Análisis Numérico Para Ingeniería

SolNE

Python

Manual de Usuario

Arturo Córdoba Villalobos

Fabián González Araya

Gustavo Segura Umaña

Joseph Vargas Blanco

Tabla de Contenidos

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

3 Cómo utilizar SolNE

*********Indicar como importar el paquete**********

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.

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 $(x_0, x_1, ..., x_n)$, 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 los siguiente:

- x_{aprox}, 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_{aprox} toma un valor infito o NaN en alguna de las iteraciones. En estos casos, cada método retorna los resultados obtenidos hasta ese momento.

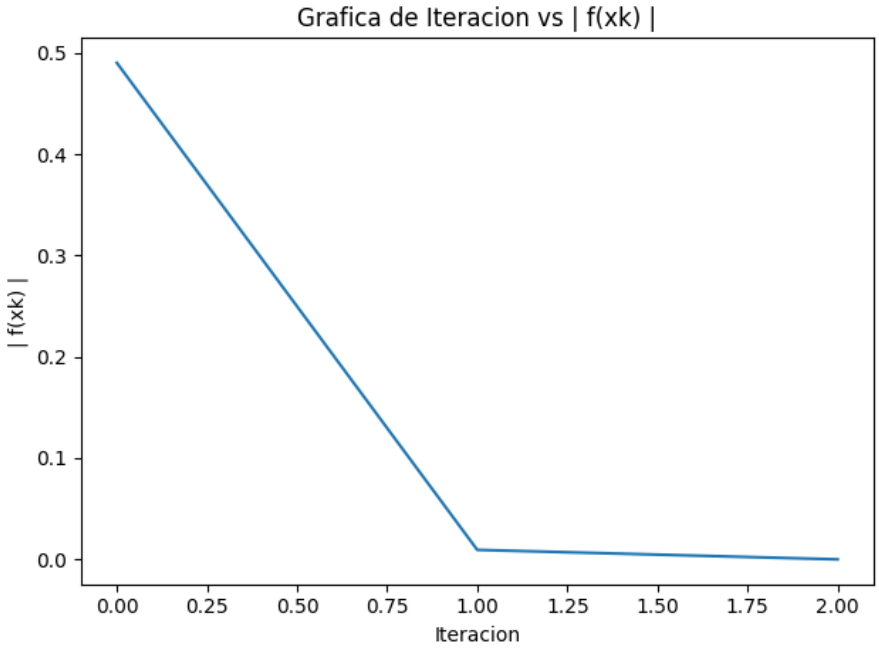

Figura 1: Gráfica obtenida al evualuar la funcion1 utlizando sne_ud_1.

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

continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_ud_2(funcion1, 5 / 7, 10 ** -5, 1)
[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)
[0.7390810817601l8, 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, 1)
[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_fd_1

Esta función implementa el método iterativo de Steffensen, 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.

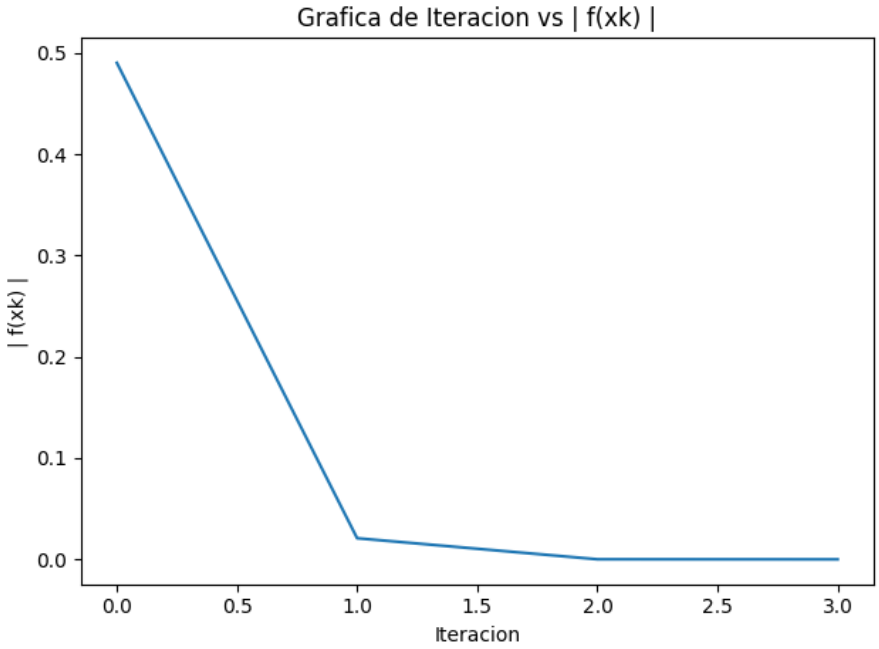

Figura 2: Gráfica obtenida al evualuar la funcion1 utlizando sne_fd_1.

```
>>> 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]
```

3.5 Función sne_fd_2

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_fd_2(funcion1, 5 / 7, 10 ** -5, 1)
[0.514935456178876, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_2(funcion2, 3 / 4, 10 ** -5, 0)
[1.146190173485739, 3]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_2(funcion3, 1 / 5, 10 ** -5, 0)
[0.7390851332151605, 3]
```

3.6 Función sne_fd_3

Esta función implementa el método iterativo M8, el cual fue tomado de [3]. Esta función recibe los argumentos explicados anteriormente al inicio de la sección 3, con dos valores iniciales x_0 y x_1. A continuación se muestran algunos ejemplos de uso.

```
>>> funcion1 = 'cos(2 * x) ** 2 - x ** 2'
>>> sne_fd_3(funcion1, 5 / 7, 1, 10 ** -5, 0)
[0.5149332646611062, 2]

>>> funcion2 = 'exp(x) - x - 2'
>>> sne_fd_3(funcion2, 3 / 4, 1, 10 ** -5, 0)
[1.146193220622202, 2]

>>> funcion3 = 'cos(x) - x'
>>> sne_fd_3(funcion3, 1 / 5, 1, 10 ** -5, 0)
[0.7390851332151646, 2]
```

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] 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.

[3] 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.