

Pontificia Universidad Javeriana



ANÁLISIS NUMÉRICO

TALLER 1

Autor:

John Stiven, Jimmy Alejandro, Pablo Veintemilla, Camilo Maldonado

1 de Octubre

TALLER 1

EDDY HERRERA

RESUMEN. En este documento se presenta el desarrollo del taller 1. Se compone de 4 puntos, Numero de Operaciones, Raíces de una Ecuación, Convergencia de Metodos Iterativos y Convergencia Acelerada.

Parte 1. Numero de Operaciones

1. Utilice el metodo de Horner para evaluar de manera óptima las derivadas de un polinomio en un valor x_0 e implemente en R o Python

implementación en Python

Se implemento una sucesión de todas las derivadas del polinomio 'p' hasta que llegue al cero que significa la derivada de una función constante o en otras palabras 'un polinomio de grado 0'. Esta implementación se puede realizar para un polinomio cualquiera representado por la lista 'p' y se evalúa dado el valor x_0 de la función `main()` utilizando el metodo de horner definido en la función `Pol()`, el tamaño menos uno de la lista determina el grado del polinomio.

```
# Evaluar un polinomio por el metodo de Horner
def Pol(p,x):
    r = 0
    for v in p:
        r = r*x + v
    return r
# Derivada de un polinomio
def dP(p):
    d = []
    n = len(p) - 1
    for i in range(len(p) - 1):
        d.append((n-i)*p[i])
    return d
# Funcion Main
def main():
    x0 = 1
    # p = x^4+2x^3+3x^2+4x+5
    p = [1,2,3,4,5]
    #####
```

```


print("Derivadas de P(x0) evaluadas en x0 = ",x0)
#####
q = p
while len(q) > 0:
    q = dP(q)
    print(len(p) - len(q)," P(x0) = {:.32f}".format(Pol(q,x0)))
main()

```

Resultados: Se evalua el polinomio $x^4 + 2x^3 + 3x^2 + 4x + 5$ en el valor $x_0 = 1.00000000001$ y los resultados obtenidos fueron:

Primera derivada:

En código: $P(x_0) = 20.00000000029999824846527189947665$ Usando la herramienta Wolfram Alpha:



WolframAlpha[®] computational intelligence.

(d/dx)(x^4+2x^3+3x^2+4x+5) where x = 1.00000000001

Extended Keyboard Upload Examples Random

Input interpretation:

$$\frac{\partial (x^4 + 2x^3 + 3x^2 + 4x + 5)}{\partial x} \text{ where } x = 1.00000000001$$

Result:

20.00000000030000

Plot:

Segunda Derivada:

En código: $P(x_0) = 30.00000000036000002978653355967253$ Usando la herramienta Wolfram Alpha:



$(d^2/dx^2)(x^4+2x^3+3x^2+4x+5)$ where $x = 1.00000000001$

Extended Keyboard Upload Examples Random

Input interpretation:

$$\frac{\partial^2(x^4 + 2x^3 + 3x^2 + 4x + 5)}{\partial x^2} \text{ where } x = 1.00000000001$$

Result:

30.00000000036000

Tercera derivada:

En código: $P(x_0) = 36.00000000024000001985768903978169$ Usando la herramienta Wolfram Alpha:



$(d^3/dx^3)(x^4+2x^3+3x^2+4x+5)$ where $x = 1.00000000001$

Extended Keyboard Upload Examples Random

Input interpretation:

$$\frac{\partial^3(x^4 + 2x^3 + 3x^2 + 4x + 5)}{\partial x^3} \text{ where } x = 1.00000000001$$

Result:

36.00000000024000

Cuarta derivada:

Nota: Se sabe que la derivada de x^n es $nx^{(n-1)}$, si derivamos esto de nuevo se obtiene $n(n-1)x^{(n-2)}$, al realizar la derivada 'n' veces sobre el polinomio entonces se tiene el valor $n(n-1)(n-2) \dots 2$ entonces se puede concluir que la derivada n-esima de un polinomio de grado n es n!.

Aplicado a este caso particular ya que el polinomio es de grado 4 al realizar la cuarta derivada se debería obtener $4! = 4*3*2 = 24$

En código: $P(x_0) = 24.000000000000000000000000000000$ Usando la herramienta Wolfram Alpha:



Input: $(d^4/dx^4)(x^4+2x^3+3x^2+4x+5)$ where $x = 1.00000000001$

Extended Keyboard Upload Examples Random

Input interpretation:


$$\frac{\partial^4 (x^4 + 2x^3 + 3x^2 + 4x + 5)}{\partial x^4} \text{ where } x = 1.00000000001$$

Result:

24

Quinta derivada: Ya que al derivar n veces el polinomio de grado n se obtiene $n!$ y este valor es un numero natural, al intentar derivar este valor aplica que la derivada de una constante es igual a cero.

En código: $P(x_0) = 0.0000000000000000000000000000000$ Usando la herramienta Wolfram Alpha:



Input: $(d^5/dx^5)(x^4+2x^3+3x^2+4x+5)$ where $x = 1.00000000001$

Extended Keyboard Upload Examples Random

Input interpretation:

$$\frac{\partial^5 (x^4 + 2x^3 + 3x^2 + 4x + 5)}{\partial x^5} \text{ where } x = 1.00000000001$$

Result:

0

Download Page POWERED BY THE WOLFRAM LANGUAGE

2. Evaluar en $x = 1.000000000001$ con $P(x) = 1 + x + x^2 + \dots + x^{50}$ la primera y la tercera derivada y encuentre el error de cálculo al compáralo con las derivadas de la expresión equivalente $Q(x) = \frac{(x^{51}-1)}{(x-1)}$ **Implementación en Python:** Usando el método `Pol()` para evaluar polinomios y `d P()` para hallar la derivada de un polinomio se construyó la solución a este punto

```
import sympy as sp
# Evaluar un polinomio por el metodo de Horner
def Pol(p, x):
    r = 0
    for v in p:
        r = r*x + v
    return r
# Derivada de un polinomio
def dP(p):
    d = []
    n = len(p) - 1
    for i in range(len(p) - 1):
        d.append((n-i)*p[i])
    return d
# Funcion Main

def main():
    x0 = 1 + 1e-11
    p = [1] * 51
    # Segunda Funcion #####
    x = (sp.var('x'))
    Q = ((x**51 - 1)/(x - 1))
    #####
    print("\nValor de P(x0) con el metodo de Horner = {:.32f}".format(Pol(p, x0)))
    print("Valor por funcion equivalente: {:.32f}".format(Q.evalf(subs={x: x0})))
    error = abs(Pol(p,x0) - Q.evalf(subs={x: x0}))
    print("Error: {:.32f}".format(error))
    #####
    d = dP(p)
    dQ = sp.diff(Q)
    print("\nValor de P'(x) con el metodo de Horner = {:.32f}".format(Pol(d, x0)))
    print("Valor por funcion quivalente: {:.32f}".format(dQ.evalf(subs={x: x0})))
    error = abs(Pol(d,x0) - dQ.evalf(subs={x: x0}))
    print("Error: {:.32f}".format(error))
    #####
    d = dP(dP(d))
    d3Q = sp.diff(sp.diff(dQ))
    print("\nValor de P'''(x) con el metodo de Horner = {:.32f}".format(Pol(d, x0)))
    print("Valor por funcion equivalente: {:.32f}".format(d3Q.evalf(subs={x: x0})))
    error = abs(Pol(d,x0) - d3Q.evalf(subs={x: x0}))
    print("Error: {:.32f}".format(error))
main()
```

y se obtuvieron los siguientes resultados:

```

Valor de P(x0) con el metodo de Horner = 51.00000001275002148304338334128261
Valor por funcion equivalente:          51.00000001275000000000000000000000
Error:  2.13162820728030e-14

Valor de P'(x) con el metodo de Horner = 1275.00000041649991544545628130435944
Valor por funcion quivalente:          1275.00000041650000000000000000000000
Error:  0

Valor de P'''(x) con el metodo de Horner = 1499400.00056377425789833068847656250000
Valor por funcion equivalente:          1499400.00056377000000000000000000000000
Error:  2.32830643653870e-10

```

Parte 2. Raíces de una Ecuación

Ejercicios

1. Implemente en R o Python un algoritmo que le permita sumar unicamente los elementos de la sub matriz 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 gráfica que muestre su orden de convergencia.

Solución:

Se ha realizado el ejercicio con 20 matrices generadas automaticamente por la herramienta random.randn() que ofrece python en la libreria numpy.

El codigo es el siguiente:

```

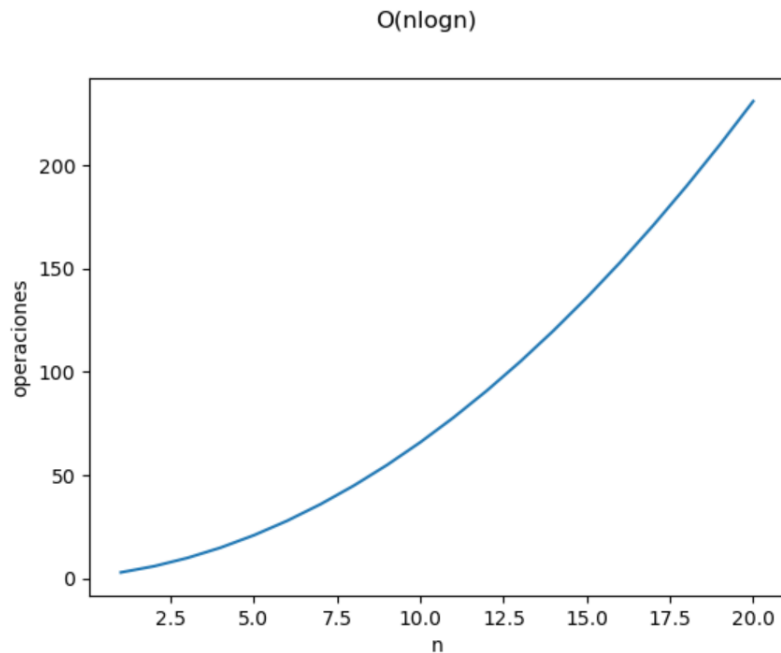
import numpy as np
import matplotlib.pyplot as plt
count = 0
bs = 0
bi = []
bc = 0
bci = []
print("para inferior presione i, para superior s")
option = input()
if(option == 's'):
    for n in range (2, 22):
        A = np.random.randn(n,n)
        for i in range(n):
            for j in range(n):
                if(j>=i):
                    count = count + A[i][j]
                    bs += 1
        print(bs, n)
        print('Suma de matriz ', count)

```

```

        bi.append(bs)
        bs = 0
        print()
    else:
        for n in range(2, 22):
            A = np.random.randn(n,n)
            for i in range(n):
                for j in range(n):
                    if(j<=i):
                        count = count + A[i][j]
                        bs += 1
            print(bs, n)
            print('Suma de matriz ', count)
            bi.append(bs)
            bs = 0
            print()
x = np.linspace(1, 20, num=20)
y = bi
plt.suptitle('O(2*n^2)')
plt.xlabel('n')
plt.ylabel('operaciones')
plt.plot(x,y)
plt.show()

```



Se recorre la matriz y se identifican los elementos que deben ser incluidos en la solución. Para el caso de la matriz superior, se seleccionan los elementos cuando $j \geq i$ y al contrario para la inferior.

Se toman los datos de la cantidad de operaciones realizadas para cada matriz y se le grafica para cada caso desde $n = 2$ hasta $n = 22$, esto para tener suficientes medidas para la grafica de convergencia.

Pruebas:

```
36 operaciones para n = 8
Suma de matriz -10.600404763171964

45 operaciones para n = 9
Suma de matriz -7.182761687198888

55 operaciones para n = 10
Suma de matriz -12.37329318128559

66 operaciones para n = 11
Suma de matriz -6.187559824121052

78 operaciones para n = 12
Suma de matriz -24.17605108745268

91 operaciones para n = 13
Suma de matriz -25.365646457860173

105 operaciones para n = 14
Suma de matriz -37.40538230180919

120 operaciones para n = 15
Suma de matriz -34.04557017933367

136 operaciones para n = 16
Suma de matriz -36.48337623868603

153 operaciones para n = 17
Suma de matriz -26.498094974929913

171 operaciones para n = 18
Suma de matriz 3.6096493065642195

190 operaciones para n = 19
Suma de matriz 12.84193227760997

210 operaciones para n = 20
Suma de matriz 17.487374857264047

231 operaciones para n = 21
Suma de matriz -5.401733505327044
```

2. Implemente en R o Python un algoritmo que le permita sumar los n^2 primeros números naturales al cuadrado. Imprima varias pruebas, para diferentes valores de n y exprese $f(n)$ en notación $O()$ con una gráfica que muestre su orden de convergencia.

Solución:

Se ha realizado el ejercicio con 20 valores de n

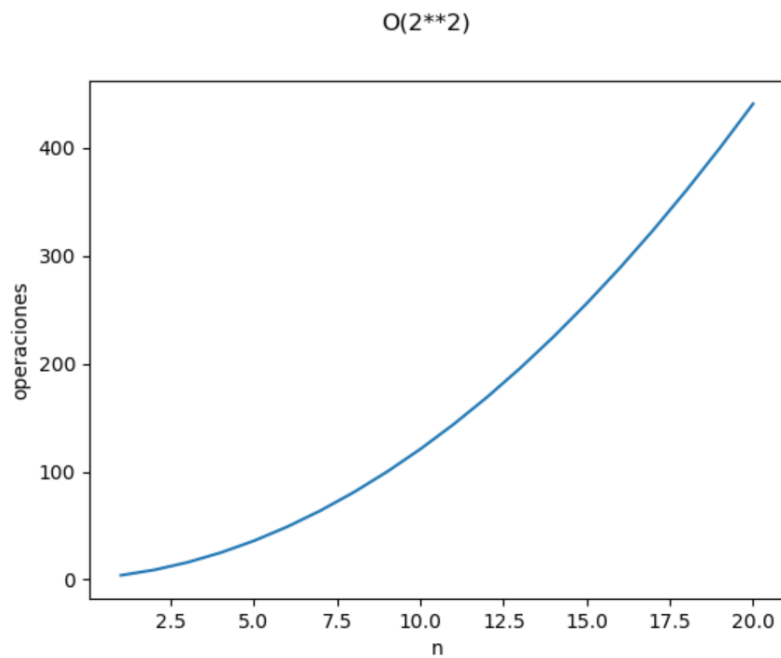
El código es el siguiente:

```
import numpy as np
import matplotlib.pyplot as plt

acum = 0
cont = 0
conti = []
for n in range(2, 22):
    for i in range(n**2):
        acum += i**2
        cont += 1
    print(acum)
    conti.append(cont)
    cont = 0

x = np.linspace(1, 20, num=20)
y = conti
plt.suptitle('O(2**2)')
plt.xlabel('n')
plt.ylabel('operaciones')
plt.plot(x,y)
plt.show()
```

Convergencia:



Se recorren los primeros n números

Pruebas:

```

2  =>  4
14
3  =>  9
218
4  => 16
1458
5  => 25
6358
6  => 36
21268
7  => 49
59292
8  => 64
144636
9  => 81
318516
10 => 100
646866
11 => 121
1230086
12 => 144
2215070
13 => 169
3809754
14 => 196
6300424
15 => 225
10072024
16 => 256
15631704
17 => 289
23635848
18 => 324
34920822
19 => 361
50537682
20 => 400
71791082
21 => 441
100282622

```

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 solución de ecuación no lineal, encuentre la altura máxima que alcanza el cohete.

Solución:

Para la solucion de este ejercicio se tomó la derivada de la función de trayectoria y se aplicaron los metodos de brent y de bisección para encontrar el punto de corte (los ceros) que es, en la derivada la que nos da el t que da la funcion de trayectoria el valor máximo.

El cohete volará a 878.4807692307693 m con brent
 El cohete volará a 878.4807692307692 m con biseccion

Parte 3. Convergencia de Métodos Iterativos

Ejercicios:

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

1. Sean $f(x) = \ln(x+2)$ y $g(x) = \sin(x)$ dos funciones de valor real.
 - a) Utilice la siguiente formula recursiva con $E = 10^{-16}$ para determinar aproximadamente el punto de intersección.:

$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-1} - x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

Solución:

Utilizando el método correspondiente y ejecutándolo en R, tenemos lo siguiente:

```
Iteracion # 0 valor: 1.485705674304172330124629297642968595028
Iteracion # 1 valor: 2.281868404290545629130799121427005605382
Iteracion # 2 valor: 1.033758274851175602209842275494552818148
Iteracion # 3 valor: 0.3327769268984742979384193665225429878688
Iteracion # 4 valor: 1.684749959000812606313610333626902838466
Iteracion # 5 valor: 3.687792362259584491398747186797415337726
Iteracion # 6 valor: 1.365131463956961741719848710580549840702
Iteracion # 7 valor: 1.095877755964146240956266162463137950286
Iteracion # 8 valor: 11.55096516927093568869389841568875512550
Iteracion # 9 valor: 0.3130932835731341115228417454634889579091
Iteracion # 10 valor: -1.725012625558363384871162585962861512630
Iteracion # 11 valor: -0.9843379538088663007746426436591351563101
Iteracion # 12 valor: -1.530154526419864517987118420119841454178
Iteracion # 13 valor: -1.750262550337807081570965987356829560289
Iteracion # 14 valor: -1.613073205464596593422828607317497638237
Iteracion # 15 valor: -1.628090593080915489921339576697779076026
Iteracion # 16 valor: -1.631536274652495796394430529705014342546
Iteracion # 17 valor: -1.631443128818894017866118815909000747467
Iteracion # 18 valor: -1.631443596903503413699695895188591899361
Iteracion # 19 valor: -1.631443596968884869020385776816715516096
Iteracion # 20 valor: -1.631443596968884822896061383153441574973
```

Después de 20 iteraciones, el valor de la intersección es de

$x = -1,631443596968884822896061383153441574973$

con un error de: $-7,680^{-17}$.

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

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Solución:

En este punto, la ecuación utilizada en punto 1a, fue modificada para que el método en R del punto anterior también sirviera para el punto 1b. Para esto, la ecuación no empieza desde n-2 sino en n-1, es por esto que x_n se transforma en x_{n+1} y a todas las x_n de la ecuación se le suma 1 para poder obtener una equivalencia entre la ecuación del punto 1a con la ecuación del punto 1b. Ya teniendo esto en claro se procedió a ejecutar el código en R y obtener el valor aproximado de la intersección el cual es de:

[illegible]

Con los resultados obtenidos, obtenemos un error de 0 para $f(1)$ y un error de $e^{2.765162956017394436652935071618337150486e21}$ para $f(2)$.

Ejercicio: Sea $f(x) = e^x x^2$

a) Demuestre que Tiene un cero de multiplicidad 2 en $x = 0$

$$f(x) = (x - 0)^2 * q(x)$$

$$e^x - x - 1 = x^2 * q(x)$$

$$\frac{e^x - x - 1}{x^2} = q(x)$$

Reemplazando $q(x)$ en la ecuación obtenemos:

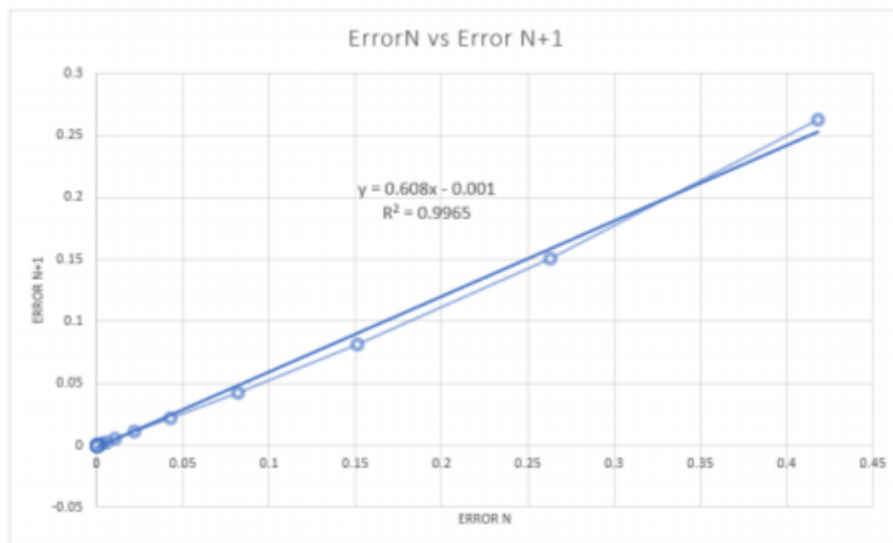
$$f(x) = x^2 * \frac{e^x - x - 1}{x^2}$$

Con esta ecuación se puede afirmar que $x=0$ satisface la raíz de multiplicidad 2.

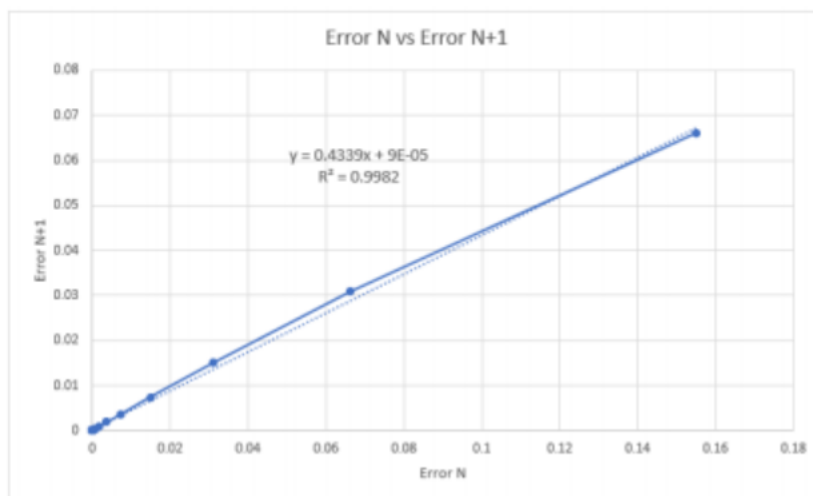
b) Utilizando el metodo de Newton con $p_0 = 1$ verifique que converge a cero pero no de forma cuadratica Con los resultados obtenidos en R se realizó la gráfica de la comparación de los errores:

1	0.41802329313067355
2	0.26292166595850802
3	0.15105886802504795
4	0.081647299137989113
5	0.042553170074067288
6	0.021738018307945847
7	0.010988297888028936
8	0.0055244828148080707
9	0.0027698901687939711
10	0.0013868655217479909
11	0.00069391390640960514
12	0.00034707736931453078
13	0.00017356880506783995
14	8.6791935023070596e-05
15	4.3397849540299617e-05
16	2.1699397179760574e-05
17	1.0849822556837258e-05
18	5.4249347556939693e-06
19	2.7124805213663479e-06
20	1.3561828300359917e-06
21	6.7810621549089107e-07
22	3.3919774053129604e-07
23	1.6965205894818476e-07
24	8.5233729703832846e-08
25	4.224419676155565e-08
26	2.1220239416608511e-08
27	0

- c) Utilizando el metodo de Newton generalizado, mejora la tasa de rendimiento? explique su respuesta



Método de Newton: Error n vs Error n+1



Método Newton generalizado: Error n vs Error n+1

En el método de Newton se obtuvieron 52 iteraciones y el resultado fue de $6.31549252804435653079117176862e16$. Por otro lado, el método de Newton generalizado arrojó 50 iteraciones y un resultado de $8.26470834189378609696212648726e16$. En las gráficas se puede observar que ambas tienen una línea de tendencia lineal y aunque el resultado obtenido sea parecido, se puede decir que el método de Newton tiene mejor rendimiento ya que realiza más iteraciones y el resultado posee un error mayor al error del valor teórico.

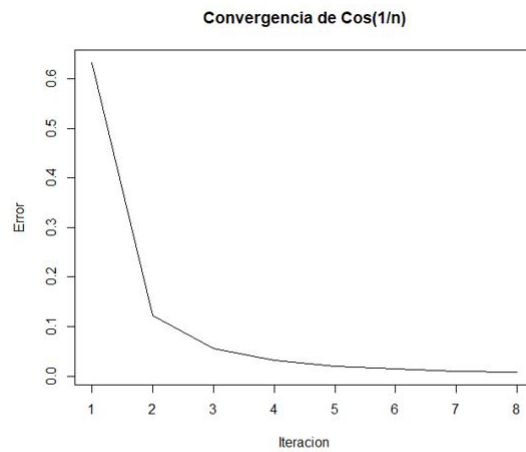
Parte 4. Convergencia Acelerada

Dada la sucesión

1. Verifique el tipo de convergencia en $x = 1$ independientemente de origen

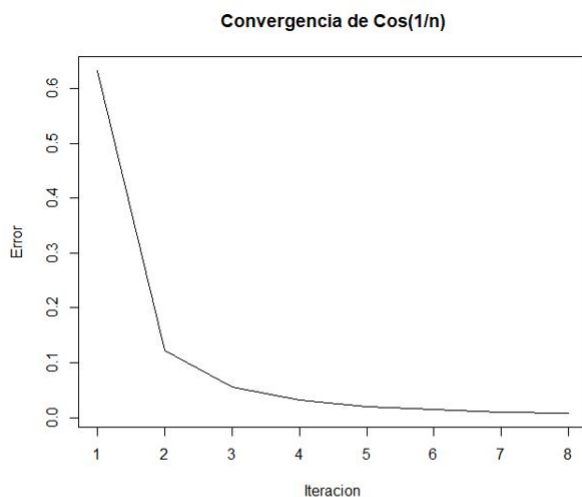
Solucion:

La grafica tiene un comporta exponencial negativo esto lo podemos apreciar en las siguientes graficas.



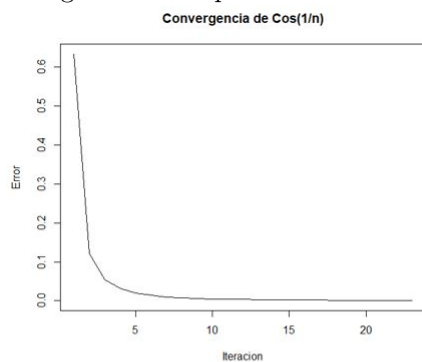
```
sumatoria(0.7,1.0,1e-2)
iteracion  x      f.x. Sumatoria      error
1 0.7 0.5403023 0.3664452 0.633554781
2 1.4 0.8775826 1.2440278 0.122417438
3 2.1 0.9449569 2.1889847 0.055043054
4 2.8 0.9689124 3.1578971 0.031087578
5 3.5 0.9800666 4.1379637 0.019933422
6 4.2 0.9861432 5.1241070 0.013856768
7 4.9 0.9898133 6.1139202 0.010186740
8 5.6 0.9921977 7.1061179 0.007802333
```

Esta grafica corresponde a un valor de $x_0 = 0.7$ y un error de $E = 10^{-2}$



```
Sumatoria(0.000002,1.0,1e-2)
iteracion  x      f.x. Sumatoria      error
1 2.0e-06 0.5403023 0.3664452 0.633554781
2 4.0e-06 0.8775826 1.2440278 0.122417438
3 6.0e-06 0.9449569 2.1889847 0.055043054
4 8.0e-06 0.9689124 3.1578971 0.031087578
5 1.0e-05 0.9800666 4.1379637 0.019933422
6 1.2e-05 0.9861432 5.1241070 0.013856768
7 1.4e-05 0.9898133 6.1139202 0.010186740
8 1.6e-05 0.9921977 7.1061179 0.007802333
```

Esta grafica corresponde a un valor de $x_0 = 0.000002$ y un error de $E = 10^{-2}$



```
Sumatoria(0.3,1.0,1e-3)
iteracion  x      f.x. Sumatoria      error
1 0.3 0.5403023 0.3664452 0.6335547812
2 0.6 0.8775826 1.2440278 0.1224174381
3 0.9 0.9449569 2.1889847 0.0550430537
4 1.2 0.9689124 3.1578971 0.0310875783
5 1.5 0.9800666 4.1379637 0.0199334222
6 1.8 0.9861432 5.1241070 0.0138567684
7 2.1 0.9898133 6.1139202 0.0101867396
8 2.4 0.9921977 7.1061179 0.0078023328
9 2.7 0.9938335 8.0999514 0.0061664915
10 3.0 0.9950042 9.0949556 0.0049958347
11 3.3 0.9958706 10.0908262 0.0041293863
12 3.6 0.9965298 11.0873560 0.0034702133
13 3.9 0.9970429 12.0843988 0.0029571213
14 4.2 0.9974501 13.0818489 0.0025499360
15 4.5 0.9977786 14.0796275 0.0022213993
16 4.8 0.9980475 15.0776750 0.0019524893
17 5.1 0.9982704 16.0759454 0.0017296050
18 5.4 0.9984572 17.0744026 0.0015428130
19 5.7 0.9986153 18.0730179 0.0013847219
20 6.0 0.9987503 19.0717681 0.0012497396
21 6.3 0.9988664 20.0706346 0.0011335726
22 6.6 0.9989671 21.0696017 0.0010328800
23 6.9 0.9990550 22.0686567 0.0009450307
```

Esta grafica corresponde a un valor de $x_0 = 0.3$ y un error de $E = 10^{-3}$

2. Compare los primeros terminos con la sucesion A_n

Como veremos en las siguientes tablas se puede apreciar que en el caso sumatoria esta inicia con un error bajo y por ende se acerca a el límite especificado, pero así mismo se demora varias iteraciones para ir afinando el error. Este método funciona muy bien cuando se busca un error alto. Por otro lado, para Eitken inicia con un gran error, pero lo va reduciendo con rapidez esto nos lleva a afirma que, aunque para errores grandes no es mejor que el otro método, cuando se necesita un error mas bajo es la mejor opción. Esta mejora que produce Eitken se aprecia mejor con errores mayores a $E = 10^{-2}$.

Aitken

```
it 1 : solucion 25.787050160004227 error: 25.150430160004227
it 2 : solucion -0.008262554266931232 error: 25.79531271427116
it 3 : solucion 13.05265520816747 error: 13.060917762434402
it 4 : solucion 8.662430765285851 error: 4.390224442881619
it 5 : solucion 9.76688974178791 error: 1.104458976502059
```

] "Sumatoria"

iteracion	x	f.x.	Sumatoria	error
1	0.7	0.5403023	0.3664452	0.633554781
2	1.4	0.8775826	1.2440278	0.122417438
3	2.1	0.9449569	2.1889847	0.055043054
4	2.8	0.9689124	3.1578971	0.031087578

3.