UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIA DE LA COMPUTACIÓN

Evaluación de Algoritmos de Optimización de Primer Orden para la Optimización Topológica Robusta de Dispositivos Nanofotónicos

TESIS

Para optar el título profesional de Licenciado en Ciencia de la Computación

AUTOR:

José Leonidas García Gonzales

ASESOR

Jorge Luis Gonzalez Reaño

Lima - Perú

25 de junio de 2022

Índice general

Pág.

Índice de tablas

Índice de figuras

Capítulo 1

Marco Teórico

En el presente capítulo se introducen conceptos fundamentales relacionados al diseño inverso de dispositivos fotónicos. Para ello se desarrolla seis secciones. Primero, se describen las propiedades físicas de interés de un *bend* y WDM. Segundo, se explica como parametrizar la región de diseño de estos dispositivos. Tercero, se detalla los pasos necesarios para poder simular computacionalmente los diseños realizados. Cuarto, se explica la estrategia de optimización a seguir con la parametrización señalada. Quinto, se describen tres algoritmos que se usaran en el presente trabajo: (i) algoritmos genéticos (GA), (ii) *particle swarm optimization* (CMA), (iii) *covariance matrix adaptation evolution strategy* (CMA-ES). Finalmente, se expone que transformaciones se puede aplicar dentro de la estrategia a seguir.

TODO: Actualizar esto al final.

1.1 Dispositivos de estudio

1.1.1 *Bend*

Un *bend* es un dispositivo fotónico que se encarga de guiar el giro de un haz de ondas.

En general, al estudiar dispositivos fotónicos es de especial interés la distribución del campo eléctrico ($\boldsymbol{E}$). Este nos permite visualizar la distribución de la energía en un dispositivo calculando lo siguiente:

$$|\boldsymbol{E}|^2 = |\boldsymbol{E_x}|^2 + |\boldsymbol{E_y}|^2 + |\boldsymbol{E_z}|^2, \tag{1.1}$$

donde $\boldsymbol{E_x}, \boldsymbol{E_y}, \boldsymbol{E_z}$ representan las componentes del campo eléctrico en los ejes x, y, z, respectivamente (Lukas Chrostowski, 2010).

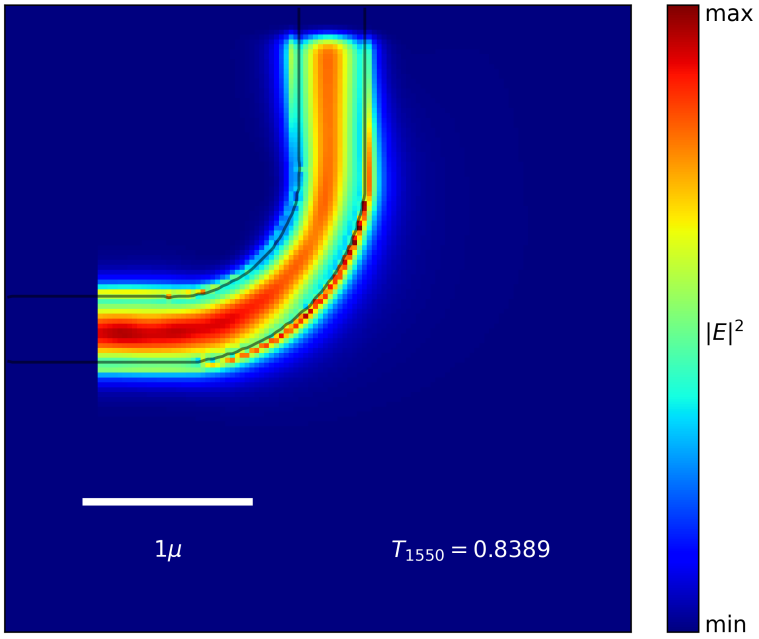


FIGURA 1.1: Representación de $|\boldsymbol{E}|^2$ de un *bend* de $1\mu m$ de radio obtenido usando 3D FDFD bajo una resolución de $30nm$.

Tradicionalmente, un *bend* consiste en una una guía de onda horizontal usada como entrada y una guía de onda vertical usada como salida, estas son conectadas por una guía de onda con la forma de un cuarto de circunferencia de radio r y con el mismo grosor de las guías de onda de entrada y salida. En la Figura 1.1 se muestra un *bend* tradicional de radio $r = 1\mu m$. Como se observa en la imagen, parte significativa de la energía se pierde en la región curva. Esto se debe a que el radio de curvatura es muy pequeño, con un valor más grande (e.g. $r = 10\mu m$) las pérdidas se vuelven casi nulas (Lukas Chrostowski, 2010).

Para evitar ambigüedades, dos puntos adicionales a remarcar son: (i) cuando nos referimos al radio estamos haciendo mención al radio medio de curvatura y (ii) todos los diseños mostrados en este trabajo tienen un profundidad de 220nm.

Observar en una gráfica el valor de $|\boldsymbol{E}|^2$ nos ayuda a entender el funcionamiento de un dispositivo. Por otro lado, una manera de cuantificar que tan bien funciona un diseño es mediante el cálculo de la transmitancia (T). Este valor se define como la relación entre la potencia del flujo que sale del dispositivo con la potencia del flujo que ingresa (Christiansen y Sigmund, 2021).

Seguidamente, sea λ la longitud de onda de la entrada y $\boldsymbol{P}$ los parámetros que caracterizan al diseño, denotaremos como $T_\lambda(\boldsymbol{P})$ a la transmitancia asociada al dispositivo obtenido con la parametrización $\boldsymbol{P}$ en la longitud de onda λ. Luego, definimos la función objetivo (f_{obj}) para un *bend*, también conocido en el área como figura de mérito (FOM), mediante la siguiente ecuación (Su *et al.*, 2020):

$$f_{obj}(\boldsymbol{P}) = max\left\{T_{1550}(\boldsymbol{P})\right\}. \tag{1.2}$$

En síntesis, la idea detrás de estas definiciones es describir un *bend* mediante una parametrización $\boldsymbol{P}$ (Sección 1.2). Luego, usando algoritmos de optimización, buscar entre las distintas combinaciones de los parámetros aquella configuración que optimice la función f_{obj} (Sección 1.6). De este modo, estaremos encontrando un diseño con una elevada transmitancia, es decir con un buen desempeño.

1.1.2 *Wavelength Demultiplexer* (WDM)

Un WDM es un dispositivo fotónico que se encarga de guiar un haz de ondas de acuerdo a su longitud de onda. Por ejemplo, estos pueden trabajar con dos longitudes de

onda y guían las de una longitud por la guía de onda superior y las de otra longitud por la guía de onda inferior.

Análogo al caso del *bend*, utilizaremos la transmitancia para cuantificar el desempeño del dispositivo. Pero, en el presente trabajo usaremos un WDM con dos guías de salida, por ello, utilizaremos como notación $T_{\lambda}^{(1)}(\boldsymbol{P})$ para representar la transmitancia en la guía de salida superior cuando se recibe un haz de longitud de onda λ en un diseño descrito por los parámetros $\boldsymbol{P}$ y $T_{\lambda}^{(2)}(\boldsymbol{P})$ para la guía de salida inferior.

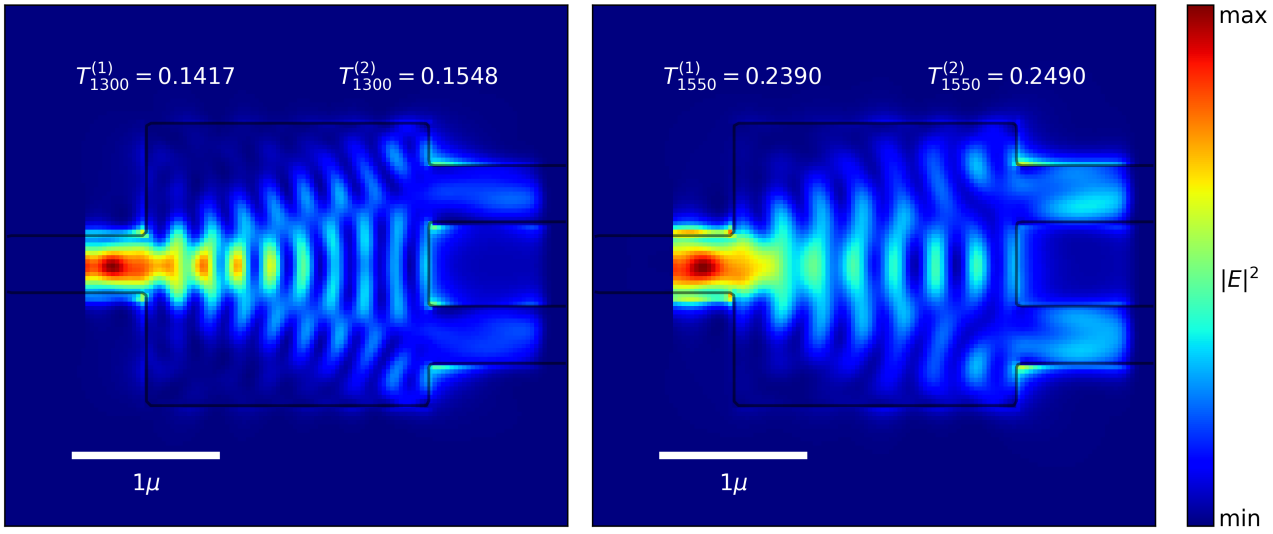


FIGURA 1.2: Representación de $|\boldsymbol{E}|^2$ de un WDM obtenido usando 3D FDFD bajo una resolución de $30nm$.

En la Figura 1.2 se muestra un WDM donde las guías de onda son unidas por una región rectangular de $2.0\mu m \times 2.0\mu m$. En el lado izquierdo se muestra la representación de $|\boldsymbol{E}|^2$ cuando se usa como entrada un haz de ondas de $1300nm$ de longitud, en el lado derecho la entrada es de $1550nm$. En ambos casos el diseño está funcionando más como un *splitter* que como un WDM. Un *splitter* es un dispositivo fotónico que divide la potencia del flujo de la entrada por las guías de salida en una determinada proporción. El diseño intuitivo para este dispositivo es el presentado en la imagen. Similar al caso del *bend*, con unas dimensiones un poco más grandes se puede conseguir pérdidas casi nulas

de energía (Lukas Chrostowski, 2010). Sin embargo, como se observa en los valores de la transmitancia, este diseño intuitivo no es apropiado para un WDM.

Basándonos en Su *et al.* (2020), definimos su FOM como

$$f_{obj}(\boldsymbol{P}) = max \left\{ \frac{\left(T^{(1)}_{1300}(\boldsymbol{P})\right)^2 + \left(1 - T^{(2)}_{1300}(\boldsymbol{P})\right)^2 + \left(1 - T^{(1)}_{1550}(\boldsymbol{P})\right)^2 + \left(T^{(2)}_{1550}(\boldsymbol{P})\right)^2}{4} \right\}. \tag{1.3}$$

La Ecuación 1.3 busca maximizar la transmitancia por la guía de onda superior y minimizarla para la guía de onda inferior cuando se recibe una longitud de onda de $1300nm$ y lo contrario para una longitud de onda de $1550nm$. Cabe destacar que la división por cuatro se realiza para asegurar que f_{obj} solamente tenga valores en el intervalo $[0, 1]$, al igual que sucede con la función objetivo del *bend*.

La idea para optimizar un WDM es la misma descrita en la anterior sección. En el resto del capítulo se describe en más detalle los siguientes pasos necesarios para lograr esto.

1.2 Parametrización

Tanto para el *bend* como para el WDM se define una región de diseño mediante una parametrización ($\boldsymbol{P}$) que pueda mapear un gran conjunto de dispositivos. Una de la estrategias más populares para esta tarea es usar parametrización basada en píxeles. Esta estrategia consiste en definir $\boldsymbol{P}$ como una matriz de n filas y m columnas con valores en el intervalo $[0, 1]$ (Molesky *et al.*, 2018).

Por ejemplo, en la Figura 1.3 se ha definido la matriz $\boldsymbol{P} : [1, n] \times [1, m] \rightarrow [0, 1]$ con $n = m = 18$ y se ha graficado sus valores usando escala gris (0 corresponde al color

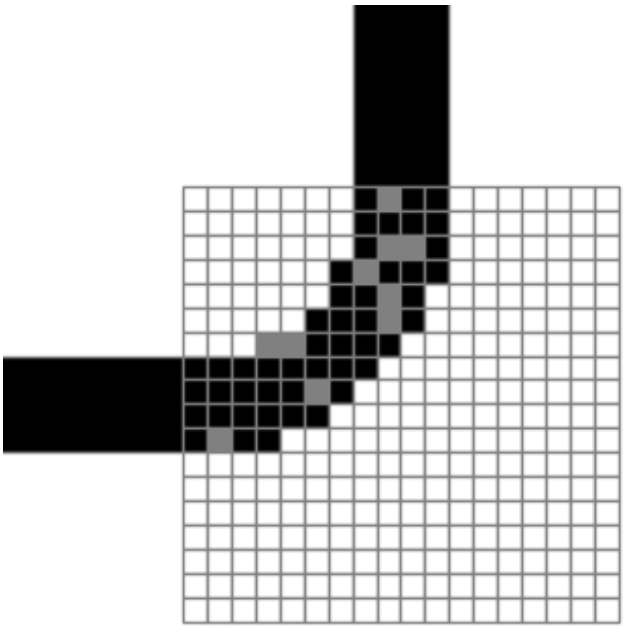


FIGURA 1.3: Parametrización basada en píxeles para un *bend* definiendo $\boldsymbol{P}$ como una matriz de 18×18.

blanco, 1 al negro y los demás valores a diferentes intensidades del gris). De esta manera observamos como $\boldsymbol{P}$ logra definir la geometría de un diseño. Adicionalmente, conforme se incrementa el valor de n y m se logra definir detalles con mayor precisión.

Sin embargo, lo anterior solo nos permite describir la geometría de los diseños, no sus propiedades. Además, no queda claro qué representan las regiones grises. Por ello, para describir las propiedades del diseño es necesario asociar a $\boldsymbol{P}$ alguna propiedad física. Esto lo realizamos calculando la permitividad ($\boldsymbol{\varepsilon}$) mediante la siguiente ecuación:

$$\boldsymbol{\varepsilon}(x, y) = \varepsilon_{Si} + (1 - \boldsymbol{P}(x, y))\varepsilon_{SiO_2} \mid x \in [1, n] \wedge y \in [1, m], \tag{1.4}$$

donde $\varepsilon_{Si} \approx 3.48^2$ es la permitividad del silicio (Si) y $\varepsilon_{SiO_2} \approx 1.44^2$ es la permitividad del óxido de silicio (SiO_2). De esta manera, la celda ubicada en la fila x, columna y de la geometría descrita por $\boldsymbol{P}(x, y)$ tiene una permitividad de valor $\boldsymbol{\varepsilon}$(x, y).

Con la Ecuación 1.4 estamos asociando a cada rectángulo de la geometría descrita por $\boldsymbol{P}$ un valor de permitividad en el rango $[\varepsilon_{SiO2} = 1.44, 3.48 = \varepsilon_{Si}]$. De esta manera, $\boldsymbol{P}(x, y) = 1$ describe que el rectángulo ubicado en la posición (x, y) es de Si, un valor de $\boldsymbol{P}(x, y) = 0$ describe la presencia de SiO_2 y $0 < \boldsymbol{P}(x, y) < 1$ hace referencia a algún material cuya permitividad es $\boldsymbol{\varepsilon}(x, y)$.

No obstante, un inconveniente de lo descrito es que $\boldsymbol{P}$ puede mapear a materiales inexistestes (regiones grises), mayor detalle de esta dificultad se estudia en la Sección 1.4. Por otro lado, con la descripción de la permitividad ya podemos calcular los campos eléctricos con lo cual se puede obtener el valor de f_{obj} definido para el *bend* y WDM. Por esta razón se realizan simulaciones electromagnéticas y se resuelven las inversas de las ecuaciones de Maxwell, esto permite obtener los campos eléctricos (Su *et al.*, 2020). Afortunadamente, para este proceso se puede utilizar distintos programas que implementan diversos métodos numéricos para realizar los cálculos (Sección 1.3).

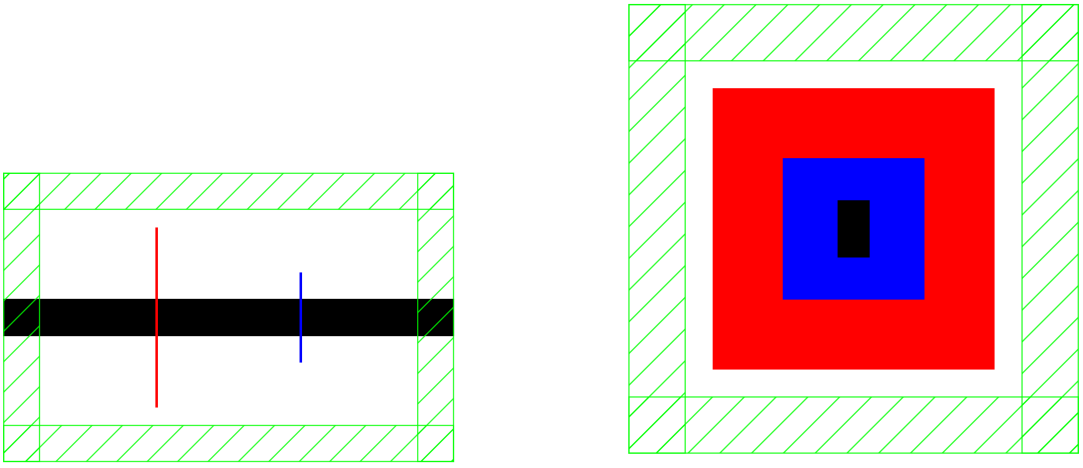
1.3 Simulación

Una vez tenemos definido un dispositivo con regiones fijas (guías de onda) y una región de diseño (descrita por la parametrización), es necesario incorporar tres elementos adicionales (Oskooi *et al.*, 2010, Su *et al.*, 2020):

1. **Fuente:** Suele representarse como un rectángulo en un plano perpendicular al flujo que pasa por el lugar donde este se ubica. Simula la emición de un haz de ondas por el diseño.

2. **Monitores:** Suelen representarse como un rectángulo similar a la fuente. Capturan información en su ubicación (e.g. valores del campo eléctrico).

3. ***Perfectly Matched Layer* (PML):** Representan las condiciones de frontera en la simulación. Se utilizan para limitar el espacio donde se deberá realizar las simulaciones computacionales.



(a) Vista frontal (b) Vista lateral

FIGURA 1.4: Configuración de simulación para una guía de onda.

En la Figura 1.4 se ilustra una guía de onda con los elementos mencionados. El rectángulo de color negro representa la guía de onda, la región roja la fuente, la región azul el monitor y lo verde el PML. Guiándonos de la figura de la izquierda, al realizar una simulación con esta configuración el flujo iría de la región roja a la región azul. Sin embargo, este seguiría expandiéndose por la guía de onda hasta llegar a la región verde (PML). En palabras sencillas, podemos interpretar el PML como una caja que permite aislar nuestro sistema. Por otro lado, la figura de la derecha representa el mismo diseño pero desde una vista lateral (recordemos que son diseños en 3D). Aquí podemos evidenciar como la fuente y el monitor son representados por rectángulos, no por rectas.

Siguiendo lo descrito, podemos utilizar diversos programas para configurar y ejecutar simulaciones de un *bend* y WDM. Una alternativa es usar SPINS-B junto a Maxwell-B, paquetes de Python de código abierto. Estos paquetes funcionan bajo una arquitectura cliente-servidor donde SPINS-B es el cliente y Maxwell-B es el servidor.

Por un lado, SPINS-B permite configurar diseños (fuente, monitores, PML, parametrización, región de diseño, etc) y solicitar al servidor que obtenga sus propiedades. Un aspecto interesante de este paquete es que trabaja con bloques llamados nodos. Estos representan operaciones fundamentales como suma, resta, multiplicación, etc. Luego, el programa guarda nuestra función objetivo (f_{obj}) usando un grafo que tiene estos nodos de vértices. Lo interesante de este enfoque es que permite calcular de forma eficiente (independientemente de las dimensiones de $\boldsymbol{P}$) el valor de $\nabla f_{obj}(\boldsymbol{P})$ mediante diferenciación automática (Kochenderfer y Wheeler, 2019, Su *et al.*, 2020).

Por otro lado, Maxwell-B implementa un método numérico conocido como *finite-difference frequency domain* (FDFD) para resolver numéricamente las ecuaciones de Maxwell y obtener así determinadas propiedades de los diseños que recibe. Es importante destacar que la implementación de esta librería es en GPUs de NVIDIA y permite aprovechar múltiples GPUs a la vez.

Un último detalle a señalar es que para las simulaciones debemos definir su resolución, esto lo representamos con el parámetro dx. Su valor se utiliza para discretizar nuestro espacio de simulación en una malla de cubos de dimensiones $dx \times dx \times dx$. En general, cuanto más pequeño es este valor los resultados son más precisos, pero la cantidad de memoria y el tiempo de simulación se incrementan considerablemente.

1.4 Estrategia de optimización

Como fue descrito en la Sección 1.2, podemos representar diseños con materiales que no existen. Para evitar esto se necesita obtener que $\boldsymbol{P}(x, y)$ de la Ecuación 1.4 tenga valores enteros. Sin embargo, al optimizar la función f_{obj} no podemos asegurar esta condición. Ante esta dificultad, Su *et al.* (2020) trabaja en dos etapas:

1. **Optimización continua**

En esta etapa se optimiza la función f_{obj} sin imponer ninguna restricción. En la Sección 1.6 se detalla algoritmos que suelen usarse para este fin.

2. **Optimización discreta**

   Se utiliza el resultado de la optimización continua como punto inicial para el algoritmo de optimización que se escoja. Luego, se trabaja por iteraciones. En cada iteración se aplica una transformación a cada diseño antes de evaluar su FOM. Esta transformación se escoge de tal manera que ayude a ir convergiendo a un diseño fabricable, es decir, a tener $\boldsymbol{P}(x, y) = 0$ o $\boldsymbol{P}(x, y) = 1$, más detalles en la Sección 1.5. Así, la idea de realizarlo por iteraciones es ir discretizando nuestro diseño de forma suave e intentando mantener un buen valor del FOM. Por ello, es crucial utilizar el resultado de una iteración como punto inicial de la próxima.

1.5 Transformaciones

En esta sección se discute dos transformaciones populares en optimización topológica para imponer restricciones de fabricación en nuestreos diseños: (i) filtro por densidad y (ii) proyección.

1.5.1 Filtro por Densidad

Antes de definir el filtro por densidad, necesitamos definir el concepto de bola cerrada. Para ello, tenemos que definir el concepto de función distancia en nuestra matriz de parametrización. Así, sea $x, x' \in [1, n] \wedge y, y' \in [1, m]$, definimos la distancia entre las posiciones (x, y) y (x', y') de la matriz $\boldsymbol{P}$ como:

$$dis(x, y, x', y') = |x - x'| * psize_y + |y - y'| * psize_x, \tag{1.5}$$

donde $psize_x$ es el valor de la longitud horizontal que representa cada celda de $\boldsymbol{P}$ y $psize_y$ es el recíproco para la longitud vertical.

De esta manera, definimos la bola cerrada ($\overline{B}_r(x,y)$) de centro (x,y) y radio $0 < r$ mediante el siguiente conjunto:

$$\overline{B}_r(x,y) = \{(x',y') | 1 \leq x' \leq n \wedge 1 \leq m \leq m \wedge dis(x,y,x',y') \leq r\}. \tag{1.6}$$

Seguidamente, usando estos conceptos en la matriz $\boldsymbol{P}$ se define el filtro por densidad ($\widetilde{\boldsymbol{P}}$) mediante la ecuación:

$$\widetilde{\boldsymbol{P}}(x,y) = \frac{\sum\limits_{(x',y') \in \overline{B}_{r_f}(x,y)} w_{x,y}(x',y') A(x',y') \boldsymbol{P}(x',y')}{\sum\limits_{(x',y') \in \overline{B}_{r_f}(x,y)} w_{x,y}(x',y') A(x',y')}, \tag{1.7}$$

$$w_{x,y}(x',y') = max(0, r_f - dis(x,y,x',y')), \tag{1.8}$$

donde $A(x,y)$ es el área de la celda representada por $\boldsymbol{P}(x,y)$. Por otro lado, el concepto de mínimo radio de curvatura (r_f) hace referencia a que un diseño se puede dibujar con una circunferencia de radio r_f. El objetivo de la Ecuación 1.8 es tratar de imponer este concepto al diseño.

Finalmente, de la Ecuación 1.7 podemos obtener el valor de su derivada parcial respecto al diseño original calculando lo siguiente:

$$\frac{\partial \widetilde{\boldsymbol{P}}(x,y)}{\partial \boldsymbol{P}(x^*,y^*)} = \frac{w_{x,y}(x^*,y^*) A(x^*,y^*)}{\sum\limits_{(x',y') \in \overline{B}_{r_f}(x,y)} w_{x,y}(x',y') A(x',y')}. \tag{1.9}$$

1.5.2 Proyección

La proyección descrita en la sección anterior ayuda a obtener diseños que eviten regiones punteagudas; sin embargo, esta transformación suele generar regiones grises. Por este motivo, es común acompañar el filtro por densidad con una proyección ($\widetilde{\widetilde{\boldsymbol{P}}}$) descrita como:

$$\widetilde{\widetilde{\boldsymbol{P}}}(x,y) = \frac{\tanh(\beta \times \eta) + \tanh(\beta \times (\widetilde{\boldsymbol{P}}(x,y) - \eta))}{\tanh(\beta \times \eta) + \tanh(\beta \times (1-\eta))}, \tag{1.10}$$

donde η y β son números escalares. Para entender el impacto de estos parámetros en la Ecuación 1.10, veamos la Figura 1.5. En la figura se está graficando esta ecuación con un valor fijo de $\eta = 0.5$ y distintos valores de β. Como podemos observar, con $\beta = 1$ tenemos prácticamente la función identidad y conforme aumenta el valor de β la función se aproxima más a una función escalonada con quiebre en 0.5. En realidad, en general el quiebre se realiza en el valor que se le de a η.

Además, de la Ecuación 1.10 podemos obserner el valor de su derivada parcial respecto al diseño $\widetilde{\boldsymbol{P}}$ mediante la siguiente ecuación:

$$\frac{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x,y)}{\partial \widetilde{\boldsymbol{P}}(x,y)} = \frac{(1 - \tanh^2(\beta \times (\widetilde{\boldsymbol{P}}(x,y) - \eta)))\beta}{\tanh(\beta \times \eta) + \tanh(\beta \times (1-\eta))}, \tag{1.11}$$

Con el objetivo de visualizar estas transformaciones, observemos la Figura 1.6. En la Figura 1.6(a) se representa el diseño que representa una parametrización $\boldsymbol{P}$. A su derecha se encuentra la Figura 1.6(b), aquí se aplicó el filtro de densidad utilizando un radio de curvatura $r_f = 80nm$. Como podemos observar, este diseño posee regiones grises, mas ha logrado eliminar ciertas regiones punteagudas. Seguidamente, en las Figuras 1.6(c), 1.6(d), 1.6(e) se muestra el diseño de la Figura 1.6(b) tras aplicar la proyección descrita con $\beta = 2^6$ y distintos valores de η. Al utilizar $\eta = \eta_i = 0.5$ (Figura 1.6(d))

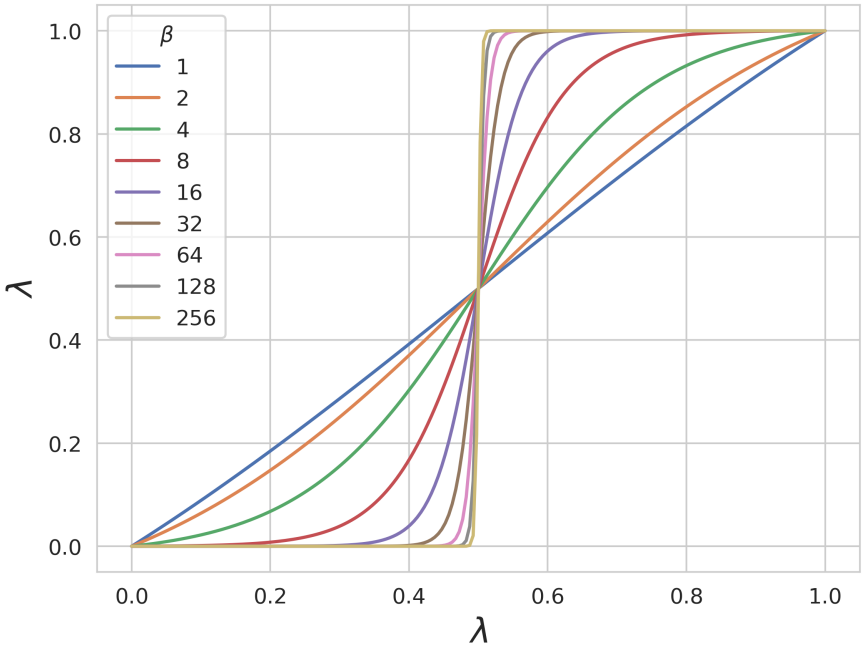


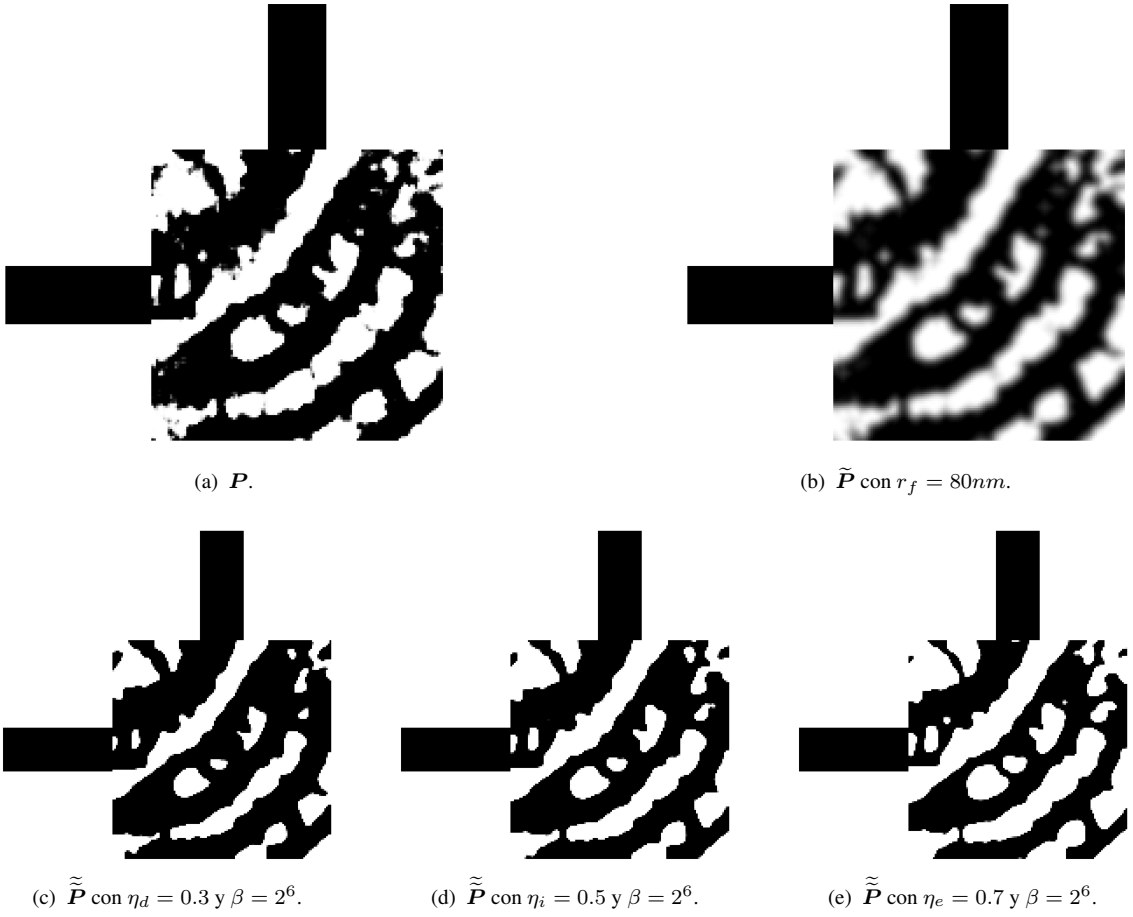
FIGURA 1.5: Función de discretización con $\eta = 0.5$ y distintos valores de β.

estamos simplemente discretizando el diseño. Por otro lado, al usar $\eta = \eta_d = 0.3$ (Figura 1.6(d)) estamos simulando que el diseño se ha dilatado (i.e. la región de Si del diseño intenta expandirse). Finalmente, al usar $\eta = \eta_e = 0.7$ (Figura 1.6(e)) estamos simulando que el diseño ha erosionado (i.e. la región de SiO_2 del diseño intenta expandirse).

Particularme, suele convenir $\eta_d = 1 - \eta_e$. Además, notemos que estos parámetros deben seleccionarse con cuidado; caso contrario el diseño podría simular una dilatación o erosión muy exageradas.

$$\frac{\partial f_{obj}}{\partial \boldsymbol{P}(x,y)} = \sum_{(x',y') \in B_{r_f}(x,y)} \frac{\partial f_{obj}}{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x',y')} \frac{\partial \widetilde{\widetilde{\boldsymbol{P}}}(x',y')}{\partial \widetilde{\boldsymbol{P}}(x',y')} \frac{\partial \widetilde{\boldsymbol{P}}(x',y')}{\partial \boldsymbol{P}(x,y)} \tag{1.12}$$

Una discusión más detallada de estas transformaciones se puede encontrar en Lazarov *et al.* (2016).



(a) $\boldsymbol{P}$. (b) $\widetilde{\boldsymbol{P}}$ con $r_f = 80nm$.

(c) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_d = 0.3$ y $\beta = 2^6$. (d) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_i = 0.5$ y $\beta = 2^6$. (e) $\widetilde{\widetilde{\boldsymbol{P}}}$ con $\eta_e = 0.7$ y $\beta = 2^6$.

FIGURA 1.6: Aplicación del filtro de densidad y proyección a una parametrización $\boldsymbol{P}$.

1.6 Algoritmos de Optimización

Finalmente, con todo lo descrito en este capítulo, ya podemos optimizar nuestros dispositivos. Ahora, notemos que nuestro problema se ha reducido a maximizar f_{obj}. Sin pérdida de generalidad, en la presente sección se describen algoritmos para minimizar la función $f = -f_{obj}$. Esto es posible ya que maximizar una función es equivalente a minimizar su negativo (Kochenderfer y Wheeler, 2019).

Para minimizar la función f, sea $\mathscr{P}$ el conjunto de todas las matrices $\boldsymbol{P} : [1, n] \times [1, m] \to [0, 1]$, queremos encontrar algún mínimo global $\boldsymbol{P}^* \in \mathscr{P}$, donde este elemento

se define como una matriz que satisface que $f(\boldsymbol{P}^*) \leq f(\boldsymbol{P}), \quad \forall \boldsymbol{P} \in \mathscr{P}$. Dicho de otra manera, nuestro problema es encontrar $\boldsymbol{P}^*$; sin embargo, esto es una tarea computacionalmente dificil de resolver (Angeris *et al.*, 2021).

Un problema más viable es encontrar $\sigma > 0 \wedge \boldsymbol{P}^+ \in \mathscr{P}$ tal que $f(\boldsymbol{P}^+) \leq f(\boldsymbol{P}), \quad \forall \boldsymbol{P} \in \mathscr{P} : ||\boldsymbol{P} - \boldsymbol{P}^+|| < \sigma$. Es decir, es más práctico encontrar $\boldsymbol{P}^+$ (conocido como mínimo local) el cual es un elemento que es un mínimo global si restringimos el espacio de búsqueda a cierta región alrededor de este elemento.

Debemos resaltar que ninguno de los algoritmos de esta sección puede asegurar encontrar el mínimo global; sin embargo, sus estrategias de búsqueda suelen permitir encontrar óptimos locales adecuados (Angeris *et al.*, 2021, Schneider *et al.*, 2019). Además, todos estos algoritmos se aseguran de mantener los valores de $\boldsymbol{P}$ en el intérvalo $[0, 1]$ aplicando $\boldsymbol{P}(x, y) \leftarrow max(0, min(1, \boldsymbol{P}(x, y)))$, mas este detalle se obvia en las explicaciones por simplicidad en la notación.

1.6.1 *Genetic Algorithms* (GA)

Como se describe en el Algoritmo 1 (Kochenderfer y Wheeler, 2019), la idea es comenzar generando una población (*population*) de tamaño $population\_size$. Esta población es un conjunto de matrices de $n \times m$ generadas a partir de una parametrización inicial $\boldsymbol{P}$, línea 1. Los siguientes tres pasos se ejecutan por k iteraciones. Primero, se realiza un proceso de selección para obtener las mejores parametrizaciones (*parents*), línea 3. Segundo, los seleccionados se encargan de producir la nueva generación (*children*), línea 4. Tercero, la nueva generación muta obteniendo nuevas características, línea 5.

Como se observa en el algorithm 1, tenemos las siguientes funciones:

- $generate\_population()$: retorna $population\_size$ matrices $\boldsymbol{P'}$ tal que $|\boldsymbol{P'}(x, y) - \boldsymbol{P}(x, y)| \in U(-GA\_range, GA\_range), \quad \forall x \in [1, n] \wedge y \in [1, m]$.

Algorithm 1: *Genetic Algorithms* (GA)

Data: $\boldsymbol{P}, population\_size, GA\_range, n\_selected\_parents, prob\_mutation$
Result: $min(population)$

```
population = generate_population()
for t = 0; t < k; t++ do
    parents = select(population)
    children = crossover(parents)
    population = mutation(children)
```

- $select(population)$: retorna $n\_selected\_parents$ elementos de $population$ de acuerdo a la probabilidad $prob_i$ dada por la ecuación

$$prob_i = \frac{max(f) - f^{(i)}}{\sum_j max(f) - f^{(j)}}, \tag{1.13}$$

  donde $f^{(i)}$ representa el valor de f aplicado a la $i-$ésima parametrización de $population$ y $max(f)$ es el máximo valor de f al haber sido aplicado a todas las matrices de $population$.

- $crossover(parents)$: retorna $population\_size$ nuevas parametrizaciones. Cada nueva matriz $\boldsymbol{S}$ es la combinación de dos parametrizaciones aleatorias $\boldsymbol{P_1}$ y $\boldsymbol{P_2}$ seleccionados de $parents$. En esta combinación, para cada $x \in [1, n] \wedge y \in [1, m]$ se define con igual probabilidad que $\boldsymbol{S}(x, y) = \boldsymbol{P_1}(x, y)$ o $\boldsymbol{S}(x, y) = \boldsymbol{P_2}(x, y)$.

- $mutation(children)$: retorna $children$, pero a cada elemento de todas estas matrices le agrega un valor en $U(-GA\_range, GA\_range)$ con probabilidad $prob\_mutation$.

1.6.2 *Particle Swarm Optimization* (PSO)

Podemos pensar este algoritmo como un caso especial del Algoritmo 1 (Kochenderfer y Wheeler, 2019). La idea es visualizar el $i-$ésimo individuo como una partícula definida por su posición ($\boldsymbol{P^{(i)}}$), velocidad ($\boldsymbol{V}^{(i)}$) y la mejor posición encontrada por la

partícula ($\boldsymbol{P}_b^{(i)}$). Para nuestro problema la matriz de parametrización representa la posición y la velocidad es una matriz que guía la exploración a otras parametrizaciones.

La idea principal del algoritmo es que cada particula acumula velocidad en una dirección favorable dada por: (i) la mejor posición encontrada hasta el momento por esta partícula y (ii) la mejor posición encontrada por la población completa. Como consecuencia, los individuos se pueden mover independientemente de perturbaciones locales. Adicionalmente, agregando caminos aleatorios los individuos incorporan comportamientos impredecibles que puede permitirles encontrar potenciales mejores elementos. Esta idea se sintetiza en el Algoritmo 2.

Algorithm 2: *Particle Swarm Optimization* (PSO)

Data: $\boldsymbol{P}, population\_size, PSO\_range, \omega, c_1, c_2$
Result: $\boldsymbol{P_b}$

```
population = generate_population()
for t = 0; t < k; t++ do
    P_b = select(population)
    population = mutation(population, P_b)
```

Como se observa en el algorithm 2, tenemos las siguientes funciones:

- $generate\_population()$: retorna $population\_size$ matrices $\boldsymbol{P'}$ tal que $|\boldsymbol{P'}(x,y) - \boldsymbol{P}(x,y)| \in U(-PSO\_range, PSO\_range), \quad \forall x \in [1,n] \wedge y \in [1,m]$. Además genera las matrices V con valores en $U(0,1)$.

- $select(population)$: retorna el individuo con el menor valor de f encontrado hasta el momento.

- $mutation(population, \boldsymbol{P_b})$: Aplica las siguientes transformaciones a *population*:

$$\boldsymbol{P}^{(i)} \leftarrow \boldsymbol{P}^{(i)} + \boldsymbol{V}^{(i)}, \tag{1.14}$$

$$\boldsymbol{V}^{(i)} \leftarrow \omega \boldsymbol{V}^{(i)} + c_1 r_1 \left(\boldsymbol{P}_{\boldsymbol{b}}^{(i)} - \boldsymbol{P}^{(i)} \right) + c_2 r_2 \left(\boldsymbol{P}_{\boldsymbol{b}} - \boldsymbol{P}^{(i)} \right), \tag{1.15}$$

donde $\boldsymbol{P_b}$ es la mejor posición encontrada globalmente, ω representa la tendencia de la particula de conservar su velocidad actual, c_1 y c_2 cuantifica la atracción relativa con $\boldsymbol{P}_{\boldsymbol{b}}^{(i)}$ y $\boldsymbol{P_b}$ respectivamente, y $r_1, r_2 \in U(0, 1)$ representan el comportamiento impredecible.

1.6.3 *Covariance Matrix Adapatation Evolution Strategy* (CMA-ES)

La idea general de esta estrategia evolutiva, mostrada en el Algoritmo 3, es mantener: (i) un vector $\boldsymbol{\mu}$ p-dimensional, (ii) una matriz $\boldsymbol{C}$ y (iii) un número σ para ir generando n inviduos p-dimensionales a partir una distribución $\mathcal{N}(\mu, \sigma^2 \boldsymbol{C})$.

Tomar puntos de esta distribución limita el espacio de búsqueda a una hiperelipse. Así, el algoritmo evalua puntos en esta región limitada. Luego, usando los valores obtenidos, se puede decidir entre: (i) mover la hiperelipse a otra región del espacio de búsqueda (ii) expandir or reducir la región cubierta por la distribución. El algoritmo de CMA-ES trabaja iterativamente sobre esta idea hasta que la hiperelipse termina casi degenerándose en un punto, potencialmente un óptimo.

Algorithm 3: CMA-ES

Data: $\boldsymbol{P}, population\_size, \sigma$
Result: $\boldsymbol{\mu}$

```
1 μ = flatten(P)
2 for t = 0; t < k; t++ do
3     sample() // Obtener population_size puntos de N(μ, σ²C)
4     update() // Ecuación 1.16
5     control() // Ecuación 1.17
6     adapt() // Ecuación 1.19
```

Entrando en más detalles del Algoritmo 3, en la línea 1 simplemente se linealiza la matriz $\boldsymbol{P}$ en un vector $\boldsymbol{\mu}$ de dimensión $p = n \times m$ (i.e. se unen sus vectores filas en

orden para formar un solo vector). Luego, dentro del bucle en la iteración t, en la línea 3 se genera $population\_size$ puntos p-dimensionales $\boldsymbol{x_i}$ de la distribución $\mathcal{N}(\boldsymbol{\mu},\ \sigma^2\boldsymbol{C})$, donde estos son ordenados ascendentemente de acuerdo al valor de f. En la línea 4 actualizamos la media $\boldsymbol{\mu}$ usando una promedio ponderado dado por

$$\boldsymbol{\mu}^{(\boldsymbol{t+1})} \leftarrow \sum_{i=1}^{n} w_i \boldsymbol{x_i}, \tag{1.16}$$

donde w_i son valores fijos y escogidos de tal manera que proporcionen mayor contribución a los puntos con menor valor al evaluarlos en f. Esto permite mover la media $\boldsymbol{\mu}$ en una dirección favorable.

Seguidamente, se necesita actualizar σ para expandir o reducir la hiperelipse en la siguiente iteración. Por este motivo, la línea 5 controla este valor mediante las ecuaciones

$$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{c_\sigma}{d_\sigma}\underbrace{\left(\frac{||\boldsymbol{p_\sigma}||}{\mathbb{E}||\mathcal{N}(0,\mathbf{I})||} - 1\right)}_{\text{evolution path comparison}}\right), \tag{1.17}$$

$$\mathbb{E}||\mathcal{N}(0,\mathbf{I})|| = \sqrt{2}\left(\frac{\Gamma\left(\frac{p+1}{2}\right)}{\Gamma\left(\frac{p}{2}\right)}\right), \tag{1.18}$$

donde $\boldsymbol{p_\sigma}$ es una variable que acumula los pasos llevados, $c_\sigma \in [0,1]$ es una variable que determina el tiempo acumulado para p_σ y $d_\sigma \approx 1$ es un parámetro que determina el ratio de posibilidad de cambio de $\sigma^{(t+1)}$. La principal parte de la Ecuación 1.17 es el término *evolution path comparison*, aquí se compara el tamaño de p_σ con su tamaño esperado bajo selección aleatoria. De esta comparación podemos controlar si el valor de σ debe incrementarse, disminuirse o permanecer igual.

Finalmente, en la línea 6 cambiamos $\boldsymbol{C}$ a una dirección favorable usando

$$\boldsymbol{C}^{(\boldsymbol{t+1})} \leftarrow \overbrace{\left(1 - c_1 c_c (1 - h_\sigma)(2 - c_c) - c_1 - c_\mu\right)\boldsymbol{C}^{(\boldsymbol{t})}}^{\text{cumulative update}} + \underbrace{c_1 \boldsymbol{p_C} \boldsymbol{p_C}^T}_{\text{rank-one update}} + \underbrace{c_\mu \sum_{i=1}^{n} w_i' \boldsymbol{\delta}^{(\boldsymbol{i})} \left(\boldsymbol{\delta}^{(\boldsymbol{i})}\right)^T}_{\text{rank-}\mu\text{ update}}, \quad (1.19)$$

donde $c_\mu \leq 1$ es el radio de aprendizaje para el término *rank-μ update*, $c_1 \leq 1 - c_\mu$ es el radio de aprendizaje para el término *rank-one update*, $c_c \in [0, 1]$ es el radio de aprendizaje para el término *cumulative update*, h_σ es la evaluación bajo la función unitaria usado para actualizar apropiadamente el camino evolutivo, $\boldsymbol{p_C}$ es un vector acumulativo usado para actualizar la matriz de covarianza, w_i' son los coeficientes de ponderación modificados y $\boldsymbol{\delta}^{(\boldsymbol{i})}$ son las desviaciones seleccionadas.

En la Ecuación 1.19, el primer término (*cumulative update*) mantiene información de la anterior matriz de convarianza. El segundo término (*rank-one update*) permite expandir la distribución en una dirección favorable. El tercer término (*rank-μ update*) incrementa la búsqueda en espacios donde es probable encontrar buenas soluciones. La combinación de estos tres términos actualiza $\boldsymbol{C}$ de tal manera que mueva la hiperelipse en una dirección favorable. Para una descripción más detallada del algoritmo, revisar los trabajos de Hansen (2016) y Kochenderfer y Wheeler (2019).

1.6.4 *Gradient Descent* (GD)

Este algoritmo comienza desde un punto inicial, en nuestro caso la matriz $\boldsymbol{P}$. Luego, usa la derivada en ese punto para guiar su búsqueda iterativamente, esto lo realiza mediante la siguiente ecuación:

$$\boldsymbol{P}^{(t+1)} \leftarrow \boldsymbol{P}^{(t)} - \gamma \nabla \boldsymbol{P}^{(t)}, \tag{1.20}$$

donde t representa la iteración actual y γ (conocido como el ratio de aprendizaje) determina la magnitud de como actualizar $\boldsymbol{P}^{(t)}$. Particularmente, un enfoque razonable para buscar convergencia a un mínimo local es actualizar γ en cada iteración mediante la ecuación

$$\gamma^{(t+1)} = \frac{|(\boldsymbol{P}^{(t)} - \boldsymbol{P}^{(t-1)})^T (\nabla f(\boldsymbol{P}^{(t)}) - \nabla f(\boldsymbol{P}^{(t-1)}))|}{||\nabla f(\boldsymbol{P}^{(t)}) - \nabla f(\boldsymbol{P}^{(t-1)})||^2}. \tag{1.21}$$

Un detalle importante a señalar es que por simplicidad en la notación, la Ecuación 1.21 se describió de esta manera; sin embargo, en realidad debemos trabajar con la matriz $\boldsymbol{P}$ después de ser linealizada para que las operaciones de esta ecuación estén bien definidas. Mayores detalles del algoritmo se pueden encontrar en Demidova y Gorchakov (2020).

1.6.5 *Method of Moving asymptotes* (MMA)

Method of moving asymptotes (MMA) es un algoritmo de optimización local basado en la gradiente, es decir, de primer orden. La idea del algoritmo es comenzar en un punto $\boldsymbol{P}$ y formar una aproximación de f alrededor de $\boldsymbol{P}$ utilizando: (i) la gradiente de f, (ii) los límites de los valores de P y (iii) una función de penalidad cuadrática. Lo interesante de esta aproximación es que es convexa y separable, lo cual permite resolverla de forma sencilla. Así, el algoritmo comienza calculando la aproximación, luego resuelve la optimización utilizando la aproximación. De este modo, surgen dos posibles escenarios: (i) la aproximación encuentra un buen elemento que se usa como punto de inicio para repetir el procedimiento o (ii) se repite el proceso limitando la contribución de la función de penalidad.

En realidad en el presente trabajo usaremos una versión mejorada del algoritmo conocida como *Conservative Convex Separable Approximation* (CCSA) debido a su popularidad en optimización topológica, pero nos seguiremos refiriendo al algoritmo como MMA por convención. Esta variante tiene la particularidad de lograr converger a un mínimo local independientemente del punto inicial. Los detalles se pueden revisar en Svanberg (2002).

1.6.6 L-BFGS-B

Limited-memory Broyden–Fletcher–Goldfarb–Shanno with boundaries (L-BFGS-B) es un algoritmo de optimización que se caracteriza por estimar la inversa de la matriz hessiana utilizando una cantidad de memoria de orden lineal. El algoritmo comienza con un punto inicial $\boldsymbol{P}$. Luego, internamente mantiene el historial de las últimas evaluaciones de la función f y ∇f para determinar una dirección para actualizar $\boldsymbol{P}$. Esto es posible precondicionando la gradiente con información de curvatura obtenida por la aproximación de la matriz hessiana. El lector interesado puede revisar los detalles del algoritmo en Liu y Nocedal (1989).

TODO: Resumen del capítulo

REFERENCIAS BIBLIOGRÁFICAS

Angeris, G., Vučković, J., and Boyd, S. (2021). Heuristic methods and performance bounds for photonic design. *Opt. Express*, 29(2):2827–2854.

Christiansen, R. E. and Sigmund, O. (2021). Inverse design in photonics by topology optimization: tutorial. *J. Opt. Soc. Am. B*, 38(2):496–509.

Demidova, L. A. and Gorchakov, A. V. (2020). Research and study of the hybrid algorithms based on the collective behavior of fish schools and classical optimization methods. *Algorithms*, 13(4).

Hansen, N. (2016). The CMA Evolution Strategy: A Tutorial.

Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for Optimization*. The MIT Press.

Lazarov, B. S., Wang, F., and Sigmund, O. (2016). Length scale and manufacturability in density-based topology optimization. *Archive of Applied Mechanics*, 86(1-2):189–218.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528.

Lukas Chrostowski (2010). *Silicon Photonics Design: From Device to System*.

Molesky, S., Lin, Z., Piggott, A. Y., Jin, W., Vucković, J., and Rodriguez, A. W. (2018). Inverse design in nanophotonics. *Nature Photonics*, 12(11):659–670.

Oskooi, A. F., Roundy, D., Ibanescu, M., Bermel, P., Joannopoulos, J. D., and Johnson, S. G. (2010). Meep: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications*, 181(3):687–702.

Schneider, P. I., Garcia Santiago, X., Soltwisch, V., Hammerschmidt, M., Burger, S., and Rockstuhl, C. (2019). Benchmarking Five Global Optimization Approaches for Nano-optical Shape Optimization and Parameter Reconstruction. *ACS Photonics*, 6(11):2726–2733.

Su, L., Vercruysse, D., Skarda, J., Sapra, N. V., Petykiewicz, J. A., and Vučković, J. (2020). Nanophotonic inverse design with SPINS: Software architecture and practical considerations. *Applied Physics Reviews*, 7(1).

Svanberg, K. (2002). A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, pages 555–573.