

ANÁLISIS NUMÉRICO PARA INGENIERÍA



SOLNE

PYTHON

## Manual de Usuario

*Arturo Córdoba Villalobos*

*Fabián González Araya*

*Gustavo Segura Umaña*

*Joseph Vargas Blanco*

# Tabla de Contenidos

<b>1</b>	<b>¿Qué es SolNE?</b>	<b>2</b>
<b>2</b>	<b>Instalación</b>	<b>2</b>
2.1	Requisitos . . . . .	2
2.1.1	Python3 . . . . .	2
2.1.2	Python3 Pip . . . . .	2
2.1.3	SymPy . . . . .	2
2.1.4	NumPy . . . . .	2
2.1.5	Matplotlib . . . . .	3
2.2	Instalación SolNE . . . . .	3
<b>3</b>	<b>Cómo utilizar SolNE</b>	<b>3</b>
3.1	Función sne_ud_1 . . . . .	4
3.2	Función sne_ud_2 . . . . .	4
3.3	Función sne_ud_3 . . . . .	5
3.4	Función sne_ud_4 . . . . .	6
3.5	Función sne_ud_5 . . . . .	6
3.6	Función sne_ud_6 . . . . .	7
3.7	Función sne_fd_1 . . . . .	7
3.8	Función sne_fd_2 . . . . .	8
3.9	Función sne_fd_3 . . . . .	9
3.10	Función sne_fd_4 . . . . .	9
3.11	Función sne_fd_5 . . . . .	10
3.12	Función sne_fd_6 . . . . .	10
<b>4</b>	<b>Referencias Bibliográficas</b>	<b>12</b>

# 1 ¿Qué es SolNE?

SolNE es un paquete cuyo objetivo es la obtención de una aproximación a la solución de una ecuación no lineal de la forma  $f(x) = 0$ . Es una biblioteca para Python que provee doce métodos iterativos diferentes, seis que emplean el uso de derivadas y seis que no utilizan derivadas. Aquellos métodos que incluyen el cálculo de derivadas empiezan con el prefijo *sne\_ud\_#* (*solving nonlinear equation - using derivative*), y los que no empiezan con el prefijo *sne\_fg\_#* (*solving nonlinear equation - free derivative*), donde # es el número de método.

## 2 Instalación

En este manual se muestran los pasos para la instalación de SolNE en Ubuntu. En cada sección se muestran los comandos que deben ser ejecutados en la consola para la instalación de cada dependencia específica. Los comandos se encuentran en un orden específico de ejecución, por lo que se recomienda empezar por la sección 2.1.1.

### 2.1 Requisitos

Este paquete utiliza SymPy para el uso de matemática simbólica, NumPy para verificar que el resultado no sea NaN o infinito, y Matplotlib para realizar las gráficas de error.

#### 2.1.1 Python3

```
$ sudo apt-get install python3
```

#### 2.1.2 Python3 Pip

```
$ sudo apt-get install python3-pip
```

#### 2.1.3 SymPy

```
$ sudo pip3 install sympy
```

#### 2.1.4 NumPy

```
$ sudo pip3 install numpy
```

### 2.1.5 Matplotlib

```
$ sudo pip3 install matplotlib
```

## 2.2 Instalación SolNE

1. Descomprima el archivo llamado SolNE.zip
2. Ingrese en la captera donde se encuentran los archivos setup.py y test.py
3. Abra esta dirección en la consola.
4. Ejecute el siguiente comando para verificar que su dispositivo cuenta con las dependencias necesarias:

```
$ python3 test.py
```

5. Si las pruebas no se ejecutan con éxito, instale las dependencias faltantes. Si se ejecutan correctamente ingrese el comando `exit()` para cerrar python.
6. Ejecute el siguiente comando en la consola:

```
$ pip3 install .
```

7. Una vez finalizada la instalación puede cerrar la consola.

## 3 Cómo utilizar SolNE

Una vez realizada la instalación, basta con importar el paquete de la siguiente manera:

```
>>> import SolNE
```

Como se indicó anteriormente, aquellos métodos que incluyen el cálculo de derivadas empiezan con el prefijo *sne\_ud\_#* (*solving nonlinear equation - using derivative*), y los que no empiezan con el prefijo *sne\_fg\_#* (*solving nonlinear equation - free derivative*), donde # es el número de método. Todas las funciones ingresadas deben estar en términos de  $x$ .

De manera general, cada método recibe los siguientes argumentos:

- *str\_funcion*: string que representa la función  $f(x)$ . Se debe utilizar la sintaxis definida por Python para el uso de operadores matemáticos, como el de la suma (+), resta(-), multiplicación (\*), división (/) y exponente (\*\*). En el caso de funciones trigonométricas o exponenciales ir al siguiente enlace para ver la sintaxis: <https://docs.python.org/3/library/math.html>.
- Valores iniciales  $(x_0, x_1, \dots, x_n)$ , los cuales son necesarios para que el método funcione.

- Tolerancia  $tol > 0$ , determina el criterio de parada para cada método iterativo, el cual está definido por  $|f(x_k)| < tol$ , donde  $x_k$  es la  $k$ -ésima iteración que aproxima la raíz de la ecuación  $f(x) = 0$ .
- *graph*: este parámetro permite mostrar la gráfica de iteraciones ( $k$ ) versus errores ( $|f(x_k)|$ ) del método iterativo. Si *graf* = 0, entonces no se mostrará la gráfica. Si *graf* = 1 o se omite el parámetro *graf*, entonces sí se mostrará la gráfica.

De manera general, cada método retorna una lista con dos elementos, el primero corresponde a  $x_{approx}$  calculado, y el segundo a la cantidad de iteraciones *iter*, adicionalmente, se mostrará o no la gráfica de error según el valor de *graph*. Todos estos valores se explican a continuación:

- $x_{approx}$ , el cual es la aproximación a la solución de la ecuación  $f(x) = 0$ .
- *iter*, el cual representa el número de iteraciones que se utilizaron para aproximar el cero de la función con una tolerancia *tol*.
- Si *graf* = 1 o se omite el parámetro *graf*, entonces se mostrará la gráfica de iteraciones ( $k$ ) versus errores ( $|f(x)|$ ) del método iterativo.

Cada método se detiene si el denominador respectivo es cero (para evitar la división entre cero), si el valor  $x_{approx}$  toma un valor infinito o NaN en alguna de las iteraciones. En estos casos, cada método retorna los resultados obtenidos hasta ese momento.

### 3.1 Función sne\_ud\_1

Esta función implementa el método iterativo de Weerakoon-Fernando, el cual fue tomado de [1]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_1(funcion1, 5 / 7, 10 ** -5, 1)
[0.5149332928105104, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_1(funcion2, 3 / 4, 10 ** -5, 0)
[1.1461932206137209, 3]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_1(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851157170514, 2]
```

### 3.2 Función sne\_ud\_2

Esta función implementa el método iterativo de Chun-Kim, el cual fue tomado de [1]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A

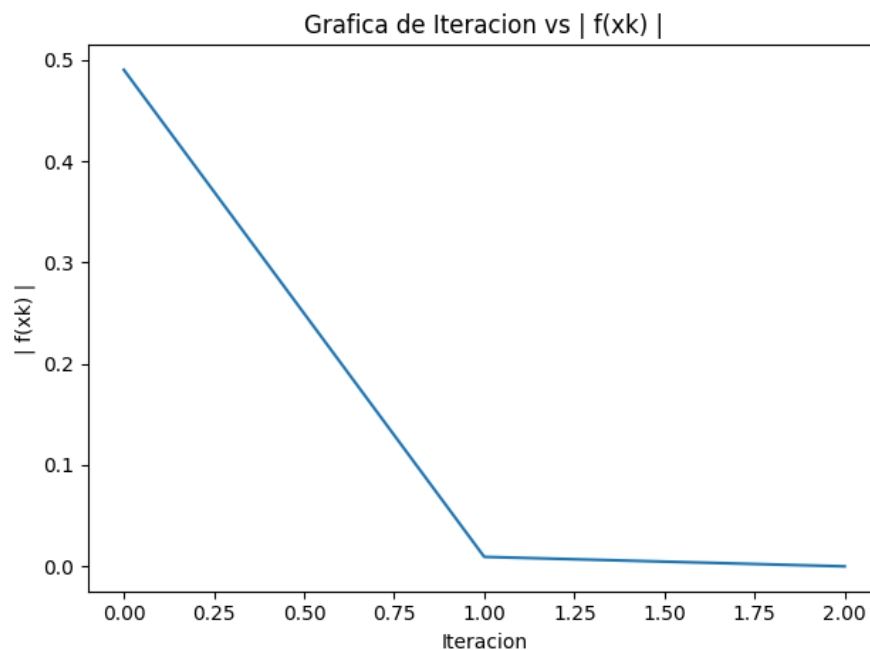


Figura 1: Gráfica obtenida al evaluar la funcion1 utilizando sne\_ud\_1.

continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_2(funcion1, 5 / 7, 10 ** -5, 0)
[0.5149331280095247, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_2(funcion2, 3 / 4, 10 ** -5, 0)
[1.1461932206205268, 4]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_2(funcion3, 1 / 5, 10 ** -5, 0)
[0.739081081760118, 2]
```

### 3.3 Función sne\_ud\_3

Esta función implementa el método iterativo de Özban-Homeier, el cual fue tomado de [1]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_3(funcion1, 5 / 7, 10 ** -5, 0)
[0.5149340851889033, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_3(funcion2, 3 / 4, 10 ** -5, 0)
[1.146193224080175, 2]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_3(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851092490082, 2]
```

### 3.4 Función sne\_ud\_4

Esta función implementa el método iterativo de Darvishi-Barati, el cual fue tomado de [1]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_4(funcion1, 5 / 7, 10 ** -5, 0)
[0.514935456178876, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_4(funcion2, 3 / 4, 10 ** -5, 0)
[1.146190173485739, 3]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_4(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851332151605, 3]
```

### 3.5 Función sne\_ud\_5

Esta función implementa el método de Ostrowski de cuarto orden, el cual fue tomado de [2]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_5(funcion1, 5 / 7, 10 ** -5, 0)
[0.51493326456136, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_5(funcion2, 3 / 4, 10 ** -5, 0)
[1.1461932226178897, 2]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_5(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851332264071, 2]
```

### 3.6 Función sne\_ud\_6

Esta función implementa el método de Traub, el cual fue tomado de [3]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_6(funcion1, 5 / 7, 10 ** -5, 0)
[0.514935456178876, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_ud_6(funcion2, 3 / 4, 10 ** -5, 0)
[1.1461901734857387, 3]

>>> funcion3 = 'cos(x) - x'
>>> sne_ud_6(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851332151605, 3]
```

### 3.7 Función sne\_fd\_1

Esta función implementa el método iterativo de Steffensen, el cual fue tomado de [4]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.



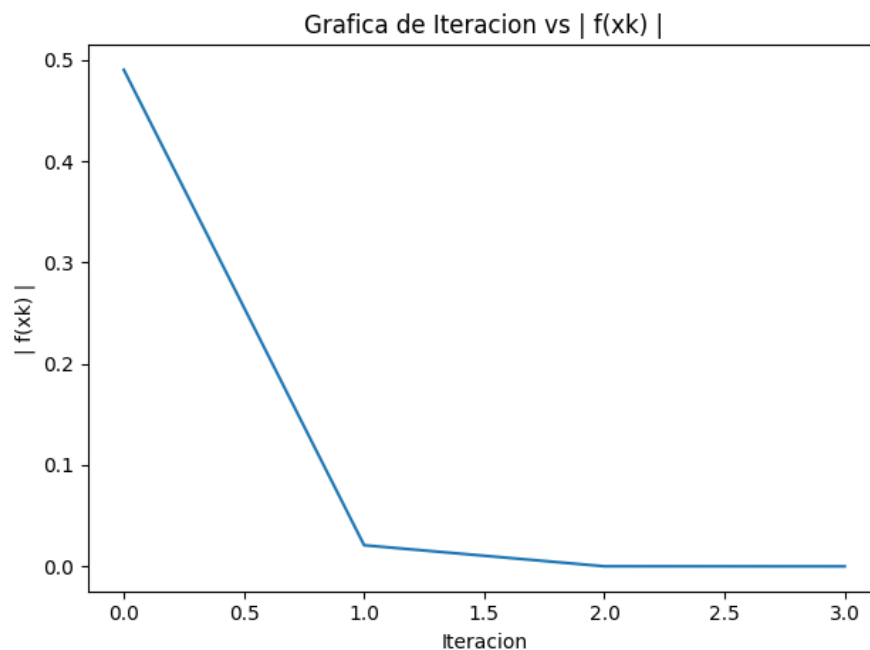


Figura 2: Gráfica obtenida al evaluar la funcion1 utilizando sne\_fd\_1.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_1(funcion1, 5 / 7, 10 ** -5, 1)
[0.5149332651840688, 3]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_1(funcion2, 3 / 4, 10 ** -5, 0)
[1.146196310975045, 9]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_1(funcion3, 1 / 5, 10 ** -5, 0)
[0.739085129255462, 3]
```

### 3.8 Función sne\_fd\_2

Esta función implementa el método iterativo M8, el cual fue tomado de [5]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con dos valores iniciales  $x_0$  y  $y$ , con  $y \neq 0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_2(funcion1, 5 / 7, 1, 10 ** -5, 0)
[0.5149332646611062, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_2(funcion2, 3 / 4, 1, 10 ** -5, 0)
[1.146193220622202, 2]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_2(funcion3, 1 / 5, 1, 10 ** -5, 0)
[0.7390851332151646, 2]
```

### 3.9 Función sne\_fd\_3

Esta función implementa el método IODF (*Improved Ostrowski's method free from derivatives*), el cual fue tomado de [6]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un solo valor inicial  $x_0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_3(funcion1, 5 / 7, 10 ** -5, 0)
[0.5149332646610815, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_3(funcion2, 3 / 4, 10 ** -5, 0)
[1.1461932206205827, 2]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_3(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851332151607, 2]
```

### 3.10 Función sne\_fd\_4

Esta función implementa el método iterativo M4, el cual fue tomado de [5]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con dos valores iniciales  $x_0$  y  $y$ , con  $y \neq 0$ . A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_4(funcion1, 5 / 7, 1, 10 ** -5, 0)
[0.5149332646603385, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_4(funcion2, 3 / 4, 1, 10 ** -5, 0)
[1.1461932206206447, 3]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_4(funcion3, 1 / 5, 1, 10 ** -5, 0)
[0.7390851332151617, 2]
```

### 3.11 Función sne\_fd\_5

Esta función implementa el método mejorado de Müller y Bisección, el cual fue tomado de [7]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con el límite izquierdo del intervalo ( $a$ ), y el límite derecho ( $b$ ). A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_5(funcion1, 0, 5, 10 ** -5, 0)
[0.5149301973028021, 7]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_5(funcion2, -5, 5, 10 ** -5, 0)
[1.1461885868072625, 128]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_5(funcion3, -5, 5, 10 ** -5, 0)
[0.7390794200572189, 24]
```

### 3.12 Función sne\_fd\_6

Esta función implementa el método paramétrico de Ostrowski-Chun, el cual fue tomado de [8]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con un valor inicial  $x_0$ , y las constantes  $a_1$ ,  $a_2$ ,  $b_1$ ,  $b_2$ ,  $alpha$ , las cuales tienen que ser un número real diferente de cero. A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_6(funcion1, 3 / 4, 1, 2, 0.3, 2, 0.7, 10 ** -5, 0)
[0.5149306605677052, 22]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_6(funcion2, 3 / 4, 1, 2, 0.3, 2, 0.7, 10 ** -5, 0)
[1.1461947355781947, 7]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_6(funcion3, 3 / 4, 1, 2, 0.3, 2, 0.7, 10 ** -5, 0)
[0.7390840785564095, 5]
```

## 4 Referencias Bibliográficas

- [1] R. Kiran, L. Li, and K. Khandelwal, “Performance of cubic convergent methods for implementing nonlinear constitutive models,” *Computers & Structures*, vol. 156, pp. 83–100, 2015.
- [2] R. Behl, A. Cordero, S. S. Motsa, and J. R. Torregrosa, “Construction of fourth-order optimal families of iterative methods and their dynamics,” *Applied Mathematics and Computation*, vol. 271, pp. 89–101, 2015.
- [3] D. Budzko, A. Cordero, and J. R. Torregrosa, “A new family of iterative methods widening areas of convergence,” *Applied Mathematics and Computation*, vol. 252, pp. 405–417, 2015.
- [4] A. Cordero and J. R. Torregrosa, “A class of steffensen type methods with optimal order of convergence,” *Applied Mathematics and Computation*, vol. 217, no. 19, pp. 7653–7659, 2011.
- [5] P. Bakhtiari, A. Cordero, T. Lotfi, K. Mahdiani, and J. R. Torregrosa, “Widening basins of attraction of optimal iterative methods,” *Nonlinear Dynamics*, vol. 87, no. 2, pp. 913–938, 2017.
- [6] A. Cordero, J. L. Hueso, E. Martínez, and J. R. Torregrosa, “Steffensen type methods for solving nonlinear equations,” *Journal of computational and applied mathematics*, vol. 236, no. 12, pp. 3058–3064, 2012.
- [7] X. Wu, “Improved muller method and bisection method with global and asymptotic superlinear convergence of both point and interval for solving nonlinear equations,” *Applied mathematics and computation*, vol. 166, no. 2, pp. 299–311, 2005.
- [8] A. Cordero, J. G. Maimó, J. R. Torregrosa, and M. P. Vassileva, “Solving nonlinear problems by ostrowski–chun type parametric families,” *Journal of Mathematical Chemistry*, vol. 53, no. 1, pp. 430–449, 2015.