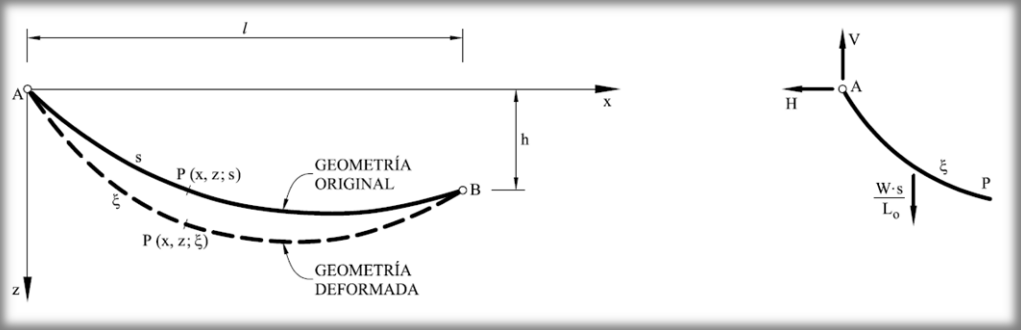
CIMNE International Center for Numerical Methods in Engineering

Métodos Numéricos avanzados

Trabajo de la asignatura

Máster en Métodos Numéricos para Cálculo y Diseño en Ingeniería



Quevedo González, Fernando José
Febrero 2014

INDICE

INTRODUCCIÓN

En el trabajo práctico se plantea la resolución del problema de la catenaria de un cable suspendido de dos apoyos a diferente altura y sometido, únicamente a su propio peso.

Dicho problema físico encuentra su representación matemática en un sistema de ecuaciones diferenciales de primer orden, cuya resolución conduce a un sistema de ecuaciones no lineales. Es último sistema el que, con ayuda de diferentes métodos numéricos, se resuelve en el presente documento.

Los métodos a emplear pueden encuadrarse en tres grupos: los métodos de Newton-Raphson, los métodos cuasi-Newton y los métodos Newton-secantes. Los métodos cuasi-Newton presentan un esquema que ofrece un menor coste numérico y, constituyendo una optimización del método de Newton-Raphson. Finalmente, los métodos Newton-secantes son simplificaciones del método Newton-Raphson, buscando en todo caso, una mayor simplicidad y un menor coste del cálculo/actualización de la matriz de rigidez (que, en teoría ofrecerá una mayor velocidad de convergencia).

A pesar de que la solución del problema por cualquiera de estos métodos numéricos, conduce a un resultado igualmente válido del problema planteado, se ha elegido implementar los diferentes métodos para, así poder lograr una mejor comprensión de los mismos, comparar la mecánica de resolución de cada uno y la eficiencia, tanto a nivel de rapidez computacional (o número de iteraciones hasta la convergencia), como a nivel de posibles errores en la solución.

Con respecto al problema planteado, estableciendo ecuaciones de equilibrio y operando las mismas, puede reducirse a dos ecuaciones no lineales, las cuales constituyen el problema numérico a resolver. Para ello, es necesario el cálculo del jacobiano del sistema, que proporcionará la matriz de rigidez tangente del sistema, necesaria para el cálculo. El sistema se resuelve en términos de las reacciones en los extremos y, posteriormente se obtiene, a partir de las mismas, la deformada del cable así como la tensión del mismo, haciendo uso de las ecuaciones básicas de equilibrio del sistema.

PROGRAMACION DE LA SOLUCION

Explicación general del programa

Para la implementación del esquema de solución, que permita a su vez la utilización de diferentes métodos, se ha elegido un esquema en el que se tiene una función principal (main), de la que "cuelgan" a modo de subrutinas los diferentes métodos.

Además, se han definido una serie de subrutinas adicionales que realizan tareas repetitivas y comunes a la mayoría de los métodos, con el objetivo de simplificar el programa, logrando una mejor estructuración del mismo. En la tabla 1 se resumen las diferentes subrutinas utilizadas.

	Subrutina	Descripción
	Main	Contiene el programa principal.
Subrutinas adicionales	**Iniciales**	Inicializa la solución.
	Coef	Calcula los coeficientes de las ecuaciones del problema.
	Tangente	Calcula la matriz de rigidez tangente.
	Crout	Factoriza (método de Crout) la matriz de rigidez tangente.
	Incr	Obtiene le incremento en la solución, resolviendo el sistema matricial de ecuaciones.
	Error	Calcula el error cometido en la solución.
	Deftens	Calcula la deformada (catenaria) y la tensión del cable.
Subrutinas de los métodos	**Fullnewton**	Implementación del método Newton-Raphson.
	Newtonmod	Implementa el método de la rigidez inicial.
	Cuasinewton	Implementación de los métodos cuasi-Newton. Hace llamadas a las funciones Bfgs y Broy.
	Bfgs	Implementación del método BFGS.
	Broy	Implementación del método de Broyden.
	Newtonsecantes	Implementación de los métodos Newton- secantes. Hace llamadas a las funciones Bfgs_secante y Broy_secante.
	Bfgs_secante	Implementación del método BFGS en su versión secante.
	Broyden_secante	Implementación del método de Broyden en su versión secante.

Tabla 1 – Subrutinas que componen el programa.

El esquema general de todos los métodos presentados es similar (Fig.1); por tanto, el planteamiento de todas las subrutinas es asimismo similar: partiendo de una solución inicial, se calculan los coeficientes de las ecuaciones que describen el problema y, a partir de éstos, la matriz de rigidez tangente. Con dicha matriz (en su forma invertida o factorizada) se resuelve el sistema matricial de ecuaciones, calculándose el nuevo incremento de la solución (Δu). Con dicho incremento, se calcula el error cometido (r) y en base a dicho error se toma la decisión de finalizar el programa (si es inferior al valor fijado como referencia, r*), o continuar con el siguiente incremento o iteración, en cuyo caso, se recalculan los coeficientes y se recomienza el esquema.

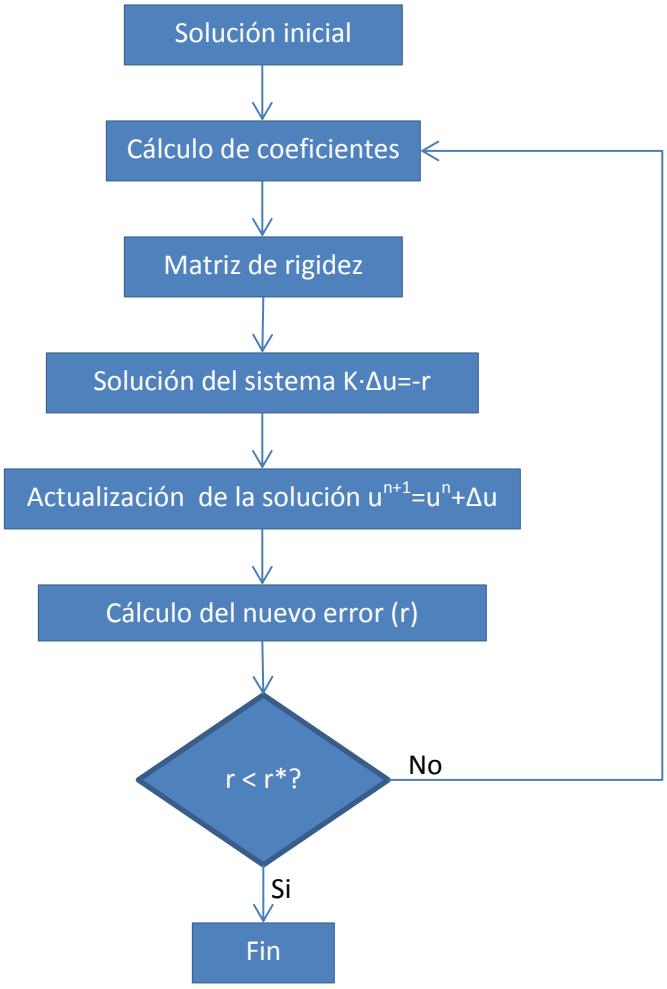


Fig. 1 – Algoritmo de los métodos.

Las diferencias entre los métodos radican en el cálculo de la matriz de rigidez tangente y su actualización. Así, el método de Newton-Raphson calcula y actualiza dicha matriz en cada incremento de carga, mientras un método Newton modificado, como es el método de la rigidez inicial (presentado en este trabajo) sólo calcula la matriz de rigidez en la primera iteración.

Los métodos cuasinewton plantean modificaciones al cálculo de la matriz de rigidez tangente, de manera que, generalmente plantean una actualización de la misma mucho menos costosa que el propio cálculo de dicha matriz, la cual es directamente calculada únicamente para la primera iteración.

De igual modo, los métodos secantes presentados únicamente calculan la matriz de rigidez tangente inicial, modificando las fórmulas de actualización para obtener una actualización de dicha matriz que realmente proporcione una matriz secante, en base a las soluciones actual y anterior. Es decir, lo que realmente cambia es el paso de resolución del sistema de ecuaciones antes mencionado, que es transformado para resolver el problema secante (en lugar del tangente).

A continuación se procederá a explicar más detalladamente cada una de las subrutinas, se hará una breve descripción de su esquema de cálculo (en el caso de las subrutinas que implementan los diferentes métodos) y se presentará y explicará el código del programa.

Cabe destacar que durante todo el cálculo se utilizan variables de doble precisión.

Subrutina “main”

Esta es la rutina principal del programa. En ella se hacen llamadas a las subrutinas de los diferentes métodos, así como a las subrutinas de inicialización. Además se almacenan los valores iniciales en las variables correspondientes y se realiza la escritura de resultados en ficheros. Cabe destacar que los bucles de cálculo de las diferentes iteraciones están incluidos en esta rutina principal, constituyendo las diferentes subrutinas de los métodos, como se verá, una única iteración de los mismos.

Se puede ver que se utilizan dos ficheros de resultados: uno que contiene la solución y el error en las diferentes iteraciones, y otro que va contener el resultado final en términos de la deformada y la tensión en el cable.

Por otro lado, en ocasiones las variables utilizadas difieren en nombre de las utilizadas al interior de las diferentes subrutinas, lo cual tiene el propósito de permitir el empleo global (común a diferentes subrutinas) de dichas variables.

```
program main
 integer kk
 real*8 l,h,Lo,W,E,A,tol,emax
 real*8 :: c(9),u(2),qe(2),r(2),Low(2,2),Up(2,2),x(1000,4)
 real*8 :: Hk(2,2),Ho(2,2),prod(2,2),s(2),sg(2),sga(2),y(2)

 open (unit=15,file='iteraciones.dat',status='unknown')
! DATOS PROBLEMA
 E=1.5d7
 A=2d-4
 l=20d0
 h=8.5d0
 Lo=28d0
 W=0.85d0
 tol=1d-10
 maxe=2d0*tol  !Me aseguro que entro en el bucle de cálculo
 kk=100
! CALCULO COEF QUE NO CAMBIAN CON LA SOLUCION
 c(5)=Lo/W
 c(7)=Lo/(E*A)

! INICIALIZO "VECTOR DE CARGAS Y VARIABLES
 qe(1)=l      !Inicializo el vector de "cargas externas aplicadas"
 qe(2)=h
 n=0          !Inicializo el contador de iteraciones
!  SOLUCION INICIAL
 call iniciales (l,Lo,W,u)            !Calculo valores iniciales
 call error (W,qe,c,u,r,emax)         !Evaluo error solucion inicial
!  ITERACIONES
 kk=1
 write (15,*) kk,u,emax  !-->"iteraciones.dat" iter. act.,sol.,error
do while (abs(emax).gt.tol)
  call fullnewton (W,c,qe,u,r,emax)
! call newtonmod (W,c,qe,u,r,Low,Up,emax)    !K= cte. pasa de
                                             !iter. en iter. (Low, Up)
! call cuasinewton  (tol,W,c,qe,u,r,Hk,Ho,prod,emax)
! call newtonsecantes_v2 (Low,Up,c,u,r,qe,s,sg,sga,y,W,tol,emax)
  write (15,*) kk,u,emax ! -->"iteraciones.dat" iter. act.,sol.,error
  kk=kk+1
 enddo

 open (unit=16,file='resultados.dat',status='unknown')
 call deftens (c,u,x,1000,Lo,W)      !Calculo la deformada y la tensión
 kk=1
 do while (kk.le.1000)
  write (16,*) x(kk,1),x(kk,2),x(kk,3),x(kk,4) !"resultados.dat":coord.s
  kk=kk+1                                      !del cable,c.x,c.z,tens.
 enddo
 close (15)
 close(16)
 stop
end
```

Subrutina "iniciales"

Se trata de una subrutina simple, cuyo código podría estar incluido directamente en la función "main" (ya que sólo se hace una única llamada), pero que ha sido creada para dotar de mayor estructura y claridad al código.

En ella se calculan los valores iniciales de la solución del problema (almacenados en el vector de 2 componentes (u), a partir de los datos: peso propio del cable (W), longitud inicial del cable (Lo) y longitud entre apoyos (l).

```
subroutine iniciales (l,Lo,W,u)
 real*8 Lo,W,l
 real*8 :: u(2)

 u(1)=(W*l/(2d0*Lo))*dsqrt(l/(6d0*(Lo-l)))
 u(2)=W/2d0

 return
end
```

Se puede observar que la subrutina trabaja con 3 variables reales y un vector de dos componentes reales. Las variables reales constituyen datos que se "pasan" desde el programa "main" a la subrutina. Por su parte el vector (u) va a almacenar la solución durante todo el algoritmo (reacciones horizontal, H, y vertical, V), y aquí se almacena en el mismo la solución inicial.

Subrutina "coef"

Las ecuaciones matemáticas que representan el problema y que permiten el cálculo de la matriz de rigidez tangente, presentan multitud de términos en común. Es por ello que, para simplificar el programa, dichos términos se han agrupado en coeficientes que se almacenan en un vector.

Este vector posee componentes que son constantes y otras que varían con la solución y, por tanto han de ser calculados en cada iteración. Es precisamente el cálculo de estas componentes variables con la solución el que se realiza en esta subrutina.

Los coeficientes calculados son

$$c(1) = \frac{V}{H}$$

$$c(2) = \sqrt{1 + \left(\frac{V}{H}\right)^2}$$

$$c(3) = \frac{V - W}{H}$$

$$c(4) = \sqrt{1 + \left(\frac{V - W}{H}\right)^2}$$

$$c(5) = \frac{L_0}{W}$$

$$c(6) = \frac{H \cdot L_0}{W}$$

$$c(7) = \frac{L_0}{E \cdot A}$$

$$c(8) = ln\left(\frac{V}{H} + \sqrt{1 + \left(\frac{V}{H}\right)^2}\right)$$

$$c(9) = ln\left(\frac{V - W}{H} + \sqrt{1 + \left(\frac{V - W}{H}\right)^2}\right)$$

Donde L_0 y W son la longitud inicial y el peso propio del cable ya mencionados, E es el módulo de elasticidad; A, el área de la sección transversal del cable. Por su parte, H y V son las reacciones horizontal y vertical, respectivamente.

De estas componentes, las únicas que no varían con la solución son la componente c(5) y la c(7). El cálculo de estas dos componentes se realiza en la rutina principal “main”.

```
subroutine coef (W,u,c)
 real*8 W
 real*8 :: c(9),u(2)

 c(1)=u(2)/u(1)
 c(2)=dsqrt(1+c(1)*c(1))
 c(3)=(u(2)-W)/u(1)
 c(4)=dsqrt(1+c(3)*c(3))
 c(6)=u(1)*c(5)

! EVITO ERRORES DE REDONDEO AL CALCULAR
SINH-1 (PDAD SINH-1(-X)=-SINH-1(X))

 if (c(1).lt.0d0) then
  c(8)=-1d0
 else
  c(8)=1d0
 endif

 if (c(3).lt.0d0) then
  c(9)=-1d0
 else
  c(9)=1d0
 endif

 c(8)=c(8)*dlog(abs(c(1))+c(2))
 c(9)=c(9)*dlog(abs(c(3))+c(4))

 return
end
```

En este caso se tienen dos vectores de variables reales y una variable real. Sin embargo, ninguna de estas variables es nueva, sino que han sido ya presentadas en las subrutinas anteriores, cobrando aquí el mismo significado.

Se puede observar que se utilizan las fórmulas de cálculo aproximado del seno hiperbólico, así como la propiedad que permite extraer el signo negativo del argumento, para evitar errores de redondeo en el cálculo, tal como se sugiere en el enunciado.

Subrutina “tangente”

Esta subrutina calcula la matriz de rigidez tangente a partir de los coeficientes previamente calculados. Asimismo, según las necesidades del método de solución empleado, la subrutina invierte dicha matriz tangente o hace una llamada a la subrutina “crout” para factorizar la matriz.

```
subroutine tangente (c,K,L,Up,flag1)
 integer flag1
 real*8 val,det
 real*8 :: c(9),K(2,2),L(2,2),Up(2,2)

! CALCULO MATRIZ DE RIGIDEZ TANGENTE
 K(1,2)=c(5)*(1d0/c(2)-1d0/c(4))
 K(2,1)=-c(5)*(c(2)-c(4)-c(1)*c(1)/c(2)+c(3)*c(3)/c(4))
 K(2,2)=c(7)+c(5)*(c(1)/c(2)-c(3)/c(4))
 K(1,1)=c(5)*(c(8)-c(9))-K(2,2)+2d0*c(7)

!CALCULO LA INVERSA / FACTORIZO (SEGUN EL METODO)
 if (flag1.eq.1) then    !flag1=1 : he de calcular la inversa
  det=K(1,1)*K(2,2)-K(1,2)*K(2,1)
  K=K/det
  val=K(1,1)
  K(1,1)=K(2,2)
  K(2,2)=val
 else
  call crout (K,L,Up)    !flag1=0 : he de factorizar K=L·U
 endif

 return
end
```

En la subrutina se utiliza una variable entera (flag1) como variable lógica de control, que permitirá cambiar entre el cálculo de la inversa (flag1=1) y la factorización de la matriz de rigidez (K). Dicha matriz, así como las matrices triangulares inferior (L) y superior (Up) son utilizadas como variables que almacenan el resultado de esta subrutina y son "devueltas" para poder proseguir con el cálculo de la solución.

Además se emplean las variables adicionales "val" y "det", necesarias para el cálculo de la inversa cuando sea necesario.

Subrutina "crout"

Esta subrutina factoriza la matriz de rigidez como el producto de una matriz triangular inferior por una triangular superior, según el método de Crout.

$$K = L \cdot U$$

```
subroutine crout (K,L,Up)
 real*8 :: K(2,2),L(2,2),Up(2,2)

! ASIGNO LOS VALORES CONOCIDOS DE
ANTEMANO (POR EL PROPIO METODO)
 Up(1,1)=1d0
 Up(2,1)=0d0
 Up(2,2)=1d0
 L(1,2)=0d0
 L(1,1)=K(1,1)
 L(2,1)=K(2,1)

! CALCULO EL RESTO DE TERMINOS
 Up(1,2)=K(1,2)/L(1,1)
 L(2,2)=K(2,2)-L(2,1)*Up(1,2)

 return
end
```

Esta subrutina trabaja con las 3 matrices mencionadas en la subrutina anterior (K, L, Up), y se puede observar que su razón de ser no es otra que dotar de claridad y simplificar la subrutina anterior.

Subrutina “incr”

Esta matriz calcula el incremento de la solución correspondiente a la iteración actual, resolviendo para ello el sistema de ecuaciones

$$K \cdot \Delta u = -r$$

Siendo K la matriz de rigidez ya citada, Δu el incremento en la solución a aplicar en el paso actual y r el residuo (calculado en el paso anterior), definido como la diferencia entre las cargas aplicadas (qe) y las cargas obtenidas con la última solución.

La subrutina permite la resolución directa (cuando se tiene la inversa de K) o empleando una estrategia en dos pasos: uno directo y otro por remonte (cuando se tiene la matriz factorizada como el producto de una matriz triangular inferior, L por una triangular superior U). Ha de notarse, que dado a la forma en que se realiza el cálculo del vector residuos, se tiene directamente “-r”, por lo que dicho signo negativo no queda reflejado en la subrutina actual.

```
subroutine incr (K,L,Up,r,inc,flag1)
 integer flag1
 real*8 :: r(2),inc(2),L(2,2),Up(2,2),K(2,2)

!flag1=1 : TENGO LA INVERSA DE K
!flag1=0 : TENGO K FACTORIZADA (L·U)

 if (flag1.eq.0) then
! RESUELVO EL SISTEMA L*y=r (DIRECTO)
  inc(1)=r(1)/L(1,1)
  inc(2)=(r(2)-L(2,1)*inc(1))/L(2,2)
! RESUELVO EL SISTEMA Up*u=y (REMONTE)
  inc(2)=inc(2)/Up(2,2)
  inc(1)=(inc(1)-Up(1,2)*inc(2))/Up(1,1)
 else
! RESOLUCION DIRECTA
  inc(1)=(r(1)*K(1,1)+r(2)*K(1,2))
  inc(2)=(r(1)*K(2,1)+r(2)*K(2,2))
 endif
 return
end
```

Se puede apreciar que esta subrutina emplea las matrices anteriormente mencionadas, así como un vector para almacenar el resultado, que no es otro que el incremento a aplicar (inc), otro vector que contiene el error cometido con la solución obtenida con el incremento inmediatamente anterior (r) y una variable entera (flag1), que hace de variable lógica de control para diferenciar los casos en los que se tiene la inversa de la matriz de rigidez (flag1=1) o esta matriz factorizada (flag1=0).

Subrutina “error”

Esta subrutina calcula el vector de residuos para cada incremento en la carga, así como el error máximo (máximo residuo en valor absoluto). Dicho error es utilizado para el criterio de fin del programa. Por su parte, el vector de residuos se emplea para calcular un nuevo incremento de la solución.

```
subroutine error (W,qe,c,u,r,emax)
 real*8 W,emax
 real*8 :: c(9),u(2),r(2),qe(2)

 call coef (W,u,c)
!Recalculo coeficientes para la nueva solucion

 r(1)=u(1)*c(7)+c(6)*(c(8)-c(9))
!Evaluo las cargas internas que...
 r(2)=W*c(7)*(u(2)/W-0.5d0)+c(6)*(c(2)-c(4))
!...resultan de los "despl." obtenidos (Calculo f y g)
 r=r-qe
!Evaluo el error cometido (realmente obtengo -r)

 if (abs(r(1)).gt.abs(r(2))) then
  emax=r(1)
 else
  emax=r(2)
 endif

 return
end
```

Para dicho cálculo del vector de residuos y del error máximo cometido en el incremento, se emplean 2 variables reales, una para almacenar dicho error máximo y otra que almacena el peso propio del cable; así como 4 vectores, que almacenan los coeficientes de las ecuaciones (c), la solución (u), los residuos (r) y las cargas externas (qe).

Dado que cuando se hace la llamada a esta subrutina siempre se tienen nuevos valores de las variables, lo primero que se hace es actualizar los coeficientes correspondientes, para lo que se hace una llamada a la función coef. Seguidamente se calculan los residuos, obteniéndose directamente "-r", por lo que, tal como se comentó con anterioridad, a la hora de calcular un nuevo incremento de la solución no ha de contemplarse dicho signo negativo.

Finalmente, el mayor valor absoluto de los valores de este vector de residuos es almacenado en la variable que contiene el error cometido en la iteración actual.

Subrutina "deftens"

Esta subrutina, al igual que la subrutina "iniciales" tiene su razón de ser en la clarificación y estructuración del código, ya que sólo se hace la llamada a esta subrutina una vez en todo el código. En ella, a partir de la solución final al problema, se calcula la deformada y la tensión del cable.

```
subroutine deftens (c,u,x,n,Lo,W)
 integer n,i
 real*8 Lo,W,s,inc
 real*8 :: c(9),u(2),x(1000,4)

 i=1
 s=0d0
 inc=Lo/n
 do while (i.le.(n+1))
! RECALCULO COEFICIENTES QUE INCLUYEN LA S
  c(3)=(u(2)-s/c(5))/u(1)
  c(4)=dsqrt(1d0+c(3)*c(3))
  if (c(3).lt.0d0) then
   c(9)=-1d0
  else
   c(9)=1d0
  endif
  c(9)=c(9)*dlog(abs(c(3))+c(4))
!OBTENGO DEFORMADA Y TENSIÓN
  x(i,1)=s
  x(i,2)=u(1)*s*c(7)/Lo+c(6)*(c(8)-c(9))        !Calculo la coordenada x
  x(i,3)=c(7)*s*(u(2)/W-s/(2d0*Lo))/c(5)+c(6)*(c(2)-c(4)) !Coordenada.z
  x(i,4)=u(1)*c(4)                              !Calculo la Tensión

  i=i+1
  s=s+inc
  if (s.ge.Lo) then
   s=Lo
  endif
 enddo
 return
end
```

Se puede observar que se utilizan dos variables enteras: la variable “i” a modo de contador, y la variable “n” que establece el número de puntos utilizados para obtener la deformada y la distribución de tensiones en el cable. En este caso, se emplean 1000 puntos para la obtención tanto de la deformada, como de la distribución de tensiones.

Ademas se emplean cuatro variables reales que se corresponden con la longitud inicial del cable (L_0), su peso propio (W), el incremento o distancia entre dos puntos para el cálculo de la deformada y la distribución de tensiones (inc) y la coordenada en dirección del cable, tal como utilizada en el enunciado (s).

Finalmente, tres vectores conteniendo los coeficientes (c), la solución (u) y otro que va a almacenar la deformada y las tensiones en el cable (x) son igualmente utilizados.

Al incluirse la coordenada s, hay coeficientes que cambian porque dependen de dicha coordenada (anteriormente, al considerarse la totalidad del cable, eran sustituidos por L_0). Dichos coeficientes son recalculados en primer lugar. Seguidamente, se calculan

y almacenan en “x” las coordenadas cartesianas de la deformada del cable y la tensión. Asimismo, la coordenada “s” es almacenada en “x”. Finalmente, para tener la certeza de que se obtiene la solución en el extremo derecho del cable, lo cual no estaría garantizado debido a errores de redondeo en el cálculo del incremento (inc), cuando “s” supera “L_0”, dichas variables se igualan.

Subrutina “fullnewton”

Esta subrutina implementa el método de Newton-Raphson. Este método presenta exactamente el mismo esquema mostrado en la sección anterior, esto es, a partir de los valores iniciales de la solución, se calcula la matriz de rigidez tangente, se resuelve el sistema para calcular el incremento en la solución, se calcula la nueva solución y se recalcula el error de esta nueva solución, en caso de no ser suficientemente buena, el método continua iterando, para lo cual recalcula la matriz de rigidez tangente, un nuevo incremento, un nuevo error, y así sucesivamente hasta obtener una solución suficientemente buena.

En el caso particular del programa desarrollado, esta subrutina hace llamadas a las subrutinas “tangente”, “incr” y “error”, descritas con anterioridad, con lo que el código resta razonablemente simple y claro.

```
subroutine fullnewton (W,c,qe,u,r,emax)
 real*8 W,s,emax
 real*8 :: c(9),qe(2),u(2),r(2),inc(2),K(2,2),L(2,2),Up(2,2)

 call tangente (c,K,L,Up,0)   !Calculo K. flag1=0: factorizacion K
 call incr (K,L,Up,r,inc,0)   !Incremento en la solucion
 u=u-inc                      !Actualizo solucion
 call error (W,qe,c,u,r,emax) !Residuo + error

 return
end
```

Se puede observar que, puesto que lo que se hace es llamar a ciertas subrutinas ya descritas, todas las variables utilizadas en esta subrutina han sido ya descritas en las subrutinas anteriores, utilizándose incluso los mismos simbolos.

Subrutina “newtonmod”

Los métodos Newton modificados, surgen con el ánimo de aumentar la velocidad de convergencia del método de Newton-Raphson, que debido a la necesidad de calcular en cada iteración la matriz de rigidez tangente, puede hacerse muy lento a medida que aumenta la envergadura del problema.

En este caso se ha escogido el método de la rigidez inicial, el cual presenta el mismo esquema que el método de Newton-Raphson, con la salvedad de que la matriz de rigidez tangente sólo es calculada y factorizada en la primera iteración. Evidentemente, este método requerirá más iteraciones para obtener la solución, sin embargo, cada iteración es más rápida, ya que no ha de recalcularse y refactorizarse la matriz de rigidez tangente.

Al igual que en el caso anterior, se hacen llamadas a las funciones "tangente" (una vez solamente), "incr" y "error".

```
subroutine newtonmod (W,c,qe,u,r,L,Up,emax)
 real*8 W,s,emax
 real*8 :: c(9),qe(2),u(2),r(2),inc(2),K(2,2),L(2,2),Up(2,2)

 if (abs(r(2))).eq.abs(qe(2))) then    !Calculo K en el 1° incr.
  call tangente (c,K,L,Up,0)           !flag1=0: factorizacion K
 endif
 call incr (K,L,Up,r,inc,0)            !Incremento en la solucion
 u=u-inc                               !Actualizo solucion
 call error (W,qe,c,u,r,emax)          !Residuo + error

 return
end
```

Se puede observar que el cálculo de la matriz de rigidez tangente (obteniéndose la inversa) solo se realiza si se cumple la condición "if", impuesta en base a la igualdad del residuo y el vector de cargas externas o aplicadas (qe), lo cual solo se cumple en la primera iteración. Por lo demás, el algoritmo es idéntico a la subrutina anterior.

Subrutina "cuasinewton"

Esta subrutina realmente agrupa las dos rutinas que contienen los métodos cuasinewton: "bfgs" (método BFGS) y "broy" (método de Broyden).

Al igual que en el caso anterior, estos métodos sólo requieren el cálculo de la matriz de rigidez tangente en la primera iteración, pero a diferencia del método anterior, dicha matriz es actualizada en cada iteración. Al requerirse la inversa de la matriz de rigidez, ésta no es factorizada. La matriz de rigidez original es almacenada durante todo el proceso (Ho), al mismo tiempo que, para cada iteración una "copia" de la matriz es actualizada (H).

Con dicha matriz actualizada se calcula una nueva solución de manera similar al método de Newton; esto es, se calcula el incremento en la solución, se actualiza la

solución y se calcula el error cometido. Además, se calcula la diferencia en el error entre dos iteraciones consecutivas, necesaria para la actualización de la matriz H.

Con estos datos (variación del error entre dos iteraciones, incremento de la solución, matriz de rigidez original y matriz de rigidez actualizada) se llama a la subrutina "broy" o a la subrutina "bfgs", que calcularán la actualización de la matriz H.

```
subroutine cuasinewton (tol,W,c,qe,u,r,H,Ho,prod,emax)
 integer k
 real*8 emax,val,det,W,tol
 real*8 :: u(2),r(2),s(2),y(2),omega(2),v(2),c(9),qe(2),inc(2)
 real*8 :: prod(2,2),Ho(2,2),H(2,2),L(2,2),Up(2,2)

!Inicializo la matriz secante inversa
 k=0
 if (abs((r(2))).eq.abs(qe(2))) then  ! Calculo K en el 1° incr.
  call tangente (c,Ho,L,Up,1)         ! flag1=1: K inversa (K=Ho)
  H=Ho
  k=1                                 ! k=1 : primera iteracion
 endif

 call incr (H,L,Up,r,inc,1)            !Incremento en la solución
 u=u-inc                               !Actualizo solución
 y=r                                   !Recuerdo r anterior
 call error (W,qe,c,u,r,emax)          !Residuo + error
 y=r-y                                 !Diferencia fzas internas
                                       ! entre incr. consecutivos
 inc=-inc                              !Paso a los métodos s=-inc
 call broy (H,Ho,prod,inc,y,k)         !Método de Broyden
!call bfgs (H,inc,y)                   !Método BFGS.
 return
end
```

Se puede observar, que en lugar de la letra "K" para la matriz tangente, se utiliza "Ho", por similitudes con el algoritmo de estos métodos, que utiliza dicha nomenclatura. Además, el cálculo de K se hace, como ya se ha señalado, únicamente en el primer incremento. El resto de variables tienen el mismo significado ya mencionado con anterioridad. Además se emplea el vector "y" para almacenar el residuo del incremento anterior y poder calcular la diferencia entre residuos, necesaria para el método.

La subrutina calcula, en primer lugar, el incremento en la solución, seguidamente almacena el error de la iteración anterior en "y", para pasar a actualizar la matriz de rigidez según el método deseado. Dicha actualización es almacenada, para cada iteración, en la matriz "H"; y es realizada a partir de la matriz de rigidez inicial "H_0".

En último lugar, ha de señalarse que el cambio entre los dos métodos se realiza de manera "manual". Es por ello que la llamada a la subrutina "bfgs" se encuentra

"comentada", ya que en el caso presentado se utiliza el método de broyden (subrutina "broy"), pero simplemente con descomentar una subrutina y comentar la otra se obtiene el resultado con la subrutina deseada.

Subrutina "bfgs"

Esta subrutina procede a la actualización de la matriz de rigidez tangente (H), de acuerdo con el método BFGS. Dicha actualización se realiza en base a la matriz de rigidez inmediatamente anterior.

```
subroutine bfgs(H,s,y)
 integer ci,cj
 real*8 den
 real*8 :: H(2,2),Hn(2,2),p1(2,2),s(2),y(2)

den=y(1)*s(1)+y(2)*s(2) !Calculo el denominador
 do ci=1,2       !Calculo los corchetes (son el mismo traspuesto)
  do cj=1,2
   p1(ci,cj)=-s(ci)*y(cj)/den
   if (ci.eq.cj) then
    p1(ci,cj)=p1(ci,cj)+1d0
   endif
  enddo
 enddo

 do ci=1,2       !Calculo producto primer corchete por H anterior
  do cj=1,2
   Hn(ci,cj)=p1(ci,1)*H(1,cj)+p1(ci,2)*Ho(2,cj)
  enddo
 enddo

 do ci=1,2 !Mult.por 2°corchete.Sumo el término restante-->nueva H
  do cj=1,2
   H(ci,cj)=Hn(ci,1)*p1(cj,1)+Hn(ci,2)*p1(cj,2)+s(ci)*s(cj)/den
  enddo
 enddo

return
end
```

A parte de la matriz "H" y los vectores "s" e "y", que contienen la actualización de la matriz de rigidez, el incremento en la solución y la variación del residuo entre dos iteraciones consecutivas, respectivamente, se tienen las matrices adicionales "Hn" y "p1", cuya función es almacenar la matriz de rigidez tal como es "pasada" a la subrutina y servir de apoyo al cálculo, respectivamente. Además, se emplea la variable real "den", que sirve de apoyo al cálculo.

Subrutina “broy”

Esta subrutina actualiza la matriz de rigidez tangente (H), segun el método de Broyden. Realizándose la actualización con respecto a la matriz de rigidez original.

```
subroutine broy (H,Ho,prod,s,y,k)
 integer k
 real*8 val
 real*8 :: v(2),H(2,2),s(2),y(2),corch(2,2)
 real*8 :: omega(2),alm(2,2),prod(2,2),Ho(2,2)
! ACTUAL.MATRIZ(H):PRODUCTO DE CORCHETE POR MATRIZ ORIGINAL (Ho)
! CALCULO DEL CORCHETE PREMULTIPLICADOR
do ci=1,2
  v(ci)=H(ci,1)*y(1)+H(ci,2)*y(2)
enddo

 val=s(1)*v(1)+s(2)*v(2)
 omega=(s-v)/val                    !   Obtengo el valor de omega
! Lo multiplico por -incr.ant.--> corchete de la iteracion actual
 do ci=1,2
  do cj=1,2
   corch(ci,ctj)=omega(ci)*s(cj)
   if (ci.eq.cj) then
    corch(ci,cj)=corch(ci,cj)+1d0    !Sumo la matriz identidad
   endif
  enddo
 enddo
! Multiplico corchete act.por los anteriores-->corchete premult.
 if (k.eq.1) then      !1° iteración: no hay producto que realizar
  prod=corch
 else
  alm=prod
  do ci=1,2
   do cj=1,2
    prod(ci,cj)=alm(ci,1)*corch(1,cj)+alm(ci,2)*corch(2,cj)
   enddo
  enddo
 endif
! CALCULO LA NUEVA MATRIZ (PRODUCTO DEL CORCHETE POR Ho)
 do ci=1,2
  do cj=1,2
   H(ci,cj)=prod(ci,1)*Ho(1,cj)+prod(ci,2)*Ho(2,cj)
  enddo
 enddo

return
end
```

La subrutina “recibe” la matriz de rigidez original (H_0), el incremento en la solución (s), la diferencia en el vector residuo entre dos iteraciones consecutivas (y).

Ademas la matriz “prod” y el entero “k” (contador de iteraciones) han de “pasarse” de iteración en iteración. El resto de vectores, matrices y variables son auxiliares al calculo.

Subrutina “newtonsecantes”

Esta subrutina calcula la solución de acuerdo al sistema de solución de los métodos secantes.

```
subroutine newtonsecantes_v2 (L,Up,c,u,r,qe,s,sg,sga,y,W,tol,emax)
 integer j
 real*8 tau,rho,val1,val2,tol,W,c1,c2,c3,emax
 real*8 :: K(2,2),r(2),u(2),c(9),L(2,2),Up(2,2)
 real*8 :: s(2),sg(2),sga(2),qe(2),y(2)

 j=1
 if (abs(r(2)).eq.abs(qe(2))) then      ! Calculo K en el 1° incr.
  call tangente (c,K,L,Up,0)            !flag1=0 : obtengo k=L·U
  j=0                       !Variable de control (j=0: 1°iteracion)
 endif

 call incr (K,L,Up,r,sg,0)  !Incremento (flag1=0: K=L·U)
 sg=-sg                 !Cambio el signo del valor del incremento
 if (j.eq.0) then       !1° iteración: tomo ese valor de incremento
  s=sg
 else                   !1° iteracion, calculo el nuevo incremento
  call broyden_sec (s,sg,sga)          !Metodo de broyden secante
!  call bfgs_secante (s,y,sg,sga,r)    !Metodo bfgs secante
 endif

 u=u+s                                 !Actualizo la solucion
 y=r                                   !Lo que calculo en error es -r
 call error (W,qe,c,u,r,emax)          !Evaluo la bondad de la solucion
 y=y-r                                 !Lo que calculo en error es -r
 sga=sg                                !Recuerdo el incremento anterior

 return
end
```

Inicialmente (y sólo para la primera iteración) se calcula la matriz de rigidez tangente, la cual se obtiene en su forma factorizada. Seguidamente, se calcula el incremento en la solución. Dicho incremento se emplea para actualizar la solución: en la primera la iteración se actualiza directamente con dicho incremento, mientras que en el resto de iteraciones, el incremento es recalculado según alguno de los dos métodos (broyden o BFGS).

Una vez actualizada la solución, se calcula el error cometido y, al igual que en los métodos cuasi-Newton, se calcula la diferencia de error entre dos iteraciones consecutivas.

Ha de notarse la diferencia de planteamiento con respecto a los métodos Newton modificados: mientras que aquéllos utilizan las subrutinas “bfgs” y “broy” para actualizar la matriz de rigidez, en este caso, las subrutinas correspondientes sirven al cálculo de los nuevos incrementos en la solución de acuerdo con un esquema secante.

Subrutina “bfgs_secante”

Esta subrutina calcula, a partir del valor del incremento obtenido mediante la subrutina “incr” (sg), del incremento anterior (sga), de la diferencia de error entre dos iteraciones consecutivas (y) y del error en la iteración anterior (r), el valor del nuevo incremento en la solución (s).

```
subroutine bfgs_secante (s,y,sg,sga,r)
 real*8 den,c1,c2,c3
 real*8 :: s(2),sg(2),sga(2),r(2),y(2)

 den=s(1)*y(1)+s(2)*y(2)
 c1=(y(1)*sga(1)+y(2)*sga(2))/den                !Coef BFGS
 c2=(y(1)*sg(1)+y(2)*sg(2))/den                  !Coef BFGS
 c3=(s(1)*r(1)+s(2)*r(2))/den                    !Coef BFGS
 s=(1d0+c3)*sg-c3*sga+(c3-(1d0+c3)*c2+c3*c1)*s   !BFGS secante
 return
end
```

Ademas, se utilizan las variables adicionales den, c1, c2, c3. Todas de carácter real (de doble precisión), para facilitar el cálculo.

Subrutina “broy_secante”

Esta subrutina calcula, a partir del valor del incremento obtenido mediante la subrutina “incr” (sg) y del incremento en la iteración anterior (sga), el valor del nuevo incremento en la solución (s).

```
subroutine broyden_sec (s,sg,sga)
 real*8 tau,rho
 real*8 :: s(2),sga(2),sg(2)

 tau=s(1)*(sga(1)-sg(1))+s(2)*(sga(2)-sg(2))
 rho=(s(1)*sg(1)+s(2)*sg(2))/tau
 s=(1d0+rho)*sg+rho*(s-sga)              !Broyden secante
 return
end
```

Se utilizan las variables adicionales tau y rho, con el mismo significado que el algoritmo genérico del método.

RESULTADOS

Evolución del error

Si se observa la subrutina "main", se puede ver que se ha establecido una tolerancia para la solución (fuerzas de reacción) de 1E-10. Asimismo, se puede observar que esta condición se traduce en que se itera, con el método escogido, hasta lograr que el valor absoluto del error máximo sea inferior a dicho valor de tolerancia.

Observando las gráficas de convergencia para los diferentes métodos, se puede apreciar que todos los métodos presentados convergen rápidamente hacia valores bajos del error cometido.

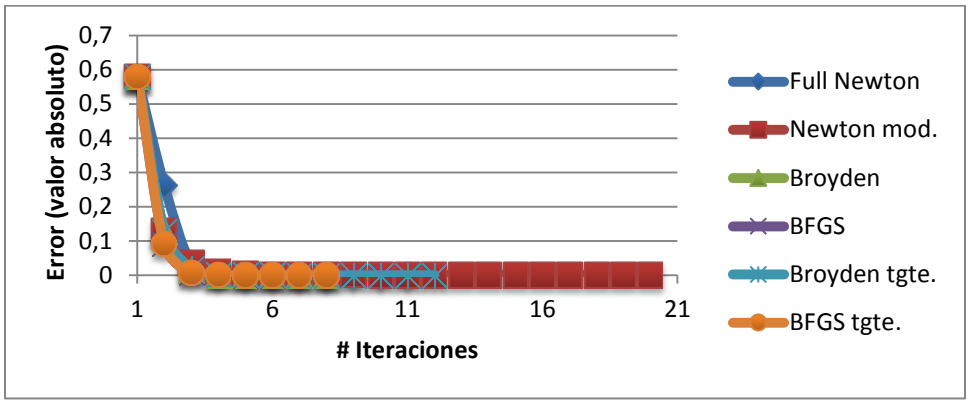


Fig. 2 – Evolución del error para los diferentes métodos.

Analizando las 4 primeras iteraciones se puede apreciar que el único método que presenta una curva ligeramente diferente del resto es el método de Newton-Raphson, alcanzando sin embargo, valores similares del error a la 3° iteración.

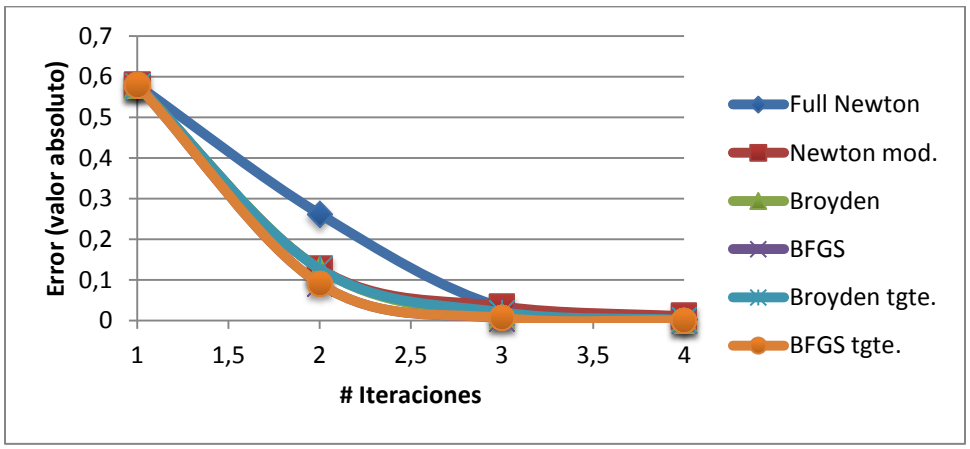


Fig. 3 – Evolución del error para los diferentes métodos.

En cuanto al error final cometido y el número de iteraciones necesarias para obtener un error inferior a la tolerancia especificada, se tiene que el método que más rápido converge es el método cuasi newton BFGS, y que todos los métodos convergen en el entorno de las 10 iteraciones, salvo el método de Newton modificado, el cual precisa de 20 iteraciones para alcanzar el error deseado.

Método	# Iteraciones	Error final
Newton-Raphson	11	5.755E-11
Newton modificado (tension inicial)	20	6.5317E-11
Cuasi newton: Broyden	8	4.5937E-12
Cuasi newton: BFGS	7	1.0463E-11
Newton secante: Broyden	12	2.6077E-12
Newton secante: BFGS	8	1.1729E-11

Tabla 2 – Numero de iteraciones y error final para los diferentes métodos.

El menor error se encuentra en el método Broyden secante, sin embargo, siendo de un orden de magnitud inferior al error cometido con el resto de métodos, excluyendo al método cuasi Newton de Broyden. En todo caso, el error cometido es muy pequeño, precisándose un bajo número de iteraciones.

Deformada

En cuanto a la deformada obtenida por los diferentes métodos, se puede observar que ésta coincide prácticamente a la perfección, lo cual no es de extrañar, teniendo en cuenta la tolerancia impuesta para la solución, que hace que la máxima diferencia en la misma entre los métodos sea del orden de 6E-11. Cabe remarcar que en la figura se muesta el valor absoluto de la coordenada z, esto es, la deflexión, y no su valor real (que, si se toma como origen el extremo superior del cable, sería negativo).

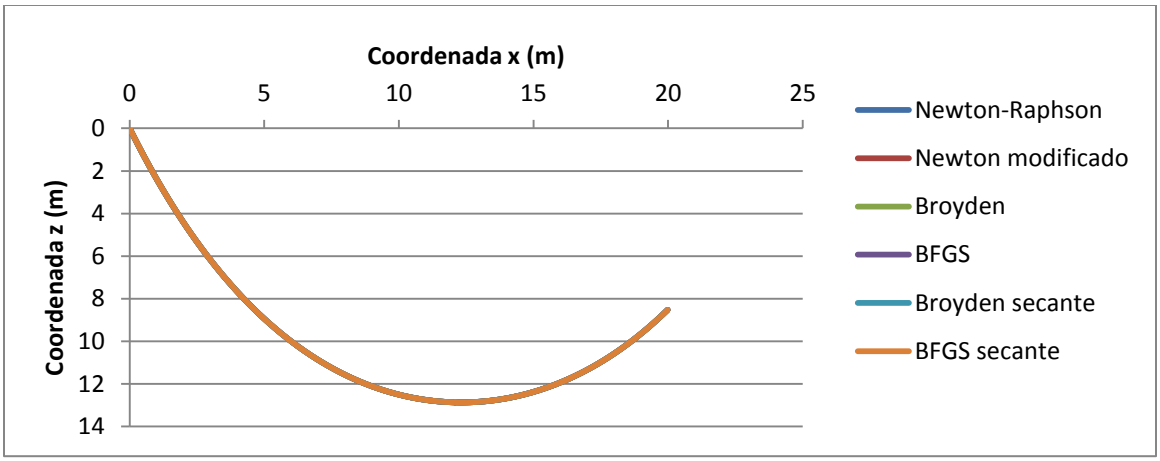


Fig. 4 – Deformada del cable para los diferentes metodos.

Así, la máxima diferencia en la coordenada z entre los diferentes métodos es de 6.30394E-11 m., la cual se produce entre los métodos de Newton-Raphson y el método BFGS, en el extremo derecho del cable.

En cuanto a la tensión axial en el cable se refiere, ésta presenta una curva similar a la obtenida para la deformada. A título de ejemplo se presenta la obtenida mediante el método de Newton-Raphson.

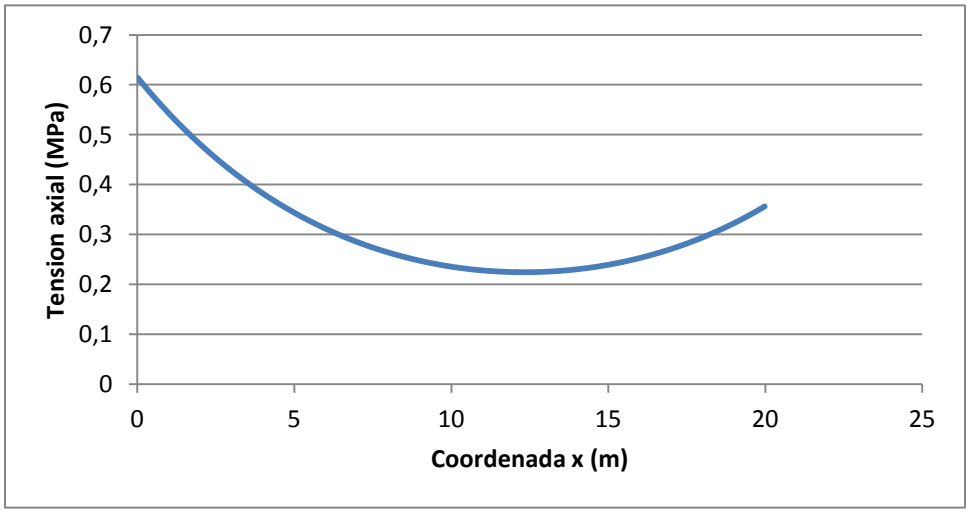


Fig. 5 – Tension axial del cable por el método de Newton-Raphson.

Se puede observar que la tensión es minima para el punto donde ocurre la deflexión máxima del cable, siendo su valor de, aproximadamente 0.22 MPa. La tensión máxima por otro lado, se produce en el apoyo situado a mayor altura, tomando un valor ligeramente superior a los 0.6 MPa.

CONCLUSIONES

Todos los métodos presentados permiten resolver satisfactoriamente el problema planteado, empleando para ello un reducido número de iteraciones, lo cual es deseable desde el punto de vista del análisis numérico.

Entre todos los métodos empleados, el menor coste por iteración lo presenta el método de Newton modificado de la tensión inicial, ya que en este caso no se recalcula ni actualiza la matriz de rigidez tangente. Seguidamente se encuentran los métodos cuasi Newton y los secantes, ya que estos plantean actualizaciones de la matriz tangente que evitan el tener que recalcular la misma. Finalmente, el método de Newton-Raphson, al exigir el cálculo de la matriz de rigidez tangente en cada iteración es el que mayor coste computacional presenta.

Así, teniendo esto en cuenta, se puede concluir que los métodos más idóneos para la resolución del problema planteado son los métodos cuasi Newton y el método BFGS secante, ya que además del hecho de que su coste por iteración es bajo, requieren de un menor número de iteraciones para hacer converger la solución (al valor del error especificado).