Taller 1 Análisis Númerico

Daniel Reyes
Edwin Turizo
Juan Pimienta
Cristobal Castrillon

February 14 2020

1. Numero de Operaciones

1.1. Teorema de Horner

1. Utilice el metodo de inducción matemática para demostrar el resultado del metodo 2

2. Implemente en R o Python para verificar los resultados del metodo de Horner
   Algoritmo realizado en Python.

```
#Metodo 3 - Algoritmo de Horner

import random
#Función del metodo de Horner
def horner(coeficientes, grado, valor):
  resultado = coeficientes[0]
  i = 1
  while(i <= grado):
    resultado = (resultado * valor) + coeficientes[i]
    i += 1
  #end while
  return (resultado,i-1)
#end def

coef=[2,-3,3,-4,-2,1,4]
grado=6
valor=1
minCoef=0
maxCoef=8;
for i in range(0,40):
  [res,iteraciones]=horner(coef,grado,valor)
  print("Numero Iteraciones: ",iteraciones, "\t Grado: ",grado)
  grado+=1;
  coef.append(random.randint(minCoef,maxCoef))
#end for
```

3. Evaluar en x = 1.0001 con $P(x) = 1 + x + x^2 + ... + x^{50}$. Encuentre el error de calculo al comparalo con la expresion equivalente $Q(x) = (x^{51} - 1)/(x - 1)$

- El resultado de evaluar x=1.0001 con el polinomio P(x) mediante el metodo Horner da igual a 52.1328212709851

```
def horner (coeficientes,grado,valor):
  resultado = coeficientes[0]
  i = 1
  while(i <= grado):
    resultado = (resultado * valor) + coeficientes[i]
    i += 1
  #end while
  return (resultado,i-1)
#end def
x=[]
cont=0
while(cont<=51):
  x.append(1)
  cont+=1
valor=1.0001
[res,iteraciones]=horner(x,51,valor)
print("Numero Iteraciones: ",iteraciones, "\t Resultado: ",res)
```

- El resultado de evaluar x=1.0001 con el polinomio Q(x) mediante una función da igual a 0.005112771 Funcion de evaluacion implementada en R

```
f = function(pX){
  return (pX^(51) - (1)) / (pX-1)
}
num=f(1.0001)
print(num)
```

1.2. Numeros Binarios

1. Encuentre los primeros 15 bits en la representacion binaria de π
   Respuesta: 010000000100100
2. Convertir los siguientes numeros binarios a base 10: 1010101; 1011.101; 10111.010101...; 111.1111...
   - 1010101 = 1.415452980112962e-39
   - 1011.101 =
   - 10111.010101... =
   - 111.1111... =
3. Convierta los siguientes numeros de base 10 a binaria: 11.25; 2/3; 30.6; 99.9
   - 11.25 = 01000001001101000000000000000000
   - 2/3 = 00111111001010101010101010101011
   - 30.6 = 01000001111101001100110011001101
   - 99.9 = 01000010110001111100110011001101

```python
#Algoritmo de conversiones binarias

import struct
import math

a = math.pi

def float_to_bin(num):
  return format(struct.unpack('!I', struct.pack('!f', num))[0], '032b')

def bin_to_float(binary):
  return struct.unpack('!f',struct.pack('!I', int(binary, 2)))[0]

numero=float_to_bin(a)
print (numero[0:15])

binary = 1010101
b = bin(binary)
print (bin_to_float (b))
```

1.3. Representación del punto flotante de los numeros Reales

1. ¿Como se ajusta un numero binario infinito en un numero finito de bits?
   Las computadoras, con un número finito de bits no pueden almacenar todos los números reales en forma exacta. La forma convencional de almacenar números reales en la memoria de una computadora es mediante el método llamado de punto flotante o floating point. Uno de los sistemas más comunes es la representación de números reales es simple precisión utilizada en la convención IEEE. En dicho sistema cada número de precisión simple ocupa 4 bytes (32 bits) que se destinan a: el signo (1 bit), un exponente (8 bits) y la parte fraccionaria de la mantisa (23 bits). De esta manera un número está determinado por estas tres cantidades.

   $$(\underbrace{0}_{\substack{\text{signo}\\(1)_{10}}}\ \underbrace{10000011}_{\substack{\text{exponente}\\(131)_{10}}}\ (1),\underbrace{00111100010000000000000}_{\substack{\text{mantisa}\\(241/1024)_{10}}})_2 \ = \ (19.765625)_{10}$$

   Hemos representado entre paréntesis la parte entera de la mantisa (que es igual a 1 siempre por convención. Debe notarse que el número final se obtiene considerando que:

   - El signo es positivo (bit de signo igual a 0).
   - El exponente se obtiene como $131 - 127 = 4$, que en sistema decimal da $2^4 = 16$.
   - La mantisa $1 + 241/1024 = 1{,}2353515625$se obtiene sumando: 1 (implícito), 1/8, 1/16, 1/32, 1/64 y 1/1024.

2. ¿Cual es la diferencia entre redondeo y recorte?
   Recorte: Se refiere a cortar la expresión en una determinada cantidad de decimales. Redondeo: Se refiera a aproximar la expresión al valor mas cercano, segun un criterio ya definido. El criterio mayor utilizado es el siguiente:

   - Sí el decimal deseado es menor que 5, se recortan los decimales siguientes.

- Sí el decimado deseado es mayor o igual que 5, se le aumenta en una unidad.

3. Idique el numero de punto flotante (IEEE) de precision doble asociado a x, el cual se denota como fl(x); para x(0.4)
   Con este algoritmo, el numero punto flotante para:
   x(0.4)= 0011111111011001100110011001100110011001100110011001100110011010
   Algoritmo de punto flotante doble en Python:

```
#Metodo de punto flotante doble
import struct

getBin = lambda x: x > 0 and "00" + str(bin(x))[2:] or "11" + str(bin(x))[3:]

def floatToBinary64(value):
  val = struct.unpack('Q', struct.pack('d', value))[0]
  return getBin(val)

binstr = floatToBinary64(0.4)
print("Punto Flotante Doble Precisionn de 0.4:", binstr)
```

1.4. Epsilon de una maquina

1. Error de redondeo. En el modelo de la arimetica de computadora IEEE, el error de redondeo relativo no es mas de la mitad del epsilon de maquina:

$$\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2}\varepsilon maq$$

   Teniendo en cuenta lo anterior, encuentre el error de redondeo para x = 0.4
   Como salidas del algoritmo tenemos:

   - Epsilon de la maquina en forma hexadecimal = 2,220446e^{16}
   - Epsilon de la maquina en forma binaria = 2^{-52}
   - El error acumulado es = 401

2. Verificar si el tipo de datos basico de R y Python es de precision doble IEEE y Revisar en R y Phython el format long

3. Encuentre la representacion en numero de maquina hexadecimal del numero real 9.4

4. Encuentre las dos raices de la ecuacion cuadratica $x^2 + 9^{12}x = 3$. Intente resolver el problema usando la arimetica de precision doble, tenga en cuenta la perdida de significancia y debe contrarestarla.

5. Explique como calcular con mayor exactitud las raicces de la ecuacion:

$$x^2 + bx - 10^{-12} = 0$$

   Donde b es un numero mayor que 100

```r
#Metodo de Error de Redondeo Epsilon - x(0.4)
t = function(maxiter = 100)
{
  n = 0;
  while(1.0+(maxiter*0.5) > 1.0)
  {
    n=n+1
    maxiter = maxiter*0.5
  }
  cat("Epsilon de la maquina en forma hexadecimal = ", maxiter, "\n")
  cat("Epsilon de la maquina en forma binaria = 2^-", n, "\n")
  suma = 1
  i = 1
  while(i<=1000){
    suma = suma + 0.40000000000
    i = i + 1
  }
  maxiter = 10000 * 0.40000000000 + 1

  cat("El error acumulado es  = ", suma, "\n")
}
```

2. Raices de una Ecuacion

1. Implemente en R o Python un algoritmo que le permita sumar unicamente los elementos de la submatriz triangular superior o triangular inferior, dada la matriz cuadrada An. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacion O() con una grafica que muestre su orden de convergencia.
   Se realiza la implementacion del codigo y se realizan pruebas emepezando desde n=5 hasta n=25.
   Algoritmo implementado en R.

```r
##Funcion de Suma Matrix Triangular

#Datos Entrada
n=5;
minN=1;
maxN=100;
for(i in 0:20){
  matriz <- matrix(minN:maxN,n,n)
  #matriz
  matrizCopy <- matriz

  #Calculo Matrices Triangulares
  matriz[lower.tri(matriz,diag=TRUE)] <- 0
  matrizUPT <- matriz
  #matrizUPT
  matrizCopy[upper.tri(matrizCopy,diag=TRUE)] <- 0
  matrizDWT <- matrizCopy
  #matrizDWT

  #Sumas Matrices Triangulares
  sumUPT=sum(matrizUPT)
  #cat("\r n=",n,"Sumatoria Matriz Triangular Superior:",sumUPT)
  sumDWT=sum(matrizDWT)
  #cat("\r n=",n,"Sumatoria Matriz Triangular Inferior:",sumDWT)
  # crear dataframe de vectores
  tabla <- data.frame(n, sumUPT, sumDWT)
  n=n+1;
  print(tabla)
}
```

n	sumUPT	sumDWT
5	170	90
6	365	190
7	693	357
8	1204	616
9	1956	996
10	3015	1530
11	2655	2155
12	2958	2612
13	3814	3046
14	5121	3906
15	5185	4545
16	5820	5220
17	7148	6092
18	7599	7026
19	8911	7391
20	9730	9460
21	10210	9810
22	11813	10822
23	12509	12081
24	15076	12176
25	13700	15700

2. Implemente en R o Python un algoritmo que le permita sumar los n2 primeros numeros naturales al cuadrado. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacion O() con una grafica que muestre su orden de convergencia.
   Se realiza la implementacion del codigo y se realizan pruebas emepezando desde n=5 hasta n=25.
   Algoritmo implementado en R.

```r
##Funcion de Suma (n2)2

#Datos Entrada
n=2;
first=1

for(i in 0:23){
  suma=0
  for(i in first:n^2){
    iCuadrado=i^2;
    suma= suma + iCuadrado;
  }
  tabla <- data.frame(n, suma)
  n=n+1;
  print(tabla)
}
```

n	suma
2	30
3	285
4	1496
5	5525
6	16206
7	40425
8	89440
9	180441
10	338350
11	597861
12	1005720
13	1623245
14	2529086
15	3822225
16	5625216
17	8087665
18	11389950
19	15747181
20	21413400
21	28686021
22	37910510
23	49485305
24	63866976
25	81575625

3. Para describir la trayectoria de un cohete se tiene el modelo:

$$y(t) = 6 + 2{,}13t^2 - 0{,}0013t^4$$

Donde, y es la altura en [m] y t tiempo en [s]. El cohete esta colocado verticalmente sobre la tierra. Utilizando dos metodos de solucion de ecuacion no lineal, encuentre la altura maxima que alcanza el cohete.

3. Convergencia de Metodos Iterativos

3.1. Parte 1

Para cada uno de los siguientes ejercicios implemente en R o Python, debe determinar el numero de iteraciones realizadas,una grafica que evidencie el tipo de convergencia del metodo y debe expresarla en notacion O()

1. Sean f(x) = ln(x + 2) y g(x) = sin(x) dos funciones de valor real.

   - Utilice la siguiente formula recursiva con $E = 10^{-8}$ para el punto de interseccion:

   $$X_n = X_{n-1} - \frac{f(X_{n-1})(X_{n-1} - X_{n-2})}{f(X_{n-1}) - f(X_{n-2})}$$

   - Aplicar el metodo iterativo siguiente con $E = 10^{-8}$ para encontrar el punto de interseccion:

   $$X_{n+1} = X_n - f(X_n)\frac{X_n - X_{n-1}}{f(X_n) - f(X_{n-1})}$$

2. Determine el valor de los coeficientes a y b tal que f(1) = 3 y f(2) = 4 con f(x) = a + (ax + b)eax+b. Obtenga la respuesta con $E = 10^{-6}$

3.2. Parte 2

Sea $f(x) = e^x - x - 1$

1. Demuestre que tiene un cero de multiplicidad 2 en x = 0

2. Utilizando el metodo de Newton con $p_0 = 1$ verifique que converge a cero pero no de forma cuadratica

3. Utilizando el metodo de Newton generalizado, mejora la tasa de rendimiento

   Las comprobaciones de cada una de los items anteriores se realizan por medio del algoritmo realizado en R. El metodo de Newton en este contexto dada la ecuacion f(x) descrita anteriormente, nos muestra que aplicandolo logra comprobar la convergencia de la funcion en el valor p_0. Se muestra a continuacion los datos obtenidos como salida del algoritmo.

```
2.500021e-05
1.250016e-05
6.250095e-06
3.125079e-06
1.562567e-06
7.81288e-07
3.907929e-07
1.953356e-07
9.871328e-08
4.922672e-08
2.667346e-08
1.002435e-08
```

4. Convergencia Acelerada

4.1. Metodo de Aitken

Dada la sucesion $x_n \infty n = 0 con \mathrm{x}_n = cos(1/n)$

1. Sean f(x) = ln(x + 2) y g(x) = sin(x) dos funciones de valor real.
2. Compare los primeros terminos con la sucesion $x_n \infty n = 0$

3. Sean $f(t) = 3sin^3(t) - 1$ y g(t) = 4sin(t) cos(t) para t ≥ 0 las ecuaciones parametricas que describe el movimiento en una particula. Utilice un metodo de solucion numerico con error de 10^{-6}para determinar donde las coordenadas coiciden

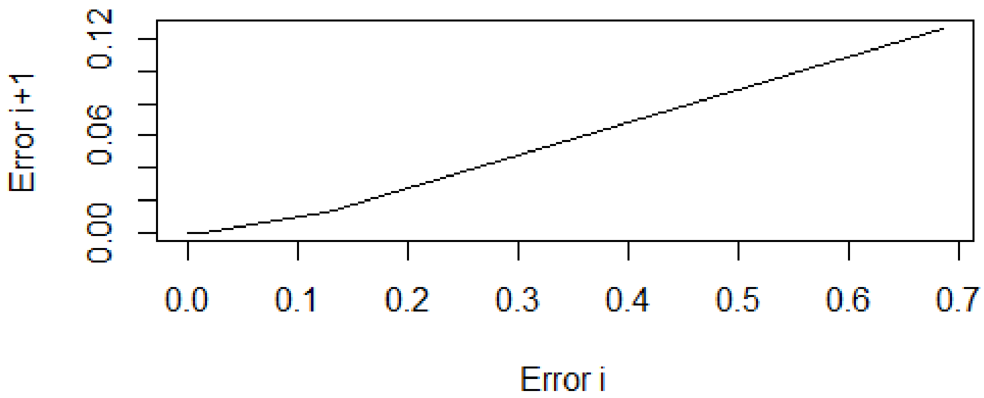
4.2. Metodo de Steffersen

1. Utilice el algoritmo de Steffersen para resolver $x^2 - cos(x)$ y compararlo con el metodo de Aitken.
   Nota Debe implementar los codigos en R, utilice Rcpp para mejorar el rendimiento

- **Metodo de Steffersen**

```
i               x_i            f(x)            Error est.
 1.00000000     -0.68507336    -0.30504713      0.68507336
 2.00000000     -0.81136839    -0.03018801      0.12629504
 3.00000000     -0.82400763    -0.00029700      0.01263924
 4.00000000     -0.82413230    -0.00000003      0.00012467
 5.00000000     -0.82413231    -0.00000000      0.00000001
Cero de funcion:  -0.8241323  con error <= 1.208145e-08 Iteraciones:  5
```

- **Metodo de Aitken**



Iteracion	Cero	Error
1.0000000	1.1004524	0.8995476
2.0000000	0.8553926	0.2450598
3.0000000	0.8246602	0.0307324
4.0000000	0.8241325	0.0005277
5.0000000	0.8241323	0.0000002

```
Raiz:  0.8241323  con valor inicial 2 , multiplicidad 1 , Error <=  0.0000002
```