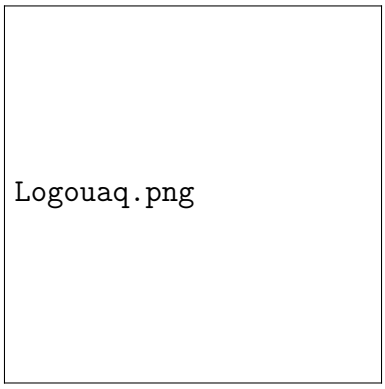
Universidad Autónoma de Querétaro

Facultad de Ingeniería



Tarea 4: Raíces múltiples

Análisis numérico

Autor:
J.A. Salinas Sánchez
Febrero 2022

Índice general

Capítulo 1

Introducción

Siguiendo con el arte de obtener respuestas de lo aparentemente irresolvible, seguimos con ecuaciones de una sola variable; pero con cómo encontrar las múltiples raíces de una ecuación. Como lo establece el teorema fundamental del álgebra, toda ecuación tiene n raíces complejas igual al grado de dicha ecuación. Dentro de las cuales se puede hallar varias reales o con un significado físico notable, pues, si bien un tiempo negativo o distancias complejas no tienen significado físico real; un tiempo real en cualquier rama de la física, o un número complejo en cuántica, sí lo tienen. Entonces, si se tiene la posibilidad de calcular cuantas raíces se pueda; los físicos/ingenieros/matemáticos estarán gustosos de hacerlo.

Ya sabiendo la importancia de calcular múltiples raíces de una ecuación, al entrar al cómo, nos damos cuenta de la existencia de varios obstáculos para hacerlo: las funciones pueden no cambiar de signo entre raíces, las raíces pueden estar muy separadas, pueden encontrarse cerca de alguna indeterminación en la función o simplemente puede no tener raíces reales. Aquí es donde entran cuantos métodos modificados se pueda; tanto para resolver los problemas derivados de las singularidades de una función, como para obtener una convergencia rápida o hallar raíces complejas o sólo reales; dependiendo de lo buscado.

1.1. Newton-Raphson modificado

A pesar de la existencia de múltiples métodos para solucionar lo anteriormente planteado, el más poderoso en cuanto a convergencia y a evasión de singularidades es el método de Newton-Raphson modificado, pues éste se aprovecha de la segunda derivada de una función para calcular con más presición la bajada de las rectas aproximativas, causando una convergencia más rápida y no salirse de la función, evitando caer en singularidades. El método NR modificado se basa en la definición de la derivada de un cosciente, introduciendo una substitución para obtener la

siguiente fórmula:

$$u(x) = \frac{f(x)}{f'(x)} \tag{1.1}$$

$$u'(x) = \frac{f'(x)f'(x) - f''(x)f(x)}{f'(x)^2} \tag{1.2}$$

$$g(x) = x - \frac{\dfrac{f(x)}{f'(x)}}{\dfrac{f'(x)^2 - f''(x)f(x)}{f'(x)^2}} \tag{1.3}$$

$$g(x) = x - \frac{f(x)f'(x)}{f'(x)^2 - f(x)f''(x)} \tag{1.4}$$

Capítulo 2

Métodología

Se utililizó el algoritmo de NR modificado para calcular las raíces múltiples, además, se hizo uso de diferentes módulos de Python3 para lograr que el usuario pudiera introducir su función mediante teclado y que el cálculo de las derivadas y de la fórmula NR modificada se hiciera de manera automática, sin necesidad de tocar el código fuente. Por último, se mueve el valor inicial de la siguiente iteración escalándolo por un factor de -100 para cambiar su signo y acercarlo a otra posible raíz. Si bien, esto puede ser poco eficiente, funciona y fue lo que se concibió en su momento.

2.1. Pseudocódigo general

```
//importar m dulos



main(){
  //ingresar funci n
  //ingresar x0
  //ingresar tolerancia
  NR(funci n ,x0,tolerancia)
}

definirfuncion(funci n){
  return f
}

NRmod(funci n ,x0,tolerancia){

  f=definirfuncion(funci n)
  dx1=derivar(f)
  dx2=derivar(dx1)
  F_NRmod=x-f*dx1/(dx1**2-f*dx1)
  intentos=0
  ea=100
  while(intentos<intentospermitidos){
    while(ea>tolerancia){
      //guardar la soluci n anterior en temp
      //calcular soluciones
    }
    //resetear ea
    //mover x0
  }
  return soluciones
}
```

2.2. Código final

```
#coding:utf8

import math
import numpy
import sympy

x=sympy.symbols('x')

def main():
    func=str(input('Introduzca su funci n: '))
    x0=float(input('Introduzca su valor inicial: '))
    emax=float(input('Introduzca su factor de tolerancia: '))
    NR(func,emax,x0)


def getfunction(f):
    global x
    return(sympy.sympify(f))


def NR(eq,ed,x0):
    global x
    xr=complex(x0)
    ecuacion=getfunction(eq)
    derivada1=sympy.diff(ecuacion)
    derivada2=sympy.diff(derivada1)
    ea=100
    e=100
    temp=0
    preliminares=[]
    resultados=[]
    f_NR=x-(((ecuacion)*(derivada1))/((derivada1)**2-((ecuacion)*(
   derivada2))))
    intentos=0

    while(intentos<1000 and round(xr.imag)==0):
        while(ea>ed):
            temp=complex(xr)
            xr=complex(f_NR.evalf(subs={x:temp}))
            if(xr!=0):
                ea=abs((xr-temp)/temp)
        if(xr.real in preliminares and xr.real not in resultados):
            resultados.append(xr.real)
        elif(xr.real not in preliminares):
            preliminares.append(xr.real)
        else:
            break
        e=ea
        ea=100
        if(abs(xr)==0):
            xr=-100*(temp+0.0001)
        elif(abs(xr)==1):
            xr=-100*(temp+0.0001)
        else:
            xr=-100*temp
        intentos+=1
        print(intentos)
    print(resultados)

if __name__ =='__main__':
```

```
60      main()
```

Capítulo 3

Ejercicios

Bibliografía

(1) hapra. S.C et Canale R.P (2015) *Métodos numéricos para ingenieros.* McGrawHill. pp (113-124)&(135-136)

(2) alls, P., 2022. Secant method. Secant Method - Mathematical Python. Available at: https://personal.math.ubc.ca/ pwalls/math-python/roots-optimization/secant/ (Accessed February 10, 2022).