*Trabajo práctico 1*

*Materia*: Métodos en Computación Científica

*Alumno*: Stessens Alejandro LU: 83262

*Fecha*: 14 de Setiembre de 2016

Ejercicio 1:

La función

f(x)=log e (x+1/x-1) en 1.1

Su resolución en Matlab es:

>> x=1.1

x =

1.1000

>> log (x+1)-log(x-1)

ans =3.0445

Y es aproximado por la serie :

Sus primeros 7 terminos :

>> x=1.1; termino=0; impar=1;

for i=1:7

termino=(1/(impar\*(x^impar)))+s;

impar=impar+2;

end

salida=s\*2

salida = 2.9166

Sus primeros 40 terminos :

>> x=1.1; termino=0; impar=1;

for i=1:30

termino=(1/(impar\*(x^impar)))+s;

impar=impar+2;

end

salida=s\*2

salida = 3.0440

Sus primeros 100 términos :

>> x=1.1; s=0; impar=1;

for i=1:100

s=(1/(impar\*(x^impar)))+s;

impar=impar+2;

end

salida=s\*2

salida = 3.0445

Conclusión:

Está serie converge lentamente al valor de f(1.1). A partir del término 30 en adelante se comporta de manera que el aumento del resultado es muy pequeño.

Ejercicio 2:

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

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

constanteDeAmortiguacion = @(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 = constanteAmortiguacion (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:

Dada la serie razón dorada de Fibonacci tenemos que:

Sus primeros 2 términos

F0=1

F1=1

Y Su formula general:

F (i+2)=F (i+1)+ F (i)

Con i=1,2,…

Sabemos que F (i+1)/f (i) converge a xn= 1+ √5

Y su resolución en Matlab es:

fprintf ('Serie Fibonacci:\n')  
f = [1 1]; i = 1;  
while f(i) + f(i+1) < 100  
f(i+2) = f(i) + f(i+1);  
i = i + 1;  
end  
fprintf('%g ',f)

La serie es:

1 1 2 3 5 8 13 21 34 55 89 …

Y sus xn son:

x1=1/1

**x1 =1**

x2=2/1

**x2 =2**

x3=3/2

**x3 =1.500000000000000**

x4=5/3

**x4 =1.666666666666667**

x5=8/5

**x5 =1.600000000000000**

x6=13/8

**x6 =1.625000000000000**

x7=21/13

**x7 =1.615384615384615**

x8=34/21

**x8 =1.619047619047619**

x9=55/34

**x9 =1.617647058823529**

x10=89/55

**x10=1.618181818181818**

y sus errores relativos son:

**er1**

x=1.618033988749895; x1=1; (x1-x)/x

ans =-0.381966011250105

**er2**

x2=2; x=1.618033988749895; (x2-x)/x

ans =0.236067977499790

**er3**

x3=1.5;x=1.618033988749895; (x3-x)/x

ans = -0.072949016875158

**er4**

x4=1.666666666666667;x=1.618033988749895; (x4-x)/x

ans = 0.030056647916492

**er5**

x5=1.600000000000000;x=1.618033988749895; (x5-x)/x

ans =-0.011145618000168

**er6**

x6=1.625000000000000;x=1.618033988749895; (x6-x)/x

ans=0.004305231718579

**er7**

x7=1.615384615384615;x=1.618033988749895; (x7-x)/x

ans = -0.001637402788632

**er8**

x8=1.619047619047619;x=1.618033988749895; (x8-x)/x

ans =6.264579760202099e-04

**er9**

x9 =1.617647058823529;x=1.618033988749895; (x9-x)/x

ans =-2.391358457586257e-04

**er10**

x10=1.618181818181818;x=1.618033988749895; (x10-x)/x

ans =9.136361346602813e-05

Sabemos que F (i+1)/f (i) converge a (1+ √5)/2

>> (1+sqrt(5))/2

ans =1.618033988749895

Ejercicio 4:

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

*Conclusión:*

Del análisis de los valores que calculan los procedimientos creados en Matlab 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.