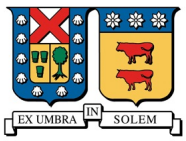

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Tarea N°1
Resolución de Ecuaciones No Lineales

Computación Científica

IPM458 - 2022

Departamento de Ing. Mecánica UTFSM

Martín Achondo Mercado

Rol: 201860005-9

Profesor: Franco Perazzo M.

13 de Mayo 2022

Resumen

En el presente trabajo se buscará implementar métodos numéricos para resolver 2 problemas planteados. El primero consiste en encontrar los esfuerzos principales para el estado de esfuerzos:

$$[\sigma] = \begin{bmatrix} -6 & 9 & -15 \\ 9 & 18 & 6 \\ -15 & 6 & -12 \end{bmatrix}$$

Para esto, lo que se realizó fue encontrar su polinomio característico y resolverlo mediante el método de Newton Rapshon. Además, se implemento una rutina de división sintética para poder ir reduciendo el polinomio y así encontrar las 3 raíces. Como resultado se obtuvo que:

$$\begin{aligned} \sigma_1 &= 21.0369 \text{ [kpsi]} \\ \sigma_2 &= 5.6706 \text{ [kpsi]} \\ \sigma_3 &= -26.7075 \text{ [kpsi]} \\ \tau_{max} &= 23.8722 \text{ [kpsi]} \end{aligned}$$

Para el segundo problema se realizó un análisis de la convergencia para el método de Newton y Broyden para el siguiente sistema no lineal para ecnontrar su solución:

$$\begin{cases} (x_1 + 3)(x_2^3 - 7) + 18 = 0 \\ \sin(x_2 e^{x_1} - 1) = 0 \end{cases}$$

Con ambos métodos se pudo llegar a la solución del problema, $x_1 = 0$ y $x_2 = 1$ de tal manera que el error sea menor al epsilon de la máquina. Con esto, se pudo contrastar la convergencia cuadrática del método de Newton, frente a la súper lineal del método de Broyden.

Todos los cálculos realizados en este trabajo se realizaron con variables de doble presición y los códigos fueron escritos en el lenguaje de Fortran.

Índice

1. Introducción **3**
1.1. Presentación del Problema 3

2. Metodología **4**
2.1. Problema 1 4
2.2. Problema 2 5

3. Resultados **6**
3.1. Problema 1 6
3.2. Problema 2 7

4. Análisis de Resultados **7**
4.1. Problema 1 7
4.2. Problema 2 8

5. Conclusión **9**

6. Referencias **9**

7. Anexos **10**
7.1. Código Pregunta 1 10
7.2. Código Pregunta 2 13

1. Introducción

En el presente trabajo se implementará un código para resolver los enunciados del problema presentado. Estos incluyen la resolución de una ecuación polinomial de grado 3 para encontrar los esfuerzos principales en un elemento de máquina y la resolución de un sistema de ecuaciones no lineal. Los códigos se elaborarán en Fortran dada su gran funcionalidad para métodos numéricos. En el documento presentarán los resultados obtenidos con sus respectivas tolerancias y un breve análisis de lo obtenido. Los códigos elaborados serán adjuntados en el anexo.

1.1. Presentación del Problema

En el presente se pide resolver lo siguiente:

1. El estado de esfuerzo en un punto de un elemento de máquina es (en kpsi) : σ_x 6, σ_y 18, σ_z 12, $\tau_{xy} = \tau_{yx} = 9$, $\tau_{xz} = \tau_{zx} = -15$ $\tau_{yz} = \tau_{zy} = = 6$, se pide encontrar los esfuerzos principales $\sigma_1 > \sigma_2 > \sigma_3$ calcular el valor del esfuerzo cortante máximo, utilizando cualquiera de las técnicas numéricas para encontrar raíces mediante un programa escrito en Fortran. Se debe recordar que $\sigma_1, \sigma_2, \sigma_3$ obtienen a partir de las raíces del siguiente polinomio cúbico:

$$\sigma^3 - C_2\sigma^2 - C_1\sigma - C_0 = 0 \tag{1}$$

Con las respectivas constantes:

$$\begin{aligned} C_2 &= \sigma_x + \sigma_y + \sigma_z \\ C_1 &= \tau_{xy}^2 + \tau_{xz}^2 + \tau_{zy}^2 - \sigma_x\sigma_y - \sigma_x\sigma_z - \sigma_z\sigma_y \\ C_0 &= \sigma_x\sigma_y\sigma_z + 2\tau_{xy}\tau_{xz}\tau_{zy} - \sigma_x\tau_{yz}^2 - \sigma_y\tau_{xz}^2 - \sigma_z\tau_{xy}^2 \end{aligned} \tag{2}$$

2. Desarrollar un programa computacional, escrito en Fortran, para resolver el siguiente sistema de ecuaciones no lineales:

$$\begin{cases} (x_1 + 3)(x_2^3 - 7) + 18 = 0 \\ \sin(x_2 e^{x_1} - 1) = 0 \end{cases} \tag{3}$$

Con el vector de partida: $\boldsymbol{x}_0 = \begin{bmatrix} -0.5 & 1.4 \end{bmatrix}^T$, utilizando:

   a) El método de Newton

   b) El método de Broyden

   c) Comparar la tasa de convergencia de ambos métodos calculando el error en cada iteración, conocida la solución exacta $\boldsymbol{x}^* = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Analizar el error para alcanzar el epsilon de la máquina

2. Metodología

2.1. Problema 1

Para el siguiente problema se pide resolver la siguiente ecuación no lineal, donde las raíces representas los esfuerzos principales del tensor de esfuerzos:

$$f(\sigma) = \sigma^3 - C_2\sigma^2 - C_1\sigma - C_0 = 0 \tag{4}$$

Para resolver esta ecuación se utilizará el método iterativo de Newton-Rapshon, el cual plantea una función iteración en el método de punto fijo. Entonces uno parte con una suposición inicial, lo reemplaza en la función iteración y así obtiene el valor de la siguiente iteración. El método entonces, sigue el siguiente algoritmo:

$$\sigma_{k+1} = \sigma_k - \frac{f(\sigma_k)}{f'(\sigma_k)} \tag{5}$$

Notar que se necesita la derivada de la función f, la cual corresponde a:

$$f'(\sigma) = 3\sigma^2 - 2C_2\sigma - C_1 \tag{6}$$

Es importante notar que este método entrega una raíz de la ecuación 4, por lo que en el mismo método se implementará la división sintética para reducir de grado el polinomio y así poder iterar para las otras raíces. Un esquema simple de este algoritmo es el siguiente implica que uno ingresa con una raíz encontrada y los coeficientes del polinomio y este devuelve los coeficientes del polinomio reducido. Lo dicho anteriormente se puede visualizar en la siguiente ecuación:

$$p_{n-1}(x) = \frac{p_n(x)}{(x-r)} \tag{7}$$

En donde r representa una raíz de p_n y p_n un polinomio de grado n, el cual corresponde a la función 4. De esta forma, se obtiene un polinomio de grado menor al que se le puede aplicar el método de Newton-Rapshon para encontrar una nueva raíz. Para cada iteración, se calculará el error relativo porcentual aproximado para tener una noción de la convergencia del método. Este se calculará como:

$$\epsilon_a = \frac{|x_k - x_{k-1}|}{x_k} \times 100\,\% \tag{8}$$

En donde k hace referencia a la iteración. Todo lo explicado anteriormente se puede visualizar en el siguiente pseudocódigo:

```
x0
A_i = [1,-C2,-C1,-C0]
call Newton_Rapshon(A_i,x_0,xs)
xf(1) = xs
call div_sintetica(A_i,xs,B_i)
FOR i=2:3
   call Newton_Rapshon(B_i,x_0,xs)
    xf(i) = xs
    call div_sintetica(B_i,xs,B_i)
END
```

Entonces, uno calcula la primera raíz, y con esta encuentra los nuevos coeficientes del polinomio al haber eliminado esa raíz, volviendo a la siguiente iteración para aplicar el método de newton a la función con coeficientes nuevos. El detalle del código se encuentra en el anexo.

Para la iteración, se probará con un valor de $x_0 = 1.1$, ya que como se demuestra en el siguiente gráfico, se encuentra cercano a las 3 raíces.

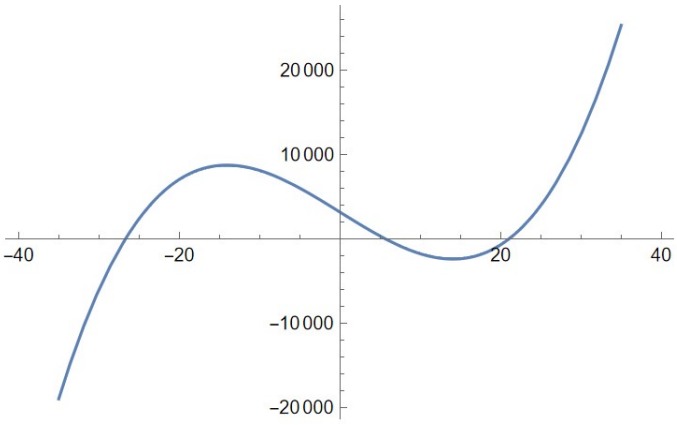

Figura 1: Gráfica de f

Además, para evaluar la convergencia, se cortará cuando el error aproximado sea menor a $0.00005\,\%$.

Importante notar que el algoritmo entregará 3 soluciones, de las cuales corresponden los esfuerzos principales. El esfuerzo cortate máximo se podrá calcular como:

$$\begin{aligned} \sigma_1 > \sigma_2 > \sigma_3 \\ \tau_{max} = \frac{|\sigma_1 - \sigma_3|}{2} \end{aligned} \tag{9}$$

2.2. Problema 2

Para el problema 2, se pide resolver el sistema no lineal presentado en la ecuación 3. Este sistema de ecuaciones se tratará en forma vectorial de la siguiente manera:

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} (x_1 + 3)(x_2^3 - 7) \\ \sin(x_2 e^{x_1} - 1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{10}$$

En donde $\boldsymbol{x} = (x_1, x_2)$. Este vector se iterará primero con el método de Newton:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{J}(\boldsymbol{x_k})^{-1}\boldsymbol{F}(\boldsymbol{x_k}) \tag{11}$$

Para evitar el cálculo de la inversa de la matriz jacobiana $\boldsymbol{J}$, se resolverá el siguiente sistema lineal:

$$\boldsymbol{J}(\boldsymbol{x_k})\boldsymbol{s}_k = -\boldsymbol{F}(\boldsymbol{x_k}) \tag{12}$$

En donde la nueva iteración de Newton queda como:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k \tag{13}$$

Para la segunda parte, se resolverá el mismo sistema pero usando el método de Broyden. Lo que se realiza es aproximar la matriz jacobiana en cada iteración.

$$\boldsymbol{B}_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \boldsymbol{F}(\boldsymbol{x_{k+1}}) - \boldsymbol{F}(\boldsymbol{x_k}) \tag{14}$$

Así, la iteración queda como:

$$\begin{aligned} \boldsymbol{B}_k \boldsymbol{s}_k &= -\boldsymbol{F}(\boldsymbol{x_k}) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{s}_k \\ \boldsymbol{y}_k &= \boldsymbol{F}(\boldsymbol{x_{k+1}}) - \boldsymbol{F}(\boldsymbol{x_k}) \\ \boldsymbol{B}_{k+1} &= \boldsymbol{B}_k + ((\boldsymbol{y}_k - \boldsymbol{B}_k \boldsymbol{s}_k)\boldsymbol{s}_k^T)/(\boldsymbol{s}_k^T \boldsymbol{s}_k) \end{aligned} \tag{15}$$

Para ambos casos se calculará error relativo de cada iteración con la norma 2 de los vectores de la siguiente manera. Donde el primero corresponde al real, y el segundo al aproximado.

$$\begin{aligned} \epsilon_r &= \sqrt{\frac{\|\boldsymbol{x}_k - \boldsymbol{x}_{real}\|}{\|\boldsymbol{x}_{real}\|}} \\ \epsilon_a &= \sqrt{\frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|}{\|\boldsymbol{x}_k\|}} \end{aligned} \tag{16}$$

Además, se usará el vector de partida $\boldsymbol{x}_0 = \begin{bmatrix} -0.5 & 1.4 \end{bmatrix}^T$. El punto de salida será cuando el error termine siendo menor al epsilon de la máquina.

Todos los cálculos en ambos problemas se realizarán con doble precisión, por lo que $\varepsilon = 2 \times 10^{-16}$. Para los sistemas lineales se usará la librería de LAPACK.

3. Resultados

3.1. Problema 1

Los valores para los esfuerzos y errores relativos aproximado asociados a cada raíz se presentan en la siguiente tabla:

Tabla 1: Resultados problema 1

Esfuerzo principal	Valor [kpsi]	ϵ_a
σ_1	21.0369	1.28E-7 %
σ_2	5.6706	3.82E-6 %
σ_3	-26.7075	1E-16 %

De aquí, el esfuerzo cortante máximo resulta:

$$\tau_{max} = 23.8722[\text{kpsi}] \tag{17}$$

3.2. Problema 2

Se presentan los resultados para ambos métodos:

Tabla 2: Resultados problema 2

Método	x_1	x_2	Iteraciones	ϵ_r
Newton	0.0000000000	1.0000000000	4	1.42E-17 %
Broyden	0.0000000000	1.0000000000	9	4.59E-17 %

Se presentan tablas con las iteraciones realizadas por cada método:

Tabla 3: Iteraciones método de Newton

Iteración	x_1	x_2	ϵ_r
1	-5.5315124128542392E-002	1.0280665787639522	6.2028185535844813E-002 %
2	-1.4035088962023601E-004	1.0001574042607961	2.1088971889110481E-004 %
3	-1.7790760992463005E-008	1.0000000055513785	1.8636764193019655E-008 %
4	-1.4239920862545272E-017	1.0000000000000000	1.4239920862545272E-017 %

Tabla 4: Iteraciones método de Broyden

Iteración	x_1	x_2	ϵ_r
1	-5.5315124128542392E-002	1.0280665787639522	6.2028185535844813E-002 %
2	5.0995249771831896E-004	1.0001236440189114	5.2472792315800794E-004 %
3	-2.3384801388646212E-004	1.0000765609106039	2.4606191625512830E-004 %
4	-4.0826287826436884E-005	1.0000135979113922	4.3031255755759909E-005 %
5	-1.3275090836813072E-007	1.0000000453383624	1.4027961640050070E-007 %
6	-5.3912129317394976E-010	1.0000000001806635	5.6858690036528383E-010 %
7	1.6679458404712356E-012	0.99999999999944278	1.7585614855125687E-012 %
8	-8.3623444740614386E-016	1.0000000000000002	8.6521203043240912E-016 %
9	4.1593590813336494E-017	1.0000000000000000	4.1593590813336494E-017 %

4. Análisis de Resultados

4.1. Problema 1

Para los resultados obtenidos en la pregunta 1, se pudo obtener los valores de los esfuerzos principales para el estado de esfuerzo planteado. Se nota que los esfuerzos están dentro del rango por lo que concuerda con la física. En las 3 iteraciones realizadas se pudo llegar a errores aproximados menores a 1E-6, lo que puede implicar convergencia. De todas formas, al evaluar los resultados en la función, se obtiene un valor aprox. de 1E-13. Lo que implica que efectivamente son raíces.

El código elaborado para esta pregunta puede ser adaptado para cualquier estado de esfuerzo que se plantee solo variando el punto de partida y las constantes. Esto se debe a que el código al incluir la división sintética, buscará todas las raíces reduciendo el polinomio característico.

4.2. Problema 2

Se nota que para los resultados obtenidos en esta pregunta, el método de Newton converge más rápido que el método de Broyden. Esto se demuestra debido a que se requirieron solo 4 iteraciones para alcanzar el error de la máquina, en comparación al segundo que tomó 9. Además, se nota que ambos llegan a errores reales relativos del orden de 1E-17, lo que es menor al epsilon de la máquina. De esta manera, existe convergencia en ambos.

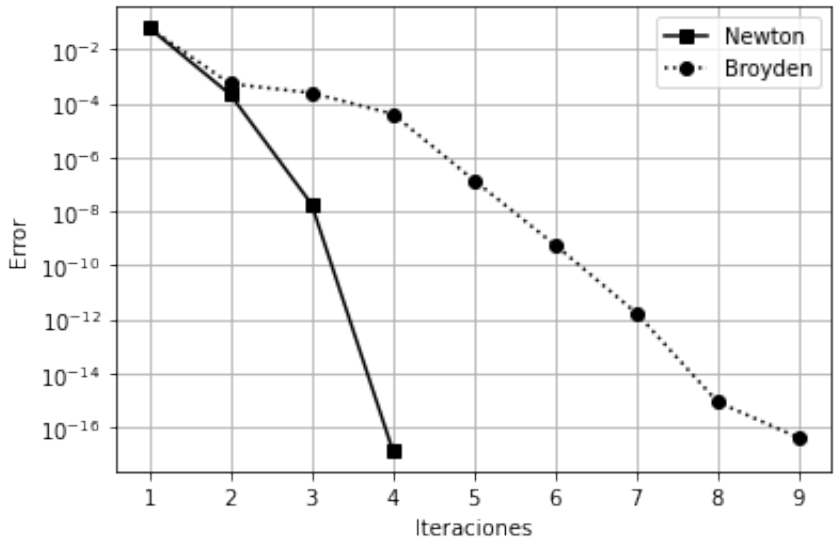

Figura 2: Error por cada iteración para ambos métodos

En la figura anterior se nota claramente como el método de Newton tiene una mucho mayor convergencia que el método de Broyden. De la figura se obtiene que el método de Newton tiene convergencia cuadrática y Braydon superlineal. Sin embargo, ambos métodos llegan al resultado esperado.

Es importante destacar que el método de Newton es más costoso ya que constantemente hay que estar evaluando la matriz jacobiana, la cual a veces puede tener expresiones muy extensas para evaluarla. Por esto, el método de Broyden ofrece la ventaja de calcularla una vez e iterar sobre ella para cada iteración.

5. Conclusión

Teniendo ya las simulaciones realizadas, se nota la gran importancia que tienen los métodos numéricos para resolver y modelar fenómenos físicos.

En el primer problema se pudo modelar el estado de esfuerzo para obtener los esfuerzos principales. Este se resolvió utilizando el polinomio característico y para encontrar sus raíces se combino el método de Newton-Rapshon con la división sintética. Es importante destacar que se llegó a errores menores a 1E-6, lo que implica una convergencia en un par de iteraciones. De todas formas, en este problema se están obteniendo valores propios de una matriz, por lo que este código se puede utilizar para obtener valores propios de matrices simétricas. Sin embargo, por eficiencia, sería recomendable utilizar otro método para matrices de mayor orden. Además, este código puede ser utilizado para encontrar las raices reales de cualquier polinomio de orden arbitrario.

Para el segundo problema, se pudo notar la diferencia en la tasa de convergencia en los dos métodos utilizados, Newton y Braydon para resolver sistemas no lineales. Para el primero, se pudo visualizar la convergencia cuadrática que tiene llegando al epsilon de la máquina (doble precisión) en solo 4 iteraciones. Para el segundo método, se pudo llegar al epsilon de la máquina pero en 9 iteraciones, ya que tiene una convergencia súper lineal. Este código puede ser utilizado para resolver cualquier sistema no lineal de 2 ecuaciones. Cabe destacar que se podría aumentar la eficiencia del código analizando el paso en el que se resueleve el sistema lineal. Sería interesante revisar si el tiempo de código varía utilizando distintos métodos incluidos en la librería de Lapack.

6. Referencias

[1] Steven C. Chapra Raymond P. Canale. (2007). Métodos numéricos para ingenieros. Mexico: The McGraw-Hill.

[2] Beer, F; Johnston, E.D.; DeWolf, J; Mazurek, D. (2010) Mecánica de Materiales, 5a edición: Mc Graw-Hill.

[3] Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney and Sorensen. LAPACK Users'Guide. Siam

[4] Perazzo, F. Solución numérica de sistemas de ecuaciones no-lineales, IPM-458. (2022)

7. Anexos

7.1. Código Pregunta 1

```
1
2  !Programa para calcular las raices de un polinomio aplicado
3  ! a el estado de esfuerzo  para obtener los esfuerzos principales.
4
5  program P1
6      implicit none
7      real(kind=8)::C0,C1,C2
8      real(kind=8) :: es,val(3),x0,K(4),kk(4),f,xx,kcopy(4),ea(3),eaf
9      integer :: n,i,z
10
11     es = 0.00005
12     x0 = 1.1567
13
14     call Constantes(C0,C1,C2)
15
16     K = (/-C0,-C1,-C2,1d0/)  !coeficientes polinomio de partida
17     n = size(K)
18
19     call newton_rapshon(es,x0,K,n,0,xx,eaf) !primera raiz
20     val(1) = xx
21     ea(1) = eaf
22     call divsin(K,n,val(1),kk)  !actualizacion de coeficientes
23     z = 1
24     do i = 2,n-1   !loop para otras raices y actualizacion de
           coeficientes
25         call newton_rapshon(es,x0,kk,n,z,xx,eaf)
26         val(i) = xx
27         ea(i) = eaf
28         kcopy = kk
29         call divsin(kcopy,n,val(i),kk)
30         z = z + 1
31     end do
32
33     write(*,*)
34     write(*,*) 'Verificacion:'
35     write(*,*) '  Raiz ecuacion,          Funcion evaluada,
                    Error relativo aproximado porcentual'
36     do i = 1,n-1
37         call evalf(val(i),K,n,0,f)
38         write(*,*) val(i),f,ea(i)
39     end do
40
41     write(*,*)
42     write(*,*)
```

```
43
44      contains
45      !subrutina para evaluar las constantes
46          subroutine Constantes(C0,C1,C2)
47              implicit none
48              real(kind=8)::ox,oy,oz,txy,tyx,txz,tzx,tzy,tyz,C0,C1,C2
49              ox = -6
50              oy = 18
51              oz = -12
52              txy = 9
53              tyx = txy
54              txz = -15
55              tzx = txz
56              tyz = 6
57              tzy = 6
58              C2 = ox + oy + oz
59              C1 = txy**2 + tyz**2 + tzx**2 - ox*oy - oy*oz - oz*ox
60              C0 = ox*oy*oz + 2*txy*tyz*tzx  - ox*tyz**2 - oy*tzx**2 -
                    oz*txy**2
61          end subroutine
62  end program P1
63
64  !subrutina para evaluar un polinomio dependiendo de los coeficientes
       asociados (input)
65  subroutine evalf(x,K,n,z,f)
66      implicit none
67      real(kind=8)::x,sum,f
68      integer::n,j,z
69      real(kind=8), intent(in)::K(n)
70      sum = 0
71      do j = 1,n-z
72          sum = sum + K(j+z)*x**(j-1)
73      end do
74      f = sum
75  end subroutine
76
77  !subrutina para evaluar la derivada un polinomio dependiendo de los
       coeficientes asociados (input)
78  subroutine evaldf(x,K,n,z,df)
79      implicit none
80      real(kind=8)::x,sum,df
81      integer::n,j,z
82      real(kind=8), intent(in)::K(n)
83      sum = 0
84      do j = 1,n-1-z
85          sum = sum + (j)*K(j+1+z)*x**(j-1)
86      end do
87      df = sum
```

```
88  end subroutine
89
90  !subrutina para division sintetica
91  subroutine divsin(K,n,r,kk)
92      integer::n,i
93      real(kind=8)::K(n)
94      real(kind=8) :: r,b(n),kk(n)
95      b(n) = K(n)
96      i = n-1
97      do while(i.ge.1)
98          b(i) = K(i) + r*b(i+1)
99          i = i-1
100     end do
101     do i=1,n
102         kk(i) = b(i)
103     end do
104 end subroutine
105
106 !subrutina para el metodo de newton rapshon
107 subroutine newton_rapshon(es,x0,K,n,z,xx,eaf)
108     implicit none
109     real(kind=8)::xx
110     integer(kind=4) :: iter
111     integer(kind=4), parameter :: imax=500
112     real(kind=8) :: x0,xr,xrold,es,ea,fr,f,df,eaf
113     integer::n,z
114     real(kind=8), intent(in)::K(n)
115     xrold = x0
116     xr = xrold
117     iter = 0
118     do while (iter.lt.imax)
119         xrold = xr
120         iter = iter + 1
121         call evalf(xrold,K,n,z,f)
122         call evaldf(xrold,K,n,z,df)
123         fr = f/df
124         xr = xrold - fr
125         if (xr.ne.0) then
126             ea = abs((xr-xrold)/xr)*100
127         end if
128         if(ea.lt.es.or.iter.gt.imax) then
129             exit
130         end if
131     end do
132     xx = xr
133     eaf = ea
134 end subroutine
```

7.2. Código Pregunta 2

```
1
2  !Programa para resolver un sistema no lineal de ecuaciones
3  !utilizando el metodo de newton y broyden
4
5  program P2
6      implicit none
7      real(kind=8) :: x0(2),xreal(2),xf(2),es
8      real(kind=8) :: a
9
10     es = epsilon(a)
11
12     x0 = (/-0.5,1.4/)
13     xreal = (/0d0,1d0/)
14
15     write(*,*)
16     write(*,*)
17     write(*,*) '======= METODO DE NEWTON ======='
18     write(*,*)
19
20     call newton(es,x0,xreal,xf)
21     write(*,*)
22     write(*,*) 'Solucion:'
23     write(*,'(3F16.10)') xf
24
25     write(*,*)
26     write(*,*)
27     write(*,*) '======= METODO DE BROYDEN ======='
28     write(*,*)
29
30     call broyden(es,x0,xreal,xf)
31     write(*,*)
32     write(*,*) 'Solucion:'
33     write(*,'(3F16.10)') xf
34
35 end program P2
36
37 ! subrutina para evaluar la funcion vecotrial
38 subroutine f_k(x,f)
39     real(kind=8)::x(2)
40     real(kind=8) :: f(2)
41     f(1) = (x(1)+3)*(x(2)**3-7)+18
42     f(2) = sin(x(2)*exp(x(1))-1)
43 end subroutine
44
45 !subrutina para evaluar la matriz jacobiana
46 subroutine jacb(x,J)
```

```
47      real(kind=8) ::x(2)
48      real(kind=8) :: J(2,2)
49      J(1,1) = x(2)**3-7
50      J(1,2) = (3*x(2)**2)*(x(1)+3)
51      J(2,1) = cos(x(2)*exp(x(1))-1)*exp(x(1))*x(2)
52      J(2,2) = cos(x(2)*exp(x(1))-1)*exp(x(1))
53  end subroutine
54
55  !subrutina para calcular el error norma2
56  subroutine norma2(x,xreal,n,norm)
57      integer :: n,i
58      real(kind=8) :: norm,n1,n2
59      real(kind=8) :: x(n),xreal(n)
60      n1 = 0
61      n2 = 0
62      do i=1,n
63          n1 = n1 + (x(i)-xreal(i))**2
64          n2 = n2 + xreal(i)**2
65      end do
66      norm = sqrt(n1/n2)
67  end subroutine
68
69  !subrutina para calcualr la norma de un vector
70  subroutine norma_vec(x,n,norm)
71      integer :: n,i
72      real(kind=8) :: norm,n1
73      real(kind=8) :: x(n)
74      n1 = 0
75      do i=1,n
76          n1 = n1 + x(i)**2
77      end do
78      norm = n1
79  end subroutine
80
81  !subrutina para utilizar el metodo de newton
82  subroutine newton(es,x0,xreal,xf)
83      implicit none
84      integer(kind=4), parameter :: imax=500,n=2
85      real(kind=8)::x0(n),xreal(n),xf(n)
86      integer(kind=4) :: iter
87      real(kind=8) :: xr(n),xrold(n),es,b(n),Acopy(n,n),norm,fr(n),J(n,n
            ),normv,norm2
88      xrold = x0
89      xr = xrold
90      iter = 0
91      write(*,*) "      Iteracion , Vector Solucion :  x1 ,  x2  , Error
            norma2 real  , Error norma2 aproximado"
92      do while (iter.lt.imax)
```

```
93          xrold = xr
94          iter = iter + 1
95          call f_k(xrold,fr)
96          b = -fr
97          call jacb(xrold,J)
98          Acopy = J
99          call sist_lineal_2d(Acopy,b)
100         xr = xrold + b
101         call norma2(xr,xreal,n,norm)
102         call norma_vec(xr,n,normv)
103         if(normv.ne.0) then
104             call norma2(xrold,xr,n,norm2)
105         end if
106         write(*,*) iter,xr(1),xr(2),norm,norm2
107         if(norm.lt.es.or.iter.gt.imax) then
108             exit
109         end if
110     end do
111     xf = xr
112 end subroutine
113
114 !subrutina para utilizar el metodo de broyden
115 subroutine broyden(es,x0,xreal,xf)
116     implicit none
117     integer(kind=4), parameter :: imax=500,n=2
118     real(kind=8)::x0(n),xreal(n),xf(n)
119     integer(kind=4) :: iter,i,j
120     real(kind=8) :: xr(n),xrold(n),es,b(n),Acopy(n,n),norm,fr(n),yk(n)
        ,Bk(n,n),frk(n),sk(n),fksk(n,n),skk,normv,norm2
121     xrold = x0
122     xr = xrold
123     iter = 0
124     write(*,*) "      Iteracion , Vector Solucion :  x1 ,  x2  , Error
        norma2 real  , Error norma2 aproximado"
125     write(*,*)
126     call jacb(xrold,Bk)
127
128     do while (iter.lt.imax)
129         xrold = xr
130         iter = iter + 1
131         call f_k(xrold,fr)
132         b = -fr
133         sk = b
134         Acopy = Bk
135         call sist_lineal_2d(Acopy,sk)
136         xr = xrold + sk
137         call norma2(xr,xreal,n,norm)
138         call norma_vec(xr,n,normv)
```

```
139         if(normv.ne.0) then
140             call norma2(xrold,xr,n,norm2)
141         end if
142         write(*,*) iter,xr(1),xr(2),norm,norm2
143         if(norm.lt.es.or.iter.gt.imax) then
144             exit
145         end if
146 
147         call f_k(xr,frk)
148 
149         yk = frk - fr
150         skk = 0
151         do i=1,n
152             do j=1,n
153                 fksk(i,j) = (frk(i))*sk(j)
154             end do
155             skk = skk + sk(i)**2
156         end do
157 
158         Bk = Bk +(fksk)/skk
159 
160     end do
161     xf = xr
162 
163 end subroutine
164 
165 ! subrutina pararesolver el sistema lineal utilizando lapack
166 subroutine sist_lineal_2d(A,b)
167     real(kind=8) :: A(2,2),b(2)
168     integer :: IPIV(2),NRHS,LDA,LDB,INFO
169     NRHS = 1
170     LDA = 2
171     LDB = 2
172     call DGESV(2,NRHS,A,LDA,IPIV,b,LDB,INFO)
173 end subroutine
```