



# UNIVERSIDAD DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías  
Informática y de Telecomunicaciones

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y  
MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Algoritmos para la  
aproximación de un conjunto  
a partir de otros.  
Caracterización matemática  
del problema y estudio  
experimental.

Presentado por:  
Laura Lázaro Soraluce

Curso académico 2024-2025



# Algoritmos para la aproximación de un conjunto a partir de otros. Caracterización matemática del problema y estudio experimental.

Laura Lázaro Soraluce

Laura Lázaro Soraluce *Algoritmos para la aproximación de un conjunto a partir de otros. Caracterización matemática del problema y estudio experimental..*

Trabajo de fin de Grado. Curso académico 2024-2025.

**Responsable de  
tutorización**

Nicolás Marín Ruiz  
*Ciencias de la Computación e Inteligencia Artificial*

Daniel Sánchez Fernández  
*Ciencias de la Computación e Inteligencia Artificial*

Doble Grado en Ingeniería  
Informática y Matemáticas

Facultad de Ciencias y  
Escuela Técnica Superior  
de Ingenierías Informática  
y de Telecomunicaciones

Universidad de Granada

**DECLARACIÓN DE ORIGINALIDAD**

D./Dña. Laura Lázaro Soraluce

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 9 de abril de 2025

Fdo: Laura Lázaro Soraluce



*Dedicatoria (opcional)*

Ver archivo preliminares/dedicatoria.tex



## **Agradecimientos**

Agradecimientos (opcional, ver archivo preliminares/agradecimiento.tex).



## **Summary**

An english summary of the project (around 800 and 1500 words are recommended).  
File: `preliminares/summary.tex`



# Índice general

Agradecimientos	v
Summary	VII
Introducción	XI
<b>I. Fundamentos matemáticos</b>	<b>1</b>
<b>1. Fundamentos Matemáticos</b>	<b>3</b>
1.1. Introducción . . . . .	3
1.1.1. Contexto y Motivación . . . . .	3
1.1.2. Definición general del problema . . . . .	3
1.1.3. Aplicaciones e impacto . . . . .	4
1.1.4. Objetivos del trabajo y metodología . . . . .	5
1.2. Formalización matemática del problema . . . . .	5
1.2.1. Convenciones y notación . . . . .	5
1.2.2. Definiciones fundamentales . . . . .	6
1.2.3. Estructuras algebraicas de conjuntos . . . . .	7
1.2.4. Caracterización matemática del problema . . . . .	9
1.3. Complejidad y clasificación computacional . . . . .	9
1.3.1. Conceptos de complejidad computacional . . . . .	9
1.3.2. Naturaleza computacional del problema de recubrimiento . . . . .	10
1.3.3. Comparación con problemas similares . . . . .	10
1.4. Programación lineal . . . . .	10
1.5. Complejidad computacional . . . . .	10
1.6. Elementos del texto . . . . .	12
1.6.1. Listas . . . . .	12
1.6.2. Tablas y figuras . . . . .	12
1.7. Entornos matemáticos . . . . .	13
1.8. Listados de código . . . . .	14
1.9. Referencias a elementos del texto . . . . .	14
1.10. Bibliografía e índice . . . . .	15
<b>2. Consideraciones elaboración TFG</b>	<b>17</b>
2.1. Métodos exactos y aproximados . . . . .	17
2.1.1. Algoritmos exactos . . . . .	17
2.1.2. Algoritmos heurísticos y aproximados . . . . .	17
2.1.3. Evaluación de rendimiento de los algoritmos . . . . .	19
2.2. Implementación computacional y estudio experimental . . . . .	19
2.2.1. Metodología de la implementación . . . . .	19
2.2.2. Experimentos realizados . . . . .	19

*Índice general*

2.2.3. Comparación de resultados . . . . .	19
2.2.4. Discusión sobre eficiencia y precisión . . . . .	19
2.3. Normativa de la comisión del Grado en Matemáticas . . . . .	19
2.4. Formato de la memoria . . . . .	20
2.5. Recomendaciones . . . . .	21
<b>II. Aproximación Informática</b>	<b>23</b>
<b>3. Conclusiones y Perspectivas Futuras</b>	<b>25</b>
3.1. Conclusiones y trabajo futuro . . . . .	25
3.1.1. Resumen de los principales hallazgos . . . . .	25
3.1.2. Limitaciones del trabajo . . . . .	25
3.1.3. Posibles líneas de investigación futuras . . . . .	25
<b>III. Resultados y análisis</b>	<b>27</b>
<b>IV. Conclusiones y Trabajo Futuro</b>	<b>29</b>
<b>A. Ejemplo de apéndice</b>	<b>31</b>
<b>Glosario</b>	<b>33</b>
<b>Bibliografía</b>	<b>35</b>

## **Introducción**

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo preliminares/introduccion.tex



# **Parte I.**

## **Fundamentos matemáticos**



# 1. Fundamentos Matemáticos

## 1.1. Introducción

### 1.1.1. Contexto y Motivación

El problema de recubrimiento de conjuntos surge en el campo de las matemáticas en el siglo XIX, cuando se empieza a formalizar el concepto en teoría de conjuntos, incluyendo nociones como álgebra de conjuntos y anillos de conjuntos, en las que profundizaremos más adelante. En el siglo XX aparece también en el ámbito topológico, cuando se habla de recubrimientos abiertos de espacios topológicos, que guarda una cierta similitud con el concepto de recubrimiento de conjuntos.

Más recientemente, en la década de 1970, tienen lugar grandes avances en el estudio de la complejidad computacional, con el Teorema de Cook y el desarrollo del concepto NP-Completo. Más adelante, la publicación de la lista de 21 problemas NP-Completos de Richard Karp, nos presenta el problema del conjunto de cobertura (Set Cover Problem), popularizando así en el ámbito de la informática el concepto de recubrimiento de conjuntos. Este consiste en: dado un conjunto de elementos,  $U$ , y una familia  $S$  de subconjuntos de  $U$ , ¿se puede encontrar un subconjunto  $C \subseteq S$  de  $k$  o menos conjuntos con unión  $U$ ? Este problema de decisión también tiene una versión de optimización: en las mismas condiciones, minimizar el número de subconjuntos que usa  $C \subseteq S$ .

Del Set Cover Problem han surgido variantes como el Exact Cover By 3-Sets ( $X_3C$ ). Esta consiste en: dado un conjunto  $X$ , con cardinal múltiplo de tres, y una familia  $C$  de subconjuntos de  $X$  de tres elementos, encontrar un subconjunto  $C' \subseteq C$  donde cada elemento de  $X$  aparece en un y sólo un elemento de  $C'$ . Aunque no es exactamente el problema que vamos a abordar en este trabajo, sí está estrechamente relacionado y tiene implicaciones y aplicaciones prácticas similares.

COMENTAR MÁS SOBRE VARIANTES FAMOSAS DEL PROBLEMA? SU IMPORTANCIA Y SUS APLICACIONES?

### 1.1.2. Definición general del problema

Una vez que conocemos el contexto histórico, el origen y la importancia del problema que nos ocupa, nos centramos en formularlo de forma explícita y en definirlo formalmente.

Dados un conjunto  $U$  y una colección  $F$  de subconjuntos de  $U$ , queremos recubrir un subconjunto  $G \subseteq U$  utilizando diferentes operaciones sobre elementos de  $F$ .

## 1. Fundamentos Matemáticos

Lo ideal sería obtener una partición exacta de  $G$ , es decir, cubrirlo completamente sin incluir elementos que no están en  $G$  y sin que haya solapamiento entre los subconjuntos escogidos.

Formalmente, dado un conjunto  $U$  y un conjunto finito  $F$  de subconjuntos finitos de  $U$ , queríamos encontrar un nuevo subconjunto  $F' \subseteq F$  tal que, mediante operaciones como unión, intersección, etc., nos permitiese construir una partición algebraica de  $G$  inducida por  $F$ .

**Definición 1.1.** Dada una familia  $F' = P_1, \dots, P_n$  de subconjuntos de  $G$ ,  $F'$  es una partición de  $G$  si:

- $\bigcup_{i \in \{1, \dots, n\}} P_i = G$ ,
- $\forall P_i, P_j$ , o bien  $P_i = P_j$ , o bien  $P_i \cap P_j = \emptyset$

[Lip98]

Dicho de otra forma, buscaríamos obtener subconjuntos disjuntos  $P_1, \dots, P_k$  que cumpliesen las siguientes condiciones:

- Forman un recubrimiento exacto de  $G$ :

$$G = \bigcup_{i=1}^k P_i$$

- Son disjuntos dos a dos:

$$P_i \cap P_j = \emptyset, \quad \forall i \neq j$$

- Cada  $P_i$  se obtiene aplicando un número finito de operaciones de unión, intersección y complemento entre elementos de  $F$ .
- No hay redundancia, es decir, no podemos eliminar ningún  $P_i$  sin dejar de cubrir  $G$ .

Sin embargo, esto no siempre va a ser posible, pues puede que haya elementos de  $G$  que no quedan cubiertos por ningún elemento de  $F$ . Por ello, en este trabajo nos centramos en encontrar la mejor aproximación posible, evaluando diferentes métricas para medir la eficiencia y analizando la complejidad computacional de las soluciones.

EXPLICAR EXACTAMENTE EL INTERÉS DEL CONCEPTO PARTICIÓN?!

### 1.1.3. Aplicaciones e impacto

El problema de recubrimiento de conjuntos tiene aplicaciones en distintas áreas, especialmente en la toma de decisiones y la optimización de recursos, aplicada también a recubrimiento de áreas físicas. Veamos algunos ejemplos relevantes:

- En marketing: colocar anuncios de forma que cubran un público objetivo, sin redundancias (sin que las personas lo vean varias veces) y sin desperdiciar recursos (sin que lo vean personas a las que no va dirigido).

## 1.2. Formalización matemática del problema

- En logística: minimizar el número de camiones utilizados en la distribución de mercancías. Tendríamos un conjunto de todas las ubicaciones de entrega, un conjunto de ubicaciones que pueden ser cubiertas por cada camión y un conjunto de ubicaciones a cubrir. En este caso añadiríamos una restricción adicional de escoger el menor número posible de camiones, es decir, minimizar el número de subconjuntos escogidos para cubrir.
- En problemas de clasificación en informática: sacar un conjunto representativo de las características para construir un modelo eficiente en términos de predicción y de complejidad. Analizamos un caso concreto de una práctica de Inteligencia de Negocio del cuatrimestre pasado: predicción de aprobación de créditos. Tendríamos un conjunto de todas las instancias, todos los solicitantes de crédito con sus características; un subconjunto de características que los modelos usan para predecir la aprobación de crédito; y un conjunto de solicitantes con una etiqueta determinada (aprobado/rechazado).
- En redes de comunicación: colocar antenas o puntos de acceso para asegurar cobertura en todo el área. Tendríamos un conjunto con todos los posibles puntos de ubicación en un área determinada, un conjunto de áreas de cobertura de cada antena, y un conjunto de ubicaciones específicas que queremos cubrir con la red.
- En biología computacional: seleccionar un conjunto de genes que sea representativo de un grupo de pacientes específico (sano/enfermo). Tendríamos un conjunto con todos los genes presentes en el conjunto de muestras biológicas, un conjunto de genes que están activos cuando se tiene una cierta enfermedad, y un conjunto de genes esenciales para diferenciar entre grupos de interés.

Estas aplicaciones muestran la importancia del recubrimiento de conjuntos en la toma de decisiones eficientes y la optimización de recursos, lo que lo convierte en un problema de gran interés no sólo en el ámbito computacional.

### 1.1.4. Objetivos del trabajo y metodología

Tras un estudio matemático exhaustivo del problema, procederemos a la parte informática. Para ello, implementaremos varios algoritmos (como Greedy, Hill Climbing, Backtracking o Programación Lineal) buscando maximizar o minimizar diferentes métricas, dependiendo del caso. Evaluaremos estos algoritmos mediante su rendimiento al cubrir el subconjunto  $G$ , basándonos en las características observadas en la fase teórica del trabajo.

DESARROLLAR!!

## 1.2. Formalización matemática del problema

### 1.2.1. Convenciones y notación

A lo largo de este trabajo, utilizaremos las siguientes convenciones de notación:

- $U$  denotará el conjunto base sobre el que trabajamos.

## 1. Fundamentos Matemáticos

- $F$  representará el conjunto finito de subconjuntos finitos de  $U$ .
  - $m$  indicará el número de elementos en  $F$ .
  - $F_1, F_2, \dots, F_m$  serán los elementos de  $F$ .
  - $G \subseteq U$  será el subconjunto que buscamos recubrir.

Salvo que se indique lo contrario, mantendremos estas notaciones a lo largo del Trabajo para evitar ambigüedades y facilitar la lectura.

Convenimos sin pérdida de generalidad que  $U \neq \emptyset$ ,  $F \neq \emptyset$  ( $m \neq 0$ ), y que  $G \neq \emptyset$ , pues en dichos casos no habría estudio que hacer. En principio, trabajaremos sin imponer restricciones al número de veces que podemos utilizar cada elementos de  $F$  en las operaciones, por lo que el hecho de que haya elementos repetidos en  $F$  equivale a coger varias veces un mismo elemento de  $F$ . Por ello, podemos considerar que no hay elementos repetidos en  $F$ .

????????? HAY QUE IMPONER QUE NINGUNO DE LOS  $F_i = \emptyset$ ? Si es que sí, no podríamos trabajar con muchas de las estructuras algebraicas que requieren que esté el vacío en ellas (anillo o álgebra de conjuntos), pero a la vez aumentaría el espacio de búsqueda. Si es que no, habría que imponer que  $\exists i \in \{1, \dots, m\} | F_i \neq \emptyset$ , no? Podríamos permitir conjuntos vacíos en  $F$ , pero excluirlos de la selección final  $F'$ .

### **1.2.2. Definiciones fundamentales**

El número de particiones posibles para un conjunto finito depende de su cardinal y se llama el número de Bell y viene definido por

- $B(0) = 1$
  - $B(n) = \sum_{k=0}^{n-1} B(k)$

Cuando consideramos subconjuntos en el contexto de conjuntos algebraicos, uno de los primeros conceptos con los que nos topamos es el de **Conjunto Potencia**. Este representa el conjunto de todos los posibles subconjuntos de un conjunto.

**Definición 1.2.** El conjunto potencia de  $U$ , denotado  $\mathcal{P}(U)$ , es el conjunto cuyos elementos son todos los subconjuntos de  $U$ . Se escribe

$$\mathcal{P}(U) = \{X | X \subseteq U\}$$

[Cru22]

Es un concepto fundamental, pues cualquier conjunto de subconjuntos que consideremos, va a ser siempre subconjunto de  $\mathcal{P}(U)$ . Aplicándolo a nuestro problema, sea cual sea el conjunto  $F$  que consideremos,  $F \subseteq \mathcal{P}(U)$ .

En este caso en el que no aplicamos más restricciones a  $F$ , no podremos asegurar que  $G$  vaya a estar recubierto.

EXPLICAR EXACTAMENTE INTERÉS DEL CONCEPTO DE CONJUNTO POTENCIA E INVESTIGAR INTERÉS DE NÚMERO DE BELL

### 1.2.2.1. Conjuntos y subconjuntos

### 1.2.2.2. Recubrimientos y particiones

### 1.2.2.3. Operaciones sobre conjuntos

La unión de conjuntos es:

- Asociativa:  $(A \cup B) \cup C = A \cup (B \cup C)$
- Comutativa:  $A \cup B = B \cup A$
- Distributiva con respecto a la intersección:  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$  y  
 $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

**LAS DOS VERSIONES DE GREEDY:** La intersección también cumple estas mismas propiedades, pero la resta NO!! Por tanto, a la hora de implementar los algoritmos, en el caso de la unión o intersección, es equivalente aplicar una operación entre el conjunto resultado que vamos construyendo y un subconjunto de F, o aplicarlo entre dos subconjuntos de F y luego unirlo con cubierto. !!!

### 1.2.3. Estructuras algebraicas de conjuntos

#### 1.2.3.1. Anillos y álgebras de conjuntos

**Definición 1.3.** Un anillo de conjuntos es una colección no vacía  $F$  de conjuntos, tal que si  $F_i, F_j \in F$  para  $i \in \{1, \dots, m\}$ , entonces  $F_i \cup F_j \in F$  y  $F_i \setminus F_j \in F$ . [Hal74]

Esto es equivalente a decir que, si  $F_i, F_j \in F$ , entonces  $\bigcup_{i=1}^n F_i$  y  $\bigcap_{i=1}^n F_i \in F$ .

Teniendo esto en cuenta, el conjunto vacío y el conjunto  $\mathcal{P}(X)$  son anillos de conjuntos.

Ejemplo sencillo:

- $U = \{1, 2, 3, 4\}$
- $F = \{\emptyset, \{1\}, \{1, 2\}, \{3, 4\}, \{1, 2, 3, 4\}\}$

Operaciones válidas:

- $\{1\} \cup \{3, 4\} = \{1, 3, 4\} \in F$
- $\{1, 2, 3, 4\} \setminus \{1, 2\} = \{3, 4\} \in F$

Si en este ejemplo  $G = \{3\}$ , no podríamos cubrirlo exactamente a partir de operaciones sobre elementos de  $F$ . Podríamos aproximararlo por  $\{3, 4\}$ , por lo que ya estaríamos teniendo en cuenta elementos que no pertenecen a  $G$ . Por lo tanto, el hecho de que  $F$  fuese un anillo de conjuntos, no garantizaría poder cubrir  $G$  de manera exacta.

**Definición 1.4.** Un semianillo es una colección no vacía  $F$  de conjuntos tal que:

- si  $F_i, F_j \in F$  entonces  $F_i \cap F_j \in F$
- si  $F_i, F_j \in F$  y  $F_i \subset F_j$ , entonces  $\exists \{C_0, \dots, C_n\}$  una colección finita de conjuntos de  $F$ , tal que  $F_i = C_0 \subset C_1 \subset \dots \subset C_n = F_j$  y  $D_i = C_i \setminus C_{i-1} \in F \quad \forall i = 1, \dots, n$ .

## 1. Fundamentos Matemáticos

[Hal74]

De nuevo, todos los semianillos siempre contienen al conjunto vacío.

Ejemplo sencillo:

- $U = \{1, 2, 3, 4\}$
- $F = \{\emptyset, \{1\}, \{2, 3\}, \{1, 2, 3\}, \{4\}\}$

Operaciones válidas:

- $\{2, 3\} \cap \{1, 2, 3\} = \{2, 3\} \in F$
- $\{1, 2, 3\} \setminus \{2, 3\} = \{1\}$

Efectivamente, la intersección de cualesquiera dos conjuntos de  $F$  pertenece también a  $F$ . Si en este ejemplo tomamos  $G = \{3\}$ , no podríamos cubrirlo exactamente con elementos de  $F$ , sólo aproximarla  $\{2, 3\}$ , por lo que ya cogemos elementos fuera de  $G$ . Por lo tanto, el hecho de que  $F$  fuese un semianillo de conjuntos, no garantizaría poder cubrir  $G$  de manera exacta.

**Definición 1.5.** Un álgebra de conjuntos (también a veces conocido como campo de conjuntos) es una colección no vacía  $F$  de conjuntos tal que:

- si  $F_i \in F$  y  $F_j \in F$ , entonces  $F_i \cup F_j \in F$
- si  $F_i \in F$ , entonces  $F'_i \in F$ .

[Hal74]

Ejemplo sencillo:

- $U = \{1, 2, 3\}$
- $F = \{\emptyset, \{1\}, \{2, 3\}, U\}$

Operaciones válidas:

- $\overline{\{1\}} = \{2, 3\} \in F$
- $\{1\} \cup \{2, 3\} = U \in F$

Si en este ejemplo cogemos  $G = \{2\}$ , no podemos cubrirlo de manera exacta con operaciones sobre los elementos de  $F$ . Sólo podríamos obtener una aproximación como  $\{2, 3\}$ , que ya contiene elementos fuera de  $G$ .

Todo álgebra de conjuntos, es también un anillo, pero la implicación contraria no es cierta.

**Definición 1.6.** Un  $\sigma$ -anillo es una colección no vacía  $F$  de conjuntos tal que

- si  $F_i, F_j \in F$ , entonces  $F_i \setminus F_j \in F$
- si  $F_i \in F$ ,  $\forall i = 1, \dots$ , entonces  $\bigcup_{i=1}^{\infty} F_i \in F$ .

[Hal74]

Esta estructura sería interesante en generalizaciones o extensiones de nuestro problema, donde  $U$  es un conjunto infinito numerable, pues trabaja con uniones infinitas.

BARAJAR QUITAR SIGMA-ANILLO SI NO ESTÁ SUFICIENTEMENTE RELACIONADO.  
EXPLICAR EN ALGÚN MOMENTO CÓMO SE TRATAN ESTAS ESTRUCTURAS EN LA PRÁCTICA, CUANTO CUESTA CONSTRUIRLAS, CON CUANTOS CONJUNTOS NOS ENCONTRARÍAMOS ( $2^{2^b}$ )...

## 1.2.4. Caracterización matemática del problema

### 1.2.4.1. Condiciones para la existencia de un recubrimiento exacto

La elección de la estructura algebraica de  $F$  es una parte fundamental de nuestro problema, pues determina las operaciones de las que disponemos, así como las garantías de alcanzar el óptimo en los algoritmos que vamos a implementar. Realmente, el único tipo de conjunto  $F$  que asegura que podemos recubrir  $G$  de manera exacta es aquel que contiene todos los singeltons de  $U$ , es decir, todos sus conjuntos unitarios.

(HABLAR DE LA MAXIMA EXPRESIVIDAD Y DEL COSTE COMPUTACIONAL ASOCIADO A SUBIR LA EXPRESIVIDAD)

### 1.2.4.2. Relación con otras estructuras

## 1.3. Complejidad y clasificación computacional

### 1.3.1. Conceptos de complejidad computacional

Estructura	Complejidad Temporal	Aproximación	Operaciones Cerradas
Familia Arbitraria	$O(2^{ F } \cdot  U )$	Sin garantía	Ninguna
Semianillo	$O( F ^2 \cdot  U )$	No constante	$\cap, \setminus$ disjuntas
Anillo	$O( F  \cdot  U )$	$O(\log k)$	$\cup, \setminus$
Álgebra	$O( F  \cdot  U )$	Exacta (si $F = 2^U$ )	$\cup, \cap, {}^c$
$\sigma$ -Anillo	No aplicable	No aplicable	$\cup_{\text{numerables}}$

Tabla 1.1.: Comparación de complejidad y garantías por estructura algebraica

REVISAR ESTA TABLA!!!

## *1. Fundamentos Matemáticos*

### **1.3.1. Problemas P, NP y NP-completos**

### **1.3.2. Naturaleza computacional del problema de recubrimiento**

### **1.3.3. Comparación con problemas similares**

#### **1.3.3.1. Set Cover Problem**

#### **1.3.3.2. Exact Cover Problem**

#### **1.3.3.3. Relación con problemas de optimización**

## **1.4. Programación lineal**

TODAVÍA NADA !!

Otra idea que merece la pena explorar al resolver nuestro problema, es la programación lineal. Sabemos que en programación lineal, tenemos que establecer una expresión matemática en términos de variables, que va a ser la que queremos minimizar o maximizar. Aplicaremos las restricciones del problema, que serán también lineales. Sabemos que la programación encuentra siempre la mejor solución y además no pierde ninguna solución posible.

Modelamos nuestro problema como programación con restricciones, para poder representar más fácilmente el uso de operaciones de conjuntos, y contemplar todas las soluciones factibles.

## **1.5. Complejidad computacional**

HABLAR DE CONCEPTOS COMO NP-COMPLETO O NP-DURO Y SET COVER PROBLEM, EXACT COVER PROBLEM O BOOLEAN SATISFIABILITY.

Este documento es una plantilla para la elaboración de un trabajo fin de Grado siguiendo los [requisitos](#) de la comisión de Grado en Matemáticas de la Universidad de Granada que, a fecha de junio de 2023, son las siguientes:

- La memoria debe realizarse con un procesador de texto científico, preferiblemente (La)TeX.
- La portada debe contener el logo de la UGR, incluir el título del TFG, el nombre del estudiante y especificar el grado, la facultad y el curso actual.
- La contraportada contendrá además el nombre del tutor o tutores.
- La memoria debe necesariamente incluir:
  - Declaración explícita firmada en la que se asume la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente. Esta declaración se puede descargar en la web del Grado.
  - un índice detallado de capítulos y secciones,
  - un resumen amplio en inglés del trabajo realizado (se recomienda entre 800 y 1500 palabras),

- una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas,
  - una bibliografía final que incluya todas las referencias utilizadas.
- Se recomienda que la extensión de la memoria sea de unas 50 páginas, sin incluir posibles apéndices.

Para generar el pdf a partir de la plantilla basta compilar el fichero `tfg.tex`. Es conveniente leer los comentarios contenidos en dicho fichero pues ayudarán a entender mejor como funciona la plantilla.

La estructura de la plantilla es la siguiente<sup>1</sup>:

- Carpeta **preliminares**: contiene los siguientes archivos
  - dedicatoria.tex** Para la dedicatoria del trabajo (opcional)
  - agradecimientos.tex** Para los agradecimientos del trabajo (opcional)
  - introduccion.tex** Para la introducción (obligatorio)
  - summary.tex** Para el resumen en inglés (obligatorio)
- **tablacontenidos.tex** Genera de forma automática la tabla de contenidos, el índice de figuras y el índice de tablas. Si bien la tabla de contenidos es conveniente incluirla, el índice de figuras y tablas es opcional. Por defecto está desactivado. Para mostrar dichos índices hay que editar este fichero y quitar el comentario a `\listoffigures` o `\listoftables` según queramos uno de los índices o los dos. En este archivo también es posible habilitar la inclusión de un índice de listados de código (si estos han sido incluidos con el paquete `listings`)

El resto de archivos de dicha carpeta no es necesario editarlos pues su contenido se generará automáticamente a partir de los metadatos que agreguemos en `tfg.tex`

- Carpeta **capitulos**: contiene los archivos de los capítulos del TFG. Añadir tantos archivos como sean necesarios. Este capítulo es `capitulo01.tex`.
- Carpeta **apendices**: Para los apéndices (opcional)
- Carpeta **img**: Para incluir los ficheros de imagen que se usarán en el documento.
- Fichero `library.bib`: Para incluir las referencias bibliográficas en formato `bibtex`. Es útil la herramienta `doi2bib` para generar de forma automática el código `bibtex` de una referencia a partir de su DOI así como la base de datos bibliográfica `MathSciNet`. Para que una referencia aparezca en el pdf no basta con incluirla en el fichero `library.bib`, es necesario además *citarla* en el documento usando el comando `\cite`. Si queremos mostrar todos las referencias incluidas en el fichero `library.bib` podemos usar `\cite{*}` aunque esta opción no es la más adecuada. Se aconseja que los elementos de la bibliografía estén citados al menos una vez en el documento (y de esa forma aparecerán de forma automática en la lista de referencias).

---

<sup>1</sup>Los nombres de las carpetas no se han acentuado para evitar problemas en sistemas con Windows

## 1. Fundamentos Matemáticos

- Fichero `glosario.tex`: Para incluir un glosario en el trabajo (opcional). Si no queremos incluir un glosario deberemos borrar el comando `\input{glosario.tex}` del fichero `tfg.tex` y posteriormente borrar el fichero `glosario.tex`
- Fichero `tfg.tex`: El documento maestro del TFG que hay que compilar con L<sup>A</sup>T<sub>E</sub>X para obtener el pdf. En dicho documento hay que cambiar la *información del título del TFG y el autor así como los tutores*.

## 1.6. Elementos del texto

En esta sección presentaremos diferentes ejemplos de los elementos de texto básico. Conviene consultar el contenido de `capitulos/capitulo01.tex` para ver cómo se han incluido.

### 1.6.1. Listas

En L<sup>A</sup>T<sub>E</sub>X tenemos disponibles los siguientes tipos de listas:

Listas enumeradas:

1. item 1
2. item 2
3. item 3

Listas no enumeradas

- item 1
- item 2
- item 3

Listas descriptivas

`termino1` descripción 1

`termino2` descripción 2

### 1.6.2. Tablas y figuras

En la [Tabla 1.2](#) o la [Figura 1.1](#) podemos ver...

Agrupados		
cabecera	cabecera	cabecera
elemento	elemento	elemento
elemento	elemento	elemento
elemento	elemento	elemento

Tabla 1.2.: Ejemplo de tabla



Figura 1.1.: Logotipo de la Universidad de Granada

## 1.7. Entornos matemáticos

La plantilla tiene definidos varios entornos matemáticos cuyo nombre es el mismo omitiendo los acentos. Así, para incluir una *proposición* usaríamos:

```
\begin{proposition}
  texto de la proposición
\end{proposition}
```

Ver el código fuente del archivo `documentacion.tex` en la carpeta `capitulos` para el resto de ejemplos.

**Teorema 1.7.** *Esto es un ejemplo de teorema.*

**Proposición 1.8.** *Ejemplo de proposición*

**Lema 1.9.** *Ejemplo de lema*

**Corolario 1.10.** *Ejemplo de corolario*

**Definición 1.11.** Ejemplo de definición

**Observación 1.12.** Ejemplo de observación

Adicionalmente está definido el entorno `teorema*` que permite incluir un teorema sin numeración:

**Teorema** (Fórmula de Gauß-Bonnet). *Sea  $S$  una superficie compacta y  $K$  su curvatura de Gauß. Entonces*

$$\int_S K = 2\pi\chi(S)$$

donde  $\chi(S)$  es la característica de Euler de  $S$ .

Las fórmulas matemáticas se escriben entre símbolos de dólar \$ si van en línea con el texto o bien usando el entorno<sup>2</sup> `equation` cuando queremos que se impriman centradas en una línea propia, como el siguiente ejemplo

$$\cos^2 x + \sin^2 x = 1. \tag{1.1}$$

Gracias al paquete `mathtools`, las ecuaciones escritas dentro del entorno `equation` llevarán numeración de forma automática si son referenciadas en cualquier parte del documento (por ejemplo la identidad Pitagórica (1.1), ver el código de los dos anteriores ejemplos y la [Sección 1.9](#) para más información sobre referencias cruzadas en el documento).

<sup>2</sup>También es posible delimitar una ecuación mediante los comandos `\[` y `\]` pero éstas nunca llevarán numeración aunque añadamos una etiqueta y las referenciamos (ver [Sección 1.9](#)).

## 1.8. Listados de código

Podemos incluir un archivo externo de código mediante el comando `lstinputlisting` especificando su nombre completo (incluyendo la extensión) y usando la opción `inputpath` para indicar la ruta hacia el fichero (siempre referida a la carpeta principal de la plantilla) así como la opción `language` para indicar el lenguaje de programación en que está escrito (esto permitirá a `LATEX` colorear adecuadamente el código). Además, si lo consideramos necesario, podemos indicar las líneas que queremos mostrar (ver el código fuente del [Código 1.1](#)). Consultar todas las opciones posibles en la [documentación del paquete `listings`](#).

```

11   for(i in 2:(num-1)) {
12     if ((num % i) == 0) {
13       flag = 0
14       break
15     }
16   }
17 }
```

Código 1.1: Extracto código (líneas de 11 a 17) del fichero `primeR.r`

Alternativamente, podemos incluir el código en un entorno `lstlisting` como el [Código 1.2](#)

```

18 def dot(v, w):
19     """Producto escalar de v y w,  $v_0 \cdot w_0 + \dots + v_n \cdot w_n$ """
20     return sum(v_i * w_i for v_i, w_i in zip(v, w))
21
22 def funcion_activacion(x):
23     """1 si la entrada es mayor o igual que 1, 0 en otro caso."""
24     return 1 if x >= 0 else 0
25
26 def perceptron(entrada, pesos):
27     """1 si el perceptrón se activa, 0 en otro caso"""
28     return funcion_activacion(dot(entrada, pesos))
```

Código 1.2: Implementación de un perceptrón

La opción `float` al incluir un listado de código permitará a dicho bloque “flotar” como si fuese un entorno `figure` y de esta manera evitaremos que se corte al final de una página.

## 1.9. Referencias a elementos del texto

Para las referencias a los elementos del texto (secciones, capítulos, teoremas,...) se procede de la siguiente manera:

- Se *marca* el elemento (justo después del mismo si se trata de un capítulo o sección o en el interior del *entorno* en otro caso), mediante el comando `\label{etiqueta}`, donde *etiqueta* debe ser un identificador único.
- Para crear una referencia al elemento en cualquier otra parte del texto se usa el comando `\ref{etiqueta}` (únicamente imprime la numeración asociada a dicho elemento, por ejemplo [1](#) o [1.7](#)) o bien `\autoref{etiqueta}` (imprime la numeración del elemento así como un texto previo indicando su tipo, por ejemplo [Capítulo 1](#) o [Teorema 1.7](#))

## 1.10. Bibliografía e índice

Esto es un ejemplo de texto en un capítulo. Incluye varias citas tanto a libros [?], artículos de investigación [Eul85], recursos online [Wik23] (páginas web), tesis [Rem56], trabajo fin de máster [Tan96], trabajo fin de grado [Doe03] así como artículos sin publicar (preprints) [CIMT22] (en estos últimos usar el campo note para añadir la información relevante).

Ver el fichero library.bib para las distintas plantillas. Cada nueva referencia debe añadirse en dicho fichero siguiendo el estilo del código bibtex según el tipo de referencia (página web, tesis, trabajo fin de grado o máster, artículo de investigación, libro,...). Alternativamente se puede usar la web <https://zbib.org> para generar automáticamente el código bibtex.



## 2. Algoritmos y Aproximación Computacional

### 2.1. Métodos exactos y aproximados

Para todos los algoritmos que vamos a implementar, utilizamos un método común para generar la familia  $F$  a partir de  $U$  y del número de subconjuntos que queremos que tenga  $F$ ,  $m$ . Para cada uno de los  $m$  subconjuntos que se van a generar, elegimos aleatoriamente un tamaño entre 1 y el número total de elementos de  $U$ , y seleccionamos elementos aleatorios del universo hasta que alcancemos el tamaño deseado. Este método asegura que todos los subconjuntos generados están contenidos en  $U$ , lo cual es un requisito para nuestro problema.

```
29 vector<Conjunto> generar_subconjuntos(int m, const Conjunto& U){  
30     vector<Conjunto> F(m);  
31     vector<int> universo(U.begin(), U.end());  
32     random_device rd;  
33     mt19937 gen(rd());  
34     uniform_int_distribution<int> dis(0, universo.size() - 1);  
35     for (int i = 0; i < m; i++) {  
36         int tam = dis(gen) % U.size() + 1;  
37         while(F[i].size() < tam) {  
38             F[i].insert(universo[dis(gen)]);  
39         }  
40     }  
41     return F;  
42 }
```

Código 2.1: Método generar\_subconjuntos

#### 2.1.1. Algoritmos exactos

##### 2.1.1.1. Búsqueda exhaustiva

##### 2.1.1.2. Programación Lineal Entera

#### 2.1.2. Algoritmos heurísticos y aproximados

##### 2.1.2.1. Algoritmo Greedy

Uno de los algoritmos que hemos elegido para la resolución de nuestro problema es un Greedy. Este tipo de algoritmos toma decisiones paso a paso, escogiendo la mejor opción local, por lo que destaca por su eficiencia computacional, ya que no necesita explorar todas las posibles combinaciones en cada paso. En cada momento, escoge la mejor opción disponible según un cierto criterio, con el objetivo de construir una buena solución. Sin embargo, estos algoritmos no garantizan que obtengamos la solución óptima, porque se puede llegar a óptimos locales, al no tener en cuenta cómo las decisiones en cada paso afectan a la final. El

## 2. Consideraciones elaboración TFG

proceso de elección de soluciones parciales, termina cuando alcanzamos un número de iteraciones máximo, cuando agotamos las opciones disponibles, o cuando tenemos una solución completa, que no mejora.

Para la implementación del algoritmo Greedy orientado a aproximar el conjunto  $G$  a partir de elementos de  $F$ , hemos escogido el lenguaje de programación C++. Por un lado, C++, al ser un lenguaje compilado, nos permite tener un tiempo de ejecución significativamente más bajo en comparación con otros lenguajes interpretados. Esto nos beneficia ya que buscamos trabajar con conjuntos de gran tamaño. Por otro lado, disponemos en C++ de estructuras de datos que pueden facilitarnos el modelado de nuestro problema: unordered set, set, vector... Aunque en Python también tenemos una sintaxis sencilla y bibliotecas útiles como NumPy o set, tiene muchas limitaciones para conjuntos con muchos datos, que es uno de los principales objetivos de este trabajo. Además, C++ permite optimizar el uso de recursos y la asignación y liberación de memoria, pues se tiene un mayor control.

En nuestro caso, buscamos construir un conjunto que aproxime lo mejor posible al conjunto objetivo  $G$ , utilizando  $F$ , y minimizando el coste:

$$\text{Coste}(\text{Covered}) = \alpha \cdot |\text{Covered} \setminus G| + \beta \cdot |G \setminus \text{Covered}|$$

en cada iteración. Nos interesa este coste, porque como hemos dicho, queremos disminuir el número de elementos que cogemos que se salen de  $G$ , así como el número de elementos de  $G$  que quedan sin cubrir. Finalmente, una vez tengamos nuestro conjunto  $\text{Covered}$ , utilizamos el índice de Jaccard para medir la efectividad de nuestra elección:

$$\text{Jaccard}(\text{Covered}) = \frac{\text{Covered} \cap G}{\text{Covered} \cup G}$$

En este algoritmo, partimos de un conjunto vacío llamado  $\text{Covered}$ , que representará el conjunto final por el que aproximamos  $G$ . En este caso, utilizamos un parámetro  $k$ , que limita el número máximo de operaciones que podemos realizar. Este parámetro sirve como cota de complejidad, y evita que el algoritmo consuma recursos computacionales excesivos. Hemos desarrollado dos variantes:

1. **Greedy por parejas:** En cada iteración consideramos todas las posibles combinaciones de dos subconjuntos  $F_i$  y  $F_j$  de la familia  $F$ , aplicando entre ellos todas las operaciones disponibles (unión, intersección y diferencia). El resultado de cada operación lo unimos con  $\text{Covered}$ , y evaluamos el costo total del nuevo conjunto. Seleccionamos aquella terna  $(F_i, F_j, Op)$  que minimice el costo, y actualizamos  $\text{Covered}$  con el conjunto obtenido. Esta variante nos permite generar conjuntos más complejos que podrían no estar presentes directamente en  $F$ , aprovechando la capacidad combinatoria del anillo de conjuntos generado por  $F$ .
2. **Greedy individual:** A diferencia de la anterior, en esta versión aplicamos directamente una operación entre  $\text{Covered}$  y un único subconjunto  $F_i \in \mathcal{F}$  en cada iteración. Entre todas las combinaciones posibles de  $(\text{Covered}, F_i, Op)$ , elegimos la que minimiza el costo respectivo. Esta versión es más eficiente computacionalmente y se acerca más a los razonamientos más clásicos de algoritmos Greedy. Sin embargo, tiene una capacidad

## 2.2. Implementación computacional y estudio experimental

limitada para generar nuevos conjuntos, por lo que puede estancarse en soluciones subóptimas, especialmente cuando los conjuntos en  $F$  no cubren adecuadamente ciertas regiones de  $G$ .

Tabla 2.1.: Comparación entre Greedy Jaccard normal y con recubrimiento (con `max_iter = 500` en todos los casos)

Tipos	U	F	G	m	Tiempo	Iteraciones	Jaccard
Normal	2 500	41	308	125	0.063 s	7	0.1364
Recubrimiento	2 500	256	308	125	0.915 s	89	0.2193
Normal	5 000	196	325	250	0.529 s	19	0.0968
Recubrimiento	5 000	502	325	250	0.948 s	64	0.1706
Normal	10 000	19	1160	500	0.056 s	3	0.1176
Recubrimiento	10 000	1003	1160	500	46.6 s	317	0.2180
Normal	20 000	327	355	1000	3.87 s	29	0.0524
Recubrimiento	20 000	2003	355	1000	1 min 33.8 s	334	0.1066
Normal	50 000	1587	610	2500	1 min 23.4 s	54	0.0414
Recubrimiento	50 000	5003	610	2500	9 min 3.7 s	500	0.1056

PENSAR EN AUMENTAR MAX\_ITER YA QUE EN EL ÚLTIMO CASO SE ALCANZAN LAS 500 ITERACIONES. EXPLICAR ESTA TABLA!!! ANALIZAR SI MERECE LA PENA EL TIEMPO EXTRA Y LAS ITERACIONES EXTRA QUE SE HACEN AL TRABAJAR CON RECUBRIMIENTO, PARA CONSEGUIR LA MEJORA QUE HAY EN JACCARD\_INDEX.

### 2.1.2.2. Hill Climbing

### 2.1.2.3. Metaheurísticas

## 2.1.3. Evaluación de rendimiento de los algoritmos

## 2.2. Implementación computacional y estudio experimental

### 2.2.1. Metodología de la implementación

### 2.2.2. Experimentos realizados

### 2.2.3. Comparación de resultados

### 2.2.4. Discusión sobre eficiencia y precisión

## 2.3. Normativa de la comisión del Grado en Matemáticas

El TFG lo rigen dos normativas:

- una a nivel general de la UGR ([Reglamento del Trabajo o Proyecto fin de Grado de la Universidad de Granada<sup>1</sup>](#)) y

<sup>1</sup><https://secretariageneral.ugr.es/sites/webugr/secretariageneral/public/inline-files/BOUGR/187/PLANTILLA%20CABECERASDoc2.pdf>

## 2. Consideraciones elaboración TFG

- otra complementaria a nivel de la Facultad de Ciencias ([Reglamento del trabajo fin de grado en la Facultad de Ciencias de la Universidad de Granada](#)<sup>2</sup>).

Además, la comisión del Grado de Matemáticas impone unos [Requisitos de la memoria](#)<sup>3</sup>. El [TFG](#) hay que elaborarlo preferiblemente en LaTeX y puede usar la plantilla disponible en [Plantilla TFG grado en matemáticas formato .tex](#)<sup>4</sup>.

Toda la información anterior puede encontrarse en la [web del Grado en Matemáticas](#)<sup>5</sup>. Es conveniente tener presente la documentación anterior para la elaboración del [TFG](#). En especial en lo relativo a las fechas de depósito del [TFG](#) para su defensa.

A continuación destaco algunos aspectos importantes de la misma:

- El plagio, entendido como la presentación de un trabajo u obra hecho por otra persona como propio o la copia de textos sin citar su procedencia y dándolos como de elaboración propia, conllevará automáticamente la calificación numérica de cero. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.
- Las memorias entregadas por parte de los estudiantes tendrán que ir firmadas sobre una declaración explícita en la que se asume la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

## 2.4. Formato de la memoria

La memoria se presentará usando un editor de textos científico, preferiblemente [LATEX](#), e incluir los siguientes apartados:

1. *Resumen en inglés*: Deberá estar escrito completamente en inglés y tener una longitud recomendada entre 800 y 1500 palabras.
2. *Introducción*. Deberá:
  - Indicar los *Objetivos del trabajo*: deberán aparecer con claridad los objetivos inicialmente previstos en la propuesta de [TFG](#) y los finalmente alcanzados con indicación de dificultades, cambios y mejoras respecto a la propuesta inicial. Si procede, es conveniente apuntar de manera precisa las interdependencias entre los distintos objetivos y conectarlos con los diferentes apartados de la memoria. Se pueden destacar aquí los aspectos formativos previos más utilizados.
  - Contextualizar el trabajo explicando antecedentes importantes para el desarrollo realizado y efectuando, en su caso, un estudio de los progresos recientes.
  - Describir el problema abordado, de forma que el lector tenga desde este momento una idea clara de la cuestión a resolver o del producto a desarrollar y una visión general de la solución alcanzada.
  - Indicar los resultados obtenidos.
  - Citar las principales fuentes consultadas.

<sup>2</sup><https://fcienicias.ugr.es/images/stories/documentos/reglamentos/reglamentoTfgCiencias23.pdf>

<sup>3</sup><https://grados.ugr.es/matematicas/pages/infoacademica/tfg/requisitosTFG>

<sup>4</sup><https://github.com/latex-mat-ugr/Plantilla-TFG/archive/master.zip>

<sup>5</sup><https://grados.ugr.es/matematicas/pages/infoacademica/trabajofingrado>

3. *Desarrollo del trabajo:* El trabajo se estructurará en partes o capítulos según convengan, con la posibilidad de incluir apéndices. Se recomienda que la extensión de esta parte (sin incluir los apéndices) sea de unas 50 páginas.
4. *Conclusiones y vías futuras:* Las conclusiones deberán incluir todas aquellas de tipo profesional y académico. Si hubiese posibles vías claras de desarrollo posterior sería interesante destacarlas aquí, poniéndolas en valor en el contexto inicial del trabajo.
5. *Bibliografía final:* Se incluirán tanto las fuentes primarias como todas aquellas cuyo peso haya sido menor en la realización del trabajo. Se recomienda un breve comentario de las referencias, ya sea individualizado, por grupos de referencias o global. En caso de incluir URLs de páginas web deberán ir acompañadas de título, autor y fecha de último acceso, entre otros datos relevantes. Se recomienda no abusar de este tipo de fuentes.

## 2.5. Recomendaciones

A la hora de abordar un trabajo como este, de cierta complejidad y extensión, es conveniente tener ciertas consideraciones desde un principio que ayuden a la organización y realización del mismo.

- La memoria deberá ceñirse a las directrices dadas en la sección precedente.
- Cualquier consulta externa (libro, artículo, página web, imagen,...) debe estar debidamente referenciada tanto en el texto como en la bibliografía al final del trabajo. La bibliografía debe de aparecer en orden alfabético (del primer autor) en el formato indicado en la plantilla.
- Se debe evitar copiar texto de forma literal, salvo citas literales, que se indicarán como tales y entrecomilladas. LaTeX proporciona el entorno quote para ello.
- Todas las imágenes y tablas incluidas en el documento deben figurar con su respectivos créditos (excepto que sean de elaboración propia). Por tanto, es recomendable guardar las referencias consultadas (direcciones web, libros) para la obtención de cualquier material gráfico o de datos.
- Si el trabajo contiene gran cantidad de vocabulario específico, conviene añadir un glosario de términos al final del mismo. Esto es mejor ir haciéndolo conforme se avanza en la redacción del trabajo.
- Es conveniente hacer un esquema inicial con la estructura general de la memoria: ¿de cuántas partes constará? ¿en qué orden? ¿qué incluirá cada una de ellas? En la plantilla proporcionada se recomienda una estructura general. Ello ayudará a organizar mejor el trabajo. No obstante, dicha estructura inicial puede ser modificada cuando el trabajo esté avanzado si el contenido lo requiere.



## **Parte II.**

### **Aproximación Informática**



## **3. Conclusiones y Perspectivas Futuras**

### **3.1. Conclusiones y trabajo futuro**

#### **3.1.1. Resumen de los principales hallazgos**

#### **3.1.2. Limitaciones del trabajo**

#### **3.1.3. Posibles líneas de investigación futuras**

Este fichero `capitulo-ejemplo.tex` es una plantilla para añadir capítulos al TFG. Para ello, es necesario:

- Crear una copia de este fichero `capitulo-ejemplo.tex` en la carpeta `capitulos` con un nombre apropiado (p.e. `capitulo01.tex`).
- Añadir el comando `\input{capitulos/capitulo01}` en el fichero principal `tfg.tex` donde queremos que aparezca dicho capítulo.



## **Parte III.**

### **Resultados y análisis**



## **Parte IV.**

### **Conclusiones y Trabajo Futuro**



## A. Ejemplo de apéndice

Los apéndices son opcionales.

Este fichero `apendice-ejemplo.tex` es una plantilla para añadir apéndices al `TFG`. Para ello, es necesario:

- Crear una copia de este fichero `apendice-ejemplo.tex` en la carpeta `apendices` con un nombre apropiado (p.e. `apendice01.tex`).
- Añadir el comando `\input{apendices/apendice01}` en el fichero principal `tfg.tex` donde queremos que aparezca dicho apéndice (debe de ser después del comando `\appendix`).



## Glosario

La inclusión de un glosario es opcional.

Archivo: `glosario.tex`

$\mathbb{R}$  Conjunto de números reales.

$\mathbb{C}$  Conjunto de números complejos.

$\mathbb{Z}$  Conjunto de números enteros.



## Bibliografía

- [CIMT22] Jesús Castro-Infantes, José M. Manzano, y Francisco Torralbo. Conjugate plateau constructions in product spaces, 2022. Preprint. arXiv: 2203.13162 [math.DG].
- [Cru22] Esteban Rubén Hurtado Cruz. Álgebra superior 1 - unidad 1.3: Potencia, producto cartesiano, familias, 2022. Último acceso: 12 de marzo de 2025.
- [Doe03] John Doe. Are we living in a simulation?, July 2003. Bachelor's Thesis, Massachusetts Institute of Technology, Cambridge, MA.
- [Eul85] Leonhard Euler. An essay on continued fractions. *Math. Systems Theory*, 18(4):295–328, 1985. Translated from the Latin by B. F. Wyman and M. F. Wyman.
- [Hal74] Paul R. Halmos. *Measure Theory*. Graduate Texts in Mathematics. Springer-Verlag, Berlin, first edición, 1974. Originally published by Litton Educational Publishing, Inc., 1950.
- [Lip98] Seymour Lipschutz. *Teoría de conjuntos y temas afines*. Schaum's Outline. McGraw-Hill, 1<sup>a</sup> edición en español edición, 1998.
- [Rem56] Robert Charles Rempel. *Relaxation Effects for Coupled Nuclear Spins*. PhD thesis, Stanford University, Stanford, CA, June 1956.
- [Tan96] Jian Tang. Spin structure of the nucleon in the asymptotic limit. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1996.
- [Wik23] Wikipedia. Leonhard Euler — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/Leonhard\\_Euler](https://en.wikipedia.org/wiki/Leonhard_Euler), 2023. [Recurso online, accedido el 27 de julio de 2023].