SIMULACIÓN NUMÉRICA DE ERRORES USANDO ALGEBRA COMPUTACIONAL

Inna K. Shingareva
Departamento de Matemáticas
Universidad de Sonora

Resumen

En este trabajo se presentan simulaciones numéricas del surgimiento y propagación de errores en cálculos numéricos. Con el apoyo del sistema Maple y en combinación con técnicas de cómputo simbólico y numérico, son considerados algunos problemas de redondeo, convergencia y estabilidad.

1 Introducción

El Análisis Numérico es el área de las Matemáticas y de las Ciencias de la Computación que crea, analiza e implementa algoritmos discretos para resolver numéricamente problemas de matemáticas continuas. Desde la década de los 80, el crecimiento en potencia y disponibilidad de las computadoras ha ayudado a utilizar modelos matemáticos más realistas en las ciencias y las ingenierías y en la aplicación del Análisis Numérico.

Muchos fenómenos no pueden ser estudiados en experimentos y frecuentemente no se pueden observar todos los detalles de un experimento. En estos casos, no se puede verificar experimentalmente si los resultados numéricos son correctos. Para validar los resultados numéricos tenemos que analizar y evaluar el surgimiento y propagación de errores en cálculos numéricos.

Por lo tanto, es importante contar con técnicas para obtener las cotas exactas o estimación de errores en los cálculos. La estimación de errores es estudiada en diferentes campos de las matemáticas numéricas [1]. Entre los campos donde se han desarrollado técnicas y herramientas para la investigación de los errores se puden mencionar: la investigación de la influencia de la aritmética computacional en los cálculos (propagación de errores, aritmética de intervalos, análisis hacia enfrente y hacia atrás), estimación de errores para el control de tamaño de pasos (en ecuaciones diferenciales ordinarias), y generación de malla adaptativa. Estos y otros campos deben ser combinados para desarrollar la cultura computacional donde los resultados numéricos siempre son acompañados por cotas del error o estimaciones del error [3].

En este trabajo, se presentan simulaciones numéricas del surgimiento y propagación de errores en cálculos numéricos. Se utiliza el sistema Maple combinado con técnicas del cómputo simbólico y numérico. El trabajo consta de tres partes donde se discuten respectivamente problemas de redondeo, de convergencia y de estabilidad.

Existen dos hechos claves para comprender lo que sucede cuando se realizan cálculos numéricos en una computadora:

- Se debe considerar cómo se da la representación de números en una computadora digital y los errores que introduce esta representación. En este caso surgen los *problemas*

de redondeo. Para cálculos con enteros (incluyendo la división de enteros), donde los resultados se convierten a números enteros o tienen valores exactos (como en el sistema Maple), los errores numéricos suceden sólo en el caso de desbordes. Para cálculos de punto flotante, los errores numéricos pueden ser insignificantes o catastróficos, dependiendo del esquema numérico que se haya seleccionado para resolver el problema.

- Se deben desarrollar procedimientos matemáticos (algoritmos) para resolver con precisión cualquier problema matemático o decir cuando un problema no puede ser resuelto con ningún grado de precisión. En este caso, surgen los *problemas de convergencia y estabilidad.* Se discuten los problemas de convergencia utilizando series de Taylor y métodos iterativos, y se consideran problemas de estabilidad con ejemplos de relaciones de recurrencia estables e inestables.

2 Problemas de Redondeo

Definición 2.1. *Denotemos con x un número exacto y con $\bar{x}$ su representación inexacta en la computadora. Supongamos que hemos almacenado el número exacto*

$$x = \pm 0.a_1a_2\dots a_ma_{m+1} \times b^c, \quad 1 \le a_1 \le b-1, \quad 0 \le a_i \le b-1, \quad i = 2,\dots,m,$$

en una computadora de mantisa m, exponente c (con el rango $m \le c \le M$), y base b. Luego existen dos procedimientos comunes para obtener $\bar{x}$ dado x:

- *(i) corte* $\bar{x} = 0.a_1a_2\dots a_m \times b^c$
- *(ii) adición* $x = \big(0.a_1a_2\dots a_ma_{m+1} + 0.0_1 0_2 \dots 0_m (\frac{b}{2})_{m+1}\big) \times b^c = 0.A_1A_2\dots A_mA_{m+1} \times b^c$ *y corte* $\bar{x} = 0.A_1A_2\dots A_m \times b^c$.

El error introducido por representar x con $\bar{x}$ se llama *error de redondeo.* Otra causa de error en la representación de un número real en una computadora es causado por el tamaño finito del exponente c. También se producen errores de redondeo cuando se suman, substraen, multiplican o dividen dos números de punto flotante. En una computadora que corta y usa la base $b = 10$ y mantisa $m = 4$ podemos construir tres números reales x_1, x_2 y x_3 tales que: $\overline{x_1 + x_2} \neq \overline{x}_1 + \overline{x}_2$, pero $\overline{x_1 - x_2} = \overline{x}_1 - \overline{x}_2$ ($x_1 = 0.22228$, $x_2 = 0.22228$), ó $\overline{\overline{(x_1 + x_2)} + x_3} \neq \overline{x_1 + \overline{(x_2 + x_3)}}$ ($x_1 = 1.0007$, $x_2 = 2.0006$, $x_3 = 3$).

Existen dos métodos para medir el tamaño de un error. Definimos el *error absoluto* como la cantidad $E_a = |x - \bar{x}|$ y el *error relativo* como la cantidad $E_r = E_a/|x|$. Una de las tareas importantes del Análisis Numérico es la de estimar estos errores.

Por ejemplo, si x es un número exacto y $\bar{x}$ su representación inexacta *(i)* (de la Definicion 2.1) en una computadora con la base 10, mantisa m y exponente c, entonces probamos que $E_r < 10^{1-m}$. El número exacto x y su representación inexacta $\bar{x}$ se pueden escribir en la forma:

$$x = 10^c \sum_{i=1}^{\infty} a_i 10^{-i}, \quad \bar{x} = 10^c \sum_{i=1}^{m} a_i 10^{-i}.$$

Calculando el error relativo, tenemos

$$E_r = \frac{10^c\left(\sum_{i=1}^{\infty} a_i 10^{-i} - \sum_{i=1}^{m} a_i 10^{-i}\right)}{10^c \sum_{i=1}^{\infty} a_i 10^{-i}} = \frac{\sum_{i=m+1}^{\infty} a_i 10^{-i}}{\sum_{i=1}^{\infty} a_i 10^{-i}} = 10^{1-m}\left(\frac{\sum_{i=1}^{\infty} a_{m+i} 10^{-1-i}}{\sum_{i=1}^{\infty} a_i 10^{-i}}\right) < 10^{1-m}.$$

Mostramos cómo se pueden definir los errores relativos y absolutos y calcular digitos significativos para algoritmos numéricos usando funciones de Maple (el conjunto de los procedimientos "Apendice.txt" que hemos escrito para este trabajo se presenta en el Apéndice). Dejamos como ejercicio al lector, que verifique las soluciones obtenidas en este trabajo.

```
# 1. Errores absolutos y relativos, digitos significativos
restart: read "Apendice.txt":
printf(" "):
printf("%18s%18s%18s%18s%8s", "Valor exacto", "Valor aproximado",
"Error absoluto", "Error relativo", "DS"\n):
for x from 0 to 0.3 by 0.06 do
  x1 := evalf(sin(x)+cos(x)):
  x2 := 1+x-1/2*x^2-1/6*x^3:
  printf("%18.12e\t%18.12e\t%18.12e\t%18.12e\t%a\n", x1, x2, Er_abs(x1, x2),
  Er_rel(x1, x2), D_s(x1, x2));
od:
```

La mayoría de las computadoras representan internamente a los números utilizando el sistema binario. Mostramos algunos ejemplos del uso de las representaciones decimal, octal y binaria de los números: $(0.9)_{10} = (0.9 \times 10^0)_{pf} = (.714631463\ldots)_8 = (.11100110011\ldots)_2$. Vemos que para una computadora decimal finita, 0.9 tiene la representación interna exacta, pero para una computadora binaria finita tiene una representación interna inexacta. Los errores de redondeo se deben a las representaciones finitas, por ejemplo, la de π, e, $\sqrt{2}$.

En general, en las computadoras existen dos representaciones binarias de punto flotante: precisión sencilla (utilizando 32 bits para cada número) y la de precisión doble (utilizando 64 bits para cada número).

Definición 2.2. *El sistema Maple representa internamente a los números utilizando el sistema decimal. El sistema Maple representa a los números de punto flotante utilizando la precisión definida por el usuario mediante la variable de ambiente* `Digits`*. La representación flotante decimal se puede obtener utilizando la función* `evalf`*. La representación flotante binaria se puede obtener utilizando la función* `evalhf`*.*

Por ejemplo, en el Maple la representación de 0.9 es

```
> 0.9, evalf(0.9), evalhf(0.9);
  0.9,  0.9,  0.900000000000000022
```

Para simular la propagación de errores de redondeo, consideramos algunos problemas de cómputo de sumas, por ejemplo,

$$\sum_{i=1}^{N} k, \quad k = 0.9, 0.1, \frac{1}{3}, \qquad \sum_{i=1}^{N} \frac{1}{i^2}, \quad \sum_{i=N}^{1} \frac{1}{i^2}, \qquad N \to \pm\infty.$$

Mostramos cómo se pueden simular la propagación de errores de redondeo, los errores relativos y absolutos y calcular digitos significativos con los siguentes procedimientos de Maple:

```
# 2. Acumulacion y propagacion del error redondeo en sumas
restart: read "Apendice.txt":
i_b := 10^5:  i_f := 10^6:  i_s := 6*10^5:
L := [0.9, 0.1, 1.0/3.0]:
for p in L do
 printf("p = " "%22.12e\n", p):
 printf("%20s%25s%20s%23s", "Valor exacto", "Valor aproximado",
 "Error absoluto", "Error relativo\n"):
   for i from i_b to i_f by i_s do
    A := Acum_redondeo(p, 1, i);
    printf("%22.12e\t %22.12e %22.12e\t %22.12e\n", A[1], A[2],
    evalf(Er_abs(A[1],A[2]),30), evalf(Er_rel(A[1], A[2]), 30));
   od:
od:

#2.1 Error redondeo en sumas hacia enfrente y hacia atras
restart: read "Apendice.txt":
unassign('i','p'):   p:=1/i^2:
A := Acum_redondeoAppl(p, 1, 1, 1000000):
unassign('i','p'):   p:=1/i^2:
B := Acum_redondeoAppl(p, 0, 1, 1000000):
printf(" "); printf("%20s %25s %20s %23s %25s", "Valor exacto",
"V_ap Enfrente", "V_ap Atras", "Error absoluto", "Error relativo\n"):
printf("%2.20f\t %2.20f\t %2.20f\t %2.16e\t %2.16e\n", A[1], A[2], B[2],
evalf(Er_abs(A[2], B[2]), 30), evalf(Er_rel(A[2], B[2]), 30));
```

Consideramos el surgimiento y propagacion de error redondeo en subtracciones, por ejemplo, $f(x) = 1 - \cos x$, $x \in [0, \pi]$, $f(x) = \sqrt{x+1} - 1$, $x \to 0$:

```
> #3.1 El surgimiento y propagacion de error redondeo en subtracciones
restart:  read "Apendice.txt":
Digits := 10:  F := 1-cos(x):   G := sin(x):
F_ex1 := unapply(combine(F*G)/G, x);
F_ap1 := unapply(F, x);  x := 5.0:
printf("%9s %20s %25s %17s %20s %23s", "x", "[1-cos(x)]",
"[1-cos(x)]*sin(x)/sin(x)", "Error absoluto", "Error relativo", "DS\n"):
for i from 1 to 15 do
 printf("%2.12e\t %2.12e\t %2.12e\t %2.12e\t %2.12e\t %10a\n",
```

```
 x, F_ap1(x), F_ex1(x), Er_abs(F_ex1(x), F_ap1(x)),
 Er_rel(F_ex1(x), F_ap1(x)), D_s(F_ex1(x), F_ap1(x)));
 x := x/2.:
od:

#3.2 Error redondeo en subtracciones (mejoramos el esquema)
restart:  read "Apendice.txt":
unassign('x'):  X := 5.0:
printf("%9s %20s %20s %25s %15s", "x", "1-cos(x)", "Funcion",
"[1-cos(x)]*sin(x)/sin(x)", "Taylor\n"):
for i from 1 to 20 do
 Fun := SubtrEqual(x, sin(x), X, 0, 1, 13):
 F_ex2:=unapply(Fun[1],x):  Er_max:=Fun[2]:
 F_ex3:=unapply(Fun[3],x):  F_ex4:=unapply(Fun[4],x):
 printf("%2.12e\t %2.12e\t %2.12e\t %2.12e\t %2.12e\n", X, F_ex3(X),
 F_ex2(X), F_ex1(X), F_ex4(X));
 X := X/2.:
od:
```

También consideramos algunos ejemplos de cálculo de raíces de polinomios de grado N ($N \to \infty$):

```
# 4. Error redondeo en calculos raices de polinomios
restart:  read "Apendice.txt":
S := PolyRoots(17, 1.0E-5*x^(15));
fsolve(S[2],x,complex,fulldigits);
fsolve(S[1],x,complex,fulldigits);
```

3 Problemas de Convergencia

3.1 Series de Taylor

Consideramos problemas de convergencia en el uso de las series de Taylor, las cuales son muy importantes en Análisis Numérico y tienen importancia tanto teórica como práctica. Si la serie converge cerca del punto x_0, la podemos utilizar para aproximar la función $f(x)$ para el valor x cerca del punto de expansión x_0. La utilidad de la expansión en serie depende de que tan rápido converge la serie, o de qué tantos términos de la serie son necesarios para alcanzar una precisión dada. En algunos ejemplos comparamos aproximaciones de Taylor de mayor orden para una función dada $f(x)$ con los valores exactos, calculamos los errores absolutos, errores relativos y estimamos el error de truncamiento.

Al simular la razón de convergencia de las series de Taylor, consideramos dos métodos para el cálculo de $\ln 2$. Primero usamos la serie de Taylor $\ln(1+x)$:

```
> Grado:=15: Fun1:=unapply(convert(taylor(ln(1+x),x,Grado),polynom),x);
```

la cual converge para $x \in (-1, 1]$. Mostramos que esta serie no converge muy rápido. Por ejemplo, para obtener 4 cifras significativas requerimos calcular 4,562 términos:

```
# 5. Convergencia de la serie ln(1+x)
restart:  read "Apendice.txt":
T_max := 15:  Term := 1.:  S := Term:
printf(" "): printf("%8s %25s %20s %25s", "i", "Valor aproximado",
"Error  absoluto", "Digitos Significativos"\n):
for i from 2 to T_max do
 Term := -((i-1)/i)*Term:  S := S + Term:
 printf("%10.5f%20.10f%20.10f%20d\n",i,S,Er_abs(ln(2),S),D_s(ln(2),S)):
od:
```

Segundo, para mejores razones de convergencia, consideramos la serie que es la diferencia de las dos expansiones en serie $\ln(1+x)$ y $\ln(1-x)$:

```
> Grado:=15: Fun2:=unapply(convert(taylor(ln(1-x),x,Grado),polynom),x);
> Fun3 := unapply(Fun1(x)-Fun2(x), x);
```

Esta serie converge más rápido que la serie original: para obtener 5 cifras significativas requerimos calcular sólo 3 términos. Mostramos cómo se pueden simular la razón de convergencia de las series de Taylor, los errores relativos y absolutos y calcular digitos significativos con los siguentes procedimientos de Maple:

```
# 6. Convergencia de la serie ln(1+x)-ln(1-x) (mejoramos el esquema)
restart:  read "Apendice.txt":
T_max := 8:  Term := 2./3.:   S := Term:
printf(" "):  printf("%8s %25s %20s %25s", "i", "Valor aproximado",
"Error  absoluto", "Digitos Significativos"\n):
for i from 1 to T_max do
  Term:=1./9.*(2*i-1)/(2*i+1)*Term:  S:=S+Term:
  printf("%10.5f %20.15f %20.15f %20d \n", i+1, S, Er_abs(ln(2), S),
  D_s(ln(2), S)):
od:
```

3.2 Métodos Iterativos

Consideremos el problema de encontrar la raíz de la ecuación $x = e^{-x}$ usando dos métodos iterativos con diferentes razones de convergencia. I) Mostramos con los siguentes procedimientos de Maple que con el primer esquema de iteración

$$x_n = e^{-x_{n-1}},$$

para obtener 6 cifras significativas, requerimos 21 iteraciones:

```
# 7.1  Metodos iterativos
restart:   read "Apendice.txt":
Digits:= 20:   V_ex := fsolve(y=exp(-y), y=0..1);
x_0 := 0.5:  x := x_0:
```

```
printf(" "): printf("%5s %20s %20s %25s", "i", "Valor aproximado",
"Error  relativo", "Digitos Significativos"\n):
for i from 1 to 20 do
  x := exp(-x):
  printf("%5d%20.15f%20.15f%20d\n",i+1,x,Er_rel(V_ex,x),D_s(V_ex,x)):
od:
```

II) Sin embargo, para el segundo método

$$x_n = x_{n-1} + \frac{e^{-x_{n-1}} - x_{n-1}}{e^{-x_{n-1}} + 1},$$

la segunda iteración concuerda con el resultado exacto al menos en 6 cifras significativas. Utilizando técnicas de cálculo simbólico y numérico, podemos simular la propagación de errores para estos esquemas:

```
# 7.2  Metodos iterativos (mejoramos el esquema)
restart:  read "Apendice.txt":
Digits:= 20:  V_ex := fsolve(y=exp(-y), y=0..1);
x_0 := 0.5:   x := x_0:
printf(" "): printf("%5s %20s %20s %25s", "i", "Valor aproximado",
"Error  relativo", "Digitos Significativos"\n):
for i from 1 to 7 do
  x := x + (exp(-x)-x)/(exp(-x)+1):
  printf("%5d%20.15f%20.15f%a\n",i+1,x,Er_rel(V_ex,x),D_s(V_ex,x)):
od:
```

4 Problemas de Estabilidad

Consideramos problemas de estabilidad en el caso de relaciones de recurrencia que aparecen frecuentemente en cálculos numéricos. En la simulación de inestabilidad de relaciones de recurrencia, consideramos los problemas asociados al cálculo de integrales:

$$I_n = \int_0^1 \frac{x^n}{4x+1} dx, \quad n = 0, 1, 2, \ldots, \quad I_0 = \int_0^1 \frac{1}{4x+1} dx = \frac{1}{4} \ln 5 = 0.4023594780$$

Usando técnicas analíticas y numéricas, mostramos que la relación de recurrencia es estable:

```
# 8.1 Problemas de estabilidad (el esquema estable)
restart:   read "Apendice.txt":
InR := evalf(int(1/(4*x+1), x=0..1));
printf(" "): printf("%5s %20s %20s %20s %25s", "i", "Recurrencia",
"Simbolico", "Numerico", "Error  relativo", "Digitos Significativos"\n):
for i from 0 to 15 do
 InR := 1/4*(1/(i+1)- InR):
```

```
 InS := evalf(simplify(int(x^i/(4*x+1), x=0..1))):
 InN := evalf(Int(x^i/(4*x+1), x=0..1)):
 printf("%5d %20.15f %20.15f %20.15f %20.15f %20d\n", i+1, InR, InS, InN,
 Er_rel(InS, InR), D_s(InS, InR)):
od:
```

En otro ejemplo

$$I_n = \int_0^1 x^n e^{x-1}\, dx, \qquad n = 0, 1, 2, \ldots.$$

Usando el método de integración por partes, tenemos

$$I_n = x^n e^{x-1}\Big|_0^1 - \int_0^1 n x^{n-1} e^{x-1} dx = 1 - n I_{n-1}, \quad n = 1, 2, \ldots,$$

donde $I_0 = \displaystyle\int_0^1 e^{x-1} dx = 1 - e^{-1} = 0.6321205588$. Si utilizamos la relación de recurrencia $I_n = 1 - nI_{n-1}$ para calcular el valor de I_n para $n = 1, 2, \ldots$ utilizando, por ejemplo, 10 cifras decimales (`Digits=10`), tendremos $I_{15} = -2.426899440$. Pero estudiando el comportamiento I_n para $n = 1, 2, \ldots$ analíticamente obtenemos que $I_n > 0$ y $I_n < I_{n-1}$. Entonces el resultado numérico I_{15} es completamente erróneo. Con otros valores de variable `Digits` obtendremos resultados diferentes pero erróneos también. Usando técnicas analíticas y numéricas, mostramos que la relación de recurrencia es inestable:

```
# 8.2 Problemas de estabilidad (el esquema inestable)
restart:   read "Apendice.txt":  Digits := 10:
InR := evalf(int(exp(x-1), x=0..1));
printf(" "): printf("%5s %20s %20s %20s", "i", "Recurrencia",
"Simbolico", "Numerico"\n):
for i from 1 to 15 do
  InR := 1-i*InR:
  InS := evalf(int(x^i*exp(x-1), x=0..1)):
  InN := evalf(Int(x^i*exp(x-1), x=0..1)):
  printf("%5d %20.15f %20.15f %20.15f\n", i+1, InR, InN, InS):
od:
```

Observamos que cualquier error redondeo hecho en el cálculo de I_0 será multiplicando por $(-1)(-2)\ldots(-n) = -n!$ para calcular I_n. La relación de recurrencia $I_n = 1 - nI_{n-1}$ es inestable.

Un método mejor sería reescribir la relación de recurrencia como

$$I_{n-1} = \frac{1 - I_n}{n}, \qquad n = \ldots, 2, 1.$$

En este caso, podemos calcular I_n comenzando con una approximación inicial, por ejemplo $I_{25} = \displaystyle\int_0^1 x^{25} e^{x-1}\, dx \approx 0$. Luego obtendremos $I_{15} = 0.059017540879298$ con 14 dígitos significativos aunque I_{25} era una aproximación no calculada. La relación de recurrencia $I_{n-1} = (1 - I_n)/n$ es estable. Mostramos el esquema estable con funciones del Maple:

```
# 8.3 Problemas de estabilidad (mejoramos el esquema inestable)
restart:  read "Apendice.txt":  Digits := 15:  InR := 0;
printf(" "): printf("%5s %20s %20s %25s %20s", "i", "Recurrencia",
"Numerico", "Digitos Significativos", "Simbolico"\n):
for i from 25 to 1 by -1 do
  InR := (1-InR)/(i+1):
  InS := evalf(int(x^i*exp(x-1), x=0..1)):
  InN := evalf(Int(x^i*exp(x-1), x=0..1)):
  printf("%5d %20.15f %20.15f %a %20.15f\n",
  i, InR, InN, D_s(InR, InN), InS):
od:
```

Apéndice

```
# 1. Error absoluto
Er_abs := proc(p_ex, p_ap)  global Ea:  Ea := abs(p_ex - p_ap):  RETURN(Ea): end;

# 2. Error relativo
Er_rel := proc(p_ex, p_ap)  global Er:
if p_ex <> 0 then  Er := abs((p_ex-p_ap)/p_ex) else Er:=0  fi: RETURN(Er): end;

# 3. Digitos significativos
D_s := proc(p_ex, p_ap)  local n:  global Ds:  assume(n>0):
if p_ex=p_ap then  Ds := " ":
   else Ds:=round(fsolve(Er_rel(p_ex, p_ap)=0.5*10^(-n),n)):
fi:
RETURN(Ds): end;


# 4. la propagacion de errores de redondeo en el computo de sumas
Acum_redondeo := proc(x, N_1, N_2) local i: global S_ex, S_ap:
S_ex:=0: S_ap:=0: S_ex:=sum('x','i'=N_1..N_2):
for i from N_1 to N_2 do S_ap:=evalhf(S_ap +x): od:
RETURN(S_ex, S_ap): end;

# 5. la propagacion de errores de redondeo en el computo de sumas hacia enfrente
Acum_redondeoF := proc(x, N_1, N_2) global i, S_exF, S_apF: S_apF := 0:
S_exF:=evalf(sum('x', 'i' = N_1..N_2), 30):
for i from N_1 to N_2 do S_apF := evalhf(S_apF +x): od:
RETURN(S_exF, S_apF): end;

# 6. la propagacion de errores de redondeo en el computo de sumas hacia atras
Acum_redondeoB := proc(x, N_1, N_2) global i, S_exB, S_apB: S_apB := 0:
S_exB := evalf(sum('x', 'i'= N_1..N_2), 30):
for i from N_2 to N_1 by -1 do S_apB := evalhf(S_apB + x): od:
```

```
RETURN(S_exB, S_apB): end;

# 7. la propagacion de errores de redondeo hacia enfrente o atras
Acum_redondeoAppl  :=  proc(P, T, N_1, N_2) global i, p, F: p := P:
  if T = 0 then F := Acum_redondeoF(p, N_1, N_2):
     else F := Acum_redondeoB(p, N_1, N_2):
  fi:
RETURN(F): end;

# 8. el surgimiento de errores de redondeo en calculos de raices de polinomios
PolyRoots := proc(N, Pert) local i: global P, Q, x:
P := sort(expand(product('x-i','i' = 1..N)));
Q := P+Pert;
RETURN(P, Q): end;

# 9. la propagacion de errores de redondeo en subtracciones
SubtrEqual:=proc(Expr1, Expr2, X, a, b, N) local F2, R: global P, Er, F1, Ser:
F1 := Expr1-Expr2: F2:=1- Expr2/Expr1 -1/10:  R:=fsolve(F2, x = a..b):
Ser := convert(series(Expr1-Expr2, x, N), polynom): Er:=R^N/N!:
 if X < round(R) then P:=Ser  else P:=F1: fi:
RETURN(P, Er, F1, Ser): end:
```

Bibliografía

[1] L. Fosdick, E. Jessup, C. Schauble, and G. Domik (1996)*An Introduction to High-Performance Scientific Computing*, MIT Press

[2] E. Haug (1989)*Computer Aided Kinematics and Dynamics of Mechanical Systems, Vol. I: Basic Methods*, Allyn and Bacon, Pub.

[3] M. Heath (1997)*Scientific Computing: An Introductory Survey*, McGraw-Hill Inc.

[4] N. Higham (1996)*Accuracy and Stability of Numerical Algorithms*, SIAM Pub.

[5] C. T. Kelley (1995)*Iterative Methods for Linear and Nonlinear Equations*, SIAM Pub.

[6] M. Overton (2001)*Numerical Computing with IEEE Floating Point Arithmetic*, SIAM Pub.