Taller 1 Análisis Numérico

Marlon Esteban Linares Zambrano
Juan Felipe Marin Florez
Brian David Hortua Viña

14 de febrero de 2020

1 Problemas (Taller 0)

1.1 Problema 1

Error de redondeo para 536.78 almacenando los cuatro primeros dígitos decimales. Tenemos que el error de redondeo está dado por la fórmula:

$$E_{redondeo} = \frac{fl(x)-x}{|x|}$$

Donde $fl(x)$ es el valor de X obtenido a través de la aproximación. Como nos dice el enunciado, el dispositivo solo almacena los 4 primeros dígitos decimales, el resto los trunca haciendo un redondeo inferior. En el caso del número en cuestión, el truncamiento quedaría:

$$536.7\mathbf{8} \rightarrow 536.\mathbf{7}$$

Entonces el error de redondeo sería:

$$\frac{|536.7-536.78|}{|536.78|} = 1,49 \cdot 10^{-4}$$

1.2 Problema 2

Algoritmo a implementar

Algorithm 1 Problema 2: calcular raíz cuadrada

procedure PROBLEMA2(n, E, x)
  $y \leftarrow \frac{1}{2}(x + \frac{n}{x})$
  while $|x - y| > E$ **do**
    $x \leftarrow y$
    $y \leftarrow \frac{1}{2}(x + \frac{n}{x})$
  end while
  return y
end procedure

1.3 Problema 3: aproximación por el teorema de Taylor

Hallar una aproximación utilizando un polinomio de Taylor para la función e^x evaluado en 0.5 y redondeado a cinco cifras significativas. Para este caso se realizará una implementación del polinomio de Taylor hasta el grado 3 con el cual se puede dar la aproximación para el punto 0.5. Se realizó lo siguiente:

$$P(0.5) = a_o + a_1(x - 0.5) = 1.5$$
$$P(0.5) = a_o + a_1(x - 0.5) + a_2(x - 0.5)^2 = 1.625$$
$$P(0.5) = a_o + a_1(x - 0.5) + a_2(x - 0.5)^2 + a_3(x - 0.5)^3 = 1.648724$$

Con el grado 3 se pudo obtener una aproximación con cinco cifras significativas al calculo de la función. Pues el valor real de $e^0.5$ corresponde a 1.6487212707

1.4 Problema 4. Error relativo y absoluto

Debemos calcular el error causado por las operaciones aritméticas del siguiente ejercicio.

La velocidad de una partícula es constante e igual a 4 m/s, medida con un error de 0.1 m/s durante un tiempo de recorrido de 5 seg medido con un error de 0.1 seg. Determine el error absoluto y el error relativo en el valor de la distancia recorrida.

$$v = 4, E_v = 0.1 \text{ (velocidad)}$$
$$t = 5, E_t = 0.1 \text{ (tiempo)}$$
$$\text{d=vt (distancia recorrida)}$$

Para calcular el error absoluto usamos la siguiente fórmula:

$$|E_{xy}| = |\overline{x}E_y| + |\overline{y}E_x|$$

Reemplazando obtenemos:

$$|E_{xy}| = 4(0.1) + 5(0.1) = 0.9$$

Para calcular el error relativo tenemos la siguiente fórmula:

$$e_{vt} = e_d = \frac{E_v}{\tilde{v}} + \frac{E_t}{\tilde{t}}$$

Reemplazando tenemos:

$$e_{vt} = e_d = e_{vt} = e_d = \frac{0.1}{4} + \frac{0.1}{5} = 0.045$$
$$0.045 \rightarrow 4.5\%$$

2 Taller 1

2.1 Numero de operaciones

2.1.1 Demostración método 3 del teorema de Horner y pseudocódigo del algoritmo

Sea $P(x) = a_0X^n + a_1X^n - 1 + ... + a_n$ un Polinomio entonces,
Método 3:

$$b_0 = a_0$$
$$b_k = a_k + b_k - 1x_0(k = 1, ..., n-1)$$
$$b_k = P(x_0)$$

Se demuestra de manera computacional el método 3 donde el resultado es n aplicando el siguiente algoritmo:

Algorithm 2 Método 3

procedure HORNER($x, listaCoeficientes$)
  $resultado \leftarrow 0$
  $contador \leftarrow 0$
  for $i \leftarrow |listaCoeficientes|$ **do**
    $resultado \leftarrow resultado * x + listaCoeficientes[i]$
    $contador \leftarrow contador + 1$
  end for
  return $reultado, contador$
end procedure

donde los resultados dan los siguientes con $x = 2$ y todos los coeficientes en 1:

grado polinomio	resultado algoritmo	resultado polinomio
0	0	0
1	1	1
2	2	3
3	3	7
4	4	15
5	5	31
6	6	63
7	7	127
8	8	255
9	9	511

Table 1: Resultados del metodo 3 aumentado en 1 en tamaño del polinomio con $x = 2$ y los coeficientes en 1

2.1.2 Evaluacion en $x = 0.0001$ con $P(x) = 1 + x + x^2 + ... + x^5 0$ y error al comparar con $Q(x) = (x^5 1 - 1)/(x - 1)$

Evaluando en $x = 0.0001$ en la primera funcion, la cual es la funcion. planteada normalmente el resultado obtenido 51.127708500135086.
Evaluando en $x = 0.0001$ en la función equivalente deducida por el Teorema de Horner el resultado obtenido es 51.12770850013501.

Para entender cual seria el error entre estas dos medidas se procede a realizar el calculo del error absoluto y del error relativo.

$$e_{abs} = |V_{real} - V_{aprox}|$$
$$e_{abs} = |51.127708500135086 - 51.12770850013501|$$
$$e_{abs} = 0.000699999999924$$

Ahora se procede a calcular el error relativo

$$\epsilon = \frac{e_{abs}}{V_{real}} * 100$$
$$\epsilon = \frac{0.0069999999999924}{51.127708500135806} * 100$$
$$\epsilon = 0.0136912062076357384 5\%$$

Para este caso tenemos que el error relativo calculado es de aproximadamente 0.0136%

2.2 Numeros binarios

Para solucionar los siguientes ejercicios se uso los siguientes algoritmos: endiendo la funcion $techo(x)$ como funcion de parte entera

Algorithm 3 Algoritmo de base 10 a 2

procedure BASE10ABINARIO($numero, numeroBits$)
    $entero \leftarrow techo(|numero|)$
    $decimal \leftarrow |numero| - techo(|numero|)$
    $contador \leftarrow 0$
    while $(entero \neq 0 \cap contador \neq numeroBits)$ **do**
        if $entero mod 2 = 0$ **then**
            $binarioParteEntera \leftarrow 0 + binarioParteEntera$
        else
            $binarioParteEntera \leftarrow 1 + binarioParteEntera$
        end if
        $contador \leftarrow contador + 1$
    end while
    while $contador \neq numeroBits$ **do**
        $decimal \rightarrow decimal * 2$
        $contador \leftarrow contador + 1$
        if $decimal > 1$ **then**
            $decimal \leftarrow decimal - 1$
            $binariopartedecimal \leftarrow binariopartedecimal + 1$
        else
            $binariopartedecimal \leftarrow binariopartedecimal + 1$
        end if
    end while
    $total \leftarrow binarioParteEntera +' .' + binariopartedecimal$
    return $total$
end procedure

Algorithm 4 Algoritmo de base 2 a 10

```
procedure binarioaBase10(numero)
    entero ← techo(|numero|)
    decimal ← |numero| − techo(|numero|)
    max ← |entero| − 1
    numb ← 0
    for i ← |entero| do
        num ← entero[i] ∗ 2^max
        max ← max − 1
        numb ← numb + num
    end for
    max ← 1
    for i ← Decimal do
        numb ← numb + deicmal[i] ∗ 1/2^max
        max ← max + 1
    end for
    return numb
end procedure
```

2.2.1 15 bit del numero *pi*

$pi = 11.0010010000111$

2.2.2 binarios a base 10

- $1010101 \rightarrow 85$
- $1011.101 \rightarrow 11.625$
- $10111.010101 \rightarrow 23.328125$
- $111.111111 \rightarrow 7.984375$

2.2.3 base 10 a binarios

- $11.25 \rightarrow 1011.001111$
- $\frac{2}{3} \rightarrow .1010101010$
- $30.6 \rightarrow 11110.10011$
- $90.9 \rightarrow 1011010.111$

2.3 Punto flotante y Epsilon de la máquina

2.3.1 Representación de un numero infinito en un numero finito de bits

De acuerdo con la especificación IEEE-754, si un valor numérico es demasiado grande para ser representado por este estándar, será representado por el valor infinito, la representación del número infinito en punto flotante se divide en 3 partes:

Valor	Signo	Exponente	Mantisa
Infinito positivo (+∞)	1	00000000	00000000000000000000000
Infinito negativo (-∞)	0	00000000	00000000000000000000000

Table 2: Representación de los valores infinito positivo y negativo en bits de acuerdo al estándar IEEE-754

2.3.2 Diferencia entre redondeo y truncamiento

Primero hay que entender a que se refiere redondear o truncar un numero decimal.

El Redondeo consiste en no considerar los decimales, cortando el número para quedarse sólo con el entero. Esto quiere decir, si queremos redondear el número 2,3, eliminaremos el 0,3 y nos quedaremos con el 2. En cambio, si el objetivo es redondear 4,9, el mecanismo de redondeo llevará a dejar de lado el 0,9 y a sumar un 0,1 para poder trabajar con el número 5. Dependiendo del numero de cifras que se deseen conservar se procede a realizar el redondeo.

19.95 conservar dos cifras → 20
19.68 conservar tres cifras → 19.7
19.35 conservar dos cifras → 19

El Truncamiento puede ser definido como un procedimiento matemático en el cual se eliminan algunas de las cifras menos significativas de la parte decimal de un número decimal, con el fin de lograr un número mucho más manejable.

3.14159265359 truncar a 4 cifras decimales → 3.1415
2.16854 truncar a 2 cifras decimales → 2.16

La principal diferencia del proceso de truncamiento con el de redondeo es que el primer método no se preocupa por hacer la aproximación al numero superior o inferior. Simplemente realiza la eliminación de las cifras.

2.3.3 Coma flotante de doble precisión

para poder mostrar la representación binaria en coma flotante de doble precisión propuesta por la IEEE en su estándar 754 se tiene que descomponer el numero en las características propuestas. Es decir: verificar su signo; su exponente; y su representación de la mantisa. Para esto el numero en cuestión (0.4) debe representarse en forma exponencial. así:

signo: 0
Exponente: 01111101
Mantisa: 10011001100110011001101
Sistema Octal: $0X3ECCCCCD$

Cabe aclarar que la máquina toma un numero ya conocido para hacer esta aproximación al numero 0.4 y esta aproximacion es:

$$0.4000000059604644775390625$$

2.3.4 Error de redondeo

Tenemos que el error de redondeo relativo no es más que la mitad del épsilon de la máquina:

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

A partir del resultado obtenido en la sección anterior, obtuvimos que:

$$\frac{fl(x)-x}{|x|} = 1.49011925 \cdot 10^{\,-8}$$

Teniendo presente que el valor del épsilon de la máquina es:

$$\epsilon_{maq} = 1.19209 \cdot 10^{-7}$$

Entonces:

$$\frac{1}{2}\epsilon_{maq} = 5.96045 \cdot 10^{-8}$$

Podemos deducir que:

$$1.49011925 \cdot 10^{\,-8} \leq 5.96045 \cdot 10^{-8}$$

Cumpliendo así el modelo de la aritmética de computadora IEEE.

2.3.5 Tipos de dato básico en R y Python

En python el float es de punto flotante de doble precision. En R, el dato tipo numérico (*numeric*) se maneja idénticamente como punto flotante de doble precisión de acuerdo al estándar IEEE 754. Desde python3, long se unificó en int, que es de coma flotante de precisión fija. En R no existe long, solo tenemos dos tipos de datos para almacenar números: *numeric e integer*.

2.3.6 Representación en hexadecimal de un número real

Para representar el valor en hexadecimal del número real 8.4, vamos a descomponera en punto flotante para poder realizar la conversión a hexadecimal

Signo: 0
Exponente: 10000010
Mantissa: 00001100110011001100110

4	0
	1
	0
	0
1	0
	0
	0
	1
0	0
	0
	0
	0
6	0
	1
	1
	0
6	0
	1
	1
	0
6	0
	1
	1
	0
6	0
	1
	1
	0
6	0
	1
	1
	0

Con base a la tabla mostrada a continuación, podemos concluir que el valor hexadecimal de 8.4 es $0x41066666$.

2.3.7 Raíces de una ecuación cuadrática

Para hallar las raices de la ecuación: $x^2 + bx - 10^{-12} = 0$, usaremos el método de bisección, que se d

2.4 Raices de una ecuación

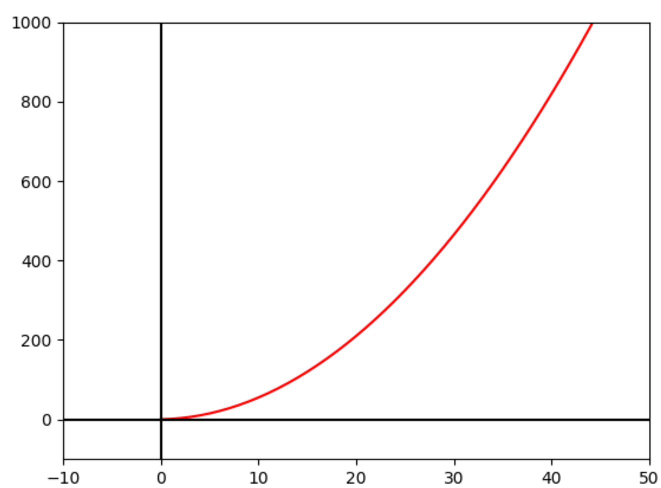
2.4.1 sumar valores de la submatriz triangular superior de una matriz dada la matriz cuadrada A_n

Algorithm 5 Algoritmo que resuelve la triangular inferior de una matriz dada

procedure TRIAGULARINFERIOR($Constantes, coeficientes$)
    $tam \leftarrow |constantes|$
    $resultado \leftarrow \emptyset$
    for $i \leftarrow tam$ **do**
        $sum \leftarrow 0$
        for $j \leftarrow i$ **do**
            $sum \leftarrow sum + coeficientes_{ixj} * resultado_j$
        end for
        $resultado_i \leftarrow \frac{constantes_i - sum}{coeficientes_{ixi}}$
    end for
    $returnresultado$
end procedure

Como se puede ver en la siguiente grafica arrojada por al programa triangular.py adjunto se puede ver la complejidad o su orden de convergencia

Figure 1: Numero de operaciones de la funcion triangular aumentando el tamaño de la matriz: $O(n + log_2(n))$

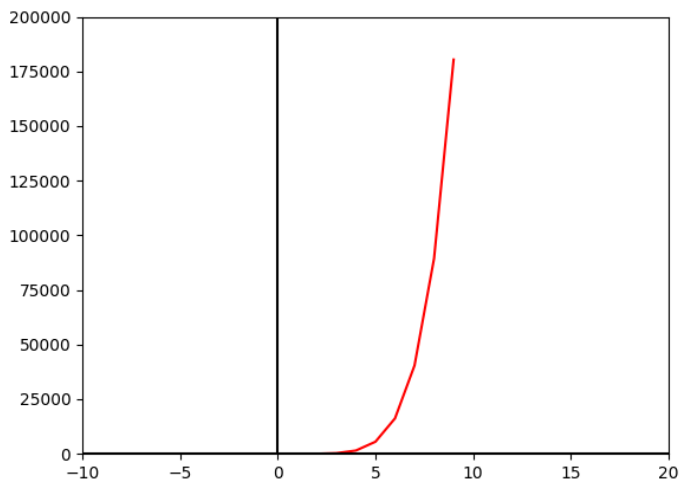


2.4.2 Algoritmo que suma los primeros n^2 numeros naturales

Algorithm 6 Algoritmo que suma los primeros n^2 numeros naturales

procedure SUMACUADRADOS(n)
    $tam \leftarrow n^2$
    $suma \leftarrow 0$
    for $i \leftarrow tam$ **do**
    end for
    $returnsuma$
end procedure

Figure 2: Resultados de aplicar la funcion $SumaCuadrados(n)$ con $n \rightarrow 10$



2.4.3 Solucion del problema de Cohete con métodos no lineales

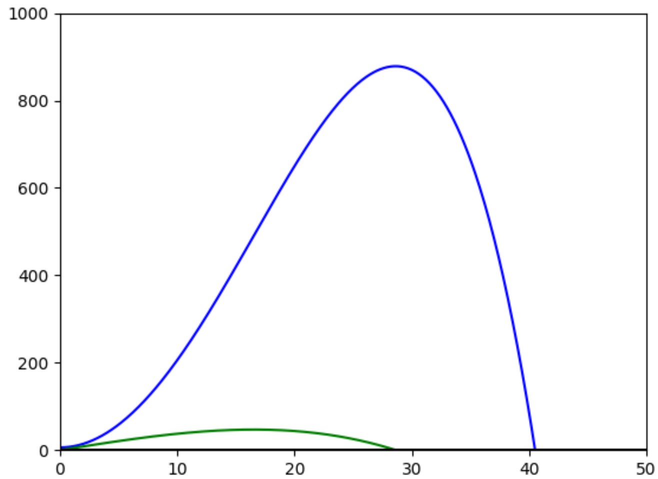
Para describir la trayectoria de un cohete se tiene el modelo:

$$y(t) = 6 + 2,13t^2 - 0.0013t^4$$

Donde, y es la altura en [m] y t tiempo en [s]. El cohete esta colocado verticalmente sobre la tierra. Utilizando dos métodos de solución no lineal, encuentre la altura máxima que alcanza el cohete.
Para este ejercicio vamos a primero encontrar la derivada para con esta encontrar el punto donde da 0 (la raiz) que es el punto mas alto que alcanza el cohete. La derivada sería la siguiente:

$$y'(t) = 4.26t - 0.0052t^3$$

Figure 3: Grafica generada en Python de la función: $y(t) = 6 + 2.13t^2 - 0.0013t^4$ en azul y la grafica $y'(t) = 4.26t - 0.052t^3$ en verde



1. Método de la bisección.

Algorithm 7 método de la bisección

procedure MÉTODOBISECCIÓN(F, x_0, x_1, e)
    while $x_1 - x_0 >= e$ **do**
        $x_2 \leftarrow \frac{x_0+x_1}{2}$
        if $f(x_2 = 0)$ **then**
            return x_2
        else
            if $f(x_0) * f(x_2) > 0$ **then**
                $x_0 = x_2$
            else
                $x_1 = x_2$
            end if
        end if
    end while
    return x_2
end procedure

usando el algoritmo con la función $y'(t) = 4.26t - 0.052t^3$ y guiándonos de la grafica poniendo los limites como $(5, 15)$ y con $e = 0.0001$ dio como resultado:

$$MetodoBiseccion(y, 25, 35, 0.0001) = 28.622207641601562$$

aplicando eso a la funcion:

$$y(28.622207641601562) = 878.4807692307692$$
$$y'(28.622207641601562) = -1.5600721781083848e - 07$$

Podemos concluir que la altura maxima del cohete fue: 878.4807692307692