TESIS LICENCIATURA EN FÍSICA

CRITERIOS BÁSICOS PARA LA PRESENTACIÓN DE LA TESIS EN EL INSTITUTO BALSEIRO TANTO DE DOCTORADO COMO DE MAESTRÍA

J. Autor
Doctorando

Dr. J. Director
Director

Dr. J. Otro más
Co-director

Miembros del Jurado
Dr. J. J. Jurado (Instituto Balseiro)
Dr. Segundo Jurado (Universidad Nacional de Cuyo)
Dr. J. Otro Jurado (Univ. Nac. de LaCalle)
Dr. J. López Jurado (Univ. Nac. de Mar del Plata)
Dr. U. Amigo (Instituto Balseiro, Centro Atómico Bariloche)

26 de Diciembre de 2022

Colisiones Atómicas – Centro Atómico Bariloche

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

A mi familia
A mis amigos
A todos los que me conocen
A toda esa otra gente que no

Índice de símbolos

Índice de contenidos

Resumen

Este es el resumen en castellano.
La tesis debe reflejar el trabajo desarrollado, mostrando la metodología utilizada, los resultados obtenidos y las conclusiones que pueden inferirse de dichos resultados.

Palabras clave: FORMATO DE TESIS, LINEAMIENTOS DE ESCRITURA, INSTITUTO BALSEIRO

Abstract

This is the title in English:
The thesis must reflect the work of the student, including the chosen methodology, the results and the conclusions that those results allow us to draw.

Keywords: THESIS FORMAT, TEMPLATES, INSTITUTO BALSEIRO

Capítulo 1

Introducción. Ondas Gravitacionales

1.1. Motivación y Objetivo

1.2. Representación Ondas Gravitacionales

1.3. Datos utilizados

Se utilizaron ondas gravitacionales generadas a partir del modelo híbrido *NRHybSur3dq8*[1] de relativdad numérica y aproximaciones post Newtonianas para colisiones de agujeros negros binarios.

Cada onda h generada se representa por una serie temporal compleja de la forma:

$$h = h_+ + ih_\times$$

Recordando que h está parametrizada por λ

$$h = h(t;\lambda) = h_\lambda(t) = h_\lambda$$

En este caso λ tendrá 3 dimensiones, $(q, \chi_{1_Z}, \chi_{2_Z})$, y estará acotado de la siguiente manera:

- Relación entre masas q: $1 \leq q \leq 8$
- Espín del agujero negro más pesado (liviano) $\chi_{1_Z}(\chi_{1_Z})$: $|\chi_{1_Z}|, |\chi_{2_Z}| < 0{,}8$

Se representa un conjunto $\mathcal{K}$ de N muestras de λ de la siguiente forma:

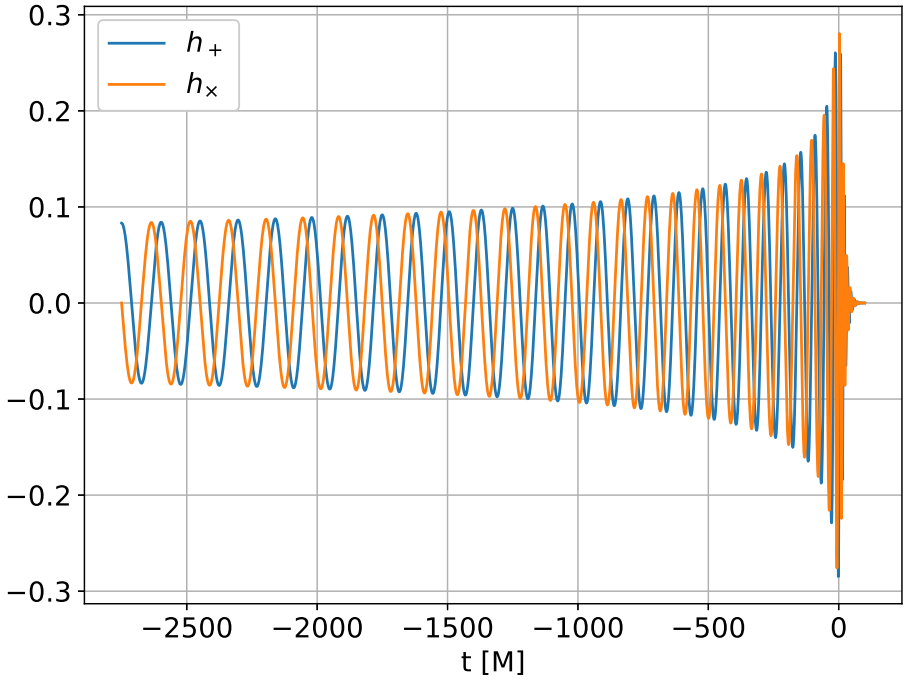

Figura 1.1: polarizaciones h_+ y $h_\times$ para $q = 3$, $\chi_{1z} = \chi_{2z} = 0$, en el modo $l = 2$, $m = 2$.

$$\mathcal{K} = \{h_{\lambda_i}\}, \quad i = 1, ..., N$$

y debido a que cada h_{λ_i} es una serie temporal, se puede representar $\mathcal{K}$ en forma de la matriz $H \in \mathbb{C}^{N \times L}$:

$$H = \begin{bmatrix} h_{\lambda_1} \\ h_{\lambda_2} \\ \vdots \\ h_{\lambda_N} \end{bmatrix} = \begin{bmatrix} h_{\lambda_1}(t_1) & h_{\lambda_1}(t_2) & \cdots & h_{\lambda_1}(t_L) \\ h_{\lambda_2}(t_1) & h_{\lambda_2}(t_2) & \cdots & h_{\lambda_2}(t_L) \\ \vdots & \vdots & \ddots & \vdots \\ h_{\lambda_N}(t_1) & h_{\lambda_N}(t_2) & \cdots & h_{\lambda_N}(t_L) \end{bmatrix}$$

Siendo L la longitud de la serie temporal. De forma que cada fila de H es una onda gravitacional.

Capítulo 2

Teoría de Aproximaciones: Bases Reducidas y Aprendizaje

2.1. Bases Reducidas

2.2. Bases Reducidas hp Greedy

El nombre del método hp greedy viene de la combinación del *"refinamiento p"* y del *"refinamiento h"*. El *refinamiento p* proviene de los métodos espectrales con bases polinomiales [2] y se refiere a la propiedad de que el error de representación disminuye al aumentar el grado del polinomio (en el caso de las bases reducidas aumenta el número de elementos en la base). Por otro lado el término de *refinamiento h* se toma prestado de los métodos de diferencias finitas, donde el tamaño de cada celda de la grilla es representado por h. En este caso el refinamiento ocurre en el espacio de los parámetros (y no en el dominio físico).

2.2.1. Refinamiento h

Partiendo de la siguiente notación:

- V: espacio de parámetros para un dado subdominio D.
- V_1, V_2 : particiones de V.
- Λ_V : parámetros *greedy* para V.
- $\hat{\Lambda}_V$: punto de anclaje para V.

El refinamiento en el dominio de los parámetros ocurre a partir de la división recursiva de cada subdominio V del dominio total D en dos subdominios V_1 y V_2. De forma que se obtiene una estructura de árbol binario.

Esta descomposición binaria del dominio está descrita en forma de pseudocódigo en el algoritmo 1.

Al algoritmo ingresan tres objetos:

- λ_V: conjunto de parámetros resultado de un muestreo de V.
- $\hat{\Lambda}_{V_1}, \hat{\Lambda}_{V_2}$: puntos de anclaje (son los primeros dos elementos de Λ_V).

Luego, para cada parámetro del conjunto λ_V se evalúa su distancia a los puntos de anclaje a partir de la *función de proximidad* $d : d(\lambda_1, \lambda_2)$:

$$d(\lambda_1, \lambda_2) = ||\lambda_1 - \lambda_2||_2,$$

de forma que se obtengan dos conjuntos; λ_{V_1} con los λ_i más proximos a $\hat{\Lambda}_{V_1}$, y λ_{V_2} con los λ_i más proximos a $\hat{\Lambda}_{V_2}$, tal que $\lambda_V = \lambda_{V_1} \cup \lambda_{V_2}$. Este resultado es la división del espacio de parámetros a partir de los puntos de anclaje.

Algoritmo 1 `Partition`$(\lambda_V, \hat{\Lambda}_{V_1}, \hat{\Lambda}_{V_2})$

Input: $\lambda_V, \hat{\Lambda}_{V_1}, \hat{\Lambda}_{V_2}$

$\lambda_{V_1} = \lambda_{V_2} = \emptyset$
for each $\lambda_i \in \lambda_V$ **do**
  if $d(\lambda_i, \hat{\Lambda}_{V_1}) < d(\lambda_i, \hat{\Lambda}_{V_2})$ **then**
    $\lambda_{V_1} = \lambda_{V_1} \cup \lambda_i$
  else if $d(\lambda_i, \hat{\Lambda}_{V_1}) > d(\lambda_i, \hat{\Lambda}_{V_2})$ **then**
    $\lambda_{V_2} = \lambda_{V_2} \cup \lambda_i$
  else
    $\lambda_{V'} = \text{random choice}([\lambda_{V_1}, \lambda_{V_2}])$
    $\lambda_{V'} = \lambda_{V'} \cup \lambda_i$
  end if
end for

Output: $\lambda_{V_1}, \lambda_{V_2}$

2.2.2. Refinamiento hp-greedy

El refinamiento hp-greedy es un método que combina el algoritmo greedy para la construcción de bases reducidas con la partición del dominio de parámetros.

Esta partición recursiva del dominio de parámetros da lugar a una estructura de árbol binario, la cual tendrá diferentes niveles l de profundidad, con un l_{max} establecido por el usuario, de forma que $l : 0 \leq l \leq l_{max}$, donde $l = 0$ es el nodo raíz. Cada nodo del árbol estará etiquetado por un conjunto de índices B_l, que parte de:

$$B_0 = (0,),$$

luego sus dos hijos ($l = 1$) tendrán las etiquetas:

$$B_1 = (0, 0,) \;\; o \;\; (0, 1,),$$

y en general:

$$B_l = (0, i_1, \cdots, i_l), \quad con \;\; i_j = \{0, 1\},$$

donde cada nivel l tendrá un máximo de 2^l nodos. Los nodos que no tengan hijos se llamarán nodos *hojas*.

El método está explicado en el algoritmo 2; partiendo de un dado dominio de parámetros V se construye una base reducida a partir de un conjunto de entrenamiento $\mathcal{T}_V = \{h_{\lambda_{V_i}}\}_{i=1}^N$, una *tolerancia greedy* ε y un n_{max} (para esto se utiliza el algoritmo [**VERIFICAR REFERENCIA**]). Si el error de representación σ es mayor que la tolerancia ε, y si la profundidad del nivel l es menor a l_{max}, entonces se realizará una partición del dominio V utilizando como puntos de anclaje a los dos primeros parámetros *greedy*. En cada dominio se realizará el mismo procedimiento hasta que se

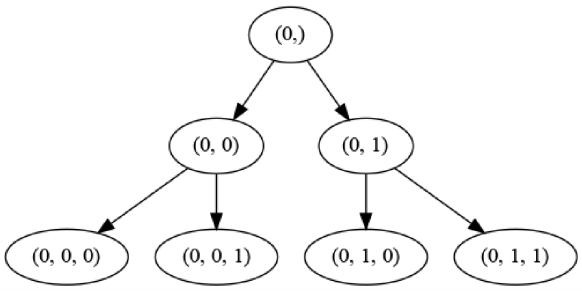

Figura 2.1: Representación de los nodos de un árbol con $l_{max} = 2$.

cumpla que $l = l_{max}$ o hasta que $\sigma \leq \varepsilon$.

Algoritmo 2 `hpGreedy`$(\mathcal{T}_V, \lambda_V, \varepsilon, n_{max}, l, l_{max}, B_l)$

Input: $\mathcal{T}_V, \lambda_V, \varepsilon, n_{max}, l, l_{max}, B_l$

rb, Λ_V, σ = `GreedyRB`$(\mathcal{T}_V, \lambda_V, \varepsilon, n_{max})$
if $\sigma > \varepsilon$ **and** $l < l_{max}$ **then**
    $\hat{\Lambda}_{V_1} = \Lambda_V[1]$
    $\hat{\Lambda}_{V_2} = \Lambda_V[2]$
    $\lambda_{V_1}, \lambda_{V_2}$ = `Partition`$(\lambda_V, \hat{\Lambda}_{V_1}, \hat{\Lambda}_{V_2})$
    out_1 = `hpGreedy`$(\mathcal{T}_{V_1}, \lambda_{V_1}, \varepsilon, n_{max}, l+1, l_{max}, (B_l, 0))$
    out_2 = `hpGreedy`$(\mathcal{T}_{V_2}, \lambda_{V_2}, \varepsilon, n_{max}, l+1, l_{max}, (B_l, 1))$
    $out = out_1 \cup out_2$
else
    $out = \{(rb, \Lambda_V, B_l)\}$
end if

Output: out

El resultado del algoritmo 2 es una estructura arbórea, donde cada nodo contiene la información de sus puntos de anclaje, por lo que en el caso de querer proyectar un conjunto de validación, cada onda gravitacional se proyectará a la base reducida del nodo hoja con el punto de anclaje más cercano al parámetro de la onda.

2.2.3. Aplicación a Ondas Gravitacionales

Se trabaja a partir de un conjunto de ondas gravitacionales con parámetro bidimensional, donde $\chi_{1_z} = \chi_{2_z} = \chi_z$, es decir que $\lambda = (q, \chi_z)$. De esta forma se puede graficar fácilmente el dominio de parámetros.

En la figura 2.2 se puede observar una representación de la partición del dominio de parámetros. En la primera imagen se pueden ver los dos puntos de anclaje, que son los primeros dos elementos de la base global construida inicialmente. En cada nueva división se construye una nueva base global con la cual se realiza la siguiente partición de cada subdominio.

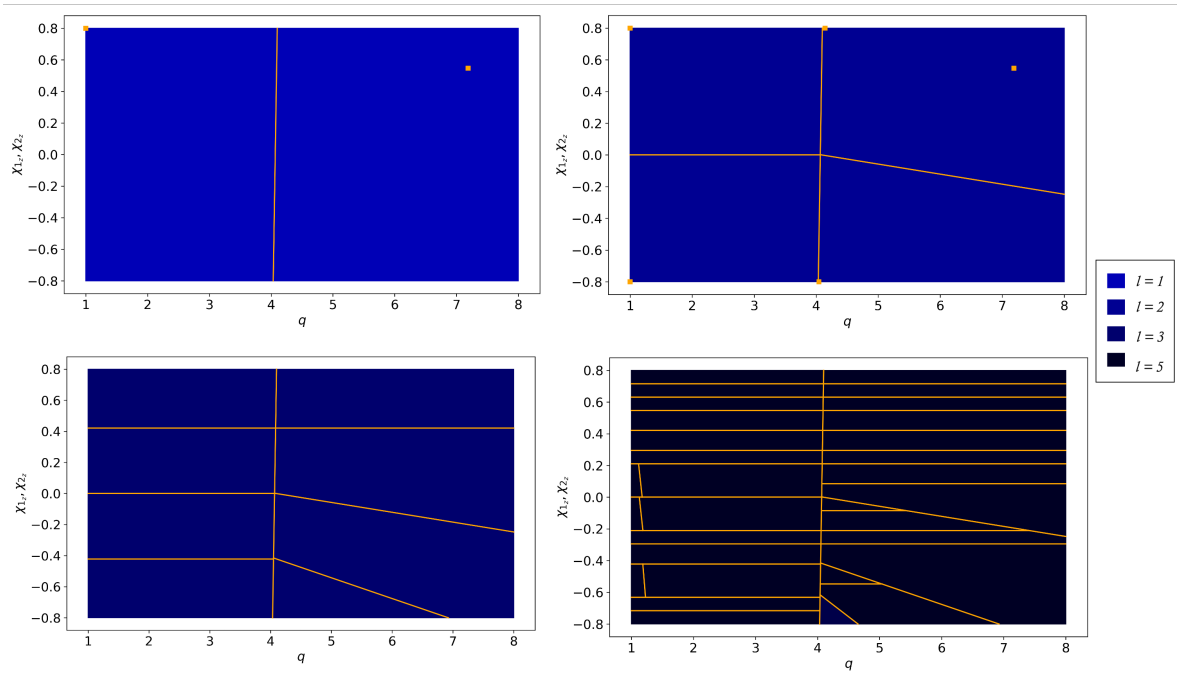

Figura 2.2: Ejemplo de partición del espacio de parámetros bidimensional para $l_{max} = 1, 2, 3$ y 5. En los primeros dos casos se muestran los puntos de anclaje.

En la figure 2.3 se compara el máximo error de representación obtenido para un conjunto de validación con una base global, es decir, con $l_{max} = 0$, y con $l_{max} = 4$. La velocidad de convergencia es claramente mayor en el segundo caso.

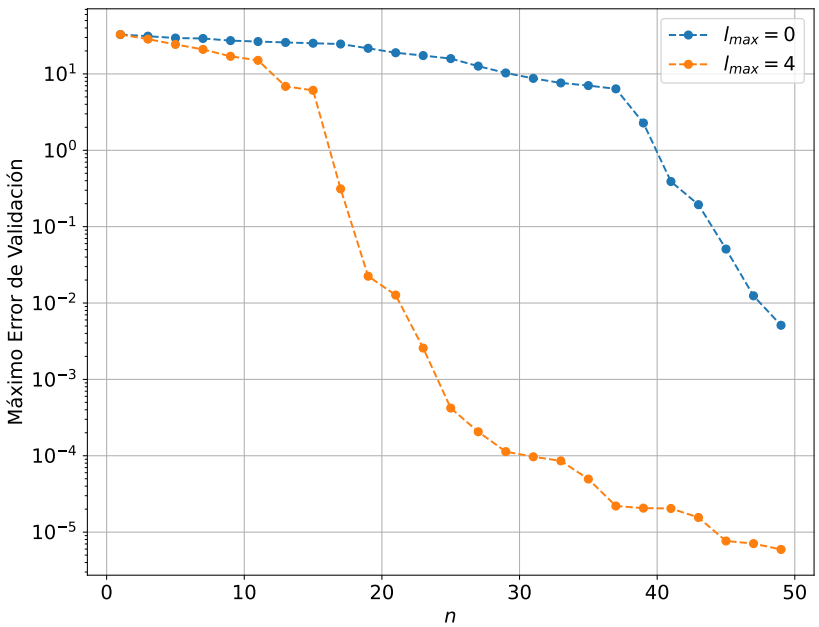

Figura 2.3: Base global ($l_{max} = 0$) versus base con $l_{max} = 4$ para distintos valores de n.

El aspecto más importante de este método es que permite disminuir la complejidad

temporal del algoritmo a la hora de proyectar la base, a cambio de aumentar la complejidad espacial, pues si bien cada subdominio tendrá un máximo de n_{max} elementos en su base, habrá un máximo de $2^{l_{max}}$ subdominios.

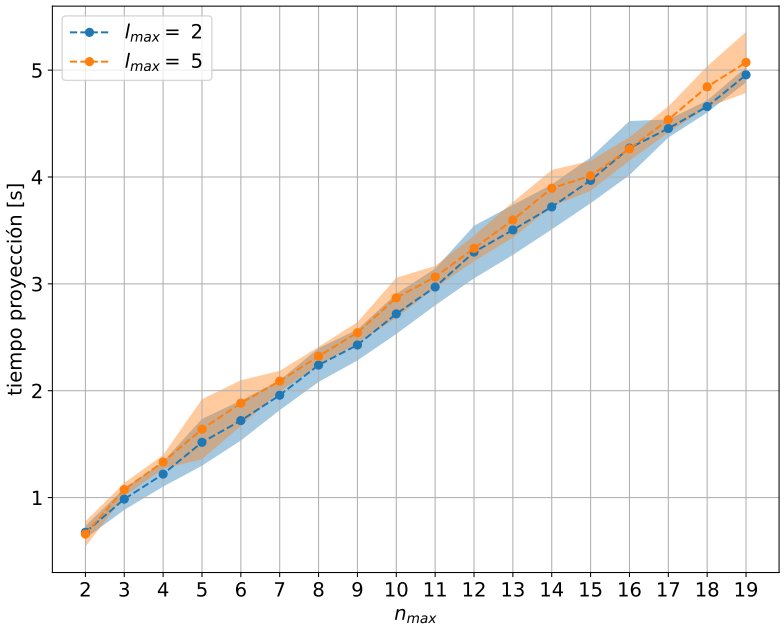

Figura 2.4: tiempos de proyección de un conjunto de validación a dos bases con distinto l_{max} en función del n_{max}. En cada caso la linea de trazo representa el valor medio, y el área de color indica una desviación estandar desde el valor medio, para cada medición.

En la figura 2.4 se graficó el tiempo de proyección de un conjunto de validación a dos bases *hp-greedy* con distinto valor de l_{max}. Se observa que el tiempo es bastante lineal en relación al n (elementos de las bases locales), y no parece ser afectado por l_{max}.

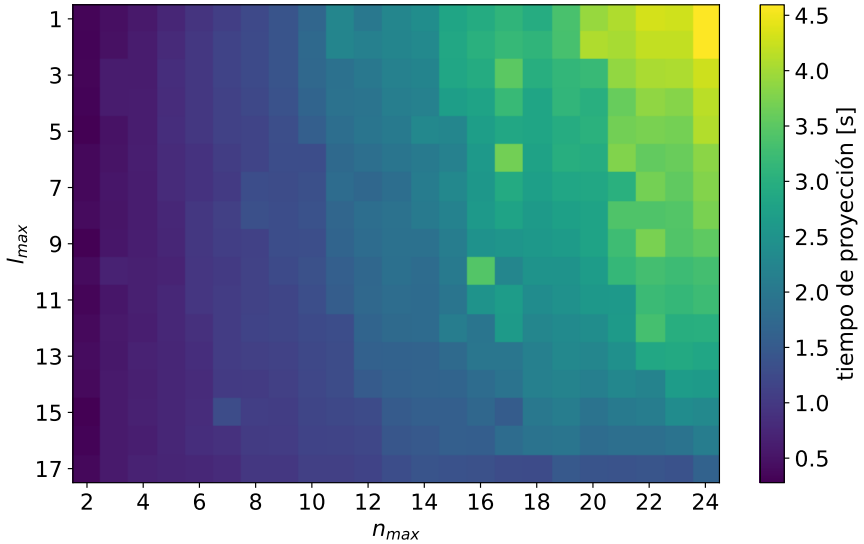

Figura 2.5: Tiempo de proyección de un conjunto de validación para diferentes valores de n_{max} y l_{max}

En la figura 2.5 se puede ver el tiempo de proyección para más valores de n_{max} y l_max. En los primeros valores de l_{max} se observa un comportamiento similar al descrito anteriormente, donde el tiempo depende casi únicamente de n_{max}. Sin embargo al aumentar el l_{max} se observa que el tiempo disminuye. Esto se puede entender en dos partes:

- **Independencia aparente entre el tiempo de proyección y** l_{max}: para realizar la proyección de cada onda del conjunto de validación en la base *hp-greedy* primero se debe buscar el subdominio (la hoja) correspondiente utilizando los puntos de anclaje de la estructura arbórea de la base. Luego se proyectará la onda en la base local del subdominio en cuestión. Si bien la búsqueda en el árbol tiene una complejidad temporal $O(l_{max})$, el trabajo de cómputo más importante es el que se realizará al momento de proyectar la base, que es independiente de l_{max}, con una complejidad temporal $O(n_{max})$. Pero esto solo se cumple hasta ciertos valores de l_{max}.

- **Disminución del tiempo de proyección al aumentar** l_{max}: ya se mencionó en más de una ocasión que por cada nivel l hay un máximo de 2^l subdomínios. Es decir que si se quiere obtener el número de elementos de todas las bases en las hojas del árbol, suponiendo un árbol denso, este número será $n_{max} \times 2^{l_{max}}$. En la figura 2.5 se utilizó un conjunto de entrenamiento con 1400 ondas, por lo que al llegar a unos valores de $l_{max} = 6$ y $n_{max} = 24$ en total debería haber 1536 elementos de base en total. Es decir, más elementos de base que ondas en el conjunto de entrenamiento. Por lo tanto al aumentar el l_{max} rápidamente se aumenta el número de subdominios, reduciendo su tamaño como resultado y reduciendo el número de elementos de las bases locales (cada subdominio tendrá una cantidad de elementos menor a n_{max}). De esta forma se explica la disminución del tiempo de proyección para valores grandes de l_{max}, consecuencia del tamaño limitado del conjunto de entrenamiento.

2.2.4. Hiperparámetros

Al momento de construir una base *hp-greedy* entran en juego cuatro hiperparámetros. Los primeros tres son los parámetros de parada;

- $\boldsymbol{n_{max}}$: determina la cantidad máxima de elementos para cada base local. A mayor cantidad de elementos el error de representación será menor, pero el tiempo requerido para proyectar un conjunto de validación a la base depende casi exclusivamente de este hiperparámetro.

- $\boldsymbol{l_{max}}$: determina la máxima profundidad de las hojas del árbol. En general al aumentar l_{max} disminuye el error de representación, pero valores muy elevados

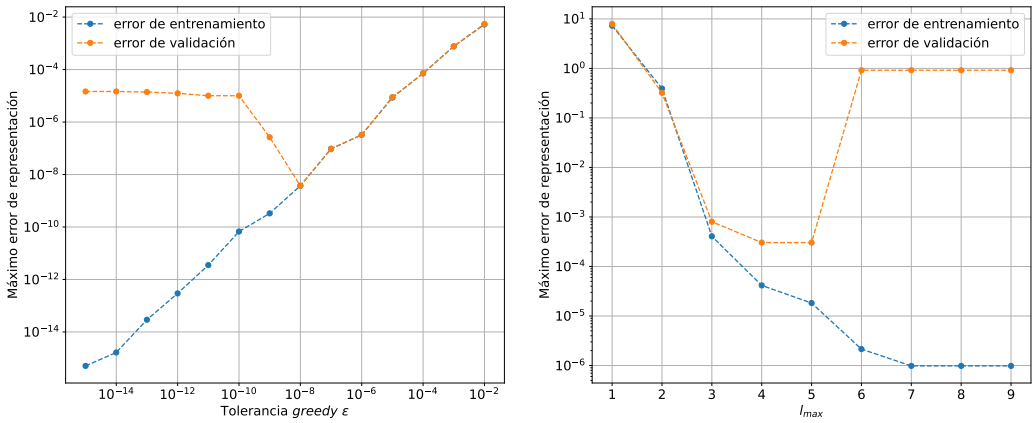

Figura 2.6: Ejemplos de *sobreajuste*. A la izquierda variando ε con $(n_{max}, l_{max}) = (25, 19)$ en un conjunto de parámetro unidimensional $(\lambda_i = q_i)$. A la derecha variando l_{max} con $(n_{max}, \varepsilon) = (20, 1 \times 10^{-6})$ en un conjunto de parámetro bidimensional$(\lambda_{ij} = (q_i, \chi_{z_j}))$.

junto a cierta combinación de hiperparámetros pueden dar lugar a sobreajustes en el modelo, un ejemplo de esto se puede ver en la figura 2.6. Este es un comportamiento típico de las estructuras arbóreas.

- $\boldsymbol{\varepsilon}$: la tolerancia *greedy* interviene tanto en el tamaño de las bases locales como en la profundidad de las hojas del árbol. Un valor de ε demasiado bajo también puede dar lugar a sobreajuste, sobre todo con valores muy altos de l_{max}. Un valor de $\varepsilon = 0$ implica que se obtiene un árbol totalmente denso, determinado únicamente por n_{max} y l_{max}, y al aumentar el valor de ε se puede pensar en la analogía de podar un árbol, de forma que se previene el sobreajuste.

Al cuarto hiperparámetro se le da el nombre de ***semilla*** y se la denota con $\hat{\mathbf{\Lambda}}_\mathbf{0}$;

- $\hat{\mathbf{\Lambda}}_\mathbf{0}$: la semnilla no es más que el primer parámetro *greedy* de la base global. En cada base local, el primer parámetro *greedy* no es relevante, pero en el caso de las bases *hp-greedy* cada semilla dará lugar a una división diferente del dominio de parámetros. En la figura 2.7 se puede ver como cuatro semillas diferentes dan lugar a cuatro curvas de error con distinta convergencia. En la figura 2.8, por otro lado, se observa el resultado de la partición del dominio para tres semillas diferentes. En general las semillas que mejor funcionan (con el conjunto de datos utilizado) son las que logran una partición regular del dominio de parámetros.

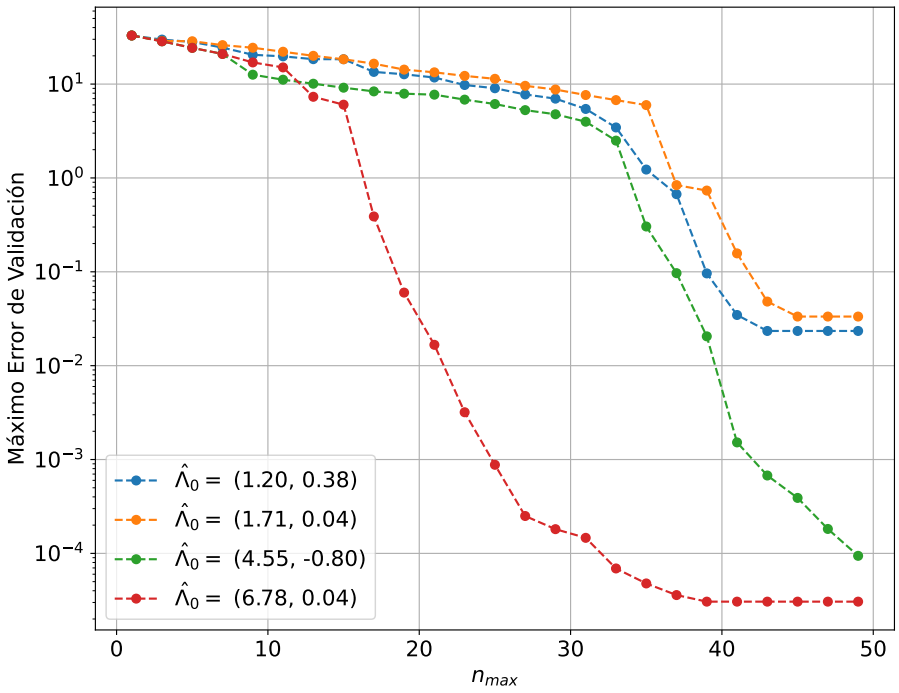

Figura 2.7: Error de validación para diferentes semillas.

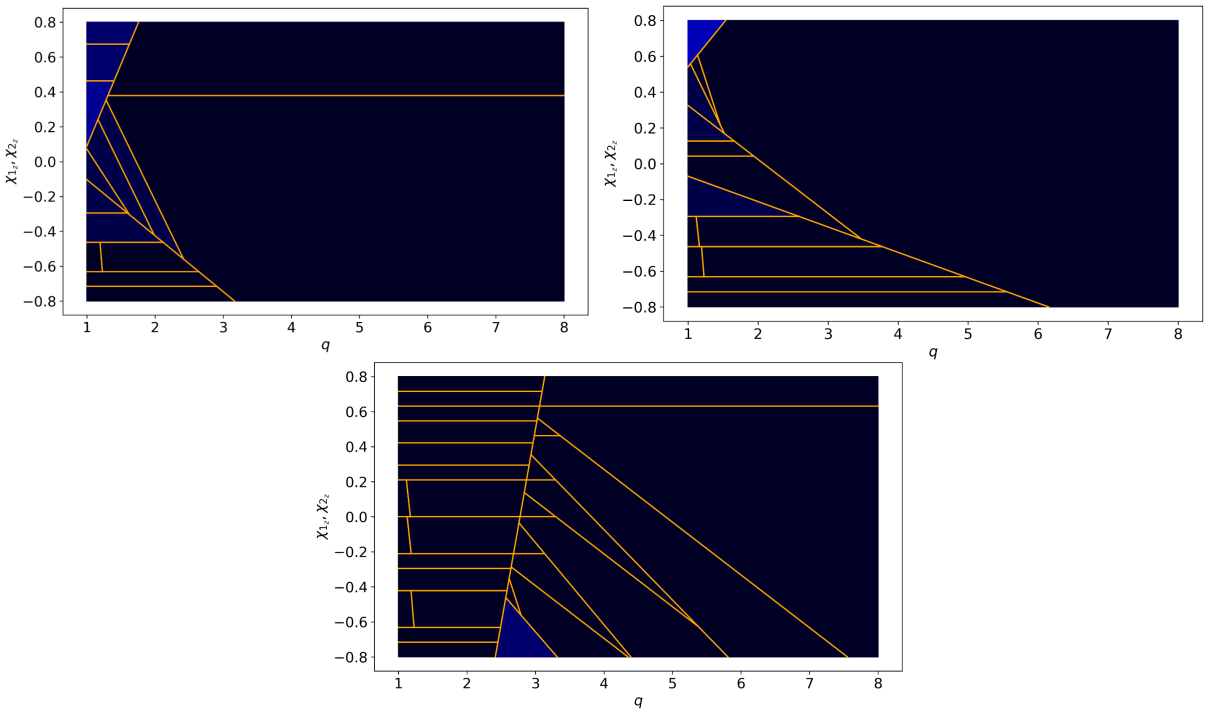

Figura 2.8: Partición del espacio de parámetros para tres semillas diferentes; a la izquierda $\hat{\Lambda}_0 = (1{,}2 \;\; 0{,}38)$, a la derecha $\hat{\Lambda}_0 = (1{,}71 \;\; 0{,}04)$ y al centro $\hat{\Lambda}_0 = (4{,}55 \;\; -0{,}8)$.

Capítulo 3

Optimización de Hiperparámetros

3.1. Planteo del Problema

Sea $f : X \to \mathbb{R}$ una función que devuelve el máximo error de validación de un modelo entrenado a partir de una combinación de hiperparámetros $\mathbf{x} \in X$, se desea encontrar $\hat{\mathbf{x}}$:

$$\mathbf{x}^* = arg\min_{\mathbf{x} \in X} f(\mathbf{x})$$

Es decir, se busca encontrar la combinación óptima de hiperparámetros dentro de un dominio X para obtener el mínimo error de representación en un dado conjunto de validación. En el caso de la construcción de una base *hp-greedy* óptima:

$$\mathbf{x} = (n_{max}, l_{max}, \varepsilon, \hat{\Lambda}_0).$$

El problema al momento de realizar esta optimización es que la función f no tiene una expresión analítica, sino es que es el resultado de entrenar el modelo y evaluar el error de representación con un conjunto de validación, lo que la hace costosa de evaluar (computacionalmente hablando). Este capítulo se centrará principalmente en la **optimización Bayesiana** [3, 4], un método que intenta reducir al mínimo el número de evaluaciones de f para encontrar $\hat{\mathbf{x}}$ y se puede colocar dentro de una categoría llamada optimización secuencial basada en modelos, o **SMBO**[5, 6] (*Secuential Model-Based Optimization*).

Además existen dos métodos muy utilizados que no utilizan modelos, los cuales son la **busqueda exaustiva** (o *grid search*) y la **búsqueda aleatoria**. Estos métodos se utilizaron en casos sencillos de optimización para realizar una comparación con la optimización bayesiana.

Comentario sobre el dominio X

Si bien la tolerancia *greedy* ε puede tomar cualquier valor real no nulo (a diferencia de n_{max}, l_{max} y $\hat{\Lambda}_0$ que toman valores discretos), para simplificar la búsqueda de $\hat{\mathbf{x}}$ se utilizaron siempre distribuciones discretas en el espacio logarítmico. Más específicamente se utilizaron conjuntos de la forma $C = \{1 \times 10^t \mid a \leq t \leq b, t \in \mathbb{Z}\}$. De esta forma X será un conjunto finito y estará definido por los valores extremos de cada hiperparámetro.

3.2. Optimización Bayesiana

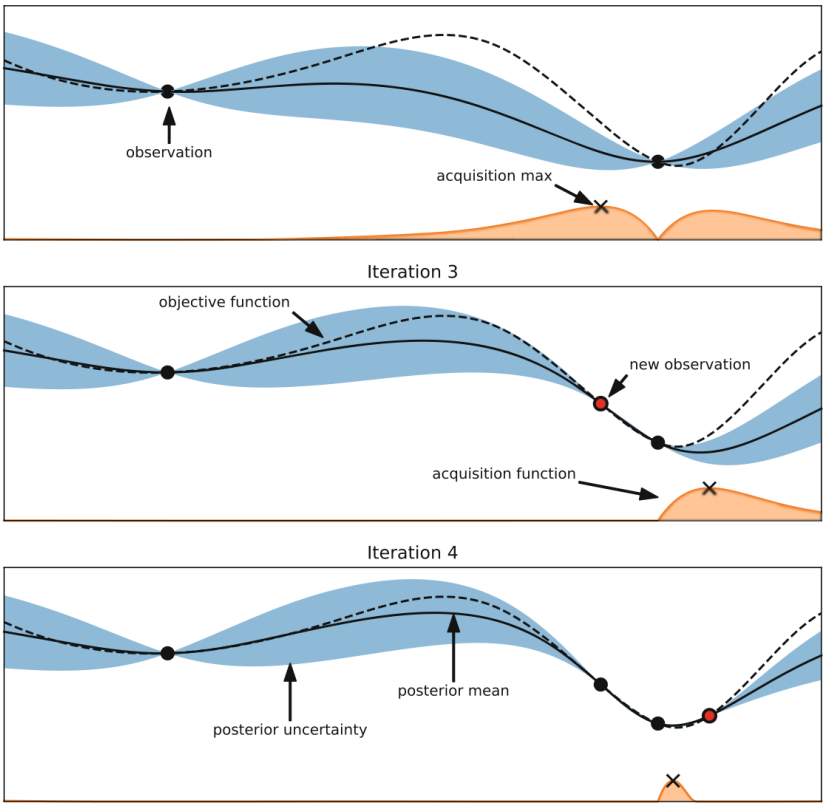

Figura 3.1: En la figura se observan tres iteraciones de una optimización bayesiana para una función sencilla con parámetro unidimensional. En linea punteada está representada la función real, mientras que con linea gruesa se representa el valor medio del modelo estadístico (en este caso construido utilizando procesos gaussianos). El área pintada en azul representa la incertidumbre del modelo, que tiende a cero en los puntos que representan las observaciones realizadas. Debajo se puede ver una función de adquisición en color naranja, que indica el siguiente punto a evaluar [7].

La optimización bayesiana es un método que utiliza la información de todas las evaluaciones realizadas de la función f para decidir que valor de $\mathbf{x}$ evaluar a continuación, reduciendo así el número necesario de evaluaciones de f para encontrar el mínimo.

Para explicar como funciona este método se parte de un formalismo llamado optimización secuencial basada en modelos, que no es más que una generalización de la

optimización bayesiana.

3.2.1. Optimización Secuencial Basada en Modelos

La idea es aproximar la función f a partir de un modelo sustituto $\mathcal{M}$.

Se parte de un conjunto de observaciones $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(k)}, y^{(k)})\}$, donde $y^{(j)} = f(\mathbf{x}^{(j)})$, a partir del cual se ajusta el modelo sustituto $\mathcal{M}$. Luego utilizando las predicciones del modelo se maximiza una función S llamada función de adquisición que elije el siguiente conjunto de hiperparámetros $\mathbf{x}_i \in X$ para evaluar la función f y se agrega el par $(\mathbf{x}_i, f(\mathbf{x}_i))$ al conjunto de observaciones D. Una vez hecho esto se vuelve a ajustar el modelo $\mathcal{M}$ y se repite el proceso, que está explicado en forma de pseudocódigo en el algoritmo 3.

Algoritmo 3 `SMBO`

Input: $f, X, S, \mathcal{M}$

1: $D = \text{InicializarMuestras}(f, X)$
2: **for** $i = 1, 2, ...$ **do**
3: $\quad \mathcal{M} = \text{AjustarModelo}(D)$
4: $\quad \mathbf{x}_i = arg\,\text{máx}_{\mathbf{x} \in X}\, \mathcal{S}(\mathbf{x}, \mathcal{M})$.
5: $\quad y_i = f(\mathbf{x}_i)$ ▷ Paso costoso
6: $\quad D = D \cup \{(\mathbf{x}_i, y_i)\}$
7: **end for**

Optimización Bayesiana

Lo que caracteriza a la optimización bayesiana dentro del formalismo de la optimización secuencial basada en modelos, es justamente la creación del modelo. En la optimización bayesiana se construye un modelo estadístico, donde se representa con $P(y|\mathbf{x})$ la predicción del modelo, siendo y el resultado de una evaluación $f(\mathbf{x})$. El nombre del método se debe a que para la construcción del modelo se utiliza el teorema de Bayes:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)\ P(y)}{P(\mathbf{x})}$$

En la terminología bayesiana, se conoce a $P(y|\mathbf{x})$ como probabilidad a posteriorí o *posterior*, que es proporcional a la probabilidad a priori o *prior* $P(y)$ por la función de verosimilitud o *likelihood* $P(\mathbf{x}|y)$. La probabilidad $P(\mathbf{x})$ es una probabilidad marginal que sirve como factor de normalización, por lo que no es de tanto interés.

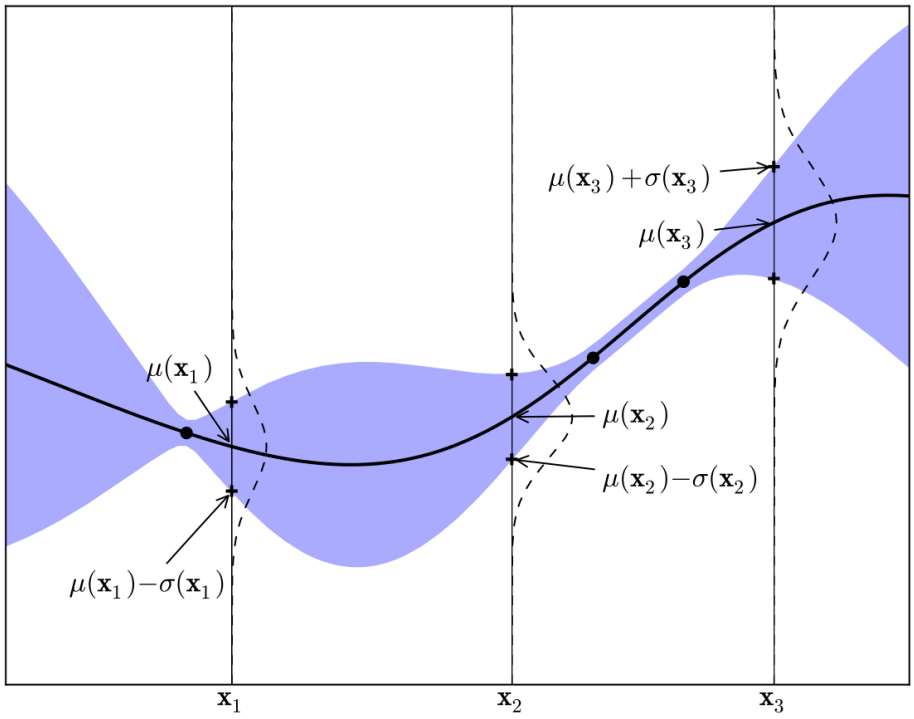

Figura 3.2: Proceso Gaussiano unidimensional con tres observaciones representadas por los puntos negros. La linea gruesa representa la media del modelo predictivo y la zona azul la varianza en cada caso. Se representa con linea de trazo las distribuciones normales para los valores x_1, x_2, y x_3[4].

Procesos Gaussianos

Una opción muy utilizada para la construcción del *prior* y actualización del *posterior* son los procesos gaussianos. Una forma sencilla de entender un proceso gaussiano es pensarlo como una función que para cada valor de x devuelve la media $\mu(x)$ y la varianza $\sigma(x)$ de una distribución normal, en el caso particular de que x sea unidimensional (ver figura 3.2). Con $\mathbf{x}$ multidimensional, se obtiene una distribución normal multivariante, caracterizada por el vector $\boldsymbol{\mu}(\mathbf{x})$ y la matriz de covarianza $\Sigma(\mathbf{x}, \mathbf{x}')$.

Sin embargo en este trabajo no se utilizan procesos gaussianos, principalmente porque parten del supuesto de que f es continua. Para una introducción a la optimización bayesiana con procesos gaussianos ver [4].

3.2.2. Mejora Esperada: Función De Adquisición

Para la elección de los puntos a evaluar en la función real se maximiza la función de adquisición S. Existen varias propuestas de funciones de adquisición, pero en este caso se utiliza la **mejora esperada** o EI(*Expected Improvement*) [8]. Sea y^* un valor de referencia, se define a la mejora esperada con respecto a y^* como:

$$EI_{y^*}(\mathbf{x}) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p(y|\mathbf{x})\, dy \tag{3.1}$$

3.2.3. Estimador de Parzen con Estructura Arbórea

El estimador de Parzen con estructura arbórea o **TPE** (*Tree-Structured Parzen Estimator*) [6] es una estrategia que modela $P(x_i|y)$ para cada $x_i \in X_i$ (es decir, que x_i representa a cada hiperparámetro por separado) a partir de dos distribuciones creadas a utilizando las observaciones D:

$$P(x_i|y) = \begin{cases} \ell(x_i) & \text{si } y < y^* \\ g(x_i) & \text{si } y \geq y^*, \end{cases} \tag{3.2}$$

Donde las densidades $\ell(x_i)$ y $g(x_i)$ se construyen a partir de dos conjuntos D_ℓ y D_g, ambos subconjuntos de D, tal que D_ℓ contiene todas las observaciones con $y < y^*$, y D_g contiene a todo el resto de forma que $D = D_\ell + D_g$. El valor de referencia y^* será un valor por encima del mejor valor observado de $f(\mathbf{x})$, que se selecciona para ser un cuantil $\gamma \in (0, 1)$ de los valores observados y tal que $P(y < y^*) = \gamma$.

Mejora Esperada con TPE

Aplicando la ecuación (3.2) a la definición de mejora esperada (3.1) se obtiene la siguiente relación [6]:

$$EI_{y^*}(x_i) \propto \left(\gamma + (1-\gamma)\frac{g(x_i)}{\ell(x_i)}\right)^{-1} \tag{3.3}$$

Es decir que para maximizar la mejora esperada se debe escoger un valor x_i que maximice el cociente $\ell(x_i)/g(x_i)$ (o minimice $g(x_i)/\ell(x_i)$).

Estimación de las Densidades

Las densidades de probabilidad se estiman utilizando ventanas de Parzen. Sea $D_x = \{x_i \mid (\mathbf{x}, y) \in D_\ell \ (o\ D_g)\}$:

$$P(x_i) = \frac{\sum_{x_i' \in D_x} w_{x_i'} k(x_i, x_i') + w_p k(x_i, x_p)}{\sum_{x_i' \in D_x} w_{x_i'} + w_p}, \tag{3.4}$$

donde $w_{x_i'}$ es el peso de la observación x_i'(ver [9]), x_p es un valor fijo *prior*, w_p es un peso *prior* (por defecto igual a 1) y k es la función *kernel*, que en este caso son distribuciones gaussianas truncadas centradas en los puntos x_i' (ver [10] para más detalle).

Algoritmo

Finalmente se puede ver el procedimiento completo del estimador de Parzen con estructura arbórea en el algoritmo 4. Un detalle importante es que el algoritmo requie-

re un valor n_c, que es el número de candidatos que se utilizarán para maximizar el cociente $\ell(x_i)/g(x_i)$ (es decir, para maximizar la función de adquisición). En la linea 6 del algoritmo se realiza el muestro de los n_c candidatos, utilizando la distribución $\ell(x_i)$, para luego seleccionar x_i^* a partir del conjunto C_i. Para este trabajo se utilizó la implementación de este algoritmo realizada en el paquete ***Optuna*** [11] escrito en el lenguaje de programación Python.

Algoritmo 4 `TPE`

Input:
$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(k)}, y^{(k)})\}$ ▷ Observaciones
$n_t \in \mathbb{N}$ ▷ número de iteraciones
$n_c \in \mathbb{N}$ ▷ número de candidatos
$\gamma \in (0, 1)$ ▷ cuantil para obtener y^*

1: **for** $t = 1, 2, ..., n_t$ **do**
2: $D_\ell = \{(\mathbf{x}, y) \in D | y < y^*, \text{ con } P(y < y^*) = \gamma\}$
3: $D_g = D - D_\ell$
4: **for** $x_i = n_{max}, l_{max}, \varepsilon, ...$ **do** ▷ Para cada hiperparámetro
5: Construir $\ell(x_i)$ con $\{x_i \mid (\mathbf{x}, y) \in D_\ell\}$ y $g(x_i)$ con $\{x_i \mid (\mathbf{x}, y) \in D_g\}$.
6: $C_i = \{x_i^{(j)} \sim \ell(x_i) | j = 1, ..., n_c\}$ ▷ muestreo de n_c candidatos para x_i^*
7: $x_i^* = arg\,\text{máx}_{x_i \in C_i}\, \ell(x_i)/g(x_i)$
8: **end for**
9: $D = D \cup \{(\mathbf{x}^*, f(\mathbf{x}^*)\}$ ▷ $\mathbf{x}^*$ es el vector construido a partir de cada x_i.
10: **end for**

Output: $\mathbf{x}$ con el mínimo valor y en D.

TPE Multivariante

Una alternativa al algoritmo 4 es el algoritmo **TPE Multivariante**, implementado en Optuna [1]. La única diferencia es que en este caso las densidades no se construyen para cada hiperparámetro por separado, sino que se utilizan ventanas de Parzen multivariadas, dónde las funciones *kernel* ahora son distribuciones gaussianas multivariadas. Es decir que en lugar de construir $\ell(x_i)$ y $g(x_i)$ para cada hiperparámetro, ahora se construye directamente $\ell(\mathbf{x})$ y $g(\mathbf{x})$.

3.3. Resultados

3.3.1. Conjunto pequeño: Comparación de métodos

Utilizando un conjunto de entrenamiento con cien ondas equidistantes en el espacio del parámetro unidimensional $q : 1 < q < 8$, se quiere optimizar el error de representación para un conjunto de validación con quinientas ondas (cinco veces más denso).

[1] El algoritmo TPE Multivariante fue introducido en la siguiente actualización de Optuna: https://github.com/optuna/optuna/pull/1767

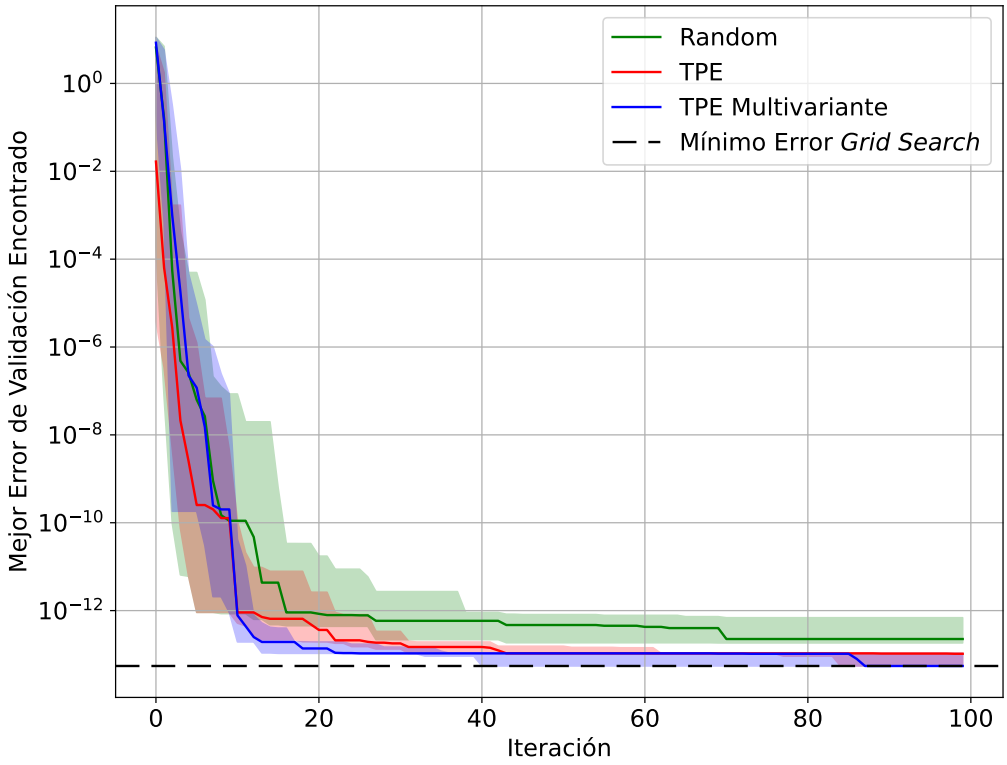

Figura 3.3: Comparación de convergencia. Se muestran los cuartiles para 20 optimizaciones realizadas, en cada caso.

Los hiperparámetros a optimizar son $\mathbf{x} = (n_{max}, l_{max}, \hat{\Lambda}_0)$, dejando ε fijo en 1×10^{-12} para simplificar la búsqueda, que se realiza en los siguientes intervalos:

$$
\begin{aligned}
n_{max} &\in \{5, 6, 7, ..., 20\}, \\
l_{max} &\in \{1, 2, 3, ..., 10\}, \\
\hat{\Lambda}_0 &\in \{q_0 \mid q_0 = 1 + i\Delta q,\ i \in \mathbb{N} : 0 \leq i \leq 99,\ \Delta q = 7/99\}.
\end{aligned}
$$

Son 16 valores de n_{max}, 10 valores de l_{max} y 100 para q_0 ($\hat{\Lambda}_0 = q_0$). Lo que hace un total de 16000 combinaciones posibles.

En la figura 3.3 se puede ver el resultado de realizar 20 optimizaciones de 100 iteraciones con tres diferentes métodos; búsqueda aleatoria (*random*), TPE y TPE multivariante (los tres métodos están implementados en Optuna [11]). En linea oscura se representa la media del mejor error a cada iteración, y la zona sombreada representa los cuartiles. Aparte, en linea de trazo se marca el mejor error obtenido realizando una búsqueda exhaustiva (*grid search*).

En las primeras 10 repeticiones los tres métodos son equivalentes, pues para el algoritmo TPE (tanto el normal como el multivariable) se parte de un muestreo aleatorio

Algoritmo	AUC (Mediana)	Mejor $Mediana(y)$ encontrada
Búsqueda Aleatoria	$2{,}59 \times 10^{-10}$	$2{,}26 \times 10^{-13}$
TPE	$1{,}20 \times 10^{-11}$	$1{,}04 \times 10^{-13}$
TPE Multivariante	$5{,}20 \times 10^{-12}$	$5{,}48 \times 10^{-14}$

Tabla 3.1: Comparación entre algoritmos de optimización.

de 10 observaciones. En la figura 3.3 se ve claramente que luego de las décima iteración se produce el cambio más notorio entre los tres métodos. Gráficamente se puede ver que ambas versiones del algoritmo TPE dan un mejor resultado que la búsqueda aleatoria, pero aparte de esto se pueden utilizar métricas como el **mejor valor encontrado** para la mediana de y o el **área bajo la curva** o **AUC** (*Area Under the Curve*)[12]. En la tabla 3.1 están los resultados de estas métricas, considerando solo las últimas 90 iteraciones.

Búsqueda Exhaustiva

La búsqueda exhaustiva o *grid search* consiste en probar todas las combinaciones posibles dentro de un espacio de hiperparámetros para seleccionar la solución óptima. Es decir que si se quiere buscar la combinación óptima de (n_{max}, l_{max}) para un rango de valores $n_{max} \in N$, $l_{max} \in L$ se deberán probar todas las combinaciones posibles del producto cartesiano $N \times L = \{(n_{max}, l_{max}) | n_{max} \in N, l_{max} \in L\}$.

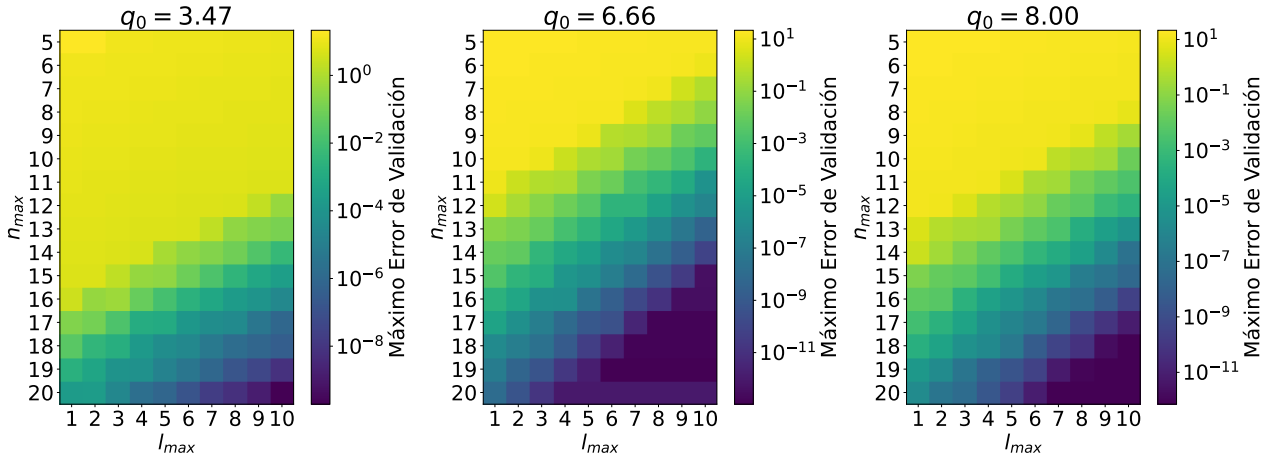

Figura 3.4: Máximo error de validación en función de n_{max} y l_{max} para tres diferentes semillas q_0.

Si bien no se puede graficar el error en función de los tres hiperparámetros a la vez, se puede obtener bastante información al dejar fijo uno o dos hiperparámetros. Por ejemplo en la figura 3.4 se observa el error de validación en función de las combinaciones posibles de n_{max} y l_{max} para tres diferentes semillas.

Luego en la figura 3.5 se ven los resultados de variar únicamente la semilla para diferentes combinaciones de n_{max} y l_{max}. En este conjunto de datos se observa que

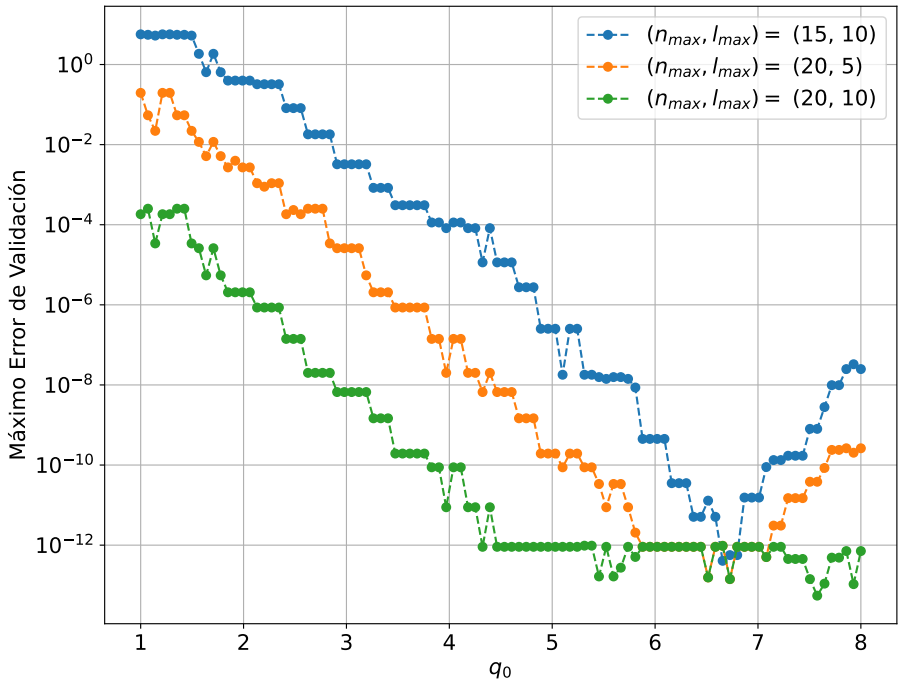

Figura 3.5: Máximo error de validación en función de la semilla q_0 para distintas combinaciones de (n_{max}, l_{max})

para $(n_{max}, l_{max}) = (20, 10)$ hay una diferencia de 9 ordenes de magnitud entre la peor y la mejor semilla (datos en color verde). Sin embargo para $(n_{max}, l_{max}) = (15, 10)$ la diferencia es de 13 ordenes de magnitud (datos de color azul). Además el valor óptimo de la semilla no coincide exactamente en estos dos ejemplos, aunque tengan un comportamiento similar. Es decir que la influencia de la semilla depende del resto de hiperparámetros, sobre todo se tiene que tener en cuenta que en este caso la tolerancia *greedy* tenía un valor $\varepsilon = 1 \cdot 10^{-12}$, por lo que no se va a obtener un resultado mucho mejor que este.

Tiempo de Optimización

Si bien la búsqueda exhaustiva garantiza encontrar el mejor resultado posible dentro del espacio de búsqueda, el tiempo necesario para realizar la búsqueda hace que el método no sea aplicable a casos relativamente complejos. En este caso sencillo, con 16000 combinaciones, la búsqueda requirió **25 horas** para completarse. En cambio las optimizaciones realizadas utilizando los algoritmos TPE tardaron una media de **8 minutos**.

Por último en la figura 3.6 se puede ver gráficamente el proceso de optimización utilizando el algoritmo TPE multivariable.

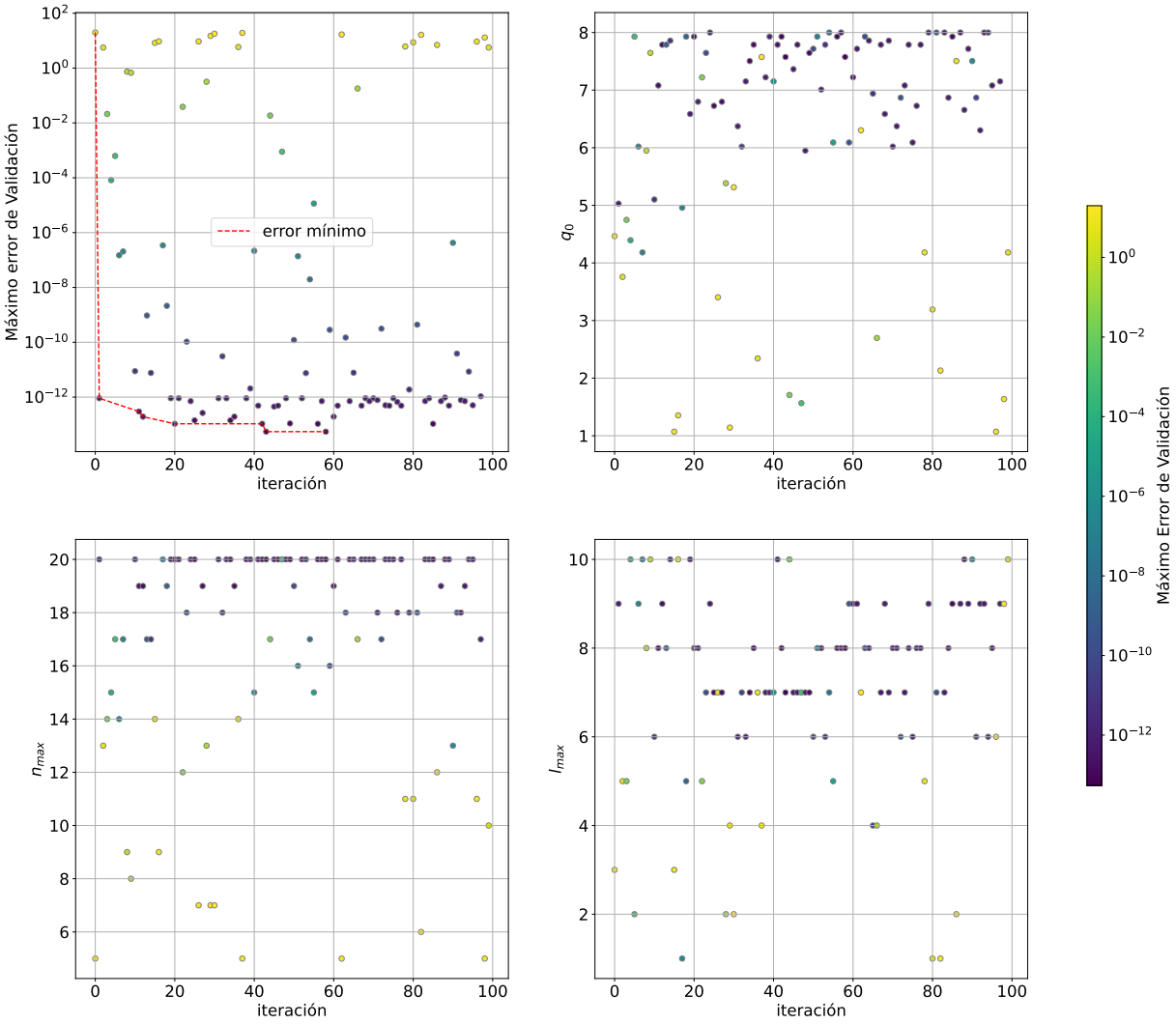

Figura 3.6: Optimización con 100 iteraciones utilizando el algoritmo TPE multivariante para el conjunto de entrenamiento con semilla unidimensional.

3.3.2. Optimización Completa

Utilizando un conjunto de entrenamiento con 70 valores discretos de q equidistantes en el rango [1, 8] y 20 de χ_z ($\chi_{z_1} = \chi_{z_2}$) en el rango [-0.8, 0.8], dando lugar a un total de 1400 funciones de onda, se muestran los resultados de optimizar el máximo error de validación utilizando un conjunto de validación con 100 valores para q y 30 vaores para χ_z con un total de 3000 funciones de onda.

La optimización se realizó en los siguientes espacios de búsqueda:

$$
\begin{aligned}
n_{max} &\in \{10, 11, 12, ..., 60\}, \\
l_{max} &\in \{2, 3, 4, ..., 20\}, \\
\varepsilon &\in \{10^{-20}, 10^{-19}, 10^{-18}, ..., 10^{-4}\}, \\
Q_0 &= \{q_0 \mid q_0 = 1 + i\Delta q,\ i \in \mathbb{N} : 0 \leq i \leq 69,\ \Delta q = 7/69\}, \\
X_0 &= \{\chi_{z_0} \mid \chi_{z_0} = -0{,}8 + j\Delta\chi_z,\ j \in \mathbb{N} : 0 \leq j \leq 19,\ \Delta\chi_z = 1{,}6/19\}, \\
\hat{\Lambda}_0 &\in \{(q_0, \chi_{z_0}) \mid q_0 \in Q_0,\ \chi_{z_0} \in X_0\}.
\end{aligned}
$$

En total se realizaron 500 iteraciones, y el mejor máximo error de validación obtenido en la iteración número 269 fue de 1.45×10^{-6} con los hiperparámetros:

$$
\begin{aligned}
n^*_{max} &= 59, \\
l^*_{max} &= 4, \\
\varepsilon^* &= 10^{-17}, \\
q^*_0 &= 7{,}899, \\
\chi^*_{z_0} &= 0{,}716.
\end{aligned}
$$

En la figura 3.7 se observa la evolución de la optimización realizada, que requirió alrededor de 8 horas para completarse. Se puede ver que para cada hiperparámetro se observa una convergencia a cierto valor, pero sin dejar de lado la exploración, es decir, que se siguen evaluando hiperparámetros fuera del rango que parece óptimo, de forma que se evita caer mucho tiempo en mínimos locales.

Importancia de los Hiperparámetros

Una vez realizada una optimización se puede estimar la importancia relativa de cada hiperparámetro con el algoritmo fANOVA [13]. Básicamente la idea es dividir la varianza total en distintos componentes que representen la varianza producida por cada hiperparámetro. En la figura 3.8 se observa a la izquierda en naranja los resultados para la optimización realizada, y a la derecha en azul se ven los resultados para otro espacio de búsqueda, esta vez con $n_{max} \in \{20, ..., 30\}$ y $l_{max} = \{2, ..., 8\}$ (es decir que se redujo el espacio de búsqueda para n_{max} y l_{max}). Se puede ver que la importancia que tienen los hiperparámetros depende claramente del espacio de búsqueda.

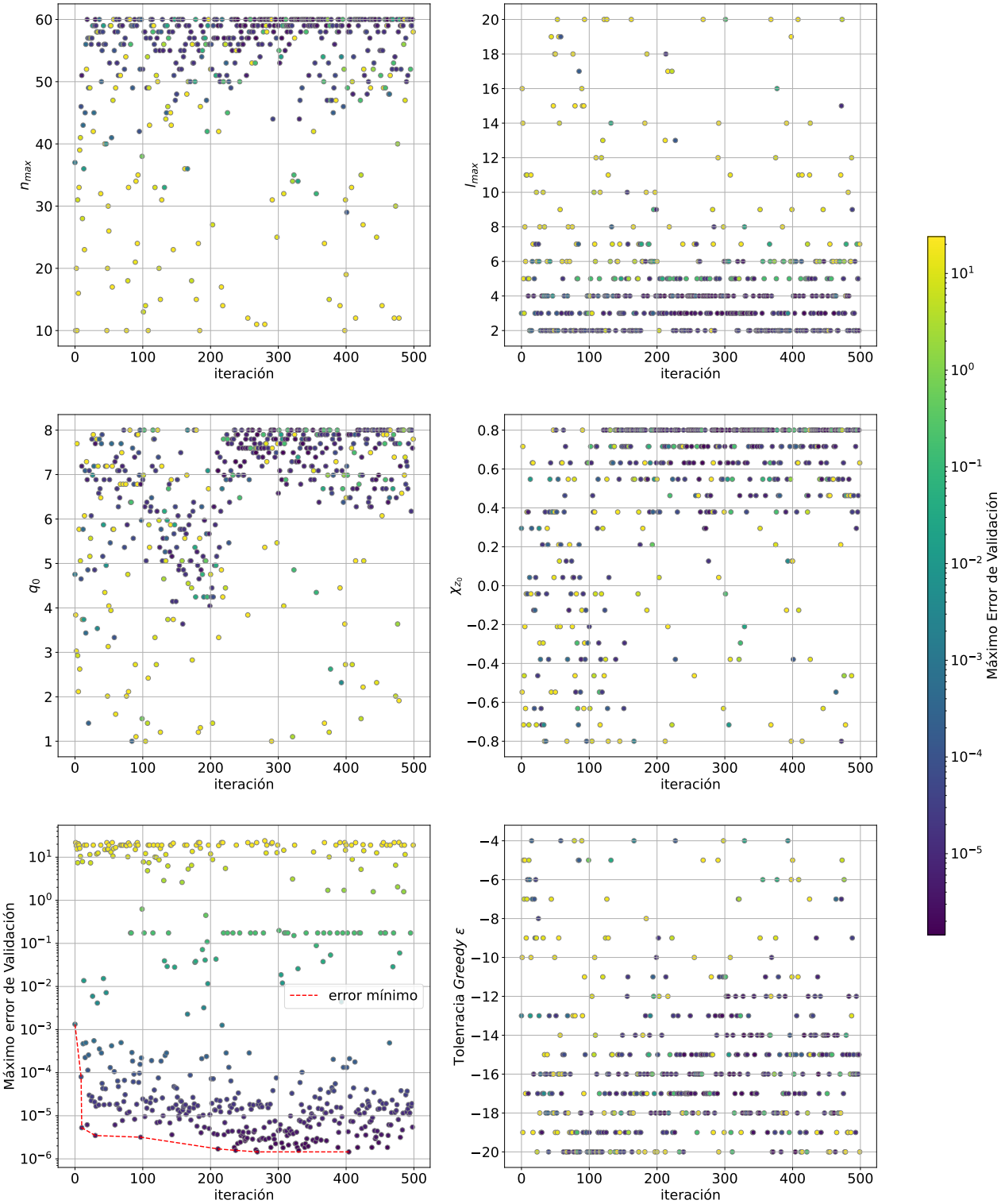

Figura 3.7: caption

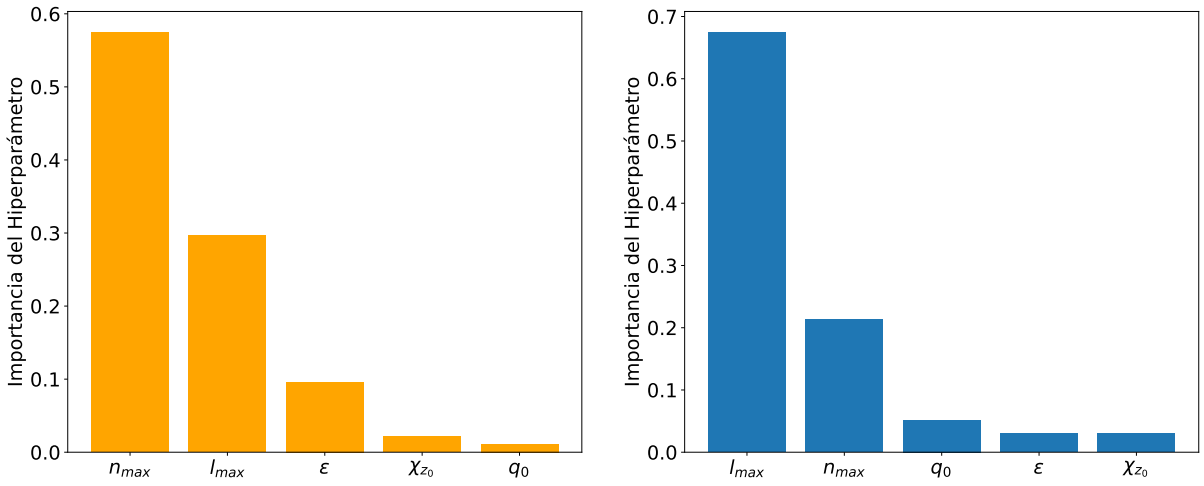

Figura 3.8: Importancia relativa de los hiperparámetros.

Bibliografía

[1] Varma, V., Field, S. E., Scheel, M. A., Blackman, J., Kidder, L. E., Pfeiffer, H. P. Surrogate model of hybridized numerical relativity binary black hole waveforms. *Physical Review D*, **99** (6), mar 2019. 1

[2] Hesthaven, J. S., Gottlieb, S., Gottlieb, D. Spectral Methods for Time-Dependent Problems. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2007. 4

[3] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., de Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, **104** (1), 148–175, 2016. 13

[4] Brochu, E., Cora, V. M., de Freitas, N. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010. URL https://arxiv.org/abs/1012.2599. 13, 16

[5] Dewancker, I., McCourt, M., Clark, S. Bayesian optimization primer, 2015. URL https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf. 13

[6] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B. Algorithms for hyper-parameter optimization. En: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger (eds.) Advances in Neural Information Processing Systems, tomo 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf. 13, 17

[7] Feurer, M., Hutter, F. Hyperparameter Optimization, págs. 3–33. Cham: Springer International Publishing, 2019. URL https://doi.org/10.1007/978-3-030-05318-5_1. 14

[8] Jones, D. A taxonomy of global optimization methods based on response surfaces. *J. of Global Optimization*, **21**, 345–383, 12 2001. 16

[9] Bergstra, J., Yamins, D., Cox, D. D. Making a science of model search, 2012. URL https://arxiv.org/abs/1209.5111. 17

[10] Ozaki, Y., Tanigaki, Y., Watanabe, S., Onishi, M. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems, 2020. URL https://doi.org/10.1145/3377930.3389817. 17

[11] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M. Optuna: A next-generation hyperparameter optimization framework. En: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2019. 18, 19

[12] Dewancker, I., McCourt, M. J., Clark, S. C., Hayes, P., Johnson, A., Ke, G. A strategy for ranking optimization methods using multiple criteria. En: AutoML@ICML. 2016. 20

[13] Hutter, F., Hoos, H., Leyton-Brown, K. An efficient approach for assessing hyperparameter importance. En: E. P. Xing, T. Jebara (eds.) Proceedings of the 31st International Conference on Machine Learning, tomo 32 de *Proceedings of Machine Learning Research*, págs. 754–762. Bejing, China: PMLR, 2014. URL https://proceedings.mlr.press/v32/hutter14.html. 23

Publicaciones asociadas

1. Mi primer aviso en la revista **ABC**, 1996
2. Mi segunda publicación en la revista **ABC**, 1997

Agradecimientos

A todos los que se lo merecen, por merecerlo