Tarea 8 - Optimización

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En el presente reporte se hablará sobre la implementación de los algoritmos de Newton y Broyden para la resolución de sistemas de ecuaciones no lineales. Se mencionará la manera en la que se hicieron las implementaciones, además de la demostración de las ejecuciones realizadas y los resultados obtenidos. Finalmente se añadirá la resolución del ejercicio 5 de la tarea.

Keywords: Método de Newton, Método de Broyden, Sistemas de ecuaciones no lineales.

1. Método de Newton

Primeramente se aplicó el método de Newton para resolver sistemas de ecuaciones no lineales, es decir encontrar un x tal que $F(x) = 0$ donde $x \in R^N$ y $F(x) \in R^N$. Por lo cual vemos que x es un vector en dimensión N y que F es una función vectorial de igual manera dimensión N. A continuación se muestra el algoritmo del método de Newton usado en esta tarea:

Algorithm 1 Método de Newton.

procedure NEWTON(x_0, F, J)
  $x_k \leftarrow x_0$.
  while Not convergencia **do**
    $J(x_k)s_k \leftarrow -F(x_k)$. ▷ Resolver el sistema de ecuaciones.
    $x_{k+1} \leftarrow x_k + s_k$.
  return x^*

En el algoritmo anterior vemos que para ir encontrando el siguiente valor de x necesitamos resolver un sistema de ecuaciones donde $J(x)$ es la matriz Jacobiana y $F(x)$ es la función a encontrar sus raíces. La actualización se realiza como $x_{k+1} = x_k + s_k$ donde s_k es el resultado del sistema de ecuaciones. Finalmente se retorna el vetor x_r el cual se obtuvo ya sea porque el algoritmo convergió o se cumplió algún otro criterio, como llegar a un número de iteraciones límite, etc.

1.1. Ejecuciones y resultados

En la presente tarea se implementó el algoritmo de Newton y además, se probó con una función no lineal de tres términos. La cual se muestra a continuación:

$$F(x) = (f_1(x), f_2(x), f_3(x))^T \tag{1}$$

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 3x_1 - cos(x_2 x_3) - 0{,}5 \\ f_2(x_1, x_2, x_3) &= x_1^2 - 81(x_2 + 0{,}1)^2 + sin(x_3) + 1{,}06 \\ f_3(x_1, x_2, x_3) &= \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{aligned} \tag{2}$$

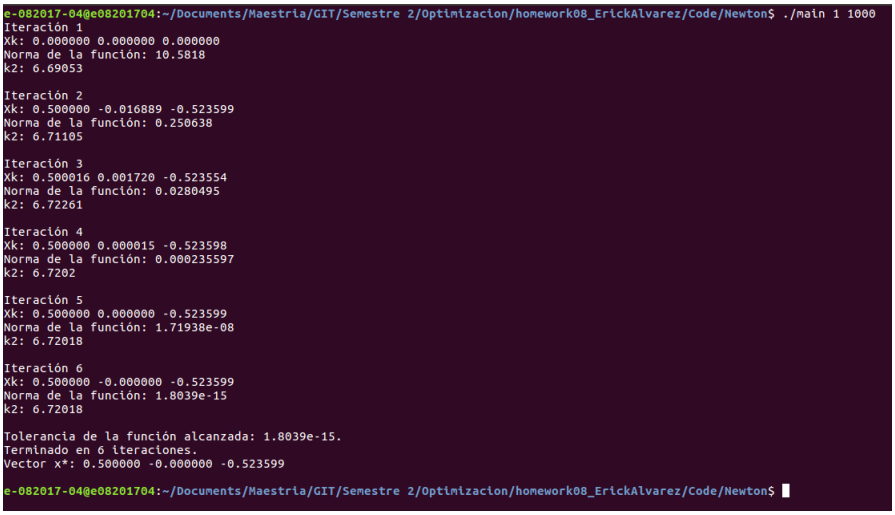
La matriz Jacobiana de dicha función es la siguiente:

$$J(x) = \begin{pmatrix} 3 & x_3 sin(x_2 x_3) & x_2 sin(x_2 x_3) \\ 2x_1 & -162(x_2 + 0{,}1) & cos(x_3) \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{pmatrix} \tag{3}$$

Para probar el programa se usaron 4 puntos iniciales diferentes, unos más cercanos a la raíz que otros, dichos puntos se mostrarán a continuación en una tabla con los resultados finales del algoritmo.
En el proceso de ejecución se fueron monitoreando ciertos valores que nos iban brindando mucha información de lo que estaba ocurriendo, tales valores fueron:

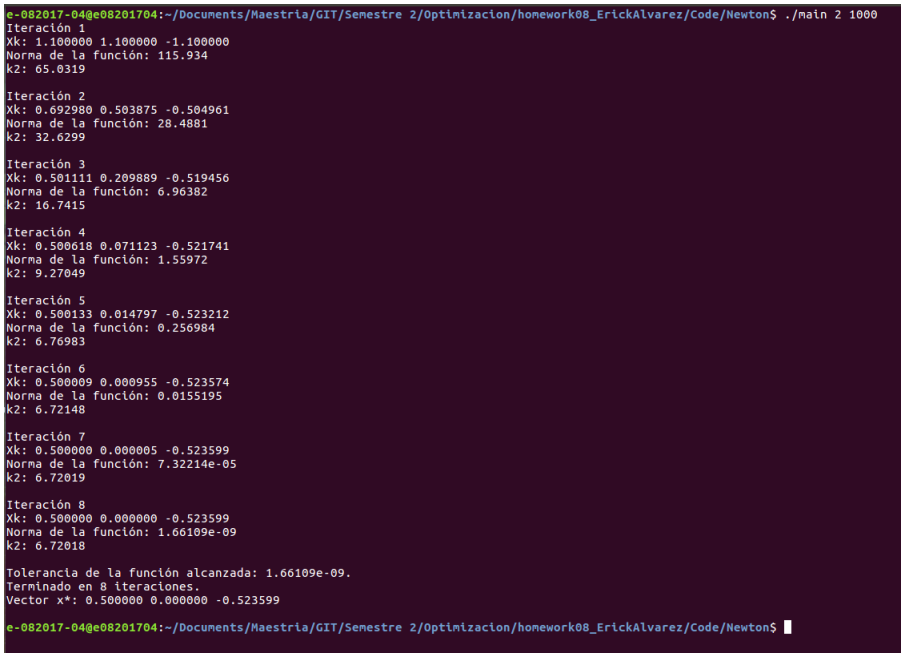
1. k: El número de iteración.
2. x_k: El vector x en la iteración k.
3. $||F(x_k)||$: La norma de la función evaluada en el punto x_k.
4. $k_2(J(x))$: El número de condición de la matriz Jacobiana.

Como un pequeño comentario, el número de condición de la matriz se calculó a través de los valores singulares de la misma. Además se definieron los límites de 1000 iteraciones y una tolerancia de $\sqrt{\epsilon_m}$ para el criterio de parada.
A continuación se mostrarán unas capturas de pantalla realizadas a la ejecución del programa con cada punto inicial.



(a) Figura 1. Ejecución del algoritmo de Newton para el primer punto inicial.

Vemos que para el primer punto inicial el cual fue $x_0 = (0, 0, 0)^T$ el algoritmo tardó 6 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.



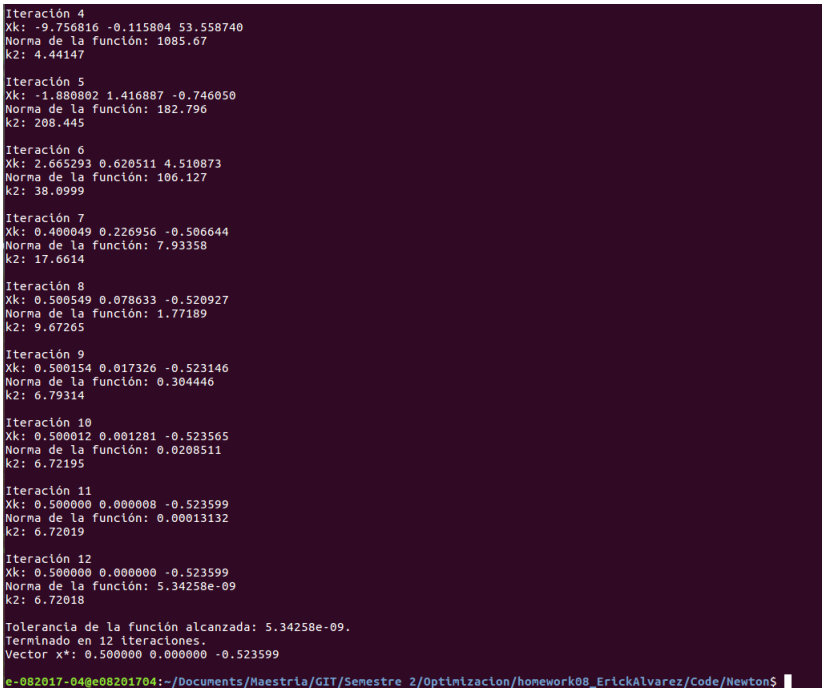
(b) Figura 2. Ejecución del algoritmo de Newton para el segundo punto inicial.

Vemos que para el segundo punto inicial el cual fue $x_0 = (1{,}1, 1{,}1, -1{,}1)^T$ el algoritmo tardó 8 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.

```
e-082017-04@e08201704:~/Documents/Maestria/GIT/Semestre 2/Optimizacion/homework08_ErickAlvarez/Code/Newton$ ./main 3 1000
Iteración 1
Xk: -10.000000 -10.000000 10.000000
Norma de la función: 7841.16
k2: 540.538

Iteración 2
Xk: -25.145339 -5.307021 -0.473599
Norma de la función: 1565.07
k2: 287.252

Iteración 3
Xk: 0.209144 -1.942190 -0.473599
Norma de la función: 274.244
k2: 93.64

Iteración 4
Xk: 0.435809 -1.023304 -0.567271
Norma de la función: 68.3398
k2: 49.2618

Iteración 5
Xk: 0.497621 -0.566924 -0.541105
Norma de la función: 16.867
k2: 25.1665

Iteración 6
Xk: 0.497069 -0.344029 -0.532521
Norma de la función: 4.02423
k2: 13.1756

Iteración 7
Xk: 0.497765 -0.242309 -0.529938
Norma de la función: 0.838112
k2: 7.69765

Iteración 8
Xk: 0.498087 -0.206004 -0.528993
Norma de la función: 0.106761
k2: 6.67846

Iteración 9
Xk: 0.498143 -0.199799 -0.528831
Norma de la función: 0.00311917
k2: 6.67903

Iteración 10
Xk: 0.498145 -0.199606 -0.528826
Norma de la función: 3.00317e-06
k2: 6.67906

Iteración 11
Xk: 0.498145 -0.199606 -0.528826
Norma de la función: 2.79466e-12
k2: 6.67906

Tolerancia de la función alcanzada: 2.79466e-12.
Terminado en 11 iteraciones.
Vector x*: 0.498145 -0.199606 -0.528826

e-082017-04@e08201704:~/Documents/Maestria/GIT/Semestre 2/Optimizacion/homework08_ErickAlvarez/Code/Newton$
```

(c) Figura 3. Ejecución del algoritmo de Newton para el tercer punto inicial.

Vemos que para el tercer punto inicial el cual fue $x_0 = (-10, -10, 10)^T$ el algoritmo tardó 11 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.



(d) Figura 4. Ejecución del algoritmo de Newton para el cuarto punto inicial.

Vemos que para el cuarto punto inicial el cual fue $x_0 = (3, -3, 3)^T$ el algoritmo tardó 12 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.

A continuación se muestra un resumen de los datos obtenidos en las ejecuciones realizadas mediante una tabla.

Cuadro 1: Resultados de la ejecución del algoritmo de Newton a los 4 puntos iniciales.

Punto inicial	**Iteración final** k	x_k	$\|\|F(x_k)\|\|$	$k(J(x_k))$
(0, 0, 0)	6	(0.5, 0, -0.523)	1.8039e-15	6.72018
(1.1, 1.1, -1.1)	8	(0.5, 0, -0.523)	1.66109e-09	6.72018
(-10, -10, 10)	11	(0.498, -0.19, -0.528)	2.79466e-12	6.67906
(3, -3, 3)	12	(0.5, 0, -0.523)	5.34258e-09	6.72018

De la Tabla anterior podemos notar que el algoritmo convergió en los 4 casos, además de que en tres de ellos llegó al mismo punto x^* excepto en el punto tres, para el cual se llegó a un vector muy cercano pero que sí cumple la tolerancia especificada.

Otra cosa que podemos notar de los resultados es que en ningún momento se tuvo un número de condición alto en la matriz, el cual se mantuvo cerca de 6.

2. Método de Broyden

Posterior a ejecutar el método de Newton, el siguiente ejercicio fue implementar el método de Broyden el cual es muy parecido al de Newton, excepto que en este algoritmo usa una aproximación de la matriz Jacobiana que recalcula en cada nueva iteración. A continuación se muestra el pseudocódigo del algoritmo de Broyden implementado en la presente tarea.

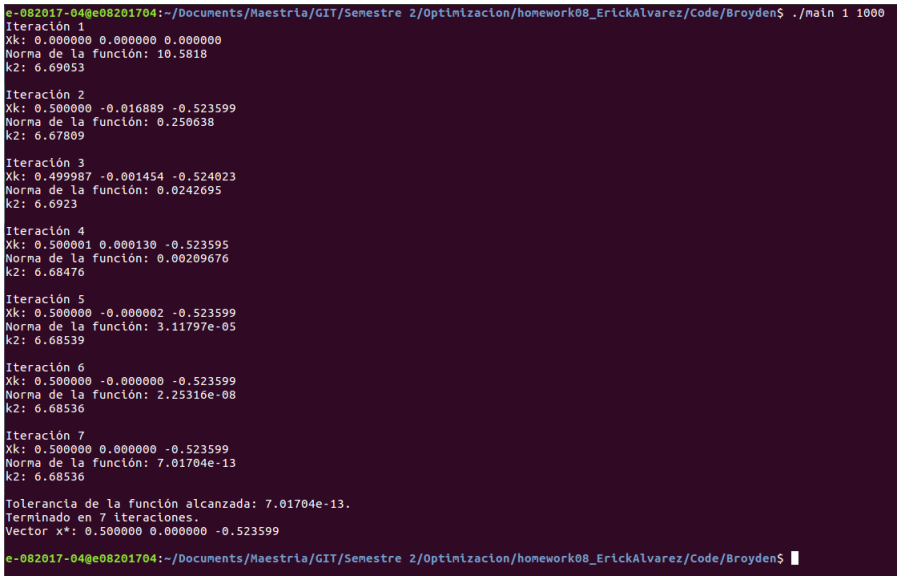
Algorithm 2 Método de Broyden.

procedure BROYDEN(x_0, F, A_0)
    $x_k \leftarrow x_0$.
    while Not convergencia **do**
        $A_k s_k \leftarrow -F(x_k)$.
        $x_{k+1} \leftarrow x_k + s_k$.
        $y_k \leftarrow F(x_{k+1}) - F(x_k)$.
        $A_{k+1} \leftarrow A_k + \frac{y_k - A_k s_k}{s_k^T s_k} s_k^T$
    return x^*

En el algoritmo anterior podemos notar que ahora no se usa la matriz Jacobiana sino una aproximación A, aunque hay que tener en cuenta que para la primer iteración debemos tener ya precalculada una A_0 la cual por ejemplo podemos usar diferencias finitar para precalcular. En la presente tarea se usó la expresión analítica empleada en el método de Newton para obtener A_0.

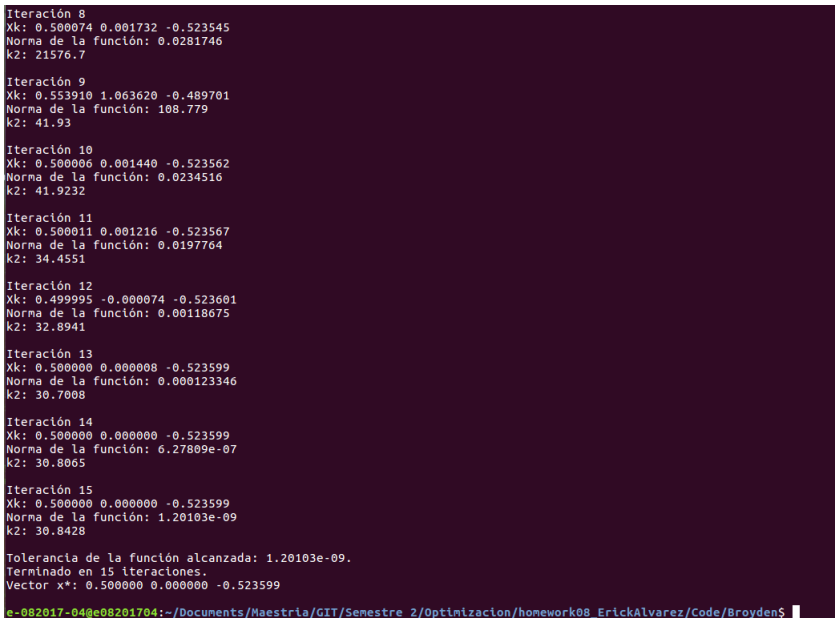
2.1. Ejecuciones y resultados

Para este algoritmo se hicieron pruebas con la misma función, puntos iniciales y criterios de parada del ejercicio pasado, por lo cual la matriz Jacobiana es la misma, además de que se monitorearon los mismos valores de interés para ver como se comportaba el algoritmo a lo largo de las iteraciones.
A continuación se mostrarán unas capturas de pantalla realizadas a la ejecución del programa con cada punto inicial.



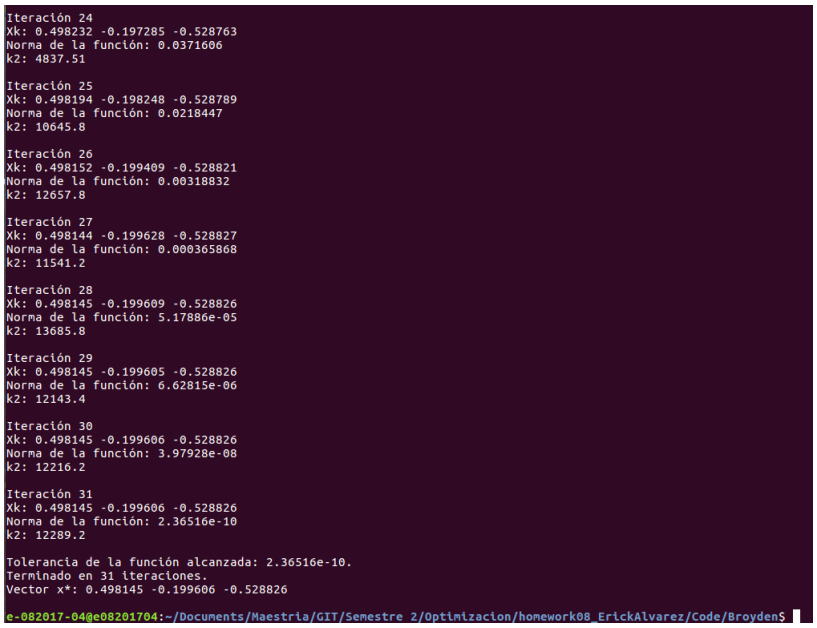
(e) Figura 5. Ejecución del algoritmo de Broyden para el primer punto inicial.

Vemos que para el primer punto inicial el cual fue $x_0 = (0, 0, 0)^T$ el algoritmo tardó 7 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.



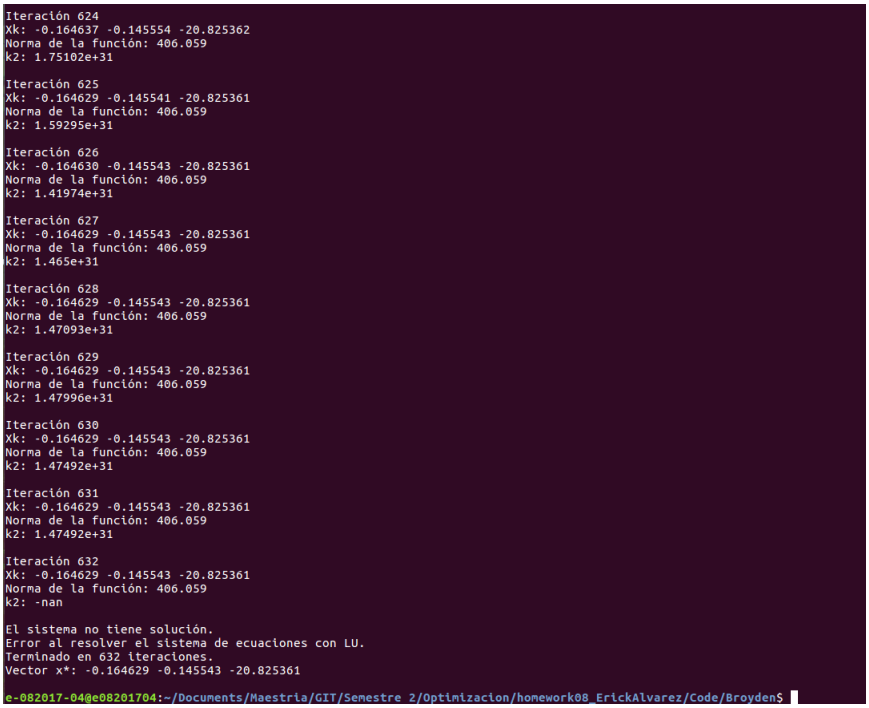
(f) Figura 6. Ejecución del algoritmo de Broyden para el segundo punto inicial.

Vemos que para el segundo punto inicial el cual fue $x_0 = (1{,}1, 1{,}1, -1{,}1)^T$ el algoritmo tardó 15 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.



(g) Figura 7. Ejecución del algoritmo de Broyden para el tercer punto inicial.

Vemos que para el tercer punto inicial el cual fue $x_0 = (-10, -10, 10)^T$ el algoritmo tardó 31 iteraciones en converger y este se detuvo debido a la tolerancia alcanzada.



(h) Figura 8. Ejecución del algoritmo de Broyden para el cuarto punto inicial.

Vemos que para el cuarto punto inicial el cual fue $x_0 = (3, -3, 3)^T$ el algoritmo no logró convergencia ya que en la iteración 632 no pudo resolver el sistema de iteraciones, y esto lo justificamos analizando el número de condición de la aproximación de la matriz Jacobiana, en donde se puede notar que dicho número es muy grande y recordamos que eso equivale a que una pequeña perturbación en el vector independiente hará que la resolución del sistema de ecuaciones de valores muy diferentes a los que debería dar, o incluso que no pueda resolver dicho sistema, como es el caso actual.

A continuación se muestra un resumen de los datos obtenidos en las ejecuciones realizadas mediante una tabla.

Cuadro 2: Resultados de la ejecución del algoritmo de Broyden a los 4 puntos iniciales.

Punto inicial	**Iteración final** k	x_k	$\|\|F(x_k)\|\|$	$k(J(x_k))$
(0, 0, 0)	7	(0.5, 0, -0.523)	7.01704e-13	6.68536
(1.1, 1.1, -1.1)	15	(0.5, 0, -0.523)	1.20103e-09	30.8428
(-10, -10, 10)	31	(0.498, -0.19, -0.528)	2.36516e-10	12289.2
(3, -3, 3)	632	(-0.164, -0.145, -20.82)	406.059	1.47492e+31

En la Tabla anterior se ve lo que anteriormente se mencionaba en el cuarto punto inicial, el número de condición resultó ser bastante alto en la última iteración y debido a eso no se pudo resolver el sistema de ecuaciones. Para los otros tres puntos vemos que si se logró llegar a un resultado en donde el vector x^* fue el mismo que en el algoritmo pasado.
Analizando el número de condición de la matriz Jacobiana en este algoritmo nos podemos percatar que pese a que se encontró la solución, este no fue tan pequeño como en el algoritmo de Newton, esto puede explicar el por qué se tardó más iteraciones el algoritmo de Broyden para encontrar la solución al sistema.

3. Ejercicio 5

Dada la función $f : R^n \to R$ y el cambio de variable $\hat{x} = Tx$, definimos la función

$$\hat{f}(\hat{x}) = f(T^{-1}\hat{x})$$

.

Usando la regla de la cadena, pruebe que

$$\nabla^2 \hat{f}(\hat{x}) = T^{-T}\nabla^2 f(x)T^{-1}$$

Solución : Usando regla la cadena y asumiendo que f es C^2 podemos encontrar primeramente $\frac{\partial \hat{f}(\hat{x})}{\partial \hat{x}_i}$ de la siguiente manera:

$$\frac{\partial \hat{f}(\hat{x})}{\partial \hat{x}_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}\frac{\partial x_j}{\partial \hat{x}_i} \tag{4}$$

Por otro lado, sabemos que:

$$\begin{aligned} x_i &= \sum_{j=1}^{n} T_{ij}^{-1}\hat{x}_j \\ \frac{\partial x_i}{\partial \hat{x}_j} &= T_{ij}^{-1} \end{aligned} \tag{5}$$

Podemos sustituir (5) en (4).

$$\frac{\partial \hat{f}(\hat{x})}{\partial \hat{x}_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} T_{ji}^{-1} \tag{6}$$

Para que el producto realizado en (6) esté bien definido podemos transponer la matriz T, asumiendo que esta tiene transpuesta.

$$\begin{aligned}\frac{\partial \hat{f}(\hat{x})}{\partial \hat{x}_i} &= \sum_{j=1}^{n} T_{ij}^{-T} \frac{\partial f}{\partial x_j} \\ \nabla \hat{f}(\hat{x}) &= T^{-T} \nabla f(x)\end{aligned} \tag{7}$$

Una vez encontrado el gradiente de $\hat{f}(\hat{x})$ podemos encontrar el Hessiano derivando (4) con respecto a algún $\hat{x}_k$ y aplicando regla de la cadena.

$$\begin{aligned}\frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \frac{\partial}{\partial \hat{x}_k}[\sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial \hat{x}_i}] \\ \frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \sum_{j=1}^{n} [\sum_{l=1}^{n} \frac{\partial^2 f}{\partial x_l x_j} \frac{\partial x_l}{\partial \hat{x}_k}] \frac{\partial x_j}{\partial \hat{x}_i} \\ \frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \sum_{j=1}^{n} [\sum_{l=1}^{n} \frac{\partial^2 f}{\partial x_l x_j} T_{lk}^{-1}] T_{ji}^{-1}\end{aligned} \tag{8}$$

En la última ecuación podemos usar la simetría del Hessiano de f para que el producto esté bien definido.

$$\begin{aligned}\frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \sum_{j=1}^{n} [\sum_{l=1}^{n} \frac{\partial^2 f}{\partial x_j x_l} T_{lk}^{-1}] T_{ji}^{-1} \\ \frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \sum_{j=1}^{n} T_{ji}^{-1} \nabla^2 f(x) T^{-1}\end{aligned} \tag{9}$$

De nuevo, asumiendo que la matriz T tiene transpuesta, la sustituimos por la misma para que el producto en (9) esté bien definido.

$$\begin{aligned}\frac{\partial^2 \hat{f}(\hat{x})}{\partial \hat{x}_k \hat{x}_i} &= \sum_{j=1}^{n} T_{ij}^{-T} \nabla^2 f(x) T^{-1} \\ \nabla^2 \hat{f}(\hat{x}) &= T^{-T} \nabla^2 f(x) T^{-1}\end{aligned} \tag{10}$$

4. Notas y comentarios finales.

En el presente reporte se analizó la implementación de dos algoritmos que resuelven sistemas de ecuaciones no lineales, el método de Newton y el método de Broyden. Los cuales son métodos iterativos que en cada paso resuelven un sistema de ecuaciones donde se ve involucrada la matriz Jacobiana, en el caso del método de Broyden, el único cambio que tiene con el método de Newton es que este trabaja con una aproximación de la Jacobiana que recalcula en cada paso.

Se hicieron pruebas de los dos algoritmos, ambos con la misma función vectorial y con los mismos puntos iniciales, en general se logró apreciar que el método de Newton produjo sus resultados en menor tiempo que el de Broyden, y en el último caso, vimos que el método de Broyden no pudo lograr la convergencia, esto debido a que la aproximación de la matriz Jacobiana se malcondicionó a tal punto que el sistema de ecuaciones no se pudo resolver.

Esto anterior nos lleva a la conclusión de que sin importar el problema que debemos resolver sea pequeño, se debe tener cuidado con la elección de los puntos iniciales, ya que algunos pueden provocar el problema que se apreció en las pruebas. De la misma forma se podría añadir un algoritmo que se encarge de condicionar la matriz en cada iteración si es que se necesita, esto para evitar problemas como el que se presenció.