Área Académica de Ingeniería en Computadores
Análisis Numérico para Ingeniería

Tarea 2

Método de Newton-Raphson

Profesor
Juan Pablo Soto Quirós

Estudiantes
Steven Badilla Soto
Fabián Crawford Barquero
Irene Muñoz Castro
Luis Morales Rodríguez
Adrián Trejos

I Semestre – 2022

Problema por resolver

El problema que queremos resolver es el solucionar un sistema de ecuaciones formado por ecuaciones no lineales, es decir obtener el valor que deben tener las variables para que cada ecuación del sistema de como resultado cero o bien f(x) = 0 donde f = (f1,…,fn) : ℝ --> ℝ y x = (x1,…,xn) (las variables de las funciones)

Método de Newton-Raphson

Para solucionar un sistema de ecuaciones con este método iterativo cada función del sistema debe ser continua y definida, y debes existir las primeras derivadas parciales de la función para cada variable. Para comenzar se debe obtener la matriz Jacobiana para este sistema de ecuaciones esta se define como $[Jf(c)]_{i,j} = \frac{\partial f_i}{\partial x_j}(c)$ para todo i,j = 1,…,n ,donde c representa los valores dados para cada variable del sistema, con el Jacobiano el método consiste en $\begin{cases} x_{k+1} = x_k - [Jf(x_k)]^{-1}f(x_k) \\ x_0 \in \mathbb{R}^n \end{cases}$, es importante notar que se puede evitar el cálculo de la inversa de $[Jf(x_k)]$ ya que $y = [Jf(x_k)]^{-1}f(x_k)$ se puede resolver como $Jf(x_k)y = f(x_k)$ aplicando algún método para la resolución de sistemas de ecuaciones, por ultimo como este es un método iterativo se debe cumplir una tolerancia de forma que error <= tolerancia, para el cálculo del error usamos $e_k = ||\, f(x_k)||$

Pseudocódigo

Entradas: vector inicial x0, vector de funciones no lineales f, vector de variables x, tolerancia > 0, iteraciones máximas > 0

Salidas: vector con la aproximación de la solución xk, iteraciones totales k, error,

```
def newton_raphson(x0,f,x,tol,iterMax):
    xk = x0
    k = 0
    error = tol +1
    while k < iterMax and error > tol:
        jac = jacobiano(f,x,xk,len(f))
        y = solve(jac,f(xk))
        xk = xk – y
        k += 1
    return [xk,k,error]
```

```
def jacobiano(f,x,xk,m):
    i = 0
    j = 0
    while i < m:
        while j < m:
            df = diff(f[i],x[j])
            jac[i,j] = df(xk)
            j += 1
        j = 0
        i += 1
    return jac
```