UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – INSTRUMENTACIÓN

DISEÑO DE UNA CAVIDAD LÁSER PARA PULSOS DE FEMTOSEGUNDOS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
JOSÉ AGUSTÍN MORENO LARIOS

TUTOR PRINCIPAL
MARTHA ROSETE AGUILAR, CENTRO DE CIENCIAS APLICADAS Y
DESARROLLO TECNOLÓGICO

MÉXICO, D. F. JULIO 2016

JURADO ASIGNADO:

Presidente:	Dr. Garduño Mejía Jesús
Secretario:	Dr. Qureshi Naser
Vocal:	Dra. Rosete Aguilar Martha
1er. Suplente:	Dr. Hernández Cordero Juan A.
2do. Suplente:	Dr. Bruce Davidson Neil Charles

Centro de Ciencias Aplicadas y Desarrollo Tecnológico

TUTOR DE TESIS:

Dra. Rosete Aguilar Martha

Rosete Aguilar Martha

FIRMA

Agradecimientos

Agradezco el apoyo incondicional de mi familia y amigos. La tarea fue ardua y gracias a sus palabras de aliento y consejo he podido continuar.

Agradezco a mi tutora, la Dra. Martha Rosete Aguilar, por la orientación que necesité para realizar mi trabajo.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por su patrocinio a lo largo de mis estudios de maestría; a la Coordinación de Estudios de Posgrado (CEP) a través del Programa de Apoyo de Estudiantes de Posgrado la Universidad Nacional Autónoma de México (UNAM) por los fondos otorgados para la presentación del trabajo "Comparison of different Kerr-lens mode locking laser design techniques" en el congreso "SPIE Photonics Europe", realizado en la ciudad de Bruselas, Bélgica del tres al siete de Abril del 2016; y a la Dirección General de Asuntos del Personal Académico (DGAPA) de la UNAM por su patrocinio en el proyecto "Diseño y Construcción de Fuentes Láser de Femtosegundos Sintonizables de Alta Eficiencia", con el número PAPIIT-IG100615.

Introducción

Los pulsos láser de femtosegundos son ampliamente utilizados en investigación y desarrollo tecnológico. Estos pulsos permiten la observación e interacción de luz láser intensa con la materia en un lapso de varios femtoseguntos (1×10^{-15} [s]) y se caracterizan por una alta concentración de energía en un intervalo temporal muy corto así como por que su potencia pico puede ser extremadamente grande, del orden de cientos de [kW] [1].

En las décadas pasadas se han publicado diversos artículos sobre el uso y el modelado de medios ópticos no lineales para el desarrollo de láseres de pulsos ultracortos. La primera realización de un láser pulsado por amarre de modos de efecto Kerr (Kerr Lens Mode-Locking, KLM) utilizando un cristal de Titanio:Zafiro generó interés por desarrollar modelos y técnicas que permitan construir una cavidad láser de pulsos ultracortos eficiente [2].

Para emisión continua se analiza la cavidad utilizando óptica paraxial y las matrices de propagación de rayos ABCD [3]. La cavidad es modelada completamente por el producto de las matrices de propagación de rayos de los componentes ópticos encontrados en un viaje redondo de un haz a través de la cavidad iniciando desde una superficie de referencia (usualmente uno de los espejos planos - totalmente reflejante o parcialmente reflejante) [4].

Para lograr la emisión de pulsos ultracortos en una cavidad bombeada por un láser de emisión continua se necesita de un elemento intracavidad cuya transmisividad o su fase sea dependiente de la intensidad del haz intracavidad. Esto se alcanza por la absorción saturable en sólidos o tintes, o por otros efectos ópticos no lineales como el efecto Kerr óptico. Algunos materiales presentan autoenfocamiento no lineal, el cual combinado con una apertura, forman una lente cuya potencia de enfoque será dependiente de la intensidad del haz intracavidad, la cual puede utilizarse para crear condiciones favorables para la emisión pulsada. Los cristales de Titanio:Zafiro (Ti:Zafiro) manifiestan dichas propiedades.

En este trabajo se presenta el desarrollo teórico para el diseño de una cavidad láser de femtosegundos

En el Capítulo 1 se discuten las matrices de propagación de rayos, las cuales permiten modelar la propagación de rayos paraxiales a través de diversos sistemas y son utilizadas para describir la cavidad láser.

En el Capítulo 2 se tratará la propagación de haces gaussianos: su descripción matemática y su relación con las matrices de propagación de rayos paraxiales.

El Capítulo 3 se dedica al análisis de una cavidad láser sin considerar efec-

tos no lineales. Se propone un modelo astigmático a través de las matrices de propagación de rayos y se derivan las ecuaciones de corrección de astigmatismo para emisión continua.

El Capítulo 4 estudia los efectos de autoenfocamiento no lineales considerados en este trabajo: el autoenfocamiento térmico y el autoenfocamiento por la intensidad del haz intracavidad (efecto Kerr). Se estudia el índice de refracción no lineal y la forma de modelar sus efectos con ayuda de las matrices de propagación de rayos.

En el Capítulo 5 se presentan diferentes técnicas para el análisis no lineal de la cavidad láser, generando mapas de amplitud y de astigmatismo para cada técnica.

En el Capítulo 6 se presentan los resultados del análisis numérico, la comparación entre los distintos modelos y la discusión de los resultados.

Parte del desarrollo de este trabajo fue presentado en el congreso SPIE Photonics Europe, que se llevó a cabo en la ciudad de Bruselas, Bélgica en Abril del 2016. Se realizó un manuscrito que fue publicado en "Proceedings of SPIE"[21].

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Matrices de propagación de rayos

Las matrices de propagación de rayos o matrices ABCD son ampliamente utilizadas para describir la propagación de rayos ópticos (óptica geométrica) a través de elementos ópticos paraxiales, tales como lentes, espejos curvos e interfaces de materiales dieléctricos. Los componentes ópticos utilizados en las cavidades resonantes láser son modelados por medio de éstas matrices. Éste capítulo tiene como fin describir las propiedades de las matrices de propagación de rayos y su utilización.

1.1. Óptica paraxial y matrices de rayos

1.1.1. Propagación en espacio libre

Considere un rayo de luz que viaja en la dirección z, con un desplazamiento transversal $r(z)$ a partir del eje con una pendiente dr/dz pequeña. Si tal rayo se propaga en el espacio libre ($n = 1$) desde un plano en z_1 a un plano $z_1' = z_1 + L$, las coordenadas del rayo en los planos de entrada y salida de la región descrita estarán dadas por las Ecuaciones (1.1a) y (1.1b).

$$r_1' = r_1 + L\frac{dr_1}{dz} \tag{1.1a}$$

$$\frac{dr_1'}{dz} = \frac{dr_1}{dz} \tag{1.1b}$$

La Ecuaciones (1.1a) y (1.1b) pueden representarse de forma matricial, como se presenta en la Ecuación (1.2)

$$\begin{pmatrix} r_1' \\ \alpha_1' \end{pmatrix} = \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ \alpha_1 \end{pmatrix} \tag{1.2}$$

donde $\alpha_1 = dr_1/dz$ y $\alpha_1' = dr_1'/dz$.

Para ser consistentes con otros trabajos sobre diseño de cavidades láser, se adoptó la siguiente convención:

- El valor del ángulo del rayo que entra o sale de un sistema óptico paraxial será positivo si la pendiente de dicho rayo es positiva.
- Para superficies curvas se considera su radio de curvatura R positivo si la superficie es cóncava.
- Se utiliza la aproximación paraxial ($\alpha \approx 0$).

La convención de signos varía dependiendo del trabajo por lo que es conveniente consultarla antes de seguir.

1.1.2. Interfaz dieléctrica curva

Considerese un rayo que se propaga en un medio dieléctrico de índice de refracción n_1 e incide en una superficie curva de radio R para ser refractada en un medio dieléctrico cuyo índice de refracción es n_2 como se muestra en la Figura 1.1

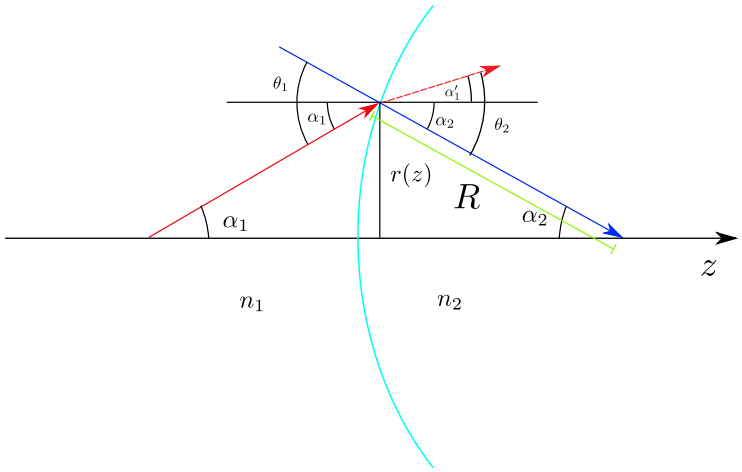


Figura 1.1: Interfaz dieléctrica curva en incidencia normal. $R < 0$ para superficies cónvexas.

donde

α_1 Es el ángulo que hace el rayo incidente con respecto al eje óptico.

α_1' Es el ángulo que hace el rayo transmitido con respecto al eje óptico.

θ_1 Es el ángulo de incidencia en la interfaz.

θ_2 Es el ángulo de transmisión refractado en la interfaz.

Se tiene que

$$\theta_1 = \alpha_1 - \alpha_2 \tag{1.3a}$$
$$\theta_2 = \alpha_1' - \alpha_2 \tag{1.3b}$$
$$\alpha_2 = -\sin\frac{r(z)}{R} \approx -\frac{r(z)}{R} \tag{1.3c}$$

Como la ley de Snell establece que $n_1 \sin\theta_1 = n_2 \sin\theta_2$, la aproximación paraxial reduce lo anterior a

$$n_1\theta_1 = n_2\theta_2 \tag{1.3d}$$
$$n_1(\alpha_1 + \alpha_2) = n_2(\alpha_1' + \alpha_2) \tag{1.3e}$$
$$n_1\alpha_1 - n_1\frac{r(z)}{R} = n_2\alpha_1' - n_2\frac{r(z)}{R} \tag{1.3f}$$
$$n_2\alpha_1' = n_1\alpha_1 + \frac{r(z)}{R}(n_2 - n_1) \tag{1.3g}$$

Dado que $r(z)_1 = r(z)_2 = r(z)$, lo anterior se puede escribir de forma matricial como en la Ecuación (1.4):

$$\begin{pmatrix} r_1' \\ n_2\alpha_1' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{n_2-n_1}{R} & 1 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ n_1\alpha_1 \end{pmatrix} \tag{1.4}$$

1.1.3. La matriz de rayos

Cuando uno traza un rayo paraxial a través de la combinación de lentes y medios que actuan como lente, las cantidades de interés son la posición r_1 y el ángulo α_1 del rayo en el plano de entrada, y las cantidades r_1' y α_1' en el plano de salida. En general existe una relación lineal entre las cantidades de entrada y salida que está escrita en forma matricial como

$$\begin{pmatrix} r_1' \\ n_2\alpha_1' \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ n_1\alpha_1 \end{pmatrix} \tag{1.5}$$

A la Ecuación (1.5) se le llama matriz de rayos o matriz ABCD del sistema. Generalmente su determinante es unitario [4]

$$AD - BC = 1. \tag{1.6}$$

Para encontrar el punto focal del lado de entrada se traza un rayo que deja el plano de salida paralelo al eje óptico ($\alpha_1' = 0$). La distancia s_1 desde el plano de entrada es calculada por

$$s_1 = \left.\frac{r_1}{\alpha_1}\right|_{\alpha_1'=0} = -\frac{n_1 D}{C}. \tag{1.7}$$

De forma similar se encuentra la distancia entre el plano de salida y el punto focal correspondiente s_2

$$s_2 = -\frac{r_1'}{\alpha_1'}\bigg|_{\alpha_1=0} = -\frac{n_2 A}{C}. \tag{1.8}$$

Para encontrar el plano principal en el lado de entrada trazamos un rayo a partir del punto focal hasta que su distancia del eje óptico sea igual a la posición r_1' del rayo de salida correspondiente

$$h_1 = \frac{r_1' - r_1}{\alpha_1}\bigg|_{\alpha_1'=0} \tag{1.9}$$

donde h_1 es la distancia entre el plano principal y el plano de entrada. En el lado de salida se encuentra que

$$h_2 = \frac{r_1' - r_1}{\alpha_1'}\bigg|_{\alpha_1=0}. \tag{1.10}$$

La distancia focal f_1 del sistema se obtiene al calcular la distancia entre el plano principal de entrada y el punto focal correspondiente. De forma semejante se realiza el cálculo para la distancia focal f_2

$$f_1 = s_1 + h_1\ \frac{r_1'}{\alpha_1}\bigg|_{\alpha_1'=0} \tag{1.11}$$

$$f_2 = s_2 + h_2 = -\ \frac{r_1}{\alpha_1'}\bigg|_{\alpha_1=0} \tag{1.12}$$

Empleando las Ecuaciones (1.5), (1.7), (1.8), (1.9), (1.10), (1.11) y (1.12) se tiene que

$$f_1 = -\frac{-n_1}{C} \tag{1.13}$$

$$f_2 = -\frac{-n_2}{C} \tag{1.14}$$

$$h_1 = \frac{n_1(D-1)}{C} \tag{1.15}$$

$$h_2 = \frac{n_2(A-1)}{C} \tag{1.16}$$

Los planos de referencia descritos anteriormente se muestran en la Figura 1.2 [6].

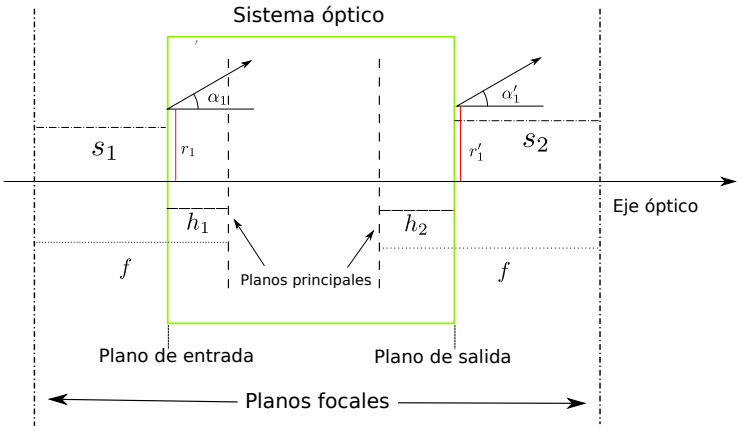


Figura 1.2: Planos de referencia en un sistema óptico.

1.2. Propagación de rayos a través de una serie de elementos

Una de las propiedades más importantes de las matrices de propagación de rayos es la de representar una serie de componentes ópticos en cascada por medio de una sóla matriz ABCD.

Supóngase un rayo descrito por el vector columna $\mathbf{a_0}$ que es propagado a través de varios elementos ópticos con matrices de rayos $\mathbf{M_1}$, ..., $\mathbf{M_n}$, arreglados en cascada. La transformación total a través de esta serie de elementos en cascada es calculada por medio de un proceso multiplicativo en cadena:

$$\mathbf{a_1} = \mathbf{M_1 a_0} \tag{1.17a}$$

$$\mathbf{a_2} = \mathbf{M_2 a_1} = \mathbf{M_2 M_1 a_0} \tag{1.17b}$$

$$\mathbf{a_3} = \mathbf{M_3 a_2} = \mathbf{M_3 M_2 M_1 a_0} \tag{1.17c}$$

El resultado general es

$$\mathbf{a_n} = [\mathbf{M_n M_{n-1} ... M_2 M_1}]\mathbf{a_0} \tag{1.18}$$

La matriz de propagación de rayos total para este sistema está dada por la Ecuación (1.19).

$$\mathbf{M_{tot}} = \mathbf{M_n M_{n-1} ... M_2 M_1} \tag{1.19}$$

La matriz $\mathbf{M_{tot}}$ permite englobar a un sistema óptico dentro de una caja negra relacionando a los rayos ubicados en el plano principal de entrada y salida.

1.3. Propagación de ondas esféricas y matrices ABCD

Las matrices de propagación de rayos permiten expresar de forma general las leyes elementales de la óptica geométrica, o la óptica de ondas esféricas, dejando fuera aberraciones de orden superior. [3] La óptica de rayos y la óptica geométrica proveen la misma información, aunque expresada de forma distinta.

Para demostrar esto imaginemos que una onda esférica ideal con radio de curvatura R puede ser vista como una colección de rayos que divergen desde un punto en común: el centro de curvatura C del frente de onda. La pendiente y el desplazamiento de cada uno de estos rayos con respecto al eje de propagación z donde el radio de curvatura es $R(z)$ - esto es, la distancia R desde el punto de origen - se relacionan por

$$n(z)\alpha = \frac{n(z)r(z)}{R(z)} \tag{1.20a}$$

ó

$$R(z) \equiv \frac{r(z)}{\alpha} \tag{1.20b}$$

La Ecuación (1.20b) implica una convención de signos en la cual una R positiva corresponde a una onda esférica convergente.

Supóngase una onda esférica con radio de curvatura $R_1 = r_1/\alpha_1$ que se propaga a través de un sistema paraxial con una matriz de propagación $ABCD$ obedeciendo la Ecuación (1.5).

El frente de onda que se encuentra a la salida de éste sistema será un frente de onda esférico con radio de curvatura $R_2 = r_1'/\alpha_1'$, que puede ser calculado por la Ecuación

$$\frac{R_2}{n_2} = \frac{Ar_1 + B\alpha_1}{Cr_1 + D\alpha_1} = \frac{A(R_1/n_1) + B}{C(R_1/n_1) + D} \tag{1.21}$$

1.4. Matrices de propagación de rayos en sistemas astigmáticos

Al usar coordenadas cartesianas en el sistema óptico, con propagación en la dirección z, un rayo general debe ser descrito por sus desplazamientos transversales en las direcciones x (tangencial) y y (sagital). Para componentes ópticos simples, las matrices de propagación de rayos son aplicadas de forma separada e independiente para los planos tangencial y sagital. Si un sistema óptico es rotacionalmente simétrico se aplican las mismas matrices ABCD para ambos planos; si el sistema es astigmático, se emplean matrices ABCD distintas para cada elemento en el plano tangencial y sagital.

1.5. Lista breve de matrices de rayos

Propagación en de un rayo a través de un medio con índice de refracción n y longitud L

$$\begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \tag{1.22}$$

Transformación de un rayo por una lente delgada cuya distancia focal sea f.

$$\begin{pmatrix} 1 & 0 \\ \frac{1}{f} & 1 \end{pmatrix} \tag{1.23}$$

Reflexión en espejo curvo de radio de curvatura R con incidencia arbitraria θ.

$$\begin{pmatrix} 1 & 0 \\ \frac{-2}{R_e} & 1 \end{pmatrix} \tag{1.24}$$

donde $R_e = R\cos\theta$ para el caso tangencial y $R_e = R/\cos\theta$ para el caso sagital.

Interfaz dieléctrica curva conformada por materiales de índice de refracción n_1 y n_2, incidencia arbitraria θ_1 con ángulo de salida θ_2 en el plano tangencial.

$$\begin{pmatrix} \frac{\cos\theta_2}{\cos\theta_1} & 0 \\ \Delta n_e/R & \frac{\cos\theta_1}{\cos\theta_2} \end{pmatrix} \tag{1.25}$$

donde $\Delta n_e = (n_2\cos\theta_2 - n_1\cos\theta_1)/\cos\theta_1\cos\theta_2$ y $n_1\sin\theta_1 = n_2\sin\theta_2$.

Interfaz dieléctrica curva conformada por materiales de índice de refracción n_1 y n_2, incidencia arbitraria θ_1 con ángulo de salida θ_2 en el plano sagital.

$$\begin{pmatrix} 1 & 0 \\ \Delta n_e/R & 1 \end{pmatrix} \tag{1.26}$$

donde $\Delta n_e = n_2\cos\theta_2 - n_1\cos\theta_1$ y $n_1\sin\theta_1 = n_2\sin\theta_2$.

Con ayuda de estos bloques básicos el diseñador óptico puede modelar sistemas ópticos más complejos como lo es una cavidad resonante láser.

Capítulo 2

Propagación de haces gaussianos

Una de las formas más sencillas de modelar un haz laser es mediante un haz gaussiano. En éste capítulo se estudia la relación entre ondas paraxiales, óptica de rayos y haces gaussianos con el fin de modelar una cavidad láser.

2.1. Ondas paraxiales y haces gaussianos

Se dice que una onda es paraxial si la normal de su frente de onda son rayos paraxiales [7]. Una forma de construirla es a partir de una onda plana $A \exp -jkz$ que actuará como una onda portadora, modulando su envolvente compleja A haciendola función de la posición que varie lentamente, de forma tal que la amplitud compleja de la onda modulada se vuelve

$$U(\mathbf{r}) = A(\mathbf{r}) \exp(-jkz) \tag{2.1}$$

donde $\mathbf{r}$ es un vector de posición cartesiano. La variación de la envolvente $A(\mathbf{r})$ y su derivada con respecto a la posición z debe ser lenta dentro de la distancia de la longitud de onda en el medio $\lambda = 2\pi/k$ para que la onda mantenga aproximadamente su naturaleza de onda plana.

La envolvente se considera aproximadamente constante en un rango de tamaño λ, de forma tal que la onda mantiene su naturaleza de onda plana en esa localidad y exhibe normales al frente de onda que son rayos paraxiales.

Para que la amplitud compleja $U(\mathbf{r})$ satisfaga la ecuación de Helmholtz, $\nabla^2 U + k^2 U = 0$, la envolvente compleja $A(\mathbf{r})$ debe satisfacer la ecuación paraxial de Helmholtz:

$$\frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} - j2k\frac{\partial A}{\partial z} = 0 \tag{2.2}$$

Una solución a la Ecuación (2.2) es una onda parabólica donde

$$A(\mathbf{r}) = \frac{A_1}{z}\exp(-jk\frac{r^2}{2z}) \tag{2.3}$$

donde A_1 es una constante y $r^2 = x^2 + y^2$. La onda parabólica es la aproximación paraxial de una onda esférica $U(r) = (A_1/r)\exp(-jkr)$ cuando x y y son mucho más pequeñas que z.

Otra solución para la ecuación paraxial de Helmholtz es un haz gaussiano. Se obtiene de la onda parabólica al usar una transformación sencillla [7]. Dado que la envolvente compleja de la onda parabólica es solución a la ecuación de Helmholtz paraxial entonces una versión desplazada de ésta también lo es, con $z = z - \xi$ donde ξ es una constante:

$$A(\mathbf{r}) = \frac{A_1}{q(z)}\exp\left[-jk\frac{r^2}{2q(z)}\right] \tag{2.4}$$

donde $q(z) = z - \xi$. Esto representa una onda parabólica centrada en el punto $z = \xi$ en lugar del origen. La Ecuación (2.4) es solución de la Ecuación (2.2) incluso cuando ξ es compleja, sin embargo, la solución adquiere propiedades distintas. Cuando $\xi = -jz_0$ donde z_0 es real, la Ecuación (2.4) resulta en la envolvente compleja de un haz gaussiano [7]

$$A(\mathbf{r}) = \frac{A_1}{q(z)}\exp\left[-jk\frac{r^2}{2q(z)}\right], q(z) = z + jz_0. \tag{2.5}$$

La cantidad $q(z)$ es el parámetro complejo q del rayo y el parámetro z_0 es el rango de Rayleigh.

Para separar la amplitud y fase de esta envolvente compleja, se escribe la función compleja $1/q(z) = 1/(z + jz_0)$ en términos de su parte real e imaginaria en la Ecuación (2.6)

$$\frac{1}{q(z)} = \frac{1}{R(z)} - j\frac{\lambda_0}{\pi n\omega^2(z)} \tag{2.6}$$

donde $R(z)$ es el radio de curvatura del frente de onda del haz, $\omega(z)$ es el radio del haz, λ_0 es la longitud de onda en el vacío y n es el índice de refracción del medio.

La amplitud compleja de un haz gaussiano es representada por la Ecuación (2.7):

$$U(\mathbf{r}) = A_0\frac{\omega_0}{\omega(z)}\exp\left[-\frac{r^2}{\omega^2(z)}\right]\exp\left[-jkz - jk - \frac{r^2}{2R(z)} + j\xi(z)\right] \tag{2.7}$$

2.2. Haces gaussianos y las matrices de propagación de rayos

2.2.1. Propagación de haces gaussianos en espacio libre

Considerando un parámetro complejo $q_0 = j(n\pi{\omega_0}^2/\lambda_0)$ tal que describa un haz gaussiano visto desde su cintura. Ahora, para describir el haz gaussiano una vez que se haya propagado en el vacío una distancia z, proponemos una ley de propagación como la mostrada en la Ecuación (2.8):

$$q = q_0 + z. \tag{2.8}$$

Obteniendo el recíproco de la Ecuación (2.8)

$$\frac{1}{q} = \frac{\lambda_0(z\lambda_0 - j\pi n{\omega_0}^2)}{z^2{\lambda_0}^2 + \pi^2 n^2 {\omega_0}^4} \tag{2.9}$$

Dado que el término $1/q$ de la Ecuación (2.9) está presente en la Ecuación (2.6), se igualan ambas ecuaciones lo que permite encontrar cómo cambia R y ω en función de un desplazamiento z. Estos cambios son mostrados en las Ecuaciones (2.10) y (2.11)

$$R = z\left[1 + \left(\frac{\pi n{\omega_0}^2}{z\lambda_0}\right)^2\right] \tag{2.10}$$

$$\omega^2 = {\omega_0}^2\left[1 + \left(\frac{z\lambda_0}{\pi n{\omega_0}^2}\right)^2\right] \tag{2.11}$$

2.3. Transformación de haces gaussianos por una lente delgada

Considerese un haz gaussiano descrito por q_1, que será modificado por una lente delgada y generará el haz gaussiano q_2. Dado que es una lente delgada, se observa que $\omega_1 = \omega_2$. La transformación de radios de curvatura es determinada por la Ecuación (1.21), donde la matriz ABCD que representa a una lente delgada es

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{pmatrix} \tag{2.12}$$

se encuentra que la transformación del radio de curvatura R_1 por una lente delgada es

$$\frac{1}{R_2} = \frac{1}{R_1} - \frac{1}{f} \tag{2.13}$$

donde f es la distancia focal de la lente delgada [8]. Los parámetros complejos q_1 y q_2 están relacionados por la Ecuación (2.14)

$$\frac{1}{q_2} = \frac{1}{q_1} - \frac{1}{f} \tag{2.14}$$

Ahora considerese que q_1 y q_2 describen dos haces gaussianos ubicados a una distancia d_1 y d_2 de una lente delgada, como se muestra en la Figura 2.1 y que q_1' y q_2' describen a los haces gaussianos justo en la lente.

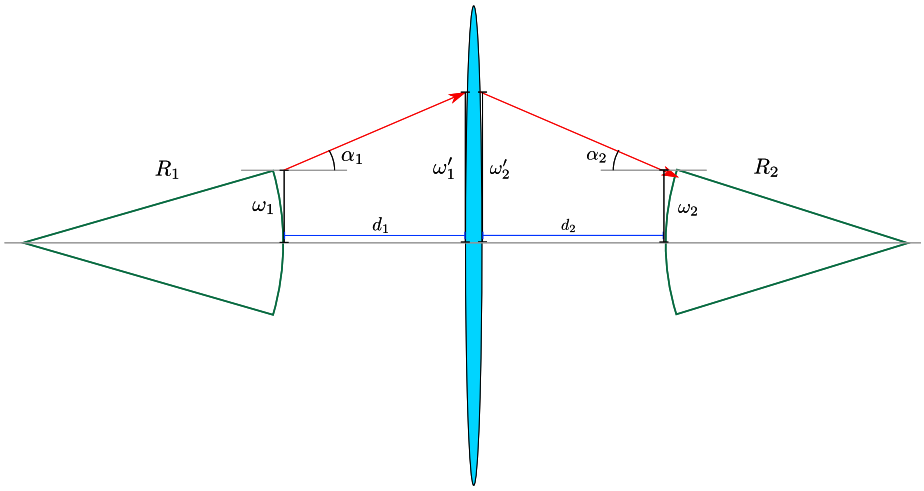


Figura 2.1: Transformación de haces gaussianos por una lente delgada,

El radio del haz ω_1' se relaciona con ω_1 por la Ecuación (2.15)

$$\omega_1' = \omega_1 + d_1 \tan\alpha_1, \tag{2.15}$$

la cual se vuelve

$$\omega_1' = \omega_1 + d_1\alpha_1 = \omega_1\left(1 + \frac{d_1}{R_1}\right). \tag{2.16}$$

Para ω_2', la relación con ω_2 se da por

$$\omega_2' = \omega_2 - d_2 \tan\alpha_2 \tag{2.17}$$

recordando que si el rayo tiene pendiente negativa el ángulo asociado es negativo. Por aproximación paraxial se tiene

$$\omega_2' = \omega_2 - d_2\alpha_2 = \omega_2\left(1 - \frac{d_2}{R_2}\right). \tag{2.18}$$

De lo anterior, los radios de curvatura R_1' y R_2' son

$$R_1' = \frac{\omega_1'}{\alpha_1} = \frac{\omega_1}{\alpha_1} + d_1 = R_1 + d_1 \tag{2.19}$$

y

$$R_2' = \frac{\omega_2'}{\alpha_2} = \frac{\omega_2}{\alpha_2} - d_2 = R_2 - d_2. \tag{2.20}$$

De las Ecuaciones (2.14), (2.19) y (2.20) se deduce que los parámetros complejos q_1 y q_2 están relacionados por

$$\frac{1}{q_2 - d_2} = \frac{1}{q_1 + d_1} - \frac{1}{f} \tag{2.21}$$

Despejando q_2 de la Ecuación (2.21):

$$q_2 = \frac{(1 - \frac{d_2}{f})q_1 + (d_1 + d_2 - \frac{d_1 d_2}{f})}{-\frac{q_1}{f} + (1 - \frac{d_1}{d_2})} \tag{2.22}$$

2.4. Propagación de haces gaussianos y matrices ABCD

Si se tiene un haz gaussiano descrito por q_1 propagándose en el vacío, puede emplearse la Ecuación (2.22) sustituyendo las distancias d_1 y d_2 por los planos principales (Ecuaciones (2.23) y (2.24)) y la distancia focal de la lente por la distancia focal (Ecuación (2.25)) del sistema (asumiendo que la distancia focal anterior y posterior son las mismas)

$$h_1 = \frac{(D-1)}{C} \tag{2.23}$$

$$h_2 = \frac{(A-1)}{C} \tag{2.24}$$

$$f_1 = -\frac{-1}{C} \tag{2.25}$$

donde $n_1 = n_2 = 1$, se obtiene

$$q_2 = \frac{Aq_1 + B}{Cq_1 + D} \tag{2.26}$$

la cual es la misma ley de propagación paraxial de ondas esféricas descritas anteriormente [3]

$$\frac{q_2}{n_2} = \frac{A\frac{q_1}{n_1} + B}{C\frac{q_1}{n_1} + D}. \tag{2.27}$$

Utilizando la Ecuación (2.27) y las matrices de propagación de rayos estudiadas en el Capítulo 1 se puede modelar la propagación de un haz gaussiano fundamental dentro de una cavidad resonante láser y estimar sus características como se realizará en los siguientes Capítulos.

Capítulo 3

Análisis lineal de la cavidad

3.1. Modelo de la cavidad lineal

Se modeló la cavidad láser mostrada en la Figura 3.1 utilizando las matrices de propagación de rayos encontradas en [3] y [9] y se empleó la metodología descrita en [4]. La cavidad resonante es modelada a partir del producto de cada matriz de propagación de rayos, describiendo un viaje redondo de un rayo de luz a través del sistema a partir de una superficie de referencia. Debido a que dicho sistema óptico es astigmático se obtienen dos matrices: una corresponde al plano tangencial y el otro al plano sagital. Se presentan dichas matrices en las Ecuaciones (3.1), (3.2), (3.3) y (3.4).

- Plano tangencial referido a EM_1

$$\begin{aligned}\begin{pmatrix} A_t & B_t \\ C_t & D_t \end{pmatrix} = &\begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1t}+\delta_{1t} \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & \frac{L}{n^3} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2t}+\delta_{2t} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2t}+\delta_{2t} \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & \frac{L}{n^3} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1t}+\delta_{1t} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \end{aligned} \quad (3.1)$$

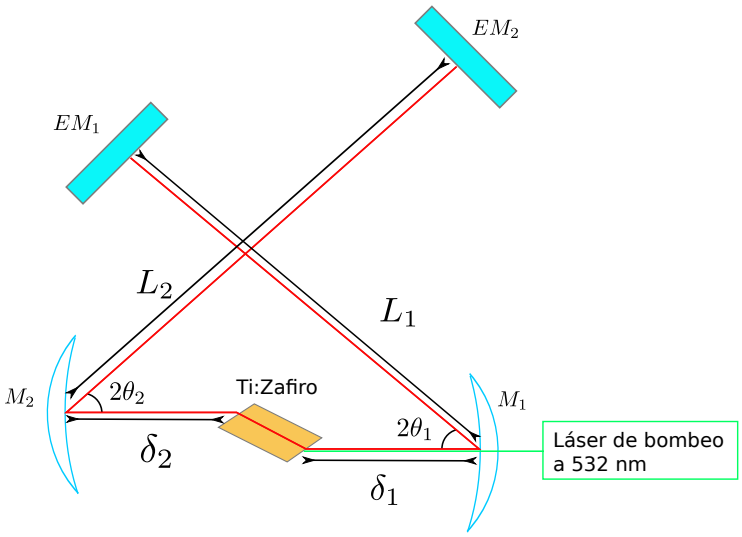


Figura 3.1: Cavidad resonante para un láser Ti:Zafiro donde L_1 y L_2 son la distancia que separa a EM_1 de M_1 y a EM_2 de M_2; δ_1 y δ_2 es la distancia entre M_1 y M_2 al cristal; y θ_1 y θ_2 es el ánglulo de incidencia del haz sobre los espejos M_1 y M_2.

- Plano sagital referido a EM_1

$$\begin{aligned}\begin{pmatrix} A_s & B_s \\ C_s & D_s \end{pmatrix} = &\begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1s}+\delta_{1s} \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2s}+\delta_{2s} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2s}+\delta_{2s} \\ 0 & 1 \end{pmatrix} \times \\ &\times \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1s}+\delta_{1s} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \end{aligned} \quad (3.2)$$

- Plano tangencial referido a EM_2

$$\begin{pmatrix} A_t & B_t \\ C_t & D_t \end{pmatrix} = \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2t}+\delta_{2t} \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & \frac{L}{n^3} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1t}+\delta_{1t} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1t}+\delta_{1t} \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & \frac{L}{n^3} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2t}+\delta_{2t} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2t}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \quad (3.3)$$

- Plano sagital referido a EM_2

$$\begin{pmatrix} A_s & B_s \\ C_s & D_s \end{pmatrix} = \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2s}+\delta_{2s} \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1s}+\delta_{1s} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{1s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{1s}+\delta_{1s} \\ 0 & 1 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & f_{2s}+\delta_{2s} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\frac{1}{f_{2s}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & L_2 \\ 0 & 1 \end{pmatrix} \quad (3.4)$$

Donde

L_1**:** Distancia entre el espejo EM1 y el espejo M1.

L_2**:** Distancia entre el espejo EM2 y el espejo M2.

$f_{1t,1s}$**:** Distancia focal tangencial o sagital del espejo M1.

$f_{2t,2s}$**:** Distancia focal tangencial o sagital del espejo M2.

$\delta_{1t,1s}$**:** Distancia que separa al espejo M1 de un extremo del cristal Ti:Zafiro referida al plano tangencial o sagital.

$\delta_{2t,2s}$**:** Distancia que separa al espejo M2 de un extremo del cristal Ti:Zafiro referida al plano tangencial o sagital.

n**:** Índice de refracción del cristal Ti:Zafiro.

L**:** Grosor del cristal. El grosor para un cristal cortado en ángulo recto y para un cristal cortado al ángulo de Brewster se muestra en la Figura 3.2.

Las Ecuaciones (3.1), (3.2) (3.3) y (3.4) representan una secuencia de elementos ópticos. Cada viaje redondo de un rayo de luz a través de esta secuencia es representado como una potencia de la matriz ABCD de la cavidad, la cual puede ser evaluada por el teorema de Sylvester [4]:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^n = \frac{1}{\sin\Theta} \cdot \begin{pmatrix} A\sin n\Theta - \sin(n-1)\Theta & B\sin n\Theta \\ C\sin n\Theta & D\sin n\Theta - \sin(n-1)\Theta \end{pmatrix} \tag{3.5}$$

Donde

$$\Theta = \arccos\left[\frac{1}{2}(A+D)\right] \tag{3.6}$$

La secuencia es considerada estable cuando la traza de la matriz obedece la desigualdad

$$-1 < \frac{1}{2}(A+D) < 1. \tag{3.7}$$

La Ecuación (3.5) muestra que la N-ésima potencia de la matriz no diverge si Θ es una cantidad real. Si Θ fuera compleja, digamos $\Theta = a + jb$, los términos proporcionales ($\sin[n\Theta]$) podrían escribirse como $\sin[n\Theta] = [\exp[(jna - nb)] + \exp[(-jna + nb)]]/2j$. La cantidad $\sin[n\Theta]$ tendría un término que crece exponencialmente con n, haciendo que la N-ésima potencia de la matriz diverge conforme N incrementa [8].

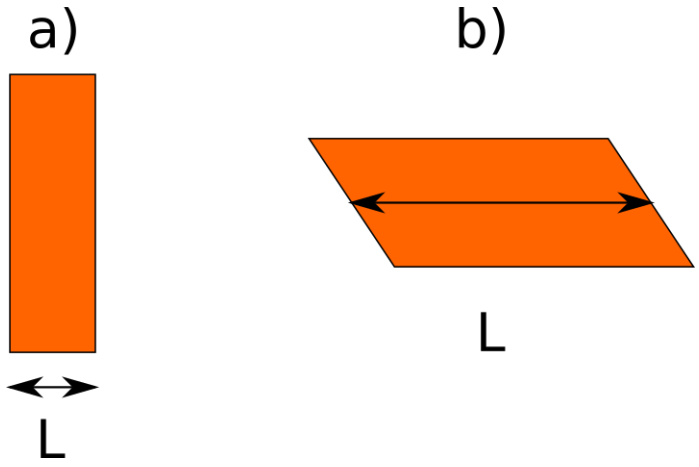


Figura 3.2: Grosor del cristal Ti:Zafiro. a) representa un cristal cortado en ángulo recto y b) representa un cristal cortado al ángulo de Brewster [5].

Al propagarse la luz en el cristal cortado al ángulo de Brewster se expande el tamaño del modo gaussiano conforme el haz se refracta dentro del cristal en el plano tangencial debido a que la interfaz aire-Titanio:Zafiro se encuentra inclinada con respecto al eje óptico, mientras que no se presentan cambios en el tamaño del modo en el plano sagital donde no se observa tal inclinación. Este efecto introduce astigmatismo debido a que el rayo presenta distintos caminos

ópticos en cada plano como se indica en las Ecuaciones (3.8) y (3.9) donde L_s y L_t es el recorrido que realiza la luz dentro del cristal [3].

$$L_s = \frac{L}{n} \tag{3.8}$$

$$L_t = \frac{L}{n^3} \tag{3.9}$$

A su vez, al incidir de forma arbitraria un haz de luz sobre un espejo curvo cambia la distancia focal para los planos tangencial y sagital como se muestra en las Ecuaciones (3.10) y (3.11) [3].

$$f_{1,2_t}(f_{1,2}, \theta_1) = f_{1,2} \cos\theta_1 \tag{3.10}$$

$$f_{1,2_s}(f_{1,2}, \theta_2) = f_{1,2}/\cos\theta_2 \tag{3.11}$$

Donde θ_1 y θ_2 son el ánglulo de incidencia del haz sobre los espejos M_1 y M_2.

Para compensar el astigmatismo lineal provocado por el cristal cortado al ángulo de Brewster se emplean espejos curvos que "doblen" el rayo en el plano principal de la cavidad, introduciendo astigmatismo de signo contrario al causado por el cristal. Algunos de los límites de estabilidad de la cavidad forman imágenes en los espejos extremos EM_1 y EM_2 (Figura 3.3) por lo que es necesario calcular la distancia del punto de la imagen relativa a cada espejo curvo en el plano sagital y tangencial, como se indica en la Ecuación (3.12d).

$$\frac{1}{D} + \frac{1}{p_{t,s}} = \frac{1}{f_{t,s}} \tag{3.12a}$$

$$\frac{1}{p_{t,s}} = \frac{1}{f_{t,s}} - \frac{1}{D} \tag{3.12b}$$

$$\frac{1}{p_{t,s}} = \frac{D - f_{t,s}}{D f_{t,s}} \tag{3.12c}$$

$$p_{t,s}(f, \theta, D) = \frac{D f_{t,s}}{D - f_{t,s}} \tag{3.12d}$$

Donde D representa la distancia del "objeto" del cual se forma la imagen. Para un haz no colimado, este "objeto" se encuentra en los espejos extremos por lo que D iguala la longitud de la rama correspondiente de la cavidad. La compensación de astigmatismo lineal se logra al resolver la Ecuación (3.13) [10].

$$\Delta p(f_1, \theta_1, D_1) + \Delta p(f_2, \theta_2, D_2) = \Delta L \tag{3.13}$$

Para este tipo de cavidades, si los ángulos θ_1 y θ_2 son distintos, se puede re-escribir la Ecuación (3.13) por las Ecuaciones (3.14a) y (3.14b).

$$\Delta p(f_1, \theta_1, D_1) = \frac{\Delta L}{2} \tag{3.14a}$$

$$\Delta p(f_2, \theta_2, D_2) = \frac{\Delta L}{2} \tag{3.14b}$$

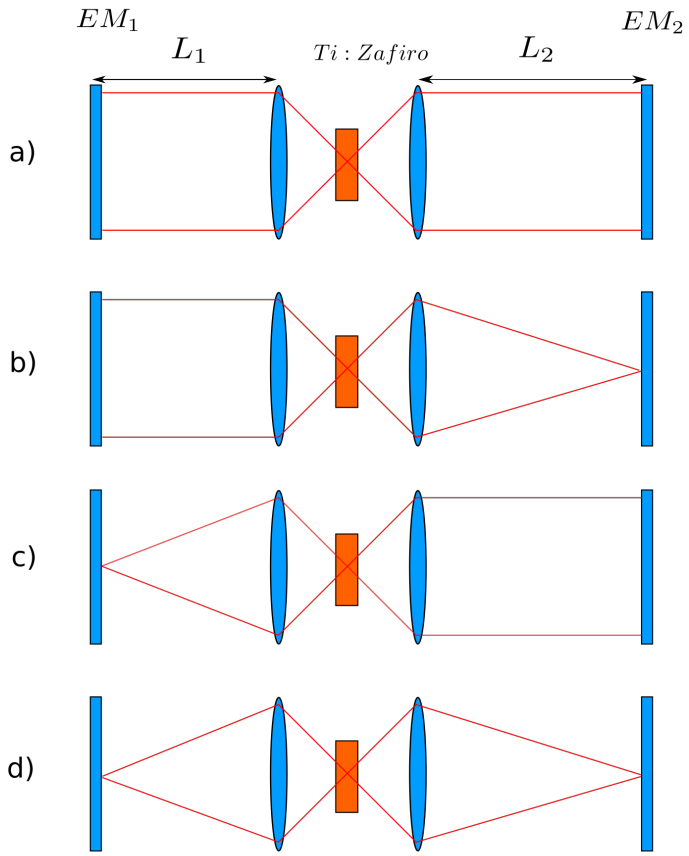


Figura 3.3: Representación geométrica del modo de la cavidad en los límites de estabilidad: a) límite plano-plano; b) límite plano-punto; c) límite punto-plano; d) límite punto-punto.

3.2. Compensación de astigmatismo con conjugado infinito

Tomando la Ecuaciones (3.14a) o (3.14b) (dependiendo del brazo con el que se trabaje) y utilizando las Ecuaciones (3.10) y (3.11) se obtiene la Ecuación (3.15a):

$$\Delta p(f,\theta,D) = f\cos\theta - \frac{f}{\cos\theta} - \frac{L}{2}\left(\frac{1}{n^3} - \frac{1}{n}\right) \tag{3.15a}$$

Definiendo a $A = \frac{L}{2}\left(\frac{1}{n^3} - \frac{1}{n}\right)$ y a $x = cos\theta$:

$$fx - \frac{f}{x} - A = 0 \tag{3.15b}$$

$$fx^2 - Ax - f = 0 \tag{3.15c}$$

$$x = \frac{A \pm \sqrt{A^2 + 4f^2}}{2f} \tag{3.15d}$$

El ángulo al que debe posicionarse el espejo para compensar el astigmatismo en la mitad de la cavidad se obtiene por:

$$\theta = \arccos(x) = \arccos\left(\frac{A \pm \sqrt{A^2 + 4f^2}}{2f}\right) \tag{3.16}$$

La distancia δ que separa el espejo cóncavo del cristal está dada por la Ecuación (3.17a) o la Ecuación (3.17b). Como se compenzó astigmatismo estas dos ecuaciones deben dar el mismo valor numérico al ser evaluadas.

$$\delta = f_s - \frac{L}{2n} \tag{3.17a}$$

$$\delta = f_t - \frac{L}{2n^3} \tag{3.17b}$$

3.3. Compensación de astigmatismo con conjugado finito

A partir de las Ecuaciones (3.14a) o (3.14b), y de la Ecuación (3.12d) se tiene:

$$\frac{fDx}{D - fx} - \frac{fD}{x\left(D - \frac{f}{x}\right)} - A = 0 \tag{3.18}$$

Resolviendo la Ecuación (3.18) para x:

$$x = \frac{A(f^2 + D) \pm \sqrt{4f^2D^4 + A^2(f^2 - D^2)^2}}{2fD(A + D)} \tag{3.19}$$

Como en la Ecuación (3.16), el ángulo que compensará el astigmatismo en la mitad de la cavidad es $\theta = \arccos(x)$.

De forma similar al caso anterior, la distancia δ que separa el espejo cóncavo del cristal se da por las Ecuaciones (3.20a) y (3.20b).

$$\delta = \frac{f_sD}{D - f_s} - \frac{L}{2n} \tag{3.20a}$$

$$\delta = \frac{f_tD}{D - f_t} - \frac{L}{2n^3} \tag{3.20b}$$

3.4. Cálculo de radio del haz a partir del sistema paraxial

Recordando la ley de propagación de haces gaussianos y matrices ABCD

$$\frac{q_2}{n_2} = \frac{A\frac{q_1}{n_1} + B}{C\frac{q_1}{n_1} + D} \tag{2.27}$$

y que un haz gaussiano es descrito por

$$\frac{1}{q} = \frac{1}{R(z)} - j\frac{\lambda_0}{n\pi\omega^2(z)}, \tag{3.21}$$

se reescribe la Ecuación (2.27), considerando que $n_1 = n_2 = 1$, para poder ser usada en la Ecuación (3.21).

$$\frac{1}{q_2} = \frac{C + D\frac{1}{q_1}}{A + B\frac{1}{q_1}} \tag{3.22}$$

Sustituyendo la Ecuación (3.22) con (3.21) se tiene que:

$$\frac{C + D\frac{1}{q_1}}{A + B\frac{1}{q_1}} = \frac{1}{R_2(z)} - j\frac{\lambda}{n\pi\omega_2^2(z)} \tag{3.23}$$

Se recuerda que los haces gaussianos son soluciones para la ecuación paraxial de Helmholtz (Ecuación (2.2)). Para que estas soluciones describan el haz que se propaga dentro de la cavidad láser es necesario que el haz al ser propagado por una cavidad resonante láser estable; en otras palabras, que satisfaga la Ecuación (3.7); se replique a sí mismo después de un viaje redondo. Esto implica que $q_1 = q_2 = q$. Sustituyendo en la Ecuación (3.22):

$$B\left(\frac{1}{q}\right)^2 + (A - D)\left(\frac{1}{q}\right) - C = 0 \tag{3.24a}$$

$$\frac{1}{q} = \frac{-(A - D) \pm \sqrt{(A - D)^2 + 4BC}}{2B} \tag{3.24b}$$

$$\frac{1}{q} = -\left(\frac{A - D}{2B}\right) \pm \frac{1}{B}\sqrt{\left(\frac{A - D}{2}\right)^2 + BC} \tag{3.24c}$$

Como $AD - BC = 1$:

$$\frac{1}{q} = -\left(\frac{A - D}{2B}\right) - \frac{j}{B}\sqrt{1 - \left(\frac{A + D}{2}\right)^2} \tag{3.24d}$$

Sustituyendo la Ecuación (3.24d) en la Ecuación (3.21) e igualando partes real e imaginarias:

$$R = \frac{2B}{D - A} \tag{3.25a}$$

$$\omega^2 = \frac{\lambda B}{n\pi}\sqrt{1 - \left(\frac{A + D}{2}\right)^2} \tag{3.25b}$$

3.5. Ejemplo numérico

Se considera una cavidad cuyos espejos curvos tengan un radio de curvatura de $R = 100$ [mm] ($f = 50$ [mm]), el brazo $L_1 = 1000$ [mm], el brazo $L_2 = 1350$ [mm], y un cristal Ti:Zafiro cuyo índice de refracción sea $n = 1.7598$ si $\lambda_0 = 810$ [nm] y posea un grosor $L = 10$ [mm].

Se calcularon los ángulos θ_1 y θ_2, cuyos valores se presentan en sus respectivas Figuras, que compensan el astigmatismo lineal en la cavidad para los cuatro límites de estabilidad (conjugados infinito-infinito, infinito-finito, finito-infinito y finito-finito) y se variará la distancia δ_2 como se muestra en la Ecuación (3.26):

$$\delta_2 = \delta + \epsilon \tag{3.26}$$

Donde δ proviene de las Ecuaciones (3.17a) o (3.20a), dependiendo del tipo de conjugado; y ϵ es un desplazamiento adicional en la separación del medio de ganancia al espejo cóncavo M_2. Se generaron gráficas que muestran el tamaño de ω para los planos tangencial y sagital utilizando las matrices de propagación de rayos generadas usando los espejos EM_1 y EM_2 como referencia, variando la distancia ϵ. A su vez se realizó y graficó $\Delta\omega = |\omega_t - \omega_s|$ para evaluar la efectividad de los ángulos calculados para compensar el astigmatismo lineal en los espejos EM_1 y EM_2. Se muestra el diagrama en la Figura 3.4.

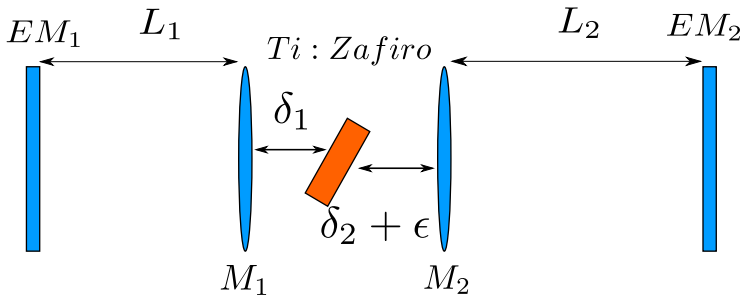


Figura 3.4: Montaje de espejos para cálculo - representación geométrica.

3.5.1. Tamaño de cintura de haz

Las gráficas del tamaño de la cintura del haz ω_t y ω_s en los espejos EM_1 y EM_2 se presentan en la Figura 3.5, la Figura 3.6, la Figura 3.7 y la Figura 3.8; donde las regiones inestables se denotan por un radio de haz $\omega_{t,s} = 0$. Al variar ϵ cambia el frente de onda del haz, donde $\epsilon = 0$ es la condición para la cual se calculó θ_1 y θ_2.

La cavidad es estable en los planos tangencial y sagital cerca del área de interés ($\epsilon \approx 0$) y el tamaño de los modos tangencial y sagital son muy similares entre sí.

En las Figuras 3.5, 3.6, 3.7 y 3.8 se realizó la corrección de astigmatismo para una región de operación en específico. Al examinar las gráficas se aprecian diferencias pequeñas si se modifica el espaciamiento entre el cristal y los espejos cóncavos mientras se conserva el ángulo de inclinación propuesto. En éstas configuraciones corregidas en astigmatismo se aprecia, además, que las regiones de estabilidad para el plano tangencial y sagital son muy similares entre sí. La forma de las gráficas es muy similar a la reportada en la literatura [10].

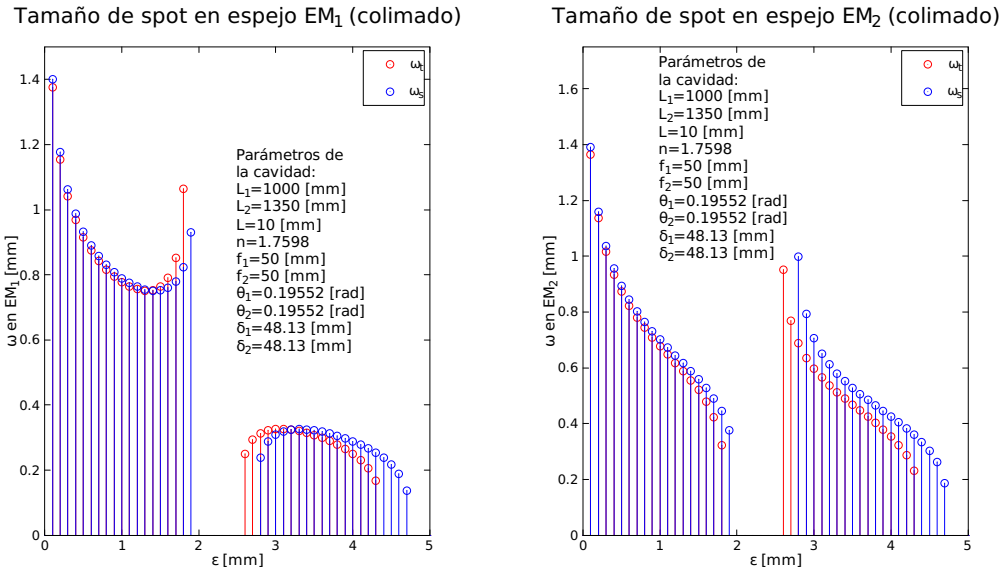


Figura 3.5: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite infinito-infinito.

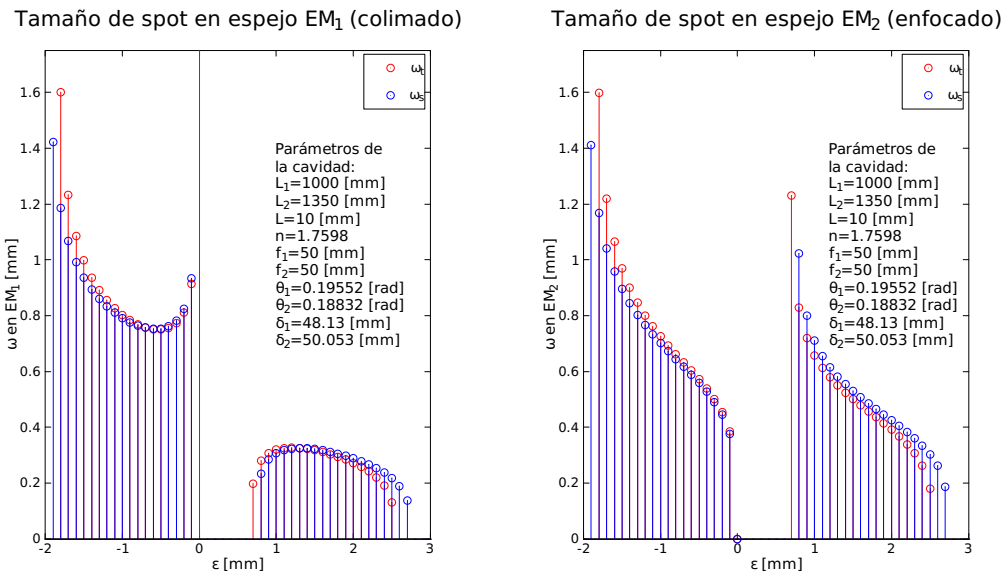


Figura 3.6: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite infinito-finito.

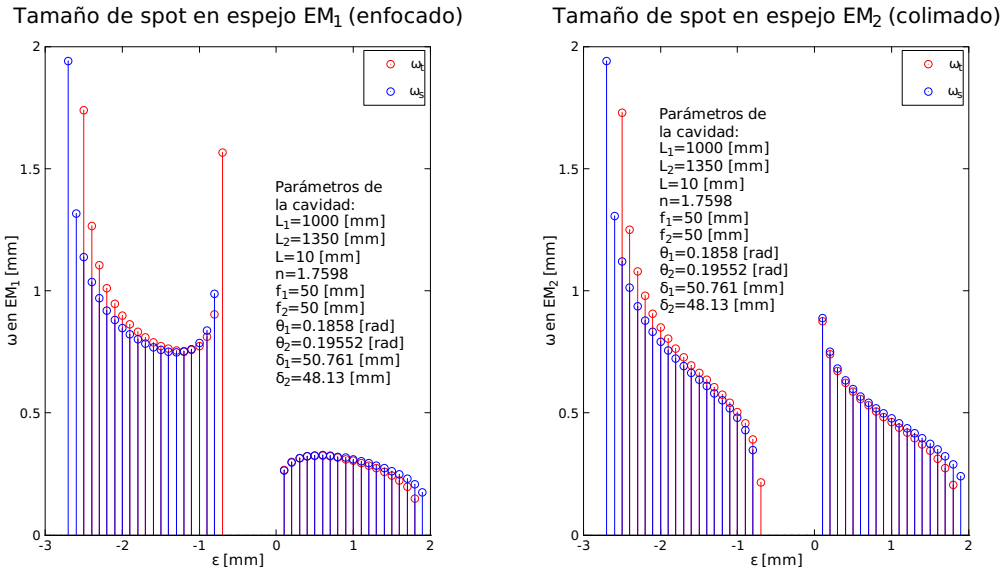


Figura 3.7: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite finito-infinito.

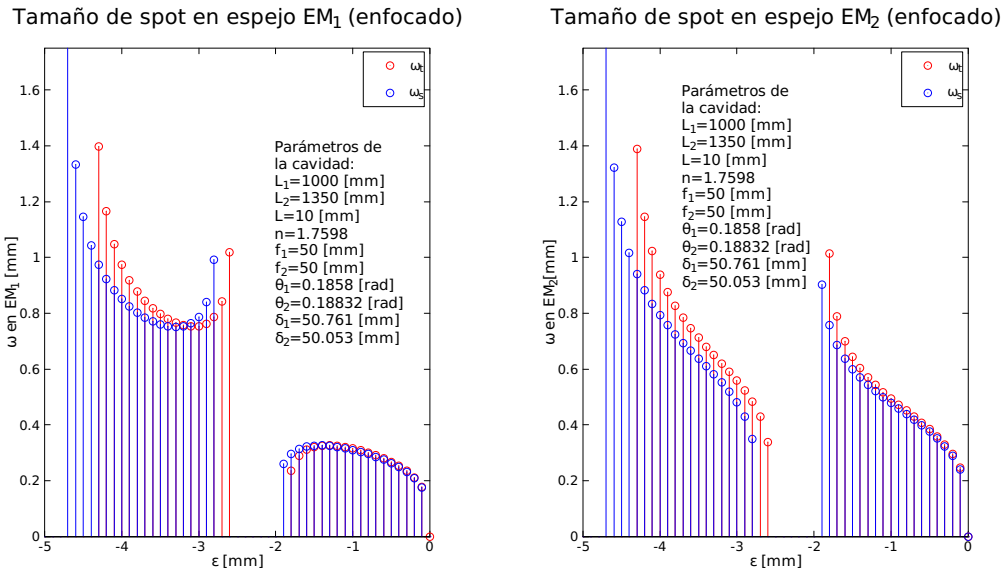


Figura 3.8: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite finito-finito.

3.5.2. Diferencia de tamaño de cintura de haz

Dado que las gráficas anteriores muestran que ω_t y ω_s para los espejos EM_1 y EM_2 son muy similares entre sí, se calculó la diferencia entre esos valores. Las gráficas resultantes se presentan en la Figura 3.9, la Figura 3.10, la Figura 3.11 y la Figura 3.12.

Se aprecia que los ángulos de compensación hacen que la diferencia de tamaño de la cintura del haz ($\Delta\omega$) en la región de interés ($\epsilon \approx 0$) sea por debajo de 30 [nm].

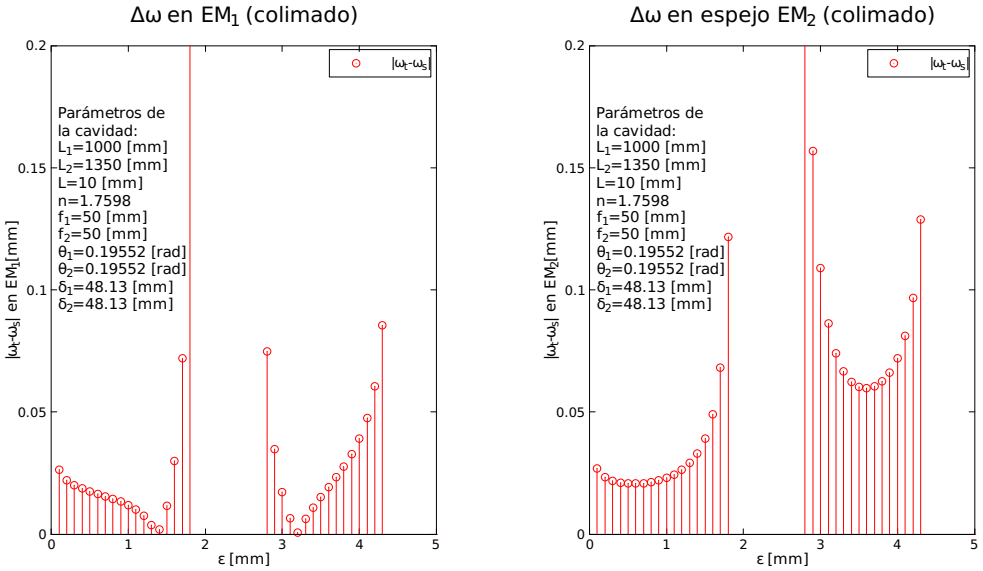


Figura 3.9: Diferencia de ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite infinito-infinito.

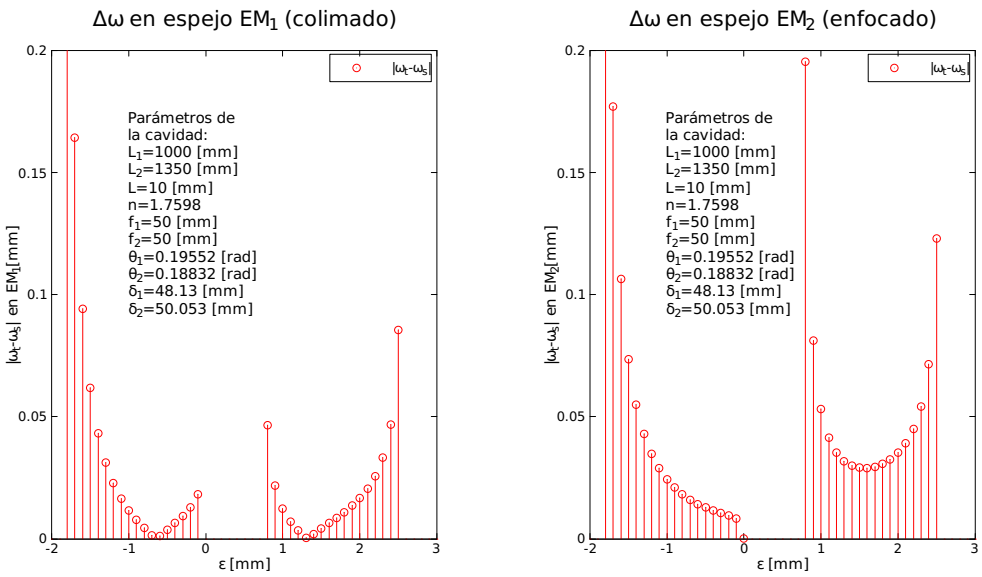


Figura 3.10: Diferencia de ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite infinito-finito.

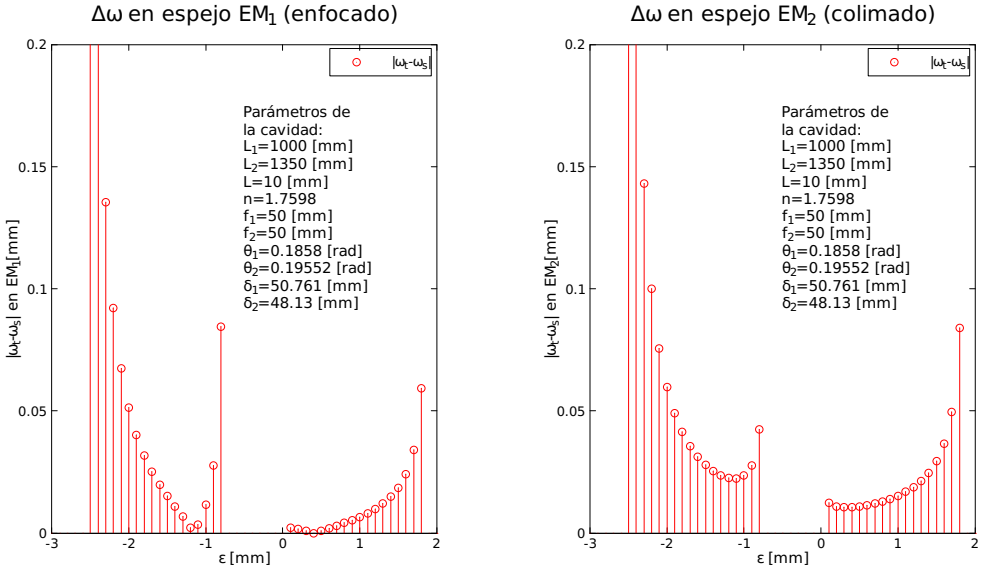


Figura 3.11: Diferencia de ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite finito-infinito.

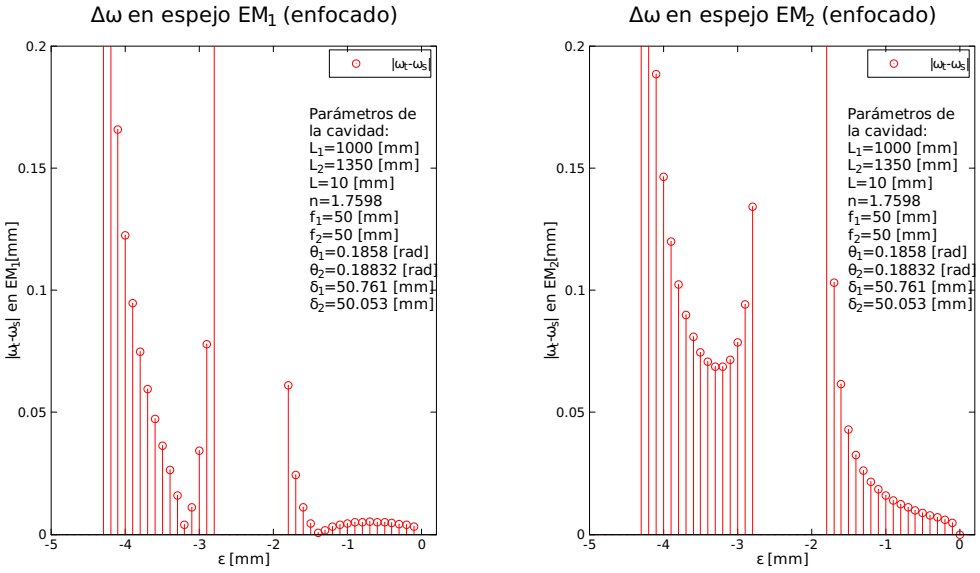


Figura 3.12: Diferencia de ω_t y ω_s en espejos EM_1 y EM_2 con θ_1 y θ_2 optimizados para el límite finito-finito.

3.5.3. Caso astigmático

Para demostrar la importancia de la compensación de astigmatismo lineal, se realizó el cálculo de ω_t y ω_s en los espejos EM_1 y EM_2 conservando las distancias δ_1 y δ_2, fijando a $\theta_1 = \theta_2$ a 0.2 [rad] (11.459°) y a 0.24435 [rad] (14 °), siendo el primer valor ligeramente superior al ángulo máximo usado para la corrección de astigmatismo en onda continua ($\theta \approx 11.2$) y el segundo es un valor varios grados mayor.

Para $\theta_1 = \theta_2 = 0.2$ [rad]

Las gráficas resultantes se presentan en la Figura 3.13, la Figura 3.14, la Figura 3.15 y la Figura 3.16.

Éste ángulo de inclinación propuesto es cercano a los valores empleados para la corrección de astigmatismo (11.4) [°]. Se observa que existe un desplazamiento entre las regiones de estabilidad para los planos tangencial y sagital en la cavidad, la cual se aprecia en el valor de ω_t y ω_s dado un valor de ϵ.

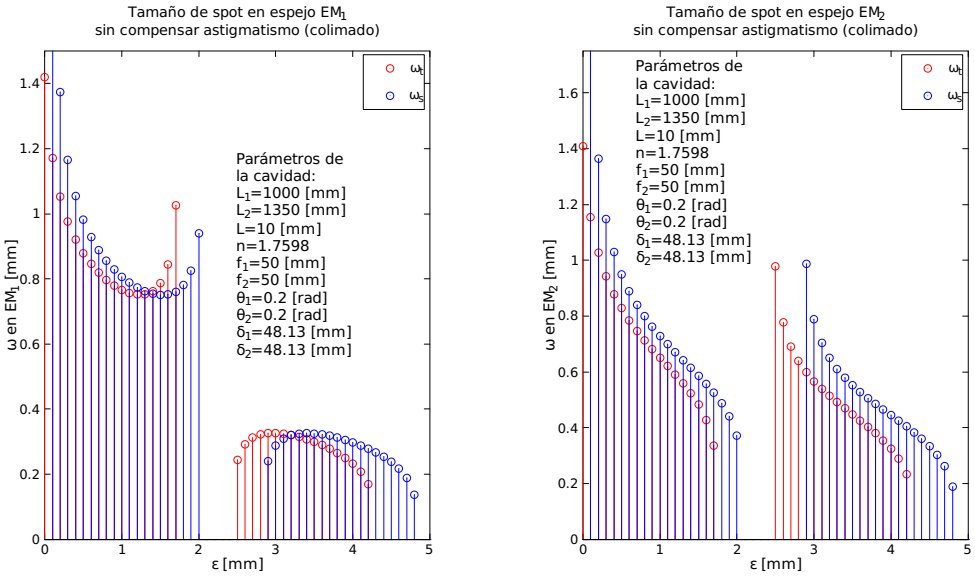


Figura 3.13: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.20$ [rad] para el límite infinito-infinito.

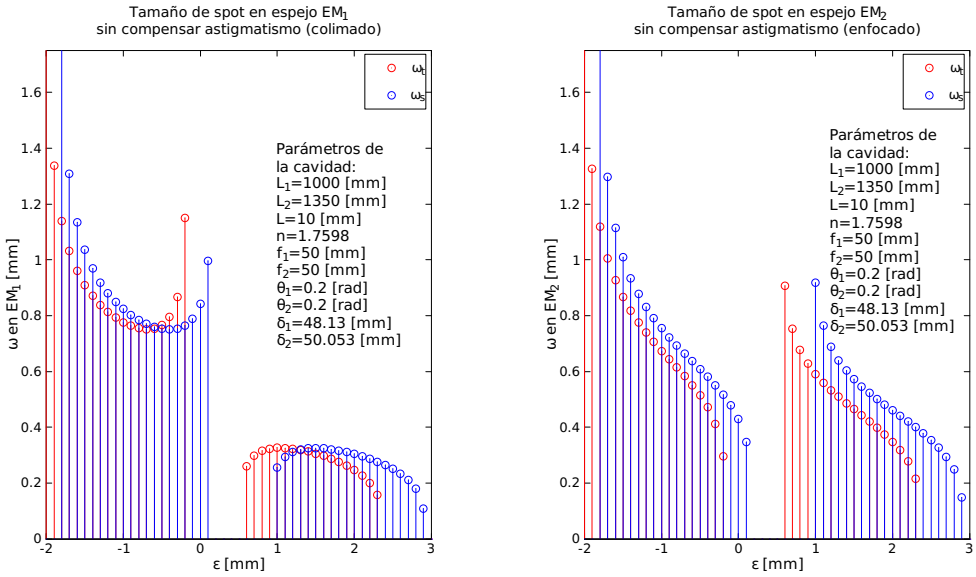


Figura 3.14: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.20$ [rad] para el límite infinito-finito.

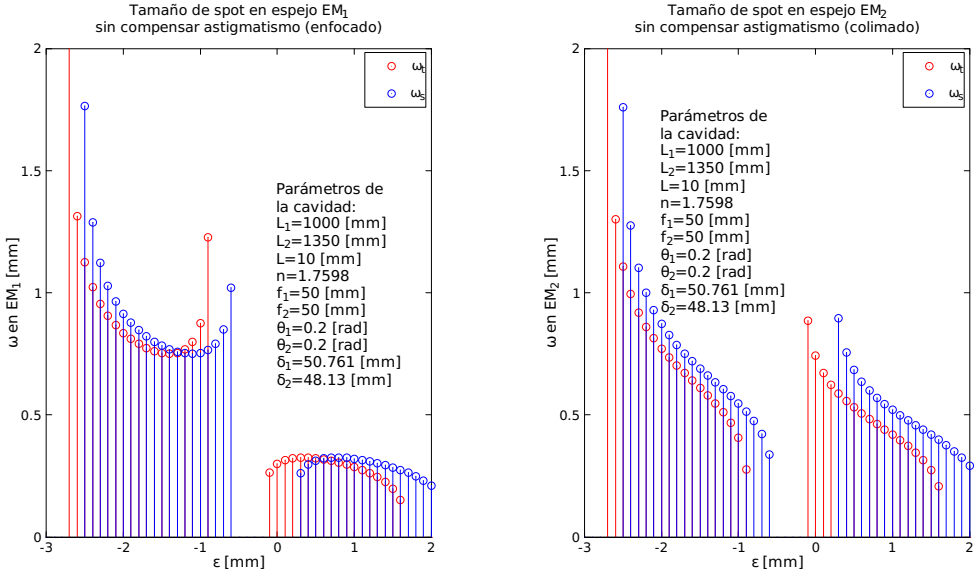


Figura 3.15: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.20$ [rad] para el límite finito-infinito.

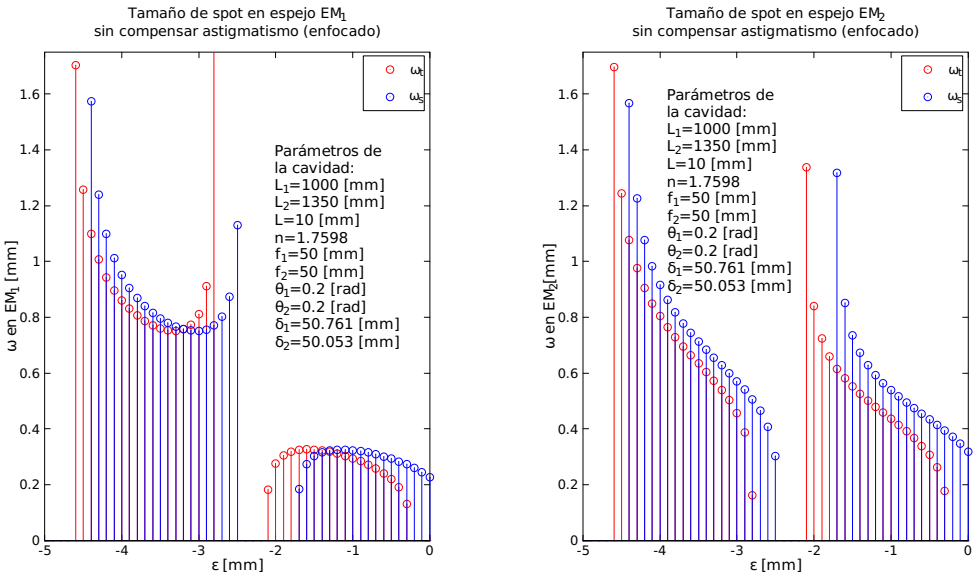


Figura 3.16: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.20$ [rad] para el límite finito-finito.

Para $\theta_1 = \theta_2 = 0.24435$ [rad]

Las gráficas resultantes se presentan en la Figura 3.17, la Figura 3.18, la Figura 3.19 y la Figura 3.20.

Al utilizar un ángulo ángulo de inclinación de los espejos cóncavos (14.0 [°]) mayor que el necesario para corregir astigmatismo se incrementan las diferencias de cómo opera la cavidad en los planos tangencial y sagital, haciendo que existan regiones en las cuales el haz puede ser estable en un plano, pero inestable en el plano ortogonal.

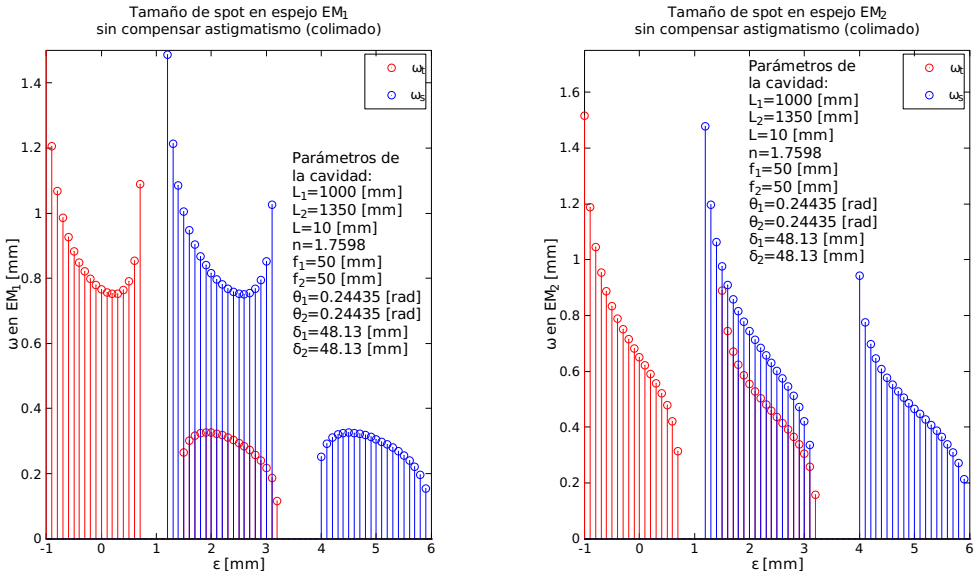


Figura 3.17: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.24435$ [rad] para el límite infinito-infinito.

Con lo anterior, se demuestra la importancia de la corrección de astigmatismo para el diseño de una cavidad láser y el cambio en el radio del haz dependiendo del espacio entre los espejos cóncavos y el cristal Titanio:Zafiro. Esta teoría permitirá generar las soluciones semilla para el análisis no lineal de la cavidad Titanio:Zafiro.

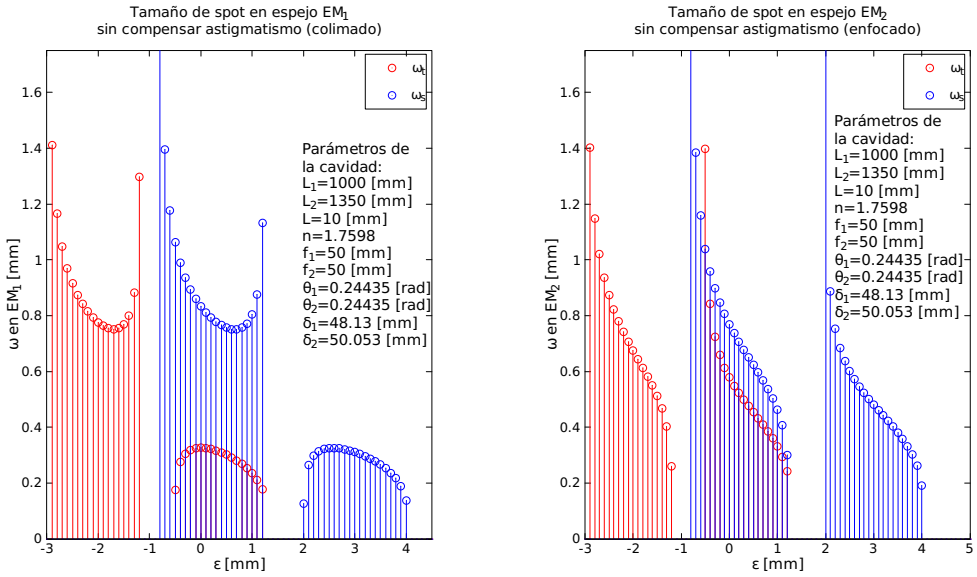


Figura 3.18: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.24435$ [rad] para el límite infinito-finito.

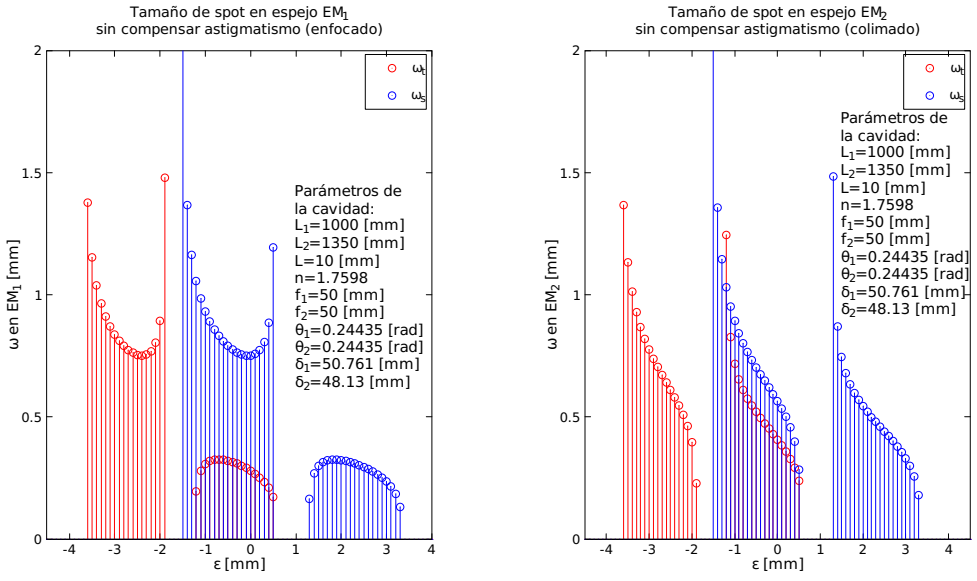


Figura 3.19: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.24435$ [rad] para el límite finito-infinito.

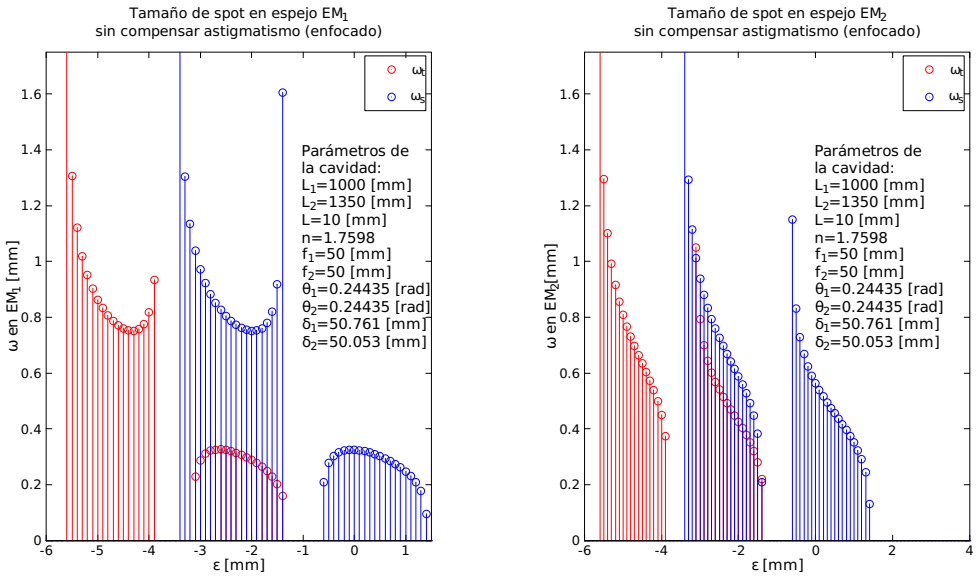


Figura 3.20: Tamaño de cintura de haz ω_t y ω_s en espejos EM_1 y EM_2 con $\theta_1 = \theta_2 = 0.24435$ [rad] para el límite finito-finito.

Capítulo 4

Modelado de efectos de autoenfocamiento

Para lograr la emisión pulsada por amarre de pulsos pasivo en la cavidad Ti:Zafiro como la presentada en la Figura 3.1, se requiere de un elemento intracavidad cuya transmisividad o bien su fase (espacial o temporal) sea dependiente de la intensidad del haz intracavidad. Esto se logra mediante la absorción saturable de sólidos y tintes o por otros efectos ópticos no lineales. Algunos materiales presentan autoenfocamiento no lineal, el cual combinado con una apertura forman un proceso de automodulación pasiva cuya efectividad depende de la intensidad, la cual es usada para generar condiciones favorables para el amarre de modos. Los cristales de Titanio:Zafiro presentan tales propiedades.

El índice de refracción del medio Kerr, n, considerando que éste índice no cambia con respecto a la dirección de propagación del haz (z) y que no existe dependencia temporal se presenta en la Ecuación (4.1):

$$n(x,y) = n_0 + \frac{\partial n}{\partial T}\Delta T(x,y) + n_2 I(x,y). \tag{4.1}$$

donde

n_0 es el índice de refracción lineal del medio.

$\frac{\partial n}{\partial T}$ es la variación del índice de refracción con respecto a la temperatura.

$n_2 I(x,y)$ es el índice de refracción no lineal dependiente de la intensidad del haz intracavidad.

La distribución de temperatura causada por el haz de bombeo y la intensidad del haz intracavidad poseen perfiles de intensidad gaussianos que cambian su dimensión conforme se propagan los haces dentro del material. Esta dependencia espacial hace que el análisis de la cavidad sea difícil, y no ofrece una solución analítica salvo para cavidades láser en anillo [11] o para una cavidad simétrica en forma de X [12]. El enfoque de solución más usado es modelar el medio Kerr

como una lámina delgada cuyo índice de refracción varía de forma parabólica (Gradient-Index lens o GRIN lens), empleando una matriz ABCD [13] o utilizando un sistema de ecuaciones diferenciales acopladas [14]. El perfil del haz gaussiano puede aproximarse mediante una expansión por serie de Taylor o por un ajuste a una ecuación parabólica por el método de mínimos cuadrados [15]. Esto último puede extenderse para haces gaussianos elípticos [16].

4.1. Ecuación paraxial de "ducto" cuadrático

El modelado de los efectos de autoenfocamiento el el cristal Titanio:Zafiro causados por los haces intracavidad y de bombeo es difícil dado al perfil de intensidad gaussiano presentado en éstos. Una solución a este problema es aproximar los perfiles de intensidad gaussianos con una ecuación parabólica, como se muestra en la Figura 4.1. Esta aproximación será válida cerca del eje de propagación y puede fallar al acercarse a los bordes, pero simplifica significativamente el tratamiento matemático. Los efectos de autoenfocamiento introducen astigmatismo no lineal en la cavidad analizada con anterioridad.

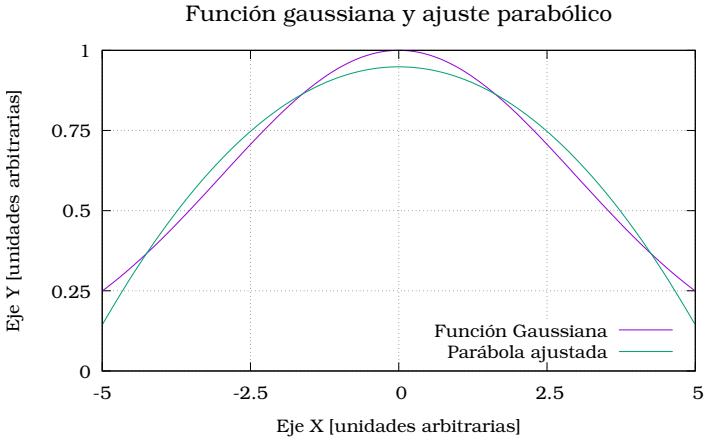


Figura 4.1: Comparación de curva gaussiana de la forma $y(x) = \exp\left(\frac{-x^2}{2\omega^2}\right)$ con una parábola de forma $y(x) = a_0 + a_1 x^2$. En este caso, $\omega = 3$, $a_0 = 0.9486$ y $a_1 = -0.0322$.

4.1.1. Ecuación paraxial del rayo

La ecuación del rayo (4.2) es usada para describir la trayectoria de un rayo de forma paramétrica mediante funciones dependientes del espacio [7]

$$\frac{d}{ds}\left(n\frac{d\mathbf{r}}{ds}\right) = \nabla n \tag{4.2}$$

donde

∇n es el gradiente del índice de refracción del medio con componentes cartesianas $\partial n/\partial x$, $\partial n/\partial y$ y $\partial n/\partial z$.

ds es una diferencial de trayectoria de la luz, $ds = \sqrt{dx^2 + dy^2 + dz^2}$.

$\mathbf{r}$ es un vector de posición formado por las funciones $x(s)$, $y(s)$ and $z(s)$.

En la aproximación paraxial, la trayectoria del rayo es casi paralela al eje de propagación (eje z) de forma que $ds \approx dz$, lo que simplifica la Ecuación (4.2) a

$$\frac{d}{dz}\left(n\frac{dx}{dz}\right) \approx \frac{\partial n}{\partial x} \tag{4.3a}$$

$$\frac{d}{dz}\left(n\frac{dy}{dz}\right) \approx \frac{\partial n}{\partial y} \tag{4.3b}$$

Dado $n = n(x, y, z)$ las Ecuaciones (4.3a) y (4.3b) pueden resolverse para las trayectorias $x(z)$ y $y(z)$.

4.1.2. Ecuación de "ducto" GRIN paraxial

Dado un índice de refracción como el mostrado en la Ecuación (4.4), substituyendo en las Ecuaciones (4.3a) y (4.3b) y considerando que $n \approx n_0$, se obtiene el sistema de ecuaciones diferenciales presentado en las Ecuaciones (4.5a) y (4.5b).

$$n(x, y) = n_0\left(1 - \frac{x^2}{2h_x^2} - \frac{y^2}{2h_y^2}\right) \tag{4.4}$$

$$\frac{d^2x}{dz^2} = \frac{1}{h_x^2}x \tag{4.5a}$$

$$\frac{d^2y}{dz^2} = \frac{1}{h_y^2}y \tag{4.5b}$$

Resolviendo el sistema de ecuaciones diferenciales para x y y y calculando la primera derivada, se encuentra la matriz de propagación de rayos paraxial para un ducto GRIN con un perfil elíptico.

$$\begin{pmatrix} x_o \\ n_0\theta_o \end{pmatrix} = \begin{pmatrix} \cos\frac{z}{h_x} & \frac{h_x}{n_0}\sin\frac{z}{h_x} \\ -\frac{n_0}{h_x}\sin\frac{z}{h_x} & n_0\cos\frac{z}{h_x} \end{pmatrix}\begin{pmatrix} x_i \\ n_0\theta_i \end{pmatrix} \tag{4.6}$$

$$\begin{pmatrix} y_o \\ n_0\phi_o \end{pmatrix} = \begin{pmatrix} \cos\frac{z}{h_y} & \frac{h_y}{n_0}\sin\frac{z}{h_y} \\ -\frac{n_0}{h_y}\sin\frac{z}{h_y} & n_0\cos\frac{z}{h_y} \end{pmatrix}\begin{pmatrix} y_i \\ n_0\phi_i \end{pmatrix} \tag{4.7}$$

Si $z \to \Delta z \approx 0$ entonces $\cos z \approx 1$ y $\sin z \approx z$ podemos reescribir las Ecuaciones (4.6) y (4.7) como:

$$\begin{pmatrix} x_o \\ n_0\theta_o \end{pmatrix} = \begin{pmatrix} 1 & \frac{\Delta z}{n_0} \\ -\frac{n_0}{h_x^2}\Delta z & 1 \end{pmatrix} \begin{pmatrix} x_i \\ n_0\theta_i \end{pmatrix} \tag{4.8}$$

$$\begin{pmatrix} y_o \\ n_0\phi_o \end{pmatrix} = \begin{pmatrix} 1 & \frac{\Delta z}{n_0} \\ -\frac{n_0}{h_y^2}\Delta z & 1 \end{pmatrix} \begin{pmatrix} y_i \\ n_0\phi_i \end{pmatrix} \tag{4.9}$$

Las Ecuaciones (4.8) y (4.9) representan una lente GRIN delgada con un perfil de índice de refracción elíptico. Esta aproximación es útil porque nos permite describir los cambios que sufre el radio del haz astigmático al propagarse en el medio de ganancia en los planos tangencial y sagital.

4.2. Aproximación del efecto Kerr

El término correspondiente para el efecto Kerr en la ecuación del índice de refracción no lineal

$$n(x,y) = n_0 + \frac{\partial n}{\partial T}\Delta T(x,y) + n_2 I(x,y). \tag{4.1}$$

es $n_2 I$, donde I es la distribución de intensidad espacial transversal del haz intracavidad. Su valor es expresado en la Ecuación (4.10)

$$I(x,y) = \frac{2P_L}{\pi\omega_t\omega_s}\exp\left(-\frac{2x^2}{{\omega_t}^2} - \frac{2y^2}{{\omega_s}^2}\right) \tag{4.10}$$

donde

$I(x,y)$ es la intensidad del haz intracavidad en el medio.

P_L es la potencia del haz intracavidad en el cristal.

ω_t, ω_s son los radios del haz en los planos tangencial y sagital del medio.

Para aproximar el término exponencial en la Ecuación (4.10) se emplea un ajuste parabólico por mínimos cuadrados ponderados, como se presenta en el trabajo de Magni [15] y Bridges [16]:

$$\int_0^\infty \int_0^\infty \left(c_1 + c_2x^2 + c_3y^2 - \exp\left[-\frac{2x^2}{\omega_t^2} - \frac{2y^2}{\omega_s^2}\right]\right)^2 \exp\left[-\frac{2x^2}{\omega_t^2} - \frac{2y^2}{\omega_s^2}\right] dxdy \tag{4.11}$$

Al resolver la Ecuación (4.11) y encontrando el valor mínimo de la función encontramos los siguientes valores para c_1, c_2, c_3

$$c_1 = \frac{3}{4} \tag{4.12a}$$

$$c_2 = -\frac{1}{2\omega_t^2} \tag{4.12b}$$

$$c_3 = -\frac{1}{2\omega_s^2} \tag{4.12c}$$

La intensidad del haz intracavidad dentro del medio de ganancia es aproximada por la Ecuación (4.13).

$$I(x,y) = \frac{2P_L}{\pi\omega_t\omega_s}\exp\left(-\frac{2x^2}{{\omega_t}^2} - \frac{2y^2}{{\omega_s}^2}\right) \approx \frac{2P_L}{\pi\omega_t\omega_s}\left(c_1 + c_2x^2 + c_3y^2\right) \tag{4.13}$$

Dado que el término correspondiente al efecto Kerr en la ecuación no lineal es n_2I, donde el valor reportado de n_2 es muy pequeño ($n_2 \approx 20 \times 10^{-20}$) [m^2/W] [13], se ignora el término lineal c_1 para nuestra aproximación.

4.3. Aproximación de lente térmica

De la Ecuación del índice de refracción no lineal (4.1) la variación de índice de refracción por causas térmicas es representado por $(\partial n/\partial T)\Delta T(x,y)$, donde $(\partial n/\partial T)$ es el cambio del índice de refracción con respecto a la temperatura y $\Delta T(x,y)$ es la distribución de cambio de temperatura en una lámina delgada del cristal Titanio:Zafiro.

Para determinar $\Delta T(x,y)$ resolvemos la ecuación de calor en estado estable para una placa delgada [17]:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{A}{k_{th}C_p\rho} = 0 \tag{4.14}$$

donde

A es el término de fuente de calor

k_{th} es la difusividad térmica del material.

C_p es el calor específico del material.

ρ es la densidad del material.

El término de fuente A se define como

$$A = \frac{Q}{tV} \tag{4.15}$$

donde Q es la cantidad de calor (energía) que entra a un volumen V durante un tiempo t. Notando que la potencia se define como energía sobre unidad de tiempo y que la irradiancia I se describe por la potencia del haz de bombeo por unidad de área, se tiene que

$$A = \frac{I_P * rea}{V} = \frac{I_P}{L} \tag{4.16}$$

Para un cristal Titanio:Zafiro el cual es bombeado por uno de los extremos usando un haz de bombeo astigmático, asumiendo que el cristal no absorbe el haz intracavidad, y que sólo una fracción de la potencia del láser de bombeo es convertida en calor se puede construir la Ecuación (4.17)

$$A = \frac{2\chi P_P}{L\pi\omega_{Pt}\omega_{Ps}} \exp\left[-\frac{2x^2}{\omega_{Pt}^2} - \frac{-2x^2}{\omega_{Ps}^2} - \alpha\xi\right] \tag{4.17}$$

donde

χ es la razón de calor disipado a potencia de bombeo.

L es la longitud del cristal.

P_P es la potencia del láser de bombeo dentro del medio.

ω_{Pt}, ω_{Ps} son los radios del haz de bombeo en los planos tangencial y sagital.

α es el coeficiente de absorción del material.

ξ es la profundidad de penetración medida desde el extremo bombeado.

Utilizando la Ecuación (4.17) como término fuente para resolver numéricamente la ecuación diferencial parcial dada por (4.14) usando diversos valores de ξ se encuentra la función de distribución de temperatura $T(x, y)$ para una lámina delgada del cristal Titanio:Zafiro.

La distribución de cambio de temperatura es representada por

$$\Delta T(x, y) = T(x, y) - T(0, 0) \tag{4.18}$$

Donde $T(0, 0)$ es la temperatura en el eje óptico. Los datos resultantes pueden ser ajustados a una elípse [18] de la forma $a_0 + a_1x^2 + a_2y^2$, dado que la fuente de calor empleada posee una forma gaussiana la cual se puede aproximar por una elípse. Debido a que el término lineal es pequeño, es despreciado para nuestro propósito.

Capítulo 5

Técnicas de modelado no lineal numéricas

La dependencia espacial no lineal del índice de refracción del cristal Titanio:Zafiro no ofrece una solución analítica. En esta sección se derivarán los métodos de propagación desacoplada matricial [13] y por ecuaciones diferenciales [14] además de el método de propagación acoplada matricial dentro del medio [15].

Para encontrar modos estables para una cavidad no lineal se emplea una solución semilla obtenida del análisis de onda continua; este modo es propagado hasta que alcance la condición de autoconsistencia, esto es, que dado un haz gaussiano que se propague por la cavidad describiendo un viaje completo, la salida debe ser el mismo haz, como se muestra en la Ecuación .

$$q_2 = \frac{Aq_1 + B}{Cq_1 + D} = q_1. \tag{5.1}$$

Dado a que se realiza una simulación numérica, la Ecuación tiene que ser adaptada. La simulación numérica se iterará hasta encontrar radios de haz ω_n y ω_{n-1} que satisfagan la Ecuación (5.2).

$$\frac{|\omega_n - \omega_{n-1}|}{\omega_n} \times 100\,\% \leq Umbral \tag{5.2}$$

Se discuten tres métodos para modelar el cristal Titanio:Zafiro: El método matricial desacoplado, el método por un sistema de ecuaciones diferenciales desacoplado y el método matricial acoplado. Estos métodos difieren en cómo el haz intracavidad es propagado dentro del medio de ganancia.

5.1. Método matricial desacoplado

Este método asume que el haz que se propaga dentro del medio Kerr es circular. Se divide el cristal en un número grande pero finito de cortes transver-

sales delgados de forma tal que las Ecuaciones (4.8) y (4.9) sean válidas. Debido a que el haz de entrada es circular, los factores parabólicos $1/h_x^2$ y $1/h_y^2$ son iguales. Una representación genérica de la matriz de propagación de rayos es presentada en la Ecuación (5.3). Para simplificar el análisis el elemento térmico de autoenfocamiento no es incluído.

$$\begin{pmatrix} r_o \\ n_0\theta_o \end{pmatrix} = \begin{pmatrix} 1 & \frac{\Delta z}{n_0} \\ -\frac{n_0}{h^2}\Delta z & 1 \end{pmatrix} \begin{pmatrix} r_i \\ n_0\theta_i \end{pmatrix} \tag{5.3}$$

Al sustituir las componentes de la matriz de la Ecuación (5.3) en la Ecuación (2.26) se obtiene

$$q' = \frac{h^2(q + \Delta z)}{h^2 - q\Delta z} \tag{5.4}$$

Debido a que el haz es circular, se define $r^2 = x^2 + y^2$, $\omega_t = \omega_s = \omega$, y $1/{h_x}^2 = 1/{h_y}^2 = 1/h^2$, entonces se substituye en la Ecuación (4.13) la cual da el perfil de intensidad responsable del efecto Kerr. El índice de refracción dado por este efecto es $n_2 I(x, y)$, obteniendo

$$\frac{1}{h^2} = -\frac{n_2 P_L}{2n_0\pi\omega^4} \tag{5.5}$$

El parámetro complejo del haz q' cambia cuando el haz se propaga a través de la lente GRIN delgada, de forma que el factor parabólico $1/h^2$ tiene que ser recalculado cada paso.

5.2. Sistema de ecuaciones diferenciales

El método consiste en derivar un sistema de ecuaciones diferenciales que describa el cambio del radio del haz y el radio de curvatura del frente de onda durante la propagación dentro del cristal Titanio:Zafiro a partir de la matriz de propagación de rayos de una lente GRIN delgada [14].

Usando las componentes de la matriz de la Ecuación (5.3) y la Ecuación (5.4), se busca por una $q' = q + dq$:

$$\frac{q'}{n_0} = \frac{Aq/n_0 + B}{Cq/n_0 + D} \tag{5.6a}$$

$$\frac{q'}{n_0}\left(-\frac{dzq}{h^2} + 1\right) = \frac{q + dz}{n_0} \tag{5.6b}$$

$$\frac{q' - q}{n_0} = \frac{dz}{n_0}\left(1 + \frac{qq'}{h^2}\right) \tag{5.6c}$$

$$\tag{5.6d}$$

Recordando que $q' = q + dq$

$$\frac{dq}{n_0} = \frac{dz}{n_0}\left(1 + \frac{q^2 + qdq}{h^2}\right) \tag{5.6e}$$

Como $dq \to 0$, $qdq \to 0$, se encuentra que

$$\frac{dq}{dz} = 1 + \frac{q^2}{h^2} \tag{5.6f}$$

Sea $p = 1/q = p_R + jp_I$, $q = 1/p$, tomando la primer derivada de q con respecto a z, y substituyendo en la Ecuación (5.6f):

$$\frac{dp_R(z)}{dz} + j\frac{dp_I(z)}{dz} = -p_R^2 + p_I^2 - 2jp_Rp_I - \frac{1}{h^2} \tag{5.7a}$$

Separando la parte real e imaginaria de la Ecuación (5.7a):

$$\frac{dp_R}{dz} = -p_R^2 + p_I^2 - \frac{1}{h^2} \tag{5.7b}$$

$$\frac{dp_I}{dz} = -2p_Rp_I \tag{5.7c}$$

El cambio en el factor parabólico $1/h^2$ (Ecuación (5.5)) es incluído mediante manipulación algebráica notando que $p_I^2 = \lambda^2/(n_0^2\pi^2\omega^4)$. El sistema de ecuaciones diferenciales final se expresa en las Ecuaciones (5.8a) y (5.8b).

$$\frac{dp_R}{dz} = -p_R^2 + p_I^2\left(1 - \frac{n_0n_2\pi P_L}{2\lambda^2}\right) \tag{5.8a}$$

$$\frac{dp_I}{dz} = -2p_Rp_I \tag{5.8b}$$

La propagación no lineal del haz dentro del medio Kerr se modela por medio del problema de valor inicial descrito por las Ecuaciónes (5.8a) y (5.8b). El sistema no lineal puede resolverse numericamente utilizando el algoritmo de Runge-Kutta de cuarto orden.

5.3. Método matricial acoplado

Éste método surge a partir de la necesidad de evaluar la propagación de haces astigmáticos al interior del medio de ganancia. El haz astigmático intracavidad está dado por la Ecuación (5.9)

$$I(x,y) = \frac{2P_L}{\pi\omega_t\omega_s}\exp\left[-\frac{2x^2}{\omega_t^2} - \frac{2y^2}{\omega_s^2}\right]. \tag{5.9}$$

Se aprecia de la Ecuación (5.9) que el análisis de propagación de haces para los planos tangencial y sagital no es independiente entre ellos. La intensidad -

y por ende el índice de refracción - dependen del radio del haz en ambos planos [16]. Al usar las herramientas matemáticas descritas en el Capítulo 4 se puede obtener un par de matrices de propagación de rayos que describan la propagación astigmática del haz intracavidad y de bombeo dentro del cristal Titanio:Zafiro al realizar una propagación de rayos acoplada (los planos tangencial y sagital son dependientes entre sí).

De las Ecuaciones (4.1) y (4.4) se encuentra que

$$\frac{1}{h_x^2} = \frac{2n_2P_L}{n_0\pi\omega_t^3\omega_s} - \frac{2a_1(\xi)}{n_0}\frac{\partial n}{\partial T} \tag{5.10a}$$

$$\frac{1}{h_y^2} = \frac{2n_2P_L}{n_0\pi\omega_t\omega_s^3} - \frac{2a_2(\xi)}{n_0}\frac{\partial n}{\partial T}. \tag{5.10b}$$

Para incluir los efectos térmicos en el índice de refracción uno debe determinar $\Delta T(x,t)$ para cada valor de ξ al propagar el haz de bombeo, creando tantas superficies como pasos usados para la propagación de haces antes de buscar modos pulsados estables. Debido a que el cristal es bombeado por uno de los extremos, se debe tener cuidado de aplicar los valores correspondientes de $a_1(\xi)$ y $a_2(\xi)$ durante la propagación de viaje redondo.

5.4. Ejemplo numérico

Se analiza la cavidad descrita en el la Sección 3.5. Para ésta evaluación, se relizó la corrección de astigmatismo lineal para el límite punto-plano (Figura 3.7) y posteriormente se variarán los valores ϵ_1 y ϵ_2 para determinar la configuración menos astigmática en emisión pulsada como se muestra en la Figura (5.1). Éste análisis comprende los modos en los espejos planos 1 y 2 (EM_1 and EM_2) en los planos tangencial y sagital.

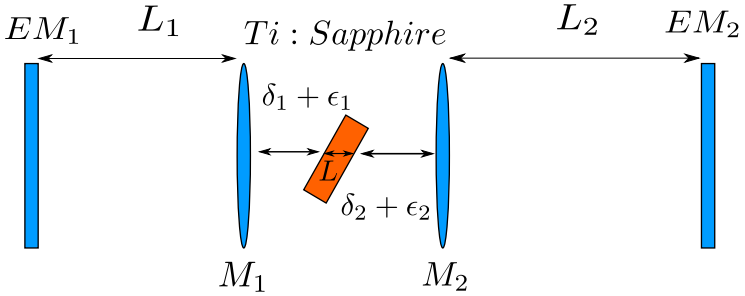


Figura 5.1: Cavidad Titanio:Zafiro usada para el análisis. Una vez que el astigmatismo de emisión continua es corregido, se modifican los valores de ϵ_1 y ϵ_2 para encontrar la configuración menos astigmática en EM_1 o en EM_2.

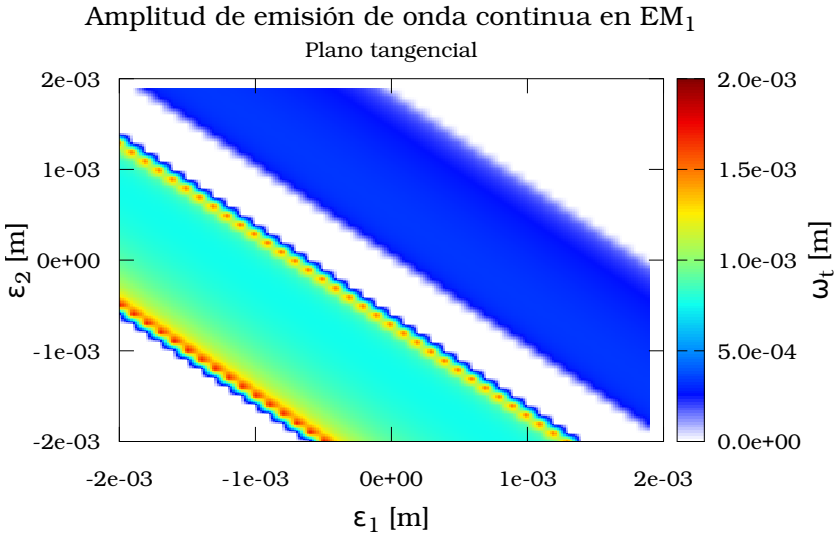
Los parámetros de ésta cavidad hipotética son L_1=1 [m], L_2=1.35 [m], L= 10 [mm], λ_0= 810 [nm], $n_{0_{\lambda_0}}$=1.7598, n_2=3×10^{-20} [m^2/W], f_1=50[mm],

f_2=50[mm] y P_L=100 [kW]. Para el análisis térmico considérese que λ_P=535 [nm], $n_{0_{\lambda_P}}$=1.7715, $k_t h$=1.3×10^{-5} [m^{2}s^{-1}], C_p=775 [J/kgK], ρ=3990 [kg/m^3], $\partial n/\partial T$=1×10^{-6} [K^{-1}], χ=0.6, P_P=8 [W], α=1[cm^{-1}], el cristal es un prisma cuadrado que mide 6 [mm] de ancho y alto, a 293-15 [K]. Asumase que el cristal se bombea en el extremo de cara a M_1, y que una vez que el haz de bombeo se haya refractado dentro del cristal, los radios del haz son ω_{P_t}=ω_{P_s}=2 [mm] sin importar los valores de ϵ_1 y ϵ_2. La mayor parte de la información se encuentra en los trabajos de Georgiev [13] y Mehendale [18].

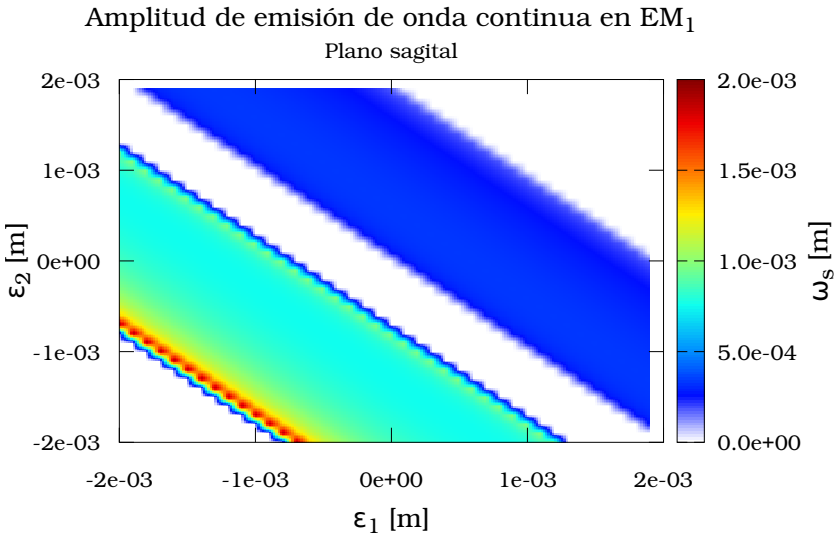
5.4.1. Análisis de onda continua

Al emplear las ecuaciones discutidas en el Capítulo 3 para encontrar las distancias δ_1, δ_2, y los semiángulos con respecto a la normal al plano de incidencia θ_1 y θ_2; al variar ϵ_1 y ϵ_2 se obtienen las soluciones de onda continua para la cavidad para los planos tangencial y sagital; al graficar estas soluciones se obtienen superficies de estabilidad, como se observa en las Figuras 5.2a y 5.2b. Al dividir los valores de las superficies de estabilidad de los planos tangencial y sagital, elemento a elemento, se obtiene la Figura 5.4, donde la razón de ω_t/ω_s representa que tan astigmático es el modo de propagación. En todas las gráficas las secciones inestables se denotan con blanco.

Del análisis de onda continua se obtiene una $\delta_1 \approx 50.8$ [mm], $\delta_2 \approx 48.1$ [mm], $\theta_1 \approx 10.65$ grados y $\theta_2 \approx 11.20$ grados. Las soluciones encontradas para la operación en onda continua se utilizan como valores semilla para la propagación en el régimen pulsado.

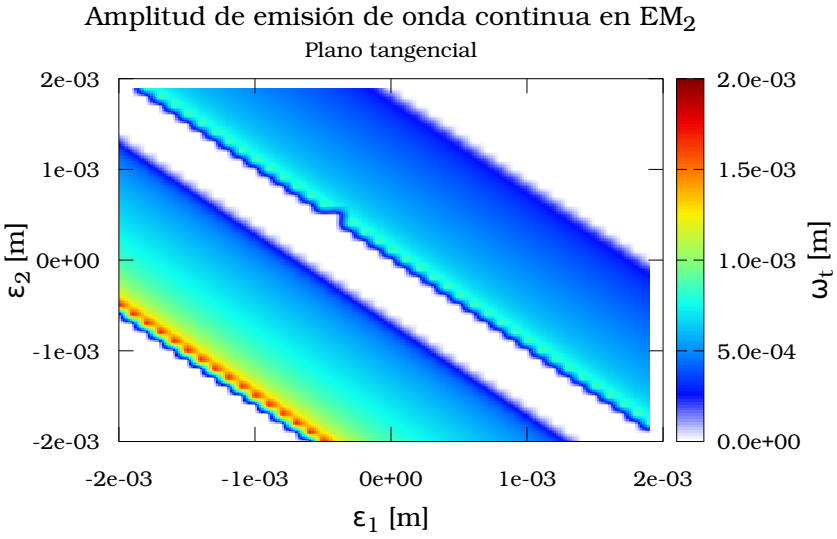


(a) Radio de haz en el plano tangencial para EM_1.

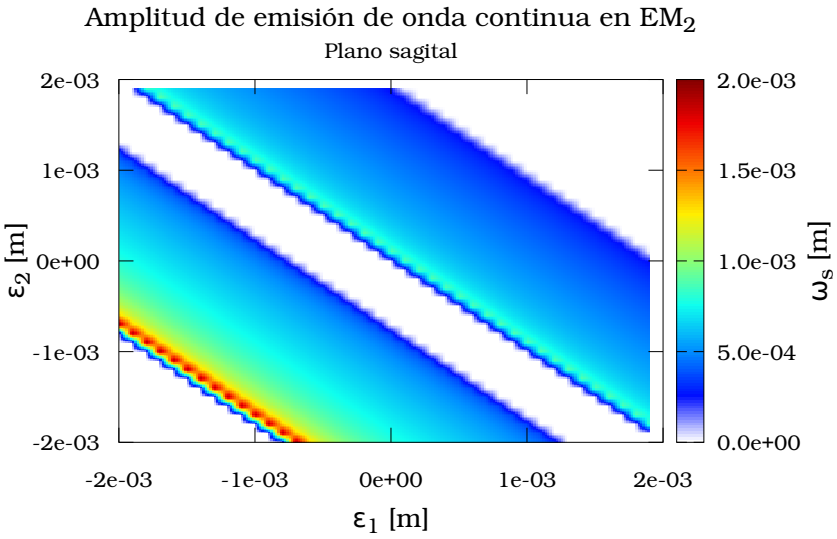


(b) Radio de haz en el plano sagital para EM_1.

Figura 5.2: Mapas de ω_t y ω_s en EM_1 para emisión continua.

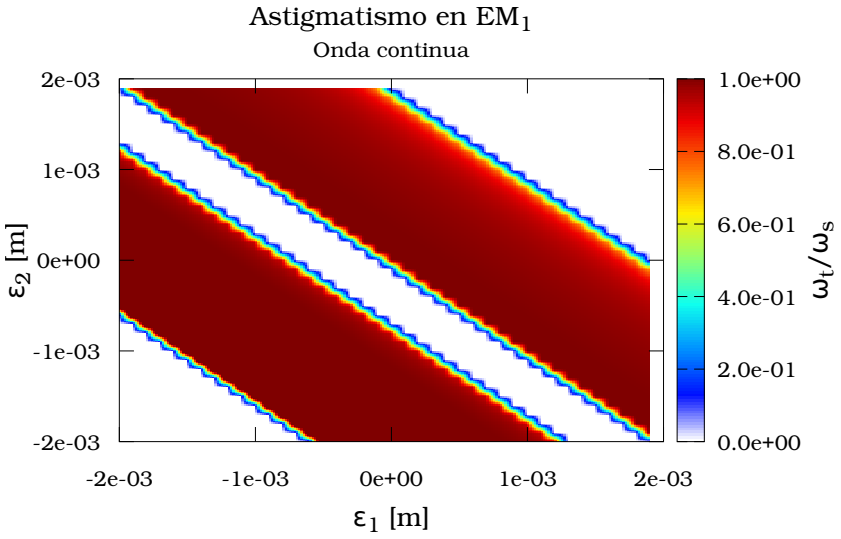


(a) Radio de haz en el plano tangencial para EM_2.

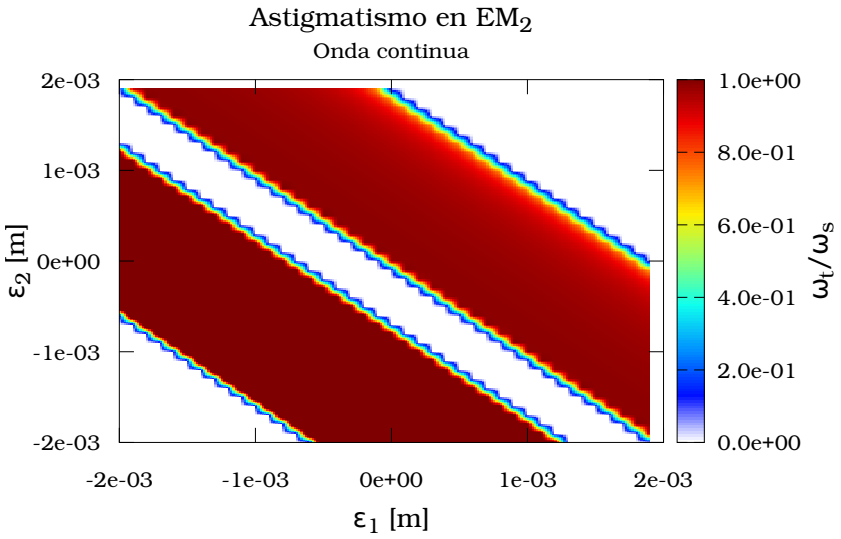


(b) Radio de haz en el plano sagital para EM_2.

Figura 5.3: Mapas de ω_t y ω_s en EM_2 para emisión continua.



(a) Mapa para EM_1.



(b) Mapa para EM_2.

Figura 5.4: Estabilidad y astigmatismo para EM_1 y EM_2 para la operación de onda continua. La cavidad es no astigmática si $\omega_t/\omega_s = 1$. Las regiones inestables se denotan por el color blanco.

5.4.2. Análisis en régimen pulsado.

El análisis en régimen pulsado se realiza al emplear los modelos del cristal Titanio:Zafiro discutidos en las Secciones 5.1, 5.2 y 5.3. El método matricial acoplado es calculado con y sin lente térmica presente. Todos los algoritmos se detienen cuando el error obtenido es menos a un umbral de 0.001 % o cuando se alcanza el número máximo de iteraciones (20×10^3) [19]. Para la propagación no lineal dentro del cristal Titanio:Zafiro se usaron 1000 láminas delgadas para modelar el autoenfocamiento por efectos Kerr y térmico.

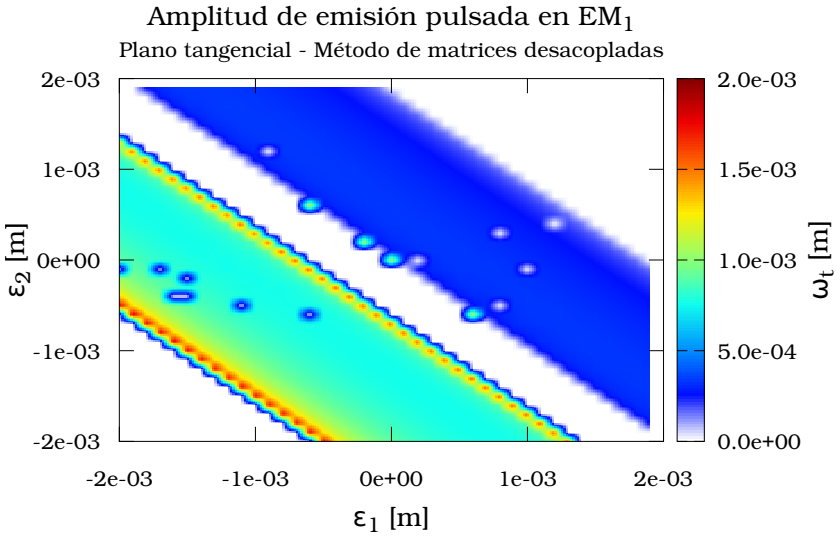
La corrección de astigmatismo en el régimen pulsado se logra al buscar valores de ϵ_1 y ϵ_2 tal que ω_t/ω_s sea lo más cercano posible a la unidad.

$$\frac{|\omega_n - \omega_{n-1}|}{\omega_n} \times 100\,\% \leq Umbral \tag{5.11}$$

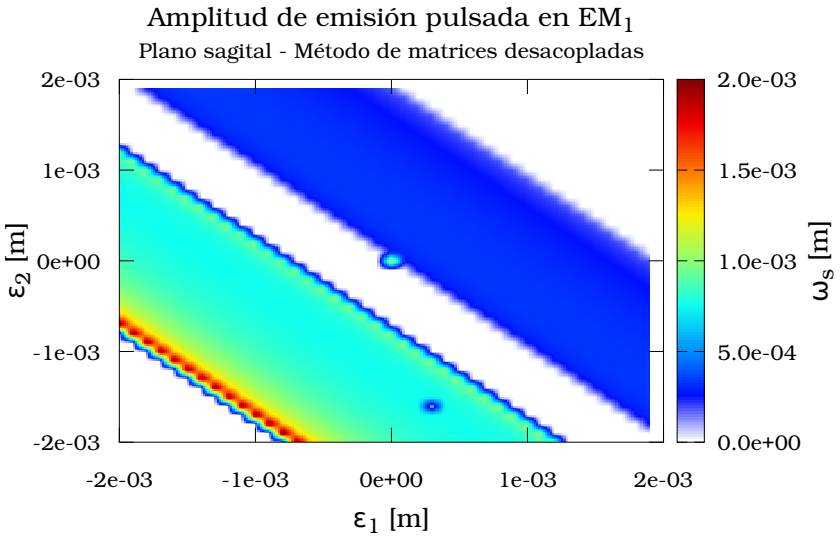
Los mapas resultantes se muestran en las Figuras 5.7, 5.10, 5.13 y 5.16.

Las regiones de estabilidad obtenidas a partir del análisis de emisión continua (Figuras 5.2 y 5.3) y emisión pulsada utilizando los métodos matricial desacoplado (Figuras 5.5 y 5.6) y por ecuaciones diferenciales son (Figuras 5.8 y 5.9) muy similares entre sí. La corrección de astigmatismo mostrada en las Figuras 5.4, 5.7, 5.10, 5.13 y 5.16 sólo aplica para el espejo de referencia (EM_1 o EM_2) Al variar la posición del cristal ajustando los valores de ϵ_1 y ϵ_2 se puede corregir astigmatismo como se muestra en las Figuras 5.4, 5.7, 5.10, 5.13 y 5.16; sin embargo, el diseñador debe considerar que puede cambiar el radio del haz en el espejo donde se corrige dicho astigmatismo. Los puntos de mayor amplitud observados en la Figura 5.14 sugieren que se requiere una mayor cantidad de iteraciones para que el análisis converja.

Método matricial desacoplado

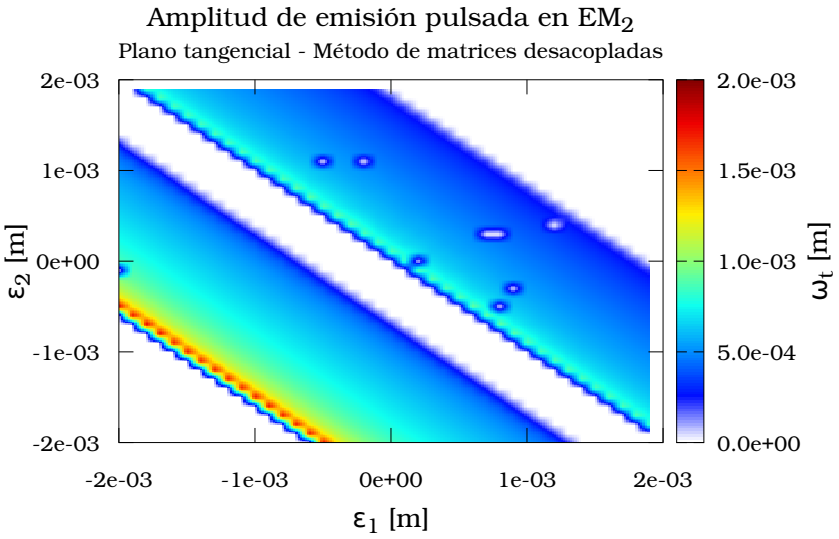


(a) Radio de haz en el plano tangencial para EM_1.

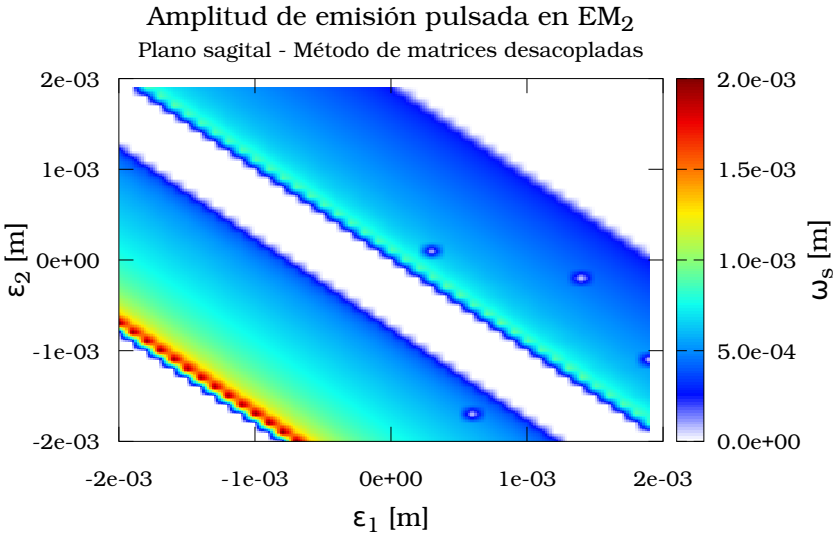


(b) Radio de haz en el plano sagital para EM_1.

Figura 5.5: Mapas de ω_t y ω_s en EM_1 para emisión pulsada utilizando el método de matrices desacopladas.

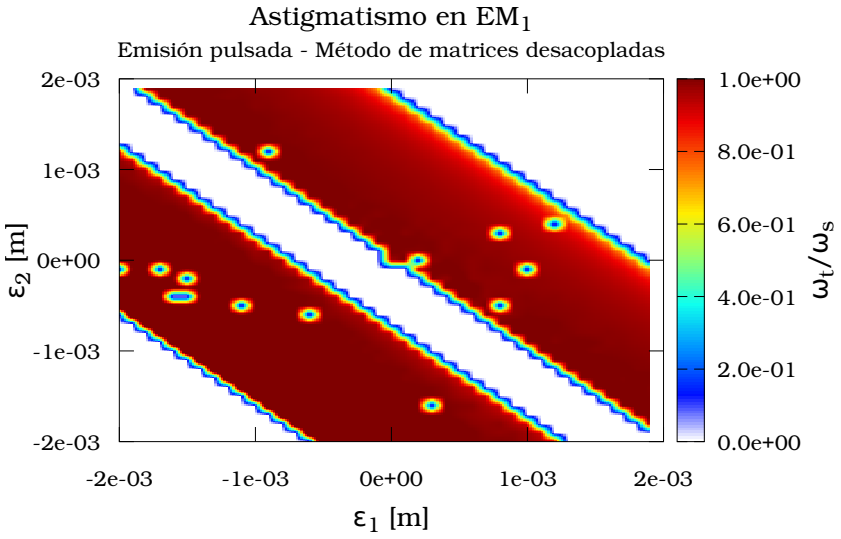


(a) Radio de haz en el plano tangencial para EM_2.

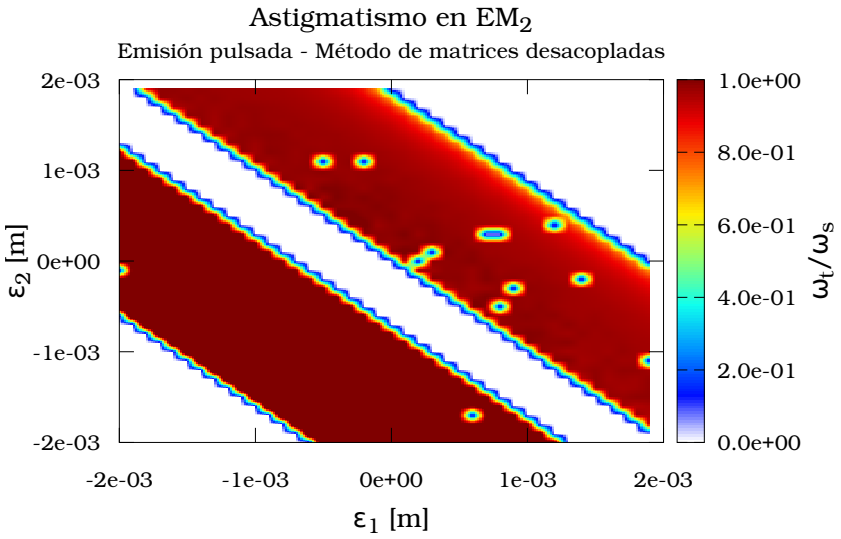


(b) Radio de haz en el plano sagital para EM_2.

Figura 5.6: Mapas de ω_t y ω_s en EM_2 para emisión pulsada utilizando el método de matrices desacopladas.



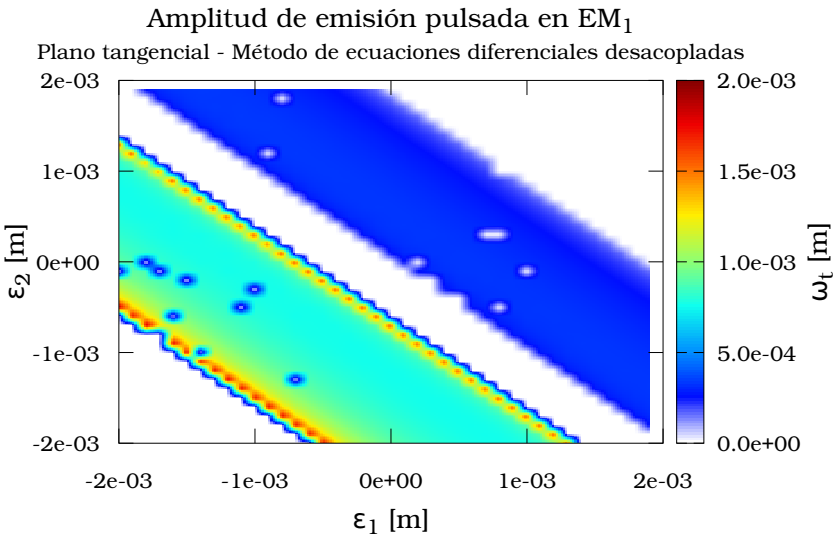
(a) Mapa de astigmatismo para EM_1.



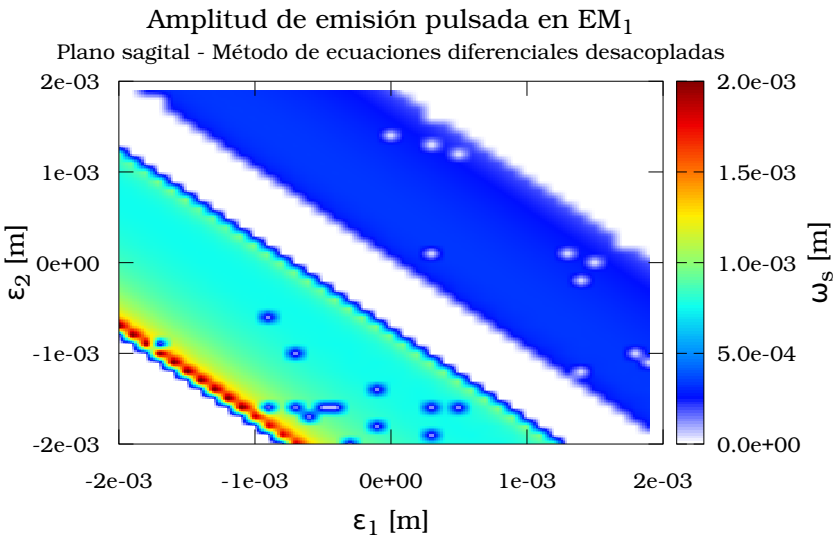
(b) Mapa de astigmatismo para EM_2.

Figura 5.7: Regiones de astigmatismo y estabilidad para EM_1 and EM_2 en operación pulsada utilizando el método matricial desacoplado. La cavidad es no astigmática si $\omega_t/\omega_s = 1$.

Método de ecuaciones diferenciales

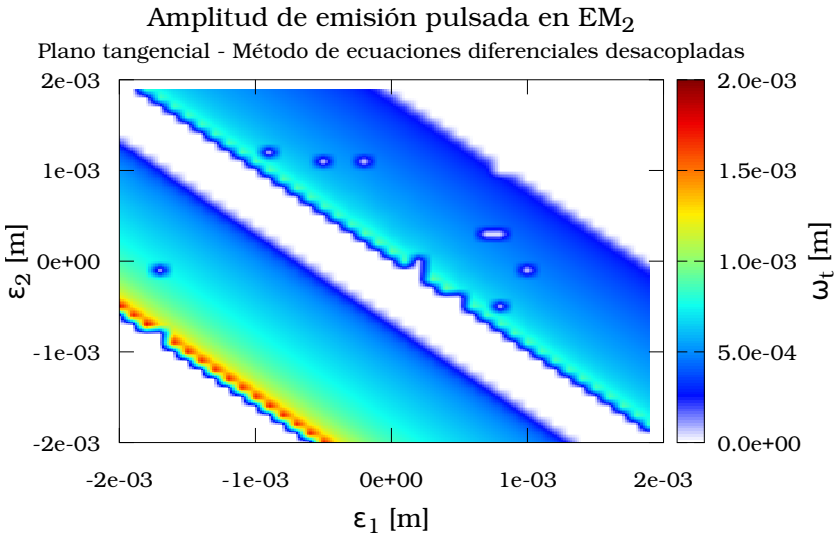


(a) Radio de haz en el plano tangencial para EM_1.

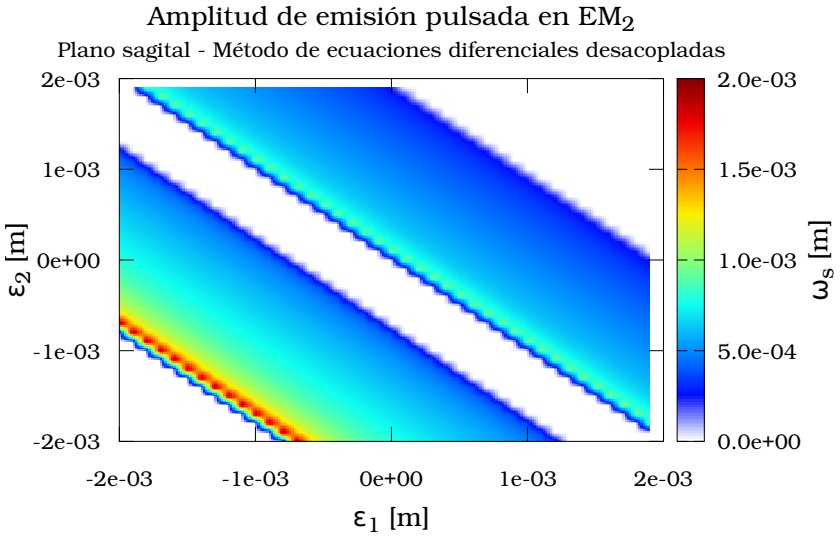


(b) Radio de haz en el plano sagital para EM_1.

Figura 5.8: Mapas de ω_t y ω_s en EM_1 para emisión pulsada utilizando el método de ecuaciones diferenciales.

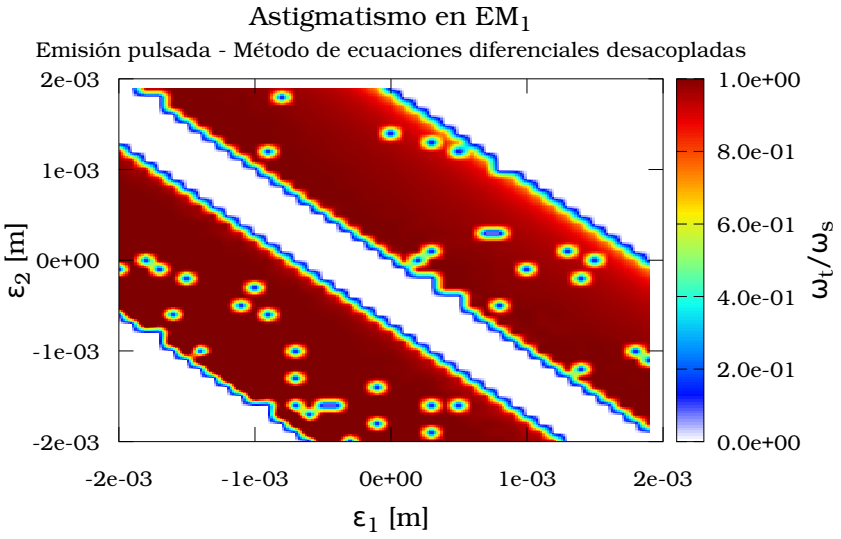


(a) Radio de haz en el plano tangencial para EM_2.

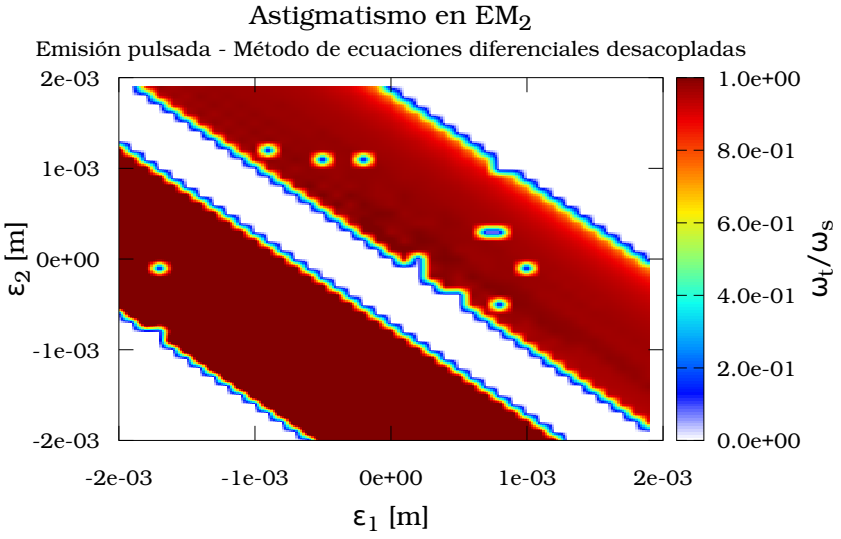


(b) Radio de haz en el plano sagital para EM_2.

Figura 5.9: Mapas de ω_t y ω_s en EM_2 para emisión pulsada utilizando el método de ecuaciones diferenciales.



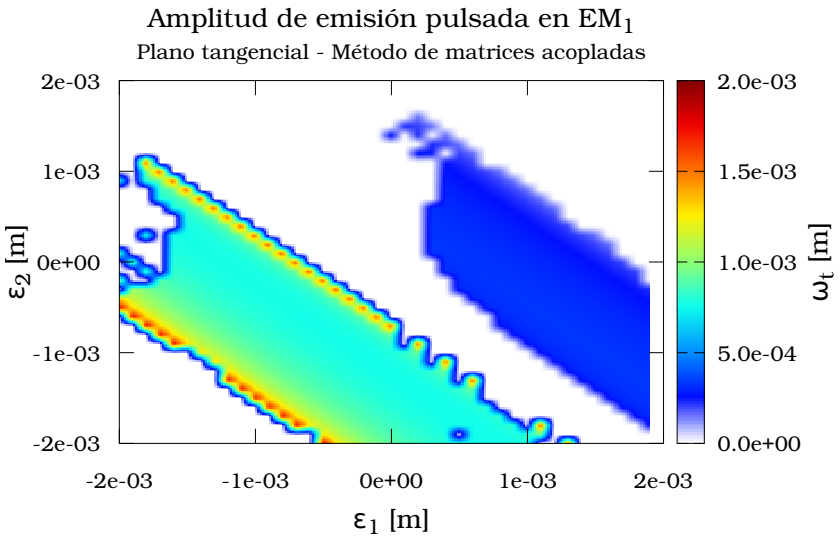
(a) Mapa de astigmatismo para EM_1.



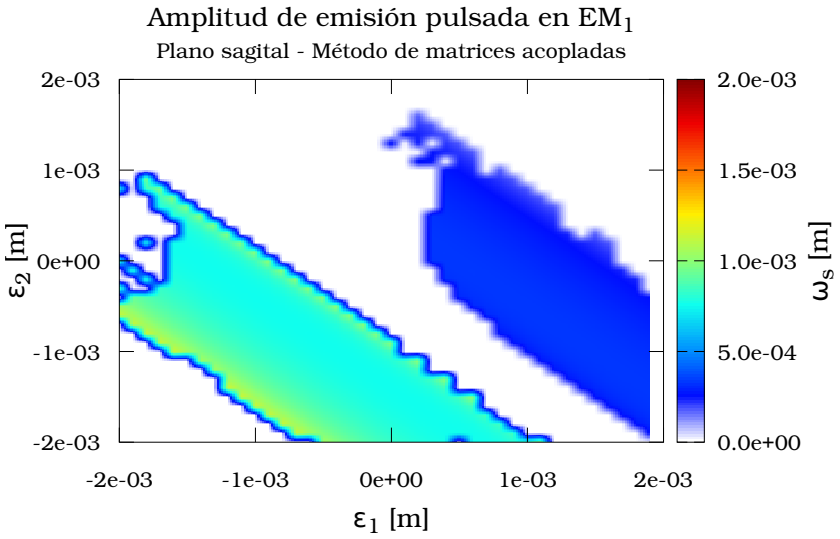
(b) Mapa de astigmatismo para EM_2.

Figura 5.10: Regiones de astigmatismo y estabilidad para EM_1 and EM_2 en operación pulsada utilizando el método de ecuaciones diferenciales. La cavidad es no astigmática si $\omega_t/\omega_s = 1$.

Método matricial acoplado

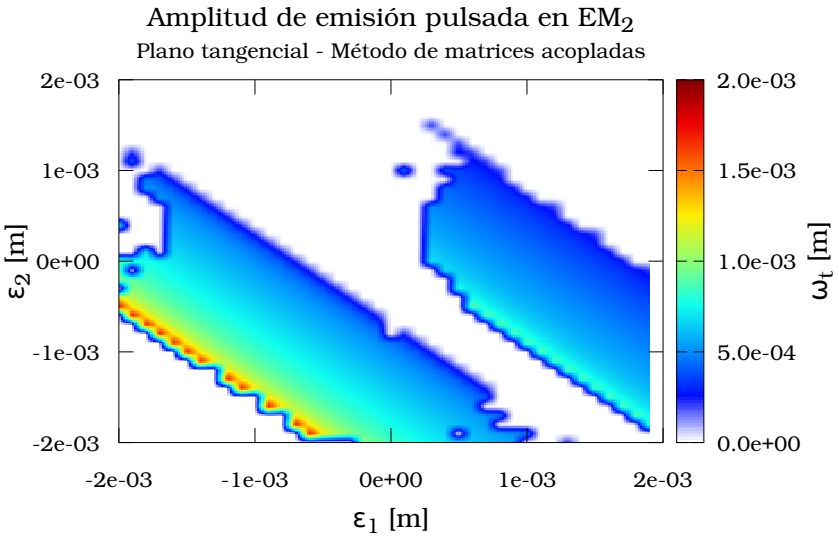


(a) Radio de haz en el plano tangencial para EM_1.

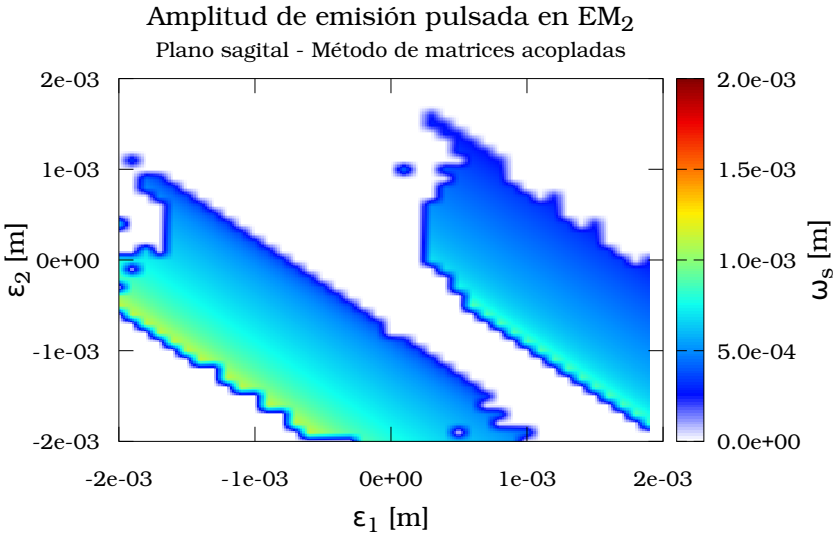


(b) Radio de haz en el plano sagital para EM_1.

Figura 5.11: Mapas de ω_t y ω_s en EM_1 para emisión pulsada utilizando el método de matrices acopladas.

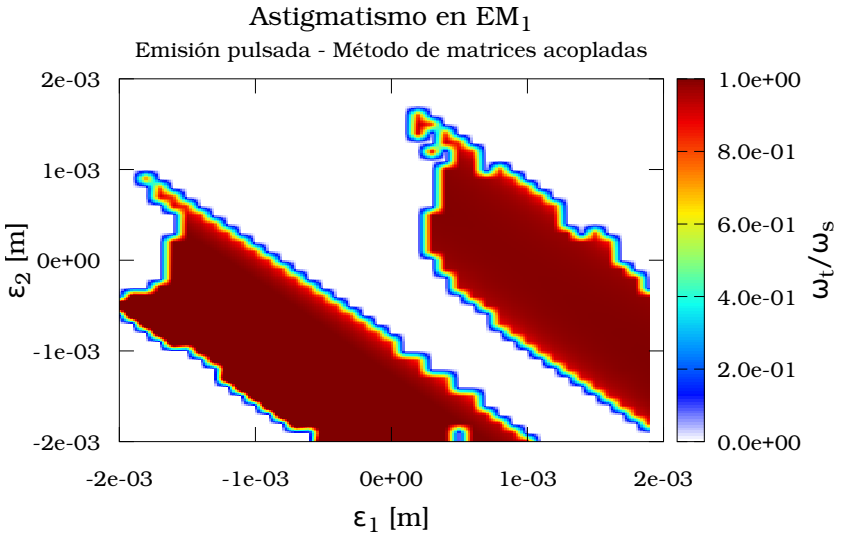


(a) Radio de haz en el plano tangencial para EM_2.

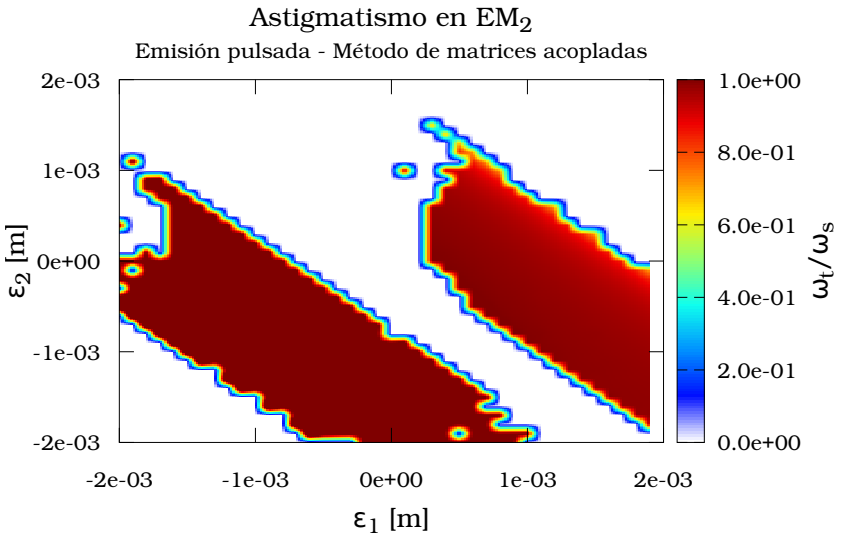


(b) Radio de haz en el plano sagital para EM_2.

Figura 5.12: Mapas de ω_t y ω_s en EM_2 para emisión pulsada utilizando el método de matrices acopladas.



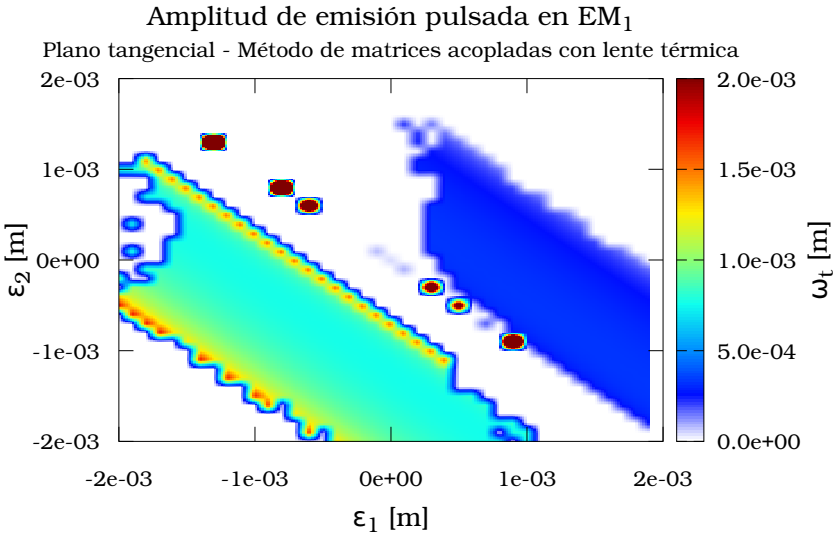
(a) Mapa de astigmatismo para EM_1.



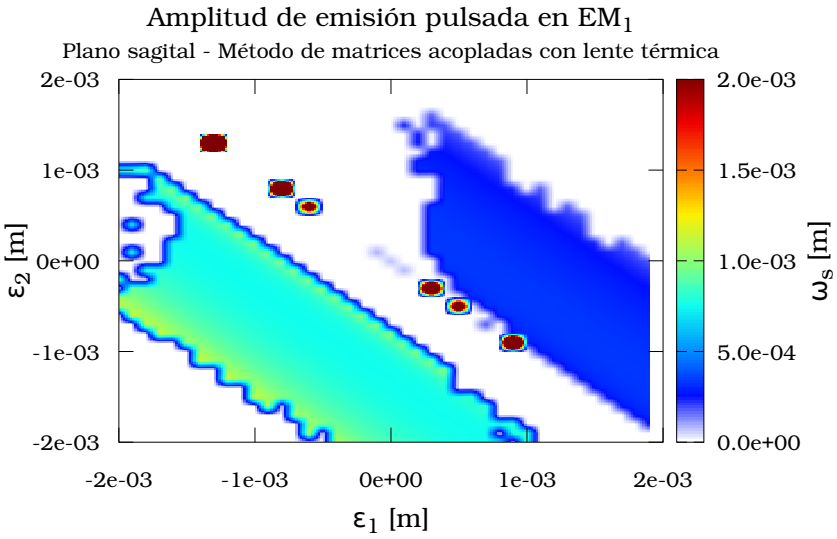
(b) Mapa de astigmatismo para EM_2.

Figura 5.13: Regiones de astigmatismo y estabilidad para EM_1 and EM_2 en operación pulsada utilizando el método de matrices acopladas. La cavidad es no astigmática si $\omega_t/\omega_s = 1$.

Método matricial acoplado con lente térmica

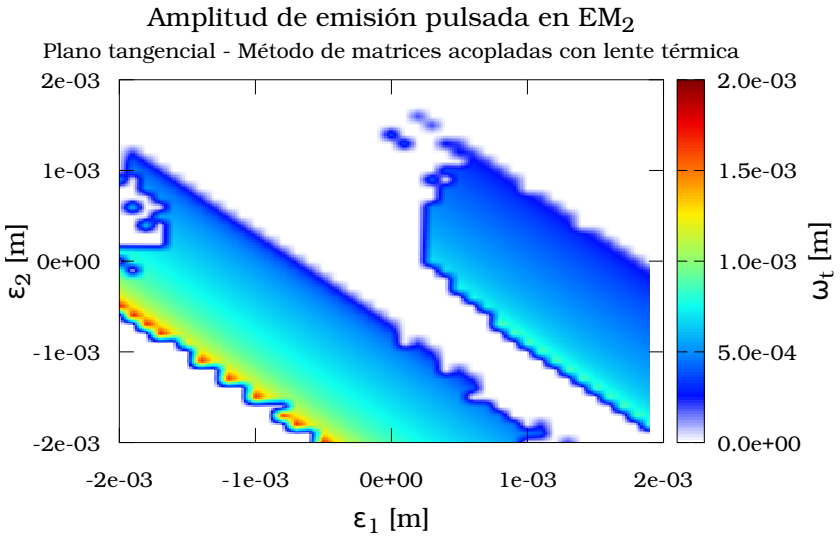


(a) Radio de haz en el plano tangencial para EM_1.

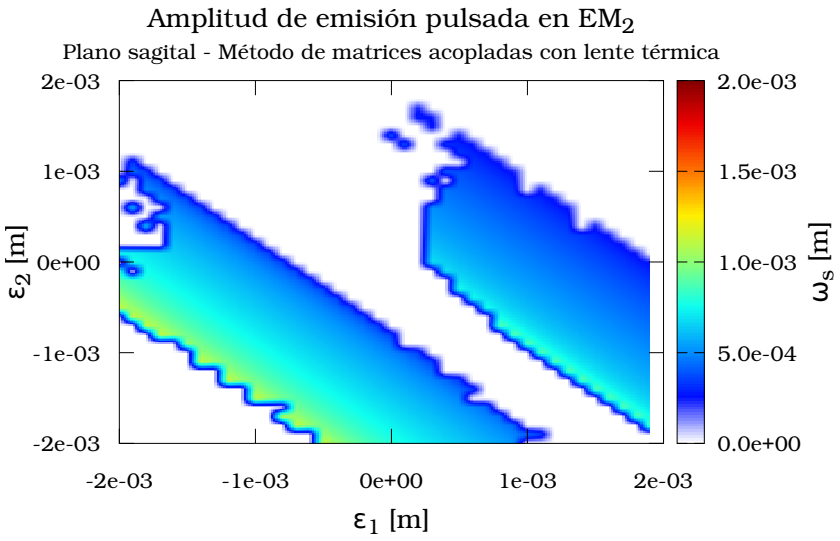


(b) Radio de haz en el plano sagital para EM_1.

Figura 5.14: Mapas de ω_t y ω_s en EM_1 para emisión pulsada utilizando el método de matrices acopladas con lente térmica.

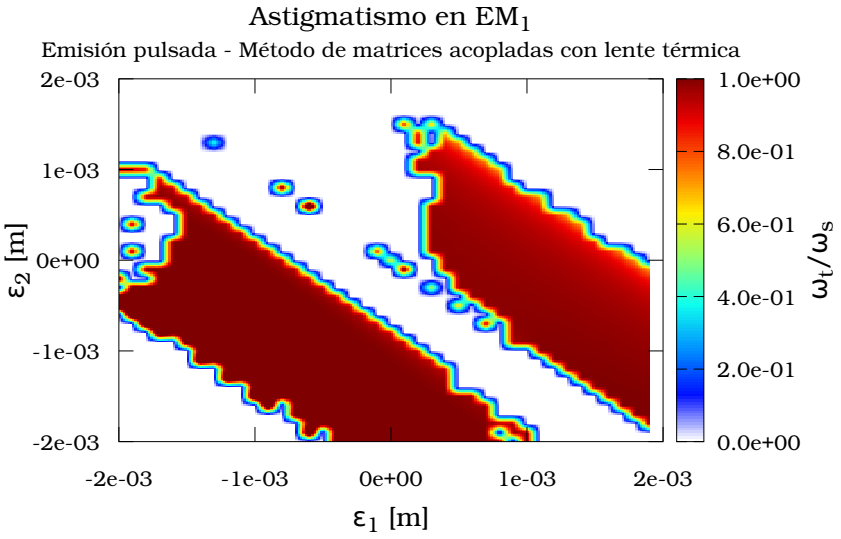


(a) Radio de haz en el plano tangencial para EM_2.

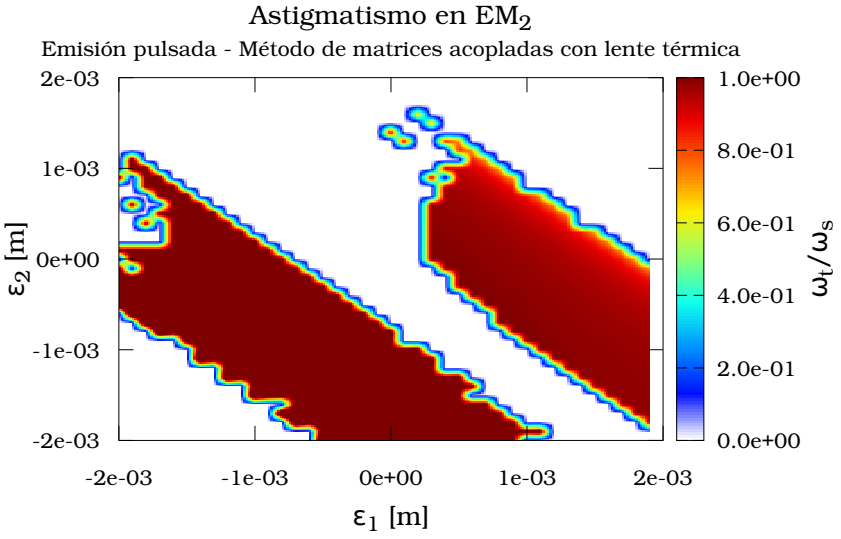


(b) Radio de haz en el plano sagital para EM_2.

Figura 5.15: Mapas de ω_t y ω_s en EM_2 para emisión pulsada utilizando el método de matrices acopladas con lente térmica.



(a) Mapa de astigmatismo para EM_1.



(b) Mapa de astigmatismo para EM_2.

Figura 5.16: Regiones de astigmatismo y estabilidad para EM_1 and EM_2 en operación pulsada utilizando el método de matrices acopladas con lente térmica. La cavidad es no astigmática si $\omega_t/\omega_s = 1$.

Capítulo 6

Resultados y discusión

6.1. Resultados

Para evaluar la similitud que existe entre los datos obtenidos, se calcula el Error Absoluto Medio (EMA), el cual se define en la Ecuación (6.1).

$$EMA = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i| \tag{6.1}$$

donde N es el número total de parejas de datos comparados, x y y son miembros de los conjuntos a comparar. Esta métrica muestra que tan cerccanos se encuentran los valores obtenidos mediante los diferentes métodos estudiados. Si el EMA es muy cercano a cero, los datos son muy similares entre sí. Estos valores se graficaron en la Figura 6.1.

La Figura 6.1 fue generada con $N = 1681$; los casos siendo

A EMA obtenido de los datos de emisión continua y de la propagación por el método de matrices desacopladas.

B EMA obtenido de los datos de emisión continua y de la propagación por el método de ecuaciones diferenciales.

C EMA obtenido de los datos de emisión continua y de la propagación por el método de matrices acopladas.

D EMA obtenido de los datos de emisión continua y de la propagación por el método de matrices acopladas y lente térmica presente.

E EMA obtenido de los datos de la propagación por los métodos de matrices desacopladas y de ecuaciones diferenciales.

F EMA obtenido de los datos de la propagación por los métodos de matrices desacopladas y de matrices acopladas.

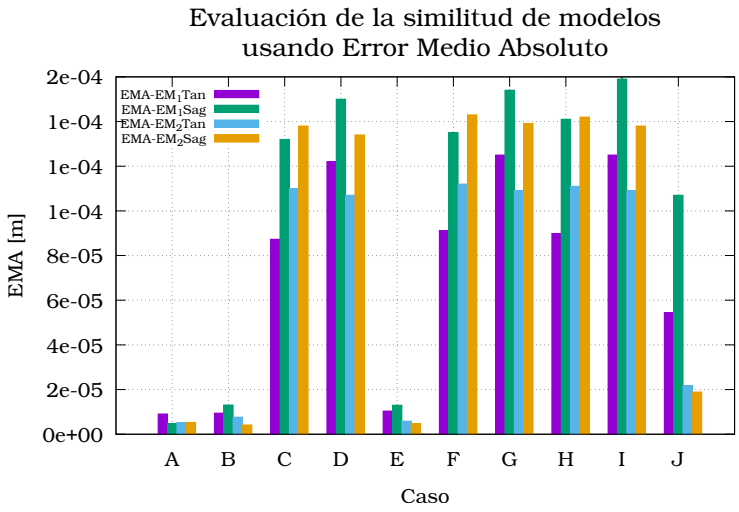


Figura 6.1: Comparación de Error Medio Absoluto

G EMA obtenido de los datos de la propagación por los métodos de matrices desacopladas y de matrices acopladas con lente térmica presente.

H EMA obtenido de los datos de la propagación por los métodos de ecuaciones diferenciales y de matrices acoplaldas.

I EMA obtenido de los datos de la propagación por los métodos de ecuaciones diferenciales y de matrices acoplaldas con lente térmica presente.

J EMA obtenido de los datos de la propagación por los métodos de matrices acopladas con y sin lente térmica presente.

Para evaluar la rapidez de cálculo, se compararon las iteraciones promedio necesarias para alcanzar la condición de autoconsistencia de cada método. Esta información se presena en la Tabla 6.1.

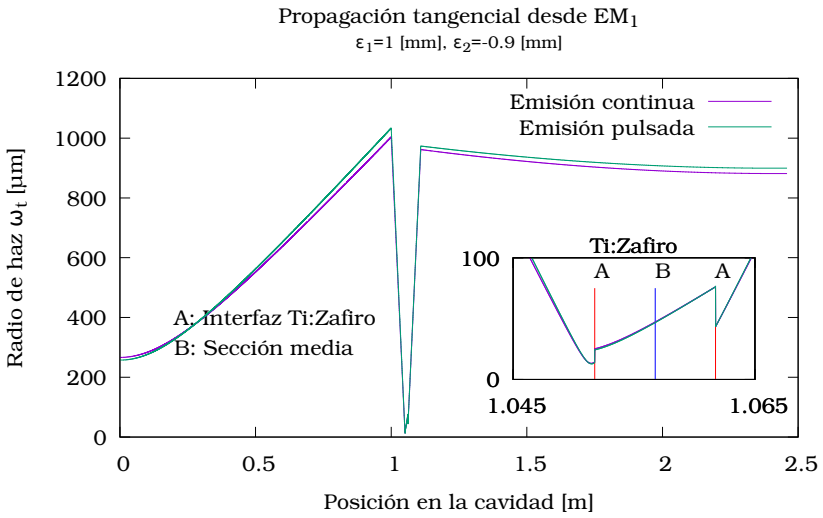
Caso	Método matricial desacoplado	Método de ecuaciones diferenciales	Método matricial acoplado	Método matricial acoplado con lente térmica
EM_1 tangential plane	580	627	7464	7789
EM_1 sagittal plane	569	790	7464	7789
EM_2 tangential plane	580	627	7029	6962
EM_2 sagittal plane	743	10	7029	6962

Tabla 6.1: Comparación de número de iteraciones promedio para alcanzar la condición de autoconsistencia.

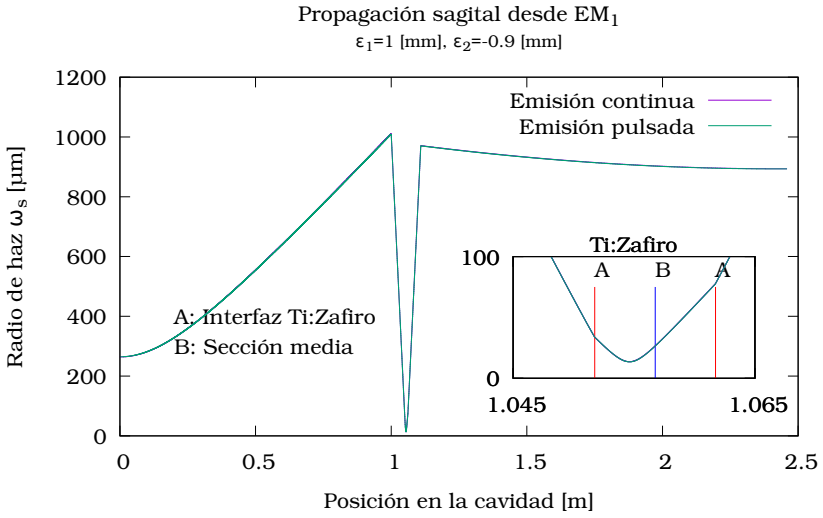
Para demostrar el efecto de ajustar ϵ_1 y ϵ_2 como se hizo en la Sección 5.4 se propagan las soluciones encontradas para emisión continua y emisión pulsada empleando el modelo matricial acoplado con lente térmica a partir de uno de los espejos de referencia (sea EM_1 o EM_2) hasta el otro.

Las gráficas en las Figuras 6.2 y 6.3 utilizan a $\epsilon_1 = 1$ [mm] y $\epsilon_2 = 0.9$ [mm] causando que el haz intracavidad se enfoque en el espejo EM_1 y sea colimado en EM_2. En dichas gráficas es apreciable la posición en donde el haz es modificado por los espejos cóncavos y el medio de ganancia.

Por otro lado, al fijar a $\epsilon_1 = -1$ [mm] y $\epsilon_2 = 0.9$ [mm] como se presenta en las Figuras 6.4 y 6.5, se modifica el comportamiento del haz intracavidad, tendiendo a tener haces prácticamente colimados en ambos brazos.

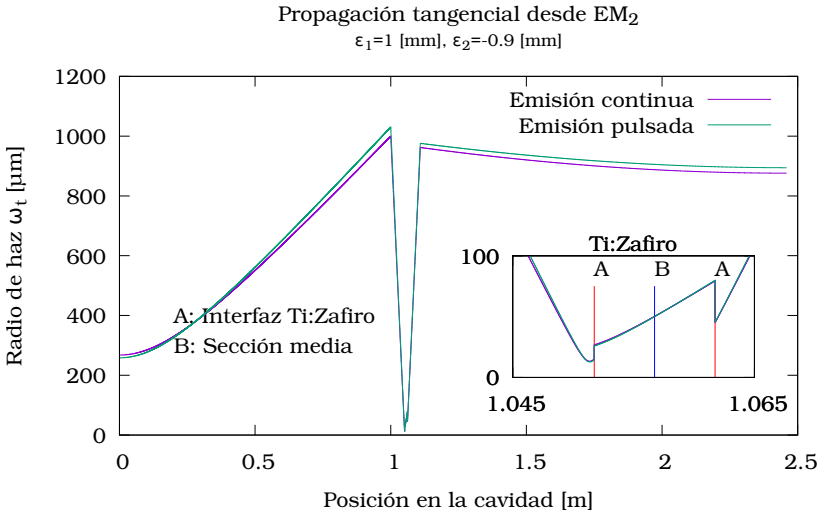


(a) Plano tangencial.

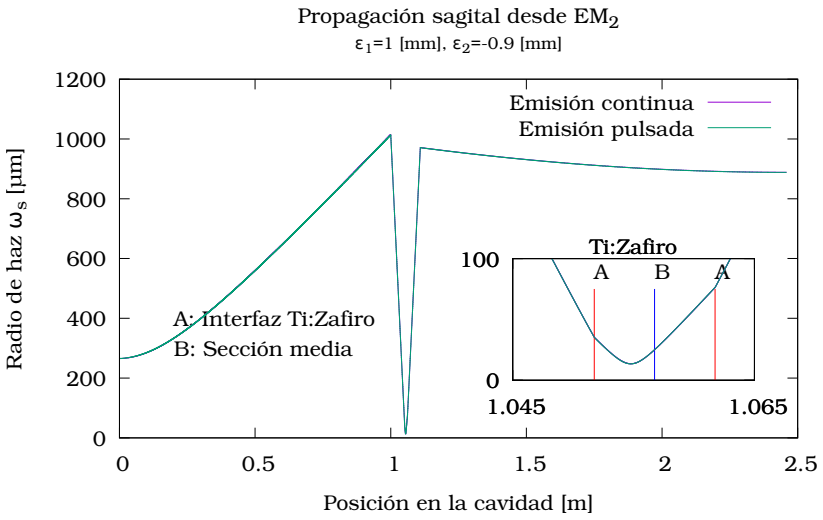


(b) Plano sagital.

Figura 6.2: Propagación del haz intracavidad para emisión continua y pulsada a partir de EM_1 utilizando $\epsilon_1 = 1$ [mm] y $\epsilon_2 = -0.9$ [mm].

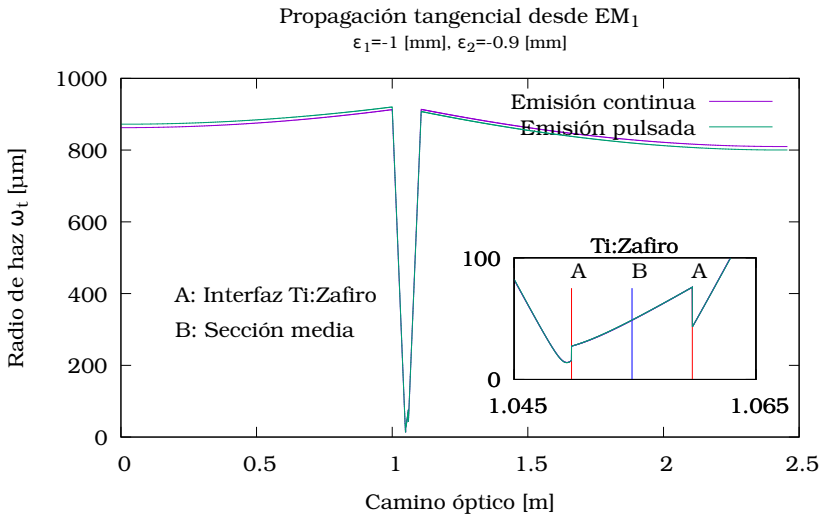


(a) Plano tangencial.

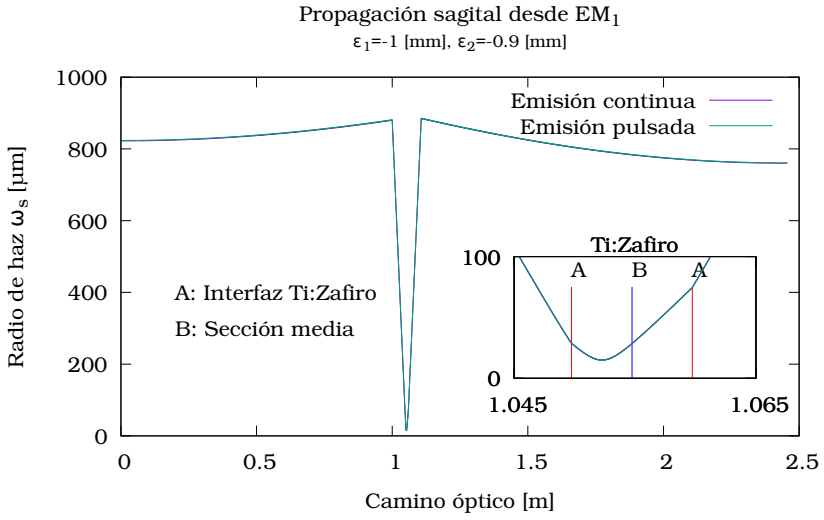


(b) Plano sagital.

Figura 6.3: Propagación del haz intracavidad para emisión continua y pulsada a partir de EM_2 utilizando $\epsilon_1 = 1$ [mm] y $\epsilon_2 = -0.9$ [mm].

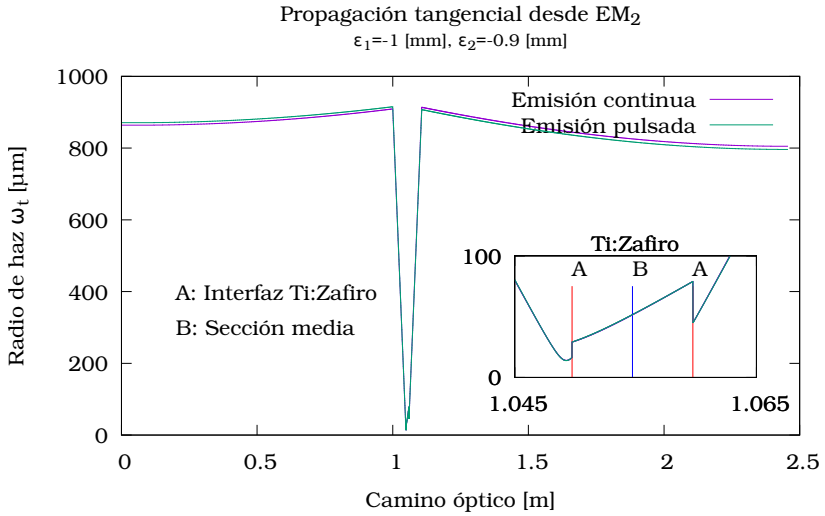


(a) Plano tangencial.

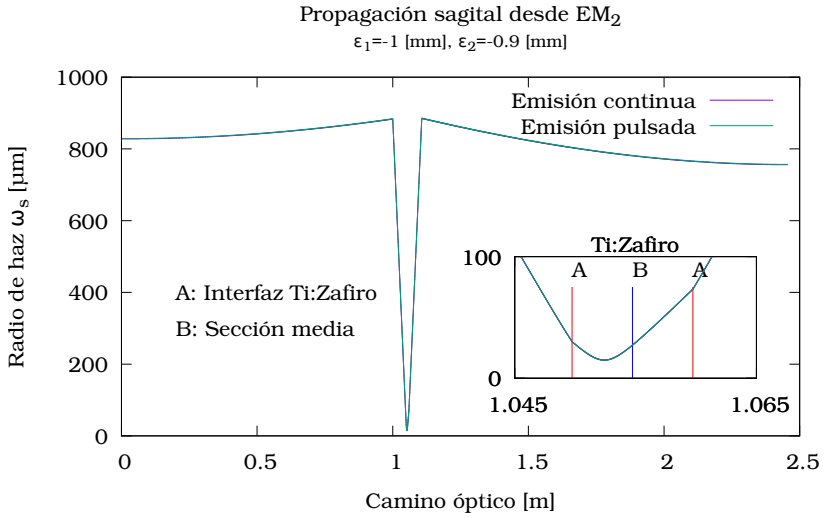


(b) Plano sagital.

Figura 6.4: Propagación del haz intracavidad para emisión continua y pulsada a partir de EM_1 utilizando $\epsilon_1 = -1$ [mm] y $\epsilon_2 = -0.9$ [mm].



(a) Plano tangencial.



(b) Plano sagital.

Figura 6.5: Propagación del haz intracavidad para emisión continua y pulsada a partir de EM_2 utilizando $\epsilon_1 = -1$ [mm] y $\epsilon_2 = -0.9$ [mm].

6.2. Discusión

Las regiones de estabilidad obtenidas a partir del análisis de emisión continua (Figuras 5.2 y 5.3) y emisión pulsada utilizando los métodos matricial desacoplado (Figuras 5.5 y 5.6) y por ecuaciones diferenciales son (Figuras 5.8 y 5.9) muy similares entre sí.

La corrección de astigmatismo mostrada en las Figuras 5.4, 5.7, 5.10, 5.13 y 5.16 sólo aplica para el espejo de referencia (EM_1 o EM_2). Por ejemplo, si se corrige astigmatismo en el espejo EM_1 y se desea conocer la forma del haz en el espejo EM_2, se tiene que propagar la solución a través de la cavidad como se hizo para obtener las Figuras 6.2, 6.3, 6.4 y 6.5. Si el astigmatismo fue corregido para EM_1 no existe garantía de que esté corregido para EM_2. Lo mismo aplica si el analisis inicia con EM_2.

Es posible corregir el astigmatismo no lineal al ajustar los valores de ϵ_1 y ϵ_2, sin embargo, el diseñador debe considerar que al hacer esto puede cambiar el parámetro complejo q en la superficie de referencia como se aprecia en las Figuras 6.2, 6.3, 6.4 y 6.5.

El Error Medio Absoluto obtenido de los datos generados por los modelos acoplados y desacoplados muestra que difieren por décimas de milímetro, como se presenta en la Figura 6.1. Esta diferencia puede cambiar el tipo de mecanismo de pérdidas requerido para alcanzar la emisión pulsada.

Los métodos de propagación de haces usando matrices acopladas requieren un número de iteraciones promedio muchas veces mayor al empleado en los métodos de propagación desacoplados, (Tabla 6.1). Dadas las diferencias en el radio del haz intracavidad calculadas, el diseñador no debe despreciar el acople espacial no lineal en la aproximación GRIN usada en el cristal.

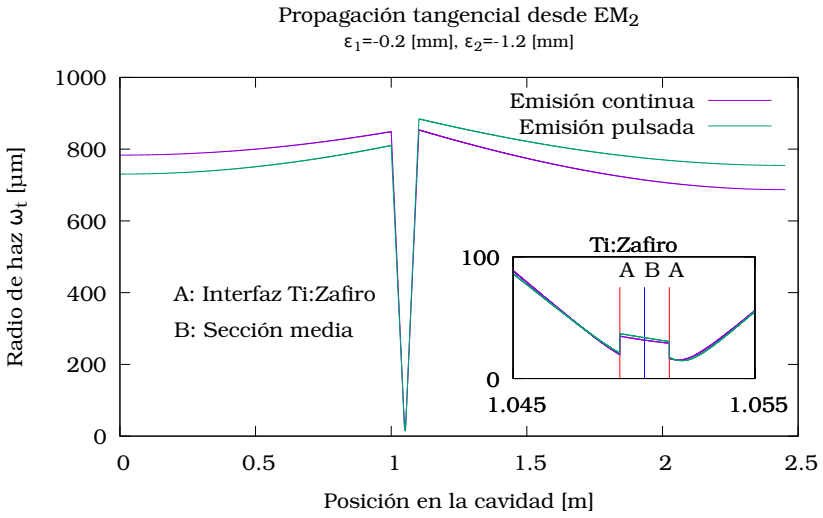
El número de iteraciones promedio requeridas para encontrar soluciones de propagación a partir del espejo EM_2 en el plano salgital es bajo comparado con el resto de los casos de la Tabla 6.1. Revisando los datos de radio de haz contra número de iteracion para este caso, se observa que no hay una clara tendencia a converger. Por otro lado, los mapas de estabilidad presentados en las Figuras 5.11, 5.12, 5.14 y 5.15 revelan que el análisis numérico no convergió para muchas soluciones de emisión continua propuestas. Lo anterior y el bajo número de iteraciones para encontrar soluciones para EM_2 en el plano sagital utilizando el método de propagación desacoplada por ecuaciones diferenciales sugiere que la condición de paro propuesta (Ecuación (5.2)) es inadecuada debido a que no provee información sobre comportamiento anómalo. Se debe diseñar un algoritmo más robusto para determinar si se alcanzó la condición de autoconsistencia para asegurar resultados confiables.

La inclusión del autoenfocamiento térmico en el método de propagación de haces por matrices genera cambios en el radio del haz intracavidad sin un impacto significativo en el número de iteraciones promedio para alcanzar la condición de autoconsistencia.

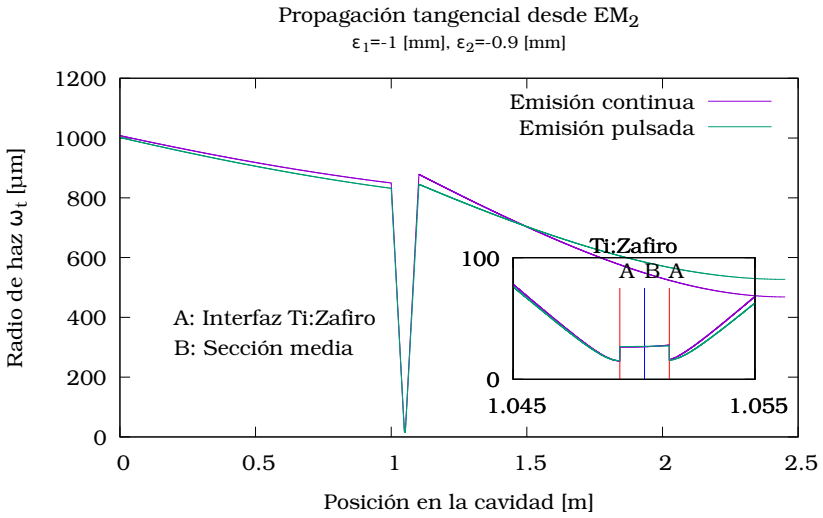
Las gráficas presentadas en Figuras 6.2, 6.3, 6.4 y 6.5 revelan diferencias en el radio del haz cerca del los espejos planos EM_1 y EM_2 durante la emisión pulsada y la emisión continua. Las gráficas generadas son consistentes con la

literatura [20].

Si el radio del haz pulsado es menor que el haz de emisión continua cerca de alguno de los espejos de referencia, se puede utilizar una apertura física para favorecer la emisión pulsada; en caso contrario, se busca realizar un acople óptico con el modo de propagación del haz pulsado dentro del cristal Titanio:Zafiro para este fin [10]. Para mostrar esto, se modeló una cavidad láser similar a la presentada en este trabajo, utilizando un cristal Titanio:Zafiro de 2.05 [mm] de longitud y una potencia intracavidad de 1.5 [MW] en los límites plano-plano y punto-plano. Se muestra la propagación del haz resultante a partir del espejo EM_2 para unos valores de ϵ_1 y ϵ_2 arbitrarios en el plano tangencial, donde se observa el comportamiento descrito.



(a) Propagación tangencial desde EM_2 para configuración plano-plano.



(b) Propagación tangencial desde EM_2 para configuración punto-plano.

Figura 6.6: Comparación del radio del haz para emisión continua contra emisión pulsada para dos formas de operación. Dependiendo que emisión tenga el radio de haz más grande, será el mecanismo de apertura a usar.

La diferencia del radio del haz de emisión pulsada y del haz de emisión continua es de escala micrométrica en el plano tangencial y prácticamente nula en el plano sagital, lo cual motiva a la búsqueda de soluciones para la cavidad en donde esta diferencia aumente con el fin de utilizar una rendija de forma satisfactoria. Se aprecia que para las configuraciones estudiadas los haces tangenciales y sagitales son astigmáticos dentro del cristal Titanio:Zafiro; el haz de bombeo debe acoplarse a ambos modos para favorecer la emisión láser. Si aumenta la potencia pico del haz intracavidad, el efecto de autoenfocamiento será mayor.

6.3. Trabajo a futuro

Este proyecto presenta oportunidades de desarrollo de software de modelado y diseño de láseres de pulsos ultracortos. Los siguientes puntos tienen la finalidad de enriquecer el trabajo presentado.

- Diseñar el sistema de acople óptico entre el haz intracavidad y el haz de bombeo para emisión pulsada.
- Mejorar la rutina de elemento finito para incluir cristales Titanio:Zafiro en forma cilíndrica.
- Diseñar una interfaz de usuario para el programa de diseño.
- Modelar cavidades de pulsos ultracortos más complejas.

Bibliografía

[1] W. R. J. Diels, *Ultrashort Laser Pulse Phenomena*. Academic Press, 2006.

[2] P. K. D. Spence and W. Sibbett, "60-fsec pulse generation form a self-mode-locked ti:sapphire laser," *Optics Letters*, vol. 16, January 1991.

[3] A. E. Siegman, *Lasers*. University Science Books, 1 ed., 1986.

[4] H. Kogelnik and T. Li, "Laser beams and resonators," *Applied Optics*, vol. 5, Octubre 1966.

[5] Altechna, "Difference between right-angle and brester-angle cut laser crystals."

[6] H. Kogelnik, "Imaging of optical modes - resonators with internal lenses," *The Bell System Technical Journal*, vol. 44, Marzo 1965.

[7] B. Saleh, *Fundamentals of Photonics*. John Wiley and Sons, 2 ed., 2007.

[8] O. Svelto, *Principles of Lasers*. Springer, 5 ed., 2010.

[9] J. T. Taché, "Ray matrices for tilted interfaces in laser resonators," *Applied Optics*, vol. 26, Febrero 1987.

[10] S. Yefet and A. Pe'er, "A review of cavity design for kerr lens mode-locked solid-state lasers," *Applied Sciences*, vol. 3, Diciembre 2013.

[11] W.-F. H. Kuei-Huei Lin, Yinchieh Lai, "Simple analytical method of cavity design for astigmatism-compensated kerr-lens mode-locked ring lasers and its applications," *Journal of the Optical Society of America B*, vol. 12, March 1995.

[12] W.-F. H. Kuei-Huei Lin, "Analytical design of symmetrical kerr-lens mode-locking laser cavities," *Journal of the Optical Society of America B*, vol. 11, May 1994.

[13] D. Georgiev, J. Herrmann, *et al.*, "Cavity design for optimum nonlinear absoption in kerr-lens mode-locked solid-state lasers," *Optics Communications*, vol. 92, Septiembre 1992.

[14] D. Huang, M. Ulman, *et al.*, "Self-focused-induced saturable loss for laser mode locking," *Optics Letters*, vol. 17, Abril 1992.

[15] V. Magni, "Abcd matrix analysis of propagation of gaussian beams through kerr media," *Optics Communications*, vol. 96, February 1993.

[16] R. Bridges, "Effect of beam ellipticity on self-mode locking in lasers," *Optics Letters*, vol. 18, December 1993.

[17] J. J. H.S. Carslaw, *Conduction of Heat in Solids*. Oxford University Press, 2 ed., 1959.

[18] M. Mehendale, "Thermal effects in laser pumped kerr-lens modelocked ti:sapphire lasers," *Optics Communications*, vol. 136, March 1997.

[19] A. Ritsataki, "A numerical model of kerr-lens mode-locking," *Optics Communications*, vol. 142, October 1997.

[20] F. Kärtner, "6.977 ultrafast optics, spring 2005," 2005. [Online. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-977-ultrafast-optics-spring-2005/lecture-notes/chapter7.pdf (Accessed 3 May, 2016). License: Creative Commons BY-NC-SA].

[21] J. A. Moreno-Larios, M. Rosete-Aguilar, and J. Garduño-Mejía, "Comparison of different kerr-lens mode locking laser design techniques," 2016.

Apéndice A

Programas desarrollados

Para el desarrollo de la tesis, se escribieron diversos programas para la simulación de una cavidad láser paraxial. El programa de diseño y el programa para visualizar la propagación del haz la cavidad fueron escritos en C y sus respectivas gráficas fueron generadas con Gnuplot.

Las gráficas observadas en el Capítulo 3 fueron generadas utilizando GNU Octave.

A.1. Gráficas del análisis lineal

Se incluyen los siguientes archivos para GNU Octave:

angulo_lineal.m Rutina para el cálculo de los ángulos para corrección de astigmatismo lineal.

distancia_cristal.m Rutina para el cálculo de la separación de los espejos cóncavos al cristal.

graficas_spot_lineal.m Rutina para generar gráficas de una cavidad corregida en astigmatismo para las zonas de operación discutidas en el Capítulo 3.

graficas_spot_lineal_diferencia.m Rutina que muestra la diferencia entre el radio del haz en el plano tangencial y sagital para dicha cavidad corregida en astigmatismo lineal.

grafica_spot_sin_compensar.m Ejemplo de cavidad utilizando un ángulo de inclinación de los espejos cóncavos arbitrario.

A.2. Código fuente de Octave

A.2.1. angulo_lineal.m

```
% Calculo de angulo para compensar astigmatismo lineal

function [theta1,theta2]=angulo_lineal(conjugado_corto,
    conjugado_largo,L,n,L1,L2,f1,f2)
  if (nargin!=8)
    usage("angulo_lineal(conjugado_corto,conjugado_largo,
        L,n,L1,L2,f1,f2)");
  endif
  A=L/2*(1/n^3 1/n);
  if(conjugado_corto=='inf')
    x_1=(A+sqrt(A^2+4*f1^2))/(2*f1);
  elseif(conjugado_corto=='fin')
    x_1=(A*(L1^2+f1^2)+sqrt(4*f1^2*L1^4+A^2*(f1^2 L1^2)
        ^2))/(2*(f1*L1*(L1+A)));
  else
    error("Especificar 'inf' (infinito) o 'fin' (finito)
        para conjugado corto\n"); %end
  endif
  if(conjugado_largo=='inf')
    x_2=(A+sqrt(A^2+4*f2^2))/(2*f2);
  elseif(conjugado_largo=='fin')
    x_2=(A*(L2^2+f2^2)+sqrt(4*f2^2*L2^4+A^2*(f2^2 L2^2)
        ^2))/(2*(f2*L2*(L2+A)));
  else
    error("Especificar 'inf' (infinito) o 'fin' (finito)
        para conjugado largo\n"); %end
  endif
  theta1=acos(x_1);
  theta2=acos(x_2);
endfunction
```

A.2.2. distancia_cristal.m

```
% Cálculo de la distancia que separa a un espejo concavo
    del cristal

function delta=distancia_cristal(tipo_conjugado,f,
    Longitud,L,n,theta)
  if (nargin!=6)
    usage("delta=distancia_cristal(tipo_conjugado,f,
        Longitud,Grosor de cristal,n,angulo de incidencia
        en espejo)")
  endif

  if((isfloat(f)||isfloat(Longitud)||isfloat(L)||isfloat(
      n))==0)
```

```
    error("Se esperan numeros como argumentos\n")
  elseif(tipo_conjugado=='inf')
    delta=f/cos(theta) L/(2*n);
  elseif(tipo_conjugado=='fin')
    delta=Longitud*f/cos(theta)/(Longitud f/cos(theta)) L
        /(2*n);
  else
    error("Especificar 'inf' (infinito) o 'fin' (finito)
        el conjugado\n");
  endif
endfunction
```

A.2.3. graficas_spot_lineal.m

```
% Rutinas para generar graficas de spot para diferentes
    condiciones
% La distancia entre el espejo M1 y el cristal permanece
     constante y la distancia
% entre el cristal y el espejo M2 se varia sumando un
    valor epsilon, imitando las
% condiciones del laboratiorio. Se supone que los
    angulos de los espejos son
% distintos.

clear;
clc;

%Declarando variables, cantidades en milimetros
L1=1000;
L2=1350;
L=10;
n=1.7598; %Indice de refraccion
f1=50;   %Dist. focal M1
f2=50;   %Dist. focal M2
lambda=810e 6;  %Longitud de onda

function [Vectores,Distancias,Angulos]=calcular(
    conjugado_corto,conjugado_largo,L1,L2,L,n,f1,f2,lambda
    )
     %Calculo de angulo
     [theta1,theta2]=angulo_lineal(conjugado_corto,
         conjugado_largo,L,n,L1,L2,f1,f2);
     Angulos=[theta1;theta2];

     %Calculo de distancias de espejos concavos a cristal
     delta1=distancia_cristal(conjugado_corto,f1,L1,L,n,
```

```
    theta1);
delta2=distancia_cristal(conjugado_largo,f2,L2,L,n,
    theta2);
Distancias=[delta1;delta2];

%Definiendo distancias focales astigmaticas
f1t=f1*cos(theta1);
f2t=f2*cos(theta2);
f1s=f1/cos(theta1);
f2s=f1/cos(theta2);

%Generando vectores
%epsilon=(f2t^2/(L2 f2t)+f1t^2/(L1 f1t)):0.1:f2t^2/(
    L2 f2t)+f1t^2/(L1 f1t);
epsilon = 10:0.1:10;
wtEM1=zeros(1,length(epsilon));
wtEM2=zeros(1,length(epsilon));
wsEM1=zeros(1,length(epsilon));
wsEM2=zeros(1,length(epsilon));

%Lazo para calculo
for index=1:length(epsilon)
  %Calculando matrices
  TangencialEM1=[1, L1; 0, 1]*[1, 0; 1/f1t, 1]*...
  [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
      epsilon(index); 0, 1]*...
  [1,0; 1/f2t, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
       0, 1]*...
  [1, 0; 1/f2t, 1]*[1, delta2+epsilon(index); 0,
      1]*[1,L/n^3; 0, 1]*...
  [1, delta1; 0, 1]*[1, 0; 1/f1t, 1]*[1, L1; 0, 1];

  SagitalEM1=[1, L1; 0, 1]*[1, 0; 1/f1s, 1]*...
  [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
      (index); 0, 1]*...
  [1,0; 1/f2s, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
       0, 1]*...
  [1, 0; 1/f2s, 1]*[1, delta2+epsilon(index); 0,
      1]*[1,L/n; 0, 1]*...
  [1, delta1; 0, 1]*[1, 0; 1/f1s, 1]*[1, L1; 0, 1];

  TangencialEM2=[1, L2; 0, 1]*[1, 0; 1/f2t, 1]*[1,
      delta2+epsilon(index); 0, 1]*...
  [1, L/n^3; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1t, 1]*
      ...
  [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1t
```

```
        , 1]*...
        [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
            epsilon(index); 0, 1]*...
        [1,0; 1/f2t, 1]*[1, L2; 0, 1];

        SagitalEM2=[1, L2; 0, 1]*[1, 0; 1/f2s, 1]*[1,
            delta2+epsilon(index); 0, 1]*...
        [1, L/n; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1s, 1]*
            ...
        [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1s
            , 1]*...
        [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
            (index); 0, 1]*...
        [1,0; 1/f2s, 1]*[1, L2; 0, 1];

        % Calculo de spots
        wtEM1(index)=spot(TangencialEM1,lambda);
        wsEM1(index)=spot(SagitalEM1,lambda);
        wtEM2(index)=spot(TangencialEM2,lambda);
        wsEM2(index)=spot(SagitalEM2,lambda);
    endfor
    Vectores=[wtEM1;wsEM1;wtEM2;wsEM2;epsilon];
endfunction

% Generando valores
[Datos_inf_inf,distancias_inf_inf,angulos_inf_inf]=
    calcular('inf','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_inf_fin,distancias_inf_fin,angulos_inf_fin]=
    calcular('inf','fin',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_inf,distancias_fin_inf,angulos_fin_inf]=
    calcular('fin','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_fin,distancias_fin_fin,angulos_fin_fin]=
    calcular('fin','fin',L1,L2,L,n,f1,f2,lambda);
%% Gráficas

% Infinito infinito
figure(1);
subplot(1,2,1)
stem(Datos_inf_inf(5,:),Datos_inf_inf(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_inf(5,:),Datos_inf_inf(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
```

```
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{1} (colimado)','
    FontSize',22)
parametros1={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_inf(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_inf(2,1)) ' [mm]'
    ]};
text(2.5,1,parametros1,'FontSize',16)
xlim([0,5])
ylim([0,1.5])
subplot(1,2,2)
stem(Datos_inf_inf(5,:),Datos_inf_inf(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_inf(5,:),Datos_inf_inf(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{2} (colimado)','
    FontSize',22)
text(1,1.5,parametros1,'FontSize',16)
xlim([0,5])
ylim([0,1.75])

%Infinito finito
figure(2);
subplot(1,2,1)
stem(Datos_inf_fin(5,:),Datos_inf_fin(1,:),'r','LineWidth
    ',1.5)
hold on;
```

```
stem(Datos_inf_fin(5,:),Datos_inf_fin(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{1} (colimado)','
    FontSize',22)
parametros2={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_fin(2,1)) ' [mm]'
    ]};
text(1,1.2,parametros2,'FontSize',16)
xlim([2,3])
ylim([0,1.75])
subplot(1,2,2)
stem(Datos_inf_fin(5,:),Datos_inf_fin(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_fin(5,:),Datos_inf_fin(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{2} (enfocado)','
    FontSize',22)
text(1.2,1.2,parametros2,'FontSize',16)
xlim([2,3])
ylim([0,1.75])

%Finito infinito
```

```
figure(3);
subplot(1,2,1)
stem(Datos_fin_inf(5,:),Datos_fin_inf(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_inf(5,:),Datos_fin_inf(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{1} (enfocado)','
    FontSize',22)
parametros3={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_inf(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_fin_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_inf(2,1)) ' [mm]'
    ]};
text(0,1.2,parametros3,'FontSize',16)
xlim([ 3,2])
ylim([0,2])
subplot(1,2,2)
stem(Datos_fin_inf(5,:),Datos_fin_inf(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_inf(5,:),Datos_fin_inf(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{2} (colimado)','
    FontSize',22)
```

```
text( 2,1.5,parametros3,'FontSize',16)
xlim([ 3,2])
ylim([0,2])

%Finito finito
figure(4);
subplot(1,2,1)
stem(Datos_fin_fin(5,:),Datos_fin_fin(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_fin(5,:),Datos_fin_fin(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{1} (enfocado)','
    FontSize',22)
parametros4={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_fin_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_fin(2,1)) ' [mm]'
    ]};
text( 2,1.2,parametros4,'FontSize',16)
xlim([ 5,0])
ylim([0,1.75])
subplot(1,2,2)
stem(Datos_fin_fin(5,:),Datos_fin_fin(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_fin(5,:),Datos_fin_fin(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2}[mm]','FontSize',16)
```

```
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title('Tamaño de spot en espejo EM_{2} (enfocado)','
    FontSize',22)
text(3.7,1.4,parametros4,'FontSize',16)
xlim([5,0])
ylim([0,1.75])

%Fin
```

A.2.4. graficas_spot_lineal_diferencia.m

```
% Rutinas para generar graficas de spot para diferentes
    condiciones
% La distancia entre el espejo M1 y el cristal permanece
     constante y la distancia
% entre el cristal y el espejo M2 se varia sumando un
    valor epsilon, imitando las
% condiciones del laboratiorio. Se supone que los
    angulos de los espejos son
% distintos.

clear;
clc;

%Declarando variables, cantidades en milimetros
L1=1000;
L2=1350;
L=10;
n=1.7598; %Indice de refraccion
f1=50;   %Dist. focal M1
f2=50;   %Dist. focal M2
lambda=810e-6; %Longitud de onda

function [Vectores,Distancias,Angulos]=calcular(
    conjugado_corto,conjugado_largo,L1,L2,L,n,f1,f2,lambda
    )
     %Calculo de angulo
     [theta1,theta2]=angulo_lineal(conjugado_corto,
         conjugado_largo,L,n,L1,L2,f1,f2);
     Angulos=[theta1;theta2];

     %Calculo de distancias de espejos concavos a cristal
     delta1=distancia_cristal(conjugado_corto,f1,L1,L,n,
         theta1);
```

```
delta2=distancia_cristal(conjugado_largo,f2,L2,L,n,
    theta2);
Distancias=[delta1;delta2];

%Definiendo distancias focales astigmaticas
f1t=f1*cos(theta1);
f2t=f2*cos(theta2);
f1s=f1/cos(theta1);
f2s=f1/cos(theta2);

%Generando vectores
%epsilon=(f2t^2/(L2 f2t)+f1t^2/(L1 f1t)):0.1:f2t^2/(
    L2 f2t)+f1t^2/(L1 f1t);
epsilon= 10:0.1:10;
wtEM1=zeros(1,length(epsilon));
wtEM2=zeros(1,length(epsilon));
wsEM1=zeros(1,length(epsilon));
wsEM2=zeros(1,length(epsilon));

%Lazo para calculo
for index=1:length(epsilon)
  %Calculando matrices
  TangencialEM1=[1, L1; 0, 1]*[1, 0; 1/f1t, 1]*...
  [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
      epsilon(index); 0, 1]*...
  [1,0; 1/f2t, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
       0, 1]*...
  [1, 0; 1/f2t, 1]*[1, delta2+epsilon(index); 0,
      1]*[1,L/n^3; 0, 1]*...
  [1, delta1; 0, 1]*[1, 0; 1/f1t, 1]*[1, L1; 0, 1];

  SagitalEM1=[1, L1; 0, 1]*[1, 0; 1/f1s, 1]*...
  [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
      (index); 0, 1]*...
  [1,0; 1/f2s, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
       0, 1]*...
  [1, 0; 1/f2s, 1]*[1, delta2+epsilon(index); 0,
      1]*[1,L/n; 0, 1]*...
  [1, delta1; 0, 1]*[1, 0; 1/f1s, 1]*[1, L1; 0, 1];

  TangencialEM2=[1, L2; 0, 1]*[1, 0; 1/f2t, 1]*[1,
      delta2+epsilon(index); 0, 1]*...
  [1, L/n^3; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1t, 1]*
      ...
  [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1t
      , 1]*...
```

```
        [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
            epsilon(index); 0, 1]*...
        [1,0;  1/f2t, 1]*[1, L2; 0, 1];

        SagitalEM2=[1, L2; 0, 1]*[1, 0;  1/f2s, 1]*[1,
            delta2+epsilon(index); 0, 1]*...
        [1, L/n; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1s, 1]*
            ...
        [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1s
            , 1]*...
        [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
            (index); 0, 1]*...
        [1,0;  1/f2s, 1]*[1, L2; 0, 1];

        % Calculo de spots
        wtEM1(index)=spot(TangencialEM1,lambda);
        wsEM1(index)=spot(SagitalEM1,lambda);
        wtEM2(index)=spot(TangencialEM2,lambda);
        wsEM2(index)=spot(SagitalEM2,lambda);
    endfor
    Vectores=[wtEM1;wsEM1;wtEM2;wsEM2;epsilon];
endfunction

% Generando valores
[Datos_inf_inf,distancias_inf_inf,angulos_inf_inf]=
    calcular('inf','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_inf_fin,distancias_inf_fin,angulos_inf_fin]=
    calcular('inf','fin',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_inf,distancias_fin_inf,angulos_fin_inf]=
    calcular('fin','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_fin,distancias_fin_fin,angulos_fin_fin]=
    calcular('fin','fin',L1,L2,L,n,f1,f2,lambda);
%% Gráficas

% Infinito infinito
figure(1);
subplot(1,2,1)
stem(Datos_inf_inf(5,:),abs(Datos_inf_inf(1,:)
    Datos_inf_inf(2,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{1}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en EM_{1} (colimado)','FontSize',22)
```

```
parametros1={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_inf(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_inf(2,1)) ' [mm]'
    ]};
text(0.1,0.15,parametros1,'FontSize',16)
xlim([0,5])
ylim([0,0.2])
subplot(1,2,2)
stem(Datos_inf_inf(5,:),abs(Datos_inf_inf(3,:)
    Datos_inf_inf(4,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{2}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{2} (colimado)','
    FontSize',22)
text(0.1,0.15,parametros1,'FontSize',16)
xlim([0,5])
ylim([0,0.2])

%Infinito finito
figure(2);
subplot(1,2,1)
stem(Datos_inf_fin(5,:),abs(Datos_inf_fin(1,:)
    Datos_inf_fin(2,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{1}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{1} (colimado)','
    FontSize',22)
parametros2={'Parámetros de ', 'la cavidad:',['L_{1}='
```

```
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_fin(2,1)) ' [mm]'
    ]};
text(1,0.15,parametros2,'FontSize',16)
xlim([2,3])
ylim([0,0.2])
subplot(1,2,2)
stem(Datos_inf_fin(5,:),abs(Datos_inf_fin(3,:)
    Datos_inf_fin(4,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{2}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{2} (enfocado)','
    FontSize',22)
text(1,0.15,parametros2,'FontSize',16)
xlim([2,3])
ylim([0,0.2])

%Finito infinito
figure(3);
subplot(1,2,1)
stem(Datos_fin_inf(5,:),abs(Datos_fin_inf(1,:)
    Datos_fin_inf(2,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{1}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{1} (enfocado)','
    FontSize',22)
parametros3={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
```

```
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_inf(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_fin_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_inf(2,1)) ' [mm]'
    ]};
text(0.5,0.15,parametros3,'FontSize',16)
xlim([3,2])
ylim([0,0.2])
subplot(1,2,2)
stem(Datos_fin_inf(5,:),abs(Datos_fin_inf(3,:)
    Datos_fin_inf(4,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{2}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{2} (colimado)','
    FontSize',22)
text(0.5,0.15,parametros3,'FontSize',16)
xlim([3,2])
ylim([0,0.2])

%Finito finito
figure(4);
subplot(1,2,1)
stem(Datos_fin_fin(5,:),abs(Datos_fin_fin(1,:)
    Datos_fin_fin(2,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{1}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{1} (enfocado)','
    FontSize',22)
parametros4={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
```

```
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_fin_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_fin(2,1)) ' [mm]'
    ]};
text(1.5,0.15,parametros4,'FontSize',16)
xlim([5,0.2])
ylim([0,0.2])
subplot(1,2,2)
stem(Datos_fin_fin(5,:),abs(Datos_fin_fin(3,:)
    Datos_fin_fin(4,:)),'r','LineWidth',1.5)
xlabel('\epsilon [mm]','FontSize',16)
ylabel('|\omega_{t} \omega_{s}| en EM_{2}[mm]','FontSize'
    ,16)
set(gca,'FontSize',14)
h=legend('|\omega_{t} \omega_{s}|');
set(h,'FontSize',14)
title('\Delta \omega en espejo EM_{2} (enfocado)','
    FontSize',22)
text(1.5,0.15,parametros4,'FontSize',16)
xlim([5,0.2])
ylim([0,0.2])

% Fin
```

A.2.5. graficas_spot_sin_compensar.m

```
% Rutinas para generar graficas de spot para diferentes
    condiciones
% La distancia entre el espejo M1 y el cristal permanece
    constante y la distancia
% entre el cristal y el espejo M2 se varia sumando un
    valor epsilon, imitando las
% condiciones del laboratiorio. Los ángulos son fijados
    a 0.2 rad.

clear;
clc;

%Declarando variables, cantidades en milimetros
```

```
L1=1000;
L2=1350;
L=10;
n=1.7598; % Indice de refraccion
f1=50;   %Dist. focal M1
f2=50;   %Dist. focal M2
lambda=810e 6;  % Longitud de onda

function [Vectores,Distancias,Angulos]=calcular(
    conjugado_corto,conjugado_largo,L1,L2,L,n,f1,f2,lambda
    )
     % Calculo de angulo
     [theta1,theta2]=angulo_lineal(conjugado_corto,
         conjugado_largo,L,n,L1,L2,f1,f2);


     % Calculo de distancias de espejos concavos a cristal
     delta1=distancia_cristal(conjugado_corto,f1,L1,L,n,
         theta1);
     delta2=distancia_cristal(conjugado_largo,f2,L2,L,n,
         theta2);
     Distancias=[delta1;delta2];

     % Cambio de angulos

     theta1=14*pi/180;
     theta2=14*pi/180;
     Angulos=[theta1;theta2];
     % Definiendo distancias focales astigmaticas
     f1t=f1*cos(theta1);
     f2t=f2*cos(theta2);
     f1s=f1/cos(theta1);
     f2s=f1/cos(theta2);

     % Generando vectores
     %epsilon=(f2t^2/(L2 f2t)+f1t^2/(L1 f1t)):0.1:f2t^2/(
         L2 f2t)+f1t^2/(L1 f1t);
     epsilon= 10:0.1:10;
     wtEM1=zeros(1,length(epsilon));
     wtEM2=zeros(1,length(epsilon));
     wsEM1=zeros(1,length(epsilon));
     wsEM2=zeros(1,length(epsilon));

     % Lazo para calculo
     for index=1:length(epsilon)
       % Calculando matrices
```

```
        TangencialEM1=[1, L1; 0, 1]*[1, 0; 1/f1t, 1]*...
        [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
            epsilon(index); 0, 1]*...
        [1,0; 1/f2t, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
             0, 1]*...
        [1, 0; 1/f2t, 1]*[1, delta2+epsilon(index); 0,
            1]*[1,L/n^3; 0, 1]*...
        [1, delta1; 0, 1]*[1, 0; 1/f1t, 1]*[1, L1; 0, 1];

        SagitalEM1=[1, L1; 0, 1]*[1, 0; 1/f1s, 1]*...
        [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
            (index); 0, 1]*...
        [1,0; 1/f2s, 1]*[1, L2; 0, 1]*[1, 0; 0, 1]*[1, L2;
             0, 1]*...
        [1, 0; 1/f2s, 1]*[1, delta2+epsilon(index); 0,
            1]*[1,L/n; 0, 1]*...
        [1, delta1; 0, 1]*[1, 0; 1/f1s, 1]*[1, L1; 0, 1];

        TangencialEM2=[1, L2; 0, 1]*[1, 0; 1/f2t, 1]*[1,
            delta2+epsilon(index); 0, 1]*...
        [1, L/n^3; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1t, 1]*
             ...
        [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1t
            , 1]*...
        [1, delta1; 0, 1]*[1, L/n^3; 0, 1]*[1, delta2+
            epsilon(index); 0, 1]*...
        [1,0; 1/f2t, 1]*[1, L2; 0, 1];

        SagitalEM2=[1, L2; 0, 1]*[1, 0; 1/f2s, 1]*[1,
            delta2+epsilon(index); 0, 1]*...
        [1, L/n; 0, 1]*[1, delta1; 0, 1]*[1,0; 1/f1s, 1]*
            ...
        [1, L1; 0, 1]*[1, 0; 0, 1]*[1, L1;0,1]*[1, 0; 1/f1s
            , 1]*...
        [1, delta1; 0, 1]*[1, L/n; 0, 1]*[1, delta2+epsilon
            (index); 0, 1]*...
        [1,0; 1/f2s, 1]*[1, L2; 0, 1];

        % Calculo de spots
        wtEM1(index)=spot(TangencialEM1,lambda);
        wsEM1(index)=spot(SagitalEM1,lambda);
        wtEM2(index)=spot(TangencialEM2,lambda);
        wsEM2(index)=spot(SagitalEM2,lambda);
    endfor
    Vectores=[wtEM1;wsEM1;wtEM2;wsEM2;epsilon];
endfunction
```

```
% Generando valores
[Datos_inf_inf,distancias_inf_inf,angulos_inf_inf]=
    calcular('inf','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_inf_fin,distancias_inf_fin,angulos_inf_fin]=
    calcular('inf','fin',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_inf,distancias_fin_inf,angulos_fin_inf]=
    calcular('fin','inf',L1,L2,L,n,f1,f2,lambda);
[Datos_fin_fin,distancias_fin_fin,angulos_fin_fin]=
    calcular('fin','fin',L1,L2,L,n,f1,f2,lambda);
%% Gráficas

% Infinito infinito
figure(1);
subplot(1,2,1)
stem(Datos_inf_inf(5,:),Datos_inf_inf(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_inf(5,:),Datos_inf_inf(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{1}';'sin compensar
    astigmatismo (colimado)'},'FontSize',15)
parametros1={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_inf(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_inf(2,1)) ' [mm]'
    ]};
text(3.5,1,parametros1,'FontSize',16)
xlim([1,6])
ylim([0,1.5])
subplot(1,2,2)
```

```
stem(Datos_inf_inf(5,:),Datos_inf_inf(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_inf(5,:),Datos_inf_inf(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{2}';'sin compensar
    astigmatismo (colimado)'},'FontSize',15)
text(2,1.5,parametros1,'FontSize',16)
xlim([1,6])
ylim([0,1.75])

%Infinito finito
figure(2);
subplot(1,2,1)
stem(Datos_inf_fin(5,:),Datos_inf_fin(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_fin(5,:),Datos_inf_fin(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{1}';'sin compensar
    astigmatismo (colimado)'},'FontSize',15)
parametros2={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_inf_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_inf_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_inf_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_inf_fin(2,1)) ' [mm]'
```

```
    ]};
text(2,1.2,parametros2,'FontSize',16)
xlim([3,4.5])
ylim([0,1.75])
subplot(1,2,2)
stem(Datos_inf_fin(5,:),Datos_inf_fin(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_inf_fin(5,:),Datos_inf_fin(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{2}';'sin compensar
    astigmatismo (enfocado)'},'FontSize',15)
text(2.2,1.3,parametros2,'FontSize',16)
xlim([3,5])
ylim([0,1.75])

%Finito infinito
figure(3);
subplot(1,2,1)
stem(Datos_fin_inf(5,:),Datos_fin_inf(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_inf(5,:),Datos_fin_inf(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{1}';'sin compensar
    astigmatismo (enfocado)'},'FontSize',15)
parametros3={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_inf(1,1)) ' [rad]'
    ],...
```

```
['\theta_{2}=' num2str(angulos_fin_inf(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_inf(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_inf(2,1)) ' [mm]'
    ]};
text(1,1.2,parametros3,'FontSize',16)
xlim([ 4.5,4])
ylim([0,2])
subplot(1,2,2)
stem(Datos_fin_inf(5,:),Datos_fin_inf(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_inf(5,:),Datos_fin_inf(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{2}';'sin compensar
    astigmatismo (colimado)'},'FontSize',15)
text(1.5,1.5,parametros3,'FontSize',16)
xlim([ 4.5,4.5])
ylim([0,2])

%Finito finito
figure(4);
subplot(1,2,1)
stem(Datos_fin_fin(5,:),Datos_fin_fin(1,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_fin(5,:),Datos_fin_fin(2,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{1} [mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{1}';'sin compensar
    astigmatismo (enfocado)'},'FontSize',15)
parametros4={'Parámetros de ', 'la cavidad:',['L_{1}='
    num2str(L1) ' [mm]'],['L_{2}=' num2str(L2) ' [mm]'
    ],...
```

```
['L=' num2str(L) ' [mm]'],['n=' num2str(n)],...
['f_{1}=' num2str(f1) ' [mm]'],['f_{2}=' num2str(f2) ' [
    mm]'],...
['\theta_{1}=' num2str(angulos_fin_fin(1,1)) ' [rad]'
    ],...
['\theta_{2}=' num2str(angulos_fin_fin(2,1)) ' [rad]'
    ],...
['\delta_{1}=' num2str(distancias_fin_fin(1,1)) ' [mm]'
    ],...
['\delta_{2}=' num2str(distancias_fin_fin(2,1)) ' [mm]'
    ]};
text(1,1.2,parametros4,'FontSize',16)
xlim([6,1.5])
ylim([0,1.75])
subplot(1,2,2)
stem(Datos_fin_fin(5,:),Datos_fin_fin(3,:),'r','LineWidth
    ',1.5)
hold on;
stem(Datos_fin_fin(5,:),Datos_fin_fin(4,:),'b','LineWidth
    ',1.5)
hold off;
xlabel('\epsilon [mm]','FontSize',16)
ylabel('\omega en EM_{2}[mm]','FontSize',16)
set(gca,'FontSize',14)
h=legend('\omega_{t}','\omega_{s}');
set(h,'FontSize',14)
title({'Tamaño de spot en espejo EM_{2}';'sin compensar
    astigmatismo (enfocado)'},'FontSize',15)
text(0,1.2,parametros4,'FontSize',16)
xlim([6,4])
ylim([0,1.75])
```

% Fin

A.3. Propagador

El propagador se conforma por los siguientes archivos:

matrices.c Se definen las funciones para realizar operaciones matriciales bidimensionales. Incluye la definición de una estructura dedicada a matrices y operaciones básicas como creación, borrado, suma, multiplicación, impresión a pantalla y escritura a archivo.

matrices.h Contiene las declaraciones de las funciones halladas en matrices.c.

lineal.c Se definen las funciones para realizar los cálculos descritos en el Capítulo 3 a partir de las condiciones límite deseadas.

lineal.h Contiene las declaraciones de las funciones halladas en lineal.c.

error_iteraciones.c Se definen funciones para guardar los datos generados por cada iteración del análisis no lineal. Los datos generados permiten visualizar como converge o no la solución propuesta para la cavidad.

error_iteraciones.h Contiene las declaraciones de las funciones halladas en error_iteraciones.h.

no_lineal.c Se definen las funciones para realizar la propagación de haces gaussianos a través de un sistema $ABCD$, se calcúla el radio del haz y el radio del frente de onda y se modela la cavidad de la Figura 3.1 utilizando el método matricial desacoplado, como se estudió en la Sección 5.1. No incluye lente térmica.

no_lineal.h Contiene las declaraciones de las funciones halladas en no_lineal.c.

no_linealRK.c Realiza el análisis no lineal de la cavidad empleando el método de ecuaciones diferenciales desacopladas visto en la Sección 5.2. No incluye lente térmica.

no_linealRK.h Contiene las declaraciones de las funciones halladas en no_linealRK.c.

termico.c Contiene las rutinas para calcular el efecto de autoenfocamiento térmico como se estudió en la Sección 4.3.

termico.h Contiene las declaraciones de las funciones halladas en termico.c.

no_linealMatAstTerm.c Funciones para análisis no lineal considerando autoenfocamiento Kerr astigmático con y sin lente térmica astigmática presente como se describió en la sección 5.3.

no_linealMatAstTerm.h Contiene las declaraciones de las funciones halladas en no_linealMatAstTerm.c.

main.c Es la rutina principal para controlar el programa.

constantes.h Archivo con constantes físicas para el diseño de la cavidad.

Los mapas de estabilidad mostrados en la Sección 5.4 fueron generados a partir de los datos calculados por éste programa y graficados con Gnuplot.

A.4. Graficador de la propagación

El programa para graficar el perfil del haz al propagarse por la cavidad se conforma por los siguientes archivos:

matrices.c Se definen las funciones para realizar operaciones matriciales bidimensionales. Incluye la definición de una estructura dedicada a matrices y operaciones básicas como creación, borrado, suma, multiplicación, impresión a pantalla y escritura a archivo.

matrices.h Contiene las declaraciones de las funciones halladas en matrices.c.

lineal.c Se definen las funciones para realizar los cálculos descritos en el Capítulo 3 a partir de las condiciones límite deseadas.

lineal.h Contiene las declaraciones de las funciones halladas en lineal.c.

no_lineal.c Se definen las funciones para realizar la propagación de haces gaussianos a través de un sistema $ABCD$, se calcúla el radio del haz y el radio del frente de onda y se modela la cavidad de la Figura 3.1 utilizando el método matricial desacoplado, como se estudió en la Sección 5.1. No incluye lente térmica.

no_lineal.h Contiene las declaraciones de las funciones halladas en no_lineal.c.

termico.c Contiene las rutinas para calcular el efecto de autoenfocamiento térmico como se estudió en la Sección 4.3.

termico.h Contiene las declaraciones de las funciones halladas en termico.c.

propGrafica.c Contiene las rutinas para generar los datos correspondientes al radio del haz con respecto a la posición en la cavidad, generando un archivo que es procesado por Gnuplot para generar las gráficas.

propGrafica.h Contiene las declaraciones de las funciones halladas en propGrafica.c.

main.c Es la rutina principal para controlar el programa.

constantes.h Archivo con constantes físicas para el diseño de la cavidad.

A.5. Código fuente en C

A.5.1. matrices.c y matrices.h

matrices.c

```
/* Librería de operaciones matriciales básicas para
   números complejos */

#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <assert.h>
#include <string.h>
#include <math.h>

typedef struct
{
```

```
    int filas;
    int columnas;
    long double complex * datos;
}matriz;

/* Crea una matriz de filas por columnas con valores
    igual a 0.
Regresa NULL si filas y columnas <=0 o un puntero a la
    nueva matriz
en caso contrario */

matriz * nuevaMatriz(int filas, int columnas)
{
    if (filas<=0||columnas<=0) return NULL;
    // Reserva para la estructura de matriz.
    matriz *m=(matriz *)calloc(1,sizeof(matriz));
    //Fijar dimensiones
    m>filas=filas;
    m>columnas=columnas;
    // Reserva para un arreglo de longitud filas *
        columnas
    m>datos=(long double complex*)calloc(filas*columnas,
        sizeof(long double complex));
    return m;
}

/* Borra matriz, no elimina variable. */

matriz * borraMatriz(matriz * mtz)
{
    assert(mtz);
    // Libera datos de la matriz
    assert (mtz>datos);
    free(mtz>datos);
    mtz>datos=NULL;
    // Libera a mtz
    free(mtz);
    mtz=NULL;
    return mtz;
}

#define elem(mtz,fila,columna) \
    mtz>datos[(columna 1)*mtz>filas+(fila 1)]

/* Copia una matriz */
```

```
matriz * copiaMatriz(matriz * mtz)
{
    if (!mtz) return NULL;
    // Crea nueva matriz para guardar copya
    matriz * cp= nuevaMatriz(mtz->filas,mtz->columnas);
    // Copia datos de la matriz a cp.
    memcpy(cp->datos, mtz->datos, mtz->filas*mtz->
        columnas*sizeof(long double complex));
    return cp;
}

/* Imprime la matriz a stdout */

int fijaElemento(matriz * mtz, int fila, int columna,
    long double complex valor)
{
    if (!mtz) return 1;
    assert(mtz->datos);
    if (fila<=0 || fila > mtz->filas || columna<=0 ||
        columna > mtz->columnas)
         return 2;

    elem(mtz,fila,columna)=valor;
    return 0;
}

/* Recupera elemento de la matriz */

long double complex obtieneElemento(matriz * mtz, int
    fila, int columna)
{
    long double complex valor;
    if(!mtz) return 1;
    assert(mtz->datos);
    if(fila <=0 || fila > mtz->filas || columna<=0 ||
        columna>mtz->columnas) return 2;
    valor=elem(mtz,fila,columna);
    return valor;
}

/* Obtiene el número de filas de la matriz */

int nFilas(matriz * mtz)
{
    int *n;
    if (!mtz) return 1;
```

```
    *n = mtz->filas;
    return *n;
    }
/* Obtiene el número de columnas de la matriz */
int nColumnas(matriz * mtz)
{
    int *n;
    if (!mtz) return  1;
    *n=mtz->columnas;
    return *n;
}
int imprimeMatriz(matriz * mtz)
{
    if (!mtz) return  1;

    int fila, columna;
    for (fila=1;fila<=mtz->filas;++fila)
    {
        for(columna=1;columna<=mtz->columnas;++columna)
        {
            //Imprime el elemento de punto flotante
            printf("%.15Le+%.15Lei ", creall(elem(mtz,
                fila,columna)),cimagl(elem(mtz,fila,
                columna)));
        }
        printf("\n");
    }
    return 0;
}

matriz * sumaMatriz(matriz * A, matriz * B)
{
    assert(A&&B);
    assert(A->filas==B->filas);
    assert(A->columnas==B->columnas);
    matriz * suma;
    suma=nuevaMatriz(A->filas,A->columnas); // A == B
    int fila, columna;
    for (columna=A->columnas;columna>=1;columna  )
        for (fila=A->filas;fila>=1;fila  )
            elem(suma,fila,columna)=elem(A,fila,columna)+
                elem(B,fila,columna);
    return suma;
}

matriz * multMatriz(matriz * A, matriz * B)
```

```
{
    assert(A&&B);
    assert(A->columnas==B->filas);
    int fila, columna, k;
    matriz * prod=nuevaMatriz(A->filas,B->columnas);
    for (columna=B->columnas;columna>=1;columna--)
        for (fila=A->filas;fila>=1;fila--)
            {
                long double complex valor=0+I*0;
                for (k=A->columnas;k>=1;k--)
                    valor+=elem(A,fila,k)*elem(B,k,
                        columna);
                elem(prod,fila,columna)=valor;
            }
    return prod;
}

matriz * cteMultMatriz(long double complex Constante,
    matriz * mtz)
{
    assert(mtz);
    matriz *prod;
    prod=nuevaMatriz(mtz->filas,mtz->columnas);
    prod=copiaMatriz(mtz);
    int fila, columna;
    for(columna=1;columna<=mtz->columnas;columna++)
        for(fila=1;fila<=mtz->filas;fila++)
            elem(prod,fila,columna)=Constante*elem(prod,
                fila,columna);
    return prod;
}

matriz * divMatricesElemAElem(matriz * A, matriz * B) //
    División elemento a elemento de A/B: A11/B11, A12/B12,
     etc. Si número es no finito, lo hace cero.
{
    assert(A&&B);
    assert(A->filas==B->filas);
    assert(A->columnas==B->columnas);
    matriz * division;
    division=nuevaMatriz(A->filas,A->columnas); // A == B
    int fila, columna;
    for (columna=A->columnas;columna>=1;columna--)
        for (fila=A->filas;fila>=1;fila--){
            elem(division,fila,columna)=elem(A,fila,
                columna)/elem(B,fila,columna);
```

```
            long double divResultadoAbs=cabsl(elem(
                division,fila,columna));
            if(isfinite(divResultadoAbs)==0)
             elem(division,fila,columna)=0;
        }
    return division;
}

void escribeMatrizArchivo(matriz * mtz, char *nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int fila, columna;
    for (fila=1;fila<=mtz->filas;fila++)
    {
        for(columna=1;columna<=mtz->columnas;columna++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le+%.15Lei,", creall(
                elem(mtz,fila,columna)),cimagl(elem(mtz,
                fila,columna)));
        }
        fprintf(archivo,"\n");
    }
    fclose(archivo);
}

void escribeMatrizArchivo_ejes(matriz * mtz, matriz *
    epsilon1, matriz * epsilon2, char *nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int fila, columna;
    fprintf(archivo, " ,epsilon 2 [m],");
    for(columna=1;columna<=mtz->columnas;columna++)
        fprintf(archivo, "%.15Le,",creall(elem(epsilon2
            ,1,columna)));
    fprintf(archivo, "\nepsilon 1 [m],\n");
    for (fila=1;fila<=mtz->filas;fila++)
    {
        fprintf(archivo,"%.15Le, ,",creall(elem(epsilon1
            ,1,fila)));
        for(columna=1;columna<=mtz->columnas;columna++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le+%.15Lei,", creall(
```

```
                elem(mtz,fila,columna)),cimagl(elem(mtz,
                fila,columna)));
        }
        fprintf(archivo,"\n");
    }
    fclose(archivo);
}

void escribeMatrizArchivo_completo(matriz * mtz, matriz *
    epsilon1, matriz * epsilon2, long double *angulos,
    long double delta1, long double delta2, char*
    conjugado_corto, char *conjugado_largo, char *nombre,
    int fila_noAstig, int columna_noAstig, long double
    astigmatismo)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int fila, columna;
    fprintf(archivo, "Configuracion optima,\n\n");
    fprintf(archivo,"delta 1 [m],Theta 1 [rad],delta 2 [m
        ],Theta 2 [rad],Conjugado corto, Conjugado largo,
        Astigmatismo (spot tan / spot sag)\n");
    fprintf(archivo,"%.15Le,%.15Le,%.15Le,%.15Le,%s,%s
        ,%.15Le\n\n",delta1+creall(obtieneElemento(
        epsilon1,1,fila_noAstig)),angulos[0],delta2+creall
        (obtieneElemento(epsilon2,1,columna_noAstig)),
        angulos[1],conjugado_corto,conjugado_largo, creall
        (astigmatismo));
    fprintf(archivo, "epsilon 1 [m],epsilon2 [m]\n");
    fprintf(archivo, "%.15Le,%.15Le,\n\n",creall(
        obtieneElemento(epsilon1,1,fila_noAstig)),creall(
        obtieneElemento(epsilon2,1,columna_noAstig)));
    fprintf(archivo, " ,epsilon 2 [m],");
    for(columna=1;columna<=mtz>columnas;columna++)
        fprintf(archivo, "%.15Le,",creall(elem(epsilon2
            ,1,columna)));
    fprintf(archivo, "\nepsilon 1 [m],\n");
    for (fila=1;fila<=mtz>filas;fila++)
    {
        fprintf(archivo,"%.15Le, ,",creall(elem(epsilon1
            ,1,fila)));
        for(columna=1;columna<=mtz>columnas;columna++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le+%.15Lei,", creall(
                elem(mtz,fila,columna)),cimagl(elem(mtz,
```

```
                fila,columna)));
        }
        fprintf(archivo,"\n");
    }
    fclose(archivo);
}

void escribeMatrizDoubleArchivo(matriz * mtz, char *
    nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int fila, columna;
    fprintf(archivo,"double datos={");
    for (fila=1;fila<=mtz->filas 1; fila++)
    {
        fprintf(archivo,"{");
        for(columna=1;columna<=mtz->columnas;columna++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le,", creall(elem(mtz,
                fila,columna)));
        }
        fprintf(archivo,"},\n");
    }
    fila=mtz->filas;
    fprintf(archivo,"{");
    for(columna=1;columna<=mtz->columnas;columna++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le,", creall(elem(mtz,
                fila,columna)));
        }
    fprintf(archivo,"}};");
    fclose(archivo);
}


// Funciones válidas para matrices de 2x2

matriz * llenaMatriz(long double complex A, long double
    complex B,long double complex C, long double complex D
    )
{
    matriz * mtz;
    mtz = nuevaMatriz(2,2);
```

```
    fijaElemento(mtz,1,1,A);
    fijaElemento(mtz,1,2,B);
    fijaElemento(mtz,2,1,C);
    fijaElemento(mtz,2,2,D);
    return mtz;
}


void llenaMatrizSinCrear(matriz * mtz, long double
    complex A, long double complex B,long double complex C
    , long double complex D)
{
    fijaElemento(mtz,1,1,A);
    fijaElemento(mtz,1,2,B);
    fijaElemento(mtz,2,1,C);
    fijaElemento(mtz,2,2,D);
}

matriz * variasMultMatriciales(matriz ** arreglo_matrices
    , int num_matrices) // A1*A2*A3*...*An
{
    matriz * temp=llenaMatriz(1,0,0,1);
    int contador=num_matrices;
    for(contador=num_matrices 1;contador>=0;contador   )
    {
        matriz * hold=multMatriz(arreglo_matrices[
            contador],temp);
        llenaMatrizSinCrear(temp,obtieneElemento(hold
            ,1,1),obtieneElemento(hold,1,2),
            obtieneElemento(hold,2,1),obtieneElemento(hold
            ,2,2));
        borraMatriz(hold);
    }

    return temp;
}
```

matrices.h

```
#ifndef MATRICES_H_INCLUDED
#define MATRICES_H_INCLUDED

#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <assert.h>
```

```
#include <string.h>

typedef struct
{
    int filas;
    int columnas;
    long double complex * datos;
}matriz;

#define elem(mtz,fila,columna) mtz->datos[(columna-1)*mtz
    ->filas+(fila-1)]

#define length(x) (sizeof(x)/sizeof((x)[0]))

/* Crea una matriz de filas por columnas con valores
    igual a 0.
Regresa NULL si filas y columnas <=0 o un puntero a la
    nueva matriz
en caso contrario */

matriz * nuevaMatriz(int filas, int columnas);
/* Borra matriz, no elimina variable. */
matriz * borraMatriz(matriz * mtz);
/* Copia una matriz */
matriz * copiaMatriz(matriz * mtz);
/* Imprime la matriz a stdout */
int fijaElemento(matriz * mtz, int fila, int columna,
    long double complex valor);
/* Recupera elemento de la matriz */
long double complex obtieneElemento(matriz * mtz, int
    fila, int columna);
/* Obtiene el número de filas de la matriz */
int nFilas(matriz * mtz);
/* Obtiene el número de columnas de la matriz */
int nColumnas(matriz * mtz);
/* Imprime matriz a stdout */
int imprimeMatriz(matriz * mtz);
matriz * sumaMatriz(matriz * A, matriz * B);
matriz * multMatriz(matriz * A, matriz * B);
matriz * cteMultMatriz(long double complex Constante,
    matriz * mtz);
matriz * divMatricesElemAElem(matriz * A, matriz * B);
void escribeMatrizArchivo(matriz * mtz, char *nombre);
void escribeMatrizArchivo_ejes(matriz * mtz, matriz *
    epsilon1, matriz * epsilon2, char *nombre);
void escribeMatrizArchivo_completo(matriz * mtz, matriz *
```

```
     epsilon1, matriz * epsilon2, long double *angulos,
    long double delta1, long double delta2, char*
    conjugado_corto, char *conjugado_largo, char *nombre,
    int fila_noAstig, int columna_noAstig, long double
    astigmatismo);
void escribeMatrizDoubleArchivo(matriz * mtz, char *
    nombre);
// Sólo para matrices de 2x2
matriz * llenaMatriz(long double complex A, long double
    complex B,long double complex C, long double complex D
    );
void llenaMatrizSinCrear(matriz * mtz, long double
    complex A, long double complex B,long double complex C
    , long double complex D);
matriz * variasMultMatriciales(matriz ** arreglo_matrices
    , int num_matrices);

#endif // MATRICES_H_INCLUDED
```

A.5.2. lineal.c y lineal.h

lineal.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "matrices.h"
#include "complex.h"

// Rutinas para cálculo lineal

void anguloLineal(char *conjugado_corto, char *
    conjugado_largo, long double L, long double n, long
    double L1, long double L2, long double f1, long double
     f2,long double * Angulos)
{
     long double A=L/2*(1/powl(n,3) 1/n);
     long double x_1=0.0, x_2=0.0;
     char inf[]="inf";
     char fin[]="fin";
     if(strcmp(conjugado_corto,inf)==0)
          x_1=(A+sqrtl(powl(A,2)+4*powl(f1,2)))/(2*f1);
          else if(strcmp(conjugado_corto,fin)==0)
          x_1=(A*(powl(L1,2)+powl(f1,2))+sqrtl(4*powl(f1,2)
              *powl(L1,4)+powl(A,2)*powl((powl(f1,2) powl(L1
              ,2)),2)))/(2*(f1*L1*(L1+A)));
```

```
        //else return 2;
    if(strcmp(conjugado_largo,inf)==0)
        x_2=(A+sqrtl(powl(A,2)+4*powl(f2,2)))/(2*f2);
        else if(strcmp(conjugado_largo,fin)==0)
        x_2=(A*(powl(L2,2)+powl(f2,2))+sqrtl(4*powl(f2,2)
            *powl(L2,4)+powl(A,2)*powl((powl(f2,2) powl(L2
            ,2)),2)))/(2*(f2*L2*(L2+A)));
        //else return 2;
    Angulos[0]=acosl(x_1);
    Angulos[1]=acosl(x_2);
}

long double distanciaCristal(char *tipo_conjugado, long
    double f, long double Longitud, long double L, long
    double n, long double theta)
{
    char inf[]="inf";
    char fin[]="fin";
    long double delta=0.0;
    if(strcmp(tipo_conjugado,inf)==0)
        delta=f/cosl(theta) L/(2*n);
    else if(strcmp(tipo_conjugado,fin)==0)
        delta=Longitud*f/cosl(theta)/(Longitud f/cosl(
            theta)) L/(2*n);
    else
        printf("\nError\n");
    return delta;

}

long double spot_lineal(matriz * ABCD, long double lambda
    )
{
    assert(ABCD);
    long double estabilidad, A, B, D, w;
    A=obtieneElemento(ABCD,1,1);
    B=obtieneElemento(ABCD,1,2);
    D=obtieneElemento(ABCD,2,2);
    estabilidad=0.5*(A+D);
    if(estabilidad<=1||estabilidad>=1) w=0;
    else
        w=sqrtl(lambda*fabs(B)/(M_PI*sqrtl(1 powl(((A+D)
            /2),2))));
    if(isfinite(w)==0)
            w=0;
    return w;
```

```
}
void calculoLineal(char *nombre, char *conjugado_corto,
    char *conjugado_largo, long double n, long double
    lambda, long double L1,long double L2,long double L,
    long double f1,long double f2, matriz *wtEM1, matriz *
    wsEM1, matriz *wtEM2, matriz *wsEM2, matriz *epsilon1,
     matriz *epsilon2,matriz *qtEM1, matriz *qsEM1, matriz
     *qtEM2, matriz *qsEM2)
{
     long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
     assert(strlen(conjugado_corto)==3);
     assert(strlen(conjugado_largo)==3);
     assert(wtEM1);
     assert(wsEM1);
     assert(wtEM2);
     assert(wsEM2);
     assert(qtEM1);
     assert(qsEM1);
     assert(qtEM2);
     assert(qsEM2);

     // Creando datos para sistema óptico
     matriz *EM1tan1,*EM1tan2,*EM1tan3,*EM1tan4,*EM1tan5,*
         EM1tan6,*EM1tan7,*EM1tan8,*EM1tan9,*EM1tan10,*
         EM1tan11,*EM1tan12,*EM1tan13,*EM1tan14,*EM1tan15;
     matriz *EM1sag1,*EM1sag2,*EM1sag3,*EM1sag4,*EM1sag5,*
         EM1sag6,*EM1sag7,*EM1sag8,*EM1sag9,*EM1sag10,*
         EM1sag11,*EM1sag12,*EM1sag13,*EM1sag14,*EM1sag15;
     matriz *EM2tan1,*EM2tan2,*EM2tan3,*EM2tan4,*EM2tan5,*
         EM2tan6,*EM2tan7,*EM2tan8,*EM2tan9,*EM2tan10,*
         EM2tan11,*EM2tan12,*EM2tan13,*EM2tan14,*EM2tan15;
     matriz *EM2sag1,*EM2sag2,*EM2sag3,*EM2sag4,*EM2sag5,*
         EM2sag6,*EM2sag7,*EM2sag8,*EM2sag9,*EM2sag10,*
         EM2sag11,*EM2sag12,*EM2sag13,*EM2sag14,*EM2sag15;
     anguloLineal(conjugado_corto,conjugado_largo,L,n,L1,
         L2,f1,f2,angulos);
     delta1=distanciaCristal(conjugado_corto,f1,L1,L,n,
         angulos[0]);
     delta2=distanciaCristal(conjugado_largo,f2,L2,L,n,
         angulos[1]);

     // Cálculo de distancias focales
     f1t=f1*cosl(angulos[0]);
     f2t=f2*cosl(angulos[1]);
     f1s=f1/cosl(angulos[0]);
     f2s=f2/cosl(angulos[1]);
```

```
/* Llenando matrices "estáticas"*/
// EM1, Caso tangencial
EM1tan1=llenaMatriz(1,L1,0,1);
EM1tan2=llenaMatriz(1,0,1/f1t,1);
EM1tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1tan4=llenaMatriz(1,L/powl(n,3),0,1);
EM1tan5=nuevaMatriz(2,2);//EM1tan5 pendiente
EM1tan6=llenaMatriz(1,0,1/f2t,1);
EM1tan7=llenaMatriz(1,L2,0,1);
EM1tan8=llenaMatriz(1,0,0,1);
EM1tan9=llenaMatriz(1,L2,0,1);
EM1tan10=llenaMatriz(1,0,1/f2t,1);
EM1tan11=nuevaMatriz(2,2);      //EM1tan11 pendiente
EM1tan12=llenaMatriz(1,L/powl(n,3),0,1);
EM1tan13=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1tan14=llenaMatriz(1,0,1/f1t,1);
EM1tan15=llenaMatriz(1,L1,0,1);

// EM1, Caso sagital
EM1sag1=llenaMatriz(1,L1,0,1);
EM1sag2=llenaMatriz(1,0,1/f1s,1);
EM1sag3=nuevaMatriz(2,2);
EM1sag4=llenaMatriz(1,L/n,0,1);
EM1sag5=nuevaMatriz(2,2);    //EM1sag5 pendiente
EM1sag6=llenaMatriz(1,0,1/f2s,1);
EM1sag7=llenaMatriz(1,L2,0,1);
EM1sag8=llenaMatriz(1,0,0,1);
EM1sag9=llenaMatriz(1,L2,0,1);
EM1sag10=llenaMatriz(1,0,1/f2s,1);
EM1sag11=nuevaMatriz(2,2);      //EM1sag11 pendiente
EM1sag12=llenaMatriz(1,L/n,0,1);
EM1sag13=nuevaMatriz(2,2);
EM1sag14=llenaMatriz(1,0,1/f1s,1);
EM1sag15=llenaMatriz(1,L1,0,1);

// EM2, Caso tangencial
EM2tan1=llenaMatriz(1,L2,0,1);
EM2tan2=llenaMatriz(1,0,1/f2t,1);
EM2tan3=nuevaMatriz(2,2);//EM2tan3=llenaMatriz(1,
    delta2+eps,0,1);
EM2tan4=llenaMatriz(1,L/powl(n,3),0,1);
EM2tan5=nuevaMatriz(2,2);
EM2tan6=llenaMatriz(1,0,1/f1t,1);
```

```
EM2tan7=llenaMatriz(1,L1,0,1);
EM2tan8=llenaMatriz(1,0,0,1);
EM2tan9=llenaMatriz(1,L1,0,1);
EM2tan10=llenaMatriz(1,0, 1/f1t,1);
EM2tan11=nuevaMatriz(2,2);
EM2tan12=llenaMatriz(1,L/powl(n,3),0,1);
EM2tan13=nuevaMatriz(2,2);//EM2tan13=llenaMatriz(1,
    delta1,0,1);
EM2tan14=llenaMatriz(1,0, 1/f2t,1);
EM2tan15=llenaMatriz(1,L2,0,1);

// EM2, Caso sagital
EM2sag1=llenaMatriz(1,L2,0,1);
EM2sag2=llenaMatriz(1,0, 1/f2s,1);
EM2sag3=nuevaMatriz(2,2);    //EM2sag3=llenaMatriz(1,
    delta2+e,0,1);
EM2sag4=llenaMatriz(1,L/n,0,1);
EM2sag5=nuevaMatriz(2,2);
EM2sag6=llenaMatriz(1,0, 1/f1s,1);
EM2sag7=llenaMatriz(1,L1,0,1);
EM2sag8=llenaMatriz(1,0,0,1);
EM2sag9=llenaMatriz(1,L1,0,1);
EM2sag10=llenaMatriz(1,0, 1/f1s,1);
EM2sag11=nuevaMatriz(2,2);
EM2sag12=llenaMatriz(1,L/n,0,1);
EM2sag13=nuevaMatriz(2,2);   //EM2sag13=llenaMatriz(1,
    delta2+e,0,1);
EM2sag14=llenaMatriz(1,0, 1/f2s,1);
EM2sag15=llenaMatriz(1,L2,0,1);

//Ciclo
for(int index1=epsilon1>columnas;index1>0;index1  )
    {
    for(int index2=epsilon2>columnas;index2>0;index2
        )
    {

                    // Llena las matrices y realiza
                        las multiplicaciones

                    // Para epsilon1 EM1: 3 y 13; EM2
                        : 5 y 11
                    llenaMatrizSinCrear(EM1tan3,1,
                        delta1+obtieneElemento(
                        epsilon1,1,index1),0,1);
                    llenaMatrizSinCrear(EM1tan13,1,
```

```
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM1sag3,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM1sag13,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM2tan5,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM2tan11,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM2sag5,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(EM2sag11,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1); //
        Optimizar asignaciones.

    // Para epsilon2  EM1: 5 y 11;
        EM2: 3 y 13
    llenaMatrizSinCrear(EM1tan5,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM1tan11,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM1sag5,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM1sag11,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM2tan3,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM2tan13,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM2sag3,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(EM2sag13,1,
```

```
                delta2+obtieneElemento(
                epsilon2,1,index2),0,1); //
                Optimizar asignaciones.
            matriz *EM1tan_piezas[]={EM1tan15
                ,EM1tan14,EM1tan13,EM1tan12,
                EM1tan11,EM1tan10,EM1tan9,
                EM1tan8,EM1tan7,EM1tan6,
                EM1tan5,EM1tan4,EM1tan3,
                EM1tan2,EM1tan1};
            matriz *EM1sag_piezas[]={EM1sag15
                ,EM1sag14,EM1sag13,EM1sag12,
                EM1sag11,EM1sag10,EM1sag9,
                EM1sag8,EM1sag7,EM1sag6,
                EM1sag5,EM1sag4,EM1sag3,
                EM1sag2,EM1sag1};
            matriz *EM2tan_piezas[]={EM2tan15
                ,EM2tan14,EM2tan13,EM2tan12,
                EM2tan11,EM2tan10,EM2tan9,
                EM2tan8,EM2tan7,EM2tan6,
                EM2tan5,EM2tan4,EM2tan3,
                EM2tan2,EM2tan1};
            matriz *EM2sag_piezas[]={EM2sag15
                ,EM2sag14,EM2sag13,EM2sag12,
                EM2sag11,EM2sag10,EM2sag9,
                EM2sag8,EM2sag7,EM2sag6,
                EM2sag5,EM2sag4,EM2sag3,
                EM2sag2,EM2sag1};
            matriz *EM1tan, *EM1sag, *EM2tan,
                *EM2sag;
            EM1tan=variasMultMatriciales(
                EM1tan_piezas,15);
            EM1sag=variasMultMatriciales(
                EM1sag_piezas,15);
            EM2tan=variasMultMatriciales(
                EM2tan_piezas,15);
            EM2sag=variasMultMatriciales(
                EM2sag_piezas,15);
            long double spot_EM1tan=
                spot_lineal(EM1tan,lambda);
            long double spot_EM1sag=
                spot_lineal(EM1sag,lambda);
            long double spot_EM2tan=
                spot_lineal(EM2tan,lambda);
            long double spot_EM2sag=
                spot_lineal(EM2sag,lambda);
long double complex q_EM1tan=1/((I*lambda)/(
```

```
                M_PI*powl(spot_EM1tan,2)));
            long double complex q_EM1sag=1/((I*lambda)/(
                M_PI*powl(spot_EM1sag,2)));
            long double complex q_EM2tan=1/((I*lambda)/(
                M_PI*powl(spot_EM2tan,2)));
            long double complex q_EM2sag=1/((I*lambda)/(
                M_PI*powl(spot_EM2sag,2)));
                        fijaElemento(wtEM1,index1,index2,
                            spot_EM1tan);
                        fijaElemento(wsEM1,index1,index2,
                            spot_EM1sag);
                        fijaElemento(wtEM2,index1,index2,
                            spot_EM2tan);
                        fijaElemento(wsEM2,index1,index2,
                            spot_EM2sag);
                        fijaElemento(qtEM1,index1,index2,
                            q_EM1tan);
                        fijaElemento(qsEM1,index1,index2,
                            q_EM1sag);
                        fijaElemento(qtEM2,index1,index2,
                            q_EM2tan);
                        fijaElemento(qsEM2,index1,index2,
                            q_EM2sag);
                        // Limpia matrices
                        EM1tan=borraMatriz(EM1tan);
                        EM1sag=borraMatriz(EM1sag);
                        EM2tan=borraMatriz(EM2tan);
                        EM2sag=borraMatriz(EM2sag);
        // TODO: Agregar cálculo de spot, agregar método
            para graficar.
        //printf("\n Sigo vivo\n\n");
                    }
        }
    // Guarda en archivo
    escribeMatrizArchivo(qtEM1,"qtEM1.csv");
    escribeMatrizArchivo(qtEM2,"qtEM2.csv");
    escribeMatrizArchivo(qsEM1,"qsEM1.csv");
    escribeMatrizArchivo(qsEM2,"qsEM2.csv");

    escribeMatrizArchivo(wtEM1,"tanEM1.csv");
    escribeMatrizArchivo(wsEM1,"sagEM1.csv");
    escribeMatrizArchivo(wtEM2,"tanEM2.csv");
    escribeMatrizArchivo(wsEM2,"sagEM2.csv");

    // Limpieza
```

```
EM1tan1=borraMatriz(EM1tan1);
EM1tan2=borraMatriz(EM1tan2);
EM1tan3=borraMatriz(EM1tan3);
EM1tan4=borraMatriz(EM1tan4);
EM1tan5=borraMatriz(EM1tan5);
EM1tan6=borraMatriz(EM1tan6);
EM1tan7=borraMatriz(EM1tan7);
EM1tan8=borraMatriz(EM1tan8);
EM1tan9=borraMatriz(EM1tan9);
EM1tan10=borraMatriz(EM1tan10);
EM1tan11=borraMatriz(EM1tan11);
EM1tan12=borraMatriz(EM1tan12);
EM1tan13=borraMatriz(EM1tan13);
EM1tan14=borraMatriz(EM1tan14);
EM1tan15=borraMatriz(EM1tan15);

EM1sag1=borraMatriz(EM1sag1);
EM1sag2=borraMatriz(EM1sag2);
EM1sag3=borraMatriz(EM1sag3);
EM1sag4=borraMatriz(EM1sag4);
EM1sag5=borraMatriz(EM1sag5);
EM1sag6=borraMatriz(EM1sag6);
EM1sag7=borraMatriz(EM1sag7);
EM1sag8=borraMatriz(EM1sag8);
EM1sag9=borraMatriz(EM1sag9);
EM1sag10=borraMatriz(EM1sag10);
EM1sag11=borraMatriz(EM1sag11);
EM1sag12=borraMatriz(EM1sag12);
EM1sag13=borraMatriz(EM1sag13);
EM1sag14=borraMatriz(EM1sag14);
EM1sag15=borraMatriz(EM1sag15);

EM2tan1=borraMatriz(EM2tan1);
EM2tan2=borraMatriz(EM2tan2);
EM2tan3=borraMatriz(EM2tan3);
EM2tan4=borraMatriz(EM2tan4);
EM2tan5=borraMatriz(EM2tan5);
EM2tan6=borraMatriz(EM2tan6);
EM2tan7=borraMatriz(EM2tan7);
EM2tan8=borraMatriz(EM2tan8);
EM2tan9=borraMatriz(EM2tan9);
EM2tan10=borraMatriz(EM2tan10);
EM2tan11=borraMatriz(EM2tan11);
EM2tan12=borraMatriz(EM2tan12);
EM2tan13=borraMatriz(EM2tan13);
EM2tan14=borraMatriz(EM2tan14);
```

```
    EM2tan15=borraMatriz(EM2tan15);

    EM2sag1=borraMatriz(EM2sag1);
    EM2sag2=borraMatriz(EM2sag2);
    EM2sag3=borraMatriz(EM2sag3);
    EM2sag4=borraMatriz(EM2sag4);
    EM2sag5=borraMatriz(EM2sag5);
    EM2sag6=borraMatriz(EM2sag6);
    EM2sag7=borraMatriz(EM2sag7);
    EM2sag8=borraMatriz(EM2sag8);
    EM2sag9=borraMatriz(EM2sag9);
    EM2sag10=borraMatriz(EM2sag10);
    EM2sag11=borraMatriz(EM2sag11);
    EM2sag12=borraMatriz(EM2sag12);
    EM2sag13=borraMatriz(EM2sag13);
    EM2sag14=borraMatriz(EM2sag14);
    EM2sag15=borraMatriz(EM2sag15);
}

/* Las rutas de propagación del haz se denominan como (1)
    y (2). La ruta (1) contempla la propagación del
    espejo plano 1 (FM1) al espejo curvo 1 (CM1),
de CM1 al espejo curvo 2 (CM2) a través del medio Kerr,
    de CM2 al espejo plano 2 (FM2) y de FM2 de vuelta a
    FM1.
La ruta (2) describe la propagación de FM1 a FM2, de FM2
    a CM2, de CM2 a CM1 atravezando el medio Kerr y de CM1
    a FM1.

Se realizarán cálculos para los planos tangencial y
    sagital de ambas rutas, compensando el astigmatismo
    lineal.
El conjugado corto será equivalente al tramo entre FM1 a
    CM1 (tramo a), y el conjugado largo será la suma de
    tramos de CM2 a FM2 y de FM2 a FM1.
Para satisfacer la condición de autoconsistencia, sólo se
    admitirán conjugados del mismo tipo, esto es,
    infinito infinito o finito finito. */

void calculoLinealAnillo(char *nombre, char *
    conjugado_corto, char *conjugado_largo, long double n,
    long double lambda, long double a, long double b,
    long double c,long double L,long double f1,long double
    f2, matriz *wt1, matriz *ws1, matriz *wt2, matriz *
    ws2, matriz *epsilon1, matriz *epsilon2,matriz *qt1,
    matriz *qs1, matriz *qt2, matriz *qs2)
```

```
{
    long double angulos[2], delta1, delta2, f1t, f1s, f2t, f2s;
    long double L1=a;
    long double L2=b+c;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto, conjugado_largo)==0);
    assert(wt1);
    assert(ws1);
    assert(wt2);
    assert(ws2);
    assert(qt1);
    assert(qs1);
    assert(qt2);
    assert(qs2);

    // Creando datos para sistema óptico
    matriz *Ruta1tan1, *Ruta1tan2, *Ruta1tan3, *Ruta1tan4, *
        Ruta1tan5, *Ruta1tan6, *Ruta1tan7;
    matriz *Ruta1sag1, *Ruta1sag2, *Ruta1sag3, *Ruta1sag4, *
        Ruta1sag5, *Ruta1sag6, *Ruta1sag7;
    matriz *Ruta2tan1, *Ruta2tan2, *Ruta2tan3, *Ruta2tan4, *
        Ruta2tan5, *Ruta2tan6, *Ruta2tan7;
    matriz *Ruta2sag1, *Ruta2sag2, *Ruta2sag3, *Ruta2sag4, *
        Ruta2sag5, *Ruta2sag6, *Ruta2sag7;
    anguloLineal(conjugado_corto, conjugado_largo, L, n, L1,
        L2, f1, f2, angulos);
    delta1=distanciaCristal(conjugado_corto, f1, L1, L, n,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo, f2, L2, L, n,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);
    f2s=f2/cosl(angulos[1]);

    /* Llenando matrices "estáticas"*/

    // Ruta 1, caso tangencial
    Ruta1tan1=llenaMatriz(1, L1, 0, 1); // L1=a
    Ruta1tan2=llenaMatriz(1, 0, 1/f1t, 1);
    Ruta1tan3=nuevaMatriz(2, 2); // Pendiente
    Ruta1tan4=llenaMatriz(1, L/powl(n, 3), 0, 1); // Medio
```

```
    Kerr (lineal)
Ruta1tan5=nuevaMatriz(2,2);//Ruta1tan5 pendiente
Ruta1tan6=llenaMatriz(1,0,1/f2t,1);
Ruta1tan7=llenaMatriz(1,L2,0,1); // L2=b+c

// Ruta 1, caso sagital
Ruta1sag1=llenaMatriz(1,L1,0,1); // L1=a
Ruta1sag2=llenaMatriz(1,0,1/f1s,1);
Ruta1sag3=nuevaMatriz(2,2); // Pendiente
Ruta1sag4=llenaMatriz(1,L/n,0,1); // Medio Kerr (
    lineal)
Ruta1sag5=nuevaMatriz(2,2);//Ruta1sag5 pendiente
Ruta1sag6=llenaMatriz(1,0,1/f2s,1);
Ruta1sag7=llenaMatriz(1,L2,0,1); // L2=b+c

// Ruta 2, caso tangencial
Ruta2tan1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2tan2=llenaMatriz(1,0,1/f2t,1);
Ruta2tan3=nuevaMatriz(2,2); // Pendiente
Ruta2tan4=llenaMatriz(1,L/powl(n,3),0,1); // Medio
    Kerr (lineal)
Ruta2tan5=nuevaMatriz(2,2);//Ruta2tan5 pendiente
Ruta2tan6=llenaMatriz(1,0,1/f1t,1);
Ruta2tan7=llenaMatriz(1,L1,0,1); // L1=a

// Ruta 2, caso sagital
Ruta2sag1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2sag2=llenaMatriz(1,0,1/f2s,1);
Ruta2sag3=nuevaMatriz(2,2); // Pendiente
Ruta2sag4=llenaMatriz(1,L/n,0,1); // Medio Kerr (
    lineal)
Ruta2sag5=nuevaMatriz(2,2);//Ruta2sag5 pendiente
Ruta2sag6=llenaMatriz(1,0,1/f1s,1);
Ruta2sag7=llenaMatriz(1,L1,0,1); // L1=a

//Ciclo
for(int index1=epsilon1>columnas;index1>0;index1  )
    {
    for(int index2=epsilon2>columnas;index2>0;index2
        )
    {

                    // Llena las matrices y realiza
                        las multiplicaciones

                    llenaMatrizSinCrear(Ruta1tan3,1,
```

```
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(Ruta1tan5,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(Ruta1sag3,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(Ruta1sag5,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(Ruta2tan3,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(Ruta2tan5,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);
    llenaMatrizSinCrear(Ruta2sag3,1,
        delta2+obtieneElemento(
        epsilon2,1,index2),0,1);
    llenaMatrizSinCrear(Ruta2sag5,1,
        delta1+obtieneElemento(
        epsilon1,1,index1),0,1);

    matriz *Ruta1tan_piezas[]={
        Ruta1tan1,Ruta1tan2,Ruta1tan3,
        Ruta1tan4,Ruta1tan5,Ruta1tan6,
        Ruta1tan7};
    matriz *Ruta1sag_piezas[]={
        Ruta1sag1,Ruta1sag2,Ruta1sag3,
        Ruta1sag4,Ruta1sag5,Ruta1sag6,
        Ruta1sag7};
    matriz *Ruta2tan_piezas[]={
        Ruta2tan1,Ruta2tan2,Ruta2tan3,
        Ruta2tan4,Ruta2tan5,Ruta2tan6,
        Ruta2tan7};
    matriz *Ruta2sag_piezas[]={
        Ruta2sag1,Ruta2sag2,Ruta2sag3,
        Ruta2sag4,Ruta2sag5,Ruta2sag6,
        Ruta2sag7};
    matriz *Ruta1tan, *Ruta1sag, *
        Ruta2tan, *Ruta2sag;

    Ruta1tan=variasMultMatriciales(
        Ruta1tan_piezas,7);
    Ruta1sag=variasMultMatriciales(
```

```
                    Ruta1sag_piezas,7);
                Ruta2tan=variasMultMatriciales(
                    Ruta2tan_piezas,7);
                Ruta2sag=variasMultMatriciales(
                    Ruta2sag_piezas,7);

                long double spot_Ruta1tan=
                    spot_lineal(Ruta1tan,lambda);
                long double spot_Ruta1sag=
                    spot_lineal(Ruta1sag,lambda);
                long double spot_Ruta2tan=
                    spot_lineal(Ruta2tan,lambda);
                long double spot_Ruta2sag=
                    spot_lineal(Ruta2sag,lambda);
        long double complex q_Ruta1tan=1/((I*lambda)
            /(M_PI*powl(spot_Ruta1tan,2)));
        long double complex q_Ruta1sag=1/((I*lambda)
            /(M_PI*powl(spot_Ruta1sag,2)));
        long double complex q_Ruta2tan=1/((I*lambda)
            /(M_PI*powl(spot_Ruta2tan,2)));
        long double complex q_Ruta2sag=1/((I*lambda)
            /(M_PI*powl(spot_Ruta2sag,2)));
                fijaElemento(wt1,index1,index2,
                    spot_Ruta1tan);
                fijaElemento(ws1,index1,index2,
                    spot_Ruta1sag);
                fijaElemento(wt2,index1,index2,
                    spot_Ruta2tan);
                fijaElemento(ws2,index1,index2,
                    spot_Ruta2sag);
                fijaElemento(qt1,index1,index2,
                    q_Ruta1tan);
                fijaElemento(qs1,index1,index2,
                    q_Ruta1sag);
                fijaElemento(qt2,index1,index2,
                    q_Ruta2tan);
                fijaElemento(qs2,index1,index2,
                    q_Ruta2sag);
                // Limpia matrices
                Ruta1tan=borraMatriz(Ruta1tan);
                Ruta1sag=borraMatriz(Ruta1sag);
                Ruta2tan=borraMatriz(Ruta2tan);
                Ruta2sag=borraMatriz(Ruta2sag);
    // TODO: Agregar cálculo de spot, agregar método
        para graficar.
    //printf("\n Sigo vivo\n\n");
```

```
            }
        }
    // Guarda en archivo
    escribeMatrizArchivo(qt1,"qt1_lin_anillo.csv");
    escribeMatrizArchivo(qt2,"qt2_lin_anillo.csv");
    escribeMatrizArchivo(qs1,"qs1_lin_anillo.csv");
    escribeMatrizArchivo(qs2,"qs2_lin_anillo.csv");

    /*escribeMatrizArchivo(wt1,"tan1.csv");
    escribeMatrizArchivo(ws1,"sag1.csv");
    escribeMatrizArchivo(wt2,"tan2.csv");
    escribeMatrizArchivo(ws2,"sag2.csv");*/

    // Limpieza

    Ruta1tan1=borraMatriz(Ruta1tan1);
    Ruta1tan2=borraMatriz(Ruta1tan2);
    Ruta1tan3=borraMatriz(Ruta1tan3);
    Ruta1tan4=borraMatriz(Ruta1tan4);
    Ruta1tan5=borraMatriz(Ruta1tan5);
    Ruta1tan6=borraMatriz(Ruta1tan6);
    Ruta1tan7=borraMatriz(Ruta1tan7);

    Ruta1sag1=borraMatriz(Ruta1sag1);
    Ruta1sag2=borraMatriz(Ruta1sag2);
    Ruta1sag3=borraMatriz(Ruta1sag3);
    Ruta1sag4=borraMatriz(Ruta1sag4);
    Ruta1sag5=borraMatriz(Ruta1sag5);
    Ruta1sag6=borraMatriz(Ruta1sag6);
    Ruta1sag7=borraMatriz(Ruta1sag7);


    Ruta2tan1=borraMatriz(Ruta2tan1);
    Ruta2tan2=borraMatriz(Ruta2tan2);
    Ruta2tan3=borraMatriz(Ruta2tan3);
    Ruta2tan4=borraMatriz(Ruta2tan4);
    Ruta2tan5=borraMatriz(Ruta2tan5);
    Ruta2tan6=borraMatriz(Ruta2tan6);
    Ruta2tan7=borraMatriz(Ruta2tan7);


    Ruta2sag1=borraMatriz(Ruta2sag1);
    Ruta2sag2=borraMatriz(Ruta2sag2);
    Ruta2sag3=borraMatriz(Ruta2sag3);
    Ruta2sag4=borraMatriz(Ruta2sag4);
    Ruta2sag5=borraMatriz(Ruta2sag5);
```

```
    Ruta2sag6=borraMatriz(Ruta2sag6);
    Ruta2sag7=borraMatriz(Ruta2sag7);

}
```

lineal.h

```
#ifndef LINEAL_H_INCLUDED
#define LINEAL_H_INCLUDED

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "matrices.h"

// Rutinas para cálculo lineal

void anguloLineal(char *conjugado_corto, char *
    conjugado_largo, long double L, long double n, long
    double L1, long double L2, long double f1, long double
     f2,long double * Angulos);
long double distanciaCristal(char *tipo_conjugado, long
    double f, long double Longitud, long double L, long
    double n, long double theta);
long double spot_lineal(matriz * ABCD, long double lambda
    );
void calculoLineal(char *nombre, char *conjugado_corto,
    char *conjugado_largo, long double n, long double
    lambda, long double L1,long double L2,long double L,
    long double f1,long double f2, matriz *wtEM1, matriz *
    wsEM1, matriz *wtEM2, matriz *wsEM2, matriz *epsilon1,
     matriz *epsilon2,matriz *qtEM1, matriz *qsEM1, matriz
     *qtEM2, matriz *qsEM2);
void calculoLinealAnillo(char *nombre, char *
    conjugado_corto, char *conjugado_largo, long double n,
     long double lambda, long double a, long double b,
    long double c,long double L,long double f1,long double
     f2, matriz *wt1, matriz *ws1, matriz *wt2, matriz *
    ws2, matriz *epsilon1, matriz *epsilon2,matriz *qt1,
    matriz *qs1, matriz *qt2, matriz *qs2);

#endif // LINEAL_H_INCLUDED
```

A.5.3. error_iteraciones.c y error_iteraciones.h

error_iteraciones.c

```
// Guardará los datos generados por cada caso.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void guardaVariaciones(long double * spots, int *
    iteraciones, int iterMax, long double epsilon1, long
    double epsilon2, char * tipo, char * subCarpeta,
    _Bool termino)
{
    char fecha[12];
    time_t rawtime;
    struct tm * timeinfo;
    time(&rawtime);
    timeinfo=localtime(&rawtime);
    strftime(fecha,30,"%F_%H%M%S",timeinfo);
    char *bandera;
    if(termino==1)
    {
        bandera=malloc(3);
        bandera[0]='s';
        bandera[1]='i';
        bandera[2]=0;
    }
    else
    {
        bandera=malloc(3);
        bandera[0]='n';
        bandera[1]='o';
        bandera[2]=0;
    }

    int caracteres=snprintf(NULL,0,"./CSV/analisis_error
        /%s/spots_%sE1_%.2Le_E2_%.2Le_%s_%s.csv",
        subCarpeta,tipo,epsilon1,epsilon2,fecha,bandera);
    char * buffer=malloc(caracteres+1);
    sprintf(buffer,"./CSV/analisis_error/%s/spots_%sE1_
        %.2Le_E2_%.2Le_%s_%s.csv",subCarpeta,tipo,epsilon1
        ,epsilon2,fecha,bandera);
    FILE *archivo;
```

```
    archivo = fopen(buffer,"w");
    fprintf(archivo,"Iteracion,W\n");
    int i;
    for(i=0;i<iterMax;i++)
    {
        //Imprime el elemento de punto flotante
        fprintf(archivo, "%d,%.15Le\n",iteraciones[i],
            spots[i]);
    }
    fclose(archivo);
    free(buffer);
    free(bandera);
}

int existeArchivo(const char *nombre)
{
   FILE *fp = fopen (nombre, "r");
   if (fp!=NULL) fclose (fp);
   return (fp!=NULL);
}

void guardaSpotIteraciones(long double spot, int iterMax,
    long double epsilon1, long double epsilon2, char *
    tipo, _Bool termino)
{
    int caracteres=snprintf(NULL,0,"./CSV/%
        s_spotIteraciones.csv",tipo);
    char *nombre=malloc(caracteres+1);
    sprintf(nombre,"./CSV/%s_spotIteraciones.csv",tipo);
    FILE *archivo;
    if(termino==0)
        spot=0.0;
    if(existeArchivo(nombre)==0)
    {
        archivo=fopen(nombre,"w");
        fprintf(archivo,"epsilon1,epsilon2,Spot[m],
            Iteraciones,\n");
        fprintf(archivo,"%.15Le,%.15Le,%.15Le,%d\n",
            epsilon1,epsilon2,spot,iterMax);
    }
    else
    {
        archivo=fopen(nombre,"a");
        fprintf(archivo,"%.15Le,%.15Le,%.15Le,%d\n",
            epsilon1,epsilon2,spot,iterMax);
    }
```

```
    fclose(archivo);
    free(nombre);
}
```

error_iteraciones.h

```
#ifndef ERROR_ITERACIONES_H_INCLUDED
#define ERROR_ITERACIONES_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void guardaVariaciones(long double * spots, int *
    iteraciones, int iterMax, long double epsilon1, long
    double epsilon2, char * tipo, char * subCarpeta,
    _Bool termino);
int existeArchivo(const char *nombre);
void guardaSpotIteraciones(long double spot, int iterMax,
     long double epsilon1, long double epsilon2, char *
    tipo, _Bool termino);


#endif // ERROR_ITERACIONES_H_INCLUDED
```

A.5.4. no_lineal.c y no_lineal.h

no_lineal.c

```
#include <stdlib.h>2
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include "matrices.h"
#include "lineal.h"
#include "error_iteraciones.h"

// Constantes para propagación en medio Kerr

    int pasosKerr=1000;


// Funciones para propagacion en rutina no lineal

// Nota: Las rutinas correran indefinidamente hasta que
    converjan.
```

```
long double complex prop_q(matriz * ABCD, long double
    complex q_in, long double n1, long double n2) //
    Propagación de un haz gaussiano a través de un sistema
     óptico ABCD   Siegman
{
    assert(ABCD);
    assert(q_in);
    assert(n1!=0&&n2!=0);
    long double complex q_out;
    q_out=n2*(obtieneElemento(ABCD,1,1)*q_in+n1*
        obtieneElemento(ABCD,1,2))/(obtieneElemento(ABCD
        ,2,1)*q_in+n1*obtieneElemento(ABCD,2,2));
    return q_out;
}


long double spot_q(long double complex q, long double n,
    long double lambda)
{
    long double w=sqrtl( lambda/(n*M_PI*cimagl(1/q)));
    return w;
}

long double radio_q(long double complex q)
{
    long double R=creal(1/creall(1/q));
    return R;
}

matriz * spot_q_matriz(matriz * q, long double n, long
    double lambda)
{
    matriz *w=nuevaMatriz(q->filas,q->columnas);
    long double complex q_ij;
    long double w_ij;
    for(int i=q->filas;i>0;i  )
        for(int j=q->columnas;j>0;j  )
            {
                q_ij=obtieneElemento(q,i,j);
                w_ij=creal(sqrtl( lambda/(n*M_PI*cimag(1/
                    q_ij))));
                if(isfinite(w_ij)==0)
                    w_ij=INFINITY;
                else
```

```
                fijaElemento(w,i,j,w_ij);
            }
    return w;
}


// Propagación de haz gaussiano en medio Kerr
   Ecuaciones desacopladas, método matricial.
// Original
long double complex propagacionKerr(int pasos, long
   double L, long double n0,long double n2, long double
   w_pump,long double complex q_in, long double chi, long
    double kth, long double Cp, long double rho, long
   double dn_dv,long double P_pump,long double P_laser,
   long double lambda0)
{
        assert(q_in);
        // Variables locales
        long double dz=L/pasos; // Grosor de lámina
        long double lambda1=lambda0/n0, w_in, parabola;
        long double complex *qProp = (long double complex
           *)calloc(pasos,sizeof(long double complex));
        long double complex A, B, C, D;
        assert(qProp);
        matriz * M_i;
        M_i=nuevaMatriz(2,2);
        int i=pasos 1;
        qProp[i]=q_in;

        for (i=pasos 1;i>0;i  )
        {
            w_in=sqrtl( lambda1/(M_PI*cimagl(1/qProp[i]))
                );
            // Factor de parábola: parábola=1/h^2
            //parabola=1/n0*dn_dv*(chi*P_pump/L)/(M_PI*
                powl(w_pump,2)*kth*Cp*rho)+1/n0*(8*n2*
                P_laser)/(M_PI*powl(w_in,4)); // Kerr y
                térmico
            //parabola=1/n0*(8*n2*P_laser)/(M_PI*powl(
                w_in,4)); // Sólo efecto Kerr  truncado
                por Taylor
            parabola=1/n0*(n2*P_laser)/(2.0*M_PI*powl(
                w_in,4)); // Sólo efecto Kerr  Ajuste
                cuadrático
            // Calculando parámetros de la matriz
            A=1.0;
```

```
            B=dz/n0;
            C= n0*parabola*dz;
            D=1.0;
            llenaMatrizSinCrear(M_i,A,B,C,D);
            qProp[i 1]=prop_q(M_i,qProp[i],n0,n0);
        }

        borraMatriz(M_i);
        long double complex regresar=qProp[0];
        assert(qProp!=NULL);
        free(qProp);
        qProp=NULL;
        return regresar;
}


long double complex * propagacionKerrMatriz(int pasos,
    long double L, long double n0,long double n2, long
    double w_pump,long double complex q_in, long double
    chi, long double kth, long double Cp, long double rho,
     long double dn_dv,long double P_pump,long double
    P_laser, long double lambda0)
{
        assert(q_in);
        // Variables locales
        long double dz=L/pasos; // Grosor de lámina
        long double lambda1=lambda0/n0, w_in, parabola;
        long double complex *qProp = (long double complex
            *)calloc(pasos,sizeof(long double complex));
        long double complex A, B, C, D;
        assert(qProp);
        matriz * M_i;
        M_i=nuevaMatriz(2,2);
        qProp[0]=q_in;
        int i=0;
        for (i=0;i<pasos 1;i++)
        {
            w_in=sqrtl( lambda1/(M_PI*cimag(1/ qProp[i]))
                );
            // Factor de parábola: parábola=1/h^2
            //parabola=1/n0*dn_dv*(chi*P_pump/L)/(M_PI*
                powl(w_pump,2)*kth*Cp*rho)+1/n0*(8*n2*
                P_laser)/(M_PI*powl(w_in,4)); // Kerr y
                térmico
            parabola=1/n0*(8.0*n2*P_laser)/(M_PI*powl(
                w_in,4.0)); // Sólo efecto Kerr
```

```
            // Calculando parámetros de la matriz
            A=1.0;
            B=dz/n0;
            C= n0*parabola*dz;
            D=1.0;
            llenaMatrizSinCrear(M_i,A,B,C,D);
            // Propagando el haz
            qProp[i+1]=prop_q(M_i,qProp[i],n0,n0);
        }
        borraMatriz(M_i);
        return qProp;
}

// Busca el valor menos astigmático en la matriz.
void confNoAstigmatica(matriz * A, long double complex *
    casiUno, int *fila, int *columna)
{
    *casiUno = obtieneElemento(A,1,1);
    *fila=1;
    *columna=1;
    for(int i=1;i<A>filas;++i)
    {
        for(int j=1;j<A>columnas;++j)
            {
                if(cabsl(obtieneElemento(A,i,j) 1)<cabsl
                    (*casiUno 1))
                {
                    *casiUno=obtieneElemento(A,i,j);
                    *fila=i;
                    *columna=j;
                }
            }
    }
    assert(obtieneElemento(A,*fila,*columna)==*casiUno);
}

// Propagación no lineal (matrices)
// Termina el lazo si alcanza el umbral solicitado
// Para epsilon1 y epsilon2

// EM1_tan

matriz * propNoLineal_tanEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
```

```
    w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM1Tan";
    char subCarpeta[]="EM1Tan_matrices";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4,q_5,q_6; //
        Parámetros complejos para propagación

    // Crea matrices para la propagación

    matriz *EM1tan1,*EM1tan2,*EM1tan3,*EM1tan4,*EM1tan5,*
        EM1tan6,*EM1tan7,*EM1tan8,*EM1tan9,*EM1tan10,*
        EM1tan11,*EM1tan12,*EM1tan13,*EM1tan14,*EM1tan15,*
        EM1tan16,*EM1tan17;
    // EM1, Caso tangencial
```

```
EM1tan1=llenaMatriz(1,L1,0,1);
EM1tan2=llenaMatriz(1,0, 1/f1t,1);
EM1tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1tan4=llenaMatriz(n0,0,0,1/n0);
EM1tan5=llenaMatriz(1/n0,0,0,n0);
EM1tan6=nuevaMatriz(2,2);//EM1tan5 pendiente
EM1tan7=llenaMatriz(1,0, 1/f2t,1);
EM1tan8=llenaMatriz(1,L2,0,1);
EM1tan9=llenaMatriz(1,0,0,1);
EM1tan10=llenaMatriz(1,L2,0,1);
EM1tan11=llenaMatriz(1,0, 1/f2t,1);
EM1tan12=nuevaMatriz(2,2);      //EM1tan11 pendiente
EM1tan13=llenaMatriz(n0,0,0,1/n0);
EM1tan14=llenaMatriz(1/n0,0,0,n0);
EM1tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1tan16=llenaMatriz(1,0, 1/f1t,1);
EM1tan17=llenaMatriz(1,L1,0,1);

// Ciclo
//matriz * q_iter = nuevaMatriz(1,iteraciones);
matriz * q_iter = nuevaMatriz(1,2);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM1tan3,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM1tan6,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1tan12,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1tan15,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);

        if(elem(q_in,index1,index2)==0)
```

```
        {elem(q_new,index1,index2)=0;
        escribeCSV=0;
        reallocFallido=0;}
    else
    {
        // Almacenamiento de spots e iteración
            usado en loop
        //long double *spots=(long douvalgrind
            realloc bytes lostble*)calloc(
            iteraciones+1,sizeof(long double));
        //int *iteracionActual=(int*)calloc(
            iteraciones+1,sizeof(int));

        fijaElemento(q_iter,1,1,obtieneElemento(
            q_in,index1,index2));
        escribeCSV=1;
        //for(int iter=1;iter<iteraciones;iter++)
        //for(int iter=iteraciones 1;iter>0;iter
            )
        for(;;)
        {
            //int ahora=iter;
            //int siguiente=iter+1;
            //int ahora=iteraciones iter;
            //int siguiente=ahora+1;
            int ahora=1;
            int siguiente=2;
            matriz * Prop1_piezas[]={EM1tan4,
                EM1tan3,EM1tan2,EM1tan1};
            matriz * Prop1=variasMultMatriciales(
                Prop1_piezas,4);
            long double complex q1_hold=
                obtieneElemento(q_iter,1,ahora);
            //q_2=prop_q(Prop1,obtieneElemento(
                q_iter,1,ahora),1,n0);
            q_2=prop_q(Prop1,q1_hold,1,n0);
            q_3=propagacionKerr(pasosKerr,L,n0,n2
                ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
                P_pump,P_laser,lambda0);
            matriz * Prop2_piezas[]={EM1tan13,
                EM1tan12,EM1tan11,EM1tan10,EM1tan9
                ,EM1tan8,EM1tan7,EM1tan6,EM1tan5};
                 // 5 a 13
            matriz * Prop2=variasMultMatriciales(
                Prop2_piezas,9);
            q_4=prop_q(Prop2,q_3,n0,n0);
```

```
q_5=propagacionKerr(pasosKerr,L,n0,n2
    ,w_pump,q_4,chi,kth,Cp,rho,dn_dv,
    P_pump,P_laser,lambda0);
q_5=propagacionKerr(pasosKerr,L,n0,n2
    ,w_pump,q_4,chi,kth,Cp,rho,dn_dv,
    P_pump,P_laser,lambda0);
matriz * Prop3_piezas[]={EM1tan17,
    EM1tan16,EM1tan15,EM1tan14}; // 14
     a 17
matriz * Prop3=variasMultMatriciales(
    Prop3_piezas,4);
q_6=prop_q(Prop3,q_5,n0,1);
fijaElemento(q_iter,1,ahora,q_6);
//printf("Actual: %f+i %f, siguiente:
     %f+i%f",creal(obtieneElemento(
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieaza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
Prop3=borraMatriz(Prop3);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_6,1,lambda0);
// Prepara vectores
if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_spt, *tmp_iter;
tmp_spt=(long double*)realloc(spots,
    cuenta*sizeof(long double));
tmp_iter=(int *)realloc(
    iteracionActual,cuenta*sizeof(int)
    );
if(tmp_spt!=NULL && tmp_iter!=NULL)
{
     spots=tmp_spt;
     iteracionActual=tmp_iter;
     spots[cuenta-1]=w_iter_nuevo;
     iteracionActual[cuenta-1]=cuenta
          -1;
```

```
                }
                else
                {
                    reallocFallido=1;
                    break;
                }
                if(w_iter_viejo>0.1||w_iter_nuevo
                    >0.1||isnan(w_iter_viejo) ||isnan(
                    w_iter_nuevo))
                {
                    termino=0;
                    escribeCSV=1;
                    fijaElemento(q_new,index1,index2
                        ,0);
                    break;
                } // Si el spot mide 20 cm es
                    demasiado
                diferencia=fabsl((w_iter_nuevo
                    w_iter_viejo)/w_iter_nuevo)*100.0;
                if(diferencia<=umbral) // Termina la
                    iteración
                {
                    //fijaElemento(q_new,index1,
                        index2,obtieneElemento(q_iter
                        ,1,siguiente));
                    fijaElemento(q_new,index1,index2,
                        obtieneElemento(q_iter,1,ahora
                        ));
                    termino=1;
                    break;
                }
                else
                {
                    fijaElemento(q_new,index1,index2
                        ,0);
                    termino=0;
                }

            }
        }
        if(reallocFallido==1)
        {
            termino=0;
            free(spots);
            free(iteracionActual);
            printf("Se acabó la memoria\n");
```

```
                reallocFallido=0;
            }
            guardaSpotIteraciones(spots[cuenta 1],cuenta,
                creall(obtieneElemento(epsilon1,1,index1))
                ,creall(obtieneElemento(epsilon2,1,index2)
                ),tipo,termino);
            if(escribeCSV==1)
                guardaVariaciones(spots,iteracionActual,
                    cuenta,creall(obtieneElemento(epsilon1
                    ,1,index1)),creall(obtieneElemento(
                    epsilon2,1,index2)),tipo,subCarpeta,
                    termino);
            if(termino==0)
                printf("Cavidad en X, EM1tan: No cumple
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo: %Le\n
                    ",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            else
                printf("Cavidad en X, EM1tan: Cumple con
                    el umbral de %Le para epsilon1=%Le,
                    epsilon2=%Le.\nError relativo: %Le\n",
                    umbral,creall(obtieneElemento(epsilon1
                    ,1,index1)),creall(obtieneElemento(
                    epsilon2,1,index2)),diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM1tan1);
    borraMatriz(EM1tan2);
    borraMatriz(EM1tan3);
    borraMatriz(EM1tan4);
    borraMatriz(EM1tan5);
    borraMatriz(EM1tan6);
    borraMatriz(EM1tan7);
    borraMatriz(EM1tan8);
    borraMatriz(EM1tan9);
    borraMatriz(EM1tan10);
    borraMatriz(EM1tan11);
    borraMatriz(EM1tan12);
    borraMatriz(EM1tan13);
    borraMatriz(EM1tan14);
```

```
    borraMatriz(EM1tan15);
    borraMatriz(EM1tan16);
    borraMatriz(EM1tan17);
    return q_new;
}

// EM1 sagital

matriz * propNoLineal_sagEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM1Sag";
    char subCarpeta[]="EM1Sag_matrices";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);
```

```
matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4,q_5,q_6; //
    Parámetros complejos para propagación

// Crea matrices para la propagación

matriz *EM1sag1,*EM1sag2,*EM1sag3,*EM1sag4,*EM1sag5,*
    EM1sag6,*EM1sag7,*EM1sag8,*EM1sag9,*EM1sag10,*
    EM1sag11,*EM1sag12,*EM1sag13,*EM1sag14,*EM1sag15,*
    EM1sag16,*EM1sag17;
// EM1, Caso saggencial
EM1sag1=llenaMatriz(1,L1,0,1);
EM1sag2=llenaMatriz(1,0, 1/f1s,1);
EM1sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1sag4=llenaMatriz(1,0,0,1);
EM1sag5=llenaMatriz(1,0,0,1);
EM1sag6=nuevaMatriz(2,2);//EM1sag5 pendiente
EM1sag7=llenaMatriz(1,0, 1/f2s,1);
EM1sag8=llenaMatriz(1,L2,0,1);
EM1sag9=llenaMatriz(1,0,0,1);
EM1sag10=llenaMatriz(1,L2,0,1);
EM1sag11=llenaMatriz(1,0, 1/f2s,1);
EM1sag12=nuevaMatriz(2,2);     //EM1sag11 pendiente
EM1sag13=llenaMatriz(1,0,0,1);
EM1sag14=llenaMatriz(1,0,0,1);
EM1sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1sag16=llenaMatriz(1,0, 1/f1s,1);
EM1sag17=llenaMatriz(1,L1,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1->columnas;index1++)
{
    for(int index2=1;index2<=epsilon2->columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
```

```
// Llenando matrices variables:
llenaMatrizSinCrear(EM1sag3,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1sag6,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag12,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag15,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);

if(elem(q_in,index1,index2)==0)
     {elem(q_new,index1,index2)=0;
     escribeCSV=0;
     reallocFallido=0;}
else
{
     fijaElemento(q_iter,1,1,obtieneElemento(
         q_in,index1,index2));
     escribeCSV=1;
     //for(int iter=1;iter<iteraciones;iter++)
     for(;;)
     {
          int ahora=1;
          int siguiente=2;
          matriz * Prop1_piezas[]={EM1sag4,
              EM1sag3,EM1sag2,EM1sag1};
          matriz * Prop1=variasMultMatriciales(
              Prop1_piezas,4);
          long double complex q1_hold=
              obtieneElemento(q_iter,1,ahora);
          //q_2=prop_q(Prop1,obtieneElemento(
              q_iter,1,ahora),1,n0);
          q_2=prop_q(Prop1,q1_hold,1,n0);
          q_3=propagacionKerr(pasosKerr,L,n0,n2
              ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
              P_pump,P_laser,lambda0);
          matriz * Prop2_piezas[]={EM1sag13,
              EM1sag12,EM1sag11,EM1sag10,EM1sag9
              ,EM1sag8,EM1sag7,EM1sag6,EM1sag5};
               // 5 a 13
          matriz * Prop2=variasMultMatriciales(
              Prop2_piezas,9);
          q_4=prop_q(Prop2,q_3,n0,n0);
          q_5=propagacionKerr(pasosKerr,L,n0,n2
              ,w_pump,q_4,chi,kth,Cp,rho,dn_dv,
              P_pump,P_laser,lambda0);
```

```
matriz * Prop3_piezas[]={EM1sag17,
    EM1sag16,EM1sag15,EM1sag14}; // 14
     a 17
matriz * Prop3=variasMultMatriciales(
    Prop3_piezas,4);
q_6=prop_q(Prop3,q_5,n0,1);
fijaElemento(q_iter,1,ahora,q_6);
//printf("Actual: %f+i %f, siguiente:
     %f+i%f",creal(obtieneElemento(
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieaza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
Prop3=borraMatriz(Prop3);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_6,1,lambda0);//
     Prepara vectores
if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_spt, *tmp_iter;
tmp_spt=(long double*)realloc(spots,
    cuenta*sizeof(long double));
tmp_iter=(int *)realloc(
    iteracionActual,cuenta*sizeof(int)
    );
if(tmp_spt!=NULL && tmp_iter!=NULL)
{
     spots=tmp_spt;
     iteracionActual=tmp_iter;
     spots[cuenta-1]=w_iter_nuevo;
     iteracionActual[cuenta-1]=cuenta
         -1;
}
else
{
     reallocFallido=1;
     break;
}
```

```
                if(w_iter_viejo>0.1||w_iter_nuevo
                    >0.1||isnan(w_iter_viejo) ||isnan(
                    w_iter_nuevo))
                {
                    termino=0;
                    escribeCSV=1;
                    fijaElemento(q_new,index1,index2
                        ,0);
                    break;
                }
                diferencia=fabsl((w_iter_nuevo
                    w_iter_viejo)/w_iter_nuevo)*100.0;
                if(diferencia<=umbral) // Termina la
                    iteración
                {
                    fijaElemento(q_new,index1,index2,
                        obtieneElemento(q_iter,1,ahora
                        ));
                    termino=1;
                    break;
                }
                else
                {
                    fijaElemento(q_new,index1,index2
                        ,0);
                    termino=0;
                }
            }
        }
        if(reallocFallido==1)
        {
            termino=0;
            free(spots);
            free(iteracionActual);
            printf("Se acabó la memoria\n");
            reallocFallido=0;
        }
        guardaSpotIteraciones(spots[cuenta 1],cuenta,
            creall(obtieneElemento(epsilon1,1,index1))
            ,creall(obtieneElemento(epsilon2,1,index2)
            ),tipo,termino);
        if(escribeCSV==1)
            guardaVariaciones(spots,iteracionActual,
                cuenta,creall(obtieneElemento(epsilon1
                ,1,index1)),creall(obtieneElemento(
                epsilon2,1,index2)),tipo,subCarpeta,
```

```
                termino);
            if(termino==0)
                printf("Cavidad en X, EM1sag: No cumple
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo: %Le\n
                    ",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            else
                printf("Cavidad en X, EM1sag: Cumple con
                    el umbral de %Le para epsilon1=%Le,
                    epsilon2=%Le.\nError relativo: %Le\n",
                    umbral,creall(obtieneElemento(epsilon1
                    ,1,index1)),creall(obtieneElemento(
                    epsilon2,1,index2)),diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM1sag1);
    borraMatriz(EM1sag2);
    borraMatriz(EM1sag3);
    borraMatriz(EM1sag4);
    borraMatriz(EM1sag5);
    borraMatriz(EM1sag6);
    borraMatriz(EM1sag7);
    borraMatriz(EM1sag8);
    borraMatriz(EM1sag9);
    borraMatriz(EM1sag10);
    borraMatriz(EM1sag11);
    borraMatriz(EM1sag12);
    borraMatriz(EM1sag13);
    borraMatriz(EM1sag14);
    borraMatriz(EM1sag15);
    borraMatriz(EM1sag16);
    borraMatriz(EM1sag17);
    return q_new;
}

// EM2 tangencial
matriz * propNoLineal_tanEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
```

```
    w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM2Tan";
    char subCarpeta[]="EM2Tan_matrices";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4,q_5,q_6; //
        Parámetros complejos para propagación

    // Crea matrices para la propagación

    matriz *EM2tan1,*EM2tan2,*EM2tan3,*EM2tan4,*EM2tan5,*
        EM2tan6,*EM2tan7,*EM2tan8,*EM2tan9,*EM2tan10,*
        EM2tan11,*EM2tan12,*EM2tan13,*EM2tan14,*EM2tan15,*
        EM2tan16,*EM2tan17;
    // EM2, Caso tangencial
```

```
EM2tan1=llenaMatriz(1,L2,0,1);
EM2tan2=llenaMatriz(1,0, 1/f2t,1);
EM2tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2tan4=llenaMatriz(n0,0,0,1/n0);
EM2tan5=llenaMatriz(1/n0,0,0,n0);
EM2tan6=nuevaMatriz(2,2);//EM2tan5 pendiente
EM2tan7=llenaMatriz(1,0, 1/f1t,1);
EM2tan8=llenaMatriz(1,L1,0,1);
EM2tan9=llenaMatriz(1,0,0,1);
EM2tan10=llenaMatriz(1,L1,0,1);
EM2tan11=llenaMatriz(1,0, 1/f1t,1);
EM2tan12=nuevaMatriz(2,2);      //EM2tan11 pendiente
EM2tan13=llenaMatriz(n0,0,0,1/n0);
EM2tan14=llenaMatriz(1/n0,0,0,n0);
EM2tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2tan16=llenaMatriz(1,0, 1/f2t,1);
EM2tan17=llenaMatriz(1,L2,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM2tan3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2tan6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);

        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;
```

```
        escribeCSV=0;
        reallocFallido=0;
        }
    else
    {
        fijaElemento(q_iter,1,1,obtieneElemento(
            q_in,index1,index2));
        escribeCSV=1;
        //for(int iter=1;iter<iteraciones;iter++)
       for(;;)
        {
            int ahora=1;
            int siguiente=2;
            matriz * Prop1_piezas[]={EM2tan4,
                EM2tan3,EM2tan2,EM2tan1};
            matriz * Prop1=variasMultMatriciales(
                Prop1_piezas,4);
            long double complex q1_hold=
                obtieneElemento(q_iter,1,ahora);
            //q_2=prop_q(Prop1,obtieneElemento(
                q_iter,1,ahora),1,n0);
            q_2=prop_q(Prop1,q1_hold,1,n0);
            q_3=propagacionKerr(pasosKerr,L,n0,n2
                ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
                P_pump,P_laser,lambda0);
            matriz * Prop2_piezas[]={EM2tan13,
                EM2tan12,EM2tan11,EM2tan10,EM2tan9
                ,EM2tan8,EM2tan7,EM2tan6,EM2tan5};
                 // 5 a 13
            matriz * Prop2=variasMultMatriciales(
                Prop2_piezas,9);
            q_4=prop_q(Prop2,q_3,n0,n0);
            q_5=propagacionKerr(pasosKerr,L,n0,n2
                ,w_pump,q_4,chi,kth,Cp,rho,dn_dv,
                P_pump,P_laser,lambda0);
            matriz * Prop3_piezas[]={EM2tan17,
                EM2tan16,EM2tan15,EM2tan14}; // 14
                 a 17
            matriz * Prop3=variasMultMatriciales(
                Prop3_piezas,4);
            q_6=prop_q(Prop3,q_5,n0,1);
            fijaElemento(q_iter,1,ahora,q_6);
            //printf("Actual: %f+i %f, siguiente:
                 %f+i %f",creal(obtieneElemento(
                q_iter,1,iter)),cimag(
                obtieneElemento(q_iter,1,iter)),
```

```
        creal(obtieneElemento(q_iter,1,
        iter+1)),cimag(obtieneElemento(
        q_iter,1,iter+1)));
    // Limpieaza
    Prop1=borraMatriz(Prop1);
    Prop2=borraMatriz(Prop2);
    Prop3=borraMatriz(Prop3);
    // Calculando spots y comparación
    w_iter_viejo=spot_q(q1_hold,1,lambda0
        );
    w_iter_nuevo=spot_q(q_6,1,lambda0);//
         Prepara vectores
    if(cuenta>iteraciones)  break; // Por
         si acaso
    ++cuenta;
    //printf("%i\n",cuenta);
    void *tmp_spt, *tmp_iter;
    tmp_spt=(long double*)realloc(spots,
        cuenta*sizeof(long double));
    tmp_iter=(int *)realloc(
        iteracionActual,cuenta*sizeof(int)
        );
    if(tmp_spt!=NULL && tmp_iter!=NULL)
    {
         spots=tmp_spt;
         iteracionActual=tmp_iter;
         spots[cuenta-1]=w_iter_nuevo;
         iteracionActual[cuenta-1]=cuenta
             -1;
    }
    else
    {
         reallocFallido=1;
         break;
    }
    if(w_iter_viejo>0.1||w_iter_nuevo
        >0.1||isnan(w_iter_viejo) ||isnan(
        w_iter_nuevo))
    {
         termino=0;
         escribeCSV=1;
         fijaElemento(q_new,index1,index2
             ,0);
         break;
    }  // Si el spot mide 20 cm es
        demasiado
```

```
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo)*100.0;
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,ahora
                    ));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spots);
        free(iteracionActual);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spots[cuenta 1],cuenta,
        creall(obtieneElemento(epsilon1,1,index1))
        ,creall(obtieneElemento(epsilon2,1,index2)
        ),tipo,termino);
    if(escribeCSV==1)
        guardaVariaciones(spots,iteracionActual,
            cuenta,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),tipo,subCarpeta,
            termino);
    if(termino==0)
        printf("Cavidad en X, EM2tan: No cumple
            con el umbral de %Le para epsilon1=%Le
            , epsilon2=%Le.\nError relativo: %Le\n
            ",umbral,creall(obtieneElemento(
            epsilon1,1,index1)),creall(
            obtieneElemento(epsilon2,1,index2)),
            diferencia);
    else
```

```
                printf("Cavidad en X, EM2tan: Cumple con
                    el umbral de %Le para epsilon1=%Le,
                    epsilon2=%Le.\nError relativo: %Le\n",
                    umbral,creall(obtieneElemento(epsilon1
                    ,1,index1)),creall(obtieneElemento(
                    epsilon2,1,index2)),diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM2tan1);
    borraMatriz(EM2tan2);
    borraMatriz(EM2tan3);
    borraMatriz(EM2tan4);
    borraMatriz(EM2tan5);
    borraMatriz(EM2tan6);
    borraMatriz(EM2tan7);
    borraMatriz(EM2tan8);
    borraMatriz(EM2tan9);
    borraMatriz(EM2tan10);
    borraMatriz(EM2tan11);
    borraMatriz(EM2tan12);
    borraMatriz(EM2tan13);
    borraMatriz(EM2tan14);
    borraMatriz(EM2tan15);
    borraMatriz(EM2tan16);
    borraMatriz(EM2tan17);
    return q_new;
}

matriz * propNoLineal_sagEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
```

```
_Bool reallocFallido=0;
long double *spots;
int * iteracionActual;
int cuenta=0;
char tipo[]="EM2Sag";
char subCarpeta[]="EM2Sag_matrices";
long double w_iter_viejo=0, w_iter_nuevo=0,
    diferencia=0;
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4,q_5,q_6; //
    Parámetros complejos para propagación

// Crea matrices para la propagación

matriz *EM2sag1,*EM2sag2,*EM2sag3,*EM2sag4,*EM2sag5,*
    EM2sag6,*EM2sag7,*EM2sag8,*EM2sag9,*EM2sag10,*
    EM2sag11,*EM2sag12,*EM2sag13,*EM2sag14,*EM2sag15,*
    EM2sag16,*EM2sag17;
// EM2, Caso saggencial
EM2sag1=llenaMatriz(1,L2,0,1);
EM2sag2=llenaMatriz(1,0, 1/f2s,1);
EM2sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2sag4=llenaMatriz(1,0,0,1);
EM2sag5=llenaMatriz(1,0,0,1);
EM2sag6=nuevaMatriz(2,2);//EM2sag5 pendiente
EM2sag7=llenaMatriz(1,0, 1/f1s,1);
EM2sag8=llenaMatriz(1,L1,0,1);
EM2sag9=llenaMatriz(1,0,0,1);
EM2sag10=llenaMatriz(1,L1,0,1);
```

```
EM2sag11=llenaMatriz(1,0, 1/f1s,1);
EM2sag12=nuevaMatriz(2,2);      //EM2sag11 pendiente
EM2sag13=llenaMatriz(1,0,0,1);
EM2sag14=llenaMatriz(1,0,0,1);
EM2sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2sag16=llenaMatriz(1,0, 1/f2s,1);
EM2sag17=llenaMatriz(1,L2,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM2sag3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2sag6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);


        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;
            escribeCSV=0;
            reallocFallido=0;
            }
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            escribeCSV=1;
            //for(int iter=1;iter<iteraciones;iter++)
            for(;;)
```

```
{
    int ahora=1;
    int siguiente=2;

    matriz * Prop1_piezas[]={EM2sag4,
        EM2sag3,EM2sag2,EM2sag1};
    matriz * Prop1=variasMultMatriciales(
        Prop1_piezas,4);
    long double complex q1_hold=
        obtieneElemento(q_iter,1,ahora);
    //q_2=prop_q(Prop1,obtieneElemento(
        q_iter,1,ahora),1,n0);
    q_2=prop_q(Prop1,q1_hold,1,n0);
    q_3=propagacionKerr(pasosKerr,L,n0,n2
        ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
        P_pump,P_laser,lambda0);
    matriz * Prop2_piezas[]={EM2sag13,
        EM2sag12,EM2sag11,EM2sag10,EM2sag9
        ,EM2sag8,EM2sag7,EM2sag6,EM2sag5};
         // 5 a 13
    matriz * Prop2=variasMultMatriciales(
        Prop2_piezas,9);
    q_4=prop_q(Prop2,q_3,n0,n0);
    q_5=propagacionKerr(pasosKerr,L,n0,n2
        ,w_pump,q_4,chi,kth,Cp,rho,dn_dv,
        P_pump,P_laser,lambda0);
    matriz * Prop3_piezas[]={EM2sag17,
        EM2sag16,EM2sag15,EM2sag14}; // 14
         a 17
    matriz * Prop3=variasMultMatriciales(
        Prop3_piezas,4);
    q_6=prop_q(Prop3,q_5,n0,1);
    fijaElemento(q_iter,1,ahora,q_6);
    //printf("Actual: %f+i %f, siguiente:
         %f+i%f",creal(obtieneElemento(
        q_iter,1,iter)),cimag(
        obtieneElemento(q_iter,1,iter)),
        creal(obtieneElemento(q_iter,1,
        iter+1)),cimag(obtieneElemento(
        q_iter,1,iter+1)));
    // Limpieaza
    Prop1=borraMatriz(Prop1);
    Prop2=borraMatriz(Prop2);
    Prop3=borraMatriz(Prop3);
    // Calculando spots y comparación
    w_iter_viejo=spot_q(q1_hold,1,lambda0
```

```
    );
w_iter_nuevo=spot_q(q_6,1,lambda0);//
    Prepara vectores
if(cuenta>iteraciones) break; // Por
    si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_spt, *tmp_iter;
tmp_spt=(long double*)realloc(spots,
    cuenta*sizeof(long double));
tmp_iter=(int *)realloc(
    iteracionActual,cuenta*sizeof(int)
    );
if(tmp_spt!=NULL && tmp_iter!=NULL)
{
    spots=tmp_spt;
    iteracionActual=tmp_iter;
    spots[cuenta-1]=w_iter_nuevo;
    iteracionActual[cuenta-1]=cuenta
        -1;
}
else
{
    reallocFallido=1;
    break;
}
if(w_iter_viejo>0.1||w_iter_nuevo
    >0.1||isnan(w_iter_viejo) ||isnan(
    w_iter_nuevo))
{
    termino=0;
    escribeCSV=1;
    fijaElemento(q_new,index1,index2
        ,0);
    break;
} // Si el spot mide 20 cm es
    demasiado
diferencia=fabsl((w_iter_nuevo-
    w_iter_viejo)/w_iter_nuevo)*100.0;
diferencia=fabsl((w_iter_nuevo-
    w_iter_viejo)/w_iter_nuevo)*100.0;
if(diferencia<=umbral) // Termina la
    iteración
{
    fijaElemento(q_new,index1,index2,
        obtieneElemento(q_iter,1,ahora
```

```
                    ));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spots);
        free(iteracionActual);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spots[cuenta 1],cuenta,
        creall(obtieneElemento(epsilon1,1,index1))
        ,creall(obtieneElemento(epsilon2,1,index2)
        ),tipo,termino);
     if(escribeCSV==1)
        guardaVariaciones(spots,iteracionActual,
            cuenta,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),tipo,subCarpeta,
            termino);
    if(termino==0)
        printf("Cavidad en X, EM2sag: No cumple
            con el umbral de %Le para epsilon1=%Le
            , epsilon2=%Le.\nError relativo: %Le\n
            ",umbral,creall(obtieneElemento(
            epsilon1,1,index1)),creall(
            obtieneElemento(epsilon2,1,index2)),
            diferencia);
    else
        printf("Cavidad en X, EM2sag: Cumple con
            el umbral de %Le para epsilon1=%Le,
            epsilon2=%Le.\nError relativo: %Le\n",
            umbral,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),diferencia);
    free(spots);
```

```
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM2sag1);
    borraMatriz(EM2sag2);
    borraMatriz(EM2sag3);
    borraMatriz(EM2sag4);
    borraMatriz(EM2sag5);
    borraMatriz(EM2sag6);
    borraMatriz(EM2sag7);
    borraMatriz(EM2sag8);
    borraMatriz(EM2sag9);
    borraMatriz(EM2sag10);
    borraMatriz(EM2sag11);
    borraMatriz(EM2sag12);
    borraMatriz(EM2sag13);
    borraMatriz(EM2sag14);
    borraMatriz(EM2sag15);
    borraMatriz(EM2sag16);
    borraMatriz(EM2sag17);
    return q_new;
}

matriz * propNoLineal_AnilloRuta1Tan(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    int pasosKerr=1000;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
```

```
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in>filas,q_in>columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4; // Parámetros
    complejos para propagación

// Crea matrices para la propagación
matriz *Ruta1tan1,*Ruta1tan2,*Ruta1tan3,*Ruta1tan4,*
    Ruta1tan5,*Ruta1tan6,*Ruta1tan7,*Ruta1tan8;
// Ruta 1, caso tangencial
Ruta1tan1=llenaMatriz(1,L1,0,1); // L1=a
Ruta1tan2=llenaMatriz(1,0,1/f1t,1);
Ruta1tan3=nuevaMatriz(2,2); // Pendiente
Ruta1tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
    entrada)
Ruta1tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
    salida)
Ruta1tan6=nuevaMatriz(2,2);//Ruta1tan6 pendiente
Ruta1tan7=llenaMatriz(1,0,1/f2t,1);
Ruta1tan8=llenaMatriz(1,L2,0,1); // L2=b+c

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1>columnas;index1++)
{
    for(int index2=1;index2<=epsilon2>columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta1tan3,1,delta1+creall
            (obtieneElemento(epsilon1,1,index1)),0,1);
                llenaMatrizSinCrear(Ruta1tan6,1,
                    delta2+creall(obtieneElemento(
                    epsilon2,1,index2)),0,1);;
```

```
if(elem(q_in,index1,index2)==0)
    {elem(q_new,index1,index2)=0;}
else
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        q_in,index1,index2));
    //for(int iter=1;iter<iteraciones;iter++)
    for(int iter=iteraciones 1;iter>0;iter  )
    {
         //int ahora=iter;
        //int siguiente=iter+1;
        int ahora=iteraciones iter;
        int siguiente=ahora+1;
        matriz * Prop1_piezas[]={Ruta1tan4,
            Ruta1tan3,Ruta1tan2,Ruta1tan1}; //
            FM1 a refracción en medio Kerr
        matriz * Prop1=variasMultMatriciales(
            Prop1_piezas,4);
        long double complex q1_hold=
            obtieneElemento(q_iter,1,ahora);
        q_2=prop_q(Prop1,q1_hold,1,n0);
        q_3=propagacionKerr(pasosKerr,L,n0,n2
            ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
            P_pump,P_laser,lambda0);
        matriz * Prop2_piezas[]={Ruta1tan8,
            Ruta1tan7,Ruta1tan6,Ruta1tan5}; //
             5 a 8
        matriz * Prop2=variasMultMatriciales(
            Prop2_piezas,4);
        q_4=prop_q(Prop2,q_3,n0,1.0);
        fijaElemento(q_iter,1,siguiente,q_4);
        //printf("Actual: %f+i %f, siguiente:
             %f+i %f",creal(obtieneElemento(
            q_iter,1,iter)),cimag(
            obtieneElemento(q_iter,1,iter)),
            creal(obtieneElemento(q_iter,1,
            iter+1)),cimag(obtieneElemento(
            q_iter,1,iter+1)));
        // Limpieza
        Prop1=borraMatriz(Prop1);
        Prop2=borraMatriz(Prop2);
        // Calculando spots y comparación
        w_iter_viejo=spot_q(q1_hold,1,lambda0
            );
        w_iter_nuevo=spot_q(q_4,1,lambda0);
        if(w_iter_viejo>0.2||w_iter_nuevo
```

```
                    >0.2) break; // Si el spot mide
                        40 cm es demasiado
                    diferencia=fabsl((w_iter_nuevo
                        w_iter_viejo)/w_iter_nuevo)*100.0;
                    if(diferencia<=umbral) // Termina la
                        iteración
                    {
                        fijaElemento(q_new,index1,index2,
                            obtieneElemento(q_iter,1,
                            siguiente));
                        termino=1;
                        break;
                    }
                    else
                    {
                        fijaElemento(q_new,index1,index2
                            ,0);
                        termino=0;
                    }
                }
            }
            if(termino==0)
                printf("Cavidad en anillo, Ruta1Tan: No
                    cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
            else
                printf("Cavidad en anillo, Ruta1Tan:
                    Cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
        }
    }
    q_iter=borraMatriz(q_iter);
    Ruta1tan1=borraMatriz(Ruta1tan1);
    Ruta1tan2=borraMatriz(Ruta1tan2);
    Ruta1tan3=borraMatriz(Ruta1tan3);
    Ruta1tan4=borraMatriz(Ruta1tan4);
    Ruta1tan5=borraMatriz(Ruta1tan5);
    Ruta1tan6=borraMatriz(Ruta1tan6);
```

```
    Ruta1tan7=borraMatriz(Ruta1tan7);
    Ruta1tan8=borraMatriz(Ruta1tan8);
    return q_new;
}

matriz * propNoLineal_AnilloRuta1Sag(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    int pasosKerr=1000;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4; // Parámetros
        complejos para propagación

    // Crea matrices para la propagación
```

```
matriz *Ruta1sag1,*Ruta1sag2,*Ruta1sag3,*Ruta1sag4,*
    Ruta1sag5,*Ruta1sag6,*Ruta1sag7,*Ruta1sag8;
// Ruta 1, caso sagital
Ruta1sag1=llenaMatriz(1,L1,0,1); // L1=a
Ruta1sag2=llenaMatriz(1,0,1/f1s,1);
Ruta1sag3=nuevaMatriz(2,2); // Pendiente
Ruta1sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
    entrada)
Ruta1sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
    )
Ruta1sag6=nuevaMatriz(2,2);//Ruta1sag6 pendiente
Ruta1sag7=llenaMatriz(1,0,1/f2s,1);
Ruta1sag8=llenaMatriz(1,L2,0,1); // L2=b+c

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1>columnas;index1++)
{
    for(int index2=1;index2<=epsilon2>columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta1sag3,1,delta1+creall
            (obtieneElemento(epsilon1,1,index1)),0,1);
                llenaMatrizSinCrear(Ruta1sag6,1,
                    delta2+creall(obtieneElemento(
                    epsilon2,1,index2)),0,1);;
        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;}
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            //for(int iter=1;iter<iteraciones;iter++)
            for(int iter=iteraciones 1;iter>0;iter  )
            {
                 //int ahora=iter;
                //int siguiente=iter+1;
                int ahora=iteraciones iter;
                int siguiente=ahora+1;
                matriz * Prop1_piezas[]={Ruta1sag4,
                    Ruta1sag3,Ruta1sag2,Ruta1sag1}; //
                    FM1 a refracción en medio Kerr
                matriz * Prop1=variasMultMatriciales(
                    Prop1_piezas,4);
                long double complex q1_hold=
```

```
    obtieneElemento(q_iter,1,ahora);
q_2=prop_q(Prop1,q1_hold,1,n0);
q_3=propagacionKerr(pasosKerr,L,n0,n2
    ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
    P_pump,P_laser,lambda0);
matriz * Prop2_piezas[]={Ruta1sag8,
    Ruta1sag7,Ruta1sag6,Ruta1sag5}; //
     5 a 8
matriz * Prop2=variasMultMatriciales(
    Prop2_piezas,4);
q_4=prop_q(Prop2,q_3,n0,1);
fijaElemento(q_iter,1,siguiente,q_4);
//printf("Actual: %f+i %f, siguiente:
     %f+i%f",creal(obtieneElemento(
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_4,1,lambda0);
if(w_iter_viejo>0.2||w_iter_nuevo
    >0.2) break;  // Si el spot mide
    40 cm es demasiado
diferencia=fabsl((w_iter_nuevo
    w_iter_viejo)/w_iter_nuevo)*100.0;
if(diferencia<=umbral) // Termina la
    iteración
{
     //fijaElemento(q_new,index1,
         index2,obtieneElemento(q_iter
         ,1,iter+1));
     fijaElemento(q_new,index1,index2,
         obtieneElemento(q_iter,1,
         siguiente));
     termino=1;
     break;
}
else
{
     fijaElemento(q_new,index1,index2
```

```
                                ,0);
                            termino=0;
                        }
                    }
                }
                if(termino==0)
                    printf("Cavidad en anillo, Ruta1Sag: No
                        cumple con el umbral de %Le para
                        epsilon1=%Le, epsilon2=%Le.\nError
                        relativo: %Le\n",umbral,creall(
                        obtieneElemento(epsilon1,1,index1)),
                        creall(obtieneElemento(epsilon2,1,
                        index2)),diferencia);
                else
                    printf("Cavidad en anillo, Ruta1Sag:
                        Cumple con el umbral de %Le para
                        epsilon1=%Le, epsilon2=%Le.\nError
                        relativo: %Le\n",umbral,creall(
                        obtieneElemento(epsilon1,1,index1)),
                        creall(obtieneElemento(epsilon2,1,
                        index2)),diferencia);
            }
        }
        q_iter=borraMatriz(q_iter);
        Ruta1sag1=borraMatriz(Ruta1sag1);
        Ruta1sag2=borraMatriz(Ruta1sag2);
        Ruta1sag3=borraMatriz(Ruta1sag3);
        Ruta1sag4=borraMatriz(Ruta1sag4);
        Ruta1sag5=borraMatriz(Ruta1sag5);
        Ruta1sag6=borraMatriz(Ruta1sag6);
        Ruta1sag7=borraMatriz(Ruta1sag7);
        Ruta1sag8=borraMatriz(Ruta1sag8);
        return q_new;
}

matriz * propNoLineal_AnilloRuta2Tan(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
```

```
//imprimeMatriz(q_in);
long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
_Bool termino=0;
long double w_iter_viejo=0, w_iter_nuevo=0,
    diferencia=0;
long double L1=a;
long double L2=b+c;
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
assert(strcmp(conjugado_corto,conjugado_largo)==0);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4; // Parámetros
    complejos para propagación

// Crea matrices para la propagación
matriz *Ruta2tan1,*Ruta2tan2,*Ruta2tan3,*Ruta2tan4,*
    Ruta2tan5,*Ruta2tan6,*Ruta2tan7,*Ruta2tan8;
// Ruta 2, caso tangencial
Ruta2tan1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2tan2=llenaMatriz(1,0,1/f2t,1);
Ruta2tan3=nuevaMatriz(2,2); // Pendiente
Ruta2tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
    entrada)
Ruta2tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
    salida)
Ruta2tan6=nuevaMatriz(2,2);//Ruta2tan6 pendiente
Ruta2tan7=llenaMatriz(1,0,1/f1t,1);
Ruta2tan8=llenaMatriz(1,L1,0,1); // L1=a

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1->columnas;index1++)
```

```
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta2tan3,1,delta2+creall
            (obtieneElemento(epsilon2,1,index2)),0,1);
                    llenaMatrizSinCrear(Ruta2tan6,1,
                        delta1+creall(obtieneElemento(
                        epsilon1,1,index1)),0,1);;
        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;}
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            //for(int iter=1;iter<iteraciones;iter++)
            for(int iter=iteraciones 1;iter>0;iter  )
            {
                 //int ahora=iter;
                //int siguiente=iter+1;
                int ahora=iteraciones iter;
                int siguiente=ahora+1;
                matriz * Prop1_piezas[]={Ruta2tan4,
                    Ruta2tan3,Ruta2tan2,Ruta2tan1}; //
                    FM1 a refracción en medio Kerr
                matriz * Prop1=variasMultMatriciales(
                    Prop1_piezas,4);
                long double complex q1_hold=
                    obtieneElemento(q_iter,1,ahora);
                q_2=prop_q(Prop1,q1_hold,1,n0);
                q_3=propagacionKerr(pasosKerr,L,n0,n2
                    ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
                    P_pump,P_laser,lambda0);
                matriz * Prop2_piezas[]={Ruta2tan8,
                    Ruta2tan7,Ruta2tan6,Ruta2tan5}; //
                     5 a 8
                matriz * Prop2=variasMultMatriciales(
                    Prop2_piezas,4);
                q_4=prop_q(Prop2,q_3,n0,1);
                fijaElemento(q_iter,1,siguiente,q_4);
                //printf("Actual: %f+i %f, siguiente:
                     %f+i %f",creal(obtieneElemento(
                    q_iter,1,iter)),cimag(
                    obtieneElemento(q_iter,1,iter)),
                    creal(obtieneElemento(q_iter,1,
```

```
                iter+1)),cimag(obtieneElemento(
                q_iter,1,iter+1)));
            // Limpieza
            Prop1=borraMatriz(Prop1);
            Prop2=borraMatriz(Prop2);
            // Calculando spots y comparación
            w_iter_viejo=spot_q(q1_hold,1,lambda0
                );
            w_iter_nuevo=spot_q(q_4,1,lambda0);
            if(w_iter_viejo>0.2||w_iter_nuevo
                >0.2) break;  // Si el spot mide
                40 cm es demasiado
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo)*100.0;
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,
                    siguiente));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(termino==0)
        printf("Cavidad en anillo, Ruta2Tan: No
            cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),diferencia);
    else
        printf("Cavidad en anillo, Ruta2Tan:
            Cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
```

```
                    index2)),diferencia);
            }
        }
        q_iter=borraMatriz(q_iter);
        Ruta2tan1=borraMatriz(Ruta2tan1);
        Ruta2tan2=borraMatriz(Ruta2tan2);
        Ruta2tan3=borraMatriz(Ruta2tan3);
        Ruta2tan4=borraMatriz(Ruta2tan4);
        Ruta2tan5=borraMatriz(Ruta2tan5);
        Ruta2tan6=borraMatriz(Ruta2tan6);
        Ruta2tan7=borraMatriz(Ruta2tan7);
        Ruta2tan8=borraMatriz(Ruta2tan8);
        return q_new;
}

matriz * propNoLineal_AnilloRuta2Sag(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
```

```
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4; // Parámetros
    complejos para propagación

// Crea matrices para la propagación
matriz *Ruta2sag1,*Ruta2sag2,*Ruta2sag3,*Ruta2sag4,*
    Ruta2sag5,*Ruta2sag6,*Ruta2sag7,*Ruta2sag8;
// Ruta 2, caso sagital
Ruta2sag1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2sag2=llenaMatriz(1,0, 1/f2s,1);
Ruta2sag3=nuevaMatriz(2,2); // Pendiente
Ruta2sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
    entrada)
Ruta2sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
    )
Ruta2sag6=nuevaMatriz(2,2);//Ruta2sag6 pendiente
Ruta2sag7=llenaMatriz(1,0, 1/f1s,1);
Ruta2sag8=llenaMatriz(1,L1,0,1); // L1=a

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1->columnas;index1++)
{
    for(int index2=1;index2<=epsilon2->columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta2sag3,1,delta2+creall
            (obtieneElemento(epsilon2,1,index2)),0,1);
                llenaMatrizSinCrear(Ruta2sag6,1,
                    delta1+creall(obtieneElemento(
                    epsilon1,1,index1)),0,1);;
        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;}
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
        //for(int iter=1;iter<iteraciones;iter++)
            for(int iter=iteraciones-1;iter>0;iter--)
            {
                //int ahora=iter;
```

```
//int siguiente=iter+1;
int ahora=iteraciones_iter;
int siguiente=ahora+1;
matriz * Prop1_piezas[]={Ruta2sag4,
    Ruta2sag3,Ruta2sag2,Ruta2sag1}; //
    FM1 a refracción en medio Kerr
matriz * Prop1=variasMultMatriciales(
    Prop1_piezas,4);
long double complex q1_hold=
    obtieneElemento(q_iter,1,ahora);
q_2=prop_q(Prop1,q1_hold,1,n0);
q_3=propagacionKerr(pasosKerr,L,n0,n2
    ,w_pump,q_2,chi,kth,Cp,rho,dn_dv,
    P_pump,P_laser,lambda0);
matriz * Prop2_piezas[]={Ruta2sag8,
    Ruta2sag7,Ruta2sag6,Ruta2sag5}; //
     5 a 8
matriz * Prop2=variasMultMatriciales(
    Prop2_piezas,4);
q_4=prop_q(Prop2,q_3,n0,1);
fijaElemento(q_iter,1,siguiente,q_4);
//printf("Actual: %f+i %f, siguiente:
     %f+i %f",creal(obtieneElemento(
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_4,1,lambda0);
if(w_iter_viejo>0.2||w_iter_nuevo
    >0.2) break; // Si el spot mide
    40 cm es demasiado
diferencia=fabsl((w_iter_nuevo
    w_iter_viejo)/w_iter_nuevo)*100.0;
if(diferencia<=umbral) // Termina la
    iteración
{
     fijaElemento(q_new,index1,index2,
         obtieneElemento(q_iter,1,
         siguiente));
```

```
                        termino=1;
                        break;
                    }
                    else
                    {
                        fijaElemento(q_new,index1,index2
                            ,0);
                        termino=0;
                    }
                }
            }
            if(termino==0)
                printf("Cavidad en anillo, Ruta2Sag: No
                    cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
            else
                printf("Cavidad en anillo, Ruta2Sag:
                    Cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
        }
    }
    q_iter=borraMatriz(q_iter);
    Ruta2sag1=borraMatriz(Ruta2sag1);
    Ruta2sag2=borraMatriz(Ruta2sag2);
    Ruta2sag3=borraMatriz(Ruta2sag3);
    Ruta2sag4=borraMatriz(Ruta2sag4);
    Ruta2sag5=borraMatriz(Ruta2sag5);
    Ruta2sag6=borraMatriz(Ruta2sag6);
    Ruta2sag7=borraMatriz(Ruta2sag7);
    Ruta2sag8=borraMatriz(Ruta2sag8);
    return q_new;
}
```

no_lineal.h

```
#ifndef NO_LINEAL_H_INCLUDED
#define NO_LINEAL_H_INCLUDED
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include "matrices.h"
#include "lineal.h"

long double complex propagacionKerr(int pasos, long
    double L, long double n0,long double n2, long double
    w_pump,long double complex q_in, long double chi, long
     double kth, long double Cp, long double rho, long
    double dn_dv,long double P_pump,long double P_laser,
    long double lambda0);
long double complex * propagacionKerrMatriz(int pasos,
    long double L, long double n0,long double n2, long
    double w_pump,long double complex q_in, long double
    chi, long double kth, long double Cp, long double rho,
     long double dn_dv,long double P_pump,long double
    P_laser, long double lambda0);
long double complex * propagacionLibre(long double L,int
    pasos, long double n0,long double complex q_in);
long double complex prop_q(matriz * ABCD, long double
    complex q_in, long double n1, long double n2); //
    Propagación de un haz gaussiano a través de un sistema
     óptico ABCD  Siegman
long double spot_q(long double complex q, long double n,
    long double lambda);
long double radio_q(long double complex q);
void confNoAstigmatica(matriz * A, long double complex *
    casiUno, int *fila, int *columna);
matriz * spot_q_matriz(matriz * q, long double n, long
    double lambda);
matriz * propNoLineal_tanEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLineal_sagEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
```

```
    w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLineal_tanEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
    w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLineal_sagEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
    w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLineal_AnilloRuta1Tan(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLineal_AnilloRuta1Sag(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLineal_AnilloRuta2Tan(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
```

```
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLineal_AnilloRuta2Sag(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);


#endif // NO_LINEAL_H_INCLUDED
```

A.5.5. no_linealRK.c y no_linealRK.h

no_linealRK.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include <assert.h>
#include "matrices.h"
#include "lineal.h"
#include "no_lineal.h"
#include "error_iteraciones.h"

int pasosKerrRK=1000;
// Resolver el sistema de ecuaciones diferenciales
    acopladas mostradas en el artículo de D. Huang.
// Emplea Runge Kutta de 4to orden. Se define a p=1/q,
    por lo que pReal=Re(p) y pImag=Im(p).
// xi queda definido por sqrt(1 P_laser/P_cr). Registrar
    la posición al momento de evaluar

// Ecuaciones diferenciales

long double dpReal(long double pReal, long double pImag,
    long double n0, long double n2, long double lambda0,
    long double P_laser) //
{
     long double evaluacion;
```

```
    //evaluacion = powl(pReal,2)+powl(pImag,2)*(1 (M_PI*n0
        *8*n2*P_laser)/lambda0);
    evaluacion = powl(pReal,2)+powl(pImag,2)*(1 (M_PI*n0*
        n2*P_laser)/(2*powl(lambda0,2))); // Ajuste
        cuadrático
    return evaluacion;
}

long double dpImag(long double pReal, long double pImag,
    long double n0)
{
    long double evaluacion;
    evaluacion = 2*pReal*pImag;
    return evaluacion;
}

// Runge Kutta 4to orden aplicado a ec. diferenciales de
    Huang, h es el tamaño de paso.

long double * pasoKerrRK(long double h, long double
    pReal_ini, long double pImag_ini, long double n0, long
     double n2, long double lambda0, long double P_laser)
{
    long double k[2][4], pReal, pReal_out, pImag,
        pImag_out;
    long double *salida=(long double *)calloc(2,sizeof(
        long double));
    memset(k,0,sizeof k);
    pReal=pReal_ini;
    pImag=pImag_ini;
    k[0][0]=dpReal(pReal,pImag,n0,n2,lambda0,P_laser);
    k[1][0]=dpImag(pReal,pImag,n0);
    pReal=pReal_ini+k[0][0]*h/2;
    pImag=pImag_ini+k[1][0]*h/2;
    k[0][1]=dpReal(pReal,pImag,n0,n2,lambda0,P_laser);
    k[1][1]=dpImag(pReal,pImag,n0);
    pReal=pReal_ini+k[0][1]*h/2;
    pImag=pImag_ini+k[1][1]*h/2;
    k[0][2]=dpReal(pReal,pImag,n0,n2,lambda0,P_laser);
    k[1][2]=dpImag(pReal,pImag,n0);
    pReal=pReal_ini+k[0][2]*h;
    pImag=pImag_ini+k[1][2]*h;
    k[0][3]=dpReal(pReal,pImag,n0,n2,lambda0,P_laser);
    k[1][3]=dpImag(pReal,pImag,n0);
    pReal_out=pReal_ini+(k[0][0]+2*(k[0][1]+k[0][2])+k
        [0][3])*h/6;
```

```
    pImag_out=pImag_ini+(k[1][0]+2*(k[1][1]+k[1][2])+k
        [1][3])*h/6;
    salida[0]=pReal_out;
    salida[1]=pImag_out;
    return salida;
}

long double complex propagacionKerrRK(long double complex
    q_in, long double lambda0, long double L, long double
    n0, long double n2, long double P_laser, int pasos)
{
    long double h=L/pasos;
    long double complex p_out, q_out;
    long double *pReal = (long double *)calloc(pasos,
        sizeof(long double));
    long double *pImag = (long double *)calloc(pasos,
        sizeof(long double));
    long double *datosPaso;
    //long double *datosPaso = (long double *)calloc(2,
        sizeof(long double));
    assert(pReal);
    assert(pImag);
    int i=pasos 1;
    pReal[i]=creall(1/q_in);
    pImag[i]=cimagl(1/q_in);
    for(i=pasos 1;i>0;i  )
    {
        datosPaso=pasoKerrRK(h,pReal[i],pImag[i],n0,n2,
            lambda0,P_laser);
        pReal[i 1]=datosPaso[0];
        pImag[i 1]=datosPaso[1];
        free(datosPaso);
    }
    p_out=pReal[0]+I*pImag[0];
    q_out=1/p_out;
    free(pReal);
    free(pImag);
    return q_out;
}

matriz * propNoLinealRK_tanEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
```

```
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM1TanRK";
    char subCarpeta[]="EM1Tan_RK";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4,q_5,q_6; //
        Parámetros complejos para propagación

    // Crea matrices para la propagación

    matriz *EM1tan1,*EM1tan2,*EM1tan3,*EM1tan4,*EM1tan5,*
        EM1tan6,*EM1tan7,*EM1tan8,*EM1tan9,*EM1tan10,*
        EM1tan11,*EM1tan12,*EM1tan13,*EM1tan14,*EM1tan15,*
        EM1tan16,*EM1tan17;
    // EM1, Caso tangencial
    EM1tan1=llenaMatriz(1,L1,0,1);
    EM1tan2=llenaMatriz(1,0, 1/f1t,1);
```

```
EM1tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1tan4=llenaMatriz(n0,0,0,1/n0);
EM1tan5=llenaMatriz(1/n0,0,0,n0);
EM1tan6=nuevaMatriz(2,2);//EM1tan5 pendiente
EM1tan7=llenaMatriz(1,0, 1/f2t,1);
EM1tan8=llenaMatriz(1,L2,0,1);
EM1tan9=llenaMatriz(1,0,0,1);
EM1tan10=llenaMatriz(1,L2,0,1);
EM1tan11=llenaMatriz(1,0, 1/f2t,1);
EM1tan12=nuevaMatriz(2,2);      //EM1tan11 pendiente
EM1tan13=llenaMatriz(n0,0,0,1/n0);
EM1tan14=llenaMatriz(1/n0,0,0,n0);
EM1tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1tan16=llenaMatriz(1,0, 1/f1t,1);
EM1tan17=llenaMatriz(1,L1,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,2);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM1tan3,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM1tan6,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1tan12,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1tan15,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);

        if(elem(q_in,index1,index2)==0)
                {elem(q_new,index1,index2)=0;
         escribeCSV=0;
         reallocFallido=0;
```

```
    }
else
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        q_in,index1,index2));
    escribeCSV=1;                    //for(int
        iter=1;iter<iteraciones;iter++)
   for(;;)
    {
        int ahora=1;
        int siguiente=2;
        matriz * Prop1_piezas[]={EM1tan4,
            EM1tan3,EM1tan2,EM1tan1};
        matriz * Prop1=variasMultMatriciales(
            Prop1_piezas,4);
        long double complex q1_hold=
            obtieneElemento(q_iter,1,ahora);
        //q_2=prop_q(Prop1,obtieneElemento(
            q_iter,1,ahora),1,n0);
        q_2=prop_q(Prop1,q1_hold,1,n0);
        //q_3=propagacionKerr(pasosKerrRK,L,
            n0,n2,w_pump,q_2,chi,kth,Cp,rho,
            dn_dv,P_pump,P_laser,lambda0);
        q_3=propagacionKerrRK(q_2,lambda0,L,
            n0,n2,P_laser,pasosKerrRK);
        matriz * Prop2_piezas[]={EM1tan13,
            EM1tan12,EM1tan11,EM1tan10,EM1tan9
            ,EM1tan8,EM1tan7,EM1tan6,EM1tan5};
             // 5 a 13
        matriz * Prop2=variasMultMatriciales(
            Prop2_piezas,9);
        q_4=prop_q(Prop2,q_3,n0,n0);
        //q_5=propagacionKerr(pasosKerrRK,L,
            n0,n2,w_pump,q_4,chi,kth,Cp,rho,
            dn_dv,P_pump,P_laser,lambda0);
        q_5=propagacionKerrRK(q_4,lambda0,L,
            n0,n2,P_laser,pasosKerrRK);
        matriz * Prop3_piezas[]={EM1tan17,
            EM1tan16,EM1tan15,EM1tan14}; // 14
             a 17
        matriz * Prop3=variasMultMatriciales(
            Prop3_piezas,4);
        q_6=prop_q(Prop3,q_5,n0,1);
        fijaElemento(q_iter,1,ahora,q_6);
        //printf("Actual: %f+i %f, siguiente:
             %f+i %f",creal(obtieneElemento(
```

```
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieaza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
Prop3=borraMatriz(Prop3);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_6,1,lambda0);//
     Prepara vectores
if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_spt, *tmp_iter;
tmp_spt=(long double*)realloc(spots,
    cuenta*sizeof(long double));
tmp_iter=(int *)realloc(
    iteracionActual,cuenta*sizeof(int)
    );
if(tmp_spt!=NULL && tmp_iter!=NULL)
{
     spots=tmp_spt;
     iteracionActual=tmp_iter;
     spots[cuenta-1]=w_iter_nuevo;
     iteracionActual[cuenta-1]=cuenta
          -1;
}
else
{
     reallocFallido=1;
     break;
}
if(w_iter_viejo>0.1||w_iter_nuevo
    >0.1||isnan(w_iter_viejo) ||isnan(
    w_iter_nuevo))
{
     termino=0;
     escribeCSV=1;
     fijaElemento(q_new,index1,index2
          ,0);
     break;
```

```
            } // Si el spot mide 20 cm es
                demasiado
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo)
                *100.00;
            if(diferencia<=umbral) // Termina la
                iteración
            {
                //fijaElemento(q_new,index1,
                    index2,obtieneElemento(q_iter
                    ,1,ahora));
                long double complex valor=
                    obtieneElemento(q_iter,1,1);
                fijaElemento(q_new,index1,index2,
                    valor);
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spots);
        free(iteracionActual);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spots[cuenta 1],cuenta,
        creall(obtieneElemento(epsilon1,1,index1))
        ,creall(obtieneElemento(epsilon2,1,index2)
        ),tipo,termino);
    if(escribeCSV==1)
        guardaVariaciones(spots,iteracionActual,
            cuenta,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),tipo,subCarpeta,
            termino);
    if(termino==0)
        printf("Cavidad en X, EM1tanRK: No cumple
```

```
                    con el umbral de %Le para epsilon1=%
                    Le, epsilon2=%Le.\nError relativo: %Le
                    \n",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            else
                printf("Cavidad en X, EM1tanRK: Cumple
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo: %Le\n
                    ",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM1tan1);
    borraMatriz(EM1tan2);
    borraMatriz(EM1tan3);
    borraMatriz(EM1tan4);
    borraMatriz(EM1tan5);
    borraMatriz(EM1tan6);
    borraMatriz(EM1tan7);
    borraMatriz(EM1tan8);
    borraMatriz(EM1tan9);
    borraMatriz(EM1tan10);
    borraMatriz(EM1tan11);
    borraMatriz(EM1tan12);
    borraMatriz(EM1tan13);
    borraMatriz(EM1tan14);
    borraMatriz(EM1tan15);
    borraMatriz(EM1tan16);
    borraMatriz(EM1tan17);
    return q_new;
}

// EM1 sagital

matriz * propNoLinealRK_sagEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
```

```
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM1SagRK";
    char subCarpeta[]="EM1Sag_RK";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4,q_5,q_6; //
        Parámetros complejos para propagación

    // Crea matrices para la propagación

    matriz *EM1sag1,*EM1sag2,*EM1sag3,*EM1sag4,*EM1sag5,*
        EM1sag6,*EM1sag7,*EM1sag8,*EM1sag9,*EM1sag10,*
        EM1sag11,*EM1sag12,*EM1sag13,*EM1sag14,*EM1sag15,*
        EM1sag16,*EM1sag17;
    // EM1, Caso saggencial
    EM1sag1=llenaMatriz(1,L1,0,1);
```

```
EM1sag2=llenaMatriz(1,0, 1/f1s,1);
EM1sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1sag4=llenaMatriz(1,0,0,1);
EM1sag5=llenaMatriz(1,0,0,1);
EM1sag6=nuevaMatriz(2,2);//EM1sag5 pendiente
EM1sag7=llenaMatriz(1,0, 1/f2s,1);
EM1sag8=llenaMatriz(1,L2,0,1);
EM1sag9=llenaMatriz(1,0,0,1);
EM1sag10=llenaMatriz(1,L2,0,1);
EM1sag11=llenaMatriz(1,0, 1/f2s,1);
EM1sag12=nuevaMatriz(2,2);      //EM1sag11 pendiente
EM1sag13=llenaMatriz(1,0,0,1);
EM1sag14=llenaMatriz(1,0,0,1);
EM1sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1sag16=llenaMatriz(1,0, 1/f1s,1);
EM1sag17=llenaMatriz(1,L1,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM1sag3,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM1sag6,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1sag12,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM1sag15,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);

        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;
            escribeCSV=0;
```

```
        reallocFallido=0;
        }
    else
    {
        fijaElemento(q_iter,1,1,obtieneElemento(
            q_in,index1,index2));
        escribeCSV=1;                    //for(int
            iter=1;iter<iteraciones;iter++)
         for(;;)
        {
            int ahora=1;
            int siguiente=2;
            matriz * Prop1_piezas[]={EM1sag4,
                EM1sag3,EM1sag2,EM1sag1};
            matriz * Prop1=variasMultMatriciales(
                Prop1_piezas,4);
            long double complex q1_hold=
                obtieneElemento(q_iter,1,ahora);
            //q_2=prop_q(Prop1,obtieneElemento(
                q_iter,1,ahora),1,n0);
            q_2=prop_q(Prop1,q1_hold,1,n0);
            //q_3=propagacionKerr(pasosKerrRK,L,
                n0,n2,w_pump,q_2,chi,kth,Cp,rho,
                dn_dv,P_pump,P_laser,lambda0);
            q_3=propagacionKerrRK(q_2,lambda0,L,
                n0,n2,P_laser,pasosKerrRK);
            matriz * Prop2_piezas[]={EM1sag13,
                EM1sag12,EM1sag11,EM1sag10,EM1sag9
                ,EM1sag8,EM1sag7,EM1sag6,EM1sag5};
                 // 5 a 13
            matriz * Prop2=variasMultMatriciales(
                Prop2_piezas,9);
            q_4=prop_q(Prop2,q_3,n0,n0);
            //q_5=propagacionKerr(pasosKerrRK,L,
                n0,n2,w_pump,q_4,chi,kth,Cp,rho,
                dn_dv,P_pump,P_laser,lambda0);
            q_5=propagacionKerrRK(q_4,lambda0,L,
                n0,n2,P_laser,pasosKerrRK);
            matriz * Prop3_piezas[]={EM1sag17,
                EM1sag16,EM1sag15,EM1sag14}; // 14
                 a 17
            matriz * Prop3=variasMultMatriciales(
                Prop3_piezas,4);
            q_6=prop_q(Prop3,q_5,n0,1);
            fijaElemento(q_iter,1,ahora,q_6);
            //printf("Actual: %f+i %f, siguiente:
```

```
        %f+i %f", creal(obtieneElemento(
        q_iter,1,iter)),cimag(
        obtieneElemento(q_iter,1,iter)),
        creal(obtieneElemento(q_iter,1,
        iter+1)),cimag(obtieneElemento(
        q_iter,1,iter+1)));
    // Limpieaza
    Prop1=borraMatriz(Prop1);
    Prop2=borraMatriz(Prop2);
    Prop3=borraMatriz(Prop3);
    // Calculando spots y comparación
    w_iter_viejo=spot_q(q1_hold,1,lambda0
        );
    w_iter_nuevo=spot_q(q_6,1,lambda0);//
         Prepara vectores
    if(cuenta>iteraciones)  break; // Por
         si acaso
    ++cuenta;
    //printf("%i\n",cuenta);
    void *tmp_spt, *tmp_iter;
    tmp_spt=(long double*)realloc(spots,
        cuenta*sizeof(long double));
    tmp_iter=(int *)realloc(
        iteracionActual,cuenta*sizeof(int)
        );
    if(tmp_spt!=NULL && tmp_iter!=NULL)
    {
         spots=tmp_spt;
         iteracionActual=tmp_iter;
         spots[cuenta-1]=w_iter_nuevo;
         iteracionActual[cuenta-1]=cuenta
             -1;
    }
    else
    {
         reallocFallido=1;
         break;
    }
    if(w_iter_viejo>0.1||w_iter_nuevo
        >0.1||isnan(w_iter_viejo) ||isnan(
        w_iter_nuevo))
    {
         termino=0;
         escribeCSV=1;
         fijaElemento(q_new,index1,index2
             ,0);
```

```
                    break;
                } // Si el spot mide 20 cm es
                    demasiado
                diferencia=fabsl((w_iter_nuevo
                    w_iter_viejo)/w_iter_nuevo)
                    *100.00;
                if(diferencia<=umbral) // Termina la
                    iteración
                {
                    fijaElemento(q_new,index1,index2,
                        obtieneElemento(q_iter,1,ahora
                        ));
                    termino=1;
                    break;
                }
                else
                {
                    fijaElemento(q_new,index1,index2
                        ,0);
                    termino=0;
                }
            }
        }
        if(reallocFallido==1)
        {
            termino=0;
            free(spots);
            free(iteracionActual);
            printf("Se acabó la memoria\n");
            reallocFallido=0;
        }
        guardaSpotIteraciones(spots[cuenta 1],cuenta,
            creall(obtieneElemento(epsilon1,1,index1))
            ,creall(obtieneElemento(epsilon2,1,index2)
            ),tipo,termino);
        if(escribeCSV==1)
            guardaVariaciones(spots,iteracionActual,
                cuenta,creall(obtieneElemento(epsilon1
                ,1,index1)),creall(obtieneElemento(
                epsilon2,1,index2)),tipo,subCarpeta,
                termino);
        if(termino==0)
            printf("Cavidad en X, EM1sagRK: No cumple
                con el umbral de %Le para epsilon1=%
                Le, epsilon2=%Le.\nError relativo: %Le
                \n",umbral,creall(obtieneElemento(
```

```
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            else
                printf("Cavidad en X, EM1sagRK: Cumple
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo: %Le\n
                    ",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM1sag1);
    borraMatriz(EM1sag2);
    borraMatriz(EM1sag3);
    borraMatriz(EM1sag4);
    borraMatriz(EM1sag5);
    borraMatriz(EM1sag6);
    borraMatriz(EM1sag7);
    borraMatriz(EM1sag8);
    borraMatriz(EM1sag9);
    borraMatriz(EM1sag10);
    borraMatriz(EM1sag11);
    borraMatriz(EM1sag12);
    borraMatriz(EM1sag13);
    borraMatriz(EM1sag14);
    borraMatriz(EM1sag15);
    borraMatriz(EM1sag16);
    borraMatriz(EM1sag17);
    return q_new;
}

// EM2 tangencial
matriz * propNoLinealRK_tanEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
```

```
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spots;
    int * iteracionActual;
    int cuenta=0;
    char tipo[]="EM2TanRK";
    char subCarpeta[]="EM2Tan_RK";
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4,q_5,q_6; //
        Parámetros complejos para propagación

    // Crea matrices para la propagación

    matriz *EM2tan1,*EM2tan2,*EM2tan3,*EM2tan4,*EM2tan5,*
        EM2tan6,*EM2tan7,*EM2tan8,*EM2tan9,*EM2tan10,*
        EM2tan11,*EM2tan12,*EM2tan13,*EM2tan14,*EM2tan15,*
        EM2tan16,*EM2tan17;
    // EM2, Caso tangencial
    EM2tan1=llenaMatriz(1,L2,0,1);
    EM2tan2=llenaMatriz(1,0,1/f2t,1);
    EM2tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
        ;
    EM2tan4=llenaMatriz(n0,0,0,1/n0);
```

```
EM2tan5=llenaMatriz(1/n0,0,0,n0);
EM2tan6=nuevaMatriz(2,2);//EM2tan5 pendiente
EM2tan7=llenaMatriz(1,0, 1/f1t,1);
EM2tan8=llenaMatriz(1,L1,0,1);
EM2tan9=llenaMatriz(1,0,0,1);
EM2tan10=llenaMatriz(1,L1,0,1);
EM2tan11=llenaMatriz(1,0, 1/f1t,1);
EM2tan12=nuevaMatriz(2,2);      //EM2tan11 pendiente
EM2tan13=llenaMatriz(n0,0,0,1/n0);
EM2tan14=llenaMatriz(1/n0,0,0,n0);
EM2tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2tan16=llenaMatriz(1,0, 1/f2t,1);
EM2tan17=llenaMatriz(1,L2,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM2tan3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2tan6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);

        if(elem(q_in,index1,index2)==0)
        {elem(q_new,index1,index2)=0;
            escribeCSV=0;
            reallocFallido=0;
            }
        else
        {
```

```
fijaElemento(q_iter,1,1,obtieneElemento(
    q_in,index1,index2));
escribeCSV=1;                    //for(int
    iter=1;iter<iteraciones;iter++)
for(;;)
{
    int ahora=1;
    int siguiente=2;

    matriz * Prop1_piezas[]={EM2tan4,
        EM2tan3,EM2tan2,EM2tan1};
    matriz * Prop1=variasMultMatriciales(
        Prop1_piezas,4);
    long double complex q1_hold=
        obtieneElemento(q_iter,1,ahora);
    //q_2=prop_q(Prop1,obtieneElemento(
        q_iter,1,ahora),1,n0);
    q_2=prop_q(Prop1,q1_hold,1,n0);
    //q_3=propagacionKerr(pasosKerrRK,L,
        n0,n2,w_pump,q_2,chi,kth,Cp,rho,
        dn_dv,P_pump,P_laser,lambda0);
    q_3=propagacionKerrRK(q_2,lambda0,L,
        n0,n2,P_laser,pasosKerrRK);
    matriz * Prop2_piezas[]={EM2tan13,
        EM2tan12,EM2tan11,EM2tan10,EM2tan9
        ,EM2tan8,EM2tan7,EM2tan6,EM2tan5};
         // 5 a 13
    matriz * Prop2=variasMultMatriciales(
        Prop2_piezas,9);
    q_4=prop_q(Prop2,q_3,n0,n0);
    //q_5=propagacionKerr(pasosKerrRK,L,
        n0,n2,w_pump,q_4,chi,kth,Cp,rho,
        dn_dv,P_pump,P_laser,lambda0);
    q_5=propagacionKerrRK(q_4,lambda0,L,
        n0,n2,P_laser,pasosKerrRK);
    matriz * Prop3_piezas[]={EM2tan17,
        EM2tan16,EM2tan15,EM2tan14}; // 14
         a 17
    matriz * Prop3=variasMultMatriciales(
        Prop3_piezas,4);
    q_6=prop_q(Prop3,q_5,n0,1);
    fijaElemento(q_iter,1,ahora,q_6);
    //printf("Actual: %f+i %f, siguiente:
         %f+i %f",creal(obtieneElemento(
        q_iter,1,iter)),cimag(
        obtieneElemento(q_iter,1,iter)),
```

```
        creal(obtieneElemento(q_iter,1,
        iter+1)),cimag(obtieneElemento(
        q_iter,1,iter+1)));
    // Limpieaza
    Prop1=borraMatriz(Prop1);
    Prop2=borraMatriz(Prop2);
    Prop3=borraMatriz(Prop3);
    // Calculando spots y comparación
    w_iter_viejo=spot_q(q1_hold,1,lambda0
        );
    w_iter_nuevo=spot_q(q_6,1,lambda0);//
         Prepara vectores
    if(cuenta>iteraciones)  break; // Por
         si acaso
    ++cuenta;
    //printf("%i\n",cuenta);
    void *tmp_spt, *tmp_iter;
    tmp_spt=(long double*)realloc(spots,
        cuenta*sizeof(long double));
    tmp_iter=(int *)realloc(
        iteracionActual,cuenta*sizeof(int)
        );
    if(tmp_spt!=NULL && tmp_iter!=NULL)
    {
         spots=tmp_spt;
         iteracionActual=tmp_iter;
         spots[cuenta-1]=w_iter_nuevo;
         iteracionActual[cuenta-1]=cuenta
             -1;
    }
    else
    {
         reallocFallido=1;
         break;
    }
    if(w_iter_viejo>0.1||w_iter_nuevo
        >0.1||isnan(w_iter_viejo) ||isnan(
        w_iter_nuevo))
    {
         termino=0;
         escribeCSV=1;
         fijaElemento(q_new,index1,index2
             ,0);
         break;
    }  // Si el spot mide 20 cm es
        demasiado
```

```
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo)
                *100.00;
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,ahora
                    ));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spots);
        free(iteracionActual);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spots[cuenta 1],cuenta,
        creall(obtieneElemento(epsilon1,1,index1))
        ,creall(obtieneElemento(epsilon2,1,index2)
        ),tipo,termino);
    if(escribeCSV==1)
        guardaVariaciones(spots,iteracionActual,
            cuenta,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),tipo,subCarpeta,
            termino);
    if(termino==0)
        printf("Cavidad en X, EM2tanRK: No cumple
            con el umbral de %Le para epsilon1=%
            Le, epsilon2=%Le.\nError relativo: %Le
            \n",umbral,creall(obtieneElemento(
            epsilon1,1,index1)),creall(
            obtieneElemento(epsilon2,1,index2)),
            diferencia);
```

```
                else
                    printf("Cavidad en X, EM2tanRK: Cumple
                        con el umbral de %Le para epsilon1=%Le
                        , epsilon2=%Le.\nError relativo: %Le\n
                        ",umbral,creall(obtieneElemento(
                        epsilon1,1,index1)),creall(
                        obtieneElemento(epsilon2,1,index2)),
                        diferencia);
                free(spots);
                free(iteracionActual);
            }
        }
        borraMatriz(q_iter);
        borraMatriz(EM2tan1);
        borraMatriz(EM2tan2);
        borraMatriz(EM2tan3);
        borraMatriz(EM2tan4);
        borraMatriz(EM2tan5);
        borraMatriz(EM2tan6);
        borraMatriz(EM2tan7);
        borraMatriz(EM2tan8);
        borraMatriz(EM2tan9);
        borraMatriz(EM2tan10);
        borraMatriz(EM2tan11);
        borraMatriz(EM2tan12);
        borraMatriz(EM2tan13);
        borraMatriz(EM2tan14);
        borraMatriz(EM2tan15);
        borraMatriz(EM2tan16);
        borraMatriz(EM2tan17);
        return q_new;
}

matriz * propNoLinealRK_sagEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
```

```
_Bool termino=0;
_Bool escribeCSV=0;
_Bool reallocFallido=0;
long double *spots;
int * iteracionActual;
int cuenta=0;
char tipo[]="EM2SagRK";
char subCarpeta[]="EM2Sag_RK";
long double w_iter_viejo=0, w_iter_nuevo=0,
    diferencia=0;
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4,q_5,q_6; //
    Parámetros complejos para propagación

// Crea matrices para la propagación

matriz *EM2sag1,*EM2sag2,*EM2sag3,*EM2sag4,*EM2sag5,*
    EM2sag6,*EM2sag7,*EM2sag8,*EM2sag9,*EM2sag10,*
    EM2sag11,*EM2sag12,*EM2sag13,*EM2sag14,*EM2sag15,*
    EM2sag16,*EM2sag17;
// EM2, Caso saggencial
EM2sag1=llenaMatriz(1,L2,0,1);
EM2sag2=llenaMatriz(1,0, 1/f2s,1);
EM2sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2sag4=llenaMatriz(1,0,0,1);
EM2sag5=llenaMatriz(1,0,0,1);
EM2sag6=nuevaMatriz(2,2);//EM2sag5 pendiente
EM2sag7=llenaMatriz(1,0, 1/f1s,1);
EM2sag8=llenaMatriz(1,L1,0,1);
```

```
EM2sag9=llenaMatriz(1,0,0,1);
EM2sag10=llenaMatriz(1,L1,0,1);
EM2sag11=llenaMatriz(1,0, 1/f1s,1);
EM2sag12=nuevaMatriz(2,2);      //EM2sag11 pendiente
EM2sag13=llenaMatriz(1,0,0,1);
EM2sag14=llenaMatriz(1,0,0,1);
EM2sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2sag16=llenaMatriz(1,0, 1/f2s,1);
EM2sag17=llenaMatriz(1,L2,0,1);

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spots=(long double*)calloc(1,sizeof(long
            double));
        iteracionActual=(int*)calloc(1,sizeof(int));
        cuenta=1;
        iteracionActual[0]=0;
        spots[0]=spot_q(obtieneElemento(q_in,index1,
            index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM2sag3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2sag6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);


        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;
            escribeCSV=0;
            reallocFallido=0;
            }
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            escribeCSV=1;                   //for(int
```

```
    iter=1;iter<iteraciones;iter++)
  for(;;)
{
    int ahora=1;
    int siguiente=2;

    matriz * Prop1_piezas[]={EM2sag4,
        EM2sag3,EM2sag2,EM2sag1};
    matriz * Prop1=variasMultMatriciales(
        Prop1_piezas,4);
    long double complex q1_hold=
        obtieneElemento(q_iter,1,ahora);
    //q_2=prop_q(Prop1,obtieneElemento(
        q_iter,1,ahora),1,n0);
    q_2=prop_q(Prop1,q1_hold,1,n0);
    //q_3=propagacionKerr(pasosKerrRK,L,
        n0,n2,w_pump,q_2,chi,kth,Cp,rho,
        dn_dv,P_pump,P_laser,lambda0);
    q_3=propagacionKerrRK(q_2,lambda0,L,
        n0,n2,P_laser,pasosKerrRK);
    matriz * Prop2_piezas[]={EM2sag13,
        EM2sag12,EM2sag11,EM2sag10,EM2sag9
        ,EM2sag8,EM2sag7,EM2sag6,EM2sag5};
         // 5 a 13
    matriz * Prop2=variasMultMatriciales(
        Prop2_piezas,9);
    q_4=prop_q(Prop2,q_3,n0,n0);
    //q_5=propagacionKerr(pasosKerrRK,L,
        n0,n2,w_pump,q_4,chi,kth,Cp,rho,
        dn_dv,P_pump,P_laser,lambda0);
    q_5=propagacionKerrRK(q_4,lambda0,L,
        n0,n2,P_laser,pasosKerrRK);
    matriz * Prop3_piezas[]={EM2sag17,
        EM2sag16,EM2sag15,EM2sag14}; // 14
         a 17
    matriz * Prop3=variasMultMatriciales(
        Prop3_piezas,4);
    q_6=prop_q(Prop3,q_5,n0,1);
                          fijaElemento(
                              q_iter,1,ahora
                              ,q_6);
    //printf("Actual: %f+i %f, siguiente:
         %f+i%f",creal(obtieneElemento(
        q_iter,1,iter)),cimag(
        obtieneElemento(q_iter,1,iter)),
        creal(obtieneElemento(q_iter,1,
```

```
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieaza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
Prop3=borraMatriz(Prop3);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_6,1,lambda0);//
     Prepara vectores
if(cuenta>iteraciones)  break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_spt, *tmp_iter;
tmp_spt=(long double*)realloc(spots,
    cuenta*sizeof(long double));
tmp_iter=(int *)realloc(
    iteracionActual,cuenta*sizeof(int)
    );
if(tmp_spt!=NULL && tmp_iter!=NULL)
{
     spots=tmp_spt;
     iteracionActual=tmp_iter;
     spots[cuenta-1]=w_iter_nuevo;
     iteracionActual[cuenta-1]=cuenta
         -1;
}
else
{
     reallocFallido=1;
     break;
}
if(w_iter_viejo>0.1||w_iter_nuevo
    >0.1||isnan(w_iter_viejo) ||isnan(
    w_iter_nuevo))
{
     termino=0;
     escribeCSV=1;
     fijaElemento(q_new,index1,index2
         ,0);
     break;
}  // Si el spot mide 20 cm es
    demasiado
diferencia=fabsl((w_iter_nuevo
```

```
                w_iter_viejo)/w_iter_nuevo);
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,ahora
                    ));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spots);
        free(iteracionActual);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spots[cuenta 1],cuenta,
        creall(obtieneElemento(epsilon1,1,index1))
        ,creall(obtieneElemento(epsilon2,1,index2)
        ),tipo,termino);
    if(escribeCSV==1)
        guardaVariaciones(spots,iteracionActual,
            cuenta,creall(obtieneElemento(epsilon1
            ,1,index1)),creall(obtieneElemento(
            epsilon2,1,index2)),tipo,subCarpeta,
            termino);
    if(termino==0)
        printf("Cavidad en X, EM2sagRK: No cumple
            con el umbral de %Le para epsilon1=%
            Le, epsilon2=%Le.\nError relativo: %Le
            \n",umbral,creall(obtieneElemento(
            epsilon1,1,index1)),creall(
            obtieneElemento(epsilon2,1,index2)),
            diferencia);
    else
        printf("Cavidad en X, EM2sagRK: Cumple
```

```
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo: %Le\n
                    ",umbral,creall(obtieneElemento(
                    epsilon1,1,index1)),creall(
                    obtieneElemento(epsilon2,1,index2)),
                    diferencia);
            free(spots);
            free(iteracionActual);
        }
    }
    borraMatriz(q_iter);
    borraMatriz(EM2sag1);
    borraMatriz(EM2sag2);
    borraMatriz(EM2sag3);
    borraMatriz(EM2sag4);
    borraMatriz(EM2sag5);
    borraMatriz(EM2sag6);
    borraMatriz(EM2sag7);
    borraMatriz(EM2sag8);
    borraMatriz(EM2sag9);
    borraMatriz(EM2sag10);
    borraMatriz(EM2sag11);
    borraMatriz(EM2sag12);
    borraMatriz(EM2sag13);
    borraMatriz(EM2sag14);
    borraMatriz(EM2sag15);
    borraMatriz(EM2sag16);
    borraMatriz(EM2sag17);
    return q_new;
}

matriz * propNoLinealRK_AnilloRuta1Tan(matriz *q_in, char
    *conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
```

```
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    int pasosKerrRK=1000;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4; // Parámetros
        complejos para propagación

    // Crea matrices para la propagación
    matriz *Ruta1tan1,*Ruta1tan2,*Ruta1tan3,*Ruta1tan4,*
        Ruta1tan5,*Ruta1tan6,*Ruta1tan7,*Ruta1tan8;
   // Ruta 1, caso tangencial
    Ruta1tan1=llenaMatriz(1,L1,0,1); // L1=a
    Ruta1tan2=llenaMatriz(1,0,1/f1t,1);
    Ruta1tan3=nuevaMatriz(2,2); // Pendiente
    Ruta1tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
        entrada)
    Ruta1tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
        salida)
    Ruta1tan6=nuevaMatriz(2,2);//Ruta1tan6 pendiente
    Ruta1tan7=llenaMatriz(1,0,1/f2t,1);
    Ruta1tan8=llenaMatriz(1,L2,0,1); // L2=b+c

    // Ciclo
    matriz * q_iter = nuevaMatriz(1,iteraciones);
    for(int index1=1;index1<=epsilon1->columnas;index1++)
    {
        for(int index2=1;index2<=epsilon2->columnas;
            index2++)
```

```
{
    // Llenando matrices variables:
    llenaMatrizSinCrear(Ruta1tan3,1,delta1+creall
        (obtieneElemento(epsilon1,1,index1)),0,1);
                llenaMatrizSinCrear(Ruta1tan6,1,
                    delta2+creall(obtieneElemento(
                    epsilon2,1,index2)),0,1);;
    if(elem(q_in,index1,index2)==0)
        {elem(q_new,index1,index2)=0;}
    else
    {
        fijaElemento(q_iter,1,1,obtieneElemento(
            q_in,index1,index2));
        //for(int iter=1;iter<iteraciones;iter++)
        for(int iter=iteraciones 1;iter>0;iter  )
        {
             //int ahora=iter;
            //int siguiente=iter+1;
            int ahora=iteraciones iter;
            int siguiente=ahora+1;
            matriz * Prop1_piezas[]={Ruta1tan4,
                Ruta1tan3,Ruta1tan2,Ruta1tan1}; //
                FM1 a refracción en medio Kerr
            matriz * Prop1=variasMultMatriciales(
                Prop1_piezas,4);
            long double complex q1_hold=
                obtieneElemento(q_iter,1,ahora);
            q_2=prop_q(Prop1,q1_hold,1,n0);
            //q_3=propagacionKerr(pasosKerrRK,L,
                n0,n2,w_pump,q_2,chi,kth,Cp,rho,
                dn_dv,P_pump,P_laser,lambda0);
            q_3=propagacionKerrRK(q_2,lambda0,L,
                n0,n2,P_laser,pasosKerrRK);
            matriz * Prop2_piezas[]={Ruta1tan8,
                Ruta1tan7,Ruta1tan6,Ruta1tan5}; //
                 5 a 8
            matriz * Prop2=variasMultMatriciales(
                Prop2_piezas,4);
            q_4=prop_q(Prop2,q_3,n0,1.0);
            fijaElemento(q_iter,1,siguiente,q_4);
            //printf("Actual: %f+i %f, siguiente:
                 %f+i %f",creal(obtieneElemento(
                q_iter,1,iter)),cimag(
                obtieneElemento(q_iter,1,iter)),
                creal(obtieneElemento(q_iter,1,
                iter+1)),cimag(obtieneElemento(
```

```
                q_iter,1,iter+1)));
            // Limpieza
            Prop1=borraMatriz(Prop1);
            Prop2=borraMatriz(Prop2);
            // Calculando spots y comparación
            w_iter_viejo=spot_q(q1_hold,1,lambda0
                );
            w_iter_nuevo=spot_q(q_4,1,lambda0);
            if(w_iter_viejo>0.2||w_iter_nuevo
                >0.2) break;  // Si el spot mide
                40 cm es demasiado
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo)
                *100.00;
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,
                    siguiente));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
    if(termino==0)
        printf("Cavidad en anillo, Ruta1TanRK: No
            cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),diferencia);
    else
        printf("Cavidad en anillo, Ruta1TanRK:
            Cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
```

```
                    index2)),diferencia);
            }
        }
        q_iter=borraMatriz(q_iter);
        Ruta1tan1=borraMatriz(Ruta1tan1);
        Ruta1tan2=borraMatriz(Ruta1tan2);
        Ruta1tan3=borraMatriz(Ruta1tan3);
        Ruta1tan4=borraMatriz(Ruta1tan4);
        Ruta1tan5=borraMatriz(Ruta1tan5);
        Ruta1tan6=borraMatriz(Ruta1tan6);
        Ruta1tan7=borraMatriz(Ruta1tan7);
        Ruta1tan8=borraMatriz(Ruta1tan8);
        return q_new;
}

matriz * propNoLinealRK_AnilloRuta1Sag(matriz *q_in, char
    *conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    int pasosKerrRK=1000;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
```

```
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4; // Parámetros
    complejos para propagación

// Crea matrices para la propagación
matriz *Ruta1sag1,*Ruta1sag2,*Ruta1sag3,*Ruta1sag4,*
    Ruta1sag5,*Ruta1sag6,*Ruta1sag7,*Ruta1sag8;
// Ruta 1, caso sagital
Ruta1sag1=llenaMatriz(1,L1,0,1); // L1=a
Ruta1sag2=llenaMatriz(1,0, 1/f1s,1);
Ruta1sag3=nuevaMatriz(2,2); // Pendiente
Ruta1sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
    entrada)
Ruta1sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
    )
Ruta1sag6=nuevaMatriz(2,2);//Ruta1sag6 pendiente
Ruta1sag7=llenaMatriz(1,0, 1/f2s,1);
Ruta1sag8=llenaMatriz(1,L2,0,1); // L2=b+c

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1->columnas;index1++)
{
    for(int index2=1;index2<=epsilon2->columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta1sag3,1,delta1+creall
            (obtieneElemento(epsilon1,1,index1)),0,1);
                llenaMatrizSinCrear(Ruta1sag6,1,
                    delta2+creall(obtieneElemento(
                    epsilon2,1,index2)),0,1);;
        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;}
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            //for(int iter=1;iter<iteraciones;iter++)
            for(int iter=iteraciones-1;iter>0;iter--)
            {
```

```
 //int ahora=iter;
//int siguiente=iter+1;
int ahora=iteraciones_iter;
int siguiente=ahora+1;
matriz * Prop1_piezas[]={Ruta1sag4,
    Ruta1sag3,Ruta1sag2,Ruta1sag1}; //
    FM1 a refracción en medio Kerr
matriz * Prop1=variasMultMatriciales(
    Prop1_piezas,4);
long double complex q1_hold=
    obtieneElemento(q_iter,1,ahora);
q_2=prop_q(Prop1,q1_hold,1,n0);
//q_3=propagacionKerr(pasosKerrRK,L,
    n0,n2,w_pump,q_2,chi,kth,Cp,rho,
    dn_dv,P_pump,P_laser,lambda0);
q_3=propagacionKerrRK(q_2,lambda0,L,
    n0,n2,P_laser,pasosKerrRK);
matriz * Prop2_piezas[]={Ruta1sag8,
    Ruta1sag7,Ruta1sag6,Ruta1sag5}; //
     5 a 8
matriz * Prop2=variasMultMatriciales(
    Prop2_piezas,4);
q_4=prop_q(Prop2,q_3,n0,1);
fijaElemento(q_iter,1,siguiente,q_4);
//printf("Actual: %f+i %f, siguiente:
     %f+i %f",creal(obtieneElemento(
    q_iter,1,iter)),cimag(
    obtieneElemento(q_iter,1,iter)),
    creal(obtieneElemento(q_iter,1,
    iter+1)),cimag(obtieneElemento(
    q_iter,1,iter+1)));
// Limpieza
Prop1=borraMatriz(Prop1);
Prop2=borraMatriz(Prop2);
// Calculando spots y comparación
w_iter_viejo=spot_q(q1_hold,1,lambda0
    );
w_iter_nuevo=spot_q(q_4,1,lambda0);
if(w_iter_viejo>0.2||w_iter_nuevo
    >0.2) break; // Si el spot mide
    40 cm es demasiado
diferencia=fabsl((w_iter_nuevo
    w_iter_viejo)/w_iter_nuevo)
    *100.00;
if(diferencia<=umbral) // Termina la
    iteración
```

```
                    {
                        //fijaElemento(q_new,index1,
                            index2,obtieneElemento(q_iter
                            ,1,iter+1));
                        fijaElemento(q_new,index1,index2,
                            obtieneElemento(q_iter,1,
                            siguiente));
                        termino=1;
                        break;
                    }
                    else
                    {
                        fijaElemento(q_new,index1,index2
                            ,0);
                        termino=0;
                    }
                }
            }
            if(termino==0)
                printf("Cavidad en anillo, Ruta1SagRK: No
                     cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
            else
                printf("Cavidad en anillo, Ruta1SagRK:
                    Cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
        }
    }
    q_iter=borraMatriz(q_iter);
    Ruta1sag1=borraMatriz(Ruta1sag1);
    Ruta1sag2=borraMatriz(Ruta1sag2);
    Ruta1sag3=borraMatriz(Ruta1sag3);
    Ruta1sag4=borraMatriz(Ruta1sag4);
    Ruta1sag5=borraMatriz(Ruta1sag5);
    Ruta1sag6=borraMatriz(Ruta1sag6);
    Ruta1sag7=borraMatriz(Ruta1sag7);
    Ruta1sag8=borraMatriz(Ruta1sag8);
    return q_new;
```

```
}

matriz * propNoLinealRK_AnilloRuta2Tan(matriz *q_in, char
    *conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
   double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
   kth, long double Cp, long double rho,long double dn_dv
   ,long double P_pump,long double P_laser,long double
   lambda0,matriz *epsilon1, matriz *epsilon2, int
   iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
    long double L1=a;
    long double L2=b+c;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    assert(strcmp(conjugado_corto,conjugado_largo)==0);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
        ); // Creando matriz de salida
    long double complex q_2,q_3,q_4; // Parámetros
        complejos para propagación

    // Crea matrices para la propagación
    matriz *Ruta2tan1,*Ruta2tan2,*Ruta2tan3,*Ruta2tan4,*
        Ruta2tan5,*Ruta2tan6,*Ruta2tan7,*Ruta2tan8;
   // Ruta 2, caso tangencial
    Ruta2tan1=llenaMatriz(1,L2,0,1); // L2=b+c
```

```
Ruta2tan2=llenaMatriz(1,0, 1/f2t,1);
Ruta2tan3=nuevaMatriz(2,2); // Pendiente
Ruta2tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
    entrada)
Ruta2tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
    salida)
Ruta2tan6=nuevaMatriz(2,2);//Ruta2tan6 pendiente
Ruta2tan7=llenaMatriz(1,0, 1/f1t,1);
Ruta2tan8=llenaMatriz(1,L1,0,1); // L1=a

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta2tan3,1,delta2+creall
            (obtieneElemento(epsilon2,1,index2)),0,1);
                    llenaMatrizSinCrear(Ruta2tan6,1,
                        delta1+creall(obtieneElemento(
                        epsilon1,1,index1)),0,1);;
        if(elem(q_in,index1,index2)==0)
            {elem(q_new,index1,index2)=0;}
        else
        {
            fijaElemento(q_iter,1,1,obtieneElemento(
                q_in,index1,index2));
            //for(int iter=1;iter<iteraciones;iter++)
            for(int iter=iteraciones 1;iter>0;iter  )
            {
                 //int ahora=iter;
                //int siguiente=iter+1;
                int ahora=iteraciones iter;
                int siguiente=ahora+1;
                matriz * Prop1_piezas[]={Ruta2tan4,
                    Ruta2tan3,Ruta2tan2,Ruta2tan1}; //
                    FM1 a refracción en medio Kerr
                matriz * Prop1=variasMultMatriciales(
                    Prop1_piezas,4);
                long double complex q1_hold=
                    obtieneElemento(q_iter,1,ahora);
                q_2=prop_q(Prop1,q1_hold,1,n0);
                //q_3=propagacionKerr(pasosKerrRK,L,
                    n0,n2,w_pump,q_2,chi,kth,Cp,rho,
```

```
                dn_dv,P_pump,P_laser,lambda0);
            q_3=propagacionKerrRK(q_2,lambda0,L,
                n0,n2,P_laser,pasosKerrRK);
            matriz * Prop2_piezas[]={Ruta2tan8,
                Ruta2tan7,Ruta2tan6,Ruta2tan5}; //
                 5 a 8
            matriz * Prop2=variasMultMatriciales(
                Prop2_piezas,4);
            q_4=prop_q(Prop2,q_3,n0,1);
            fijaElemento(q_iter,1,siguiente,q_4);
            //printf("Actual: %f+i %f, siguiente:
                 %f+i %f",creal(obtieneElemento(
                q_iter,1,iter)),cimag(
                obtieneElemento(q_iter,1,iter)),
                creal(obtieneElemento(q_iter,1,
                iter+1)),cimag(obtieneElemento(
                q_iter,1,iter+1)));
            // Limpieza
            Prop1=borraMatriz(Prop1);
            Prop2=borraMatriz(Prop2);
            // Calculando spots y comparación
            w_iter_viejo=spot_q(q1_hold,1,lambda0
                );
            w_iter_nuevo=spot_q(q_4,1,lambda0);
            if(w_iter_viejo>0.2||w_iter_nuevo
                >0.2) break;  // Si el spot mide
                40 cm es demasiado
            diferencia=fabsl((w_iter_nuevo
                w_iter_viejo)/w_iter_nuevo);
            if(diferencia<=umbral) // Termina la
                iteración
            {
                fijaElemento(q_new,index1,index2,
                    obtieneElemento(q_iter,1,
                    siguiente));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new,index1,index2
                    ,0);
                termino=0;
            }
        }
    }
```

```
            if(termino==0)
                printf("Cavidad en anillo, Ruta2TanRK: No
                    cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
            else
                printf("Cavidad en anillo, Ruta2TanRK:
                    Cumple con el umbral de %Le para
                    epsilon1=%Le, epsilon2=%Le.\nError
                    relativo: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferencia);
        }
    }
    q_iter=borraMatriz(q_iter);
    Ruta2tan1=borraMatriz(Ruta2tan1);
    Ruta2tan2=borraMatriz(Ruta2tan2);
    Ruta2tan3=borraMatriz(Ruta2tan3);
    Ruta2tan4=borraMatriz(Ruta2tan4);
    Ruta2tan5=borraMatriz(Ruta2tan5);
    Ruta2tan6=borraMatriz(Ruta2tan6);
    Ruta2tan7=borraMatriz(Ruta2tan7);
    Ruta2tan8=borraMatriz(Ruta2tan8);
    return q_new;
}

matriz * propNoLinealRK_AnilloRuta2Sag(matriz *q_in, char
    *conjugado_corto,char *conjugado_largo,long double a,
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
    n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral)
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double w_iter_viejo=0, w_iter_nuevo=0,
        diferencia=0;
```

```
long double L1=a;
long double L2=b+c;
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
assert(strcmp(conjugado_corto,conjugado_largo)==0);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz * q_new=nuevaMatriz(q_in->filas,q_in->columnas
    ); // Creando matriz de salida
long double complex q_2,q_3,q_4; // Parámetros
    complejos para propagación

// Crea matrices para la propagación
matriz *Ruta2sag1,*Ruta2sag2,*Ruta2sag3,*Ruta2sag4,*
    Ruta2sag5,*Ruta2sag6,*Ruta2sag7,*Ruta2sag8;
// Ruta 2, caso sagital
Ruta2sag1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2sag2=llenaMatriz(1,0,1/f2s,1);
Ruta2sag3=nuevaMatriz(2,2); // Pendiente
Ruta2sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
    entrada)
Ruta2sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
    )
Ruta2sag6=nuevaMatriz(2,2);//Ruta2sag6 pendiente
Ruta2sag7=llenaMatriz(1,0,1/f1s,1);
Ruta2sag8=llenaMatriz(1,L1,0,1); // L1=a

// Ciclo
matriz * q_iter = nuevaMatriz(1,iteraciones);
for(int index1=1;index1<=epsilon1->columnas;index1++)
{
    for(int index2=1;index2<=epsilon2->columnas;
        index2++)
    {
        // Llenando matrices variables:
```

```
llenaMatrizSinCrear(Ruta2sag3,1,delta2+creall
    (obtieneElemento(epsilon2,1,index2)),0,1);
            llenaMatrizSinCrear(Ruta2sag6,1,
                delta1+creall(obtieneElemento(
                epsilon1,1,index1)),0,1);;
if(elem(q_in,index1,index2)==0)
    {elem(q_new,index1,index2)=0;}
else
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        q_in,index1,index2));
//for(int iter=1;iter<iteraciones;iter++)
    for(int iter=iteraciones 1;iter>0;iter  )
    {
         //int ahora=iter;
        //int siguiente=iter+1;
        int ahora=iteraciones iter;
        int siguiente=ahora+1;
        matriz * Prop1_piezas[]={Ruta2sag4,
            Ruta2sag3,Ruta2sag2,Ruta2sag1}; //
            FM1 a refracción en medio Kerr
        matriz * Prop1=variasMultMatriciales(
            Prop1_piezas,4);
        long double complex q1_hold=
            obtieneElemento(q_iter,1,ahora);
        q_2=prop_q(Prop1,q1_hold,1,n0);
        //q_3=propagacionKerr(pasosKerrRK,L,
            n0,n2,w_pump,q_2,chi,kth,Cp,rho,
            dn_dv,P_pump,P_laser,lambda0);
        q_3=propagacionKerrRK(q_2,lambda0,L,
            n0,n2,P_laser,pasosKerrRK);
        matriz * Prop2_piezas[]={Ruta2sag8,
            Ruta2sag7,Ruta2sag6,Ruta2sag5}; //
             5 a 8
        matriz * Prop2=variasMultMatriciales(
            Prop2_piezas,4);
        q_4=prop_q(Prop2,q_3,n0,1);
        fijaElemento(q_iter,1,siguiente,q_4);
        //printf("Actual: %f+i %f, siguiente:
             %f+i%f",creal(obtieneElemento(
            q_iter,1,iter)),cimag(
            obtieneElemento(q_iter,1,iter)),
            creal(obtieneElemento(q_iter,1,
            iter+1)),cimag(obtieneElemento(
            q_iter,1,iter+1)));
        // Limpieza
```

```
                Prop1=borraMatriz(Prop1);
                Prop2=borraMatriz(Prop2);
                // Calculando spots y comparación
                w_iter_viejo=spot_q(q1_hold,1,lambda0
                    );
                w_iter_nuevo=spot_q(q_4,1,lambda0);
                if(w_iter_viejo>0.2||w_iter_nuevo
                    >0.2) break; // Si el spot mide
                    40 cm es demasiado
                diferencia=fabsl((w_iter_nuevo
                    w_iter_viejo)/w_iter_nuevo)
                    *100.00;
                if(diferencia<=umbral) // Termina la
                    iteración
               {
                     fijaElemento(q_new,index1,index2,
                         obtieneElemento(q_iter,1,
                         siguiente));
                     termino=1;
                     break;
                }
                else
                {
                     fijaElemento(q_new,index1,index2
                         ,0);
                     termino=0;
                }
           }
      }
      if(termino==0)
           printf("Cavidad en anillo, Ruta2SagRK: No
               cumple con el umbral de %Le para
               epsilon1=%Le, epsilon2=%Le.\nError
               relativo: %Le\n",umbral,creall(
               obtieneElemento(epsilon1,1,index1)),
               creall(obtieneElemento(epsilon2,1,
               index2)),diferencia);
      else
           printf("Cavidad en anillo, Ruta2SagRK:
               Cumple con el umbral de %Le para
               epsilon1=%Le, epsilon2=%Le.\nError
               relativo: %Le\n",umbral,creall(
               obtieneElemento(epsilon1,1,index1)),
               creall(obtieneElemento(epsilon2,1,
               index2)),diferencia);
}
```

```
    }
    q_iter=borraMatriz(q_iter);
    Ruta2sag1=borraMatriz(Ruta2sag1);
    Ruta2sag2=borraMatriz(Ruta2sag2);
    Ruta2sag3=borraMatriz(Ruta2sag3);
    Ruta2sag4=borraMatriz(Ruta2sag4);
    Ruta2sag5=borraMatriz(Ruta2sag5);
    Ruta2sag6=borraMatriz(Ruta2sag6);
    Ruta2sag7=borraMatriz(Ruta2sag7);
    Ruta2sag8=borraMatriz(Ruta2sag8);
    return q_new;
}
```

no_linealRK.h

```
#ifndef NO_LINEALRK_H_INCLUDED
#define NO_LINEALRK_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include <assert.h>

// Resolver el sistema de ecuaciones diferenciales
    acopladas mostradas en el artículo de D. Huang.
// Emplea Runge Kutta de 4to orden. Se define a p=1/q,
    por lo que pReal=Re(p) y pImag=Im(p).
// xi queda definido por sqrt(1 P_laser/P_cr). Registrar
    la posición al momento de evaluar

// Ecuaciones diferenciales

long double dpReal(long double pReal, long double pImag,
    long double n0, long double n2, long double lambda0,
    long double P_laser);
long double dpImag(long double pReal, long double pImag,
    long double n0);
long double * pasoKerrRK(long double h, long double
    pReal_ini, long double pImag_ini, long double n0, long
     double n2, long double lambda0, long double P_laser);
long double complex propagacionKerrRK(long double complex
     q_in, long double lambda0, long double L, long double
     n0, long double n2, long double P_laser, int pasos);
matriz * propNoLinealRK_tanEM1(matriz *q_in, char *
```

```
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLinealRK_sagEM1(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLinealRK_tanEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLinealRK_sagEM2(matriz *q_in, char *
    conjugado_corto,char *conjugado_largo,long double L1,
    long double L2, long double L, long double f1, long
    double f2, long double n0, long double n2, long double
     w_pump,long double chi, long double kth, long double
    Cp, long double rho,long double dn_dv,long double
    P_pump,long double P_laser,long double lambda0,matriz
    *epsilon1, matriz *epsilon2, int iteraciones, long
    double umbral);
matriz * propNoLinealRK_AnilloRuta1Tan(matriz *q_in, char
     *conjugado_corto,char *conjugado_largo,long double a,
     long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLinealRK_AnilloRuta1Sag(matriz *q_in, char
     *conjugado_corto,char *conjugado_largo,long double a,
```

```
    long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLinealRK_AnilloRuta2Tan(matriz *q_in, char
     *conjugado_corto,char *conjugado_largo,long double a,
     long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);
matriz * propNoLinealRK_AnilloRuta2Sag(matriz *q_in, char
     *conjugado_corto,char *conjugado_largo,long double a,
     long double b, long double c, long double L, long
    double f1, long double f2, long double n0, long double
     n2, long double w_pump,long double chi, long double
    kth, long double Cp, long double rho,long double dn_dv
    ,long double P_pump,long double P_laser,long double
    lambda0,matriz *epsilon1, matriz *epsilon2, int
    iteraciones, long double umbral);


#endif // NO_LINEALRK_H_INCLUDED
```

A.5.6. termico.c y termico.h

termico.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include <complex.h>
#include "matrices.h"
#include "no_lineal.h"


// Rutina para resolver un sistema de n ecuaciones
    algebráicas de n incógnitas
// mediante Gauss Seidel. La matriz A representa los
    coeficientes constantes, X
```

```
// son las incógnitas y B son los resultados. La ecuación
     en forma matricial
// es [A]{X}={B} El algoritmo emplea relajación para
    mejorar la convergencia, esto es, cada
// valor nuevo de x calculado y modificado por un
    promedio ponderado de los resultados de
// la iteración anterior y actual: xi_nuevo=lambda*
    xi_nuevo'+(1 lambda)*xi_viejo.
// lambda se determina de forma empírica. Si lambda=0, no
     hay modificación al sistema.

// Función Gauss Seidel

long double * gaussSeidel(long double **A, long double *B
    , int n, int iteraciones, long double umbral, long
    double lambda)
{
    long double *X=(long double *)calloc(n,sizeof(long
        double));
    for(int i=0;i<n;i++)
    {
        long double dummy=A[i][i];
        for(int j=0;j<n;j++)
        {
            A[i][j]/=dummy; //A[i][j]=A[i][j]/A[i][i];
        }
        B[i]/=dummy; //B[i]=B[i]/A[i][i];
    }
    for(int i=0;i<n;i++)
    {
        long double suma=B[i];
        for(int j=0;j<n;j++)
        {
            if(i!=j)
                suma =A[i][j]*X[j];
        }
        X[i]=suma;
    }
    for(int paso=0;paso<iteraciones;paso++)
    {
        long double error=0;
        _Bool centinela=1;
        for(int i=0;i<n;i++)
        {
            long double viejo=X[i];
            long double suma=B[i];
```

```
                for(int j=0;j<n;j++)
                {
                    if(i!=j) suma=A[i][j]*X[j];
                }
                X[i]=lambda*suma+(1.0 lambda)*viejo;
                if(centinela==1&&X[i]!=0)
                {
                    error=100.0*fabsl((X[i] viejo)/X[i]);
                    if(error>umbral) centinela=0;
                }
            }
            if(centinela==1) break;
        }
        return X;
}

// Estructura para guardar la información relevante a la
    distribución de temperatura en un plano
// Almacena ejes cartesianos, valores del plano, y
    profundiad en el material usada.

typedef struct
{
    int N; // Órden de la matriz a usar.
    int paso; // Número de paso. dz=L/(pasos 1); xi(i)=dz
        *paso(i). exp( alpha*xi)
    long double deltaX; // Diferencial en el plano
        tangencial
    long double deltaY; // Dirección en el plano sagital
    long double * X; // Vector de posición tangencial
    long double * Y; // Vector de posición sagital
    long double * T;    // Mapa de datos
}distTempPlano;

// Estructura para guardar los coeficientes del ajuste
    parabólico de la distribución de temperatura
// Almacena los coeficientes de T=a0+a1*X^2+a2*Y^2 y la
    posición correspondiente (Pos=L/xi)
// Los vectores de los coeficientes tendrán un total de
    pasosTotales pasos.

typedef struct
{
    long double a0;
    long double a1;
```

```
    long double a2;
    int paso;
}ajusteTemperaturaPlano;

// Estructura que guarda coeficientes en forma de
    arreglos.
typedef struct
{
    long double *a0;
    long double *a1;
    long double *a2;
    int *paso;
}ajusteTemperaturaCristal;


distTempPlano * planoNuevo(int N, int paso, long double
    ancho, long double alto) // Crea estructura y almacena
     datos
{
    distTempPlano *dT =(distTempPlano*)calloc(1,sizeof(
        distTempPlano));
    dT>N=N;
    dT>paso=paso;
    dT>deltaX=ancho/(N 1);
    dT>deltaY=alto/(N 1);
    dT>X=(long double*)calloc(N,sizeof(long double));
    dT>Y=(long double*)calloc(N,sizeof(long double));
    dT>X[N 1]=ancho/2; // Se hace esa resta por cuestión
         de índices
    dT>Y[N 1]=alto/2;
    for(int i=N 1;i>0;i  ) // Creación de vectores de
        posición
    {
        dT>X[i 1]=dT>X[i] dT>deltaX;
        dT>Y[i 1]=dT>Y[i] dT>deltaY;
    }
    // Plano
    dT>T=(long double *)calloc(N*N,sizeof(long double));
         //Reserva para mapa de temperatura
    return dT;
}

distTempPlano * borraPlano(distTempPlano * plano) //
    Borra estructura de plano
{
    assert(plano);
```

```
    // Libera datos
    assert(plano->T);
    assert(plano->X);
    assert(plano->Y);
    free(plano->X);
    free(plano->Y);
    free(plano->T);
    plano->T=NULL;
    plano->X=NULL;
    plano->Y=NULL;
    // Libera a plano
    free(plano);
    plano=NULL;
    return plano;
}

distTempPlano * distTempPlanoCristal(int N, long double
    ancho, long double alto, long double deltaZ, long
    double P_pump, long double chi, long double L, long
    double kth, long double Cp, long double rho, long
    double w_pump_t,long double w_pump_s, long double
    alpha,\
                                        int paso, int
                                            iteraciones,
                                            long double
                                            umbral, long
                                            double
                                            lambda_relax)
{
    assert((N%2==0)!=1);
    distTempPlano *dT=planoNuevo(N,paso,ancho,alto);
    long double xi=(long double)dT->paso*deltaZ;
        // Llena condiciones de frontera
        dT->T[0+0]=dT->T[N-1]=dT->T[(N-1)*N]=dT->T[(N-1)*
            N+N-1]=20.0+273.15; // Valores no usados en el
              algoritmo
        for(int i=1;i<N-1;i++)
        {
                dT->T[0*N+i]=20.0+273.15; // Frontera
                    izquierda
                dT->T[i*N+0]=20.0+273.15; // Frontera
                    inferior
                dT->T[(N-1)*N+i]=20.0+273.15; // Frontera
                     derecha
                dT->T[i*N+N-1]=20.0+273.15; // Frontera
                    superior
```

```
}
// Iterando
for(int paso=0;paso<iteraciones;paso++)
{
    _Bool centinela=1;
    for(int j=1;j<N-1;j++)
    {
        for(int i=1;i<N-1;i++)
        {
            long double func=(2.0*chi*P_pump)/(L
                *Cp*rho*kth*M_PI*w_pump_t*w_pump_s
                )*expl(-2.0*powl(dT->X[i],2)/powl(
                w_pump_t,2)-2.0*powl(dT->Y[j],2)/
                powl(w_pump_s,2))*expl(-alpha*xi);
            long double viejo=dT->T[i*N+j];
            dT->T[i*N+j]=(powl(dT->deltaY,2)*(dT
                ->T[(i+1)*N+j]+dT->T[(i-1)*N+j])+
                powl(dT->deltaX,2)*(dT->T[i*N+(j
                +1)]+dT->T[i*N+(j-1)])-powl(dT->
                deltaX,2)*powl(dT->deltaY,2)*func)
                /(2*(powl(dT->deltaX,2)+powl(dT->
                deltaY,2)));
            dT->T[i*N+j]=lambda_relax*dT->T[i*N+j
                ]+(1.0-lambda_relax)*viejo;
            if(centinela==1&&dT->T[i*N+j]!=0)
            {
                long double error=fabsl((dT->T[i*
                    N+j]-viejo)/dT->T[i*N+j])
                    *100.0;
                if(error>umbral) centinela=0;
            }
        }
    }
    if(centinela==1)
        {break;}
}

// Obteniento deltaT
int centro=(N+1)/2;
long double Tmax=dT->T[(centro-1)*N+centro-1]; //
    Centro ubicado en N/2, N/2. Se resta 1 por
    cuestiones de índices.
for(int j=N-1;j>=0;j--)
{
    for(int i=N-1;i>=0;i--)
    {
```

```
                dT->T[i*N+j]=dT->T[i*N+j] Tmax;
            }
        }
    //}
    // Imprimiendo valores
    /*for(int j=1;j<N-1;j++)
    {
        for(int i=1;i<N-1;i++)
        {
            printf("Delta T(%i,%i)= %Lf [K]\n",i,j,dT->T[
                i*N+j]);
        }
    }*/
    return dT;
}

ajusteTemperaturaPlano * ajusteCuadraticoPlano(
    distTempPlano * plano, int iteraciones, long double
    umbral, long double lambda_relax)
{
    // Creación de estructura
    ajusteTemperaturaPlano *ajuste =(
        ajusteTemperaturaPlano*)calloc(1,sizeof(
        ajusteTemperaturaPlano));
    // Cálculo de coeficientes
    int N=plano->N;
    long double AA,BB,CC,DD,EE,FF,GG,HH,II; //
    AA=0.0, BB=0.0, CC=0.0, DD=0.0, EE=0.0, FF=0.0, GG
        =0.0, HH=0.0, II=0.0;
    int i=0,j=0,conteo=0;
    for(j=1;j<N-1;j++)
    {
        for(i=1;i<N-1;i++)
        {
            AA++;
            BB+=powl(plano->X[i],2);
            CC+=powl(plano->Y[j],2);
            DD+=plano->T[i*N+j];
            EE+=powl(plano->X[i],4);
            FF+=powl(plano->X[i],2)*powl(plano->Y[j],2);
            GG+=plano->T[i*N+j]*powl(plano->X[i],2);
            HH+=powl(plano->Y[j],4);
            II+=plano->T[i*N+j]*powl(plano->Y[j],2);
            conteo++;
        }
    }
```

```
// Construyendo matriz

//int iteraciones=100000;
//long double umbral=0.1e 6; // Umbral en porcentaje
// Creación de matriz A
int orden=3; // Matriz de ajuste
long double **A=(long double**)calloc(orden,sizeof(
    long double *));
for(i=0;i<orden;i++)
    A[i]=(long double *)calloc(orden,sizeof(long
        double));
// Llenado de A
A[0][0]=AA;
A[0][1]=BB;
A[0][2]=CC;
A[1][0]=BB;
A[1][1]=EE;
A[1][2]=FF;
A[2][0]=CC;
A[2][1]=FF;
A[2][2]=HH;

// Creación de B
long double *B=(long double *)calloc(orden,sizeof(
    long double));

// Llenado de B
B[0]=DD;
B[1]=GG;
B[2]=II;

long double *X=gaussSeidel(A,B,orden,iteraciones,
    umbral,lambda_relax);
/*printf("Soluciones\n");
printf("a0=%Lf\n",X[0]);
printf("a1=%Lf\n",X[1]);
printf("a2=%Lf\n",X[2]);*/

/*FILE *archivo;
archivo = fopen("Ajuste cuadratico.csv","w");
fprintf(archivo, "Coeficiente,Valor,\n");
for(i=0;i<3;i++)
{
    fprintf(archivo, "a %i,%.15Le,\n",i,X[i]);
}
```

```
    fclose(archivo);*/
    ajuste->a0=X[0];
    ajuste->a1=X[1];
    ajuste->a2=X[2];
    ajuste->paso=plano->paso;

    //Limpieza
    for(int i=orden-1;i>=0; i)
        free(A[i]);
    free(A);
    free(B);
    free(X);
    return ajuste;
}

// Ajuste cuadrático ponderado a T original
ajusteTemperaturaPlano * ajusteCuadraticoPlanoPonderado(
    distTempPlano * plano, int iteraciones, long double
    umbral, long double lambda_relax)
{
    // Creación de estructura
    ajusteTemperaturaPlano *ajustePonderado =(
        ajusteTemperaturaPlano*)calloc(1,sizeof(
        ajusteTemperaturaPlano));
    // Cálculo de coeficientes
    int N=plano->N;
    long double AA,BB,CC,DD,EE,FF,GG,HH,II; //
    AA=0.0, BB=0.0, CC=0.0, DD=0.0, EE=0.0, FF=0.0, GG
        =0.0, HH=0.0, II=0.0;
    int i=0,j=0,conteo=0;
    for(j=1;j<N-1;j++)
    {
        for(i=1;i<N-1;i++)
        {
            AA+=plano->T[i*N+j];
            BB+=powl(plano->X[i],2)*plano->T[i*N+j];
            CC+=powl(plano->Y[j],2)*plano->T[i*N+j];
            DD+=plano->T[i*N+j]*plano->T[i*N+j];
            EE+=powl(plano->X[i],4)*plano->T[i*N+j];
            FF+=powl(plano->X[i],2)*powl(plano->Y[j],2)*
                plano->T[i*N+j];
            GG+=plano->T[i*N+j]*powl(plano->X[i],2)*plano
                ->T[i*N+j];
            HH+=powl(plano->Y[j],4)*plano->T[i*N+j];
            II+=plano->T[i*N+j]*powl(plano->Y[j],2)*plano
                ->T[i*N+j];
```

```
            conteo++;
        }
    }

    // Construyendo matriz

    //int iteraciones=100000;
    //long double umbral=0.1e 6; // Umbral en porcentaje
    // Creación de matriz A
    int orden=3; // Matriz de ajuste
    long double **A=(long double**)calloc(orden,sizeof(
        long double *));
    for(i=0;i<orden;i++)
        A[i]=(long double *)calloc(orden,sizeof(long
            double));
    // Llenado de A
    A[0][0]=AA;
    A[0][1]=BB;
    A[0][2]=CC;
    A[1][0]=BB;
    A[1][1]=EE;
    A[1][2]=FF;
    A[2][0]=CC;
    A[2][1]=FF;
    A[2][2]=HH;

    // Creación de B
    long double *B=(long double *)calloc(orden,sizeof(
        long double));

    // Llenado de B
    B[0]=DD;
    B[1]=GG;
    B[2]=II;

    long double *X=gaussSeidel(A,B,orden,iteraciones,
        umbral,lambda_relax);

    ajustePonderado->a0=X[0];
    ajustePonderado->a1=X[1];
    ajustePonderado->a2=X[2];
    ajustePonderado->paso=plano->paso;

    //Limpieza
    for(int i=orden 1;i>=0; i)
        free(A[i]);
```

```
    free(A);
    free(B);
    free(X);
    return ajustePonderado;
}

ajusteTemperaturaCristal * vectorAjusteCristal(long
    double complex qInTan, long double complex qInSag,
    long double alpha, long double lambdaIn, long double
    nLin, \
    long double Power, long double chi, long double L,
        long double kth, long double Cp, long double rho,
        long double dn_dT, \
    long double ancho, long double alto, int iteraciones,
         int pasos, long double umbral, long double
        lambda_relax, int N)
    {
        long double deltaZ=L/(pasos-1); // Diferencia de
            distancia en la dirección de propagación
        long double wTan=spot_q(qInTan,nLin,lambdaIn);
        long double wSag=spot_q(qInSag,nLin,lambdaIn);

        //Creación de estructura de salida
        ajusteTemperaturaCristal *ajuste=(
            ajusteTemperaturaCristal*)calloc(1,sizeof(
            ajusteTemperaturaCristal));
        ajuste->a0=(long double*)calloc(pasos,sizeof(long
             double));
        ajuste->a1=(long double*)calloc(pasos,sizeof(long
             double));
        ajuste->a2=(long double*)calloc(pasos,sizeof(long
             double));
        ajuste->paso=(int*)calloc(pasos,sizeof(int));

        // Llenado de paso[]
        for(int indice=pasos-1;indice>=0;indice--)
             ajuste->paso[indice]=indice;

        long double complex *qPropTan = (long double
            complex*)calloc(pasos,sizeof(long double
            complex));
        long double complex *qPropSag = (long double
            complex*)calloc(pasos,sizeof(long double
            complex));
        long double complex At, Bt, Ct, Dt;
        long double complex As, Bs, Cs, Ds;
```

```
assert(qPropTan);
assert(qPropSag);
matriz *MTan, *MSag;
MTan=nuevaMatriz(2,2);
MSag=nuevaMatriz(2,2);
// Cálculo inicial
qPropTan[0]=qInTan;
qPropSag[0]=qInSag;
ajuste->paso[0]=0;
long double n0Prop=nLin;
for (int i=0;i<pasos;i++)
{
    // Cálculo de spots
    long double w_pump_t=spot_q(qPropTan[i],
        n0Prop,lambdaIn);
    long double w_pump_s=spot_q(qPropSag[i],
        n0Prop,lambdaIn);
    // Cálculo de plano
    distTempPlano *plano=distTempPlanoCristal(N,
        ancho,alto,deltaZ,Power,chi,L,kth,Cp,rho,
        w_pump_s,w_pump_t,alpha,ajuste->paso[i],
        iteraciones,umbral,lambda_relax);
    // Ajuste de plano
    ajusteTemperaturaPlano *ajustePlano=
        ajusteCuadraticoPlanoPonderado(plano,
        iteraciones,umbral,lambda_relax);
    ajuste->a0[i]=ajustePlano->a0;
    ajuste->a1[i]=ajustePlano->a1;
    ajuste->a2[i]=ajustePlano->a2;
    ajuste->paso[i]=ajustePlano->paso;
    n0Prop=nLin+ajuste->a0[i]*dn_dT;
    // Formación de matrices
    // Factor de parábola=1/h^2
    long double parabola1=2*n0Prop*ajuste->a1[i]*
        dn_dT;
    long double parabola2=2*n0Prop*ajuste->a2[i]*
        dn_dT;
    // Coeficientes
    At=1;
    Bt=deltaZ/n0Prop;
    Ct= n0Prop*parabola1*deltaZ;
    Dt=1;
    As=1;
    Bs=deltaZ/n0Prop;
    Cs= n0Prop*parabola2*deltaZ;
    Ds=1;
```

```
            llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
            llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
            if(i+1<pasos)
            {
                qPropTan[i+1]=prop_q(MTan,qPropTan[i],
                    n0Prop,n0Prop);
                qPropSag[i+1]=prop_q(MSag,qPropSag[i],
                    n0Prop,n0Prop);
            }
            free(ajustePlano);
            plano=borraPlano(plano);
        }
        //Limpieza
        borraMatriz(MTan);
        borraMatriz(MSag);
        free(qPropTan);
        free(qPropSag);
        printf("Terminó ajuste térmico\n");
        return ajuste;
    }


void escribePlanoArchivoCompleto(distTempPlano * plano,
    char *nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int i,j;
    fprintf(archivo, " ,Eje tangencial [m],");
    for(i=0;i<=plano>N 1;i++)
        fprintf(archivo, "%.15Le,",plano>X[i]);
    fprintf(archivo, "\nEje sagital [m],\n");
    for (j=0;j<=plano>N 1;j++)
    {
        fprintf(archivo,"%.15Le, ,",plano>Y[j]);
        for(i=0;i<=plano>N 1;i++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le,", plano>T[i*plano
                >N+j]);
        }
        fprintf(archivo,"\n");
    }
    fclose(archivo);
}
```

```
void escribePlanoArchivo(distTempPlano * plano, char *
    nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int i,j;
    for (j=1;j<plano->N 1;j++)
    {
        for(i=1;i<plano->N 1;i++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le,", plano->T[i*plano
                ->N+j]);
        }
        fprintf(archivo,"\n");
    }
    fclose(archivo);
}

void escribePlanoArchivoOrigin(distTempPlano * plano,
    char *nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    int i,j;
    for (j=1;j<plano->N 1;j++)
    {
        for(i=1;i<plano->N 1;i++)
        {
            //Imprime el elemento de punto flotante
            fprintf(archivo, "%.15Le,%.15Le,%.15Le\n",
                plano->X[i], plano->Y[j],plano->T[i*plano
                ->N+j]);
        }
    }
    fclose(archivo);
}

void escribeCoeficientesAjuste(ajusteTemperaturaCristal *
    ajuste, int pasos, char *nombre)
{
    FILE *archivo;
    archivo = fopen(nombre,"w");
    fprintf(archivo, "Paso,a0,a1,a2\n");
    int i;
    for(i=0;i<pasos;i++)
```

```
    {
        //Imprime el elemento de punto flotante
        fprintf(archivo, "%i,%.15Le,%.15Le,%.15Le\n",
            ajuste->paso[i],ajuste->a0[i],ajuste->a2[i],
            ajuste->a2[i]);
    }
    fclose(archivo);
}
```

termico.h

```
#ifndef TERMICO_H_INCLUDED
#define TERMICO_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include <complex.h>
#include "matrices.h"
#include "no_lineal.h"

// Rutina para resolver un sistema de n ecuaciones
    algebráicas de n incógnitas
// mediante Gauss Seidel. La matriz A representa los
    coeficientes constantes, X
// son las incógnitas y B son los resultados. La ecuación
     en forma matricial
// es [A]{X}={B} El algoritmo emplea relajación para
    mejorar la convergencia, esto es, cada
// valor nuevo de x calculado y modificado por un
    promedio ponderado de los resultados de
// la iteración anterior y actual: xi_nuevo=lambda*
    xi_nuevo'+(1 lambda)*xi_viejo.
// lambda se determina de forma empírica. Si lambda=0, no
     hay modificación al sistema.

// Función Gauss Seidel

long double * gaussSeidel(long double **A, long double *B
    , int n, int iteraciones, long double umbral, long
    double lambda);
// Estructura para guardar la información relevante a la
    distribución de temperatura en un plano
// Almacena ejes cartesianos, valores del plano, y
    profundiad en el material usada.
```

```
typedef struct
{
    int N; // Órden de la matriz a usar.
    int paso; // Número de paso. dz=L/(pasos 1); xi(i)=dz
        *paso(i). exp( alpha*xi)
    long double deltaX; // Diferencial en el plano
        tangencial
    long double deltaY; // Dirección en el plano sagital
    long double * X; // Vector de posición tangencial
    long double * Y; // Vector de posición sagital
    long double * T;   // Mapa de datos
}distTempPlano;

// Estructura para guardar los coeficientes del ajuste
    parabólico de la distribución de temperatura
// Almacena los coeficientes de T=a0+a1*X^2+a2*Y^2 y la
    posición correspondiente (Pos=L/xi)
// Los vectores de los coeficientes tendrán un total de
    pasosTotales pasos.

typedef struct
{
    long double a0;
    long double a1;
    long double a2;
    int paso;
}ajusteTemperaturaPlano;

// Estructura que guarda coeficientes en forma de
    arreglos.
typedef struct
{
    long double *a0;
    long double *a1;
    long double *a2;
    int *paso;
}ajusteTemperaturaCristal;


distTempPlano * planoNuevo(int N, int paso, long double
    ancho, long double alto); // Crea estructura y
    almacena datos
distTempPlano * borraPlano(distTempPlano * plano); //
    Borra estructura de plano
```

```
distTempPlano * distTempPlanoCristal(int N, long double
    ancho, long double alto, long double deltaZ, long
    double P_pump, long double chi, long double L, long
    double kth, long double Cp, long double rho, long
    double w_pump_t,long double w_pump_s, long double
    alpha,\
                                        int paso, int
                                            iteraciones,
                                            long double
                                            umbral, long
                                            double
                                            lambda_relax);
ajusteTemperaturaPlano * ajusteCuadraticoPlano(
    distTempPlano * plano, int iteraciones, long double
    umbral, long double lambda_relax);
ajusteTemperaturaPlano * ajusteCuadraticoPlanoPonderado(
    distTempPlano * plano, int iteraciones, long double
    umbral, long double lambda_relax);
ajusteTemperaturaCristal * vectorAjusteCristal(long
    double complex qInTan, long double complex qInSag,
    long double alpha, long double lambdaIn, long double
    nLin, \
     long double Power, long double chi, long double L,
         long double kth, long double Cp, long double rho,
         long double dn_dT, \
     long double ancho, long double alto, int iteraciones,
          int pasos, long double umbral, long double
         lambda_relax, int N);
void escribePlanoArchivoCompleto(distTempPlano * plano,
    char *nombre);
void escribePlanoArchivo(distTempPlano * plano, char *
    nombre);
void escribePlanoArchivoOrigin(distTempPlano * plano,
    char *nombre);
void escribeCoeficientesAjuste(ajusteTemperaturaCristal *
    ajuste, int pasos, char *nombre);

#endif // TERMICO_H_INCLUDED
```

A.5.7. no_linealMatAstTerm.c y no_linealMatAstTerm.h

no_linealMatAstTerm.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

```
#include <complex.h>
#include "matrices.h"
#include "lineal.h"
#include "termico.h"
#include "no_lineal.h"
#include "error_iteraciones.h"

// Programa para la propagación no lineal considerando
    haces astigmáticos para efecto Kerr y térmico

// Constantes para propagación en medio Kerr

     //int pasosKerr=1000;

// Estructura para sacar dos números complejos de la
    propagación Kerr acoplada

typedef struct
{
     long double complex qTan;
     long double complex qSag;
}propAstigmatica;

// Original
propAstigmatica * propagacionKerrAcoplada(int pasos, long
     double L, long double n0,long double n2, long double
    complex qInTan, \
                                                long double
                                                    complex
                                                    qInSag,
                                                    long
                                                    double chi
                                                    , long
                                                    double kth
                                                    , long
                                                    double Cp,
                                                     \
                                                long double
                                                    rho, long
                                                    double
                                                    dn_dv,long
                                                     double
                                                    P_laser,
                                                    long
                                                    double
                                                    lambda0, \
```

```
                                                    ajusteTemperaturaCristal
                                                        *
                                                    vectorPlano
                                                    , _Bool
                                                    ladoBombeo
                                                    )
{
        propAstigmatica *salida=(propAstigmatica*)calloc
            (1,sizeof(propAstigmatica)); // Reserva de
            espacio
        long double deltaZ=L/(pasos-1); // Diferencia de
            distancia en la dirección de propagación
        long double wTan=spot_q(qInTan,n0,lambda0);
        long double wSag=spot_q(qInSag,n0,lambda0);

        long double complex *qPropTan = (long double
            complex*)calloc(pasos,sizeof(long double
            complex));
        long double complex *qPropSag = (long double
            complex*)calloc(pasos,sizeof(long double
            complex));
        long double complex At, Bt, Ct, Dt;
        long double complex As, Bs, Cs, Ds;
        assert(qPropTan);
        assert(qPropSag);
        matriz *MTan, *MSag;
        MTan=nuevaMatriz(2,2);
        MSag=nuevaMatriz(2,2);
        // Cálculo inicial
        qPropTan[0]=qInTan;
        qPropSag[0]=qInSag;
        long double n0Prop=n0;
        // Identificar si se propaga desde el lado
            bombeado o no, cambiar coeficientes
        // de ajuste acorde a esto.
        if(ladoBombeo==1)
        {
            for (int i=0;i<pasos;i++)
            {
                // Cálculo de spots
                long double wTan=spot_q(qPropTan[i],
                    n0Prop,lambda0);
                long double wSag=spot_q(qPropSag[i],
                    n0Prop,lambda0);
                // Cálculo de plano
                //n0Prop=n0+vectorPlano->a0[i]*dn_dv+n2*
```

```
                P_laser/(M_PI*wTan*wSag)*3.0/4.0; //
                    Factores lineal, térmico y Kerr
            n0Prop=n0; // Descartados términos
                lineales por ser pequeños.
            // Formación de matrices
            // Factor de parábola=1/h^2
            long double parabola1 = 2.0*vectorPlano >
                a1[i]*dn_dv/n0Prop+(n2*P_laser)/(
                n0Prop*M_PI*powl(wTan,3)*wSag);
            long double parabola2 = 2.0*vectorPlano >
                a2[i]*dn_dv/n0Prop+(n2*P_laser)/(
                n0Prop*M_PI*powl(wSag,3)*wTan);
            // Coeficientes
            At=1;
            Bt=deltaZ/n0Prop;
            Ct= n0Prop*parabola1*deltaZ;
            Dt=1;
            As=1;
            Bs=deltaZ/n0Prop;
            Cs= n0Prop*parabola2*deltaZ;
            Ds=1;
            llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
            llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
            if(i+1<pasos)
            {
                qPropTan[i+1]=prop_q(MTan,qPropTan[i
                    ],n0Prop,n0Prop);
                qPropSag[i+1]=prop_q(MSag,qPropSag[i
                    ],n0Prop,n0Prop);
            }
        }
    }
    else
    {
        int indiceTermico=pasos-1;
        for (int i=0;i<pasos;i++)
        {
            // Cálculo de spots
            //long double w_pump_t=spot_q(qPropTan[i
                ],n0Prop,lambda0);
            //long double w_pump_s=spot_q(qPropSag[i
                ],n0Prop,lambda0);
            long double wTan=spot_q(qPropTan[i],
                n0Prop,lambda0);
            long double wSag=spot_q(qPropSag[i],
                n0Prop,lambda0);
```

```
                    // Cálculo de plano
                    n0Prop=n0;
                    // Formación de matrices
                    // Factor de parábola=1/h^2
                    long double parabola1 = 2.0*vectorPlano >
                        a1[indiceTermico]*dn_dv/n0Prop+(n2*
                        P_laser)/(n0Prop*M_PI*powl(wTan,3)*
                        wSag);
                    long double parabola2 = 2.0*vectorPlano >
                        a2[indiceTermico]*dn_dv/n0Prop+(n2*
                        P_laser)/(n0Prop*M_PI*powl(wSag,3)*
                        wTan);
                    // Coeficientes
                    At=1;
                    Bt=deltaZ/n0Prop;
                    Ct= n0Prop*parabola1*deltaZ;
                    Dt=1;
                    As=1;
                    Bs=deltaZ/n0Prop;
                    Cs= n0Prop*parabola2*deltaZ;
                    Ds=1;
                    llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
                    llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
                    if(i+1<pasos)
                    {
                        qPropTan[i+1]=prop_q(MTan,qPropTan[i
                            ],n0Prop,n0Prop);
                        qPropSag[i+1]=prop_q(MSag,qPropSag[i
                            ],n0Prop,n0Prop);
                    }
                    indiceTermico   ;
                }
            }
            salida >qTan=qPropTan[pasos 1];
            salida >qSag=qPropSag[pasos 1];
            //Limpieza
            borraMatriz(MTan);
            borraMatriz(MSag);
            free(qPropTan);
            free(qPropSag);
            return salida;
        }


// Propagación no lineal (matrices)
// Termina el lazo si alcanza el umbral solicitado
```

```
// Para epsilon1 y epsilon2

// EM1   Busca que sean estables en ambos planos.

matriz ** propNoLinealEM1Astigmatico(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser,\
                                        long double lambda0,
                                            matriz *epsilon1,
                                            matriz *epsilon2,
                                            int iteraciones,
                                            int pasosKerr,
                                            long double umbral
                                            )
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spotsTan, *spotsSag;
    int * iteracionActualTan, *iteracionActualSag;
    int cuenta=0;
    char tipoTan[]="EM1Tan_Acoplado", tipoSag[]="
        EM1Sag_Acoplado";
    char subCarpetaTan[]="EM1Tan_matAcop";
    char subCarpetaSag[]="EM1Sag_matAcop";
    long double wIterTanViejo=0, wIterTanNuevo=0,
        wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
         diferenciaSag=0; // Umbral no porcentual
```

```
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cos(angulos[0]);
f2t=f2*cos(angulos[1]);
f1s=f1/cos(angulos[0]);
f2s=f2/cos(angulos[1]);

matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
    // Espacio para dos matrices
q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
     // Creando matriz tangencial
q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
     // Creando matriz tangencial
long double complex *qPropTan=(long double complex*)
    calloc(6,sizeof(long double complex)); //
    Parámetros para propagación
long double complex *qPropSag=(long double complex*)
    calloc(6,sizeof(long double complex));

// Crea matrices para la propagación

matriz *EM1tan1,*EM1tan2,*EM1tan3,*EM1tan4,*EM1tan5,*
    EM1tan6,*EM1tan7,*EM1tan8,*EM1tan9,*EM1tan10,*
    EM1tan11,*EM1tan12,*EM1tan13,*EM1tan14,*EM1tan15,*
    EM1tan16,*EM1tan17;
matriz *EM1sag1,*EM1sag2,*EM1sag3,*EM1sag4,*EM1sag5,*
    EM1sag6,*EM1sag7,*EM1sag8,*EM1sag9,*EM1sag10,*
    EM1sag11,*EM1sag12,*EM1sag13,*EM1sag14,*EM1sag15,*
    EM1sag16,*EM1sag17;
// EM1, Caso tangencial
EM1tan1=llenaMatriz(1,L1,0,1);
EM1tan2=llenaMatriz(1,0,1/f1t,1);
EM1tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1tan4=llenaMatriz(n0,0,0,1/n0);
EM1tan5=llenaMatriz(1/n0,0,0,n0);
EM1tan6=nuevaMatriz(2,2);//EM1tan5 pendiente
EM1tan7=llenaMatriz(1,0,1/f2t,1);
```

```
EM1tan8=llenaMatriz(1,L2,0,1);
EM1tan9=llenaMatriz(1,0,0,1);
EM1tan10=llenaMatriz(1,L2,0,1);
EM1tan11=llenaMatriz(1,0,1/f2t,1);
EM1tan12=nuevaMatriz(2,2);      //EM1tan11 pendiente
EM1tan13=llenaMatriz(n0,0,0,1/n0);
EM1tan14=llenaMatriz(1/n0,0,0,n0);
EM1tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1tan16=llenaMatriz(1,0,1/f1t,1);
EM1tan17=llenaMatriz(1,L1,0,1);
// EM1, Caso saggencial
EM1sag1=llenaMatriz(1,L1,0,1);
EM1sag2=llenaMatriz(1,0,1/f1s,1);
EM1sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1sag4=llenaMatriz(1,0,0,1);
EM1sag5=llenaMatriz(1,0,0,1);
EM1sag6=nuevaMatriz(2,2);//EM1sag5 pendiente
EM1sag7=llenaMatriz(1,0,1/f2s,1);
EM1sag8=llenaMatriz(1,L2,0,1);
EM1sag9=llenaMatriz(1,0,0,1);
EM1sag10=llenaMatriz(1,L2,0,1);
EM1sag11=llenaMatriz(1,0,1/f2s,1);
EM1sag12=nuevaMatriz(2,2);      //EM1sag11 pendiente
EM1sag13=llenaMatriz(1,0,0,1);
EM1sag14=llenaMatriz(1,0,0,1);
EM1sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1sag16=llenaMatriz(1,0,1/f1s,1);
EM1sag17=llenaMatriz(1,L1,0,1);


// Ciclo
matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
    para tangencial, 2 para sagital
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
     for(int index2=1;index2<=epsilon2 >columnas;
         index2++)
     {
          spotsTan=(long double*)calloc(1,sizeof(long
              double));
          spotsSag=(long double*)calloc(1,sizeof(long
              double));
          iteracionActualTan=(int *)calloc(1,sizeof(int
```

```
    ));
iteracionActualSag=(int *)calloc(1,sizeof(int
    ));
cuenta=1;
iteracionActualTan[0]=0;
iteracionActualSag[0]=0;
spotsTan[0]=spot_q(obtieneElemento(qInTan,
    index1,index2),1,lambda0);
spotsSag[0]=spot_q(obtieneElemento(qInSag,
    index1,index2),1,lambda0);
// Llenando matrices variables:
llenaMatrizSinCrear(EM1tan3,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1tan6,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1tan12,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1tan15,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1sag3,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1sag6,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag12,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag15,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);

if((elem(qInTan,index1,index2)!=0)&&(elem(
    qInSag,index1,index2)!=0)) // Si no son
    cero, realiza rutina
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        qInTan,index1,index2));
    fijaElemento(q_iter,2,1,obtieneElemento(
        qInSag,index1,index2));
    escribeCSV=1;
    for(;;)
    {
        int ahora=1;
        int siguiente=2;
        matriz * Prop1Tan_piezas[]={EM1tan4,
            EM1tan3,EM1tan2,EM1tan1};
        matriz * Prop1Tan=
            variasMultMatriciales(
            Prop1Tan_piezas,4);
```

```
matriz * Prop1Sag_piezas[]={EM1sag4,
    EM1sag3,EM1sag2,EM1sag1};
matriz * Prop1Sag=
    variasMultMatriciales(
    Prop1Sag_piezas,4);
// Almacenando primer q
qPropTan[0]=obtieneElemento(q_iter,1,
    ahora);
qPropSag[0]=obtieneElemento(q_iter,2,
    ahora);
// Cálculo de q2
qPropTan[1]=prop_q(Prop1Tan,qPropTan
    [0],1,n0);
qPropSag[1]=prop_q(Prop1Sag,qPropSag
    [0],1,n0);
// Cálculo de q3
propAstigmatica *q3=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[1],qPropSag[1],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,1);
qPropTan[2]=q3>qTan;
qPropSag[2]=q3>qSag;
free(q3);
// Más matrices
matriz * Prop2Tan_piezas[]={EM1tan13,
    EM1tan12,EM1tan11,EM1tan10,EM1tan9
    ,EM1tan8,EM1tan7,EM1tan6,EM1tan5};
     // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,9);
matriz * Prop2Sag_piezas[]={EM1sag13,
    EM1sag12,EM1sag11,EM1sag10,EM1sag9
    ,EM1sag8,EM1sag7,EM1sag6,EM1sag5};
     // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,9);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,n0);
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,n0);
// Propagación de regreso Kerr,
    cálculo q5
```

```
propAstigmatica *q5=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[3],qPropSag[3],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,0);
qPropTan[4]=q5->qTan;
qPropSag[4]=q5->qSag;
free(q5);
// Más matrices
matriz * Prop3Tan_piezas[]={EM1tan17,
    EM1tan16,EM1tan15,EM1tan14}; // 14
     a 17
matriz * Prop3Tan=
    variasMultMatriciales(
    Prop3Tan_piezas,4);
matriz * Prop3Sag_piezas[]={EM1sag17,
    EM1sag16,EM1sag15,EM1sag14}; // 14
     a 17
matriz * Prop3Sag=
    variasMultMatriciales(
    Prop3Sag_piezas,4);
// Cálculo de q6
qPropTan[5]=prop_q(Prop3Tan,qPropTan
    [4],n0,1);
qPropSag[5]=prop_q(Prop3Sag,qPropSag
    [4],n0,1);
// Guarda elementos
fijaElemento(q_iter,1,ahora,qPropTan
    [5]);
fijaElemento(q_iter,2,ahora,qPropSag
    [5]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop3Tan=borraMatriz(Prop3Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);
Prop3Sag=borraMatriz(Prop3Sag);

// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[5],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
```

```
    lambda0);
wIterSagNuevo=spot_q(qPropSag[5],1,
    lambda0);

if(cuenta>iteraciones)  break; // Por
    si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_sptTan, *tmp_sptSag, *
    tmp_iterTan, *tmp_iterSag;
tmp_sptTan=(long double*)realloc(
    spotsTan,cuenta*sizeof(long double
    ));
tmp_sptSag=(long double*)realloc(
    spotsSag,cuenta*sizeof(long double
    ));
tmp_iterTan=(int *)realloc(
    iteracionActualTan,cuenta*sizeof(
    int));
tmp_iterSag=(int *)realloc(
    iteracionActualSag,cuenta*sizeof(
    int));
if(tmp_sptTan!=NULL && tmp_sptSag!=
    NULL && tmp_iterTan!=NULL &&
    tmp_iterSag!=NULL)
{
     spotsTan=tmp_sptTan;
     spotsSag=tmp_sptSag;
     iteracionActualTan=tmp_iterTan;
     iteracionActualSag=tmp_iterSag;
     spotsTan[cuenta-1]=wIterTanNuevo;
     spotsSag[cuenta-1]=wIterSagNuevo;
     iteracionActualTan[cuenta-1]=
         cuenta-1;
     iteracionActualSag[cuenta-1]=
         cuenta-1;
}
else
{
     reallocFallido=1;
     break;
}
if(wIterTanViejo>0.1||wIterTanNuevo
    >0.1||wIterSagViejo>0.1||
    wIterSagNuevo>0.1  ||\
    isnan(wIterTanViejo)||isnan(
```

```
            wIterTanNuevo)||isnan(
            wIterSagViejo)||isnan(
            wIterSagNuevo))
        {
            termino=0;
            escribeCSV=1;
            fijaElemento(q_new[0],index1,
                index2,0);
            fijaElemento(q_new[1],index1,
                index2,0);
            break;
        }
    diferenciaTan=fabsl((wIterTanNuevo
        wIterTanViejo)/wIterTanNuevo)
        *100.00;
    diferenciaSag=fabsl((wIterSagNuevo
        wIterSagViejo)/wIterSagNuevo)
        *100.00;
    if(diferenciaTan<=umbral&&
        diferenciaSag<=umbral) // Termina
        la iteración
    {
        fijaElemento(q_new[0],index1,
            index2,obtieneElemento(q_iter
            ,1,ahora));
        fijaElemento(q_new[1],index1,
            index2,obtieneElemento(q_iter
            ,2,ahora));
        termino=1;
        escribeCSV=1;
        break;
    }
    else
    {
        if(diferenciaTan<=umbral)
            fijaElemento(q_new[0],index1,
                index2,obtieneElemento(
                q_iter,1,ahora));
        else
            fijaElemento(q_new[0],index1,
                index2,0);
        if(diferenciaSag<=umbral)
            fijaElemento(q_new[1],index1,
                index2,obtieneElemento(
                q_iter,2,ahora));
        else
```

```
                    fijaElemento(q_new[1],index1,
                        index2,0);
                termino=0;
            }
        }
    }
    else
    {
        if(elem(qInTan,index1,index2)==0)
            fijaElemento(q_new[0],index1,index2
                ,0.0);
        if(elem(qInSag,index1,index2)==0)
            fijaElemento(q_new[1],index1,index2
                ,0.0);
        escribeCSV=0;
        reallocFallido=0;
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spotsTan[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoTan,termino);
    guardaSpotIteraciones(spotsSag[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoSag,termino);
    if(escribeCSV==1)
    {
        guardaVariaciones(spotsTan,
            iteracionActualTan,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),tipoTan,subCarpetaTan,termino
            );
        guardaVariaciones(spotsSag,
            iteracionActualSag,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
```

```
                creall(obtieneElemento(epsilon2,1,
                index2)),tipoSag,subCarpetaSag,termino
                );
        }
        if(termino==0)
            printf("Cavidad en X, EM1: No cumple con
                el umbral de %Le para epsilon1=%Le,
                epsilon2=%Le.\nError relativo
                tangencial: %Le\nError relativo
                sagital: %Le\n",umbral,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),diferenciaTan,diferenciaSag);
        else
            printf("Cavidad en X, EM1: Cumple con el
                umbral de %Le para epsilon1=%Le,
                epsilon2=%Le.\nError relativo
                tangencial: %Le\nError relativo
                sagital: %Le\n",umbral,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),diferenciaTan,diferenciaSag);
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
    }
}
// Limpieza
borraMatriz(q_iter);
borraMatriz(EM1tan1);
borraMatriz(EM1tan2);
borraMatriz(EM1tan3);
borraMatriz(EM1tan4);
borraMatriz(EM1tan5);
borraMatriz(EM1tan6);
borraMatriz(EM1tan7);
borraMatriz(EM1tan8);
borraMatriz(EM1tan9);
borraMatriz(EM1tan10);
borraMatriz(EM1tan11);
borraMatriz(EM1tan12);
borraMatriz(EM1tan13);
borraMatriz(EM1tan14);
borraMatriz(EM1tan15);
borraMatriz(EM1tan16);
```

```
    borraMatriz(EM1tan17);
    borraMatriz(EM1sag1);
    borraMatriz(EM1sag2);
    borraMatriz(EM1sag3);
    borraMatriz(EM1sag4);
    borraMatriz(EM1sag5);
    borraMatriz(EM1sag6);
    borraMatriz(EM1sag7);
    borraMatriz(EM1sag8);
    borraMatriz(EM1sag9);
    borraMatriz(EM1sag10);
    borraMatriz(EM1sag11);
    borraMatriz(EM1sag12);
    borraMatriz(EM1sag13);
    borraMatriz(EM1sag14);
    borraMatriz(EM1sag15);
    borraMatriz(EM1sag16);
    borraMatriz(EM1sag17);
    return q_new;
}

matriz ** propNoLinealEM2Astigmatico(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser,\
                                        long double lambda0,
                                            matriz *epsilon1,
                                            matriz *epsilon2,
                                            int iteraciones,
                                            int pasosKerr,
                                            long double umbral
                                            )
```

```
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2], delta1, delta2, f1t, f1s, f2t, f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spotsTan, *spotsSag;
    int * iteracionActualTan, *iteracionActualSag;
    int cuenta=0;
    char tipoTan[]="EM2Tan_Acoplado", tipoSag[]="
        EM2Sag_Acoplado";
    char subCarpetaTan[]="EM2Tan_matAcop";
    char subCarpetaSag[]="EM2Sag_matAcop";
    long double wIterTanViejo=0, wIterTanNuevo=0,
        wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
         diferenciaSag=0; // Umbral no porcentual
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cos(angulos[0]);
    f2t=f2*cos(angulos[1]);
    f1s=f1/cos(angulos[0]);
    f2s=f2/cos(angulos[1]);

    matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
        // Espacio para dos matrices
    q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
         // Creando matriz tangencial
    q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
         // Creando matriz tangencial
    long double complex *qPropTan=(long double complex*)
        calloc(6,sizeof(long double complex)); //
        Parámetros para propagación
    long double complex *qPropSag=(long double complex*)
        calloc(6,sizeof(long double complex));

    // Crea matrices para la propagación
```

```
matriz *EM2tan1,*EM2tan2,*EM2tan3,*EM2tan4,*EM2tan5,*
    EM2tan6,*EM2tan7,*EM2tan8,*EM2tan9,*EM2tan10,*
    EM2tan11,*EM2tan12,*EM2tan13,*EM2tan14,*EM2tan15,*
    EM2tan16,*EM2tan17;
matriz *EM2sag1,*EM2sag2,*EM2sag3,*EM2sag4,*EM2sag5,*
    EM2sag6,*EM2sag7,*EM2sag8,*EM2sag9,*EM2sag10,*
    EM2sag11,*EM2sag12,*EM2sag13,*EM2sag14,*EM2sag15,*
    EM2sag16,*EM2sag17;

// EM2, Caso tangencial
EM2tan1=llenaMatriz(1,L2,0,1);
EM2tan2=llenaMatriz(1,0, 1/f2t,1);
EM2tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2tan4=llenaMatriz(n0,0,0,1/n0);
EM2tan5=llenaMatriz(1/n0,0,0,n0);
EM2tan6=nuevaMatriz(2,2);//EM2tan5 pendiente
EM2tan7=llenaMatriz(1,0, 1/f1t,1);
EM2tan8=llenaMatriz(1,L1,0,1);
EM2tan9=llenaMatriz(1,0,0,1);
EM2tan10=llenaMatriz(1,L1,0,1);
EM2tan11=llenaMatriz(1,0, 1/f1t,1);
EM2tan12=nuevaMatriz(2,2);      //EM2tan11 pendiente
EM2tan13=llenaMatriz(n0,0,0,1/n0);
EM2tan14=llenaMatriz(1/n0,0,0,n0);
EM2tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2tan16=llenaMatriz(1,0, 1/f2t,1);
EM2tan17=llenaMatriz(1,L2,0,1);
// EM1, Caso saggencial
EM2sag1=llenaMatriz(1,L2,0,1);
EM2sag2=llenaMatriz(1,0, 1/f2s,1);
EM2sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2sag4=llenaMatriz(1,0,0,1);
EM2sag5=llenaMatriz(1,0,0,1);
EM2sag6=nuevaMatriz(2,2);//EM2sag5 pendiente
EM2sag7=llenaMatriz(1,0, 1/f1s,1);
EM2sag8=llenaMatriz(1,L1,0,1);
EM2sag9=llenaMatriz(1,0,0,1);
EM2sag10=llenaMatriz(1,L1,0,1);
EM2sag11=llenaMatriz(1,0, 1/f1s,1);
EM2sag12=nuevaMatriz(2,2);      //EM2sag11 pendiente
EM2sag13=llenaMatriz(1,0,0,1);
EM2sag14=llenaMatriz(1,0,0,1);
EM2sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
```

```
    ,0,1);
EM2sag16=llenaMatriz(1,0, 1/f2s,1);
EM2sag17=llenaMatriz(1,L2,0,1);


// Ciclo
matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
    para tangencial, 2 para sagital
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        spotsTan=(long double*)calloc(1,sizeof(long
            double));
        spotsSag=(long double*)calloc(1,sizeof(long
            double));
        iteracionActualTan=(int *)calloc(1,sizeof(int
            ));
        iteracionActualSag=(int *)calloc(1,sizeof(int
            ));
        cuenta=1;
        iteracionActualTan[0]=0;
        iteracionActualSag[0]=0;
        spotsTan[0]=spot_q(obtieneElemento(qInTan,
            index1,index2),1,lambda0);
        spotsSag[0]=spot_q(obtieneElemento(qInSag,
            index1,index2),1,lambda0);
        // Llenando matrices variables:
        llenaMatrizSinCrear(EM2tan3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2tan6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2tan15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2sag3,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
        llenaMatrizSinCrear(EM2sag6,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag12,1,delta1+creall(
            obtieneElemento(epsilon1,1,index1)),0,1);
        llenaMatrizSinCrear(EM2sag15,1,delta2+creall(
            obtieneElemento(epsilon2,1,index2)),0,1);
```

```
if((elem(qInTan,index1,index2)!=0)&&(elem(
    qInSag,index1,index2)!=0)) // Si no son
    cero, realiza rutina
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        qInTan,index1,index2));
    fijaElemento(q_iter,2,1,obtieneElemento(
        qInSag,index1,index2));
    spotsTan=(long double*)calloc(1,sizeof(
        long double));
    spotsSag=(long double*)calloc(1,sizeof(
        long double));
    iteracionActualTan=(int *)calloc(1,sizeof
        (int));
    iteracionActualSag=(int *)calloc(1,sizeof
        (int));
    cuenta=1;
    iteracionActualTan[0]=0;
    iteracionActualSag[0]=0;
    spotsTan[0]=spot_q(obtieneElemento(qInTan
        ,index1,index2),1,lambda0);
    spotsSag[0]=spot_q(obtieneElemento(qInSag
        ,index1,index2),1,lambda0);
    escribeCSV=1;
    for(;;)
    {
        int ahora=1;
        int siguiente=2;
        matriz * Prop1Tan_piezas[]={EM2tan4,
            EM2tan3,EM2tan2,EM2tan1};
        matriz * Prop1Tan=
            variasMultMatriciales(
            Prop1Tan_piezas,4);
        matriz * Prop1Sag_piezas[]={EM2sag4,
            EM2sag3,EM2sag2,EM2sag1};
        matriz * Prop1Sag=
            variasMultMatriciales(
            Prop1Sag_piezas,4);
        // Almacenando primer q
        qPropTan[0]=obtieneElemento(q_iter,1,
            ahora);
        qPropSag[0]=obtieneElemento(q_iter,2,
            ahora);
        // Cálculo de q2
        qPropTan[1]=prop_q(Prop1Tan,qPropTan
            [0],1,n0);
```

```
qPropSag[1]=prop_q(Prop1Sag,qPropSag
    [0],1,n0);
// Cálculo de q3
propAstigmatica *q3=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[1],qPropSag[1],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,0);
qPropTan[2]=q3>qTan;
qPropSag[2]=q3>qSag;
free(q3);
// Más matrices
matriz * Prop2Tan_piezas[]={EM2tan13,
    EM2tan12,EM2tan11,EM2tan10,EM2tan9
    ,EM2tan8,EM2tan7,EM2tan6,EM2tan5};
     // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,9);
matriz * Prop2Sag_piezas[]={EM2sag13,
    EM2sag12,EM2sag11,EM2sag10,EM2sag9
    ,EM2sag8,EM2sag7,EM2sag6,EM2sag5};
     // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,9);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,n0);
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,n0);
// Propagación de regreso Kerr,
    cálculo q5
propAstigmatica *q5=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[3],qPropSag[3],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,1);
qPropTan[4]=q5>qTan;
qPropSag[4]=q5>qSag;
free(q5);
// Más matrices
matriz * Prop3Tan_piezas[]={EM2tan17,
    EM2tan16,EM2tan15,EM2tan14}; // 14
     a 17
matriz * Prop3Tan=
```

```
    variasMultMatriciales(
    Prop3Tan_piezas,4);
matriz * Prop3Sag_piezas[]={EM2sag17,
    EM2sag16,EM2sag15,EM2sag14}; // 14
     a 17
matriz * Prop3Sag=
    variasMultMatriciales(
    Prop3Sag_piezas,4);
// Cálculo de q6
qPropTan[5]=prop_q(Prop3Tan,qPropTan
    [4],n0,1);
qPropSag[5]=prop_q(Prop3Sag,qPropSag
    [4],n0,1);
// Guarda elementos
fijaElemento(q_iter,1,ahora,qPropTan
    [5]);
fijaElemento(q_iter,2,ahora,qPropSag
    [5]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop3Tan=borraMatriz(Prop3Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);
Prop3Sag=borraMatriz(Prop3Sag);

// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[5],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
    lambda0);
wIterSagNuevo=spot_q(qPropSag[5],1,
    lambda0);

if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_sptTan, *tmp_sptSag, *
    tmp_iterTan, *tmp_iterSag;
tmp_sptTan=(long double*)realloc(
    spotsTan,cuenta*sizeof(long double
    ));
```

```
tmp_sptSag=(long double*)realloc(
    spotsSag,cuenta*sizeof(long double
    ));
tmp_iterTan=(int *)realloc(
    iteracionActualTan,cuenta*sizeof(
    int));
tmp_iterSag=(int *)realloc(
    iteracionActualSag,cuenta*sizeof(
    int));
if(tmp_sptTan!=NULL && tmp_sptSag!=
    NULL && tmp_iterTan!=NULL &&
    tmp_iterSag!=NULL)
{
     spotsTan=tmp_sptTan;
     spotsSag=tmp_sptSag;
     iteracionActualTan=tmp_iterTan;
     iteracionActualSag=tmp_iterSag;
     spotsTan[cuenta 1]=wIterTanNuevo;
     spotsSag[cuenta 1]=wIterSagNuevo;
     iteracionActualTan[cuenta 1]=
         cuenta 1;
     iteracionActualSag[cuenta 1]=
         cuenta 1;
}
else
{
     reallocFallido=1;
     break;
}
if(wIterTanViejo>0.1||wIterTanNuevo
    >0.1||wIterSagViejo>0.1||
    wIterSagNuevo>0.1 ||\
    isnan(wIterTanViejo)||isnan(
        wIterTanNuevo)||isnan(
        wIterSagViejo)||isnan(
        wIterSagNuevo))
    {
          termino=0;
          escribeCSV=1;
          fijaElemento(q_new[0],index1,
              index2,0);
          fijaElemento(q_new[1],index1,
              index2,0);
          break;
    }
diferenciaTan=fabsl((wIterTanNuevo
```

```
                    wIterTanViejo)/wIterTanNuevo)
                    *100.00;
                diferenciaSag=fabsl((wIterSagNuevo
                    wIterSagViejo)/wIterSagNuevo)
                    *100.00;
                if(diferenciaTan<=umbral&&
                    diferenciaSag<=umbral) // Termina
                    la iteración
                {
                    fijaElemento(q_new[0],index1,
                        index2,obtieneElemento(q_iter
                        ,1,ahora));
                    fijaElemento(q_new[1],index1,
                        index2,obtieneElemento(q_iter
                        ,2,ahora));
                    termino=1;
                    escribeCSV=1;
                    break;
                }
                else
                {
                    if(diferenciaTan<=umbral)
                        fijaElemento(q_new[0],index1,
                            index2,obtieneElemento(
                            q_iter,1,ahora));
                    else
                        fijaElemento(q_new[0],index1,
                            index2,0);
                    if(diferenciaSag<=umbral)
                        fijaElemento(q_new[1],index1,
                            index2,obtieneElemento(
                            q_iter,2,ahora));
                    else
                        fijaElemento(q_new[1],index1,
                            index2,0);
                    termino=0;
                }
            }
        }
        else
        {
            if(elem(qInTan,index1,index2)==0)
                fijaElemento(q_new[0],index1,index2
                    ,0.0);
            if(elem(qInSag,index1,index2)==0)
                fijaElemento(q_new[1],index1,index2
```

```
                ,0.0);
        escribeCSV=0;
        reallocFallido=0;
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spotsTan[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoTan,termino);
    guardaSpotIteraciones(spotsSag[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoSag,termino);
    if(escribeCSV==1)
    {
        guardaVariaciones(spotsTan,
            iteracionActualTan,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),tipoTan,subCarpetaTan,termino
            );
        guardaVariaciones(spotsSag,
            iteracionActualSag,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),tipoSag,subCarpetaSag,termino
            );
    }
    if(termino==0)
        printf("Cavidad en X, EM2: No cumple con
            el umbral de %Le para epsilon1=%Le,
            epsilon2=%Le.\nError relativo
            tangencial: %Le\nError relativo
            sagital: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),diferenciaTan,diferenciaSag);
```

```
            else
                printf("Cavidad en X, EM2: Cumple con el
                    umbral de %Le para epsilon1=%Le,
                    epsilon2=%Le.\nError relativo
                    tangencial: %Le\nError relativo
                    sagital: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferenciaTan,diferenciaSag);
            free(spotsTan);
            free(spotsSag);
            free(iteracionActualTan);
            free(iteracionActualSag);
        }
    }
    // Limpieza
    free(qPropTan);
    free(qPropSag);
    borraMatriz(q_iter);
    borraMatriz(EM2tan1);
    borraMatriz(EM2tan2);
    borraMatriz(EM2tan3);
    borraMatriz(EM2tan4);
    borraMatriz(EM2tan5);
    borraMatriz(EM2tan6);
    borraMatriz(EM2tan7);
    borraMatriz(EM2tan8);
    borraMatriz(EM2tan9);
    borraMatriz(EM2tan10);
    borraMatriz(EM2tan11);
    borraMatriz(EM2tan12);
    borraMatriz(EM2tan13);
    borraMatriz(EM2tan14);
    borraMatriz(EM2tan15);
    borraMatriz(EM2tan16);
    borraMatriz(EM2tan17);
    borraMatriz(EM2sag1);
    borraMatriz(EM2sag2);
    borraMatriz(EM2sag3);
    borraMatriz(EM2sag4);
    borraMatriz(EM2sag5);
    borraMatriz(EM2sag6);
    borraMatriz(EM2sag7);
    borraMatriz(EM2sag8);
    borraMatriz(EM2sag9);
    borraMatriz(EM2sag10);
```

```
    borraMatriz(EM2sag11);
    borraMatriz(EM2sag12);
    borraMatriz(EM2sag13);
    borraMatriz(EM2sag14);
    borraMatriz(EM2sag15);
    borraMatriz(EM2sag16);
    borraMatriz(EM2sag17);
    return q_new;
}

matriz ** propNoLinealAnilloRuta1Astigmatico(matriz *
   qInTan, matriz *qInSag, ajusteTemperaturaCristal *
   vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double a,
                                            long double b,
                                            long double c,\
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser,\
                                        long double lambda0,
                                            matriz *epsilon1,
                                            matriz *epsilon2,
                                            int iteraciones,
                                            int pasosKerr,
                                            long double umbral
                                            )
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double wIterTanViejo=0, wIterTanNuevo=0,
        wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
         diferenciaSag=0; // Umbral no porcentual
    long double L1=a, L2=b+c;
```

```
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
    // Espacio para dos matrices
q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
     // Creando matriz tangencial
q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
     // Creando matriz tangencial
long double complex *qPropTan=(long double complex*)
    calloc(4,sizeof(long double complex)); //
    Parámetros para propagación
long double complex *qPropSag=(long double complex*)
    calloc(4,sizeof(long double complex));

// Crea matrices para la propagación

matriz *Ruta1tan1,*Ruta1tan2,*Ruta1tan3,*Ruta1tan4,*
    Ruta1tan5,*Ruta1tan6,*Ruta1tan7,*Ruta1tan8;
matriz *Ruta1sag1,*Ruta1sag2,*Ruta1sag3,*Ruta1sag4,*
    Ruta1sag5,*Ruta1sag6,*Ruta1sag7,*Ruta1sag8;

// Ruta1, Caso tangencial
Ruta1tan1=llenaMatriz(1,L1,0,1); // L1=a
Ruta1tan2=llenaMatriz(1,0,1/f1t,1);
Ruta1tan3=nuevaMatriz(2,2); // Pendiente
Ruta1tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
    entrada)
Ruta1tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
    salida)
Ruta1tan6=nuevaMatriz(2,2);//Ruta1tan6 pendiente
Ruta1tan7=llenaMatriz(1,0,1/f2t,1);
Ruta1tan8=llenaMatriz(1,L2,0,1); // L2=b+c
```

```
// Ruta 1, caso sagital
 Ruta1sag1=llenaMatriz(1,L1,0,1); // L1=a
 Ruta1sag2=llenaMatriz(1,0, 1/f1s,1);
 Ruta1sag3=nuevaMatriz(2,2); // Pendiente
 Ruta1sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
     entrada)
 Ruta1sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
     )
 Ruta1sag6=nuevaMatriz(2,2);//Ruta1sag6 pendiente
 Ruta1sag7=llenaMatriz(1,0, 1/f2s,1);
 Ruta1sag8=llenaMatriz(1,L2,0,1); // L2=b+c



 // Ciclo
 matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
     para tangencial, 2 para sagital
 for(int index1=1;index1<=epsilon1 >columnas;index1++)
 {
     for(int index2=1;index2<=epsilon2 >columnas;
         index2++)
     {
         // Llenando matrices variables:
         llenaMatrizSinCrear(Ruta1tan3,1,delta1+creall
             (obtieneElemento(epsilon1,1,index1)),0,1);
                     llenaMatrizSinCrear(Ruta1tan6,1,
                         delta2+creall(obtieneElemento(
                         epsilon2,1,index2)),0,1);
                     llenaMatrizSinCrear(Ruta1sag3,1,
                         delta1+creall(obtieneElemento(
                         epsilon1,1,index1)),0,1);
                     llenaMatrizSinCrear(Ruta1sag6,1,
                         delta2+creall(obtieneElemento(
                         epsilon2,1,index2)),0,1);

         if((elem(qInTan,index1,index2)!=0)&&(elem(
             qInSag,index1,index2)!=0)) // Si no son
             cero, realiza rutina
         {
             fijaElemento(q_iter,1,1,obtieneElemento(
                 qInTan,index1,index2));
             fijaElemento(q_iter,2,1,obtieneElemento(
                 qInSag,index1,index2));
             for(int iter=iteraciones 1;iter>0;iter  )
             {
                 int ahora=iteraciones iter;
```

```
int siguiente=ahora+1;
matriz * Prop1Tan_piezas[]={Ruta1tan4
    ,Ruta1tan3,Ruta1tan2,Ruta1tan1};
matriz * Prop1Tan=
    variasMultMatriciales(
    Prop1Tan_piezas,4);
matriz * Prop1Sag_piezas[]={Ruta1sag4
    ,Ruta1sag3,Ruta1sag2,Ruta1sag1};
matriz * Prop1Sag=
    variasMultMatriciales(
    Prop1Sag_piezas,4);
// Almacenando primer q
qPropTan[0]=obtieneElemento(q_iter,1,
    ahora);
qPropSag[0]=obtieneElemento(q_iter,2,
    ahora);
// Cálculo de q2
qPropTan[1]=prop_q(Prop1Tan,qPropTan
    [0],1,n0);
qPropSag[1]=prop_q(Prop1Sag,qPropSag
    [0],1,n0);
// Cálculo de q3
propAstigmatica *q3=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[1],qPropSag[1],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,1);
qPropTan[2]=q3>qTan;
qPropSag[2]=q3>qSag;
free(q3);
// Más matrices
matriz * Prop2Tan_piezas[]={Ruta1tan8
    ,Ruta1tan7,Ruta1tan6,Ruta1tan5};
    // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,4);
matriz * Prop2Sag_piezas[]={Ruta1sag8
    ,Ruta1sag7,Ruta1sag6,Ruta1sag5};
    // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,4);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,1.0);
```

```
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,1.0);
// Guarda elementos
fijaElemento(q_iter,1,siguiente,
    qPropTan[3]);
fijaElemento(q_iter,2,siguiente,
    qPropSag[3]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);


// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[3],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
    lambda0);
wIterSagNuevo=spot_q(qPropSag[3],1,
    lambda0);
if(wIterTanViejo>0.2||wIterTanNuevo
    >0.2||wIterSagViejo>0.2||
    wIterSagNuevo>0.2) break; // Si
    el spot mide 40 cm es demasiado
diferenciaTan=fabsl((wIterTanNuevo
    wIterTanViejo)/wIterTanNuevo)
    *100.00;
diferenciaSag=fabsl((wIterSagNuevo
    wIterSagViejo)/wIterSagNuevo)
    *100.00;
if(diferenciaTan<=umbral&&
    diferenciaSag<=umbral) // Termina
    la iteración
{
     fijaElemento(q_new[0],index1,
         index2,obtieneElemento(q_iter
         ,1,ahora));
     fijaElemento(q_new[1],index1,
         index2,obtieneElemento(q_iter
         ,2,ahora));
     termino=1;
     break;
```

```
            }
            else
            {
                if(diferenciaTan<=umbral)
                    fijaElemento(q_new[0],index1,
                        index2,obtieneElemento(
                        q_iter,1,ahora));
                else
                    fijaElemento(q_new[0],index1,
                        index2,0);
                if(diferenciaSag<=umbral)
                    fijaElemento(q_new[1],index1,
                        index2,obtieneElemento(
                        q_iter,2,ahora));
                else
                    fijaElemento(q_new[1],index1,
                        index2,0);
                termino=0;
            }
        }
    }
    else
    {
        if(elem(qInTan,index1,index2)==0)
            fijaElemento(q_new[0],index1,index2
                ,0.0);
        if(elem(qInSag,index1,index2)==0)
            fijaElemento(q_new[1],index1,index2
                ,0.0);
    }
    if(termino==0)
        printf("Cavidad en anillo, Ruta 1: No
            cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo tangencial: %Le\nError
            relativo sagital: %Le\n",umbral,creall
            (obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),diferenciaTan,diferenciaSag);
    else
        printf("Cavidad en anillo, Ruta 1: Cumple
            con el umbral de %Le para epsilon1=%
            Le, epsilon2=%Le.\nError relativo
            tangencial: %Le\nError relativo
            sagital: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
```

```
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferenciaTan,diferenciaSag);

        }
    }
    // Limpieza
    free(qPropTan);
    free(qPropSag);
    borraMatriz(q_iter);
    borraMatriz(Ruta1tan1);
    borraMatriz(Ruta1tan2);
    borraMatriz(Ruta1tan3);
    borraMatriz(Ruta1tan4);
    borraMatriz(Ruta1tan5);
    borraMatriz(Ruta1tan6);
    borraMatriz(Ruta1tan7);
    borraMatriz(Ruta1tan8);
    borraMatriz(Ruta1sag1);
    borraMatriz(Ruta1sag2);
    borraMatriz(Ruta1sag3);
    borraMatriz(Ruta1sag4);
    borraMatriz(Ruta1sag5);
    borraMatriz(Ruta1sag6);
    borraMatriz(Ruta1sag7);
    borraMatriz(Ruta1sag8);
    return q_new;
}

matriz ** propNoLinealAnilloRuta2Astigmatico(matriz *
   qInTan, matriz *qInSag, ajusteTemperaturaCristal *
   vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double a,
                                            long double b,
                                            long double c,\
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
```

```
                                                    double dn_dv,long
                                                    double P_laser,\
                                                long double lambda0,
                                                    matriz *epsilon1,
                                                    matriz *epsilon2,
                                                    int iteraciones,
                                                    int pasosKerr,
                                                    long double umbral
                                                    )
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    long double wIterTanViejo=0, wIterTanNuevo=0,
        wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
         diferenciaSag=0; // Umbral no porcentual
    long double L1=a, L2=b+c;
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cosl(angulos[0]);
    f2t=f2*cosl(angulos[1]);
    f1s=f1/cosl(angulos[0]);
    f2s=f2/cosl(angulos[1]);

    matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
        // Espacio para dos matrices
    q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
         // Creando matriz tangencial
    q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
         // Creando matriz tangencial
    long double complex *qPropTan=(long double complex*)
        calloc(4,sizeof(long double complex)); //
        Parámetros para propagación
    long double complex *qPropSag=(long double complex*)
        calloc(4,sizeof(long double complex));

    // Crea matrices para la propagación
```

```
matriz *Ruta2tan1,*Ruta2tan2,*Ruta2tan3,*Ruta2tan4,*
    Ruta2tan5,*Ruta2tan6,*Ruta2tan7,*Ruta2tan8;
matriz *Ruta2sag1,*Ruta2sag2,*Ruta2sag3,*Ruta2sag4,*
    Ruta2sag5,*Ruta2sag6,*Ruta2sag7,*Ruta2sag8;

// Ruta 2, caso tangencial
Ruta2tan1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2tan2=llenaMatriz(1,0, 1/f2t,1);
Ruta2tan3=nuevaMatriz(2,2); // Pendiente
Ruta2tan4=llenaMatriz(n0,0,0,1/n0); // Medio Kerr (
    entrada)
Ruta2tan5=llenaMatriz(1/n0,0,0,n0); // Medio Kerr (
    salida)
Ruta2tan6=nuevaMatriz(2,2);//Ruta2tan6 pendiente
Ruta2tan7=llenaMatriz(1,0, 1/f1t,1);
Ruta2tan8=llenaMatriz(1,L1,0,1); // L1=a

// Ruta 2, caso sagital
Ruta2sag1=llenaMatriz(1,L2,0,1); // L2=b+c
Ruta2sag2=llenaMatriz(1,0, 1/f2s,1);
Ruta2sag3=nuevaMatriz(2,2); // Pendiente
Ruta2sag4=llenaMatriz(1,0,0,1); // Medio Kerr (
    entrada)
Ruta2sag5=llenaMatriz(1,0,0,1); // Medio Kerr (salida
    )
Ruta2sag6=nuevaMatriz(2,2);//Ruta2sag6 pendiente
Ruta2sag7=llenaMatriz(1,0, 1/f1s,1);
Ruta2sag8=llenaMatriz(1,L1,0,1); // L1=a



// Ciclo
matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
    para tangencial, 2 para sagital
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
    for(int index2=1;index2<=epsilon2 >columnas;
        index2++)
    {
        // Llenando matrices variables:
        llenaMatrizSinCrear(Ruta2tan3,1,delta2+creall
            (obtieneElemento(epsilon2,1,index2)),0,1);
                llenaMatrizSinCrear(Ruta2tan6,1,
                    delta1+creall(obtieneElemento(
                    epsilon1,1,index1)),0,1);
```

```
                llenaMatrizSinCrear(Ruta2sag3,1,
                    delta2+creall(obtieneElemento(
                    epsilon2,1,index2)),0,1);
                llenaMatrizSinCrear(Ruta2sag6,1,
                    delta1+creall(obtieneElemento(
                    epsilon1,1,index1)),0,1);

            if((elem(qInTan,index1,index2)!=0)&&(elem(
                qInSag,index1,index2)!=0)) // Si no son
                cero, realiza rutina
            {
                fijaElemento(q_iter,1,1,obtieneElemento(
                    qInTan,index1,index2));
                fijaElemento(q_iter,2,1,obtieneElemento(
                    qInSag,index1,index2));
                for(int iter=iteraciones 1;iter>0;iter  )
                {

                    int ahora=iteraciones iter;
                    int siguiente=ahora+1;
                    matriz * Prop1Tan_piezas[]={Ruta2tan4
                        ,Ruta2tan3,Ruta2tan2,Ruta2tan1};
                    matriz * Prop1Tan=
                        variasMultMatriciales(
                        Prop1Tan_piezas,4);
                    matriz * Prop1Sag_piezas[]={Ruta2sag4
                        ,Ruta2sag3,Ruta2sag2,Ruta2sag1};
                    matriz * Prop1Sag=
                        variasMultMatriciales(
                        Prop1Sag_piezas,4);
                    // Almacenando primer q
                    qPropTan[0]=obtieneElemento(q_iter,1,
                        ahora);
                    qPropSag[0]=obtieneElemento(q_iter,2,
                        ahora);
                    // Cálculo de q2
                    qPropTan[1]=prop_q(Prop1Tan,qPropTan
                        [0],1,n0);
                    qPropSag[1]=prop_q(Prop1Sag,qPropSag
                        [0],1,n0);
                    // Cálculo de q3
                    propAstigmatica *q3=
                        propagacionKerrAcoplada(pasosKerr,
                        L,n0,n2,qPropTan[1],qPropSag[1],
                        chi,kth,Cp,rho,dn_dv,P_laser,
                        lambda0,vector,0);
```

```
qPropTan[2]=q3>qTan;
qPropSag[2]=q3>qSag;
free(q3);
// Más matrices
matriz * Prop2Tan_piezas[]={Ruta2tan8
    ,Ruta2tan7,Ruta2tan6,Ruta2tan5};
    // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,4);
matriz * Prop2Sag_piezas[]={Ruta2sag8
    ,Ruta2sag7,Ruta2sag6,Ruta2sag5};
    // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,4);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,1.0);
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,1.0);
// Guarda elementos
fijaElemento(q_iter,1,siguiente,
    qPropTan[3]);
fijaElemento(q_iter,2,siguiente,
    qPropSag[3]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);


// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[3],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
    lambda0);
wIterSagNuevo=spot_q(qPropSag[3],1,
    lambda0);
if(wIterTanViejo>0.2||wIterTanNuevo
    >0.2||wIterSagViejo>0.2||
    wIterSagNuevo>0.2) break;   // Si
```

```
                el spot mide 40 cm es demasiado
            diferenciaTan=fabsl((wIterTanNuevo
                wIterTanViejo)/wIterTanNuevo)
                *100.00;
            diferenciaSag=fabsl((wIterSagNuevo
                wIterSagViejo)/wIterSagNuevo)
                *100.00;
            if(diferenciaTan<=umbral&&
                diferenciaSag<=umbral) // Termina
                la iteración
            {
                fijaElemento(q_new[0],index1,
                    index2,obtieneElemento(q_iter
                    ,1,siguiente));
                fijaElemento(q_new[1],index1,
                    index2,obtieneElemento(q_iter
                    ,2,siguiente));
                termino=1;
                break;
            }
            else
            {
                fijaElemento(q_new[0],index1,
                    index2,0);
                fijaElemento(q_new[1],index1,
                    index2,0);
                termino=0;
            }
        }
    }
    else
    {
        if(elem(qInTan,index1,index2)==0)
            fijaElemento(q_new[0],index1,index2
                ,0.0);
        if(elem(qInSag,index1,index2)==0)
            fijaElemento(q_new[1],index1,index2
                ,0.0);
    }
    if(termino==0)
        printf("Cavidad en anillo, Ruta 2: No
            cumple con el umbral de %Le para
            epsilon1=%Le, epsilon2=%Le.\nError
            relativo tangencial: %Le\nError
            relativo sagital: %Le\n",umbral,creall
            (obtieneElemento(epsilon1,1,index1)),
```

```
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferenciaTan,diferenciaSag);
            else
                printf("Cavidad en anillo, Ruta 2: Cumple
                    con el umbral de %Le para epsilon1=%
                    Le, epsilon2=%Le.\nError relativo
                    tangencial: %Le\nError relativo
                    sagital: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferenciaTan,diferenciaSag);
        }
    }
    // Limpieza
    borraMatriz(q_iter);
    borraMatriz(Ruta2tan1);
    borraMatriz(Ruta2tan2);
    borraMatriz(Ruta2tan3);
    borraMatriz(Ruta2tan4);
    borraMatriz(Ruta2tan5);
    borraMatriz(Ruta2tan6);
    borraMatriz(Ruta2tan7);
    borraMatriz(Ruta2tan8);
    borraMatriz(Ruta2sag1);
    borraMatriz(Ruta2sag2);
    borraMatriz(Ruta2sag3);
    borraMatriz(Ruta2sag4);
    borraMatriz(Ruta2sag5);
    borraMatriz(Ruta2sag6);
    borraMatriz(Ruta2sag7);
    borraMatriz(Ruta2sag8);
    free(qPropTan);
    free(qPropSag);
    return q_new;
}

matriz ** propNoLinealEM1AstigmaticoKerr(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
```

```
                                                    double n0, long
                                                    double n2, \
                                                long double chi, long
                                                     double kth, long
                                                    double Cp, \
                                                long double rho,long
                                                    double dn_dv,long
                                                    double P_laser,\
                                                long double lambda0,
                                                    matriz *epsilon1,
                                                    matriz *epsilon2,
                                                    int iteraciones,
                                                    int pasosKerr,
                                                    long double umbral
                                                    )
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spotsTan, *spotsSag;
    int * iteracionActualTan, *iteracionActualSag;
    int cuenta=0;
    char tipoTan[]="EM1Tan_AcopladoKerr", tipoSag[]="
        EM1Sag_AcopladoKerr";
    char subCarpetaTan[]="EM1Tan_matAcopKerr";
    char subCarpetaSag[]="EM1Sag_matAcopKerr";
    long double wIterTanViejo=0, wIterTanNuevo=0,
        wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
         diferenciaSag=0; // Umbral no porcentual
    assert(strlen(conjugado_corto)==3);
    assert(strlen(conjugado_largo)==3);
    anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
        L2,f1,f2,angulos);
    delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
        angulos[0]);
    delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
        angulos[1]);

    // Cálculo de distancias focales
    f1t=f1*cos(angulos[0]);
    f2t=f2*cos(angulos[1]);
    f1s=f1/cos(angulos[0]);
    f2s=f2/cos(angulos[1]);
```

```
matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
    // Espacio para dos matrices
q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
     // Creando matriz tangencial
q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
     // Creando matriz tangencial
long double complex *qPropTan=(long double complex*)
    calloc(6,sizeof(long double complex)); //
    Parámetros para propagación
long double complex *qPropSag=(long double complex*)
    calloc(6,sizeof(long double complex));

// Crea matrices para la propagación

matriz *EM1tan1,*EM1tan2,*EM1tan3,*EM1tan4,*EM1tan5,*
    EM1tan6,*EM1tan7,*EM1tan8,*EM1tan9,*EM1tan10,*
    EM1tan11,*EM1tan12,*EM1tan13,*EM1tan14,*EM1tan15,*
    EM1tan16,*EM1tan17;
matriz *EM1sag1,*EM1sag2,*EM1sag3,*EM1sag4,*EM1sag5,*
    EM1sag6,*EM1sag7,*EM1sag8,*EM1sag9,*EM1sag10,*
    EM1sag11,*EM1sag12,*EM1sag13,*EM1sag14,*EM1sag15,*
    EM1sag16,*EM1sag17;
// EM1, Caso tangencial
EM1tan1=llenaMatriz(1,L1,0,1);
EM1tan2=llenaMatriz(1,0, 1/f1t,1);
EM1tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1tan4=llenaMatriz(n0,0,0,1/n0);
EM1tan5=llenaMatriz(1/n0,0,0,n0);
EM1tan6=nuevaMatriz(2,2);//EM1tan5 pendiente
EM1tan7=llenaMatriz(1,0, 1/f2t,1);
EM1tan8=llenaMatriz(1,L2,0,1);
EM1tan9=llenaMatriz(1,0,0,1);
EM1tan10=llenaMatriz(1,L2,0,1);
EM1tan11=llenaMatriz(1,0, 1/f2t,1);
EM1tan12=nuevaMatriz(2,2);     //EM1tan11 pendiente
EM1tan13=llenaMatriz(n0,0,0,1/n0);
EM1tan14=llenaMatriz(1/n0,0,0,n0);
EM1tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1tan16=llenaMatriz(1,0, 1/f1t,1);
EM1tan17=llenaMatriz(1,L1,0,1);
// EM1, Caso saggencial
EM1sag1=llenaMatriz(1,L1,0,1);
EM1sag2=llenaMatriz(1,0, 1/f1s,1);
```

```
EM1sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM1sag4=llenaMatriz(1,0,0,1);
EM1sag5=llenaMatriz(1,0,0,1);
EM1sag6=nuevaMatriz(2,2);//EM1sag5 pendiente
EM1sag7=llenaMatriz(1,0, 1/f2s,1);
EM1sag8=llenaMatriz(1,L2,0,1);
EM1sag9=llenaMatriz(1,0,0,1);
EM1sag10=llenaMatriz(1,L2,0,1);
EM1sag11=llenaMatriz(1,0, 1/f2s,1);
EM1sag12=nuevaMatriz(2,2);      //EM1sag11 pendiente
EM1sag13=llenaMatriz(1,0,0,1);
EM1sag14=llenaMatriz(1,0,0,1);
EM1sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM1sag16=llenaMatriz(1,0, 1/f1s,1);
EM1sag17=llenaMatriz(1,L1,0,1);


// Ciclo
matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
    para tangencial, 2 para sagital
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
     for(int index2=1;index2<=epsilon2 >columnas;
         index2++)
     {
          spotsTan=(long double*)calloc(1,sizeof(long
              double));
          spotsSag=(long double*)calloc(1,sizeof(long
              double));
          iteracionActualTan=(int *)calloc(1,sizeof(int
              ));
          iteracionActualSag=(int *)calloc(1,sizeof(int
              ));
          cuenta=1;
          iteracionActualTan[0]=0;
          iteracionActualSag[0]=0;
          spotsTan[0]=spot_q(obtieneElemento(qInTan,
              index1,index2),1,lambda0);
          spotsSag[0]=spot_q(obtieneElemento(qInSag,
              index1,index2),1,lambda0);
          // Llenando matrices variables:
          llenaMatrizSinCrear(EM1tan3,1,delta1+creall(
              obtieneElemento(epsilon1,1,index1)),0,1);
          llenaMatrizSinCrear(EM1tan6,1,delta2+creall(
```

```
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1tan12,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1tan15,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1sag3,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM1sag6,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag12,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM1sag15,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);

if((elem(qInTan,index1,index2)!=0)&&(elem(
    qInSag,index1,index2)!=0)) // Si no son
    cero, realiza rutina
{
    fijaElemento(q_iter,1,1,obtieneElemento(
        qInTan,index1,index2));
    fijaElemento(q_iter,2,1,obtieneElemento(
        qInSag,index1,index2));
    escribeCSV=1;
    for(;;)
    {
        int ahora=1;
        int siguiente=2;
        matriz * Prop1Tan_piezas[]={EM1tan4,
            EM1tan3,EM1tan2,EM1tan1};
        matriz * Prop1Tan=
            variasMultMatriciales(
            Prop1Tan_piezas,4);
        matriz * Prop1Sag_piezas[]={EM1sag4,
            EM1sag3,EM1sag2,EM1sag1};
        matriz * Prop1Sag=
            variasMultMatriciales(
            Prop1Sag_piezas,4);
        // Almacenando primer q
        qPropTan[0]=obtieneElemento(q_iter,1,
            ahora);
        qPropSag[0]=obtieneElemento(q_iter,2,
            ahora);
        // Cálculo de q2
        qPropTan[1]=prop_q(Prop1Tan,qPropTan
            [0],1,n0);
        qPropSag[1]=prop_q(Prop1Sag,qPropSag
```

```
    [0],1,n0);
// Cálculo de q3
propAstigmatica *q3=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[1],qPropSag[1],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,1);
qPropTan[2]=q3>qTan;
qPropSag[2]=q3>qSag;
free(q3);
// Más matrices
matriz * Prop2Tan_piezas[]={EM1tan13,
    EM1tan12,EM1tan11,EM1tan10,EM1tan9
    ,EM1tan8,EM1tan7,EM1tan6,EM1tan5};
     // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,9);
matriz * Prop2Sag_piezas[]={EM1sag13,
    EM1sag12,EM1sag11,EM1sag10,EM1sag9
    ,EM1sag8,EM1sag7,EM1sag6,EM1sag5};
     // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,9);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,n0);
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,n0);
// Propagación de regreso Kerr,
    cálculo q5
propAstigmatica *q5=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[3],qPropSag[3],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,0);
qPropTan[4]=q5>qTan;
qPropSag[4]=q5>qSag;
free(q5);
// Más matrices
matriz * Prop3Tan_piezas[]={EM1tan17,
    EM1tan16,EM1tan15,EM1tan14}; // 14
     a 17
matriz * Prop3Tan=
    variasMultMatriciales(
```

```
    Prop3Tan_piezas,4);
matriz * Prop3Sag_piezas[]={EM1sag17,
    EM1sag16,EM1sag15,EM1sag14}; // 14
     a 17
matriz * Prop3Sag=
    variasMultMatriciales(
    Prop3Sag_piezas,4);
// Cálculo de q6
qPropTan[5]=prop_q(Prop3Tan,qPropTan
    [4],n0,1);
qPropSag[5]=prop_q(Prop3Sag,qPropSag
    [4],n0,1);
// Guarda elementos
fijaElemento(q_iter,1,ahora,qPropTan
    [5]);
fijaElemento(q_iter,2,ahora,qPropSag
    [5]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop3Tan=borraMatriz(Prop3Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);
Prop3Sag=borraMatriz(Prop3Sag);

// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[5],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
    lambda0);
wIterSagNuevo=spot_q(qPropSag[5],1,
    lambda0);

if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_sptTan, *tmp_sptSag, *
    tmp_iterTan, *tmp_iterSag;
tmp_sptTan=(long double*)realloc(
    spotsTan,cuenta*sizeof(long double
    ));
tmp_sptSag=(long double*)realloc(
```

```
        spotsSag,cuenta*sizeof(long double
        ));
    tmp_iterTan=(int *)realloc(
        iteracionActualTan,cuenta*sizeof(
        int));
    tmp_iterSag=(int *)realloc(
        iteracionActualSag,cuenta*sizeof(
        int));
    if(tmp_sptTan!=NULL && tmp_sptSag!=
        NULL && tmp_iterTan!=NULL &&
        tmp_iterSag!=NULL)
    {
         spotsTan=tmp_sptTan;
         spotsSag=tmp_sptSag;
         iteracionActualTan=tmp_iterTan;
         iteracionActualSag=tmp_iterSag;
         spotsTan[cuenta-1]=wIterTanNuevo;
         spotsSag[cuenta-1]=wIterSagNuevo;
         iteracionActualTan[cuenta-1]=
             cuenta-1;
         iteracionActualSag[cuenta-1]=
             cuenta-1;
    }
    else
    {
         reallocFallido=1;
         break;
    }
    if(wIterTanViejo>0.1||wIterTanNuevo
        >0.1||wIterSagViejo>0.1||
        wIterSagNuevo>0.1  ||\
        isnan(wIterTanViejo)||isnan(
            wIterTanNuevo)||isnan(
            wIterSagViejo)||isnan(
            wIterSagNuevo))
        {
              termino=0;
              escribeCSV=1;
              fijaElemento(q_new[0],index1,
                  index2,0);
              fijaElemento(q_new[1],index1,
                  index2,0);
              break;
        }
    diferenciaTan=fabsl((wIterTanNuevo
        -wIterTanViejo)/wIterTanNuevo)
```

```
                *100.00;
            diferenciaSag=fabsl((wIterSagNuevo
                wIterSagViejo)/wIterSagNuevo)
                *100.00;
            if(diferenciaTan<=umbral&&
                diferenciaSag<=umbral) // Termina
                la iteración
            {
                fijaElemento(q_new[0],index1,
                    index2,obtieneElemento(q_iter
                    ,1,ahora));
                fijaElemento(q_new[1],index1,
                    index2,obtieneElemento(q_iter
                    ,2,ahora));
                termino=1;
                escribeCSV=1;
                break;
            }
            else
            {
                if(diferenciaTan<=umbral)
                    fijaElemento(q_new[0],index1,
                        index2,obtieneElemento(
                        q_iter,1,ahora));
                else
                    fijaElemento(q_new[0],index1,
                        index2,0);
                if(diferenciaSag<=umbral)
                    fijaElemento(q_new[1],index1,
                        index2,obtieneElemento(
                        q_iter,2,ahora));
                else
                    fijaElemento(q_new[1],index1,
                        index2,0);
                termino=0;
            }
        }
    }
    else
    {
        if(elem(qInTan,index1,index2)==0)
            fijaElemento(q_new[0],index1,index2
                ,0.0);
        if(elem(qInSag,index1,index2)==0)
            fijaElemento(q_new[1],index1,index2
                ,0.0);
```

```
        escribeCSV=0;
        reallocFallido=0;
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
    guardaSpotIteraciones(spotsTan[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoTan,termino);
    guardaSpotIteraciones(spotsSag[cuenta 1],
        cuenta,creall(obtieneElemento(epsilon1,1,
        index1)),creall(obtieneElemento(epsilon2
        ,1,index2)),tipoSag,termino);
    if(escribeCSV==1)
    {
        guardaVariaciones(spotsTan,
            iteracionActualTan,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),tipoTan,subCarpetaTan,termino
            );
        guardaVariaciones(spotsSag,
            iteracionActualSag,cuenta,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),tipoSag,subCarpetaSag,termino
            );
    }
    if(termino==0)
        printf("Cavidad en X Kerr, EM1: No cumple
            con el umbral de %Le para epsilon1=%
            Le, epsilon2=%Le.\nError relativo
            tangencial: %Le\nError relativo
            sagital: %Le\n",umbral,creall(
            obtieneElemento(epsilon1,1,index1)),
            creall(obtieneElemento(epsilon2,1,
            index2)),diferenciaTan,diferenciaSag);
    else
```

```
                printf("Cavidad en X Kerr, EM1: Cumple
                    con el umbral de %Le para epsilon1=%Le
                    , epsilon2=%Le.\nError relativo
                    tangencial: %Le\nError relativo
                    sagital: %Le\n",umbral,creall(
                    obtieneElemento(epsilon1,1,index1)),
                    creall(obtieneElemento(epsilon2,1,
                    index2)),diferenciaTan,diferenciaSag);
            free(spotsTan);
            free(spotsSag);
            free(iteracionActualTan);
            free(iteracionActualSag);
        }
    }
    // Limpieza
    borraMatriz(q_iter);
    borraMatriz(EM1tan1);
    borraMatriz(EM1tan2);
    borraMatriz(EM1tan3);
    borraMatriz(EM1tan4);
    borraMatriz(EM1tan5);
    borraMatriz(EM1tan6);
    borraMatriz(EM1tan7);
    borraMatriz(EM1tan8);
    borraMatriz(EM1tan9);
    borraMatriz(EM1tan10);
    borraMatriz(EM1tan11);
    borraMatriz(EM1tan12);
    borraMatriz(EM1tan13);
    borraMatriz(EM1tan14);
    borraMatriz(EM1tan15);
    borraMatriz(EM1tan16);
    borraMatriz(EM1tan17);
    borraMatriz(EM1sag1);
    borraMatriz(EM1sag2);
    borraMatriz(EM1sag3);
    borraMatriz(EM1sag4);
    borraMatriz(EM1sag5);
    borraMatriz(EM1sag6);
    borraMatriz(EM1sag7);
    borraMatriz(EM1sag8);
    borraMatriz(EM1sag9);
    borraMatriz(EM1sag10);
    borraMatriz(EM1sag11);
    borraMatriz(EM1sag12);
    borraMatriz(EM1sag13);
```

```
    borraMatriz(EM1sag14);
    borraMatriz(EM1sag15);
    borraMatriz(EM1sag16);
    borraMatriz(EM1sag17);
    return q_new;
}

matriz ** propNoLinealEM2AstigmaticoKerr(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser,\
                                        long double lambda0,
                                            matriz *epsilon1,
                                            matriz *epsilon2,
                                            int iteraciones,
                                            int pasosKerr,
                                            long double umbral
                                            )
{
    // Variables
    //imprimeMatriz(q_in);
    long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s;
    _Bool termino=0;
    _Bool escribeCSV=0;
    _Bool reallocFallido=0;
    long double *spotsTan, *spotsSag;
    int * iteracionActualTan, *iteracionActualSag;
    int cuenta=0;
    char tipoTan[]="EM2Tan_AcopladoKerr", tipoSag[]="
        EM2Sag_AcopladoKerr";
    char subCarpetaTan[]="EM2Tan_matAcopKerr";
    char subCarpetaSag[]="EM2Sag_matAcopKerr";
```

```
long double wIterTanViejo=0, wIterTanNuevo=0,
    wIterSagViejo=0, wIterSagNuevo=0, diferenciaTan=0,
     diferenciaSag=0; // Umbral no porcentual
assert(strlen(conjugado_corto)==3);
assert(strlen(conjugado_largo)==3);
anguloLineal(conjugado_corto,conjugado_largo,L,n0,L1,
    L2,f1,f2,angulos);
delta1=distanciaCristal(conjugado_corto,f1,L1,L,n0,
    angulos[0]);
delta2=distanciaCristal(conjugado_largo,f2,L2,L,n0,
    angulos[1]);

// Cálculo de distancias focales
f1t=f1*cos(angulos[0]);
f2t=f2*cos(angulos[1]);
f1s=f1/cos(angulos[0]);
f2s=f2/cos(angulos[1]);

matriz ** q_new=(matriz**)calloc(2,sizeof(matriz*));
    // Espacio para dos matrices
q_new[0]=nuevaMatriz(qInTan>filas,qInTan>columnas);
     // Creando matriz tangencial
q_new[1]=nuevaMatriz(qInSag>filas,qInSag>columnas);
     // Creando matriz tangencial
long double complex *qPropTan=(long double complex*)
    calloc(6,sizeof(long double complex)); //
    Parámetros para propagación
long double complex *qPropSag=(long double complex*)
    calloc(6,sizeof(long double complex));

// Crea matrices para la propagación

matriz *EM2tan1,*EM2tan2,*EM2tan3,*EM2tan4,*EM2tan5,*
    EM2tan6,*EM2tan7,*EM2tan8,*EM2tan9,*EM2tan10,*
    EM2tan11,*EM2tan12,*EM2tan13,*EM2tan14,*EM2tan15,*
    EM2tan16,*EM2tan17;
matriz *EM2sag1,*EM2sag2,*EM2sag3,*EM2sag4,*EM2sag5,*
    EM2sag6,*EM2sag7,*EM2sag8,*EM2sag9,*EM2sag10,*
    EM2sag11,*EM2sag12,*EM2sag13,*EM2sag14,*EM2sag15,*
    EM2sag16,*EM2sag17;

// EM2, Caso tangencial
EM2tan1=llenaMatriz(1,L2,0,1);
EM2tan2=llenaMatriz(1,0, 1/f2t,1);
EM2tan3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
```

```
EM2tan4=llenaMatriz(n0,0,0,1/n0);
EM2tan5=llenaMatriz(1/n0,0,0,n0);
EM2tan6=nuevaMatriz(2,2);//EM2tan5 pendiente
EM2tan7=llenaMatriz(1,0, 1/f1t,1);
EM2tan8=llenaMatriz(1,L1,0,1);
EM2tan9=llenaMatriz(1,0,0,1);
EM2tan10=llenaMatriz(1,L1,0,1);
EM2tan11=llenaMatriz(1,0, 1/f1t,1);
EM2tan12=nuevaMatriz(2,2);      //EM2tan11 pendiente
EM2tan13=llenaMatriz(n0,0,0,1/n0);
EM2tan14=llenaMatriz(1/n0,0,0,n0);
EM2tan15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2tan16=llenaMatriz(1,0, 1/f2t,1);
EM2tan17=llenaMatriz(1,L2,0,1);
// EM1, Caso saggencial
EM2sag1=llenaMatriz(1,L2,0,1);
EM2sag2=llenaMatriz(1,0, 1/f2s,1);
EM2sag3=nuevaMatriz(2,2); //llenaMatriz(1,delta1,0,1)
    ;
EM2sag4=llenaMatriz(1,0,0,1);
EM2sag5=llenaMatriz(1,0,0,1);
EM2sag6=nuevaMatriz(2,2);//EM2sag5 pendiente
EM2sag7=llenaMatriz(1,0, 1/f1s,1);
EM2sag8=llenaMatriz(1,L1,0,1);
EM2sag9=llenaMatriz(1,0,0,1);
EM2sag10=llenaMatriz(1,L1,0,1);
EM2sag11=llenaMatriz(1,0, 1/f1s,1);
EM2sag12=nuevaMatriz(2,2);      //EM2sag11 pendiente
EM2sag13=llenaMatriz(1,0,0,1);
EM2sag14=llenaMatriz(1,0,0,1);
EM2sag15=nuevaMatriz(2,2); //llenaMatriz(1,delta1
    ,0,1);
EM2sag16=llenaMatriz(1,0, 1/f2s,1);
EM2sag17=llenaMatriz(1,L2,0,1);


// Ciclo
matriz * q_iter = nuevaMatriz(2,iteraciones); // 1
    para tangencial, 2 para sagital
for(int index1=1;index1<=epsilon1 >columnas;index1++)
{
     for(int index2=1;index2<=epsilon2 >columnas;
         index2++)
     {
          spotsTan=(long double*)calloc(1,sizeof(long
```

```
    double));
spotsSag=(long double*)calloc(1,sizeof(long
    double));
iteracionActualTan=(int *)calloc(1,sizeof(int
    ));
iteracionActualSag=(int *)calloc(1,sizeof(int
    ));
cuenta=1;
iteracionActualTan[0]=0;
iteracionActualSag[0]=0;
spotsTan[0]=spot_q(obtieneElemento(qInTan,
    index1,index2),1,lambda0);
spotsSag[0]=spot_q(obtieneElemento(qInSag,
    index1,index2),1,lambda0);
// Llenando matrices variables:
llenaMatrizSinCrear(EM2tan3,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM2tan6,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM2tan12,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM2tan15,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM2sag3,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);
llenaMatrizSinCrear(EM2sag6,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM2sag12,1,delta1+creall(
    obtieneElemento(epsilon1,1,index1)),0,1);
llenaMatrizSinCrear(EM2sag15,1,delta2+creall(
    obtieneElemento(epsilon2,1,index2)),0,1);

if((elem(qInTan,index1,index2)!=0)&&(elem(
    qInSag,index1,index2)!=0)) // Si no son
    cero, realiza rutina
{
     fijaElemento(q_iter,1,1,obtieneElemento(
         qInTan,index1,index2));
     fijaElemento(q_iter,2,1,obtieneElemento(
         qInSag,index1,index2));
     spotsTan=(long double*)calloc(1,sizeof(
         long double));
     spotsSag=(long double*)calloc(1,sizeof(
         long double));
     iteracionActualTan=(int *)calloc(1,sizeof
         (int));
```

```
iteracionActualSag=(int *)calloc(1,sizeof
    (int));
cuenta=1;
iteracionActualTan[0]=0;
iteracionActualSag[0]=0;
spotsTan[0]=spot_q(obtieneElemento(qInTan
    ,index1,index2),1,lambda0);
spotsSag[0]=spot_q(obtieneElemento(qInSag
    ,index1,index2),1,lambda0);
escribeCSV=1;
for(;;)
{
    int ahora=1;
    int siguiente=2;
    matriz * Prop1Tan_piezas[]={EM2tan4,
        EM2tan3,EM2tan2,EM2tan1};
    matriz * Prop1Tan=
        variasMultMatriciales(
        Prop1Tan_piezas,4);
    matriz * Prop1Sag_piezas[]={EM2sag4,
        EM2sag3,EM2sag2,EM2sag1};
    matriz * Prop1Sag=
        variasMultMatriciales(
        Prop1Sag_piezas,4);
    // Almacenando primer q
    qPropTan[0]=obtieneElemento(q_iter,1,
        ahora);
    qPropSag[0]=obtieneElemento(q_iter,2,
        ahora);
    // Cálculo de q2
    qPropTan[1]=prop_q(Prop1Tan,qPropTan
        [0],1,n0);
    qPropSag[1]=prop_q(Prop1Sag,qPropSag
        [0],1,n0);
    // Cálculo de q3
    propAstigmatica *q3=
        propagacionKerrAcoplada(pasosKerr,
        L,n0,n2,qPropTan[1],qPropSag[1],
        chi,kth,Cp,rho,dn_dv,P_laser,
        lambda0,vector,0);
    qPropTan[2]=q3>qTan;
    qPropSag[2]=q3>qSag;
    free(q3);
    // Más matrices
    matriz * Prop2Tan_piezas[]={EM2tan13,
        EM2tan12,EM2tan11,EM2tan10,EM2tan9
```

```
    ,EM2tan8,EM2tan7,EM2tan6,EM2tan5};
     // 5 a 13
matriz * Prop2Tan=
    variasMultMatriciales(
    Prop2Tan_piezas,9);
matriz * Prop2Sag_piezas[]={EM2sag13,
    EM2sag12,EM2sag11,EM2sag10,EM2sag9
    ,EM2sag8,EM2sag7,EM2sag6,EM2sag5};
     // 5 a 13
matriz * Prop2Sag=
    variasMultMatriciales(
    Prop2Sag_piezas,9);
// Cálculo de q4
qPropTan[3]=prop_q(Prop2Tan,qPropTan
    [2],n0,n0);
qPropSag[3]=prop_q(Prop2Sag,qPropSag
    [2],n0,n0);
// Propagación de regreso Kerr,
    cálculo q5
propAstigmatica *q5=
    propagacionKerrAcoplada(pasosKerr,
    L,n0,n2,qPropTan[3],qPropSag[3],
    chi,kth,Cp,rho,dn_dv,P_laser,
    lambda0,vector,1);
qPropTan[4]=q5>qTan;
qPropSag[4]=q5>qSag;
free(q5);
// Más matrices
matriz * Prop3Tan_piezas[]={EM2tan17,
    EM2tan16,EM2tan15,EM2tan14}; // 14
     a 17
matriz * Prop3Tan=
    variasMultMatriciales(
    Prop3Tan_piezas,4);
matriz * Prop3Sag_piezas[]={EM2sag17,
    EM2sag16,EM2sag15,EM2sag14}; // 14
     a 17
matriz * Prop3Sag=
    variasMultMatriciales(
    Prop3Sag_piezas,4);
// Cálculo de q6
qPropTan[5]=prop_q(Prop3Tan,qPropTan
    [4],n0,1);
qPropSag[5]=prop_q(Prop3Sag,qPropSag
    [4],n0,1);
// Guarda elementos
```

```
fijaElemento(q_iter,1,ahora,qPropTan
    [5]);
fijaElemento(q_iter,2,ahora,qPropSag
    [5]);

// Limpieza local
Prop1Tan=borraMatriz(Prop1Tan);
Prop2Tan=borraMatriz(Prop2Tan);
Prop3Tan=borraMatriz(Prop3Tan);
Prop1Sag=borraMatriz(Prop1Sag);
Prop2Sag=borraMatriz(Prop2Sag);
Prop3Sag=borraMatriz(Prop3Sag);

// Calculando spots y comparación
wIterTanViejo=spot_q(qPropTan[0],1,
    lambda0);
wIterTanNuevo=spot_q(qPropTan[5],1,
    lambda0);
wIterSagViejo=spot_q(qPropSag[0],1,
    lambda0);
wIterSagNuevo=spot_q(qPropSag[5],1,
    lambda0);

if(cuenta>iteraciones) break; // Por
     si acaso
++cuenta;
//printf("%i\n",cuenta);
void *tmp_sptTan, *tmp_sptSag, *
    tmp_iterTan, *tmp_iterSag;
tmp_sptTan=(long double*)realloc(
    spotsTan,cuenta*sizeof(long double
    ));
tmp_sptSag=(long double*)realloc(
    spotsSag,cuenta*sizeof(long double
    ));
tmp_iterTan=(int *)realloc(
    iteracionActualTan,cuenta*sizeof(
    int));
tmp_iterSag=(int *)realloc(
    iteracionActualSag,cuenta*sizeof(
    int));
if(tmp_sptTan!=NULL && tmp_sptSag!=
    NULL && tmp_iterTan!=NULL &&
    tmp_iterSag!=NULL)
{
     spotsTan=tmp_sptTan;
```

```
        spotsSag=tmp_sptSag;
        iteracionActualTan=tmp_iterTan;
        iteracionActualSag=tmp_iterSag;
        spotsTan[cuenta_1]=wIterTanNuevo;
        spotsSag[cuenta_1]=wIterSagNuevo;
        iteracionActualTan[cuenta_1]=
            cuenta_1;
        iteracionActualSag[cuenta_1]=
            cuenta_1;
    }
    else
    {
        reallocFallido=1;
        break;
    }
    if(wIterTanViejo>0.1||wIterTanNuevo
        >0.1||wIterSagViejo>0.1||
        wIterSagNuevo>0.1 ||\
        isnan(wIterTanViejo)||isnan(
            wIterTanNuevo)||isnan(
            wIterSagViejo)||isnan(
            wIterSagNuevo))
        {
            termino=0;
            escribeCSV=1;
            fijaElemento(q_new[0],index1,
                index2,0);
            fijaElemento(q_new[1],index1,
                index2,0);
            break;
        }
    diferenciaTan=fabsl((wIterTanNuevo
        wIterTanViejo)/wIterTanNuevo)
        *100.00;
    diferenciaSag=fabsl((wIterSagNuevo
        wIterSagViejo)/wIterSagNuevo)
        *100.00;
    if(diferenciaTan<=umbral&&
        diferenciaSag<=umbral) // Termina
        la iteración
    {
        fijaElemento(q_new[0],index1,
            index2,obtieneElemento(q_iter
            ,1,ahora));
        fijaElemento(q_new[1],index1,
            index2,obtieneElemento(q_iter
```

```
                    ,2,ahora));
                termino=1;
                escribeCSV=1;
                break;
            }
            else
            {
                if(diferenciaTan<=umbral)
                    fijaElemento(q_new[0],index1,
                        index2,obtieneElemento(
                        q_iter,1,ahora));
                else
                    fijaElemento(q_new[0],index1,
                        index2,0);
                if(diferenciaSag<=umbral)
                    fijaElemento(q_new[1],index1,
                        index2,obtieneElemento(
                        q_iter,2,ahora));
                else
                    fijaElemento(q_new[1],index1,
                        index2,0);
                termino=0;
            }
        }
    }
    else
    {
        if(elem(qInTan,index1,index2)==0)
            fijaElemento(q_new[0],index1,index2
                ,0.0);
        if(elem(qInSag,index1,index2)==0)
            fijaElemento(q_new[1],index1,index2
                ,0.0);
        escribeCSV=0;
        reallocFallido=0;
    }
    if(reallocFallido==1)
    {
        termino=0;
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
        printf("Se acabó la memoria\n");
        reallocFallido=0;
    }
```

```
        guardaSpotIteraciones(spotsTan[cuenta 1],
            cuenta,creall(obtieneElemento(epsilon1,1,
            index1)),creall(obtieneElemento(epsilon2
            ,1,index2)),tipoTan,termino);
        guardaSpotIteraciones(spotsSag[cuenta 1],
            cuenta,creall(obtieneElemento(epsilon1,1,
            index1)),creall(obtieneElemento(epsilon2
            ,1,index2)),tipoSag,termino);
        if(escribeCSV==1)
        {
            guardaVariaciones(spotsTan,
                iteracionActualTan,cuenta,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),tipoTan,subCarpetaTan,termino
                );
            guardaVariaciones(spotsSag,
                iteracionActualSag,cuenta,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),tipoSag,subCarpetaSag,termino
                );
        }
        if(termino==0)
            printf("Cavidad en X Kerr, EM2: No cumple
                con el umbral de %Le para epsilon1=%
                Le, epsilon2=%Le.\nError relativo
                tangencial: %Le\nError relativo
                sagital: %Le\n",umbral,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),diferenciaTan,diferenciaSag);
        else
            printf("Cavidad en X Kerr, EM2: Cumple
                con el umbral de %Le para epsilon1=%Le
                , epsilon2=%Le.\nError relativo
                tangencial: %Le\nError relativo
                sagital: %Le\n",umbral,creall(
                obtieneElemento(epsilon1,1,index1)),
                creall(obtieneElemento(epsilon2,1,
                index2)),diferenciaTan,diferenciaSag);
        free(spotsTan);
        free(spotsSag);
        free(iteracionActualTan);
        free(iteracionActualSag);
    }
```

```
    }
    // Limpieza
    free(qPropTan);
    free(qPropSag);
    borraMatriz(q_iter);
    borraMatriz(EM2tan1);
    borraMatriz(EM2tan2);
    borraMatriz(EM2tan3);
    borraMatriz(EM2tan4);
    borraMatriz(EM2tan5);
    borraMatriz(EM2tan6);
    borraMatriz(EM2tan7);
    borraMatriz(EM2tan8);
    borraMatriz(EM2tan9);
    borraMatriz(EM2tan10);
    borraMatriz(EM2tan11);
    borraMatriz(EM2tan12);
    borraMatriz(EM2tan13);
    borraMatriz(EM2tan14);
    borraMatriz(EM2tan15);
    borraMatriz(EM2tan16);
    borraMatriz(EM2tan17);
    borraMatriz(EM2sag1);
    borraMatriz(EM2sag2);
    borraMatriz(EM2sag3);
    borraMatriz(EM2sag4);
    borraMatriz(EM2sag5);
    borraMatriz(EM2sag6);
    borraMatriz(EM2sag7);
    borraMatriz(EM2sag8);
    borraMatriz(EM2sag9);
    borraMatriz(EM2sag10);
    borraMatriz(EM2sag11);
    borraMatriz(EM2sag12);
    borraMatriz(EM2sag13);
    borraMatriz(EM2sag14);
    borraMatriz(EM2sag15);
    borraMatriz(EM2sag16);
    borraMatriz(EM2sag17);
    return q_new;
}
```

no_linealMatAstTerm.h

```
#ifndef NO_LINEALMATASTTERM_H_INCLUDED
#define NO_LINEALMATASTTERM_H_INCLUDED
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include "matrices.h"
#include "lineal.h"
#include "termico.h"
#include "no_lineal.h"

typedef struct
{
    long double complex qTan;
    long double complex qSag;
}propAstigmatica;

// Original
propAstigmatica * propagacionKerrAcoplada(int pasos, long
    double L, long double n0,long double n2, long double
   complex qInTan, \
                                                long double
                                                   complex
                                                   qInSag,
                                                   long
                                                   double chi
                                                   , long
                                                   double kth
                                                   , long
                                                   double Cp,
                                                    \
                                                long double
                                                   rho, long
                                                   double
                                                   dn_dv,long
                                                    double
                                                   P_laser,
                                                   long
                                                   double
                                                   lambda0, \
                                                ajusteTemperaturaCristal
                                                    *
                                                   vectorPlano
                                                   , _Bool
                                                   ladoBombeo
                                                   );
```

```
matriz ** propNoLinealEM1Astigmatico(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser, \
                                        long double lambda0,
                                            matriz *epsilon1,
                                            matriz *epsilon2,
                                            int iteraciones,
                                            int pasosKerr,
                                            long double umbral
                                            );

matriz ** propNoLinealEM2Astigmatico(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                        char *conjugado_corto
                                            ,char *
                                            conjugado_largo,
                                            long double L1,
                                            long double L2, \
                                        long double L, long
                                            double f1, long
                                            double f2, long
                                            double n0, long
                                            double n2, \
                                        long double chi, long
                                             double kth, long
                                            double Cp, \
                                        long double rho,long
                                            double dn_dv,long
                                            double P_laser, \
                                        long double lambda0,
                                            matriz *epsilon1,
```

```
                                        matriz *epsilon2,
                                        int iteraciones,
                                        int pasosKerr,
                                        long double umbral
                                        );

matriz ** propNoLinealAnilloRuta1Astigmatico(matriz *
    qInTan, matriz *qInSag, ajusteTemperaturaCristal *
    vector, \
                                    char *conjugado_corto
                                        ,char *
                                        conjugado_largo,
                                        long double a,
                                        long double b,
                                        long double c,\
                                    long double L, long
                                        double f1, long
                                        double f2, long
                                        double n0, long
                                        double n2, \
                                    long double chi, long
                                         double kth, long
                                        double Cp, \
                                    long double rho,long
                                        double dn_dv,long
                                        double P_laser, \
                                    long double lambda0,
                                        matriz *epsilon1,
                                        matriz *epsilon2,
                                        int iteraciones,
                                        int pasosKerr,
                                        long double umbral
                                        );

matriz ** propNoLinealAnilloRuta2Astigmatico(matriz *
    qInTan, matriz *qInSag, ajusteTemperaturaCristal *
    vector, \
                                    char *conjugado_corto
                                        ,char *
                                        conjugado_largo,
                                        long double a,
                                        long double b,
                                        long double c,\
                                    long double L, long
                                        double f1, long
                                        double f2, long
```

```
                                        double n0, long
                                        double n2, \
                                    long double chi, long
                                         double kth, long
                                        double Cp, \
                                    long double rho,long
                                        double dn_dv,long
                                        double P_laser, \
                                    long double lambda0,
                                        matriz *epsilon1,
                                        matriz *epsilon2,
                                        int iteraciones,
                                        int pasosKerr,
                                        long double umbral
                                        );

matriz ** propNoLinealEM1AstigmaticoKerr(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                    char *conjugado_corto
                                        ,char *
                                        conjugado_largo,
                                        long double L1,
                                        long double L2, \
                                    long double L, long
                                        double f1, long
                                        double f2, long
                                        double n0, long
                                        double n2, \
                                    long double chi, long
                                         double kth, long
                                        double Cp, \
                                    long double rho,long
                                        double dn_dv,long
                                        double P_laser,\
                                    long double lambda0,
                                        matriz *epsilon1,
                                        matriz *epsilon2,
                                        int iteraciones,
                                        int pasosKerr,
                                        long double umbral
                                        );

matriz ** propNoLinealEM2AstigmaticoKerr(matriz *qInTan,
    matriz *qInSag, ajusteTemperaturaCristal *vector, \
                                    char *conjugado_corto
                                        ,char *
```

```
                                        conjugado_largo,
                                        long double L1,
                                        long double L2, \
                                    long double L, long
                                        double f1, long
                                        double f2, long
                                        double n0, long
                                        double n2, \
                                    long double chi, long
                                         double kth, long
                                        double Cp, \
                                    long double rho,long
                                        double dn_dv,long
                                        double P_laser,\
                                    long double lambda0,
                                        matriz *epsilon1,
                                        matriz *epsilon2,
                                        int iteraciones,
                                        int pasosKerr,
                                        long double umbral
                                        );

#endif // NO_LINEALMATASTTERM_H_INCLUDED
```

A.5.8. propGrafica.c y propGrafica.h

propGrafica.c

```
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include "matrices.h"
#include "lineal.h"
#include "no_lineal.h"
#include "termico.h"


/* Estructura de vector */

typedef struct
{
    int numeroElementos;
    long double posKerrIn; // Posición donde el haz entra
        al cristal
    long double posKerrOut; // Posición donde el haz sale
        del cristal
    long double posKerrMid; // Sección media del cristal.
```

```
    long double * spotCW; // Radio del haz
    long double * spotML; // Radio del haz
    long double * pos; // Posición en cavidad.
}vectorDato;

void escribeData(vectorDato *datos, char* filenameCW,
    char* filenameML)
{
    FILE *archivoCW, *archivoML;
    archivoCW=fopen(filenameCW,"w");
    archivoML=fopen(filenameML,"w");
    int i=0;
    for(i=0;i<datos>numeroElementos 1;i++)
    {
        fprintf(archivoCW,"%Le %Le\n",datos>pos[i],datos
            >spotCW[i]);
        fprintf(archivoML,"%Le %Le\n",datos>pos[i],datos
            >spotML[i]);
    }
    fclose(archivoCW);
    fclose(archivoML);
}

/* Escribe a gnuplot  Todos los datos. */
void gnuplotEscribe(vectorDato *EM1tan, vectorDato *
    EM1sag, vectorDato *EM2tan, vectorDato *EM2sag)
{
    // Nombres de archivo
    char* scriptFile="script.plot";
    char* datosEM1tanCW="datosEM1tanCW.dat";
    char* datosEM1sagCW="datosEM1sagCW.dat";
    char* datosEM2tanCW="datosEM2tanCW.dat";
    char* datosEM2sagCW="datosEM2sagCW.dat";
    char* datosEM1tanML="datosEM1tanML.dat";
    char* datosEM1sagML="datosEM1sagML.dat";
    char* datosEM2tanML="datosEM2tanML.dat";
    char* datosEM2sagML="datosEM2sagML.dat";

    // Creación de archivos de datos.
    escribeData(EM1tan, datosEM1tanCW, datosEM1tanML);
    escribeData(EM1sag, datosEM1sagCW, datosEM1sagML);
    escribeData(EM2tan, datosEM2tanCW, datosEM2tanML);
    escribeData(EM2sag, datosEM2sagCW, datosEM2sagML);

    FILE *script, *gnuplotPipe;
    script=fopen(scriptFile,"w");
```

```
//Encabezado
fprintf(script,"set encoding utf8\n");
fprintf(script,"set terminal pdf enhanced solid size
   6in,4in font \"Bookman URW,16\"\n");
fprintf(script,"set datafile separator \" \"\n");
// Conf. EM1 tan
fprintf(script,"set title \"Propagación tangencial
   desde EM_1\\n{/*0.8 $\epsilon$_1=1 [mm], $\
   epsilon$_2=0.9 [mm]}\" font \"Bookman URW,16\"
   offset 0,0\n");
fprintf(script,"set output \"EM1tan.pdf\"\n");
fprintf(script,"set multiplot\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set size 1,1\n");
fprintf(script,"set bmargin at screen 0.2\n");
fprintf(script,"set rmargin at screen 0.93\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set key font \"Bookman URW,16\"\n");
fprintf(script,"set xlabel \"Camino óptico [m]\"\n");
fprintf(script,"set ylabel \"Radio de haz $\omega$_t
   [m]\"\n");
fprintf(script,"set xrange [*:*]\n");
fprintf(script,"set xtics autofreq font \"Bookman URW
   ,16\"\n");
fprintf(script,"set yrange [*:*]\n");
fprintf(script,"set ytics 2e 4 font \"Bookman URW
   ,16\"\n");
fprintf(script,"set format x \"%%.2g\"\n");
fprintf(script,"set format y \"%%.1e\"\n");
fprintf(script, "set label \'A: Interfaz Ti:Zafiro\'
   at 0.2,4e 4\n");
fprintf(script, "set label \'B: Sección media\' at
   0.2,3e 4\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
   title \"Emisión continua\", \"%s\" using 1:2 with
   lines title \"Emisión pulsada\"\n",datosEM1tanCW,
   datosEM1tanML);

// Zoom
fprintf(script,"set size 0.45,0.2\n");
fprintf(script,"set bmargin at screen 0.3\n");
fprintf(script,"set rmargin at screen 0.88\n");
fprintf(script,"set origin 0.5,0.4\n");
fprintf(script,"set title \"Ti:Zafiro\" offset 0,1\n
   ");
fprintf(script,"set xrange [%f:%f]\n",1.045,1.065);
```

```
fprintf(script,"set yrange [%f:%f]\n",0.0,100e6);
fprintf(script,"set xtics (%f,%f) font \"Bookman URW
    ,16\"\n",1.045,1.065);
fprintf(script,"set ytics (%f,%f) font \"Bookman URW
    ,16\"\n",0.0,100e6);
fprintf(script,"set xlabel \"\"\n");
fprintf(script,"set ylabel \"\"\n");
fprintf(script,"set format x \"%%.4g\"\n");
fprintf(script,"unset key\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM1tanCW,
    datosEM1tanML);
fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1tan>
    posKerrIn,EM1tan>posKerrIn);
fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM1tan
    >posKerrMid,EM1tan>posKerrMid);
fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1tan>
    posKerrOut,EM1tan>posKerrOut);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1tan>posKerrIn);
fprintf(script, "set label \'B\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1tan>posKerrMid);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1tan>posKerrOut);
fprintf(script,"replot\n");
fprintf(script,"unset multiplot\n");

// Conf. em1 sagital
fprintf(script,"unset arrow\n");
fprintf(script,"unset label\n");
fprintf(script,"set title \"Propagación sagital desde
    EM_1\\n{/*0.8 $\epsilon$_1=1 [mm], $\epsilon$_2
    =0.9 [mm]}\" font \"Bookman URW,16\" offset 0,0\n
    ");
fprintf(script,"set output \"EM1sag.pdf\"\n");
fprintf(script,"set multiplot\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set size 1,1\n");
fprintf(script,"set bmargin at screen 0.2\n");
fprintf(script,"set rmargin at screen 0.93\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set key font \"Bookman URW,16\"\n");
```

```
fprintf(script,"set xlabel \"Camino óptico [m]\"\n");
fprintf(script,"set ylabel \"Radio de haz $\omega$_s
    [m]\"\n");
fprintf(script,"set xrange [*:*]\n");
fprintf(script,"set xtics autofreq font \"Bookman URW
    ,16\"\n");
fprintf(script,"set yrange [*:*]\n");
fprintf(script,"set ytics 2e 4 font \"Bookman URW
    ,16\"\n");
fprintf(script,"set format x \"%%.2g\"\n");
fprintf(script,"set format y \"%%.1e\"\n");
fprintf(script, "set label \'A: Interfaz Ti:Zafiro\'
    at 0.2,4e 4\n");
fprintf(script, "set label \'B: Sección media\' at
    0.2,3e 4\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM1sagCW,
    datosEM1sagML);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1sag >
    posKerrIn,EM1sag >posKerrIn);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM1sag
     >posKerrMid,EM1sag >posKerrMid);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1sag >
    posKerrOut,EM1sag >posKerrOut);
// Zoom
fprintf(script,"set size 0.45,0.2\n");
fprintf(script,"set bmargin at screen 0.3\n");
fprintf(script,"set rmargin at screen 0.88\n");
fprintf(script,"set origin 0.5,0.4\n");
fprintf(script,"set title \"Ti:Zafiro\" offset 0,1\n
    ");
fprintf(script,"set xrange [%f:%f]\n",1.045,1.065);
fprintf(script,"set yrange [%f:%f]\n",0.0,100e 6);
fprintf(script,"set xtics (%f,%f) font \"Bookman URW
    ,16\"\n",1.045,1.065);
fprintf(script,"set ytics (%f,%f) font \"Bookman URW
    ,16\"\n",0.0,100e 6);
fprintf(script,"set xlabel \"\"\n");
fprintf(script,"set ylabel \"\"\n");
fprintf(script,"set format x \"%%.4g\"\n");

fprintf(script,"unset key\n");
```

```
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM1sagCW,
    datosEM1sagML);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1sag>
    posKerrIn,EM1sag>posKerrIn);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM1sag
    >posKerrMid,EM1sag>posKerrMid);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM1sag>
    posKerrOut,EM1sag>posKerrOut);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1sag>posKerrIn);
fprintf(script, "set label \'B\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1sag>posKerrMid);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM1sag>posKerrOut);
fprintf(script,"replot\n");
fprintf(script,"unset multiplot\n");

// Conf. EM2 tan
fprintf(script,"unset arrow\n");
fprintf(script,"unset label\n");
fprintf(script,"set title \"Propagación tangencial
    desde EM_2\\n{/*0.8 $\epsilon$_1=1 [mm], $\
    epsilon$_2=0.9 [mm]}\" font \"Bookman URW,16\"
    offset 0,0\n");
fprintf(script,"set output \"EM2tan.pdf\"\n");
fprintf(script,"set multiplot\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set size 1,1\n");
fprintf(script,"set bmargin at screen 0.2\n");
fprintf(script,"set rmargin at screen 0.93\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set key font \"Bookman URW,16\"\n");
fprintf(script,"set xlabel \"Camino óptico [m]\"\n");
fprintf(script,"set ylabel \"Radio de haz $\omega$_t
    [m]\"\n");
fprintf(script,"set xrange [*:*]\n");
fprintf(script,"set xtics autofreq font \"Bookman URW
    ,16\"\n");
fprintf(script,"set yrange [*:*]\n");
fprintf(script,"set ytics 2e 4 font \"Bookman URW
    ,16\"\n");
```

```
fprintf(script,"set format x \"%%.2g\"\n");
fprintf(script,"set format y \"%%.1e\"\n");
fprintf(script, "set label \'A: Interfaz Ti:Zafiro\'
    at 0.2,4e 4\n");
fprintf(script, "set label \'B: Sección media\' at
    0.2,3e 4\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM2tanCW,
    datosEM2tanML);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2tan >
    posKerrIn,EM2tan >posKerrIn);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM2tan
     >posKerrMid,EM2tan >posKerrMid);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2tan >
    posKerrOut,EM2tan >posKerrOut);
// Zoom
fprintf(script,"set size 0.45,0.2\n");
fprintf(script,"set bmargin at screen 0.3\n");
fprintf(script,"set rmargin at screen 0.88\n");
fprintf(script,"set origin 0.5,0.4\n");
fprintf(script,"set title \"Ti:Zafiro\" offset 0,1\n
    ");
fprintf(script,"set xrange [%f:%f]\n",1.045,1.065);
fprintf(script,"set yrange [%f:%f]\n",0.0,100e 6);
fprintf(script,"set xtics (%f,%f) font \"Bookman URW
    ,16\"\n",1.045,1.065);
fprintf(script,"set ytics (%f,%f) font \"Bookman URW
    ,16\"\n",0.0,100e 6);
fprintf(script,"set xlabel \"\"\n");
fprintf(script,"set ylabel \"\"\n");
fprintf(script,"set format x \"%%.4g\"\n");
fprintf(script,"unset key\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM2tanCW,
    datosEM2tanML);
fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2tan >
    posKerrIn,EM2tan >posKerrIn);
fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM2tan
     >posKerrMid,EM2tan >posKerrMid);
```

```
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2tan>
   posKerrOut,EM2tan>posKerrOut);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
   *0.9\n",EM2tan>posKerrIn);
fprintf(script, "set label \'B\' at %Le,GPVAL_Y_MAX
   *0.9\n",EM2tan>posKerrMid);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
   *0.9\n",EM2tan>posKerrOut);
fprintf(script,"replot\n");
fprintf(script,"unset multiplot\n");

// Conf. em2 sagital
fprintf(script,"unset arrow\n");
fprintf(script,"unset label\n");
fprintf(script,"set title \"Propagación sagital desde
    EM_2\\n{/*0.8 $\epsilon$_1=1 [mm], $\epsilon$_2
   =0.9 [mm]}\" font \"Bookman URW,16\" offset 0,0\n
   ");
fprintf(script,"set output \"EM2sag.pdf\"\n");
fprintf(script,"set multiplot\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set size 1,1\n");
fprintf(script,"set bmargin at screen 0.2\n");
fprintf(script,"set rmargin at screen 0.93\n");
fprintf(script,"set origin 0,0\n");
fprintf(script,"set key font \"Bookman URW,16\"\n");
fprintf(script,"set xlabel \"Camino óptico [m]\"\n");
fprintf(script,"set ylabel \"Radio de haz $\omega$_s
   [m]\"\n");
fprintf(script,"set xrange [*:*]\n");
fprintf(script,"set xtics autofreq font \"Bookman URW
   ,16\"\n");
fprintf(script,"set yrange [*:*]\n");
fprintf(script,"set ytics 2e-4 font \"Bookman URW
   ,16\"\n");
fprintf(script,"set format x \"%%.2g\"\n");
fprintf(script,"set format y \"%%.1e\"\n");
fprintf(script, "set label \'A: Interfaz Ti:Zafiro\'
   at 0.2,4e-4\n");
fprintf(script, "set label \'B: Sección media\' at
   0.2,3e-4\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
   title \"Emisión continua\", \"%s\" using 1:2 with
   lines title \"Emisión pulsada\"\n",datosEM2sagCW,
   datosEM2sagML);
```

```
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2sag>
    posKerrIn,EM2sag>posKerrIn);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM2sag
    >posKerrMid,EM2sag>posKerrMid);
//fprintf(script,"set arrow from %Le,0 to %Le,
    GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2sag>
    posKerrOut,EM2sag>posKerrOut);
// Zoom
fprintf(script,"set size 0.45,0.2\n");
fprintf(script,"set bmargin at screen 0.3\n");
fprintf(script,"set rmargin at screen 0.88\n");
fprintf(script,"set origin 0.5,0.4\n");
fprintf(script,"set title \"Ti:Zafiro\" offset 0,1\n
    ");
fprintf(script,"set xrange [%f:%f]\n",1.045,1.065);
fprintf(script,"set yrange [%f:%f]\n",0.0,100e6);
fprintf(script,"set xtics (%f,%f) font \"Bookman URW
    ,16\"\n",1.045,1.065);
fprintf(script,"set ytics (%f,%f) font \"Bookman URW
    ,16\"\n",0.0,100e6);
fprintf(script,"set xlabel \"\"\n");
fprintf(script,"set ylabel \"\"\n");
fprintf(script,"set format x \"%%.4g\"\n");
fprintf(script,"unset key\n");
fprintf(script,"plot \"%s\" using 1:2 with lines
    title \"Emisión continua\", \"%s\" using 1:2 with
    lines title \"Emisión pulsada\"\n",datosEM2sagCW,
    datosEM2sagML);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2sag>
    posKerrIn,EM2sag>posKerrIn);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'blue\'\n",EM2sag
    >posKerrMid,EM2sag>posKerrMid);
fprintf(script,"set arrow from %Le,0 to %Le,
   GPVAL_Y_MAX*0.75 nohead lc rgb \'red\'\n",EM2sag>
    posKerrOut,EM2sag>posKerrOut);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM2sag>posKerrIn);
fprintf(script, "set label \'B\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM2sag>posKerrMid);
fprintf(script, "set label \'A\' at %Le,GPVAL_Y_MAX
    *0.9\n",EM2sag>posKerrOut);
fprintf(script,"replot\n");
```

```
    fprintf(script,"unset multiplot\n");
    fclose(script);
    printf("Fin\n");

}

/* Rutina para gráficar propagación lineal  Conlleva un
    error por la discreti
zación de la propagación de rayos (en lugar de una matriz
     para propagar en espacio,
se utilizan muchas matrices */

void propagacionKerrGrafica(int pasos, long double deltaZ
    , long double n0,long double n2,
                            long double complex qInTan,
                                long double complex
                                qInSag, long double
                                complex *qTan, \
                                        long double
                                            complex *
                                            qSag, long
                                             double *
                                            wTan, long
                                             double *
                                            wSag, long
                                             double
                                            chi, long
                                            double kth
                                            , long
                                            double Cp,
                                             \
                                        long double
                                            rho, long
                                            double
                                            dn_dv,long
                                             double
                                            P_laser,
                                            long
                                            double
                                            lambda0, \
                                        ajusteTemperaturaCristal
                                             *
                                            vectorPlano
                                            , _Bool
                                            ladoBombeo
                                            )
```

```
{
        long double wInTan=spot_q(qInTan,n0,lambda0);
        long double wInSag=spot_q(qInSag,n0,lambda0);
        long double complex At, Bt, Ct, Dt;
        long double complex As, Bs, Cs, Ds;
        matriz *MTan, *MSag;
        MTan=nuevaMatriz(2,2);
        MSag=nuevaMatriz(2,2);
        // Cálculo inicial
        long double n0Prop=n0;
        // Identificar si se propaga desde el lado
            bombeado o no, cambiar coeficientes
        // de ajuste acorde a esto.
        if(ladoBombeo==1)
        {
            // Primer propagación
            wInTan=spot_q(qInTan,n0,lambda0);
            wInSag=spot_q(qInSag,n0,lambda0);
            n0Prop=n0; // Descartados términos lineales
                por ser pequeños.
            // Formación de matrices
            // Factor de parábola=1/h^2
            long double parabola1=2.0*vectorPlano>a1
                [0]*dn_dv/n0Prop+(n2*P_laser)/(n0Prop*M_PI
                *powl(wInTan,3)*wInSag);
            long double parabola2=2.0*vectorPlano>a2
                [0]*dn_dv/n0Prop+(n2*P_laser)/(n0Prop*M_PI
                *powl(wInSag,3)*wInTan);
            // Coeficientes
            At=1;
            Bt=deltaZ/n0Prop;
            Ct=n0Prop*parabola1*deltaZ;
            Dt=1;
            As=1;
            Bs=deltaZ/n0Prop;
            Cs=n0Prop*parabola2*deltaZ;
            Ds=1;
            llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
            llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
            qTan[0]=prop_q(MTan,qInTan,n0Prop,n0Prop);
            qSag[0]=prop_q(MSag,qInSag,n0Prop,n0Prop);
            wTan[0]=spot_q(qTan[0],n0,lambda0);
            wSag[0]=spot_q(qSag[0],n0,lambda0);
            for (int i=0;i<pasos;i++)
            {
                // Cálculo de spots
```

```
                wInTan=spot_q(qTan[i],n0Prop,lambda0);
                wInSag=spot_q(qSag[i],n0Prop,lambda0);
                // Cálculo de plano
                //n0Prop=n0+vectorPlano->a0[i]*dn_dv+n2*
                    P_laser/(M_PI*wTan*wSag)*3.0/4.0; //
                    Factores lineal, térmico y Kerr
                n0Prop=n0; // Descartados términos
                    lineales por ser pequeños.
                // Formación de matrices
                // Factor de parábola=1/h^2
                parabola1 = 2.0*vectorPlano->a1[i]*dn_dv/
                    n0Prop+(n2*P_laser)/(n0Prop*M_PI*powl(
                    wInTan,3)*wInSag);
                parabola2 = 2.0*vectorPlano->a2[i]*dn_dv/
                    n0Prop+(n2*P_laser)/(n0Prop*M_PI*powl(
                    wInSag,3)*wInTan);
                // Coeficientes
                At=1;
                Bt=deltaZ/n0Prop;
                Ct= n0Prop*parabola1*deltaZ;
                Dt=1;
                As=1;
                Bs=deltaZ/n0Prop;
                Cs= n0Prop*parabola2*deltaZ;
                Ds=1;
                llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
                llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
                if(i+1<pasos)
                {
                    qTan[i+1]=prop_q(MTan,qTan[i],n0Prop,
                        n0Prop);
                    qSag[i+1]=prop_q(MSag,qSag[i],n0Prop,
                        n0Prop);
                    wTan[i+1]=spot_q(qTan[i],n0,lambda0);
                    wSag[i+1]=spot_q(qSag[i],n0,lambda0);
                }
            }
        }
        else
        {
            int indiceTermico=pasos-1;
            long double wInTan=spot_q(qInTan,n0Prop,
                lambda0);
            long double wInSag=spot_q(qInSag,n0Prop,
                lambda0);
            // Cálculo de plano
```

```
n0Prop=n0;
// Formación de matrices
// Factor de parábola=1/h^2
long double parabola1 = 2.0*vectorPlano >a1[
    indiceTermico]*dn_dv/n0Prop+(n2*P_laser)/(
    n0Prop*M_PI*powl(wInTan,3)*wInSag);
long double parabola2 = 2.0*vectorPlano >a2[
    indiceTermico]*dn_dv/n0Prop+(n2*P_laser)/(
    n0Prop*M_PI*powl(wInSag,3)*wInTan);
// Coeficientes
At=1;
Bt=deltaZ/n0Prop;
Ct= n0Prop*parabola1*deltaZ;
Dt=1;
As=1;
Bs=deltaZ/n0Prop;
Cs= n0Prop*parabola2*deltaZ;
Ds=1;
llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
qTan[0]=prop_q(MTan,qInTan,n0Prop,n0Prop);
qSag[0]=prop_q(MSag,qInSag,n0Prop,n0Prop);
wTan[0]=spot_q(qTan[0],n0,lambda0);
wSag[0]=spot_q(qSag[0],n0,lambda0);
indiceTermico  ;
for (int i=0;i<pasos;i++)
{
    // Cálculo de spots
    wInTan=spot_q(qTan[i],n0Prop,lambda0);
    wInSag=spot_q(qSag[i],n0Prop,lambda0);
    // Cálculo de plano
    n0Prop=n0;
    // Formación de matrices
    // Factor de parábola=1/h^2
    parabola1 = 2.0*vectorPlano >a1[
        indiceTermico]*dn_dv/n0Prop+(n2*
        P_laser)/(n0Prop*M_PI*powl(wInTan,3)*
        wInSag);
    parabola2 = 2.0*vectorPlano >a2[
        indiceTermico]*dn_dv/n0Prop+(n2*
        P_laser)/(n0Prop*M_PI*powl(wInSag,3)*
        wInTan);
    // Coeficientes
    At=1;
    Bt=deltaZ/n0Prop;
    Ct= n0Prop*parabola1*deltaZ;
```

```
                    Dt=1;
                    As=1;
                    Bs=deltaZ/n0Prop;
                    Cs= n0Prop*parabola2*deltaZ;
                    Ds=1;
                    llenaMatrizSinCrear(MTan,At,Bt,Ct,Dt);
                    llenaMatrizSinCrear(MSag,As,Bs,Cs,Ds);
                    if(i+1<pasos)
                    {
                        qTan[i+1]=prop_q(MTan,qTan[i],n0Prop,
                            n0Prop);
                        qSag[i+1]=prop_q(MSag,qSag[i],n0Prop,
                            n0Prop);
                        wTan[i+1]=spot_q(qTan[i],n0,lambda0);
                        wSag[i+1]=spot_q(qSag[i],n0,lambda0);
                    }
                    indiceTermico   ;
                }
            }
            //Limpieza
            borraMatriz(MTan);
            borraMatriz(MSag);
        }

void graficaPropagacion(char *conjugado_corto, char *
    conjugado_largo, int iteraciones, long double umbral,
    int N, long double Epsilon1, long double Epsilon2,
                        long double lambdaPump, long double
                            n0, long double n2, long double
                            P_laser, long double P_pump, long
                             double chi, long double L,
                        long double kth, long double Cp,
                            long double rho, long double
                            dn_dv, long double ancho, long
                            double alto, long double w_pump_t
                            ,
                        long double w_pump_s, long double
                            nPump, long double lambda0, long
                            double L1, long double L2,
                        long double f1, long double f2)
{
    // Lente térmica   en el futuro se podrá propagar
        desde la fuente láser
    long double complex pT,pS,qT,qS;
     long double angulos[2],delta1,delta2,f1t,f1s,f2t,f2s
         ;
```

```
pT=1/100e 3 I*535e 9/(nPump*M_PI*powl(w_pump_t,2));
qT=1/pT;
pS=1/100e 3 I*535e 9/(nPump*M_PI*powl(w_pump_s,2));
qS=1/pS;
long double alpha=1/1e 2;
int pasos=1000;
int pasosKerr=1000;
long double lambda_relax=1.5; // Valor de relajación
    para el método Liebmann (Gauss Seidel) de sol. de
    EDP.
ajusteTemperaturaCristal *ajuste=vectorAjusteCristal(
    qT,qS,alpha,lambdaPump,n0,P_pump,chi,L,kth,Cp,rho,
    dn_dv,ancho,alto,iteraciones,pasos,umbral,
    lambda_relax,N);
printf("Cálculo térmico terminado\n");
long double paso=(long double)L/pasosKerr;

// Cálculo CW
matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
    epsilon1,*epsilon2;
matriz *wt1_ml, *ws1_ml, *wt2_ml, *ws2_ml, *astigML1,
     *astigML2;

// Llenando epsilon1 y epsilon2
epsilon1=nuevaMatriz(1,1);
epsilon2=nuevaMatriz(1,1); // Columnas
fijaElemento(epsilon1,1,1,Epsilon1);
fijaElemento(epsilon2,1,1,Epsilon2);

// Creando matrices receptoras
wt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
ws1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
wt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
ws2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qs1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qs2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
```

```
matriz **qAstigEM1_ML;
matriz **qAstigEM2_ML;

// Cálculo lineal
calculoLineal("lol.csv",conjugado_corto,
    conjugado_largo,n0,lambda0,L1,L2,L,f1,f2,wt1,ws1,
    wt2,ws2,epsilon1,epsilon2,qt1,qs1,qt2,qs2);

// Cálculo no lineal

qAstigEM1_ML=propNoLinealEM1Astigmatico(qt1,qs1,
    ajuste,conjugado_corto,conjugado_largo,L1,L2,L,f1,
    f2,n0,n2,chi,kth,Cp,rho,dn_dv,P_laser,lambda0,
    epsilon1,epsilon2,iteraciones,pasosKerr,umbral);
qAstigEM2_ML=propNoLinealEM2Astigmatico(qt2,qs2,
    ajuste,conjugado_corto,conjugado_largo,L1,L2,L,f1,
    f2,n0,n2,chi,kth,Cp,rho,dn_dv,P_laser,lambda0,
    epsilon1,epsilon2,iteraciones,pasosKerr,umbral);

// Cálculo de spots
wt1_ml=spot_q_matriz(qAstigEM1_ML[0],1,lambda0);
ws1_ml=spot_q_matriz(qAstigEM1_ML[1],1,lambda0);
wt2_ml=spot_q_matriz(qAstigEM2_ML[0],1,lambda0);
ws2_ml=spot_q_matriz(qAstigEM2_ML[1],1,lambda0);

// Cálculo de distancias
anguloLineal("fin","inf",L,n0,L1,L2,f1,f2,angulos);
delta1=distanciaCristal("fin",f1,L1,L,n0,angulos[0]);
delta2=distanciaCristal("inf",f2,L2,L,n0,angulos[1]);

// Cálculo de distancias focales
f1t=f1*cosl(angulos[0]);
f2t=f2*cosl(angulos[1]);
f1s=f1/cosl(angulos[0]);
f2s=f2/cosl(angulos[1]);

delta1=delta1+Epsilon1;
delta2=delta2+Epsilon2;

// Búsqueda de configuración menos astigmática
int fila1, columna1,fila2, columna2;
long double complex valor1, valor2;

/// Inicio Gráfica
// matrices a usar
```

```
    matriz *propLibre, *espCurv1Tan, *espCurv1Sag, *
        espCurv2Tan, *espCurv2Sag,
        *brewsterEntradaTan, *brewsterSalidaTan, *
            brewsterEntradaSag, *brewsterSalidaSag,
        *pasoCristal;

// Llenado de matrices
propLibre=llenaMatriz(1,paso,0,1);
espCurv1Tan=llenaMatriz(1,0, 1/f1t,1);
espCurv1Sag=llenaMatriz(1,0, 1/f1s,1);
espCurv2Tan=llenaMatriz(1,0, 1/f2t,1);
espCurv2Sag=llenaMatriz(1,0, 1/f2s,1);
brewsterEntradaTan=llenaMatriz(n0,0,0,1/n0);
brewsterSalidaTan=llenaMatriz(1/n0,0,0,n0);
brewsterEntradaSag=llenaMatriz(1,0,0,1);
brewsterSalidaSag=llenaMatriz(1,0,0,1);
pasoCristal=llenaMatriz(1,paso/n0,0,1);

/// Espejo EM1

/// Datos para propagación
long double complex *q0EM1tanCW, *q1EM1tanCW, *
    q2EM1tanCW, *q3EM1tanCW, *q4EM1tanCW; // Vectores
    para almacenar q propagada.
long double *w0EM1tanCW, *w1EM1tanCW, *w2EM1tanCW, *
    w3EM1tanCW, *w4EM1tanCW; // Vectores para spot
long double complex *q0EM1sagCW, *q1EM1sagCW, *
    q2EM1sagCW, *q3EM1sagCW, *q4EM1sagCW; // Vectores
    para almacenar q propagada.
long double *w0EM1sagCW, *w1EM1sagCW, *w2EM1sagCW, *
    w3EM1sagCW, *w4EM1sagCW; // Vectores para spotlong
     double complex *q0EM1tan, *q1EM1tan, *q2EM1tan, *
    q3EM1tan, *q4EM1tan; // Vectores para almacenar q
    propagada.

long double complex *q0EM1tanML, *q1EM1tanML, *
    q2EM1tanML, *q3EM1tanML, *q4EM1tanML; // Vectores
    para almacenar q propagada.
long double *w0EM1tanML, *w1EM1tanML, *w2EM1tanML, *
    w3EM1tanML, *w4EM1tanML; // Vectores para spot
long double complex *q0EM1sagML, *q1EM1sagML, *
    q2EM1sagML, *q3EM1sagML, *q4EM1sagML; // Vectores
    para almacenar q propagada.
long double *w0EM1sagML, *w1EM1sagML, *w2EM1sagML, *
    w3EM1sagML, *w4EM1sagML; // Vectores para spotlong
     double complex *q0EM1tan, *q1EM1tan, *q2EM1tan, *
```

```
    q3EM1tan, *q4EM1tan; // Vectores para almacenar q
    propagada.
// Estimación de tamaño y reserva de memoriade vector
     q0.
int contador=0;
int numeroPasosq0EM1 = L1/paso+1;  // Incluye
    reflexión en espejo curvo

q0EM1tanCW=(long double complex*)calloc(
    numeroPasosq0EM1,sizeof(long double complex));
w0EM1tanCW=(long double*)calloc(numeroPasosq0EM1,
    sizeof(long double));
q0EM1sagCW=(long double complex*)calloc(
    numeroPasosq0EM1,sizeof(long double complex));
w0EM1sagCW=(long double*)calloc(numeroPasosq0EM1,
    sizeof(long double));

q0EM1tanML=(long double complex*)calloc(
    numeroPasosq0EM1,sizeof(long double complex));
w0EM1tanML=(long double*)calloc(numeroPasosq0EM1,
    sizeof(long double));
q0EM1sagML=(long double complex*)calloc(
    numeroPasosq0EM1,sizeof(long double complex));
w0EM1sagML=(long double*)calloc(numeroPasosq0EM1,
    sizeof(long double));


// Propagación en q0EM1

q0EM1tanCW[0]=obtieneElemento(qt1,1,1);
w0EM1tanCW[0]=spot_q(q0EM1tanCW[0],1,lambda0);
q0EM1sagCW[0]=obtieneElemento(qs1,1,1);
w0EM1sagCW[0]=spot_q(q0EM1sagCW[0],1,lambda0);

q0EM1tanML[0]=obtieneElemento(qAstigEM1_ML[0],1,1);
w0EM1tanML[0]=spot_q(q0EM1tanML[0],1,lambda0);
q0EM1sagML[0]=obtieneElemento(qAstigEM1_ML[1],1,1);
w0EM1sagML[0]=spot_q(q0EM1sagML[0],1,lambda0);

for(contador=1;contador<numeroPasosq0EM1 1;contador
    ++)
{
     q0EM1tanCW[contador]=prop_q(propLibre,q0EM1tanCW[
         contador 1],1,1);
     w0EM1tanCW[contador]=spot_q(q0EM1tanCW[contador
         ],1,lambda0);
```

```
        q0EM1sagCW[contador]=prop_q(propLibre,q0EM1sagCW[
            contador-1],1,1);
        w0EM1sagCW[contador]=spot_q(q0EM1sagCW[contador
            ],1,lambda0);

        q0EM1tanML[contador]=prop_q(propLibre,q0EM1tanML[
            contador-1],1,1);
        w0EM1tanML[contador]=spot_q(q0EM1tanML[contador
            ],1,lambda0);
        q0EM1sagML[contador]=prop_q(propLibre,q0EM1sagML[
            contador-1],1,1);
        w0EM1sagML[contador]=spot_q(q0EM1sagML[contador
            ],1,lambda0);
    }
    contador--;
    q0EM1tanCW[contador+1]=prop_q(espCurv1Tan,q0EM1tanCW[
        contador],1,1);
    w0EM1tanCW[contador+1]=spot_q(q0EM1tanCW[contador
        +1],1,lambda0);
    q0EM1sagCW[contador+1]=prop_q(espCurv1Sag,q0EM1sagCW[
        contador],1,1);
    w0EM1sagCW[contador+1]=spot_q(q0EM1sagCW[contador
        +1],1,lambda0);
    q0EM1tanML[contador+1]=prop_q(espCurv1Tan,q0EM1tanML[
        contador],1,1);
    w0EM1tanML[contador+1]=spot_q(q0EM1tanML[contador
        +1],1,lambda0);
    q0EM1sagML[contador+1]=prop_q(espCurv1Sag,q0EM1sagML[
        contador],1,1);
    w0EM1sagML[contador+1]=spot_q(q0EM1sagML[contador
        +1],1,lambda0);

    // Creación de q1
    int numeroPasosq1EM1 = delta1/paso+1; // incluye
        refracción en cristal
    q1EM1tanCW=(long double complex*)calloc(
        numeroPasosq1EM1,sizeof(long double complex));
    w1EM1tanCW=(long double*)calloc(numeroPasosq1EM1,
        sizeof(long double));
    q1EM1sagCW=(long double complex*)calloc(
        numeroPasosq1EM1,sizeof(long double complex));
    w1EM1sagCW=(long double*)calloc(numeroPasosq1EM1,
        sizeof(long double));
    q1EM1tanML=(long double complex*)calloc(
        numeroPasosq1EM1,sizeof(long double complex));
    w1EM1tanML=(long double*)calloc(numeroPasosq1EM1,
```

```
    sizeof(long double));
q1EM1sagML=(long double complex*)calloc(
    numeroPasosq1EM1,sizeof(long double complex));
w1EM1sagML=(long double*)calloc(numeroPasosq1EM1,
    sizeof(long double));
// Propagación en q1EM1tan
q1EM1tanCW[0]=prop_q(propLibre,q0EM1tanCW[contador
    +1],1,1);
w1EM1tanCW[0]=spot_q(q1EM1tanCW[0],1,lambda0);
q1EM1sagCW[0]=prop_q(propLibre,q0EM1sagCW[contador
    +1],1,1);
w1EM1sagCW[0]=spot_q(q1EM1sagCW[0],1,lambda0);
q1EM1tanML[0]=prop_q(propLibre,q0EM1tanML[contador
    +1],1,1);
w1EM1tanML[0]=spot_q(q1EM1tanML[0],1,lambda0);
q1EM1sagML[0]=prop_q(propLibre,q0EM1sagML[contador
    +1],1,1);
w1EM1sagML[0]=spot_q(q1EM1sagML[0],1,lambda0);


for(contador=1;contador<numeroPasosq1EM1 1;contador
    ++)
{
    q1EM1tanCW[contador]=prop_q(propLibre,q1EM1tanCW[
        contador 1],1,1);
    w1EM1tanCW[contador]=spot_q(q1EM1tanCW[contador
        ],1,lambda0);
    q1EM1sagCW[contador]=prop_q(propLibre,q1EM1sagCW[
        contador 1],1,1);
    w1EM1sagCW[contador]=spot_q(q1EM1sagCW[contador
        ],1,lambda0);
    q1EM1tanML[contador]=prop_q(propLibre,q1EM1tanML[
        contador 1],1,1);
    w1EM1tanML[contador]=spot_q(q1EM1tanML[contador
        ],1,lambda0);
    q1EM1sagML[contador]=prop_q(propLibre,q1EM1sagML[
        contador 1],1,1);
    w1EM1sagML[contador]=spot_q(q1EM1sagML[contador
        ],1,lambda0);
}
contador  ;
q1EM1tanCW[contador+1]=prop_q(brewsterEntradaTan,
    q1EM1tanCW[contador],1,n0);
w1EM1tanCW[contador+1]=spot_q(q1EM1tanCW[contador+1],
    n0,lambda0);
q1EM1sagCW[contador+1]=prop_q(brewsterEntradaSag,
```

```
    q1EM1sagCW[contador],1,n0);
w1EM1sagCW[contador+1]=spot_q(q1EM1sagCW[contador+1],
    n0,lambda0);
q1EM1tanML[contador+1]=prop_q(brewsterEntradaTan,
    q1EM1tanML[contador],1,n0);
w1EM1tanML[contador+1]=spot_q(q1EM1tanML[contador+1],
    n0,lambda0);
q1EM1sagML[contador+1]=prop_q(brewsterEntradaSag,
    q1EM1sagML[contador],1,n0);
w1EM1sagML[contador+1]=spot_q(q1EM1sagML[contador+1],
    n0,lambda0);
int posProp=contador+1;

// Creación de q2
int numeroPasosq2EM1 = pasos+1; // incluye refracción
     hacia afuera del cristal
q2EM1tanCW=(long double complex*)calloc(
    numeroPasosq2EM1,sizeof(long double complex));
w2EM1tanCW=(long double *)calloc(numeroPasosq2EM1,
    sizeof(long double));
q2EM1sagCW=(long double complex*)calloc(
    numeroPasosq2EM1,sizeof(long double complex));
w2EM1sagCW=(long double *)calloc(numeroPasosq2EM1,
    sizeof(long double));
q2EM1tanML=(long double complex*)calloc(
    numeroPasosq2EM1,sizeof(long double complex));
w2EM1tanML=(long double *)calloc(numeroPasosq2EM1,
    sizeof(long double));
q2EM1sagML=(long double complex*)calloc(
    numeroPasosq2EM1,sizeof(long double complex));
w2EM1sagML=(long double *)calloc(numeroPasosq2EM1,
    sizeof(long double));


/// Propagación en cristal

q2EM1tanCW[0]=prop_q(pasoCristal,q1EM1tanCW[contador
    +1],n0,n0);
w2EM1tanCW[0]=spot_q(q2EM1tanCW[0],n0,lambda0);
q2EM1sagCW[0]=prop_q(pasoCristal,q1EM1sagCW[contador
    +1],n0,n0);
w2EM1sagCW[0]=spot_q(q2EM1sagCW[0],n0,lambda0);
//q2EM1tanML[0]=prop_q(pasoCristal,q1EM1tanML[
    contador+1],n0,n0);
//w2EM1tanML[0]=spot_q(q2EM1tanML[0],n0,lambda0);
//q2EM1sagML[0]=prop_q(pasoCristal,q1EM1sagML[
```

```
    contador+1],n0,n0);
//w2EM1sagML[0]=spot_q(q2EM1sagML[0],n0,lambda0);
for(contador=1;contador<numeroPasosq2EM1_1;contador
    ++)
{
    q2EM1tanCW[contador]=prop_q(pasoCristal,
        q2EM1tanCW[contador-1],n0,n0);
    w2EM1tanCW[contador]=spot_q(q2EM1tanCW[contador],
        n0,lambda0);
    q2EM1sagCW[contador]=prop_q(pasoCristal,
        q2EM1sagCW[contador-1],n0,n0);
    w2EM1sagCW[contador]=spot_q(q2EM1sagCW[contador],
        n0,lambda0);
}
propagacionKerrGrafica(numeroPasosq2EM1_1,paso,n0,n2,
    q1EM1tanML[posProp],q1EM1sagML[posProp],q2EM1tanML
    ,q2EM1sagML,w2EM1tanML,w2EM1sagML,chi,kth,Cp,rho,
    dn_dv,P_laser,lambda0,
                            ajuste,"1");
//contador=999; // numeroPasosq2EM1 1
contador   ;
q2EM1tanCW[contador+1]=prop_q(brewsterSalidaTan,
    q2EM1tanCW[contador],n0,1);
w2EM1tanCW[contador+1]=spot_q(q2EM1tanCW[contador
    +1],1,lambda0);
q2EM1sagCW[contador+1]=prop_q(brewsterSalidaSag,
    q2EM1sagCW[contador],n0,1);
w2EM1sagCW[contador+1]=spot_q(q2EM1sagCW[contador
    +1],1,lambda0);
q2EM1tanML[contador+1]=prop_q(brewsterSalidaTan,
    q2EM1tanML[contador],n0,1);
w2EM1tanML[contador+1]=spot_q(q2EM1tanML[contador
    +1],1,lambda0);
q2EM1sagML[contador+1]=prop_q(brewsterSalidaSag,
    q2EM1sagML[contador],n0,1);
w2EM1sagML[contador+1]=spot_q(q2EM1sagML[contador
    +1],1,lambda0);

// Creación de q3
int numeroPasosq3EM1 = delta2/paso+1; // Incluye
    reflexión en espejo curvo.
q3EM1tanCW=(long double complex*)calloc(
    numeroPasosq3EM1, sizeof(long double complex));
w3EM1tanCW=(long double*)calloc(numeroPasosq3EM1,
    sizeof(long double));
q3EM1sagCW=(long double complex*)calloc(
```

```
    numeroPasosq3EM1, sizeof(long double complex));
w3EM1sagCW=(long double*)calloc(numeroPasosq3EM1,
    sizeof(long double));q3EM1tanML=(long double
    complex*)calloc(numeroPasosq3EM1, sizeof(long
    double complex));
w3EM1tanML=(long double*)calloc(numeroPasosq3EM1,
    sizeof(long double));
q3EM1sagML=(long double complex*)calloc(
    numeroPasosq3EM1, sizeof(long double complex));
w3EM1sagML=(long double*)calloc(numeroPasosq3EM1,
    sizeof(long double));

// Propagación de q3
q3EM1tanCW[0]=prop_q(propLibre,q2EM1tanCW[contador
    +1],1,1);
w3EM1tanCW[0]=spot_q(q3EM1tanCW[0],1,lambda0);
q3EM1sagCW[0]=prop_q(propLibre,q2EM1sagCW[contador
    +1],1,1);
w3EM1sagCW[0]=spot_q(q3EM1sagCW[0],1,lambda0);
q3EM1tanML[0]=prop_q(propLibre,q2EM1tanML[contador
    +1],1,1);
w3EM1tanML[0]=spot_q(q3EM1tanML[0],1,lambda0);
q3EM1sagML[0]=prop_q(propLibre,q2EM1sagML[contador
    +1],1,1);
w3EM1sagML[0]=spot_q(q3EM1sagML[0],1,lambda0);

for(contador=1;contador<numeroPasosq3EM1 1;contador
    ++)
{
    q3EM1tanCW[contador]=prop_q(propLibre,q3EM1tanCW[
        contador 1],1,1);
    w3EM1tanCW[contador]=spot_q(q3EM1tanCW[contador
        ],1,lambda0);
    q3EM1sagCW[contador]=prop_q(propLibre,q3EM1sagCW[
        contador 1],1,1);
    w3EM1sagCW[contador]=spot_q(q3EM1sagCW[contador
        ],1,lambda0);
    q3EM1tanML[contador]=prop_q(propLibre,q3EM1tanML[
        contador 1],1,1);
    w3EM1tanML[contador]=spot_q(q3EM1tanML[contador
        ],1,lambda0);
    q3EM1sagML[contador]=prop_q(propLibre,q3EM1sagML[
        contador 1],1,1);
    w3EM1sagML[contador]=spot_q(q3EM1sagML[contador
        ],1,lambda0);
}
```

```
contador   ;
q3EM1tanCW[contador+1]=prop_q(espCurv2Tan,q3EM1tanCW[
    contador],1,1);
w3EM1tanCW[contador+1]=spot_q(q3EM1tanCW[contador
    +1],1,lambda0);
q3EM1sagCW[contador+1]=prop_q(espCurv2Sag,q3EM1sagCW[
    contador],1,1);
w3EM1sagCW[contador+1]=spot_q(q3EM1sagCW[contador
    +1],1,lambda0);
q3EM1tanML[contador+1]=prop_q(espCurv2Tan,q3EM1tanML[
    contador],1,1);
w3EM1tanML[contador+1]=spot_q(q3EM1tanML[contador
    +1],1,lambda0);
q3EM1sagML[contador+1]=prop_q(espCurv2Sag,q3EM1sagML[
    contador],1,1);
w3EM1sagML[contador+1]=spot_q(q3EM1sagML[contador
    +1],1,lambda0);

// Creación de q4
int numeroPasosq4EM1= L2/paso;
q4EM1tanCW=(long double complex*)calloc(
    numeroPasosq4EM1, sizeof(long double complex));
w4EM1tanCW=(long double *)calloc(numeroPasosq4EM1,
    sizeof(long double));
q4EM1sagCW=(long double complex*)calloc(
    numeroPasosq4EM1, sizeof(long double complex));
w4EM1sagCW=(long double *)calloc(numeroPasosq4EM1,
    sizeof(long double));
q4EM1tanML=(long double complex*)calloc(
    numeroPasosq4EM1, sizeof(long double complex));
w4EM1tanML=(long double *)calloc(numeroPasosq4EM1,
    sizeof(long double));
q4EM1sagML=(long double complex*)calloc(
    numeroPasosq4EM1, sizeof(long double complex));
w4EM1sagML=(long double *)calloc(numeroPasosq4EM1,
    sizeof(long double));
// propagación de q4
q4EM1tanCW[0]=prop_q(propLibre,q3EM1tanCW[contador
    +1],1,1);
w4EM1tanCW[0]=spot_q(q4EM1tanCW[0],1,lambda0);
q4EM1sagCW[0]=prop_q(propLibre,q3EM1sagCW[contador
    +1],1,1);
w4EM1sagCW[0]=spot_q(q4EM1sagCW[0],1,lambda0);
q4EM1tanML[0]=prop_q(propLibre,q3EM1tanML[contador
    +1],1,1);
w4EM1tanML[0]=spot_q(q4EM1tanML[0],1,lambda0);
```

```
q4EM1sagML[0]=prop_q(propLibre,q3EM1sagML[contador
    +1],1,1);
w4EM1sagML[0]=spot_q(q4EM1sagML[0],1,lambda0);
for(contador=1;contador<numeroPasosq4EM1 1;contador
    ++)
{
    q4EM1tanCW[contador]=prop_q(propLibre,q4EM1tanCW[
        contador 1],1,1);
    w4EM1tanCW[contador]=spot_q(q4EM1tanCW[contador
        ],1,lambda0);
    q4EM1sagCW[contador]=prop_q(propLibre,q4EM1sagCW[
        contador 1],1,1);
    w4EM1sagCW[contador]=spot_q(q4EM1sagCW[contador
        ],1,lambda0);
    q4EM1tanML[contador]=prop_q(propLibre,q4EM1tanML[
        contador 1],1,1);
    w4EM1tanML[contador]=spot_q(q4EM1tanML[contador
        ],1,lambda0);
    q4EM1sagML[contador]=prop_q(propLibre,q4EM1sagML[
        contador 1],1,1);
    w4EM1sagML[contador]=spot_q(q4EM1sagML[contador
        ],1,lambda0);
}
contador  ;
q4EM1tanCW[contador+1]=prop_q(propLibre,q4EM1tanCW[
    contador],1,1);
w4EM1tanCW[contador+1]=spot_q(q4EM1tanCW[contador
    +1],1,lambda0);
q4EM1sagCW[contador+1]=prop_q(propLibre,q4EM1sagCW[
    contador],1,1);
w4EM1sagCW[contador+1]=spot_q(q4EM1sagCW[contador
    +1],1,lambda0);
q4EM1tanML[contador+1]=prop_q(propLibre,q4EM1tanML[
    contador],1,1);
w4EM1tanML[contador+1]=spot_q(q4EM1tanML[contador
    +1],1,lambda0);
q4EM1sagML[contador+1]=prop_q(propLibre,q4EM1sagML[
    contador],1,1);
w4EM1sagML[contador+1]=spot_q(q4EM1sagML[contador
    +1],1,lambda0);


/// Escritura EM1

/// Estructura para guardar
vectorDato *EM1tan=(vectorDato *)calloc(1,sizeof(
```

```
    vectorDato));
EM1tan->numeroElementos=numeroPasosq0EM1+
    numeroPasosq1EM1+numeroPasosq2EM1+numeroPasosq3EM1
    +numeroPasosq4EM1 4; // Ultima propagación no hay
    refracción.
EM1tan->pos=(long double *)calloc(EM1tan->
    numeroElementos,sizeof(long double));
EM1tan->spotCW=(long double *)calloc(EM1tan->
    numeroElementos,sizeof(long double));
EM1tan->spotML=(long double *)calloc(EM1tan->
    numeroElementos,sizeof(long double));
vectorDato *EM1sag=(vectorDato *)calloc(1,sizeof(
    vectorDato));
EM1sag->numeroElementos=numeroPasosq0EM1+
    numeroPasosq1EM1+numeroPasosq2EM1+numeroPasosq3EM1
    +numeroPasosq4EM1 4; // Ultima propagación no hay
    refracción.
EM1sag->pos=(long double *)calloc(EM1sag->
    numeroElementos,sizeof(long double));
EM1sag->spotCW=(long double *)calloc(EM1sag->
    numeroElementos,sizeof(long double));
EM1sag->spotML=(long double *)calloc(EM1sag->
    numeroElementos,sizeof(long double));


for(int i=0;i<numeroPasosq0EM1 1;i++)
{
    EM1tan->pos[i]=i*paso;
    EM1tan->spotCW[i]=w0EM1tanCW[i];
    EM1tan->spotML[i]=w0EM1tanML[i];
    EM1sag->pos[i]=i*paso;
    EM1sag->spotCW[i]=w0EM1sagCW[i];
    EM1sag->spotML[i]=w0EM1sagML[i];
}
for(int i=0;i<numeroPasosq1EM1 1;i++)
{
    EM1tan->pos[i+numeroPasosq0EM1 1]=(i+
        numeroPasosq0EM1 1)*paso;
    EM1tan->spotCW[i+numeroPasosq0EM1 1]=w1EM1tanCW[i
        ];
    EM1tan->spotML[i+numeroPasosq0EM1 1]=w1EM1tanML[i
        ];
    EM1sag->pos[i+numeroPasosq0EM1 1]=(i+
        numeroPasosq0EM1 1)*paso;
    EM1sag->spotCW[i+numeroPasosq0EM1 1]=w1EM1sagCW[i
        ];
```

```
        EM1sag>spotML[i+numeroPasosq0EM1_1]=w1EM1sagML[i
            ];
    }
    for(int  i=0;i<numeroPasosq2EM1_1;i++)
    {
        if(i==0)
        {
            EM1tan>posKerrIn=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso;
            EM1sag>posKerrIn=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso; // Inicio de
                 cristal
        }
        if(i==(int)(numeroPasosq2EM1_1)/2)
        {
            EM1tan>posKerrMid=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso; // Mitad de
                 cristal
            EM1sag>posKerrMid=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso; // Mitad de
                 cristal
        }
        if(i==(numeroPasosq2EM1_2))
        {
            EM1tan>posKerrOut=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso; // Fin de
                 cristal
            EM1sag>posKerrOut=(i+numeroPasosq0EM1+
                numeroPasosq1EM1_2)*paso; // Fin de
                 cristal
        }
        EM1tan>pos[i+numeroPasosq0EM1+numeroPasosq1EM1
            2]=(i+numeroPasosq0EM1+numeroPasosq1EM1_2)*
            paso;
        EM1tan>spotCW[i+numeroPasosq0EM1+
            numeroPasosq1EM1_2]=w2EM1tanCW[i];
        EM1tan>spotML[i+numeroPasosq0EM1+
            numeroPasosq1EM1_2]=w2EM1tanML[i];
        EM1sag>pos[i+numeroPasosq0EM1+numeroPasosq1EM1
            2]=(i+numeroPasosq0EM1+numeroPasosq1EM1_2)*
            paso;
        EM1sag>spotCW[i+numeroPasosq0EM1+
            numeroPasosq1EM1_2]=w2EM1sagCW[i];
        EM1sag>spotML[i+numeroPasosq0EM1+
            numeroPasosq1EM1_2]=w2EM1sagML[i];
    }
```

```
for(int i=0;i<numeroPasosq3EM1 1; i++)
{
    EM1tan>pos[i+numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1 3]=(i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3)*paso;
    EM1tan>spotCW[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3]=
        w3EM1tanCW[i];
    EM1tan>spotML[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3]=
        w3EM1tanML[i];
    EM1sag>pos[i+numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1 3]=(i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3)*paso;
    EM1sag>spotCW[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3]=
        w3EM1sagCW[i];
    EM1sag>spotML[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1 3]=
        w3EM1sagML[i];
}
for(int i=0;i<numeroPasosq4EM1 1; i++)
{
    EM1tan>pos[i+numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1+numeroPasosq3EM1 4]=(i+
        numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1+numeroPasosq3EM1 4)*paso;
    EM1tan>spotCW[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1+
        numeroPasosq3EM1 4]=w4EM1tanCW[i];
    EM1tan>spotML[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1+
        numeroPasosq3EM1 4]=w4EM1tanML[i];
    EM1sag>pos[i+numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1+numeroPasosq3EM1 4]=(i+
        numeroPasosq0EM1+numeroPasosq1EM1+
        numeroPasosq2EM1+numeroPasosq3EM1 4)*paso;
    EM1sag>spotCW[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1+
        numeroPasosq3EM1 4]=w4EM1sagCW[i];
    EM1sag>spotML[i+numeroPasosq0EM1+
        numeroPasosq1EM1+numeroPasosq2EM1+
        numeroPasosq3EM1 4]=w4EM1sagML[i];
}

// Limpieza de vectores
```

```
free(q0EM1tanCW);
free(q1EM1tanCW);
free(q2EM1tanCW);
free(q3EM1tanCW);
free(q4EM1tanCW);
free(w0EM1tanCW);
free(w1EM1tanCW);
free(w2EM1tanCW);
free(w3EM1tanCW);
free(w4EM1tanCW);
free(q0EM1tanML);
free(q1EM1tanML);
free(q2EM1tanML);
free(q3EM1tanML);
free(q4EM1tanML);
free(w0EM1tanML);
free(w1EM1tanML);
free(w2EM1tanML);
free(w3EM1tanML);
free(w4EM1tanML);
free(q0EM1sagCW);
free(q1EM1sagCW);
free(q2EM1sagCW);
free(q3EM1sagCW);
free(q4EM1sagCW);
free(w0EM1sagCW);
free(w1EM1sagCW);
free(w2EM1sagCW);
free(w3EM1sagCW);
free(w4EM1sagCW);
free(q0EM1sagML);
free(q1EM1sagML);
free(q2EM1sagML);
free(q3EM1sagML);
free(q4EM1sagML);
free(w0EM1sagML);
free(w1EM1sagML);
free(w2EM1sagML);
free(w3EM1sagML);
free(w4EM1sagML);

/// FIN EM1
/// INICIO EM2

long double complex *q0EM2tanCW, *q1EM2tanCW, *
    q2EM2tanCW, *q3EM2tanCW, *q4EM2tanCW; // Vectores
```

```
    para almacenar q propagada.
long double *w0EM2tanCW, *w1EM2tanCW, *w2EM2tanCW, *
    w3EM2tanCW, *w4EM2tanCW; // Vectores para spot
long double complex *q0EM2sagCW, *q1EM2sagCW, *
    q2EM2sagCW, *q3EM2sagCW, *q4EM2sagCW; // Vectores
    para almacenar q propagada.
long double *w0EM2sagCW, *w1EM2sagCW, *w2EM2sagCW, *
    w3EM2sagCW, *w4EM2sagCW; // Vectores para spot
long double complex *q0EM2tanML, *q1EM2tanML, *
    q2EM2tanML, *q3EM2tanML, *q4EM2tanML; // Vectores
    para almacenar q propagada.
long double *w0EM2tanML, *w1EM2tanML, *w2EM2tanML, *
    w3EM2tanML, *w4EM2tanML; // Vectores para spot
long double complex *q0EM2sagML, *q1EM2sagML, *
    q2EM2sagML, *q3EM2sagML, *q4EM2sagML; // Vectores
    para almacenar q propagada.
long double *w0EM2sagML, *w1EM2sagML, *w2EM2sagML, *
    w3EM2sagML, *w4EM2sagML; // Vectores para spot

// Estimación de tamaño y reserva de memoriade vector
     q0.

int numeroPasosq0EM2 = L2/paso+1;  // Incluye
    reflexión en espejo curvo

q0EM2tanCW=(long double complex*)calloc(
    numeroPasosq0EM2,sizeof(long double complex));
w0EM2tanCW=(long double*)calloc(numeroPasosq0EM2,
    sizeof(long double));
q0EM2sagCW=(long double complex*)calloc(
    numeroPasosq0EM2,sizeof(long double complex));
w0EM2sagCW=(long double*)calloc(numeroPasosq0EM2,
    sizeof(long double));
q0EM2tanML=(long double complex*)calloc(
    numeroPasosq0EM2,sizeof(long double complex));
w0EM2tanML=(long double*)calloc(numeroPasosq0EM2,
    sizeof(long double));
q0EM2sagML=(long double complex*)calloc(
    numeroPasosq0EM2,sizeof(long double complex));
w0EM2sagML=(long double*)calloc(numeroPasosq0EM2,
    sizeof(long double));


// Propagación en q0EM2

q0EM2tanCW[0]=obtieneElemento(qt2,1,1);
```

```
w0EM2tanCW[0]=spot_q(q0EM2tanCW[0],1,lambda0);
q0EM2sagCW[0]=obtieneElemento(qs2,1,1);
w0EM2sagCW[0]=spot_q(q0EM2sagCW[0],1,lambda0);
q0EM2tanML[0]=obtieneElemento(qAstigEM2_ML[0],1,1);
w0EM2tanML[0]=spot_q(q0EM2tanML[0],1,lambda0);
q0EM2sagML[0]=obtieneElemento(qAstigEM2_ML[1],1,1);
w0EM2sagML[0]=spot_q(q0EM2sagML[0],1,lambda0);

for(contador=1;contador<numeroPasosq0EM2 1;contador
    ++)
{
     q0EM2tanCW[contador]=prop_q(propLibre,q0EM2tanCW[
         contador 1],1,1);
     w0EM2tanCW[contador]=spot_q(q0EM2tanCW[contador
         ],1,lambda0);
     q0EM2sagCW[contador]=prop_q(propLibre,q0EM2sagCW[
         contador 1],1,1);
     w0EM2sagCW[contador]=spot_q(q0EM2sagCW[contador
         ],1,lambda0);
     q0EM2tanML[contador]=prop_q(propLibre,q0EM2tanML[
         contador 1],1,1);
     w0EM2tanML[contador]=spot_q(q0EM2tanML[contador
         ],1,lambda0);
     q0EM2sagML[contador]=prop_q(propLibre,q0EM2sagML[
         contador 1],1,1);
     w0EM2sagML[contador]=spot_q(q0EM2sagML[contador
         ],1,lambda0);
}
contador   ;
q0EM2tanCW[contador+1]=prop_q(espCurv1Tan,q0EM2tanCW[
    contador],1,1);
w0EM2tanCW[contador+1]=spot_q(q0EM2tanCW[contador
    +1],1,lambda0);
q0EM2sagCW[contador+1]=prop_q(espCurv1Sag,q0EM2sagCW[
    contador],1,1);
w0EM2sagCW[contador+1]=spot_q(q0EM2sagCW[contador
    +1],1,lambda0);
q0EM2tanML[contador+1]=prop_q(espCurv1Tan,q0EM2tanML[
    contador],1,1);
w0EM2tanML[contador+1]=spot_q(q0EM2tanML[contador
    +1],1,lambda0);
q0EM2sagML[contador+1]=prop_q(espCurv1Sag,q0EM2sagML[
    contador],1,1);
w0EM2sagML[contador+1]=spot_q(q0EM2sagML[contador
    +1],1,lambda0);
```

```
// Creación de q1
int numeroPasosq1EM2 = delta2/paso+1; // incluye
    refracción en cristal
q1EM2tanCW=(long double complex*)calloc(
    numeroPasosq1EM2,sizeof(long double complex));
w1EM2tanCW=(long double*)calloc(numeroPasosq1EM2,
    sizeof(long double));
q1EM2sagCW=(long double complex*)calloc(
    numeroPasosq1EM2,sizeof(long double complex));
w1EM2sagCW=(long double*)calloc(numeroPasosq1EM2,
    sizeof(long double));
q1EM2tanML=(long double complex*)calloc(
    numeroPasosq1EM2,sizeof(long double complex));
w1EM2tanML=(long double*)calloc(numeroPasosq1EM2,
    sizeof(long double));
q1EM2sagML=(long double complex*)calloc(
    numeroPasosq1EM2,sizeof(long double complex));
w1EM2sagML=(long double*)calloc(numeroPasosq1EM2,
    sizeof(long double));
// Propagación en q1EM2tanML
q1EM2tanCW[0]=prop_q(propLibre,q0EM2tanCW[contador
    +1],1,1);
w1EM2tanCW[0]=spot_q(q1EM2tanCW[0],1,lambda0);
q1EM2sagCW[0]=prop_q(propLibre,q0EM2sagCW[contador
    +1],1,1);
w1EM2sagCW[0]=spot_q(q1EM2sagCW[0],1,lambda0);
q1EM2tanML[0]=prop_q(propLibre,q0EM2tanML[contador
    +1],1,1);
w1EM2tanML[0]=spot_q(q1EM2tanML[0],1,lambda0);
q1EM2sagML[0]=prop_q(propLibre,q0EM2sagML[contador
    +1],1,1);
w1EM2sagML[0]=spot_q(q1EM2sagML[0],1,lambda0);


for(contador=1;contador<numeroPasosq1EM2 1;contador
    ++)
{
    q1EM2tanCW[contador]=prop_q(propLibre,q1EM2tanCW[
        contador 1],1,1);
    w1EM2tanCW[contador]=spot_q(q1EM2tanCW[contador
        ],1,lambda0);
    q1EM2sagCW[contador]=prop_q(propLibre,q1EM2sagCW[
        contador 1],1,1);
    w1EM2sagCW[contador]=spot_q(q1EM2sagCW[contador
        ],1,lambda0);
    q1EM2tanML[contador]=prop_q(propLibre,q1EM2tanML[
```

```
            contador 1],1,1);
        w1EM2tanML[contador]=spot_q(q1EM2tanML[contador
            ],1,lambda0);
        q1EM2sagML[contador]=prop_q(propLibre,q1EM2sagML[
            contador 1],1,1);
        w1EM2sagML[contador]=spot_q(q1EM2sagML[contador
            ],1,lambda0);
    }
    contador   ;
    q1EM2tanCW[contador+1]=prop_q(brewsterEntradaTan,
        q1EM2tanCW[contador],1,n0);
    w1EM2tanCW[contador+1]=spot_q(q1EM2tanCW[contador+1],
        n0,lambda0);
    q1EM2sagCW[contador+1]=prop_q(brewsterEntradaSag,
        q1EM2sagCW[contador],1,n0);
    w1EM2sagCW[contador+1]=spot_q(q1EM2sagCW[contador+1],
        n0,lambda0);
    q1EM2tanML[contador+1]=prop_q(brewsterEntradaTan,
        q1EM2tanML[contador],1,n0);
    w1EM2tanML[contador+1]=spot_q(q1EM2tanML[contador+1],
        n0,lambda0);
    q1EM2sagML[contador+1]=prop_q(brewsterEntradaSag,
        q1EM2sagML[contador],1,n0);
    w1EM2sagML[contador+1]=spot_q(q1EM2sagML[contador+1],
        n0,lambda0);
    posProp=contador+1;

    // Creación de q2
    int numeroPasosq2EM2 = pasos+1; // incluye refracción
         hacia afuera del cristal
    q2EM2tanCW=(long double complex*)calloc(
        numeroPasosq2EM2,sizeof(long double complex));
    w2EM2tanCW=(long double *)calloc(numeroPasosq2EM2,
        sizeof(long double));
    q2EM2sagCW=(long double complex*)calloc(
        numeroPasosq2EM2,sizeof(long double complex));
    w2EM2sagCW=(long double *)calloc(numeroPasosq2EM2,
        sizeof(long double));
    q2EM2tanML=(long double complex*)calloc(
        numeroPasosq2EM2,sizeof(long double complex));
    w2EM2tanML=(long double *)calloc(numeroPasosq2EM2,
        sizeof(long double));
    q2EM2sagML=(long double complex*)calloc(
        numeroPasosq2EM2,sizeof(long double complex));
    w2EM2sagML=(long double *)calloc(numeroPasosq2EM2,
        sizeof(long double));
```

```
/// Propagación en cristal
q2EM2tanCW[0]=prop_q(pasoCristal,q1EM2tanCW[contador
    +1],n0,n0);
w2EM2tanCW[0]=spot_q(q2EM2tanCW[0],n0,lambda0);
q2EM2sagCW[0]=prop_q(pasoCristal,q1EM2sagCW[contador
    +1],n0,n0);
w2EM2sagCW[0]=spot_q(q2EM2sagCW[0],n0,lambda0);
//q2EM2tanML[0]=prop_q(pasoCristal,q1EM2tanML[
    contador+1],n0,n0);
//w2EM2tanML[0]=spot_q(q2EM2tanML[0],n0,lambda0);
//q2EM2sagML[0]=prop_q(pasoCristal,q1EM2sagML[
    contador+1],n0,n0);
//w2EM2sagML[0]=spot_q(q2EM2sagML[0],n0,lambda0);
for(contador=1;contador<numeroPasosq2EM2_1;contador
    ++)
{
     q2EM2tanCW[contador]=prop_q(pasoCristal,
         q2EM2tanCW[contador_1],n0,n0);
     w2EM2tanCW[contador]=spot_q(q2EM2tanCW[contador],
         n0,lambda0);
     q2EM2sagCW[contador]=prop_q(pasoCristal,
         q2EM2sagCW[contador_1],n0,n0);
     w2EM2sagCW[contador]=spot_q(q2EM2sagCW[contador],
         n0,lambda0);
}
propagacionKerrGrafica(numeroPasosq2EM2_1,paso,n0,n2,
    q1EM2tanML[posProp],q1EM2sagML[posProp],q2EM2tanML
    ,q2EM2sagML,w2EM2tanML,w2EM2sagML,chi,kth,Cp,rho,
    dn_dv,P_laser,lambda0,
                          ajuste,"0");
//contador=999; // numeroPasosq2EM2_1
contador  ;
q2EM2tanCW[contador+1]=prop_q(brewsterSalidaTan,
    q2EM2tanCW[contador],n0,1);
w2EM2tanCW[contador+1]=spot_q(q2EM2tanCW[contador
    +1],1,lambda0);
q2EM2sagCW[contador+1]=prop_q(brewsterSalidaSag,
    q2EM2sagCW[contador],n0,1);
w2EM2sagCW[contador+1]=spot_q(q2EM2sagCW[contador
    +1],1,lambda0);
q2EM2tanML[contador+1]=prop_q(brewsterSalidaTan,
    q2EM2tanML[contador],n0,1);
w2EM2tanML[contador+1]=spot_q(q2EM2tanML[contador
    +1],1,lambda0);
```

```
q2EM2sagML[contador+1]=prop_q(brewsterSalidaSag,
    q2EM2sagML[contador],n0,1);
w2EM2sagML[contador+1]=spot_q(q2EM2sagML[contador
    +1],1,lambda0);

// Creación de q3
int numeroPasosq3EM2 = delta1/paso+1; // Incluye
    reflexión en espejo curvo.
q3EM2tanCW=(long double complex*)calloc(
    numeroPasosq3EM2, sizeof(long double complex));
w3EM2tanCW=(long double*)calloc(numeroPasosq3EM2,
    sizeof(long double));
q3EM2sagCW=(long double complex*)calloc(
    numeroPasosq3EM2, sizeof(long double complex));
w3EM2sagCW=(long double*)calloc(numeroPasosq3EM2,
    sizeof(long double));
q3EM2tanML=(long double complex*)calloc(
    numeroPasosq3EM2, sizeof(long double complex));
w3EM2tanML=(long double*)calloc(numeroPasosq3EM2,
    sizeof(long double));
q3EM2sagML=(long double complex*)calloc(
    numeroPasosq3EM2, sizeof(long double complex));
w3EM2sagML=(long double*)calloc(numeroPasosq3EM2,
    sizeof(long double));

// Propagación de q3
q3EM2tanCW[0]=prop_q(propLibre,q2EM2tanCW[contador
    +1],1,1);
w3EM2tanCW[0]=spot_q(q3EM2tanCW[0],1,lambda0);
q3EM2sagCW[0]=prop_q(propLibre,q2EM2sagCW[contador
    +1],1,1);
w3EM2sagCW[0]=spot_q(q3EM2sagCW[0],1,lambda0);
q3EM2tanML[0]=prop_q(propLibre,q2EM2tanML[contador
    +1],1,1);
w3EM2tanML[0]=spot_q(q3EM2tanML[0],1,lambda0);
q3EM2sagML[0]=prop_q(propLibre,q2EM2sagML[contador
    +1],1,1);
w3EM2sagML[0]=spot_q(q3EM2sagML[0],1,lambda0);

for(contador=1;contador<numeroPasosq3EM2 1;contador
    ++)
{
    q3EM2tanCW[contador]=prop_q(propLibre,q3EM2tanCW[
        contador 1],1,1);
    w3EM2tanCW[contador]=spot_q(q3EM2tanCW[contador
        ],1,lambda0);
```

```
        q3EM2sagCW[contador]=prop_q(propLibre,q3EM2sagCW[
            contador-1],1,1);
        w3EM2sagCW[contador]=spot_q(q3EM2sagCW[contador
            ],1,lambda0);
        q3EM2tanML[contador]=prop_q(propLibre,q3EM2tanML[
            contador-1],1,1);
        w3EM2tanML[contador]=spot_q(q3EM2tanML[contador
            ],1,lambda0);
        q3EM2sagML[contador]=prop_q(propLibre,q3EM2sagML[
            contador-1],1,1);
        w3EM2sagML[contador]=spot_q(q3EM2sagML[contador
            ],1,lambda0);
    }
    contador  ;
    q3EM2tanCW[contador+1]=prop_q(espCurv2Tan,q3EM2tanCW[
        contador],1,1);
    w3EM2tanCW[contador+1]=spot_q(q3EM2tanCW[contador
        +1],1,lambda0);
    q3EM2sagCW[contador+1]=prop_q(espCurv2Sag,q3EM2sagCW[
        contador],1,1);
    w3EM2sagCW[contador+1]=spot_q(q3EM2sagCW[contador
        +1],1,lambda0);
    q3EM2tanML[contador+1]=prop_q(espCurv2Tan,q3EM2tanML[
        contador],1,1);
    w3EM2tanML[contador+1]=spot_q(q3EM2tanML[contador
        +1],1,lambda0);
    q3EM2sagML[contador+1]=prop_q(espCurv2Sag,q3EM2sagML[
        contador],1,1);
    w3EM2sagML[contador+1]=spot_q(q3EM2sagML[contador
        +1],1,lambda0);

    // Creación de q4
    int numeroPasosq4EM2= L1/paso;
    q4EM2tanCW=(long double complex*)calloc(
        numeroPasosq4EM2, sizeof(long double complex));
    w4EM2tanCW=(long double *)calloc(numeroPasosq4EM2,
        sizeof(long double));
    q4EM2sagCW=(long double complex*)calloc(
        numeroPasosq4EM2, sizeof(long double complex));
    w4EM2sagCW=(long double *)calloc(numeroPasosq4EM2,
        sizeof(long double));
    q4EM2tanML=(long double complex*)calloc(
        numeroPasosq4EM2, sizeof(long double complex));
    w4EM2tanML=(long double *)calloc(numeroPasosq4EM2,
        sizeof(long double));
    q4EM2sagML=(long double complex*)calloc(
```

```
    numeroPasosq4EM2, sizeof(long double complex));
w4EM2sagML=(long double *)calloc(numeroPasosq4EM2,
    sizeof(long double));
// propagación de q4
q4EM2tanCW[0]=prop_q(propLibre,q3EM2tanCW[contador
    +1],1,1);
w4EM2tanCW[0]=spot_q(q4EM2tanCW[0],1,lambda0);
q4EM2sagCW[0]=prop_q(propLibre,q3EM2sagCW[contador
    +1],1,1);
w4EM2sagCW[0]=spot_q(q4EM2sagCW[0],1,lambda0);
q4EM2tanML[0]=prop_q(propLibre,q3EM2tanML[contador
    +1],1,1);
w4EM2tanML[0]=spot_q(q4EM2tanML[0],1,lambda0);
q4EM2sagML[0]=prop_q(propLibre,q3EM2sagML[contador
    +1],1,1);
w4EM2sagML[0]=spot_q(q4EM2sagML[0],1,lambda0);
for(contador=1;contador<numeroPasosq4EM2 1;contador
    ++)
{
    q4EM2tanCW[contador]=prop_q(propLibre,q4EM2tanCW[
        contador 1],1,1);
    w4EM2tanCW[contador]=spot_q(q4EM2tanCW[contador
        ],1,lambda0);
    q4EM2sagCW[contador]=prop_q(propLibre,q4EM2sagCW[
        contador 1],1,1);
    w4EM2sagCW[contador]=spot_q(q4EM2sagCW[contador
        ],1,lambda0);
    q4EM2tanML[contador]=prop_q(propLibre,q4EM2tanML[
        contador 1],1,1);
    w4EM2tanML[contador]=spot_q(q4EM2tanML[contador
        ],1,lambda0);
    q4EM2sagML[contador]=prop_q(propLibre,q4EM2sagML[
        contador 1],1,1);
    w4EM2sagML[contador]=spot_q(q4EM2sagML[contador
        ],1,lambda0);
}
contador  ;
q4EM2tanCW[contador+1]=prop_q(propLibre,q4EM2tanCW[
    contador],1,1);
w4EM2tanCW[contador+1]=spot_q(q4EM2tanCW[contador
    +1],1,lambda0);
q4EM2sagCW[contador+1]=prop_q(propLibre,q4EM2sagCW[
    contador],1,1);
w4EM2sagCW[contador+1]=spot_q(q4EM2sagCW[contador
    +1],1,lambda0);
q4EM2tanML[contador+1]=prop_q(propLibre,q4EM2tanML[
```

```
    contador],1,1);
w4EM2tanML[contador+1]=spot_q(q4EM2tanML[contador
    +1],1,lambda0);
q4EM2sagML[contador+1]=prop_q(propLibre,q4EM2sagML[
    contador],1,1);
w4EM2sagML[contador+1]=spot_q(q4EM2sagML[contador
    +1],1,lambda0);


/// Escritura EM2tanML

/// Estructura para guardar
vectorDato *EM2tan=(vectorDato *)calloc(1,sizeof(
    vectorDato));
EM2tan>numeroElementos=numeroPasosq0EM2+
    numeroPasosq1EM2+numeroPasosq2EM2+numeroPasosq3EM2
    +numeroPasosq4EM2 4; // Ultima propagación no hay
    refracción.
EM2tan>pos=(long double *)calloc(EM2tan>
    numeroElementos,sizeof(long double));
EM2tan>spotCW=(long double *)calloc(EM2tan>
    numeroElementos,sizeof(long double));
EM2tan>spotML=(long double *)calloc(EM2tan>
    numeroElementos,sizeof(long double));
vectorDato *EM2sag=(vectorDato *)calloc(1,sizeof(
    vectorDato));
EM2sag>numeroElementos=numeroPasosq0EM2+
    numeroPasosq1EM2+numeroPasosq2EM2+numeroPasosq3EM2
    +numeroPasosq4EM2 4; // Ultima propagación no hay
    refracción.
EM2sag>pos=(long double *)calloc(EM2sag>
    numeroElementos,sizeof(long double));
EM2sag>spotCW=(long double *)calloc(EM2sag>
    numeroElementos,sizeof(long double));
EM2sag>spotML=(long double *)calloc(EM2sag>
    numeroElementos,sizeof(long double));



for(int i=0;i<numeroPasosq0EM2 1;i++)
{
    EM2tan>pos[i]=(numeroPasosq0EM2+numeroPasosq1EM2
        +numeroPasosq2EM2+numeroPasosq3EM2+
        numeroPasosq4EM2 5 i)*paso;
    EM2tan>spotCW[i]=w0EM2tanCW[i];
    EM2tan>spotML[i]=w0EM2tanML[i];
```

```
        EM2sag->pos[i]=(numeroPasosq0EM2+numeroPasosq1EM2
            +numeroPasosq2EM2+numeroPasosq3EM2+
            numeroPasosq4EM2_5_i)*paso;
        EM2sag->spotCW[i]=w0EM2sagCW[i];
        EM2sag->spotML[i]=w0EM2sagML[i];
    }
    for(int i=0;i<numeroPasosq1EM2_1;i++)
    {
        EM2tan->pos[i+numeroPasosq0EM2_1]=(
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2+numeroPasosq4EM2_4_i)*paso;
        EM2tan->spotCW[i+numeroPasosq0EM2_1]=w1EM2tanCW[i
            ];
        EM2tan->spotML[i+numeroPasosq0EM2_1]=w1EM2tanML[i
            ];
        EM2sag->pos[i+numeroPasosq0EM2_1]=(
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2+numeroPasosq4EM2_4_i)*paso;
        EM2sag->spotCW[i+numeroPasosq0EM2_1]=w1EM2sagCW[i
            ];
        EM2sag->spotML[i+numeroPasosq0EM2_1]=w1EM2sagML[i
            ];
    }
    for(int i=0;i<numeroPasosq2EM2_1;i++)
    {
        if(i==0)
        {
            EM2tan->posKerrIn=(numeroPasosq2EM2+
                numeroPasosq3EM2+numeroPasosq4EM2_3_i)*
                paso; //Inicio de cristal
            EM2sag->posKerrIn=(numeroPasosq2EM2+
                numeroPasosq3EM2+numeroPasosq4EM2_3_i)*
                paso; //Inicio de cristal
        }
        if(i==(numeroPasosq2EM2_1)/2)
        {
            EM2tan->posKerrMid=(numeroPasosq2EM2+
                numeroPasosq3EM2+numeroPasosq4EM2_3_i)*
                paso; //Mitad de cristal
            EM2sag->posKerrMid=(numeroPasosq2EM2+
                numeroPasosq3EM2+numeroPasosq4EM2_3_i)*
                paso; //Mitad de cristal
        }
        if(i==(numeroPasosq2EM2_2))
        {
            EM2tan->posKerrOut=(numeroPasosq2EM2+
```

```
                numeroPasosq3EM2+numeroPasosq4EM2 3 i)*
                paso; // Fin de cristal
            EM2sag>posKerrOut=(numeroPasosq2EM2+
                numeroPasosq3EM2+numeroPasosq4EM2 3 i)*
                paso; // Fin de cristal
        }
        EM2tan>pos[i+numeroPasosq0EM2+numeroPasosq1EM2
            2]=(numeroPasosq2EM2+numeroPasosq3EM2+
            numeroPasosq4EM2 3 i)*paso;
        EM2tan>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2 2]=w2EM2tanCW[i];
        EM2tan>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2 2]=w2EM2tanML[i];
        EM2sag>pos[i+numeroPasosq0EM2+numeroPasosq1EM2
            2]=(numeroPasosq2EM2+numeroPasosq3EM2+
            numeroPasosq4EM2 3 i)*paso;
        EM2sag>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2 2]=w2EM2sagCW[i];
        EM2sag>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2 2]=w2EM2sagML[i];
    }
    for(int i=0;i<numeroPasosq3EM2 1;i++)
    {
        EM2tan>pos[i+numeroPasosq0EM2+numeroPasosq1EM2+
            numeroPasosq2EM2 3]=(numeroPasosq3EM2+
            numeroPasosq4EM2 2 i)*paso;
        EM2tan>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2 3]=
            w3EM2tanCW[i];
        EM2tan>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2 3]=
            w3EM2tanML[i];
        EM2sag>pos[i+numeroPasosq0EM2+numeroPasosq1EM2+
            numeroPasosq2EM2 3]=(numeroPasosq3EM2+
            numeroPasosq4EM2 2 i)*paso;
        EM2sag>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2 3]=
            w3EM2sagCW[i];
        EM2sag>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2 3]=
            w3EM2sagML[i];
    }
    for(int i=0;i<numeroPasosq4EM2 1;i++)
    {
        EM2tan>pos[i+numeroPasosq0EM2+numeroPasosq1EM2+
            numeroPasosq2EM2+numeroPasosq3EM2 4]=(
```

```
            numeroPasosq4EM2 1 i)*paso;
        EM2tan>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2 4]=w4EM2tanCW[i];
        EM2tan>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2 4]=w4EM2tanML[i];
        EM2sag>pos[i+numeroPasosq0EM2+numeroPasosq1EM2+
            numeroPasosq2EM2+numeroPasosq3EM2 4]=(
            numeroPasosq4EM2 1 i)*paso;
        EM2sag>spotCW[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2 4]=w4EM2sagCW[i];
        EM2sag>spotML[i+numeroPasosq0EM2+
            numeroPasosq1EM2+numeroPasosq2EM2+
            numeroPasosq3EM2 4]=w4EM2sagML[i];
    }

    //Limpieza de vectores
    free(q0EM2tanCW);
    free(q1EM2tanCW);
    free(q2EM2tanCW);
    free(q3EM2tanCW);
    free(q4EM2tanCW);
    free(w0EM2tanCW);
    free(w1EM2tanCW);
    free(w2EM2tanCW);
    free(w3EM2tanCW);
    free(w4EM2tanCW);
    free(q0EM2tanML);
    free(q1EM2tanML);
    free(q2EM2tanML);
    free(q3EM2tanML);
    free(q4EM2tanML);
    free(w0EM2tanML);
    free(w1EM2tanML);
    free(w2EM2tanML);
    free(w3EM2tanML);
    free(w4EM2tanML);
    free(q0EM2sagCW);
    free(q1EM2sagCW);
    free(q2EM2sagCW);
    free(q3EM2sagCW);
    free(q4EM2sagCW);
    free(w0EM2sagCW);
    free(w1EM2sagCW);
```

```
free(w2EM2sagCW);
free(w3EM2sagCW);
free(w4EM2sagCW);
free(q0EM2sagML);
free(q1EM2sagML);
free(q2EM2sagML);
free(q3EM2sagML);
free(q4EM2sagML);
free(w0EM2sagML);
free(w1EM2sagML);
free(w2EM2sagML);
free(w3EM2sagML);
free(w4EM2sagML);


/// FIN EM2

/// Escribe script
gnuplotEscribe(EM1tan,EM1sag,EM2tan,EM2sag);


/// Limpieza final

free(EM1tan>pos);
free(EM1tan>spotCW);
free(EM1tan>spotML);
free(EM1tan);
free(EM1sag>pos);
free(EM1sag>spotCW);
free(EM1sag>spotML);
free(EM1sag);
free(EM2tan>pos);
free(EM2tan>spotCW);
free(EM2tan>spotML);
free(EM2tan);
free(EM2sag>pos);
free(EM2sag>spotCW);
free(EM2sag>spotML);
free(EM2sag);

wt1=borraMatriz(wt1);
ws1=borraMatriz(ws1);
wt2=borraMatriz(wt2);
ws2=borraMatriz(ws2);
qt1=borraMatriz(qt1);
qs1=borraMatriz(qs1);
```

```
    qt2=borraMatriz(qt2);
    qs2=borraMatriz(qs2);


    wt1_ml=borraMatriz(wt1_ml);
    ws1_ml=borraMatriz(ws1_ml);
    wt2_ml=borraMatriz(wt2_ml);
    ws2_ml=borraMatriz(ws2_ml);
    free(qAstigEM1_ML[0]);
    qAstigEM1_ML[0]=NULL;
    free(qAstigEM1_ML[1]);
    qAstigEM1_ML[1]=NULL;
    free(qAstigEM1_ML);
    qAstigEM1_ML=NULL;
    free(qAstigEM2_ML[0]);
    qAstigEM2_ML[0]=NULL;
    free(qAstigEM2_ML[1]);
    qAstigEM2_ML[1]=NULL;
    free(qAstigEM2_ML);
    qAstigEM1_ML=NULL;
    free(ajuste->a0);
    ajuste->a0=NULL;
    free(ajuste->a1);
    ajuste->a1=NULL;
    free(ajuste->a2);
    ajuste->a2=NULL;
    free(ajuste->paso);
    ajuste->paso=NULL;
    free(ajuste);
    ajuste=NULL;
    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}
```

propGrafica.h

```
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include "matrices.h"
#include "lineal.h"
#include "no_lineal.h"
#include "termico.h"


/* Estructura de vector */
```

```
typedef struct
{
    int numeroElementos;
    long double posKerrIn; // Posición donde el haz entra
        al cristal
    long double posKerrOut; // Posición donde el haz sale
        del cristal
    long double posKerrMid; // Sección media del cristal.
    long double * spotCW; // Radio del haz
    long double * spotML; // Radio del haz
    long double * pos; // Posición en cavidad.
}vectorDato;

void escribeData(vectorDato *datos, char* filenameCW,
    char* filenameML);

/* Escribe a gnuplot   Todos los datos. */
void gnuplotEscribe(vectorDato *EM1tan, vectorDato *
    EM1sag, vectorDato *EM2tan, vectorDato *EM2sag);


/* Rutina para gráficar propagación lineal   Conlleva un
    error por la discreti
zación de la propagación de rayos (en lugar de una matriz
     para propagar en espacio,
se utilizan muchas matrices */

void propagacionKerrGrafica(int pasos, long double deltaZ
    , long double n0,long double n2,
                            long double complex qInTan,
                                long double complex
                                qInSag, long double
                                complex *qTan, \
                                        long double
                                            complex *
                                            qSag, long
                                             double *
                                            wTan, long
                                             double *
                                            wSag, long
                                             double
                                            chi, long
                                            double kth
                                            , long
                                            double Cp,
```

```
                                                    \
                                                long double
                                                    rho, long
                                                    double
                                                    dn_dv,long
                                                     double
                                                    P_laser,
                                                    long
                                                    double
                                                    lambda0, \
                                                ajusteTemperaturaCristal
                                                    *
                                                    vectorPlano
                                                    , _Bool
                                                    ladoBombeo
                                                    );
void graficaPropagacion(char *conjugado_corto, char *
    conjugado_largo, int iteraciones, long double umbral,
    int N, long double Epsilon1, long double Epsilon2,
                        long double lambdaPump, long double
                            n0, long double n2, long double
                            P_laser, long double P_pump, long
                             double chi, long double L,
                        long double kth, long double Cp,
                            long double rho, long double
                            dn_dv, long double ancho, long
                            double alto, long double w_pump_t
                            ,
                        long double w_pump_s, long double
                            nPump, long double lambda0, long
                            double L1, long double L2,
                        long double f1, long double f2);
```

constantes.h

```
#ifndef CONSTANTES_H_INCLUDED
#define CONSTANTES_H_INCLUDED
 /* Constantes para programas */
#include <complex.h>

     long double L1=1.0;
     long double L2=1.35;
     long double L=10.0e 3;
     long double n0=1.7598;
     long double n2=3.0e 20;
     long double nPump=1.7715;
```

```
    long double w_pump=75.0e 6;
    long double kth=1.3e 5;
    long double chi=0.6;
    long double Cp=775.0;
    long double rho=3990.0;
    long double P_pump=8.0;
    long double P_laser=100.0e3;
    long double dn_dv=1e 6;
    long double lambda0=810.0e 9;
    long double lambdaPump=535.0e 9;
    long double f1=50.0e 3;
    long double f2=50.0e 3;

#endif // CONSTANTES_H_INCLUDED
```

main.c del propagador láser

```
#include <stdio.h>
#include <stdlib.h>
#include "error_iteraciones.h"
#include <complex.h>
#include <time.h>
#include "lineal.h"
#include "no_lineal.h"
#include "matrices.h"
#include "constantes.h"
#include "no_linealRK.h"
#include "termico.h"
#include "no_linealMatAstTerm.h"

void guardaSpots(char *nombre, long double *spots, long
    double *potencia, int pasos)
{
    // Guarda spots primero y luego potencias.
    FILE *archivo;
    archivo = fopen(nombre,"w");
    for(int i=0;i<pasos;i++)
        fprintf(archivo,"%Le,",spots[i]);
    fprintf(archivo,"\n");
    for(int i=0;i<pasos;i++)
        fprintf(archivo,"%Le,",potencia[i]);
    fprintf(archivo,"\n");
    fclose(archivo);
}

void cavidadXRK(int iteraciones, long double umbral)
```

```
{
    matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
        epsilon1,*epsilon2,*astigLin1,*astigLin2;
    matriz *qt1_ml, *qs1_ml, *qt2_ml, *qs2_ml, *wt1_ml, *
        ws1_ml, *wt2_ml, *ws2_ml, *astigML1, *astigML2;

    // Llenando epsilon1 y epsilon2
    epsilon1=nuevaMatriz(1,41);
    epsilon2=nuevaMatriz(1,41); // Columnas
    fijaElemento(epsilon1,1,1, 2.0e 3);
    fijaElemento(epsilon2,1,1, 2.0e 3);
    for(int index=2;index<=41;index++)
    {
        fijaElemento(epsilon1,1,index,obtieneElemento(
            epsilon1,1,index 1)+0.1e 3);
        fijaElemento(epsilon2,1,index,obtieneElemento(
            epsilon2,1,index 1)+0.1e 3);
    }

    // Creando matrices receptoras
    wt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    ws1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    wt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    ws2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qs1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qs2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );

    // Cálculo lineal
    calculoLineal("lol.csv","fin","inf",n0,lambda0,L1,L2,
        L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,qs1,
        qt2,qs2);
    // Astigmatismo lineal (tan/sag)
    astigLin1=divMatricesElemAElem(wt1,ws1);
    astigLin2=divMatricesElemAElem(wt2,ws2);

    // Cálculo no lineal
```

```
qt1_ml=propNoLinealRK_tanEM1(qt1,"fin","inf",L1,L2,L,
    f1,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qs1_ml=propNoLinealRK_sagEM1(qs1,"fin","inf",L1,L2,L,
    f1,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qt2_ml=propNoLinealRK_tanEM2(qt2,"fin","inf",L1,L2,L,
    f1,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qs2_ml=propNoLinealRK_sagEM2(qs2,"fin","inf",L1,L2,L,
    f1,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);

// Cálculo de spots
wt1_ml=spot_q_matriz(qt1_ml,1,lambda0);
ws1_ml=spot_q_matriz(qs1_ml,1,lambda0);
wt2_ml=spot_q_matriz(qt2_ml,1,lambda0);
ws2_ml=spot_q_matriz(qs2_ml,1,lambda0);

// Cálculo de astigmatismo (tan/sag)
astigML1=divMatricesElemAElem(wt1_ml,ws1_ml);
astigML2=divMatricesElemAElem(wt2_ml,ws2_ml);

// Cálculo de distancias
long double delta1, delta2, angulos[2];
anguloLineal("fin","inf",L,n0,L1,L2,f1,f2,angulos);
delta1=distanciaCristal("fin",f1,L1,L,n0,angulos[0]);
delta2=distanciaCristal("inf",f2,L2,L,n0,angulos[1]);

// Búsqueda de configuración menos astigmática
int fila1, columna1,fila2, columna2;
long double complex valor1, valor2;
confNoAstigmatica(astigML1,&valor1,&fila1,&columna1);
confNoAstigmatica(astigML2,&valor2,&fila2,&columna2);

// Escritura de archivos
escribeMatrizArchivo(qt1_ml,"qt1_ml_RK.csv");
escribeMatrizArchivo(qs1_ml,"qs1_ml_RK.csv");
escribeMatrizArchivo(qt2_ml,"qt2_ml_RK.csv");
escribeMatrizArchivo(qs2_ml,"qs2_ml_RK.csv");
escribeMatrizArchivo_ejes(astigLin1,epsilon1,epsilon2
    ,"astigLinEM1.csv");
```

```
    escribeMatrizArchivo_ejes(astigLin2,epsilon1,epsilon2
        ,"astigLinEM2.csv");
    escribeMatrizArchivo_completo(astigML1,epsilon1,
        epsilon2,angulos,delta1,delta2,"Finito","Infinito"
        ,"astigML1_RK.csv",fila1,columna1,valor1);
    escribeMatrizArchivo_completo(astigML2,epsilon1,
        epsilon2,angulos,delta1,delta2,"Finito","Infinito"
        ,"astigML2_RK.csv",fila2,columna2,valor2);

    wt1=borraMatriz(wt1);
    ws1=borraMatriz(ws1);
    wt2=borraMatriz(wt2);
    ws2=borraMatriz(ws2);
    qt1=borraMatriz(qt1);
    qs1=borraMatriz(qs1);
    qt2=borraMatriz(qt2);
    qs2=borraMatriz(qs2);
    astigLin1=borraMatriz(astigLin1);
    astigLin2=borraMatriz(astigLin2);

    qt1_ml=borraMatriz(qt1_ml);
    qs1_ml=borraMatriz(qs1_ml);
    qt2_ml=borraMatriz(qt2_ml);
    qs2_ml=borraMatriz(qs2_ml);
    wt1_ml=borraMatriz(wt1_ml);
    ws1_ml=borraMatriz(ws1_ml);
    wt2_ml=borraMatriz(wt2_ml);
    ws2_ml=borraMatriz(ws2_ml);
    astigML1=borraMatriz(astigML1);
    astigML2=borraMatriz(astigML2);

    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}

void cavidadXMatrices(int iteraciones, long double umbral
    )
{
    matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
        epsilon1,*epsilon2,*astigLin1,*astigLin2;
    matriz *qt1_ml, *qs1_ml, *qt2_ml, *qs2_ml, *wt1_ml, *
        ws1_ml, *wt2_ml, *ws2_ml, *astigML1, *astigML2;

    // Llenando epsilon1 y epsilon2
    epsilon1=nuevaMatriz(1,41);
    epsilon2=nuevaMatriz(1,41); // Columnas
```

```
fijaElemento(epsilon1,1,1, 2.0e 3);
fijaElemento(epsilon2,1,1, 2.0e 3);
for(int index=2;index<=41;index++)
{
    fijaElemento(epsilon1,1,index,obtieneElemento(
        epsilon1,1,index 1)+0.1e 3);
    fijaElemento(epsilon2,1,index,obtieneElemento(
        epsilon2,1,index 1)+0.1e 3);
}

// Creando matrices receptoras
wt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
ws1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
wt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
ws2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qs1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
qs2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
    );
//astigLin1=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//astigLin2=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);

//qt1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//qs1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//qt2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//qs2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//wt1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//ws1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
//wt2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
    columnas);
```

```
//ws2_ml=nuevaMatriz(epsilon1 >columnas, epsilon2 >
    columnas);
//astigML1=nuevaMatriz(epsilon1 >columnas, epsilon2 >
    columnas);
//astigML2=nuevaMatriz(epsilon1 >columnas, epsilon2 >
    columnas);

// Cálculo lineal
//calculoLineal("lol.csv","inf","inf",n0,lambda0,a,b,
    c,L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,
    qs1,qt2,qs2);
calculoLineal("lol.csv","fin","inf",n0,lambda0,L1,L2,
    L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,qs1,
    qt2,qs2);
// Astigmatismo lineal (tan/sag)
astigLin1=divMatricesElemAElem(wt1,ws1);
astigLin2=divMatricesElemAElem(wt2,ws2);

// Cálculo no lineal
qt1_ml=propNoLineal_tanEM1(qt1,"fin","inf",L1,L2,L,f1
    ,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qs1_ml=propNoLineal_sagEM1(qs1,"fin","inf",L1,L2,L,f1
    ,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qt2_ml=propNoLineal_tanEM2(qt2,"fin","inf",L1,L2,L,f1
    ,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);
qs2_ml=propNoLineal_sagEM2(qs2,"fin","inf",L1,L2,L,f1
    ,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
    P_laser,lambda0,epsilon1,epsilon2,iteraciones,
    umbral);

// Cálculo de spots
wt1_ml=spot_q_matriz(qt1_ml,1,lambda0);
ws1_ml=spot_q_matriz(qs1_ml,1,lambda0);
wt2_ml=spot_q_matriz(qt2_ml,1,lambda0);
ws2_ml=spot_q_matriz(qs2_ml,1,lambda0);

// Cálculo de astigmatismo (tan/sag)
astigML1=divMatricesElemAElem(wt1_ml,ws1_ml);
astigML2=divMatricesElemAElem(wt2_ml,ws2_ml);
```

```
// Cálculo de distancias
long double delta1, delta2, angulos[2];
anguloLineal("fin","inf",L,n0,L1,L2,f1,f2,angulos);
delta1=distanciaCristal("fin",f1,L1,L,n0,angulos[0]);
delta2=distanciaCristal("inf",f2,L2,L,n0,angulos[1]);

// Búsqueda de configuración menos astigmática
int fila1, columna1,fila2, columna2;
long double complex valor1, valor2;
confNoAstigmatica(astigML1,&valor1,&fila1,&columna1);
confNoAstigmatica(astigML2,&valor2,&fila2,&columna2);

// Escritura de archivos
escribeMatrizArchivo(qt1_ml,"qt1_ml_matriz.csv");
escribeMatrizArchivo(qs1_ml,"qs1_ml_matriz.csv");
escribeMatrizArchivo(qt2_ml,"qt2_ml_matriz.csv");
escribeMatrizArchivo(qs2_ml,"qs2_ml_matriz.csv");
escribeMatrizArchivo_ejes(astigLin1,epsilon1,epsilon2
    ,"astigLinEM1.csv");
escribeMatrizArchivo_ejes(astigLin2,epsilon1,epsilon2
    ,"astigLinEM2.csv");
escribeMatrizArchivo_completo(astigML1,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML1_matriz.csv",fila1,columna1,valor1);
escribeMatrizArchivo_completo(astigML2,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML2_matriz.csv",fila2,columna2,valor2);

wt1=borraMatriz(wt1);
ws1=borraMatriz(ws1);
wt2=borraMatriz(wt2);
ws2=borraMatriz(ws2);
qt1=borraMatriz(qt1);
qs1=borraMatriz(qs1);
qt2=borraMatriz(qt2);
qs2=borraMatriz(qs2);
astigLin1=borraMatriz(astigLin1);
astigLin2=borraMatriz(astigLin2);

qt1_ml=borraMatriz(qt1_ml);
qs1_ml=borraMatriz(qs1_ml);
qt2_ml=borraMatriz(qt2_ml);
qs2_ml=borraMatriz(qs2_ml);
wt1_ml=borraMatriz(wt1_ml);
ws1_ml=borraMatriz(ws1_ml);
wt2_ml=borraMatriz(wt2_ml);
```

```
    ws2_ml=borraMatriz(ws2_ml);
    astigML1=borraMatriz(astigML1);
    astigML2=borraMatriz(astigML2);

    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}

void CavidadXMatricesAcopladas(int iteraciones, long
   double umbral, int N)
{
    // Lente térmica  en el futuro se podrá propagar
       desde la fuente láser
    long double w_pump_t=2e-3;
    long double w_pump_s=2e-3;
    long double complex pT,pS,qT,qS;
    long double ancho=6e-3, alto=6e-3;
    pT=1/100e-3*I*535e-9/(nPump*M_PI*powl(w_pump_t,2));
    qT=1/pT;
    pS=1/100e-3*I*535e-9/(nPump*M_PI*powl(w_pump_s,2));
    qS=1/pS;
    long double alpha=1/1e-2;
    int pasos=1000;
    int pasosKerr=1000;
    long double lambda_relax=1.5; // Valor de relajación
       para el método Liebmann (Gauss Seidel) de sol. de
       EDP.
    ajusteTemperaturaCristal *ajuste=vectorAjusteCristal(
       qT,qS,alpha,lambdaPump,n0,P_pump,chi,L,kth,Cp,rho,
       dn_dv,ancho,alto,iteraciones,pasos,umbral,
       lambda_relax,N);
    printf("Cálculo térmico terminado\n");

    // Cálculo CW
    matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
       epsilon1,*epsilon2,*astigLin1,*astigLin2;
    matriz *wt1_ml, *ws1_ml, *wt2_ml, *ws2_ml, *astigML1,
        *astigML2;

    // Llenando epsilon1 y epsilon2
    epsilon1=nuevaMatriz(1,41);
    epsilon2=nuevaMatriz(1,41); // Columnas
    fijaElemento(epsilon1,1,1, 2.0e-3);
    fijaElemento(epsilon2,1,1, 2.0e-3);
    for(int index=2;index<=41;index++)
    {
```

```
        fijaElemento(epsilon1,1,index,obtieneElemento(
            epsilon1,1,index 1)+0.1e 3);
        fijaElemento(epsilon2,1,index,obtieneElemento(
            epsilon2,1,index 1)+0.1e 3);
    }

    // Creando matrices receptoras
    wt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    ws1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    wt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    ws2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qt1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qs1=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qt2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    qs2=nuevaMatriz(epsilon1>columnas,epsilon2>columnas
        );
    /*astigLin1=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    astigLin2=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);

    qt1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    qs1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    qt2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    qs2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);*/
    matriz **qAstigEM1_ML;
    matriz **qAstigEM2_ML;
    /*wt1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    ws1_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    wt2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
    ws2_ml=nuevaMatriz(epsilon1>columnas,epsilon2>
        columnas);
```

```
astigML1=nuevaMatriz(epsilon1 >columnas, epsilon2 >
    columnas);
astigML2=nuevaMatriz(epsilon1 >columnas, epsilon2 >
    columnas);*/

// Cálculo lineal
//calculoLineal("lol.csv","inf","inf",n0,lambda0,a,b,
    c,L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,
    qs1,qt2,qs2);
calculoLineal("lol.csv","fin","inf",n0,lambda0,L1,L2,
    L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,qs1,
    qt2,qs2);
// Astigmatismo lineal (tan/sag)
astigLin1=divMatricesElemAElem(wt1,ws1);
astigLin2=divMatricesElemAElem(wt2,ws2);

// Cálculo no lineal

qAstigEM1_ML=propNoLinealEM1Astigmatico(qt1,qs1,
    ajuste,"fin","inf",L1,L2,L,f1,f2,n0,n2,chi,kth,Cp,
    rho,dn_dv,P_laser,lambda0,epsilon1,epsilon2,
    iteraciones,pasosKerr,umbral);
qAstigEM2_ML=propNoLinealEM2Astigmatico(qt2,qs2,
    ajuste,"fin","inf",L1,L2,L,f1,f2,n0,n2,chi,kth,Cp,
    rho,dn_dv,P_laser,lambda0,epsilon1,epsilon2,
    iteraciones,pasosKerr,umbral);

// Cálculo de spots
wt1_ml=spot_q_matriz(qAstigEM1_ML[0],1,lambda0);
ws1_ml=spot_q_matriz(qAstigEM1_ML[1],1,lambda0);
wt2_ml=spot_q_matriz(qAstigEM2_ML[0],1,lambda0);
ws2_ml=spot_q_matriz(qAstigEM2_ML[1],1,lambda0);

// Cálculo de astigmatismo (tan/sag)
astigML1=divMatricesElemAElem(wt1_ml,ws1_ml);
astigML2=divMatricesElemAElem(wt2_ml,ws2_ml);

// Cálculo de distancias
long double delta1, delta2, angulos[2];
anguloLineal("fin","inf",L,n0,L1,L2,f1,f2,angulos);
delta1=distanciaCristal("fin",f1,L1,L,n0,angulos[0]);
delta2=distanciaCristal("inf",f2,L2,L,n0,angulos[1]);

// Búsqueda de configuración menos astigmática
int fila1, columna1,fila2, columna2;
```

```
long double complex valor1, valor2;
confNoAstigmatica(astigML1,&valor1,&fila1,&columna1);
confNoAstigmatica(astigML2,&valor2,&fila2,&columna2);

// Escritura de archivos
escribeMatrizArchivo(qAstigEM1_ML[0],"
    qt1_ml_matriz_acoplado.csv");
escribeMatrizArchivo(qAstigEM1_ML[1],"
    qs1_ml_matriz_acoplado.csv");
escribeMatrizArchivo(qAstigEM2_ML[0],"
    qt2_ml_matriz_acoplado.csv");
escribeMatrizArchivo(qAstigEM2_ML[1],"
    qs2_ml_matriz_acoplado.csv");
escribeMatrizArchivo_ejes(astigLin1,epsilon1,epsilon2
    ,"astigLinEM1.csv");
escribeMatrizArchivo_ejes(astigLin2,epsilon1,epsilon2
    ,"astigLinEM2.csv");
escribeMatrizArchivo_completo(astigML1,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML1_matriz_acoplado.csv",fila1,columna1,
    valor1);
escribeMatrizArchivo_completo(astigML2,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML2_matriz_acoplado.csv",fila2,columna2,
    valor2);

wt1=borraMatriz(wt1);
ws1=borraMatriz(ws1);
wt2=borraMatriz(wt2);
ws2=borraMatriz(ws2);
qt1=borraMatriz(qt1);
qs1=borraMatriz(qs1);
qt2=borraMatriz(qt2);
qs2=borraMatriz(qs2);
astigLin1=borraMatriz(astigLin1);
astigLin2=borraMatriz(astigLin2);

wt1_ml=borraMatriz(wt1_ml);
ws1_ml=borraMatriz(ws1_ml);
wt2_ml=borraMatriz(wt2_ml);
ws2_ml=borraMatriz(ws2_ml);
astigML1=borraMatriz(astigML1);
astigML2=borraMatriz(astigML2);
free(qAstigEM1_ML[0]);
qAstigEM1_ML[0]=NULL;
free(qAstigEM1_ML[1]);
```

```
    qAstigEM1_ML[1]=NULL;
    free(qAstigEM1_ML);
    qAstigEM1_ML=NULL;
    free(qAstigEM2_ML[0]);
    qAstigEM2_ML[0]=NULL;
    free(qAstigEM2_ML[1]);
    qAstigEM2_ML[1]=NULL;
    free(qAstigEM2_ML);
    qAstigEM1_ML=NULL;
    free(ajuste->a0);
    ajuste->a0=NULL;
    free(ajuste->a1);
    ajuste->a1=NULL;
    free(ajuste->a2);
    ajuste->a2=NULL;
    free(ajuste->paso);
    ajuste->paso=NULL;
    free(ajuste);
    ajuste=NULL;
    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}

void CavidadXMatricesAcopladasSoloKerr(int iteraciones,
   long double umbral, int N)
{
    // Lente térmica  en el futuro se podrá propagar
        desde la fuente láser
    long double w_pump_t=1.0;
    long double w_pump_s=1.0;
    long double complex pT,pS,qT,qS;
    long double ancho=6e-3, alto=6e-3;
    pT=1/(100e-3*I*535e-9/(nPump*M_PI*pow(w_pump_t,2)));
    qT=1/pT;
    pS=1/(100e-3*I*535e-9/(nPump*M_PI*pow(w_pump_s,2)));
    qS=1/pS;
    long double alpha=1/1e-2;
    int pasos=1000;
    int pasosKerr=1000;
    long double lambda_relax=1.5; // Valor de relajación
        para el método Liebmann (Gauss Seidel) de sol. de
        EDP.
    ajusteTemperaturaCristal *ajuste=vectorAjusteCristal(
        qT,qS,alpha,lambdaPump,n0,0,chi,L,kth,Cp,rho,dn_dv
        ,ancho,alto,iteraciones,pasos,umbral,lambda_relax,
        N);
```

```
printf("Cálculo térmico terminado\n");

// Cálculo CW
matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
    epsilon1,*epsilon2,*astigLin1,*astigLin2;
matriz *wt1_ml, *ws1_ml, *wt2_ml, *ws2_ml, *astigML1,
     *astigML2;

// Llenando epsilon1 y epsilon2
epsilon1=nuevaMatriz(1,41);
epsilon2=nuevaMatriz(1,41); // Columnas
fijaElemento(epsilon1,1,1, 2.0e 3);
fijaElemento(epsilon2,1,1, 2.0e 3);
for(int index=2;index<=41;index++)
{
     fijaElemento(epsilon1,1,index,obtieneElemento(
         epsilon1,1,index 1)+0.1e 3);
     fijaElemento(epsilon2,1,index,obtieneElemento(
         epsilon2,1,index 1)+0.1e 3);
}

// Creando matrices receptoras
wt1=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
ws1=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
wt2=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
ws2=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
qt1=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
qs1=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
qt2=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );
qs2=nuevaMatriz(epsilon1 >columnas,epsilon2 >columnas
    );

matriz **qAstigEM1_ML;
matriz **qAstigEM2_ML;

// Cálculo lineal
//calculoLineal("lol.csv","inf","inf",n0,lambda0,a,b,
    c,L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,
    qs1,qt2,qs2);
```

```
calculoLineal("lol.csv","fin","inf",n0,lambda0,L1,L2,
    L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,qs1,
    qt2,qs2);
// Astigmatismo lineal (tan/sag)
astigLin1=divMatricesElemAElem(wt1,ws1);
astigLin2=divMatricesElemAElem(wt2,ws2);

// Cálculo no lineal

qAstigEM1_ML=propNoLinealEM1AstigmaticoKerr(qt1,qs1,
    ajuste,"fin","inf",L1,L2,L,f1,f2,n0,n2,chi,kth,Cp,
    rho,dn_dv,P_laser,lambda0,epsilon1,epsilon2,
    iteraciones,pasosKerr,umbral);
qAstigEM2_ML=propNoLinealEM2AstigmaticoKerr(qt2,qs2,
    ajuste,"fin","inf",L1,L2,L,f1,f2,n0,n2,chi,kth,Cp,
    rho,dn_dv,P_laser,lambda0,epsilon1,epsilon2,
    iteraciones,pasosKerr,umbral);

// Cálculo de spots
wt1_ml=spot_q_matriz(qAstigEM1_ML[0],1,lambda0);
ws1_ml=spot_q_matriz(qAstigEM1_ML[1],1,lambda0);
wt2_ml=spot_q_matriz(qAstigEM2_ML[0],1,lambda0);
ws2_ml=spot_q_matriz(qAstigEM2_ML[1],1,lambda0);

// Cálculo de astigmatismo (tan/sag)
astigML1=divMatricesElemAElem(wt1_ml,ws1_ml);
astigML2=divMatricesElemAElem(wt2_ml,ws2_ml);

// Cálculo de distancias
long double delta1, delta2, angulos[2];
anguloLineal("fin","inf",L,n0,L1,L2,f1,f2,angulos);
delta1=distanciaCristal("fin",f1,L1,L,n0,angulos[0]);
delta2=distanciaCristal("inf",f2,L2,L,n0,angulos[1]);

// Búsqueda de configuración menos astigmática
int fila1, columna1,fila2, columna2;
long double complex valor1, valor2;
confNoAstigmatica(astigML1,&valor1,&fila1,&columna1);
confNoAstigmatica(astigML2,&valor2,&fila2,&columna2);

// Escritura de archivos
escribeMatrizArchivo(qAstigEM1_ML[0],"
    qt1_ml_matriz_acopladoKerr.csv");
escribeMatrizArchivo(qAstigEM1_ML[1],"
    qs1_ml_matriz_acopladoKerr.csv");
escribeMatrizArchivo(qAstigEM2_ML[0],"
```

```
    qt2_ml_matriz_acopladoKerr.csv");
escribeMatrizArchivo(qAstigEM2_ML[1],"
    qs2_ml_matriz_acopladoKerr.csv");
escribeMatrizArchivo_ejes(astigLin1,epsilon1,epsilon2
    ,"astigLinEM1.csv");
escribeMatrizArchivo_ejes(astigLin2,epsilon1,epsilon2
    ,"astigLinEM2.csv");
escribeMatrizArchivo_completo(astigML1,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML1_matriz_acopladoKerr.csv",fila1,columna1
    ,valor1);
escribeMatrizArchivo_completo(astigML2,epsilon1,
    epsilon2,angulos,delta1,delta2,"Finito","Infinito"
    ,"astigML2_matriz_acopladoKerr.csv",fila2,columna2
    ,valor2);

wt1=borraMatriz(wt1);
ws1=borraMatriz(ws1);
wt2=borraMatriz(wt2);
ws2=borraMatriz(ws2);
qt1=borraMatriz(qt1);
qs1=borraMatriz(qs1);
qt2=borraMatriz(qt2);
qs2=borraMatriz(qs2);
astigLin1=borraMatriz(astigLin1);
astigLin2=borraMatriz(astigLin2);

wt1_ml=borraMatriz(wt1_ml);
ws1_ml=borraMatriz(ws1_ml);
wt2_ml=borraMatriz(wt2_ml);
ws2_ml=borraMatriz(ws2_ml);
astigML1=borraMatriz(astigML1);
astigML2=borraMatriz(astigML2);
free(qAstigEM1_ML[0]);
qAstigEM1_ML[0]=NULL;
free(qAstigEM1_ML[1]);
qAstigEM1_ML[1]=NULL;
free(qAstigEM1_ML);
qAstigEM1_ML=NULL;
free(qAstigEM2_ML[0]);
qAstigEM2_ML[0]=NULL;
free(qAstigEM2_ML[1]);
qAstigEM2_ML[1]=NULL;
free(qAstigEM2_ML);
qAstigEM1_ML=NULL;
free(ajuste>a0);
```

```
    ajuste->a0=NULL;
    free(ajuste->a1);
    ajuste->a1=NULL;
    free(ajuste->a2);
    ajuste->a2=NULL;
    free(ajuste->paso);
    ajuste->paso=NULL;
    free(ajuste);
    ajuste=NULL;
    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}


void cavidadXRK_soloSag2(int iteraciones, long double
    umbral)
{
    matriz *wt1,*ws1,*wt2,*ws2,*qt1,*qs1,*qt2,*qs2,*
        epsilon1,*epsilon2,*astigLin1,*astigLin2;
    matriz *qt1_ml, *qs1_ml, *qt2_ml, *qs2_ml, *wt1_ml, *
        ws1_ml, *wt2_ml, *ws2_ml, *astigML1, *astigML2;

    // Llenando epsilon1 y epsilon2
    epsilon1=nuevaMatriz(1,41);
    epsilon2=nuevaMatriz(1,41); // Columnas
    fijaElemento(epsilon1,1,1, 2.0e 3);
    fijaElemento(epsilon2,1,1, 2.0e 3);
    for(int index=2;index<=41;index++)
    {
        fijaElemento(epsilon1,1,index,obtieneElemento(
            epsilon1,1,index 1)+0.1e 3);
        fijaElemento(epsilon2,1,index,obtieneElemento(
            epsilon2,1,index 1)+0.1e 3);
    }

    // Creando matrices receptoras
    wt1=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    ws1=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    wt2=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    ws2=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    qt1=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
```

```
    qs1=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    qt2=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );
    qs2=nuevaMatriz(epsilon1->columnas,epsilon2->columnas
        );

    // Cálculo lineal
    calculoLineal("lol.csv","fin","inf",n0,lambda0,L1,L2,
        L,f1,f2,wt1,ws1,wt2,ws2,epsilon1,epsilon2,qt1,qs1,
        qt2,qs2);
    // Astigmatismo lineal (tan/sag)

    // Cálculo no lineal
    qs2_ml=propNoLinealRK_sagEM2(qs2,"fin","inf",L1,L2,L,
        f1,f2,n0,n2,w_pump,chi,kth,Cp,rho,dn_dv,P_pump,
        P_laser,lambda0,epsilon1,epsilon2,iteraciones,
        umbral);

    // Cálculo de spots
    ws2_ml=spot_q_matriz(qs2_ml,1,lambda0);

    // Escritura de archivos

    wt1=borraMatriz(wt1);
    ws1=borraMatriz(ws1);
    wt2=borraMatriz(wt2);
    ws2=borraMatriz(ws2);
    qt1=borraMatriz(qt1);
    qs1=borraMatriz(qs1);
    qt2=borraMatriz(qt2);
    qs2=borraMatriz(qs2);
    ws2_ml=borraMatriz(ws2_ml);
    astigML1=borraMatriz(astigML1);
    astigML2=borraMatriz(astigML2);

    epsilon1=borraMatriz(epsilon1);
    epsilon2=borraMatriz(epsilon2);
}

int main_old()
{
    int iteraciones=10000;
    long double umbral=1.0e3; // Porcentaje
    int N=51;
    time_t inicioMatrices, finMatrices, inicioRK, finRK,
```

```
        inicioMatAcop, finMatAcop, inicioMatAcopKerr,
        finMatAcopKerr;
    double tiempoMatrices, tiempoRK, tiempoMatAcop,
        tiempoMatAcopKerr;
    time(&inicioMatrices);
    cavidadXMatrices(iteraciones,umbral);
    time(&finMatrices);
    tiempoMatrices=difftime(finMatrices,inicioMatrices);
    time(&inicioRK);
    //cavidadXRK(iteraciones,umbral);
    time(&finRK);
    tiempoRK=difftime(finRK,inicioRK);
    time(&inicioMatAcop);
    //CavidadXMatricesAcopladas(iteraciones,umbral,N);
    time(&finMatAcop);
    tiempoMatAcop=difftime(finMatAcop,inicioMatAcop);
    time(&inicioMatAcopKerr);
    CavidadXMatricesAcopladasSoloKerr(iteraciones,umbral,
        N);
    time(&finMatAcopKerr);
    tiempoMatAcop=difftime(finMatAcopKerr,
        inicioMatAcopKerr);
    printf("Tiempo transcurrido para rutina con matrices:
         %f [s]\n",tiempoMatrices);
    printf("Tiempo transcurrido para rutina con Runge
        Kutta: %f [s]\n",tiempoRK);
    printf("Tiempo transcurrido para rutina matricial con
         planos acoplados y lente térmica: %f [s]\n",
        tiempoMatAcop);
    printf("Tiempo transcurrido para rutina matricial con
         planos acoplados sin lente térmica: %f [s]\n",
        tiempoMatAcopKerr);
    return 0;
}

int main()
{
    int iteraciones=10000;
    long double umbral=1.0e3; // Porcentaje
    cavidadXRK_soloSag2(iteraciones,umbral);
    return 0;
}
```

main.c del graficador de la propagación

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include "constantes.h"
#include "propGrafica.h"

int main()
{
    long double epsilon1 = 0.001;
    long double epsilon2 = 0.0009;
    //long double epsilon1 = 0.001;
    //long double epsilon2 = 0.0009;

    long double w_pump_t=2e 3;
    long double w_pump_s=2e 3;
    long double ancho=6e 3, alto=6e 3;
    graficaPropagacion("fin","inf",50000,0.0001,51,
        epsilon1,epsilon2,lambdaPump,n0,n2,P_laser,P_pump,
        chi,L,kth,Cp,rho,dn_dv,ancho,alto,w_pump_t,
        w_pump_s,nPump,lambda0,L1,L2,f1,f2);

    system("gnuplot script.plot");
    return 0;
}
```