Análisis Numérico

Proyecto Final

Método de Newton para sistemas no lineales

- Implementación y teoría del método -

00076015 Carlos Javier Burgos Martinez

00006715 David Bejamín Ayala Giralt

00388913 Diego José Eguizabal Liu

00058615 Karla Esperanza López Méndez

00353715 Mario Cecilio De Leon Recinos

00004315 Rafael Enrique Cruz Aparicio

00088116 Yury Alejandro Rivera Quintanilla

July 1, 2019

Contents

1 Definiciones

El problema computacional que se intenta resolver es encontrar soluciones para ecuaciones de sistemas no lineales. Las cuales seran resueltas con el metodo de newton adaptado para sistemas no lineales. Este metodo ha sido modificado de tal forma que el algoritmico pueda efectuar la transformacion a un sitema de $\mathbb{R}^n$.

Para construir dicho algoritmo que lleve a una solución del método de punto-fijo en un caso unidimensional, obtuvimos una función ϕ con las propiedades

$$g(x) = x - \phi(x)f(x)$$

que da una convergencia cuadrática en el punto fijo p de la función g.De esta condición el método de Newton evolucionó al escoger $\phi(x) = 1/f'(x)$ asumiendo que $f'(x) \neq 0$.

Un enfoque similar en el caso n-dimensional implica una matriz:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \tag{1}$$

Donde cada una de las entradas $a_{ij}(x)$ es una función de $\mathbb{R}^n$ a $\mathbb{R}$. Esto requiere que $A(x)$ sea encontrado para que

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \mathbf{A}(\mathbf{x})^{-\mathbf{1}}\mathbf{F}(\mathbf{x})$$

genere una convergencia cuadrática para la solución $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, asumiendo que $A(x)$ es no singular en el punto fijo $\mathbf{p}$ de $\mathbf{G}$.

Suponemos que p es una solución de $G(x) = x$. Si existe un número $\delta > 0$ con la propiedad que:

(i) $\partial g_i/\partial x_j$ sea continua en $N_\delta = x| \quad \|x - p\|$ para toda $i = 1, 2, ..., n$ y toda $j = 1, 2, ///, n$;

(ii) $\partial^2 g_i(x)/\partial x_j \partial x_k$ sea continua y $\left|\partial^2 g_i(x)/(\partial x_j \partial x_k)\right| \leq M$ para alguna constante M siempre que $x \in N_\delta$ para toda $i = 1, 2, ..., n, j = 1, 2, ..., n$ y toda $k = 1, 2, ..., n$;

(iii) $\partial g_i(p)/\partial x_k = 0$ para toda $i = 1, 2, ..., n$ y toda $k = 1, 2, ..., n$.

Entonces existe un número $\hat{\delta} \leq \delta$ tal que la sucesión generada por $x^k = G(x^{k-1})$ converge cuadráticamente a p para cualquier elección de x^0 a condición de que $\left\|x^0 - p\right\| < \hat{\delta}$

$$\left\|x^k - p\right\|_\infty \leq \frac{n^2 M}{2} \quad \left\|x^{k-1} - p\right\|_\infty^2 \text{ para toda k}\geq 1$$

Para utilizar el teorema supongamos que $A(x)$ es una matriz de nXn de funciones de $\mathbb{R}^n$ a $\mathbb{R}$ en la forma de la ecuación 1, cuyos elementos específicos se escogerán más adelante. Supongamos además que $A(x)$ es no singular cerca de una solución p de $F(x) = O$, y denotemos con $b_{ij}(x)$ el elemento de $A(x)^{-1}$ en el i-ésimo renglón y en la j-ésima columna.

Dado que $G(x) = x - A(x)^{-1}F(x)$, tenemos $g_i(x) = x_i - \sum_{j=1}^{n} b_{ij}(x)f_j(x)$

$$\frac{\partial g_i}{\partial x_k}(\mathbf{x}) = \begin{cases} 1 - \sum_{j=1}^{n}\left(b_{ij}(\mathbf{x})\frac{\partial f_j}{\partial x_k}(\mathbf{x}) + \frac{\partial b_{ij}}{\partial x_k}(\mathbf{x})\, f_j(\mathbf{x})\right), & \text{si } i = k, \\ -\sum_{j=1}^{n}\left(b_{ij}(\mathbf{x})\frac{\partial f_j}{\partial x_k}(\mathbf{x}) + \frac{\partial b_{ij}}{\partial x_k}(\mathbf{x}) f_j(\mathbf{x})\right), & \text{si } i \neq k. \end{cases}$$

(2)

El teorema que se menciono anteriormente implica que necesitamos $\partial g_i(p)/\partial x_k$ para toda $i = 1, 2, ..., n$ y toda $k = 1, 2, ..., n$. Esto significa que, para toda $i = k$,

$$0 = 1 - \sum_{j=1}^{n} b_{ij}(p)\frac{\partial f_j}{\partial x_i}(p),$$

por lo que

$$\sum_{j=1}^{n} b_{ij}(p)\frac{\partial f_j}{\partial x_i}(p) = 1 \tag{3}$$

Cuando $k \neq 1$

$$0 = -\sum_{j=1}^{n} b_{ij}(p)\frac{\partial f_j}{\partial x_i}(p)$$

por lo que

$$\sum_{j=1}^{n} b_{ij}(p)\frac{\partial f_j}{\partial x_i}(p) = 0 \tag{4}$$

Al definir la matriz $J(x)$ por medio de

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \dots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix} \tag{5}$$

vemos que las condiciones de las ecuacions (3) y (4) requieren

$$A(p)^{-1}J(p) = I, \text{la matriz identidad,}$$

por lo que

$$A(p) = J(p).$$

En consecuencia, una eleccion apropiada de $A(x)$ es $A(x) = J(x)$, dado que entonces se cumple la condicion(iii) del teorema.

La funcion G esta definida por

$$G(x) = x - J(x)^{-1}F(x)$$

y el procedimiento de la iteracion funcional pasa de seleccionar x^0 a generar, para $k \geq 1$,

$$x^k = G(x^{k-1}) = x^{k-1} - J(x^{k-1})^{-1}F(x^{k-1})$$

Con esto se concluye que esto es metodo de Newton para sistemas no lineales, y generalmente se espera que de una convergencia cuadratica, siempre y cuando se conozca un valor inicial suficientemente preciso y exista $J(p)^{-1}$.

La debilidad de este metodo es que se tiene la necesidad de calcular e invertir la matriz $J(x)$ en cada paso.

Matriz Jacobiana: Es una matriz que está conformada por las derivadas parciales, de primer orden de una función.

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \dots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}$$

Campo vectorial es una función vectorial de varias variables en la que a cada punto de su dominio se le asigna el vector correspondiente a una determinada magnitud vectorial que actúa sobre dicho punto.

$$F : A \subset \mathbb{R}^n \longrightarrow \mathbb{R}$$

Espacio vectorial es la estructura matemática que se crea a partir de un conjunto no vacío y que cumple con diversos requisitos y propiedades iniciales. Esta estructura surge mediante una operación de suma (interna al conjunto) y una operación de producto entre dicho conjunto y un cuerpo.

$$\mathbb{R}_n = x_1, x_1, ..., x_n, \text{ con } x_i \in \mathbb{R}$$

En $\mathbb{R}_n$, la suma de vectores y el producto por un escalar se definen asi:

$$\text{Sean } u = u_1, u_2, ..., u_n \text{ y } v_1, v_2, ..., v_n \in \mathbb{R}_n$$

$$u + v = (u_1 + v_1, u_2 + v_2, ..., u_n + v_n) \in \mathbb{R}_n$$

$$\alpha v = \alpha v_1, \alpha v_2, ..., \alpha v_3 \in \mathbb{R}_n$$

Vector nulo:Es el vector de coordenadas (0,0). O bien, se define como el vector de longitud o módulo cero. La definición de vector nulo es una convención matemática útil para resolver ecuaciones vectoriales. Juega el papel de elemento neutro para la suma de vectores.

Matriz inversa: En matemáticas, en particular en álgebra lineal, una matriz cuadrada A de orden n se dice que es invertible, no singular, no degenerada o regular si existe otra matriz cuadrada de orden n, llamada matriz inversa de A y representada como A^{-1}, tal que:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}.$$

,donde In es la matriz identidad de orden n y el producto utilizado es el producto de matrices usual.

El conjunto $\mathbb{R}_n$ es el producto cartesiano $\mathbb{R}x...x\mathbb{R} = \mathbb{R}_n$, cuyos elementos son las n-uplas de números reales $(x_1, ..., x_n), x_1 \in \mathbb{R}, i = 1, ..., n$, llamados vectores de n componentes.

2 Desarrollo del programa

Este programa fue desarrollado con la intención de permitir resolver sistemas de ecuaciones no lineales, de manera numérica, usando el método de Newton adaptado para estos tipos de sistemas. Estos sistemas son un conjunto de ecuaciones no lineales que tienen múltiples variables y describen fenómenos bastante complejos como los simulados por computadora.

2.1 Algoritmo

El algoritmo que utilizamos en este caso fue el método de Newton adaptado para trabajar con sistemas no lineales de n funciones y n variables que mapean $\mathbb{R}^3$ a $\mathbb{R}^3$ de forma iterativa. Como ya se mencionó en esta forma, el método de Newton debe trabajar para resolver varias ecuaciones con varias incógnitas es decir, un sistema de ecuaciones. La característica principal de este sistema es ser no lineal, por lo que los métodos convencionales para resolver el SEL resultan ser no útiles. A consecuencia de esto se necesita otro tipo de método. Uno de los candidatos es el método de Newton, sin embargo este método esta definido para una variable y se deduce de un polinomio de series de potencias, un polinomio de taylor de segundo grado para ser precisos. Por lo que, este método no puede aplicarse directamente al sistema y necesita ser adaptado. Describimos los cambios de forma general a continuación.

Si recordamos la forma unidimensional del método de Newton:

$$x_{k+1} = x_k + \frac{f(x_k)}{f'(x_k)}$$

Se observa que cada iteración está dada por el valor resultante de la iteración anterior o del valor inicial, más la función evaluada en ese mismo punto, dividido entre la derivada de la función evaluada en ese mismo punto nuevamente. El primer cambio necesario para usar el método de Newton en un sistema no lineal es definirlo de forma matricial, esto se logra desarrollando una aproximación por polinomio de taylor de grado 2 a la función $f(x_0, x_1, x_3, ...x_n)$. De este último polinomio agrupamos todas las derivadas parciales y las convertimos a una forma matricial, generando así la matriz Jacobiana que al invertirla juega el mismo papel que la derivada de la función para el caso unidimensional o el método de Newton sin adaptar. El siguiente cambio importante, es que ya no se itera sobre un punto si no sobre dos vectores columna, el primero que contiene las coordenadas del punto de la iteración anterior o de la inicial y el segundo es un vector con las coordenadas imágenes de $f(x_n)$.

El método adaptado para resolver sistemas de ecuaciones no lineales aún posee una forma y características similares a la de su forma inicial. La diferencia radica en que el método generalizado no trabaja iterando sobre $\mathbb{R}$ para encontrar la raíz de f sino que este itera sobre un campo vectorial de $\mathbb{R}^n$ definido por las n funciones que necesita resolver. En otras palabras el método retorna como respuesta un vector columna con las soluciones o el vector solución.

Forma generarizada del método de Newton.

$$\bar{X}_{k+1} = \bar{X}_k + J(\bar{X})^{-1} F(\bar{X})$$

Pero para este caso el papel de la derivada lo juega la matriz de derivadas parciales de cada función, es decir, la matriz Jacobiana que nos permite mapear $\mathbb{R}^3$ a $\mathbb{R}^3$ y en lugar iterar sobre un punto x de $\mathbb{R}$ ahora iteramos en el vector $\bar{X}$.
Desafortunadamente, el cálculo de una matriz Jacobiana luego debe ser invertido y el cálculo de una matriz inversa resulta demasiado costoso en términos de procesamiento por lo que se optó por usar una forma alternativa al método que genera un SEL al final el cual, es mucho más rápido de resolver

$$J(\bar{X}_k)y = -F(\bar{X}_k)$$

Siendo y un vector que satisfaga la igualdad, es de la necesidad de encontrar las coordenadas de ese vector de donde se origina el SEL previamente mencionado. Finalmente la respuesta para esa iteración será la suma del vector $\bar{X}$ y el vector y

2.2 Pseudocódigo

El Pseudocódigo para el proyecto se divide en 3 bloques principales de código:

1. Configuración de parámetros
2. Bucle
3. Salida

2.2.1 Configuración de parámetros

Durante la configuración de parámetros el programa necesita recibir como entrada varias cosas, entre ellas están: el número n de incognitas y de funciones que se desea calcular, luego la precisión deseada que luego servirá para saber en que momento detener el programa. Finalmente se necesitarán las ecuaciones que deben ser solucionadas.

2.2.2 Bucle

Esta es la parte computacionalmente más exigente del programa, en esta parte se necesitan calcular los siguientes valores:

- Matriz Jacobiana de la iteración
- El vector $F(\bar{X})$
- Un SEL al fin de cada iteración

2.2.3 Salida

Esta es la parte donde en caso de éxito, los valores numéricos obtenidos en el vector columna se le muestran al usuario en un formato amigable así como la tabla con las iteraciones realizadas, o en caso de fallo un mensaje de error y si es posible las tablas de iteración que el programa haya logrado ejecutar.

3 Resultados Practicos

3.0.1 Ejemplo

Para mostrar la implementacion de Newton en un sistema no lineal, se ha considerado un sistema de ecuaciones de 3 incognitas en $\mathbb{R}^3$, el cual tiene una solucion aproximada en $(0.5,\ 0,\ -0.52359877)^t$. y una aproximacion inicial $(0.1,\ 0.1,\ -0.1)^t$

$$Eq_1 = 3x_0 - \cos(x_1 x_2) - \frac{1}{2} = 0$$

$$Eq_2 = x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + 1.06 = 0$$

$$Eq_3 = \epsilon^{-x_0 x_1} + 20x_2 + \frac{10\pi - 3}{3} = 0$$

Calculo de matriz Jacobiana J(x_0, x_1, x_2)

$$\begin{bmatrix} 3 & x_2 \sin(x_1 x_2) & x_1 \sin(x_1 x_2) \\ 2x_0 & -162x_1 - 16.2 & \cos(x_2) \\ -x_1 e^{-x_0 x_1} & -x_0 e^{-x_0 x_1} & 20 \end{bmatrix}$$

Evaluar matriz jacobiana en $(0.1, 0.1, -0.1)$

$$J(x) = \begin{bmatrix} 3.0 & 0.0009999833333416667 & 0.0009999833333416667 \\ 0.2 & -32.4 & 0.995004165278026 \\ -0.0990049833749168 & -0.0990049833749168 & 20.0 \end{bmatrix}$$

Evaluar F(x) en $(0.1, 0.1, -0.1)$

$$F(x) = \begin{bmatrix} -1.19995000041667 & -2.07016658335317 & 12.4620253457151 \end{bmatrix}$$

Resolver el sistema de ecuaciones lineales J(x)**y** $= -$**F**(x)

Al resolver dicho sistema de euaciones lineales, se elimina el problema de calcular la matriz inversa del jacobiano, aumentando la eficiencia del metodo.

$$SEL = \begin{bmatrix} 0.400217339249327 & -0.0805103934985596 & -0.621518638243312 \end{bmatrix}$$

Sumar $\overline{X} + \overline{Y}$

Donde $\overline{X}$ es el vector con las coordenadas iniciales (0.1,0.1,-0.1) y $\overline{Y}$ el vector solucion del SEL previamente resuelto.

$$\begin{bmatrix} 0.500217339249327 & 0.0194896065014404 & -0.521518638243312 \end{bmatrix}$$

Estos nuevos puntos son la solucion de la primera iteración por el metodo de Newton, donde el valor inicial de la proxima iteracion es: $(0.5002173392, 0.0194896065, -0.5215186382)^t$.

Es imporante considerar que se iterará hasta que la **norma 2** de $\overline{Y}$ no sea menor que la tolerancia.

3.0.2 Implementación

El codigo esta escrito en **Python** utilizando librerias de **Sympy** y **Numpy** para simplificar calculos.

Funcion principal encargada en solucionar las ecuaciones no lineales por el metodo de Newton

```
import numpy as np
from sympy import*
from sympy.interactive import printing;
printing.init_printing(use_latex=true);
from IPython.display import display, Latex



def Newton_system(F, J, x, eps):
"""
Solve nonlinear system F=0 by Newton's method.
J is the Jacobian of F. Both F and J must be functions of x.
At input, x holds the start value. The iteration continues
until ||F|| < eps.
"""
F_value = F(x)
#display(Latex('$$ F(x) = '+ latex(simplify(F_value)) + '$$'))
F_norm = np.linalg.norm(F_value, ord=2)  # l2 norm of vector
iteration_counter = 0
while abs(F_norm) > eps and iteration_counter < 100:
delta = np.linalg.solve(J(x), -F_value)
display(Latex('$$ F(x) = '+ latex(simplify(F_value)) + '$$'))
display(Latex('$$ J(x) = '+ latex(simplify(J(x))) + '$$'))
display(Latex('$$ SEL = '+ latex(simplify(delta)) + '$$'))
x = x + delta
display(Latex('$$ SolucionSistema = '+ latex(simplify(x)) + '$$'))
F_value = F(x)
F_norm = np.linalg.norm(F_value, ord=2)
iteration_counter += 1

# Here, either a solution is found, or too many iterations
if abs(F_norm) > eps:
```

```
iteration_counter = -1
return x, iteration_counter
```

- **Funcion en la cual se ingresa el sistema a evaluar y su respectivo jacobiano**

```
def test_Newton_system1():
from numpy import cos, sin, pi, exp

def F(x):
eq1=3*x[0] - cos(x[1]*x[2]) -1/2
eq2=x[0]**2 -81*(x[1]+0.1)**2 +sin(x[2])+1.06
eq3= 1/exp(x[0]*x[1])+20*x[2]+(10*pi-3)/3
return np.array(
[eq1,eq2,eq3])

def J(x):
return np.array([[3,    x[2]*sin(x[1]*x[2]), x[1]*sin(x[1]*x[2])],
[  2*x[0],   -162*x[1] - 16.2,         cos(x[2])],
[-x[1]*exp(-x[0]*x[1]), -x[0]*exp(-x[0]*x[1]),20]])
#
expected = np.array([0.5,0,-0.52359877])
tol = 1e-9
x, n = Newton_system(F, J, x=np.array([0.1,0.1,0.1]), eps=1e-9)
error_norm = np.linalg.norm(expected - x, ord=2)
if error_norm < tol:
print('la norma del error es mas peque a que la tolerancia')
print('Numero maximo de iteraciones exedido')
print ('norm of error =%g' % error_norm)
print(' la tolerancia es', tol)
```

3.0.3 Análisis del codigo y su implementación

- La computadora puede calcular sin mayor problemas el jacobiano, con la libreria de Sympy,

pero esto nos causo una lentitud excesiva, por lo cual se opto a dejar de manera explicita el jacobiano en la funcion.

- No utilizamos por las mismas razones algebra simbolica ya que este proceso nos demoraba demasiado al momento de resolver el sistema de ecuaciones lineales.
- Al eliminar esos obstaculos, el tiempo de ejecuacion ha incrementado satisfactoriamente
- Se han realizado ejercicios de prueba del libro de Burden y se han obtenido los mismos resultados en menor iteraciones, dando por entendido que su implementacion es eficiente.
- El calculo de la inversa nunca fue una opcion implementarla para la solucion de estos sistemas no lineales.

En el libro de Burden se obtuvo la misma precision en 8 iteraciones, mientras nuestro programa lo calculo en 5, con una tolerancia de 10^{-9}

$$\textit{SolucionSistema} = \begin{bmatrix} 0.500217339249327 & 0.0194896065014404 & -0.521518638243312 \end{bmatrix}$$

$$\textit{SolucionSistema} = \begin{bmatrix} 0.500014270519337 & 0.00159198938309455 & -0.523557180114757 \end{bmatrix}$$

$$\textit{SolucionSistema} = \begin{bmatrix} 0.500000113950141 & 1.2497660849872 \cdot 10^{-5} & -0.523598448689747 \end{bmatrix}$$

$$\textit{SolucionSistema} = \begin{bmatrix} 0.500000000007136 & 7.82391898286304 \cdot 10^{-10} & -0.523598775577834 \end{bmatrix}$$

$$\textit{SolucionSistema} = \begin{bmatrix} 0.5 & 3.41773492877925 \cdot 10^{-18} & -0.523598775598299 \end{bmatrix}$$

References

[1] RL Burden, J Douglas Faires, and AC Reynolds. Numerical analysis brooks-cole. *Boston, Mass, USA,*, 2010.

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[3] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.