*Trabajo práctico 1*

*Materia*: Métodos en Computación Científica

*Alumno*: Duarte Daniela LU: 88187

*Fecha*: 28 de Agosto de 2013

Ejercicio 1:

Para realizar el cálculo épsilon machine se realizo el siguiente algoritmo propuesto por la cátedra:

%Calculo el epsilon machine

s=1.0;

for i=1:100

s=0.5\*s;

t=s+1.0;

if (t<=1.0)

ep=2.0\*s

break

end

end

Para verificar el resultado se utilizo el comando de matlab : eps.

Por ambos caminos el resultado fue:

eps = 2.2204e-16

Ejercicio 2

Primero cargamos la función que calcula la constante de amortiguación. Para esto

Definimos una nueva función constAmortiguacion(m,l,h,a,r).

constAmortiguacion = @(l,h,a,r) (6 \* pi \* 0.3445 \* l / (h^3)) \* (((a - h/2)^2) - (r^2)) \* ((((a^2)-(r^2)) / (a - (h/2)))-h);

Calculamos el valor usando como referencia los valores *m* = 0.3445 Pa.s, *l* =10 cm,

*h*= 0.1 cm, *a* = 2cm, y *r* = 0.5 cm.

valor\_referencia = constAmortiguacion (0.3445,10, 0.1, 2, 0.5)

Ahora, calculamos *la función* para los diferentes valores. El resultado será almacenado en *a,b,c*

(constantes de amortiguación con error). Calculamos el error absoluto y el error relativo

guardando el resultado en error\_absoluto y error\_relativorespectivamente.

% inciso a)

a = constAmortiguacion(9.999, 0.09, 1.999, 0.499)

error\_absoluto = a - valor\_referencia

error\_relativo = error\_absoluto / valor\_referencia

% inciso b)

disp('------ Inciso b) ------');

b = constAmortiguacion(10.001, 0.101, 2.001, 0.501)

error\_absoluto = b - valor\_referencia

error\_relativo = error\_absoluto / valor\_referencia

% inciso c)

disp('------ Inciso c) ------');

c = constAmortiguacion(9.999, 0.101, 2.001, 0.499)

error\_absoluto = c - valor\_referencia

error\_relativo = error\_absoluto / valor\_referencia

En la siguiente tabla se presentan los resultados obtenidos:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***l*** | ***h*** | ***a*** | ***r*** | **Constante Amortiguación** | **Error Absoluto** | **Error Relativo** |
| 10 | 0.1 | 2 | 0.5 | 4.2056141926645e+05 | - |  |
| 9.999 | 0.09 | 1.999 | 0.499 | 5.8098142543172e+05 | 1.6042000616e+05 | 0.38144251663662 |
| 10.001 | 0.101 | 2.001 | 0.501 | 4.0835277301135e+05 | -1.2208646255e+04 | -0.0290294014044 |
| 9.999 | 0.101 | 2.001 | 0.499 | 4.0873065231814e+05 | -1.1830766948e+04 | -0.0281308898209 |

*Conclusión:*

De la observación de los valores obtenidos mostrados en la tabla anterior se puede concluir que el la mayor variación del valor h en el inciso a) es la que produce el mayor error absoluto y relativo en el cálculo de la función, esto se debe a que el valor h es el más cercano a cero y por lo tanto sufre de mayor sensibilidad en la representación y redondeo.

Ejercicio 3:

El polinomio de Wilkinson se utiliza para el análisis de la estabilidad numérica.

Primero generamos el polinomio P(Z) con la siguiente rutina:

% defino P(Z)

P=[1 -1]

for n=2:20,

P=conv(P, [1 -n]);

% para el cálculo en doble precisión

format long;

end

Luego la siguiente rutina permite computar raíces

% defino p(Z)

p = @(x) (x-20).\*(x-19).\*(x-18).\*(x-17).\*(x-16).\*(x-15).\*(x-14).\*(x-13).\*(x-12).\*(x-11).\*(x-10).\*(x-9).\*(x-8).\*(x-7).\*(x-6).\*(x-5).\*(x-4).\*(x-3).\*(x-2).\*(x-1);

% cálculo las raíces de P(Z) en doble precisión

raices = roots(P)

Con P(Z) y p(Z) se pasan a chequear la calidad se las raíces:

% calcular |P(z)|

for i=1:20,

raiz = raices(i);

valor= polyval(P,raiz);

disp(abs(valor));

end

% calcular |p(z)|

for i=1:20,

raiz = raices(i);

valor= p(raiz);

disp(abs(valor));

end

% calcular |z-k|

for i=1:20,

raiz = raices(i);

valor= raiz - (21 - i);

disp(abs(valor));

end

Los resultados de los 3 cálculos (P(zk), p(zk), z-k)se muestran en la siguiente tabla:

*Conclusión:*

Del análisis de los valores mostrados en la tabla se puede observar que los valores del polinomio  y sus raíces son extremadamente sensibles al redondeo de errores incluso si los cálculos se realizan con doble precisión.