Análisis numérico

Facultad de Ciencias, UNAM
SISTEMAS DE ECUACIONES NO LINEALES

Jorge Zavaleta Sánchez

Semestre 2022-1

Índice

Preliminares

[1]:
```
# Modulos
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets # Controles
from ipywidgets import interact, interact_manual
```

[2]:
```
# Funciones auxiliares

## Derivada numerica
def derivada(f,x,tol = 1e-12, maxiter = 500):
    """Calcula la derivada de f en x (f'(x)) mediante diferencias progresivas.

    - Entrada >
      f (function)  - Funcion de la cual se quiere conocer su derivada.
      x (float)     - Punto donde se quiere conocer el valor de f'.
      tol (float)   - Opcional. Tolerancia para la diferencia entre dos␣
 ↪aproximaciones.
      maxiter (int) - Opcional. Numero maximo de iteraciones.

    - Salida >
      dfx (float) - Valor aproximado de f'(x).
    """
    d,h,it = np.inf,0.5,1
    dfx_a = f(x+1) - f(x)
    while abs(d) > tol and it < maxiter:
```

```
        dfx = (f(x+h) - f(x)) / h
        d = dfx - dfx_a
        dfx_a = dfx
        h /= 2
        it += 1
    return dfx

## Funciones de prueba
fun1 = lambda x:x**2-1.0
fun2 = lambda x:x**2-4.0*np.sin(x)
```

Ceros de una función

Dada una función continua $f\colon [a,b] \subset \mathbb{R} \to \mathbb{R}$, se quiere encontrar un cero o raíz de f. Esto es, se quiere saber si existe $x_0 \in [a,b]$ tal que $f(x_0) = 0$. Existen varios algoritmos para buscar ceros de funciones y a continuación se mostrarán algunos de ellos.

Método de bisección

El algoritmo más simple para la búsqueda de raíces de una función continua f en el intervalo $[a,b]$ es el método de bisección. Si existe una raíz en el intervalo $[a,b]$, el método consiste en ir dividiendo el intervalo a la mitad y seleccionar el subintervalo que contiene la raíz. Para garantizar la existencia de la raíz en el intervalo $[a,b]$ y en los subintervalos subsecuentes, el método se basa en el siguiente teorema

Teorema 1 (Bolzano)**.** *Si f es una función continua en el intervalo $[a,b]$ con $f(a)f(b) < 0$, entonces existe $c \in (a,b)$ tal que $f(c) = 0$.*

La idea es ir cambiando los extremos de los subintervalos para cumplir con las hipótesis del teorema y así garantizar que se seleccione el intervalo que contiene la raíz

Algoritmo (método de bisección)

Algoritmo 1: Bisección

Entrada: f función continua en $[a,b]$, a y b
Salida: Raíz de f

Definir tol
Mientras $(b-a) > tol$ **hacer**
  Calcular
$$m = \frac{b+a}{2}$$
  Si $f(b)f(m) \leq 0$ **entonces**
    $a = m$
  de lo contrario
    $b = m$
devolver m

[3]:
```
def biseccion(f,a,b,tol = 1e-8, maxiter = 500):
    """Busca un cero de la funcion f mediante el metodo de biseccion en el intervalo (a,b).
    - Entrada >
      f (function)    - Funcion de la cual se quiere conocer su raiz.
      a (float)       - Extremo inferior del intervalo.
      b (float)       - Extremo superior del intervalo.
```

```
        tol (float)     - Opcional. Tolerancia para la diferencia entre dos␣
↪aproximaciones.
        maxiter (float) - Opcional. Numero maximo de iteraciones.

    - Salida >
        cero (float) - Valor aproximado de un cero .
        it (int)     - Número de iteraciones que le toma al metodo encontrar un␣
↪cero.
    """
    it = 0
    while abs(b-a) > tol and it < maxiter:
        it += 1
        m = (b+a)/2
        if f(b)*f(m) <= 0:
            a = m
        else:
            b = m
    return m,it
```

Ejemplos

[4]:
```
# Utilizando las funciones de prueba
x,i = biseccion(fun1,-5,5)
print(f'Un cero de la función f1 es {x:2.4e} ya que f({x:2.4e}) = {fun1(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
x,i = biseccion(fun2,-4,1)
print(f'\nUn cero de la función f2 es {x:2.4e} ya que f({x:2.4e}) = {fun2(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
```

```
Un cero de la función f1 es 1.0000e+00 ya que f(1.0000e+00) = 1.1176e-08
El método necesito 30 iteraciones

Un cero de la función f2 es -5.5879e-09 ya que f(-5.5879e-09) = 2.2352e-08
El método necesito 29 iteraciones
```

Nota: *Si la función evaluada en los extremos del intervalo de búsqueda no tienen signos contrarios, el algoritmo puede no converger al cero a pesar de que se encuentre del intervalo.*

En el ejemplo anterior para la función $f(x) = x^2 - 4\sin(x)$, si se toma el intervalo de búsqueda $[-4, 2]$ el método no converge ya que $f(-4)f(2) > 0$.

Método de Newton

El *método de Newton* (conocido también como el método de *Newton-Raphson*) también es un algoritmo para buscar aproximaciones de los ceros o raíces de una función real. Este método incluso puede ser usado para buscar el máximo o mínimo de una función, al buscar los ceros de su primera derivada.

Una forma de obtener el algoritmo es desarrollando la función $f(x)$ en series de Taylor, alrededor del punto x_n

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + (x - x_n)^2 \frac{f''(x_n)}{2!} + \cdots$$

Si se trunca el desarrollo a partir del término de grado 2, y evaluamos en x_{n+1}

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

Si además se supone que x_{n+1} tiende a la raíz, se ha de cumplir que $f(x_{n+1}) = 0$, luego, sustituyendo en la expresión anterior, obtenemos el algoritmo al despejar x_{n+1}

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Esta última ecuación nos da un algoritmo iterativo que genera una sucesión de puntos a partir de un valor inicial arbitrario x_0 que generalmente convergerá siempre que este estimado inicial esté lo suficientemente cerca del cero desconocido, y que $f'(x_0) \neq 0$.

Algoritmo (método de Newton)

Algoritmo 2: Newton

Entrada: f función continua en $[a, b]$, x_0 aproximación inicial
Salida: Raíz de f

Definir *tol*
$n = 0$
Mientras $|x_n - x_{n+1}| > tol$ **hacer**
  Si $f'(x_n) = 0$ **entonces**
    detener
  Calcular
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
  $n = n + 1$
devolver x_n

[5]:
```python
def newton(f,x,tol = 1e-8, maxiter = 500):
    """Busca un cero de la funcion f mediante el metodo de Newton
    - Entrada >
      f (function)    - Funcion de la cual se quiere conocer su raiz.
      x (float)       - Punto inicial.
      tol (float)     - Opcional. Tolerancia para la diferencia entre dos aproximaciones.
      maxiter (float) - Opcional. Numero maximo de iteraciones.

    - Salida >
      cero (float) - Valor aproximado de un cero .
```

```
        it (int)      - Número de iteraciones que le toma al metodo encontrar un␣
↪cero.
    """
    it,h = 0,1
    # Se calcula el cero mediante Newton
    while abs(h) > tol and it < maxiter:
        try:
            h = f(x)/derivada(f,x)
        except ZeroDivisionError:
            print(f'La derivada de la función en {x} es cero')
            return None
        cero = x - h
        x = cero
        it += 1
    return cero,it
```

Ejemplos

[6]:
```
# Utilizando las funciones de prueba
x,i = newton(fun1,5)
print(f'Un cero de la función f1 es {x:2.4e} ya que f({x:2.4e}) = {fun1(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
x,i = newton(fun2,-0.5)
print(f'\nUn cero de la función f2 es {x:2.4e} ya que f({x:2.4e}) = {fun2(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
```

```
Un cero de la función f1 es 1.0000e+00 ya que f(1.0000e+00) = 0.0000e+00
El método necesito 7 iteraciones

Un cero de la función f2 es -1.0572e-18 ya que f(-1.0572e-18) = 4.2286e-18
El método necesito 4 iteraciones
```

[7]:
```
# Tomando un punto inicial no adecuado.
r = newton(lambda x:x**2-1,0)
print(r)
```

```
La derivada de la función en 0 es cero
None
```

Método de la secante

Dado que no siempre es fácil tener información de la derivada en el método de Newton, se puede usar una aproximación para esta. Usando diferencias progresivas

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

y tomando $h = x_{n-1} - x_n$ se tiene que

$$f'(x_n) = \frac{f(x_{n-1}) - f(x_n)}{x_{n-1} - x_n} = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Luego, sustituyendo la última expresión en

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

se tiene que

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Al igual que para el método de Newton, esta última ecuación nos da un algoritmo iterativo que genera una sucesión de puntos a partir de dos valores iniciales arbitrarios x_0 y x_1.

Algoritmo (método de la secante)

Algoritmo 3: Secante

Entrada: f función continua en $[a, b]$, x_0 y x_1 aproximaciones iniciales.
Salida: Raíz de f

Definir *tol*
$n = 1$
Mientras $|x_n - x_{n+1}| > tol$ **hacer**
  Si $f(x_n) - f(x_{n-1}) = 0$ **entonces**
    detener
  Calcular

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

  $n = n + 1$
devolver x_{n+1}

[8]:
```python
def secante(f,x0,x1,tol = 1e-8, maxiter = 500):
    """Busca un cero de la funcion f mediante el metodo de la secante
    - Entrada >
      f (function)    - Funcion de la cual se quiere conocer su raiz.
      x0 (float)      - Primera aproximación.
      x1 (float)      - Segunda aproximación.
      tol (float)     - Opcional. Tolerancia para la diferencia entre dos␣
 ↪aproximaciones.
      maxiter (float) - Opcional. Numero maximo de iteraciones.

    - Salida >
      cero (float) - Valor aproximado de un cero .
```

```
    it (int)     - Número de iteraciones que le toma al metodo encontrar un␣
↪cero.
    """
    it = 0
    while abs(x0 - x1) > tol and it < maxiter:
        try:
            x0,x1 = x1,x0 - f(x0)*(x1-x0)/(f(x1)-f(x0))
        except ZeroDivisionError:
            print('La recta secante no tiene interseccion con el eje x')
            return None
        it += 1
    return x1,it
```

Ejemplos

[9]:
```
# Utilizando las funciones de prueba
x,i = secante(fun1,-5,2)
print(f'Un cero de la función f1 es {x:2.4e} ya que f({x:2.4e}) = {fun1(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
x,i = secante(fun2,-4,2)
print(f'\nUn cero de la función f2 es {x:2.4e} ya que f({x:2.4e}) = {fun2(x):2.
↪4e}')
print(f'El método necesito {i} iteraciones')
```

```
Un cero de la función f1 es 1.0000e+00 ya que f(1.0000e+00) = 0.0000e+00
El método necesito 9 iteraciones

Un cero de la función f2 es 1.9338e+00 ya que f(1.9338e+00) = 1.2434e-14
El método necesito 6 iteraciones
```

[10]:
```
# Tomando puntos iniciales no adecuados.
r = secante(lambda x:x**2-1,-2,2)
print(r)
```

```
La recta secante no tiene interseccion con el eje x
None
```