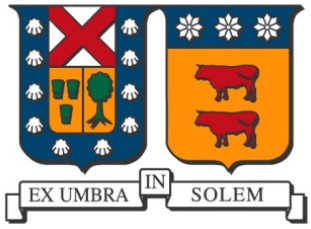
2021-05

INTERPOLACIÓN DEL TIPO LAGRANGE PARA DATOS 2D DISPERSOS

GARCIA GALARSE, LUCAS

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“INTERPOLACIÓN DEL TIPO LAGRANGE PARA DATOS 2D DISPERSOS”

LUCAS GARCÍA GALARSE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Claudio Torres
Profesor Correferente: Roberto León

Mayo - 2021

AGRADECIMIENTOS

Se acerca el final de un largo proceso, un punto que durante muchos años se vio lejano y en más de una ocasión inalcanzable. Este libro, punto de inflexión entre mis estudios y mi vida profesional, es testamento del trabajo de meses y la preparación de años.

Sin embargo, todo el esfuerzo hubiese sido insuficiente sin el invaluable apoyo de los demás: A los amigos que conocí en mi paso por la universidad y a los que me acompañan desde el colegio, por los buenos momentos que pasamos juntos y por darme fuerza en las situaciones difíciles; al profesor Claudio, por sus valiosos consejos; a mi gatita, por hacerme compañía en las noches de estudio; y finalmente a mi familia, por soportarme durante la pandemia y por hacer todo esto posible.

A todos ustedes, Gracias!

RESUMEN

Resumen— Se presenta un algoritmo de interpolación para funciones reales de 2 variables inspirado en el algoritmo de interpolación de Lagrange. La propuesta se basa en realizar una interpolación del tipo Lagrange compleja y obtener la parte real del resultado. Además se presentan mejoras a la propuesta para reducir el costo computacional adaptando el algoritmo a su forma baricéntrica y reducir las oscilaciones del interpolador generando valores imaginarios convenientes.

El desempeño del algoritmo en términos de error de interpolación es comparable al de otras alternativas ampliamente utilizadas para cantidades bajas de puntos. Para obtener resultados razonables con cantidades más altas de puntos se concluye que es necesario emplear polinomios de mayor grado en la generación de los mencionados valores imaginarios convenientes y se propone como trabajo futuro una forma más eficiente de obtenerlos. [1]

Palabras Clave— Interpolación; Lagrange; Complejo; 2D; Sin grilla.

ABSTRACT

***Abstract*—** An interpolation algorithm for 2 variable real functions is presented, inspired in the Lagrange interpolation algorithm. The proposed method is based on performing a complex Lagrange interpolation and obtaining the real part of the result. Improvements are also presented to reduce the computational cost of the proposal by adapting it to it's barycentric form, and to reduce the oscillations of the interpolator generating convenient imaginary values.

The performance of the algorithm in terms of interpolation error is similar to other widely used alternative interpolants, for a low number of nodes. To obtain reasonable results with a higher number of nodes it is necessary to employ a higher degree polynomial in the generation of the mentioned convenient imaginary values and as future work, a more efficient way to obtain them should be studied. [2]

***Keywords*—** Interpolation; Lagrange; Complex; 2D; Meshfree.

[1] El código está disponible en `https://github.com/Lucas18496/Interpolacion-Lagrange-2D`

[2] Code is available at `https://github.com/Lucas18496/Interpolacion-Lagrange-2D`

GLOSARIO

1D: De una dimensión. Refiriéndose a funciones de 1 variable.
2D: De dos dimensiones. Refiriéndose a funciones de 2 variables.
IDW: Algoritmo de interpolación ponderando por el inverso de la distancia.
NNI: Algoritmo de interpolación por “vecino más cercano”.
PPSN: Conjunto de nodos que asegura unicidad en la interpolación polinomial.
RBF: Funciones de base radial.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La interpolación es una herramienta matemática ampliamente utilizada en distintos campos en ciencias e ingeniería, consiste en la obtención de nuevos valores a partir de un conjunto de valores dados. La función que hace posible la obtención de estos nuevos puntos es llamada función interpoladora y dependiendo del tipo de problema se pueden requerir otras características de esta como por ejemplo continuidad, diferenciabilidad, entre otras.

Para funciones escalares reales de 1 variable, un conocido algoritmo de interpolación es el de Lagrange, que permite generar una función interpoladora polinomial a partir de la sumatoria de términos $L_k(x)$ convenientemente construidos de modo que su valor es 1 en el punto x_k de la muestra inicial y 0 en los demás puntos.

El objetivo de este trabajo es obtener un algoritmo de interpolación inspirado en el algoritmo de Lagrange y aplicable a datos provenientes de funciones escalares reales de 2 variables, lo que será llamado algoritmo de Lagrange 2D. Si bien ya existen propuestas similares a lo anteriormente dicho, estas están restringidas a distribuciones específicas de los puntos a interpolar (generalmente un patrón de grilla), por lo que la propuesta de este trabajo será aplicable a cualquier distribución en el conjunto de puntos dado.

Para conseguir este objetivo, se consideraron diferentes maneras de generalizar el algoritmo de Lagrange a funciones de 2 variables, se seleccionó la que resultó más efectiva en pruebas preliminares, se mejoró reduciendo su costo computacional y se agregaron correcciones que permiten reducir oscilaciones innecesarias de la función interpoladora.

Finalmente la propuesta de solución que se presentará consiste en realizar una interpolación de Lagrange compleja en forma baricéntrica considerando el dominio de interpolación como si se tratara del plano complejo, de ella se obtendrá la parte real siendo esta la función interpoladora. Además, se pudo comprobar que definir la componente imaginaria del valor interpolado de manera conveniente permite mejorar considerablemente el desempeño de la función obtenida, siendo esta corrección también agregada al algoritmo.

En el documento se presentará en el Capítulo 1 una definición detallada de lo que se busca conseguir con el algoritmo que se propondrá, luego en el Capítulo 2 se presentará el estado del arte respecto a diferentes propuestas de interpolación 2D con un especial énfasis a métodos similares al de Lagrange, en el Capítulo 3 se definirá a fondo el algoritmo de interpolación y sus respectivas mejoras, para después, en el Capítulo 4, respaldar las mejoras realizadas así como comparar el desempeño del algoritmo propuesto frente a otros métodos de interpolación a través de diferentes experimentos numéricos, presentando finalmente las conclusiones relevantes del trabajo en el Capítulo 5.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

Un caso particular de interpolación es aquel donde el conjunto de puntos se interpola a través de una función escalar de 2 dimensiones, situación que en el presente trabajo es denominada “interpolación 2D”. Existen diversos algoritmos de interpolación 2D, algunos de los cuales trabajan sobre el supuesto de que los puntos a interpolar siguen un patrón de “grilla”, generalmente regular o equiespaciada, sobre el dominio y otros que de un modo más general no hacen supuestos sobre la distribución de los puntos, los que son denominados algoritmos “libres de grilla” (o *meshfree* en inglés) al trabajar sobre un conjunto de datos dispersos. La diferencia entre ambos escenarios se muestra gráficamente en la Figura 1, donde para 9 nodos a interpolar, los puntos rojos siguen una distribución de grilla regular, mientras que los puntos azules tienen una distribución “libre de grilla”.

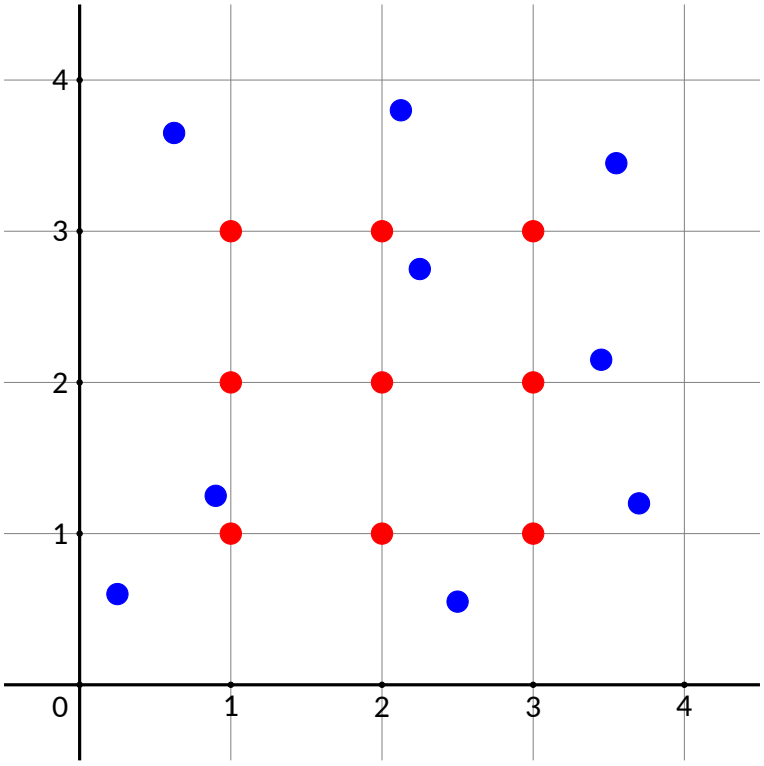


Figura 1: Distribución en grilla v/s libre de grilla.
Fuente: Elaboración propia.

Entre las diversas técnicas de interpolación unidimensional, una bastante conocida es la de los polinomios de Lagrange, que consiste en la generación de un polinomio interpolador mediante una serie de productorias que se suman entre sí. Sin embargo, en la literatura se puede constatar la falta de una definición estándar “libre de grilla” de un algoritmo de interpolación basado en polinomios de Lagrange para datos en 2D.

Es en este contexto que se percibe la oportunidad de plantear un algoritmo que generalice el concepto de interpolación de Lagrange en 2 dimensiones sobre un dominio real, con una propuesta aplicable a aproximaciones “libres de grilla”, esperando que el mismo resulte en una alternativa competente a otros algoritmos ya existentes que permiten interpolar datos provenientes de funciones escalares de 2 variables.

1.1. OBJETIVOS

A continuación se define el objetivo general y los objetivos específicos que se trabajarán en el desarrollo del presente trabajo:

1.1.1. OBJETIVO GENERAL

Desarrollar un algoritmo basado en la interpolación de Lagrange que permita generar una función interpoladora a partir de datos dispersos provenientes de una función escalar de 2 dimensiones, buscando reducir en lo posible el error de interpolación.

1.1.2. OBJETIVOS ESPECÍFICOS

- Plantear diferentes algoritmos de interpolación basados en el algoritmo de Lagrange aplicables a datos dispersos en $\mathbb{R}^2$.
- Comparar el desempeño de los algoritmos planteados entre sí y con respecto a otros algoritmos de interpolación en 2 dimensiones, tomando como medida el error.
- Optimizar el costo computacional del algoritmo planteado que obtenga los mejores resultados en la evaluación anterior.

1.2. IMPACTO DE LA SOLUCIÓN

El planteamiento de un algoritmo de interpolación constituye una herramienta aplicable en múltiples contextos, en [Fasshauer, 2003] el autor aborda diferentes disciplinas en las que resultan útiles las aproximaciones libres de grilla, según explica “la motivación original para este tipo de aproximación proviene de aplicaciones en geodesia, geofísica, mapeo, y meteorología; sin embargo, luego se encontrarán aplicaciones para la misma en solución numérica de ecuaciones diferenciales parciales, inteligencia artificial, teoría de aprendizaje, redes neuronales, procesamiento de señales, teoría de muestras, estadísticas, finanzas, y optimización”.

Respecto a la efectividad de un algoritmo de interpolación tipo Lagrange, en [Gasca y Sauer, 2000] se cuestiona la utilidad de la interpolación polinomial a raíz del fenómeno de Runge, sin embargo, dado que el grado del polinomio es el que se relaciona con la magnitud de las oscilaciones, se concluye que para datos en altas dimensiones la interpolación polinomial podría resultar una herramienta razonable para un número moderado de nodos. En [De Boor y Ron, 1992] se establece que la interpolación polinomial es útil, en general, para hacer aproximaciones locales; más aún, si se maneja bien, por ejemplo, utilizando puntos de Chebyshev, es uno de los métodos más eficientes para lograr aproximación local.

Cabe mencionar que una interpolación del tipo Lagrange no requiere la resolución de sistemas de ecuaciones lineales, a diferencia de otros algoritmos utilizados en aproximaciones libres de grilla como las funciones de base radial (o *RBF* por sus siglas en inglés).

Dado que no existe un modo único de interpolar un conjunto de puntos dado, en una aplicación práctica se preferirá un cierto algoritmo de interpolación en base a distintos criterios que resulten más útiles, como por ejemplo la continuidad de la función interpolante, el menor error en la aproximación o el menor costo computacional. Es por esto que resulta valioso investigar herramientas de interpolación innovadoras de modo que se puedan dilucidar las ventajas de la misma o el contexto en que resulte más práctico aplicarlas.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. INTERPOLACIÓN DE LAGRANGE EN 1D

Se busca interpolar N puntos, presentados como pares (x_k, y_k), con $k \in \{1, 2, ...N\}$ para los que se cumple que $x_i \neq x_j \; \forall i \neq j$, a través de un polinomio $P(x)$ de grado $(N-1)$ o menor. La fórmula para obtener el polinomio interpolador de Lagrange es:

$$P(x) = \sum_{k=1}^{N} y_k L_k(x), \tag{1}$$

donde:

$$L_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^{N} \frac{x - x_j}{x_k - x_j}.$$

Dada esta definición de $L_k(x)$, se obtiene que $L_k(x_k) = 1$ y $L_k(x_j) = 0, \forall i \neq j$. Con esto, se asegura que el polinomio resultante interpola los N puntos.

Más aún, según el principal teorema de interpolación polinomial presentado en [Sauer, 2006], el polinomio obtenido es el único polinomio de grado $(N-1)$ o menor que interpola los n puntos dados.

Un problema que se encuentra asociado a la interpolación polinomial es la incidencia del llamado fenómeno de Runge: en la medida que aumentan los puntos utilizados a su vez aumenta el grado del polinomio interpolador, lo que produce, sobre todo en los bordes del dominio, oscilaciones de magnitudes crecientes que provocan un aumento en el error de interpolación. Es posible minimizar estas oscilaciones si se utiliza como distribución de puntos a los nodos de Chebyshev [Mathews *et al.*, 2004].

2.2. INTERPOLACIÓN DE LAGRANGE EN 2D

En este caso se busca interpolar N puntos $(x_k, y_k, z_k) \in \mathbb{R}^3$, $k \in \{1, ...N\}$, para los que se cumple $(x_i, y_i) \neq (x_j, y_j) \; \forall i \neq j$, a través de una función $P(x, y)$ que puede o no ser un polinomio.

2.2.1. UNICIDAD EN LA INTERPOLACIÓN

En el caso de interpolación polinomial en 1D, es N el número de constantes que define de manera única un polinomio $P(x)$ de grado $(N-1)$ o menor. Del mismo modo, dados N puntos, existe un único polinomio de grado $(N-1)$ o menor que interpola el conjunto.

Esta relación, sin embargo, no aplica para interpolación polinomial en 2D. Un polinomio de 2 variables $P(x,y)$ de grado $(N-1)$ o menor se define de manera única a través de $\frac{(N+1)N}{2}$ constantes. Sin embargo, no necesariamente existe un único polinomio que interpole un conjunto cualquiera de $\frac{(N+1)N}{2}$ puntos, ya que según implica el teorema de Mairhuber-Curtis [Mairhuber, 1956], no es posible asegurar unicidad en una interpolación polinomial multivariada sobre una distribución cualquiera de datos.

Según [Liang y Lu, 1998, Liang *et al.*, 2002] contar con un conjunto de al menos $\frac{(N+1)N}{2}$ puntos es condición necesaria, pero no suficiente, para que los mismos sean interpolados de manera única por un polinomio $P(x,y)$ de grado $(N-1)$ o menor. Un conjunto de puntos que asegura unicidad en la interpolación polinomial recibe el nombre de *properly posed set of nodes* o PPSN.

2.3. ESTADO DEL ARTE

2.3.1. INTERPOLACIÓN TIPO LAGRANGE 2D EN LA LITERATURA

En [Bozorgmanesh *et al.*, 2009] se presenta un ejemplo de uso práctico de interpolación en 2 dimensiones, donde se busca generar una función que determine el tamaño resultante de las partículas en función de la presión y temperatura de un cierto proceso industrial. Esto se consigue a través de una interpolación de los datos obtenidos utilizando polinomios de Lagrange. En concreto los autores proponen, para N posibles valores distintos de x y M posibles valores distintos de y en el conjunto de puntos y para $f(x,y)$ función escalar de 2 variables que se desea aproximar, la siguiente fórmula:

$$P(x,y) = \sum_{i=0}^{N} \sum_{j=0}^{M} f(x_i, y_j) L_{ij}(x,y) \tag{2}$$

Donde:

$$L_{ij}(x,y) = L_i(x)L_j(y)$$
$$L_i(x) = \prod_{\substack{s=0 \\ s\neq i}}^{N} \frac{x - x_s}{x_i - x_s}, \quad L_j(y) = \prod_{\substack{s=0 \\ s\neq j}}^{M} \frac{y - y_s}{y_j - y_s}$$

Por otro lado, [Burkardt, 2019] implementa en diferentes lenguajes de programación un algoritmo de interpolación 2D basado en polinomios de Lagrange, el que matemáticamente sigue la misma estructura que la propuesta anterior.

En [Yi y Yao, 2019] se busca resolver un caso de la ecuación de telégrafo para lo que se emplea un algoritmo de interpolación de Lagrange baricéntrica. La estructura del interpolador es similar a la anterior, aunque para 3 dimensiones:

$$u(x,y,t) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{\tau} \Phi_i(x)\Phi_j(y)\Phi_k(t)u(x_i,y_j,t_k) \tag{3}$$

Sin embargo, en las propuestas presentadas en (2) y (3) se trabaja implícitamente sobre el supuesto que los puntos a interpolar vienen dados por un patrón de "grilla" sobre el dominio. El patrón de "grilla" descrito implica que el modelo asume que cada punto (x_i, y_j, z_{ij}) en el caso 2D o (x_i, y_j, t_k, u_{ijk}) en el caso 3D es conocido para cualquier combinación de i, j e i, j, k respectivamente.

Otro algoritmo de interpolación polinomial es propuesto paralelamente por los autores de [Saniee, 2008] y [Calvi, 2005], basándose en la regla de Cramer. Dados $d+1$ puntos, X vector y $f(X)$ una función escalar, se busca obtener un polinomio $p(X)$, tal que $p(X_i) = f(X_i)$ para $i \in \{0, ...d\}$.

Con los puntos a interpolar es posible obtener la matriz de Vandermonde, el determinante de dicha matriz es representado por VDM$(X_0, ..., X_d)$. Con VDM$(X_0, ..., X_d) \neq 0$ se da una condición necesaria y suficiente para asegurar unicidad en la interpolación polinomial, en otras palabras, el conjunto de puntos se cataloga como PPSN.

Si se cumple la condición anterior, entonces el polinomio interpolador se obtiene mediante:

$$p(X) = \sum_{j=0}^{d} f(X_j)\frac{\text{VDM}(X_0, ...X_{j-1}, X, X_{j+1}, ...X_d)}{\text{VDM}(X_0, ...X_d)}$$

Donde VDM$(X_0, ...X_{j-1}, X, X_{j+1}, ...X_d)$ corresponde al determinante de la matriz obtenida al reemplazar el punto correspondiente a la j-ésima fila de la matriz de Vandermonde

(originalmente X_j) por el vector de incógnitas X. Cabe destacar que este algoritmo no presupone un patrón de grilla en los puntos a interpolar, por lo que es posible aplicarlo sobre datos dispersos, siempre y cuando la distribución de los mismos sea tal que el determinante VDM$(X_0, ..., X_d)$ no sea nulo.

En [Sauer, 1995] se presenta el pseudocódigo de un algoritmo de interpolación polinomial que también opera sobre un conjunto PPSN de datos dispersos. Tomando como base la ecuación (1), este algoritmo hace una analogía entre los polinomios $L_i(x)$ y vectores ortonormales. En otras palabras, para obtener los $L_i(x)$ el método busca generar iterativamente polinomios "ortogonales" entre sí. Para este propósito, se parte con un conjunto inicial de polinomios y se aplica un algoritmo basado en Gram-Schmidt para obtener los valores buscados $L_i(x)$. Una propuesta similar es implementada en MATLAB y C en el algoritmo presentado en [de Boor, 2000].

2.3.2. OTROS ALGORITMOS DE INTERPOLACIÓN 2D

Se presentan algoritmos de interpolación aplicables al tipo de problema tratado en este trabajo pero que no están directamente relacionados con el algoritmo de Lagrange o con interpolación polinomial.

La estructura general de interpolación por splines en dos dimensiones consiste en la construcción de una función interpoladora continua, la que se genera por partes, dividiendo el dominio de interpolación en cuadrados de igual tamaño y definiendo una función para cada uno de estos sub-dominios [Peters y Reif, 2008]. Una restricción asociada a esta propuesta es que solo es aplicable a puntos en una grilla equiespaciada, los que corresponderán a los vértices de los mencionados cuadrados.

La interpolación por vecino más cercano o NNI (*Nearest neighbour interpolation*) es un algoritmo simple de interpolación comúnmente utilizado en aplicaciones gráficas (magnificación de imágenes, aplicación de texturas, etcétera). La función interpoladora entrega como resultado el valor del nodo más cercano en el conjunto de datos [Han, 2013], en base a alguna norma como la distancia euclidiana. Esta interpolación es equivalente a realizar una teselación de Voronoi sobre el dominio asignando a cada polígono el valor correspondiente a cada punto.

Las principales ventajas de NNI son su simplicidad y reducido tiempo de ejecución, sin embargo, la función interpoladora producida es discontinua.

Una interpolación del tipo *piecewise linear* sobre una triangulación se divide en 2 pasos. El primero es la generación de alguna triangulación del conjunto de puntos, por ejemplo, triangulación de Delaunay; luego se interpola mediante una función lineal asociada a cada triángulo, de modo que el interpolador es continuo sobre el dominio [Barnhill, 1977]. De este modo es posible interpolar mediante una función continua una distribución de puntos

dispersa, sin embargo, la función interpoladora no es diferenciable.

La interpolación ponderando por el inverso de la distancia o IDW (*Inverse distance weighting*) consiste en generar el interpolador a través de una sumatoria de términos asociados a cada punto en el conjunto de datos, cada uno de estos términos es ponderado por el inverso de alguna medida de distancia al punto, por ejemplo, distancia euclidiana. Un ejemplo de este tipo de algoritmos es la interpolación de Shepard [Shepard, 1968] definida como:

$$P(x,y) = \begin{cases} \dfrac{\sum_{i=1}^{N} f(x_i,y_i)\,\|(x,y)-(x_i,y_i)\|_2^{-u}}{\sum_{i=1}^{N}\|(x,y)-(x_i,y_i)\|_2^{-u}} & \text{, si } \|(x,y)-(x_i,y_i)\|_2 \neq 0 \\ f(x_i,y_i) & \text{, si } \|(x,y)-(x_i,y_i)\|_2 = 0 \end{cases}$$

El parámetro $u \geq 1$ determina la influencia que tiene la distancia a un punto en la interpolación. En general para mayores valores de u la influencia de puntos más cercanos es mayor y para valores de u menores crece la influencia de puntos más lejanos. Cuando $u \longrightarrow \infty$ el resultado es equivalente a NNI.

La interpolación por funciones de base radial o RBF (*Radial basis functions*) según [Fasshauer, 2007] interpola el conjunto de N puntos mediante una sumatoria de funciones radiales $\kappa(r)$:

$$P(x,y) = \sum_{j=1}^{N} c_j \kappa\left(\|(x,y)-(x_j,y_j)\|_2\right)$$

Dada la condición $P(x_i,y_i) = f(x_i,y_i),\ \forall i \in \{1,...N\}$ es posible determinar los valores de las variables c_j a través de la resolución de un sistema de ecuaciones lineales.

El tipo de RBF viene definido por la función $\kappa(r)$, algunos ejemplos notables son RBF por función de distancia, la más simple de todas, con $\kappa(r) = r$; RBF gaussiana con $\kappa(r) = e^{-(\varepsilon r)^2}$ y RBF *multiquadric* con $\kappa(r) = \sqrt{1+(\varepsilon r)^2}$. El parámetro ε presente en las últimas 2 RBF es llamado parámetro de forma.

Algunas consideraciones a tener en cuenta con este tipo de interpolación es que es necesario resolver un sistema de ecuaciones para generar la función $P(x,y)$ y para el caso de RBF gaussiana o *multiquadric* se debe considerar el valor óptimo para el parámetro ε: En general con valores de ε más cercanos a 0 se obtiene un interpolador más exacto, pero se eleva el número de condición de la matriz asociada al sistema de ecuaciones, generándose un compromiso en la elección del valor apropiado para este parámetro.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

En este capítulo se presentará la justificación detrás de las decisiones de diseño tomadas y la secuencia de pasos que dan origen a la propuesta de solución. Los algoritmos de interpolación implementados son 2, como se puede ver en la Figura 2 representados por la clases **PartReLagrange2D** y **BarLagrange2D**.

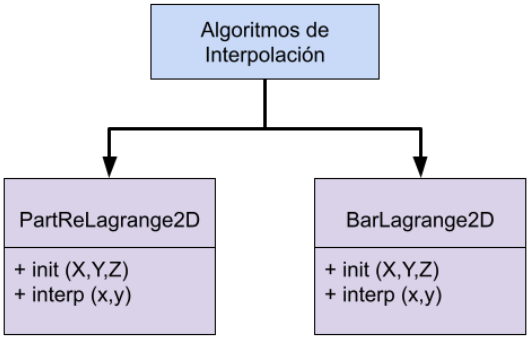


Figura 2: Algoritmos de interpolación implementados.
Fuente: Elaboración propia.

El primer algoritmo de interpolación, **PartReLagrange2D** (Por Lagrange 2D parte real), es presentado en la sección 3.2 y el interpolador corresponde a la parte real del polinomio complejo resultante de una interpolación de Lagrange 1D compleja. El segundo algoritmo, **BarLagrange2D** (Por Lagrange 2D baricéntrico), es detallado en la sección 3.3 y corresponde a llevar el primer algoritmo de interpolación a su forma baricéntrica. Este segundo algoritmo es el que se utilizará en adelante como parte de la propuesta de solución.

Finalmente en la sección 3.4 se explica la necesidad de definir de manera conveniente la parte imaginaria del valor en el eje z de los puntos a interpolar. La generación del interpolador entonces, como se puede ver en la Figura 3, corresponde primero a definir esta parte imaginaria conveniente de modo que minimice una medida de las oscilaciones y luego utilizarla en el algoritmo de interpolación en forma baricéntrica **BarLagrange2D**.

3.1. INTERPOLACIÓN DE LAGRANGE COMPLEJA

En primera instancia es posible hacer una interpolación 2D haciendo un símil entre el dominio de la función y el plano complejo. De este modo los puntos (x_k, y_k, z_k) pueden representarse como (w_k, z_k) donde $w_k = x_k + iy_k$ los que se pueden interpolar con el algo-

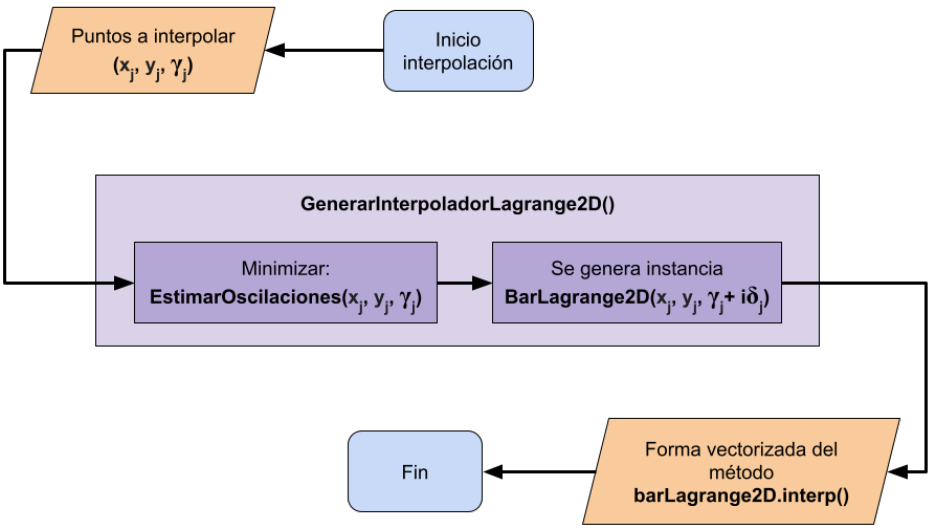


Figura 3: Generación del interpolador a partir de los puntos dados.
Fuente: Elaboración propia.

ritmo de Lagrange en 1D utilizando variable compleja en lugar de real, como se explica en [Bolotnikov, 2014].

Sin embargo, es necesario que la función generada entregue números reales y no complejos como es el caso, lo que se hace entonces es tomar solo la parte real de la función interpoladora antes definida:

$$P(x,y) = \text{Re}\Big(\sum_{k=1}^{N} z_k L_k(x,y)\Big), \quad L_k(x,y) = \frac{\prod_{\substack{j=1 \\ j\neq k}}^{N}((x+iy)-(x_j+iy_j))}{\prod_{\substack{j=1 \\ j\neq k}}^{N}((x_k+iy_k)-(x_j+iy_j))} \tag{4}$$

Donde la función $\text{Re}(x)$ corresponde a la parte real de x. Dado que $L_k = 1$ si $(x = x_k) \wedge (y = y_k)$ y $L_k = 0$ si $(x = x_j) \wedge (y = y_j), \forall j \neq k$, el algoritmo presentado de hecho interpola los puntos y es válido para cualquier distribución de los mismos, no siendo necesario el uso de un patrón de grilla o una distribución del tipo *posed*.

3.2. OBTENCIÓN DE LA PARTE REAL

En la fórmula (4) existe un costo computacional asociado a obtener la parte real de una expresión compleja, además del costo asociado a la obtención de valores puramente imaginarios que no se utilizarán (pues la interpolación solo se construirá con la parte real del resultado). Con todo, resulta conveniente expresar de diferente manera la interpolación, considerando sólo los cálculos necesarios para obtener la parte real.

En primera instancia se tiene que $(x_k, y_k, z_k) \in \mathbb{R}^3; \forall k \in \{1, ...N\}$, sin embargo, por motivos que se verán más adelante, es conveniente expresar z_k de un modo más general, como un número complejo:

$$z_k = \gamma_k + i\delta_k; \quad \gamma_k, \delta_k \in \mathbb{R} \ \forall k \in \{1, ...N\}$$

Llevando este número a su forma exponencial se tiene:

$$z_k = |z_k| \ \text{exp}\,(\, i \ \text{arctan}_2(\gamma_k, \delta_k)), \quad |z_k| = \|(\gamma_k, \delta_k)\|_2$$

En la expresión anterior y en futuras instancias se utilizará la función $\text{arctan}_2(x, y)$, la cual se define de la siguiente manera:

$$\text{arctan}_2(x, y) = \begin{cases} 0, & \text{si } x = 0,\ y = 0 \\ \alpha, \text{tal que } \cos(\alpha) = \frac{x}{\sqrt{x^2+y^2}} \text{ y } \sin(\alpha) = \frac{y}{\sqrt{x^2+y^2}}, & \text{en otro caso} \end{cases}$$

Se define luego $l_k(x, y)$ y utilizando también la forma exponencial de los números complejos:

$$l_k(x, y) = \prod_{\substack{j=1 \\ j\neq k}}^{N} ((x + iy) - (x_j + iy_j)), \quad L_k(x, y) = \frac{l_k(x, y)}{l_k(x_k, y_k)}$$

$$l_k(x, y) = \prod_{\substack{j=1 \\ j\neq k}}^{N} \left(\|(x, y) - (x_j, y_j)\|_2 \ \text{exp}\,(i \,\text{arctan}_2(x - x_j, y - y_j)) \right)$$

$$= \left(\prod_{\substack{j=1 \\ j\neq k}}^{N} \|(x, y) - (x_j, y_j)\|_2 \right) \text{exp} \left(i \sum_{\substack{j=1 \\ j\neq k}}^{N} \text{arctan}_2(x - x_j, y - y_j) \right)$$

Con base en lo anterior es posible definir $L_k(x,y)$ y posteriormente su parte real considerando la forma polar de los números complejos:

$$L_k(x,y) = |z_k| \left(\prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{\|(x,y)-(x_j,y_j)\|_2}{\|(x_k,y_k)-(x_j,y_j)\|_2} \right) \times$$

$$\exp\left(i \left(\arctan_2(\gamma_k,\delta_k) + \sum_{\substack{j=1 \\ j\neq k}}^{N} \arctan_2(x-x_j, y-y_j) - \sum_{\substack{j=1 \\ j\neq k}}^{N} \arctan_2(x_k-x_j, y_k-y_j) \right) \right)$$

$$\therefore \operatorname{Re}(L_k(x,y)) = \left(|z_k| \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{\|(x,y)-(x_j,y_j)\|_2}{\|(x_k,y_k)-(x_j,y_j)\|_2} \right) \times$$

$$\cos\left(\arctan_2(\gamma_k,\delta_k) \sum_{\substack{j=1 \\ j\neq k}}^{N} \left(\arctan_2(x-x_j, y-y_j) - \arctan_2(x_k-x_j, y_k-y_j) \right) \right)$$

De este modo es posible definir finalmente la función interpoladora:

$$P(x,y) = \sum_{k=1}^{N} L_{1k}(x,y)\, L_{2k}(x,y) \tag{5}$$

$$L_{1k}(x,y) = |z_k| \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{\|(x,y)-(x_j,y_j)\|_2}{\|(x_k,y_k)-(x_j,y_j)\|_2},$$

$$L_{2k}(x,y) = \cos\left(\arctan_2(\gamma_k,\delta_k) + \sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x-x_j, y-y_j) - \arctan_2(x_k-x_j, y_k-y_j) \Big) \right)$$

3.2.1. CONEXIÓN CON LAGRANGE 1D

Es posible probar que para datos en 1D el interpolador presentado en (5) es equivalente a la interpolación de Lagrange clásica (1). Para esta demostración se tiene que $y = 0$, $y_i = 0,\ \forall i \in \{1, ...N\}$ y $z_k \in \mathbb{R},\ z_k = \gamma_k + 0i$; entonces, tomando como base algún k:

$$L_{1k}(x, 0) = |z_k| \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{|x - x_j|}{|x_k - x_j|},$$

$$L_{2k}(x, 0) = \cos\left(\arctan_2(\gamma_k, 0) + \sum_{\substack{j=1 \\ j\neq k}}^{N} (\arctan_2(x - x_j, 0) - \arctan_2(x_k - x_j, 0)) \right)$$

Dependiendo de los valores de x y z_k se pueden distinguir dos posibles casos.

- **Caso 1:** $(x - x_j \neq 0, \forall j \in \{1, ...N\}\ \wedge\ z_k \neq 0)$
  Notar que dado $\arctan_2(a - b, 0) = \theta$:
  - $\theta = 0,$ si $(a - b) > 0$
  - $\theta = \pi,$ si $(a - b) < 0$

  Del mismo modo, $\arctan_2(\gamma_k, 0) = 0$ si $z_k > 0$; $\arctan_2(\gamma_k, 0) = \pi$ si $z_k < 0$.
  Entonces:

$$\arctan_2(\gamma_k, 0) + \sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x - x_j, 0) - \arctan_2(x_k - x_j, 0) \Big) = c\pi\,,\ c \in Z$$

  En base a lo anterior se tienen dos posibles escenarios:

$$c \text{ es impar } \Leftrightarrow\ z_k \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{x - x_j}{x_k - x_j} < 0$$

$$c \text{ es par } \Leftrightarrow\ z_k \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{x - x_j}{x_k - x_j} > 0$$

Dado que $\cos(c\pi) = 1 \Leftrightarrow c$ es par y $\cos(c\pi) = -1 \Leftrightarrow c$ es impar, entonces:

$$L_{1k}(x,0)\, L_{2k}(x,0) = z_k \prod_{\substack{j=1 \\ j\neq k}}^{N} \frac{x - x_j}{x_k - x_j}$$

Es decir, el producto de los términos $L_{1k}(x,0)\, L_{2k}(x,0)$ es igual al término $L_k(x)$ de Lagrange en 1D.

- **Caso 2:** $(\exists x_j \mid x - x_j = 0 \ \vee \ z_k = 0)$
  Este caso es trivial, pues $L_{1k}(x,0) = 0$ por lo que $L_{1k}(x,0)\, L_{2k}(x,0) = L_k(x) = 0$.

Por lo tanto, el interpolador $P(x,y)$ presentado en (5) aplicado a datos en 1D es equivalente al algoritmo de interpolación de Lagrange en 1D.

3.2.2. SOBRE EL COSTO COMPUTACIONAL

Tiempo de cómputo:

Para esta y futuras instancias en las que se presente el costo computacional en términos de tiempo de ejecución, se tendrán las siguientes consideraciones: se utilizará como unidad de medida $[f]$ correspondiente al tiempo de ejecución requerido para una operación elemental (suma, resta, multiplicación o división) de punto flotante, se asumirá que el costo de todas estas operaciones es el mismo y se tendrán en cuenta las siguientes constantes:

- $C_a =$ costo de ejecutar la operación $\arctan_2$.
- $C_s =$ costo de ejecutar la operación raíz cuadrada.
- $C_t =$ costo de ejecutar la función sin o la función cos.

En este contexto existen dos procesos importantes a tomar en cuenta, uno es el que se llamará de “precálculo” y consiste en la generación de la función interpoladora y obtención de valores que dependan de los puntos dados (x_k, y_k, z_k), esta operación solo debe realizarse una vez, a menos que el conjunto de puntos cambie. El otro proceso es la interpolación misma, es decir, la ejecución de la función $P(x,y)$ para un par de valores x e y dados, naturalmente esta operación se ejecutará con mayor frecuencia en la mayoría de los casos prácticos.

- Precálculo:
  - El denominador de $L_{1k}(x,y)$ en (5) corresponde a $\sqrt{(x_k-x_j)^2+(y_k-y_j)^2}$ ejecutándose $(N-1)$ veces, lo que corresponde a: $(N-1)(5+C_s)[f]$. Sumado a lo anterior se ejecutan $(N-2)$ productos lo que da un costo para cada L_{1k} de: $(N-1)(5+C_s)+(N-2)[f]$. Considerando esto para $k\in\{1,...N\}$ el costo total es de:
    $(N^2-N)(5+C_s)+(N^2-2N)[f]=(6+C_s)N^2-(7+C_s)N[f]$.
  - Obtener $|z_k|=\sqrt{\gamma_k^2+\delta_k^2}$ tiene un costo de: $3+C_s[f]$, sumando dividirlo por el denominador anterior se tiene un costo de: $4+C_s[f]$. Ejecutándose esto para cada $k\in\{1,...N\}$, el costo total es de:
    $(4+C_s)N[f]$.
  - Es posible precalcular $-\sum_{j=1,\,j\neq k}^{N}\arctan 2(x_k-x_j,y_k-y_j)$ con un costo asociado de: $(N-1)(2+C_a)+(N-2)[f]$. Considerando esto para $k\in\{1,...N\}$ el costo total es de:
    $(3+C_a)N^2-(4+C_a)N[f]$.
  - También se precalcula $\arctan_2(\gamma_k,\delta_k)$ lo que tiene un costo de: $C_a[f]$, además se debe sumar al valor obtenido en el punto anterior, lo que da un costo de: $1+C_a[f]$. Finalmente se considera esto para $k\in\{1,...N\}$, lo que da un costo total de:
    $(1+C_a)N[f]$.
  - Con todo, el costo total del proceso de generar el interpolador (precalcular valores) es de:
    $(9+C_a+C_s)N^2-6N[f]$.
- Interpolación:
  - Costo de calcular el numerador de L_{1k} más una multiplicación: $(N-1)(5+C_s)+(N-2)+1[f]$. Considerando esto para $k\in\{1,...N\}$ el costo total es de:
    $(6+C_s)N^2-(6+C_s)N[f]$.
  - Para L_{2k} se debe considerar el costo de calcular $\sum_{j=1,\,j\neq k}^{N}\arctan 2(x-x_j,y-y_j)$ sumado al costo de una suma y un coseno: $(N-1)(2+C_a)+(N-2)+(C_t+1)[f]$. Considerando esto para $k\in\{1,...N\}$ el costo total es de:
    $(3+C_a)N^2-(3+C_a-C_t)N[f]$.
  - Luego el costo de los productos $L_{1k}\times L_{2k}$:
    $N[f]$.
  - Finalmente el costo de la sumatoria de los valores antes obtenidos:
    $(N-1)[f]$.

- Dando como resultado el costo total de interpolar un par de puntos (x, y): $(9 + C_a + C_s)N^2 - (7 + C_a + C_s - C_t)N - 1[f]$.

Se puede destacar de estos resultados el hecho de que tanto el proceso de precálculo como el de interpolación implican un costo del orden de $O(n^2)$ en términos de tiempo.

Memoria requerida:

Para el costo en términos de uso de memoria del algoritmo se utilizan bits $[b]$ como unidad de medida y se considerará para las variables el espacio requerido para almacenar un número racional en *double precision* de $d[b] = 64[b]$. En este caso también se divide el análisis en las fases de precálculo e interpolación.

- Precálculo:
  - Almacenamiento del vector X: $Nd[b]$
  - Almacenamiento del vector Y: $Nd[b]$
  - Almacenamiento de vectores Z: $2Nd[b]$
  - Valores precalculados de L_{1k} y L_{2k}: $2Nd[b]$
  - Otros valores auxiliares utilizados: $4Nd + 5d[b]$
  - Total: $10Nd + 5d[b]$
    $= 640N + 320[b]$
- Interpolación:
  - Valores de L_{1k} y L_{2k}: $2Nd[b]$
  - Resultado final: $d[b]$
  - Otros valores auxiliares utilizados: $7Nd + 2d[b]$
  - Total: $9Nd + 3d[b]$
    $576N + 192[b]$

En este caso, el espacio utilizado por el algoritmo para ambos procesos es de orden $O(n)$.

3.2.3. IMPLEMENTACIÓN

Se presenta a continuación la implementación del algoritmo de interpolación en Python 3 [Van Rossum y Drake, 2009] utilizando la biblioteca de computación numérica Numpy [Harris *et al.*, 2020]. Se utilizan en lo posible operaciones vectorizadas y se organizan los cálculos dentro del código en pro de minimizar el número de operaciones realizadas. Con

todo, el costo computacional del algoritmo es menor al estimado previamente, pero dentro del mismo orden de magnitud.

Se considerará la clase **PartReLagrange2D**. Al generar una instancia de la misma se realizarán las operaciones de precálculo a través del algoritmo presentado en la función 1.

Luego el algoritmo de interpolación en sí, correspondiente a ejecutar el método **PartReLagrange2D**.interp que a partir de valores dados de (x, y) entrega una estimación de z utilizando en el proceso los valores antes calculados, este se muestra en la función 2.

[3]Notar que la función definida por Numpy np.arctan2(y, x) recibe sus variables de entrada en orden inverso a la función definida en este documento previamente arctan$_2(x, y)$.

Función 1: PrecálculoLagrange2D

Data: X, Y vectores, Z matriz $(2 \times N)$ representando puntos a interpolar $(X[k], Y[k], Z[0,k] + iZ[1,k])$.
Result: instancia de la clase **PartReLagrange2D**.
begin

self.$X \longleftarrow X$
self.$Y \longleftarrow Y$
self.$Zre \longleftarrow Z[0,:]$
self.$Zim \longleftarrow Z[1,:]$
self.$n\_puntos \longleftarrow X$.size

self.$L1\_den \longleftarrow$ np.zeros(self.n_puntos)
self.$L2\_neg \longleftarrow$ np.zeros(self.n_puntos)

for k ***in range***(*self*.n_puntos) **:**
$k\_dif\_X \longleftarrow X[k] -$ self.X
$k\_dif\_Y \longleftarrow Y[k] -$ self.Y
$k\_dif\_X[k] \longleftarrow 1$
$k\_dif\_Y[k] \longleftarrow 1$

$L1k\_den\_X \longleftarrow$ np.square(k_dif_X)
$L1k\_den\_Y \longleftarrow$ np.square(k_dif_Y)
$L1k\_den \longleftarrow$ np.sqrt(np.prod($L1k\_den\_X + L1k\_den\_Y$))
self.$L1\_den[k] \longleftarrow L1k\_den$

$k\_dif\_X[k] \longleftarrow 0$
$k\_dif\_Y[k] \longleftarrow 0$
$L2k\_neg \longleftarrow$ np.sum(np.arctan2(k_dif_Y, k_dif_X)) [3]
self.$L2\_neg[k] \longleftarrow L2k\_neg$

$Z\_abs \longleftarrow$ np.sqrt(np.square(self.Zre) + np.square(self.Zim))
$Z\_ang \longleftarrow$ np.arctan2(self.Zim, self.Zre)
self.$L1\_den \longleftarrow$ np.divide(Z_abs, self.$L1\_den$)
self.$L2\_neg \longleftarrow$ self.$L2\_neg - Z\_ang$

Función 2: InterpolaciónLagrange2D
Data: x, y.
Result: z.
begin
$L1\_num \longleftarrow$ np.zeros(self.n_puntos)
$L2\_pos \longleftarrow$ np.zeros(self.n_puntos)

for k ***in range***(*self*.n_puntos) **:**
$k\_dif\_X \longleftarrow x -$ self.X
$k\_dif\_Y \longleftarrow y -$ self.Y
$k\_dif\_X[k] \longleftarrow 1$
$k\_dif\_Y[k] \longleftarrow 1$

$L1k\_num\_X \longleftarrow$ np.square(k_dif_X)
$L1k\_num\_Y \longleftarrow$ np.square(k_dif_Y)
$L1k\_num \longleftarrow$ np.sqrt(np.prod($L1k\_num\_X + L1k\_num\_Y$))
$L1\_num[k] \longleftarrow L1k\_num$

$k\_dif\_X[k] \longleftarrow 0$
$k\_dif\_Y[k] \longleftarrow 0$
$L2k\_pos \longleftarrow$ np.sum(np.arctan2(k_dif_Y, k_dif_X))
$L2\_pos[k] \longleftarrow L2k\_pos$

$L1 \longleftarrow$ np.multiply($L1\_num,$ self.$L1\_den$)
$L2 \longleftarrow$ np.cos($L2\_pos -$ self.$L2\_neg$)
$L \longleftarrow$ np.multiply($L1, L2$)
$z \longleftarrow$ np.sum(L)
return z

3.3. LAGRANGE 2D EN FORMA BARICÉNTRICA

Una de las ventajas que presenta el algoritmo de Lagrange en 1D es la posibilidad de modificarlo para presentar al interpolador en su forma baricéntrica, lo que permite reducir de manera considerable el tiempo de cómputo necesario para interpolar algún punto.

Con este objetivo en mente se obtendrá un equivalente a la forma baricéntrica a partir de la función interpoladora de Lagrange 2D presentada anteriormente y se harán las modificaciones correspondientes al algoritmo que la implementa. La metodología seguida para la generación de la forma baricéntrica está basada en la presentada por los autores en [Berrut y Trefethen, 2004].

Se toma como base la fórmula presentada en (5), luego se definen $l(x, y)$ y w_k:

$$l(x,y) = \prod_{j=1}^{N} \|(x,y) - (x_j, y_j)\|_2 \ , \qquad w_k = \frac{|z_k|}{\sum\limits_{\substack{j=1 \\ j\neq k}}^{N} \|(x_k, y_k) - (x_j, y_j)\|_2}$$

En base a lo anterior es posible redefinir L_{1k}:

$$L_{1k}(x,y) = \frac{l(x,y)\, w_k}{\|(x,y) - (x_k, y_k)\|_2} \qquad (6)$$
$$\text{para } (x,y) \neq (x_k, y_k)$$

Siguiendo con L_{2k} como estaba definido en (5):

$$L_{2k}(x,y) = \cos\left(\arctan_2(\gamma_k, \delta_k) + \sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x - x_j, y - y_j) \Big) - \sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x_k - x_j, y_k - y_j) \Big) \right)$$

$$L_{2k}(x,y) = \cos\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x - x_j, y - y_j) \Big) - \left(\sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\arctan_2(x_k - x_j, y_k - y_j) \Big) - \arctan_2(\gamma_k, \delta_k) \right) \right)$$

A partir de esto, se define α_k y se redefine $L_{2k}(x,y)$ utilizando la propiedad trigonométrica del coseno de la diferencia:

$$\alpha_k = \sum_{\substack{j=1 \\ j\neq k}}^{N} (\text{arctan}_2(x_k - x_j, y_k - y_j)) - \text{arctan}_2(\gamma_k, \delta_k)$$

$$L_{2k}(x,y) = \cos\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} \Big(\text{arctan}_2(x - x_j, y - y_j)\Big) - \alpha_k\right)$$

$$L_{2k}(x,y) = \sin\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} (x - x_j, y - y_j)\right) \sin(\alpha_k) + \cos\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} (x - x_j, y - y_j)\right) \cos(\alpha_k)$$

Se define la función $\theta(x,y)$ como:

$$\theta(x,y) = \sum_{j=1}^{N} \text{arctan}_2(x - x_j, y - y_j)$$

Entonces es posible redefinir la siguiente expresión:

$$\sin\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} (x - x_j, y - y_j)\right) = \sin\big(\theta(x,y) - \text{arctan}_2(x - x_k, y - y_k)\big)$$

$$= \sin\big(\theta(x,y)\big) \cos\left(\text{arctan}_2(x - x_k, y - y_k)\right) - \cos\big(\theta(x,y)\big) \sin\left(\text{arctan}_2(x - x_k, y - y_k)\right)$$

De modo similar se hace el cambio:

$$\cos\left(\sum_{\substack{j=1 \\ j\neq k}}^{N} (x - x_j, y - y_j)\right) = \cos\big(\theta(x,y) - \text{arctan}_2(x - x_k, y - y_k)\big)$$

$$= \cos\big(\theta(x,y)\big) \cos\left(\text{arctan}_2(x - x_k, y - y_k)\right) + \sin\big(\theta(x,y)\big) \sin\left(\text{arctan}_2(x - x_k, y - y_k)\right)$$

Luego, se expresa $L_{2k}(x, y)$ como:

$$\begin{aligned}\sin(\alpha_k)\left(\sin(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)-\cos(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right) \quad (7)\\ +\cos(\alpha_k)\left(\cos(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)+\sin(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right)\end{aligned}$$

Al combinar los resultados obtenidos en (6) y (7), se obtiene lo que los autores en [Berrut y Trefethen, 2004] denominan como la primera forma baricéntrica:

$$P_1(x, y) = \sum_{k=1}^{N} L_{1k}(x, y)\, L_{2k}(x, y)$$
$$L_{1k}(x, y) = \frac{l(x, y)\, w_k}{\|(x, y) - (x_k, y_k)\|_2},$$

$$\begin{aligned}L_{2k}(x,y) = \sin(\alpha_k)\left(\sin(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)-\cos(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right)\\ +\cos(\alpha_k)\left(\cos(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)+\sin(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right)\end{aligned}$$

En esta instancia es posible utilizar esta propuesta para interpolar la función constante $F(x, y) = 1$ tomando como base los puntos (x_k, y_k) para $k \in \{1, ...N\}$. Se llamará a esta función interpoladora $\hat{P}_1(x, y)$.

$$\hat{P}_1(x, y) = \sum_{k=1}^{N} \hat{L}_{1k}(x, y)\, \hat{L}_{2k}(x, y)$$
$$\hat{L}_{1k}(x, y) = \frac{l(x, y)\, \hat{w}_k}{\|(x, y) - (x_k, y_k)\|_2},$$

$$\begin{aligned}\hat{L}_{2k}(x,y) = \sin(\hat{\alpha}_k)\left(\sin(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)-\cos(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right)\\ +\cos(\hat{\alpha}_k)\left(\cos(\theta)\cos\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)+\sin(\theta)\sin\left(\arctan_2\left(\frac{y-y_k}{x-x_k}\right)\right)\right)\end{aligned}$$

Es importante señalar que la diferencia entre $\hat{P}_1(x, y)$ y $P_1(x, y)$ radica en los términos $\hat{w}_k$ y $\hat{\alpha}_k$, los que se pueden definir a partir de los términos ya conocidos:

$$\hat{w}_k = \frac{w_k}{|z_k|}, \qquad \hat{\alpha}_k = \alpha_k + \text{arctan}_2(\gamma_k, \delta_k)$$

Finalmente se divide $P_1(x, y)$ por $\hat{P}_1(x, y)$ para obtener lo que se conoce como segunda forma baricéntrica, representado por la función $P(x, y)$. De este modo, dado que se cancela el término $l(x, y)$ ya no es necesario calcularlo.

$$P(x,y) = \frac{\displaystyle\sum_{k=1}^{N} \frac{w_k\, L_{2k}(x,y)}{\|(x,y)-(x_k,y_k)\|_2}}{\displaystyle\sum_{k=1}^{N} \frac{\hat{w}_k\, \hat{L}_{2k}(x,y)}{\|(x,y)-(x_k,y_k)\|_2}} \tag{8}$$

La ventaja de presentar el interpolador de esta manera, es que tanto los valores de w_k como α_k se pueden precalcular, mientras que los valores de $l(x, y)$ y $\theta(x, y)$ solo es necesario calcularlos una vez durante el proceso de interpolación.

3.3.1. SOBRE EL COSTO COMPUTACIONAL

Siguiendo el mismo procedimiento que en la sección 3.2.2 se obtendrá el costo computacional asociado al algoritmo de interpolación usando representación baricéntrica.

Tiempo de cómputo:

- Precálculo:
  - El denominador de w_k en (8) corresponde a $\sqrt{(x_k - x_j)^2 + (y_k - y_j)^2}$ ejecutándose $(N - 1)$ veces, lo que corresponde a: $(N - 1)(5 + C_s)[f]$. Sumado a lo anterior se ejecutan $(N - 2)$ productos lo que da un costo para cada w_k de: $(N - 1)(5 + C_s) + (N - 2)[f]$. Considerando esto para $k \in \{1, ...N\}$ el costo total es de:
    $(N^2 - N)(5 + C_s) + (N^2 - 2N)[f] = (6 + C_s)N^2 - (7 + C_s)N[f]$.
  - Obtener $|z_k| = \sqrt{\gamma_k^2 + \delta_k^2}$ tiene un costo de: $3 + C_s[f]$, sumado a dividirlo por el denominador antes calculado, se tiene un costo de: $4 + C_s[f]$. Ejecutándose lo anterior para cada $k \in \{1, ...N\}$, el costo total es de:
    $(4 + C_s)N[f]$.
  - Para obtener el primer término de $\alpha_k \sum_{j=1,\, j\neq k}^{N} \text{arctan}\, 2(x_k - x_j, y_k - y_j)$ se tiene un costo asociado de: $(N - 1)(2 + C_a) + (N - 2)[f]$. Considerando esto para $k \in \{1, ...N\}$ el costo total es de:

$(3 + C_a)N^2 - (4 + C_a)N[f]$.

- También se precalcula $\arctan_2(\gamma_k, \delta_k)$ lo que tiene un costo de: $C_a[f]$, además se debe restar al valor obtenido en el punto anterior, lo que da un costo de: $1+C_a[f]$. Finalmente se considera esto para $k \in \{1, ...N\}$, lo que da un costo total de: $(1 + C_a)N[f]$.
- Con todo, el costo total del proceso de generar el interpolador (precalcular valores) es de: $(9 + C_a + C_s)N^2 - 6N[f]$.

- Interpolación:
  - Costo de calcular $\theta(x, y)$: Se tiene $\arctan_2(x - x_j, y - y_j)$ ejecutándose para $j \in \{1, ...N\}$ con un costo de $(2 + C_a)N[f]$. Considerando además $(N - 1)$ sumas se obtiene un costo total de: $(3 + C_a)N - 1[f]$
  - Costo asociado a calcular el denominador común $\sqrt{(x - x_k)^2 + (y - y_k)^2}$ de $(5 + C_s)[f]$. Considerando esto para $k \in \{1, ...N\}$ se tiene un costo total de: $(5 + C_s)N[f]$
  - Costo asociado a L_{2k}: Se requiere calcular $\arctan_2(x - x_k, y - y_k)$ con un costo de $(2 + C_a)[f]$. A esto se agregan los productos y operaciones trigonométricas, llegando a un costo de $(11 + C_a + 10C_t)[f]$. Finalmente se aplica lo anterior para cada $k \in \{1, ...N\}$ y se considera el cálculo de L_{2k} asociado al numerador y denominador, dando un costo total de: $(22 + 2C_a + 20C_t)N[f]$
  - Se considera el producto entre w_k y L_{2k} y la división correspondiente, ocurriendo tanto en el numerador como en el denominador por un costo de $4[f]$. Se aplica lo anterior para cada $k \in \{1, ...N\}$ dando un costo total de: $4N[f]$
  - Se tiene el costo de ejecutar las sumatorias del numerador y denominador, más una división: $2N - 1[f]$
  - Lo anterior da como resultado el costo total de interpolar un par de puntos (x, y) de: $(36 + 3C_a + C_s + 20C_t)N - 2[f]$

Se puede observar que el costo computacional en términos de tiempo de ejecución es, en la operación de precálculo, igual entre la propuesta anterior 3.2.2 (parte real de Lagrange complejo) y la forma baricéntrica. Sin embargo, cuando se trata de realizar la interpolación de un punto dado, la forma baricéntrica sólo tiene un costo del orden de $O(n)$.

Memoria requerida:

- Precálculo:
  - Almacenamiento del vector X: $Nd[b]$
  - Almacenamiento del vector Y: $Nd[b]$
  - Almacenamiento de vectores Z: $2Nd[b]$
  - Valores de w_k: $Nd[b]$
  - Valores de $\hat{w}_k$: $Nd[b]$
  - Valores de α_k: $Nd[b]$
  - Valores de $\hat{\alpha}_k$: $Nd[b]$
  - Valores almacenados de sin/cos de α_k: $2Nd[b]$
  - Valores almacenados de sin/cos de $\hat{\alpha}_k$: $2Nd[b]$
  - Otros valores auxiliares utilizados: $4Nd + 4d[b]$
  - Total: $16Nd + 4d[b]$
    $= 1024N + 256[b]$
- Interpolación:
  - Valor almacenado de θ: $d[b]$
  - Valor almacenado de sin/cos de θ: $2d[b]$
  - Valores de L_k: $Nd[b]$
  - Valores de $\hat{L}_k$: $Nd[b]$
  - Valores del divisor común: $Nd[b]$
  - Resultado final: $d[b]$
  - Otros valores auxiliares utilizados: $8Nd + 2d[b]$
  - Total: $11Nd + 6d[b]$
    $= 704N + 384[b]$

Se puede apreciar que el costo en términos de uso de memoria es mayor en la forma baricéntrica del algoritmo, lo que prueba un compromiso entre un reducido tiempo de cómputo para el interpolador a precio de un mayor consumo de memoria tanto en el precálculo como en la interpolación.

Sin embargo, se ha de señalar que en ambos procesos el costo espacial de Lagrange en forma baricéntrica sigue siendo del orden $O(n)$ y es en todos los casos bajo para el estándar de los computadores actuales (se necesitaría interpolar una data de aproximadamente 7×10^6 puntos para saturar 1 Gigabyte de memoria en el proceso de precálculo). Por ello, es la forma baricéntrica la que se utilizará en el futuro.

3.3.2. IMPLEMENTACIÓN

Se presenta a continuación la implementación del algoritmo de interpolación en su forma baricéntrica, ahora se considera la clase **BarLagrange2D**. Para generar una instancia de la misma se deben realizar las operaciones de precálculo a través del algoritmo presentado en la función 3.

Luego, el algoritmo de interpolación, correspondiente a ejecutar el método **BarLagrange2D**.interp, se presenta en la función 4. Para evitar un error de división por 0 se agrega una condición de modo que siempre que (x, y) sea un punto que se encuentra en la data, en lugar de ejecutarse la interpolación se entrega simplemente el valor correspondiente z_k de la misma data.

Función 3: PrecálculoLagrange2DBaricéntrico

Data: X, Y vectores, Z matriz $(2 \times N)$ representando puntos a interpolar $(X[k], Y[k], Z[0,k] + iZ[1,k])$.

Result: instancia de la clase **BarLagrange2D**.

begin

self.$X \longleftarrow X$
self.$Y \longleftarrow Y$
self.$Zre \longleftarrow Z[0,:]$
self.$Zim \longleftarrow Z[1,:]$

self.$n\_puntos \longleftarrow X.size$
self.$W \longleftarrow$ np.zeros(self.n_puntos)
self.$W\_noZ \longleftarrow$ np.zeros(self.n_puntos)
self.$\alpha \longleftarrow$ np.zeros(self.n_puntos)
self.$\alpha\_noZ \longleftarrow$ np.zeros(self.n_puntos)

for k ***in range***(*self*.n_puntos) **:**
$k\_dif\_X \longleftarrow X[k] -$ self.X
$k\_dif\_Y \longleftarrow Y[k] -$ self.Y
$k\_dif\_X[k] \longleftarrow 1$
$k\_dif\_Y[k] \longleftarrow 1$

.

$Wk\_X \longleftarrow$ np.square(k_dif_X)
$Wk\_Y \longleftarrow$ np.square(k_dif_Y)
$Wk \longleftarrow 1/$np.sqrt(np.prod($Wk\_X + Wk\_Y$))
self.$W[k] \longleftarrow Wk$
self.$W\_noZ[k] \longleftarrow Wk$

$k\_dif\_X[k] \longleftarrow 0$
$k\_dif\_Y[k] \longleftarrow 0$
$\alpha_k \longleftarrow$ np.sum(np.arctan2(k_dif_Y, k_dif_X))
self.$\alpha[k] \longleftarrow \alpha_k$
self.$\alpha\_noZ[k] \longleftarrow \alpha_k$

$Z\_abs \longleftarrow$ np.sqrt(np.square(self.Zre) + np.square(self.Zim))
self.$W \longleftarrow$ np.multiply(self.W, Z_abs)
self.$\alpha \longleftarrow$ self.$\alpha -$ np.arctan2(self.Zim, self.Zre)

self.$s\_\alpha \longleftarrow$ np.sin(self.α)
self.$c\_\alpha \longleftarrow$ np.cos(self.α)
self.$s\_\alpha\_noZ \longleftarrow$ np.sin(self.α)
self.$c\_\alpha\_noZ \longleftarrow$ np.cos(self.α)

Función 4: InterpoladiónLagrange2DBaricéntrica

Data: x, y.
Result: z.
begin
    if $(x$ ***in*** *self.*$X)$ ***and*** $(y$ ***in*** *self.*$Y)$ **:**
        $ind\_x \longleftarrow$ np.where(self.$X == x)[0]$
        $ind\_y \longleftarrow$ np.where(self.$Y == y)[0]$
        $intersec \longleftarrow$ np.intersect1d(ind_x, ind_y)
        if $intersec$.*shape*$[0] > 0$ **:**
            return self.$Zre[intersec[0]]$

    $X\_dif \longleftarrow x -$ self.X
    $Y\_dif \longleftarrow y -$ self.Y
    $ang \longleftarrow$ np.arctan2(Y_dif, X_dif)
    $c\_t \longleftarrow$ np.cos(ang)
    $s\_t \longleftarrow$ np.sin(ang)

    $suma\_cuad \longleftarrow$ np.square$(X_dif) +$ np.square(Y_dif)

    $\theta \longleftarrow$ np.sum(ang)
    $c\_\theta \longleftarrow$ np.cos(θ)
    $s\_\theta \longleftarrow$ np.sin(θ)

    $Lk\_t1 \longleftarrow s\_\theta * c\_t - c\_\theta * s\_t$
    $Lk\_t2 \longleftarrow c\_\theta * c\_t + s\_\theta * s\_t$
    $Lk \longleftarrow$ np.multiply(self.$s\_\alpha, Lk\_t1) +$ np.multiply(self.$c\_\alpha, Lk\_t2)$
    $Lk\_noZ \longleftarrow$
     np.multiply(self.$s\_\alpha\_noZ, Lk\_t1) +$ np.multiply(self.$c\_\alpha\_noZ, Lk\_t2)$

    $div \longleftarrow$ np.sqrt$(suma_cuad)$
    $num \longleftarrow$ np.sum(np.divide(np.multiply(self.$W, Lk), div))$
    $den \longleftarrow$ np.sum(np.divide(np.multiply(self.$W\_noZ, Lk\_noZ), div))$
    $z \longleftarrow num/den$
    return z

3.4. CORRECCIÓN USANDO PARTE IMAGINARIA

En principio el algoritmo basado en la parte real de Lagrange complejo presentado en la sección 3.2 es suficiente para interpolar una función real de 2 variables. Luego el algoritmo de Lagrange 2D en forma baricéntrica de la sección 3.3 produce un interpolador que es matemáticamente equivalente al anterior, aunque con la ventaja de reducir en un orden de magnitud

el tiempo de cómputo requerido para el proceso de interpolación.

Sin embargo, las pruebas iniciales realizadas dan cuenta de un comportamiento indeseado que merma la efectividad del algoritmo: La función interpoladora generada a partir de una serie de puntos tiende a presentar una serie de oscilaciones que aumentan en número y magnitud en la medida que se incrementa el número de puntos.

Un ejemplo de este fenómeno se puede ver en la Figura 4 (Gráfico generado utilizando Wolfram Mathematica 12.1 [Wolfram Research, 2020]) donde a partir de la función $F(x, y) = x + y$ se toman 4 y 9 puntos equiespaciados los que se usan en cada caso para generar una función interpoladora. Se puede observar que a pesar de la simplicidad de la función original, se producen una serie de las antes mencionadas oscilaciones que empeoran en la medida en que la data provista al algoritmo de interpolación aumenta.

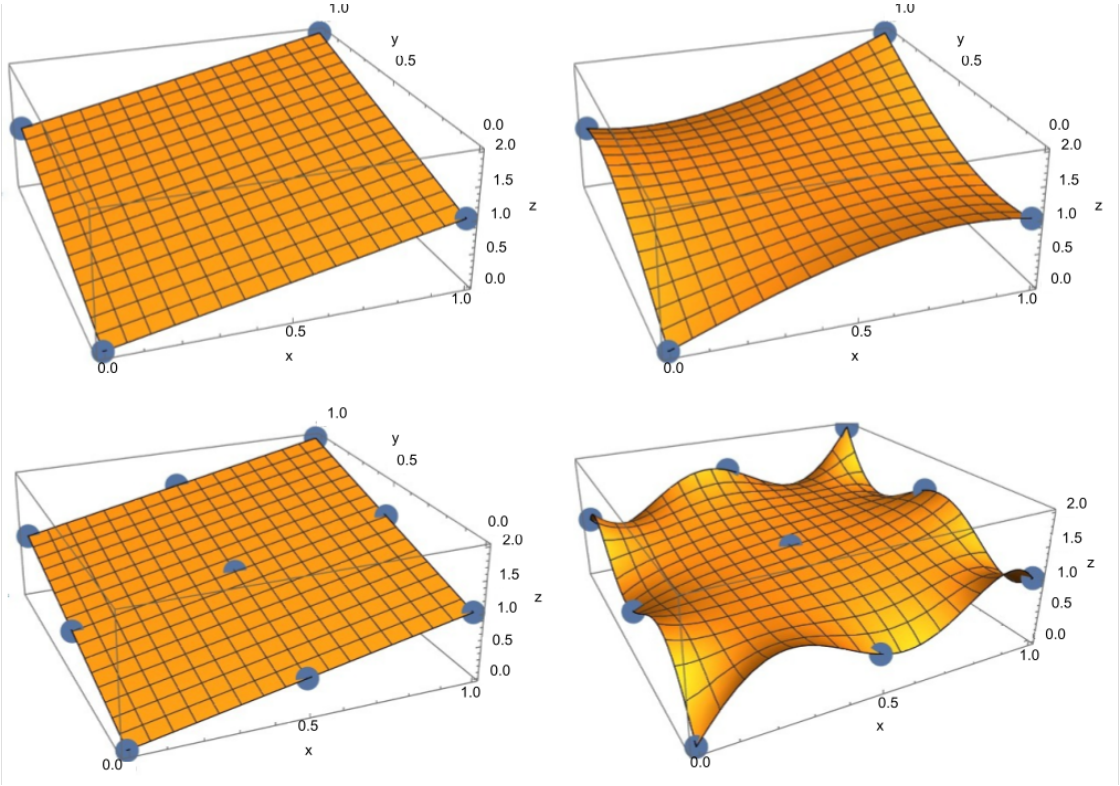


Figura 4: Comparación entre la función original (izquierda) y la obtenida interpolando 4 y 9 puntos equiespaciados (derecha).

Fuente: Elaboración propia.

Esta situación recuerda al fenómeno de Runge, pero se produce por un motivo diferente: La función interpoladora corresponde a la parte real del único polinomio complejo de grado mínimo que interpola los puntos $(x_k + iy_k, \gamma_k)$, esto implícitamente no sólo requiere que la parte real del polinomio complejo interpole los valores γ_k donde corresponda, sino que también exige que su parte imaginaria sea 0 en cada punto de la data.

Lo que se hará para solucionar este problema es entonces sumar un valor imaginario conveniente $i\delta_k$, $\delta_k \in \mathbb{R} \forall k \in \{1, ...N\}$ a cada valor γ_k original que permita reducir las oscilaciones, proceso que será llamado por simplicidad "corrección imaginaria" y realizar la interpolación usando los nuevos valores corregidos $z_k = \gamma_k + i\delta_k$. Es a razón de esto que los algoritmos antes propuestos aceptan valores complejos para el parámetro z_k.

Es importante mencionar que dado que el polinomio complejo señalado interpola los puntos y que el interpolador producido por el algoritmo corresponde solo a la parte real de dicho polinomio, cualquier corrección imaginaria agregada no cambiará el hecho de que los puntos originales son interpolados, más sí cambiará la estructura de la función interpoladora.

3.4.1. PROPUESTA DE CORRECCIÓN

La idea es suponer una estructura para la parte imaginaria que se agregará a cada punto y optimizarla. Se asume una función g que genera esta corrección imaginaria, se realiza la interpolación, se establece una medida de las oscilaciones de la función interpoladora y se ajustan los parámetros de g de modo que produzca una corrección que permita minimizar las oscilaciones del interpolador. Un diagrama de este proceso se presenta en la Figura 5.

Se supondrá una estructura del tipo $z_k = \gamma_k + i\,\text{Im}(g(x_k + iy_k))$ de modo que la corrección imaginaria corresponde a $\delta_k = \text{Im}(g(x_k + iy_k))$ para una función $g : \mathbb{C} \longrightarrow \mathbb{C}$. Se establecerá la función g como un polinomio complejo de segundo grado, debido a que el algoritmo de interpolación propuesto está basado a su vez en la generación de un polinomio complejo, y que este grado es suficiente para producir resultados satisfactorios en tiempos razonables. En la práctica se pudo observar la tendencia de que a mayor grado del polinomio complejo g se podían reducir más efectivamente las oscilaciones a precio de aumentar el tiempo de cómputo requerido para generar la corrección imaginaria.

Así, la estructura de la función g corresponde a:

$$g(c) = \phi_1\, c^2 + \phi_2\, c + \phi_3 + i\psi_1\, c^2 + i\psi_2\, c + i\psi_3\,,$$
$$\text{con } c = x + iy$$

De ella, se utilizará sólo la parte imaginaria para la corrección:

$$\text{se tiene } c^2 = x^2 + 2ixy - y^2\,,$$
$$g(c) = \phi_1(x^2 + 2ixy - y^2) + \phi_2(x + iy) + \phi_3 + i\psi_1(x^2 + 2ixy - y^2) + i\psi_2(x + iy) + i\psi_3$$
$$\text{Im}(g(c)) = \phi_1 2xy + \phi_2 y + \psi_1(x^2 - y^2) + \psi_2 x + \psi_3 \quad (9)$$

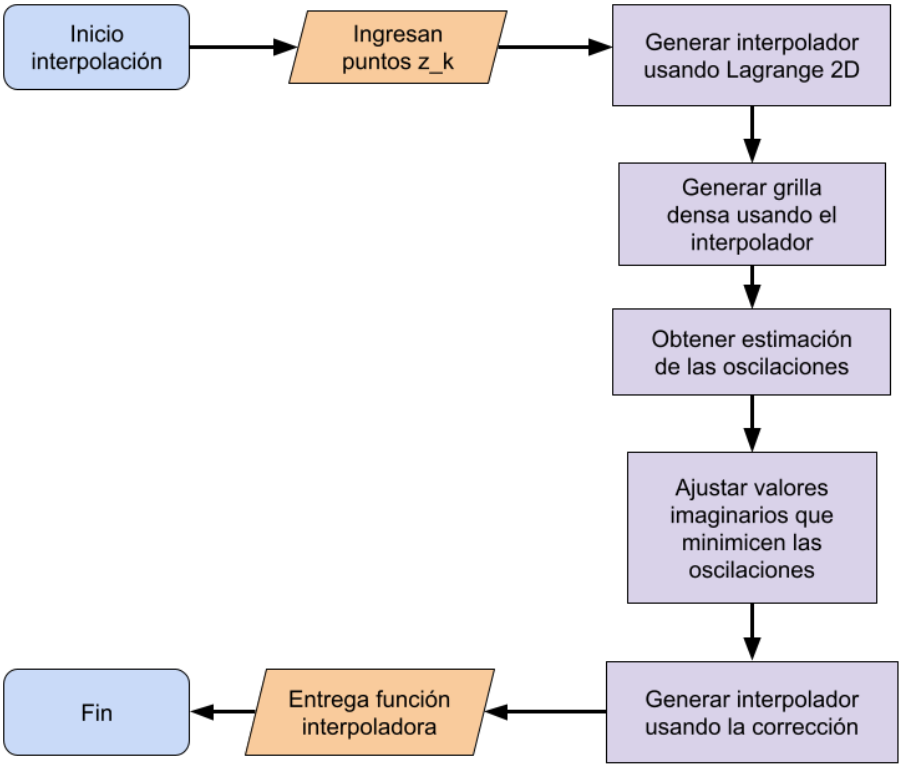


Figura 5: Diagrama de flujo de la propuesta de interpolación tipo Lagrange 2D.
Fuente: Elaboración propia.

Por lo tanto, dada esta definición de g se tienen 5 parámetros ϕ_1, ϕ_2, ψ_1, ψ_2, ψ_3; que es necesario optimizar para minimizar las oscilaciones del interpolador.

Por otro lado, dada una función interpoladora es necesario establecer una estimación cuantitativa de sus oscilaciones. La métrica a utilizar será la integral de la norma al cuadrado del gradiente (se utiliza el cuadrado porque es más fácil de calcular y su mínimo se encuentra en el mismo punto), sobre el dominio a interpolar Ω :

$$\int_{\Omega} \| \nabla P(x,y) \|_2^2 \, d\Omega = \int_{\Omega} \left(\left(\frac{\partial P(x,y)}{\partial x} \right)^2 + \left(\frac{\partial P(x,y)}{\partial y} \right)^2 \right) d\Omega$$

En la práctica se obtendrá una aproximación de esta medida generando una serie de puntos con el interpolador en un patrón de grilla regular densa. Se estimarán las derivadas parciales mediante diferencias finitas y la integral a través de cuadratura por regla del rectángulo.

Es importante señalar que la medida de las oscilaciones y la eventual minimización de las mismas es posible sin conocer directamente la función que se busca aproximar, como sería

la función $F(x, y) = x + y$ en el ejemplo de la Figura 4.

Finalmente, para optimizar los parámetros que minimizan las oscilaciones se utiliza el algoritmo BFGS [Wright y Nocedal, 1999] implementado en el método optimize.minimize de SciPy [Virtanen *et al.*, 2020]. El proceso completo de optimización de los parámetros, aplicación de la corrección y generación de la función interpoladora se presenta en el algoritmo 5.

En este, la función **CorrIm** genera la corrección imaginaria a partir de los valores x e y de los puntos a interpolar y los parámetros $arr\_param = (\phi_1, \phi_2, \psi_1, \psi_2, \psi_3)$; es decir, corresponde a la función $\mathrm{Im}(g(c))$ presentada en (9). Mientras que la función **EstOscil** entrega una estimación de las oscilaciones de una función interpoladora dada, los detalles de su funcionamiento se presentan en la función 6.

Función 5: GenerarInterpoladorLagrange2D

Data: X, Y, Z vectores representando los puntos a interpolar;
$x\_min, x\_max, y\_min, y\_max$ límites del dominio de interpolación; n_param número de parámetros usados por la función $g()$ de la corrección imaginaria;
n_pts_grilla número de puntos por lado en la grilla usada para evaluar las oscilaciones.

Result: fun_interp función interpoladora.

begin

global $X\_GLOB \longleftarrow X$
global $Y\_GLOB \longleftarrow Y$
global $Z\_GLOB \longleftarrow Z$
global $MIN\_X\_GLOB \longleftarrow x\_min$
global $MAX\_X\_GLOB \longleftarrow x\_max$
global $MIN\_Y\_GLOB \longleftarrow y\_min$
global $MAX\_Y\_GLOB \longleftarrow y\_max$
global $N\_PTS\_GLOB \longleftarrow n\_pts\_grilla$

$arr\_param \longleftarrow$ np.zeros(n_param)
$min\_out \longleftarrow$ optimize.minimize(**EstOscil**, arr_param)
$arr\_param \longleftarrow min\_out$.x

$Zim \longleftarrow$ **CorrIm**(X, Y, arr_param)
$bar\_lag \longleftarrow$ **BarLagrange2D**$(X, Y,$ np.array$([Z, Zim]))$
$fun\_interp \longleftarrow$ np.vectorize$(bar_lag$.interp$)$

return fun_interp

Finalmente, las funciones np.gradient y **EstInteg** se encargan respectivamete de estimar el valor de las derivadas parciales de la función interpoladora sobre el dominio por medio de diferencias finitas y estimar el valor de la integral de una función dada sobre el dominio por cuadratura.

Función 6: EstimarOscilaciones

Data: arr_param vector de parámetros usados por la función $g()$ de la corrección imaginaria.
Result: fun_interp función interpoladora.
begin

$Zim \longleftarrow$ **CorrIm**$(X\_GLOB, Y\_GLOB, arr\_param)$

$bar\_lag \longleftarrow$ **BarLagrange2D**$(X\_GLOB, Y\_GLOB,$ np.array$([Z_GLOB, Zim]))$
$fun\_interp \longleftarrow$ np.vectorize$(bar_lag$.interp$)$

$step\_x \longleftarrow (MAX\_X\_GLOB - MIN\_X\_GLOB)/N\_PTS\_GLOB$
$step\_y \longleftarrow (MAX\_Y\_GLOB - MIN\_Y\_GLOB)/N\_PTS\_GLOB$
$MG \longleftarrow$ np.mgrid$[MIN_X_GLOB : (MAX_X_GLOB + step_x) :$
$step\_x, MIN\_Y\_GLOB : (MAX\_Y\_GLOB + step\_y) : step\_y]$
$X\_test \longleftarrow MG[0]$
$Y\_test \longleftarrow MG[1]$

$Z\_test \longleftarrow fun\_interp(X\_test$.flatten$(), Y_test$.flatten$())$
$Z\_test\_rs \longleftarrow$ np.reshape$(Z_test, (N_PTS_GLOB + 1, N_PTS_GLOB + 1))$
$dX \longleftarrow$ np.gradient$(Z_test_rs, step_x,$ axis $= 0)$
$dY \longleftarrow$ np.gradient$(Z_test_rs, step_y,$ axis $= 1)$
$sum\_cuad \longleftarrow$ np.power$(dX, 2) +$ np.power$(dY, 2)$
$est\_osc \longleftarrow$ **EstInteg**$(X\_test, Y\_test, sum\_cuad)$

return est_osc

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se mostrarán los resultados de experimentos realizados a componentes del algoritmo de interpolación que permitan dar respaldo empírico a las decisiones de diseño tomadas en el capítulo anterior, además de pruebas que permiten medir el desempeño de la propuesta frente a otras técnicas de interpolación. Se realizarán pruebas para medir el costo computacional, pruebas que tiempo y error para distintas "correcciones imaginarias" y pruebas de error de interpolación para la propuesta final.

- **Pruebas de costo computacional:** Sección 4.1. En estos experimentos se compara el tiempo de ejecución requerido por el algoritmo propuesto inicialmente de Lagrange 2D por parte real contra su forma baricéntrica, para las operaciones de precálculo e interpolación.
    - **Precálculo** Sección 4.1.1.
    - **Interpolación** Sección 4.1.2.
- **Pruebas de corrección imaginaria:** Sección 4.2. En estos experimentos se muestra el efecto que tienen diferentes propuestas de "corrección imaginaria" en el error de interpolación y los tiempos de ejecución.
- **Pruebas de error de interpolación:** Sección 4.3. En esta serie de experimentos se compara el desempeño de la propuesta final con otros algoritmos de interpolación. Se utilizaron 4 distribuciones de puntos expuestas a continuación, para cada una de ellas se compararon los algoritmos utilizando 6 funciones de prueba, tomando como medida los errores absoluto y relativo de interpolación, lo que corresponde a 12 tablas de resultados por cada distribución de puntos.
    - **Puntos en grilla equiespaciada:** Sección 4.3.1.
    - **Puntos de Chebyshev:** Sección 4.3.2.
    - **Puntos de Halton:** Sección 4.3.3.
    - **Puntos que minimizan iterativamente el error:** Sección 4.3.4.

4.1. PRUEBAS DE COSTO COMPUTACIONAL

En esta sección se busca corroborar de modo práctico la superioridad, en términos de tiempo de ejecución, de la representación baricéntrica. Resultado teórico expuesto en las secciones 3.2.2 y 3.3.1.

Se utilizará como base para estas pruebas la función que en [Fasshauer, 2007] es referida como *Franke's test function*, visible en la Figura 6 y definida en el dominio $D = [0, 1]^2$ como:

$$F(x,y) = \frac{3}{4}e^{-\frac{1}{4}((9x-2)^2+(9y-2)^2)} + \frac{3}{4}e^{-\frac{1}{49}(9x+1)^2-\frac{1}{10}(9y+1)^2} + \frac{1}{2}e^{-\frac{1}{4}((9x-7)^2+(9y-3)^2)} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2} \tag{10}$$

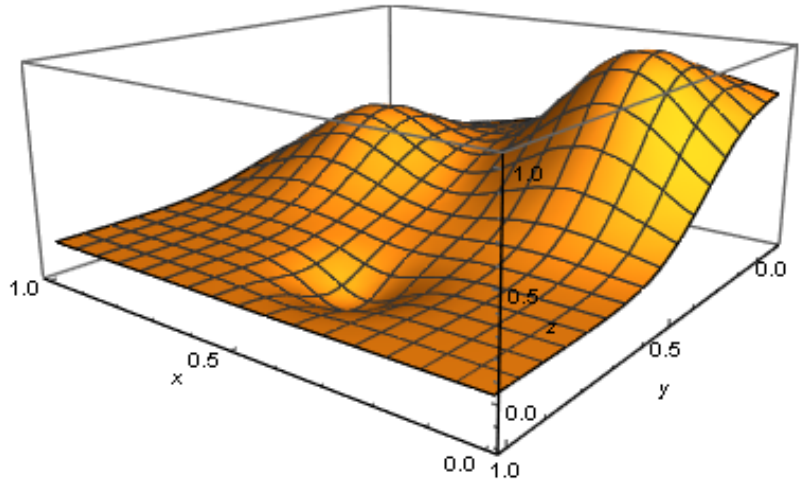


Figura 6: Gráfico *Franke's test function*.
Fuente: Elaboración propia.

A partir de ella se obtendrán desde 5 hasta 50 puntos cuasi-aleatorios utilizando una distribución de Halton [Halton, 1964], los que se interpolarán en cada caso utilizando el algoritmo de parte real de Lagrange complejo (llamado "Propuesta inicial" en los gráficos) y la representación baricéntrica del mismo.

4.1.1. PRECÁLCULO

En el gráfico de la Figura 7 (Gráfico generado utilizando Matplotlib [Hunter, 2007]) se mide el tiempo requerido para generar el interpolador en cada caso. Para cada cantidad de puntos el experimento se repite 50 veces mostrándose el promedio de los tiempos obtenidos en escala logarítmica.

Se puede apreciar que el tiempo de cómputo requerido para la operación de precálculo aumenta a una tasa bastante similar entre ambos algoritmos en la medida que aumenta el número de puntos a interpolar.

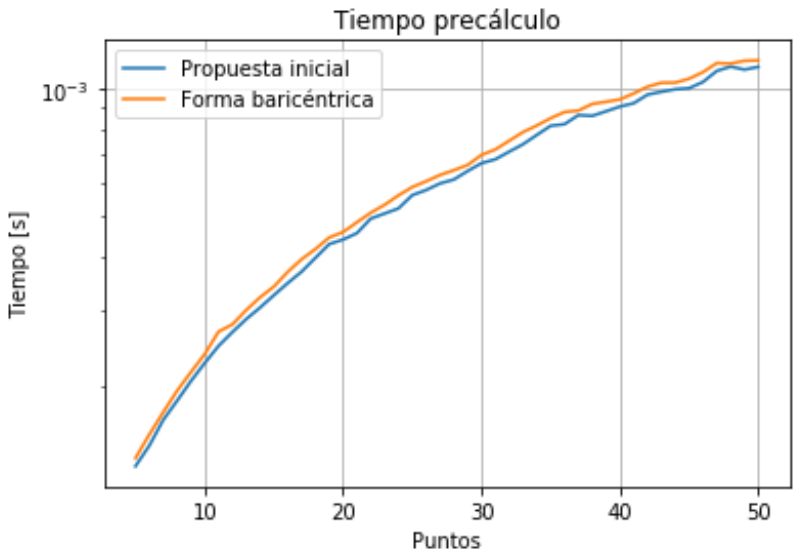


Figura 7: Tiempo utilizado en la operación de precálculo para 5 a 50 puntos de Halton.
Fuente: Elaboración propia.

4.1.2. INTERPOLACIÓN

En el gráfico de la Figura 8, para cada algoritmo, se mide el tiempo requerido para obtener una estimación de un punto dado (x_t, y_t) que no se encuentra en la data, a partir de un interpolador que pasa por N puntos (en otras palabras, corresponde al tiempo necesario para ejecutar $P(x_t, y_t)$ dado que $P(x, y)$ es una función que interpola los N puntos dados). Otra vez el resultado corresponde al promedio de los tiempos obtenidos luego de repetir 50 veces el experimento para cada cantidad de puntos.

Resulta evidente la ventaja del algoritmo en forma baricéntrica en esta operación que es, en la práctica, la que se suele ejecutar con mayor frecuencia.

4.2. PRUEBAS DE CORRECCIÓN IMAGINARIA

En esta sección se busca mostrar el efecto que tiene la corrección imaginaria propuesta en la práctica, para ello se realizarán pruebas de error y de tiempo de ejecución interpolando una función relativamente simple, e^{x+y} (mostrada en la Figura 9) en el dominio $D = [0, 1]^2$.

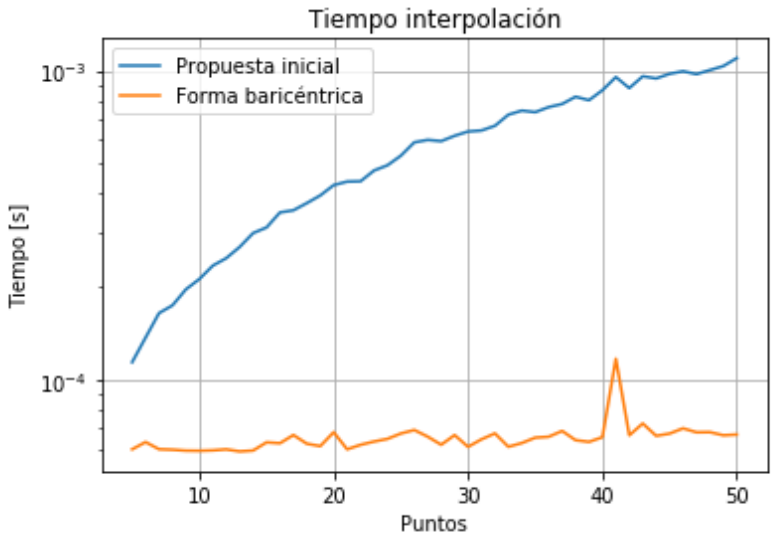


Figura 8: Tiempo utilizado en para interpolar un punto, dado un interpolador generado con 5 a 50 puntos de Halton.

Fuente: Elaboración propia.

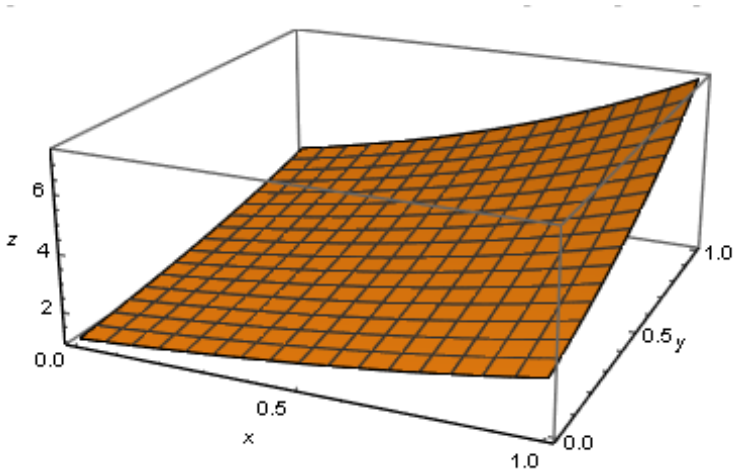


Figura 9: Gráfico de la función e^{x+y} usada en las pruebas.

Fuente: Elaboración propia.

En todos los casos se utilizará el interpolador de Lagrange 2D en forma baricéntrica, sin embargo, se considerarán 3 posibles casos para la corrección imaginaria. No utilizar corrección, utilizar una corrección a través de un polinomio complejo de grado 2 como se propuso en la ecuación (9) y utilizar una corrección a través de un polinomio complejo de grado 1:

$$\text{con } c = x + iy,$$
$$g(c) = \phi c + i\psi c$$
$$\iff \text{Im}(g(c)) = \psi x + \phi y$$

Para la interpolación se utilizarán de 2 a 15 puntos de Halton. Se obtendrá el promedio del error absoluto calculado con una grilla de 25×25 puntos equiespaciados y el tiempo de ejecución considerado corresponderá al proceso completo: Generación de la corrección imaginaria, generación del interpolador (precálculo) y obtención de las estimaciones para cada uno de los 25×25 puntos. En ambos casos los resultados se dan en escala logarítmica.

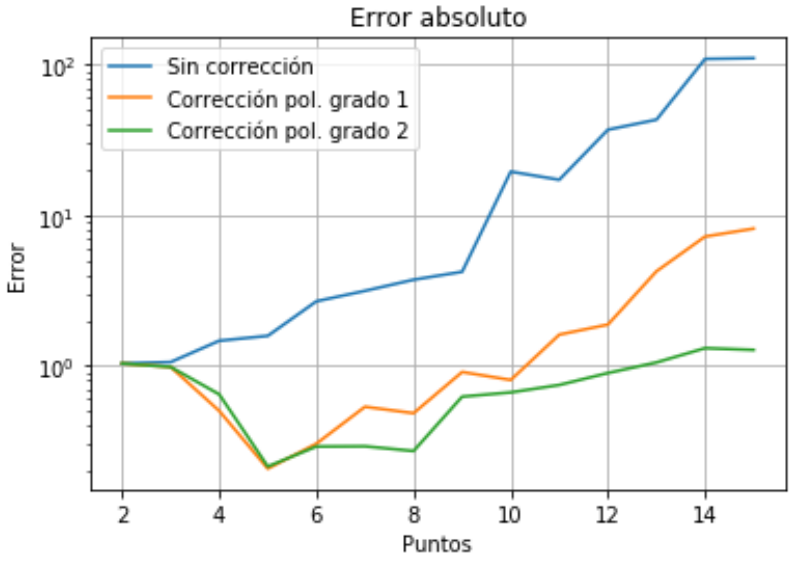


Figura 10: Promedio de error absoluto usando diferentes correcciones.
Fuente: Elaboración propia.

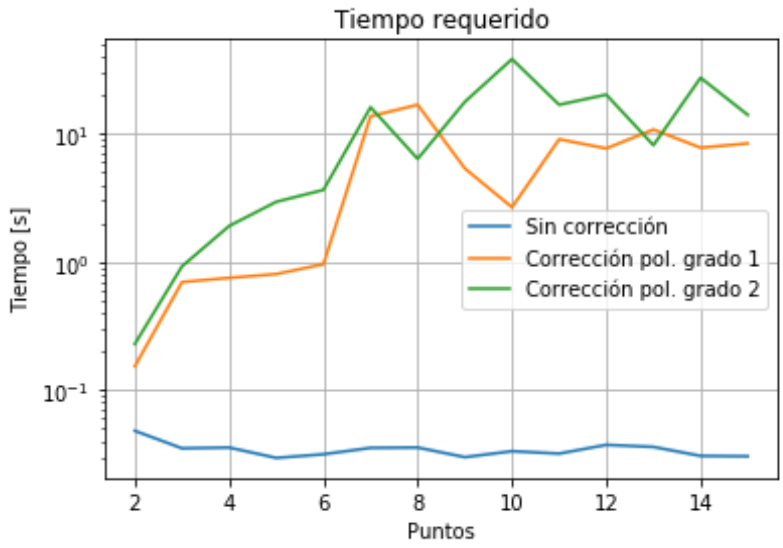


Figura 11: Tiempo de precálculo e interpolación usando diferentes correcciones.
Fuente: Elaboración propia.

En los resultados se puede apreciar el efecto que tienen las oscilaciones en el error de interpolación: Si no se utiliza ninguna corrección este tiende a aumentar rápidamente en la medida que aumentan los puntos. Al aplicar algún tipo de corrección se puede palear este efecto en cierta medida pero no completamente, dado que el error tiende a ir en aumento aproximadamente desde los 5 y los 8 puntos para las correcciones con polinomios de grado 1 y 2 respectivamente.

En general el error de interpolación es menor mientras mayor es el grado del polinomio utilizado en la corrección y se esperarían resultados mejores para correcciones generadas con polinomios de grado más alto, sin embargo, esto significa que existen más parámetros que es necesario optimizar al momento de generar el interpolador y como se puede ver en gráfico 11 implican tiempos de cómputo mayores, siendo necesario un compromiso entre la calidad del interpolador y el tiempo de ejecución requerido.

4.3. PRUEBAS DE ERROR DE INTERPOLACIÓN

El objetivo de estas pruebas es comparar en diferentes escenarios el desempeño del algoritmo propuesto de Lagrange 2D con otros métodos de interpolación. Se utilizarán como medidas los promedios de error absoluto y error relativo, los que se calcularán para cada interpolador generando una “grilla densa” de 25×25 puntos sobre el dominio, para cada uno de los cuales serán obtenidos ambos tipos de error y luego promediados. Dada la alta variabilidad en el orden de magnitud de los resultados, se usa escala logarítmica en base 10.

En el caso del error relativo, para evitar una posible división por 0, este se define como $e_{\text{rel}} = \dfrac{e_{\text{abs}}}{|z| + \epsilon}$ con e_{abs} error absoluto, z valor esperado en el punto y un valor pequeño $\epsilon > 0$. Para las pruebas se utilizará $\epsilon = 10^{-10}$.

Las evaluaciones se realizarán interpolando puntos que sigan un patrón de grilla equiespaciada, puntos de Chebyshev, puntos de Halton y puntos generados de forma iterativa de modo que minimicen el error. Los métodos de interpolación a comparar serán, además de la propuesta de de Lagrange 2D, interpolación por vecino más cercano (NNI), RBF *multiquadric* y gaussiana, y Lagrange en grilla solo para las pruebas en grilla equiespaciada y puntos de Chebyshev. Para el parámetro ε de las RBF se usa el promedio de la distancia entre los puntos a interpolar.

Las funciones de prueba que se utilizarán, a partir de las cuales se obtendrán los puntos a interpolar, serán 6. La primera de ellas es *Franke's test function* definida previamente en la ecuación (10); el resto corresponden a funciones de prueba presentadas en [Mühlenstädt y Kuhnt, 2009] y se definen a continuación:

- Función 2: Definida en la ecuación (11) en el dominio $D = [0{,}05, 0{,}2] \times [5, 30]$ y visible en la Figura 12

$$F(x, y) = 0{,}9996(1090{,}91 + 4xy)e^{x\pi/2} \tag{11}$$

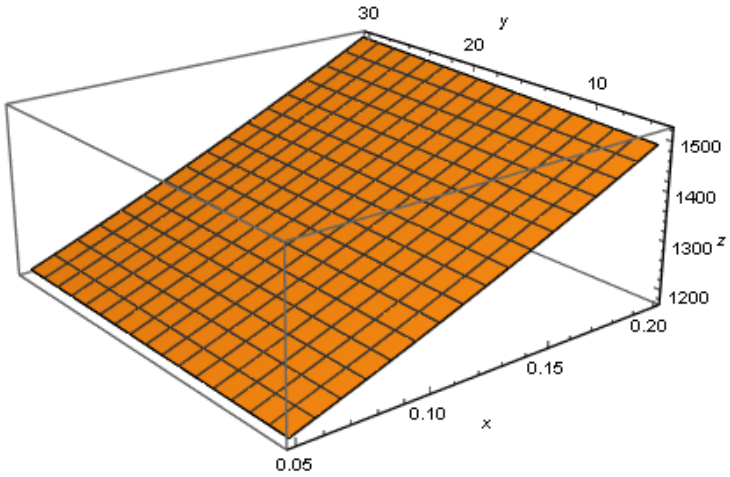


Figura 12: Gráfico función 2 de prueba.
Fuente: Elaboración propia.

- Función 3: Definida en la ecuación (12) en el dominio $D = [-5, 5]^2$, visible en la Figura 13

$$F(x, y) = 1{,}0316 + 4x^2 - 2{,}1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4 \tag{12}$$

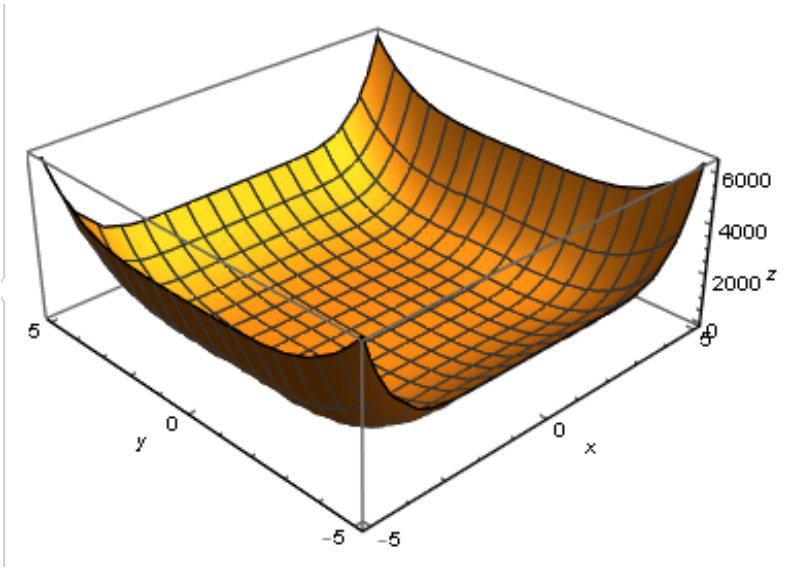


Figura 13: Gráfico función 3 de prueba.
Fuente: Elaboración propia.

- Función 4: Definida en la ecuación (13) en el dominio $D = [-2, 2]^2$, visible en la Figura 14

$$F(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2} \quad (13)$$

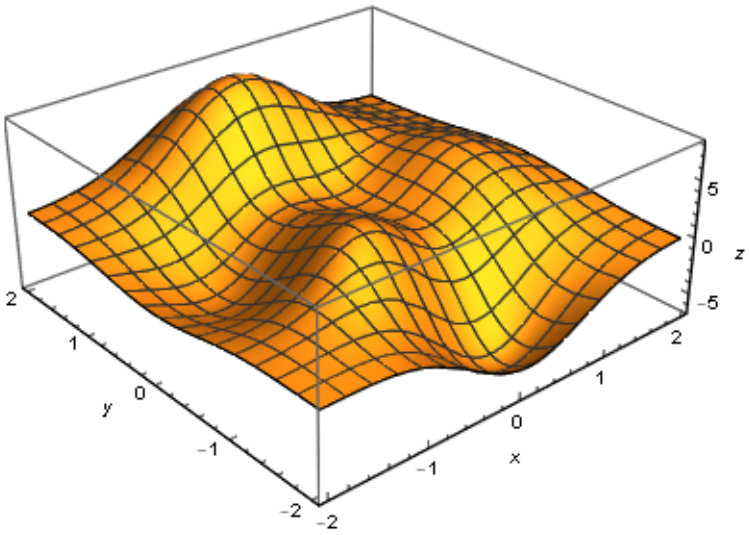


Figura 14: Gráfico función 4 de prueba.
Fuente: Elaboración propia.

- Función 5: Definida en la ecuación (14) en el dominio $D = [0, 1]^2$, visible en la Figura 15

$$F(x, y) = |x^2 + \text{sin}(0{,}5\pi y) - y| \tag{14}$$

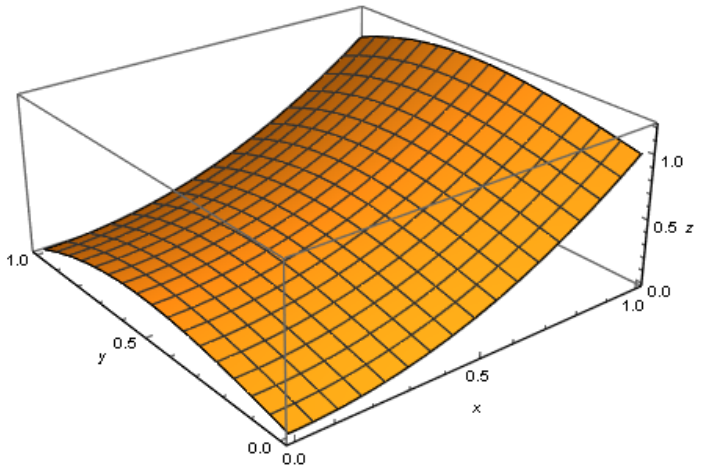


Figura 15: Gráfico función 5 de prueba.
Fuente: Elaboración propia.

- Función 6: Definida en la ecuación (15) en el dominio $D = [0, 1]^2$, visible en la Figura 16

$$F(x, y) = \text{cos}\left(4\pi\sqrt{(x - 0{,}25)^2 + (y - 0{,}25)^2}\right) \tag{15}$$

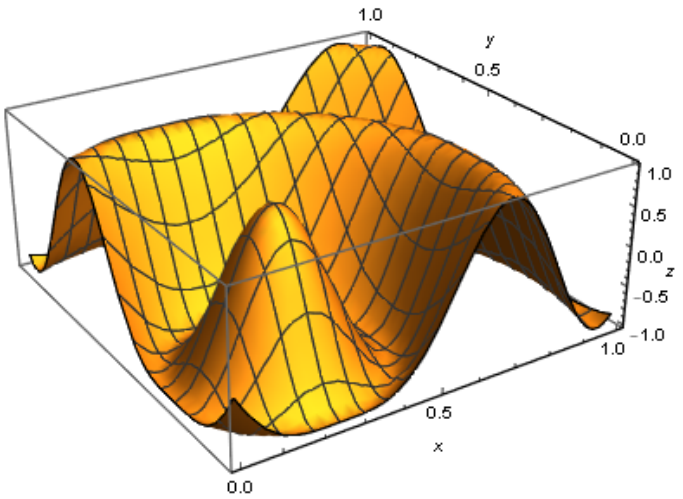


Figura 16: Gráfico función 6 de prueba.
Fuente: Elaboración propia.

4.3.1. PUNTOS EN GRILLA EQUIESPACIADA

Primero se realizan pruebas de error utilizando puntos con valores (x, y) generados a partir de una grilla equiespaciada de $n \times n$ sobre el dominio con $n \in \{2, ..,7\}$.

Función 1:

Tabla 1: Función 1, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.816	-0.886	-0.908	-0.914	-0.846
3×3	-0.815	-0.915	-0.932	-0.966	-0.912
4×4	-0.654	-1.126	-1.163	-1.22	-1.074
5×5	1.117	-1.229	-1.714	-1.738	-1.166
6×6	1.603	-1.393	-1.942	-1.79	-1.265
7×7	6.084	-1.703	-2.193	-1.947	-1.296

Tabla 2: Función 1, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	2.016	1.283	1.406	0.676	-0.053
3×3	1.945	0.959	1.171	0.588	0.446
4×4	2.911	1.447	1.163	0.617	0.262
5×5	4.956	1.521	0.845	0.414	0.146
6×6	5.69	1.469	0.532	0.222	-0.001
7×7	10.049	1.224	0.049	-0.109	-0.144

Para la función 1 los métodos que obtuvieron los mejores resultados fueron las 2 formas de RBF. Para cantidades bajas de puntos Lagrange 2D obtiene resultados inferiores pero similares a los otros métodos, sin embargo, su desempeño empeora a partir de los 4×4 puntos.

Función 2:

Tabla 3: Función 2, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.262	-15.566	-0.331	0.291	0.024
3×3	-0.35	-15.709	-0.392	0.279	-0.298
4×4	-0.515	-15.513	-0.481	0.263	-0.476
5×5	0.935	-15.503	-0.517	0.243	-0.593
6×6	1.779	-15.425	-0.566	0.269	-0.701
7×7	3.835	-15.309	-0.609	0.197	-0.764

Tabla 4: Función 2, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.423	-15.901	-0.59	-0.048	-0.31
3×3	-0.57	-16.008	-0.65	-0.061	-0.573
4×4	-0.658	-15.827	-0.74	-0.077	-0.738
5×5	0.698	-15.782	-0.776	-0.097	-0.847
6×6	1.712	-15.693	-0.825	-0.07	-0.958
7×7	3.643	-15.622	-0.869	-0.143	-1.013

En la función 2 los resultados de Lagrange 2D son iguales o mejores que los de RBF y NNI hasta antes de los 5×5 puntos, en este caso destaca por sobre todos los excelentes resultados obtenidos por Lagrange en grilla.

Función 3:

Tabla 5: Función 3, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	3.714	3.714	3.647	3.561	3.714
3×3	3.183	3.038	3.113	3.3	3.327
4×4	3.203	2.882	2.953	3.11	3.076
5×5	4.037	2.179	2.636	2.898	2.911
6×6	4.948	2.067	2.51	2.76	2.796
7×7	8.304	-12.602	2.276	2.61	2.718

Tabla 6: Función 3, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	2.704	2.704	2.62	2.449	2.704
3×3	0.633	1.26	1.354	1.568	0.835
4×4	0.78	1.651	1.714	1.748	0.474
5×5	0.686	0.305	0.957	1.136	0.292
6×6	1.382	0.707	1.136	1.157	0.125
7×7	4.655	-15.113	0.387	0.573	0.065

La función 3 presenta valores altos en el dominio lo que explica el alto error absoluto. En términos de error relativo Lagrange 2D obtiene mejores resultados que las RBF hasta antes de los 6×6 puntos, se desempeña mejor que Lagrange en grilla hasta antes de los 5×5 puntos y mejor que NNI antes de los 4×4 puntos.

Llama la atención que para 7×7 puntos se ve una drástica caída del error para Lagrange en grilla, esto se debe a que la función a interpolar corresponde a un polinomio de dos variables de grado 6, mismo grado de los polinomios L_i, L_j de este interpolador cuando se tienen 7 puntos por lado.

Función 4:

Tabla 7: Función 4, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	0.291	0.29	0.29	0.29	0.289
3×3	0.121	0.125	0.08	0.02	0.029
4×4	0.595	0.21	0.076	0.035	0.102
5×5	1.113	-0.011	-0.087	-0.116	0.031
6×6	3.478	-0.146	-0.531	-0.531	-0.07
7×7	6.409	-0.342	-0.742	-0.775	-0.133

Tabla 8: Función 4, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	0.004	0.005	0.004	0.003	0.024
3×3	0.541	0.621	0.576	0.487	0.368
4×4	1.323	0.501	0.263	0.205	0.321
5×5	2.105	0.499	0.3	0.236	0.289
6×6	4.514	0.243	-0.215	-0.27	0.073
7×7	7.433	0.073	-0.426	-0.397	0.121

Para la función 4 Lagrange 2D obtiene resultados similares a Lagrange en grilla y NNI hasta antes de los 4×4 puntos, donde estos últimos lo superan. Las RBF en general obtienen mejores resultados en esta función.

Función 5:

Tabla 9: Función 5, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.886	-1.05	-1.039	-0.878	-0.599
3×3	-1.268	-1.884	-1.633	-1.429	-0.89
4×4	-1.523	-2.957	-1.78	-1.483	-1.05
5×5	-0.190	-4.101	-2.117	-1.604	-1.172
6×6	0.666	-5.296	-2.273	-1.684	-1.267
7×7	4.563	-6.559	-2.479	-1.773	-1.333

Tabla 10: Función 5, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.282	-0.305	-0.353	-0.388	-0.114
3×3	-0.451	-1.308	-1.059	-0.922	-0.427
4×4	-0.458	-2.343	-1.16	-1.057	-0.577
5×5	1.618	-3.43	-1.509	-1.253	-0.688
6×6	2.695	-4.597	-1.635	-1.339	-0.771
7×7	6.535	-5.807	-1.77	-1.454	-0.827

En la función 5 Lagrange 2D tiene un desempeño superior a NNI y comparable al de RBF gaussiana hasta antes de los 5×5 puntos. Los mejores resultados en esta prueba fueron obtenidos por Lagrange en grilla, seguido de RBF *multiquadric*.

Función 6:

Tabla 11: Función 6, $\log_{10}$ de error absoluto para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.200	-0.191	-0.181	-0.175	-0.082
3×3	-0.217	-0.208	-0.19	-0.199	-0.218
4×4	-0.080	-0.274	-0.294	-0.305	-0.281
5×5	1.529	-0.253	-0.603	-0.723	-0.4
6×6	2.895	-0.499	-0.661	-0.703	-0.468
7×7	6.771	-0.789	-1.045	-1.053	-0.494

Tabla 12: Función 6, $\log_{10}$ de error relativo para puntos en grilla regular.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	7.759	7.773	7.739	7.733	7.982
3×3	7.523	7.754	7.808	7.823	7.792
4×4	7.902	7.805	7.872	7.885	7.996
5×5	8.310	8.111	7.503	7.224	8.181
6×6	9.247	7.844	7.556	7.489	7.89
7×7	11.289	7.355	7.027	7.072	7.83

Finalmente en la función 6, habiendo varios valores cercanos a 0, se tienen en general valores altos de error relativo para todos los interpoladores. El desempeño de todos los métodos es bastante similar en términos de error absoluto hasta antes de los 4×4 puntos, a partir de donde Lagrange 2D obtiene peores resultados.

En general se puede decir que Lagrange 2D obtiene buenos resultados en términos de error para cantidades bajas de puntos. En todos los casos se puede ver que para cierta cantidad de puntos a interpolar en adelante, el desempeño del interpolador de hecho empeora debido a la alta amplitud de las oscilaciones generadas. La cantidad de puntos a partir de la cual se produce este fenómeno varía en cada caso aunque oscila entre los 3×3 y los 5×5 puntos.

4.3.2. PUNTOS DE CHEBYSHEV

Se utilizarán puntos de Chebyshev en una grilla de $n \times n$ con $n \in \{2, ..,7\}$, donde cada componente x e y corresponde a los nodos de Chebyshev a lo largo del dominio del respectivo eje.

Función 1:

Tabla 13: Función 1, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.816	-0.886	-0.908	-0.914	-0.881
3×3	-0.815	-0.915	-0.932	-0.966	-0.871
4×4	-0.618	-0.943	-0.975	-1.388	-0.912
5×5	-0.457	-1.357	-1.548	-1.122	-1.08
6×6	-0.394	-1.423	-1.491	-1.366	-1.118
7×7	1.356	-1.779	-1.886	-1.494	-1.22

Tabla 14: Función 1, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	2.016	1.283	1.406	0.676	-0.093
3×3	1.945	0.959	1.171	0.588	0.352
4×4	2.546	1.313	0.981	0.175	0.041
5×5	3.111	1.013	0.75	0.514	0.57
6×6	3.460	0.679	0.354	0.126	-0.12
7×7	5.119	0.627	0.38	-0.029	0.11

En la función 1 el desempeño de Lagrange 2D es comparable al del resto de los métodos hasta antes de los 4×4 puntos. En general RBF obtiene mejores resultados en esta prueba por un estrecho margen.

Función 2:

Tabla 15: Función 2, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.262	-15.538	-0.331	0.291	0.034
3×3	-0.35	-15.663	-0.392	0.279	-0.287
4×4	-0.442	-15.437	-0.474	0.263	-0.408
5×5	-0.689	-15.339	-0.496	0.271	-0.529
6×6	1.223	-15.309	-0.541	0.271	-0.615
7×7	1.872	-15.24	-0.565	0.238	-0.69

Tabla 16: Función 2, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.424	-15.874	-0.59	-0.048	-0.309
3×3	-0.57	-15.981	-0.65	-0.061	-0.572
4×4	-0.686	-15.753	-0.727	-0.077	-0.709
5×5	-0.912	-15.644	-0.745	-0.069	-0.822
6×6	0.913	-15.652	-0.788	-0.069	-0.905
7×7	1.603	-15.586	-0.812	-0.102	-0.977

Para la función 2 Lagrange 2D obtiene resultados similares a RBF *multiquadric* y NNI, y superiores a RBF gaussiana hasta antes de los 6×6 puntos. Otra vez, Lagrange en grilla presenta el mejor desempeño con errores mucho menores al resto de los algoritmos.

Función 3:

Tabla 17: Función 3, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	3.714	3.714	3.647	3.561	3.714
3×3	3.183	3.038	3.113	3.3	3.327
4×4	3.098	2.915	2.963	3.03	3.003
5×5	3.048	2.133	2.587	2.828	2.671
6×6	3.214	2.191	2.467	2.679	2.7
7×7	4.195	-12.509	2.035	2.52	2.46

Tabla 18: Función 3, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	2.704	2.704	2.62	2.449	2.704
3×3	0.633	1.26	1.354	1.568	0.835
4×4	1.395	2.111	2.121	1.857	1.106
5×5	0.324	0.734	1.316	1.395	0.475
6×6	0.21	1.272	1.415	1.307	0.312
7×7	0.797	-14.09	0.552	1.021	0.139

En la función 3, como se explicó anteriormente, se tienen valores altos de error absoluto en general. Es interesante que por un lado Lagrange 2D obtiene resultados similares o peores en términos de error absoluto a los otros métodos, pero en general superiores desde el punto de vista del error relativo.

Función 4:

Tabla 19: Función 4, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	0.291	0.29	0.29	0.29	0.288
3×3	0.121	0.125	0.08	0.02	0.048
4×4	0.326	0.148	0.105	0.13	0.16
5×5	0.396	0.066	0.031	-0.149	0.076
6×6	0.98	-0.245	-0.347	-0.176	-0.014
7×7	1.375	-0.378	-0.451	-0.261	-0.042

Tabla 20: Función 4, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	0.004	0.005	0.004	0.003	0.021
3×3	0.541	0.621	0.576	0.487	0.405
4×4	0.838	0.392	0.319	0.168	0.37
5×5	0.939	0.581	0.523	0.289	0.407
6×6	1.886	0.056	-0.028	-0.032	0.21
7×7	2.396	0.126	0.055	0.026	0.24

En la función 4 tanto Lagrange en grilla como los algoritmos de RBF y NNI obtienen resultados bastante similares. Lagrange 2D también obtiene resultados comparables hasta antes de los 4×4 puntos a partir de donde estos empeoran.

Función 5:

Tabla 21: Función 5, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.886	-1.05	-1.039	-0.878	-0.583
3×3	-1.268	-1.884	-1.633	-1.429	-0.868
4×4	-1.387	-2.891	-1.855	-0.903	-1.007
5×5	-1.538	-3.991	-2.331	-1.036	-1.116
6×6	-1.509	-5.172	-2.466	-1.189	-1.202
7×7	0.031	-6.432	-2.779	-1.313	-1.278

Tabla 22: Función 5, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.282	-0.305	-0.353	-0.388	-0.098
3×3	-0.451	-1.308	-1.059	-0.922	-0.4
4×4	-0.551	-2.344	-1.285	-0.52	-0.526
5×5	-0.101	-3.447	-1.712	-0.735	-0.635
6×6	-0.049	-4.641	-1.871	-0.896	-0.723
7×7	1.447	-5.897	-2.164	-0.986	-0.796

Para la función 5 el desempeño de Lagrange 2D fue similar a RBF gaussiana y NNI hasta antes de los 7×7 puntos, siendo superado en primer lugar por Lagrange en grilla y segundo por RBF *multiquadric*.

Función 6:

Tabla 23: Función 6, $\log_{10}$ de error absoluto para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	-0.200	-0.191	-0.181	-0.175	-0.094
3×3	-0.217	-0.208	-0.19	-0.199	-0.185
4×4	0.110	-0.165	-0.178	-0.248	-0.202
5×5	-0.040	-0.359	-0.41	-0.449	-0.274
6×6	0.263	-0.418	-0.417	-0.485	-0.36
7×7	1.790	-0.954	-1.132	-0.866	-0.467

Tabla 24: Función 6, $\log_{10}$ de error relativo para puntos de Chebyshev.
Fuente: Elaboración Propia.

	Lagrange 2D	Lagrange en grilla	RBF multiquadric	RBF gaussiana	NNI
2×2	7.759	7.773	7.739	7.733	7.982
3×3	7.523	7.754	7.808	7.823	7.911
4×4	8.137	8.101	8.081	8.004	8.192
5×5	7.699	7.906	7.846	7.687	7.937
6×6	7.988	7.739	7.739	7.575	7.872
7×7	7.979	7.486	7.133	6.296	8.121

Finalmente en la función 6, como se explicó anteriormente, se tienen en general valores altos de error relativo. Para el error absoluto Lagrange 2D obtiene resultados similares a los demás métodos hasta antes de los 4×4 puntos. El resto de los algoritmos obtiene resultados similares entre sí, teniendo un mejor desempeño las RBF por un pequeño margen.

Los resultados para Lagrange 2D son similares a los obtenidos en la serie de pruebas anteriores: Para cierta cantidad de puntos en adelante, la cual varía dependiendo de la función de la cual provienen los puntos a interpolar, el desempeño del algoritmo en términos del error empeora. Una diferencia que se puede notar, sin embargo, es que el uso de puntos de Chebyshev de hecho resulta beneficioso para Lagrange 2D a diferencia de los demás algoritmos de interpolación en las pruebas presentadas.

En general el desempeño de Lagrange en grilla, los algoritmos de RBF y NNI fue ligeramente inferior en la serie de pruebas usando puntos de Chebyshev a aquel sobre una grilla equiespaciada; esto se explica por el hecho de que los puntos de Chebyshev tienen a concentrarse en los bordes, teniéndose menos “información” de la parte central de la función. La principal ventaja de los puntos de Chebyshev es minimizar el fenómeno de Runge, pero este no se presenta en RBF ni NNI y probablemente la cantidad de puntos por lado sea demasiado baja (un máximo de 7 puntos por lado) como para ser un problema en Lagrange en grilla.

Para Lagrange 2D, por otro lado, el uso de puntos de Chebyshev de hecho produce mejores resultados, siendo el interpolador efectivo para intervalos mayores de puntos y obteniéndose valores menores de error. Es probable que los puntos de Chebyshev reduzcan la incidencia de las oscilaciones presentadas en la sección 3.4 de modo similar a como reducen las oscilaciones del fenómeno de Runge en Lagrange 1D.

4.3.3. PUNTOS DE HALTON

Como se discutió al principio del documento, una de las principales características buscadas en el interpolador presentado de Lagrange 2D es la capacidad de interpolar una distribución de puntos libre de grilla, es por esto que se realiza una serie de pruebas calculando los errores para interpolaciones realizadas con n puntos de Halton, para $n \in \{2, .., 30\}$.

Función 1:

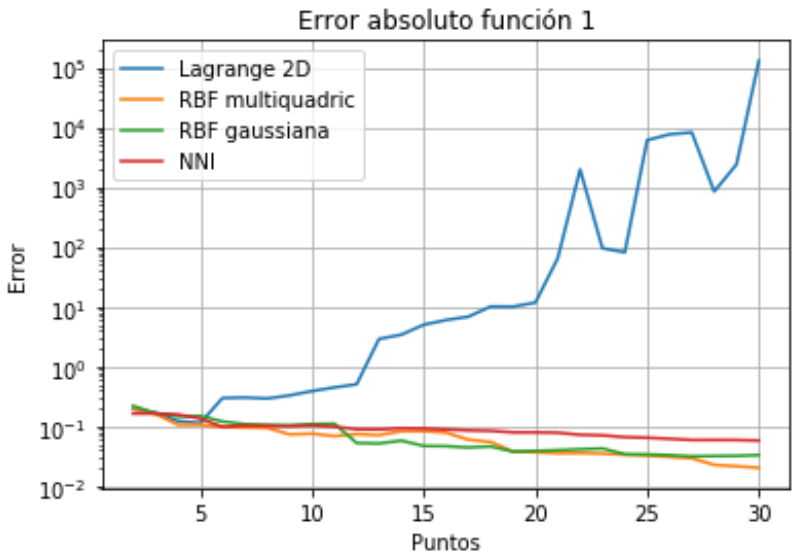


Figura 17: Gráfico *Error absoluto en función 1 para puntos de Halton*.
Fuente: Elaboración propia.

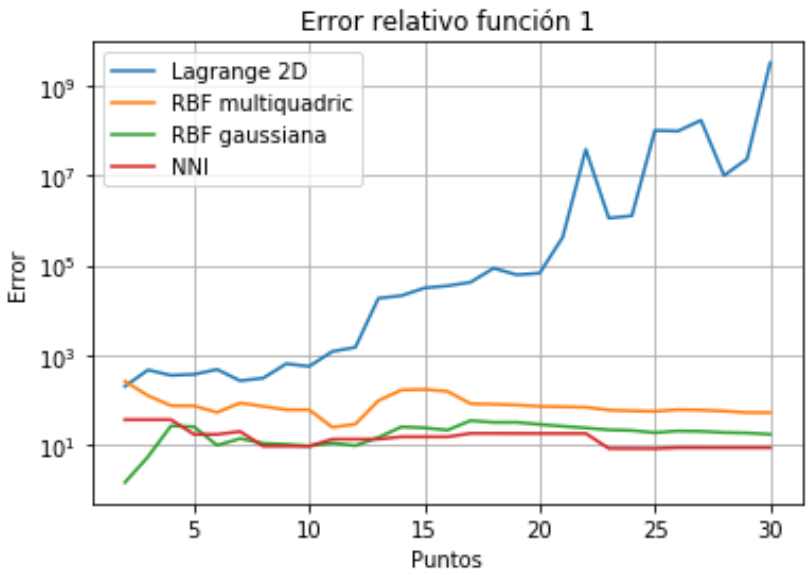


Figura 18: Gráfico *Error relativo en función 1 para puntos de Halton*.
Fuente: Elaboración propia.

En la función 1 Lagrange 2D tiene un desempeño comparable a los otros algoritmos hasta los 5 puntos, a partir de donde entrega valores más altos de error. Pero es aproximadamente a los 12 puntos donde este se dispara y crece a un ritmo acelerado en la medida en que se agregan puntos.

Función 2:

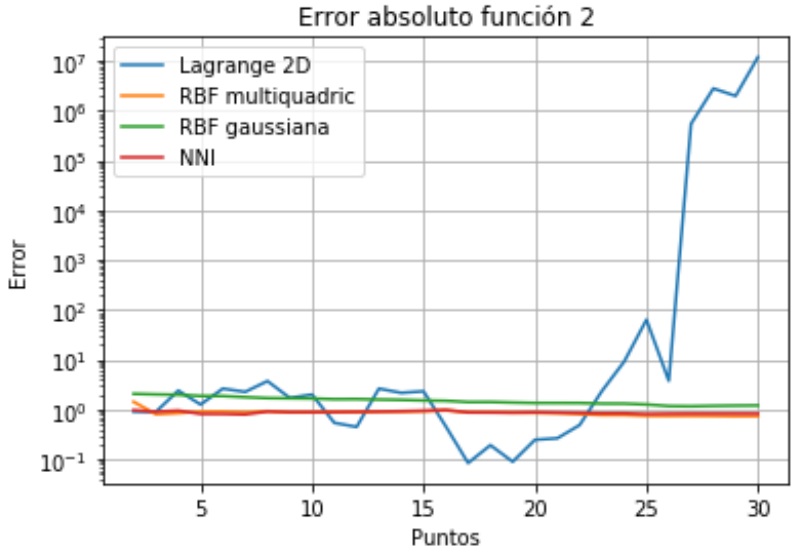


Figura 19: Gráfico *Error absoluto en función 2 para puntos de Halton*.
Fuente: Elaboración propia.

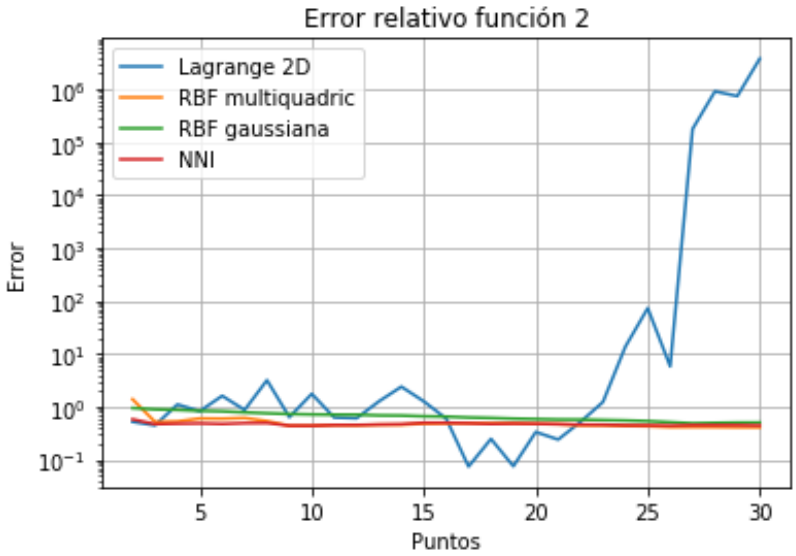


Figura 20: Gráfico *Error relativo en función 2 para puntos de Halton*.
Fuente: Elaboración propia.

Para la función 2 se tiene un mejor desempeño de Lagrange 2D en comparación a la función anterior, llama la atención la alta variabilidad en las medida de ambos tipos de error a diferencia de lo que ocurre con las RBF y NNI. En general los resultados de Lagrange 2D

son similares a los de los otros métodos, siendo marginalmente peor hasta los 15 puntos y obteniendo resultados ligeramente mejores que el resto entre los 15 y 22 puntos; es desde aquí en adelante que el error de interpolación para Lagrange 2D crece a un ritmo acelerado.

Función 3:

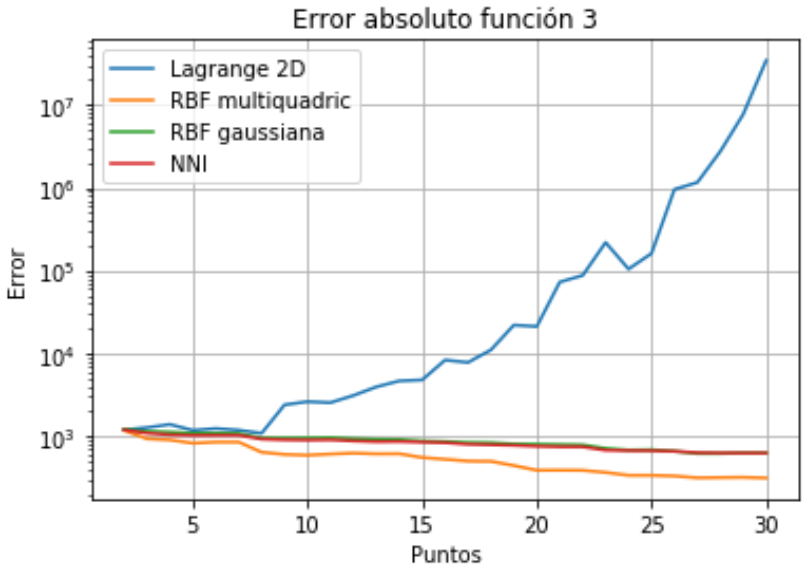


Figura 21: Gráfico *Error absoluto en función 3 para puntos de Halton*.
Fuente: Elaboración propia.

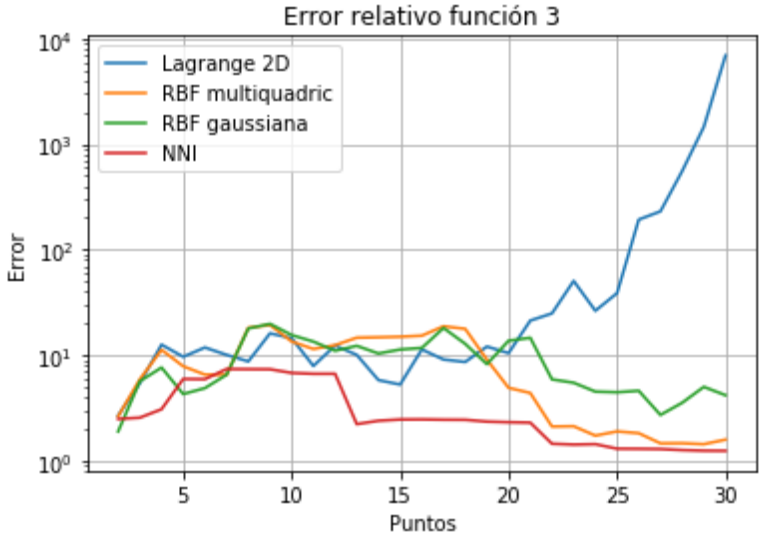


Figura 22: Gráfico *Error relativo en función 3 para puntos de Halton*.
Fuente: Elaboración propia.

En la función 3 el desempeño de Lagrange 2D en términos de error absoluto es similar al del resto de los algoritmos hasta aproximadamente los 8 puntos, sin embargo, para las medidas de error relativo se tiene que el comportamiento de Lagrange 2D es comparable al de las RBF hasta los 20 puntos.

Función 4:

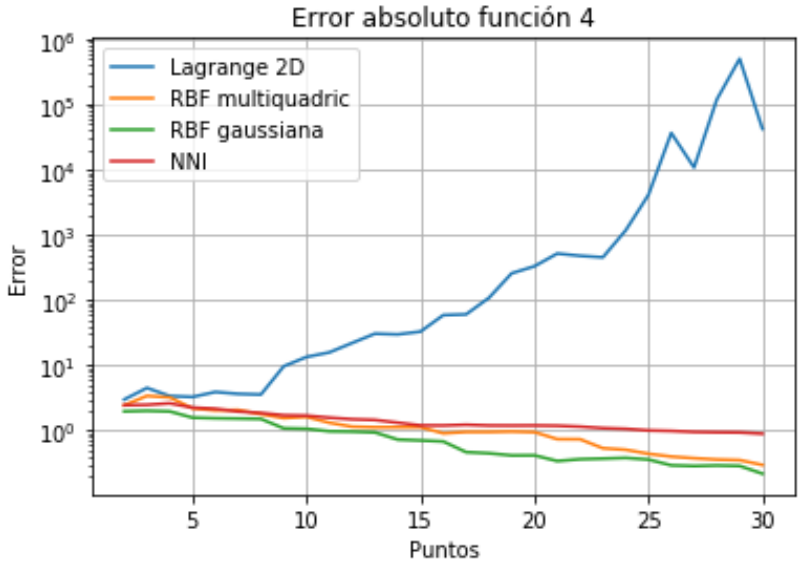


Figura 23: Gráfico *Error absoluto en función 4 para puntos de Halton*.
Fuente: Elaboración propia.

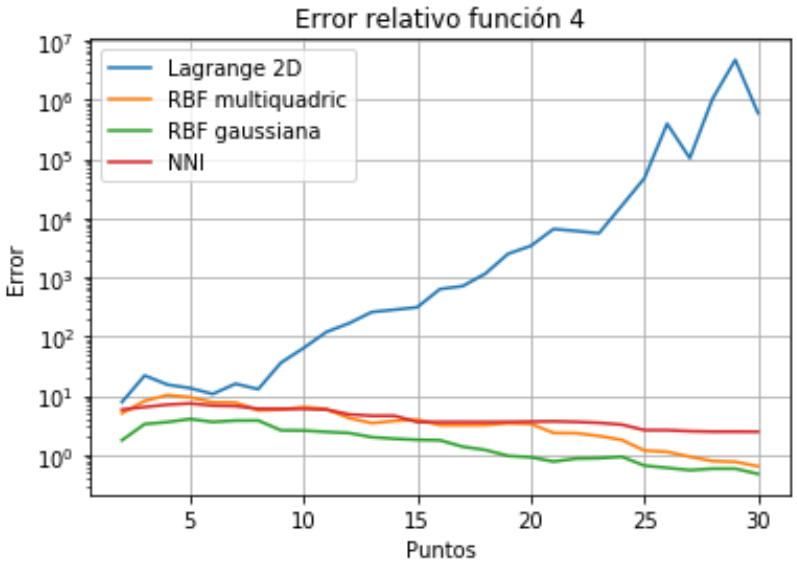


Figura 24: Gráfico *Error relativo en función 4 para puntos de Halton*.
Fuente: Elaboración propia.

En la función 4 los resultados obtenidos de Lagrange 2D fueron similares aunque ligeramente peores al del resto de los interpoladores hasta aproximadamente los 8 puntos, desde donde el error para Lagrange 2D se dispara. Los resultados obtenidos por RBF *multiquadric* y NNI son similares entre sí, y el menor error de interpolación se obtuvo con RBF gaussianas.

Función 5:

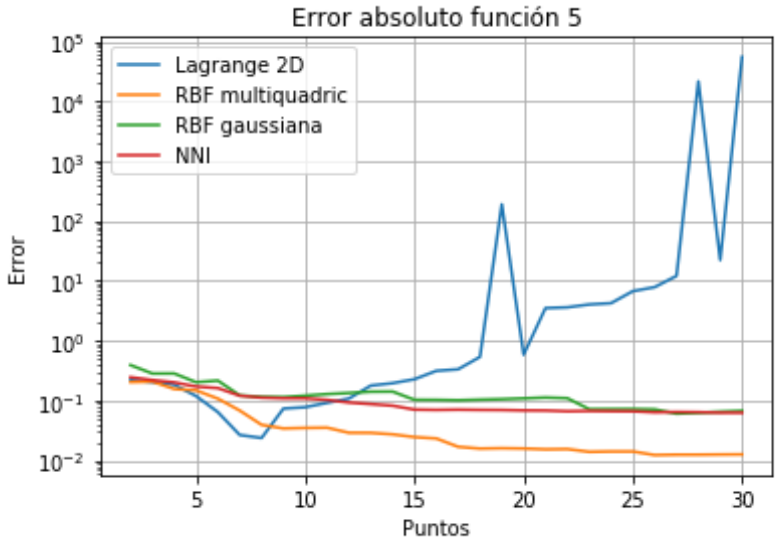


Figura 25: Gráfico *Error absoluto en función 5 para puntos de Halton*.
Fuente: Elaboración propia.

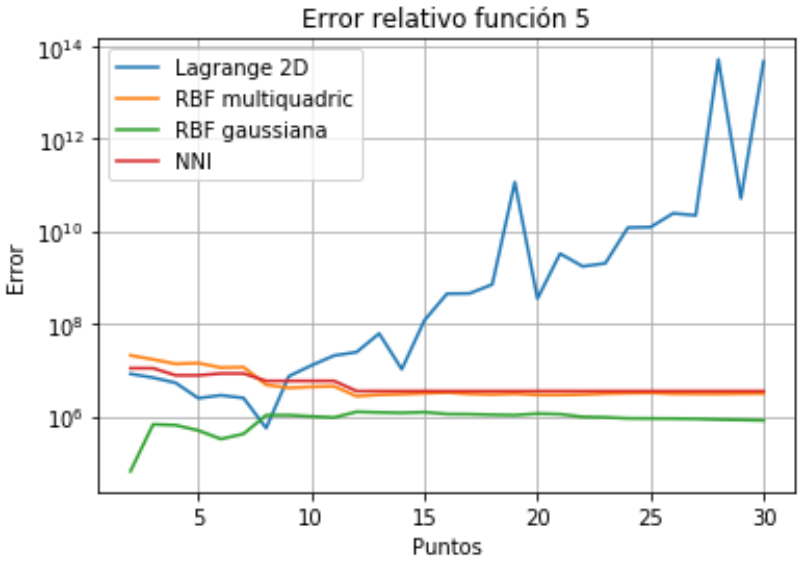


Figura 26: Gráfico *Error relativo en función 5 para puntos de Halton*.
Fuente: Elaboración propia.

En la función 5 el comportamiento de Lagrange 2D en términos de error absoluto es en principio comparable a RBF *multiquadric* y mejor RBF gaussiana y NNI; para el error relativo los

resultados de Lagrange 2D son en principio mejores que RBF *multiquadric* y NNI, aunque es superado por RBF gaussiana. En ambos casos el desempeño de Lagrange 2D empeora a partir de los 8 puntos.

Función 6:

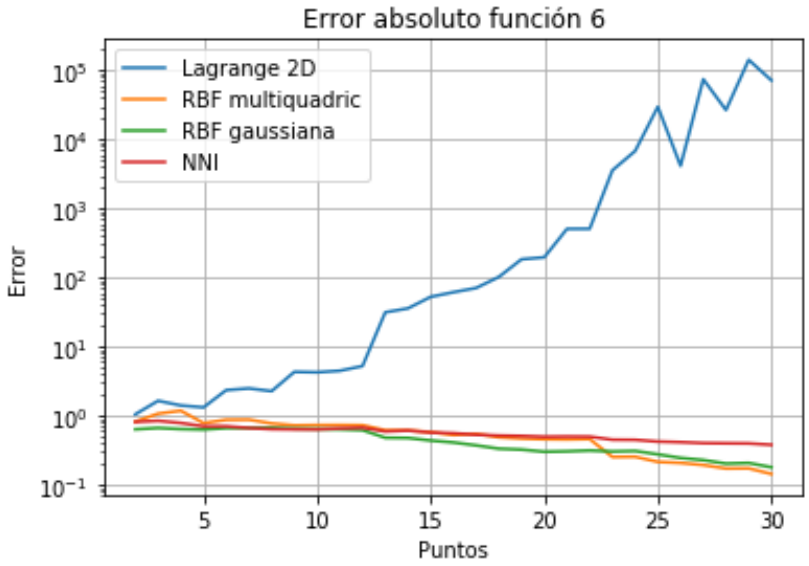


Figura 27: Gráfico *Error absoluto en función 6 para puntos de Halton*.
Fuente: Elaboración propia.

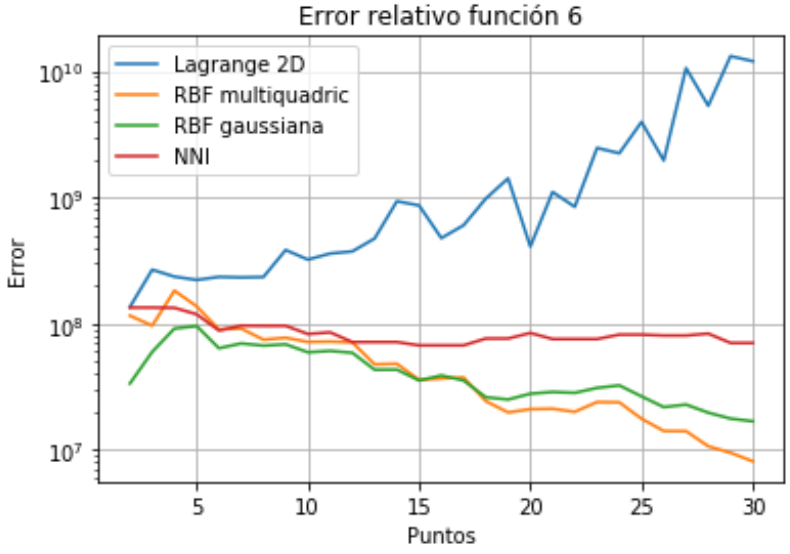


Figura 28: Gráfico *Error relativo en función 6 para puntos de Halton*.
Fuente: Elaboración propia.

Finalmente en la función 6 el desempeño de Lagrange 2D fue ligeramente peor al del resto de los métodos inicialmente, empeorando luego de forma considerable a partir de los 5 puntos.

Para la mayoría de las pruebas, en un principio el desempeño de Lagrange 2D no es muy diferente al de los otros métodos, llegando a ser superior a algunos de estos para algunas funciones; sin embargo, se puede ver con mayor claridad ahora el la tendencia a aumentar el error que ocurre cuando se tiene una cierta cantidad de puntos en adelante. Con algunas diferencias entre casos particulares, este instante a partir del cual el error de interpolación va en rápido aumento en la medida en que se añaden puntos parece encontrarse aproximadamente en los 8 puntos.

4.3.4. PUNTOS QUE MINIMIZAN ITERATIVAMENTE EL ERROR

Para esta prueba se parte con 2 puntos de Halton, en cada iteración se hace una evaluación del error utilizando una “grilla densa” y se agrega el siguiente punto en la posición donde el error absoluto sea máximo. El resultado mostrado corresponde al promedio de errores absoluto y relativo en cada iteración.

Función 1:

Tabla 25: Función 1, $\log_{10}$ de error absoluto para puntos reduciendo el error. Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	-0.709	-0.696	-0.656	-0.784
3	-0.497	-0.714	-0.956	-0.873
4	-0.123	-0.563	-1.038	-0.914
5	-0.272	-0.516	-1.256	-0.902
6	-0.326	-0.73	-1.372	-0.908
7	-0.49	-0.882	-1.3	-0.929
8	-0.664	-0.899	-1.332	-0.954
9	-0.467	-1.031	-1.303	-0.96
10	0.529	-1.143	-1.247	-0.965

Tabla 26: Función 1, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	2.306	2.419	0.167	1.564
3	2.88	2.427	-0.024	1.563
4	3.272	2.668	0.908	1.563
5	3.195	2.879	1.121	1.564
6	2.925	2.427	0.701	1.564
7	2.829	2.443	1.767	1.564
8	2.677	2.444	1.829	1.758
9	2.811	1.297	1.868	1.758
10	4.074	1.715	2.083	1.759

En la función 1 el desempeño de Lagrange 2D fue inferior al del resto de los algoritmos, aunque obtuvo resultados razonables hasta los 9 puntos; los mejores resultados en esta prueba fueron obtenidos por las RBF.

Función 2:

Tabla 27: Función 2, $\log_{10}$ de error absoluto para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	-0.041	0.156	0.317	-0.011
3	0.172	-0.103	0.283	0.031
4	-0.086	-0.066	0.269	0.035
5	-0.149	-0.253	0.26	0.032
6	0.682	-0.414	0.255	0.013
7	0.37	-0.318	0.245	-0.025
8	0.559	-0.507	0.231	-0.08
9	0.475	-0.431	0.216	-0.066
10	0.241	-0.585	0.201	-0.105

Tabla 28: Función 2, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	-0.281	0.144	-0.022	-0.234
3	0.146	-0.431	-0.045	-0.174
4	-0.354	-0.35	-0.046	-0.193
5	-0.422	-0.534	-0.046	-0.204
6	0.479	-0.61	-0.042	-0.22
7	0.176	-0.59	-0.043	-0.233
8	0.418	-0.678	-0.047	-0.252
9	0.304	-0.671	-0.051	-0.226
10	0.124	-0.764	-0.057	-0.26

Para la función 2, Lagrange 2D junto con RBF *multiquadric* obtienen los mejores resultados hasta los 5 puntos. Desde este punto en adelante el desempeño de Lagrange 2D empeora notablemente aunque no se distancia demasiado de los resultados obtenidos por RBF gaussiana.

Función 3:

Tabla 29: Función 3, $\log_{10}$ de error absoluto para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	3.079	3.073	3.085	3.078
3	3.343	3.073	3.131	3.136
4	3.338	2.989	3.158	3.253
5	2.986	3.051	3.17	3.147
6	3.241	3.101	3.154	3.199
7	3.326	2.938	3.193	3.208
8	3.42	2.88	3.173	3.155
9	3.158	2.9	3.066	3.075
10	3.321	2.824	2.955	3.016

Tabla 30: Función 3, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	0.428	0.421	0.278	0.396
3	1.606	1.677	1.372	0.439
4	1.537	1.447	1.198	0.602
5	1.252	1.635	1.419	0.548
6	1.463	2.007	1.745	0.578
7	1.822	1.752	2.016	0.582
8	1.651	1.684	1.871	0.581
9	1.099	1.676	1.606	0.568
10	1.513	1.491	1.302	0.557

En la función 3, como en casos anteriores, se puede ver que el error absoluto es bastante alto en general. En cuanto al error relativo tanto Lagrange 2D como las RBF presentaron un considerable aumento en el error al incrementarse el número de puntos; esto también ocurre en parte con NNI aunque no de manera tan pronunciada.

En otras palabras, el desempeño de todos los interpoladores en términos de error relativo fue mejor con solo 2 puntos. Esto se explica por la estructura de la función 3, la que presenta valores altos cerca de los bordes del dominio y valores comparativamente pequeños cerca del centro, y el hecho de que la prueba se realiza colocando iterativamente puntos donde sea máximo el error absoluto. Es probable que el error absoluto sea más alto cerca de los bordes, lugar donde se agregan los puntos y es precisamente donde el error relativo es menor.

Función 4:

Tabla 31: Función 4, $\log_{10}$ de error absoluto para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	0.475	0.387	0.294	0.389
3	0.875	0.61	0.107	0.432
4	0.71	0.377	-0.051	0.466
5	0.781	0.643	0.114	0.393
6	0.826	0.449	-0.126	0.349
7	0.715	0.338	-0.216	0.269
8	0.805	0.219	-0.328	0.178
9	0.826	0.266	-0.386	0.148
10	0.864	0.034	-0.324	0.131

Tabla 32: Función 4, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	0.894	0.706	0.249	0.768
3	1.403	1.042	0.217	0.836
4	1.456	0.985	0.41	1.09
5	1.336	1.405	0.703	1.04
6	1.555	1.108	0.252	1.019
7	1.332	0.889	0.185	0.863
8	1.426	0.513	0.051	0.497
9	1.397	0.595	0.01	0.48
10	1.372	0.506	0.102	0.48

En la función 4 el desempeño de Lagrange 2D fue comparable, aunque inferior al del resto de los algoritmos. De todos ellos fue RBF gaussiana la que obtuvo los mejores resultados.

Función 5:

Tabla 33: Función 5, $\log_{10}$ de error absoluto para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	-0.656	-0.689	-0.406	-0.609
3	-0.985	-0.869	-0.71	-0.727
4	-1.104	-0.906	-1.028	-0.768
5	-1.152	-1.264	-1.29	-0.793
6	-1.209	-1.561	-1.356	-0.828
7	-1.204	-1.587	-1.225	-0.846
8	-1.496	-1.622	-1.36	-0.87
9	-1.454	-1.772	-1.334	-0.893
10	-1.685	-1.799	-1.379	-0.958

Tabla 34: Función 5, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	6.913	7.313	4.81	7.036
3	6.631	6.595	5.712	7.036
4	6.776	6.788	6.022	7.036
5	6.455	6.447	6.12	7.036
6	5.623	6.353	6.01	7.036
7	6.659	5.335	5.922	7.036
8	6.593	5.31	5.696	6.984
9	-0.48	4.916	5.616	6.984
10	-0.974	4.773	5.563	6.66

En la función 5, en términos de error absoluto el desempeño de Lagrange 2D es superior al de RBF gaussiana y NNI, siendo solo superado por RBF *multiquadric*. En cuanto al error relativo, se tienen valores bastante altos en general, con la excepción de los resultados obtenidos por Lagrange 2D para 9 y 10 puntos los que fueron particularmente buenos.

Función 6:

Tabla 35: Función 6, $\log_{10}$ de error absoluto para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	0.022	-0.084	-0.195	-0.088
3	0.213	0.007	-0.22	-0.029
4	0.081	-0.155	-0.177	-0.074
5	0.122	0.193	-0.243	-0.113
6	0.05	-0.12	-0.269	-0.097
7	0.045	-0.112	-0.304	-0.157
8	0.049	-0.111	-0.351	-0.194
9	-0.024	-0.14	-0.374	-0.218
10	0.057	-0.063	-0.359	-0.243

Tabla 36: Función 6, $\log_{10}$ de error relativo para puntos reduciendo el error.
Fuente: Elaboración Propia.

	Lagrange 2D	RBF multiquadric	RBF gaussiana	NNI
2	8.133	8.064	7.527	8.128
3	8.364	7.851	7.726	8.131
4	8.322	7.821	7.838	8.131
5	8.237	8.234	7.753	8.131
6	8.172	7.81	7.724	8.134
7	8.283	8.145	7.746	8.134
8	8.325	7.985	7.749	8.158
9	8.181	7.874	7.661	8.158
10	8.248	8.06	7.673	8.158

Finalmente en la función 6 los mejores resultados fueron obtenidos por RBF gaussiana y NNI. Lagrange 2D entrega resultados similares, aunque peores, a los de RBF *multiquadric*.

Los resultados obtenidos por Lagrange 2D en esta serie de pruebas son en la mayoría de los casos similares a los de los otros interpoladores. En algunos casos inferiores como en las funciones 1, 4 y 6; y en otros superiores como ocurrió en las funciones 2 y 5. En general, con pocos puntos el algoritmo de Lagrange 2D pudo obtener resultados razonables.

Por otro lado, si bien la heurística utilizada para agregar los puntos puede haber contribuido a los buenos resultados obtenidos por Lagrange 2D, también puede resultar contraproducente en algunos contextos, como se pudo ver en el caso de la función 3.

CAPÍTULO 5
CONCLUSIONES

En el trabajo fue posible dar cumplimiento al principal objetivo planteado, desarrollando un algoritmo de interpolación basado en Lagrange aplicable a funciones escalares de 2 variables reales, sobre una distribución de puntos dispersa (libre de grilla). Además se adaptó el algoritmo de interpolación a su forma baricéntrica, lo que permite obtener los mismos resultados, pero reducir considerablemente el costo computacional de la ejecución de la función interpoladora, disminuyendo en un orden de magnitud la complejidad computacional de este proceso.

La implementación de la propuesta consiste en realizar una interpolación de Lagrange 1D compleja tomando de entrada los valores del eje x como la parte real y los valores del eje y como la parte imaginaria, la suma de ellos corresponde al primer término utilizado en la interpolación. En cuanto al segundo término, este corresponde a los valores del eje z como la parte real y una serie de valores convenientemente generados, llamados "corrección imaginaria'' que corresponderán, naturalmente, a la parte imaginaria. Dado que se producen sólo valores complejos, se toma la parte real para constituir la función interpoladora. Esta es la razón por la que el interpolador sólo es aplicable a funciones reales de variable real, siendo necesaria una modificación en el algoritmo si se busca adaptarlo para la interpolación de funciones 2D complejas.

Es posible realizar la interpolación de un conjunto de puntos utilizando sólo los valores del eje z como parte real del segundo término en la interpolación compleja, sin embargo, se pudo constatar que hacer esto en la práctica implica la aparición de oscilaciones en la función interpoladora que no hacen sino aumentar en número y magnitud en la medida que aumentan los puntos. Estas oscilaciones generan en la mayoría de los casos prácticos valores crecientes de error de interpolación haciendo contraproducente el incrementar la cantidad de puntos en la interpolación, teniendo un efecto similar al fenómeno de Runge para interpolación polinomial, aunque más preocupante pues se presenta incluso para cantidades bajas de puntos.

La solución que se encontró para este problema fue la generación de una serie de valores imaginarios complementarios a la data proveniente de los valores del eje z y que fue llamada "corrección imaginaria''. Esta corrección vendría a constituir la parte imaginaria del segundo término utilizado en la interpolación compleja e independiente de cuales sean los valores de esta corrección, no se cambiará el hecho de que los puntos son interpolados. Lo que se hace es generar los valores de esta corrección de forma conveniente de modo que se minimicen las oscilaciones de la función interpoladora.

La heurística utilizada para generar la corrección imaginaria fue asumir que esta sigue la estructura de un polinomio complejo de segundo grado, de modo que se tiene una cantidad finita de parámetros que es necesario optimizar. En cuanto a la función objetivo, se estiman

las oscilaciones como la integral sobre el dominio de la suma al cuadrado de las derivadas parciales de la función interpoladora. Resolver este problema de minimización es necesario para la obtención de la "corrección imaginaria'' óptima y, por ende, para la obtención del interpolador.

Se pudo comprobar de manera práctica que el uso de esta corrección imaginaria permite mitigar de manera considerable la incidencia negativa de esta oscilaciones, a precio de aumentar el tiempo de cómputo. Se pudo comprobar también a través de pruebas con diferentes correcciones que utilizar polinomios de mayor grado para la "corrección imaginaria'' otorga mejores resultados respecto a la calidad del interpolador, pero significa mayores tiempos de cómputo. Dicho esto, para cualquier corrección utilizada el fenómeno está aún presente y para cantidades suficientemente altas de puntos el error de interpolación tiende a aumentar en la medida que más puntos se agregan.

Respecto al desempeño del interpolador propuesto frente a otros algoritmos de interpolación en las pruebas de error, se pudo ver que para una cantidad baja de puntos (aproximadamente 8 o menos) los resultados obtenidos por la propuesta de Lagrange 2D son similares a los que producen RBF gaussiana, *multiquadric* y NNI; con variaciones que dependen de la función en particular. Para más puntos, sin embargo, el error crece a una tasa elevada debido a las oscilaciones, por lo que sería necesaria una "corrección imaginaria'' diferente, por ejemplo, generada con un polinomio de mayor grado.

5.1. TRABAJO FUTURO

Las siguientes recomendaciones se hacen dirigidas al lector interesado en profundizar, ampliar o mejorar la propuesta presentada en el trabajo:

- El objetivo planteado en un principio fue la construcción de un algoritmo de interpolación 2D, es decir, para funciones escalares de 2 variables. Se podría extender este concepto a funciones de 3 o más variables, generalización que se podría conseguir con la utilización de cuaterniones.
- Podría resultar útil buscar la forma de extender la propuesta a funciones vectoriales o a variables complejas.
- Resultaría ideal poder determinar de manera analítica la corrección óptima para minimizar las oscilaciones del interpolador. De este modo se obtendrían mejores funciones interpoladoras y se reduciría considerablemente el tiempo de cómputo por no ser necesario resolver un problema de minimización.
- Buscar otros criterios para optimizar la "corrección imaginaria'' a utilizar diferente a la medida de las oscilaciones, o utilizar una fórmula diferente para medirlas.

- Probar utilizar funciones diferentes, no necesariamente un polinomio, para generar la “corrección imaginaria’’ y evaluar si se obtienen mejores resultados.
- Podrían mejorar los resultados o reducirse el tiempo de cómputo si se utiliza un *initial guess* diferente en la minimización de los parámetros de la “corrección imaginaria’’ (en este momento se inicializan todos en 0). Sería conveniente, de ser posible, establecer una heurística que permita determinar estos *initial guess* convenientes. Del mismo modo se podría estudiar la utilización de otro algoritmo de optimización que resulte más conveniente para este contexto en particular.

REFERENCIAS BIBLIOGRÁFICAS

[Barnhill, 1977] Barnhill, R. E. (1977). Representation and approximation of surfaces. En *Mathematical software*, pp. 69–120. Elsevier.

[Berrut y Trefethen, 2004] Berrut, J.-P. y Trefethen, L. N. (2004). Barycentric lagrange interpolation. *SIAM review*, 46(3):501–517.

[Bolotnikov, 2014] Bolotnikov, V. (2014). Lagrange interpolation problem for quaternion polynomials. *Comptes Rendus Mathematique*, 352(7-8):577–581.

[Bozorgmanesh *et al.*, 2009] Bozorgmanesh, A., Otadi, M., SAFE, K. A., Zabihi, F., y BARKHORDARI, A. M. (2009). Lagrange two-dimensional interpolation method for modeling nanoparticle formation during ress process.

[Burkardt, 2019] Burkardt, J. (2019). Polynomial interpolation in 2d using lagrange polynomials.

[Calvi, 2005] Calvi, J.-P. (2005). Lectures on multivariate polynomial interpolation.

[de Boor, 2000] de Boor, C. (2000). Computational aspects of multivariate polynomial interpolation: Indexing the coefficients. *Advances in Computational Mathematics*, 12(4):289–301.

[De Boor y Ron, 1992] De Boor, C. y Ron, A. (1992). Computational aspects of polynomial interpolation in several variables. *Mathematics of Computation*, 58(198):705–727.

[Fasshauer, 2003] Fasshauer, G. E. (2003). 603 handouts and worksheets.

[Fasshauer, 2007] Fasshauer, G. E. (2007). *Meshfree approximation methods with MATLAB*, volumen 6. World Scientific.

[Gasca y Sauer, 2000] Gasca, M. y Sauer, T. (2000). Polynomial interpolation in several variables. *Advances in Computational Mathematics*, 12(4):377.

[Halton, 1964] Halton, J. (1964). Radical-inverse quasi-random point sequence [g5]. *Comm. ACM*, 7(12):701–702.

[Han, 2013] Han, D. (2013). Comparison of commonly used image interpolation methods. En *Proceedings of the 2nd international conference on computer science and electronics engineering*, pp. 1556–1559. Atlantis Press.

[Harris *et al.*, 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., y Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[Liang *et al.*, 2002] Liang, X.-Z., Cui, L.-H., y Zhang, J.-L. (2002). Properly posed set of nodes for bivariate lagrange interpolation along an algebraic curve. En *Analysis, combinatorics and computing*, pp. 295–304.

[Liang y Lu, 1998] Liang, X.-Z. y Lu, C.-M. (1998). Properly posed set of nodes for bivariate lagrange interpolation. *Approximation Theory IX*, 1:189–196.

[Mairhuber, 1956] Mairhuber, J. C. (1956). On haar's theorem concerning chebychev approximation problems having unique solutions. *Proceedings of the American Mathematical Society*, 7(4):609–615.

[Mathews *et al.*, 2004] Mathews, John H and Fink, Kurtis D and others (2004). *Numerical methods using MATLAB*, volumen 4. Pearson prentice hall Upper Saddle River, NJ.

[Mühlenstädt y Kuhnt, 2009] Mühlenstädt, T. y Kuhnt, S. (2009). Comparing different interpolation methods on two-dimensional test functions. *Proceedings of the ENBIS-EMSE, St. Etienne*, 142.

[Peters y Reif, 2008] Peters, J. y Reif, U. (2008). Generalized splines. *Subdivision Surfaces*, pp. 39–55.

[Saniee, 2008] Saniee, K. (2008). A simple expression for multivariate lagrange interpolation.

[Sauer, 1995] Sauer, T. (1995). Computational aspects of multivariate polynomial interpolation. *Advances in Computational Mathematics*, 3(3):219–237.

[Sauer, 2006] Sauer, T. (2006). *Numerical Analysis*. Pearson Addison Wesley.

[Shepard, 1968] Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. En *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524.

[Van Rossum y Drake, 2009] Van Rossum, G. y Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[Virtanen *et al.*, 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., y SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[Wolfram Research, 2020] Wolfram Research, I. (2020). Mathematica, version 12.1. Champaign, IL, 2020.

[Wright y Nocedal, 1999] Wright, S. y Nocedal, J. (1999). Numerical optimization. *Springer Science*, 35(67-68).

[Yi y Yao, 2019] Yi, S.-C. y Yao, L.-Q. (2019). A steady barycentric lagrange interpolation method for the 2d higher-order time-fractional telegraph equation with nonlocal boundary condition with error analysis. *Numerical Methods for Partial Differential Equations*, 35(5):1694–1716.