |r_A| \leq \frac{2p}{1} + 3\ (2^{-23}) \leq 10^{-4} \Leftrightarrow p \leq (1/2)\left(10^{-4} - 3\ (2^{-23})\right)\ [\text{m}] = 0{,}0498\ [\text{mm}]$$

Por tanto, para que el cálculo se realice bien en un ordenador utilizando simple precisión, la regla debe medir longitudes con un error máximo de 0.0498 mm.

2.— Se desea dibujar en un plotter la curva $y = f(x)$ para valores de x comprendidos en el intervalo $[a, b]$. Para ello se confecciona la siguiente subrutina:

```
SUBROUTINE CURVA(A,B,N)
IMPLICIT REAL*4 (A-H,O-Z)
IMPLICIT INTEGER*4 (I-N)
DELTA=(B-A)/FLOAT(N)
X=A
Y=F(A)
CALL MOVE(X,Y)
DO I=1,N
  X=A+FLOAT(I)*DELTA
  Y=F(X)
  CALL DRAW(X,Y)
ENDDO
RETURN
END
```

donde la subrutina `MOVE(X,Y)` mueve la pluma sin dibujar hasta el punto de coordenadas (x, y), la subrutina `DRAW(X,Y)` mueve la pluma dibujando una línea recta desde la posición anterior hasta el punto de coordenadas (x, y), y `F(X)` es una función del tipo `REAL*4 FUNCTION` que se confecciona separadamente. Según el manual del compilador FORTRAN, para las variables de tipo `REAL*4` se destinan m =24 bits para almacenar la mantisa (incluido el signo).

Se pide:

- **a)** Explicar muy brevemente como funciona la subrutina.
- **b)** Hallar razonadamente (en primera aproximación) un valor máximo de N a partir del cual no se mejoren los resultados, debido a que la precisión del ordenador no permite discriminar entre dos valores consecutivos de x
- **c)** Hallar razonadamente (en primera aproximación) un valor máximo de N a partir del cual no se mejoren los resultados, debido a que la precisión del ordenador no permite discriminar entre dos valores consecutivos de $f(x)$
- **d)** Teniendo en cuenta además que el plotter interpreta las coordenadas (x, y) en centímetros, y que la resolución máxima de la pluma es de 0.1 milímetros, ¿cuál es el valor máximo de N que debe utilizarse en la práctica?

NOTA: Pueden efectuarse las simplificaciones que se juzguen razonables, ya que se habla de "primera aproximación", siempre y cuando se justifiquen.

Sol. 2.a)

El intervalo de estudio $[a, b]$ se discretiza en $n+1$ puntos equiespaciados una distancia $\delta = \dfrac{b-a}{n}$ y cada punto (x_i, y_i) se calcula como $x_i = a + \delta\, i,\ y_i = f(x_i);\ i = 0, \ldots, n$

La pluma se va moviendo de forma discreta por los puntos calculados. Gráficamente:

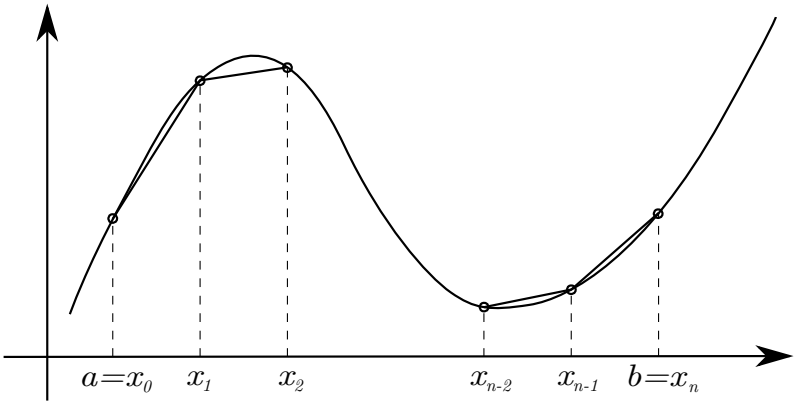

Figura 1: Esquema de dibujo del plotter mediante poligonal a trozos.

La subrutina dibuja una poligonal a trozos (ver figura 1).

Sol. 2.b)

Llamamos δ al espaciado entre dos puntos contiguos de forma que:

$$|x_{i+1} - x_i| = |\delta|$$

El error en el punto i será $E_{x_i} = x_i\, r_{x_i}$, pero en primera aproximación podemos despreciar la propagación de errores en el cálculo de x_i, es decir, suponemos $r_{x_i} \approx r^A_{x_i}$.

Sabemos además que:

$$\left|r^A_{x_i}\right| \leq r_M = \frac{1}{2}\, 2^{-24+2} = 2^{-23}$$

Luego:

$$|E_{x_i}| = |x_i\, r_{x_i}| \leq |x_i|\, r_M \leq \max_i |x_i|\ r_M$$

Buscamos los valores de δ que el ordenador es capaz de representar ya que no tiene sentido que $|E_{x_i}| > \delta$, por lo que exigimos que:

$$\max_i |x_i|\ r_M \ \leq\ |\delta| \ =\ \frac{|b-a|}{n}$$

por tanto:

$$\boxed{n \leq N_x = \frac{|b-a|}{\max\limits_i |x_i|} \frac{1}{r_M}; \quad \text{con } r_M \; = \; 2^{-23}}$$

Sol. 2.c)

Procedemos de la misma forma para los valores de la función:

$$|y_{i+1} - y_i| = |\mu_i| = |f(x_{i+1} - f(x_i)| \approx \left|f'(x_i)\,(x_{i+1} - x_i)\right| = \left|f'(x_i)\right| \; |\delta|$$

El error será: $E_{y_i} = y_i \, r_{y_i} = f(x_i) \, r_{y_i}$

En primera aproximación despreciamos los errores de almacenamiento de las operaciones intermedias en el cálculo de $f(x) \Rightarrow$

$$r_{y_i} \approx \frac{f'(x_i)}{f(x_i)} \, x_i \, r_{x_i} + r^A_{y_i} \qquad ; \qquad r_{x_i} \approx r^A_{x_i} \qquad ; \qquad \left\{ \begin{array}{l} \left|f^A_{x_i}\right| \; \leq \; r_M \\ \left|f^A_{y_i}\right| \; \leq \; r_M \end{array} \right.$$

Por tanto:

$$|E_{y_i}| \approx \left|f'(x_i) \, x_i \, r^A_{x_i} + f(x_i) \, r^A_{y_i}\right| \leq \left|f'(x_i) \, x_i + f(x_i)\right| \; r_M \leq \max_i \left|f'(x_i) \, x_i + f(x_i)\right| \; r_M$$

Exigimos que:

$$\max_i \left|f'(x_i) \, x_i + f(x_i)\right| \; r_M \leq \max_i |\mu_i| \approx \max_i \left|f'(x_i)\right| \; |\delta|$$

luego:

$$\max_i \left|f'(x_i) \, x_i + f(x_i)\right| \; r_M \leq \max_i \left|f'(x_i)\right| \frac{|b-a|}{n}$$

de modo que:

$$\boxed{n \leq N_y = \frac{|b-a| \; \max\limits_i |f'(x_i)|}{\max\limits_i |f'(x_i) \, x_i + f(x_i)|} \, \frac{1}{r_M}, \quad \text{con } r_M \; = \; 2^{-23}}$$

Sol. 2.d)

$$\left. \begin{array}{l} |x_{i+1} - x_i| = |\delta| \\ |y_{i+1} - y_i| = |\mu_i| \approx |f'(x_i)| \; |\delta| \end{array} \right\} \Rightarrow |\Delta\delta_i| = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \approx |\delta| \; \sqrt{1 + (f'(x_i))^2}$$

Exigimos que:

$$0{,}01\text{cm} \leq \max_i \sqrt{1 + (f'(x_i))^2} \; |\delta| = \max_i \sqrt{1 + (f'(x_i))^2} \; \frac{|b-a|}{n}$$

por tanto:

$$\boxed{n \leq N_\Delta = \frac{|b-a| \max\limits_i \sqrt{1 + (f'(x_i))^2}}{0{,}01 \text{ cm}}}$$

En la práctica:

$$\boxed{n \leq \text{mín}\{N_x,\, N_y,\, N_\Delta\}}$$

3.— Presentar diversos ejemplos patológicos de operaciones numéricas en los que se demuestre que en un ordenador digital:

a) No deben restarse números "parecidos".

b) No debe dividirse por números "pequeños".

c) Es preferible sumar en primer lugar los números más pequeños.

d) El orden de los factores altera el producto.

Indicar cuál es la justificación teórica de estas aseveraciones. Realizar los ejemplos en sistema decimal mediante una calculadora.

Realizamos los ejemplos con 3 cifras significativas en base 10. Denominamos $\widehat{z}$ al valor calculado correspondiente al valor exacto z.

Sol. 3.a)

Sean dos números a y b tales que:

$$r_a = r_a^I + r_a^A \qquad \text{y} \qquad r_b = r_b^I + r_b^A$$

La operación resta y su correspondiente error relativo serán:

$$z = a - b \qquad \rightarrow \qquad r_z = \frac{a}{a-b} r_a + \frac{-b}{a-b} r_b + r_z^A \Rightarrow$$

$$\Rightarrow r_z = \left[\frac{a}{a-b} r_a^I + \frac{-b}{a-b} r_b^I\right] + \left[\frac{a}{a-b} r_a^A + \frac{-b}{a-b} r_b^A + r_z^A\right]$$

Cuando los números son muy parecidos ($a \approx b$) los coeficientes de propagación de errores son muy grandes.

No deben restarse números parecidos porque el error relativo puede ser muy grande.

Ejemplo 1: $y = 1 - (1-x)(1+x)$ $\quad |x| << 1$ $\quad$ (3 cifras significativas en base 10)
para $x = 0{,}100\ 10^{-2}$:

$$\begin{cases} \widehat{1-x} = 0{,}999\ 10^0 \\ \widehat{1+x} = 0{,}100\ 10^1 \\ \widehat{(1-x)(1+x)} = 0{,}999\ 10^0 \\ \widehat{y} = 0{,}1\ 10^{-2} \end{cases}$$

Pero en realidad $y = 0{,}1\ 10^{-5}$, luego $r_y = \dfrac{y - \widehat{y}}{y} = -999 \approx -100000\,\%$

Estos problemas se pueden evitar simplemente operando la función:

$$y = 1 - (1-x)(1+x) = 1 - (1 - x^2) = x^2 \quad \Rightarrow \quad y = x^2$$

Ejemplo 2: $y = \tan(x) - \sin(x) \qquad |x| << 1$
para $x = 0{,}100\ 10^0$:

$$\begin{cases} \widehat{\tan(x)} = 0{,}100\ 10^0 \\ \widehat{\sin(x)} = 0{,}100\ 10^0 \\ \widehat{y} = 0{,}000\ 10^0 \end{cases}$$

Pero en realidad $y = 0{,}000501255$, luego $r_y = \dfrac{y - \widehat{y}}{y} = 1 \approx 100\,\%$
De nuevo se pueden evitar estos problemas operando con la expresión:

$$y = \tan(x) - \sin(x) = \sin(x)\left(\frac{1}{\cos(x)} - 1\right) = \sin(x)\ \frac{1 - \cos(x)}{\cos(x)} = \frac{\sin(x)\ (1 - \cos^2(x))}{\cos(x)\ (1 + \cos(x))} =$$

$$= \frac{\sin^3(x)}{\cos(x)\ (1 + \cos(x))}$$

También se podría utilizar una fórmula asintóticamente equivalente:

$$\sin(x) \approx x - \frac{x^3}{3!} + \theta(x^5)$$

$$\tan(x) \approx x + \frac{x^3}{3} + \hat{\theta}(x^5)$$

$$\Rightarrow y \approx \frac{x^3}{2}$$

Sol. 3.b)

Sean dos números a y b tales que:

$$r_a = r_a^I + r_a^A \qquad \text{y} \qquad r_b = r_b^I + r_b^A$$

La operación división y su correspondiente error relativo serán:

$$z = a/b \qquad \rightarrow \qquad r_z = \left(\frac{1}{1 - r_b}\right) r_a + \left(\frac{-1}{1 - r_b}\right) r_b + r_z^A \approx r_a - r_b + r_z^A \Rightarrow$$

Luego el error absoluto:

$$E_z = z\, r_z, \qquad E_z \approx \left[\frac{a}{b}(r_a^I - r_b^I)\right] + \left[\frac{a}{b}(r_a^A - r_b^A + r_z^A)\right]$$

Cuando $|b| << 1$ los coeficientes de propagación de errores en el error absoluto crecen mucho.

Sin embargo, en los resultados de los cálculos, el error absoluto no tiene demasiada importancia en general. Lo importante es que el error relativo sea pequeño. (Se exceptúan aquellos casos en los que los errores absolutos deban ser necesariamente pequeños, como en el caso del ejemplo del dibujo por ordenador donde el error absoluto en las coordenadas de los puntos es lo importante)

Hay un caso en que sí es importante el error absoluto: cuando el resultado es cero (ya que en este caso el error relativo no tiene sentido). Por este motivo es normal que la condición de convergencia de un algoritmo iterativo se escriba en términos de error relativo y en error absoluto máximo admisible tipo:

$$\text{Si } [|x_{k+1} - x_k| \leq \text{máx}\,(\varepsilon,\, r\,|x_{k+1}|)] \quad \rightarrow \quad \text{STOP}$$

Es preferible programar esta condición de esta forma en lugar de escribir

$$\text{Si } \left[\left|\frac{x_{k+1} - x_k}{x_{k+1}}\right| \leq r\right] \quad \rightarrow \quad \text{STOP}$$

Para evitar problemas cuando $|x_{k+1}| \rightarrow 0$

Sol. 3.c)

Es preferible sumar en primer lugar los números pequeños, para que su suma adquiera un valor significativo antes de añadirle los números grandes.

Ejemplo: $y = 0{,}1\ 10^1 + \overbrace{0{,}1\ 10^{-2} + 0{,}1\ 10^{-2} + ... + 0{,}1\ 10^{-2}}^{1000 \text{ veces}}$

Si los cálculos se realizan en este orden, se obtiene $\hat{y} = 0{,}1\ 10^1$, pero en realidad $y = 0{,}2\ 10^1$. Por tanto $r_y = \dfrac{y - \hat{y}}{y} = 0{,}5 = 50\,\%$

Este error se puede evitar simplemente con el orden de las operaciones, operando primero la suma de los términos más pequeños y después sumarlo al término de mayor orden.

$$y = \overbrace{0{,}1\ 10^{-2} + 0{,}1\ 10^{-2} + ... + 0{,}1\ 10^{-2}}^{1000 \text{ veces}} + 0{,}1\ 10^1$$

Sol. 3.d)

El orden de los factores altera el producto, ya que los resultados intermedios (y sus errores de almacenamiento) son diferentes.

Sean a, b y c tales que:

$$r_a = r_a^I + r_a^A \qquad , \qquad r_b = r_b^I + r_b^A \qquad , \qquad r_c = r_c^I + r_c^A$$

$$\Rightarrow \begin{cases} z_1 = (a\,b)\,c & \rightarrow \quad r_{z_1} \approx (r_a + r_b + r_{ab}^A) + r_c + r_{z_1}^A \\ z_2 = a\,(b\,c) & \rightarrow \quad r_{z_2} \approx r_a + (r_b + r_c + r_{bc}^A) + r_{z_2}^A \end{cases}$$

Y se puede observar que r_{ab} y r_{bc} pueden diferir, así como $r_{z_1}^A$ y $r_{z_2}^A$.

Ejemplo:

$$\begin{cases} z_1 = (0{,}5\ 10^0 \times 0{,}2\ 10^1) \times\ 0{,}999\ 10^0 & \rightarrow \quad \hat{z_1} = 0{,}999\ 10^0 \\ z_2 = 0{,}5\ 10^0 \times (0{,}2\ 10^1 \times\ 0{,}999\ 10^0) & \rightarrow \quad \hat{z_2} = 0{,}100\ 10^1 \end{cases}$$

4.— En un cálculo es preciso evaluar la función:

$$f(x) = 1 - (1 - x)(1 + x) = x^2$$

para valores de x tales que $|x| << 1$. Se utiliza un ordenador que utiliza m bits para el almacenamiento de la mantisa (incluido el signo) en coma flotante, y que redondea por aproximación.

a) ¿Se debe calcular $f(x)$ como $1-(1-x)(1+x)$ o como x^2? (En el último caso el cálculo se realiza multiplicando la variable x por sí misma).

b) Establecer las cotas del error relativo en los dos casos.

c) ¿Qué operación es mejor según el valor de x?

d) Presentar un ejemplo **numérico** de la conveniencia de usar uno u otro método cuando los cálculos se realizan manualmente con ayuda de una calculadora, y se redondea por aproximación con tres dígitos decimales significativos.

Sol. 4.a)

La función se debe calcular como $f(x) = x^2$, pues la fórmula original $f(x) = 1-(1-x)(1+x)$ producirá un gran error relativo cuando $|x| << 1$ ya que entonces el término $(1-x)(1+x) \approx 1$ y se restan números muy parecidos. Para valores de $|x|$ pequeños obtendremos como resultado el valor 0.

Sol. 4.b)

$$f_1(x) = 1-(1-x)(1+x) \Rightarrow \begin{cases} A = 1-x & \rightarrow \quad r_A = \dfrac{1}{1-x}\cancel{r_1} + \dfrac{-x}{1-x} r_x + r_A^A \\ B = 1+x & \rightarrow \quad r_B = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x} r_x + r_b^A \\ C = A * B & \rightarrow \quad r_C = r_A + r_B + r_C^A \\ F1 = 1-C & \rightarrow \quad r_{F1} = \dfrac{1}{1-C}\cancel{r_1} + \dfrac{-C}{1-C} r_C + r_{F1}^A \end{cases}$$

$$\Rightarrow r_{F1} = \frac{-(1-x)(1+x)}{1-(1-x)(1+x)} \left[\frac{-x}{1-x} r_x + r_A^A + \frac{x}{1+x} r_x + r_B^A + r_C^A \right] + r_{F1}^A$$

$$= \frac{-(1-x^2)}{x^2} \left[\frac{-2x^2}{1-x^2} r_x + r_A^A + r_B^A + r_C^A \right] + r_{F1}^A$$

$$= 2\, r_x - \frac{(1-x^2)}{x^2} [r_A^A + r_B^A + r_C^A] + r_{F1}^A$$

$$= 2\, r_x^I + \left(2\, r_x^A - \frac{(1-x^2)}{x^2} [r_A^A + r_B^A + r_C^A] + r_{F1}^A \right)$$

Podemos acotar los errores de almacenamiento por el error de máquina r_M:

$$|r_{F1}| \leq 2 \left| r_x^I \right| + \left(2 + \left| \frac{1-x^2}{x^2} \right| 3 + 1 \right) r_M$$

$$= 2 \left| r_x^I \right| + 3 \left(1 + \left| \frac{1-x^2}{x^2} \right| \right) r_M$$

$$|x| << 1 \Rightarrow |r_{F1}| \leq 2 \left| r_x^I \right| + 3 \left(1 + \frac{1-x^2}{x^2} \right) r_M = 2 \left| r_x^I \right| + \frac{3}{x^2} r_M$$

Por tanto,

$$\boxed{|r_{F1}| \leq 2\,|r_x^I| + \frac{3}{x^2}\,r_M \qquad \text{para } |x| << 1}$$

$$f_2(x) = x^2 \Rightarrow \left\{ \begin{array}{l} F2 = x * x \end{array} \right. \rightarrow \quad r_{F2} = r_x + r_x + r_{F2}^A$$

$$\Rightarrow r_{F2} = 2\,r_x + r_{F2}^A = 2r_x^I + 2r_x^A + r_{F2}^A$$

De nuevo acotamos los errores de almacenamiento por el error de máquina y obtenemos una cota del error relativo:

$$\boxed{|r_{F2}| \leq 2\ |r_x^I| + 3\,r_M}$$

Sol. 4.c)

Si nos fijamos en las cotas del error relativo para ambos casos, se puede ver claramente que siempre es mejor la fórmula $f(x) = x^2$ ya que ninguno de los términos del error se amplifica de forma exagerada. En cambio la fórmula original amplifica enormemente los errores de almacenamiento de los resultados intermedios A, B y C para valores de $|x| << 1$.

Sol. 4.d)

$$x = 0{,}100\ 10^{-2} \Rightarrow \left\{ \begin{array}{ll} \hat{f}_2(x) = 0{,}100\ 10^{-5} & \text{valor calculado correcto} \\ \hat{f}_1(x) = 0{,}100\ 10^{-2} & \text{valor calculado incorrecto} \end{array} \right.$$

5.— Las n primeras potencias φ^1, φ^2,...,φ^n del número áureo $\varphi = (-1+\sqrt{5})/2$ pueden obtenerse sin realizar ninguna multiplicación, ya que se cumple la relación:

$$\varphi^i = \varphi^{i-2} - \varphi^{i-1}$$

Demostrar que esta forma de efectuar los cálculos es numéricamente inestable, mientras que la multiplicación reiterada es numéricamente estable.

Sol. 5.

Algoritmo inestable: $\qquad \varphi \;=\; (-1+\sqrt{5})/2 \;\approx\; 0{,}61803398875$

$$\left\{ \begin{array}{ll} x_0 = 1 & r_{x_0} = \cancel{r_{x_0}^I} + \cancel{r_{x_0}^A} = 0 \\ x_1 = \varphi & r_{x_1} = r_{x_1}^I + r_{x_1}^A \\ x_2 = x_0 - x_1 & r_{x_2} = \dfrac{x_0}{x_0 - x_1} r_{x_0} + \dfrac{-x_1}{x_0 - x_1} r_{x_1} + r_{x_2}^A = \dfrac{1}{\varphi^2} r_{x_0} - \dfrac{1}{\varphi} r_{x_1} + r_{x_2}^A \\ x_3 = x_1 - x_2 & r_{x_3} = \dfrac{x_1}{x_1 - x_2} r_{x_1} + \dfrac{-x_2}{x_1 - x_2} r_{x_2} + r_{x_3}^A = \dfrac{1}{\varphi^2} r_{x_1} - \dfrac{1}{\varphi} r_{x_2} + r_{x_3}^A \\ \quad \vdots & \quad \vdots \\ x_k = x_{k-2} - x_{k-1} & r_{x_k} = \dfrac{x_{k-2}}{x_{k-2} - x_{k-1}} r_{x_{k-2}} + \dfrac{-x_{k-1}}{x_{k-2} - x_{k-1}} r_{x_{k-1}} + r_{x_k}^A = \dfrac{1}{\varphi^2} r_{x_{k-2}} - \dfrac{1}{\varphi} r_{x_{k-1}} + r_{x_k}^A \end{array} \right.$$

Para simplificar las expresiones cambiamos la notación $R_k = r_{x_k}$ y $r_{k-1} = r^A_{x_k}$, de modo que:

$$\Rightarrow \begin{cases} R_2 = \varphi^{-2}R_0 - \varphi^{-1}R_1 + r_1 = -\varphi^{-1}R_1 + r_1 \\ R_3 = \varphi^{-2}R_1 - \varphi^{-1}R_2 + r_2 = 2\varphi^{-2}R_1 - \varphi^{-1}r_1 + r_2 \\ R_4 = \varphi^{-2}R_2 - \varphi^{-1}R_3 + r_3 = -3\varphi^{-3}R_1 + 2\varphi^{-2}r_1 - \varphi^{-1}r_2 + r_3 \\ R_5 = \varphi^{-2}R_3 - \varphi^{-1}R_4 + r_4 = 5\varphi^{-4}R_1 - 3\varphi^{-3}r_1 + 2\varphi^{-2}r_2 - \varphi^{-1}r_3 + r_4 \\ \vdots \\ R_k = \varphi^{-2}R_{k-2} - \varphi^{-1}R_{k-1} + r_{k-1} = (-1)^{k-1}F_k\,\varphi^{-(k-1)}R_1 + \sum_{i=1}^{k-1}(-1)^{k-1-i}F_{k-i}\,\varphi^{-(k-1-i)}r_i \end{cases}$$

siendo F_j el término $j-$ésimo de la sucesión de Fibonacci $= \{1, 1, 2, 3, 5, 8, ...\}$

De modo que si $R_1 = r^I_\varphi + r_0$, entonces:

$$R_k = (-1)^{k-1}F_k\,\varphi^{-(k-1)}r^I_\varphi + \sum_{i=0}^{k-1}(-1)^{k-1-i}F_{k-i}\,\varphi^{-(k-1-i)}r_i$$

$$\Rightarrow |R_k| \leq \varphi^{-(k-1)}\left[F_k\left|r^I_\varphi\right| + \sum_{i=0}^{k-1}F_{k-i}\,\varphi^i\,|r_i|\right]$$

Teniendo en cuenta que $r_i = r^A_{x_{i+1}} \rightarrow |r_i| \leq r_M$ entonces:

$$\boxed{|R_k| \leq \frac{1}{\varphi^{(k-1)}}\left[F_k\left|r^I_\varphi\right| + \left(\sum_{i=0}^{k-1}F_{k-i}\,\varphi^i\right)r_M\right]}$$

Es decir:

$$|R_k| \leq A_k\left|r^I_\varphi\right| + B_k\,r_M \begin{cases} A_k = \dfrac{F_k}{\varphi^{k-1}} \\ B_k = \left(\displaystyle\sum_{i=0}^{k-1}\frac{F_{k-i}\,\varphi^i}{\varphi^{k-1}}\right) = \sum_{i=0}^{k-1}\frac{F_{k-i}}{\varphi^{k-i-1}} = \sum_{i=0}^{k-1}A_{k-i} = \sum_{j=1}^{k}A_j \end{cases}$$

Y se puede comprobar fácilmente que los coeficientes A_k y B_k crecen fuertemente a medida que aumenta el valor de k (tabla 1):

Tabla 1: Valores de los coeficientes de la serie.

k	A_k	B_k
1	1	1
5	≈34	≈55
10	≈4181	≈6765
20	≈63 10^6	≈102 10^6

Luego, tanto el error inherente en el dato inicial como los errores de almacenamiento de las operaciones intermedias se amplifican de forma descontrolada ⇒ INESTABLE.

NOTA:

Puede demostrarse que el término k-ésimo de la sucesión de Fibonacci vale:

$$F_k = \frac{(1+\varphi)^k - (-\varphi)^k}{1+2\,\varphi}, \qquad \text{con } \varphi = \frac{-1+\sqrt{5}}{2}$$

y que, por inducción,

$$\begin{aligned} &F_1 = 1 \\ &F_2 = 1 \\ &\dots \\ &F_k = F_{k-2} + F_{k-1} \qquad \forall k \geq 3 \end{aligned}$$

Luego,

$$A_k = \frac{1}{1+2\varphi}\left[\frac{(1+\varphi)^k}{\varphi^{k-1}} - \frac{(-\varphi)^k}{\varphi^{k-1}}\right] = \frac{\varphi}{1+2\varphi}\left[\left(\frac{1+\varphi}{\varphi}\right)^k - (-1)^k\right] = \frac{\varphi}{1+2\varphi}\left[\underbrace{\left(1+\frac{1}{\varphi}\right)}_{\approx 2{,}618}^k - (-1)^k\right]$$

De modo que el término crece exponencialmente.

$$\begin{aligned} B_k &= \sum_{j=1}^{k} A_j = \frac{\varphi}{1+2\varphi}\sum_{j=1}^{k}\left[\left(1+\frac{1}{\varphi}\right)^j - (-a)^j\right] \\ &= \frac{\varphi}{1+2\varphi}\left[\frac{\left(1+\frac{1}{\varphi}\right) - \left(1+\frac{1}{\varphi}\right)^{k+1}}{1-\left(1+\frac{1}{\varphi}\right)} - \frac{(-1)-(-1)^{k+1}}{1-(-1)}\right] \\ &= \frac{\varphi}{1+2\varphi}\left[\varphi\left(1+\frac{1}{\varphi}\right)\left(\left(1+\frac{1}{\varphi}\right)^k - 1\right) + \frac{1-(-1)^k}{2}\right] \\ &= \frac{\varphi}{1+2\varphi}\left[(1+\varphi)\left(\underbrace{\left(1+\frac{1}{\varphi}\right)}_{\approx 2{,}618}^k - 1\right) + \frac{1-(-1)^k}{2}\right] \end{aligned}$$

De modo que el término también crece exponencialmente.

Algoritmo estable: $\qquad \varphi = (-1+\sqrt{5})/2 \approx 0{,}61803398875$

$$\begin{cases} x_0 = 1 & r_{x_0} = \cancel{r^I_{x_0}} + \cancel{r^A_{x_0}} = 0 \\ x_1 = \varphi & r_{x_1} = r^I_{x_1} + r^A_{x_1} \\ x_2 = x_1\ x_1 & r_{x_2} = r_{x_1} + r_{x_1} + r^A_{x_2} = 2\ r_{x_1} + r^A_{x_2} \\ x_3 = x_2\ x_1 & r_{x_3} = r_{x_2} + r_{x_1} + r^A_{x_3} = 3\ r_{x_1} + r^A_{x_2} + r^A_{x_3} \\ \vdots & \vdots \\ x_k = x_{k-1}\ x_1 & r_{x_k} = r_{x_{k-1}} + r_{x_1} + r^A_{x_k} = k\ r_{x_1} + \sum_{i=2}^{k} r^A_{x_i} \end{cases}$$

Luego:

$$r_{x_k} = k\ r^I_{x_1} + \left(k\ r^A_{x_1} + \sum_{i=2}^{k} r^A_{x_i}\right)$$

$$\Rightarrow |r_{x_k}| \leq k \ \left|r^I_{x_1}\right| + k \ \left|r^A_{x_1}\right| + \sum_{i=2}^{k} \left|r^A_{x_i}\right|$$

$$\Rightarrow |r_{k_k}| \leq k \ \left|r^I_{x_1}\right| + k \ r_M + (k-1) \ r_M$$

$$\Rightarrow \boxed{|r_{x_k}| \leq k \ \left|r^I_{x_1}\right| + (2k-1) \ r_M}$$

En el peor de los casos, el error inherente y los errores de almacenamiento de las operaciones intermedias crecen de forma lineal ⇒ ESTABLE.

Además la cota superior será muy conservadora, con lo que el error relativo será pequeño en general, incluso después de realizar un número elevado de iteraciones.

6.— Se desea calcular aproximadamente e^x para diversos valores de x en el intervalo [0,10] sumando los primeros n términos de la serie de Taylor. El cálculo se efectuará en un ordenador que utiliza 24 bits para la mantisa en coma flotante (incluyendo el signo). Efectuar un estudio sencillo (sin tener en cuenta el efecto de la propagación del error de redondeo) que establezca un valor máximo del número de términos a considerar, por encima del cuál no se obtengan mejoras apreciables en la aproximación.

Sol. 6.

El desarrollo en serie de Taylor de la función e^x es:

$$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!} + R_n(x) \qquad ; \qquad R_n(x) = \frac{e^\xi}{(n+1)!} x^{n+1} \qquad \xi \in [0, x]$$

Si tenemos en cuenta que no tendremos en cuenta la propagación del error en las operaciones, podemos plantear la resolución del problema mediante dos criterios:

1) No tiene sentido seguir sumando términos cuando el nuevo término es del orden del error de máquina o inferior:

$$\left|\frac{x^{n+1}}{(n+1)!}\right| \ \leq \ |e^x| \, r_M$$

$$x > 0 \quad \Rightarrow \quad \frac{e^{-x} \, x^{n+1}}{(n+1)!} \ \leq \ r_M$$

$$\max_{x \in [0,10]} \frac{e^{-x} \, x^{n+1}}{(n+1)!} = \begin{cases} \dfrac{e^{-(n+1)} \, (n+1)^{n+1}}{(n+1)!} & \text{para } x = n+1 \leq 10 \\ \dfrac{e^{-10} \, 10^{n+1}}{(n+1)!} & \text{para } x = 10 \leq n+1 \end{cases}$$

Luego, no tiene sentido seguir cuando:

$$\boxed{\frac{e^{-Z_n} Z_n^{n+1}}{(n+1)!} \leq \frac{1}{2} \, 2^{-m+2} \qquad \text{con} \qquad \begin{cases} Z_n = n+1 & \text{si} \quad n \leq 9 \\ Z_n = 10 & \text{si} \quad n \geq 9 \end{cases}}$$

2) No tiene sentido seguir sumando términos cuando el error de truncamiento es del orden del error de máquina o inferior:

$$\left|\frac{e^{\xi}\, x^{n+1}}{(n+1)!}\right| \;\leq\; |e^{x}|\, r_{M}$$

$$x>0 \quad \Rightarrow \quad \frac{e^{\xi-x}\, x^{n+1}}{(n+1)!} \;\leq\; r_{M}$$

$$\max_{x\in[0,10],\,\xi\in[0,x]} \frac{e^{\xi-x}\, x^{n+1}}{(n+1)!} = \frac{10^{n+1}}{(n+1)!}$$

Luego, no tiene sentido seguir cuando:

$$\boxed{\frac{10^{n+1}}{(n+1)!} \;\leq\; \frac{1}{2}\, 2^{-m+2}}$$

7.— En un proceso de cálculo se requiere evaluar repetidas veces la función $f(x) = \sqrt{1+x} - 1$. Es sabido, que para valores pequeños de x, existe, entre otras, la aproximación asintótica a la función anterior $f(x) \approx f_0(x) = x/2$. Los cálculos se efectuarán en un ordenador digital. ¿Es posible que los resultados obtenidos empleando la aproximación asintótica f_0 sean mejores que los resultados de los cálculos obtenidos empleando la propia función $f(x)$? En caso afirmativo: ¿para que rango de valores de x?; ¿cuál sería este rango si se utiliza precisión simple (24 bits para la mantisa, incluido el signo)? Ya sea el caso afirmativo o negativo, presentar diversos ejemplos numéricos que corroboren el resultado, empleando la base decimal y realizando las operaciones en coma flotante con tres dígitos.

Sol. 7.

Es posible que la fórmula $f_0(x)$ dé resultados más aproximados que la fórmula original para valores de $|x|$ muy pequeños, pues el error de truncamiento de $f_0(x)$ es cada vez más pequeño cuando $|x| \to 0$, mientras que el error de rendondeo al calcular $f(x)$ puede ser muy grande cuando $|x| \to 0$, debido a que $1+x \approx 1$. De hecho, para valores de $|x|$ suficientemente pequeños esta fórmula dará un valor igual a 0, lo que supone un 100 % de error relativo.

Analizamos la propagación de errores y los errores de truncamiento:

1) $\quad f_1(x) = \sqrt{1+x} - 1$

$$\begin{cases} Y = 1+x & \rightarrow \quad r_Y = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x} r_x + r_Y^A \\ Z = \sqrt{Y} & \rightarrow \quad r_Z = \dfrac{1}{2}\, r_Y + r_Z^A \\ F1 = Z - 1 & \rightarrow \quad r_{F1} = \dfrac{Z}{Z-1} r_Z + \dfrac{-1}{Z-1}\cancel{r_1} + r_{F1}^A \end{cases}$$

$$\begin{aligned}\Rightarrow r_{F1} &= \frac{\sqrt{1+x}}{\sqrt{1+x}-1}\left(\frac{1}{2}\left(\frac{x}{1+x}r_x + r_Y^A\right) + r_Z^A\right) + r_{F1}^A \\ &= \left(\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\,\frac{1}{2}\,\frac{x}{1+x}r_x^I\right) + \left[\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\left(\frac{1}{2}\left(\frac{x}{1+x}r_x^A + r_Y^A\right) + r_Z^A\right) + r_{F1}^A\right]\end{aligned}$$

$$|r_{F1}| \leq \left|\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\,\frac{1}{2}\,\frac{x}{1+x}\right|\,|r_x^I| + \left[\left|\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\right|\left(\frac{1}{2}\left(\left|\frac{x}{1+x}\right|+1\right)+1\right)+1\right] r_M$$

2) $f_0(x) = x/2$

$\{F0 = x/2 \quad \rightarrow \quad r_{F0} = r_x + \cancel{r_2} + \cancel{r_{F0}^A}^{\,0}$ (dividir por potencias de 2 en binario solo modifica el exponente y por tanto no introduce un nuevo error de almacenamiento)

De modo que:

$$r_{F0} \;=\; r_x \;=\; r_x^I + r_x^A$$

Pero lo que buscamos no es $r_{F0} = \dfrac{f_0(x) - \widehat{f}_0(x)}{f_0(x)}$.

Tenemos que tener en cuenta el error de truncamiento dado que estamos utilizando una aproximación a la función que realmente queremos evaular. El error relativo total es:

$$r_{F0}^* = \frac{f(x) - \widehat{f}_0(x)}{f(x)} = \frac{f(x) - f_0(x)(1 - r_{F0})}{f(x)} = \underbrace{\frac{f(x) - f_0(x)}{f(x)}}_{\text{truncamiento}} + \underbrace{\frac{f_0(x)}{f(x)}}_{\approx 1}\overbrace{r_{F0}}^{\text{redondeo}}$$

Por tanto:

$$\begin{aligned}r_{F0}^* &= \frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)} + \frac{x/2}{(\sqrt{1+x}-1)}(r_x^I + r_x^A) \\ &= \frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)} + \frac{x/2}{(\sqrt{1+x}-1)}r_x^I + \frac{x/2}{(\sqrt{1+x}-1)}r_x^A\end{aligned}$$

$$|r_{F0}^*| \leq \left|\frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)}\right| + \left|\frac{x/2}{(\sqrt{1+x}-1)}\right||r_x^I| + \left|\frac{x/2}{(\sqrt{1+x}-1)}\right| r_M$$

Estudiamos que ocurre en ambos casos cuando $|x| << 1 \Rightarrow \begin{cases}(1+x) \rightarrow 1 \\ (\sqrt{1+x}) \rightarrow 1 \\ (\sqrt{1+x}-1) \rightarrow x/2 \\ ((\sqrt{1+x}-1) - x/2) \rightarrow -x^2/8\end{cases}$.

Luego:

$$\begin{cases} |r_{F1}| \leq R_1 \approx \left|\dfrac{1}{x/2}\,\dfrac{1}{2}\,x\right| \left|r_x^I\right| + \left[\left|\dfrac{1}{x/2}\right|\left(\dfrac{1}{2}\left(|x|+1\right)+1\right)+1\right] r_M = \left|r_x^I\right| + \left(2+\dfrac{3}{|x|}\right) r_M \\ |r_{F0}^*| \leq R_0 \approx \left|\dfrac{-x^2/8}{x/2}\right| + \left|\dfrac{x/2}{x/2}\right| \left|r_x^I\right| + \left|\dfrac{x/2}{x/2}\right| r_M = |x/4| + \left|r_x^I\right| + r_M \end{cases}$$

La evaluación de la función como $f_0(x)$ será mejor siempre y cuando $R_0 \leq R_1$, es decir:

$$\left|\frac{x}{4}\right| + \left|r_x^I\right| + r_M \leq \left|r_x^I\right| + \left[2+\frac{3}{|x|}\right] r_M$$

$$\Rightarrow \left|\frac{x}{4}\right| \leq \left[1+\frac{3}{|x|}\right] r_M$$

Que se cumplirá cuando:

$$\left|\frac{x}{4}\right| \leq \frac{3}{|x|} r_M$$

De modo que:

$$x^2 \leq 12\, r_M \;\Leftrightarrow\; |x| \leq 2\sqrt{3}\,\sqrt{r_M}$$

Para valores de $|x| \leq 2\sqrt{3}\,\sqrt{r_M}$ la fórmula $f_0(x)$ tiene una cota de error menor que la de la función original $f(x)$ a pesar del error de truncamiento.

Precisión simple $\Rightarrow$ $m = 24$ bits $\Rightarrow$ $r_M = 2^{-23}$. Por tanto la fórmula $f_0(x)$ será mejor para valores $|x| \leq 1{,}19604\,10^{-3}$

En una calculadora de 10 cifras:

$$r_M = 1/2\,10^{-(10+1)+2} = 1/2\,10^{-9} \;\Rightarrow\; |x| \;\leq\; 7{,}74597\,10^{-5}$$

$$\begin{cases} x = 0{,}0001 \rightarrow & f_1(x) = 0{,}4998750\,10^{-4}, & f_0(x) = 0{,}5\,10^{-4} \\ x = 0{,}00001 \rightarrow & f_1(x) = 0{,}5\,10^{-5}, & f_0(x) = 0{,}5\,10^{-5} \end{cases}$$

En una calculadora de 3 cifras:

$$r_M = 1/2\,10^{-(3+1)+2} = 1/2\,10^{-2} \;\Rightarrow\; |x| \;\leq\; 0{,}244949$$

8.— En un proceso de cálculo es necesario evaluar la función $f(x) = e^{-1/x}$ para diversos valores de $x \in [0{,}05, 0{,}10]$ con un error relativo $r_f \leq 10^{-8}$. Si los valores de x se conocen con un error relativo $r_x \leq 10^{-5}$, y suponiendo que podamos aumentar la precisión del ordenador hasta donde sea necesario, ¿puede obtenerse un resultado satisfactorio? En caso contrario, ¿con qué precisión sería necesario conocer los datos?

En cualquier caso, ¿qué tipo de precisión –simple o doble– debe usarse si los cálculos se efectúan en un ordenador digital? Se supondrá que en simple y en doble precisión se utilizan 24 y 53 bits, respectivamente, para almacenar la mantisa (incluido el signo).

Sol. 8.

Si suponemos que el ordenador tiene precisión infinita esto implica que todos los errores de almacenamiento son nulos y por tanto $r_x = r_x^I$.

De esta forma, el error en la evaluación de la función $f(x) = e^{-1/x}$ será:

$$r_f = \frac{f'(x)}{f(x)} x\, r_x = \frac{e^{-1/x}(1/x^2)}{e^{-1/x}} x\, r_x = \frac{r_x}{x} = \frac{r_x^I}{x} \Rightarrow |r_f| = \frac{\left|r_x^I\right|}{|x|}$$

Dado que $\left|r_x^I\right| \leq 10^{-5}$ y que $x \in [0{,}05,\ 0{,}10]$, la cota superior del error será:

$$|r_f| \leq \frac{10^{-5}}{0{,}05} = 0{,}2\ 10^{-3}$$

Por tanto, no podemos asegurar que $|r_f| \leq 10^{-8}$

Ejemplo.

$$\left.\begin{array}{l} x = 0{,}05 \ \rightarrow\ f(x) = 2{,}0611536\ 10^{-9} \\ \widehat{x} = 0{,}05\ (1 - 10^{-5}) \ \rightarrow\ f(\widehat{x}) = 2{,}0607414\ 10^{-9} \end{array}\right\} \Rightarrow \frac{f(x) - f(\widehat{x})}{f(x)} = 0{,}19998\ 10^{-3}$$

Para lograr la precisión buscada necesitamos que el error en el dato inicial sea:

$$|r_f| = \frac{\left|r_x^I\right|}{|x|} \leq 10^{-8} \Rightarrow \left\{\begin{array}{l} \left|r_x^I\right| \leq |x|\ 10^{-8} \\ x \in [0{,}05,\ 0{,}10] \end{array}\right\} \Rightarrow \left|r_x^I\right| \leq 0{,}05\ 10^{-8} = 0{,}5\ 10^{-9}$$

Analizamos ahora la precisión que tendríamos que utilizar en un ordenador digital para obtener la precisión deseada:

$$\left\{\begin{array}{llllllll} \text{Simple precisión} & \rightarrow & m = 24 & \rightarrow & r_M = 1/2\ 2^{-24+2} & = & 2^{-23} & \approx\ 0{,}119\ 10^{-6} \\ \text{Doble precisión} & \rightarrow & m = 53 & \rightarrow & r_M = 1/2\ 2^{-53+2} & = & 2^{-22} & \approx\ 0{,}222\ 10^{-15} \end{array}\right\}$$

Simple precisión no será suficiente pues ni siquiera podemos almacenar el dato x con seguridad ya que $|r_x| \leq \left|r_x^I\right| + r_M$, y ya solo el error de máquina $r_M > 10^{-8}$.

Necesitaríamos utilizar doble precisión. Y esperamos que la doble precisión sea suficiente.

Estudio completo

$$\left\{\begin{array}{lll} & r_x = r_x^I + r_x^A & ;\ \left|r_x^A\right| \leq r_M \\ Y = -1/X & r_y = 1\ \cancel{r_1}^{\,0} - 1\ r_x + r_y^A & ;\ \left|r_y^A\right| \leq r_M \\ F = EXP(Y) & r_f = \dfrac{g'(y)}{g(y)} y\, r_y + r_f^A = \dfrac{\cancel{e^y}}{\cancel{e^y}} y\, r_y + r_f^A & ;\ \left|r_f^A\right| \leq r_M \end{array}\right.$$

$$\begin{aligned} r_f &= \frac{-1}{x}\left(-(r_x^I + r_x^A) + r_y^A\right) + r_f^A \\ &= \frac{1}{x} r_x^I + \left(\frac{1}{x} r_x^A - \frac{1}{x} r_y^A + r_f^A\right) \end{aligned}$$

$$|r_f| \leq \frac{1}{|x|}\left|r_x^I\right| + \left(\frac{1}{|x|} + \frac{1}{|x|} + 1\right) r_M = \frac{1}{|x|}\left|r_x^I\right| + \left(\frac{2}{|x|} + 1\right) r_M$$

En doble precisión $r_M = 2^{-52} = 0{,}222\ 10^{-15}$ y si consideramos que $x \in [005,\ 0{,}10]$ entonces se puede acotar el error como:

$$|r_f| \leq \frac{|r_x^I|}{0{,}05} + \left(\frac{2}{0{,}05} + 1\right) 2^{-52}$$

Queremos estar seguros de que $|r_f| \leq 10^{-8}$, de modo que:

$$\boxed{|r_x^I| \leq 0{,}499999545\ 10^{-9}}$$

Luego, efectivamente, el cálculo es posible en doble precisión. Obsérvese que la precisión requerida para el dato x es algo mayor que la que se obtuvo anteriormente sin considerar errores de almacenamiento en las operaciones.

9.— En un proceso de cálculo es preciso evaular varios millones de veces la función $f(x) = \ln(1+x)$, siempre para valores de x positivos ($x > 0$) y pequeños ($x << 1$). La función del compilador Fortran de la que se dispone es muy precisa, pero requiere un tiempo de computación relativamente elevado (del orden del equivalente a varias decenas de operaciones elementales).

Para reducir el tiempo de computación se valora la posibilidad de aproximar la función utilizando los dos primeros términos de su desarrollo de Taylor, admitiéndose esta simplificación cuando el error relativo sea inferior (en valor absoluto) a cien veces el error de almacenamiento de la máquina. Se confeccionan las siguientes subrutinas:

```
SUBROUTINE EXACTA(X,Y1)
REAL*4 X,Z,Y1
Z = 1. + X
Y1 = ALOG(Z)
RETURN
END
```

```
SUBROUTINEAPROX(X,Y2)
REAL*4 X,T,Y2
T = 1. - X/2.
Y2 = X*T
RETURN
END
```

Sabiendo que el ordenador trabaja en coma flotante redondeando por aproximación y destinando m=24 bits a la mantisa (incluido 1 bit para el signo), y que los valores de `X` son exactos (en el sentido de que su error inherente es nulo, aunque en general estarán afectados del correspondiente error de almacenamiento), se pide:

a) Analizar instrucción por instrucción las subrutinas anteriores, obteniendo los errores relativos de las variables en cada paso.

b) Obtener el error relativo del valor `Y1` calculado mediante la subrutina `EXACTA` respecto al valor exacto de $f(x)$. Obtener una cota del error y explicar qué sucede cuando x tiende a cero.

c) Obtener el error relativo del valor `Y2` calculado mediante la subrutina `APROX` respecto al valor exacto de $f(x)$. Obtener una cota del error y explicar qué sucede cuando x tiende a cero.

d) Discutir para qué rango de valores de `X` puede utilizarse la subrutina `APROX` en lugar de la subrutina `EXACTA`.

e) ¿Es posible que la subrutina `APROX` proporcione resultados más precisos que la subrutina `EXACTA`? En caso afirmativo, ¿para qué valores de `X`?

Nota: El desarrollo de Taylor (con el resto de Lagrange) de la función $f(x)$ es:

$$f(x) = -\sum_{i=1}^{n} (-1)^i \frac{x^i}{i} + R_n(x) \qquad ; \qquad R_n(x) = -1 \left(\frac{-1}{1+\xi}\right)^{n+1} \frac{x^{n+1}}{n+1} \qquad \xi \in (0,x)$$

Sol. 9.a)

Dado que en el enunciado se indica que el error inherente del dato es nulo, entonces:

$$r_x^I = 0 \;\Rightarrow\; r_x = r_x^A \qquad \left|r_x^A\right| \leq r_M$$

EXACTA:

$$\begin{cases} Z = 1. + X & \rightarrow \quad r_z = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x}\, r_x + r_z^A \\ Y1 = \ln(Z) & \rightarrow \quad r_{y1} = \dfrac{1/z}{\ln(z)}\, z\, r_z + r_{y1}^A \end{cases}$$

$$\Rightarrow \boxed{r_{y1} = \frac{1}{\ln(1+x)} \left[\frac{x}{1+x}\, r_x^A + r_z^A\right] + r_{y1}^A}$$

APROXIMADA:

$$\begin{cases} T = 1. - X/2 & \rightarrow \quad r_t = \dfrac{1}{1-x/2}\cancel{r_1} + \dfrac{-x/2}{1-x/2}\, r_x + r_t^A \\ Y2 = X * T & \rightarrow \quad r_{y2} = r_x + r_t + r_{y2}^A \end{cases}$$

$$\Rightarrow \boxed{r_{y2} = \left(1 - \frac{x/2}{1-x/2}\right) r_x^A + r_t^A + r_{y2}^A}$$

Téngase en cuenta que en esta expresión del error se ha considerado que multiplicar por potencias enteras de la base de numeración ($B = 2$) no introduce un nuevo error en la operación de división ni de almacenamiento.

Sol. 9.b)

$$\begin{aligned} \text{Conocemos } r_{y1} &= \frac{y_1 - \hat{y_1}}{y_1} \text{ siendo } \hat{y_1} \text{ el valor calculado} \\ \text{Buscamos } r_{y1}^* &= \frac{f(x) - \hat{y_1}}{f(x)} = \underbrace{\frac{f(x) - y_1}{f(x)}}_{\text{truncamiento}} + \underbrace{\frac{y_1}{f(x)}}_{\approx 1} \underbrace{\frac{y_1 - \hat{y_1}}{y_1}}_{\text{redondeo}} \end{aligned}$$

Como la fórmula es exacta, $f(x) = y_1$, por tanto:

$$r_{y1}^* = r_{y1} = \frac{1}{\ln(1+x)} \left[\frac{x}{1+x}\, r_x^A + r_z^A\right] + r_{y1}^A$$

$$\left|r_{y1}^*\right| \leq \left[\frac{1}{|\ln(1+x)|} \left(\left|\frac{x}{1+x}\right| + 1\right) + 1\right] r_M \equiv R_1$$

$$\text{cuando} \left\{ \begin{array}{l} x \to 0 \\ x > 0 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} 1+x \to 1 \\ \ln(1+x) \to x \end{array} \right\} \Rightarrow R_1 \approx \left(2 + \frac{1}{|x|}\right) r_M \Rightarrow R_1 \to \infty$$

En la práctica, cuando $x \to 0$ la fórmula dará como resultado 0, pues x es insignificante frente a 1, con lo que el error tiende al 100 % (no a ∞).

Sol. 9.c)

$$\begin{aligned} \text{Conocemos } r_{y2} &= \frac{y_2 - \hat{y_2}}{y_2} \text{ siendo } \hat{y_2} \text{ el valor calculado} \\ \text{Buscamos } r^*_{y2} &= \frac{f(x) - \hat{y_2}}{f(x)} = \underbrace{\frac{f(x) - y_2}{f(x)}}_{\text{truncamiento}} + \underbrace{\frac{y_2}{f(x)}}_{\approx 1} \underbrace{\frac{y_2 - \hat{y_2}}{y_2}}_{\text{redondeo}} \end{aligned}$$

$$\text{Sabemos que } f(x) = \underbrace{x - \frac{x^2}{2}}_{y_2} + \underbrace{\left(\frac{1}{1+\xi}\right)^3 x^3/3}_{R_2(x)} \quad \text{con} \quad \xi \in [0, x] \qquad \text{(Desarrollo de Taylor)}$$

Luego el error relativo de truncamiento será $\dfrac{f(x) - y_2}{f(x)} = \dfrac{R_2(x)}{f(x)}$, por tanto:

$$r^*_{y2} = \frac{\dfrac{1}{(1+\xi)^3}\dfrac{x^3}{3}}{\ln(1+x)} + \frac{(x - x^2/2)}{\ln(1+x)} r_{y2}$$

$$= \frac{1}{(1+\xi)^3}\frac{x^3/3}{\ln(1+x)} + \frac{(x - x^2/2)}{\ln(1+x)}\left[\frac{1-x}{1-x/2} r^A_x + r^A_t + r^A_{y2}\right]$$

$$= \frac{1}{(1+\xi)^3}\frac{x^3/3}{\ln(1+x)} + \frac{1}{\ln(1+x)}\left[x(1-x)\, r^A_x + x(1-x/2)\, r^A_t + x(1-x/2)\, r^A_{y2}\right]$$

$$\left|r^*_{y2}\right| \le \left|\frac{1}{(1+\xi)^3}\right| \frac{\left|x^3/3\right|}{|\ln(1+x)|} + \frac{|x|}{|\ln(1+x)|}\left[|1-x| + |1-x/2| + |1-x/2|\right] r_M$$

Teniendo en cuenta que $x > 0$, $x << 1$ y que $\xi \in [0, x]$:

$$\left|r^*_{y2}\right| \le \frac{x^3/3}{\ln(1+x)} + \frac{x(3-2x)}{\ln(1+x)} r_M \equiv R_2$$

$$\text{cuando} \left\{ \begin{array}{l} x \to 0 \\ x > 0 \end{array} \right\} \Rightarrow \{\ln(1+x) \approx x\} \Rightarrow R_2 \approx \frac{x^3/3}{x} + \frac{x(3-2x)}{x} r_M = \frac{x^2}{3} + (3-2x)\, r_M$$

$$R_2 \to 3\, r_M \qquad \text{cuando} \qquad x \to 0$$

Sol. 9.d)

Tal y como se indica se aceptará la validez de la subrutina `APROX` cuando el error relativo sea inferior a cien veces el error de máquina:

$$\frac{x^2}{3} + (3-2x)\, r_M \le 100\, r_M \Rightarrow x^2 \le 3(97+2x)\, r_M \Rightarrow \boxed{x \le \sqrt{291\ 2^{-23}} \approx 0{,}006}$$

Sol. 9.e)

La subrutina `APROX` funcionará mejor que `EXACTA` cuando:

$$\frac{x^2}{3} + (3 - 2x)\, r_M \leq (2 + \frac{1}{x})\, r_M \Leftrightarrow x^2 \leq 3(2 + \frac{1}{x} - 3 + 2x)\, r_M = 3(\frac{1}{x} + 2x - 1)$$

Podemos despreciar $(2x - 1)$ frente a $1/x \Rightarrow$

$$x^2 \leq \frac{3}{x}\, r_M \Leftrightarrow \boxed{x \leq \sqrt[3]{3\, 2^{-23}} \approx 0{,}007}$$

10.— Un programa FORTRAN incluye la instrucción `X=N`, donde `X` es una variable real del tipo `REAL*4` (simple precisión) o de tipo `REAL*8` (doble precisión),y `N` es una constante entera positivo de tipo `INTEGER*4`.

Deducir cuál es el valor de `N` a partir del cual se pueden producir errores de almacenamiento en simple precisión y en doble precisión.

Sol. 10.

La instrucción `X=N` puede producir error de almacenamiento cuando:

$$N \geq (\underbrace{10 \ldots 01}_{m \text{ bits}})_2 = 2^{m-1} + 2^0 = 2^{m-1} + 1$$

siendo m el número de bits que se destinan a la mantisa (incluido el signo), pues a partir de este valor el número N puede tener más bits significativos que los que caben en la mantisa y será necesario redondear.

Si el primer bit de la mantisa no se almacena (que es lo más frecuente) entonces cabrá un bit más y los errores de almacenamiento se pueden producir cuando:

$$N \geq 2^m + 1$$

Por tanto:

$$\begin{cases} \texttt{REAL}*4 & \rightarrow & m = 24 & \rightarrow & N \geq 16\,777\,217 \\ \texttt{REAl}*8 & \rightarrow & m = 53 & \rightarrow & N \geq 2^{53} + 1 \end{cases}$$

Se puede observar que en doble precisión nunca habrá errores de almacenamiento pues el entero más grande que se puede considerar es $N = 2^{e-1} - 1$ con $e = 32$ es decir $N = 2^{31} - 1$.

---

11. Realizar un programa FORTRAN que encuentre la solución del problema anterior mediante experimentación numérica.

Sol. 11.

```
c=======================================================================
c================================================Programa AlmacenarEntero
c=======================================================================
c-----------------------------------------------------------------------
c Este programa encuentra el primer numero entero que no se puede almacenar
c exactamente en una variable del tipo REAL*4.
c
c Nota: El programa se debe compilar sin optimizatión (opcion: -o0) para
c       impedir que el compilador lo corrija.
c-----------------------------------------------------------------------
implicit integer*4 (i-n)
implicit real*4 (a-h,o-z)
logical seguir
parameter (MXINT=2*(2**30-1)+1)     !calculamos (2**31-1) sin overflow

i=0
seguir=.TRUE.
do while(seguir)
  i=i+1
  x=i       !equivale a x=real(i)
  j=x       !equivale a j=int(x)
  seguir=(i.lt.MXINT).and.(i.eq.j)
enddo

if(i.ne.j)then
  write(6,*) i,x
else
  write(6,200)
endif

100 format(' Problema en el termino i=',i20,
   .       '                 ----> x=',f30.9)
200 format(' Todos los enteros de tipo INTEGER*4'//
   .       ' caben en un real*4')

read(5,'()')

end
```

12.— En un ordenador digital se calcula la suma

$$S_n = \overbrace{a + a + ... + a}^{"n" \text{veces}}$$

donde a es un número cualquiera cuyo error inherente es r_a^I. Las operaciones se realizan en coma flotante, destinando m bits a la mantisa (incluido el signo). Se pide:

a) Calcular el error total del resultado que se obtiene al realizar la operación anterior. Indicar qué parte del error total se debe a la propagación del error inherente del dato a y qué parte se debe a la propagación de los errores de almacenamiento de las operaciones intermedias.

b) Analizar cómo crece el error máximo al aumentar el valor de n. Discutir en qué medida este valor máximo puede considerarse exagerado.

c) Repetir los apartados anteriores suponiendo que el cálculo se realiza en la forma $S_n = n \cdot a$. Comparar los resultados obtenidos en los dos casos.

d) Relacionar estas conclusiones teóricas con el planteamiento del problema 8 de la práctica anterior y realizar una predicción teórica del PK a partir del cuál pueden producirse los problemas a los que se hace referencia en aquel ejercicio. En su caso, comparar la predicción teórica con los resultados "experimentales" obtenidos por simulación numérica.

Sol. 12.a)

Analizamos el error en cada operación:

$$\begin{cases} S_1 = a & \rightarrow \quad r_{S_1} = r_a \\ S_2 = S_1 + a & \rightarrow \quad r_{S_2} = \dfrac{S_1}{S_1+a}\, r_{S_1} + \dfrac{a}{S_1+a}\, r_a + r^A_{S_2} \\ S_3 = S_2 + a & \rightarrow \quad r_{S_3} = \dfrac{S_2}{S_2+a}\, r_{S_2} + \dfrac{a}{S_2+a}\, r_a + r^A_{S_3} \\ S_4 = S_3 + a & \rightarrow \quad r_{S_4} = \dfrac{S_3}{S_3+a}\, r_{S_3} + \dfrac{a}{S_3+a}\, r_a + r^A_{S_4} \\ \vdots & \quad\;\; \vdots \\ S_n = S_{n-1} + a & \rightarrow \quad r_{S_n} = \dfrac{S_{n-1}}{S_{n-1}+a}\, r_{S_{n-1}} + \dfrac{a}{S_{n-1}+a}\, r_A + r^A_{S_n} \end{cases}$$

Si tenemos en cuenta que $S_i = i \cdot a$ entonces:

$$\frac{S_i}{S_i+a} = \frac{i}{i+1} \qquad \text{y} \qquad \frac{a}{S_i+a} = \frac{1}{i+1}$$

Luego:

$$\begin{cases} r_{S_1} &=& r_a \\ r_{S_2} &=& 1/2\; r_{S_1} + 1/2\; r_a + r^A_{S_2} &=& r_a + r^A_{S_2} \\ r_{S_3} &=& 2/3\; r_{S_2} + 1/3\; r_a + r^A_{S_3} &=& r_a + r^A_{S_3} + 2/3\; r^A_{S_2} \\ r_{S_4} &=& 3/4\; r_{S_3} + 1/4\; r_a + r^A_{S_4} &=& r_a + r^A_{S_4} + 3/4(r^A_{S_3} + 2/3\; r^A_{S_2}) \\ \vdots & & & & \\ r_{S_n} &=& \dfrac{n-1}{n}\, r_{S_{n-1}} + \dfrac{1}{n} r_a + r^A_{S_n} &=& r_a + r^A_{S_n} + \dfrac{n-1}{n}\left(r^A_{S_{n-1}} + ... + \dfrac{4}{5}\left(r^A_{S_4} + \dfrac{3}{4}\left(r^A_{S_3} + \dfrac{2}{3} r^A_{S_2}\right)\right)...\right) \end{cases}$$

$$\Rightarrow r_{S_n} = \underbrace{r^I_a}_{\text{error inherente}} \underbrace{+ r^A_a + r^A_{S_n} + \frac{n-1}{n}\left(r^A_{S_{n-1}} + ... + \frac{4}{5}\left(r^A_{S_4} + \frac{3}{4}\left(r^A_{S_3} + \frac{2}{3} r^A_{S_2}\right)\right)...\right)}_{\text{errores de almacenamiento}}$$

Vemos que el error inherente del dato se propaga tal cual es.

Sol. 12.b)

Acotamos el error:

$$\begin{aligned}
|r_{S_n}| &\leq |r_a^I| + \left[2 + \frac{n-1}{n}\left(1 + ... + \frac{4}{5}\left(1 + \frac{3}{4}\left(1 + \frac{2}{3}\right)\right)...\right)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{n}\left((n-1) + ... + 4 + 3 + 2\right)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{n}\,\frac{2+(n-1)}{2}\,(n-2)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{2n}\,(n+1)(n-2)\right] r_M \\
&= |r_a^I| + \frac{1}{2n}\left[4n + (n+1)(n-2)\right] r_M \\
&= |r_a^I| + \frac{1}{2n}[4n + n^2 - 2n + n - 2]\, r_M \\
&= |r_a^I| + \frac{1}{2n}[n^2 + 3n - 2]\, r_M \\
&= |r_a^I| + [n/2 + 3/2 - 1/n]\, r_M
\end{aligned}$$

Luego

$$\boxed{|r_{S_n}|^{max} \sim |r_a^I| + \frac{n}{2}\, r_M} \qquad \text{(Cuando } n \text{ es grande)}$$

El error crece linealmente con el valor de n.

Este error máximo es extremadamente improbable, pues para que se alcance, todos los errores de almacenamiento deben ser iguales a su máximo (el error de máquina) y del mismo signo. El resultado final será presumiblemente más pequeño. En la práctica los errores de almacenamiento tendrán valores diversos entre $(-r_M)$ y (r_M). Por lo tanto, para cuantificar este efecto podríamos realizar un cálculo estadístico y obtener el error probable (la esperanza matemática de r_{S_n})

Sol. 12.c)

Si realizamos la operación como $S_n = n \cdot a$ entonces:

$$r_{S_n} = r_n + r_a + r_{S_n}^a$$

En general salvo que n sera muy grande $r_n = 0$, por tanto

$$\boxed{r_{S_n} = r_a^I + r_a^A + r_{S_n}^A}$$

Igual que en el caso anterior, el error inherente en el dato se propaga tal cual es.

$$\boxed{|r_{S_n}| \leq |r_a^I| + 2\, r_M}$$

En el cálculo mediante sumas reiteradas el error crece a medida que se realizan más sumas, mientras que en el cálculo mediante multiplicación el error se mantiene en el mismo nivel.

Sol. 12.d)

El efecto acumulado de los errores de almacenamiento comenzaría a notarse cuando:

$$(n \cdot a)\,|r_{S_n}| \approx \frac{1}{2}\,0{,}001, \qquad \text{con} \qquad a = 0{,}1$$

pues el error podría ya alterar el tercer decimal.

Esto podría empezar a pasar cuando:

$$(n \cdot a)\,|r_{S_n}|^{max} \approx \frac{1}{2}\,0{,}001, \qquad \text{con} \qquad a = 0{,}1$$

con

$$|r_{S_n}|^{max} \sim \left|r_a^I\right| + \frac{n}{2}\,r_M$$

con $r_M = \dfrac{1}{2}\,2^{-m+2}$ y para n grande, $\left|r_a^I\right| << \dfrac{n}{2}\,r_M$:

$$(n\ 0{,}1)\frac{n}{2}\,2^{-m+2} \approx \frac{1}{2}\,0{,}001 \Longleftrightarrow n^2 \approx 10^{-2}\,2^{m-1} \Leftrightarrow \boxed{n \approx 10^{-1}\,2^{\frac{m-1}{2}}}$$

De esta forma, para variables tipo:

$$\texttt{REAL*4} \rightarrow n \approx 290$$

$$\texttt{REAL*8} \rightarrow n \approx 6\,710\,886$$

Los ensayos numéricos con los programas propuestos (SumaIterada_R4 y SumaIterada_R8) arrojan los siguientes resultados:

$$\texttt{REAL}*4 \quad \rightarrow \quad n = 703 \qquad\quad S_n = 70{,}299$$

$$\texttt{REAL}*8 \quad \rightarrow \quad n = 17\,704\,899 \quad S_n = 1\,770\,489{,}901$$

Luego el fenómeno tarda más en producirse de lo que habíamos previsto. Esto se debe a que la cota del error r_{S_n} es exagerada en relación con el error que se produce en la práctica.

13.— Un ingeniero programa habitualmente en Fortran y utiliza determinado ordenador digital. El ingeniero desea conocer cuál es el error de máquina r_M con el que se realizan los cálculos, tanto para las variables de tipo `REAL*4` (simple precisión) como para las variables de tipo `REAL*8` (doble precisión).

El ingeniero no tiene acceso a los manuales ni a ningún tipo de documentación en la que se explique cómo se almacenan los datos y cuántos bit se destinan al almacenamiento de la mantisa en cada caso.

Se pide:

a) Relacionar el error de máquina r_M con el valor de $\varepsilon > 0$ ("*machine* ε"), correspondiente al menor número real positivo tal que $\mathcal{A}(1+\varepsilon) \neq \mathcal{A}(1)$, siendo $\mathcal{A}(x)$ el valor almacenado correspondiente a x.

b) Realizar un programa Fortran que permita obtener experimentalmente el valor del ε ("*machine* ε") para los dos tipos de precisión (simple y doble) en el ordenador en el que se ejecute el programa.

c) Probar el programa en un ordenador y comparar los resultados obtenidos con las predicciones teóricas. Explicar las discrepancias, en su caso.

Sol. 13.a)

$$\mathcal{A}(1) = (0{,}100\ldots000)_B \; B^1$$

$$\mathcal{A}(1+\epsilon) = (0.\underbrace{100\ldots001}_{(m-1)\ cifras})_B \; B^1$$

Luego:

$$\varepsilon = \frac{1}{2}(0.\underbrace{000\ldots001}_{(m-1)\ cifras})_B \; B^1$$

El término $\dfrac{1}{2}$ aparece debido al redondeo por aproximación.

$$\Rightarrow \varepsilon = \frac{1}{2}\; B^{-(m-1)}\; B^1 = \frac{1}{2}\; B^{-m+2}$$

Por tanto:

$$\boxed{\varepsilon = r_{_M} = \frac{1}{2}\; B^{-m+2}}$$

Para un ordenador digital $B = 2 \Rightarrow$.

$$\epsilon = r_{_M} = 2^{-m+1} \rightarrow \begin{cases} \texttt{REAL}*4 \;\rightarrow\; m = 24 \;\Rightarrow\; \varepsilon = 2^{-23} \\ \texttt{REAL}*8 \;\rightarrow\; m = 53 \;\Rightarrow\; \varepsilon = 2^{-52} \end{cases}$$

Pero en la práctica es normal que no se almacene el primer bit (ya que siempre es 1 excepto para el número 0). De esta forma se dispone de un bit más para la mantisa, luego:

$$\epsilon = r_{_M} = 2^{-m} \rightarrow \begin{cases} \texttt{REAL}*4 \;\rightarrow\; m = 24 \;\Rightarrow\; \varepsilon = 2^{-24} \\ \texttt{REAL}*8 \;\rightarrow\; m = 53 \;\Rightarrow\; \varepsilon = 2^{-53} \end{cases}$$

Sol. 13.b)

```
c=====================================Programa MachineEpsilon
c=========================================================
      real*4 eps
      real*8 deps
c.....Simple Precision (REAL*4)
      call MEps_Simple(le2,eps)
      write(6,100) le2,eps
  100 format(' REAL*4 -> Machine Epsilon = 2**',i3,' = ',e15.6)

c.....Doble Precision (REAL*8)
      call MEps_Doble(lde2,deps)
      write(6,200) lde2,deps
 200 format(' REAL*8 -> Machine Epsilon = 2**',i3,' = ',d15.6)
      end

c------------------------------Subroutine MEps_Simple-----
c       Obtencion del machine epsilon en simple precision
c-----------------------------------------------------------
      subroutine MEps_Simple(le2,eps)
      implicit real*4 (a-h,o-z)
      uno=1.e+00
      dos=2.e+00
      eps=1.e+00
      le2=0
      q =uno+eps/dos
      do while (uno.ne.q)
        eps=eps/dos
        le2=le2-1
        q =uno+eps/dos
      enddo

      return
      end
c-------------------------Subroutine MEps_Doble----------
c        Obtencion delmachine epsilon en doble precision
c-----------------------------------------------------------
      subroutine MEps_Doble(le2,eps)
      implicit real*8 (a-h,o-z)
      uno=1.d+00
      dos=2.d+00
      eps=1.d+00
      le2=0
      q =uno+eps/dos
      do while (uno.ne.q)
        eps=eps/dos
        le2=le2-1
        q=uno+eps/dos
```

```
enddo

return
end
```
