Reto 1: Sistemas no lineales

D. R. Ramírez, C. Castrillón, J. M. Torres

Marzo de 2021

1 Introducción

En el presente documento se muestran los resultados obtenidos tras implementar y probar algunos métodos numéricos para sistemas no lineales. Los métodos implementados en código que se analizarán en el documento actual son: el *algoritmo de Brent* y el *método Regula Falsi*. Posteriormente se mostrará como el uso de las funciones de ciertos módulos de Python, como SymPy y ScyPy, pueden facilitar la implementación de lo que se quiere lograr.

Figure 1: Logo de Python

2 Algoritmo de Brent

2.1 Diagrama de flujo

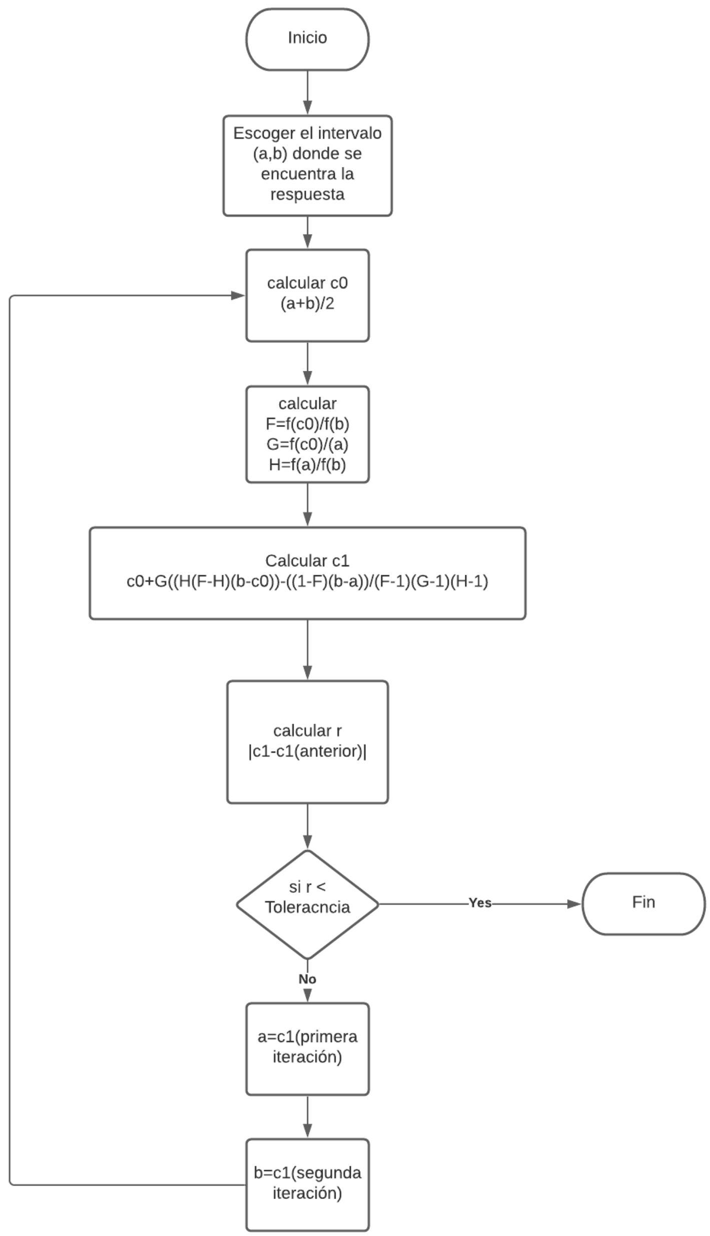

Figure 2: Diagrama de flujo Algoritmo de Brent

2.2 Evaluación para todos los casos

En la siguiente figura se pueden observar los parámetros que fueron tomados en cuenta para la solución de la ecuación $f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27}$ implementando el método de Brent. Se consideraron el número de iteraciones, una tolerancia, el resultado, la comparación del resultado teórico y el experimental (Error relativo) y el tiempo de ejecución.

Numero Iteraciones	Tolerancia	Resultado	Error Relativo	Tiempo Ejecución (s)
518	1,00E-05	0,667758483418624	0,16377%	0,08573222160339
11800	1,00E-07	0,666905477789765	0,03582%	9,23841500282287
1186652	1,00E-10	0,666690746386363	0,00361%	989,9773063659660

Figure 3: Gráfica error Ei y Ei+1

2.3 Comparación de los casos 1 y 2

En la figura tres se observan los resultados para tres tolerancias distintas,10^{-6},10^{-8} y 10^{-11}.Para cumplir con estas tolerancias se gastaron diferentes cantidades de iteraciones y tiempo para llegar a los resultados. Estos resultados se compararon con el valor teórico el cual es $\frac{2}{3}$ y se obtuvieron errores relativos cada vez mas pequeños. Pero, para obtener un error mas pequeño se tuvieron que gastar muchos mas recursos.

En el caso de la tolerancia igual a 10^{-11} le tomo al programa del método implementado nada mas y nada menos que 1'186.652 iteraciones y un tiempo aproximado de 990 segundos, lo cual equivale a 16 minutos y medio. Por lo anterior se expresa que esta cantidad de recursos es muy grande comparada a las que se usan con las otras dos tolerancias.

Por otro lado, se cree que aunque con la tolerancia de 10^{-6} se utilice una cantidad de tiempo expresada en décimas de segundo y las iteraciones sean casi 23 veces menos que las de la tolerancia 10^{-8}, el tiempo de ejecución relativamente bueno para obtener un resultado con un Error relativo menor, lo cual es el objetivo, acercarse lo mas posible a la solución.

2.4 Gráficas

2.4.1 Error Ei y Ei+1

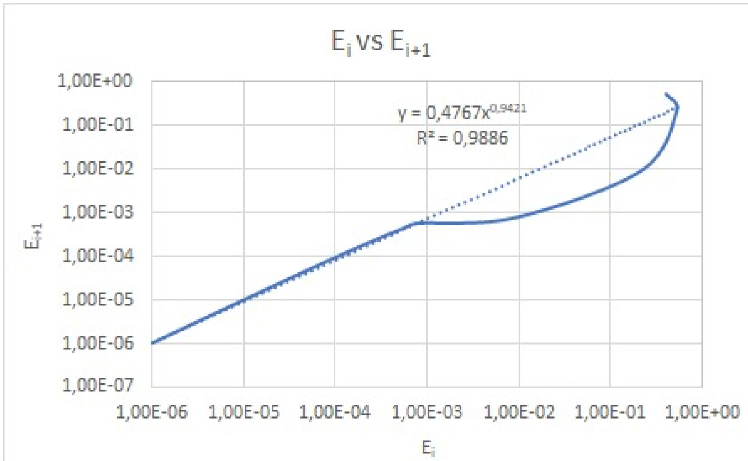

Figure 4: Gráfica error Ei y Ei+1

2.4.2 Error vs Iteración

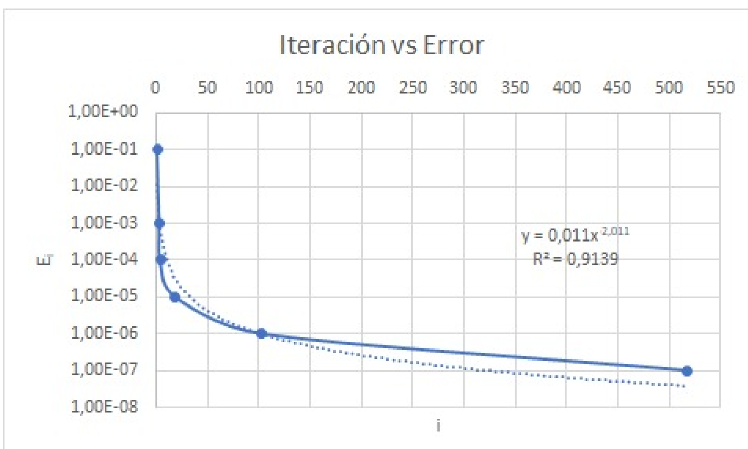

Figure 5: Gráfica error vs iteración

3 Estimación de raíces por Posición Falsa

3.1 Diagrama de flujo y explicación del método

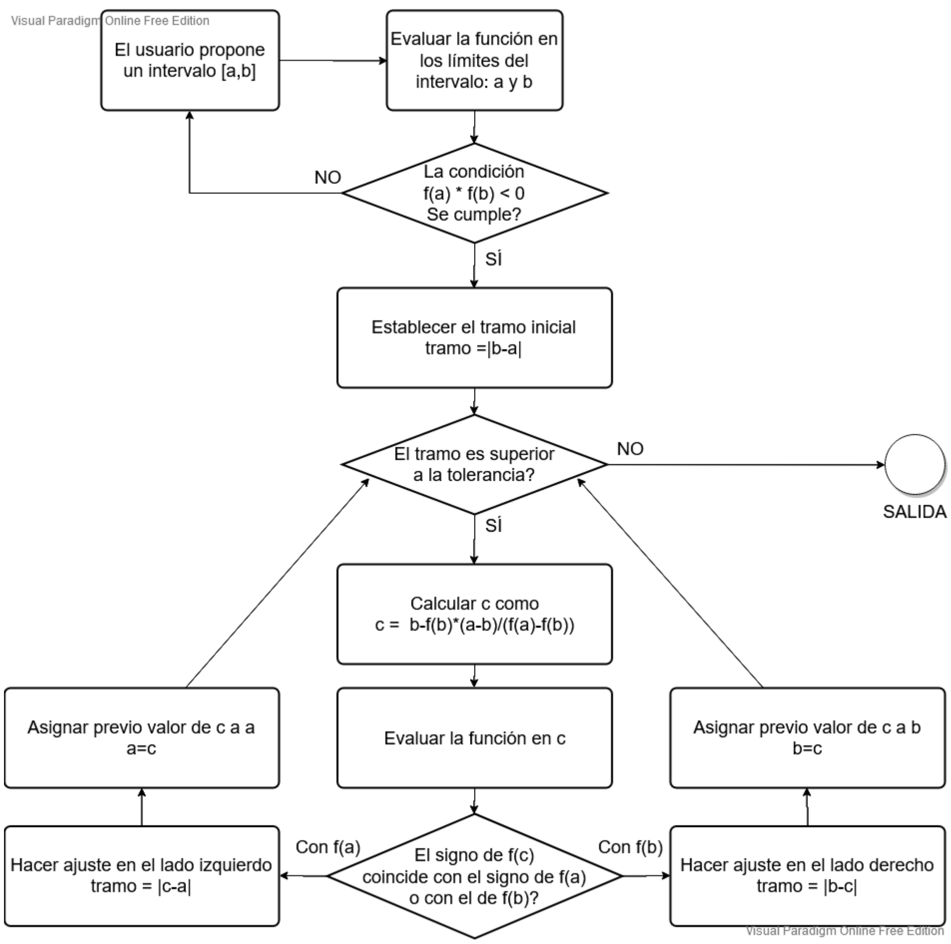

Figure 6: Diagrama de flujo del método de Posición Falsa

El método *Regula Falsi* se describe de la siguiente manera: la recta que inicia en a y termina en b; y pasa por el punto c (intersección con el eje x), se aproxima a la función (f(x)) dada. En consecuencia al cambio de tramo —i.e. cambio en el valor de uno de los extremos del intervalo en que se está evaluando la función— que se hace tras cada iteración del algoritmo, la recta cambia su longitud y, de mayor importancia, su pendiente. Por consiguiente, tal punto de intersección (c) varía, y su proximidad a la raíz de la función presente en el intervalo, incrementa. Para ilustrar lo que está sucediendo, se observa que la recta se desplaza hasta que el punto c se acerca tanto como sea posible (teniendo en cuenta la tolerancia dada) a la solución. El valor de c se obtiene mediante despeje al hallar la equivalencia por semejanza de triángulos entre la sección positiva de la recta, y la sección negativa de la misma.

3.2 Evaluación para todos los casos

Función: -3*x^3 + 60*x + 57 - 300/x
Tolerancia: 2^-16
Intervalo [1,3]

Aproximación	Ei	Ei+1
2.5371900826	0.46280991735537200000	0.28312339680493800000
2.2540666858	0.28312339680493800000	0.14750430888439800000
2.1065623770	0.14750430888439800000	0.07196229694328780000
2.0346000800	0.07196229694328780000	0.03420493229113350000
2.0003951477	0.03420493229113350000	0.01608297797111950000
1.9843121697	0.01608297797111950000	0.00752642003592041000
1.9767857497	0.00752642003592041000	0.00351466191614613000
1.9732710878	0.00351466191614613000	0.00163965912028540000
1.9716314287	0.00163965912028540000	0.00076458709218574200
1.9708668416	0.00076458709218574200	0.00035645855129740000
1.9705103830	0.00035645855129740000	0.00016616846984507300
1.9703442146	0.00016616846984507300	0.00007745838049233450
1.9702667562	0.00007745838049233450	0.00003610596713343730
1.9702306502	0.00003610596713343730	0.00001683004413055570
1.9702138202	0.00001683004413055570	0.00000784493814909304
1.9702059752	0.00000784493814909304	

Figure 7: Tabla para el caso 1

La anterior tabla presenta la información correspondiente a la estimación de la raíz para cada iteración, junto al error estimado. En última instancia, la estimación se aproxima a 1.9702059752.

Función: -3*x^3 + 60*x + 57 - 300/x
Tolerancia: 2^-50
Intervalo [3, 4.6]

Aproximación	Ei	Ei+1
2.5371900826	0.46280099173553720000000000000000	0.28312339680493800000000000000000
2.2540666858	0.28312339680493800000000000000000	0.14750430888439800000000000000000
2.1065623770	0.14750430888439800000000000000000	0.07196229694328780000000000000000
2.0346000800	0.07196229694328780000000000000000	0.03420493229113350000000000000000
2.0003951477	0.03420493229113350000000000000000	0.01608297797111950000000000000000
1.9843121697	0.01608297797111950000000000000000	0.00752642003592041000000000000000
1.9767857497	0.00752642003592041000000000000000	0.00351466191614613000000000000000
1.9732710878	0.00351466191614613000000000000000	0.00163965912028540000000000000000
1.9716314287	0.00163965912028540000000000000000	0.00076458709218574200000000000000
1.9708668416	0.00076458709218574200000000000000	0.00035645855129740000000000000000
1.9705103830	0.00035645855129740000000000000000	0.00016616846984507300000000000000
1.9703442146	0.00016616846984507300000000000000	0.00007745838049233450000000000000
1.9702667562	0.00007745838049233450000000000000	0.00003610596713343730000000000000
1.9702306502	0.00003610596713343730000000000000	0.00001683004413055570000000000000
1.9702138202	0.00001683004413055570000000000000	0.00000784493814909304000000000000
1.9702059752	0.00000784493814909304000000000000	0.00000365672970192854000000000000
1.9702023185	0.00000365672970192854000000000000	0.00000170449510994430000000000000
1.9702006140	0.00000170449510994430000000000000	0.00000079450833379013300000000000
1.9701998195	0.00000079450833379013300000000000	0.00000037034037303484900000000000
1.9701994492	0.00000037034037303484900000000000	0.00000017262497120640300000000000
1.9701992765	0.00000017262497120640300000000000	0.00000008046484101242860000000000
1.9701991961	0.00000008046484101242860000000000	0.00000003750668531310450000000000
1.9701991586	0.00000003750668531310450000000000	0.00000001748280089782863000000000
1.9701991411	0.00000001748280089782863000000000	0.00000000814917666680514000000000
1.9701991329	0.00000000814917666680514000000000	0.00000000379853593024392000000000
1.9701991291	0.00000000379853593024392000000000	0.00000000177059322759021000000000
1.9701991274	0.00000000177059322759021000000000	0.00000000082531803613505800000000
1.9701991265	0.00000000082531803613505800000000	0.00000000038470138186141800000000
1.9701991262	0.00000000038470138186141800000000	0.00000000017931900408996100000000
1.9701991260	0.00000000017931900408996100000000	0.00000000008358513881034920000000
1.9701991259	0.00000000008358513881034920000000	0.00000000003896105660317060000000
1.9701991259	0.00000000003896105660317060000000	0.00000000001816058414760840000000
1.9701991258	0.00000000001816058414760840000000	0.00000000000846545056276681000000
1.9701991258	0.00000000000846545056276681000000	0.00000000000394573262951780000000
1.9701991258	0.00000000000394573262951780000000	0.00000000000183919546259403000000
1.9701991258	0.00000000000183919546259403000000	0.00000000000085731421961554500000
1.9701991258	0.00000000000085731421961554500000	0.00000000000039945824426013100000
1.9701991258	0.00000000000039945824426013100000	0.00000000000018629542353210100000
1.9701991258	0.00000000000018629542353210100000	0.00000000000008704148513061220000
1.9701991258	0.00000000000008704148513061220000	0.00000000000004041211809635570000
1.9701991258	0.00000000000004041211809635570000	0.00000000000001887379141862760000
1.9701991258	0.00000000000001887379141862760000	0.00000000000000865973959207622000
1.9701991258	0.00000000000000865973959207622000	0.00000000000000421884749357559000
1.9701991258	0.00000000000000421884749357559000	0.00000000000000199840144432528000
1.9701991258	0.00000000000000199840144432528000	0.00000000000000066613381477509300
1.9701991258	0.00000000000000066613381477509300	

La anterior tabla presenta la información correspondiente a la estimación de la raíz para cada iteración, junto al error estimado. En última instancia, la estimación se aproxima a 1.9701991258.

Función: -3*x^3 + 60*x + 57 - 300/x

Tolerancia: 2^-16

Intervalo [1,3]

Aproximación	Ei	Ei+1
4.1168533870	1.1168533869792900000	0.2289357907678040000
4.3457891777	0.2289357907678040000	0.0237178384943810000
4.3695070162	0.0237178384943810000	0.0022345098294023200
4.3717415261	0.0022345098294023200	0.0002085626729595440
4.3719500887	0.0002085626729595440	0.0000194496171079805
4.3719695384	0.0000194496171079805	0.0000018136358503895
4.3719713520	0.0000018136358503895	

Figure 9: Tabla para el caso 3

La anterior tabla presenta la información correspondiente a la estimación de la raíz para cada iteración, junto al error estimado. En última instancia, la estimación se aproxima a 4.371971352.

Función: -3*x^3 + 60*x + 57 - 300/x
Tolerancia: 2^-50
Intervalo [3, 4.6]

Aproximación	Ei	Ei+1
4.1168533870	1.11685338697929000000000000000	0.22893579076780400000000000000
4.3457891777	0.22893579076780400000000000000	0.02371783849438100000000000000
4.3695070162	0.02371783849438100000000000000	0.00223450982940232000000000000
4.3717415261	0.00223450982940232000000000000	0.00020856267295954400000000000
4.3719500887	0.00020856267295954400000000000	0.00001944961710798050000000000
4.3719695384	0.00001944961710798050000000000	0.00000181363585038951000000000
4.3719713520	0.00000181363585038951000000000	0.00000016911643818673300000000
4.3719715211	0.00000016911643818673300000000	0.00000001576962116445200000000
4.3719715369	0.00000001576962116445200000000	0.00000000147047174436920000000
4.3719715384	0.00000000147047174436920000000	0.00000000013711609625488500000
4.3719715385	0.00000000013711609625488500000	0.00000000001278621653000300000
4.3719715385	0.00000000001278621653000300000	0.00000000000119282361765726000
4.3719715385	0.00000000000119282361765726000	0.00000000000011102230246251500
4.3719715385	0.00000000000011102230246251500	0.00000000000000976996261670137
4.3719715385	0.00000000000000976996261670137	0.22802846149545100000000000000
4.3719715385	0.22802846149545100000000000000	0.00000000000000000000000000000
4.3719715385	0.00000000000000000000000000000	

Figure 10: Tabla para el caso 4

La anterior tabla presenta la información correspondiente a la estimación de la raíz para cada iteración, junto al error estimado. En última instancia, la estimación se aproxima a 4.3719715385.

3.3 Comparación de los casos 1 y 2

Dado que las estimaciones, para cada uno de los casos (i.e. la implementación con Regula Falsi, y la implementación con SciPy y SymPy) se planteó de manera diferente —para Regula Falsi se encontró una expresión f(x), mientras que la implementación con librerías implicó la búsqueda del resultado mediante un sistema de ecuaciones de dos incógnitas—, no es posible compararlas de manera directa.

3.4 Gráficas

Para la curva $x^2 + xy = 10$ (1):

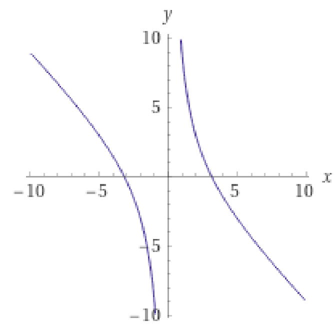

Figure 11: Gráfica para la curva 1

Para la curva $y + 3xy^2 = 57$ (2):

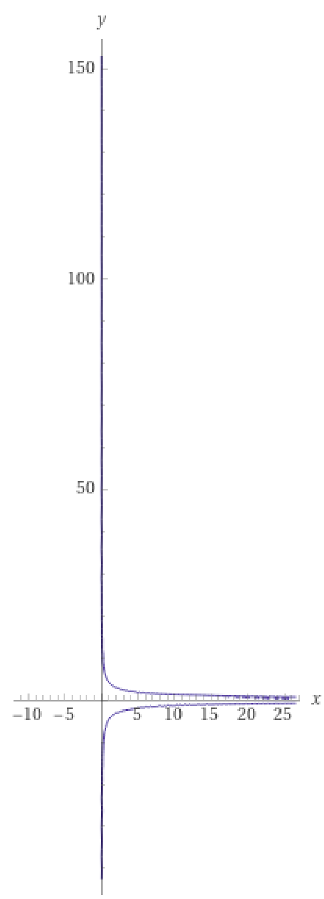

Figure 12: Gráfica para la curva 2

Para la función $f(x) = -3x^3 + 60x + 57 - 300/x$, correspondiente a la intersección entre las curvas (1) y (2):

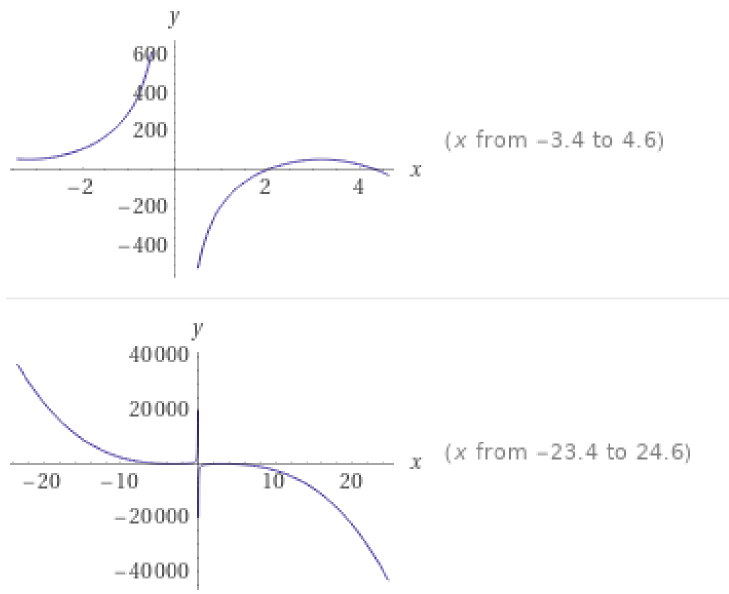

Figure 13: Gráficas para la expresión resultante (intersección)

3.4.1 Error Ei+1 y Ei

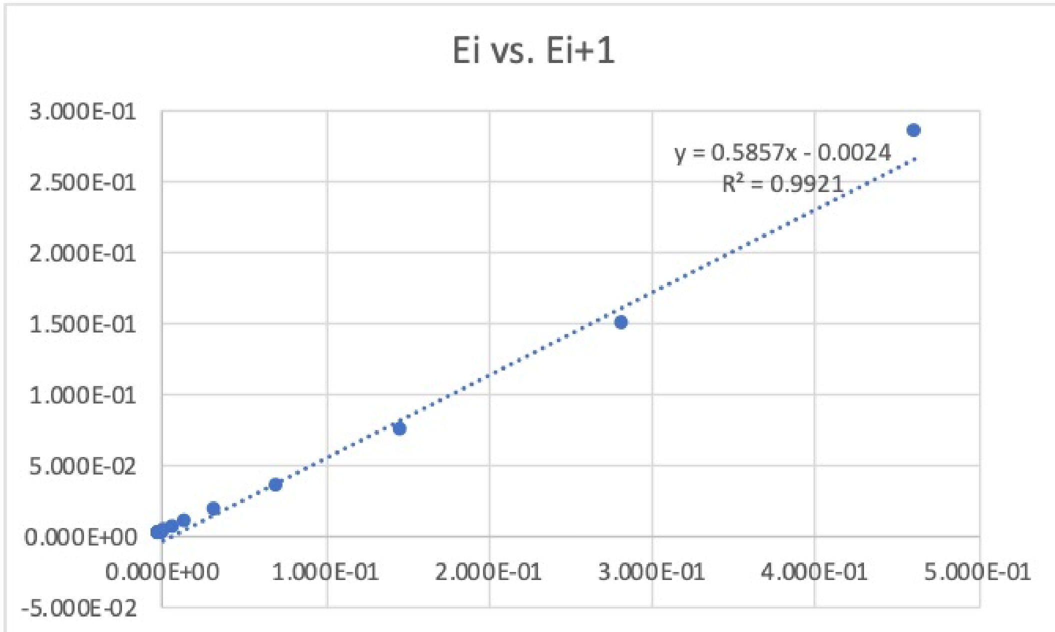

Figure 14: Ei vs. Ei+1 para el caso 1

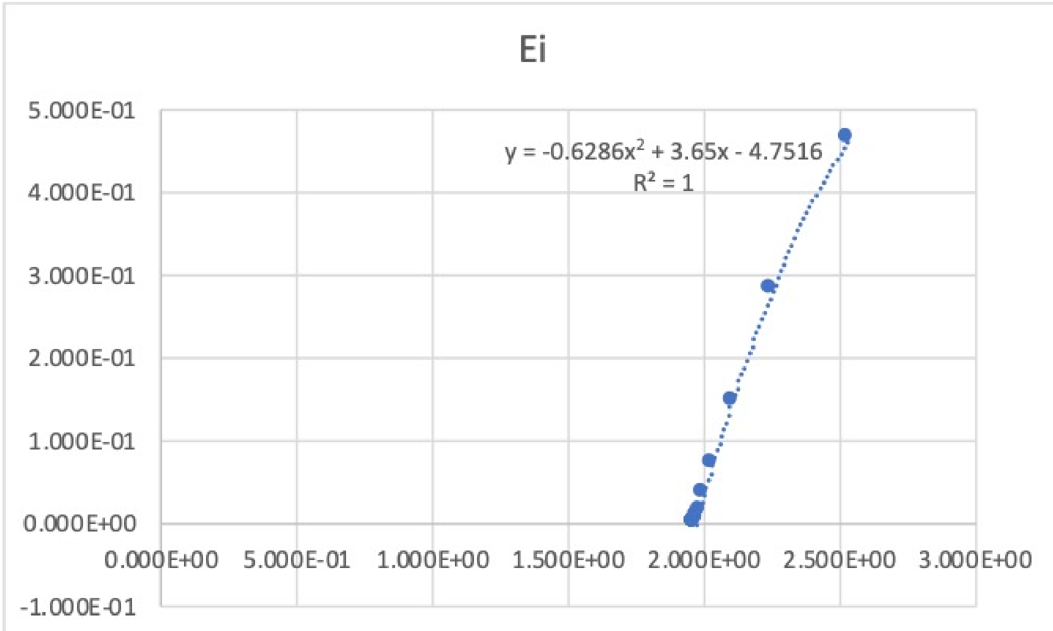

Figure 15: Aproximación vs. Error en i para el caso 1

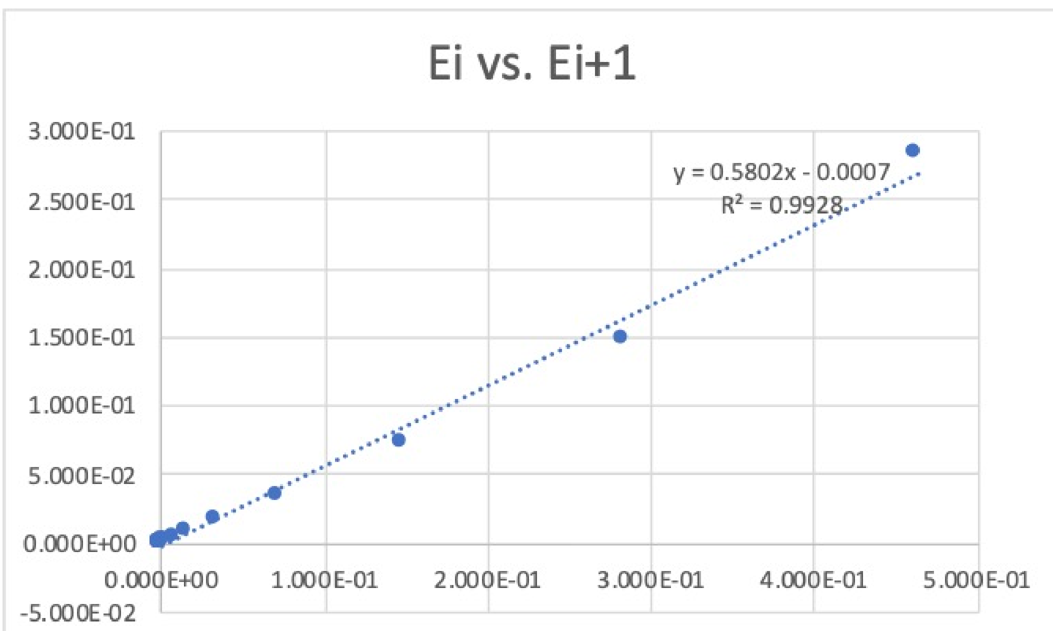

Figure 16: Ei vs. Ei+1 para el caso 1

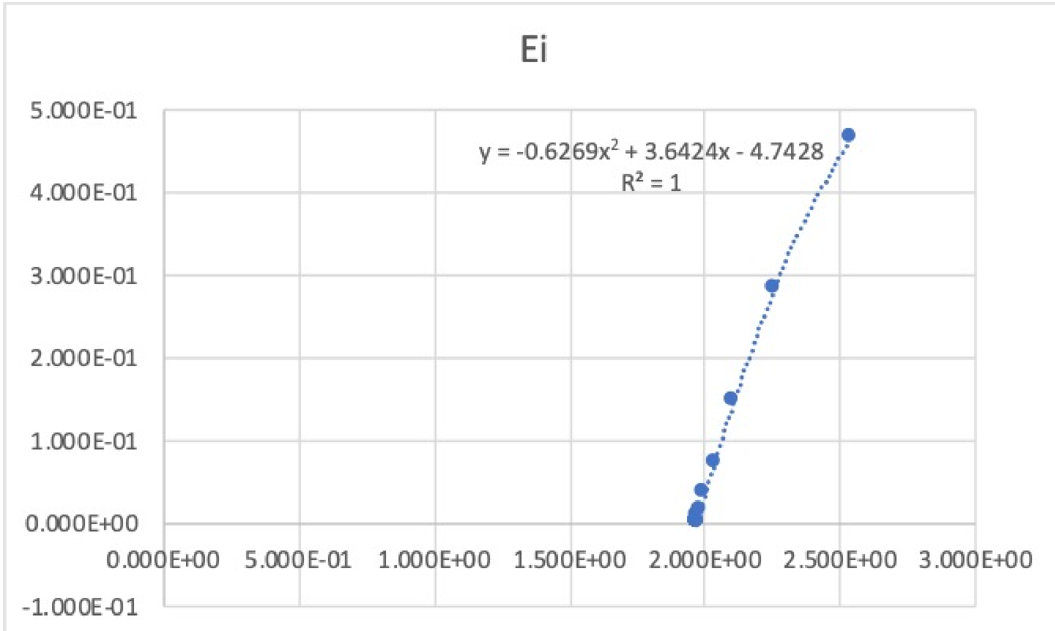

Figure 17: Aproximación vs. Error en i para el caso 2

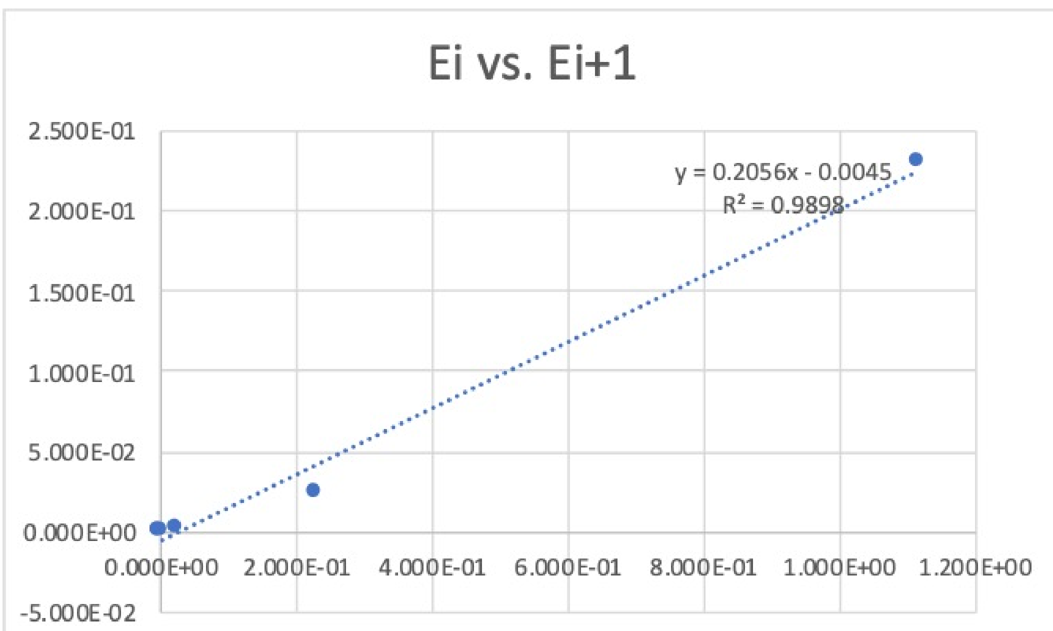

Figure 18: Ei vs. Ei+1 para el caso 1

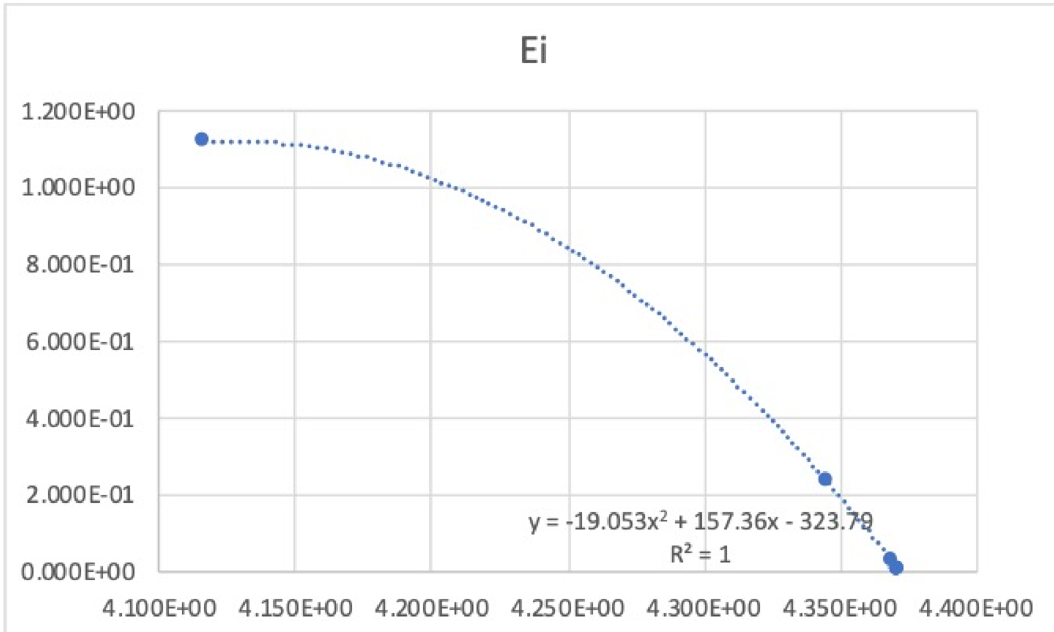

Figure 19: Aproximación vs. Error en i para el caso 3

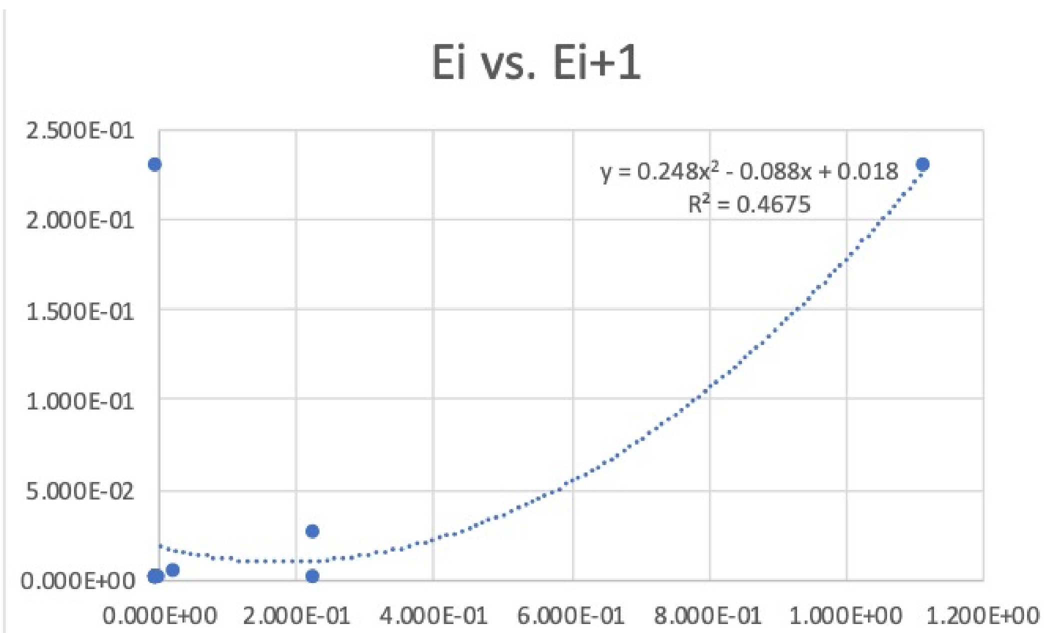

Figure 20: Ei vs. Ei+1 para el caso 1

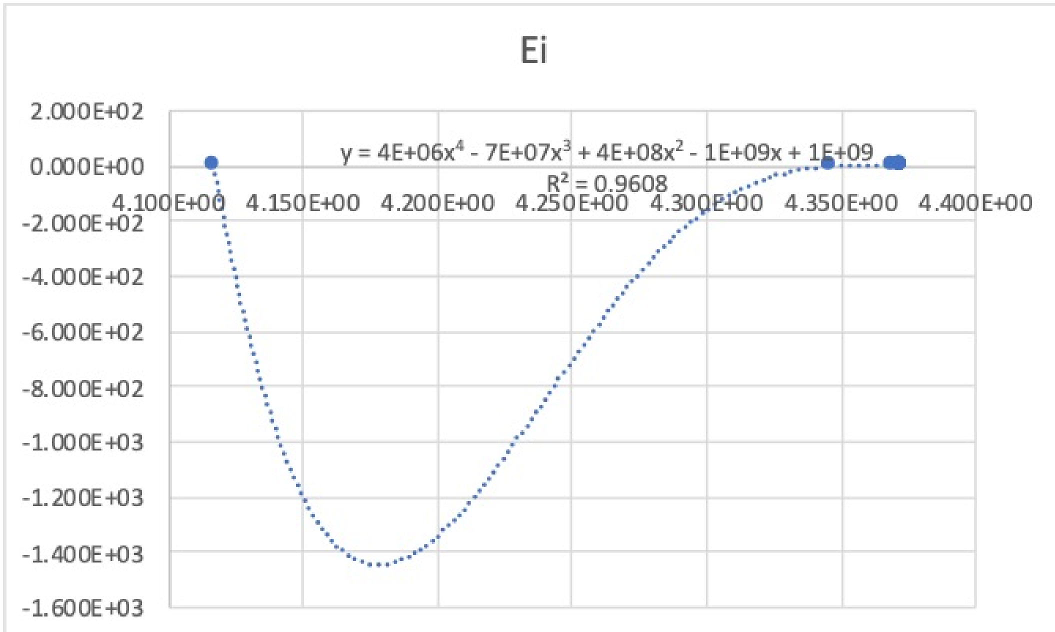

Figure 21: Aproximación vs. Error en i para el caso 4

3.4.2 Error vs. Iteración

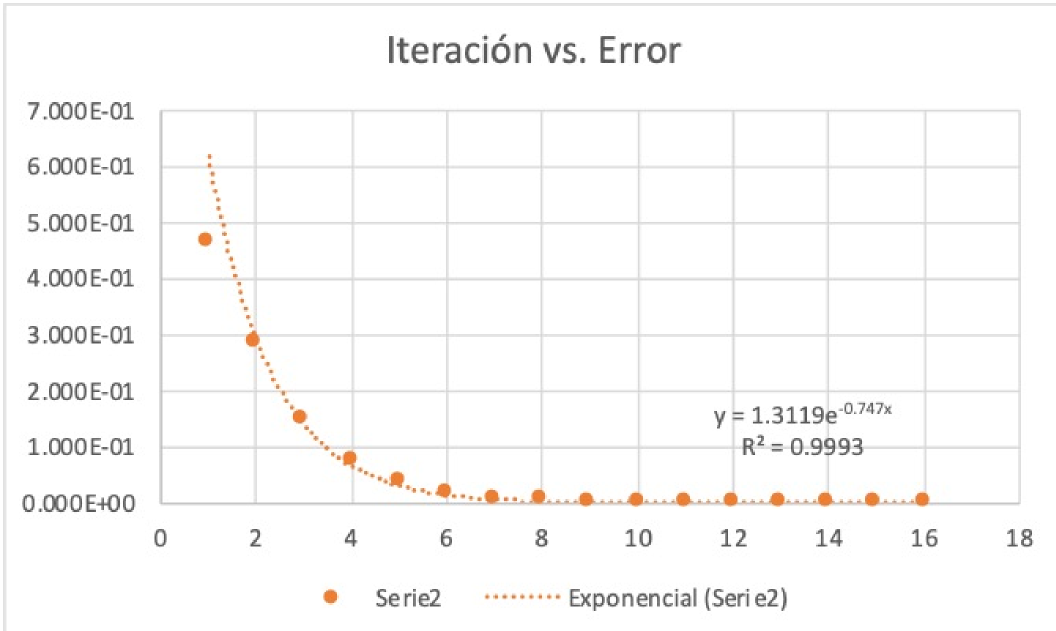

Figure 22: Iteración vs. Error caso 1

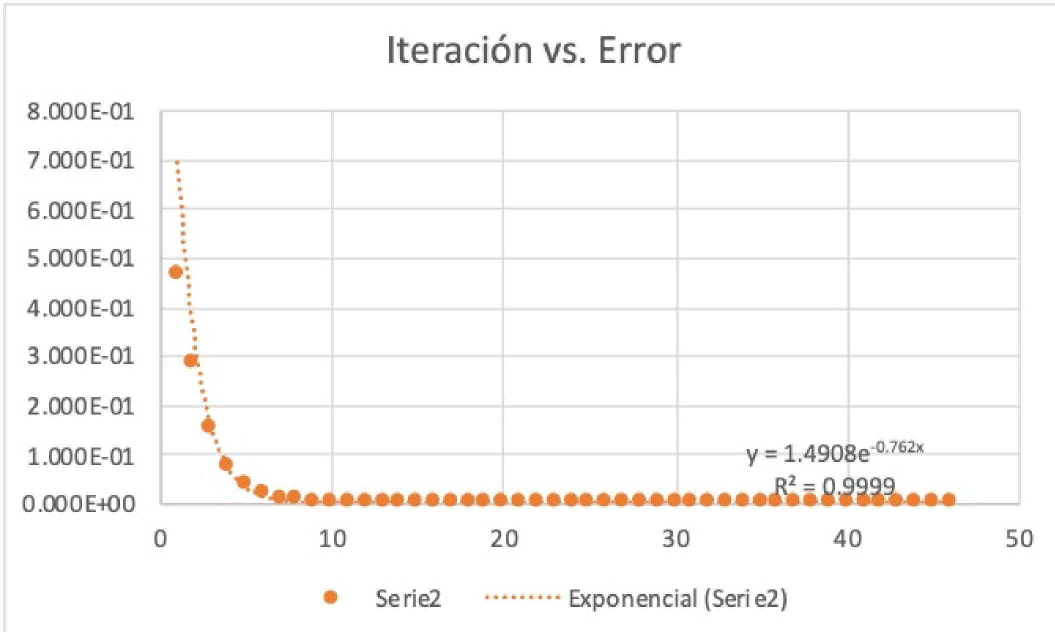

Figure 23: Iteración vs. Error caso 2

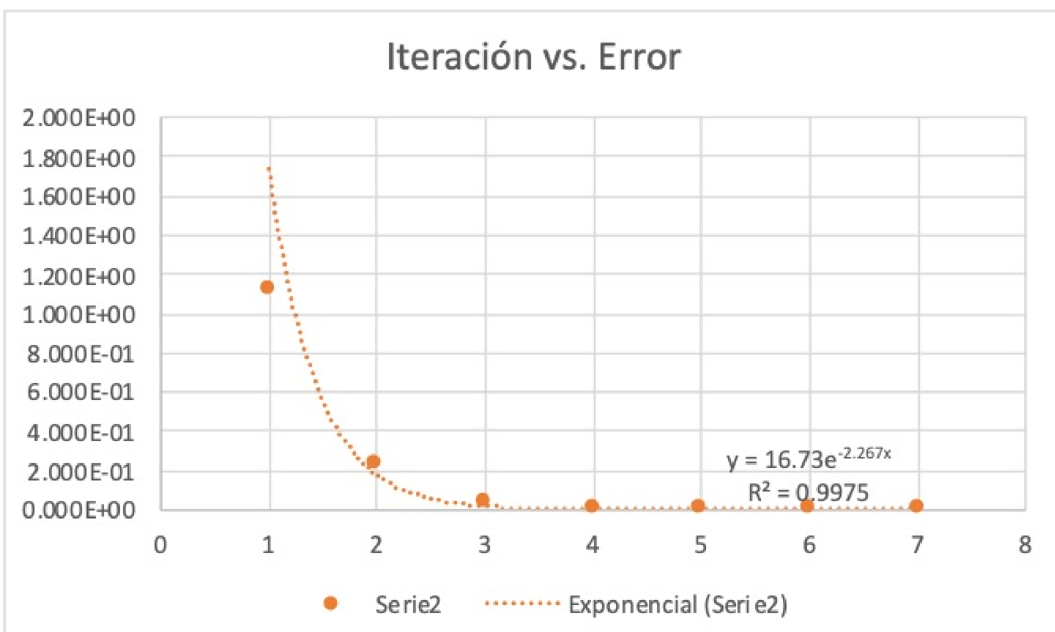

Figure 24: Iteración vs. Error caso 3

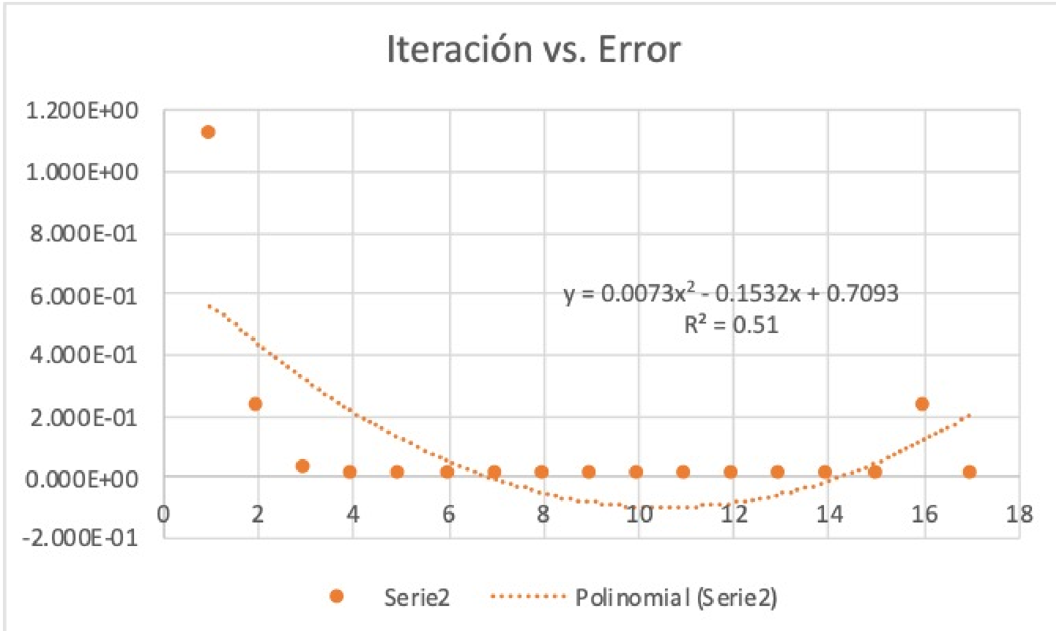

Figure 25: Iteración vs. Error caso 4

4 Implementación de los métodos usando librerías

En este apartado se busca resolver los problemas de las primeras secciones pero mediante el uso de módulos de Python. Los módulos que se utilizaron para resolver los problemas fueron *ScyPy* y *SymPy*.

4.1 Algoritmo de Brent

Para encontrar las raíces del polinomio $f(x) = x^3 - 2x^2 + \frac{4x}{3} - \frac{8}{27}$ se hace uso de la función *brentq* incluida en el paquete *scypy.optimize*. La función *brentq* encuentra una raíz de una función en un intervalo cerrado usando el método clásico de Brent. De esta manera se encuentra un cero de la función en el intervalo de cambio de signo [a, b]. El método de Brent es una versión segura del método secante que usa extrapolación cuadrática inversa [1]. El método de Brent utiliza un polinomio de interpolación de Lagrange de grado 2. Brent afirma que este método convergerá siempre y cuando los valores de la función sean computables dentro de una región dada que contenga una raíz. Dados tres puntos x1, x2 y x3, el método de Brent ajusta x como una función cuadrática de y y luego usa la fórmula de interpolación [2].

A continuación se muestran todos los parámetros que la función puede recibir:

- **f**: *función elemental*

Función de Python que devuelve un número. La función debe ser continua y debe tener signos opuestos.

- **a**: *escalar*
  El limite inferior del intervalo cerrado.

- **b**: *escalar*
  El limite superior del intervalo cerrado.

- **xtol** (parámetro opcional): *numero*
  Tolerancia admitida. La raíz calculada x0 satisfará np.allclose (x, x0, atol = xtol, rtol = rtol), donde x es la raíz exacta. El parámetro no debe ser negativo.

- **rtol** (parámetro opcional): *numero*
  Tolerancia admitida. La raíz calculada x0 satisfará np.allclose (x, x0, atol = xtol, rtol = rtol), donde x es la raíz exacta. El parámetro no puede ser menor que su valor predeterminado de 4 * np.finfo (float) .eps.

- **maxiter** (parámetro opcional): *entero*
  El numero máximo de iteraciones. Si no se logra la convergencia en iteraciones máximas, se genera un error. Este valor debe ser siempre superior o igual a 0.

- **args** (parámetro opcional): *tupla*
  Una tupla que contiene argumentos adicionales para la función f.

- **full_output** (parámetro opcional): *booleano*
  Si full_output es False, se devuelve la raíz. Si full_output es True, el valor de retorno es (x, r), donde x es la raíz y r es un objeto de tipo RootResults.

- **disp** (parámetro opcional): *booleano*
  Si disp es True, se genera un RuntimeError si el algoritmo no convergió. De lo contrario, el estado de convergencia se registra en cualquier objeto de retorno de RootResults.

A continuación se muestra el resultado del primer punto utilizando los módulos descritos previamente:

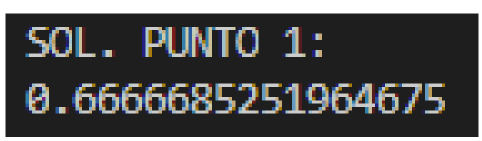

Figure 26: Solución punto 1 con *SciPy*

4.2 Aproximación a la raíz por posición falsa

Para hallar la intersección entre las curvas $x^2 + xy = 10$ y $y + 3xy^2 = 57$ se hace uso de la función *solve* incluida en el paquete *sympy.solvers.solvers*. La función *solve* resuelve ecuaciones y sistemas de ecuaciones algebraicas [3]. Para hacer uso de esta función, la expresión que se pasa como parámetro debe estar organizada de tal forma que solo se pase el lado de la ecuación que tenga todos los términos, por ende no se incluye ni la igualdad ni el termino nulo del otro lado de la ecuación.

A continuación se muestran todos los parámetros que la función puede recibir:

- **f**: *expresión(es)*
  Puede ser una sola expresión o varias expresiones (una igualdad, una expresión relacional, un sistema de ecuaciones, etc.).

- **símbolo(s)**: *caracter(es)*
  Las incógnitas o variables para las cuales se debe resolver la expresión pasada por parámetro.

El resto de parámetros que se le pueden pasar a la función *solve* son parámetros opcionales y no aportan nada relevante para este documento, si desea saber mas acerca de ellos puede visitar la documentación que aparece en las referencias.

A continuación se muestra el resultado del segundo punto utilizando los módulos descritos previamente:

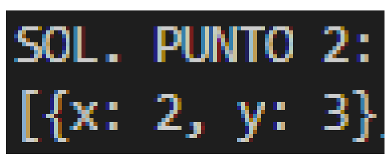

Figure 27: Solución punto 2 con *SymPy*

References

[1] *scipy.optimize.brentq — SciPy v1.6.1 Reference Guide*. 1. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brentq.html` (visited on 03/13/2021).

[2] *Brent's Method – from Wolfram MathWorld*. 2. URL: `https://mathworld.wolfram.com/BrentsMethod.html` (visited on 03/13/2021).

[3] *Solvers — SymPy 1.7.1 documentation*. 3. URL: `https://docs.sympy.org/latest/modules/solvers/solvers.html` (visited on 03/13/2021).