Universidad Autónoma de Querétaro
Facultad de Ingeniería
Doctorado en Ingeniería

Dinámica no lineal de elementos mecánicos en contacto mediante el método de Isogeometría

Tesis

Que como parte de los requisitos para obtener el Grado de
Doctor en Ingeniería

Presenta:
M.C. Stephanie Virginia Camacho Gutiérrez

Dirigido por:
Dr. Juan Carlos Jáuregui Correa

Dr. Juan Carlos Jáuregui Correa
Presidente

Dr. Aurelio Domínguez González
Secretario

Dr. Carlos López Cajún
Vocal

Dr. Roberto Gómez Loenzo
Suplente

Dr. Manuel Toledano Ayala
Suplente

Centro Universitario, Querétaro, Qro.
Agosto, 2020
México

Resumen

En este trabajo de tesis se propuso un método para resolver el problema de contacto no lineal entre dos cuerpos, cuyo comportamiento se asumió como deformación plana. Esta propuesta se basó en la ecuación integral de la frontera (*BIE,* por sus siglas en inglés) y en el método de análisis isogeométrico. A diferencia de las investigaciones que dividen la frontera en elementos, en este método se utilizó un "macroelemento", reduciendo los grados de libertad requeridos para resolver el problema. Asimismo, se utilizó el algoritmo de cúmulo de partículas para encontrar la deformación entre los cuerpos. Los resultados obtenidos se compararon con métodos probados, como el método de elementos finitos y el método analítico basado en el modelo de Hertz.

Summary

In this thesis work, a method is proposed to solve the non-linear problem of contact between two bodies, whose behavior is assumed as plane strain. This approach is based on the integral boundary equation (BIE) and the isogeometric analysis method. Unlike research that divide the boundary into elements, this method uses an "macro-element"; reducing the degrees of freedom required to solve the problem. Also, the particle swarm optimization was used to find the deformation between the bodies. The results were compared with proven methods, such as the finite element method and the analytical method based on the Hertz model.

Dedicatorias

Al amor de mi vida, mejor amigo y compañero de aventuras: Iván, gracias por animarme y apoyarme siempre.

A mis mamás, hermanos y familia, gracias por estar pendientes de mí y también por apoyarme en cada una de mis decisiones.

Agradecimientos

A mi director de tesis, Dr. Juan Carlos Jáuregui, muchas gracias por apoyarme y por la disponibilidad para resolver mis dudas y explicarme muchas veces lo que no entendía.

A mis sinodales, Dr. Aurelio, Dr. Roberto, Dr. Toledano y Dr. López-Cajún, por el tiempo que se tomaron en revisar este trabajo, y por las observaciones que me brindaron a lo largo de este proyecto de investigación.

A mis compañeros de doctorado, por sus ánimos y consejos.

A mis profesores, sus observaciones y enseñanzas me ayudaron mucho.

A la Universidad Autónoma de Querétaro y al Consejo Nacional de Ciencia y Tecnología (CONACYT), por apoyarme con becas. Espero retribuir ese esfuerzo en cada trabajo que realice.

Índice General

Índice de Figuras

Glosario

A: Matriz formada por los coeficientes de los kernels de tracción y de la matriz de salto C

$a_{contacto}$: Ancho del área de contacto

b: Vector de con desplazamientos y tracciones prescritos

B: Vector de puntos de control para generar una curva con *NURBS*

BEM: Siglas en inglés del método numérico *"Boundary element method*", en español este método se conoce como "Elementos Frontera"

B-Spline: Función de interpolación definida a trozos por un polinomio, de manera recursiva, mediante un vector de nudos y puntos de control, para generar curvas.

C: Matriz de salto

C_1: Constante usada en el algoritmo de *PSO*

$\mathrm{C}_{\max}$: Constante usada en el algoritmo de *PSO*

CAD: Siglas en inglés de software "*Computer Aided Design*" ("Diseño asistido por computadora).

CAE: Siglas en inglés de software "*Computer Aided Engineering*" ("Ingeniería asistida por computadora")

$\mathrm{d}_{\max}$: Distancia máxima entre los puntos de carga y campo

E: Coeficiente de elasticidad o modulo de Young

e: Entidad "elemento" del *BEM*

f: Fuerza

factor: Factor de escala de la matriz $\mathrm{A/U}$

g: Grado de la curva *NURBS*

GDL: Siglas de "Grados de Libertad"

Gen: Número de ciclos que se ejecutará el *PSO*

G_{N}: Mejor posición global de las partículas

I: Perímetro del cuerpo deformado

I_{or}: Perímetro del cuerpo bajo análisis antes de aplicar una carga

IGA: Siglas en inglés del método "*Isogeometric Analysis*"

IGA-BEM: Acrónimo en inglés de "*Isogeometric Boundary Element Method*" ("Método de Elementos Frontera Isogeométrico")

J: Jacobiano

k: Orden de la curva *NURBS* $(\mathrm{grado} - 1)$

L_{i}: Mejor posición local de la partícula

L_{on}: Longitud

lb: Límite inferior de las variables de diseño

M: Función de forma cuadrática

m: Multiplicidad del nudo κ

MEF: Siglas del "Método de Elementos Finitos". En inglés se abrevia como *FEM* ("*Finite Element Method*")

n: Vector normal

N: Funciones base *B-Spline*

n_{cp}: Al sumarle uno, se convierte en el número de puntos de control de la *NURBS*

n_{e}: Número de elementos usados en el *BEM*

$\mathrm{N}_{\mathrm{points}}$: Número de puntos de carga/campo usados en el *BEM* propuesto

$\mathrm{N}_{\mathrm{pointsDraw}}$: Número de puntos usados para dibujar la curva *NURBS*

$\mathrm{N}_{\mathrm{swarm}}$: Tamaño de la población usado en el algoritmo *PSO*

NURBS: Siglas en inglés de una función "*Non-Uniform Rational B-Spline*" ("*B-Spline* No-uniforme Racional") que se usa para generar curvas

P: Punto de carga en el contorno del cuerpo

p: Punto de carga en el interior del cuerpo

PSO: Siglas en inglés del método "*Particle Swarm Optimization*" ("Optimización por cúmulo de partículas")

Q: Punto de campo

R: Función base racional de una curva *NURBS*

r: Distancia entre el punto de carga y el punto de campo

rand: Número aleatorio

S: Superficie

s: Parámetro que controla la curvatura de una *NURBS*

T: Kernel de tracciones de la solución de Kelvin

t: Tracción

t_{int}: Tracciones en el interior del cuerpo bajo análisis

tam: Tamaño del vector de nudos

T-Spline: Superficie generada con *NURBS* y modelado de subdivisión

U: Kernel de desplazamientos de la solución de Kelvin

u: Desplazamiento

u_{int}: Desplazamientos en el interior del cuerpo bajo análisis

ub: Límite superior de las variables de diseño

V: Volumen

V_i: Velocidad de la partícula en el algoritmo de *PSO*

W: Vector de pesos para generar una curva con *NURBS*

w_i: Elementos del vector de pesos

X, x: Coordenada en el eje X

X_i: Posición de una partícula en el algoritmo de *PSO*

y: Coordenada en el eje Y

α: Punto de control de una curva *NURBS*

β: Punto medio entre dos puntos de control

γ: Punto en una curva *NURBS* alineado con un punto de control

Γ: Contorno del cuerpo bajo análisis

ε: Deformación

ζ: Variable del sistema local coordenado del *BEM*

η: Número de funciones base diferentes de cero

κ_i: Elemento i del vector de nudos

Λ: Curva generada con *NURBS*

μ: Modulo de cizalladura

Ξ: Vector de nudos usado para generar una curva *NURBS*

ξ: Parámetro usado para generar una curva *NURBS*

ϖ: Sistema de tracciones y esfuerzos del teorema de Betti

ρ: Sistema de tracciones y esfuerzos del teorema de Betti

ϱ: Variable de penalización usada en la implementación del *PSO*

σ: Esfuerzo

τ: Carga aplicada en los cuerpos bajo análisis

τ_L: Carga por unidad de longitud

υ: Coeficiente de Poisson

Ψ: Vector de desplazamientos y tracciones desconocidos

Ω: Matriz con los kernels acomodados para resolver el sistema de ecuaciones de *BEM*

CAPÍTULO I

1. Introducción

La mecánica del medio continuo es la encargada de estudiar esfuerzos, deformaciones y flujos que ocurren en sólidos, líquidos y gases. Como en la mayoría de las ramas de ingeniería, en mecánica del medio continuo también se utilizan métodos numéricos para resolver problemas prácticos. Entre estos métodos, tenemos el de diferencias finitas (*FD*, por sus siglas en inglés), el de elementos finitos (MEF), y el de elementos frontera (*BEM*, por sus siglas en inglés). Estas técnicas consisten en dividir al cuerpo de estudio (medio continuo) en elementos discretos, los cuales se analizan individualmente de acuerdo con las ecuaciones constitutivas del problema. Posteriormente, se usan relaciones adicionales para ensamblar los elementos y de esta forma obtener el valor, por ejemplo, de desplazamientos o deformaciones en el cuerpo.

Actualmente el método de elementos finitos constituye la base de la mayoría de los softwares comerciales de ingeniería asistida por computadora (*CAE*), como ANSYS, COMSOL, SOLIDWORKS, etc. Sin embargo, de acuerdo a un estudio reportado por Cottrell *et al.* [1], de todo el tiempo requerido para obtener la simulación de una pieza en software CAE, el 20% se consume en la elaboración del modelo en software *CAD* (Diseño Asistido por Computadora); el 60% del tiempo se utiliza para generar la geometría adecuada en el análisis se utiliza aproximadamente, y sólo el 20% se destina para el análisis en sí. Por tanto, se requieren de nuevos métodos para optimizar el tiempo empleado en el análisis por *CAE*.

En 2004 Hughes *et al.* [2] propuso utilizar las curvas *NURBS* (*Non-Uniform Rational B-Spline*) generadas en el modelo *CAD* como modelo de análisis en lugar de usar el típico método de interpolación, disminuyendo así el tiempo empleado para el análisis y el error derivado de la aproximación geométrica. Este concepto se llamó Análisis Isogeométrico (*IGA*, por sus siglas en inglés).

En esta tesis se presenta una metodología basada en *IGA* para analizar el contacto entre dos elementos mecánicos. En este primer capítulo se muestran los antecedentes, el planteamiento del problema y los objetivos de la tesis. En el siguiente capítulo, se abordan temas claves como las *NURBS*, el método de elementos frontera y los algoritmos de optimización. Posteriormente, en el capítulo III se explica la metodología propuesta. En el capítulo IV se muestran los resultados obtenidos con esta metodología y se comparan con los métodos de elementos finitos y elementos frontera. Finalmente, en el capítulo V se concluye y se muestran las perspectivas del trabajo.

1.1 Antecedentes

Inicialmente, *IGA* se utilizó en conjunto con elementos finitos, [2]–[14], pero eventualmente se combinó con los métodos de elementos frontera ([15]–[23]) y celdas finitas ([24]–[26]). El método de elementos frontera resuelve los problemas de mecánica de medios continuos usando el teorema de

Betti, las ecuaciones de Navier-Cauchy y el teorema de la divergencia. Este último teorema simplifica la solución de los problemas del medio continuo, modelándolos únicamente en el contorno [27]. Este modelado consiste en dividir el contorno del cuerpo bajo análisis en un conjunto discreto de funciones, denominados elementos con nodos, y en describir los desplazamientos y tracciones con funciones polinómicas. Dado que el *BEM* es una consecuencia del teorema Betti, existen dos conjuntos de desplazamientos y tracciones. Un conjunto de ellos se conoce de antemano a través de entidades llamadas *kernels* de tracción y desplazamiento, que son válidos para cualquier geometría en equilibrio. El otro conjunto pertenece al cuerpo bajo análisis, que es el problema a resolver. Naturalmente, para obtener una solución única, se deben aplicar condiciones de frontera tales como tracciones y desplazamientos prescritos.

En 2012 Simpson *et al.* [28] establecieron las pautas para combinar el método de elementos frontera con análisis isogeométrico. Por ejemplo, los elementos se generaron a partir del vector de nudos de las *NURBS* obtenidas del *CAD,* y las funciones base de estas curvas reemplazaron las funciones de forma polinomiales que se usan típicamente en el método de elementos frontera. Asimismo, aplicaron su metodología para obtener las deformaciones de los siguientes cuerpos: una placa en L, una placa infinita con un agujero dentro, un reactor nuclear y una llave ajustable con mango.

Posteriormente, Scott *et al.* [29] refinaron el método *IGA-BEM* al usar *T-Splines* en lugar de NURBS para resolver problemas elastoestáticos lineales, ellos demostraron la eficacia de su método analizando una hélice y haciendo una prueba de parche. Luego, Lian *et al.* [16] continuaron esta línea de investigación usando *T-Splines* para optimizar el análisis tridimensional de cuerpos elásticos, y eligiendo los puntos de control como variables para modificar la geometría. Asimismo, Takahashi y Matsumoto [30] utilizaron el *IGA-BEM* combinado con el método rápido multipolar (*FMM*, por su siglas en inglés) para disminuir los puntos de control en las *NURBS*. Ellos mostraron que dicha combinación puede lograr la misma exactitud que el método tradicional *BEM,* pero con menos grados de libertad.

Por otra parte, la combinación *IGA-BEM* no sólo se ha centrado en disminuir los grados de libertad, cambiar las funciones base o resolver problemas lineales. También se ha implementado en otros rubros interesantes. Por ejemplo, Heltai *et al.* [31] usaron *IGA-BEM* para analizar los flujos de Stock 3D y Gong y Dong [32] desarrollaron un método para calcular las integrales singulares de un problema de potencial 3D. Peng *et al.* [33] utilizaron un algoritmo geométrico para propagar fracturas e *IGA-BEM* para analizarlas. Venas y Kvamsal [34] utilizaron *IGA-BEM* para resolver el problema de dispersión acústica de un submarino.

Las aplicaciones de *IGA-BEM* en una amplia variedad de problemas han dado buenos resultados. Sin embargo, la solución depende de los *kernels* calculados en los puntos de interpolación que describen el contorno, restringiendo el dominio de la solución al dominio de la discretización geométrica. En este trabajo se propone una metodología basada en *IGA-BEM* usando

una sola función *NURBS* continua, por lo que la discretización de la geometría se hace independiente de los puntos de evaluación, y se separa el cálculo de los *kernels* de las funciones de interpolación. Estas propiedades permiten modificar el contorno para resolver problemas como el de contacto entre dos cuerpos.

El contacto mecánico se ha estudiado ampliamente. Johnson [35] describe los modelos de contacto analíticos más conocidos, tal como, el modelo de Hertz, el modelo KJR, el modelo Bradley, entre otros. El modelo de Hertz relaciona el área de contacto con las propiedades del material, al igual que el modelo JKR, pero este último agrega las fuerzas de interacción interfacial. Finalmente, el modelo Bradley considera las interacciones externas de Van der Waals que agregan carga al contacto.

Entre los métodos numéricos usados en mecánica de medios continuos para analizar el contacto, el algoritmo más popular es el de maestro-esclavo. Este método consiste en proyectar los nodos del cuerpo elástico (esclavo) en el cuerpo rígido (maestro) cuando se aplica una fuerza. Si los nodos del cuerpo esclavo penetran el cuerpo del maestro, se utiliza una fuerza en el área de contacto para regresar los nodos del esclavo al exterior del maestro (Figura 1-1). De esta manera, se encuentra la deformación entre dos cuerpos. Este algoritmo se ha utilizado en conjunto con MEF ([36]–[38]), y recientemente se ha combinado con *IGA* ([3]–[5], [39]–[42]). Sin embargo, la principal desventaja de este método es que la fuerza de contacto aplicada no es una fuerza real. Adicionalmente, cada vez que se emplea esta fuerza, se debe encontrar nuevamente la zona de contacto y también se debe verificar que no haya penetración. En caso de que haya, el proceso se repite.

Cuando se utiliza BEM, el problema se resuelve de manera diferente. Los nodos que se encuentran en la zona de contacto deben cumplir con dos condiciones generales: debe haber continuidad entre sus desplazamientos (no superpuestos) y sus tracciones deben ser las mismas, pero en la dirección opuesta. Como el área de contacto no se conoce de antemano, el análisis comienza con una primera zona de contacto, que se modifica hasta que los nodos satisfagan las condiciones de contacto. Aunque el contacto se modela más fácilmente con *BEM* que con MEF, el análisis presenta las desventajas de *BEM*, como la sensibilidad a la selección de los puntos de análisis que conduce a errores en el cálculo numérico de las integrales.

Figura 1-1 Contacto maestro-esclavo.

También se tiene al método "*meshless*", cuyas funciones base se definen por un conjunto de nodos dispersos que no generan un enmallado como en el MEF convencional [43]–[49]. Sin

embargo, los resultados que se obtienen con este método son sensibles a la elección de los nodos dispersos tal como sucede con MEF.

1.2 Planteamiento del Problema

Para analizar un elemento mecánico por medio de un software de ingeniería asistido por computadora, los diseños son trasladados a una geometría apropiada (enmallado). Este proceso consume alrededor de un 80% del análisis completo y los diseños cada vez son más complejos [1].

Asimismo, el resultado obtenido al resolver un problema de contacto entre elementos mecánicos con software CAE, basado en elementos finitos, contiene errores en el cálculo al aproximar las áreas de contacto con el método de maestro-esclavo. Para disminuir este error se debe hacer un enmallado más fino, aumentando los grados de libertad (GDL). Sin embargo, entre más grados de libertad se tengan, más grande es el error. Por tanto, debe haber un equilibrio entre los grados de libertad y el tamaño de la malla.

El problema de contacto también se resuelve con modelos analíticos, sin embargo, con estos modelos no se obtiene el comportamiento local de la zona de contacto.

1.3 Justificación

Dado que los diseños mecánicos se han vuelto cada vez más complejos, se requiere de análisis más eficientes (en tiempo) y exactos (en soluciones). Al utilizar el análisis isogeométrico como base para desarrollar una metodología que resuelva el problema de contacto entre elementos mecánicos, se podrían obtener modelos con menor cantidad de grados de libertad, sin perder la capacidad de solución, incluyendo los fenómenos locales.

1.4 Hipótesis

Mediante el uso del análisis isogeométrico, definido a través de datos de las curvas NURBS, se obtiene un análisis más exacto del fenómeno de contacto dinámico entre elementos mecánicos con menor número de grados de libertad que con los métodos existentes.

1.5 Objetivos

1.5.1 Objetivo general

Desarrollar una metodología basada en el análisis isogeométrico que represente el fenómeno de contacto dinámico entre elementos mecánicos con menor número de grados de libertad que con los métodos existentes.

1.5.2 Objetivos particulares

1. Determinar la función de contacto dinámico entre dos elementos mecánicos simples para integrarla al análisis isogeométrico.
2. Programar el método propuesto para analizar la dinámica de dos elementos mecánicos simples en contacto.
3. Establecer la bondad de ajuste del método propuesto al comparar sus resultados con métodos existentes.

CAPÍTULO II

2. Fundamentación teórica

La metodología de contacto entre elementos mecánicos que se propone en este trabajo está compuesta por tres partes principales: la modelación geométrica mediante curvas NURBS, la teoría de elementos frontera y la aplicación de un algoritmo de optimización para encontrar el equilibrio. Por tal motivo, este capítulo contiene un resumen de estas partes.

2.1 Modelación geométrica mediante curvas NURBS

Un modelo geométrico generado con software CAD se hace mediante objetos gráficos llamados "primitivas" que el usuario manipula a fin de crear un dibujo o un elemento tridimensional. Las primitivas más importantes son los puntos, las líneas, los segmentos rectos, las cónicas, los objetos compuestos, las curvas paramétricas, las superficies analíticas, las superficies paramétricas y los sólidos [50]. De estos objetos, los más transcendentales para fines de este trabajo son las curvas paramétricas, dentro de las cuales se encuentran las curvas *NURBS*.

Las *NURBS* son un tipo de curva paramétrica capaz de representar con precisión líneas, curvas, sólidos y superficies de manera recursiva y con menor cantidad de información que otras aproximaciones (por ejemplo, los polinomios de Lagrange). Una *NURBS* también se considera como una función polinomial por tramos de grado g $(k-1)$. De [51], una *NURBS* se define como:

$$(\Lambda) = \frac{\sum_{i=1}^{n_{cp}+1} B_i w_i N_{i,k}(\xi)}{\sum_{i=1}^{n_{cp}+1} w_i N_{i,k}(\xi)} \qquad a < \xi < b, \quad 2 \leq k \leq n_{cp}+1 \tag{2.1}$$

Donde B_i simboliza a los puntos de control, w_i a los pesos y $N_{i,p}(\xi)$ a las funciones base *B-Spline* de orden k definidas por el parámetro ξ.

2.1.1 Elementos de una curva NURBS

- **Orden de la curva $\boldsymbol{k}$**: Es el grado de la curva más uno. El orden máximo de una *NURBS* es igual al número de vértices del polígono de control. Asimismo, cualquier tramo de la curva será C^{k-2} continua. Por ejemplo, si el orden de la curva es 3 y no se repite ningún nodo, entonces toda la curva será C^1 continua.
- **Puntos de control $\boldsymbol{B_i}$**: Estos $n_{cp}+1$ puntos de control conforman lo que se denomina "polígono de control" (o *convex hull*), que da forma a la curva. Los puntos generados con la ecuación (2.1) yacen dentro de este polígono de control. En la Figura 2-1 se muestra el efecto de mover un punto de control en la forma de la curva. Si los puntos de control se repiten, la curva se atrae hacia ese punto, y aunque se genere una esquina aguda (Figura 2-2), la diferenciabilidad se mantiene y la curva es C^{k-2} continua.

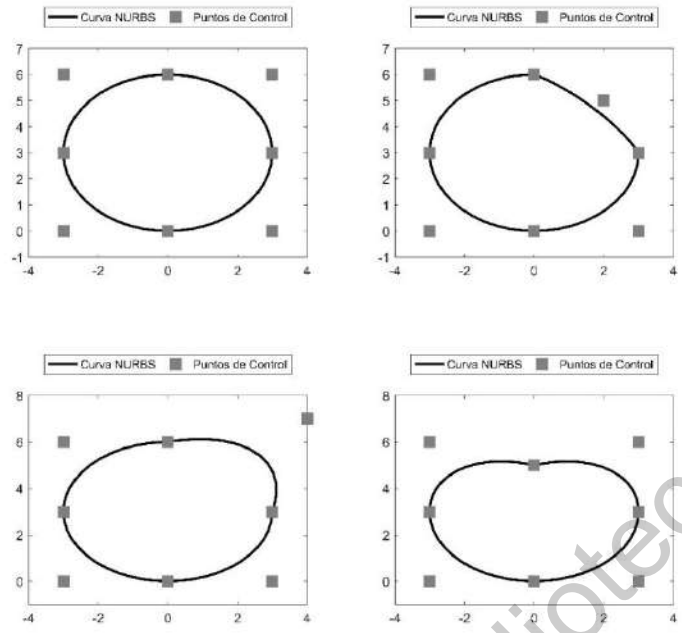


Figura 2-1 Efecto de mover un punto de control en la forma de la curva NURBS.

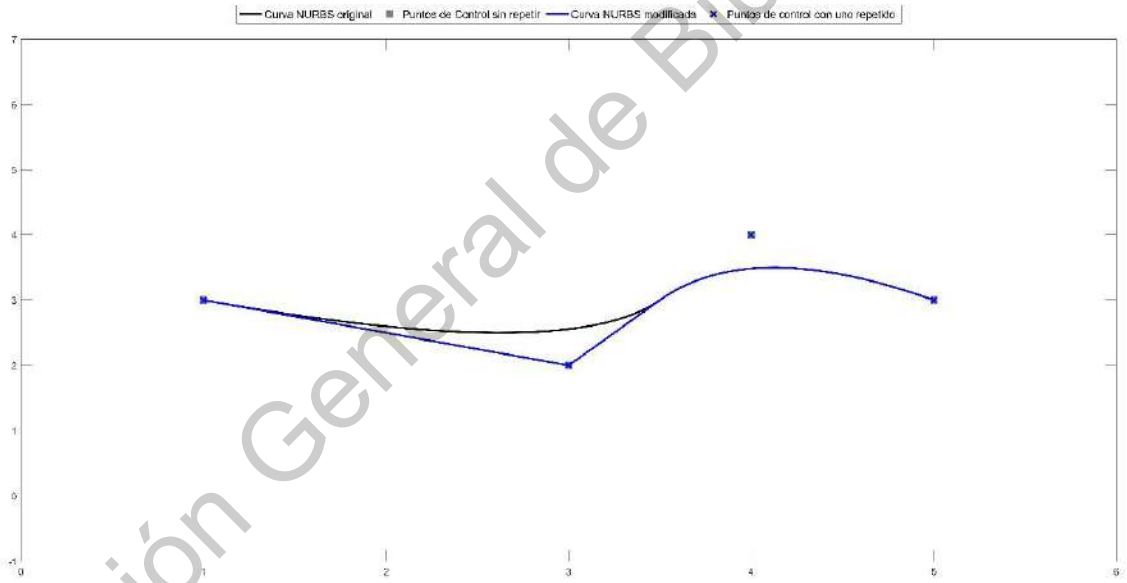


Figura 2-2 Efecto de repetir un nodo de control. La curva de color negro representa una curva NURBS con puntos de control sin repetir, y la curva azul representa una Curva NURBS con el segundo nodo repetido $k-1$ veces.

- **Parámetro ξ**: Como se muestra en la ecuación (2.2), el parámetro ξ que genera la curva varía de un valor mínimo "a" a un máximo "b".
- **Vector de nudos Ξ**: El valor de las funciones base $N_{i,g}$depende del parámetro ξ y del vector de nudos Ξ, o "*knot vector", ecuación* (2.2).

$$\Xi = \left\{a, \kappa_2, \kappa_3 \dots, \kappa_{n_{cp}}, b\right\} \tag{2.2}$$

La influencia de este vector se aprecia en el valor obtenido de las funciones base (Figura 2-3). Algunas características de este vector son:

- Cada elemento del vector debe cumplir con la relación $\kappa_i \leq \kappa_{i+1}$.
- Se clasifican en vectores periódicos y abiertos, uniformes y no uniformes.
- Un vector uniforme es aquél cuyos elementos están espaciados linealmente.
- Un vector abierto uniforme es aquél que tiene nudos interiores espaciados equitativamente y tiene k multiplicidad en valores de los nudos de sus extremos (es decir, los valores del nudo se repiten k veces en los extremos, Figura 2-3 c).
- Los elementos de un vector no uniforme pueden estar espaciados desigualmente y pueden ser múltiples a los extremos o en el interior del vector (Figura 2-3 a, b).
- Una curva generada con un vector periódico de nudos no cubre generalmente los extremos de un polígono de control cerrado, sin embargo, si los puntos de control asociados a estos extremos se repiten $k-2$ veces, se genera una curva cerrada.

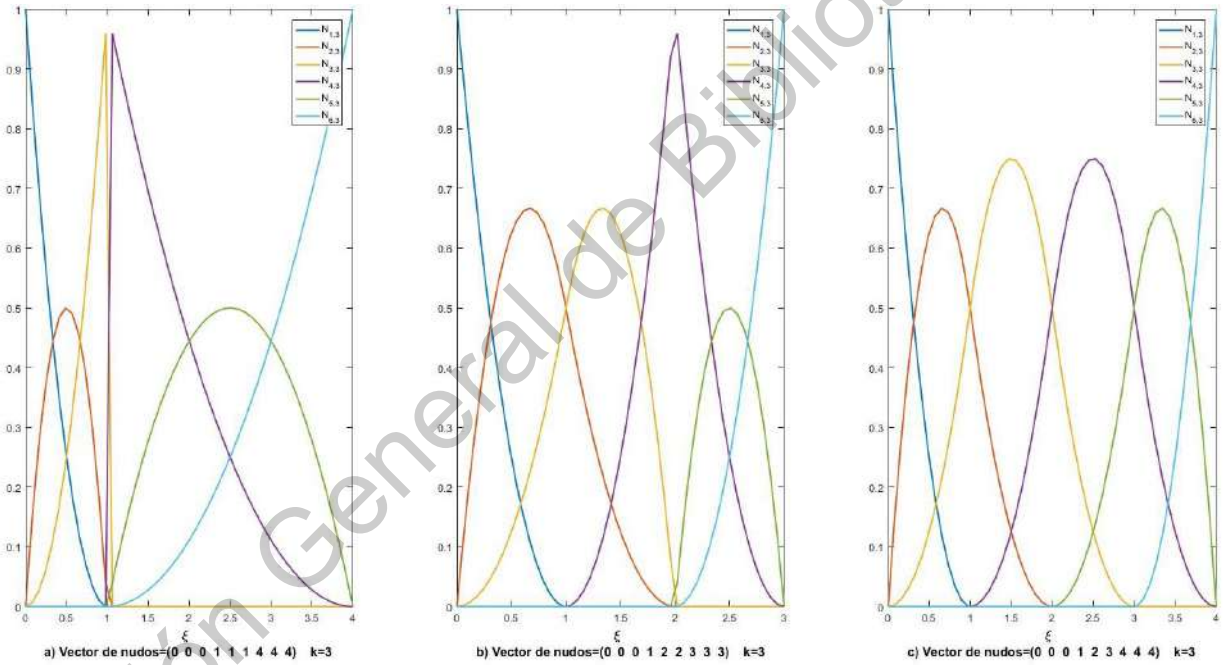


Figura 2-3 Influencia del vector de nudos en la generación de funciones base racionales de segundo grado (o tercer orden).

- Las funciones base se calculan por cada rango de κ.
- Si se repiten valores en el vector de nudos (multiplicidad m), se reduce la diferenciabilidad de la función base a $k-m-1$ en el nudo κ_i.
- La multiplicidad m del nudo debe obedecer la relación $m \leq k-1$.
- El tamaño del vector de nudos será de $n_{cp}+1+k$.

- **Funciones base *B-Spline* $N_{i,p}(\xi)$**: Están definidas por las ecuaciones (2.3, a y b)

$$N_{i,1}(\xi) = \begin{cases} 1 & \text{si } \kappa_i \leq \xi \leq \kappa_{i+1} \\ 0 & \text{otro caso} \end{cases} \quad \text{(a)}$$

$$N_{i,\mathrm{k}}(\xi) = \frac{\xi - \kappa_i}{\kappa_{i+\mathrm{k}-1} - \kappa_i} N_{i,\mathrm{k}-1}(\xi) + \frac{\kappa_{i+\mathrm{k}} - \xi}{\kappa_{i+\mathrm{k}} - \kappa_{i+1}} N_{i+1,\mathrm{k}-1}(\xi) \quad \text{(b)} \tag{2.3}$$

Cuando la función base resulte en $0/0$, se tomará simplemente como 0.

- **Funciones base racionales *B-Spline*:** Una curva *NURBS* también se define como la proyección de una curva no-racional polinomial *B-Spline* en un espacio coordenado homogéneo 4-D [1], [52]. Al trasladar esta *B-Spline* a un espacio físico 3-D, se le asocia un vector de pesos W. Por tanto, las funciones base racionales asociadas a esta curva son:

$$R_{i,k}(\xi) = \frac{w_i N_{i,k}(\xi)}{\sum_{i=1}^{n_{cp}+1} w_i N_{i,k}(\xi)} \tag{2.4}$$

De tal manera que, una curva *NURBS* puede escribirse simplemente como:

$$\Lambda(\xi) = \sum_{i=1}^{n_{cp}+1} B_i R_{i,k}(\xi) \tag{2.5}$$

- **Pesos w_i**: Los pesos aumentan el control de la forma de la curva, la elección de estos pesos puede producir secciones cónicas si el orden de la curva es igual a 3. La Figura 2-4 muestra la influencia de estos pesos en la forma de la curva. La Tabla 1 muestra el tipo de sección resultante de acuerdo a la elección del peso.

Tabla 1 Sección de curva resultante al variar el peso

Peso w_2	Sección generada
$\boldsymbol{w_2 = 0}$	Línea recta
$\boldsymbol{0 < w_2 < 1}$	Segmento de curva elíptico
$\boldsymbol{w_2 = 1}$	Segmento de curva parabólico
$\boldsymbol{w_2 > 1}$	Segmento de curva hiperbólico

2.1.2 Primera derivada de una curva NURBS

La primera derivada de una curva *NURBS* se obtiene al derivar las ecuaciones (2.4) y (2.5) [52]:

$$\Lambda'(\xi) = \sum_{i=1}^{n+1} B_i R'_{i,k}(\xi) \tag{2.6}$$

$$R'_{i,p}(\xi) = \frac{w_i N'_{i,k}(\xi)}{\sum_{i=1}^{n_{cp}+1} w_i N_{i,k}} - \frac{w_i N_{i,k} \sum_{i=1}^{n_{cp}+1} w_i N'_{i,k}}{\left(\sum_{i=1}^{n_{cp}+1} w_i N_{i,k}\right)^2} \tag{2.7}$$

Donde $N'_{i,k}$ es:

$$N'_{i,k} = \frac{N_{i,k-1}(\xi) + (\xi - \kappa_i) N'_{i,k-1}(\xi)}{\kappa_{i+k-1} - \kappa_i} + \frac{(\kappa_{i+k} - \xi) N'_{i+1,k-1}(\xi) - N_{i+1,k-1}(\xi)}{\kappa_{i+k} - \kappa_{i+1}} \tag{2.8}$$

Para más información acerca de las curvas *NURBS* se recomienda la lectura del libro de David F. Rogers [52] para teoría y del libro de Piegl y Tiller [51] para programar *NURBS* con algoritmos optimizados.

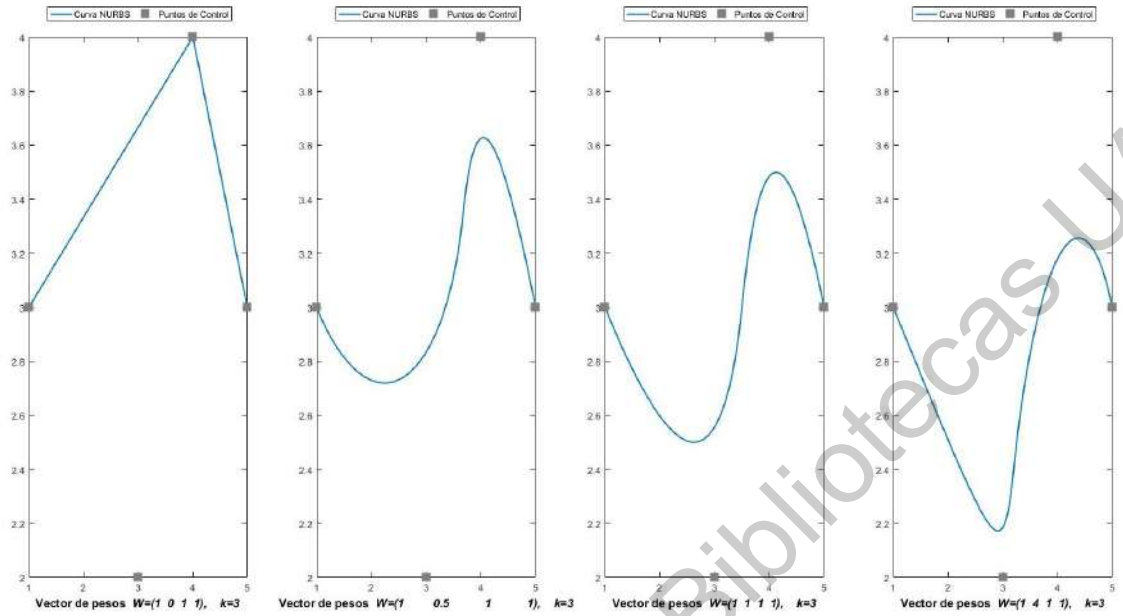


Figura 2-4 Efecto del vector de pesos en la curva NURBS

2.1.3 Círculo generado con NURBS

Un círculo puede crearse utilizando 9 puntos de control, que generarán un polígono de control con cuatro segmentos de 90° (Figura 2-5). De la literatura disponible [52] se toma el orden de la curva ($k = 3$), el vector de nudos (2.9) y el vector de pesos (2.10) para dibujar el círculo.

$$\Xi = [0 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 2 \;\; 2 \;\; 3 \;\; 3 \;\; 4 \;\; 4 \;\; 4] \tag{2.9}$$

$$W = [1 \;\; \sqrt{2}/_2 \;\; 1 \;\; \sqrt{2}/_2 \;\; 1 \;\; \sqrt{2}/_2 \;\; 1 \;\; \sqrt{2}/_2 \;\; 1 \;\; \sqrt{2}/_2 \;\; 1] \tag{2.10}$$

Del vector de nudos, se establece que el rango del parámetro es $0 \leq \xi \leq 4$ y que la curva estará compuesta de 4 tramos cuadráticos racionales, puesto que hay 4 intervalos de los valores del vector de nudos. En el apéndice A, se muestra el código para generar el círculo de la Figura 2-5. Los puntos de control asociados a esta curva son:

$$B = [0 \;\; 0; 3 \;\; 0; 3 \;\; 3; 3 \;\; 6; 0 \;\; 6; -3 \;\; 6; -3 \;\; 3; -3 \;\; 0; 0 \;\; 0] \tag{2.11}$$

2.2 Elementos frontera para problemas elastoestáticos

Existen varias formas de derivar las ecuaciones de elementos frontera para problemas elastoestáticos, sin embargo, para fines de este trabajo se usó la formulación “directa” de Becker [53]. Para llegar a dichas ecuaciones, se utiliza el “teorema del Trabajo recíproco” o “teorema de Betti”, el cual establece que el trabajo hecho por un sistema de esfuerzos (ϖ) sobre los

desplazamientos del sistema (ρ) es igual al trabajo hecho por los esfuerzos del sistema (ρ) sobre los desplazamientos del sistema (ϖ), ecuación (2.12).

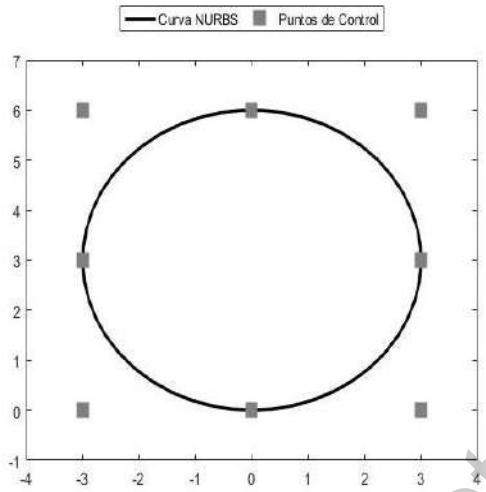


Figura 2-5 Círculo dibujado con NURBS

$$\int_V \sigma_{ij}^{\varpi} \varepsilon_{ij}^{\rho} dV = \int_V \sigma_{ij}^{\rho} \varepsilon_{ij}^{\varpi} dV \tag{2.12}$$

Si las deformaciones se escriben en función de los desplazamientos:

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.13}$$

La expresión (2.12) se convierte en:

$$\int_V \sigma_{ij}^{\varpi} \frac{\partial {u_i}^{\rho}}{\partial x_j} dV = \int_V \sigma_{ij}^{\rho} \frac{\partial {u_i}^{\varpi}}{\partial x_j} dV \tag{2.14}$$

Expandiendo el primer término de la ecuación (2.14):

$$\int_V \sigma_{ij}^{\varpi} \frac{\partial {u_i}^{\rho}}{\partial x_j} dV = \int_V \left[\frac{\partial}{\partial x_j}\left(\sigma_{ij}^{\varpi} u_i^{\rho}\right) - \frac{\partial \sigma_{ij}^{\varpi}}{\partial x_j} u_i^{\rho}\right] dV \tag{2.15}$$

Si en la ecuación anterior se utilizan las fuerzas en equilibrio de un elemento diferencial de un cuerpo:

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \tag{2.16}$$

Entonces la expresión (2.15) se transforma en:

$$\int_V \sigma_{ij}^{\varpi} \frac{\partial {u_i}^{\rho}}{\partial x_j} dV = \int_V \left[\frac{\partial}{\partial x_j}\left(\sigma_{ij}^{\varpi} u_i^{\rho}\right)\right] dV + \int_V f_i^{\varpi} u_i^{\rho} dV \tag{2.17}$$

Para convertir las integrales de volumen en integrales de superficie, se aplica el teorema de la divergencia:

$$\int_V \frac{\partial}{\partial x_j} dV = \int_\Gamma f_i n_i dS \tag{2.18}$$

Entonces la ecuación (2.14) se transforma en (2.19):

$$\int_V \sigma_{ij}^{\varpi} \frac{\partial u_i{}^{\rho}}{\partial x_j}\, dV = \int_\Gamma \left(\sigma_{ij}^{\varpi} u_i^{\rho}\right) n_j dS + \int_V f_i^{\varpi}\, u_i^{\rho}\, dV \tag{2.19}$$

La ecuación anterior puede simplificarse si los esfuerzos se transforman en tracciones como se muestra en la ecuación (2.20). n_j es un vector unitario que apunta al exterior del cuerpo.

$$t_i = \sigma_{ij} n_j \tag{2.20}$$

Si también se realiza este procedimiento con σ_{ij}^{ρ} de (2.14), entonces, se llegará a la siguiente forma del teorema de Betti:

$$\int_\Gamma t_i^{\varpi} u_i^{\rho} dS + \int_V f_i^{\varpi} u_i^{\rho} dV = \int_\Gamma t_i^{\rho} u_i^{\varpi} dS + \int_V f_i^{\rho} u_i^{\varpi} dV \tag{2.21}$$

Donde t_i^{ϖ} y t_i^{ρ} representan las tracciones, u_i^{ϖ} y u_i^{ρ} los desplazamientos y f_i^{ϖ} y f_i^{ρ} las cargas del sistema (ϖ) y (ρ), respectivamente. Para resolver la ecuación (2.21), se toma al sistema (ϖ) como el conjunto a ser resuelto y al sistema (ρ) como un conjunto conocido. Este último sistema corresponde a la solución fundamental de Kelvin, la cual es válida para cualquier cuerpo en equilibrio. Al aplicar esta solución en la ecuación (2.21), como se describe en [54], se llega a la identidad Somigliana para desplazamientos:

$$u_i(p) = -\int_\Gamma T_{ij}(p,Q) u_i(Q) dS + \int_\Gamma t_i(Q) U_{ij}(p,Q) dS + \int_V f_i(q) U_{ij}(p,Q) dV \tag{2.22}$$

T_{ij} y U_{ij} simbolizan los *kernels* de tracción y desplazamiento de la solución de Kelvin, los cuales proporcionan las tracciones y los desplazamientos de cualquier punto Q en la superficie cuando una carga se aplica en el punto interior p (Figura 2-6). Las variables u_i, t_i y f_i representan a los desplazamientos, tracciones y fuerzas del sistema (ϖ). Las ecuaciones (2.23) y (2.24) describen los *kernels* de desplazamiento y tracción para un problema bidimensional elastoestático. Estos *kernels* dependen del coeficiente de Poisson υ, del módulo de cizalladura μ y de la distancia $r[p,Q]$, ecuación (2.25), entre el punto de carga (p) y el punto de campo (Q). Por cada punto de carga y de campo, se calculan los *kernels* de desplazamiento y tracción.

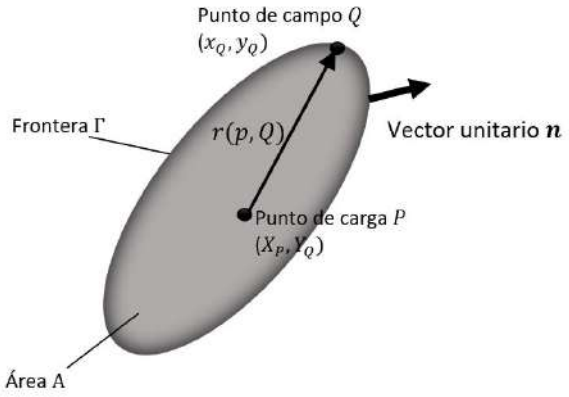


Figura 2-6 Solución de Kelvin, dominio bidimensional

$$U_{ij}\begin{cases} U_{xx}(p,Q)=\dfrac{1}{8\pi\mu(1-\upsilon)}\left[(3-4\nu)\ln\left(\dfrac{1}{r}\right)+(\dfrac{\partial r}{\partial x})^2\right] \\ U_{xy}(p,Q)=U_{yx}(p,Q)=\dfrac{1}{8\pi\mu(1-\upsilon)}\dfrac{\partial r}{\partial x}\dfrac{\partial r}{\partial y} \\ U_{yy}(p,Q)=\dfrac{1}{8\pi\mu(1-\upsilon)}\left[(3-4\nu)\ln\left(\dfrac{1}{r}\right)+(\dfrac{\partial r}{\partial y})^2\right] \end{cases} \tag{2.23}$$

$$T_{ij}\begin{cases} T_{xx}(p,Q)=\dfrac{-1}{4\pi(1-\upsilon)r}\left(\dfrac{\partial r}{\partial n}\right)\left[(1-2\upsilon)+2\left(\dfrac{\partial r}{\partial y}\right)^2\right] \\ T_{xy}(p,Q)=\dfrac{-1}{4\pi(1-\upsilon)r}\left[2\dfrac{\partial r}{\partial x}\dfrac{\partial r}{\partial y}\dfrac{\partial r}{\partial n}+(1-2\upsilon)(\dfrac{\partial r}{\partial y}n_x-\dfrac{\partial r}{\partial x}n_y)\right] \\ T_{yx}(p,Q)=\dfrac{-1}{4\pi(1-\upsilon)r}\left[2\dfrac{\partial r}{\partial x}\dfrac{\partial r}{\partial y}\dfrac{\partial r}{\partial n}-(1-2\upsilon)(\dfrac{\partial r}{\partial y}n_x-\dfrac{\partial r}{\partial x}n_y)\right] \\ T_{yy}(p,Q)=\dfrac{-1}{4\pi(1-\upsilon)r}\left(\dfrac{\partial r}{\partial n}\right)\left[(1-2\upsilon)+2\left(\dfrac{\partial r}{\partial y}\right)^2\right] \end{cases} \tag{2.24}$$

$$r[p,Q]=\sqrt{(X_p-x_Q)^2+(Y_p-y_Q)^2} \tag{2.25}$$

Las ecuaciones (2.22) a (2.25) tiene al punto de carga (p) en el interior del cuerpo y al punto de campo (Q) en el contorno del cuerpo, por tanto, para conseguir el método de elementos frontera, el punto de carga tiene que moverse al contorno. La ecuación (2.26) representa la ecuación Somigliana con el punto de carga en la frontera y con ausencia de fuerzas de cuerpo. Al término $C(P)$ se le conoce como "término de salto"; este elemento mueve al punto de carga (p) del interior al contorno; así, el punto de carga ahora se escribe como (P).

$$C(P)u_i(P)+\int_\Gamma T_{ij}(P,Q)u_i(Q)dS=\int_\Gamma U_{ij}(P,Q)t_i(Q)dS \tag{2.26}$$

Para resolver numéricamente la ecuación anterior, el cuerpo bajo análisis se divide en "elementos" (Figura 2-7), que a su vez están constituidos por "nodos". La geometría de estos elementos se describe mediante funciones de forma que pueden ser lineales, cuadráticas, cúbicas o de orden superior. Si se eligen funciones de forma cuadráticas, la geometría del elemento se genera mediante:

$$\begin{aligned} x(\zeta)&=\textstyle\sum_{i=1}^{3}M_1(\zeta)x_1+M_2(\zeta)x_2+M_3(\zeta)x_3 \\ y(\zeta)&=\textstyle\sum_{i=1}^{3}M_1(\zeta)y_1+M_2(\zeta)y_2+M_3(\zeta)y_3 \end{aligned} \tag{2.27}$$

Donde ζ simboliza la variable del sistema coordenado local del elemento que varía de -1 a 1. x_i y y_i representan las coordenadas de los nodos que definen a cada elemento: un nodo en medio y otros dos nodos en sus extremos (Figura 2-8). Finalmente, M_i representa a las funciones cuadráticas que deben cumplir con las siguientes condiciones:

$$\begin{aligned} M_i(\zeta)&=1 \ \text{ en el nodo } i \\ M_i(\zeta)&=0 \ \text{ en los otros nodos} \end{aligned} \tag{2.28}$$

Las funciones que cumplen estas condiciones son:

$$M_1(\zeta) = \frac{-\zeta}{2}(1-\zeta)$$
$$M_2(\zeta) = (1+\zeta)(1-\zeta) \quad (2.29)$$
$$M_3(\zeta) = \frac{\zeta}{2}(1+\zeta)$$

Si estas funciones se utilizan para describir las variaciones de las variables desconocidas, como desplazamientos (2.30) y tracciones (2.31), entonces se dice que el elemento es isoparamétrico. Es importante mencionar que existe cierta discontinuidad entre los polinomios que representan a los elementos cuando el problema se resuelve numéricamente. Si en algún elemento se aplica tracción o se le asigna desplazamiento prescrito, los elementos vecinos no se verán afectados por esas cargas o desplazamientos, aun cuando compartan nodos.

$$u_x(\zeta) = \sum_{i=1}^{3} M_1(\zeta)(u_x)_1 + M_2(\zeta)(u_x)_2 + M_1(\zeta)(u_x)_3$$
$$u_y(\zeta) = \sum_{i=1}^{3} M_1(\zeta)(u_y)_1 + M_2(\zeta)(u_y)_2 + M_1(\zeta)(u_y)_3 \quad (2.30)$$

$$t_x(\zeta) = \sum_{i=1}^{3} M_1(\zeta)(t_x)_1 + M_2(\zeta)(t_x)_2 + M_1(\zeta)(t_x)_3 \quad (2.31)$$
$$t_y(\zeta) = \sum_{i=1}^{3} M_1(\zeta)(t_y)_1 + M_2(\zeta)(t_y)_2 + M_1(\zeta)(t_y)_3$$

Para generar los *kernels*, cada nodo se toma como punto de carga (P), y cada punto de campo (Q) se obtiene mediante las funciones de forma de la ecuación (2.27), (Figura 2-9). Sin embargo, en la diagonal de las matrices de *kernels* ocurre que el punto de carga (P) es igual al punto de campo (Q), o que el punto (P) es igual a cualquier nodo del elemento que define al punto de campo. Cuando esto sucede, los valores de los *kernels* son singulares o muy cercanos a serlo. Para evitar estas singularidades, se puede seguir el procedimiento detallado en [53].

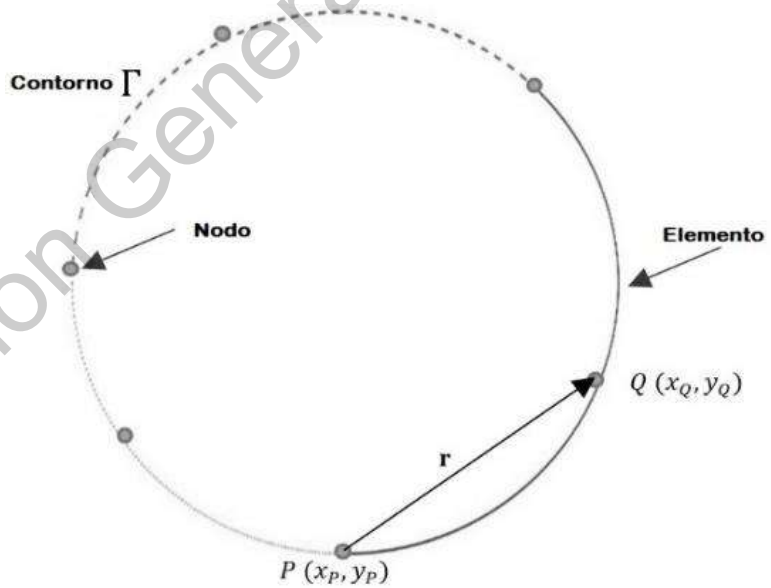


Figura 2-7 Nodos y elementos usados en el método de elementos frontera

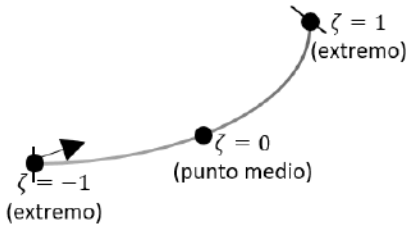


Figura 2-8 Elemento cuadrático, definido por tres nodos

Luego, para integrar los *kernels* calculados se utiliza el método de cuadratura Gaussiana. Dado que el cálculo de estos *kernels* se hace desde el sistema coordenado local ζ, entonces se utilizará el Jacobiano, ecuación (2.32), para transformar la ecuación (2.26) del dominio Γ al dominio de ζ. Las ecuaciones (2.33) y (2.34) representan las derivadas de las coordenadas $x(\zeta)$ y $y(\zeta)$ y de las funciones de forma $M_i(\zeta)$ con respecto a ζ, que se utilizan en el cálculo del Jacobiano. La ecuación (2.35) muestra las componentes del vector unitario que se utiliza en el cálculo de los *kernels* de tracciones, y también para convertir los esfuerzos en tracciones, ecuación (2.20).

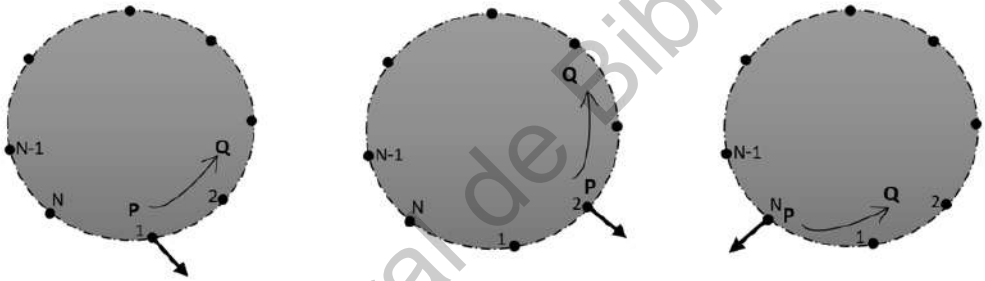


Figura 2-9 Iterando el punto de carga P para calcular los kernels

$$J(\zeta) = \frac{d\Gamma}{d\zeta} = \sqrt{\left[\frac{dx(\zeta)}{d\zeta}\right]^2 + \left[\frac{dy(\zeta)}{d\zeta}\right]^2} \tag{2.32}$$

$$\frac{dx(\zeta)}{d\zeta} = \frac{dM_1(\zeta)}{d\zeta}x_1 + \frac{dM_2(\zeta)}{d\zeta}x_2 + \frac{dM_3(\zeta)}{d\zeta}x_3$$
$$\frac{dy(\zeta)}{d\zeta} = \frac{dM_1(\zeta)}{d\zeta}y_1 + \frac{dM_2(\zeta)}{d\zeta}y_2 + \frac{dM_3(\zeta)}{d\zeta}y_3 \tag{2.33}$$

$$\frac{dM_1(\zeta)}{d\zeta} = \zeta - \frac{1}{2}$$
$$\frac{dM_2(\zeta)}{d\zeta} = -2\zeta \tag{2.34}$$
$$\frac{dM_3(\zeta)}{d\zeta} = \zeta + \frac{1}{2}$$

$$n_x = \frac{1}{J(\zeta)}\left[\frac{dy(\zeta)}{d\zeta}\right]$$
$$n_y = -\frac{1}{J(\zeta)}\left[\frac{dx(\zeta)}{d\zeta}\right] \tag{2.35}$$

Finalmente, la ecuación (2.26) puede reescribirse como:

$$C(P)u_i(P) + \sum_{e=1}^{n_e}\sum_{c=1}^{3}\left[\int_{-1}^{1} T_{ij}\left(P, Q(\zeta)\right)M_c(\zeta)J(\zeta)d\zeta\right]u_i = \sum_{e=1}^{n_e}\sum_{c=1}^{3}\left[\int_{-1}^{1} U_{ij}\left(P, Q(\zeta)\right)M_c(\zeta)J(\zeta)d\zeta\right]t_i \tag{2.36}$$

Donde ζ es la variable local para describir la geometría con funciones de forma cuadráticas; u_i representa los desplazamientos y t_i a las tracciones en el elemento e que forman parte del problema a resolver. n_e denota el número total de elementos, M simboliza las funciones de forma cuadráticas, y J representa la transformación Jacobiana para convertir las coordenadas de la frontera Γ a coordenadas locales. El rango de integración se establece en [-1,1] por el uso de la cuadratura Gaussiana.

Para que el problema se resuelva, se formará un sistema algebraico de ecuaciones:

$$[\boldsymbol{A}][\boldsymbol{u}] = [\boldsymbol{U}][\boldsymbol{t}] \tag{2.37}$$

Donde A simboliza a los coeficientes de los *kernels* de tracción y a los elementos de la matriz de salto asociados a los puntos de carga P. La matriz U representa los *kernels* de desplazamientos también en función de los puntos P. Los vectores $\boldsymbol{u}$ y $\boldsymbol{t}$ simbolizan las variables del problema a ser resuelto. Para que el sistema tenga solución única, se aplicarán condiciones de frontera en cada nodo, ya sean tracciones (t_x y t_y) o desplazamientos (u_x y u_y) prescritos. Si el nodo no tiene algún valor prescrito, se asumirá que el nodo estará libre de esfuerzos; es decir, que las tracciones t_x y t_y serán igual a cero.

Antes de resolver el sistema de ecuaciones de (2.37), se tomará en cuenta que los *kernels* de tracción son de magnitud mucho mayor que los *kernels* de desplazamiento, por lo que debe aplicarse un factor de escala. Becker [53] calcula este factor dividiendo el módulo de elasticidad E entre la distancia máxima (d_{max}) que hay entre los nodos, ecuación (2.38). Por otra parte, en el programa propuesto por Scott *et al.* [29] utilizan como factor de escala a la traza de la matriz $\boldsymbol{A}/\boldsymbol{U}$ de (2.37).

$$factor = \frac{E}{d_{max}} \tag{2.38}$$

2.3 Algoritmos de optimización

Un algoritmo de optimización es un método para encontrar el máximo o el mínimo de una función llamada función objetivo o función de ajuste. Hay una amplia variedad de algoritmos de optimización que podríamos clasificar en programación matemática, técnicas de procesos estocásticos y métodos estadísticos, Rao [55]. Dentro de la categoría de programación matemática tenemos a las técnicas de optimización no tradicional o moderna, que se desarrollaron con el auge de las computadoras y se implementan fácilmente para resolver problemas complejos. Estas técnicas no tradicionales se pueden catalogar en algoritmos basados en la naturaleza, algoritmos basados en trayectorias y en algoritmos basados en población o con un solo punto de búsqueda [56]. Dentro de la categoría de algoritmos basados en la naturaleza se encuentran los algoritmos evolutivos y los algoritmos de inteligencia de manada.

La premisa de los algoritmos evolutivos se basa en la supervivencia del más fuerte; los mejores "individuos" (soluciones candidatas para la función objetivo) se seleccionan para reproducirlos, y mediante los operadores de combinación y mutación, se crea la siguiente generación; el ciclo se repetirá hasta que se alcance un criterio de terminación [57]. En esta subclasificación de algoritmos encontramos a los algoritmos genéticos [58], estrategias de evolución [59], [60], programación evolutiva [61] y evolución diferencial [62].

Los algoritmos de inteligencia de manada se fundamentan en el comportamiento colectivo y en las interacciones descentralizadas de cualquier conjunto de animales con la capacidad de reaccionar a cambios ambientales y tomar decisiones. En esta categoría se encuentra el algoritmo de cúmulo de partículas [63], la optimización por colonia de hormigas [64] y el algoritmo del murciélago [65], entre otros.

Dentro de la clase de algoritmos basados en trayectoria, las técnicas más representativas son recocido simulado [66] y búsqueda tabú [67]. Estos algoritmos se caracterizan por seguir una ruta, y encontrar una mejor solución explorando a sus alrededores; bajo ciertas condiciones permiten peores soluciones para evitar mínimos o máximos locales y así, llegar a la mejor solución global. Respecto a la clasificación de algoritmos basados en población o en un solo punto de búsqueda, podemos englobar a todos los algoritmos basados en la naturaleza en la primera categoría, y a los algoritmos basados en la trayectoria en la segunda categoría.

Estas técnicas no tradicionales se han aplicado en una amplia diversidad de problemas de optimización: mejora de procesos [68], [69], diseño de construcciones [70], ahorro de consumo energético [71], planeación de rutas [72]–[79], diseño de elementos eléctricos y electrónicos [80]–[82], diseño de elementos mecánicos [83]–[85], entre muchas otras aplicaciones. Entre las técnicas no tradicionales destacan los algoritmos genéticos, el algoritmo de cúmulo de partículas, y recientemente, el algoritmo del murciélago, por su facilidad de implementación y convergencia. Para elegir el método de optimización más eficaz, se realizó una comparación entre estos métodos sobresalientes; los resultados obtenidos y la metodología usada para compararlos se reportan en [86]. De este estudio, se determinó que el algoritmo más eficiente es el de cúmulo de partículas.

2.3.1 Cúmulo de partículas

El algoritmo de cúmulo de partículas, o *PSO* por sus siglas en inglés *(Particle Swarm Optimization*), fue propuesto en 1995 por Kennedy y Eberhart [87]. Este algoritmo está inspirado en el comportamiento de manadas de animales. El diagrama de flujo de este método se muestra en la Figura 2-10. El primer paso para ejecutarlo es establecer el tamaño de la población (N_{swarm}), cada individuo de esta población tiene un conjunto de variables de optimización, o variables de diseño, que representan su posición (X_i) en el espacio de soluciones. Estos individuos cambian su posición en el espacio ajustando sus velocidades V_i. Como X_i y V_i dependen de los valores pasados de velocidad y posición, cada partícula (o individuo) guarda su mejor posición (L_i). También se guarda la mejor posición de todas las partículas (G_N). Las posiciones y velocidades se actualizan con las

ecuaciones (2.39) y (2.40). Del libro de Clerc [88], se obtienen valores de referencia para las constantes C_1 y C_{max}. Asimismo, los valores de posición deben de pertenecer al espacio de solución deseado, es decir, deben corresponder a un rango de valores que se establezca previamente.

$$V_{i+1} = C_1 V_i + C_{max} rand(0,1)(L_i - X_i) + C_{max} rand(0,1)(G_N - X_i) \quad (2.39)$$

$$X_{i+1} = X_i + V_i \quad (2.40)$$

Para obtener la mejor posición local L_i de cada individuo y la mejor posición global (G_N), se sustituyen en cada iteración las variables de diseño de cada individuo en la función objetivo. Si los nuevos valores son mejores que los anteriores, entonces se actualizan estas variables como se describe en el párrafo anterior.

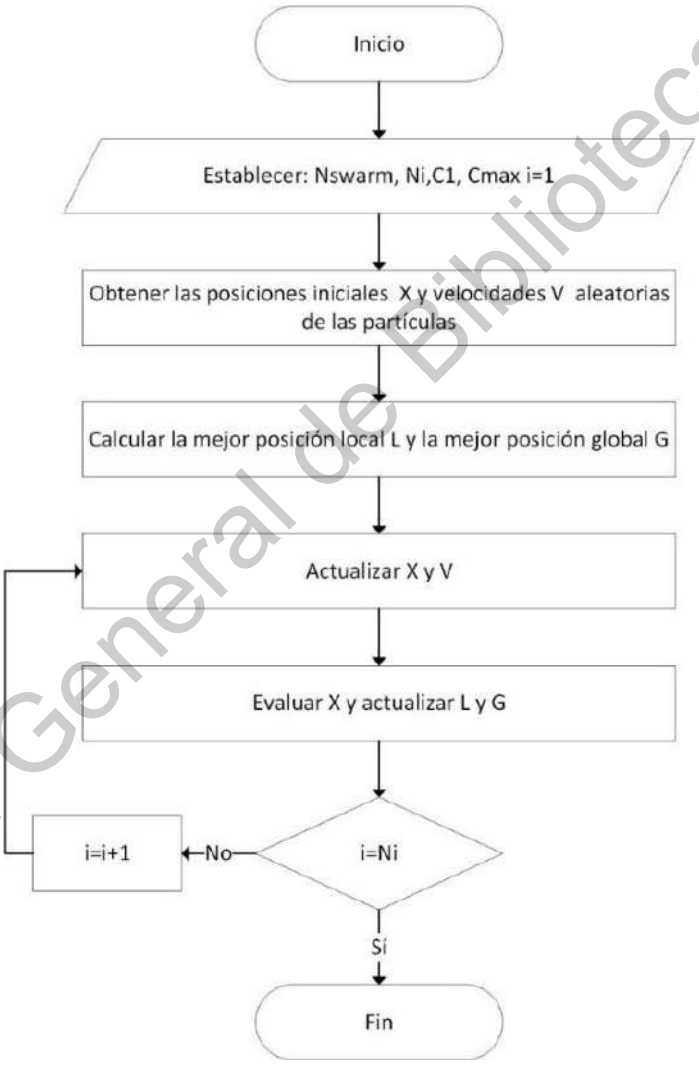


Figura 2-10 Diagrama de flujo del algoritmo de cúmulo de partículas

CAPÍTULO III

3. Metodología

Para resolver el problema de contacto dinámico entre elementos mecánicos, se estableció la ecuación de contacto. Dado que la mayoría de las propuestas conocidas hasta ahora tratan de resolver el problema desde la implementación numérica, se optó por otro camino: replantear la formulación base.

Sin embargo, para replantear esta formulación base se tenía que elegir entre continuar con elemento finito, o tomar alguno de los otros métodos numéricos usados en mecánica del medio continúo. Después de haber hecho el estado del arte, fue claro que con elementos frontera se disminuirían los nodos requeridos para efectuar el análisis.

En esta investigación, se tomaron las mejores características de *IGA* para superar las desventajas del *BEM* convencional. Los puntos de análisis de la geometría se separaron de los cuerpos en contacto y se utilizó un algoritmo de optimización para encontrar el área de contacto. De esta forma, los puntos de control de las *NURBS* se modificaron repetidamente hasta encontrar el área correcta. Con el método propuesto, el dominio de la solución es continuo al definir el contorno con una sola *NURBS* y, con menos puntos de análisis. Para validar la metodología propuesta se analizó un cilindro bajo una carga y el contacto entre dos cilindros después de aplicar una carga.

3.1 Método propuesto de elementos frontera isogeométrico

Las pautas para combinar el *IGA-BEM* se establecieron por Simpson *et al.* [53]; así, la ecuación (2.36) en términos de curvas *NURBS* se transforma en (3.1). Como se observa, el parámetro ξ sustituye al parámetro ζ, y las funciones de forma cuadráticas se sustituyen por funciones base racionales R_c.

$$C(P)u_i(P) + \sum_{e=1}^{n_e}\sum_{c=1}^{k}\left[\int_{-1}^{1} T_{ij}\left(P, Q(\xi)\right)R_c(\xi)J^e(\xi)d\xi\right]u_i^e = \sum_{e=1}^{n_e}\sum_{c=1}^{k}\left[\int_{-1}^{1} U_{ij}\left(P, Q(\xi)\right)R_c(\xi)J^e(\xi)d\xi\right]t_i^e \tag{3.1}$$

A diferencia del método planteado por Simpson, con el método propuesto el contorno del cuerpo se genera con una sola NURBS, es decir, con un “macroelemento”. De tal manera que no hay necesidad de integrar los elementos. Sin embargo, se requiere la integración del Jacobiano porque el macroelemento representa el contorno completo del cuerpo. El resultado de esta integración es el perímetro del cuerpo.

En adición a las condiciones de frontera, en el método propuesto se tienen tres conjuntos que definen el problema a analizar:

- Los valores del parámetro ξ para obtener los nodos P y Q en el contorno, que generaran los *kernels* de desplazamiento.

- Los puntos de control, B ,que definen la forma del cuerpo.
- Los valores del parámetro ξ para calcular el Jacobiano del macroelemento.

La ecuación integral de la frontera propuesta se muestra en la ecuación (3.2), donde η representa los valores de ξ para calcular el Jacobiano, los cuales son diferentes de los valores de ξ para calcular los nodos P y Q.

$$C(P(\xi))u_i(P(\xi)) + T_{ij}(P(\xi), Q(\xi))u_i(Q(\xi)) \int_\Gamma J(\eta)d\eta = U_{ij}(P(\xi), Q(\xi))t_i(Q(\xi)) \int_\Gamma J(\eta)d\eta \tag{3.2}$$

En el *BEM* convencional, el termino de salto C se calcula indirectamente al utilizar consideraciones de cuerpo rígido [53]. Sin embargo, estas consideraciones no son válidas debido a la naturaleza de las *NURBS* [28], por tanto, el termino de salto se calcula explícitamente como en [89]. No obstante, la matriz del término de salto se reduce a (3.3), debido a la forma geométrica (circular) de los cuerpos bajo análisis. Esta matriz solo afecta a la diagonal de los *kernels* de tracción cuando el punto de carga (P), coincide con el punto de (Q).

$$C(P) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \tag{3.3}$$

Puesto que la frontera Γ es función de ξ, el término que corresponde a la transformación Jacobiana $J(\eta)$ se define como:

$$J(\eta) = J(\eta) = \frac{d\Gamma}{d\eta} = \sqrt{\left(\frac{dx(\xi)}{d\xi}\right)^2 + \left(\frac{dy(\xi)}{d\xi}\right)^2} \tag{3.4}$$

El diagrama de flujo del método propuesto de elementos frontera isogeométrico se muestra en la Figura 3-1. Como se observa, el primer paso consiste en leer las propiedades del material y los datos de las NURBS que definen el cuerpo bajo análisis, es decir, obtener el módulo de Young (E), el coeficiente de Poisson (ν), los puntos de control (B), el vector de nudos (Ξ), y el orden de la curva (k). Luego, se elige una cantidad N_{points} de puntos de carga/campo. Las coordenadas de estos puntos se generan al reemplazar el valor de ξ en la función *NURBS.*

El siguiente paso es calcular los *kernels* por cada combinación de puntos de carga y de campo. Cada *kernel* se guarda en una matriz global de desplazamientos (U) y tracciones (T) de tamaño $(N_{points}\ x\ 2)x(N_{points}\ x\ 2)$. Cuando el punto de carga coincide con el punto de campo, el cálculo del *kernel* de tracción se salta y se reemplaza por el termino salto descrito en la ecuación (3.3). En el caso del *kernel* de desplazamiento, se agrega o se resta una tolerancia al punto de campo, con el objetivo de evitar que el término $(1/r)$ se vaya al infinito. Los *kernels* globales obtenidos se multiplican por la integral del contorno del cuerpo, ecuación (3.4); si se requiere un valor más exacto de esta integral, se deberán agregar más valores del parámetro η, que son diferentes de los valores de ξ que se usan para el cálculo de los puntos de campo y de carga.

Siguiendo el diagrama de flujo, el siguiente paso es aplicar las condiciones de frontera para generar una solución única del tipo $\boldsymbol{\Omega}\Psi = \boldsymbol{b}$. Es decir, es necesario acomodar las variables desconocidas de un lado, ya sean desplazamientos o tracciones y del otro lado, los valores conocidos. Los *kernels* se guardarán en la matriz $\boldsymbol{\Omega}$, y los valores prescritos de tracción y de desplazamiento se colocarán en el vector $\boldsymbol{b}$. Luego, se resuelve el sistema de ecuaciones para encontrar las tracciones y desplazamientos desconocidos Ψ. Las siguientes ecuaciones representan algunos elementos adicionales para calcular los *kernels*. Los términos $\frac{dy(\xi)}{d\xi}$ y $\frac{dx(\xi)}{d\xi}$ se obtienen de la ecuación (2.7).

$$\frac{dr}{dx} = \frac{X_Q(\xi) - X_P(\xi)}{r(P(\xi), Q(\xi))} \tag{3.5}$$

$$\frac{dr}{dy} = \frac{Y_Q(\xi) - Y_P(\xi)}{r(P(\xi), Q(\xi))} \tag{3.6}$$

$$n_x = \frac{dx}{dn} = \frac{1}{J(\xi)}\left[\frac{dy(\xi)}{d\xi}\right] \tag{3.7}$$

$$n_y = \frac{dy}{dn} = -\frac{1}{J(\xi)}\left[\frac{dx(\xi)}{d\xi}\right] \tag{3.8}$$

$$\frac{dr}{dn} = \frac{dr}{dx}\frac{dx}{dn} + \frac{dr}{dy}\frac{dy}{dn} \tag{3.9}$$

3.2 Algoritmo de contacto

El contacto se analizó siguiendo las ecuaciones unilaterales descritas en los problemas de Sognorini (contacto entre un cuerpo elástico y uno rígido) y de Winkler-Westergaard (contacto entre dos cuerpos elásticos). Una de estas ecuaciones, (3.10), establece que los desplazamientos normales de un cuerpo con respecto a otro, son menores o iguales a cero; por lo tanto, no hay penetración entre ellos.

$$u(x) \cdot n|_\Gamma \leq 0 \tag{3.10}$$

Entonces, si a dos cuerpos que están en contacto se les aplica una carga, se deformarán sin penetrarse y alcanzarán el equilibrio. Si este contacto se analiza con el método propuesto de la sección 3.1, podemos plantear las siguientes ecuaciones:

$$C(P(\xi))u_i(P(\xi)) + T_{ij}(P(\xi), Q(\xi))u_i(Q(\xi)) \int_{\Gamma_1} J(\eta)d\eta + T_{ij}(P(\xi), Q(\xi))u_i(Q(\xi)) \int_{\Gamma_{AB}} J(\eta)d\eta = U_{ij}(P(\xi), Q(\xi))t_i(Q(\xi)) \int_{\Gamma_1} J(\eta)d\eta + U_{ij}(P(\xi), Q(\xi))t_i(Q(\xi)) \int_{\Gamma_{AB}} J(\eta)d\eta \tag{3.11}$$

$$C(P(\xi))u_i(P(\xi)) + T_{ij}(P(\xi), Q(\xi))u_i(Q(\xi)) \int_{\Gamma_2} J(\eta)d\eta + T_{ij}(P(\xi), Q(\xi))u_i(Q(\xi)) \int_{\Gamma_{AB}} J(\eta)d\eta = U_{ij}(P(\xi), Q(\xi))t_i(Q(\xi)) \int_{\Gamma_2} J(\eta)d\eta + U_{ij}(P(\xi), Q(\xi))t_i(Q(\xi)) \int_{\Gamma_{AB}} J(\eta)d\eta \tag{3.12}$$

Donde Γ_1 y Γ_2 representan los contornos de los cuerpos 1 y 2, respectivamente, que no están en contacto. Γ_{AB} simboliza la región en común entre ambos cuerpos (Figura 3-2), es decir, la zona de contacto. Como esta zona se desconoce después de aplicar la carga, entonces se usará un

algoritmo de optimización para encontrarla. En la Figura 3-3 se muestra el diagrama de flujo de la metodología de contacto propuesta.

El primer paso del método de contacto propuesto consiste en ejecutar el algoritmo de la Figura 3-1. Después, se encontrará la intersección entre los dos cuerpos definida por los puntos A y B. Si estos cuerpos tienen el mismo módulo de Young, y si la carga es lo suficientemente grande, la naturaleza del método *IGA-BEM* podría generar penetración entre los cuerpos. Sin embargo, los cuerpos deberían deformarse sin penetrarse, tal como lo establece la ecuación (3.10). Esta deformación depende de la fuerza aplicada y de las propiedades de los materiales. Para encontrar la deformación correcta, se modificarán los puntos de control utilizando el algoritmo de cúmulo de partículas, y se dejará al área de contacto fija. Las nuevas geometrías deberán tener las mismas áreas que los cuerpos originales.

Para asegurar que el área de contacto sea correcta, se reaplicará la misma carga. Si la zona de contacto no cambia, o el cambio es menor a una tolerancia, el algoritmo se detendrá. De lo contrario, seguirá iterando hasta encontrar la zona de contacto correcta.

3.2.1 Optimización con cúmulo de partículas para encontrar los cuerpos deformados

Para ejecutar el *PSO* descrito en la sección 2.3.1, se estableció primeramente la función objetivo, ecuación (3.13), donde I corresponde al perímetro del cuerpo deformado e I_{or} simboliza el perímetro original del cuerpo (antes de aplicar la carga). Luego, se programó el algoritmo para que variara los puntos de control que definen los cuerpos deformados, asumiendo que la zona de contacto es una línea recta, Figura 3-4.

$$I_{or} - I = 0 \tag{3.13}$$

Por consiguiente, se eligieron como variables de diseño a los puntos de control (B) que están al exterior de la zona de contacto. Para optimizar estas variables, se utilizaron las ecuaciones de velocidad y posición del *PSO*, (2.39) y (2.40). Donde la variable X_i representa los puntos de control que se estarán optimizando en cada iteración. Estos datos se sustituyen en la función objetivo (3.13). Si el perímetro calculado es mayor o menor que el perímetro original, se aplicará una penalización sumando o restando una cantidad a ese perímetro obtenido, ecuaciones (3.14) y (3.15).

La variable ϱ representa una cantidad proporcional a la diferencia entre los valores I e I_{or}. Para actualizar la mejor posición local de cada partícula, L_i, la I penalizada se comparará con el mejor perímetro local guardado, si el nuevo valor es mejor que el guardado, entonces se actualizará la mejor posición local de la partícula L_i. Este mismo procedimiento se hace para actualizar la mejor posición global de las partículas.

$$Si\ I > I_{or} \Longrightarrow I = I + abs(\varrho I) \tag{3.14}$$

$$Si\ I < I_{or} \Longrightarrow I = I - (\varrho I) \tag{3.15}$$

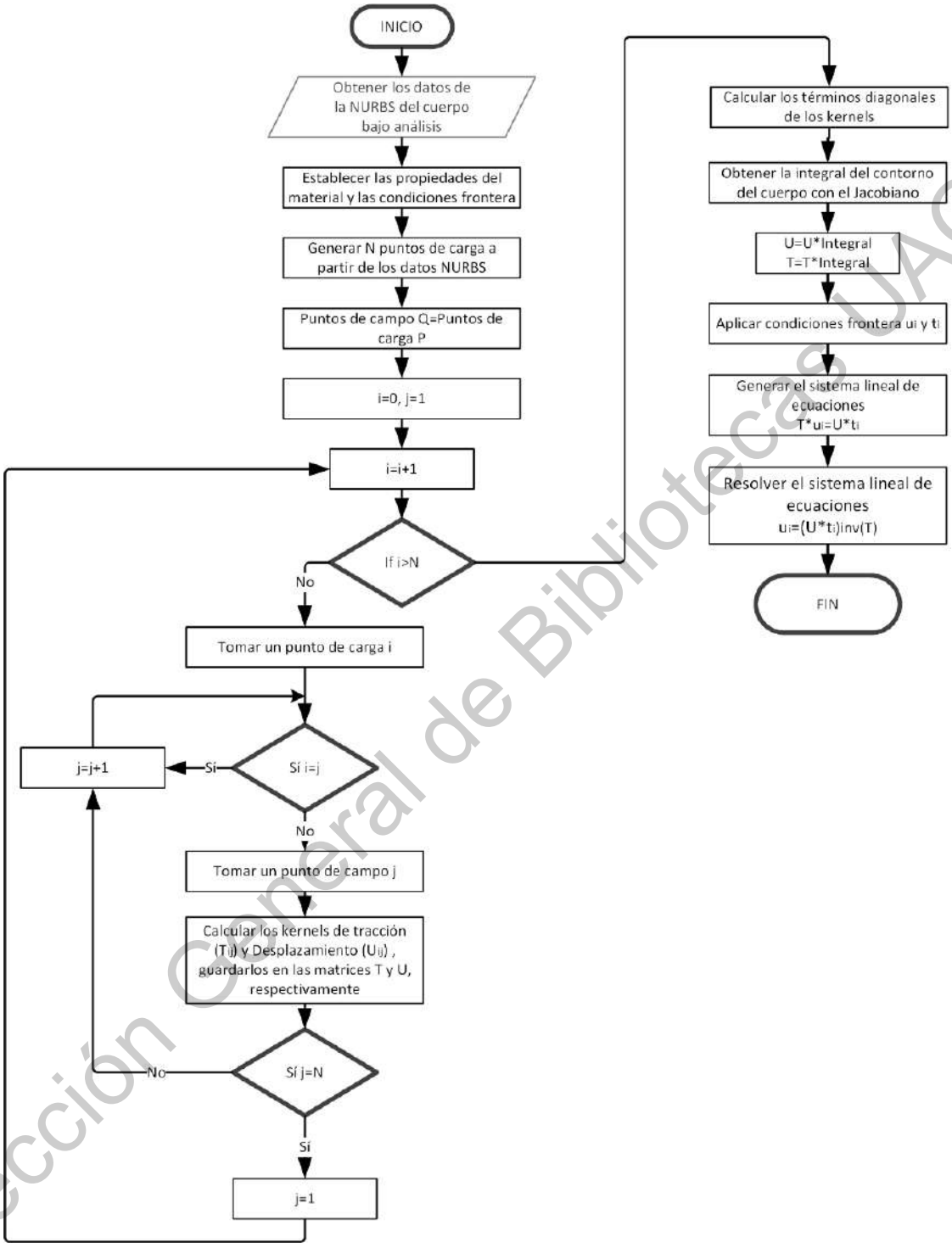


Figura 3-1 Diagrama de flujo del método propuesto de elementos frontera isogeométrico

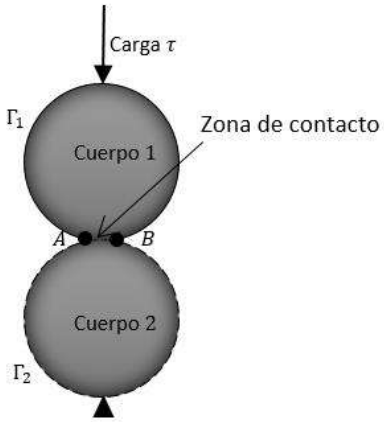


Figura 3-2 Zona de contacto entre dos cuerpos

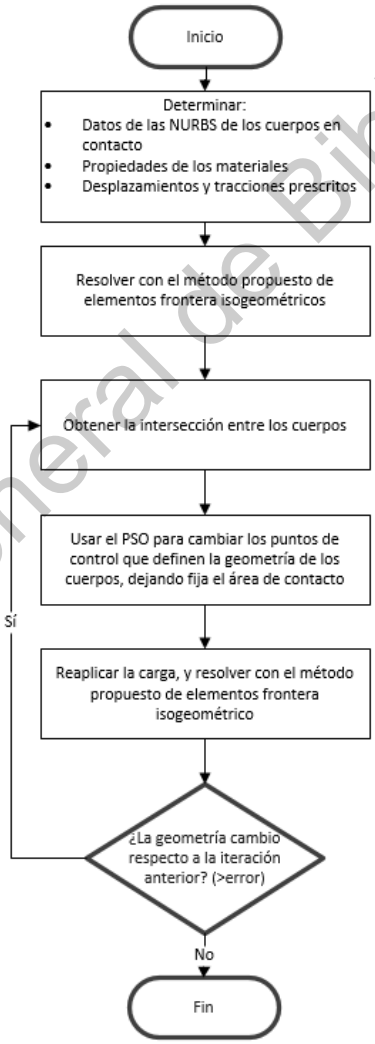


Figura 3-3 Diagrama de flujo para analizar el contacto entre dos cuerpos con el método propuesto IGA-BEM.

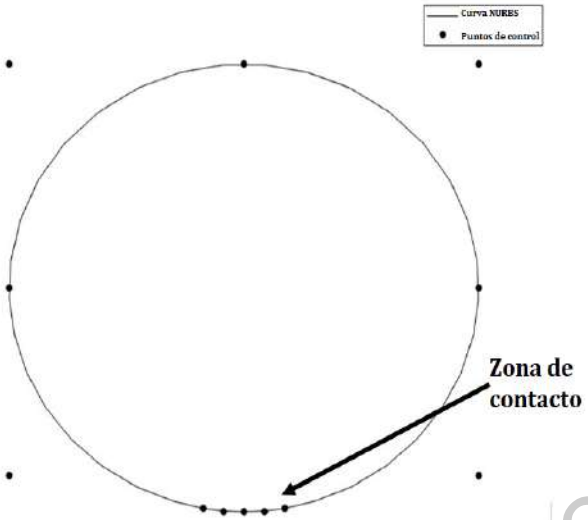


Figura 3-4 Puntos de control y zona de contacto.

3.3 Validación de los algoritmos

3.3.1 Validación del método propuesto del método propuesto de elementos frontera isogeométrico

Para validar la metodología de la sección 3.1, se hizo un programa en Matlab para resolver un problema de deformación plana. Los resultados obtenidos se compararon con el método de elementos finitos (usando un software *CAE*) y con el método de elementos frontera convencional (basado en el programa de Becker [53]). La Figura 3-5 muestra el problema a resolver: una carga se aplica en la parte superior del cilindro, y en su lado opuesto se agrega una restricción de desplazamiento. Como se trata de un problema plano, no es necesario crear el modelo en 3D del cilindro, solo se requiere analizar su sección transversal. Las dimensiones del cilindro y las propiedades de su material se especifican en Tabla 2, así como el valor de la carga τ. Los resultados de este problema se presentan en la siguiente sección.

Tabla 2 Propiedades del cuerpo bajo análisis y carga aplicada

Carga τ	Módulo de Young E	Coeficiente de Poisson v	Diámetro del círculo
$-4500\ N$, Componente Y	$2x10^{11}\ Pa$	0.3	6 mm

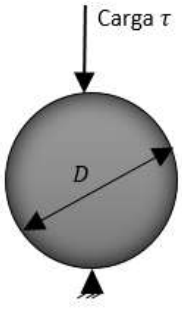


Figura 3-5 Cilindro bajo una carga τ y un punto fijo.

3.3.2 Validación del método de contacto propuesto

Para validar al algoritmo de contacto se resolvió el problema de contacto entre dos cilindros (Figura 3-6) y sus resultados se compararon con los obtenidos por el método de elementos finitos y por el modelo analítico de Hertz; este problema también se analizó como deformación plana. Las especificaciones de los materiales se describen en la Tabla 2. Se asumió un comportamiento simétrico en la deformación para ambos elementos. Los resultados de esta validación se describen en la sección 4.

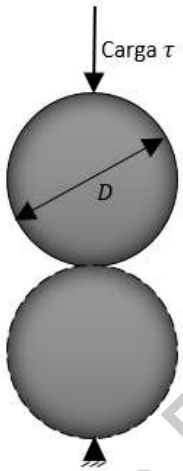


Figura 3-6 Cilindros en contacto bajo una carga τ y con un punto fijo.

3.4 Implementación en software de las metodologías propuestas

La programación de las metodologías del *IGA-BEM* propuesto y del algoritmo de contacto está basada en lo descrito en las secciones 3.1 y 3.2. Sin embargo, en este apartado se complementan los lineamientos de dichas secciones, y se enlistan los programas generados en el software de Matlab, cuyos códigos se encuentran en la sección de Apéndices.

3.4.1 Implementación del IGA-BEM propuesto

El algoritmo de implementación se muestra en la Figura 3-7 y en la Figura 3-8; el programa del apéndice que corresponde a este algoritmo es "Principal.m". El primer paso del algoritmo es definir el orden de la curva k, el vector de nudos Ξ, los puntos de control B y el vector de pesos W. Las características de la curva *NURBS* que genera la sección transversal del cilindro (problema a resolver), se definieron en la sección 2.1.3. Luego, mediante la función "*size*" de Matlab se obtiene el tamaño del vector de nudos (tam). El siguiente paso es calcular el número n de funciones base mediante la siguiente ecuación:

$$n = tam - k \tag{3.16}$$

Esta operación se hace para implementar el algoritmo optimizado de *NURBS* de [51], el cual solo calcula las funciones base que son diferentes de cero. Luego, se determina el número de puntos de análisis N_{points}; para este problema se eligieron 24 puntos. Posteriormente, utilizando el rango del vector de nudos, $(0 \leq \xi \leq 4)$, se obtienen $N_{points} - 2$ valores de ξ espaciados linealmente (función

"*linspace*"). Para evitar que el primer y el último número de ξ generen la misma coordenada (NURBS cerrada), se le resta $1/100$ al extremo del rango antes de generar los valores de ξ $(4-1/100)$. Luego, se agregan dos valores de ξ donde se aplicará la carga τ. Es decir, la carga τ estará distribuida en dos puntos. Así, los valores de ξ para generar las coordenadas de los puntos de análisis son:

$$\begin{bmatrix} 0.0, 0.1995, 0.3890, 0.5786, 0.7681, 0.9576, \\ 1.1471, 1.3367, 1.5262, 1.7157, 1.9052, 1.9900^{*}, \\ 2.01^{*}, 2.0948, 2.2843, 2.4738, 2.6633, 2.8529, 3.0424, \\ 3.2319, 3.4214, 3.6110, 3.8005, 3.99 \end{bmatrix} \tag{3.17}$$

Después, utilizando el algoritmo optimizado para generar *NURBS* de [51], se obtendrán las coordenadas de los puntos de análisis con base en los valores de (3.17). En el apéndice, la función que genera los nodos se llama "DeriveNURBS_Point.m", esta función también calcula las derivadas de los puntos de análisis, utilizando la ecuación (2.6). La Tabla 3 muestra las coordenadas de los puntos de análisis obtenidas a partir de los valores del parámetro ξ de (3.17).

Tabla 3 Coordenadas de los puntos de análisis

No. De punto de análisis	X	Y	No. De punto de análisis	X	Y	No. De punto de análisis	X	Y
1	0.0425489	0.00030175	9	2.0273152	5.21133288	17	-2.60878875	4.4812904
2	0.87930492	0.13175614	10	1.25531267	5.7247367	18	-2.9298933	3.64476761
3	1.69907715	0.52752415	11	0.41156306	5.97163521	19	-2.99448034	2.81810032
4	2.37841234	1.17157042	12*	0.0425489	5.99969825	20	-2.81986465	1.97610383
5	2.81986465	1.97610383	13*	-0.0425489	5.99969825	21	-2.37841234	1.17157042
6	2.99448034	2.81810032	14	-0.41156306	5.97163521	22	-1.69907715	0.52752415
7	2.9298933	3.64476761	15	-1.25531267	5.7247367	23	-0.87930492	0.13175614
8	2.60878875	4.4812904	16	-2.0273152	5.21133288	24	-0.0425489	0.00030175

**Puntos donde se aplica la carga*

Posteriormente, se determina el módulo de Young $(E = 2x10^{11})$ y el coeficiente de Poisson $(\nu = .3)$ y se calcula el módulo de corte μ:

$$\mu = \frac{E}{2 * (1+\nu)} \tag{3.18}$$

Luego, se leen los desplazamientos y tracciones prescritos del archivo "Prescritos.xlsx", utilizando la función "xlsread". Finalmente, se calculan los *kernels*. Se toma un punto de análisis i como punto de carga, y los demás puntos de análisis como puntos de campo (j). Cuando el punto de carga i coincida con el punto de campo j, el cálculo de *kernels* se salta.

Por ejemplo, para calcular el $kernel_{i=1,j=2}$, se toma como nodo de carga P a la coordenada del punto de análisis 1, y a las coordenadas del punto de análisis 2 como nodo de campo Q. Luego, se calcula la distancia r entre los nodos utilizando la ecuación (2.25) y se obtienen los valores de $\frac{dr}{dx}$ y $\frac{dr}{dy}$ con las ecuaciones (3.5) y (3.6). Posteriormente, se calcula el Jacobiano $\frac{d\Gamma}{d\xi}$ con (3.4), y se

obtienen las normales n_x y n_y, con (3.7) y (3.8), para finalmente calcular $\frac{dr}{dn}$ con (3.9). Estos valores se sustituyen en las ecuaciones de los *kernels* (2.23) y (2.24), y se guardan los resultados en las matrices de *kernels* global T y U. Este proceso se hace por cada punto de análisis, de tal manera que el tamaño de cada matriz global será de $[N_{points} \cdot 2]\ x\ [N_{points} \cdot 2]$.

El siguiente paso, es calcular las diagonales de las matrices globales de *kernels*. La diagonal de la matriz de tracciones se calcula sustituyendo los valores de la matriz de salto, ecuación (3.3). Es decir:

$$T_{jj} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{3.19}$$

Para calcular la diagonal de la matriz de desplazamientos, por cada punto de análisis, se toman sus coordenadas como puntos de carga, luego, al valor de ξ correspondiente a este punto de análisis se le incrementa una tolerancia de 2^{-40} (constante de Matlab llamada "*eps*"), multiplicada por $1x10^{12}$ ($\cong .9095$). El valor de la tolerancia se obtuvo experimentalmente. Entonces, se generan las coordenadas con este nuevo valor de ξ. Con $r \neq 0$, se procede al cálculo de *kernel* de desplazamiento con (2.23).

Posteriormente, se calcula la integral del contorno del macroelemento utilizando la ecuación 3.4. Sin embargo, los valores de ξ para este cálculo (denotados ahora como η) son diferentes de los valores de ξ que se utilizaron para generar las coordenadas de los puntos de carga/campo. Para el problema a resolver, y con el objetivo de obtener un valor más preciso de la integral de contorno, se utilizaron 50 puntos de η. Nuevamente, utilizando la función "linspace", se generaron 50 números espaciados linealmente dentro del rango del vector de nudos Ξ.

Luego, utilizando el programa "DeriveNURBS_point.m", se obtuvieron las coordenadas y las derivadas correspondientes al parámetro η . Las derivadas servirán para calcular los respectivos valores del Jacobiano $\frac{d\Gamma}{d\eta}$. Luego, aplicando la regla de Simpson 1/3 compuesta se integraron los valores del Jacobiano, obteniendo el perímetro del cuerpo bajo análisis I; este resultado se verifica fácilmente, puesto que se trata de un círculo. El siguiente paso en el algoritmo general es multiplicar los *kernels* por el valor de esta integral.

Posteriormente, se colocan las tracciones conocidas en el vector de tracciones t, y se multiplican por las normales n_x y $-n_y$ correspondientes al nodo donde se aplica la tracción, siguiendo parte del procedimiento establecido por [53]. Es importante mencionar que los signos de estas normales impactan en los resultados obtenidos, de hecho, el *BEM* es muy sensible al cálculo de estos elementos.

Luego, se borran las columnas y renglones correspondientes a los desplazamientos prescritos, y se resuelve para encontrar los desplazamientos desconocidos u:

$$u = (U/T) * t \tag{3.20}$$

Después, se leen las coordenadas de los nodos internos para calcular los desplazamientos internos. Luego, se toma cada nodo interno como el nodo de carga P_i, y se toman las coordenadas

de los puntos de análisis originales como nodos de campo Q_j. Así, se procede a calcular la distancia r entre estos puntos, las derivadas $\frac{dr}{dx}$ y $\frac{dr}{dy}$, los Jacobianos $\frac{d\Gamma}{d\xi}$, las normales asociadas, n_x y n_y y las derivadas $\frac{dr}{dn}$-. Posteriormente, se calculan los *kernels* asociados a estos puntos de carga y campo. En este caso, no habrá *kernels* indeterminados. Posteriormente, se leen las tracciones prescritas en los puntos de campo t_Q. Entonces, para calcular los desplazamientos internos se usan las ecuaciones (3.21) y (3.22), que se originan de la ecuación (2.22), como se describe en [53]. Las variables u_x y u_y representan los desplazamientos de los nodos de campo correspondientes.

$$u_{x_{int}}(i) = u_{x_{int}}(i) + [-u_x(j) \cdot T_{xx} - u_y(j) \cdot T_{xy} + t_x(i) \cdot U_{xx} + t_y(i) \cdot U_{xy} \quad (3.21)$$

$$u_{y_{int}}(i) = u_{y_{int}}(i) + [-u_x(j) \cdot T_{yx} - u_y(j) \cdot T_{yy} + t_x(i) \cdot U_{yx} + t_y(i) \cdot U_{yy} \quad (3.22)$$

Finalmente, se finaliza el programa al graficar la curva original y la curva deformada. La curva deformada se obtiene al sumar los desplazamientos obtenidos por (3.20) a las coordenadas de los puntos de análisis originales. Los resultados se muestran con gráficas de contorno que se hicieron con el software "*Origin*".

3.4.2 Implementación del algoritmo de contacto

La Figura 3-9 muestra el algoritmo de implementación del método de contacto. Como se ve, el primer paso es agregar puntos de control cercanos a la zona de contacto (Figura 3-4), sin alterar la forma circular. Entonces, tomando las ventajas de la curva NURBS y como referencia la sección 4.5 de [52], se determinó lo siguiente:

- Las secciones cónicas se determinan con tres puntos de control, los puntos de los extremos quedan sobre la curva y el punto medio controla la abertura de la curva (esto también se puede lograr moviendo los pesos, como se especificó en la Tabla 1).
- Por ejemplo, si conocemos el punto sobre la curva γ_1 que está alineado con el punto de control central α_2 (Figura 3-10), se puede determinar este punto de control.
- Los pesos asociados a estos nuevos puntos de control se obtienen calculando el coseno del ángulo que hay entre los puntos del extremo de esta sección cónica.
- En el ejemplo que se quiere resolver, se conocen los puntos sobre la curva γ_i, y los puntos de control $\alpha_1, \alpha_3, \alpha_5, \alpha_7$ y α_9 que están sobre la curva (Figura 3-10).

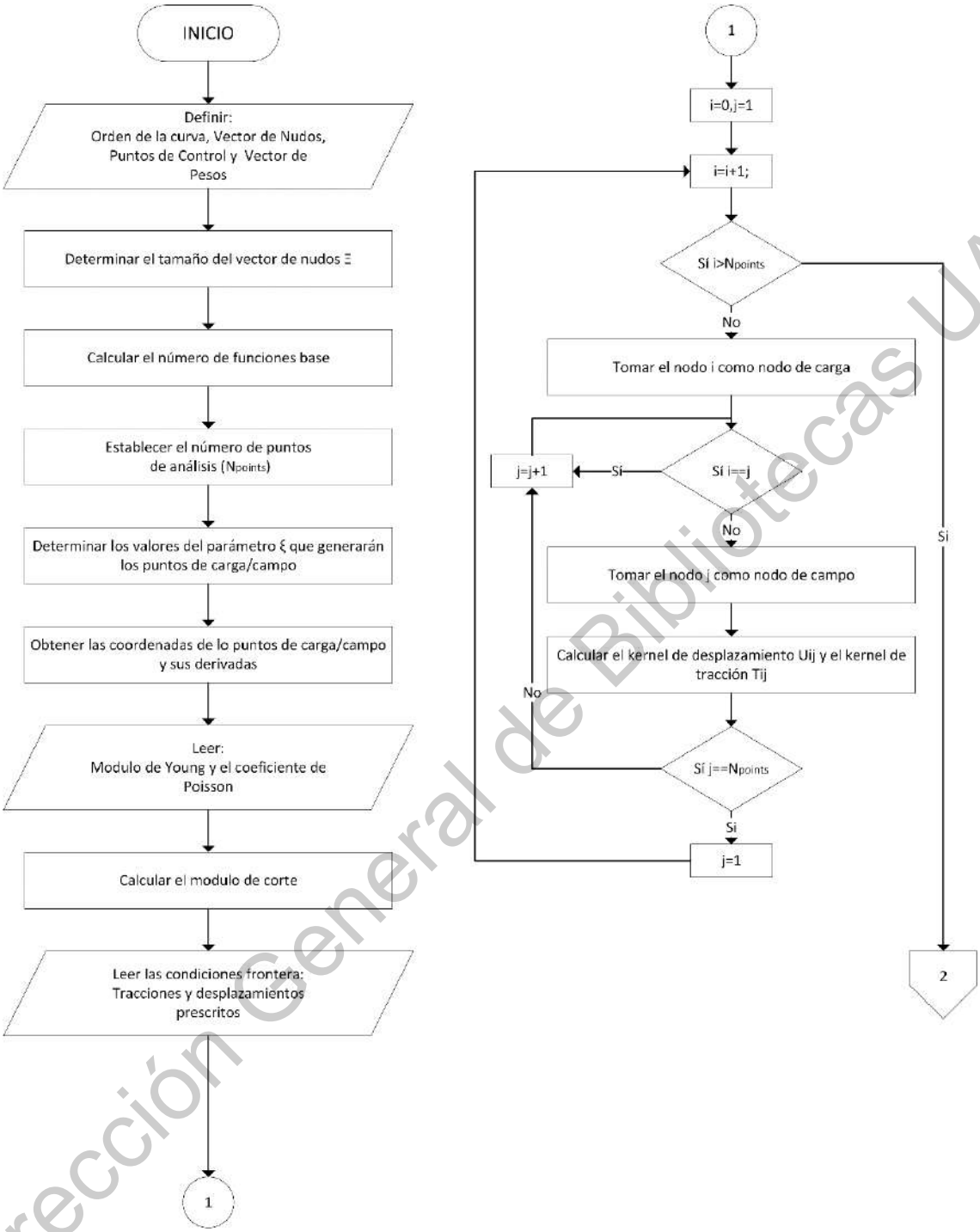


Figura 3-7 Implementación del algoritmo IGA-BEM propuesto (a)

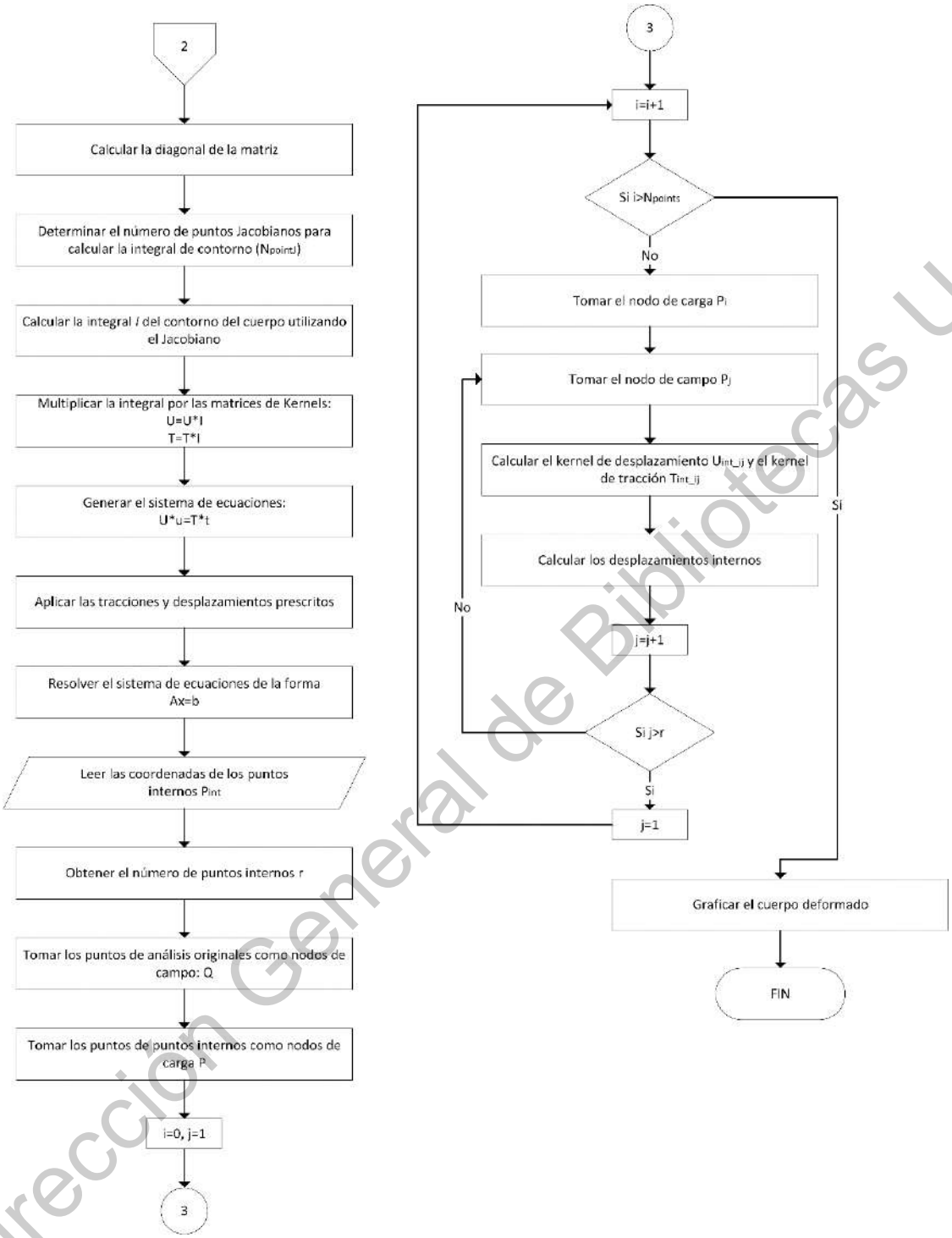


Figura 3-8 Implementación del algoritmo IGA-BEM propuesto (b)

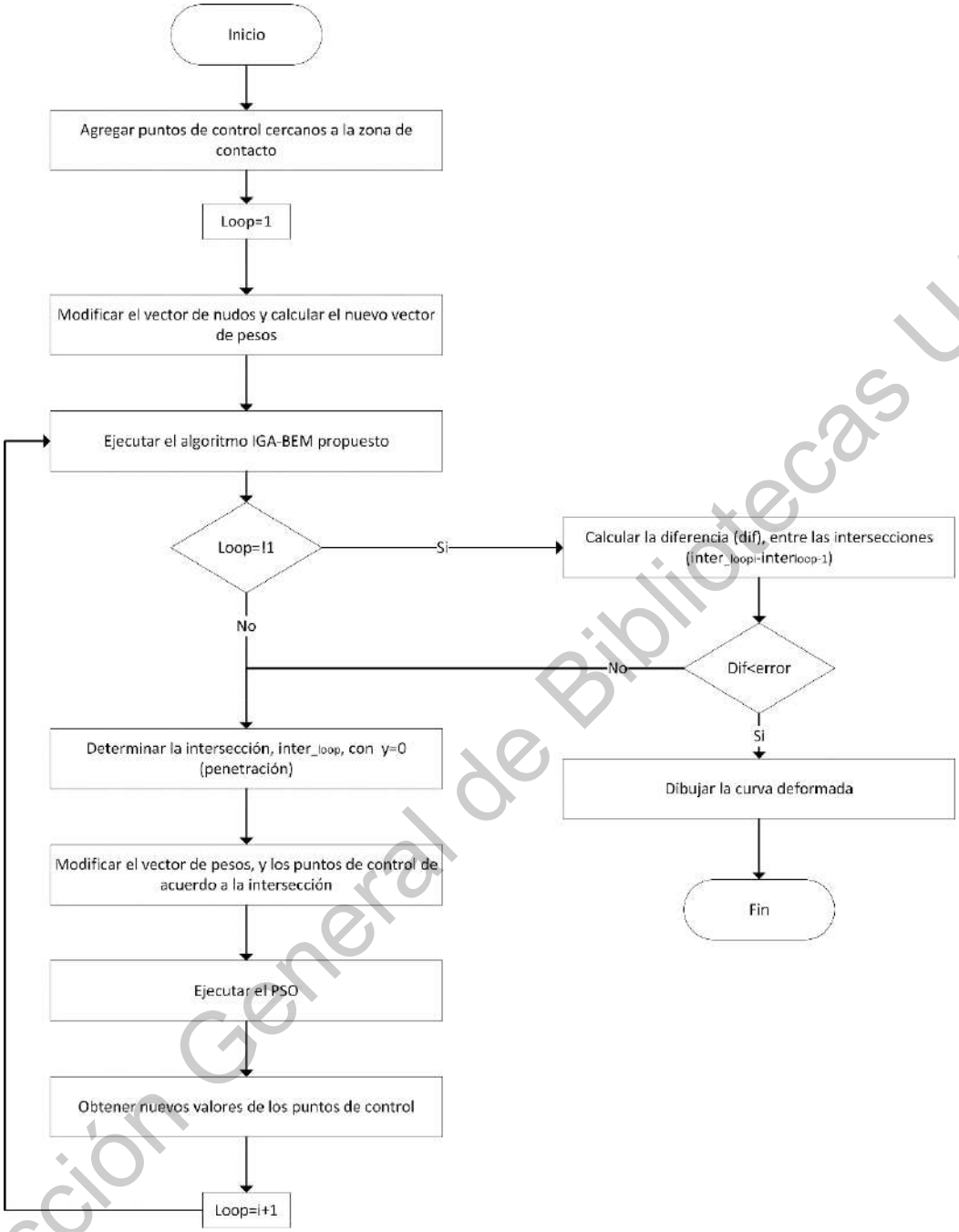


Figura 3-9 Implementación del algoritmo de contacto

En la Figura 3-10 no se distinguen todos los puntos γ_i, sin embargo, los que faltan (γ_2 y γ_3) son muy cercanos a α_4 y α_6. Los puntos de control que no están numerados en la figura, no se modificaron, ya que los puntos agregados solo afectan a la mitad inferior del círculo. Los puntos de control conocidos se describen en la Tabla 4 y los puntos γ_i en la Tabla 5.

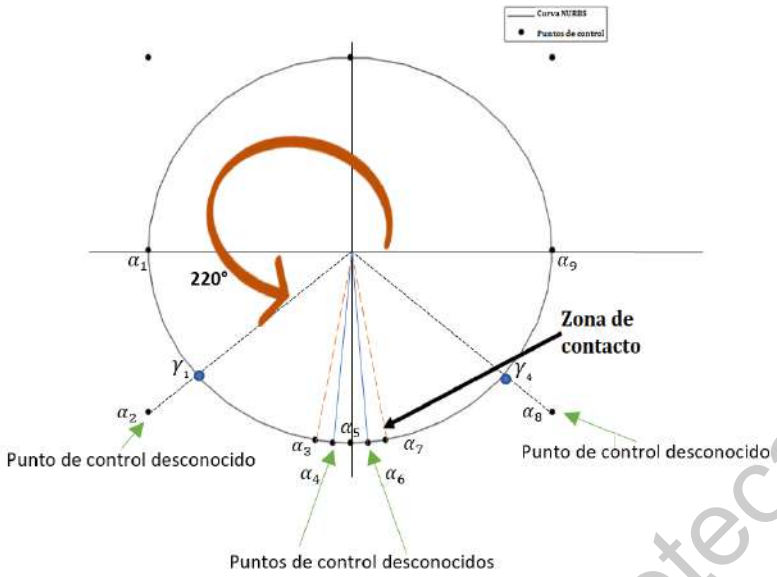


Figura 3-10 Puntos de control agregados

Tabla 4 Coordenadas puntos de control agregados

No. de nodo en Figura 3-10	Coordenada x	Coordenada y
α_1	-3	3
α_3	$3 \cdot cos(260°)$	$3 \cdot sin(260°) + 3$
α_5	0	0
α_7	$3 \cdot cos(280°)$	$3 \cdot sin(280°) + 3$
α_9	3	3

Tabla 5 Coordenadas de los puntos γ_i

γ_i	Coordenada x	Coordenada y
γ_1	$3 \cdot cos(220°)$	$3 \cdot cos(220°)+3$
γ_2	$3 \cdot cos(265°)$	$3 \cdot sin(265°) + 3$
γ_3	$3 \cdot cos(275°)$	$3 \cdot cos(275°) + 3$
γ_4	$3 \cdot cos(320°)$	$3 \cdot cos(320°) + 3$

Como se mencionó, los pesos de los puntos de control desconocidos se obtienen al calcular el coseno de los ángulos entre los puntos de control sobre la curva:

Tabla 6 Pesos de los puntos de control agregados

Peso	
w_{α_2}	$cos(40°)$
w_{α_4}	$cos(5°)$
w_{α_6}	$cos(5°)$
w_{α_8}	$cos(40°)$

Finalmente, para obtener los puntos de control desconocidos se utilizan las ecuaciones (3.23)-(3.25). Donde s es un parámetro que controla curvatura y β es el punto medio entre los puntos de control α_{i-1} y α_{i+1}, como se observa en la Figura 3-11. Los vectores con los nuevos puntos de control, nudos y pesos que generan al círculo de la Figura 3-10, se especifican en la Tabla 12. El programa que realiza los cálculos de los puntos de control se llama “NURBSParameters.m”.

$$\alpha_i = [\gamma_i - (1 - s_i) \cdot \beta_i]/s_i \quad (3.23)$$

$$s_i = w_{\alpha_i}/(w_{\alpha_i} + 1) \quad (3.24)$$

$$\beta_i = (\alpha_{i-1} + \alpha_{i+1})/2 \quad (3.25)$$

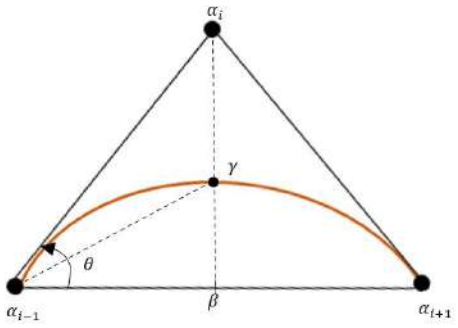


Figura 3-11 Arco generado con una NURBS

Siguiendo el diagrama de flujo de la Figura 3-9, el siguiente paso es ejecutar el programa del *IGA-BEM* propuesto ("principal2.m"). Es importante mencionar, que, aunque se esté resolviendo el contacto entre dos cilindros, al asumir que las deformaciones son simétricas en un problema de deformación plana, no es necesario resolver para ambos cuerpos. Entonces, se resuelve para un cuerpo, y si su deformación sobrepasa el límite de (0,0), entonces habrá penetración entre ambos cuerpos. El paso consecutivo es encontrar la intersección entre estos cuerpos. Para ello, se genera la curva deformada con más puntos ($N_{pointsDraw}$), luego, se encuentra el primer punto a la izquierda con $y < 0$ y el primer punto a la derecha con $y < 0$. Estos puntos generaran la línea de intersección, Figura 3-12.

Posteriormente, se modifica el vector de pesos y los puntos de control de acuerdo a las coordenadas de los puntos de intersección. Estas coordenadas sustituirán los valores de los puntos de control α_3 y α_7. A los puntos α_4 y α_6 pueden asignárseles cualquier valor, ya que sus respectivos pesos w_{α_4} y w_{α_6} se harán cero para generar una zona de contacto con forma de línea recta (Figura 4-10). Esta asignación de pesos se realiza en el programa "NewValues.m".

El siguiente paso en el método propuesto es variar algunos puntos de control con el algoritmo de optimización de cumulo de partículas. En la Figura 3-13 se muestra el algoritmo de implementación de este método, cuyo código asociado es "PSO.m". El primer movimiento en este programa es determinar el número de variables de optimización, es decir, el número de variables que el algoritmo buscará optimizar. Como se ha descrito anteriormente, estas corresponden a las coordenadas de algunos puntos de control: $\alpha_4, \alpha_5, \alpha_6$ y α_7. Los demás puntos de control α se dejaron fijos para no generar formas abstractas.

Posteriormente, se establecieron los rangos de las variables a optimizar y se definió el tamaño de la población, $N_{swarm} = 25$, el número de ciclos que se ejecutará el algoritmo (Gen), el número de intervalos y los valores de las constantes c_1 y C_{max}. Los valores de estas constantes se tomaron de

[88]. Después, se generaron valores aleatorios iniciales de las posiciones de las partículas y de sus velocidades, mediante los programas “InitialPopulation.m” e “InitialVelocity.m”. Los valores de posición aleatorios que se generaron deben estar contenidos en el rango de valores de la Tabla 7. Luego, se utilizó la función “InitialEvaluatingS_UptadatingP_g.m” para obtener los mejores valores locales L_i y el mejor valor global G.

La siguiente parte del programa consiste en actualizar los valores de las posiciones (considerando los valores de la Tabla 7) y las velocidades utilizando el programa “UpdateVandX”, el cual se basó en las ecuaciones (2.39) y (2.40). Luego, se evalúan estos valores actualizados sustituyéndolos en la función objetivo (3.13). Si el resultado de la evaluación es diferente de cero, entonces se penalizará a la función objetivo con las ecuaciones (3.14) y (3.15). Por consiguiente, se determinará si estos valores penalizados son mejores que los mejores valores locales L_i o que el mejor valor global G; si se cumplen estas condiciones, entonces se actualizarán los mejores valores locales y el mejor valor global, y se guardarán los valores asociados de las variables de diseño. Esta parte del programa PSO se estará ejecutando hasta que se cumpla Gen número de ciclos.

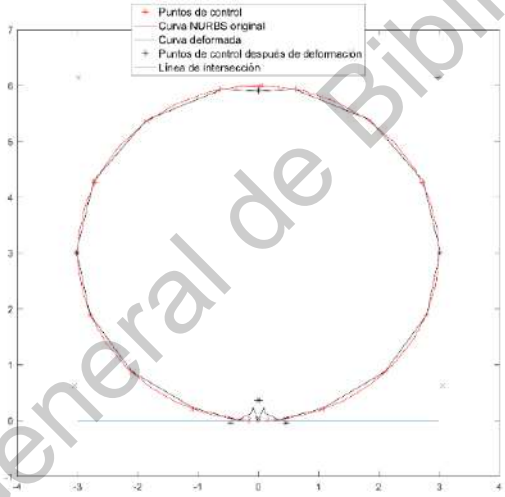


Figura 3-12 Línea de intersección

Tabla 7 Rango de las variables a optimizar

Rango	α_{4_x}	α_{4_y}	α_{5_x}	α_{5_y}	α_{6_x}	α_{6_y}	α_{7_y}
ub (límite superior)	4	3	4	3	4	6	6
lb (límite inferior)	0	0	3	2	3	3	3

Para verificar que el algoritmo de optimización funciona, se graficaron los valores globales y se revisó la convergencia del programa y la dispersión de las partículas, Figura 3-14 y Figura 3-15, comparando el primer ciclo de optimización vs. el último. En algunas ocasiones, la curva optimizada resultará en extremo convexa, de tal manera que el algoritmo de optimización se deberá ejecutar más de una vez. Cuando la forma generada sea factible, se ejecutará nuevamente el *IGA-BEM* contacto propuesto con los puntos de control optimizados (“Principal2.m”). Si no hay diferencia significativa entre las longitudes de las intersecciones, entonces el algoritmo general de contacto

finalizará. Si no, se ejecutará nuevamente para encontrar nuevos puntos de control y se repetirá el proceso de validación. Como trabajo futuro, se agregarán restricciones de forma para automatizar el proceso de optimización.

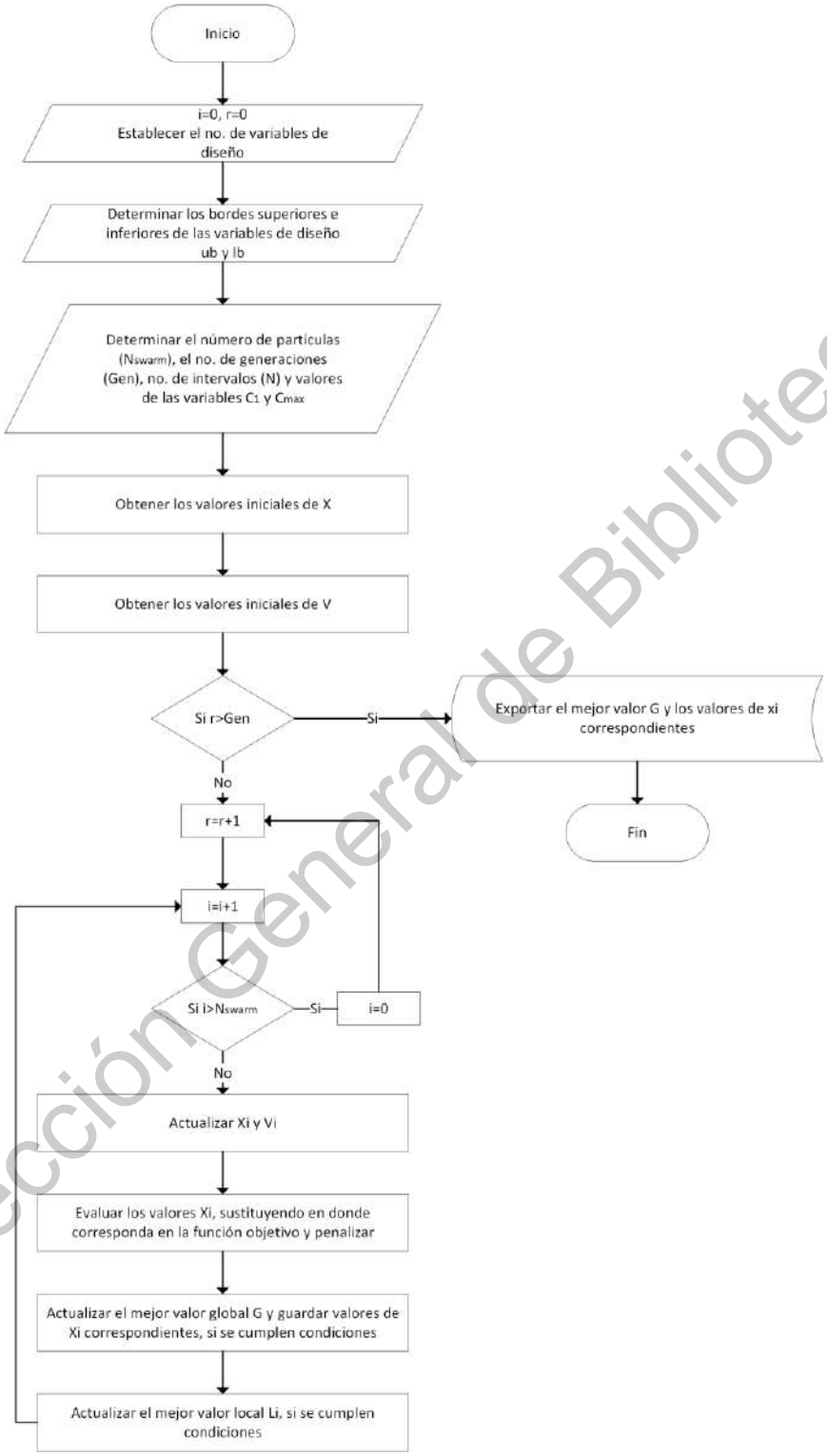


Figura 3-13 Implementación del PSO

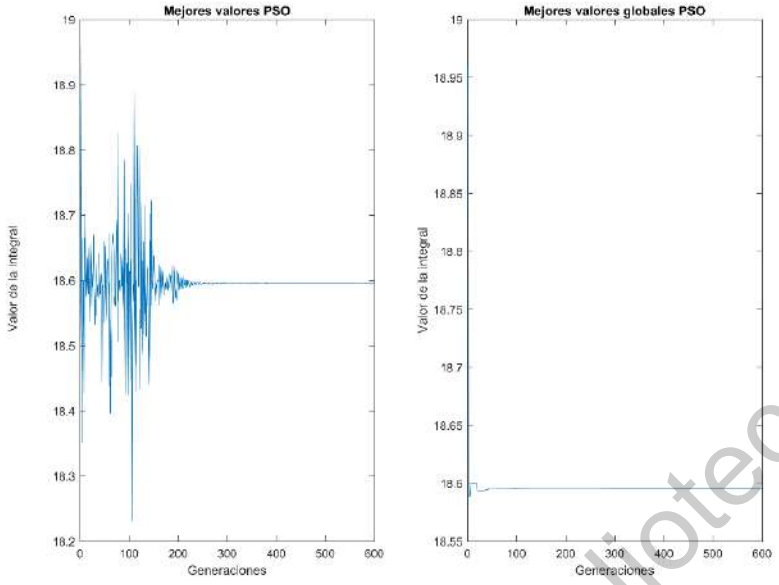


Figura 3-14 Convergencia PSO

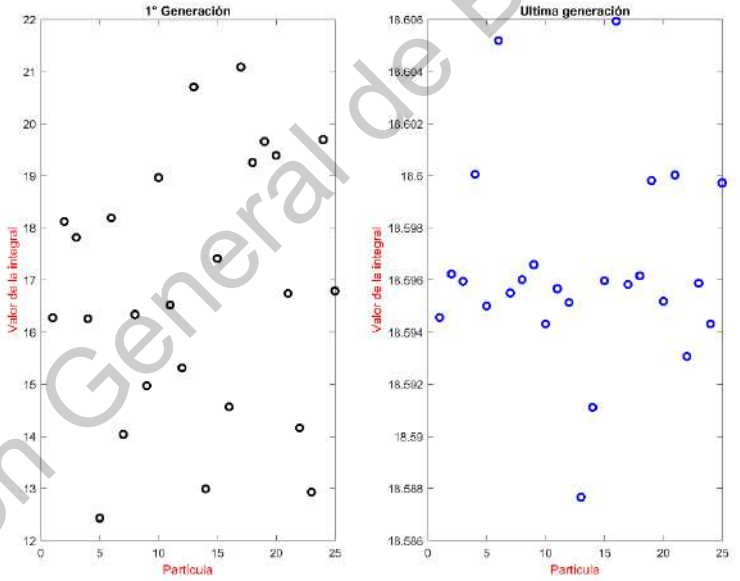


Figura 3-15 Dispersión de las partículas, 1° generación vs última

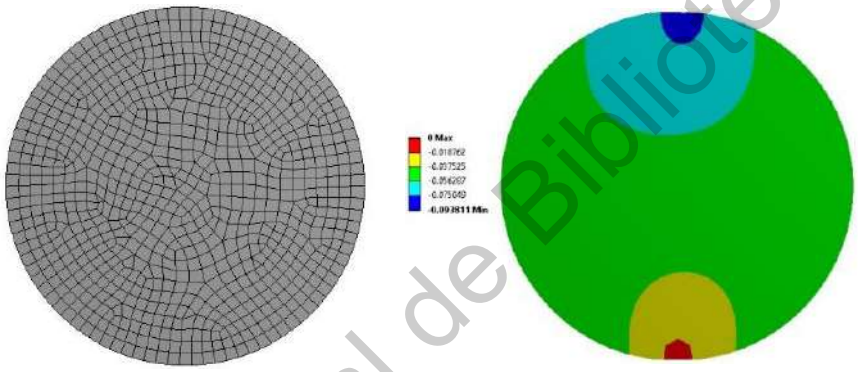
CAPÍTULO IV

4. RESULTADOS Y DISCUSIÓN

4.1 Problema del cilindro bajo una carga τ y un punto fijo

4.1.1 Resultados utilizando un software comercial basado en MEF

Se utilizó el software ANSYS para resolver el problema presentado en la Figura 3-5. Se aplicó una carga prescrita en la parte superior del círculo, y se distribuyó entre dos nodos vecinos. Luego, se agregaron restricciones de desplazamientos en los nodos del lado opuesto. Se eligió una malla fina, Figura 4-1 (a), que generó 2566 nodos y 827 elementos. El resultado del problema se muestra en la Figura 4-1 (b).

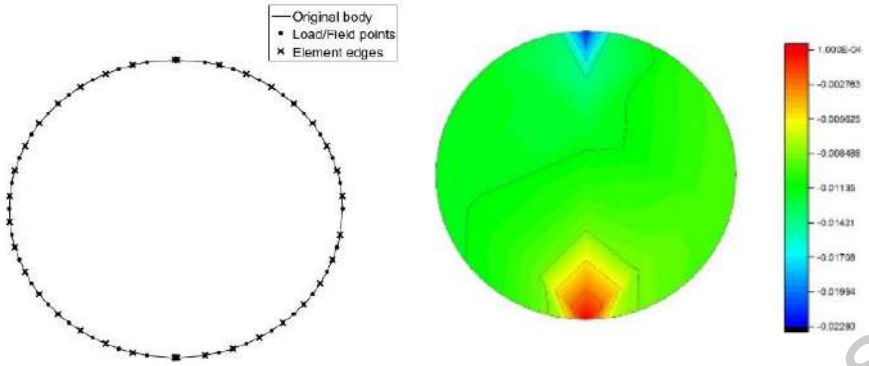


(**a**) (**b**)

Figura 4-1 Cilindro bajo carga τ analizado con MEF. (a)Malla fina generada en el software (b) Cilindro deformado, los colores representan los desplazamientos en la dirección Y.

4.1.2 Resultados empleando el método convencional de elementos frontera

El método convencional BEM, que se usó en este apartado, está basado en el programa propuesto por [53]. Se eligieron 71 nodos y 36 elementos distribuidos en el contorno del cuerpo, Figura 4-2 (a). También, se utilizaron 6 puntos de integración para la Cuadratura Gaussiana y se optó por funciones de forma cuadráticas para describir las tracciones y desplazamientos. Además, se tomaron las propiedades del material descrito en la Tabla 8. El resultado de esta implementación se muestra en la Figura 4-2(b). Se agregaron 13 puntos internos para generar los desplazamientos internos.



(**a**) (**b**)

Figura 4-2 Cilindro bajo una carga τ, analizado con el BEM convencional. (a) Puntos de análisis usados como puntos de carga y límites de los elementos. (b) Cilindro deformado, los colores representan los desplazamientos en la dirección Y.

4.1.3 Resultados usando el método propuesto de elementos frontera isogeométrico

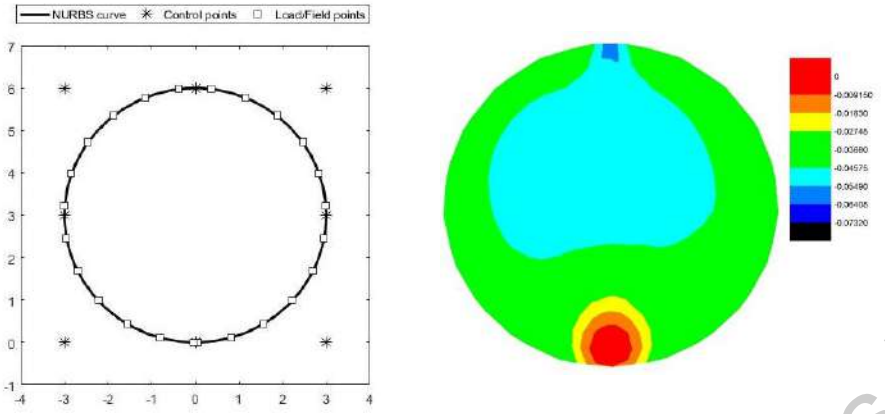
Para generar la sección transversal del cilindro (círculo), se establecieron los puntos de control, el vector de nudos, el vector de pesos y el orden de la curva. Estos valores se detallan en la Tabla 8. Como se describió en la sección 3.1, los puntos de carga/campo se generaron en el contorno del cuerpo, Figura 4-3 (a), se eligieron 24 valores del parámetro ξ para obtener dichos puntos mediante la ecuación (2.1). La Tabla 9 especifica elementos adicionales para ejecutar el programa del método propuesto. El resultado obtenido con este método se muestra en la Figura 4-3 (b). Se añadieron puntos de carga internos para generar adecuadamente la gráfica de contorno.

Tabla 8 Datos para generar la curva NURBS del problema de deformación plana

Vector de nudos	$\Xi = [0,0,0,1,1,2,2,3,3,4,4,4]$
Puntos de control	$B = [0\ \ 0; 3\ \ 0; 3\ \ 3; 3\ \ 6; 0\ \ 6; -3\ \ 6; -3\ \ 3; -3\ \ 0; 0\ \ 0]$
Vector de pesos	$W = [1,\ \frac{\sqrt{2}}{2},\ 1,\ \frac{\sqrt{2}}{2},\ 1,\ \frac{\sqrt{2}}{2},\ 1,\ \frac{\sqrt{2}}{2},\ 1\ ]$
Orden de la curva	$k = 3$

Tabla 9 Datos para analizar el problema de deformación plana con el método propuesto

Número de puntos de campo/carga	24
Número de elementos	1
Número de puntos de control (para la geometría NURBS)	9
Número de puntos Jacobianos (para calcular el perímetro)	50
Número de puntos de carga internos	13



(**a**) (**b**)

Figura 4-3 Cilindro bajo una carga τ, analizado con el método propuesto IGA-BEM. (a) Puntos de carga/campo usados como puntos de análisis, y puntos de control para generar la curva NURBS. (b) Cilindro deformado, los colores representan los desplazamientos en la dirección Y.

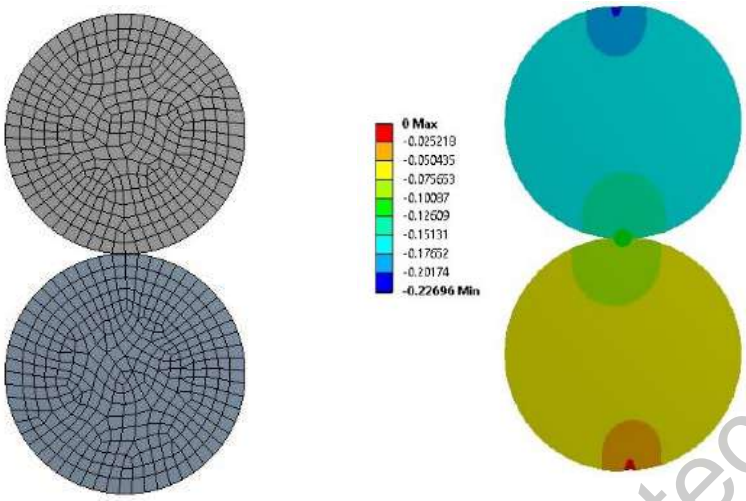
4.2 Problema de deformación plana de dos elementos en contacto

4.2.1 Problema de contacto resuelto con MEF

Este problema también se resolvió con el software de ANSYS. Los parámetros usados en esta simulación se describen en Tabla 10 . Se eligió una malla fina para obtener resultados más precisos, Figura 4-4(a). Para controlar la penetración entre los cuerpos se utilizó el método de maestro-esclavo, que se explicó brevemente en la introducción, en conjunto con el método de Lagrange aumentado. La Figura 4-4(b) muestra los cuerpos deformados, el mapa de colores representa los desplazamientos en Y. La máxima deformación se localiza en el área de aplicación de la carga. La zona de contacto mide aproximadamente 0.65 mm (Figura 4-5).

Tabla 10 Parámetros para resolver el problema con MEF

Orden del elemento	Cuadrático
Tipo de contacto	Sin fricción
Comportamiento	Simétrico
Método para resolver el contacto	Lagrange aumentado
Método de detección	Nodal-normal al objetivo
Número de nodos	2184
Número de elementos	696



(**a**) (**b**)

Figura 4-4 Contacto entre dos cuerpos analizados con software basado en FEM. (a) Malla usada en el software (b) Desplazamiento en Y de los cuerpos en contacto después de aplicar una carga τ

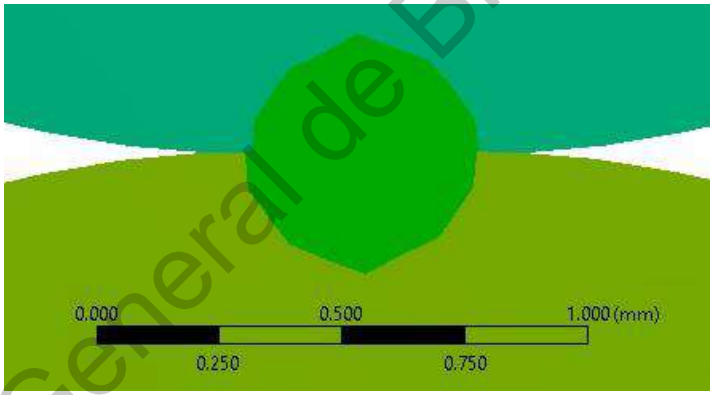


Figura 4-5 Acercamiento al área de contacto entre dos cuerpos analizado con FEM

4.2.2 Problema de contacto resuelto con el modelo de Hertz

De acuerdo a [90], el ancho del área rectangular de contacto entre dos cilindros se determina utilizando la ecuación (4.1), si estos elementos tienen el mismo módulo de elasticidad ($E = E_1 = E_2$), y el mismo coeficiente de Poisson ($v = 0.3$). En esta ecuación, τ_L representa la carga por unidad de longitud (τ/L_{ON}), K_D es la curvatura relativa ($^{D_1 \cdot D_2}/_{D_1 + D_2}$), y $a_{contacto}$ simboliza al ancho del área de contacto rectangular. Información más detallada sobre el contacto de Hertz puede encontrarse en [35]. La longitud del cilindro es de $100\,mm$.

$$a_{contacto} = 2.15\sqrt{\frac{\tau_L K_D}{E}} \tag{4.1}$$

Si se sustituyen los valores de la Tabla 2 y los descritos en el párrafo anterior, el ancho del área de contacto es $a_{contacto} = .056\, mm$. Siguiendo la teoría de Hertz, se pueden obtener datos adicionales como la presión de contacto, el esfuerzo cortante máximo y la profundidad donde ocurre. Estos valores se describen en la Tabla 11.

Tabla 11 Resultados del problema de contacto obtenidos con la teoría de Hertz

Ancho del área de contacto	$0.056\, mm$
Presión de contacto máxima Hertziana	$1024.4\, MPa$
Esfuerzo cortante máximo	$308\, MPa$
Profundidad del esfuerzo cortante máximo	$0.022\, mm$

4.2.3 Problema de contacto resuelto con el algoritmo de contacto propuesto

Para utilizar el algoritmo de la sección 3.2, se agregaron algunos puntos de control cerca del punto en común (0,0). Los datos de las NURBS para crear el área transversal de los cilindros en contacto se especifican en la Tabla 12. Como se observa, los pesos se cambiaron para mantener la forma circular de los cilindros. La Tabla 13 especifica los datos para resolver el problema con el *IGA-BEM* propuesto. Los puntos de carga se generaron con 24 valores de ξ distribuidos entre el valor máximo y mínimo del vector de nudos. A diferencia de los parámetros descritos en la sección 4.1.3, en este algoritmo se incrementó el número de puntos de control de 9 a 13 (Figura 4-6). Finalmente, el número de puntos η para calcular el perímetro se dejó en 50. Sin embargo, este valor se calcula una sola vez. La Figura 4-7 muestra a los cilindros deformados después de aplicar la carga τ, y antes de utilizar el algoritmo de optimización. La Figura 4-8 es un acercamiento al área de contacto entre los dos cuerpos, como se observa hay penetración. En la Tabla 13 se especifican los parámetros de entrada del algoritmo de optimización; los valores de $C1$ y $C2$ se establecieron de acuerdo a las recomendaciones hechas por [88]. La Figura 4-9 muestra los cilindros en contacto después de haber aplicado el algoritmo de optimización para encontrar sus deformaciones. En la Figura 4-10 se ve un acercamiento al área de contacto optimizada; como se ve, la zona de contacto es una línea recta y no hay penetración entre los cuerpos. La longitud de esta zona es de aproximadamente 62 mm.

Tabla 12 Datos para crear las curvas del problema de contacto

Vector de nudos	$\Xi = [0,0,0,1,1,2,2,3,3,4,4,5,5,6,6,6]$
Puntos de control, cilindro superior	$B = [0\ \ 0; 0.2625\ \ 0; 0.5209\ \ 0.0456; 3\ \ 0.04827; 3\ \ 3; 3\ \ 6; 0\ \ 6; -3\ \ 6; -3\ \ \ 3;$ $-3\ 0.4827; -0.5209\ \ \ 0.0456;\ -0.2625\ \ 0; 0\ \ 0]$
Puntos de control, cilindro interior	$B = [0\ \ 0; 0.2625\ \ 0; 0.5209\ \ -0.0456; 3\ \ -0.04827; 3\ \ -3;\ \ 3\ \ -6; 0\ \ -6;$ $-3\ \ -6; -3\ \ \ -3; -3\ \ -0.4827; -0.5209\ \ -0.0456;\ -0.2625\ \ 0; 0\ \ 0]$
Vector de pesos	$W = [1, 0.9962, 1, 0.7660, 1, 0.7071, 1, 0.7071, 1, 0.7660, 1, 0.9962, 1]$
Orden de la curva	$k = 3$

Tabla 13 Datos para analizar el problema de contacto

Números de puntos de carga/campo	24
Número de elementos	1
Número de puntos de control (para la *NURBS*)	13
Número de puntos Jacobianos	50

Tabla 14 Datos para ejecutar el algoritmo de PSO

Número de generaciones	300
Tamaño de la población (N_{swarm})	25
C_1	0.8
C_{max}	1.62

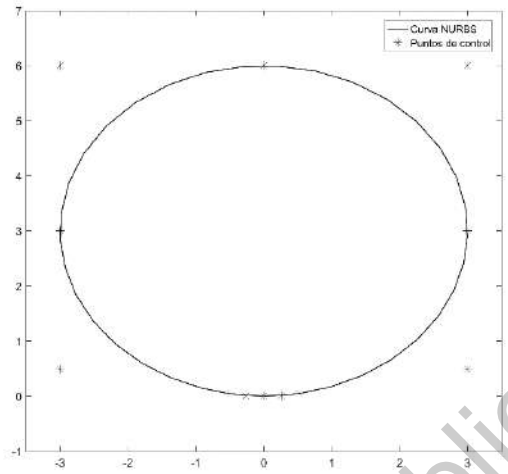


Figura 4-6 Puntos de control agregados a cada círculo cercanos al área de contacto

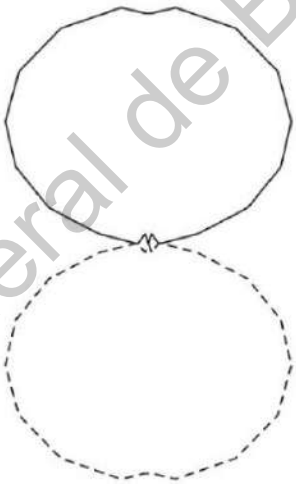


Figura 4-7 Contacto entre dos cuerpos después de aplicar una carga con el IGA-BEM propuesto, antes de aplicar el algoritmo de contacto

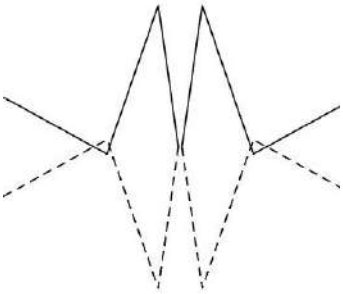


Figura 4-8 Acercamiento al área de contacto entre dos cuerpos analizados con el IGA-BEM propuesto antes de optimizar

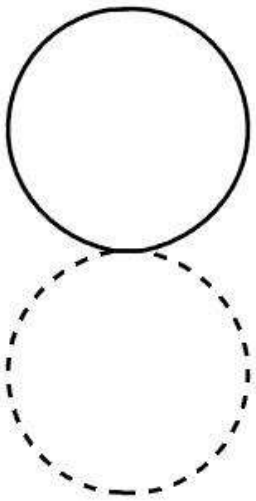


Figura 4-9 Contacto entre dos cuerpos analizados con el método de contacto propuesto.

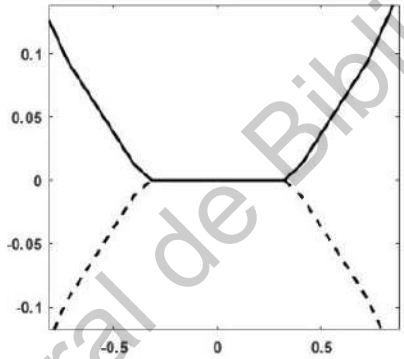


Figura 4-10 Acercamiento al área de contacto entre dos cuerpos analizados con el método de contacto propuesto.

4.3 Discusión: Análisis de los resultados obtenidos

4.3.1 Deformación plana de un cilindro

Como se observó en la sección 4.1, los tres métodos generaron cuerpos deformados similares. Con una malla fina, el máximo desplazamiento que se obtuvo con MEF fue de $-0.094\ mm$, mientras que con el método convencional de elementos frontera fue de $-0.023\ mm$, y con el método propuesto de *IGA-BEM* fue de $-0.073\ mm$. El máximo desplazamiento, en los tres casos, se obtuvo en la zona donde se aplicó la carga P. Para resolver el problema, en el software MEF se utilizaron 2566 nodos, con el método convencional *BEM* se usaron 71 nodos de análisis y con el método propuesto *IGA-BEM* se utilizaron 24 puntos de análisis. Sin embargo, para obtener el comportamiento interno del cuerpo, se agregaron puntos de campo internos con los métodos *IGA-BEM* propuesto y *BEM* convencional. En contraste, dada la naturaleza de MEF, no se requirieron nodos adicionales.

4.3.2 Deformación plana de dos cilindros en contacto

Respecto a este problema, en la Tabla 15 se muestra una comparación entre el ancho de las áreas de contacto que se obtuvieron con el modelo de Hertz, el software MEF y el método de contacto

propuesto. Tomando la solución analítica como referencia, el error obtenido con MEF fue del 16%, mientras que con el propuesto fue del 10%. En ambos casos, la interferencia entre los cuerpos se eliminó después de aplicar la carga. Con MEF el problema se resolvió usando un total de 2184 nodos, equivalente a 4368 grados de libertad (GDL). Con el método de contacto propuesto, el problema se resolvió con 24 puntos de análisis (48 GDL), pero el número de puntos de control que definieron la forma del círculo se incrementó de 9 a 13 puntos.

Tabla 15 Comparación de los anchos de las áreas de contacto

Método	Ancho del área de contacto (mm)	Variación con respecto al método analítico	GDL para resolver el problema
Analítico (Modelo Hertz)	0.56	0	-
MEF	0.65	16%	4368
Método de contacto propuesto	0.62	11%	48

CAPÍTULO V

5. Conclusiones y perspectivas

Como se describió en el capítulo 1, los métodos que se usan para analizar el contacto entre elementos mecánicos conllevan a errores en los resultados al aproximar el área de contacto o al propagar del error, como se menciona en [91]. Esta propagación se relaciona con el tamaño de la matriz: entre más elementos tenga la matriz, más grande será el error. Para evitar estos errores, se propuso utilizar análisis isogeométrico, puesto que el enmallado se hace en función de curvas *NURBS,* que generan líneas cónicas y rectas con más exactitud comparadas a las generadas con polinomios de Lagrange que comúnmente son usados en los métodos numéricos de mecánica del medio continúo.

Después de la revisión del estado del arte, se determinó que la combinación de análisis isogeométrico y elementos frontera (*IGA-BEM*) llevaría a un análisis más exacto, al utilizar menos grados de libertad en la resolución de problemas de contacto entre elementos mecánicos. De esta manera se replanteó la ecuación integral de la frontera para combinarla con *IGA*, y luego para analizar el contacto entre dos cuerpos.

El método propuesto *IGA-BEM* obtuvo resultados consistentes al resolver el problema de deformación plana de un cilindro bajo carga comparándolos con los métodos probados de MEF y *BEM*. Aunque los valores de los desplazamientos que se obtuvieron por los tres métodos fueron diferentes, la deformación generada fue similar. Sin embargo, se requirieron menos grados de libertad en el *IGA-BEM* propuesto, debido a que se utilizó una sola *NURBS* para generar el contorno de cuerpo completo. Además, los puntos de carga/campo se separaron de los puntos que representan la geometría, a diferencia del *BEM* convencional. Asimismo, al utilizar un macroelemento se garantizó la continuidad entre los polinomios que definen al cuerpo bajo análisis.

Respecto al problema de contacto no lineal entre elementos mecánicos, se compararon los anchos de las zonas de contacto obtenidos por el método de elementos finitos y por el método de contacto propuesto. Estos variaron 16% y 10%, respectivamente, comparados con el obtenido con el modelo analítico de Hertz. Sin embargo, los GDL usados con MEF para resolver el problema fueron 4368, mientras que con el método propuesto fueron solo 48 GDL.

Por tanto, el análisis de los cilindros en contacto mediante el método de contacto propuesto se hizo con menor número de grados de libertad comparado con MEF y se consiguió un resultado aceptable al compararlo con el modelo analítico de Hertz. Disminuir los grados de libertad es muy importante cuando se tienen problemas multifísicos, donde se requiere invertir matrices.

Los resultados son prometedores, sin embargo, se requiere trabajo futuro para probar que la metodología es válida en otras situaciones donde los desplazamientos no son simétricos o factores adicionales, como la fricción, sean considerados. Asimismo, se podrían utilizar otras curvas paramétricas para representar la geometría, como las *T-Splines*.

Referencias

[1] J. A. Cottrell, T. J. R. Hughes, y Y. Bazilevs, *Isogeometric analysis: toward integration of CAD and FEA*, 1a ed. John Wiley & Sons, 2009.

[2] T. J. R. Hughes, J. A. Cottrell, y Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement", *Computer Methods in Applied Mechanics and Engineering*, vol. 194, núm. 39–41, pp. 4135–4195, 2005, doi: 10.1016/j.cma.2004.10.008.

[3] I. Temizer, P. Wriggers, y T. J. R. Hughes, "Contact treatment in isogeometric analysis with NURBS", *Computer Methods in Applied Mechanics and Engineering*, vol. 200, núm. 9–12, pp. 1100–1112, feb. 2011, doi: 10.1016/j.cma.2010.11.020.

[4] L. De Lorenzis, P. Wriggers, y G. Zavarise, "A mortar formulation for 3D large deformation contact using NURBS-based isogeometric analysis and the augmented Lagrangian method", *Comput Mech*, vol. 49, núm. 1, pp. 1–20, ene. 2012, doi: 10.1007/s00466-011-0623-4.

[5] M. E. Matzen, T. Cichosz, y M. Bischoff, "A point to segment contact formulation for isogeometric, NURBS based finite elements", *Computer Methods in Applied Mechanics and Engineering*, vol. 255, pp. 27–39, mar. 2013, doi: 10.1016/j.cma.2012.11.011.

[6] V. Agrawal y S. S. Gautam, "Varying-order NURBS discretization: An accurate and efficient method for isogeometric analysis of large deformation contact problems", *Computer Methods in Applied Mechanics and Engineering*, vol. 367, p. 113125, ago. 2020, doi: 10.1016/j.cma.2020.113125.

[7] Y. Bazilevs y T. J. R. Hughes, "NURBS-based isogeometric analysis for the computation of flows about rotating components", *Comput Mech*, vol. 43, núm. 1, Art. núm. 1, dic. 2008, doi: 10.1007/s00466-008-0277-z.

[8] E. Brivadis, A. Buffa, B. Wohlmuth, y L. Wunderlich, "Isogeometric mortar methods", *Computer Methods in Applied Mechanics and Engineering*, vol. 284, pp. 292–319, feb. 2015, doi: 10.1016/j.cma.2014.09.012.

[9] L. De Lorenzis, P. Wriggers, y T. J. R. Hughes, "Isogeometric contact: a review: Isogeometric contact: a review", *GAMM-Mitteilungen*, vol. 37, núm. 1, Art. núm. 1, jul. 2014, doi: 10.1002/gamm.201410005.

[10] L. De Lorenzis, J. A. Evans, T. J. R. Hughes, y A. Reali, "Isogeometric collocation: Neumann boundary conditions and contact", *Computer Methods in Applied Mechanics and Engineering*, vol. 284, pp. 21–54, feb. 2015, doi: 10.1016/j.cma.2014.06.037.

[11] R. Dimitri, L. De Lorenzis, P. Wriggers, y G. Zavarise, "NURBS- and T-spline-based isogeometric cohesive zone modeling of interface debonding", *Comput Mech*, vol. 54, núm. 2, Art. núm. 2, ago. 2014, doi: 10.1007/s00466-014-0991-7.

[12] R. Dimitri, L. De Lorenzis, M. A. Scott, P. Wriggers, R. L. Taylor, y G. Zavarise, "Isogeometric large deformation frictionless contact using T-splines", *Computer Methods in Applied Mechanics and Engineering*, vol. 269, pp. 394–414, feb. 2014, doi: 10.1016/j.cma.2013.11.002.

[13] M. Dittmann, M. Franke, İ. Temizer, y C. Hesch, "Isogeometric Analysis and thermomechanical Mortar contact problems", *Computer Methods in Applied Mechanics and Engineering*, vol. 274, pp. 192–212, jun. 2014, doi: 10.1016/j.cma.2014.02.012.

[14] Y. Guo, M. Ruess, y Z. Gürdal, "A contact extended isogeometric layerwise approach for the buckling analysis of delaminated composites", *Composite Structures*, vol. 116, pp. 55–66, sep. 2014, doi: 10.1016/j.compstruct.2014.05.006.

[15] J. Vallepuga Espinosa y A. Foces Mediavilla, "Boundary element method applied to three dimensional thermoelastic contact", *Engineering Analysis with Boundary Elements*, vol. 36, núm. 6, Art. núm. 6, jun. 2012, doi: 10.1016/j.enganabound.2011.12.010.

[16] H. Lian, P. Kerfriden, y S. P. A. Bordas, "Shape optimization directly from CAD: An isogeometric boundary element approach using T-splines", *Computer Methods in Applied Mechanics and Engineering*, vol. 317, pp. 1–41, abr. 2017, doi: 10.1016/j.cma.2016.11.012.

[17] L. L. Chen, H. Lian, Z. Liu, H. B. Chen, E. Atroshchenko, y S. P. A. Bordas, "Structural shape optimization of three dimensional acoustic problems with isogeometric boundary element

methods", *Computer Methods in Applied Mechanics and Engineering*, vol. 355, pp. 926–951, oct. 2019, doi: 10.1016/j.cma.2019.06.012.
[18] M. Taus, G. J. Rodin, T. J. R. Hughes, y M. A. Scott, "Isogeometric boundary element methods and patch tests for linear elastic problems: Formulation, numerical integration, and applications", *Computer Methods in Applied Mechanics and Engineering*, vol. 357, p. 112591, dic. 2019, doi: 10.1016/j.cma.2019.112591.
[19] F. L. Sun, C. Y. Dong, y H. S. Yang, "Isogeometric boundary element method for crack propagation based on Bézier extraction of NURBS", *Engineering Analysis with Boundary Elements*, vol. 99, pp. 76–88, feb. 2019, doi: 10.1016/j.enganabound.2018.11.010.
[20] M. E. Yildizdag, I. T. Ardic, A. Kefal, y A. Ergin, "An isogeometric FE-BE method and experimental investigation for the hydroelastic analysis of a horizontal circular cylindrical shell partially filled with fluid", *Thin-Walled Structures*, vol. 151, p. 106755, jun. 2020, doi: 10.1016/j.tws.2020.106755.
[21] E. O. Inci, L. Coox, O. Atak, E. Deckers, y W. Desmet, "Applications of an isogeometric indirect boundary element method and the importance of accurate geometrical representation in acoustic problems", *Engineering Analysis with Boundary Elements*, vol. 110, pp. 124–136, ene. 2020, doi: 10.1016/j.enganabound.2019.10.009.
[22] A. M. Shaaban, C. Anitescu, E. Atroshchenko, y T. Rabczuk, "Isogeometric boundary element analysis and shape optimization by PSO for 3D axi-symmetric high frequency Helmholtz acoustic problems", *Journal of Sound and Vibration*, vol. 486, p. 115598, nov. 2020, doi: 10.1016/j.jsv.2020.115598.
[23] M. Yoon, J. Lee, y B. Koo, "Shape design optimization of thermoelasticity problems using isogeometric boundary element method", *Advances in Engineering Software*, vol. 149, p. 102871, nov. 2020, doi: 10.1016/j.advengsoft.2020.102871.
[24] E. Rank, M. Ruess, S. Kollmannsberger, D. Schillinger, y A. Düster, "Geometric modeling, isogeometric analysis and the finite cell method", *Computer Methods in Applied Mechanics and Engineering*, vol. 249–252, pp. 104–115, dic. 2012, doi: 10.1016/j.cma.2012.05.022.
[25] T. Hoang, C. V. Verhoosel, F. Auricchio, E. H. van Brummelen, y A. Reali, "Mixed Isogeometric Finite Cell Methods for the Stokes problem", *Computer Methods in Applied Mechanics and Engineering*, vol. 316, pp. 400–423, abr. 2017, doi: 10.1016/j.cma.2016.07.027.
[26] S. C. Divi, C. V. Verhoosel, F. Auricchio, A. Reali, y E. H. van Brummelen, "Error-estimate-based adaptive integration for immersed isogeometric analysis", *Computers & Mathematics with Applications*, p. S0898122120301590, may 2020, doi: 10.1016/j.camwa.2020.03.026.
[27] F. J. Rizzo, "An integral equation approach to boundary value problems of classical elastostatics", *Quart. Appl. Math.*, vol. 25, núm. 1, pp. 83–95, abr. 1967, doi: 10.1090/qam/99907.
[28] R. N. Simpson, S. P. A. Bordas, J. Trevelyan, y T. Rabczuk, "A two-dimensional Isogeometric Boundary Element Method for elastostatic analysis", *Computer Methods in Applied Mechanics and Engineering*, vol. 209–212, pp. 87–100, feb. 2012, doi: 10.1016/j.cma.2011.08.008.
[29] M. A. Scott *et al.*, "Isogeometric boundary element analysis using unstructured T-splines", *Computer Methods in Applied Mechanics and Engineering*, vol. 254, pp. 197–221, feb. 2013, doi: 10.1016/j.cma.2012.11.001.
[30] T. Takahashi y T. Matsumoto, "An application of fast multipole method to isogeometric boundary element method for Laplace equation in two dimensions", *Engineering Analysis with Boundary Elements*, vol. 36, núm. 12, pp. 1766–1775, dic. 2012, doi: 10.1016/j.enganabound.2012.06.004.
[31] L. Heltai, M. Arroyo, y A. DeSimone, "Nonsingular isogeometric boundary element method for Stokes flows in 3D", *Computer Methods in Applied Mechanics and Engineering*, vol. 268, pp. 514–539, 2014.
[32] Y. P. Gong y C. Y. Dong, "An isogeometric boundary element method using adaptive integral method for 3D potential problems", *Journal of Computational and Applied Mathematics*, vol. 319, pp. 141–158, ago. 2017, doi: 10.1016/j.cam.2016.12.038.
[33] X. Peng, E. Atroshchenko, P. Kerfriden, y S. P. A. Bordas, "Isogeometric boundary element methods for three dimensional static fracture and fatigue crack growth", *Computer Methods in Applied Mechanics and Engineering*, vol. 316, pp. 151–185, abr. 2017, doi: 10.1016/j.cma.2016.05.038.

[34] J. V. Venås y T. Kvamsdal, “Isogeometric boundary element method for acoustic scattering by a submarine”, *Computer Methods in Applied Mechanics and Engineering*, p. 112670, oct. 2019, doi: 10.1016/j.cma.2019.112670.
[35] K. L. Johnson, *Contact mechanics*. Cambridge university press, 1987.
[36] D. Di Capua y C. Agelet de Saracibar, “A direct elimination algorithm for quasi-static and dynamic contact problems”, *Finite Elements in Analysis and Design*, vol. 93, pp. 107–125, ene. 2015, doi: 10.1016/j.finel.2014.09.001.
[37] F. J. Cavalieri y A. Cardona, “Numerical solution of frictional contact problems based on a mortar algorithm with an augmented Lagrangian technique”, *Multibody Syst Dyn*, vol. 35, núm. 4, pp. 353–375, dic. 2015, doi: 10.1007/s11044-015-9449-8.
[38] D. M. Neto, M. C. Oliveira, y L. F. Menezes, “Surface Smoothing Procedures in Computational Contact Mechanics”, *Arch Computat Methods Eng*, vol. 24, núm. 1, pp. 37–87, ene. 2017, doi: 10.1007/s11831-015-9159-7.
[39] R. Dimitri, L. De Lorenzis, M. A. Scott, P. Wriggers, R. L. Taylor, y G. Zavarise, “Isogeometric large deformation frictionless contact using T-splines”, *Computer Methods in Applied Mechanics and Engineering*, vol. 269, pp. 394–414, feb. 2014, doi: 10.1016/j.cma.2013.11.002.
[40] Y. Guo, M. Ruess, y Z. Gürdal, “A contact extended isogeometric layerwise approach for the buckling analysis of delaminated composites”, *Composite Structures*, vol. 116, pp. 55–66, sep. 2014, doi: 10.1016/j.compstruct.2014.05.006.
[41] I. Temizer, M. M. Abdalla, y Z. Gürdal, “An interior point method for isogeometric contact”, *Computer Methods in Applied Mechanics and Engineering*, vol. 276, pp. 589–611, jul. 2014, doi: 10.1016/j.cma.2014.03.018.
[42] I. Temizer y C. Hesch, “Hierarchical NURBS in frictionless contact”, *Computer Methods in Applied Mechanics and Engineering*, vol. 299, pp. 161–186, feb. 2016, doi: 10.1016/j.cma.2015.11.006.
[43] Siraj-ul-Islam y S. Ismail, “Meshless collocation procedures for time-dependent inverse heat problems”, *International Journal of Heat and Mass Transfer*, vol. 113, pp. 1152–1167, oct. 2017, doi: 10.1016/j.ijheatmasstransfer.2017.06.028.
[44] Y. Cai, P. Sun, H. Zhu, y T. Rabczuk, “A mixed cover meshless method for elasticity and fracture problems”, *Theoretical and Applied Fracture Mechanics*, vol. 95, pp. 73–103, jun. 2018, doi: 10.1016/j.tafmec.2018.01.011.
[45] G. Kosec *et al.*, “Weak and strong from meshless methods for linear elastic problem under fretting contact conditions”, *Tribology International*, vol. 138, pp. 392–402, oct. 2019, doi: 10.1016/j.triboint.2019.05.041.
[46] J. Gong, D. Zou, X. Kong, Y. Qu, y J. Liu, “An extended meshless method for 3D interface simulating soil-structure interaction with flexibly distributed nodes”, *Soil Dynamics and Earthquake Engineering*, vol. 125, p. 105688, oct. 2019, doi: 10.1016/j.soildyn.2019.05.027.
[47] Q. Zhao, C.-M. Fan, F. Wang, y W. Qu, “Topology optimization of steady-state heat conduction structures using meshless generalized finite difference method”, *Engineering Analysis with Boundary Elements*, vol. 119, pp. 13–24, oct. 2020, doi: 10.1016/j.enganabound.2020.07.002.
[48] I. Ahmad, M. N. Khan, M. Inc, H. Ahmad, y K. S. Nisar, “Numerical simulation of simulate an anomalous solute transport model via local meshless method”, *Alexandria Engineering Journal*, vol. 59, núm. 4, pp. 2827–2838, ago. 2020, doi: 10.1016/j.aej.2020.06.029.
[49] J. Gong, D. Zou, X. Kong, J. Liu, y K. Chen, “A coupled meshless-SBFEM-FEM approach in simulating soil-structure interaction with cross-scale model”, *Soil Dynamics and Earthquake Engineering*, vol. 136, p. 106214, sep. 2020, doi: 10.1016/j.soildyn.2020.106214.
[50] R. Alvarez Cuervo y J. Roces García, *Introducción al diseño paramétrico: con Autodesk Mechanical Desktop*. Asturias: Universidad de Oviedo, 2005.
[51] L. Piegl y T. Wayne, *The NURBS Book*, 2a ed. Springer-Verlag Berlin Heidelberg, 1997.
[52] D. F. Rogers, *An introduction to NURBS: With historical perspective*. Morgan Kaufmann Publishers, 2001.
[53] Becker, A. A., *The boundary element method in engineering: a complete course*. Cambridge: Mc-Graw Hill International, 1992.
[54] T. A. Cruse, “Numerical solutions in three dimensional elastostatics”, *International Journal of Solids and Structures*, vol. 5, núm. 12, pp. 1259–1274, dic. 1969, doi: 10.1016/0020-7683(69)90071-7.

[55] S. S. Rao, *Engineering optimization: theory and practice*, 4th ed. Hoboken, N.J: John Wiley & Sons, 2009.
[56] M. Birattari, L. Paquete, y K. Varrentrapp, “Classification of metaheuristics and design of experiments for the analysis of components”, 2001.
[57] G. J. F. Jones, “CGA 04-Genetic and Evolutionary Algorithms”, 1990.
[58] D. E. Goldberg y J. H. Holland, “Genetic algorithms and machine learning”, 1988.
[59] I. Rechenberg, “Cybernetic solution path of an experimental problem”, *Royal Aircraft Establishment Library Translation 1122*, 1965.
[60] H.-P. P. Schwefel, *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., 1993.
[61] J. R. Koza, M. A. Keane, y M. J. Streeter, “Evolving inventions”, *Scientific American*, vol. 288, núm. 2, pp. 52–59, 2003.
[62] K. Price, R. M. Storn, y J. A. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.
[63] J. Kennedy y R. Eberhart, “Particle swarm optimization”, en *Proceedings of ICNN’95-International Conference on Neural Networks*, 1995, vol. 4, pp. 1942–1948.
[64] M. Dorigo, M. Birattari, y T. Stutzle, “Ant colony optimization”, *IEEE computational intelligence magazine*, vol. 1, núm. 4, pp. 28–39, 2006.
[65] X.-S. Yang, “A new metaheuristic bat-inspired algorithm”, en *Nature inspired cooperative strategies for optimization (NICSO 2010)*, Springer, 2010, pp. 65–74.
[66] P. J. Van Laarhoven y E. H. Aarts, “Simulated annealing”, en *Simulated annealing: Theory and applications*, Springer, 1987, pp. 7–15.
[67] F. Glover, “Tabu search—part I”, *ORSA Journal on computing*, vol. 1, núm. 3, pp. 190–206, 1989.
[68] E.-N. Dragoi, S. Curteanu, A.-I. Galaction, y D. Cascaval, “Optimization methodology based on neural networks and self-adaptive differential evolution algorithm applied to an aerobic fermentation process”, *Applied Soft Computing*, vol. 13, núm. 1, pp. 222–238, 2013.
[69] Y. Wang, G. Bu, Y. Wang, T. Zhao, Z. Zhang, y Z. Zhu, “Application of a simulated annealing algorithm to design and optimize a pressure-swing distillation process”, *Computers & Chemical Engineering*, vol. 95, pp. 97–107, 2016.
[70] F. Bre, A. S. Silva, E. Ghisi, y V. D. Fachinotti, “Residential building design optimisation using sensitivity analysis and genetic algorithm”, *Energy and Buildings*, vol. 133, pp. 853–866, 2016.
[71] X. Li, K. Xing, Y. Wu, X. Wang, y J. Luo, “Total energy consumption optimization via genetic algorithm in flexible manufacturing systems”, *Computers & Industrial Engineering*, vol. 104, pp. 188–200, 2017.
[72] M. Roca-Riu, M. Estrada, y C. Trapote, “The design of interurban bus networks in city centers”, *Transportation research part A: policy and practice*, vol. 46, núm. 8, pp. 1153–1165, 2012.
[73] A. Mahapatro y P. M. Khilar, “Detection and diagnosis of node failure in wireless sensor networks: A multiobjective optimization approach”, *Swarm and Evolutionary Computation*, vol. 13, pp. 74–84, 2013.
[74] K. V. Narasimha, E. Kivelevitch, B. Sharma, y M. Kumar, “An ant colony optimization technique for solving min–max multi-depot vehicle routing problem”, *Swarm and Evolutionary Computation*, vol. 13, pp. 63–73, 2013.
[75] P. K. Das, H. S. Behera, y B. K. Panigrahi, “A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning”, *Swarm and Evolutionary Computation*, vol. 28, pp. 14–28, 2016.
[76] J. Wu, H. Wu, Y. Song, Y. Cheng, W. Zhao, y Y. Wang, “Genetic algorithm trajectory plan optimization for EAMA: EAST Articulated Maintenance Arm”, *Fusion Engineering and Design*, vol. 109, pp. 700–706, 2016.
[77] R. Alvizu, X. Zhao, G. Maier, Y. Xu, y A. Pattavina, “Energy efficient dynamic optical routing for mobile metro-core networks under tidal traffic patterns”, *Journal of Lightwave Technology*, vol. 35, núm. 2, pp. 325–333, 2016.
[78] K. K. Fung, G. F. Lewis, y X. Wu, “The optimisation of low-acceleration interstellar relativistic rocket trajectories using genetic algorithms”, *Acta Astronautica*, vol. 133, pp. 258–268, 2017.
[79] P. V. Silvestrin y M. Ritt, “An iterated tabu search for the multi-compartment vehicle routing problem”, *Computers & Operations Research*, vol. 81, pp. 192–202, 2017.

[80] K.-S. Low y T.-S. Wong, “A multiobjective genetic algorithm for optimizing the performance of hard disk drive motion control system”, *IEEE Transactions on Industrial Electronics*, vol. 54, núm. 3, pp. 1716–1725, 2007.
[81] J. Zhao, F. Yao, H. Wang, Y. Mi, y Y. Wang, “Research on application of genetic algorithm in optimization design of transformer”, en *2011 4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT)*, 2011, pp. 955–958.
[82] E. M. Bates, W. J. Birmingham, y C. Romero-Talamás, “Design Optimization of Nested Bitter Magnets”, *IEEE Transactions on Magnetics*, vol. 53, núm. 3, pp. 1–10, 2016.
[83] C. Gologlu y M. Zeyveli, “A genetic approach to automate preliminary design of gear drives”, *Computers & Industrial Engineering*, vol. 57, núm. 3, pp. 1043–1051, 2009.
[84] S. Gupta, R. Tiwari, y S. B. Nair, “Multi-objective design optimisation of rolling bearings using genetic algorithms”, *Mechanism and Machine Theory*, vol. 42, núm. 10, pp. 1418–1443, 2007.
[85] S. Corbera, J. L. Olazagoitia, y J. A. Lozano, “Multi-objective global optimization of a butterfly valve using genetic algorithms”, *ISA transactions*, vol. 63, pp. 401–412, 2016.
[86] S. V. Camacho-Gutiérrez, J. C. Jáuregui-Correa, y A. Dominguez, “Optimization of Excitation Frequencies of a Gearbox Using Algorithms Inspired by Nature”, *J. Vib. Eng. Technol.*, vol. 7, núm. 6, pp. 551–563, dic. 2019, doi: 10.1007/s42417-019-00149-6.
[87] J. Kennedy y R. Eberhart, “Particle swarm optimization”, *1995 IEEE International Conference on Neural Networks (ICNN 95)*, vol. 4, pp. 1942–1948, 1995, doi: 10.1109/ICNN.1995.488968.
[88] M. Clerc, *Particle swarm optimization*, vol. 93. John Wiley & Sons, 2010.
[89] M. Guiggiani, “A self-adaptive co-ordinate transformation for efficient numerical evaluation of general boundary element integrals”, *Int. J. Numer. Meth. Engng.*, vol. 26, núm. 7, pp. 1683–1684, jul. 1988, doi: 10.1002/nme.1620260715.
[90] W. C. Young, R. G. Budynas, y A. M. Sadegh, *Roark’s formulas for stress and strain*, vol. 7. McGraw-Hill New York, 2002.
[91] M. Lefebvre, R. K. Keeler, R. Sobie, y J. White, “Propagation of errors for matrix inversion”, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 451, núm. 2, pp. 520–528, sep. 2000, doi: 10.1016/S0168-9002(00)00323-5.

Apéndices

Apéndice A. Programas IGA-BEM propuesto y de contacto

El diagrama de interacción del programa basado en el IGA-BEM propuesto se muestra en la Figura Apéndice 1. El programa “Principal” es la interfaz base, que manda a llamar a la función “DeriveNURBS_Point” para generar los puntos de las curvas NURBS, y a la función “SaveNi” para guardar los valores de las funciones base. Las condiciones de frontera se obtienen de un archivo externo de Excel conocido como “Datos.xlsx”. En este mismo archivo se guardan los resultados de desplazamientos y tracciones. Las figuras mostradas en la sección de resultados se generaron con en el software de “*Origin*” y los resultados guardados en Excel.

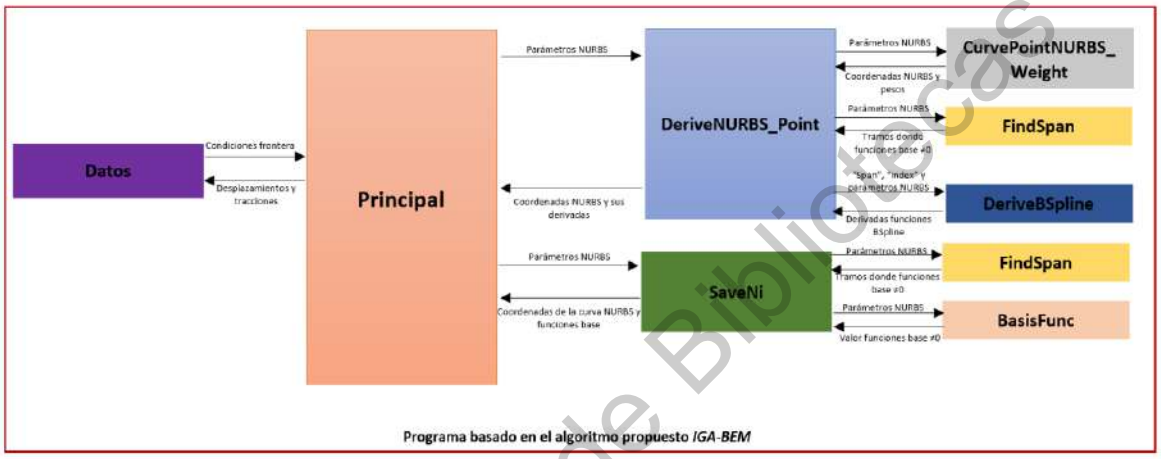


Figura Apéndice 1 Diagrama de interacción entre los subprogramas del algoritmo IGA-BEM propuesto.

Código del programa IGA-BEM propuesto

Principal.m

```
%%Programa principal para el análisis de un cuerpo usando el método
%%isogeométrico-elementos frontera (IGA-BEM)
clearvars
clc

%%Definimos el cuerpo de análisis%%
%Orden de la curva NURBS
p = 3;
%Vector de nudos
U=[ 0 0 0 1 1 2 2 3 3 4 4 4];
%Puntos de control
P=[0 0;3 0;3 3;3 6;0 6;-3 6;-3 3;-3 0;0 0];
%Pesos
W=[1 sqrt(2)/2 1 sqrt(2)/2 1 sqrt(2)/2 1 sqrt(2)/2 1];
%Tamaño del vector de nudos (m)
[~,m]=size(U);
%No. de funciones base
n=m-p;
%No. de puntos de control
[Cpoints,~]=size(P(:,1));
%Puntos para dibujar la curva NURBS
Npoints=22+2;
f=1/100;
u_aux=linspace(f,U(end)-f,Npoints-2);
u=zeros(1,Npoints);
u(1,1:11)=u_aux(1,1:11);
u(1,12)=1.99; %Este valor de nudo lo agregamos para que la fuerza quedé aplicada en (0,6)
u(1,13)=2.01;
u(1,14:end)=u_aux(1,12:end);
%Inicializamos matrices
C=zeros(Npoints,2);
Cder=zeros(Npoints,2);
Ni=zeros(Npoints,Cpoints);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas, que mas
%adelante se volverán puntos de carga
for i=1:Npoints
    [C(i,:),Cder(i,:)]=DeriveNURBS_Point(n,p,u(i),P,U,W);
    %Guardamos los valores de las funciones base para posteriormente
```

```
    %encontrar los nuevos puntos de control
    [~,Ni(i,:)]=SaveNi(n,p,u(i),P,U,Cpoints,W);
end
plot(C(:,1),C(:,2),'r*')

hold on
%%
%%Establecemos las características del cuerpo
%Módulo de Young
E=2e11;
%E=1;
%Coeficiente de Poisson
v=.3;
%Modulo de corte
miu=E/(2*(1+v));
%%
%Establecemos las condiciones fronteras, se tendrán que modificar las
%columnas de los xlsread para obtener adecuadamente los datos
%Desplazamientos (No. Nodo, Dirección (x=1,y=2),Cantidad)
DesPre=xlsread('Preescritos_D.xlsx','Desplazamientos','A2:C5');
%Tracciones
TracPre=xlsread('Preescritos_D.xlsx','Tracciones','A2:C5');
%%
%%%Calculamos los rangos del parámetro del vector de nudos y las normales a
%%%los puntos que estamos analizando
Rango=[0 1;1 2;2 3;3 4];
Nor=zeros(Npoints,2);
for i=1:Npoints
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    Nor(i,1)=1*(Cder(i,2))/Jacobiano;
    Nor(i,2)=-1*(Cder(i,1))/Jacobiano;
end
%%
%%%%Calculo de Kernels
DMAX=0;
T=zeros(Npoints*2,Npoints*2);
D=zeros(Npoints*2,Npoints*2);
Derivada=zeros(Npoints,Npoints);
for i=1:Npoints
    XP=C(i,1);
    YP=C(i,2);
    IROW1=2*i-1;
    IROW2=IROW1+1;
    for j=1:Npoints
        if i==j
            continue
        else
            XQ=C(j,1);
            YQ=C(j,2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            if r>DMAX
                DMAX=r;
            end
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            %dx/dpsi=Cder(j,1) y dy/dpsi=Cder(j,2) (psi es la variable de
            %la NURBS)
            Jacobiano=sqrt(Cder(j,1)^2+Cder(j,2)^2);
            nx=1*(Cder(j,2))/Jacobiano;
            ny=-1*(Cder(j,1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            %Derivada(i,j)=drdn;%%
            ICOL1=2*j-1;
            ICOL2=ICOL1+1;
            %Calculamos kernels de desplazamiento
            %Factor=1; %Lo hacemos uno para que no afecte las matrices, original 1/(8*pi*miu*(1-v))
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IROW1,ICOL1)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IROW1,ICOL2)=FactorA*drdx*drdy;
            %Uyx
            D(IROW2,ICOL1)=FactorA*drdx*drdy;
            %Uyy
            D(IROW2,ICOL2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %Calculamos kernels de tracción
            %Factor=-1/r;
            Factor=-1/(4*pi*(1-v)*r);% Original
            %Txx
            T(IROW1,ICOL1)=Factor*drdn*((1-2*v)+2*(drdx^2));
            %Txy
            T(IROW1,ICOL2)=Factor*(2*drdx*drdy*drdn+(1-2*v)*(drdy*nx-drdx*ny));
            %Tyx
            T(IROW2,ICOL1)=Factor*(2*drdx*drdy*drdn-(1-2*v)*(drdy*nx-drdx*ny));
            %Tyy
            T(IROW2,ICOL2)=Factor*drdn*((1-2*v)+2*(drdy^2));
        end
    end
end
%%
%Calculamos la diagonal de la matriz T->Tij+Ci con el término de salto como
%en el artículo de Simpson (2012) y también calculamos la diagonal de la
%matriz U, haciendo que Q se aleje de P cuando coincidan en una razón
%+eps*algo
```

```
Cij=[.5 0;0 .5]; %Matriz de salto, solo aplica para la sección transversal de un círculo
for i=1:Npoints
    IRow=2*i-1;
    IRow2=IRow+1;
            %Diagonales, consideraciones cuerpo rígido
            %Txx
            T(IRow,IRow)=Cij(1,1); %T(IRow,IRow)-T(IRow,ICol);
            %Txy
            T(IRow,IRow2)=Cij(1,2);%T(IRow,IRow2)-T(IRow,ICol2);
            %Tyx
            T(IRow2,IRow)=Cij(2,1);%T(IRow,IRow2);
            %T(IRow2,IRow)=T(IRow2,IRow)-T(IRow,ICol);<-No funciona para la
            %deformación de un círculo
            %Tyy
            T(IRow2,IRow2)=Cij(2,2);
            %Diagonales de la matriz U, con r indeterminado
            XP=C(i,1);
            YP=C(i,2);
            xi=u(i)+eps*1e12; %Incrementamos el valor de Q
            [Cxi,Derxi]=DeriveNURBS_Point(n,p,xi,P,U,W);
            XQ=Cxi(1);
            YQ=Cxi(2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            Jacobiano=sqrt(Derxi(1)^2+Derxi(2)^2);
            nx=1*(Derxi(2))/Jacobiano;
            ny=-1*(Derxi(1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IRow,IRow)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IRow,IRow2)=FactorA*drdx*drdy;
            %Uyx
            D(IRow2,IRow)=FactorA*drdx*drdy;
            %Uyy
            D(IRow2,IRow2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %D(IRow:IRow2,IRow:IRow2)
end
%%
%Calculamos la integral de superficie, aplicando la regla de Simpson 1/3
%compuesta, el Jacobiano lo representamos con más número de puntos que con
%el cálculo anterior para aproximar mejor la forma del círculo
%Puntos para dibujar la curva NURBS
NpointJ=50;
uJ=linspace(1/NpointJ,U(end),NpointJ);
CJ=zeros(NpointJ,2);
Cder_J=zeros(NpointJ,2);
Jacob=zeros(NpointJ,1);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas
for i=1:NpointJ
    [CJ(i,:),Cder_J(i,:)]=DeriveNURBS_Point(n,p,uJ(i),P,U,W);
    Jacob(i,1)=sqrt(Cder_J(i,1)^2+Cder_J(i,2)^2);
end
plot(CJ(:,1),CJ(:,2),'r');
npointJ=NpointJ-1;
b=U(end);
a=U(1);
h=(b-a)/npointJ;
%Integral del jacobiano de psi
I= h/3*(Jacob(1)+2*sum(Jacob(3:2:end-2))+4*sum(Jacob(2:2:end))+Jacob(end));
%Multiplicamos por el perímetro y en el caso de  la matriz  D, por el factor
%de escalamiento (Ec. 4.62)
D=D.*I;
T=T.*I;
%%
%%Aplicamos las condiciones fronteras a los Kernels para formar la matriz A*
%%Y B* (4.50 y 4.58 Libro Becker)
[N_dis,~]=size(DesPre);
[N_trac,~]=size(TracPre);
TTracPres=zeros(Npoints*2,1);
%Colocamos las tracciones conocidas en el vector tracciones
for i=1:N_trac
    index=TracPre(i,1)*2-2+TracPre(i,2); %Ubicación de la tracción prescrita
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    nx=1*(Cder(i,2))/Jacobiano;
    ny=-1*(Cder(i,1))/Jacobiano;
   %Las presiones prescritas las normalizamos con el factor SCALE
    if TracPre(i,2)==1
        TTracPres(index,1)=TracPre(i,3)*nx;
    else
        TTracPres(index,1)=TracPre(i,3)*-ny;
    end
end

%Borramos las columnas y renglones correspondientes a desplazamientos conocidos
AuxTrac=TTracPres;
for I = length(DesPre(:,1)):-1:1
    GDL = DesPre(I,1)*2-2+DesPre(I,2);
    T(GDL,:)=[];
    T(:,GDL)=[];
    D(GDL,:)=[];
    D(:,GDL)=[];
```

```
    AuxTrac(GDL,:)=[];
end

%Resolvemos el sistema de ecuaciones
KJ=D/T;
UU=KJ*AuxTrac;

%Ordenamos los desplazamientos
UUNew=zeros(Npoints*2,1);
a=1;
for i=1:Npoints*2
    if a<=length(DesPre(:,1))
        index=DesPre(a,1)*2-2+DesPre(a,2);
    end
    if i==index
        UUNew(i,1)=DesPre(a,3);
        a=a+1;
    else
        UUNew(i,1)=UU(i-a+1);
    end
end
%%
%Obteniendo el cuerpo deformado con NURBS

%a)Sumamos los desplazamientos a los puntos que usamos como puntos de carga
AUX=UUNew(1:2:end);
AUX2=UUNew(2:2:end);
scale=1;
Cnew=C+[scale*UUNew(1:2:end) scale*UUNew(2:2:end)];
%plot(Cnew(:,1),Cnew(:,2),'k')
UUNewXY=[UUNew(1:2:end) UUNew(2:2:end)];
TracXY=[TTracPres(1:2:end) TTracPres(2:2:end)];
%b)Obtenemos la matriz inversa de Ni y lo multiplicamos por los nuevos
%datos de la curva deformada para obtener los puntos de control
Pnew=inv(Ni'*Ni)*Ni'*Cnew;

%c)Graficamos la curva deformada
NpointsDraw=40;
Cdef=zeros(NpointsDraw,2);
%f=1/100;
u_new=linspace(U(1),U(end),NpointsDraw);
for i=1:NpointsDraw
    Cdef(i,:)=CurvePointNURBS(n,p,u_new(i),Pnew,U,W);
end

plot(Cnew(:,1),Cnew(:,2),'--k')
plot(Pnew(:,1),Pnew(:,2),'*k')
hor=zeros(1,10);
xhor=linspace(-3,3,10);
plot(xhor,hor);
hold off
figure (2)
plot(C(:,1),C(:,2),'r',Cnew(:,1),Cnew(:,2),'b*');
%%
%Empezamos a calcular los puntos internos
%Coordenadas de los puntos internos
XINT=xlsread('Preescritos_D.xlsx','PInternos','A2:A14');
YINT=xlsread('Preescritos_D.xlsx','PInternos','B2:B14');
%Obtenemos el número de puntos internos
[r,~]=size(XINT);
%Inicializamos las variables internas
UXINT=zeros(r,1);
UYINT=zeros(r,1);
for i=1:r
    %Tomar cada nodo interno como el nodo de carga "NodeP"
    XP=XINT(i);
    YP=YINT(i);
    for j=1:Npoints
        %Tomar los nodos del cálculo original como el nodo de campo "NodeQ"
        XQ=C(j,1);
        YQ=C(j,2);
        r=sqrt((XP-XQ)^2+(YP-YQ)^2);
        drdx=(XQ-XP)/r;
        drdy=(YQ-YP)/r;
        %dx/dpsi=Cder(j,1) y dy/dpsi=Cder(j,2) (psi es la variable de
        %la NURBS)
        Jacobiano=sqrt(Cder(j,1)^2+Cder(j,2)^2);
        nx=1*(Cder(j,2))/Jacobiano;
        ny=-1*(Cder(j,1))/Jacobiano;
        %Derivada(i,j)=drdn;%%
        drdn=drdx*nx+drdy*ny;
        %Desplazamientos ya conocidos
        UX=UUNewXY(j,1);
        UY=UUNewXY(j,2);
        %Tracciones ya conocidas
        TR=TracXY(j,1);
        TZ=TracXY(j,2);
        %Calculamos kernels de desplazamiento
        %Factor=1; %Lo hacemos uno para que no afecte las matrices, original 1/(8*pi*miu*(1-v))
        FactorA=1/(8*pi*miu*(1-v));
        %Uxx
        UXX=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
        %Uxy
        UXY=FactorA*drdx*drdy;
```

```
        %Uyx
        UYX=FactorA*drdx*drdy;
        %Uyy
        UYY=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
        %Calculamos kernels de tracción
        %Factor=-1/r;
        Factor=-1/(4*pi*(1-v)*r);% Original
        %Txx
        TXX=Factor*drdn*((1-2*v)+2*(drdx^2));
        %Txy
        TXY=Factor*(2*drdx*drdy*drdn+(1-2*v)*(drdy*nx-drdx*ny));
        %Tyx
        TYX=Factor*(2*drdx*drdy*drdn-(1-2*v)*(drdy*nx-drdx*ny));
        %Tyy
        TYY=Factor*drdn*((1-2*v)+2*(drdy^2));
        %Caculamos los desplazamientos internos usando la ecuación integral
        %4.32 de Becker
        UXINT(i,1)=UXINT(i,1)+(-UX*TXX-UY*TXY+TR*UXX+TZ*UXY);
        UYINT(i,1)=UYINT(i,1)+(-UX*TYX-UY*TYY+TR*UYX+TZ*UYY);
        %A diferencia del libro de Becker, no incluimos ningún factor en
        %esta ecuación ("FACT")
    end
end
%%
%Guardamos datos para graficar en origin
xlswrite('Preescritos_D.xlsx',CJ,'PlotXY','A2');%Guardamos coordenadas del círculo
xlswrite('Preescritos_D.xlsx',UUNewXY(:,2),'Salida_UY','C2'); %Guardamos UY
xlswrite('Preescritos_D.xlsx',Cnew,'Salida_UY','A2'); %Guardamos coordenadas X y
xlswrite('Preescritos_D.xlsx',XINT,'Salida_UYint','A2'); %Guardamos coordenadas X punto interno
xlswrite('Preescritos_D.xlsx',YINT,'Salida_UYint','B2'); %Guardamos coordenadas Y punto interno
xlswrite('Preescritos_D.xlsx',UYINT,'Salida_UYint','C2'); %Guardamos desplazamientos Y internos
```

DeriveNURBSpoint.m

```
function[C,Cder]=DeriveNURBS_Point(n,p,u,P,U,W)
%Este programa calcula la derivada de un punto de una curva NURB basado en
%la ecuación 4.7 del libro "NURBS' Book"
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%P=vector con los puntos de control
%W=Pesos de la curva
%C=Punto de la curva de acuerdo al valor de u

WPoints=[P W'];
WPoints(:,1:2)=[WPoints(:,1).*WPoints(:,3) WPoints(:,2).*WPoints(:,3)];
span=FindSpan(n,p,u,U);
%Obtenemos el punto de la curva NURB
[C,Weight]=CurvePointNURBS_Weight(n,p,u,P,U,W);
C1=[0 0 0];
for i=1:p
    index=span-p+i;
    [D]=DeriveBSpline(span,index,p,u,U);
    %Aquí obtenemos la derivada de A'(u) y W'(u)
    C1=C1+D*WPoints(index,:);
end
%Obtenemos la derivada de un punto de una curva NURB (Ec. 4.7):
Cder=(C1(1,1:2)-C1(1,3).*C)./Weight;
```

CurvePointNURBS_weight.m

```
function [C,Weight]=CurvePointNURBS_Weight(n,p,u,P,U,W)
%Este programa calcula el valor de un punto de acuerdo al valor del
%parámetro u
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%P=vector con los puntos de control
%w=Vector de pesos
%C=Punto de la curva de acuerdo al valor de u
WPoints=[P W'];
WPoints(:,1:2)=[WPoints(:,1).*WPoints(:,3) WPoints(:,2).*WPoints(:,3)];
span=FindSpan(n,p,u,U);
N=BasisFunc(span,u,p,U);
Cw=[0 0 0];
for i=1:p
    Cw=Cw+N(i)*WPoints(span-p+i,:);
end
C=Cw(1,1:2)./Cw(1,3);
Weight=Cw(1,3);
```

FindSpan.m

```
function[span]=FindSpan(n,p,u,U)
%Programa para encontrar en que intervalo yace el valor de u
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
if u==U(n+1)
    span=n;
elseif u==U(p)
    span=p;
else
%     low=p
%     high=n+1
%     mid=round((low+high)/2)
    for i=p+1:n+1
        if u < U(i)
            span=i-1;
            break
        elseif u==U(i)
            f=find(U==u);
            %Checamos si el valor de u se repite más de una vez, y tomamos
            %el índice del último valor repetido como el span
            [~,c]=size(f);
            if c >= 2
                span=f(end);
                break
            else
            %Si el valor no está repetido el valor del span es i
                span=i;
                break
            end
        end
    end
end
```

DeriveBSpline.m

```
function[D]=DeriveBSpline(span,i,p,u,U)
%Programa para calcular la primer derivada de una B-Spline
%span=índice de acuerdo al valor del parámetro u en el vector de nudos
%i=Índice de la función base a derivar, ejemplo: N'i,p
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%Siguiendo la ecuación 2.7 del libro "NURBS' book":

%1)Obtenemos las funciones base diferentes de cero Nspan,p-1
FunBase=CallFuncBasis(span,p-1,u,U);
%2)Buscamos si el índice i y el índice i+1 en las funciones base diferentes
%de cero, si no están, entonces la función Ni,p=0 o Ni+1,p=0
Index1=find(FunBase(:,1)==i);
if isempty(Index1)
    Nip=0;
else
    Nip=FunBase(Index1,2);
end
Index2=find(FunBase(:,1)==i+1);
if isempty(Index2)
    Ni_1p=0;
else
    Ni_1p=FunBase(Index2,2);
end
%Escribimos la función tal como la ecuación 2.7, note que donde está p se
%le quita uno, ya que en matlab trabajamos con el orden y no con el grado
%de la curva
DenLeft=U(i+p-1)-U(i);
if DenLeft==0
    left=0;
else
    left=((p-1)/DenLeft)*Nip;
end
DenRight=U(i+p)-U(i+1);
if DenRight==0
    right=0;
else
    right=((p-1)/DenRight)*Ni_1p;
end
D=left-right;
```

SaveNi.m

```
function [C,Ni]=SaveNi(n,p,u,P,U,Cpoints,W)
%Este programa calcula el valor de un punto de acuerdo al valor del
%parámetro u
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%P=vector con los puntos de control
%w=Vector de pesos
%C=Punto de la curva de acuerdo al valor de u
WPoints=[P W'];
WPoints(:,1:2)=[WPoints(:,1).*WPoints(:,3) WPoints(:,2).*WPoints(:,3)];
span=FindSpan(n,p,u,U);
N=BasisFunc(span,u,p,U);
Ni=zeros(1,Cpoints);
Cw=[0 0 0];
Weight=0;
for i=1:p
    Cw=Cw+N(i)*WPoints(span-p+i,:);
    Ni(1,span-p+i)=N(i)*WPoints(span-p+i,3);
    Weight=Weight+N(i)*WPoints(span-p+i,3);
end
C=Cw(1,1:2)./Cw(1,3);
Ni(1,:)=Ni(1,:)./Weight;
```

BasisFunc.m

```
function [C,Ni]=SaveNi(n,p,u,P,U,Cpoints,W)
%Este programa calcula el valor de un punto de acuerdo al valor del
%parámetro u
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%P=vector con los puntos de control
%w=Vector de pesos
%C=Punto de la curva de acuerdo al valor de u
WPoints=[P W'];
WPoints(:,1:2)=[WPoints(:,1).*WPoints(:,3) WPoints(:,2).*WPoints(:,3)];
span=FindSpan(n,p,u,U);
N=BasisFunc(span,u,p,U);
Ni=zeros(1,Cpoints);
Cw=[0 0 0];
Weight=0;
for i=1:p
    Cw=Cw+N(i)*WPoints(span-p+i,:);
    Ni(1,span-p+i)=N(i)*WPoints(span-p+i,3);
    Weight=Weight+N(i)*WPoints(span-p+i,3);
end
C=Cw(1,1:2)./Cw(1,3);
Ni(1,:)=Ni(1,:)./Weight;
```

El diagrama mostrado en la Figura Apéndice 2 representa la interacción entre los subprogramas que se implementaron para validar el algoritmo de contacto propuesto en Figura 3-3, Figura 3-7 y Figura 3-8. El programa base es “Principal1”, donde se implementa la mayor parte del algoritmo de contacto propuesto. Sin embargo, esta interfaz interactúa con elementos adicionales, como el subprograma “NURBSParameters” y el subprograma “NewValues”. “NURBSParameters” agrega puntos de control adicionales y calcula los pesos asociados a dichos puntos. “NewValues” manda a llamar al programa “PSO”, que obtiene los valores optimizados de los puntos de control. En la Figura Apéndice 3 se muestra un diagrama de los subprogramas del algoritmo de optimización. Información detallada de los programas se describe en el capítulo 3.

Finalmente, en la Figura Apéndice 4 se muestra el diagrama de interacción de los subprogramas para validar al algoritmo de contacto, se toman los puntos de control optimizados del programa “Principal1.m”, y se guardan en el archivo “PlotOther.m”, luego se ejecuta el programa “Principal2.m”. Si ya no hay cambios significativos en la línea de contacto, entonces se asume que

el problema de contacto se resolvió y se obtienen los desplazamientos internos para posteriormente guardarse en el archivo “Datos.xlsx” y graficarse en Origin.

En caso de que la línea de contacto haya tenido cambios significativos, entonces, se vuelve a ejecutar el algoritmo de optimización PSO y se generan nuevos puntos de control. Para validar la nueva solución, se sustituyen estos puntos de control en “PlotOther.m” y se ejecuta nuevamente el programa de “Principal2.m”, repitiéndose el proceso hasta encontrar la zona de contacto correcta.

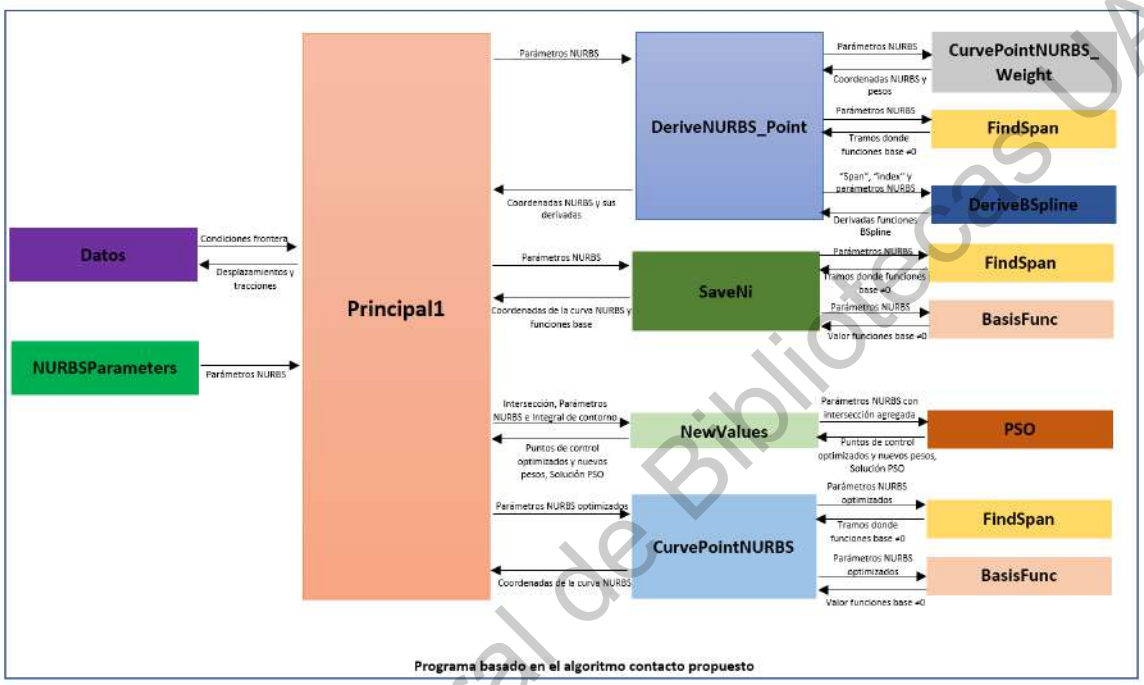


Figura Apéndice 2 Diagrama de interacción entre los subprogramas del algoritmo propuesto de contacto IGA-BEM

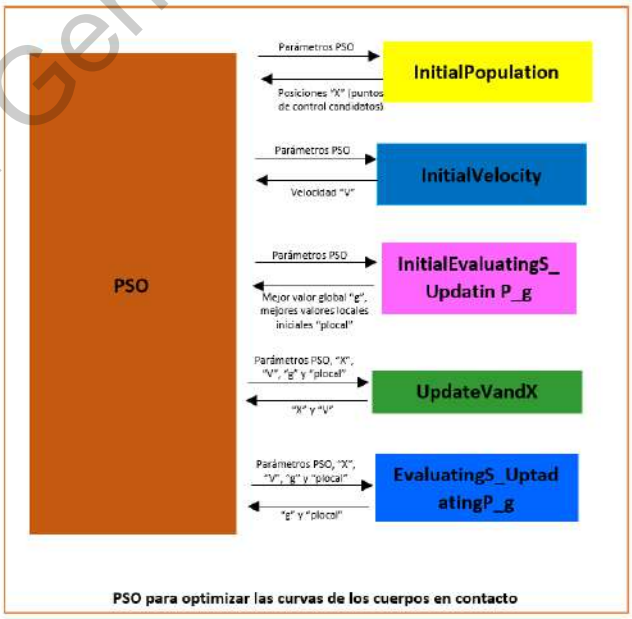


Figura Apéndice 3 Interacción entre los subprogramas asociados al algoritmo PSO

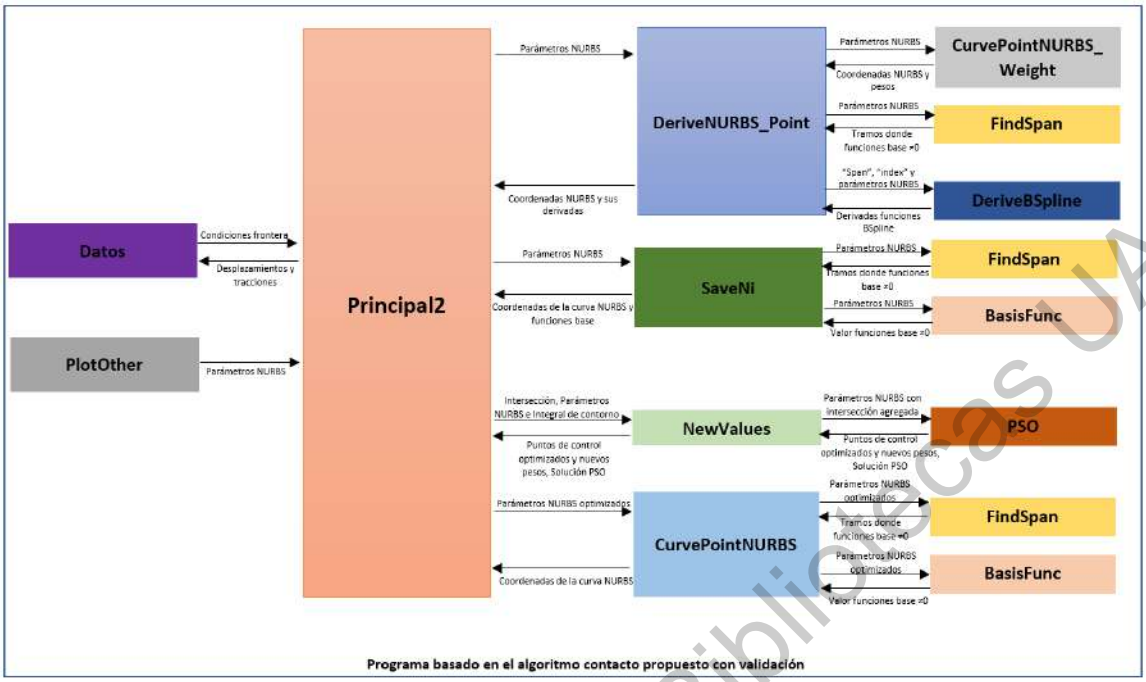


Figura Apéndice 4 Interacción del programa para validar y analizar el contacto entre dos elementos mecánicos

Código del programa del algoritmo de contacto IGA-BEM propuesto

Principal1.m

```
%%Programa principal para el análisis de un cuerpo usando el método
%%isogeométrico-elementos frontera (IGA-BEM)
clearvars
clc

%%Definimos el cuerpo de análisis%%
%Orden de la curva NURBS
p = 3;
%Vector de nudos, Puntos de control y Pesos
[U,P,W] = NURBSParameters( );
%Tamaño del vector de nudos (m)
[~,m]=size(U);
%No. de funciones base
n=m-p;
%No. de puntos de control
[Cpoints,~]=size(P(:,1));
%Puntos para dibujar la curva NURBS
Npoints=22+2;
f=1/100;
u_aux=linspace(f,U(end)-f,Npoints-2);
u=zeros(1,Npoints);
u(1,1:11)=u_aux(1,1:11);
u(1,12)=2.99; %Este valor de nudo lo agregamos para que la fuerza quedé aplicada en (0,6)
u(1,13)=3.01;
u(1,14:end)=u_aux(1,12:end);
%u=u_aux;
%Inicializamos matrices
C=zeros(Npoints,2);
Cder=zeros(Npoints,2);
Ni=zeros(Npoints,Cpoints);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas, que mas
%adelante se volverán puntos de carga
for i=1:Npoints
    [C(i,:),Cder(i,:)]=DeriveNURBS_Point(n,p,u(i),P,U,W);
    %Guardamos los valores de las funciones base para posteriormente
    %encontrar los nuevos puntos de control
    [~,Ni(i,:)]=SaveNi(n,p,u(i),P,U,Cpoints,W);
end
plot(C(:,1),C(:,2),'r*')
hold on
%%
%%Establecemos las características del cuerpo
%Modulo de Young
E=2e11;
```

```
%E=1;
%Coeficiente de Poisson
v=.3;
%Modulo de corte
miu=E/(2*(1+v));
%%
%Establecemos las condiciones frontera, se tendrán que modificar las
%columnas de los xlsread para obtener adecuadamente los datos
%Desplazamientos (No. Nodo, Dirección (x=1,y=2),Cantidad)
DesPre=xlsread('Preescritos_E.xlsx','Desplazamientos','A2:C5');
%Tracciones
TracPre=xlsread('Preescritos_E.xlsx','Tracciones','A2:C5');
%%
%%%Calculamos los rangos del parámetro del vector de nudos y las normales a
%%%los puntos que estamos analizando
Rango=[0 1;1 2;2 3;3 4];
Nor=zeros(Npoints,2);
for i=1:Npoints
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    Nor(i,1)=1*(Cder(i,2))/Jacobiano;
    Nor(i,2)=-1*(Cder(i,1))/Jacobiano;
end
%%
%%%%Calculo de Kernels
DMAX=0;
T=zeros(Npoints*2,Npoints*2);
D=zeros(Npoints*2,Npoints*2);
Derivada=zeros(Npoints,Npoints);
for i=1:Npoints
    XP=C(i,1);
    YP=C(i,2);
    IROW1=2*i-1;
    IROW2=IROW1+1;
    for j=1:Npoints
        if i==j
            continue
        else
            XQ=C(j,1);
            YQ=C(j,2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            if r>DMAX
                DMAX=r;
            end
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            %dx/dpsi=Cder(j,1) y dy/dpsi=Cder(j,2) (psi es la variable de
            %la NURBS)
            Jacobiano=sqrt(Cder(j,1)^2+Cder(j,2)^2);
            nx=1*(Cder(j,2))/Jacobiano;
            ny=-1*(Cder(j,1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            %Derivada(i,j)=drdn;%%
            ICOL1=2*j-1;
            ICOL2=ICOL1+1;
            %Calculamos kernels de desplazamiento
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IROW1,ICOL1)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IROW1,ICOL2)=FactorA*drdx*drdy;
            %Uyx
            D(IROW2,ICOL1)=FactorA*drdx*drdy;
            %Uyy
            D(IROW2,ICOL2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %Calculamos kernels de tracción
            %Factor=-1/r;
            Factor=-1/(4*pi*(1-v)*r);% Original
            %Txx
            T(IROW1,ICOL1)=Factor*drdn*((1-2*v)+2*(drdx^2));
            %Txy
            T(IROW1,ICOL2)=Factor*(2*drdx*drdy*drdn+(1-2*v)*(drdy*nx-drdx*ny));
            %Tyx
            T(IROW2,ICOL1)=Factor*(2*drdx*drdy*drdn-(1-2*v)*(drdy*nx-drdx*ny));
            %Tyy
            T(IROW2,ICOL2)=Factor*drdn*((1-2*v)+2*(drdy^2));
        end
    end
end
%%
%Calculamos la diagonal de la matriz T->Tij+Ci con el término de salto como
%en el artículo de Simpson (2012) y también calculamos la diagonal de la
%matriz U, haciendo que Q se aleje de P cuando coincidan en una razón
%+eps*algo
Cij=[.5 0;0 .5]; %Matriz de salto, solo aplica para la sección transversal de un círculo
for i=1:Npoints
    IRow=2*i-1;
    IRow2=IRow+1;
            %Diagonales, consideraciones cuerpo rígido
            %Txx
            T(IRow,IRow)=Cij(1,1);
            %Txy
            T(IRow,IRow2)=Cij(1,2);
            %Tyx
            T(IRow2,IRow)=Cij(2,1);
```

```
            %T(IRow2,IRow)=T(IRow2,IRow)-T(IRow,ICol);<-No funciona para la
            %deformación de un círculo
            %Tyy
            T(IRow2,IRow2)=Cij(2,2);
            %Diagonales de la matriz U, con r indeterminado
            XP=C(i,1);
            YP=C(i,2);
            xi=u(i)+eps*1e12; %Incrementamos el valor de Q
            [Cxi,Derxi]=DeriveNURBS_Point(n,p,xi,P,U,W);
            XQ=Cxi(1);
            YQ=Cxi(2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            Jacobiano=sqrt(Derxi(1)^2+Derxi(2)^2);
            nx=1*(Derxi(2))/Jacobiano;
            ny=-1*(Derxi(1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IRow,IRow)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IRow,IRow2)=FactorA*drdx*drdy;
            %Uyx
            D(IRow2,IRow)=FactorA*drdx*drdy;
            %Uyy
            D(IRow2,IRow2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %D(IRow:IRow2,IRow:IRow2)
end
%%
%Calculamos la integral de supeficie, aplicando la regla de Simpson 1/3
%compuesta, el Jacobiano lo representamos con más número de puntos que con
%el cálculo anterior para aproximar mejor la forma del círculo
%Puntos para dibujar la curva NURBS
NpointJ=50;
uJ=linspace(1/NpointJ,U(end),NpointJ);
CJ=zeros(NpointJ,2);
Cder_J=zeros(NpointJ,2);
Jacob=zeros(NpointJ,1);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas
for i=1:NpointJ
    [CJ(i,:),Cder_J(i,:)]=DeriveNURBS_Point(n,p,uJ(i),P,U,W);
    Jacob(i,1)=sqrt(Cder_J(i,1)^2+Cder_J(i,2)^2);
end
plot(CJ(:,1),CJ(:,2),'r');
npointJ=NpointJ-1;
b=U(end);
a=U(1);
h=(b-a)/npointJ;
%Integral del jacobiano de psi
Integral= h/3*(Jacob(1)+2*sum(Jacob(3:2:end-2))+4*sum(Jacob(2:2:end))+Jacob(end));
%Multiplicamos por el perímetro
D=D.*Integral;
T=T.*Integral;

%%
%%Aplicamos las condiciones frontera a los Kernels para formar la matriz A*
%%Y B* (4.50 y 4.58 Libro Becker)
[N_dis,~]=size(DesPre);
[N_trac,~]=size(TracPre);
TTracPres=zeros(Npoints*2,1);
%Colocamos las tracciones conocidas en el vector tracciones
for i=1:N_trac
    index=TracPre(i,1)*2-2+TracPre(i,2); %Ubicación de la tracción prescrita
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    nx=1*(Cder(i,2))/Jacobiano;
    ny=-1*(Cder(i,1))/Jacobiano;
   %Las presiones prescritas las normalizamos con el factor SCALE
    if TracPre(i,2)==1
        TTracPres(index,1)=TracPre(i,3)*nx;%/SCALEF;
    else
        TTracPres(index,1)=TracPre(i,3)*ny;%/SCALEF;
    end
end

%Borramos las columnas y renglones correspondientes a desplazamientos conocidos
AuxTrac=TTracPres;
for I = length(DesPre(:,1)):-1:1
    GDL = DesPre(I,1)*2-2+DesPre(I,2);
    T(GDL,:)=[];
    T(:,GDL)=[];
    D(GDL,:)=[];
    D(:,GDL)=[];
    AuxTrac(GDL,:)=[];
end

%Resolvemos el sistema de ecuaciones
KJ=D/T;
UU=KJ*AuxTrac;

%Ordenamos los desplazamientos
UUNew=zeros(Npoints*2,1);
a=1;
for i=1:Npoints*2
```

```
    if a<=length(DesPre(:,1))
        index=DesPre(a,1)*2-2+DesPre(a,2);
    end
    if i==index
        UUNew(i,1)=DesPre(a,3);
        a=a+1;
    else
        UUNew(i,1)=UU(i-a+1);
    end
end
%%
%Obteniendo el cuerpo deformado con NURBS

%a)Sumamos los desplazamientos a los puntos que usamos como puntos de carga
AUX=UUNew(1:2:end);
AUX2=UUNew(2:2:end);
scale=1;
Cnew=C+[scale*UUNew(1:2:end) scale*UUNew(2:2:end)];
%plot(Cnew(:,1),Cnew(:,2),'k')
UUNewXY=[UUNew(1:2:end) UUNew(2:2:end)];
TracXY=[TTracPres(1:2:end) TTracPres(2:2:end)];

plot(Cnew(:,1),Cnew(:,2),'b--');
%b)Obtenemos la matriz inversa de Ni y lo multiplicamos por los nuevos
%datos de la curva deformada para obtener los puntos de control
Pnew=inv(Ni'*Ni)*Ni'*Cnew;

%c)Graficamos la curva deformada
NpointsDraw=40;
Cdef=zeros(NpointsDraw,2);
%f=1/100;
u_new=linspace(U(1),U(end),NpointsDraw);
for i=1:NpointsDraw
    Cdef(i,:)=CurvePointNURBS(n,p,u_new(i),Pnew,U,W);
end

%%
%Esta parte calcula la intersección con la línea y=0
Inter=zeros(2,2);
flagder=0;
flagizq=0;
for i=1:NpointsDraw
    %Encontramos el primer punto a la izquierda con y<0
    if Cdef(i,2)<0&&Cdef(i,1)<0&&flagizq==0
        Inter(1,:)=Cdef(i,:);
        flagizq=1;
    %Encontramos el primer punto a la derecha con y<0
    elseif Cdef(i,2)<0&&Cdef(i,1)>0
        Inter(2,:)=Cdef(i,:);
    end
end
longInter=norm(Inter(1,2)-Inter(2,1)); %Calculamos la longitud de la intersección
xlswrite('Preescritos_D.xlsx',longInter,'Interseccion','A2') %Guardamos la longitud de la intersección
%%
% %En esta sección encontramos los nuevos puntos de control de la figura
% %deformada con algoritmo de optimización
[Pnueva,Wnuevo,Best,Sol,SolF,lastPl]=NewValues(Inter,U,P,W,p,Integral);
%De las soluciones optimizadas por el PSO elegimos la mejor, y copiamos sus respectivos valores en el archivo "PlotOther",
%luego corremos el archivo "Principal2", si ya no hay cambios en la línea de contacto, entonces asumimos que el problema
de
%contacto se resolvió, puesto que se encontró la deformación de los
%cuerpos.
```

NURBSParameters.m

```
function [U,P,W] = NURBSParameters( )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
r=3;
k=3;
p1=[-3 3];
%p2 lo desconocemos
p3=[r*cosd(260) r*sind(260)+k];
S1=[r*cosd(220) r*sind(220)+k];
h2_1=cosd(40);
s1=h2_1/(h2_1+1);
M1=(p3+p1)/2;
p2=(S1-(1-s1)*M1)/s1;

%p4 lo desconocemos
p5=[0 0];
S2=[r*cosd(265) r*sind(265)+k];
h2_2=cosd(5);
s2=h2_2/(h2_2+1);
M2=(p3+p5)/2;
p4=(S2-(1-s2)*M2)/s2;

%p6 lo desconocemos
p7=[r*cosd(280) r*sind(280)+k];
S3=[r*cosd(275) r*sind(275)+k];
h2_3=cosd(5);
s3=h2_3/(h2_3+1);
```

```
M3=(p5+p7)/2;
p6=(S3-(1-s3)*M3)/s3;

%p8 lo desconocemos
p9=[3 3];
S4=[r*cosd(320) r*sind(320)+k];
h2_4=cosd(40);
s4=h2_4/(h2_4+1);
M4=(p7+p9)/2;
p8=(S4-(1-s4)*M4)/s4;


%%Definimos el cuerpo de análisis%%

%Vector de nudos
U=[0 0 0 1 1 2 2 3 3 4 4 5 5 6 6 6];
%Puntos de control
P=[0 0;p6(1) p6(2);p7(1) p7(2);p8(1) p8(2);3 3;3 6;0 6;-3 6;-3 3;p2(1) p2(2);p3(1) p3(2);p4(1) p4(2);0 0];
%Pesos
W=[1 h2_3 1 h2_4 1 sqrt(2)/2 1 sqrt(2)/2 1 h2_1 1 h2_2 1];

end
```

NewValues.m

```
function [Pnew,W,Best,Sol,SolF,lastPl]=NewValues(Inter,U,P,W,p,Ior)
%Actualizamos los puntos de control con las coordenadas de las intersecciones y generamos la línea recta
P(11,:)=Inter(1,:);
P(3,:)=Inter(2,:);
P(12,:)=(P(1,:)+P(11,:))/2;
P(2,:)=(P(1,:)+P(3,:))/2;
W(12)=0;
W(2)=0;
[Pnew,Best,Sol,SolF,lastPl]=PSO(P,W,U,p,Ior);
DibujaSol(lastPl,U,W,p,P)
```

CurvePointNURBS.m

```
function [C]=CurvePointNURBS(n,p,u,P,U,W)
%Este programa calcula el valor de un punto de acuerdo al valor del
%parámetro u
%n=no. de funciones base = m-p (m es el número de elementos del vector
%nudos)
%p=orden de la curva (grado de la curva+1)
%u=parámetro a evaluar
%U=vector de nudos
%P=vector con los puntos de control
%w=Vector de pesos
%C=Punto de la curva de acuerdo al valor de u
WPoints=[P W'];
WPoints(:,1:2)=[WPoints(:,1).*WPoints(:,3) WPoints(:,2).*WPoints(:,3)];
span=FindSpan(n,p,u,U);
N=BasisFunc(span,u,p,U);
Cw=[0 0 0];
for i=1:p
    Cw=Cw+N(i)*WPoints(span-p+i,:);
end
C=Cw(1,1:2)./Cw(1,3);
```

Principal2.m

```
%%Programa principal para el análisis de un cuerpo usando el método
%%isogeométrico-elementos frontera (IGA-BEM)
clearvars
clc

%%Definimos el cuerpo de análisis%%
%Orden de la curva NURBS
p = 3;
%Vector de nudos, Puntos de control y Pesos optimizado
[P,W,U]=PlotOther();
%Tamaño del vector de nudos (m)
[~,m]=size(U);
%No. de funciones base
n=m-p;
%No. de puntos de control
[Cpoints,~]=size(P(:,1));
%Puntos para dibujar la curva NURBS
Npoints=22+2;
f=1/100;
u_aux=linspace(f,U(end)-f,Npoints-2);
u=zeros(1,Npoints);
u(1,1:11)=u_aux(1,1:11);
u(1,12)=2.99; %Este valor de nudo lo agregamos para que la fuerza quedé aplicada en (0,6)
u(1,13)=3.01;
u(1,14:end)=u_aux(1,12:end);
%u=u_aux;
%Inicializamos matrices
```

```
C=zeros(Npoints,2);
Cder=zeros(Npoints,2);
Ni=zeros(Npoints,Cpoints);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas, que mas
%adelante se volverán puntos de carga
for i=1:Npoints
    [C(i,:),Cder(i,:)]=DeriveNURBS_Point(n,p,u(i),P,U,W);
    %Guardamos los valores de las funciones base para posteriormente
    %encontrar los nuevos puntos de control
    [~,Ni(i,:)]=SaveNi(n,p,u(i),P,U,Cpoints,W);
end
plot(C(:,1),C(:,2),'r*')
hold on
%%
%%Establecemos las características del cuerpo
%Modulo de Young
E=2e11;
%E=1;
%Coeficiente de Poisson
v=.3;
%Modulo de corte
miu=E/(2*(1+v));
%%
%Establecemos las condiciones frontera, se tendrán que modificar las
%columnas de los xlsread para obtener adecuadamente los datos
%Desplazamientos (No. Nodo, Dirección (x=1,y=2),Cantidad)
DesPre=xlsread('Preescritos_D.xlsx','Desplazamientos','A2:C5');
%Tracciones
TracPre=xlsread('Preescritos_D.xlsx','Tracciones','A2:C5');
%Leemos la longitud de la intersección de la iteración anterior
longInter=xlsread('Preescritos_D.xlsx','Interseccion','A2');
%Leemos la tolerancia para comparar las longitudes de las intersecciones
tol=xlsread('Preescritos_D.xlsx','Tolerancia','A2');
%%
%%%Calculamos los rangos del parámetro del vector de nudos y las normales a
%%%los puntos que estamos analizando
Rango=[0 1;1 2;2 3;3 4];
Nor=zeros(Npoints,2);
for i=1:Npoints
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    Nor(i,1)=1*(Cder(i,2))/Jacobiano;
    Nor(i,2)=-1*(Cder(i,1))/Jacobiano;
end
%%
%%%%Calculo de Kernels
DMAX=0;
T=zeros(Npoints*2,Npoints*2);
D=zeros(Npoints*2,Npoints*2);
Derivada=zeros(Npoints,Npoints);
for i=1:Npoints
    XP=C(i,1);
    YP=C(i,2);
    IROW1=2*i-1;
    IROW2=IROW1+1;
    for j=1:Npoints
        if i==j
            continue
        else
            XQ=C(j,1);
            YQ=C(j,2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            if r>DMAX
                DMAX=r;
            end
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            %dx/dpsi=Cder(j,1) y dy/dpsi=Cder(j,2) (psi es la variable de
            %la NURBS)
            Jacobiano=sqrt(Cder(j,1)^2+Cder(j,2)^2);
            nx=1*(Cder(j,2))/Jacobiano;
            ny=-1*(Cder(j,1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            %Derivada(i,j)=drdn;%%
            ICOL1=2*j-1;
            ICOL2=ICOL1+1;
            %Calculamos kernels de desplazamiento
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IROW1,ICOL1)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IROW1,ICOL2)=FactorA*drdx*drdy;
            %Uyx
            D(IROW2,ICOL1)=FactorA*drdx*drdy;
            %Uyy
            D(IROW2,ICOL2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %Calculamos kernels de tracción
            %Factor=-1/r;
            Factor=-1/(4*pi*(1-v)*r);% Original
            %Txx
            T(IROW1,ICOL1)=Factor*drdn*((1-2*v)+2*(drdx^2));
            %Txy
            T(IROW1,ICOL2)=Factor*(2*drdx*drdy*drdn+(1-2*v)*(drdy*nx-drdx*ny));
            %Tyx
            T(IROW2,ICOL1)=Factor*(2*drdx*drdy*drdn-(1-2*v)*(drdy*nx-drdx*ny));
```

```
            %Tyy
            T(IROW2,ICOL2)=Factor*drdn*((1-2*v)+2*(drdy^2));
        end
    end
end
%%
%Calculamos la diagonal de la matriz T->Tij+Ci con el término de salto como
%en el artículo de Simpson (2012) y también calculamos la diagonal de la
%matriz U, haciendo que Q se aleje de P cuando coincidan en una razón
%+eps*algo
Cij=[.5 0;0 .5]; %Matriz de salto, solo aplica para la sección transversal de un círculo
for i=1:Npoints
    IRow=2*i-1;
    IRow2=IRow+1;
            %Diagonales, consideraciones cuerpo rígido
            %Txx
            T(IRow,IRow)=Cij(1,1); %T(IRow,IRow)-T(IRow,ICol);
            %Txy
            T(IRow,IRow2)=Cij(1,2);%T(IRow,IRow2)-T(IRow,ICol2);
            %Tyx
            T(IRow2,IRow)=Cij(2,1);%T(IRow,IRow2);
            %T(IRow2,IRow)=T(IRow2,IRow)-T(IRow,ICol);<-No funciona para la
            %deformación de un círculo
            %Tyy
            T(IRow2,IRow2)=Cij(2,2);
            %Diagonales de la matriz U, con r indeterminado
            XP=C(i,1);
            YP=C(i,2);
            xi=u(i)+eps*1e12; %Incrementamos el valor de Q
            [Cxi,Derxi]=DeriveNURBS_Point(n,p,xi,P,U,W);
            XQ=Cxi(1);
            YQ=Cxi(2);
            r=sqrt((XP-XQ)^2+(YP-YQ)^2);
            drdx=(XQ-XP)/r;
            drdy=(YQ-YP)/r;
            Jacobiano=sqrt(Derxi(1)^2+Derxi(2)^2);
            nx=1*(Derxi(2))/Jacobiano;
            ny=-1*(Derxi(1))/Jacobiano;
            drdn=drdx*nx+drdy*ny;
            FactorA=1/(8*pi*miu*(1-v));
            %Uxx
            D(IRow,IRow)=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
            %Uxy
            D(IRow,IRow2)=FactorA*drdx*drdy;
            %Uyx
            D(IRow2,IRow)=FactorA*drdx*drdy;
            %Uyy
            D(IRow2,IRow2)=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
            %D(IRow:IRow2,IRow:IRow2)
end
%Definimos el factor de escalamiento, ec. 4.63 libro de Becker
SCALEF=E/DMAX;
%%
%Calculamos la integral de supeficie, aplicando la regla de Simpson 1/3
%compuesta, el Jacobiano lo representamos con más número de puntos que con
%el cálculo anterior para aproximar mejor la forma del círculo
%Puntos para dibujar la curva NURBS
NpointJ=50;
uJ=linspace(1/NpointJ,U(end),NpointJ);
CJ=zeros(NpointJ,2);
Cder_J=zeros(NpointJ,2);
Jacob=zeros(NpointJ,1);
%Obtenemos las coordenadas para dibujar la curva y sus derivadas
for i=1:NpointJ
    [CJ(i,:),Cder_J(i,:)]=DeriveNURBS_Point(n,p,uJ(i),P,U,W);
    Jacob(i,1)=sqrt(Cder_J(i,1)^2+Cder_J(i,2)^2);
end
plot(CJ(:,1),CJ(:,2),'r');
npointJ=NpointJ-1;
b=U(end);
a=U(1);
h=(b-a)/npointJ;
%Integral del jacobiano de psi
Integral= h/3*(Jacob(1)+2*sum(Jacob(3:2:end-2))+4*sum(Jacob(2:2:end))+Jacob(end));
%Multiplicamos por el perímetro
D=D.*Integral;
T=T.*Integral;

%%
%%Aplicamos las condiciones frontera a los Kernels para formar la matriz A*
%%Y B* (4.50 y 4.58 Libro Becker)
[N_dis,~]=size(DesPre);
[N_trac,~]=size(TracPre);
TTracPres=zeros(Npoints*2,1);
%Colocamos las tracciones conocidas en el vector tracciones
for i=1:N_trac
    index=TracPre(i,1)*2-2+TracPre(i,2); %Ubicación de la tracción prescrita
    Jacobiano=sqrt(Cder(i,1)^2+Cder(i,2)^2);
    nx=1*(Cder(i,2))/Jacobiano;
    ny=-1*(Cder(i,1))/Jacobiano;
    if TracPre(i,2)==1
        TTracPres(index,1)=TracPre(i,3)*nx;
    else
        TTracPres(index,1)=TracPre(i,3)*ny;
```

```
    end
end
%Borramos las columnas y renglones correspondientes a desplazamientos conocidos
AuxTrac=TTracPres;
for I = length(DesPre(:,1)):-1:1
    GDL = DesPre(I,1)*2-2+DesPre(I,2);
    T(GDL,:)=[];
    T(:,GDL)=[];
    D(GDL,:)=[];
    D(:,GDL)=[];
    AuxTrac(GDL,:)=[];
end
%Resolvemos el sistema de ecuaciones
KJ=D/T;
UU=KJ*AuxTrac;
%Ordenamos los desplazamientos
UUNew=zeros(Npoints*2,1);
a=1;
for i=1:Npoints*2
    if a<=length(DesPre(:,1))
        index=DesPre(a,1)*2-2+DesPre(a,2);
    end
    if i==index
        UUNew(i,1)=DesPre(a,3);
        a=a+1;
    else
        UUNew(i,1)=UU(i-a+1);
    end
end
%%
%Obteniendo el cuerpo deformado con NURBS

%a)Sumamos los desplazamientos a los puntos que usamos como puntos de carga
AUX=UUNew(1:2:end);
AUX2=UUNew(2:2:end);
scale=1;
Cnew=C+[scale*UUNew(1:2:end) scale*UUNew(2:2:end)];
%plot(Cnew(:,1),Cnew(:,2),'k')
UUNewXY=[UUNew(1:2:end) UUNew(2:2:end)];
TracXY=[TTracPres(1:2:end) TTracPres(2:2:end)];
plot(Cnew(:,1),Cnew(:,2),'b--');
%b)Obtenemos la matriz inversa de Ni y lo multiplicamos por los nuevos
%datos de la curva deformada para obtener los puntos de control
Pnew=inv(Ni'*Ni)*Ni'*Cnew;
%c)Graficamos la curva deformada
NpointsDraw=40;
Cdef=zeros(NpointsDraw,2);
%f=1/100;
u_new=linspace(U(1),U(end),NpointsDraw);
for i=1:NpointsDraw
    Cdef(i,:)=CurvePointNURBS(n,p,u_new(i),Pnew,U,W);
end
%%
%Esta parte calcula la intersección con la línea y=0
Inter=zeros(2,2);
flagder=0;
flagizq=0;
for i=1:NpointsDraw
    %Encontramos el primer punto a la izquierda con y<0
    if Cdef(i,2)<0&&Cdef(i,1)<0&&flagizq==0
        Inter(1,:)=Cdef(i,:);
        flagizq=1;
    %Encontramos el primer punto a la derecha con y<0
    elseif Cdef(i,2)<0&&Cdef(i,1)>0
        Inter(2,:)=Cdef(i,:);
    end
end
%%
%Sección A
NewlongInter=norm(Inter(1,2)-Inter(2,1)); %Calculamos la longitud de la intersección (nueva)
diferencia=abs(NewlongInter-longInter);
if diferencia <= tol
    fprintf('La deformación optima ha sido encontrada')
else
    fprintf('Correr el algoritmo de optimización nuevamente, la deformación optima no se ha encontrado')
end
%->Si no hay cambios significativos (diferencia <= tol), entonces comente la
%sección B, y active la sección C, corra nuevamente el programa para
%encontrar los desplazamientos internos
%->Si diferencia >= tol, entonces comente la sección C, y active la sección B
%para correr el PSO nuevamente, cargue los nuevos puntos optimizados en
%"PlotOther", y revise nuevamente los resultados de la sección A. Repita
%este proceso hasta que no haya cambios significativos en la línea de
%intersección

%%
% %Sección B
% %En esta sección encontramos los nuevos puntos de control de la figura
% %deformada con algoritmo de optimización
% [Pnueva,Wnuevo,Best,Sol,SolF,lastPl]=NewValues(Inter,U,P,W,p,Integral);
% %De las soluciones optimizadas por el PSO elegimos la mejor, y copiamos sus respectivos valores en el archivo
"PlotOther",
% %luego corremos el archivo "Principal2", si ya no hay cambios en la línea de contacto, entonces asumimos que el problema
de
```

```
% %contacto se resolvió, puesto que se encontró la deformación de los
% %cuerpos.
% %Grafica la curva original (C), curva deformada (C+U) y curva deformada con nuevos
% %puntos de control (Cnew)
% plot(Cnew(:,1),Cnew(:,2),'k')
% plot(Pnew(:,1),Pnew(:,2),'*k')
%%
%Sección C
%Empezamos a calcular los puntos internos
%Coordenadas de los puntos internos
XINT=xlsread('Preescritos_D.xlsx','PInternos','A2:A14');
YINT=xlsread('Preescritos_D.xlsx','PInternos','B2:B14');
%Obtenemos el número de puntos internos
[r,~]=size(XINT);
%Inicializamos las variables internas
UXINT=zeros(r,1);
UYINT=zeros(r,1);
for i=1:r
    %Tomar cada nodo interno como el nodo de carga "NodeP"
    XP=XINT(i);
    YP=YINT(i);
    for j=1:Npoints
        %Tomar los nodos del cálculo original como el nodo de campo "NodeQ"
        XQ=C(j,1);
        YQ=C(j,2);
        r=sqrt((XP-XQ)^2+(YP-YQ)^2);
        drdx=(XQ-XP)/r;
        drdy=(YQ-YP)/r;
        %dx/dpsi=Cder(j,1) y dy/dpsi=Cder(j,2) (psi es la variable de
        %la NURBS)
        Jacobiano=sqrt(Cder(j,1)^2+Cder(j,2)^2);
        nx=1*(Cder(j,2))/Jacobiano;
        ny=-1*(Cder(j,1))/Jacobiano;
        %Derivada(i,j)=drdn;%%
        drdn=drdx*nx+drdy*ny;
        %Desplazamientos ya conocidos
        UX=UUNewXY(j,1);
        UY=UUNewXY(j,2);
        %Tracciones ya conocidas
        TR=TracXY(j,1);
        TZ=TracXY(j,2);
        %Calculamos kernels de desplazamiento
        FactorA=1/(8*pi*miu*(1-v));
        %Uxx
        UXX=FactorA*((3-4*v)*log(1/r)+(drdx)^2);
        %Uxy
        UXY=FactorA*drdx*drdy;
        %Uyx
        UYX=FactorA*drdx*drdy;
        %Uyy
        UYY=FactorA*((3-4*v)*log(1/r)+(drdy)^2);
        %Calculamos kernels de tracción
        %Factor=-1/r;
        Factor=-1/(4*pi*(1-v)*r);% Original
        %Txx
        TXX=Factor*drdn*((1-2*v)+2*(drdx^2));
        %Txy
        TXY=Factor*(2*drdx*drdy*drdn+(1-2*v)*(drdy*nx-drdx*ny));
        %Tyx
        TYX=Factor*(2*drdx*drdy*drdn-(1-2*v)*(drdy*nx-drdx*ny));
        %Tyy
        TYY=Factor*drdn*((1-2*v)+2*(drdy^2));
        %Caculamos los desplazamiento internos usando la ecuación integral
        %4.32 de Becker
        UXINT(i,1)=UXINT(i,1)+(-UX*TXX-UY*TXY+TR*UXX+TZ*UXY);
        UYINT(i,1)=UYINT(i,1)+(-UX*TYX-UY*TYY+TR*UYX+TZ*UYY);
        %A diferencia del libro de Becker, no incluimos ningún factor en
        %esta ecuación ("FACT")
    end
end
%%
%Guardamos datos para graficar en origin
xlswrite('Preescritos_D.xlsx',CJ,'PlotXY','A2');%Guardamos coordenadas del círculo
xlswrite('Preescritos_D.xlsx',UUNewXY(:,2),'Salida_UY','C2'); %Guardamos UY
xlswrite('Preescritos_D.xlsx',Cnew,'Salida_UY','A2'); %Guardamos coordenadas X y
xlswrite('Preescritos_D.xlsx',XINT,'Salida_UYint','A2'); %Guardamos coordenadas X punto interno
xlswrite('Preescritos_D.xlsx',YINT,'Salida_UYint','B2'); %Guardamos coordenadas Y punto interno
xlswrite('Preescritos_D.xlsx',UYINT,'Salida_UYint','C2'); %Guardamos desplazamientos Y internos
```

PlotOther.m

```
function[Plast,W,U]=PlotOther()
%W= Vector de pesos calculado
%U=Vector de nudos
%Best=Los puntos de control optimizados con PSO (los tomamos de acuerdo a
%la gráfica que seleccionamos como forma óptima)
%Inter=Coordenadas Y de la intersección
W=[1,0.996194698091746,1,0.766044443118978,1,0.707106781186548,1,0.707106781186548,1,0.766044443118978,1,0.996194698091746
,1];
U=[0 0 0 1 1 2 2 3 3 4 4 5 5 6 6 6];
Best=[3.34455263535777 0.690862725723739 3.40833362918675 2.69542243668422 3 5.30222340782314 5.19308298600570];
Inter=[-0.320950494716956,-0.00548782422300218;0.320537579641164,-0.00559264811597138];
```

```
Pnew=[0,0;0.262465990577772,-6.32645343114235e-16;0.520944533000791,0.045576740963376 1;3,0.482701106468160;3,3;3,6;0,6;-
3,6;-3,3;-3,0.482701106468160;-0.520944533000791,0.045576740963376 1;-0.262465990577772,-6.32645343114235e-16;0,0];
Pnew(11,:)=Inter(1,:);
Pnew(3,:)=Inter(2,:);
Pnew(12,:)=(Pnew(1,:)+Pnew(11,:))/2;
Pnew(2,:)=(Pnew(1,:)+Pnew(3,:))/2;
W(12)=0;
W(2)=0;
Pnew(4,1)=Best(1,1);    %X4
Pnew(4,2)=Best(1,2);    %Y4
Pnew(5,1)=Best(1,3);    %X5
Pnew(5,2)=Best(1,4);    %Y5
Pnew(6,1)=Best(1,5);    %X6
Pnew(6,2)=Best(1,6);    %Y6
Pnew(7,2)=Best(1,7);    %Y7
Pnew(8,1)=-Best(1,5);   %X8
Pnew(8,2)=Best(1,6);    %Y8
Pnew(9,1)=-Best(1,3);   %X9
Pnew(9,2)=Best(1,4);    %Y9
Pnew(10,1)=-Best(1,1);  %X10
Pnew(10,2)=Best(1,2);   %Y10
Plast=Pnew;
```

PSO.m

```
function [Pnew,Best,Sol,SolF,lastPl]=PSO(P,W,U,p,Ior)
%Empieza el algoritmo de optimización
%%% X4 Y4 X5 Y5 X6 Y6 Y7)
ub=[4  3  4  3  4  6  6];              %Bordes superiores (X4,Y4,X5,Y5,X6,Y6,Y7)
lb=[0  0  3  2  3  3  3];              %Bordes inferiores
Nswarm=25;                             %Cantidad de partículas
Nvar=7;
%No. de variables
Gen=600;                               %No. de generaciones
N=4;                                   %No. de intervalos
c1=0.8;
Cmax=1.62;
%Población inicial
[x]=InitialPopulation(Nswarm,Nvar,ub,lb);
[v]=InitialVelocity(Nswarm,Nvar,ub,lb,N);
[plocal,g,sol,bestva,funfree]=InitialEvaluatingS_UptadatingP_g(p,x,Nswarm,Nvar,P,W,U,Ior);
%Inicializando matrices
Sol=zeros(1,Gen);       %Mejores valores integrales por generación con restricciones
SolF=zeros(1,Gen);      %Mejores valores integrales por generación sin restricciones
Best=zeros(Gen,Nvar);   %Población con los mejores valores integrales por generación
Sol(1,1)=sol;
SolF(1,1)=funfree;
Best(1,:)=bestva;
p1=plocal;
ggraph(1,1)=g(1,1);
tic
for i=1:Gen-1
    %Actualizando v y x
    [v,x]=UpdateVandX(c1,Cmax,v,x,g,plocal,Nswarm,Nvar,ub,lb);
    %Evaluando S y actualizando plocal y g
    [plocal,g,sol,bestva,funfree]=EvaluatingS_UptadatingP_g(plocal,x,Nswarm,p,P,W,U,Ior,g);
    %Mejores valores
    ggraph(i+1,1)=g(1,1);
    Sol(1,i+1)=sol;
    SolF(1,i+1)=funfree;
    Best(i+1,:)=bestva;
    if i==Gen-1
        lastPl=plocal;
    end
end
toc
figure
subplot(1,2,1)
plot(p1(:,1),'ko','LineWidth',2)
%ylim([0 400])
xlabel('Particle','Color','r')
ylabel('Integral Value','Color','r')
title('1st Generation')
subplot(1,2,2)
plot(plocal(:,1),'bo','LineWidth',2)
%ylim([0 250])
xlabel('Particle','Color','r')
ylabel('Integral value','Color','r')
title('Last generation')
figure
subplot(1,2,1)
plot(Sol)
title('Best values PSO')
xlabel('Generations')
ylabel('Integral Value')
subplot(1,2,2)
plot(ggraph)
title('Global Best values PSO')
xlabel('Generations')
ylabel('Integral Value')
```

InitialPopulation.m

```
function[pop]=InitialPopulation(Npop,Nvar,ub,lb)
%Esta función genera la población inicial
%Npop=Tamaño de la población
%Nvar=Número de variables
%ub=Límites superiores de las variables
%lb=Límites inferiores de las variables

pop=rand(Npop,Nvar);           %Población con números aleatorios
%%Unnormalized matrix
for i=1:Npop
    for j=1:Nvar
       pop(i,j)=(ub(j)-lb(j))*pop(i,j)+lb(j); %Población con números entre los limites lb y ub
    end
end
```

InitialVelocity.m

```
function[v]=InitialVelocity(Nswarm,Nvar,ub,lb,N)
%Esta función genera la velocidad inicial de las partículas
%Npop=Tamaño de la población
%Nvar=Número de variables
%ub=Límites superiores de las variables
%lb=Límites inferiores de las variables
%N=Constante asociada a la velocidad
%Normalized matrix
v=rand(Nswarm,Nvar);                    %Velocidad aleatoria de la población entre uno y cero
%%Unnormalized matrix
for i=1:Nswarm
    for j=1:Nvar
        %Velocidad aleatoria de la población entre los límites establecidos
        v(i,j)=((ub(j)-lb(j))/N)*v(i,j);
    end
end
```

InitialEvaluatingS_UptadatingP_g.m

```
%p=orden de la curva
%pop=Población de variables candidatas
%Npop=Número de individuos en la población
%Nvar=Número de variables
%Pmod=Puntos de control original de la NURBS
%W=Pesos de la NURBS
%U=vector de nudos
%Ior=Integral del contorno original
%Tamaño del vector de nudos (m)
%Salida:
%plocal=mejor posición del individuo
%g=mejor posición global
%sol=mejor posición global (integral con restricciones)
[~,m]=size(U);
%No. de funciones base
n=m-p;
%Número de puntos para dibujar la curva
NpDraw=40;
%Parámetro para dibujar la curva
u_nueva=linspace(U(1),U(end),NpDraw);
%Inicializamos los puntos de la curva de prueba
Cder_J=zeros(NpDraw,2);
Jacob=zeros(NpDraw,1);
%Obtenemos las constantes para calcular el Jacobiano
npointJ=NpDraw-1;
b=U(end);
a=U(1);
h=(b-a)/npointJ;
%Inicializamos vectores
Integral=zeros(Npop,1); %Valor de la integral con restricciones
fitpop=zeros(Npop,1);  %Valor de la integral sin restricciones
plocal=zeros(Npop,Nvar+1); %Mejor valor de la integral por individuo de la población
for i=1:Npop
    Pnew=P;
    Pnew(4,1)=pop(i,1);    %X4
    Pnew(4,2)=pop(i,2);    %Y4
    Pnew(5,1)=pop(i,3);    %X5
    Pnew(5,2)=pop(i,4);    %Y5
    Pnew(6,1)=pop(i,5);    %X6
    Pnew(6,2)=pop(i,6);    %Y6
    Pnew(7,2)=pop(i,7);    %Y7
    Pnew(8,1)=-pop(i,5);   %X8
    Pnew(8,2)=pop(i,6);    %Y8
    Pnew(9,1)=-pop(i,3);   %X9
    Pnew(9,2)=pop(i,4);    %Y9
    Pnew(10,1)=-pop(i,1);  %X10
    Pnew(10,2)=pop(i,2);   %Y10
    %Encontramos el Jacobiano para saber si su perímetro sigue siendo el
    %mismo que el original
    for k=1:NpDraw
        [~,Cder_J(k,:)]=DeriveNURBS_Point(n,p,u_nueva(k),Pnew,U,W);
        Jacob(k,1)=sqrt(Cder_J(k,1)^2+Cder_J(k,2)^2);
```

```
    end
    %Integral del jacobiano de psi
    Integral(i)= h/3*(Jacob(1)+2*sum(Jacob(3:2:end-2))+4*sum(Jacob(2:2:end))+Jacob(end));
    %Restricción sobre conservación de área
    if Integral(i)==Ior
        kI=0;
    else
        kI=(Ior-Integral(i))/Ior;
    end
    fitpop(i)=Integral(i); %Valor de la integral obtenida sin restricciones
    if Integral(i)>Ior
        Integral(i)=Integral(i)+abs(kI*Integral(i)); %Integral obtenida con restricciones
    elseif Integral (i)<Ior
        Integral(i)=Integral(i)-(kI*Integral(i)); %Integral obtenida con restricciones
    end
    if Integral(i)<0
        Integral(i)=50;      %En caso que la penalización sea muy alta, y vuelva negativa el valor de la integral lo
hacemos muy grande
    end
    plocal(i,:)=[Integral(i) pop(i,:)];
end
%Obtenemos el mejor valor, ordenamos de acuerdo al absoluto de la resta entre valor original de la
%integral y la integral penalizada, entre más cercano sea a cero, mejor es
%el valor
[~,Index]=sort(abs(Ior-Integral),'ascend');
%Mejor valor de la integral en la población con restricciones
sol=Integral(Index(1));
%Población con el mejor valor de la integral
bestva=pop(Index(1),:);
%Mejor valor de la integral sin restricciones
funfree=fitpop(Index(1),1);
%Mejores valores globales
g=[sol bestva];
```

UpdateVandX.m

```
function[v,x]=UpdateVandX(c1,Cmax,v,x,g,plocal,Nswarm,Nvar,ub,lb)

for i=1:Nswarm
    for j=1:Nvar
        v(i,j)=c1*v(i,j)+Cmax*rand*(plocal(i,j+1)-x(i,j))+Cmax*rand*(g(j+1)-x(i,j));
        x(i,j)=x(i,j)+v(i,j);
        if x(i,j)>ub(j)
            v(i,j)=0;
            x(i,j)=ub(j);
        end
        if x(i,j)<lb(j)
            v(i,j)=0;
            x(i,j)=lb(j);
        end
    end
end
```

EvaluatingS_UptadatingP_g.m

```
function [plocal,g,sol,bestva,funfree]=EvaluatingS_UptadatingP_g(plocal,pop,Npop,p,P,W,U,Ior,g)
%%%Esta función selecciona a los individuos más aptos de la población%%%
%p=orden de la curva
%pop=Población de variables candidatas
%Npop=Número de individuos en la población
%Nvar=Número de variables
%P=Puntos de control original de la NURBS
%W=Pesos de la NURBS
%U=vector de nudos
%Ior=Integral del contorno original
%Tamaño del vector de nudos (m)
%Salida:
%plocal=mejor posición del individuo
%g=mejor posición global
%sol=mejor posición global (integral con restricciones)
[~,m]=size(U);
%No. de funciones base
n=m-p;
%Número de puntos para dibujar la curva
NpDraw=40;
%Parámetro para dibujar la curva
u_nueva=linspace(U(1),U(end),NpDraw);
%Inicializamos los puntos de la curva de prueba
Cder_J=zeros(NpDraw,2);
Jacob=zeros(NpDraw,1);
%Obtenemos las constantes para calcular el Jacobiano
npointJ=NpDraw-1;
b=U(end);
a=U(1);
h=(b-a)/npointJ;
%Inicializamos vectores
Integral=zeros(Npop,1); %Valor de la integral con restricciones
fitpop=zeros(Npop,1);  %Valor de la integral sin restricciones
%plocal=zeros(Npop,Nvar+1); %Mejor valor de la integral por individuo de la población
for i=1:Npop
```

```
Pnew=P;
    Pnew(4,1)=pop(i,1);    %X4
    Pnew(4,2)=pop(i,2);    %Y4
    Pnew(5,1)=pop(i,3);    %X5
    Pnew(5,2)=pop(i,4);    %Y5
    Pnew(6,1)=pop(i,5);    %X6
    Pnew(6,2)=pop(i,6);    %Y6
    Pnew(7,2)=pop(i,7);    %Y7
    Pnew(8,1)=-pop(i,5);   %X8
    Pnew(8,2)=pop(i,6);    %Y8
    Pnew(9,1)=-pop(i,3);   %X9
    Pnew(9,2)=pop(i,4);    %Y9
    Pnew(10,1)=-pop(i,1);  %X10
    Pnew(10,2)=pop(i,2);   %Y10
    %Encontramos el Jacobiano para saber si su perímetro sigue siendo el
    %mismo que el original
    for k=1:NpDraw
        [~,Cder_J(k,:)]=DeriveNURBS_Point(n,p,u_nueva(k),Pnew,U,W);
        Jacob(k,1)=sqrt(Cder_J(k,1)^2+Cder_J(k,2)^2);
    end
    %Integral del jacobiano de psi
    Integral(i)= h/3*(Jacob(1)+2*sum(Jacob(3:2:end-2))+4*sum(Jacob(2:2:end))+Jacob(end));
    %Restricción sobre conservación de área
    if Integral(i)==Ior
        kI=0;
    else
        kI=(Ior-Integral(i))/Ior;
    end
    fitpop(i)=Integral(i); %Valor de la integral obtenida sin restricciones
    if Integral(i)>Ior
        Integral(i)=Integral(i)+abs(kI*Integral(i)); %Integral obtenida con restricciones
    elseif Integral(i)<Ior
        Integral(i)=Integral(i)-(kI*Integral(i)); %Integral obtenida con restricciones
    end
    if Integral(i)<0
        Integral(i)=50;      %En caso que la penalización sea muy alta, y vuelva negativa el valor de la integral lo
hacemos muy grande
    end
    if abs(Ior-Integral(i))< abs(plocal(i,1)-Ior)
        plocal(i,:)=[Integral(i) pop(i,:)];
    end
end
%Obtenemos el mejor valor, ordenamos de acuerdo al absoluto de la resta entre valor original de la
%integral y la integral penalizada, entre más cercano sea a cero, mejor es
%el valor
% Integral
% fitpop
[~,Index]=sort(abs(Ior-Integral),'ascend');
%Mejor valor de la integral en la población con restricciones
sol=Integral(Index(1));
%Población con el mejor valor de la integral
bestva=pop(Index(1),:);
%Mejor valor de la integral sin restricciones
funfree=fitpop(Index(1),1);
%Actualizamos el mejor valor global
if abs(Ior-sol)< abs(Ior-g(1,1))
    g(1,:)=[sol bestva];
end
```

Apéndice B. Programa para dibujar un círculo

```
%%Programa para dibujar círculo%%
%Orden de la curva NURBS
p = 3;
%Vector de nudos
%U=[0 0 0 1 2 2 3 3 4 4 5 6 6 6];
U=[0 0 0 1 1 2 2 3 3 4 4 4];
%Puntos de control
P=[0 0;3 0;3 3;3 6;0 6;-3 6;-3 3;-3 0;0 0];
%Pesos
W=[1 sqrt(2)/2 1 sqrt(2)/2 1 sqrt(2)/2 1 sqrt(2)/2 1];
%Tamaño del vector de nudos (m)
[~,m]=size(U);
%No. de funciones base
n=m-p;
NpointJ=50;
uJ=linspace(U(1),U(end),NpointJ);
CJ=zeros(NpointJ,2);
Cder_J=zeros(NpointJ,2);
%Jacob=zeros(NpointJ,1);
figure('Color','w')
%Curva 1
for i=1:NpointJ
    [CJ(i,:),Cder_J(i,:)]=DeriveNURBS_Point(n,p,uJ(i),P,U,W);
    %Jacob(i,1)=sqrt(Cder_J(i,1)^2+Cder_J(i,2)^2);
end
%subplot(2,2,1)
plot(CJ(:,1),CJ(:,2),'k','LineWidth',2);
hold on
plot(P(:,1),P(:,2),'s',...
    'LineWidth',2,...
    'MarkerSize',10,...
    'MarkerEdgeColor',[0.5,0.5,0.5],...
    'MarkerFaceColor',[0.5,0.5,0.5])
axis([-4 4 -1 7])
legend('Curva NURBS','Puntos de Control','Location','northoutside','Orientation','horizontal')
```

El código de la función “DeriveNURBS_Point.m” se describió en el apéndice A.

Apéndice C. Programa *BEM* basado en el libro de Becker

El código del programa *BEM,* usado para validar la metodología de este trabajo de tesis, está basado en el capítulo 12 y apéndices del libro de Becker [53]. El diagrama de interacción del programa *BEM* principal con subprogramas (funciones) se muestra en Figura Apéndice 5.

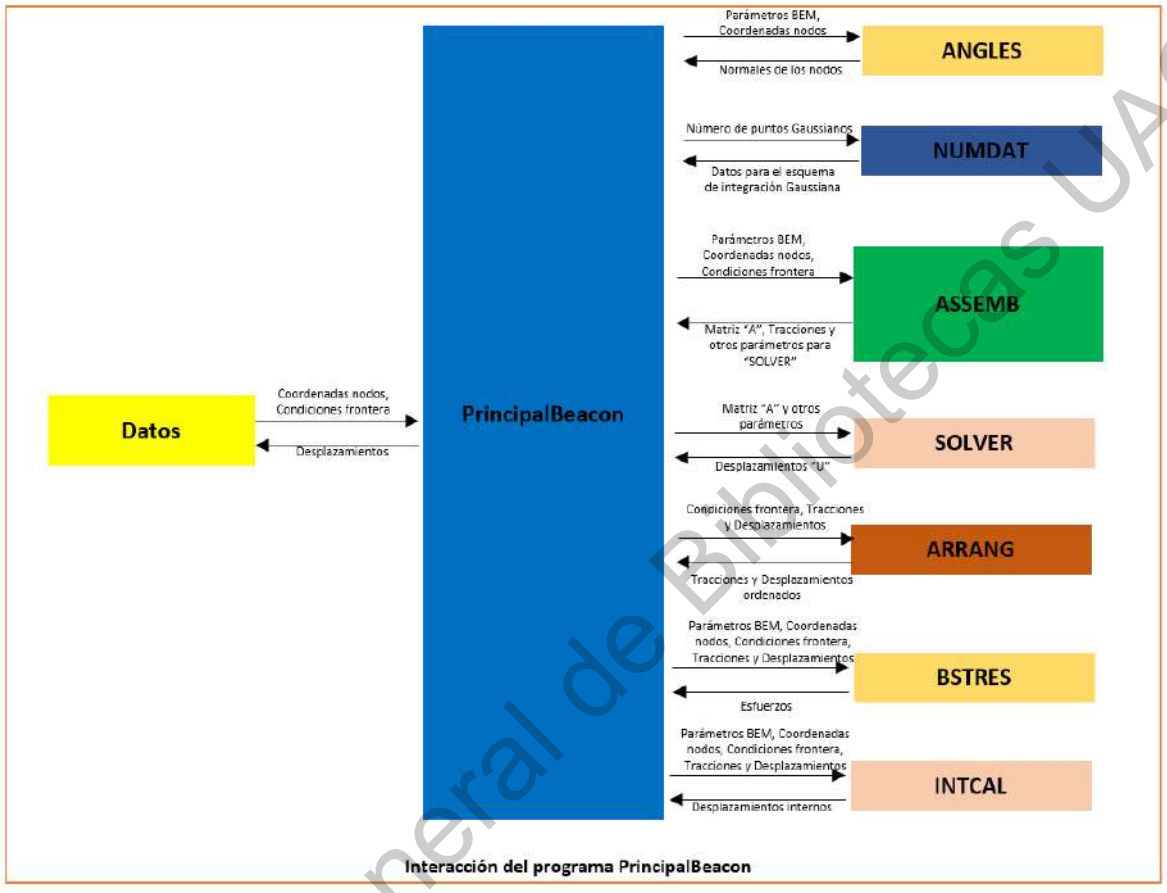


Figura Apéndice 5 Interacción del programa PrincipalBeacon, basado en el código de Becker

Código del programa de elementos frontera convencional

PrincipalBeacon.m

```
%%%Boundary Elements in Continuum Mechanics BEACON%%%
%%%Program based on the book "The Boundary Element Method in Engineering,
%%%A complete Course" by A.A. Becker. Programmed by Stephanie V. Camacho Gutierrez

MAXELE=100;                 %Maximum number of elements
MAXNOD=2*MAXELE;            %Maximum number of nodes
MAXINT=200;                 %Maximum number of internal nodes

ZETA=[-1 0 1];              %Local intrinsic coordinate parameter
NGAUSS=4;                   %Integration points for The Gaussian Quadrature, 4 or 6
IGEOM=2;                    %1=Axisymetric,2=2D Plane Strain, 3=2D Plane Stress
E=200000;                   %Young's modulus
XNU=.3;                     %Poisson's ratio

NNODES=71;                  %Nodes number
NELEMS=36;                  %Elements number
NBCU=3;                     %Prescribed displacements number
NBCS=1;                     %Prescribed stresses number
NINPTS=13;                   %Internal points number
A=zeros(2*NNODES,2*NNODES+1);
U=zeros(2*NNODES,1);
                %%%READ INPUT DATA%%%
%Read Nodal Coordinates%
X=xlsread('DataBEACON.xlsx','NodalCoor','B2:B72');
Y=xlsread('DataBEACON.xlsx','NodalCoor','C2:C72');
%Read Element Data
NODE1=xlsread('DataBEACON.xlsx','ElementData','B2:B37');
```

```
NODE2=xlsread('DataBEACON.xlsx','ElementData','C2:C37');
NODE3=xlsread('DataBEACON.xlsx','ElementData','D2:D37');
%Read Prescribed Displacements B.C.
ISTORU=xlsread('DataBEACON.xlsx','DispBC','A2:A4');
IDIREC=xlsread('DataBEACON.xlsx','DispBC','B2:B4');
PRESU=xlsread('DataBEACON.xlsx','DispBC','C2:E4');
%Read Stresses B.C.
PRESSXX=zeros(NBCS,3);
PRESSYY=zeros(NBCS,3);
PRESSXY=zeros(NBCS,3);
ISTORS=xlsread('DataBEACON.xlsx','StressBC','B1:D1');
ISTORS=unique(ISTORS)';
aux1=xlsread('DataBEACON.xlsx','StressBC','B2:D2');
aux2=xlsread('DataBEACON.xlsx','StressBC','B3:D3');
aux3=xlsread('DataBEACON.xlsx','StressBC','B4:D4');
for i=1:NBCS
    var=(i-1)*3+1;
    PRESSXX(i,:)=aux1(1,var:var+2);
    PRESSYY(i,:)=aux2(1,var:var+2);
    PRESSXY(i,:)=aux3(1,var:var+2);
end
%Read Internal Points
XINT=xlsread('DataBEACON.xlsx','InternalPoints','B2:B14');
YINT=xlsread('DataBEACON.xlsx','InternalPoints','C2:C14');
           %%%         Finish READ DATA         %%%

%Calculate the normal angles at each node
[ANGNOR]=ANGLES(NELEMS,NODE1,NODE2,NODE3,X,Y,ZETA);
%Calculate the numerical data for the gaussian integrations schemes and
%elliptic integrals
[AA1,BB1,CC1,DD1,XG,CG,XGL,CGL]=NUMDAT(NGAUSS);
%Assemble the solution matrix
[A,XTRAC,YTRAC,SCALEF,NEQN]=ASSEMB(NNODES,IGEOM,E,XNU,NBCS,NODE1,NODE2,NODE3,X,Y,ZETA,PRESSXX,PRESSXY,PRESSYY,NELEMS,NGAUS
S,A,ISTORS,AA1,BB1,CC1,DD1,XG,CG,XGL,CGL,NBCU,ISTORU,IDIREC,PRESU);
%Solve the system of linear equations
[U]=SOLVER(NEQN,A,U);
%Arrange the computed variables in suitable arrays
[XTRAC,YTRAC,U]=ARRANG(NBCU,ISTORU,NODE1,NODE2,NODE3,X,Y,IDIREC,U,SCALEF,XTRAC,YTRAC,PRESU);
%Calculate the boundary stresses
[SIG11,SIG12,SIG22,SIG33,SIGXX,SIGYY,SIGXY,SIGZZ,TTRAC,PTRAC,DISPT,DISPN]=BSTRES(NNODES,NELEMS,NODE1,NODE2,NODE3,X,Y,XTRAC
,YTRAC,U,E,XNU,ZETA,IGEOM);
%Calculate the internal displacements
[UXINT,UYINT]=INTCAL(XINT,YINT,E,XNU,NODE1,NODE2,NODE3,X,Y,NELEMS,U,XTRAC,YTRAC,NGAUSS,CG,XG);
%plot(X,Y);
hold on
UX=U(1:2:end);
UY=U(2:2:end);
U_XY=[U(1:2:end) U(2:2:end)];
% plot(UX+X,UY+Y);
% Xmov=X+UX;
% Ymov=Y+UY;
plot(X,Y,'k');
hold on
%plot(Xmov,Ymov,'b--');%X y desplazados después de aplicar la carga
plot(X,Y,'ko','LineWidth',2,...
    'MarkerSize',2,...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0,0,0])
ylim([-1 7])
xlim([-4 4]); %Para graficar los puntos de carga
XEl=[X(NODE1);X(NODE3)];
YEl=[Y(NODE1);Y(NODE3)];
plot(XEl,YEl,'kx','LineWidth',2)%,...
%       'MarkerSize',3,...
%     'MarkerEdgeColor','k',...
%     'MarkerFaceColor',[0,0,0])
hold off
```

ANGLES.m

```
function [ANGNOR]=ANGLES(NELEMS,NODE1,NODE2,NODE3,X,Y,ZETA)
TOLER=1E-20;
ANGNOR=zeros(NELEMS,3);
for IELEM=1:NELEMS
    [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
    for IC=1:3
        NODE=NORDER(IC);
        H=ZETA(IC);
        [SHAPF,SHAPD]=SHAPE(H);
        [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODE);
        if FLAG==1
            break;
        end
        if abs(XNORM)< TOLER
            XNORM=TOLER;
        end
        if abs(YNORM)< TOLER
            YNORM=TOLER;
        end
        THETA=atan(abs(YNORM)/abs(XNORM));
        if (XNORM<0) && (YNORM>0)
            ANG=pi-THETA;
```

```
        elseif (XNORM>0) && (YNORM<0)
            ANG=2*pi-THETA;
        elseif (XNORM<0) && (YNORM<0)
            ANG=pi+THETA;
        else
            ANG=THETA;
        end
        ANGNOR(IELEM,IC)=ANG;
    end
end
```

NUMDAT.m

```
function [AA1,BB1,CC1,DD1,XG,CG,XGL,CGL]=NUMDAT(NGAUSS)
%Store the elliptic coefficient data for Axisymmetric cases or the Gaussian
%Quadrature for 4 and 6 integration points
AA1(1)=0.096578619622;
AA1(2)=0.031559431627;
AA1(3)=0.023761224857;
AA1(4)=0.025962888452;
AA1(5)=0.0066398011146;
BB1(1)=0.12499929597;
BB1(2)=0.070148757782;
BB1(3)=0.044983875539;
BB1(4)=0.018751660276;
BB1(5)=0.0018472341632;
CC1(1)=0.44315287472;
CC1(2)=0.057566998484;
CC1(3)=0.031761145524;
CC1(4)=0.030662347457;
CC1(5)=0.0076529606032;
DD1(1)=0.24999920273;
DD1(2)=0.093564907830;
DD1(3)=0.054260524448;
DD1(4)=0.021836021169;
DD1(5)=0.0021247918284;

if NGAUSS==4
    XG(1)=-0.861136311594053;
    XG(2)=-0.339981043584856;
    CG(1)=0.347854845137454;
    CG(2)=0.652145154862546;
    XGL(1)=0.0414484801993832;
    XGL(2)=0.245274914320602;
    XGL(3)=0.556165453560276;
    XGL(4)=0.848982394532985;
    CGL(1)=0.383464068145135;
    CGL(2)=0.386875317774762;
    CGL(3)=0.190435126950142;
    CGL(4)=0.0392254871299598;
else
    XG(1)=-0.932469514203152;
    XG(2)=-0.661209386466264;
    XG(3)=-0.238619186083197;
    CG(1)=0.171324492379170;
    CG(2)=0.360761573048139;
    CG(3)=0.467913934572691;
    XGL(1)=0.0216340058441169;
    XGL(2)=0.129583399154951;
    XGL(3)=0.314020449914766;
    XGL(4)=0.538657217351802;
    XGL(5)=0.756915337377403;
    XGL(6)=0.922668851372120;
    CGL(1)=0.238763662578548;
    CGL(2)=0.308286573273947;
    CGL(3)=0.245317446563210;
    CGL(4)=0.142008756566477;
    CGL(5)=0.0554546223248863;
    CGL(6)=0.0101689586929323;
end
%Arrange the symmetric part of the ordinary gaussian quadrature
N=1+(NGAUSS/2);
for I=N:NGAUSS
    XG(I)=-XG(NGAUSS+1-I);
    CG(I)=CG(NGAUSS+1-I);
end
```

ASSEMB.m

```
function
[A,XTRAC,YTRAC,SCALEF,NEQN]=ASSEMB(NNODES,IGEOM,E,XNU,NBCS,NODE1,NODE2,NODE3,X,Y,ZETA,PRESSXX,PRESSXY,PRESSYY,NELEMS,NGAUS
S,A,ISTORS,AA1,BB1,CC1,DD1,XG,CG,XGL,CGL,NBCU,ISTORU,IDIREC,PRESU)
%Assemble the A and B Matrices

NEQN=2*NNODES;
%For plane stress cases, use "effective material properties", eq. 4.10
if IGEOM==3
    E=E*(1+2*XNU)/(1+XNU)^2;
    XNU=XNU/(1+XNU);
end
```

```
%Shear modulus, eq. 4.6
XMU=E/(2*(1+XNU));

%Transform the prescribed stresses into tractions, eq. 2.39
XTRAC=zeros(NELEMS,3);
YTRAC=zeros(NELEMS,3);
for IBC=1:NBCS  %17
    IELEM=ISTORS(IBC);
    [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
    for IC=1:3 %18
        NODE=NORDER(IC);
        H=ZETA(IC);
        [~,SHAPD]=SHAPE(H);
        [FLAG,XNORM,YNORM,~]=JACOBI(SHAPD,CORDX,CORDY,NODE);
        if FLAG==1
            error('THE JACOBIAN IS ZERO');
        end
        XTRAC(IELEM,IC)=PRESSXX(IBC,IC)*XNORM+PRESSXY(IBC,IC)*YNORM;
        YTRAC(IELEM,IC)=PRESSXY(IBC,IC)*XNORM+PRESSYY(IBC,IC)*YNORM;
    end%18
end%17

%Calculate the maximum distance between any two points, to be used for
%scaling purposes.
DMAX=0;
for II=1:NNODES %11
    for JJ=1:NNODES %12
        if JJ==II
            continue
        end
        DIST=sqrt((X(II)-X(JJ))^2+(Y(II)-Y(JJ))^2);
        if DIST>DMAX
            DMAX=DIST;
        end
    end %12
end %11

%Define a suitable scaling factor for interchanged elements of the A and B
%Matrices, eq. 4.63
SCALEF=E/DMAX;

%Initialize the matrices used to store the R.H.S.
RHS1=zeros(NNODES,1);
RHS2=zeros(NNODES,1);
BN1=zeros(NNODES,1);
BN2=zeros(NNODES,1);
%Initialize matrices NONSIN
GSHAPF=zeros(6,1);
GUXX=zeros(6,1);
GUXY=zeros(6,1);
GUYX=zeros(6,1);
GUYY=zeros(6,1);
GTXX=zeros(6,1);
GTXY=zeros(6,1);
GTYX=zeros(6,1);
GTYY=zeros(6,1);
GFACT=zeros(6,1);

%IMPORTANT SECTION: Take each node in turn as the load point "NODEP"
for NODEP=1:NNODES %2
    XP=X(NODEP);
    YP=Y(NODEP);
    %NODEP
    %Initialize the A and B matrices to start a new row
    A11=zeros(NNODES,1); %3
    A12=zeros(NNODES,1);
    A21=zeros(NNODES,1);
    A22=zeros(NNODES,1);
    F11=zeros(NNODES,1);
    F12=zeros(NNODES,1);
    F21=zeros(NNODES,1);
    F22=zeros(NNODES,1);
    B11=zeros(NELEMS,3); %8
    B12=zeros(NELEMS,3); %9
    B21=zeros(NELEMS,3);
    B22=zeros(NELEMS,3);

    %Take each element in turn as the FIELD ELEMENT "IELEMQ"
    for IELEMQ=1:NELEMS %4
        %IELEMQ
        [CORDX,CORDY,NORDER]=ARRAY3(IELEMQ,NODE1,NODE2,NODE3,X,Y);
        for IC=1:3 %5
            NODEQ=NORDER(IC);
            %For axisymmetric cases, check if "NODEP" is on Z-Axis
            RELXP=XP/DMAX;
            if IGEOM==1 && RELXP<1E-16

[A11,A12,A21,A22,B11,B12,B21,B22]=PZAXIS(NGAUSS,XG,NODEQ,CORDX,CORDY,A11,A12,A21,A22,B11,B12,B21,B22,IELEMQ,IC);
                continue
            end
            %Perform the numerical integrations
            ISELF=0;
            if NODEP==NODE1(IELEMQ) || NODEP==NODE2(IELEMQ) || NODEP==NODE3(IELEMQ)
                %fprintf('Entro caso nodos iguales')
```

```
            ISELF=1;
        end
        if ISELF==0
            %Nonsingular integrations

[A11,A12,A21,A22,B11,B12,B21,B22,GUXX,GUXY,GUYX,GUYY,GTXX,GTXY,GTYX,GTYY,GFACT]=NONSIN(NGAUSS,CORDX,CORDY,NODEQ,IC,IGEOM,X
MU,XNU,XP,YP,ISELF,AA1,BB1,CC1,DD1,CG,XG,A11,A12,A21,A22,B11,B12,B21,B22,IELEMQ,GSHAPF,GUXX,GUXY,GUYX,GUYY,GTXX,GTXY,GTYX,
GTYY,GFACT);
        else
            %Singular integrations, non-logarithmic part

[A11,A12,A21,A22,B11,B12,B21,B22]=SINGU1(NGAUSS,XG,NODEP,NODE1,NODE2,NODE3,IELEMQ,CORDX,CORDY,NODEQ,IGEOM,XMU,XNU,XP,YP,IS
ELF,AA1,BB1,CC1,DD1,CG,A11,A12,A21,A22,B11,B12,B21,B22,IC);
            %Singular integrations, logarithmic part

[A11,A12,A21,A22,B11,B12,B21,B22]=SINGU2(NGAUSS,NODEP,IELEMQ,NODE1,NODE2,NODE3,XGL,CGL,CORDX,CORDY,NODEQ,IGEOM,XMU,XNU,XP,
YP,ISELF,AA1,BB1,CC1,DD1,A11,A12,A21,A22,B11,B12,B21,B22,IC);
        end
      end %5
  end %4
  %So far, all coefficients of A and B matrices have been calculated,
  %except for the diagonal terms of A
  if IGEOM==1
      [A12,A22,A11,A21]=DIAGAX(NNODES,NODEP,A12,A22,A11,A21,NELEMS,NODE1,NODE2,NODE3,X,Y,E,BN1,BN2);
  else
      [A11,A12,A21,A22]=DIAG2D(NNODES,NODEP,A11,A12,A21,A22);
  end
  %Apply the boundary conditions to the matrix

[A11,A21,A12,A22,RHS1,RHS2]=BCAPPL(NBCU,NBCS,ISTORU,ISTORS,IDIREC,NODE1,NODE2,NODE3,X,Y,F11,F21,F12,F22,B11,B21,B12,B22,SC
ALEF,NNODES,PRESU,NODEP,A11,A21,A12,A22,RHS1,RHS2,XTRAC,YTRAC);

  %For axisymmetric cases, make sure that the coefficient of the A-matrix in
  %the R-Direction is non-zero. If NODEP is on the Z axis. This may occur
  %because of applying the B.C.
  if IGEOM==1 && RELXP<1E-15
      A11(NODEP)=1E-2;
  end
  %Store A11,A12,A21,A22 in a two-dimensional array to indicate them
  %positions in the overall solution matrix
  IROW1=2*NODEP-1;
  IROW2=IROW1+1;
  for I=1:NNODES %7
   ICOL1=2*I-1;
   ICOL2=ICOL1+1;
   A(IROW1,ICOL1)=A11(I);
   A(IROW1,ICOL2)=A12(I);
   A(IROW2,ICOL1)=A21(I);
   A(IROW2,ICOL2)=A22(I);
  end
  %Store the R.H.S. of the equations as an extension of the A matrix, to
  %be ready for the solver subprogram
  JMAX=NEQN+1;
  A(IROW1,JMAX)=RHS1(NODEP);
  A(IROW2,JMAX)=RHS2(NODEP);
end%2
```

ARRAY3.m

```
function [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y)
%Arrange the nodes of element into the array "NORDER"
NORDER=[NODE1(IELEM) NODE2(IELEM) NODE3(IELEM)];
%Arrange the coordinates of the nodes into coordinates arrays of size 3
CORDX=zeros(1,3);
CORDY=zeros(1,3);
for IC=1:3
    NODE=NORDER(IC);
    CORDX(IC)=X(NODE);
    CORDY(IC)=Y(NODE);
end
```

SHAPE.m

```
function[SHAPF,SHAPD]=SHAPE(H)
%Calculate shape functions and their derivatives, as functions of H
%Parameter

%Calculating quadratic shape functions, eq. 3.20
SHAPF(1)=0.5*H*(H-1);
SHAPF(2)=1-H^2;
SHAPF(3)=0.5*H*(H+1);

%Calculating derivatives of the shape functions, eq. 3.29
SHAPD(1)=H-0.5;
SHAPD(2)=-2.0*H;
SHAPD(3)=H+.5;
```

JACOBI.m

```
function [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODE)
%Calculate the Jacobean and the unit normal components

%Calculate the derivatives of the X and Y coordinates, Ec. 4.47
DXDH=0.0;
DYDH=0.0;
for IC=1:3
    DXDH=DXDH+SHAPD(IC)*CORDX(IC);
    DYDH=DYDH+SHAPD(IC)*CORDY(IC);
end

%Calculate the Jacobian, Eq. 4.41
XJACOB=sqrt(DXDH^2+DYDH^2);

if XJACOB < 1E-10
    disp('THE JACOBIAN IS ZERO AT NODE %d',NODE);
    FLAG=1;
else
    FLAG=0;
end

%Calculate the unit normal components, eq. 4.46
XNORM=DYDH/XJACOB;
YNORM=-DXDH/XJACOB;
```

NONSIN.m

```
function[A11,A12,A21,A22,B11,B12,B21,B22,GUXX,GUXY,GUYX,GUYY,GTXX,GTXY,GTYX,GTYY,GFACT]=NONSIN(NGAUSS,CORDX,CORDY,NODEQ,IC
,IGEOM,XMU,XNU,XP,YP,ISELF,AA1,BB1,CC1,DD1,CG,XG,A11,A12,A21,A22,B11,B12,B21,B22,IELEMQ,GSHAPF,GUXX,GUXY,GUYX,GUYY,GTXX,GT
XY,GTYX,GTYY,GFACT)
%To integrate the non-singular kernels using ordinary gaussian quadrature
%formulae


%Make the variable "H" equal to the ordinary gaussian quadrature coordinate
for IG=1:NGAUSS
    H=XG(IG);
    [SHAPF,SHAPD]=SHAPE(H);
    [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODEQ);
    if FLAG==1
        break;
    end
    %In order to speed up execution time, the kernels are calculated only
    %once for the first node of the element "IELEMQ" and then stored in new
    %arrays beginning with the letter "G"
    GSHAPF(IG)=SHAPF(IC);
    FN1=3;
    FN2=3;
    ILOG=3;
    if IC==1
        if IGEOM==1

[UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KERAXI(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,ISELF,ILOG,AA1,BB1,CC1,DD1,FN1,FN2,XNORM,YNORM
);
        else
            [UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KER2D(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,XNORM,YNORM,ISELF,ILOG,FN1,FN2);
        end
        %Store the kernels in new arrays, to avoid calculating them for
        %second and third nodes of the element
        GUXX(IG)=UXX;
        GUXY(IG)=UXY;
        GUYX(IG)=UYX;
        GUYY(IG)=UYY;
        GTXX(IG)=TXX;
        GTXY(IG)=TXY;
        GTYX(IG)=TYX;
        GTYY(IG)=TYY;
        if IGEOM==1
            GFACT(IG)=2*pi*XJACOB*CG(IG)*XQ;
        else
            GFACT(IG)=XJACOB*CG(IG);
        end
    end
    %Form the sum-matrix associated with this particular node
    FACT=GFACT(IG)*GSHAPF(IG);
    A11(NODEQ)=A11(NODEQ)+FACT*GTXX(IG);
    A12(NODEQ)=A12(NODEQ)+FACT*GTXY(IG);
    A21(NODEQ)=A21(NODEQ)+FACT*GTYX(IG);
    A22(NODEQ)=A22(NODEQ)+FACT*GTYY(IG);
    B11(IELEMQ,IC)=B11(IELEMQ,IC)+FACT*GUXX(IG);
    B12(IELEMQ,IC)=B12(IELEMQ,IC)+FACT*GUXY(IG);
    B21(IELEMQ,IC)=B21(IELEMQ,IC)+FACT*GUYX(IG);
    B22(IELEMQ,IC)=B22(IELEMQ,IC)+FACT*GUYY(IG);
end
```

SINGU1.m

```
function[A11,A12,A21,A22,B11,B12,B21,B22]=SINGU1(NGAUSS,XG,NODEP,NODE1,NODE2,NODE3,IELEMQ,CORDX,CORDY,NODEQ,IGEOM,XMU,XNU,
XP,YP,ISELF,AA1,BB1,CC1,DD1,CG,A11,A12,A21,A22,B11,B12,B21,B22,IC)
%Perform the singular integrations, when "NODEP" is inside the field element
%"IELEMQ". This sub-program deals with the non-logarithmic parts.

%Use ordinary gaussian quadrature, since the integrals are non-logarithmic
for IG=1:NGAUSS %1
    H=XG(IG);
    %If P is the first node of the element
    if NODEP==NODE1(IELEMQ)
        %fprintf('Entró caso NODEP==NODE1')
        FN1=(H-2)*CORDX(1)+2*(1-H)*CORDX(2)+H*CORDX(3);
        FN2=(H-2)*CORDY(1)+2*(1-H)*CORDY(2)+H*CORDY(3);
    end
    %If P is the second node of the element
    if NODEP==NODE2(IELEMQ)
        %fprintf('Entró caso NODEP==NODE2')
        FN1=0.5*(H-1)*CORDX(1)-H*CORDX(2)+0.5*(1+H)*CORDX(3);
        FN2=0.5*(H-1)*CORDY(1)-H*CORDY(2)+0.5*(1+H)*CORDY(3);
    end
    if NODEP==NODE3(IELEMQ)
        %fprintf('Entró caso NODEP==NODE3')
        FN1=-H*CORDX(1)+2*(1+H)*CORDX(2)-(H+2)*CORDX(3);
        FN2=-H*CORDY(1)+2*(1+H)*CORDY(2)-(H+2)*CORDY(3);
    end
    %Calculate the shape functions and the Jacobean
    [SHAPF,SHAPD]=SHAPE(H);
    [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODEQ);
    %Calculate the non-logarithmic parts of the kernels
    ILOG=0;
    if IGEOM==1

[UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KERAXI(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,ISELF,ILOG,AA1,BB1,CC1,DD1,FN1,FN2,XNORM,YNORM
);
    else
        [UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KER2D(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,XNORM,YNORM,ISELF,ILOG,FN1,FN2);
    end
        if IGEOM==1
        FACT=2*pi*XJACOB*CG(IG)*SHAPF(IC)*XQ;
    else
        FACT=XJACOB*CG(IG)*SHAPF(IC);
    end
    %Form the sub-matrix associated with this particular node
    A11(NODEQ)=A11(NODEQ)+FACT*TXX;
    A12(NODEQ)=A12(NODEQ)+FACT*TXY;
    A21(NODEQ)=A21(NODEQ)+FACT*TYX;
    A22(NODEQ)=A22(NODEQ)+FACT*TYY;
    B11(IELEMQ,IC)=B11(IELEMQ,IC)+FACT*UXX;
    B12(IELEMQ,IC)=B12(IELEMQ,IC)+FACT*UXY;
    B21(IELEMQ,IC)=B21(IELEMQ,IC)+FACT*UYX;
    B22(IELEMQ,IC)=B22(IELEMQ,IC)+FACT*UYY;
end
```

KER2D.m

```
function[UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KER2D(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,XNORM,YNORM,ISELF,ILOG,FN1,FN2)
%Calculate 2D Kernels

%Calculate the coordinates of the element "IELEMQ", eq. 4.39
% CORDX
% CORDY
XQ=SHAPF(1)*CORDX(1)+SHAPF(2)*CORDX(2)+SHAPF(3)*CORDX(3);
YQ=SHAPF(1)*CORDY(1)+SHAPF(2)*CORDY(2)+SHAPF(3)*CORDY(3);

%Calculate the displacement kernels, eq. 4.23
XA=1.0/(8*pi*XMU*(1-XNU));
RDIST=sqrt((XP-XQ)^2+(YP-YQ)^2);
XN1=3-4*XNU;
XN2=1-2*XNU;
DRDX=(XQ-XP)/RDIST;
DRDY=(YQ-YP)/RDIST;
DRDN=DRDX*XNORM+DRDY*YNORM;

%Disp. Kernels for "P" not inside the element "IELEMQ"
if ISELF == 0
    UXX=XA*(XN1*log(1/RDIST)+DRDX*DRDX);
    UXY=XA*DRDX*DRDY;
    UYX=UXY;
    UYY=XA*(XN1*log(1/RDIST)+DRDY*DRDY);
end

%Dips. Kernels for "P" inside "IELEMQ" (Non-logarithmic parts)
if ISELF==1 && ILOG==0
    S1=-0.5*log(FN1*FN1+FN2*FN2);
    UXX=XA*(XN1*S1+DRDX*DRDX);
    UXY=XA*DRDX*DRDY;
    UYX=UXY;
    UYY=XA*(XN1*S1+DRDY*DRDY);
```

```
end

%Disp. Kernels for "P" inside "IELEMQ" (Logarithmic parts)
if ISELF==1 && ILOG==1
    UXX=XA*XN1;
    UXY=0.0;
    UYX=UXY;
    UYY=XA*XN1;
end

%Calculate the traction kernels, eq. 4.27
%Note that there is no logarithmic singularity in 2D traction Kernels
XB=-1/(4*pi*RDIST*(1-XNU));
TXX=XB*DRDN*(XN2+2*DRDX*DRDX);
TXY=XB*(DRDN*2*DRDX*DRDY-XN2*(DRDX*YNORM-DRDY*XNORM));
TYX=XB*(DRDN*2*DRDX*DRDY-XN2*(DRDY*XNORM-DRDX*YNORM));
TYY=XB*DRDN*(XN2+2*DRDY*DRDY);
if ISELF ==1 && ILOG==1
    TXX=0;
    TXY=0;
    TYX=0;
    TYY=0;
End
```

SINGU2.m

```
function[A11,A12,A21,A22,B11,B12,B21,B22]=SINGU2(NGAUSS,NODEP,IELEMQ,NODE1,NODE2,NODE3,XGL,CGL,CORDX,CORDY,NODEQ,IGEOM,XMU
,XNU,XP,YP,ISELF,AA1,BB1,CC1,DD1,A11,A12,A21,A22,B11,B12,B21,B22,IC)
%PERFORM THE SINGULAR INTEGRATIONS, WHEN "NODEP" is INSIDE THE FIELD ELEMENT "IELEMQ".
%THIS SUBPROGRAM DEALS WITH THE LOGARITHMIC PARTS.

%Use the logarithmic gaussian quadrature coordinates
for IG=1:NGAUSS %1
    %If P is the first node of the element
    if NODEP==NODE1(IELEMQ)
        H=2*XGL(IG)-1.0;
        COEFF=2.0;
    end
    %If P is the second node of the element(First sub-element)
    if NODEP==NODE2(IELEMQ)
        ISUBEL=1;
        H=XGL(IG);
        COEFF=1.0;
    end
    %If P is the third node of the element
    if NODEP==NODE3(IELEMQ)
        H=1-2*XGL(IG);
        COEFF=2.0;
    end
    %Calculate the shape functions and the Jacobian
    [SHAPF,SHAPD]=SHAPE(H);
    [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODEQ);
    %Calculate the logarithmic part of the kernels
    ILOG=1;
    FN1=0;
    FN2=0;
    if IGEOM==1

[UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KERAXI(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,ISELF,ILOG,AA1,BB1,CC1,DD1,FN1,FN2,XNORM,YNORM
);
    else
        [UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KER2D(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,XNORM,YNORM,ISELF,ILOG,FN1,FN2);
    end
    if IGEOM==1
        FACTL=2*pi*COEFF*XJACOB*CGL(IG)*SHAPF(IC)*XQ;
    else
        FACTL=COEFF*XJACOB*CGL(IG)*SHAPF(IC);
    end
    %Form the sub-matrix associated with this particular node
    A11(NODEQ)=A11(NODEQ)+FACTL*TXX;
    A12(NODEQ)=A12(NODEQ)+FACTL*TXY;
    A21(NODEQ)=A21(NODEQ)+FACTL*TYX;
    A22(NODEQ)=A22(NODEQ)+FACTL*TYY;
    B11(IELEMQ,IC)=B11(IELEMQ,IC)+FACTL*UXX;
    B12(IELEMQ,IC)=B12(IELEMQ,IC)+FACTL*UXY;
    B21(IELEMQ,IC)=B21(IELEMQ,IC)+FACTL*UYX;
    B22(IELEMQ,IC)=B22(IELEMQ,IC)+FACTL*UYY;
    %Repeat the calculations for the second sub-element (When P is the
    %second node of the element)
    if NODEP==NODE2(IELEMQ)
        ISUBEL=ISUBEL+1;
    end
    if NODEP==NODE2(IELEMQ) && ISUBEL==2
        H=-XGL(IG);
        COEFF=1.0;
        [SHAPF,SHAPD]=SHAPE(H);
        [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODEQ);
        if IGEOM==1

[UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY]=KERAXI(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,ISELF,ILOG,AA1,BB1,CC1,DD1,FN1,FN2,XNORM,YNORM);
        else
            [UXX,UXY,UYX,UYY,TXX,TXY,TYX,TYY,XQ]=KER2D(SHAPF,CORDX,CORDY,XMU,XNU,XP,YP,XNORM,YNORM,ISELF,ILOG,FN1,FN2);
        end
        if IGEOM==1
```

```
            FACTL=2*pi*COEFF*XJACOB*CGL(IG)*SHAPF(IC)*XQ;
        else
            FACTL=COEFF*XJACOB*CGL(IG)*SHAPF(IC);
        end
        %Form the sub-matrix associated with this particular node
        A11(NODEQ)=A11(NODEQ)+FACTL*TXX;
        A12(NODEQ)=A12(NODEQ)+FACTL*TXY;
        A21(NODEQ)=A21(NODEQ)+FACTL*TYX;
        A22(NODEQ)=A22(NODEQ)+FACTL*TYY;
        B11(IELEMQ,IC)=B11(IELEMQ,IC)+FACTL*UXX;
        B12(IELEMQ,IC)=B12(IELEMQ,IC)+FACTL*UXY;
        B21(IELEMQ,IC)=B21(IELEMQ,IC)+FACTL*UYX;
        B22(IELEMQ,IC)=B22(IELEMQ,IC)+FACTL*UYY;
        ISUBEL=0;
    end
end
```

DIAG2D.m

```
function[A11,A12,A21,A22]=DIAG2D(NNODES,NODEP,A11,A12,A21,A22)
%Calculate the diagonal terms of the A-matrix in 2D Cases

%Using rigid displacements condition, eq. 4.57
ARR=0;
ARZ=0;
AZR=0;
AZZ=0;
for NODE=1:NNODES
    if NODE == NODEP
        continue;
    end
    ARR=ARR-A11(NODE);
    ARZ=ARZ-A12(NODE);
    AZR=AZR-A21(NODE);
    AZZ=AZZ-A22(NODE);
end
%Put the diagonal terms in their positions in the A-Matrix
A11(NODEP)=ARR;
A12(NODEP)=ARZ;
A21(NODEP)=AZR;
A22(NODEP)=AZZ;
```

BCAPPL.m

```
function[A11,A21,A12,A22,RHS1,RHS2]=BCAPPL(NBCU,NBCS,ISTORU,ISTORS,IDIREC,NODE1,NODE2,NODE3,X,Y,F11,F21,F12,F22,B11,B21,B1
2,B22,SCALEF,NNODES,PRESU,NODEP,A11,A21,A12,A22,RHS1,RHS2,XTRAC,YTRAC)
%Apply the B.C. to the matrices A and B, so that all the unknowns are on
%the L.H.S.

for IBC=1:NBCU %1
    IELEM=ISTORU(IBC);
    ID=IDIREC(IBC);
    [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
    for IC=1:3 %2
        NODE=NORDER(IC);
        %Note that the relevant B-Coefficients must be multiplied by the
        %scaling factor
        if ID==1
            F11(NODE)=F11(NODE)-B11(IELEM,IC)*SCALEF;
            F21(NODE)=F21(NODE)-B21(IELEM,IC)*SCALEF;
        elseif ID==2
            F12(NODE)=F12(NODE)-B12(IELEM,IC)*SCALEF;
            F22(NODE)=F22(NODE)-B22(IELEM,IC)*SCALEF;
        end
    end%2
end%1

%Initialize the array NCHECK, which is used to make sure that common nodes
%between two elements are not treated twice.
NCHECK=zeros(NNODES,1); %3

%Now, multiply the relevant A-coefficient by the prescribed disp. and move
%them to the R.H.S. (and reverse their sign)

for IBC=1:NBCU %4
    IELEM=ISTORU(IBC);
    ID=IDIREC(IBC);
    [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
    for IC=1:3 %5
        NODE=NORDER(IC);
        DISP=PRESU(IBC,IC);
        if NCHECK(NODE)==IDIREC(IBC)
            continue
        end
        if ID==1
            RHS1(NODEP)=RHS1(NODEP)-A11(NODE)*DISP;
            RHS2(NODEP)=RHS2(NODEP)-A21(NODE)*DISP;
        elseif ID==2
            RHS1(NODEP)=RHS1(NODEP)-A12(NODE)*DISP;
            RHS2(NODEP)=RHS2(NODEP)-A22(NODE)*DISP;
```

```
        end
        %Now, make the relevant B-Coefficient (now saved ub arrays "F"
        %occupied the positions of the relevant A-Coefficients.
        if ID==1
            A11(NODE)=F11(NODE);
            A21(NODE)=F21(NODE);
        elseif ID==2
            A12(NODE)=F12(NODE);
            A22(NODE)=F22(NODE);
        end
        NCHECK(NODE)=IDIREC(IBC);
    end %5
end%4
%Consider the prescribed Stress B.C.
%Multiply the B-Coefficients by the prescribed traction and then add them
%to the R.H.S.
for IBC=1:NBCS %6
    IELEM=ISTORS(IBC);
    [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
    for IC=1:3 %7
        NODE=NORDER(IC);
        RHS1(NODEP)=RHS1(NODEP)+B11(IELEM,IC)*XTRAC(IELEM,IC)+B12(IELEM,IC)*YTRAC(IELEM,IC);
        RHS2(NODEP)=RHS2(NODEP)+B21(IELEM,IC)*XTRAC(IELEM,IC)+B22(IELEM,IC)*YTRAC(IELEM,IC);
    end%7
end%6
```

SOLVER.m

```
function[U]=SOLVER(NEQN,A,U)
%To solve the simultaneous linear equations using gaussian elimination with
%partial pivoting. The right-hand side of the equations have been saved as
%an extension of [A]

%Calculate the mean value of the coefficients
AMEAN=0;
for IROW=1:NEQN
    for ICOL=1:NEQN
        AMEAN=AMEAN+abs(A(IROW,ICOL));
    end
end
AMEAN=AMEAN/(NEQN*NEQN);

%Perform partial pivoting by searching the leading column of [A]
JMAX=NEQN+1;
for IEQN=1:NEQN-1 %3
    IMIN=IEQN+1;
    IMAX=IEQN;
    for I=IMIN:NEQN %4
        if abs(A(I,IEQN))>abs(A(IMAX,IEQN))
            IMAX=I;
        end
    end %4
    if IMAX ~= IEQN
        for J=IEQN:JMAX %6
            AA=A(IEQN,J);
            A(IEQN,J)=A(IMAX,J);
            A(IMAX,J)=AA;
        end %6
    end
    %Eliminate X(IEQN) from the remaining equations, avoiding division by
    %zero (if the pivotal element is zero)
    if abs(A(IEQN,IEQN)/AMEAN)<1E-8 %5
        error('Error, the solution matrix has a zero determinant')
    else
        for I=IMIN:NEQN %7
            FACT=A(I,IEQN)/A(IEQN,IEQN);
            for J=IMIN:JMAX %8
                A(I,J)=A(I,J)-FACT*A(IEQN,J);
            end %8
        end %7
        %fprintf('Entró en caso B2,%d',IEQN)
    end
end %3
%Use back substitution, avoiding division by zero
if abs(A(NEQN,NEQN)/AMEAN)<1E-8
    error('Error, the solution matrix has a zero determinant')
else
    U(NEQN)=A(NEQN,JMAX)/A(NEQN,NEQN);
    for L=2:NEQN  %9
        I=NEQN+1-L;
        SUM=A(I,JMAX);
        IP1=I+1;
        for J=IP1:NEQN %10
            SUM=SUM-A(I,J)*U(J);
        end
        U(I)=SUM/A(I,I);
    end %9
end
```

ARRANG.m

```
function[XTRAC,YTRAC,U]=ARRANG(NBCU,ISTORU,NODE1,NODE2,NODE3,X,Y,IDIREC,U,SCALEF,XTRAC,YTRAC,PRESU)
%To arrange the computed variables (the unknowns) into displacements and
%tractions

%For elements with prescribed disp. B.C. the computed unknown is TRACTION
if NBCU==0
    error('Missing prescribed displacements B.C.')
else
    for IBC=1:NBCU %1
        IELEM=ISTORU(IBC);
        [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
        for IC=1:3 %2
            NODE=NORDER(IC);
            IROW1=2*NODE-1;
            IROW2=IROW1+1;
            %Note that the computed disp must be re-scaled, eq. 4.62
            if IDIREC(IBC)==1
                XTRAC(IELEM,IC)=U(IROW1)*SCALEF;
            else
                YTRAC(IELEM,IC)=U(IROW2)*SCALEF;
            end
        end %2
    end%1
    %Now, insert the prescribed disp. into their relevant position in the
    %array "U"
    for IBC=1:NBCU %3
        IELEM=ISTORU(IBC);
        [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
        for IC=1:3 %4
            NODE=NORDER(IC);
            IROW=2*(NODE-1)+IDIREC(IBC);
            U(IROW)=PRESU(IBC,IC);
        end %4
    end%3
```

BSTRES.m

```
function[SIG11,SIG12,SIG22,SIG33,SIGXX,SIGYY,SIGXY,SIGZZ,TTRAC,PTRAC,DISPT,DISPN]=BSTRES(NNODES,NELEMS,NODE1,NODE2,NODE3,X
,Y,XTRAC,YTRAC,U,E,XNU,ZETA,IGEOM)
%Calculate the boundary stresses from the already computed values of
%displacements and tractions

%Initialize the stress arrays
SIG11=zeros(NNODES,1); %4
SIG12=zeros(NNODES,1);
SIG22=zeros(NNODES,1);
SIG33=zeros(NNODES,1);
SIGXX=zeros(NNODES,1);
SIGYY=zeros(NNODES,1);
SIGXY=zeros(NNODES,1);
SIGZZ=zeros(NNODES,1);
NCHECK=zeros(NNODES,1);%4

TTRAC=zeros(NELEMS,3);
PTRAC=zeros(NELEMS,3);

DISPT=zeros(NNODES,1);
DISPN=zeros(NNODES,1);
%Start the loop for the elements
for IELEM=1:NELEMS %1
     [CORDX,CORDY,NORDER]=ARRAY3(IELEM,NODE1,NODE2,NODE3,X,Y);
        for IC=1:3 %2
            NODE=NORDER(IC);
            IROW1=2*NODE-1;
            %Calculate the unit tangential vector components, eq. 4.44
            H=ZETA(IC);
            [SHAPF,SHAPD]=SHAPE(H);
            [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODE);
            TANG1=-YNORM;
            TANG2=XNORM;
            %Now, transform the tractions from global to local coordinates,
            %using the unit normal components, eq. 4.67
            TTRAC(IELEM,IC)=YTRAC(IELEM,IC)*XNORM-XTRAC(IELEM,IC)*YNORM;
            PTRAC(IELEM,IC)=XTRAC(IELEM,IC)*XNORM+YTRAC(IELEM,IC)*YNORM;
            %Similarly, transform the disp. from global to local
            %coordinates
            DISPT(NODE)=U(2*NODE)*XNORM-U(2*NODE-1)*YNORM;
            DISPN(NODE)=U(2*NODE-1)*XNORM+U(2*NODE)*YNORM;
            %Calculate the strains in the local directions, eq. 4.66
            XX1=0;
            XX2=0;
            for IIC=1:3 %3
                NN=NORDER(IIC);
                KROW1=2*NN-1;
                KROW2=KROW1+1;
                XX1=XX1+SHAPD(IIC)*U(KROW1);
                XX2=XX2+SHAPD(IIC)*U(KROW2);
```

```
            end %3
            E11=(XX1*TANG1+XX2*TANG2)/XJACOB;
            %For axisymmetric cases, calculate the hoop strain "EE3". For
            %2D geometries, "E33=0" ever for plane stress, since E and NU
            %have already modified.
            E33=0;
            if IGEOM==1 && X(NODE)>1E-20
                E33=U(IROW1)/X(NODE);
            end
            %Using Hooke's law, calculate local stresses, eq. 4.68
            FACT=XNU*E*E33/(1-XNU^2);
            S11=FACT+(E*E11/(1+XNU)+XNU*PTRAC(IELEM,IC))/(1-XNU);
            S12=TTRAC(IELEM,IC);
            S22=PTRAC(IELEM,IC);
            S33=E*E33+XNU*(S11+S22);
            %Using standard transformation matrix, calculate global
            %stresses, eq. 4.70
            SRR=S11*YNORM^2+S22*XNORM^2-2*S12*XNORM*YNORM;
            SZZ=S11*XNORM^2+S22*YNORM^2+2*S12*XNORM*YNORM;
            SRZ=(S22-S11)*XNORM*YNORM+S12*(XNORM^2-YNORM^2);
            if IGEOM==1 && X(NODE)<1E-20
                S33=SRR;
            end
            if IGEOM==3
                S33=0;
            end
            SHOOP=S33;
            %To produce nodal stresses, average the stress values on the
            %common nodes between two elements

            %Note: This averaging of stress values can be avoided,
            %resulting in discontinuous values of stress at corners
            FACT=1;
            if NCHECK(NODE)==1
                FACT=2;
            end
            SIG11(NODE)=(SIG11(NODE)+S11)/FACT;
            SIG22(NODE)=(SIG22(NODE)+S22)/FACT;
            SIG12(NODE)=(SIG12(NODE)+S12)/FACT;
            SIG33(NODE)=(SIG33(NODE)+S33)/FACT;
            SIGXX(NODE)=(SIGXX(NODE)+SRR)/FACT;
            SIGYY(NODE)=(SIGYY(NODE)+SZZ)/FACT;
            SIGXY(NODE)=(SIGXY(NODE)+SRZ)/FACT;
            SIGZZ(NODE)=(SIGZZ(NODE)+SHOOP)/FACT;
            NCHECK(NODE)=1;
        end
end
```

INTCAL.m

```
function [UXINT,UYINT]=INTCAL(XINT,YINT,E,XNU,NODE1,NODE2,NODE3,X,Y,NELEMS,U,XTRAC,YTRAC,NGAUSS,CG,XG)
%To calculate the internal displacement
[r,~]=size(XINT);
%Initialize the arrays of the internal variables
UXINT=zeros(r,1);
UYINT=zeros(r,1);
%Shear modulus, eq. 4.6
XMU=E/(2*(1+XNU));
for i=1:r
    %Take each internal point in turn as the load point "NodeP"
    XP=XINT(i);
    YP=YINT(i);
    Toler=1e-20;
    DIFF=0.5-XNU;
    if DIFF<Toler
        XNU=0.5-Toler;
    end
    %Take each element on the boundary in turn as the field element "IELEMQ"
    for IELEMQ=1:NELEMS
        [CORDX,CORDY,NORDER]=ARRAY3(IELEMQ,NODE1,NODE2,NODE3,X,Y);
        for IC=1:3
            NODEQ=NORDER(IC);
            IROW1=2*NODEQ-1;
            IROW2=IROW1+1;
            UX=U(IROW1,1);
            UY=U(IROW2,1);
            TR=XTRAC(IELEMQ,IC);
            TZ=YTRAC(IELEMQ,IC);
            for IG=1:NGAUSS
                H=XG(IG);
                [SHAPF,SHAPD]=SHAPE(H);
                [FLAG,XNORM,YNORM,XJACOB]=JACOBI(SHAPD,CORDX,CORDY,NODEQ);
                if FLAG==1
                    error('THE JACOBIAN IS ZERO');
                end
                %Calculate the coordinates of the element "IELEMQ", eq. 4.39
                XQ=SHAPF(1)*CORDX(1)+SHAPF(2)*CORDX(2)+SHAPF(3)*CORDX(3);
                YQ=SHAPF(1)*CORDY(1)+SHAPF(2)*CORDY(2)+SHAPF(3)*CORDY(3);
                %Calculate the displacement kernels, eq. 4.23
                XA=1.0/(8*pi*XMU*(1-XNU));
                RDIST=sqrt((XP-XQ)^2+(YP-YQ)^2);
                XN1=3-4*XNU;
```

```
                XN2=1-2*XNU;
                DRDX=(XQ-XP)/RDIST;
                DRDY=(YQ-YP)/RDIST;
                DRDN=DRDX*XNORM+DRDY*YNORM;
                %Displacement Kernels
                UXX=XA*(XN1*log(1/RDIST)+DRDX*DRDX);
                UXY=XA*DRDX*DRDY;
                UYX=UXY;
                UYY=XA*(XN1*log(1/RDIST)+DRDY*DRDY);
                %Traction Kernels
                XB=-1/(4*pi*RDIST*(1-XNU));
                TXX=XB*DRDN*(XN2+2*DRDX*DRDX);
                TXY=XB*(DRDN*2*DRDX*DRDY-XN2*(DRDX*YNORM-DRDY*XNORM));
                TYX=XB*(DRDN*2*DRDX*DRDY-XN2*(DRDY*XNORM-DRDX*YNORM));
                TYY=XB*DRDN*(XN2+2*DRDY*DRDY);
                FACT=XJACOB*CG(IG)*SHAPF(IC);
                %Calculate the internal displacements using the B.I.E. of
                %EQ. 4.32
                UXINT(i,1)=UXINT(i,1)+FACT*(-UX*TXX-UY*TXY+TR*UXX+TZ*UXY);
                UYINT(i,1)=UYINT(i,1)+FACT*(-UX*TYX-UY*TYY+TR*UYX+TZ*UYY);
            end
        end
    end
end
```

Apéndice D. Publicaciones

- Congresos Nacionales:
    1. **Metaheuristic algorithms: Review and Comparison.** CONIIN 2017
    2. **Optimización del diseño preliminar de una transmisión mecánica mediante cúmulo de partículas.** SOMIIM 2017
       http://somim.org.mx/memorias/memorias2017/articulos/A1_190.pdf
       ISSN 2448-5551
    3. **Análisis elastoestático mediante el método isogeométrico de elementos frontera.** SOMIIM 2018
       http://somim.org.mx/memorias/memorias2018/articulos/A3_182.pdf
       ISSN 2448-5551
- Congreso Internacional/ Capítulo de libro:
    4. **A plane strain problem solved by the isogeometric boundary element method.** IFFToM 2019/ Springer Nature
       https://doi.org/10.1007/978-3-030-20131-9_267
- Revista Indexada:
    5. **Optimization of Excitation Frequencies of a Gearbox Using Algorithms Inspired by Nature.** JVET, 2019.
       https://doi.org/10.1007/s42417-019-00149-6
    6. **An application of Isogeometric Analysis and Boundary Integral Element Method for Solving Nonlinear Contact Problems.** Applied Sciences, 2020.
       https://doi.org/10.3390/app10072345