Eficiencia de la Iteración de Newton de Orden 3, en la Solución Numérica por el Método de Disparo

Aro Huanacuni, Alex[1,*] ; **Santander Mamani, Oscar**[1]

[1] *Universidad Nacional del Altiplano, Departamento de Ciencias Físico Matemáticas, Puno, Perú*

Resumen: El método de disparo es una técnica iterativa que usa velocidad de disparo para obtener una solución óptima. El propósito de la investigación es verificar la eficacia de las iteraciones de Newton de orden 3 adaptadas y acopladas a RK4, en el método disparo. Se realizó un experimento numérico con ejemplos mediante el método de disparo en las ecuaciones diferenciales ordinarias no lineal de orden 2, y se obtuvo velocidades de disparo y errores en cada iteración. Los resultados obtenidos sobre velocidades finales fueron satisfactorios con las iteraciones NW, NWO3-1, NWO3-2 y NWO3-3 aplicadas al método de disparo y de los cuales, NWO3-2 y NWO3-3 tienen alta precisión en los experimentos realizados.

Palabras claves: Eficiencia, Newton Orden 3, Velocidad de disparo, Método de disparo

Efficiency of Newton's Iteration of Order 3, in the Numerical Solution by the Shooting Method

Abstract: The shooting method is an iterative technique that uses shooting speed to obtain an optimal solution. The purpose of the research is to verify the effectiveness of Newton iterations of order 3 adapted and coupled to RK4, in the shooting method. Numerical experiments were performed with examples using the shooting method in nonlinear ordinary differential equations of order 2, and shooting velocities and errors were obtained in each iteration.The results obtained on final velocities were satisfactory with the iterations NW, NWO3-1, NWO3-2 and NWO3-3 applied to the shooting method and of which, NWO3-2 and NWO3-3 have high accuracy in the experiments performed.

Keywords: Efficiency, Newton Order 3, Speed of shooting, Method of shooting

1. INTRODUCCIÓN

El método de disparo es una técnica utilizada para encontrar soluciones de ecuaciones diferenciales ordinarias con valores en la frontera y se efectúa realizando disparos a ciertas velocidades hasta alcanzar el blanco que se sitúa en la frontera. El problema de valores en la frontera (PVF), estudiado por Bailey & Shampine (1968), está dado de la forma

$$x'' = f(t,x,x'),\ x(a) = \alpha,\ x(b) = \beta,\ t \in \langle a,b \rangle, \qquad (1)$$

Este tipo de ecuaciones tiene soluciones analíticas en algunos casos y en otras se hace más tedioso como en el caso no lineal, pero se puede garantizar soluciones únicas implícitas bajo las condiciones suficientes planteadas y demostradas por Schrader (1969), Ha (2001) y Keller (2018). Por su parte, Burden et al. (2017) consideran para análisis del problema (1), el problema de valor inicial (PVI) dada por

$$x'' = f(t,x,x'),\ x(a,\eta) = \alpha,\ x'(a,\eta) = \eta, \qquad (2)$$

con η velocidad de disparo y $\eta_0 = (\beta - \alpha)/(b-a)$ velocidad de disparo inicial para la parte numérica. La solución exacta $x = x(t,\eta)$ de (2) está garantizada por las condiciones mencionadas anteriormente, que son fundamentales para formular la solución numérica del problema (1). En la formulación, la aproximación numérica busca la velocidad de disparo η de mayor precisión, que satisface la ecuación $x(b,\eta) - \beta = 0$, y esto garantiza una buena aproximación a la solución exacta de (1) en la frontera $t = b$. Para encontrar velocidades se puede utilizar técnicas iterativas como: bisección, secante, punto fijo, Newton y otros. El rendimiento de estos métodos son diferentes, sin embargo la iteración de Newton clásico (NW) es utilizada por muchos investigadores por su rendimiento y ciertas modificaciones como el de Osborne (1969). Para sistema diferencial, Attili & Syam (2008) aplicaron la técnica de descomposición adomiana y NW en ambos casos en el método de disparo y obtuvieron buenos resultados. Las iteraciones de Runge Kutta de orden 4 (RK4) y de NW pueden ser acoplados al método disparo en la solución de diferentes problemas como de Dirichlet, Sturm-Liouville, Robin y rotación del eje motor (Ahsan & Farrukh, 2013; Granas et al., 1979; Liu, 2006; Mataušek, 1974). En los problemas, es difícil obtener en forma explícita la solución del PVF, por lo que se recurre a métodos numéricos como el de Euler, Diferencias finitas, Runge Kutta y entre otros para aproxi-

*ayaro@unap.edu.pe
Recibido: 30/03/2023
Aceptado: 04/10/2023
Publicado en línea: 09/02/2024
10.33333/rp.vol53n1.07

mar a la solución exacta con alta precisión y también se utiliza para describir el comportamiento cualitativo de ciertos modelos relacionados con las ecuaciones diferenciales. El método RK4, propuesto por Kutta (1901), e iteración de Newton clásico serán aplicados a la ecuación (1), en el estudio conjuntamente con Newton modificadas de orden 3. En la presente investigación, se realizó un experimento numérico con método de disparo utilizando RK4 e iteraciones adaptadas de Newton del mismo orden 3, y es aplicado al PVF no lineal (1) orden 2, con el propósito de determinar la eficiencia de iteraciones adaptadas en la aproximación del PVF. Para obtener resultados numéricos esperados, se utilizó Python 3 por su mayor uso en la actualidad.

2. MÉTODOS

En la ecuación (2), se deriva con respecto a η y se evalúa en $\eta = \eta_k$ con $(k = 0,\ldots,n)$, se obtiene un nuevo PVI:

$$\frac{\partial x''}{\partial \eta} = \frac{\partial f}{\partial x}\frac{dx}{d\eta} + \frac{\partial f}{\partial x'}\frac{dx'}{d\eta},\ \frac{\partial x}{\partial \eta}(a,\eta_k) = 0,\ \frac{\partial x'}{\partial \eta}(a,\eta_k) = 1, \quad (3)$$

donde x', denota la derivada con respecto a t.
Considerando $x(t,\eta_k) = u_1(t)$, $x'(t,\eta_k) = u_2(t)$, $\frac{\partial x}{\partial \eta}(t,\eta_k) = v_1(t)$ y $\frac{\partial x'}{\partial \eta}(t,\eta_k) = v_2(t)$, en las ecuaciones (2) y (3) se obtiene las siguientes nuevas ecuaciones:

$$\begin{cases} u_1' = & u_2,\ u_1(a) = \alpha, \\ u_2' = & f(t,u_1,u_2),\ u_2(a) = \eta_k \end{cases} \quad (4)$$

y

$$\begin{cases} v_1' = & v_2,\ v_1(a) = 0, \\ v_2' = & \frac{\partial f}{\partial u_1}(t,u_1,u_2)v_1 + \frac{\partial f}{\partial u_2}(t,u_1,u_2)v_2,\ v_2(a) = 1. \end{cases} \quad (5)$$

Para obtener solución numérica $U_j = (u_1(t_j,\eta_k), u_2(t_j,\eta_k))$, para $j \geq 1$, en tiempo t_j con velocidad de disparo η_k, se aplicó la tabla moderna de RK4 utilizada por Butcher (1996) al sistema (4), como sigue

$$K_1 = hF\left(t_{j-1}, U_{j-1}\right), \quad (6)$$

$$K_2 = hF\left(t_{j-1} + \frac{h}{2}, U_{j-1} + \frac{1}{2}K_1\right), \quad (7)$$

$$K_3 = hF\left(t_{j-1} + \frac{h}{2}, U_{j-1} + \frac{1}{2}K_2\right), \quad (8)$$

$$K_4 = hF(t_{j-1} + h, U_{j-1} + K_3), \quad (9)$$

$$U_j = U_{j-1} + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right), \quad (10)$$

para $F(t,U) = (u_2, f(t,u_1,u_2))$, $U = (u_1,u_2)$ y $U_0 = (\alpha,\eta_0)$. Utilizando $G(t,V,U) = (v_2, \frac{\partial f}{\partial u_1}(t,u_1,u_2)v_1 + \frac{\partial f}{\partial u_2}(t,u_1,u_2)v_2)$, con $V = (v_1,v_2)$ y $V_0 = (0,1)$, se obtiene de forma análoga $V_j = (v_1(t_j,\eta_k), v_2(t_j,\eta_k))$ por RK4 aplicado a $G(t,V,U_j)$.
Los resultados obtenidos por RK4 permite obtener las siguientes aproximaciones $x(t_j,\eta_k) \approx u_1(t_j,\eta_k)$, $x'(t_j,\eta_k) \approx u_2(t_j,\eta_k)$ y $\frac{\partial x}{\partial \eta}(t_j,\eta_k) \approx v_1(t_j,\eta_k)$, y estos son utilizados en NW, para determinar una nueva velocidad de disparo s_k y reformulando nuevamente las ecuaciones (4) y (5) para $\eta_k = s_k$, se obtiene nuevas aproximaciones $u_1(b,s_k)$ y $v_1(b,s_k)$ para velocidad de disparo s_k, así sucesivamente hasta lograr aproximación eficiente. Estos valores numéricos obtenidos son asociados a velocidad de disparo η_k, y el siguiente paso es buscar velocidades de disparo por iteraciones adaptadas propuestas en la investigación. A continuación, presentamos iteraciones adaptadas de orden 3, de dos pasos con el propósito de construir $\eta_1,\ldots,\eta_n$, a partir de η_0.
La iteración Newton orden 3 (NWO3-1), propuesta por Weerakoon & Fernando (2000), y al sustituir por $u_1(b,\eta_k)$ y $u_1(b,s_k)$ (aproximaciones de x en $t = b$ asociados a las velocidades η_k y s_k, respectivamente) y, $v_1(b,\eta_k)$ y $v_1(b,s_k)$ (son aproximación a $\frac{\partial x}{\partial \eta}(b,\eta_k)$ y $\frac{\partial x}{\partial \eta}(b,s_k)$, respectivamente) quedan adaptadas de la siguiente manera:

$$s_k = \eta_k - \frac{u_1(b,\eta_k) - \beta}{v_1(b,\eta_k)} \quad (11)$$

$$\eta_{k+1} = \eta_k - 2\frac{u_1(b,s_k) - \beta}{v_1(b,\eta_k) + v_1(b,s_k)},\ k = 0,\ldots,n, \quad (12)$$

donde s_k y η_{k+1} son velocidades de disparo de primer y segundo paso, respectivamente.
La otra perspectiva iteración Newton orden 3 (NWO3-2), dada por Darvishi & Barati (2007), y utilizando las aproximaciones mencionadas en la iteración anterior, tiene la forma adaptada dada por:

$$s_k = \eta_k - \frac{u_1(b,\eta_k) - \beta}{v_1(b,\eta_k)} \quad (13)$$

$$\eta_{k+1} = \eta_k - \frac{u_1(b,s_k) + u_1(b,\eta_k) - 2\beta}{v_1(b,\eta_k)},\ k = 0,\ldots,n, \quad (14)$$

donde s_k y η_{k+1} son velocidades de disparo de primer y segundo paso, respectivamente.
Magreñán Ruiz & Argyros (2014) usan la técnica iteración de orden 3 (NWO3-3), y manera similar como los casos anteriores quedan adaptadas las velocidades de disparo de la siguiente manera:

$$s_k = \eta_k - \frac{u_1(b,\eta_k) - \beta}{v_1(b,\eta_{-}k)} \quad (15)$$

$$\eta_{k+1} = s_k - \frac{u_1(b,s_k) - \beta}{v_1(b,\eta_k)},\ k = 0,\ldots,n, \quad (16)$$

donde s_k y η_{k+1} son velocidades de disparo de primer y segundo paso, respectivamente.
Estas iteraciones son diferentes pero del mismo orden 3, y fueron demostradas con rigurosidad por los autores mencionados y dichas adaptaciones fueron utilizadas para verificar si las velocidades de disparos encontrados son efectivos. Para la presente investigación, se realizó el experimento con las siguientes iteraciones adaptada de NW, NWO3-1, NWO3-2, NWO3-3 y RK4, en la determinación de velocidades de disparo óptimo con la condición de pare $|u_1(b,\eta_k) - \beta| < tol$, donde tol denota la tolerancia, además se pretende estimar los errores de aproximación a la solución exacta. Asimismo, para la implementación del script del método de disparo se utilizó el diagrama de flujo de la Figura 1 , los paquetes NumPy (2022) y SymPy (2022), y se realizó en Python 3, con interfaz Jupyter Notebook.

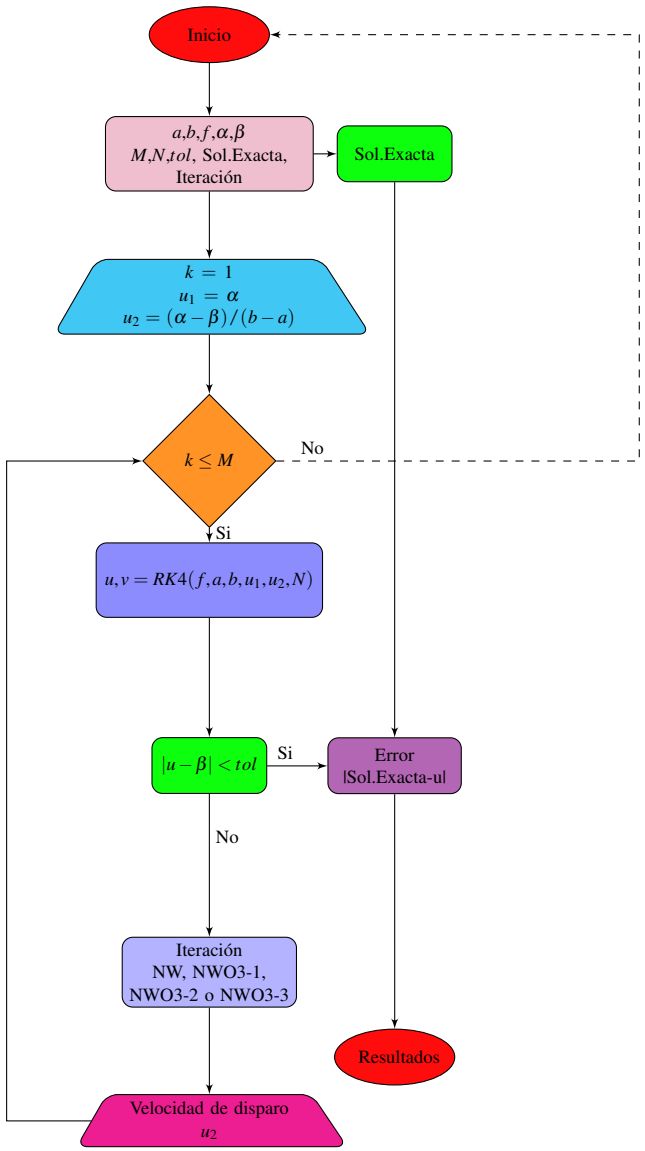

Figura 1. Diagrama de flujo del método de disparo

3. RESULTADOS Y DISCUSIÓN

El experimento se realizó con la implementación de *class Method_diaparo()* (ver, apéndices A y B) y los ejemplos:

3.1 Ejemplo 1

Considere el problema de valores en la frontera (Filipov et al., 2017):

$$x'' = -\frac{3x^2x'}{t},\ x(1) = \frac{1}{\sqrt{2}},\ x(2) = \frac{2}{\sqrt{5}}, \qquad (17)$$

donde $x(t) = t/\sqrt{t^2+1}$, es solución exacta.
Observe la Tabla 1, los errores de aproximación a la solución exacta en la frontera $t = \frac{2}{\sqrt{5}}$, generados a diferentes velocidades obtenidas por iteraciones adaptadas. Las iteraciones NWO3-1, NWO3-2 y NWO3-3 se desempeñan con alta precisión que NW. Además, estas iteraciones requieren un promedio 4 o 3 procesos en la estimación; las velocidades finales obtenidas por iteraciones NWO3-2 y NWO3-3 tienen excelentes aproximaciones en la frontera que los demás, como se puede ver en la Figura 2, 1.a), 1.b), 1.c) y 1.d).

Tabla 1. Velocidades de disparo determinados por $|u_1(2,\eta_k) - 2/\sqrt{5}| < 10^{-6}$, NW, NWO3-1, NWO3-2 y NWO3-3 con velocidad inicial $\eta_0 = 0.18732041$, en el Ejemplo 1

Iteración	N	k	**Velocidad** η_k	**Error** $\lvert u_1(2,\eta_k) - 2/\sqrt{5}\rvert$
NW	10	0	0.18732041	$8.33042178 \times 10^{-2}$
		1	0.34483077	$4.17409946 \times 10^{-3}$
		2	0.35343823	$5.46837987 \times 10^{-5}$
		3	0.35355153	$5.96827138 \times 10^{-7}$
	20	0	0.18732041	$8.33041975 \times 10^{-2}$
		1	0.34528631	$3.95582105 \times 10^{-3}$
		2	0.35348789	$3.12490276 \times 10^{-5}$
		3	0.35355298	$1.77547096 \times 10^{-7}$
	30	0	0.18732041	$8.33041964 \times 10^{-2}$
		1	0.34544693	$3.87879728 \times 10^{-3}$
		2	0.35350412	$2.35151177 \times 10^{-5}$
		3	0.35355319	$9.02327065 \times 10^{-8}$
	40	0	0.18732041	$8.33041962 \times 10^{-2}$
		1	0.34552881	$3.83953140 \times 10^{-3}$
		2	0.35351217	$1.96784501 \times 10^{-5}$
		3	0.35355327	$5.70059140 \times 10^{-8}$
NWO3-1	10	0	0.18732041	$8.33042178 \times 10^{-2}$
		1	0.35165765	$9.05161726 \times 10^{-4}$
		2	0.35353219	$9.82880362 \times 10^{-6}$
		3	0.35355256	$1.06983022 \times 10^{-7}$
	20	0	0.18732041	$8.33041975 \times 10^{-2}$
		1	0.35233152	$5.83463330 \times 10^{-4}$
		2	0.35354644	$3.29959069 \times 10^{-6}$
		3	0.35355331	$1.86884414 \times 10^{-8}$
	30	0	0.18732041	$8.33041964 \times 10^{-2}$
		1	0.35256914	$4.69974834 \times 10^{-4}$
		2	0.35354962	$1.79476060 \times 10^{-6}$
		3	0.35355337	$6.86235535 \times 10^{-9}$
	40	0	0.18732041	$8.33041962 \times 10^{-2}$
		1	0.35269030	$4.12107629 \times 10^{-4}$
		2	0.35355090	$1.18778164 \times 10^{-6}$
		3	0.35355338	$3.42712281 \times 10^{-9}$
NWO3-2	10	0	0.18732041	$8.33042178 \times 10^{-2}$
		1	0.35272309	$3.96160193 \times 10^{-4}$
		2	0.35355268	$4.98226820 \times 10^{-8}$
	20	0	0.18732041	$8.33041975 \times 10^{-2}$
		1	0.35278755	$3.65644335 \times 10^{-4}$
		2	0.35355333	$1.30788557 \times 10^{-8}$
	30	0	0.18732041	$8.33041964 \times 10^{-2}$
		1	0.35280959	$3.55132580 \times 10^{-4}$
		2	0.35355337	$6.07323403 \times 10^{-9}$
	40	0	0.18732041	$8.33041962 \times 10^{-2}$
		1	0.35282071	$3.49825478 \times 10^{-4}$
		2	0.35355338	$3.56956209 \times 10^{-9}$
NWO3-3	10	0	0.18732041	$8.33042178 \times 10^{-2}$
		1	0.35272309	$3.96160193 \times 10^{-4}$
		2	0.35355268	$4.98226818 \times 10^{-8}$
	20	0	0.18732041	$8.33041975 \times 10^{-2}$
		1	0.35278755	$3.65644335 \times 10^{-4}$
		2	0.35355333	$1.30788556 \times 10^{-8}$
	30	0	0.18732041	$8.33041964 \times 10^{-2}$
		1	0.35280959	$3.55132580 \times 10^{-4}$
		2	0.35355337	$6.07323403 \times 10^{-9}$
	40	0	0.18732041	$8.33041962 \times 10^{-2}$
		1	0.35282071	$3.49825478 \times 10^{-4}$
		2	0.35355338	$3.56956176 \times 10^{-9}$

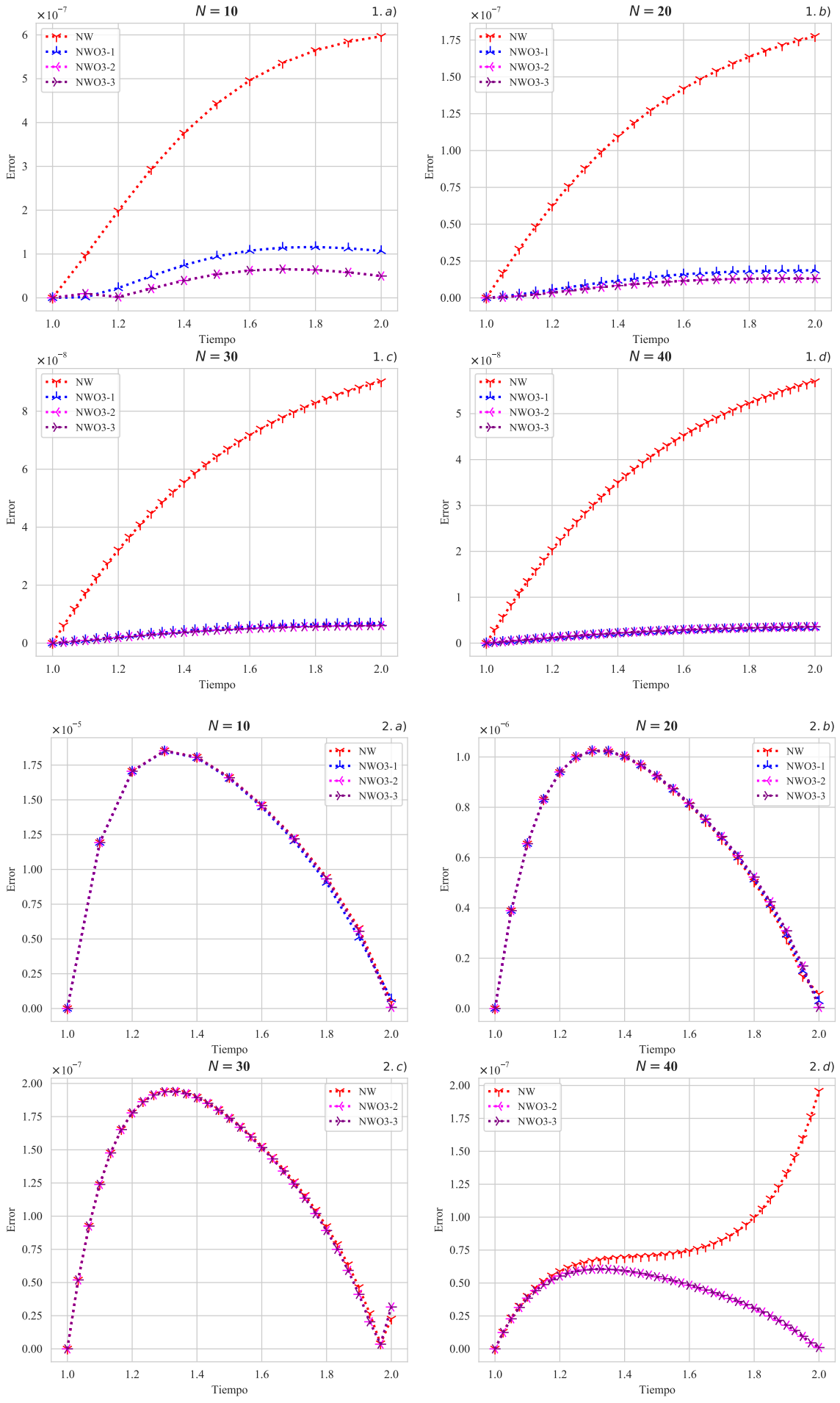

Figura 2. Errores de aproximación a la solución exacta por iteración NW, NWO3-1, NWO3-2 y NWO3-3 a lo largo del intervalo $[1,2]$, para $N = 10, 20, 30, 40$ y a velocidades finales η_k. 1.a), 1.b), 1.c) y 1.d) son errores para el Ejemplo 1 y 2.a), 2.b), 2.c) y 2.d) son errores para el Ejemplo 2

Las iteraciones utilizadas en el Ejemplo 1, a diferentes pasos se obtuvieron de forma satisfactoria las velocidades de disparo y así como sus aproximaciones a la solución exacta. Estos resultados pueden ser replicados a pasos más refinados.

3.2 Ejemplo 2

Sea el problema de valores en la frontera que considera Ha (2001):

$$x'' = 2x^3 - 6x - 2t^3,\ x(1) = 2,\ x(2) = 2.5, \qquad (18)$$

cuya solución exacta es $x(t) = (t^2+1)/t$.
La iteración NWO3-1 no determina velocidades de disparo para 30 y 40 pasos, debido a la divergencia de la velocidad de disparo y los demás, lo cual permitió alcanzar aproximaciones en la frontera con $t = 2.5$, ver Tablas 2 y 3.

Tabla 2. Velocidades de disparo determinado por $|u_1(2,\eta_k) - 2.5| < 10^{-6}$, NW y NWO3-1, iniciando $\eta_0 = 0.5$ en el Ejemplo 2

Iteración	N	k	**Velocidad** η_k	**Error** $\lvert u_1(2,\eta_k) - 2.5\rvert$
NW	10	0	0.50000000	1.57950953×10^{2}
		1	0.08341190	1.25249309
		2	0.00439167	$5.41334520 \times 10^{-2}$
		3	−0.00018506	$2.65300411 \times 10^{-3}$
		4	0.00004277	$1.47130714 \times 10^{-4}$
		5	0.00003014	$8.11190472 \times 10^{-6}$
		6	0.00003084	$4.47387389 \times 10^{-7}$
	20	0	0.50000000	1.77913305×10^{3}
		1	0.12128427	2.03652136
		2	0.02204041	$2.85075996 \times 10^{-1}$
		3	0.00021781	$2.65655168 \times 10^{-3}$
		4	−0.00000378	$7.34036004 \times 10^{-5}$
		5	0.00000234	$2.05706002 \times 10^{-6}$
		6	0.00000217	$5.76246051 \times 10^{-8}$
	30	0	0.50000000	4.58868564×10^{4}
		1	−0.23664092	1.93721273
		2	0.10392841	1.65854857
		3	0.01839778	$2.36033621 \times 10^{-1}$
		4	0.00028657	$3.52587048 \times 10^{-3}$
		5	−0.00000477	$6.41937569 \times 10^{-5}$
		6	0.00000055	$1.20428081 \times 10^{-6}$
		7	0.00000045	$2.25802430 \times 10^{-8}$
	40	0	0.50000000	6.93565868×10^{6}
		1	−3.40338089	7.90252662
		2	0.01104305	$1.39350371 \times 10^{-1}$
		3	0.00008111	$9.97339394 \times 10^{-4}$
		4	−0.00000098	$1.38920473 \times 10^{-5}$
		5	0.00000016	$1.95747490 \times 10^{-7}$
NWO3-1	10	0	0.50000000	1.57950953×10^{2}
		1	−0.29974344	2.25670816
		2	−0.11589228	1.14144361
		3	−0.00628252	$7.65838082 \times 10^{-2}$
		4	0.00035463	$3.98459836 \times 10^{-3}$
		5	0.00001288	$2.20355096 \times 10^{-4}$
		6	0.00003179	$1.21509639 \times 10^{-5}$
		7	0.00003075	$6.70143617 \times 10^{-7}$
	20	0	0.50000000	1.77913305×10^{3}
		1	−0.25413734	2.03121130
		2	−0.09285544	$9.52524058 \times 10^{-1}$
		3	−0.00411759	$5.02800912 \times 10^{-2}$
		4	0.00011184	$1.35072947 \times 10^{-3}$
		5	−0.00000090	$3.78763211 \times 10^{-5}$
		6	0.00000226	$1.06101454 \times 10^{-6}$
		7	0.00000218	$2.97226559 \times 10^{-8}$
	30	0	0.50000000	–
	40	0	0.50000000	–

Tabla 3. Velocidades de disparo determinado por $|u_1(2,\eta_k) - 2.5| < 10^{-6}$, NWO3-2 y NWO3-3, iniciando $\eta_0 = 0.5$ en el Ejemplo 2

Iteración	N	k	**Velocidad** η_k	**Error** $\lvert u_1(2,\eta_k) - 2.5\rvert$
NWO3-2	10	0	0.50000000	1.57950953×10^{2}
		1	0.08010851	1.19236030
		2	0.00082903	$9.83220498 \times 10^{-3}$
		3	0.00003306	$2.77249143 \times 10^{-5}$
		4	0.00003306	$8.43114010 \times 10^{-8}$
	20	0	0.50000000	1.77913305×10^{3}
		1	0.12085076	2.02658198
		2	0.00805983	$1.00993764 \times 10^{-1}$
		3	0.00000174	$5.40117776 \times 10^{-6}$
		4	0.00000218	$4.23893320 \times 10^{-9}$
	30	0	0.50000000	4.58868564×10^{4}
		1	0.23660982	1.93704166
		2	−0.18755890	1.64806280
		3	−0.08176044	$8.55690275 \times 10^{-1}$
		4	−0.00576814	$7.01602412 \times 10^{-2}$
		5	−0.00000682	$8.95503778 \times 10^{-5}$
		6	0.00000044	$3.15567332 \times 10^{-8}$
	40	0	0.50000000	$6.93565868e \times 10^{6}$
		1	−3.40337644	7.90251538
		2	−0.04921419	$5.47666655 \times 10^{-1}$
		3	−0.00128997	$1.58436173 \times 10^{-2}$
		4	−0.00000027	$5.13568843 \times 10^{-6}$
		5	0.00000014	$1.01954090 \times 10^{-9}$
NWO3-3	10	0	0.50000000	1.57950953×10^{2}
		1	0.08010851	1.19236030
		2	0.00082903	$9.83220498 \times 10^{-3}$
		3	0.00003306	$2.77249143 \times 10^{-5}$
		4	0.00003081	$8.43114001 \times 10^{-8}$
	20	0	0.50000000	1.77913305×10^{3}
		1	0.12085076	2.02658198
		2	0.00805983	$1.00993764 \times 10^{-1}$
		3	0.00000174	$5.40117776 \times 10^{-6}$
		4	0.00000218	$4.23893320 \times 10^{-9}$
	30	0	0.50000000	4.58868564×10^{4}
		1	−0.23660982	1.93704166
		2	−0.18755890	1.64806280
		3	−0.08176044	$8.55690275 \times 10^{-1}$
		4	−0.00576814	$7.01602412 \times 10^{-2}$
		5	−0.00000682	$8.95503778 \times 10^{-5}$
		6	0.00000044	$3.15567332 \times 10^{-8}$
	40	0	0.50000000	6.93565868×10^{6}
		1	−3.40337644	7.90251538
		2	−0.04921419	$5.47666655 \times 10^{-1}$
		3	−0.00128997	$1.58436173 \times 10^{-2}$
		4	−0.00000027	$5.13568842 \times 10^{-6}$
		5	0.00000014	$1.01954090 \times 10^{-9}$

El error mínimo en la frontera para velocidad estimada es $1.01954090 \times 10^{-9}$ y se realizó con NWO3-2 y NWO3-3, para $N = 40$ en el Ejemplo 2 y los demás resultados también son eficaces, pero en menor cantidad de dígitos de precisión ver en las Tablas 2 y 3. Los errores obtenidos por NWO3-1, NWO3-2 y NWO3-3 son menores al resultado obtenido por Ha (2001) y estas iteraciones requieren realizar menor cantidad de operaciones.
En la Figura 2, 2.a), 2.b), 2.c) y 2.d), se observan los errores de aproximación a la solución exacta con velocidades estimadas por iteraciones adaptadas. Dentro de los cuales, NWO3-2 y NWO3-3 son precisas en 8 dígitos a la solución exacta en la frontera para $N = 20$ y $N = 40$, y en 7 dígitos para $N = 10$ y $N = 30$. Sin embargo, las iteraciones NWO3-2 y NWO3-3 son excelentes para aproximar para cualquier paso refinado.

4. CONCLUSIONES

Las iteraciones NWO3-2, NWO3-3 y RK4 aplicadas al método de disparo son eficientes para aproximar a las soluciones exactas de problemas de valores en la frontera de orden 2. Además, NWO3-1 y NW desempeñan en forma similar y requieren mayor cantidad de procesos que las iteraciones NWO3-2 y NWO3-3.

REFERENCIAS

Ahsan, M., & Farrukh, S. (2013). A new type of shooting method for nonlinear boundary value problems. *Alexandria Engineering Journal, 52*(4), 801–805. https://doi.org/10.1016/j.aej.2013.07.001

Attili, B., & Syam, M. (2008). Efficient shooting method for solving two point boundary value problems. *Chaos, Solitons and Fractals, 35*(5), 895–903. https://doi.org/10.1016/j.chaos.2006.05.094

Bailey, P., & Shampine, L. (1968). On shooting methods for two-point boundary value problems. *Journal of Mathematical Analysis and Applications, 23*(2), 235–249. https://doi.org/10.1016/0022-247X(68)90064-4

Burden, R., Faires, J., & Burden, A. (2017). *Análisis Numérico* (10th ed.). Cengage Learning.

Butcher, J. (1996). History of Runge-Kutta methods. *Appllied Numerical Mathematics, 20*(3), 247–260. https://doi.org/10.1016/0168-9274(95)00108-5

Darvishi, M., & Barati, A. (2007). A third-order Newton-type method to solve systems of nonlinear equations. *Applied Mathematics and Computation, 187*(2), 630–635. https://doi.org/10.1016/j.amc.2006.08.080

Filipov, S., Gospodinov, I., & Faragó, I. (2017). Shooting-projection method for two-point boundary value problems. *Applied Mathematics Letters, 72*, 10–15. https://doi.org/10.1016/j.aml.2017.04.002

Granas, A., Guenther, R., & Lee, J. (1979). The Shooting Method for the Numerical Solution of a Class of Nonlinear Boundary Value Problems. *SIAM Journal on Numerical Analysis, 16*(5), 828–836. https://doi.org/10.1137/0716062

Ha, S. N. (2001). A nonlinear shooting method for two-point boundary value problems. *Computers and Mathematics with Applications, 42*(10–11), 1411–1420. https://doi.org/10.1016/S0898-1221(01)00250-4

Keller, H. B. (2018). *Numerical Methods for Two-Point Boundary-Value Problems.* Dover Publications.

Kutta, W. (1901). Beitrag zur näherungsweisen Integration otaler Differentialgleichungen. *Zeit. Math. Phys., 46*, 435–453.

Liu, C. (2006). The Lie-group shooting method for nonlinear two-point boundary value problems exhibiting multiple solutions. *CMES - Computer Modeling in Engineering and Sciences, 13*(2), 149–163. https://doi.org/10.3970/cmes.2006.013.149

Magreñán Ruiz, Á., & Argyros, I. (2014). *Two-step Newton methods. Journal of Complexity, 30*(4), 533–553. https://doi.org/10.1016/j.jco.2013.10.002

Mataušek, M. (1974). Direct shooting method, linearization, and nonlinear algebraic equations. *Journal of Optimization Theory and Applications, 14*(2), 199–212. https://doi.org/10.1007/BF00932940

NumPy. (2022). *NumPy User Guide v1.3.* https://numpy.org/doc/1.23/numpy-ref.pdf

Osborne, M. (1969). On shooting methods for boundary value problems. *Journal of Mathematical Analysis and Applications, 27*(2), 417–433. https://doi.org/10.1016/0022-247X(69)90059-6

Schrader, K. (1969). Existence theorems for second order boundary value problems. *Journal of Differential Equations, 5*(3), 572–584. https://doi.org/10.1016/0022-0396(69)90094-1

SymPy. (2022). *SymPy Documentation.* https://github.com/sympy/sympy/releases

Weerakoon, S., & Fernando, T. (2000). A variant of Newton's method with accelerated third-order convergence. *Applied Mathematics Letters, 13*(8), 87–93. https://doi.org/10.1016/S0893-9659(00)00100-2

BIOGRAFÍAS

Alex Youn, Aro Huanacuni es Magíster en Matemáticas por la Universidad Federal de Río de Janeiro (UFRJ-Brasil) y Licenciado en Ciencias Físico Matemáticas de la Universidad Nacional del Altiplano, Puno. Sus áreas de investigación son la geometría simpléctica, geometría compleja y análisis numérico. Desde 2017, es profesor a tiempo completo en la Universidad Nacional del Altiplano.

Oscar, Santander Mamani es Licenciado en Ciencias Físico Matemáticas e Ingeniero de Minas de la Universidad Nacional del Altiplano, Puno. Cuenta con el grado de Magíster en Informática con mención en Matemática y Simulación Computacional. Sus áreas de investigación son el análisis complejo, métodos numéricos, geomecánica y SSOMA.

Apéndice A. Implementación del método de disparo en Python 3:

```
class Method_disparo():
    def __init__(self,f):
        self.f=f
    def RungeKutta_4(self,a,b,u1,u2,N):
        f_np=lambdify([t,[x,x1]],self.f,modules="numpy")
        F=lambda t,u:np.array([u[1],f_np(t,[u[0],u[1]])])
        f_x=lambdify([t,[x,x1]],f.diff(x),modules="numpy")
        f_x1=lambdify([t,[x,x1]],f.diff(x1),modules="numpy")
        G=lambda t,v,u:np.array([v[1],f_x(t,u)*v[0]+f_x1(t,u)*v[1]])
        h=(b-a)/N
        tam=2
        u=np.zeros((tam,N+1))
        v=np.zeros((tam,N+1))
        t1=a+np.arange(N+1)*h
        k=1
        u[0:tam,0]=np.array([u1,u2])
        v[0:tam,0]=np.array([0,1])
        for i in range(N):
            K1=h*F(t1[i],u[0:tam,i])
            K2=h*F(t1[i]+(1/2)*h,u[0:tam,i]+(1/2)*K1)
            K3=h*F(t1[i]+(1/2)*h,u[0:tam,i]+(1/2)*K2)
            K4=h*F(t1[i]+h,u[0:tam,i]+K3)
            u[0:tam,i+1]=u[0:tam,i]+(1/6)*(K1+2*K2+2*K3+K4)
            K_1=h*G(t1[i],v[0:tam,i],u[0:tam,i])
            K_2=h*G(t1[i]+(1/2)*h,v[0:tam,i]+(1/2)*K_1,u[0:tam,i])
            K_3=h*G(t1[i]+(1/2)*h,v[0:tam,i]+(1/2)*K_2,u[0:tam,i])
            K_4=h*G(t1[i]+h,v[0:tam,i]+K_3,u[0:tam,i])
            v[0:tam,i+1]=v[0:tam,i]+(1/6)*(K_1+2*K_2+2*K_3+K_4)
        return t1,u,v
    def Aprox_numeric(self,method,a,b,alpha,beta,N,M,tol):
        k=1
        u1=alpha
        u2=(beta-alpha)/(b-a)
        Vacio=np.empty((0, N+1), float)
        print("="*60)
        while k<=M:
            t1,u,v=Method_disparo.RungeKutta_4(self,a,b,u1,u2,N)
            u_result=np.array([u[0,:]])
            X=np.append(Vacio,u_result,axis=0)
            Vacio=X
            print("Velocidad %s:{:^15.8f}".format(u2)%(k-1))
            if abs(u[0,N]-beta)<=tol:
                return t1,u[0,:]
                break
            else:
                if method=="Newthon":
                    u2+=-(u[0,N]-beta)/(v[0,N])
                elif method=="Newthon3_1":
                    u11=u2-(u[0,N]-beta)/(v[0,N])
                    t1,u10,v10=Method_disparo.RungeKutta_4(self,a,b,u1,u11,N)
                    u2=u2-2*(u[0,N]-beta)/(v[0,N]+v10[0,N])
                elif method=="Newthon3_2":
                    u11=u2-(u[0,N]-beta)/(v[0,N])
                    t1,u10,v10=Method_disparo.RungeKutta_4(self,a,b,u1,u11,N)
                    u2=u2-(u[0,N]+u10[0,N]-2*beta)/(v[0,N])
                elif method=="Newthon3_3":
```

```
                    u11=u2-(u[0,N]-beta)/(v[0,N])
                    t1,u10,v10=Method_disparo.RungeKutta_4(self,a,b,u1,u11,N)
                    u2=u11-(u10[0,N]-beta)/(v[0,N])
            k+=1
        else:
            print("No es suficiente",M)
class Error(Method_disparo):
    def __init__(self,f,g):
        super().__init__(f)
        self.g=g
    def error_aprox(self,method,a,b,alpha,beta,N,M,tol):
        t1,x_1=Method_disparo(f).Aprox_numeric(method,a,b,alpha,beta,N,M,tol)
        g_np=lambdify(t,g,modules="numpy")
        x_exact=g_np(t1)
        return t1,abs(x_1-x_exact)
```

Apéndice B. Declaración de sentencias.
Ejemplo 1:

```
import numpy as np
from sympy import*
t,x,x1=symbols("t x x1")
f=-3*(x**2)*x1/t
Aprox=Method_disparo(f)
Result1=[Aprox.Aprox_numeric("Newthon",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40)]
Result2=[Aprox.Aprox_numeric("Newthon3_1",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40)]
Result3=[Aprox.Aprox_numeric("Newthon3_2",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40)]
Result4=[Aprox.Aprox_numeric("Newthon3_3",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40)]
Result1, Result2, Result3, Result4
====================================
t,x,x1=symbols("t x x1")
f=-3*(x**2)*x1/t
g=t/(t**2+1)**0.5
Result=Error(f,g)
Error1=Result.error_aprox("Newthon",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40) ]
Error2=[Result.error_aprox("Newthon3_1",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40) ]
Error3=[Result.error_aprox("Newthon3_2",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40) ]
Error4=[Result.error_aprox("Newthon3_3",1,2,1/np.sqrt(2),2/np.sqrt(5),i,400,1e-6) for
i in (10,20,30,40) ]
Error1, Error2, Error3, Error4
```

Ejemplo 2:

```
from sympy import*
import numpy as np
t,x,x1=symbols("t x x1")
f=2*x**3-6*x-2*t**3
Aprox=Method_disparo(f)
Resultado1=[Aprox.Aprox_numeric("Newthon",1,2,2,2.5,k,400e+9,1e-6) for
k in (10,20,30,40)]
Resultado2=[Aprox.Aprox_numeric("Newthon3_1",1,2,2,2.5,k,400e+9,1e-6) for
```

```
k in (10,20)]
Resultado3=[Aprox.Aprox_numeric("Newthon3_2",1,2,2,2.5,k,400,1e-6) for
k in (10,20,30,40)]
Resultado4=[Aprox.Aprox_numeric("Newthon3_3",1,2,2,2.5,k,400,1e-6) for
k in (10,20,30,40)]
==================================
g=(t**2+1)/t
Result=Error(f,g)
Errorej1=[Result.error_aprox("Newthon",1,2,2,2.5,k,40000,1e-6) for k in (10,20,30,40)]
Errorej2=[Result.error_aprox("Newthon3_1",1,2,2,2.5,k,40000,1e-6) for k in (10,20)]
Errorej3=[Result.error_aprox("Newthon3_2",1,2,2,2.5,k,40000,1e-6) for k in (10,20,30,40)]
Errorej4=[Result.error_aprox("Newthon3_3",1,2,2,2.5,k,40000,1e-6) for k in (10,20,30,40)]
Errorej1,Errorej2,Errorej3, Errorej4
```