Pontificia Universidad Javeriana

Taller Analisis Numerico

Profesora

Eddy Herrera Daza

Integrantes

Diego Arroyo
arroyodiego@javeriana.edu.co
git: DiegoArroyoG

Santiago Caro
santiago.caro@javeriana.edu.co
git: SantiagoCaroDuque

Nicolas Lopez
lopezn.i@javeriana.edu.co
git: NicolasLopezFer

7 de agosto de 2019

1. Tabla de Contenidos

2. Introducción

En este taller se van a tratar temas referentes a la resolución de raíces reales en ecuaciones no lineas, tales que, dado un $f(x)$ se obtenga un $f(r) = 0$, entonces r sera una raíz real de la ecuación; para lograr esto se analizaran y se pondrán en practica métodos matemáticos iterativos, como: Método de Bisección, Método de Punto Fijo, Método de Newton, entre otros.

Además de métodos numéricos usados para diferentes fines, cada uno de estos problemas tendrá una breve descripción, un diseño que consta de unos valores de entrada y unos de salida y el algoritmo responsable de solucionar el problema dado. Además también tendrán, en caso de ser necesario una gráfica en donde se muestra el orden de convergencia de cada uno de los algoritmos.

3. Método de Bisección

El método consiste en calcular de forma iterativa el punto medio ($c = (a + b)/2$) en el intervalo $[a, b]$ y luego sustituirlo por el intervalo $[a, c]$ o $[c, b]$, manteniendo la condición de que una de las variables debe ser positiva y la otra negativa, hasta que la distancia entre a y b sea muy pequeña, entonces el ultimo valor calculado (c) sera muy cercano a la raíz de la ecuación.

3.1. Diseño

3.1.1. Valores de entrada

E : Tolerancia. a : Inicio del intervalo. b : Final del intervalo.

3.1.2. Valores de salida

c : Valor cercano a la raíz de la ecuación.

3.2. Algoritmo

Algorithm 1 Método de Bisección.

```
1: procedure BISECCIÓN(E, a, b)
2:     if f(a) * f(b) > 0 then
3:         return −1
4:     end if
5:     while |b − a| > E do
6:         c = (a + 2) ÷ 2
7:         if c = 0 then
8:             return c
9:         end if
10:        if f(a) * f(c) < 0 then
11:            b ← c
12:        else if  then
13:            a ← c
14:        end if
15:    end while
16:    return c
17: end procedure
```

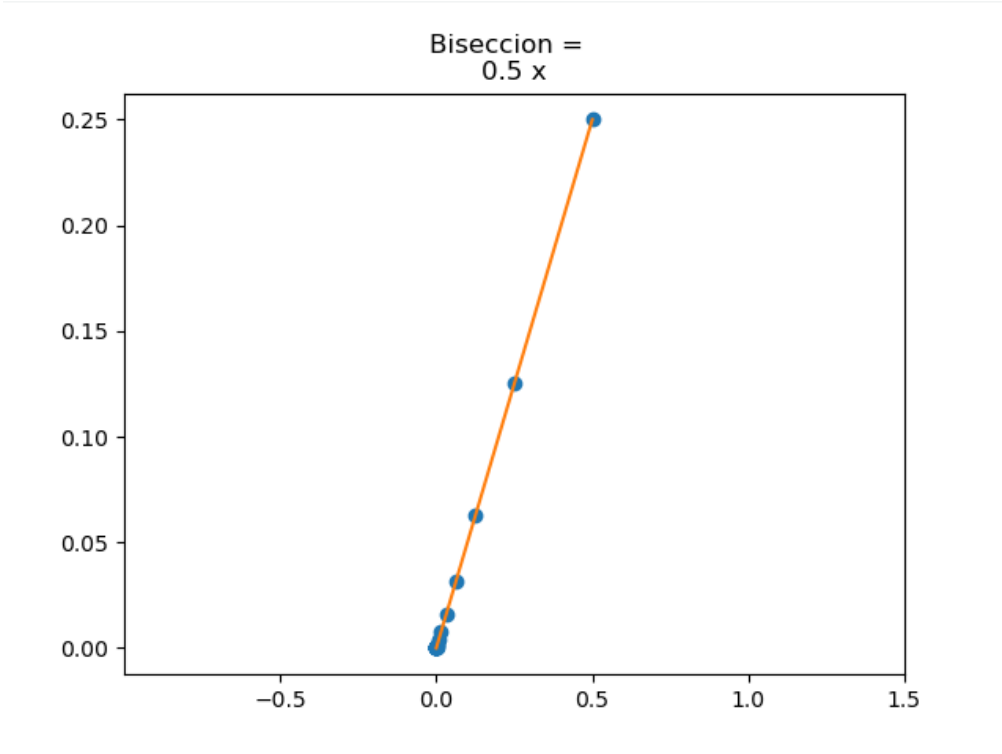

Figura 1: Gráfica de Convergencia del Método de Bisección

4. Método de Punto Fijo

El método de punto fijo consiste en despejar un $f(x)$ en función de x, es decir, obtener un $g(x) = x$, dado un x inicial que iterativamente se le asignara el valor de $g(x)$, finalmente cuando esto se cumpla, se tendrá un $(g(x) = x) \approx r \approx (f(x) = 0)$, donde x sera el valor aproximado de la raíz de la ecuación.

4.1. Diseño

4.1.1. Valores de entrada

E : Tolerancia. x : Valor inicial.

4.1.2. Valores de salida

x : Valor final, cercano a la raíz de la ecuación.

4.2. Algoritmo

Algorithm 2 Método de Punto Fijo.

```
1: procedure PUNTO FIJO(E, x)
2:     while |g(x) − x| > E do
3:         x ← g(x)
4:     end while
5:     return x
6: end procedure
```

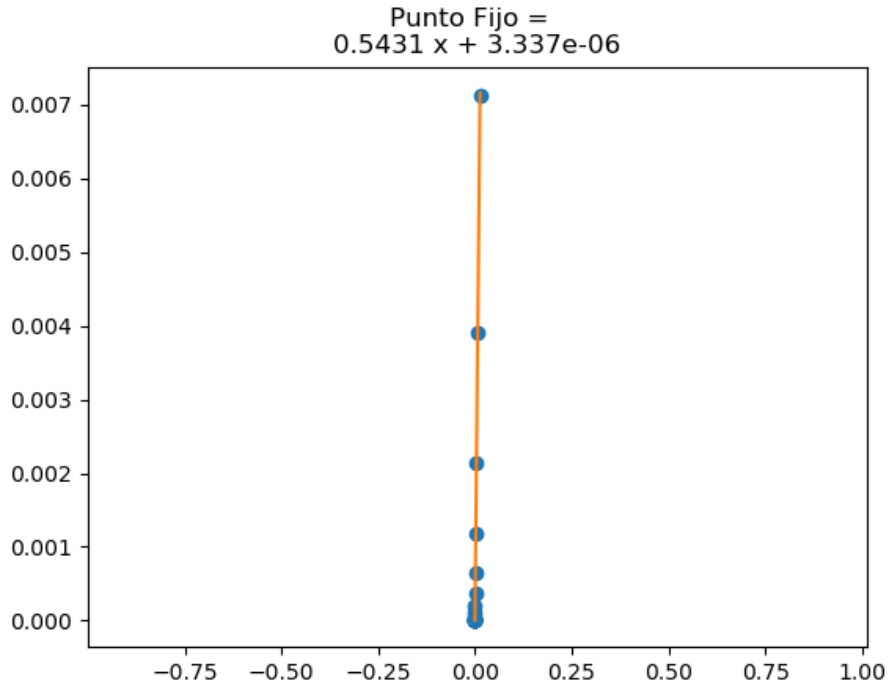

Figura 2: Gráfica de Convergencia del Método Punto Fijo

5. Método de Newton

El método de newton consiste en una formula iterativa, dado un $f(x)$, se halla su $f'(x) \neq 0$ y desde un x inicial se resuelve la formula $x_{i+1} = x_i - (f(x_i) \div f'(x_i))$.

5.1. Diseño

5.1.1. Valores de entrada

E : Tolerancia. x : Valor inicial.

5.1.2. Valores de salida

x_i : Valor final, cercano a la raíz de la ecuación.

5.2. Algoritmo

Algorithm 3 Método de Newton.

1: **procedure** NEWTON(E, x)
2: $x_i \leftarrow x - (f(x) \div f'(x))$
3: **while** $|x_i - x| > E$ **do**
4: $x \leftarrow x_i$
5: $x_i \leftarrow x - \frac{f(x)}{f'(x)}$
6: **end while**
7: **return** x_i
8: **end procedure**

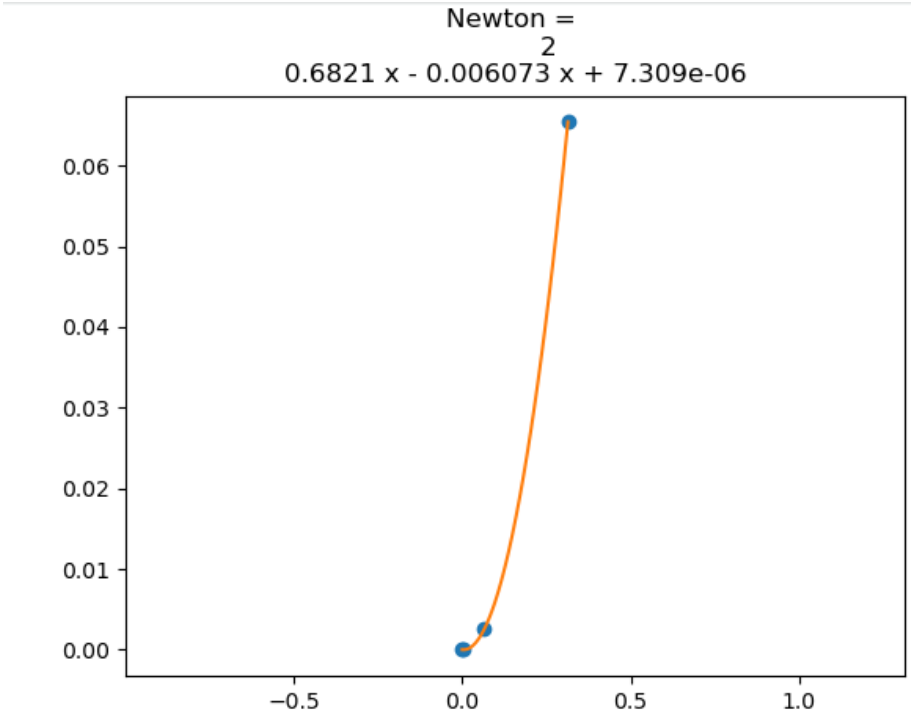

Figura 3: Gráfica de Convergencia del Método Newton

6. Método de la Linea Secante

Este método es un derivado del Método de Newton pero tiene una ventaja clara sobre este y es que no depende de que exista la primera derivada para poder encontrar las raíces, o que en dado caso la derivada sea 0. Es un método, como Newton, que la encuentra de forma iterativa. Ejecuta el algoritmo la cantidad de veces que sea necesaria hasta que el error relativo sea menor que la tolerancia establecida anteriormente.

6.1. Diseño

6.1.1. Valores de entrada

- E : Tolerancia.
- x : Valor inicial.
- a : Inicio de rango, donde la linea secante toca por primera vez la función.
- b : Inicio de rango, donde la linea secante toca por segunda y ultima vez la función.

6.1.2. Valores de salida

x_i : Valor final, cercano a la raíz de la ecuación.

6.2. Algoritmo

Algorithm 4 Método de linea Secante.

1: **procedure** SECANTE(E, x, a, b)
2: $x_i \leftarrow x - \frac{f(x)}{\frac{f(b)-f(a)}{b-a}}$
3: **while** $|x_i - x| > E$ **do**
4: $x \leftarrow x_i$
5: $x_i \leftarrow x - \frac{f(x)}{\frac{f(b)-f(a)}{b-a}}$
6: **end while**
7: **return** x_i
8: **end procedure**

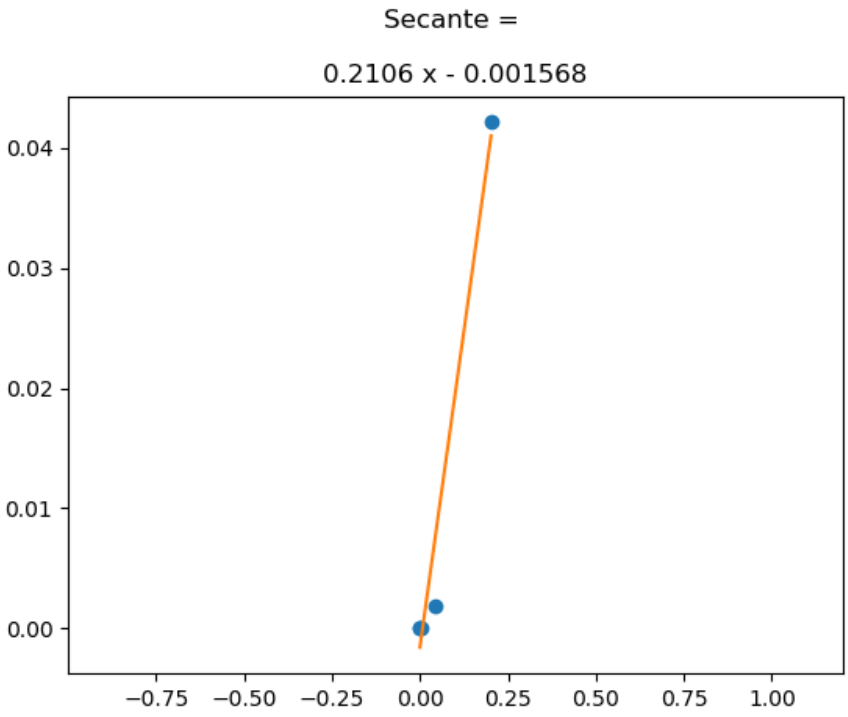

Figura 4: Gráfica de Convergencia del Método Secante

7. Método de Steffensen

El método de Steffensen al igual que Newton se basa en una formula iterativa a partir de un $f(x)$, pero en esta formula se elimina la derivada para evitar casos complejos o $f'(x) = 0$, quedando así, $x_{i+1} = x_i - ([f(x_i)]^2 \div (f(x_i + f(x_i)) - f(x_i)))$.

7.1. Diseño

7.1.1. Valores de entrada

E : Tolerancia. x : Valor inicial.

7.1.2. Valores de salida

x_i : Valor final, cercano a la raíz de la ecuación.

7.2. Algoritmo

Algorithm 5 Método de Newton.

```
1: procedure NEWTON(E, x)
2:     x_i ← x − ([f(x)]^2 ÷ (f(x + f(x)) − f(x)))
3:     while |x_i − x| > E do
4:         x ← x_i
5:         x_i ← x − ([f(x)]^2 ÷ (f(x + f(x)) − f(x)))
6:     end while
7:     return x_i
8: end procedure
```

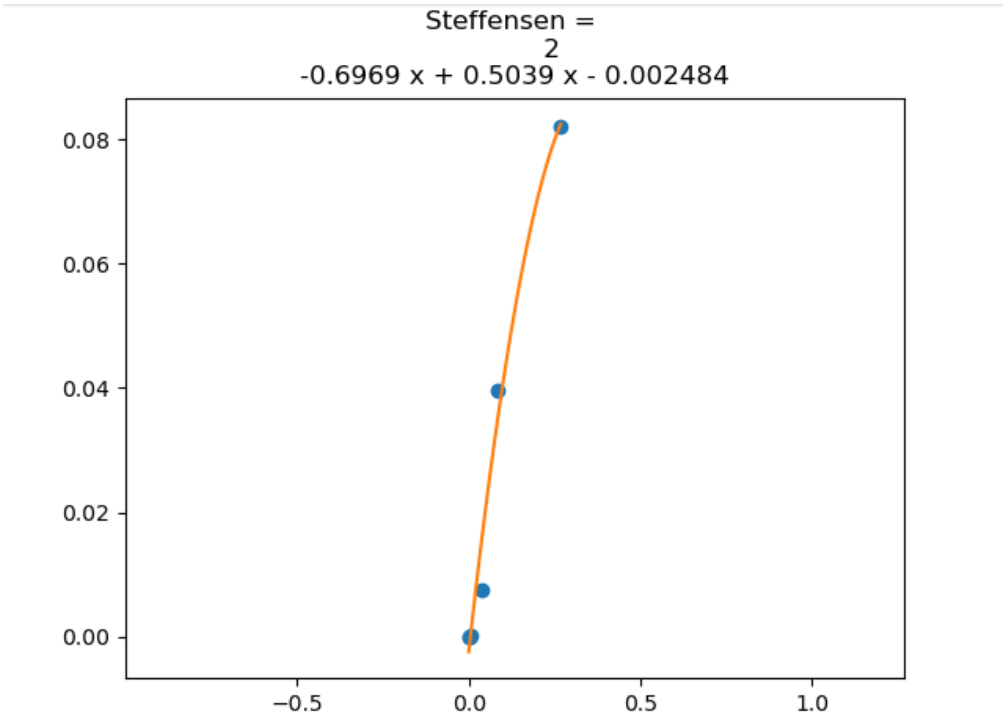

Figura 5: Gráfica de Convergencia del Método Steffensen

8. Punto 13

Este punto del libro se trata de encontrar un algoritmo iterativo el cual encuentre cualquier raíz de cualquier numero aleatorio Real. Una de las condiciones de este algoritmo es que tiene que converger de una manera cuadrática y solo puede usar operaciones fundamentales de la Aritmética (Suma, Resta, Multiplicación y División)

8.1. Diseño

Encontramos un algoritmo que con una ecuación resuelve este problema. Hay que tener en cuenta que este algoritmo tiene dentro de su ecuación un numero que es elevado a otro numero, pero para no salirnos de las condiciones propuestas por el ejercicio decidimos que esta potencia se iba a calcular de manera iterativa también, realizando solamente las multiplicaciones y cumpliendo que solo se utiliza la aritmética fundamental. Este algoritmo entonces usa la siguiente ecuación:

$$x_{k+1} = \frac{1}{n}\left[(n-1)x_k + \frac{A}{x_k^{n-1}}\right]$$

8.1.1. Valores de Entrada

- $n \rightarrow$ Índice de la raíz .

- $a \rightarrow$ Radicando.
- $E \rightarrow$ Tolerancia de la respuesta del problema.
- $x_0 \rightarrow$ Valor inicial de la respuesta.

8.1.2. Valores de Salida

- $x_i \rightarrow$ Valor final de la n-esima raíz del radicando a con una tolerancia de E

8.2. Algoritmo

Algorithm 6 Algoritmo que encuentra la n-esima raiz de a

```
1: procedure N-ESIMARAIZ(a, n, x, E)
2:     r ← (1/n) * ((n − 1) * x + (a/power(x, n − 1)))
3:     while abs(r − x) > E do
4:         x ← r
5:         r ← (1/n) * ((n − 1) * x + (a/power(x, n − 1)))
6:     end while
7:     return r
8: end procedure
```

Algorithm 7 Algoritmo que hace una potencia de forma iterativa

```
1: procedure POWER(num, pot)
2:     r ← num
3:     for i ← 0 to pot − 1 do
4:         r ← r * num
5:     end for
6:     return r
7: end procedure
```

9. Punto 26

Este punto es una de las aplicaciones para los algoritmos que fueron desarrollados en clase y además se trata de una aplicación de la vida real en donde se ve claramente cual es el uso de estos algoritmos.

9.1. Diseño

Para esto entonces usamos la definición de la derivada, sabemos que cuando la derivada se iguala a 0 es que este es un punto máximo o mínimo para la función. En este caso sabemos que en el intervalo de $[25, 30]$ se trata de un máximo, este nos da entonces el tiempo de vuelo y sabiendo esto y reemplazando en la ecuación original sabemos entonces la altura máxima.

9.1.1. Valores de Entrada

- $E \rightarrow$ Error permitido con el que se calcula el tiempo de vuelo.
- $a \rightarrow$ Inicio del intervalo.
- $b \rightarrow$ Fin del intervalo
- $f(x) \rightarrow$ Función de vuelo.
- $f'(x) \rightarrow$ Derivada de la Función de vuelo.

9.1.2. Valores de Salida

- $t \rightarrow$ Tiempo de vuelo.
- $h \rightarrow$ Altura máxima del vuelo

9.2. Algoritmo

Algorithm 8 Algoritmo que obtiene el tiempo máximo y altura máxima

```
1: procedure VUELO(E, a, b, f(x), f'(x))
2:     t ← biseccion(E, a, b, f'(x))
3:     h ← f(t)
4:     return t, h
5: end procedure
```

10. Punto 27

10.1. Diseño

Este punto da dos ecuaciones perimétricas $x(t), y(t)$ las cuales tienen la misma variable t pero la primera da las coordenadas en x y la segunda las coordenadas en y. La idea es conseguir los puntos en donde se intersectan en un rango definido.

Para esto vamos a usar uno de los algoritmos ya propuestos antes pero con un cambio, para saber donde son iguales estas dos ecuaciones son iguales lo que hacemos es igualarlas y tomarla como una sola ecuación nueva $h(t)$.

Lo primero que hay que verificar es si hay una intersección entre estas dos ecuaciones en el rango definido. Para esto entonces hay que calcular $h(a)$ $h(b)$ en donde a es donde empieza el intervalo y b donde acaba y multiplicar estos dos valores, si su resultado es menor a 0, significa que si hay un lugar en donde se intersectan. Y por ultimo entonces con el método de Bisección que es el que recibe un intervalo, se encuentra la raíz y este es el punto en donde se intersectan.

10.1.1. Valores de entrada

- $[a, b]$ Intervalo en el cual existe una intersección entre las dos ecuaciones
- $h(t)$ La ecuación final que une a las dos anteriores

10.1.2. Valores de Salida

- t_i en donde $x(t_i) = y(t_i)$

10.2. Algoritmo

Algorithm 9 Punto 27

```
1: procedure 27(h, a, b)
2:     if Revisar(h, a, b) then
3:         Si hay un punto donde son iguales
4:         r ← biseccin(0,000000010, 0, π/2)
5:     end if
6: end procedure
```

Algorithm 10 Revisar intersección

```
1: procedure REVISAR(h, a, b)
2:     if h(a) * h(b) < 0 then
3:         return True
4:     else
5:         return False
6:     end if
7: end procedure
```

11. Taller: 1

Este problema resuelve cuanto es el error de redondeo que hay entre un numero con 5 cifras significativas y su redondeo con 4 cifras significativas.

11.1. Diseño

Para esto entonces se uso la fórmula básica para obtener el error la cual es:

$$E = \frac{V_{exp} - V_{teo}}{V_{teo}}$$

En donde estamos hablando que el valor experimental es el valor con 4 cifras significativas y el teorico es el valor con 5 cifras significativas.

11.1.1. Valores de Entrada

- $V_{exp}(a) \rightarrow$ Valor con 4 cifras significativas
- $V_{teo}(b) \rightarrow$ Valor con 5 cifras significativas

11.1.2. Valores de Salida

- $E \rightarrow$ Error de redondeo entre estas dos cifras.

11.2. Algoritmo

Algorithm 11 Calcular Error

```
1: procedure ERROR(a, b)
2:     x ← |a − b|
3:     y ← x/b
4:     E ← y ∗ 100
5:     return E
6: end procedure
```

12. Taller: 2

La idea de este problema es implementar este algoritmo iterativo que encuentra la raíz cuadrada de un numero real con cierta tolerancia determinada desde antes en el mismo programa. Este es un algoritmo iterativo ya que con un valor inicial calcula un nuevo valor hasta que cumpla con la condición que su error con respecto al calculo anterior sea menos que la tolerancia.

12.1. Diseño

En este caso el mismo problema nos da la formula para resolver el problema en general de la raíz cuadrada. En este caso entonces esta formula seria:

$$Y \leftarrow \frac{1}{2}(x + \frac{n}{x})$$

En donde Y es el nuevo valor que esta siendo calculado, n es el radicando y x es el valor inicial de la función o el valor calculado en la iteración anterior.

12.1.1. Valores de Entrada

- $n \rightarrow$ Radicando de la raíz cuadrada.
- $E \rightarrow$ Error permitido o tolerancia del problema.
- $x \rightarrow$ Valor inicial con el cual se va a trabajar para encontrar la respuesta

12.1.2. Valores de Salida

- $y \rightarrow$ Valor de la raíz cuadrada de n con una tolerancia E

12.2. Algoritmo

Algorithm 12 Raiz Cuadrada de un numero

```
1: procedure SqRoot(n, E, x)
2:     Y ← ½(x + n/x)
3:     while |x-y|>E do
4:         x ← y
5:         Y ← ½(x + n/x)
6:     end while
7:     return Y
8: end procedure
```

13. Taller: 3

Este es un problema que nos lleva a la ejecución de un algoritmo iterativo, ya que por la misma definición de las Series de Taylor se trata de repetir una suma con unos valores predeterminados.

13.1. Diseño

Haciendo el adecuado proceso para obtener la serie de Taylor tenemos que la serie tiene la siguiente forma:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

En donde sabemos que para este caso $x = 0{,}5$ y realmente la suma no irá hasta infinito si no hasta que cumpla la condición de coincidir con 5 cifras significativas del valor real. Este valor real será tomado de la plataforma de *Wolfram Alpha* ya que el resultado obtenido tiene una gran cantidad de cifras.

13.1.1. Valores de Entrada

- $x \rightarrow$ Valor a evaluar la serie de Taylor
- $E \rightarrow$ Error o tolerancia a tener en cuenta para el calculo del resultado
- $teo \rightarrow$ Valor teórico del resultado real obtenido de Wolfram Alpha

13.1.2. Valores de Salida

- $y \rightarrow$ valor experimental de e^0,5 con la serie de Taylor con 5 cifras significativas iguales a las del valor teorico.

13.2. Algoritmo

Algorithm 13 Encuentra el Valor de la serie de taylor de e^x

```
1: procedure TAYLOR(x, E, teo)
2:     con ← 0
3:     y ← x^con / con!
4:     while |teo-y|>E do
5:         con ← con + 1
6:         y ← x^con / con!
7:     end while
8:     return y
9: end procedure
```

14. Taller: 4

Este problema ilustra como la propagación de errores por la incertidumbre de ciertas medidas o datos que se tienen y como se tienen que tener en cuenta al momento de obtener otro dato aparte que sea un derivado de estos primeros.

14.1. Diseño

Primero hay que tener clara las ecuaciones de Error Absoluto y Error relativo las cuales son:

$$E_A = v * E_t + t * E_v$$

$$E_R = \frac{E_v}{v} + \frac{E_t}{t}$$

Sabiendo que el error absoluto tiene una medida en metros (El cual entonces nos da un rango de variación) y el Relativo en porcentaje.

14.1.1. Valores de Entrada

- $v \rightarrow$ Dato de la velocidad medida.
- $E_v \rightarrow$ incertidumbre de la medida de velocidad.
- $t \rightarrow$ Dato del tiempo medido.
- $E_t \rightarrow$ incertidumbre de la medida del tiempo.

14.1.2. Valores de Salida

- $E_A \rightarrow$ Error Absoluto del calculo de la distancia.
- $E_R \rightarrow$ Error Relativo del calculo de la distancia.

14.2. Algoritmo

Algorithm 14 Encuentra errores absolutos y Relativos del calculo de distancia

```
1: procedure ERRORESDISTANCIA(v, t, ev, et)
2:     d ← v * t
3:     E_a ← v * E_t + t * E_v
4:     E_r ← E_v/v + E_t/t
5:     return d, E_a, E_r
6: end procedure
```

15. Taller: 5

La idea de este ejercicio es que dado un polinomio de un grado en especifico, que en este caso es de grado 4 y un valor de x_0 encontrar el valor del polinomio realizando la menor cantidad de multiplicaciones posibles.

15.1. Diseño

La manera que se resolverá este problema en especifico es ignorar aquellos datos que tengan 0 cómo coeficiente y además usando datos anteriores, sabiendo que $x^4 = (x^2)^2$ por lo tanto entonces para x^4 en vez de hacer 4 multiplicaciones se hacen solo dos. Para esto entonces se tendrá una colección con todos los resultados de los valores con potencias pares, facilitando el calculo de las próximas potencias pares.

15.1.1. Valores de Entrada

- $poli \rightarrow$ Coeficientes de cada uno de los términos del polinomio
- $val \rightarrow$ valor en el cual se va a calcular el polinomio.

15.1.2. Valores de Salida

- $r \rightarrow$ Valor del polinomio en el punto deseado

15.2. Algoritmo

Algorithm 15 Calcular Polinomio

```
1: procedure CALCULAR(poli, val)
2:     r ← 0
3:     poli ← poli.reverse()
4:     prev ← [|poli|/2|]
5:     for i ← 0 to|poli| do
6:         mul ← 0
7:         if i ≠ 0 ∧ i%2 = 0 then
8:             if i = 2 then
9:                 prev.append(val^2)
10:            else
11:                prev.append(prev[|prev|]^2)
12:            end if
13:            mul ← prev[|prev|]
14:        else
15:            mul ← val^i
16:            if poli[i] ≠ 0 then
17:                r ← r + (poli[i] * mul)
18:            end if
19:        end if
20:    end for
21:    return r
22: end procedure
```