Pontificia Universidad Católica de Valparaíso
Instituto de Física

Tarea 1: Errores y representación de datos

Nicolás Sepúlveda Quiroz [1]
Pontificia Universidad Católica de Valparaíso

Resumen

El análisis de los errores absolutos y relativos asociados a los cálculos numéricos es esencial para entender en qué medida las soluciones entregadas por los computadores son correctos. En este informe se presentan cuatro problemáticas que utilizan métodos iterativos y se analizan los errores asociados y formas de minimizarlos.

18 de diciembre, 2019

[1] nicosepulveda4673@gmail.com

Índice

1. Introducción

El estudio de problemas mediante cálculos en el computador hace uso de algoritmos, secuencias de pasos predeterminadas para llegar a una solución. Para realizarlos, se emplean los métodos numéricos, iterativos o directos. Los métodos iterativos, como cualquier cómputo, tienen asociados errores de redondeo, truncamiento, o eliminación de cifras significativas [1]. El análisis de cómo estos errores se desarrollan es de gran importancia para la resolución correcta de problemas matemáticos para comprender fenómenos naturales o sociales. En este informe se presentan cuatro problemáticas que requieren de la utilización de métodos numéricos, y se analiza cómo los errores afectan las soluciones. En ellas se ven involucradas relaciones de recurrencia, para el calculo de sumatorias o la aproximación de funciones.

2. Cómputo numérico de la constante de Euler y metodo de acelerción

La constante de Euler es una constante matemática que tiene principal recurrencia en teoría de números, con relaciones a la función Zeta de Riemman, la función Gamma, y en el area de la física, asociada a casos límites en soluciones de agujeros negros y amplitudes de scattering [2].Esta se define como:

$$\gamma = \lim_{n\to\infty} \left[\sum_{k=0}^{n} \frac{1}{k} - log(n) \right]. \tag{2.1}$$

Al calcular numéricamente esta constante, se usan métodos iterativos para realizar la suma, lo que naturalmente introduce limitaciones asociadas al mayor número que una suma puede alcanzar, al menor

número representable cuando se suman términos muy pequeños, y a los errores numéricos como el de redondeo. Es útil entonces buscar métodos de aceleración que permitan, desde la matemática, realizar los cálculos en menor número de iteraciones; por ejemplo, a través de una expresión equivalente a 2.1 pero que converja al valor en menos pasos.

Se ha realizado una estimación del mayor valor de la suma que el computador calcularía en una semana. Para ello, se ha estimado el tiempo que tarda una iteración del ciclo mediante un promedio simple de lo tardado por las primeras 5000 iteraciones, con pasos de 100 iteraciones. La función utilizada en Python es $perf\_counter()$, como se puede observar en el código 1. El valor entregado para el promedio es $7{,}412000000073249x10^{-7}$, con lo cual se estima que en una semana se realizarían 10000000 iteraciones. Usando el valor aproximado de la constante de Euler, y tomando n el número de iteraciones estimado, se encuentra que el valor de la suma s en una semana sería:

$$\gamma = 0{,}577215664901532 = s - log(n) = s - 16{,}1180; s \approx 15{,}54088$$

Para la estimación numérica de la constante de Euler, se utilizó el código 1 del cual se obtiene la figura 1 que muestra el valor de la constante γ para 5000 iteraciones en incrementos de n iguales a 100.

Se puede mostrar que una modificación en la expresión 2.1 mantiene el valor de γ:

$$\gamma = \lim_{n\to\infty}\left[\sum_{k=0}^{n}\frac{1}{k} - log(n)\right] = \lim_{n\to\infty}\left[\sum_{k=0}^{n}\frac{1}{k} - log(n+\frac{1}{2})\right], \qquad (2.2)$$

pues se verifica que [3]:

$$lim_{n\to\infty}\left[\sum_{k=0}^{n}\frac{1}{k} - log(n+\frac{1}{2})\right]$$

$$= lim_{n\to\infty}\left[\sum_{k=0}^{n}\frac{1}{k} - log(n) - log(n+\frac{1}{2}) + log(n)\right] \qquad (2.3)$$

y luego

$$\lim_{n\to\infty} log(\frac{1}{1+\frac{1}{2n}}) = 0. \qquad (2.4)$$

En la figura 2 se comparan los ratios de convergencia de las ecuaciones 2.1 y 2.2, y se observa cómo en la segunda la suma converge más rápido.

En [3] se muestra cómo las modificaciones de la forma $log(n+\alpha)$ con $\alpha > -n$ cambian la calidad de la aproximación para valores finitos de

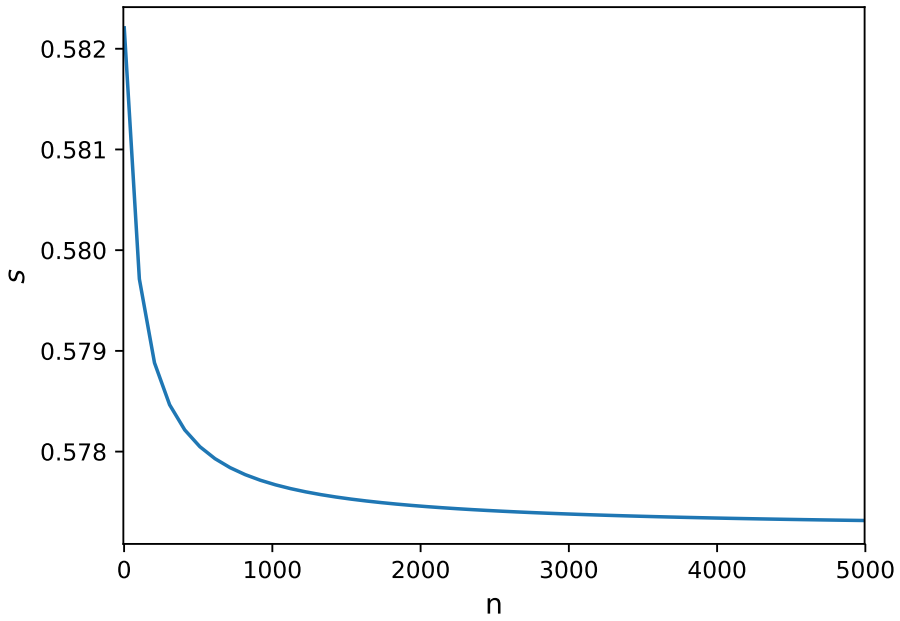

FIGURA 1. Valor de la constante de Euler (s) en función del número de términos n en la sumatoria de .

n, que es lo que se tiene en un análisis numérico. Mediante la definición de una variable de error $\epsilon(\alpha)$ como la diferencia entre la ecuación modificada y el valor dado por la definición, y la diferenciación de esta con respecto a α, se encuentra que el valor que minimiza este error es $\sim \frac{1}{2}$.

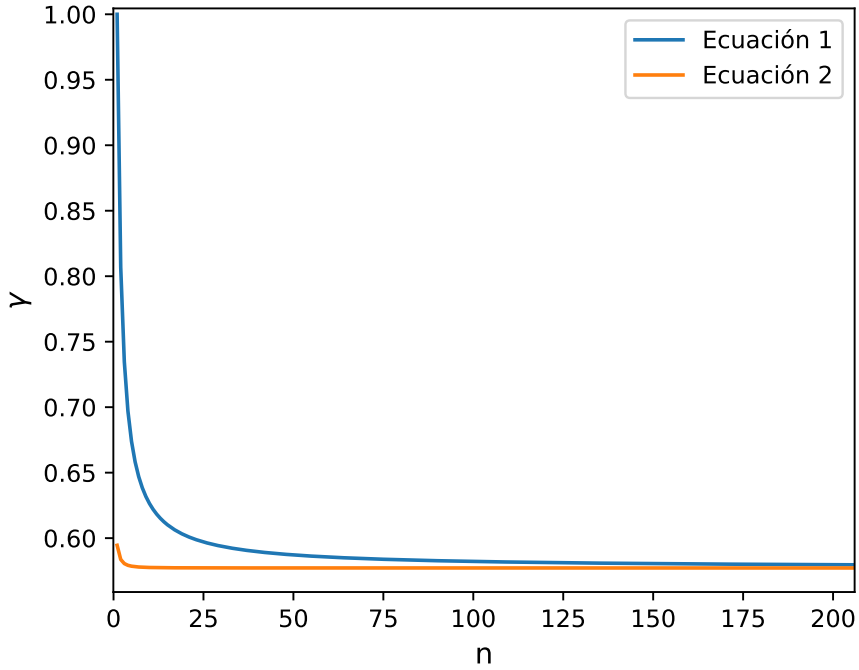

FIGURA 2. Comparación del valor de la constante de Euler en función del número de términos n en la sumatoria de .

3. SOBRE LAS RELACIONES DE RECURRENCIA.

Una forma de resolver problemas en métodos numéricos es el empleo de algoritmos iterativos con el fin de aproximarse a la solución. Sin embargo, las operaciones hechas sobre el valor del cómputo anterior son fuentes sistemáticas de errores que pueden ser significativos con respecto a la solución encontrada. Algoritmos que son estables se aproximan en cada paso a la solución exacta del problema, y la construcción de estos es de suma importancia para realizar cálculos correctos. En esta sección se analizará una relación de recurrencia y los errores numéricos asociados.

Se tiene la expresión:

$$p_n = \int_0^1 dx x^n e^x, \tag{3.1}$$

y se comprueba que los números p_n satisfacen $1 = p_1 > p_2 > p_3 > ... > 0$, pues escribiendo la integral como una suma de Riemman, se

puede mostrar que $p_n - p_{n+1} > 0$:

$$\lim_{n->\infty} \sum_{i=1}^{n} \frac{(i/n)^m e^{i/n}}{n} - \frac{(i/n)^m e^{i/m}}{n} > 0, \tag{3.2}$$

y agrupando se encuentra

$$\lim_{n->\infty} \sum_{i=1}^{n} \frac{e^{i/n}}{n^{m+2}} i^m (n-i) > 0 \tag{3.3}$$

que es verdadero ya que todos los términos son mayores que cero en el límite $n- > \infty$. Para encontrar la relación de recurrencia asociada se escribe el término para $n+1$:

$$p_{n+1} = \int_0^1 dx x^{n+1} e^x. \tag{3.4}$$

Integrando por partes con los cambios de variable

$$u = x^{n+1};\ du = (n+1)x^n dx;\ dv = e^x dx;\ v = e^x, \tag{3.5}$$

se encuentra

$$p_{n+1} = -\int_0^1 e^x (n+1) x^n dx + x^{n+1} e^x \Big|_0^1 = e - (n+1) p_n \tag{3.6}$$

En la columna a de la tabla 3 se observa como las igualdades anteriormente demostradas no se cumplen.

Como es mostrado en [4, 5], este algoritmo es numéricamente inestable cuando es aplicado hacia adelante, pues los errores asociados al redondeo se vuelven cada vez mayores a medida que n aumenta. El error absoluto asociado al resultado numérico p'_n es descrito por

$$p'_n = p_n + \epsilon_n, \tag{3.7}$$

y expresado explícitamente en función de n:

$$\epsilon_n = p'_n - p_n = (e - n p'_{n-1}) - (e - n p_{n-1}) = -n(p_{n-1} - p'_{n-1}) = -n\epsilon_{n-1}, \tag{3.8}$$

lo que implica que para la iteración n el error es proporcional a $n!$ veces el error e_0; existe un crecimiento del error que para un valor definido de n en adelante causará errores numéricos.

El efecto del crecimiento de ϵ_n debido a que es múltiplo de ϵ_{n-1} puede ser mitigado realizando la operación inversa, es decir, ejecutando el programa en reversa. De este modo se tiene que la relación de recurrencia para p_{n-1} es

$$p_{n-1} = \frac{e - p_n}{n} \tag{3.9}$$

a)	b)	c)	d)
1.000.E+00	1.000000	1.000000	1.000000
-2.817.E-01	0.718282	0.718282	0.718282
3.845.E+00	0.563436	0.563436	0.563436
-1.651.E+01	0.464536	0.464536	0.464536
1.018.E+02	0.395600	0.395600	0.395600
-7.096.E+02	0.344685	0.344685	0.344685
5.680.E+03	0.305490	0.305490	0.305490
-5.111.E+04	0.274362	0.274362	0.274362
5.111.E+05	0.249028	0.249028	0.249028
-5.623.E+06	0.228002	0.228002	0.228002
6.747.E+07	0.210265	0.210265	0.210265
-8.771.E+08	0.195100	0.195100	0.195100
1.228.E+10	0.181983	0.181983	0.181982
-1.842.E+11	0.170524	0.170524	0.170527
2.947.E+12	0.160426	0.160416	0.160373
-5.010.E+13	0.151461	0.151632	0.152320
9.018.E+14	0.143447	0.140541	0.128845
-1.713.E+16	0.136243	0.188546	0.399072
3.427.E+17	0.129664	-0.864086	-4.864086
-7.197.E+18	0.125000	20.000000	100.000000

FIGURA 3. Tabla 1: Valores de p_n calculados con $a)$ el algoritmo 3.6; $b) - c) - d)$ el algoritmo 3.9 con diferentes p_{20}.

genera un error asociado al resultado numérico $p'_n - 1$:

$$\epsilon_{n-1} = p'_{n-1} - p_{n-1} = \frac{1}{n}((e - p'_n) - (e - p_n)) = -\frac{1}{n}\epsilon_n. \tag{3.10}$$

Por lo tanto, el valor del error absoluto para las iteración asociada a p_1 es una fracción proporcional a $\frac{1}{n!}$ del error inicial de p_n. Los valores de esta solución estable se muestran en la columna $b)$ de la tabla 3.

Cuando se usan valores de p_{20} que son en principio, mayores que 1, se rompen las desigualdades demostradas. No obstante, el algoritmo sigue convergiendo al valor $p_1 = 1$ luego de la desestabilización inducida por el término p_{20}. Como se observa en 3.9, un valor de p_n mayor que e implica $p_{n-1} < 0$ y $p_{n-2} > 0$; y solo cuando la diferencia en el numerador sea positiva, es decir, para un $p(k) < e$ la suma va a converger. En las columnas $c)$ y $d)$ de la tabla 3 se observa que solo es necesario dos iteraciones para la estabilización de la solución y la verificación de

x	exp(x)	exp(x) [1]	exp(x) [2]	exp(x) [3]
1	2.718281828459040	2.718281525573190	2.718281722772260	2.718281828459040
-1	0.367879441171442	0.367879482162589	0.367879455474593	0.367879441171442
0.5	1.648721270700120	1.648721168154760	1.648721371479290	1.648721270700120
-0.123	0.884263662560821	0.884263849834450	0.884263849834450	0.884263662560821
-25.5	8.42E-12	1.94047027052706E-11	8.42345956860573E-12	8.42346375446862E-12
-1776	ERROR	6.31E-55	0	0
3.14159	23.1406312269649	23.1406302657924	23.1406325278203	23.1406312269549

FIGURA 4. Valor de $exp(x)$ según los distintos métodos.

las desigualdades; sin embargo, para valores muy grandes de p_{20} son necesarias más iteraciones, de lo que se desprende que hay un valor máximo de p_{20} para el cual estas se cumplen.

4. ANÁLISIS DE LA FUNCIÓN e^x Y UN MÉTODO DE ACELERACIÓN.

La importancia de la función exponencial radica en su propiedad fundamental: su razón de cambio es proporcional al valor de la función; comportamiento ampliamente observado en la naturaleza, tanto en ciencias exactas como sociales. En este problema se analizan los métodos iterativos para el cálculo de esta y sus errores numéricos.

La descomposición de la función e^x en serie de Taylor es:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \tag{4.1}$$

suma que se ha calculado numéricamente para los términos mayores que $\epsilon = -10^6$. Como se observa en el código 3, para los valores negativos de x se ha utilizado el inverso de la suma. Asimismo, se ha empleado un algoritmo de reducción con el cual se comparan los errores asociados y el tiempo de cómputo, es decir, el número de iteraciones necesarias para calcular el valor de la función.

Este consiste en la expresión

$$e^x = 2^m e^u \tag{4.2}$$

que elimina las potencias de 2 y usa el código del primer método para $|u| < log(2)/2$; el cálculo es mucho mas rápido debido a la cota superior de x en e^x.

En la tabla 4 se muestra la función evaluada numéricamente en distintos valores x para los métodos 1 y 2 en las columnas $exp(x)[1]$ y $exp(x)[2]$ respectivamente. En la tabla 5 se encuentras los errores relativo y absoluto asociados a los valores de la función obtenidos en la calculadora del computador $exp(x)$.

x	Abs(1)	Rel(1)	Abs(2)	Rel(2)	Abs(3)	Rel(3)
1.00E+00	3.03E-07	1.11E-07	1.06E-07	3.89E-08	4.44E-16	1.63E-16
-1.00E+00	4.10E-08	1.11E-07	1.43E-08	3.89E-08	2.78E-16	7.54E-16
5.00E-01	1.03E-07	6.22E-08	1.01E-07	6.11E-08	0.00E+00	0.00E+00
-1.23E-01	1.87E-07	2.12E-07	1.87E-07	2.12E-07	0.00E+00	0.00E+00
-2.55E+01	1.10E-11	1.30E+00	4.60E-16	5.46E-05	4.64E-16	5.51E-05
-2.00E+02	3.26E-32	2.35E+55	7.07E-95	5.11E-08	2.89E-100	2.09E-13
3.14E+00	9.61E-07	4.15E-08	1.30E-06	5.62E-08	1.00E-11	4.32E-13

FIGURA 5. Errores absoluto y relativo asociados a $exp(x)$ según los distintos métodos.

Con el fin de visualizar por qué los errores relativos para los números de valor absoluto grande crece considerablemente, se muestra el gráfico de los términos n-ésimos de la suma en función de n para $x = 1$ y $x = 3,1415$. En la figura 7 se puede observar que a medida que x crece, el valor del término n fluctúa en una función ascendente y luego descendente. Entonces mientras más grande x, más iteraciones son necesarias para que el valor de los términos comience a decrecer, y la cantidad de iteraciones para lograr minimizar el n-ésimo término bajo el valor ϵ es mucho mayor. Esto puede explicar por qué con valores como $x = -25{,}5$ el valor de la suma en el método 1 crezca tanto, y al mismo tiempo que el error relativo asociado al resultado sea grande; pues las iteraciones necesarias para cumplir con el ϵ requerido son mucho mayores que 25. No obstante, el método 2 al reducir el valor a calcular por la exponencial ($x \to u$), es capaz de reducir el número de iteraciones y por ende aproximarse mejor al valor de la calculadora.
Dado lo anterior, se concluye que cuando el exponente es muy grande, los números involucrados en la suma son muy grandes y ocupan las 25 iteraciones sin ser menores que el límite. Si el número es pequeño, el valor ϵ se alcanza rápidamente. Entonces, no hay mayor diferencia entre el método 1 y el 2 cuando se usan números pequeños pero si cuando son muy grandes.

La columna $exp(x)[3]$ de la tabla 4 corresponde a los valores calculador por un tercer método: la parte par de la fracción continuada de Gauss. Se muestran sus errores absoluto y relativo en las últimas dos columnas de la tabla 5 y se observa que es una mejor aproximación que los otros dos métodos dado que los errores relativos asociados son órdenes de magnitud menores.

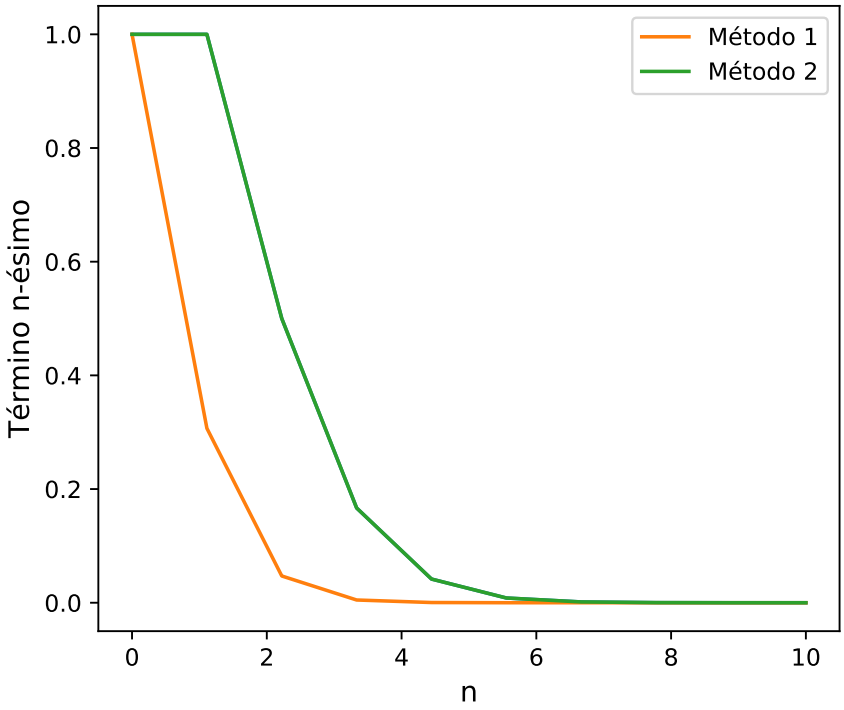

FIGURA 6. Término n-ésimo de la suma con respecto a n; con $x = 1$.

5. Mapa logístico

El mapa logístico es una relación de recurrencia ampliamente usada para el estudio de sistemas caóticos y descripción de procesos naturales, como por ejemplo, un modelo demográfico. Como fue explicado, las relaciones de recurrencia al utilizar algoritmos iterativos llevan asociados errores que pueden afectar la estabilidad de la solución dependiendo de como estos se propagan en los cómputos. En esta sección se analizan las relaciones correspondientes a este sistema.

El mapa logístico es definido por

$$x_{n+1} = rx_n(1 - x_n), \tag{5.1}$$

con $r \in [0, 4]$ y $x_n \in [0, 1]$. En su versión distribuida:

$$x_{n+1} = r(x_n - x_n x_n) \tag{5.2}$$

En la figura 8 se grafican las soluciones x_n para los primeros 100 valores de n, con $x_0 = 0{,}857$ y $r = 1 + \sqrt{8} - 10^{-3}$ En esta se observa

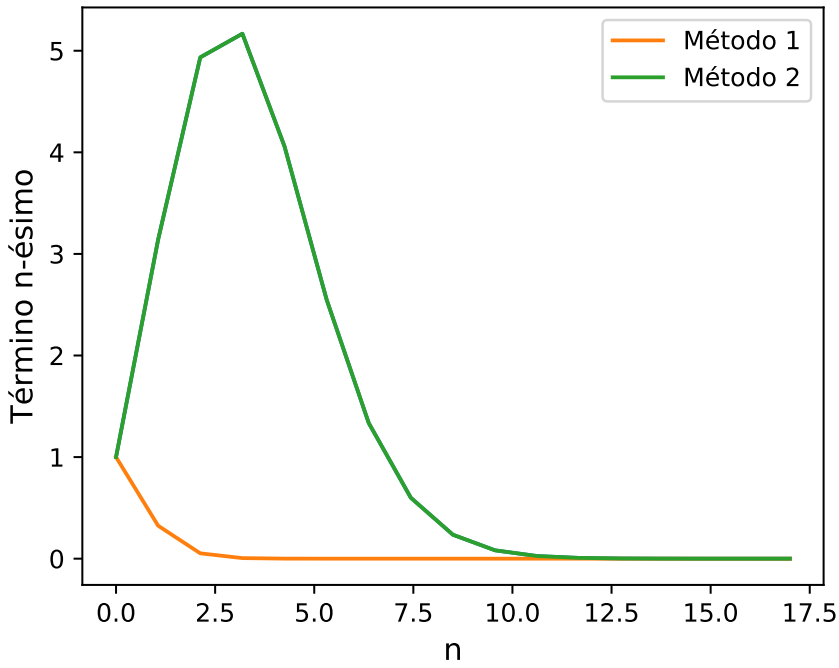

FIGURA 7. Término n-ésimo de la suma con respecto a n; con $x = 3,1415$.

que en las últimas iteraciones las funciones comienzan a diferir, hecho atribuible a errores numéricos asociados al error, pues analíticamente las ecuaciones 5.1 y 5.2 son idénticas. En la figura 9 se muestra la solución con el mismo x_0 pero con $r = 1 + \sqrt{8} + 10^{-3}$; en este caso, las soluciones no difieren para ningún $n < 100$. La diferencia de ambos casos es explicada por una transición de una zona de caos a una estable de las soluciones para el parámetro r, correspondiente al inicio del período 3 del mapa logístico con $r = 1 + \sqrt{8}$ [6]. Así, soluciones con valores de r que están por debajo de este punto de transición presentan un comportamiento caótico, que se traduce en la sensibilidad de estas a pequeñas variaciones de las condiciones iniciales, y en el caso de los cómputos numéricos, a las variaciones inducidas por los errores de redondeo. Por otro lado, las soluciones por encima de este límite son menos afectadas por los errores numéricos.

Con $r = 4$ se tiene desarrollo completo del caos, y para esta condición se tiene solución analítica del problema. En la figura 10 se muestran

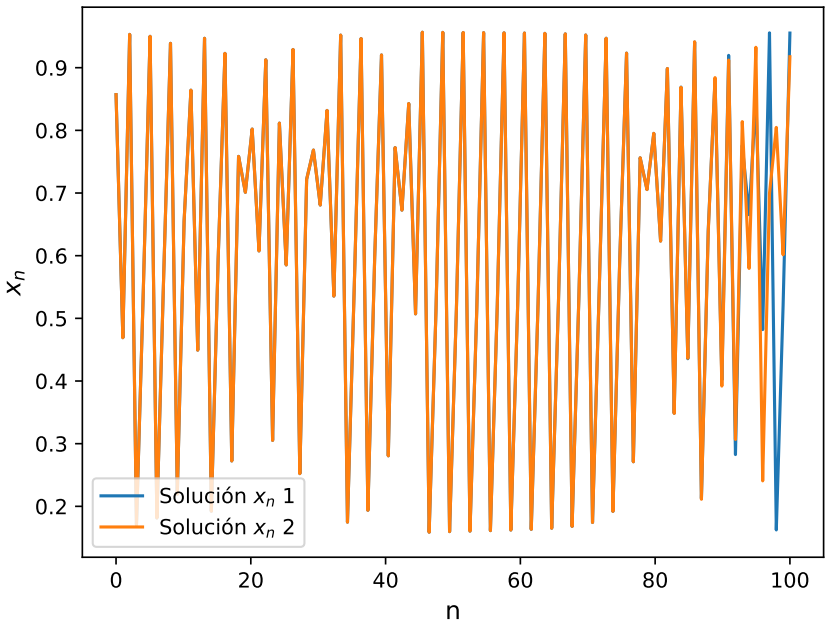

FIGURA 8. Solución x_n vs. n; con $r = 1 + \sqrt{8} - 10^{-3}$

las soluciones según las ecuaciones 5.1, 5.2 y la analítica:

$$x_n = sin^2(2^n arcsin\sqrt{x_0}), \tag{5.3}$$

para las primeras 100 iteraciones. Se observa cómo desde aproximadamente la iteración 50 las soluciones numéricas difieren de la analítica. En [6] comentan cómo esta perdida de precisión puede ser interpretada como un rompimiento de las propiedades asociativas y distributivas del álgebra numérico. Por otro lado, se observa en la figura 11 cómo existen dos picos en los errores relativos de las soluciones numéricas con respecto a la analítica, coincidentes con dos puntos en la figura 10 que son cercanos a cero, los cuales hacen crecer la función de error relativo al estar en el denominador de la expresión. Se observa que esto no ocurre en la zona de las primeras iteraciones, posiblemente debido a que los errores absolutos son cercanos a cero.

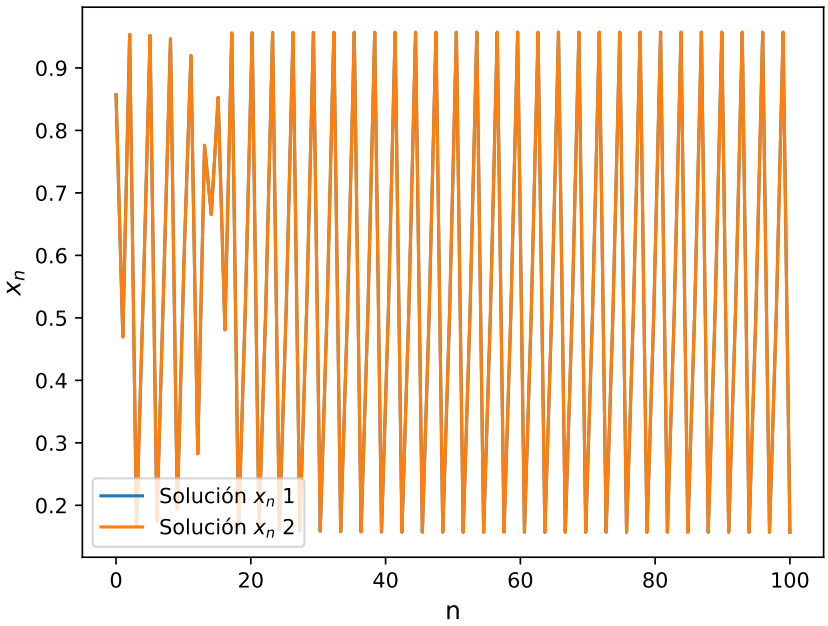

FIGURA 9. Solución x_n vs. n; con $r = 1 + \sqrt{8} + 10^{-3}$

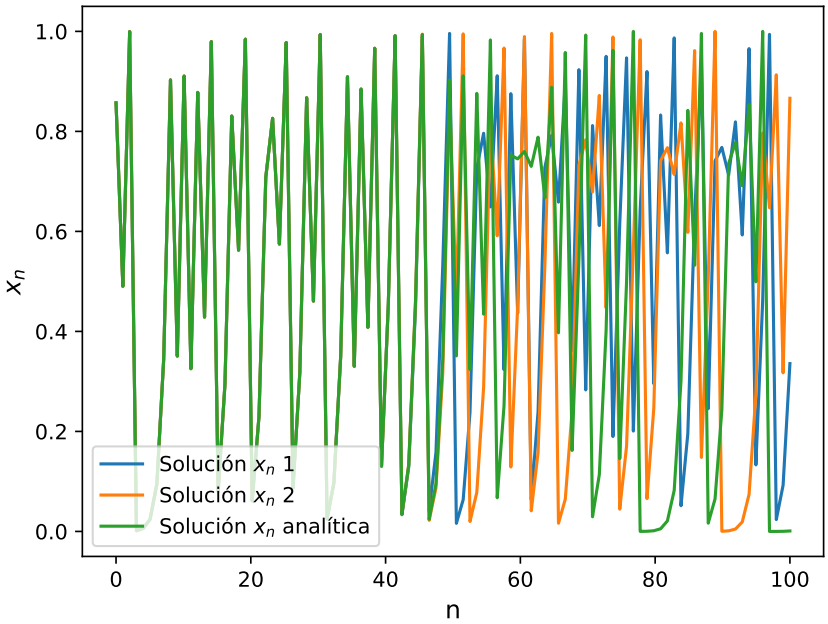

FIGURA 10. Comparación de soluciones x_n según los distintos métodos.

.

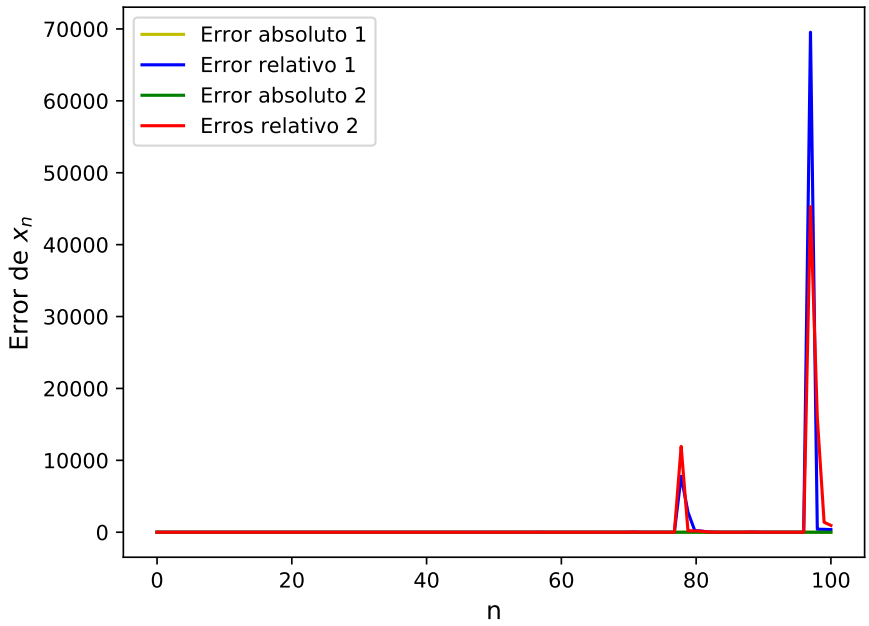

FIGURA 11. Errores absoluto y relativo asociados a los dos algoritmos ejecutados.

.

6. AGRADECIMIENTOS

Agradecimientos a Pablo Orellana por ayudarme a resolver el cálculo de las desigualdades del segundo problema, y a Mackarena Garrido por ayudarme a solucionar muchas dudas de programación.

7. ANEXO

Código 1.

```
import time
import math
from math import log
import matplotlib.pyplot as plt
import numpy as np
from numpy import linspace

from time import perf_counter


s=0
```

```
n=5000
SUMA=[]
TABLA=[]
TIEMPO=[]
ss=[]
for i in range(1,n+1):
    start = perf_counter()
    x=i
    s+=1/x
    stop = perf_counter()
    ss.append(s)
    duration = stop - start

    S=s-log(x)
    SUMA.append(S)
    if i%100==0:
        print('tiempo : '+str(duration)+'en la iteraci n : '+str(i))
        TIEMPO.append(duration)

    if i%100==0:
        TABLA.append(S)




#segnda parte

s=0
n=5000
SUMA2=[]
for i in range(1,n+1):
    x=i
    s+=1/x
    S=s-log(x+1/2)
    #if i%100==0:
    #    print(S)
    SUMA2.append(S)

#print(SUMA)
#print(SUMA2)
#gr fico
X=linspace(1,5000,5000)
t=linspace(1,5000,50)
print(len(X))
print(len(TIEMPO))

#plt.plot(X, ss)
```

```
#plt.show()

plt.show()

plt.plot(t,TABLA)
plt.xlabel('n', fontsize=12)
plt.ylabel('$s$', fontsize=12)
plt.show()

print("El tiempo promedio por iteraci n es :"+str(sum(TIEMPO)/len(TIEMPO)))
plt.plot(t,TIEMPO)
plt.show()

np.savetxt("foo.csv", TABLA, delimiter=",")

plt.plot(X,SUMA, label='Ecuaci n 1')
plt.plot(X,SUMA2, label='Ecuaci n 2')
plt.xlabel('n', fontsize=12)
plt.ylabel('$\\gamma$', fontsize=12)
plt.savefig("grafico1.jpg")
plt.legend()



plt.show()


#tiempo promedio: 7.412000000073249*10^-7
```

Código 2.

```
#Tarea1(2).py
import math
from math import log
import matplotlib.pyplot as plt
import numpy as np
from numpy import linspace


p=[0]*20
#a)----------
p[0]=1
for i in range(1,20):
    print(i)
    p[i]=np.e-((i+1)+1)*p[i-1]
    print(p[i])
p=np.asarray(p)
#b)
pb=[0]*20
```

```
pb[19]=1/8
for i in reversed(range(0,19)):
    print(i)
    pb[i]=(pb[i+1]-np.e)/(-(i+1)-1)
    print(pb[i])
pb1=np.asarray(pb)

pb=[0]*20
pb[19]=20
for i in reversed(range(0,19)):
    print(i)
    pb[i]=(pb[i+1]-np.e)/(-(i+1)-1)
    print(pb[i])
pb2=np.asarray(pb)

pb=[0]*20
pb[19]=10000000000
for i in reversed(range(0,19)):
    print(i)
    pb[i]=(pb[i+1]-np.e)/(-(i+1)-1)
    print(pb[i])
pb3=np.asarray(pb)


print(p)
print(pb1)
print(pb2)
print(pb3)

n=linspace(1,20,20)
#plt.plot(n,p)
plt.plot(n,pb1)
plt.plot(n,pb2)
plt.plot(n,pb3)
np.savetxt("t1-2.txt", np.r_[p,pb1,pb2,pb3], delimiter=';')

plt.show()
```

Código 3.

```
import math
from math import log
import matplotlib.pyplot as plt
import numpy as np
from numpy import linspace, log


x_vec=[1,-1,0.5,-0.123,-25.5,-200,3.14159,-40,0.2]
real=[2.718281828459045,0.367879441171442,1.648721270700128,0.884263662560821,0.00
```

```
numerico1=[]
numerico2=[]
numerico3=[]
terminos1=[]
terminos2=[]


for v in x_vec:
    print('con x =:'+str(v))
    x=v
    suma=0
    L=26


    factorial=-1

    for n in range(0,L-1):
        if n==0:
            factorial=1
        else:

            factorial=factorial*n


        termino=abs(x**n)/(factorial)

        if abs(termino)>10**(-6):
            suma=suma+termino
            print('esta es la iteraci n con n='+str(n))
            print('este es el n- simo termino: '+str(termino))
            print('esta es la suma: '+str(suma))
            if x==1:
                terminos1.append(termino)
        else:
            break
            print('La suma se hizo en '+str(n-1)+' iteraciones.')

    if x>0 or x==0:
        print('La suma e^x(a) es : '+str(suma))
        numerico1.append(suma)
    else:
        print('La suma e^x(a) es : '+str(1/suma))
        numerico1.append(1/suma)
```

```
#-------------------------------------------------------------
#ahora e**x=2**m * e**u
#puedo usar el codigo anterior para calcular e**u
    z=x/log(2)
    if x<0:
        m=int(z-1/2)
    else:
        m=int(z+1/2)
    w=z-m
    u=w*log(2)

    print('u es '+str(u))
#luego corro el mismo c digo de e^x pero con la variable u en vez de x
    suma=0
    L=26
    factorial=1

    for n in range(0,L-1):
        if n==0:
            factorial=1
        else:
            factorial=factorial*n


        termino=abs(u**n)/(factorial)



        if abs(termino)>10**(-6):
            suma=suma+termino
            print('esta es la iteraci n con n='+str(n))
            print('este es el n- simo termino: '+str(termino))
            print('esta es la suma: '+str(suma))
            if x==1:
                terminos2.append(termino)
        else:
            break
            print('La suma se hizo en '+str(n-1)+' iteraciones.')


    if u>0 or u==0:

        suma2=2**m*(suma)
    else:

        suma2=2**m*(1/suma)
```

```
    numerico2.append(suma2)
    print('La suma e^x(a) es : '+str(suma2))



# parte c) ----------------------

#vuelvo a resetear la suma
    suma=0
    s=2+u**2*(2520+28*u**2)/(15120+420*u**2+u**4)
    suma=(s+u)/(s-u)
#print('la suma e^u es :'+str(suma))
    suma3=2**m*suma
    print('La suma e^x(c) es : '+str(suma3))
    numerico3.append(suma3)


numerico1=np.asarray(numerico1)
numerico2=np.asarray(numerico2)
numerico3=np.asarray(numerico3)
#print(numerico1)
valores=np.column_stack((x_vec,real,numerico1,numerico2,numerico3))
#print(valores)


e_a1=abs(real-numerico1)
e_a2=abs(real-numerico2)
e_a3=abs(real-numerico3)

e_r1=e_a1/real
e_r2=e_a2/real
e_r3=e_a3/real

errores=np.column_stack((x_vec,e_a1,e_r1,e_a2,e_r2,e_a3,e_r3))
np.savetxt("valores.txt", valores,delimiter=";")
np.savetxt("errores.txt",errores,delimiter=";")
print('terminos1')
print(terminos1)

X=linspace(0,len(terminos1),len(terminos1))
plt.plot(X,terminos1)

while len(terminos2)<len(terminos1):
        terminos2.append(0)
print('terminos2')
print(terminos2)
plt.plot(X,terminos2,label='M todo 1')
```

```
plt.plot(X,terminos1,label='M todo 2')
plt.xlabel('n',fontsize=12)
plt.ylabel('T rmino n- simo ',fontsize=12)
plt.legend()
plt.show()
```

Código 4.

```
#Tarea1(4).py
import math
from math import log
import matplotlib.pyplot as plt
import numpy as np
from numpy import linspace
#ECUACIN 11----------------
x=[0]*100
r=1+np.sqrt(8)+10**(-3)
#r=4
x[0]=0.857

for i in range(1,100):
    x[i]=r*x[i-1]*(1-x[i-1])



print('Con la ecuaci n (11) : '+str(x))

#ECUACIN 12----------------
x2=[0]*100
x2[0]=0.857
for i in range(1,100):
    x2[i]=r*(x2[i-1]-x2[i-1]*x2[i-1])

print('Con la ecuaci n (12) : '+str(x))
n=np.linspace(0,100,100) #vector con n meros n

xa=[0]*100
xa[0]=0.857
print(xa)
for i in range(1,100):
    xa[i]=(np.sin((2**i)*(np.arcsin(np.sqrt(xa[0])))))**2
print(xa)
x=np.asarray(x)
x2=np.asarray(x2)
xa=np.asarray(xa)
e_a1=xa-x
e_r1=abs(xa-x)/xa
e_a2=xa-x2
e_r2=abs(xa-x2)/xa
```

```
plt.plot(n,x,label='Soluci n $x_n$ 1')
plt.plot(n,x2,label='Soluci n $x_n$ 2')
#plt.plot(n,xa,label='Soluci n $x_n$ anal tica')
plt.xlabel('n', fontsize=12)
plt.ylabel('$x_n$', fontsize=12)
plt.legend(loc=3)
plt.show()


plt.plot(n,e_a1,'y',label='Error absoluto 1')
plt.plot(n,e_r1,'b',label='Error relativo 1')
plt.plot(n,e_a2,'g',label='Error absoluto 2')
plt.plot(n,e_r2,'r',label='Erros relativo 2')
plt.xlabel('n', fontsize=12)
plt.ylabel('Error de $x_n$', fontsize=12)
plt.legend()
plt.show()
```

8. Referencias

1. http://www.ehu.eus/pegonzalez/I.Teleco/Apuntes/tema1.pdf
2. Súkeník M. y Šima J.: THE EULER – MASCHERONI CONSTANT AND ITS APPLICATION IN PHYSICAL RESEARCH
3. Havil, Julian (2003). Gamma: Exploring Euler's Constant.
4. https://scipython.com/book/chapter-9-general-scientific-programming/examples/numerical-stability-of-an-integral-solved-by-recursion/
5. https://www.math.ubc.ca/ israel/m210/lesson32.pdf
6. Oteo J. y Ros J.: Double precision errors in the logistic map: Statistical study and dynamical interpretation