Análisis Numérico Para Ingeniería

SolNE

Octave

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

2.1.1 Octave

La última versión se puede descargar desde https://www.octave.org o buscando "GNU Octave" en la tienda de software de Ubuntu. Se debe instalar con las opciones predeterminadas.

2.1.2 Python3

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

2.1.3 Python3 Pip

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

2.1.4 SymPy

```
$ sudo pip3 install sympy
```

Abrir GNU Octave y desde la línea de comandos de Octave escribir:

```
>> pkg install -forge symbolic
```

Cargue el paquete simbólico con el comando Octave:

```
>> pkg load symbolic
```

2.2 Instalación SolNE

1. Descomprima el archivo llamado SolNE.zip
2. Ingrese en la captera donde se encuentra el archivo SolNE.tar.gz
3. Abra esta dirección en la consola.
4. Ejecute octave-cli al estar en esta carpeta en la consola:

```
$ octave-cli
```

5. En la línea de comandos de Octave, ejecute el siguiente comando para instalar el paquete:

```
$ pkg install SolNE.tar.gz
```

6. Si no se muestra ningún mensaje la instalación ha sido completada satisfactoriamente.

3 Cómo utilizar SolNE

Una vez realizada la instalación, ya sea desde octave-cli o desde la versión gráfica del programa, ejecute en la línea de comandos lo siguiente para cargar el paquete:

```
>> pkg load 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 $(x_0, x_1, ..., x_n)$, 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_{aprox} 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_{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.

3.1 Función sne_ud_1

Esta función implementa el método iterativo de Weerakoon-Fernando, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_1(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 2

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_ud_1(funcion2, 3/4, 10^-5);
x_aprox2 = 1.3678
iter2 = 4
```

3.2 Función sne_ud_2

Esta función implementa el método iterativo de Chun-Kim, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_2(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 2

>> funcion2 = 'exp(x) - x - 2';
>> [x_aprox2, iter2] = sne_ud_2(funcion2, 3/4, 10^-5);
x_aprox2 = 1.1462
iter2 = 4
```

3.3 Función sne_ud_3

Esta función implementa el método iterativo de Özban-Homeier, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_3(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 3

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_ud_3(funcion2, 3/4, 10^-5);
x_aprox2 = 1.3678
iter2 = 3
```

3.4 Función sne_ud_4

Esta función implementa el método iterativo de Darvishi-Barati, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_4(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 3

>> funcion2 = 'exp(x) - x - 2';
>> [x_aprox2, iter2] = sne_ud_4(funcion2, 3/4, 10^-5);
x_aprox2 = 1.1462
iter2 = 3
```

3.5 Función sne_ud_5

Esta función implementa el método de Ostrowski de cuarto orden, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_5(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 2

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_ud_5(funcion2, 3/4, 10^-5);
x_aprox2 = 1.1462
iter2 = 3
```

3.6 Función sne_ud_6

Esta función implementa el método de Traub, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_ud_6(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 3

>> funcion2 = 'exp(x) - x - 2';
>> [x_aprox2, iter2] = sne_ud_6(funcion2, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 3
```

3.7 Función sne_fd_1

Esta función implementa el método iterativo de Steffensen, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_fd_1(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 3

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_fd_1(funcion2, 3/4, 10^-5);
x_aprox2 = 1.3678
iter2 = 4
```

3.8 Función sne_fd_2

Esta función implementa el método iterativo M8, el cual fue tomado de [**?**]. 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';
>> [x_aprox1, iter1] = sne_fd_2(funcion1, 3/4, 1, 10^-5);
x_aprox1 = 0.51493
iter1 = 2

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_fd_2(funcion2, 3/4, 1, 10^-5);
x_aprox2 = 1.3678
iter2 = 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 [**?**]. 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';
>> [x_aprox1, iter1] = sne_fd_3(funcion1, 3/4, 10^-5);
x_aprox1 = 0.51493
iter1 = 2

>> funcion2 = 'exp(x) - x^3 - x';
>> [x_aprox2, iter2] = sne_fd_3(funcion2, 3/4, 10^-5);
x_aprox2 = 1.3678
iter2 = 2
```

4 Referencias Bibliográficas