Analisis Numerico - Reto 1

Julian Carrillo Chiquisa
Cristian Camilo Contreras Borja
Kevin Andrés Garzón Ospina
Jenifer Medina Yepez
Oscar Andrés Pacheco Turizo
Pontificia Universidad Javeriana
Bogotá D.C

13 de marzo de 2021

Resumen

En el siguiente documento presentara la documentacion acerca del Reto numero 1 de la materia analisis de algortimos en el cual se presentaran 3 puntos importantes el primero es la aplicacion del algotimo de brent, el segundo es la interseccion entre curvas y finalmente se presentaran las librerias NumPy y SciPy las cuales pertenecen al lenguaje de Python.

1. Algoritmo de Brent

Aplicar el algoritmo de Brent para encontrar las raices del polinomio, con un error menor de 2 elevado a la menos 50:

$$f(x) = x^3 - 2x^2 + 4x/3 - 8/27$$

1.1. Metodo Secante

Es un método para encontrar la raíz de una función de forma iterativa. Se considera una variación del método de Newton-Raphson donde en vez de calcular la derivada de la función en el punto de estudio, teniendo en mente la definición de derivada, se aproxima la pendiente a la recta que une la función evaluada en el punto de estudio y en el punto de la iteración anterior.

Implementacion del Metodo Secante :

```
def secant(f, x1, x2, tol):
    error = 1e3
    acum = []
    iteraciones = []
    n = 0
    x3 = 0
    while error > tol:
        x3 = x1 - ((x2 - x1) / (f(x2) - f(x1))) * f(x1)
        x1 = x2
        x2 = x3
        error = abs(f(x3))
        acum.append(error)
        #print(error)
        iteraciones.append(n)
        n += 1
    print("Solucion Aproximada: {:.20f}".format(x3))
    print("Numero de iteraciones: {:d}".format(n))

    plt.plot(iteraciones,acum)
    plt.xlabel('Iteaciones')
    plt.ylabel('Error')
    plt.title('METODO DE SECANTE CON ' + str(tol))
    plt.show()
```

Figura 1: Implementacion Metodo Secante.

Diagrama de flujo de la Implementacion:

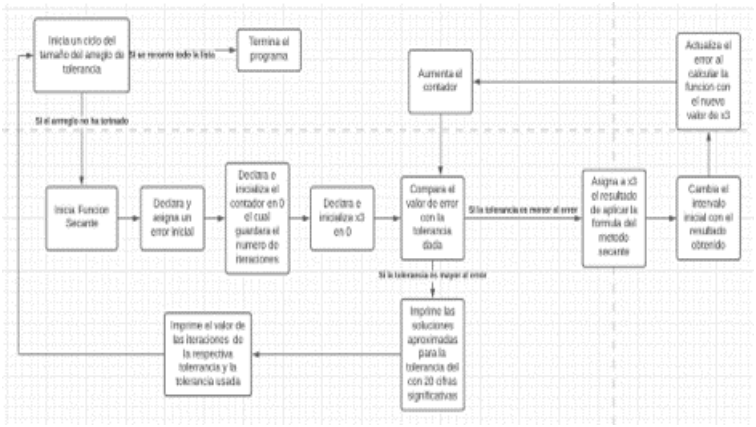

Figura 2: Diagrama de Flujo Metodo Secante.

La solucion obtenida mediante este metodo aplicado a la ecuacion dada es:

```
Solucion Aproximada: 0.66666833071300513058
Numero de iteraciones: 42
```

Figura 3: Solucion Ejercicio 1 (Metodo Secante).

Graficas:

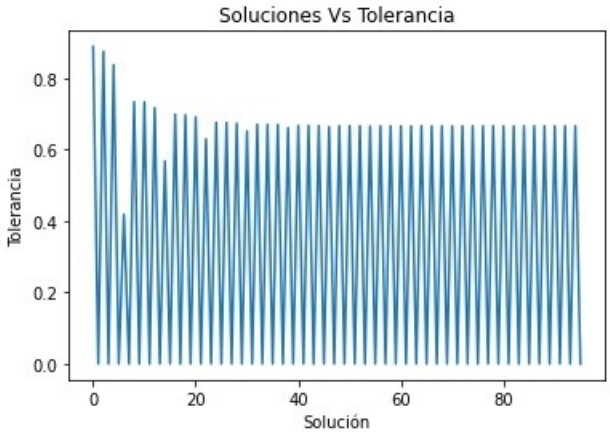

Figura 4: Grafica entre solcuiones y tolerancias.

Figura 5: Grafica numero de iteraciones frente a solucion de iteracion.

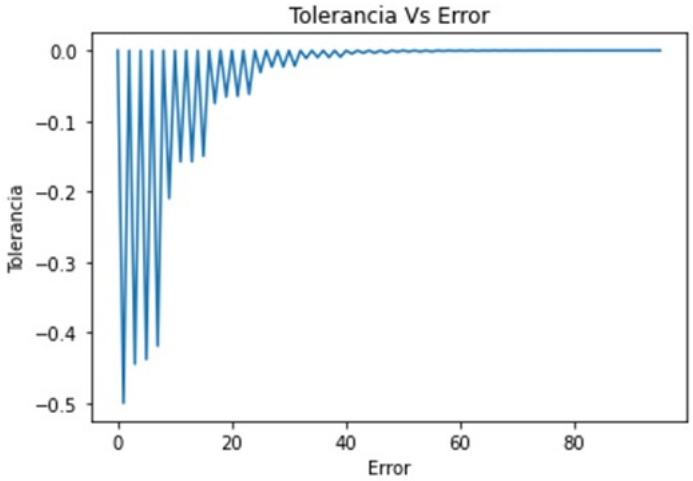

Figura 6: Grafica tolerancia contra error.

1.2. Algoritmo Brent

El método de Brent es un algoritmo híbrido de búsqueda de raíces que combina el método de bisección , secante y la interpolación cuadrática inversa, este tiene la confiabilidad de la bisección, pero tambien puede ser tan rápido como otros metodos.

Implementacion Algoritmo de Brent :

```python
def Brent(a,b,machep,t,f):
    sa = a
    sb = b
    fa = f(sa)
    fb = f(sb)
    c = sa
    fc = fa
    e = sb-sa
    d = e
    k=0
    while(True):
        if (abs(fc)<abs(fb)):
            sa=sb;sb=c ;c = sa;fa = fb;fb = fc;fc=fa
        tol = 2.0*machep*abs(sb)+t
        m= 0.5*(c-sb)
        if(abs(m)<=tol or fb == 0 ):break
        if (abs(e)<tol or abs(fa)<=abs(fb)):
            e=m;d=e
        else:
            s = fb/fa
            if (sa==c):
                p = 2.0*m*s; q = 1.0 -s
            else:
                q = fa/fc;r = fb/fc
                p = s*(2.0*m*1.0*(q-r)-(sb-sa)*(r-1.0))
                q = (q-1.0)*(r-1.0)*(s-1.0)
            if (0<p):
                q = -q
            else:
                p = -p
            s=e;e=d
            if((2.0*p<3.0*m*q-abs(tol*q))and (p<abs(0.5*s*q))):
                d = p/q
            else:
                e = m; d = e
        sa = sb; fa=fb
        if(tol<abs(d)):
            sb = sb +d
        elif(0<m):
            sb = sb+tol
        else:
            sb = sb-tol
        fb = f(sb)
        if((0.0<fb and 0.0<fc) or (fb<= 0.0 and fc <= 0.0) ):
            c = sa; fc = fa; e = sb-sa; d=e
        k+=1

    value = sb
    print("Numero de iteraciones: {:d}".format(k))
    return value
```

Figura 7: Implementacion Algortimo de Brent.

Diagrama de flujo de la Implementacion:

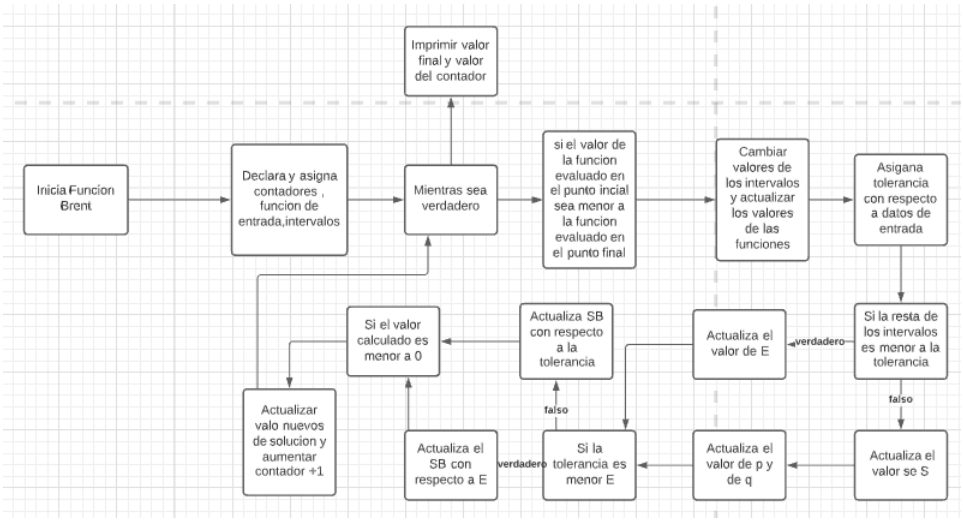

Figura 8: Diagrama de Flujo Algoritmo de Brent.

La solucion obtenida mediante este metodo aplicado a la ecuacion dada es:

```
Numero de iteraciones: 48
Solucion Aproximada: 0.66666848274847190226
```

Figura 9: Solucion Ejercicio 1 (Algoritmo de Brent).

Comparacion de las dos soluciones obtenidas:

Tras haber aplicado dos formas de solucion de raices podemos observar como el numero de iteraciones con respecto al algoritmo de Brent es mayor al del metodo secante debido a que este algoritmo como ya se habia dicho previamente hace una combinacion del funcionamiento del secante, en cuanto a la significancia manejada fue la misma en los dos casos y los resultados tuvieron similitudes hasta la septima cifra significativa en donde el metodo secante cuenta con mayor acercamiento a la respuesta esperada.

2. Intersección entre curvas

Aplicar la tecnica de aproximacion a la raiz, que desarrollo en el trabajo en grupo, para encontrar la interseccion entre:

$$x^2 + xy = 10; y + 3xy^2 = 57$$

Tras aplicar una version mejorada del metodo secante usado anteriormente se llego a esta implementacion:

```python
import matplotlib.pyplot as plt
import numpy as np

def ciclo(f,x1,x2,aux):
    for i in aux:
        secant(f, x1, x2, i)

def secant(f, x1, x2, tol):
    error = 1e3
    acum = []
    iteraciones = []
    n = 0
    x3 = 0
    while error > tol:
        x3 = x1 - ((x2 - x1) / (f(x2) - f(x1))) * f(x1)
        x1 = x2
        x2 = x3
        error = abs(f(x3))
        acum.append(error)
        #print(error)
        iteraciones.append(n)
        n += 1
    print("Solucion Aproximada: {:.20f}".format(x3))
    print("Numero de iteraciones: {:d}".format(n))

    plt.plot(iteraciones,acum)
    plt.xlabel('Iteaciones')
    plt.ylabel('Error')
    plt.title('METODO DE SECANTE CON ' + str(tol))
    plt.show()

def secantXY(f, g, x1, x2, y1, y2, tol):
    arreglo = []
    errorx = 1e3
    errory = 1e3
    acum = []
    iteraciones = []
    n = 0
    x3 = 0
    y3 = 0
    while errorx > tol and errory > tol:
        x3 = x1 - ((x2 - x1) / (f(x2,y2) - f(x1,y1))) * f(x1,y1)
        y3 = y1 - ((y2 - y1) / (g(x2,y2) - g(x1,y1))) * g(x1,y1)
        x1 = x2
        x2 = x3
        y1 = y2
        y2 = y3
        errorx = abs(f(x3,y3))
        errory = abs(g(x3,y3))
        acum.append(errorx)
        #print(error)
        iteraciones.append(n)
        n += 1
        arreglo = np.append(arreglo,[n,y3])
    print("Solucion AproximadaX: {:.20f}".format(x3))
    print("Solucion AproximadaY: {:.20f}".format(y3))
    print("Numero de iteraciones: {:d}".format(n))
    print(arreglo)

    plt.plot(arreglo)
    plt.xlabel('Iteraciones')
    plt.ylabel('Solucion en y')
    plt.title('Metodo de la secante con ' + str(tol))
    plt.show()
```

Figura 10: Implementacion Algoritmo para la Interseccion entre curvas.

```
#---------------------------------------------------------------
Pruebas
#---------------------------------------------------------------#

def fun(x):
    return x**3 - (2*x**2) + (4*x)/3 - (8/27)

def fun2(x,y):
    return x**2+x*y-10

def fun3(x,y):
    return y+3*x*y**2-57

secantXY(fun2, fun3, 1.5, 10, 3.5, 10, 10e-4)
```

Figura 11: Implementacion Algoritmo para la Interseccion entre curvas parte 2.

Diagrama de flujo de la implementacion

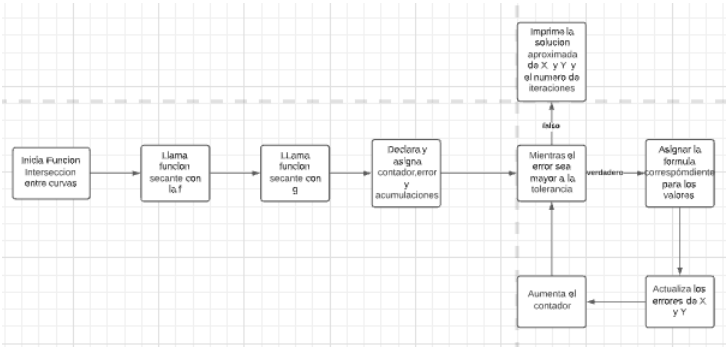

Figura 12: Diagrama de flujo del Algoritmo para la Interseccion entre curvas.

La solucion obtenida mediante el algortimo aplicado a las ecuaciones dada es:

```
Solucion AproximadaX: 1.99849348385561542685
Solucion AproximadaY: 3.00494621568519493238
Numero de iteraciones: 19
```

Figura 13: Solucion Punto 2

Graficas:

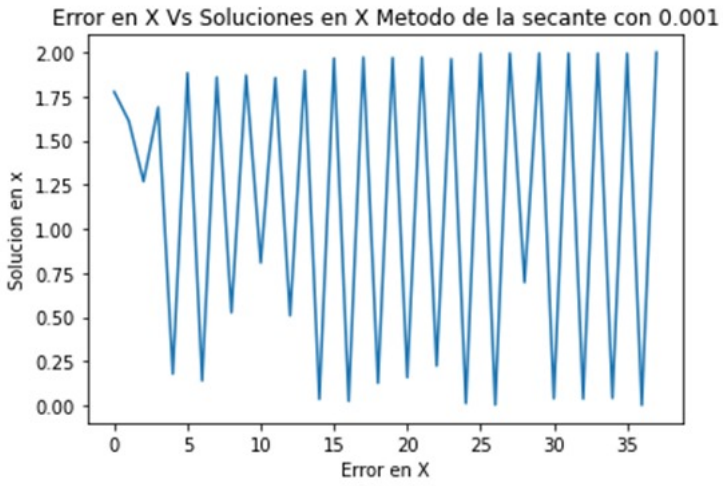

Figura 14: Grafica de error en x contra lsolucuion en x .

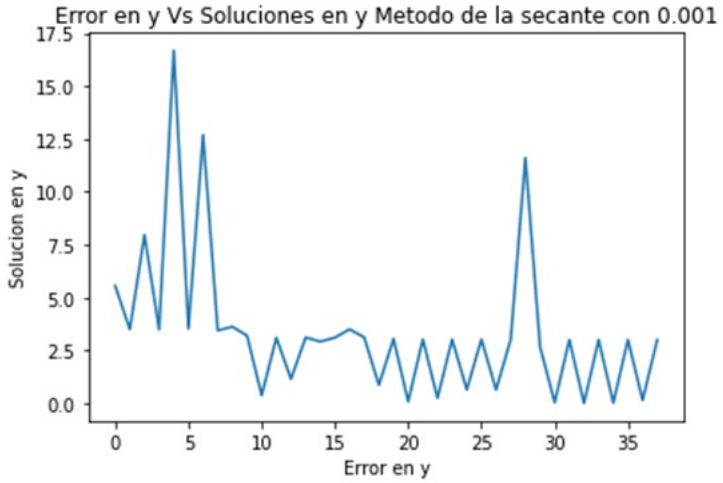

Figura 15: Grafica error en y contra solucion en y.

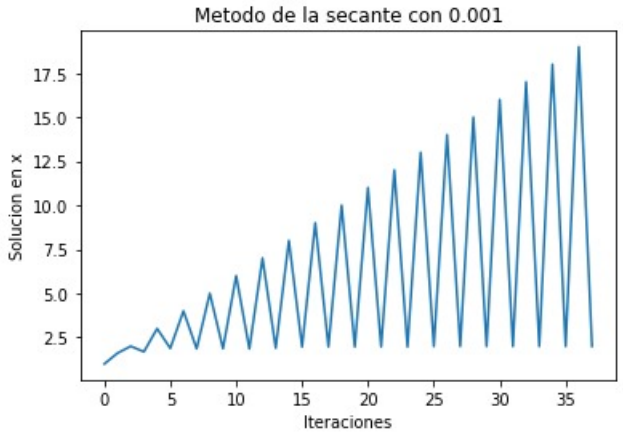

Figura 16: Grafica de iteraciones contra la solucion en x.

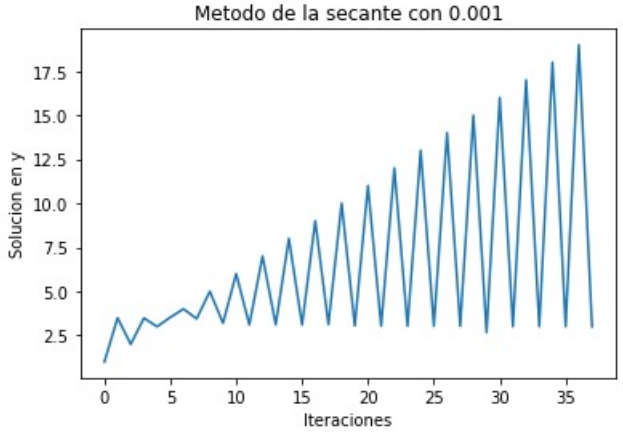

Figura 17: Grafica de iteraciones contra la solucion en y.

Tras analizar el problema de interseccion entre curvas con las dos ecuaciones dadas decidimos aplicar el metodo de la secante ya mencionado anteriormente y realizar unas modificaciones para que pueda aceptar ambos parametros y asi calcular el valor esperado con un error menor de 2 elevado a la menos 16, al analizar el resultado obtuvimos que tras realizar un numero inferior de iteraciones con este metodo modificado dandonos una aproximacion cercana a la esperada.

3. Librerías en Python

Python es un lenguaje de programación que nos sirve de herramienta para la implementación de métodos numéricos. Este lenguaje contiene múltiples librerías que nos sirven de apoyo para las implementaciones, dos de ellas son NumPy y SciPy.

3.1. NumPy

Es una biblioteca de código abierto para el lenguaje de programación Python que significa "Numerical Python", diseañada para matemáticas, ciencias e ingeniería que da soporte para crear vectores y matrices grandes multidimensionales, junto a colección de funciones matemáticas de alto nivel para operar con ellas, por lo que proporciona rutinas básicas para la manipulación de estos grandes conjuntos y matrices de datos numéricos

Numpy maneja una estructura de datos llamada "ndarray"diseñada para matrices de n dimensiones, son matrices descritas de forma homogenea, es decir, que el tipo de los elementos de la matriz debe ser el mismo.

3.1.1. Importación

Existen varias formas de importar la librería Numpy.
La forma extandar es mediante una declaración de importación simple:

```
>>> import numpy
```

Figura 18: Primera forma de importar NumPy

Pero, si queremos utilizar numpy.x para llamadas a funciones de NumPy, debemos utilizar la abreviación np.x y la importanción debe ser de la siguiente manera:

```
>>> import numpy as np
```

Figura 19: Segunda forma de importar NumPy

3.1.2. Rutinas

NumPy contiene una gran variedad de rutinas que están agrupada en los siguientes grupos

- Rutinas de creación de matrices
- Rutinas de manipulación de matrices
- Operaciones binarias
- Operaciones de cadena
- Interfaz de función ajena de tipos c
- Funciones de soporte de fecha y hora

- Rutinas de tipos de datos
- Rutinas aceleradas por SciPy
- Funciones matemáticas con dominio automático
- Manejo de errores de coma flotante
- Transformada discreta de Fourier
- Programación funcional
- Funciones de ayuda específicas de NumPy
- Rutinas de indexación
- Entrada y salida
- Álgebra lineal
- Funciones lógicas
- Operaciones de matriz enmascarada
- Funciones matemáticas
- Biblioteca de matrices
- Rutinas diversas
- Matrices de relleno
- Polinimios
- Muestreo aleatorio
- Establecer rutinas
- Clasificar, buscar y contar
- Estadísticas
- Soporte de prueba
- Funciones de ventana

Algunas de las rutinas que son más relevantes se describen a contiuación

Rutina	**Descripción**
Diag	Extrae una diagonal o construye una matriz diagonal
Diagflat	Crea una matriz bidimensional con la entrada aplanada como diagonal.
Tri	Una matriz con 1 en la diagonal dada y por debajo de ella y 0 en cualquier otro lugar
Copyto	Copia valores de una matriz a otra y difunde según sea necesario
Shape	Devuelve la forma de una matriz.
Seterr	Establece cómo se manejan los errores de punto flotantes
Geterr	Obtiene la forma actual de manejar errores de punto flotante
Fft	Calcula la Transformada de Fourier discreta unidimensional
Ifft	Calcula la transformada de Fourier discreta inversa unidimensional
Fft2	Calcula la Transformada de Fourier discreta bidimensional
Fftn	Calcula la Transformada de Fourier discreta N-dimensional
Ifftn	Calcula la Transformada de Fourier discreta inversa N-dimensional
Apply.along.axix	Aplica una función a cortes 1-D a lo largo del eje dado
Lookfor	Realiza una búsqueda de palabras clave en cadenas de documentos
Info	Obtiene información de ayuda para una función, clase o módulo
Set.printoptions	Determinan la forma en que se muestran los números de punto flotante, las matrices y otros objetos
Linalg.multi.dot	Calcula el producto escalar de dos o más matrices en una sola llamada de función, mientras selecciona automáticamente el orden de evaluación más rápido
Inner	Producto interno de dos matrices
Linalg.eig	Calcula los valores propios y los vectores propios rectos de una matriz cuadrada.
Linalg.norm	Matriz o norma vectorial
Linalg.solve	Resuelve una ecuación de matriz lineal o un sistema de ecuaciones escalares lineales.
Linalg.tensorsolve	Resuelve la ecuación del tensorpara x. ax=b
Linalg.inv	Calcula la inversa (multiplicativa) de una matriz
Around	Redondea uniformemente al número dado de decimales
Round	Redondea una matriz al número dado de decimales
Rint	Redondea al número entero más cercano hacia cero
Setbufsize	Establece el tamaño del búfer utilizado en ufuncs
Getbufsize	Devuelve el tamaño del búfer utilizado en ufuncs

Rutina	Descripción
Polynomial.polynomial	Serie de potencia
Polynomial.chebyshev	Serie Chebyshev
Polynomial.hermite	Serie Hermite
Polynomial.hermite.e	Serie HermiteE
Polynomial.laguerre	Serie Laguerre
Polynomial.legendre	Serie Legendre

3.2. SciPy

Es una biblioteca libre y de código abierto para Python, que significa "Scientific Python", contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT y procesamiento de señales e imagenes. Amplía la funcionalidad de NumPy con una colección de algoritmos útiles como minimización, transformación de fourier, regresión y demás técnicas matemáticas aplicadas. Ahorra enormes cantidades de tiempo en aplicaciones de computación científica gracias a su biblioteca de rutinas probadas y su estructura de datos es un vector multidimensional proporcionado por el módulo NumPy.

3.2.1. Importanción

Para importar la librería de SciPy es necesario que primero se haya importado la de NumPy y se usa el siguiente comando:

```
>>> import scipy
```

Figura 20: Forma de importar SciPy

Si deseamos obtener mayor información sobre los paquetes que ofrece la biblioteca usamos el comando help:

```
>>> help(scipy)
```

Figura 21: Ayuda SciPy

3.2.2. Módulos

SciPy contiene más de veinte módulos diseñados para funciones específicas.

Módulo	Descripción
Cluster	Cuantización Vectorial
Fftpack	Algoritmos de transformación discreta de Fourier
Integrate	Rutinas de integración
Interpolate	Herramientas de interpolación
Io	Entrada y salida de datos
Lib	Wrappers de Python a bibliotecas externas
Lib.lapack	Wrappers a la biblioteca LAPACK
Linalg	Rutinas de álgebra lineal
Misc	Otras herramientas
Ndimage	Paquete de imágenes n-dimenasional
Odr	Regresión de distancia ortogonal
Optimize	Herramientas de optimización
Signal	Herramientas de procesamiento de señal
Sparse	Matrices dispersas
Sparse.linalg	Álgebra lineal dispersa
Sparse.linalg.dsolve	Solucionadores lineales
Sparse.linalg.dsolve.umfpack	Interfaz a la biblioteca UMFPACK
Sparse.linalg.eigen.lobpcg	Método de gradiente conjugado
Special	Funciones Airy
Lib.blas	Conenedores a la biblioteca BLAS
Sparse.linalg.eigen	Solucionadores de valores propios dispersos
Stats	Funciones estadísticas
Spatial	Estructuras de datos espaciales y algoritmos

4. Bibliografia

-https://es.wikipedia.org/wiki/NumPy
-https://es.wikipedia.org/wiki/SciPy
-https://es.qaz.wiki/wiki/Brent's$_{m}ethod-http://metodosnumericoscem.weebly.com/uploads/2/5/9/7/25971049/mn_{c}lase_{1}2_{s}ol_{s}is_{e}cuaciones_{v}2.pdf$
$https://numpy.org/doc/stable/reference/routines.html$