Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería.

Área Académica de Matemáticas y Física

Amalgama: Automatización computacional del análisis numérico para modelos de partículas con materia oscura

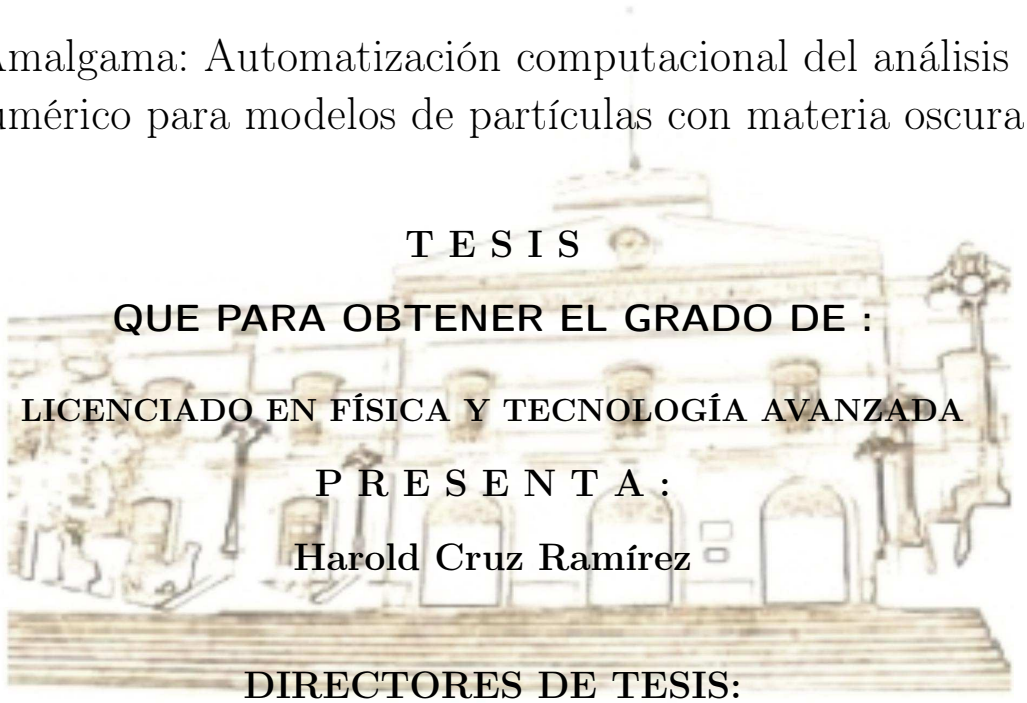
T E S I S

QUE PARA OBTENER EL GRADO DE :

LICENCIADO EN FÍSICA Y TECNOLOGÍA AVANZADA

P R E S E N T A :

Harold Cruz Ramírez



DIRECTORES DE TESIS:

Dr. Lao Tsé López Lozano
Dra. Alba Leticia Carrillo Monteverde

Mineral de la Reforma Hgo.
Marzo 2026

Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
School of Engineering and Basic Sciences

Mineral de la Reforma, Hgo., a 26 de febrero de 2026

Número de control: ICBI-D/315/2026
Asunto: Autorización de impresión.

MTRA. OJUKY DEL ROCÍO ISLAS MALDONADO
DIRECTORA DE ADMINISTRACIÓN ESCOLAR DE LA UAEH

Con Título Quinto, Capítulo II, Capítulo V, Artículo 51 Fracción IX del Estatuto General de nuestra Institución, por este medio, le comunico que el Jurado asignado al egresado de la Licenciatura en Física y Tecnología Avanzada **Harold Cruz Ramírez,** quien presenta el trabajo de titulación **"Amalgama: Automatización computacional del análisis numérico para modelos de partículas con materia oscura",** ha decidido, después de revisar fundamento en lo dispuesto en el Título Tercero, Capítulo I, Artículo 18 Fracción IV; dicho trabajo en la reunión de sinodales, **autorizar la impresión del mismo**, una vez realizadas las correcciones acordadas.

A continuación, firman de conformidad los integrantes del Jurado:

Presidente:	Dr. Adolfo Enrique Guevara Escalante	Adolfo Guevara
Secretario:	Dra. Alba Leticia Carrillo Monteverde	Alba Leticia Carrillo M.
Vocal:	Dr. Lao Tsé López Lozano	
Suplente:	Dr. Gerardo Hernández Tomé	

Sin otro particular por el momento, reciba un cordial saludo.

Atentamente
"Amor, Orden y Progreso"

Mtro. Gabriel Vergara Rodríguez
Director del ICBI

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO ICBI DIRECCIÓN

GVR/MMM

Ciudad del Conocimiento, Carretera Pachuca-Tulancingo Km. 4.5 Colonia Carboneras, Mineral de la Reforma, Hidalgo, México. C.P. 42184
Teléfono: 771 71 720 00 Ext. 40001
direccion_icbi@uaeh.edu.mx, vergarar@uaeh.edu.mx

A mi familia,
especialmente a mi madre y hermanas,
por ser mi apoyo e inspiración.

AGRADECIMIENTOS

A mi madre, F. Martha Ramirez Reyes, por ser mi mayor inspiración. Gracias por guiarme, alentarme y por nunca dejar de creer en mi. Este logro también es tuyo.

A mis hermanas, Liz, Gandy y Yoisi. Gracias por su apoyo y amor incondicional. Es por ustedes que sé el valor de la familia.

A mis abuelos, por cuidarme y por sus consejos para convertirme en la persona que soy ahora.

A mis directores de tesis, Dr. Lao-Tse López Lozano y Dra. Alba Leticia Carrillo Monteverde. Gracias por compartir sus conocimientos, por su paciencia y dedicación en cada asesoría, por guiarme en este proceso y por motivarme para dar lo mejor de mi.

A mis amigos, Rafa, Alfredo, Joe y Fer por siempre caminar a mi lado y permitirme ser parte de sus vidas. Gracias por todas las risas, chistes y aventuras juntos.

A mi pareja, Bu, por tu amor y presencia, especialmente en los momentos difíciles, y por las esperanzas futuras que me motivan a continuar.

Índice general

Resumen

El análisis fenomenológico de algunos modelos de materia oscura requiere el uso secuencial de múltiples programas especializados. Entre los más utilizados se encuentran FeynRules, el cual genera las reglas de Feynman permitidas por el modelo, CalcHEP para calcular los elementos de matriz y micrOMEGAs, el cual evalúa observables cosmológicos. El flujo de trabajo tradicional presenta tres limitaciones principales que incrementan significativamente el tiempo de análisis y la posibilidad de errores: gestión manual de archivos entre herramientas, ejecución secuencial de cálculos repetitivos, y fragmentación en la presentación de resultados.

En este proyecto se desarrolló Amalgama, un programa escrito en Python 3 que gestiona casi sin intervención del usuario la llamada de FeynRules, CalcHEP y micrOMEGAs en un flujo de trabajo unificado. El programa automatiza la generación de scripts de Mathematica para FeynRules y gestiona la creación de proyectos en micrOMEGAs. Implementa paralelización del cálculo de densidad reliquia mediante el módulo `concurrent.futures` y genera automáticamente visualizaciones de resultados. La arquitectura modular facilita futuras extensiones del programa.

La validación se realizó a través del Modelo Estándar más un singlete escalar, comparando resultados con publicaciones establecidas. El análisis de desempeño en tres configuraciones de hardware con 12 y 48 núcleos demostró reducir los tiempos de computo significativamente. En la configuración de mayor capacidad se procesaron más de 2,600 evaluaciones por segundo. El programa reduce un análisis que tomaría días en el proceso tradicional a ejecuciones de minutos. Se desarrolló adicionalmente un instalador automatizado compatible con Arch Linux, Ubuntu, Debian y Fedora que gestiona todas las dependencias necesarias.

Amalgama representa una herramienta validada y funcional que unifica, automatiza y optimiza el análisis de modelos de materia oscura, cumpliendo los objetivos establecidos y proporcionando una base extensible para análisis fenomenológicos futuros.

Capítulo 1: Introducción

La física de partículas y la física en general se han caracterizado históricamente por su naturaleza innovadora, manifestándose tanto en el desarrollo de metodologías novedosas y hallazgos científicos, como en la incorporación de recursos computacionales y algoritmos avanzados. La integración de estos métodos computacionales ha marcado un antes y un después en el desarrollo de conocimientos en el área teórica y experimental al permitirnos realizar cálculos complejos o simulando experimentos que antes se pensaban inaccesibles.

El desarrollo de estas tecnologías destaca por su capacidad de resolver problemáticas específicas, adaptándose continuamente para enfrentar desafíos nuevos que podrían emerger, por ejemplo, durante el proceso de análisis o procesamiento de datos. Bajo esta premisa, el presente trabajo se enfoca en resolver una problemática particular: la optimización del cálculo de la densidad reliquia de materia oscura fría usando FeynRules[4, 6, 13] y micrOMEGAS[3, 7, 8].

1.1. Antecedentes.

La determinación de la abundancia de materia oscura en el universo se convirtió en una de las problemáticas más desafiantes en el área de partículas. Durante décadas se ha abordado este problema mediante la solución de la ecuación de evolución de Boltzmann[5], que describe la evolución de densidad numérica de una partícula en un universo en expansión de tipo Friedmann-Lemaître, aproximadamente plano, lleno de radiación, materia bariónica y materia oscura.

$$\dot{n} = -3Hn - \langle \sigma_{eff} v \rangle (n^2 - n_{eq}^2) \tag{1.1}$$

La ecuación de Boltzmann en la forma anterior representa el marco teórico mínimo para conectar la física de partículas con la cosmología observable. Su importancia radica en incorporar simultáneamente dos procesos físicos fundamentales: el término de expansión $(3Hn)$

describe la dilución cosmológica de partículas debido a la expansión del universo, mientras que el término de interacción $\langle\sigma_{eff}v\rangle(n^2 - n_{eq}^2)$ representa los procesos cuánticos de aniquilación y creación que tienden a mantener el equilibrio térmico.

La interacción entre estos términos produce el mecanismo de *freeze-out*: en el universo primitivo, las interacciones mantienen el equilibrio térmico; al expandirse y enfriarse el universo, la tasa de expansión H eventualmente supera la tasa de interacción. Las partículas dejan de aniquilarse y su densidad se "congela", estableciendo la abundancia reliquia actual.

Esta ecuación describe la transición que fijó la cantidad de materia oscura en el cosmos, permitiendo transformar una propiedad microscópica (sección eficaz de aniquilación) en una predicción cosmológica observable: $\Omega_{\rm DM}h^2$. Esto la convierte en la herramienta indispensable para validar cualquier candidato a materia oscura.

Calcular $\langle\sigma_{eff}v\rangle$ requiere la evaluación de un gran número de amplitudes de aniquilación y coaniquilación[5]. La evolución de la física computacional y la adopción de algoritmos especializados transformó este panorama, permitiendo la automatización y estandarización de cálculos como este.

1.2. Obtención de $\Omega_{\rm DM}h^2$

La densidad reliquia, $\Omega_{DM}h^2$, representa la fracción de densidad crítica del universo correspondiente a la materia oscura en la actualidad. Las mediciones de precisión del Fondo Cósmico de Microondas realizadas por Planck en 2018 establecen su valor en $\Omega_{DM}h^2 = 0{,}120 \pm 0{,}001$[2]. Por otro lado, su determinación teórica requiere resolver la ecuación de evolución de Boltzmann desde el universo primitivo hasta el presente, capturando el proceso de freeze-out que fijó la abundancia actual[23, 24].

Para la obtención de la densidad reliquia es necesario resolver la ecuación de abundancia de materia oscura $Y(T)$, donde $Y(T) = n(T)/s(T)$ representa la densidad numérica de partículas normalizada por la densidad de entropía del universo:

$$\frac{\mathrm{d}Y}{\mathrm{d}T} = \sqrt{\frac{\pi g_*(T)}{45}} M_p \langle\sigma_{eff}v\rangle \left(Y(T)^2 - Y_{eq}(T)^2\right) \tag{1.2}$$

Esta expresión se deriva de la ecuación de Boltzmann presentada anteriormente. La trans-

formación se realiza para expresar la evolución en función de la temperatura T en lugar del tiempo y utilizando la variable de abundancia $Y = n/s$, donde s es la densidad de entropía, lo que elimina el término de dilución cosmológica $3Hn$. En esta nueva formulación M_p es la masa de Planck que surge al relacionar la tasa de expansión H con la temperatura a través de las ecuaciones de Friedmann. El término $\sqrt{\pi g_*(T)/45}$ codifica los grados de libertad relativistas efectivos $g_*(T)$ y la dependencia cosmológica, mientras que $\langle \sigma_{\text{eff}} \nu \rangle$ representa la sección eficaz de aniquilación promediada térmicamente. El factor $(Y^2 - Y_{\text{eq}}^2)$ actúa como fuerza impulsora hacia el equilibrio, análogo al término $(n^2 - n_{\text{eq}}^2)$ de la ecuación original.

La densidad reliquia en la actualidad, en unidades de la densidad crítica, está dada por:

$$\Omega_{DM} h^2 = \frac{\rho_{DM}^0}{\rho_{crit}} h^2 = \frac{m_{DM} s_0 Y_0}{\rho_{crit}} h^2 \tag{1.3}$$

donde $\rho_{crit} = 3H_0^2/8\pi G$ es la densidad crítica actual, s_0 es la densidad de entropía presente, m_{DM} es la masa de la partícula de materia oscura, y Y_0 es el resultado de la integración de la ecuación (1.2) evaluada en el presente[5, 18].

En este trabajo se utiliza MICROMEGAS, código que resuelve la ecuación (1.2) numéricamente sin aproximaciones[8] para obtener la densidad reliquia de candidatos de materia oscura en modelos más allá del Modelo Estándar.

Como se evidenciará a lo largo de este trabajo, ningún software individualmente resulta suficiente para realizar un análisis completo de modelos de materia oscura. Aunque MICROMEGAS ha demostrado ser una herramienta confiable, característica que le ha dado gran popularidad en la comunidad de física de partículas, no puede considerarse el punto de partida directo para modelos más allá del Modelo Estándar, requiriendo un procesamiento previo del modelo. Reconociendo esta limitación, los desarrolladores integraron CALCHEP como generador de eventos para calcular secciones eficaces a nivel árbol[7], entregando a la comunidad una herramienta computacional más completa.

Analizar un modelo más allá del Modelo Estándar requiere el conocimiento de los canales de decaimiento de las nuevas partículas introducidas, así como sus probabilidades. CALCHEP aporta código generado en C que calcula los elementos de matriz requeridos para las evaluaciones numéricas realizadas posteriormente en MICROMEGAS. Sin embargo, esto introduce una nueva complicación: la creación de los archivos de entrada que CALCHEP requiere. El software necesita cuatro archivos fundamentales: `varsN.mdl` (parámetros del modelo),

`funcN.mdl` (restricciones), `prtclsN.mdl` (contenido de partículas) y `lgrngn.mdl` (reglas de Feynman).

La obtención de las reglas de Feynman es, por tanto, el primer paso en el camino hacia la determinación de $\Omega_{DM}h^2$. Paquetes especializados como LanHEP[35], SARAH[37] y FeynRules automatizan este proceso generando las reglas de Feynman en formatos compatibles con múltiples generadores de eventos[4]. Estos paquetes requieren únicamente la declaración del contenido de partículas, parámetros y lagrangianos en la sintaxis adecuada. En esencia, se trata de traducir el modelo físico a un formato interpretable por los generadores de reglas de Feynman.

1.3. Planteamiento del problema.

El cálculo de la densidad reliquia de materia oscura requiere la integración de múltiples herramientas computacionales especializadas, cada una optimizada para una etapa específica del proceso. Si bien esta diversidad permite gran flexibilidad en el análisis, también introduce complicaciones ampliamente reconocidas por los usuarios.

La principal problemática se encuentra en las incompatibilidades entre estos software: diferentes lenguajes de programación, formatos de entrada y salida, e intercambio de datos poco estandarizado. Aunque han existido intentos de estandarización —como el formato UFO (Universal FeynRules Output) desarrollado para unificar la escritura de reglas de Feynman[16], y la integración nativa entre micrOMEGAS y CalcHEP— estos no resuelven completamente las dificultades presentes al momento de trabajar con más de uno de ellos que impactan en la eficiencia del proceso de análisis: tiempos de cómputo extendidos, gestión manual compleja de archivos entre las diferentes aplicaciones, y presentación de los resultados en formatos poco legibles que requieren de una interpretación y tratamiento.

Este trabajo aborda tres limitaciones del proceso actual: la gestión manual de los archivos, la obtención manual y secuencial de los puntos en el espacio de parámetros (m_{DM}, λ_{h-DM}) para generar las gráficas de contorno, y la representación poco legible de los resultados. De estos, el segundo representa el principal cuello de botella en el análisis de modelos, ya que requiere la ejecución manual e independiente (entre sí) de micrOMEGAS para cada punto.

1.4. Propuesta de solución al problema.

Para atender las problemáticas identificadas previamente, se desarrolló una solución que integra tres herramientas especializadas mediante Python, gestionando automáticamente el flujo de trabajo completo del análisis: FeynRules (con Mathematica), CalcHEP y micrOMEGAS.

Esta integración permite al usuario partir únicamente de la definición del modelo teórico y obtener directamente $\Omega_{DM}h^2$ del candidato de materia oscura y realizar gráficas de contorno para explorar regiones del espacio de parámetros donde el modelo resulte viable o de interés particular para su análisis. Adicionalmente, se implementó paralelización en el cálculo de $\Omega_{DM}h^2$, transformando el proceso lineal tradicional en uno paralelo, reduciendo significativamente los tiempos de cómputo al aprovechar las arquitecturas de los computadores modernos.

1.5. Estructura del trabajo.

El producto principal de este trabajo es la creación de Amalgama, denominación que alude al propósito central de fusionar Mathematica, CalcHEP y micrOMEGAS. Este es un programa desarrollado en Python completamente funcional en sistemas operativos Linux, pensado y probado en Debian 13, Ubuntu 24.04 LTS, Fedora 42 y ArchLinux. La implementación incluye un instalador automatizado que gestiona la descarga de dependencias desde repositorios oficiales y su compilación para su uso inmediato. La validación del programa se realizó mediante la implementación del Modelo Estándar extendido con un singlete escalar, comparando la gráfica obtenida con publicaciones centradas en el estudio de este tipo de modelos para verificar su confiabilidad.

El presente trabajo de tesis esta estructurado de la siguiente manera. El Capítulo 2 presenta el estado del arte de la física de partículas, incluyendo el Modelo Estándar y los principales candidatos de materia oscura. El Capítulo 3 describe el software existente para fenomenología en física de partículas y las metodologías actuales de exploración del espacio de parámetros. El Capítulo 4 detalla la metodología empleada en el desarrollo de Amalgama, su funcionalidad completa y proceso de instalación. El Capítulo 5 presenta los resultados obtenidos mediante la aplicación del programa al modelo de singlete escalar y analiza los tiempos de ejecución. Finalmente, el Capítulo 6 expone las conclusiones del trabajo y sus perspectivas futuras.

Capítulo 2: Estado del arte de la física de partículas

La física de partículas constituye el marco teórico fundamental para comprender la estructura más básica de la materia y las fuerzas que generan las interacciones entre ella. A lo largo de más de un siglo, desde el descubrimiento del electrón por J. J. Thomson en 1897[38] hasta la confirmación experimental de la existencia del bosón de Higgs en 2012[1, 11], esta área de la física ha experimentado una evolución constante con la creación, validación y refinamiento de modelos teóricos que buscan explicar las observaciones experimentales.

El Modelo Estándar[21, 33, 39] representa el logro más significativo de esta búsqueda. Actualmente, este modelo es la teoría más sólida que tenemos en física de partículas, con una precisión experimental extraordinaria que ha validado sus predicciones en múltiples experimentos. Las partículas e interacciones descritas en este modelo han logrado explicar satisfactoriamente la mayoría de las observaciones experimentales. No obstante, el Modelo Estándar presenta limitaciones fundamentales que evidencian que es una teoría incompleta. Diversos fenómenos observados experimentalmente permanecen sin una explicación teórica sólida que los respalde, revelando la necesidad de extender esta teoría.

En este capítulo se abordará no solamente el Modelo Estándar y su contenido, sino también una de las incompatibilidades más importantes entre esta teoría y la naturaleza observada: la materia oscura. Se mostrarán las evidencias experimentales que justifican su existencia y, particularmente, cómo la comunidad científica ha intentado desarrollar marcos teóricos que proporcionen una justificación teórica sólida para este fenómeno.

2.1. Modelo estándar.

El Modelo Estándar representa el mayor triunfo de la física de partículas. A través de principios fundamentales de simetría, el Modelo Estándar describe la interacción entre diferentes campos cuánticos y las dinámicas de las partículas asociadas a ellos.

Esta teoría es el resultado de décadas de trabajo teórico y experimental, integradas en un marco unificado que describe tres de las cuatro fuerzas fundamentales de la naturaleza: la fuerza electromagnética, la fuerza nuclear fuerte y la fuerza débil. Es crucial destacar que la fuerza débil actúa tanto en el sector hadrónico (cambiando el sabor de los quarks) como en el leptónico (mediando desintegraciones como la beta y procesos como la dispersión neutrino-electrón). Componentes teóricos como la Electrodinámica Cuántica (QED, por sus siglas en inglés), que describe las interacciones electromagnéticas mediante el intercambio de fotones; la Cromodinámica Cuántica (QCD, por sus siglas en inglés), que explica las interacciones fuertes entre quarks a partir del intercambio de gluones; la Teoría Electrodébil, que unifica las fuerzas electromagnética y débil; y el Mecanismo de Higgs, responsable de la generación de masa de las partículas, constituyen los pilares fundamentales sobre los cuales se construyó el Modelo Estándar.

2.1.1. Partículas fundamentales.

El descubrimiento del electrón por J. J. Thomson en 1897 marcó el inicio de la física de partículas moderna, revelando la existencia de constituyentes fundamentales más allá del átomo. Sin embargo, fue el descubrimiento del pión en 1947 por Cecil Frank Powell[26] lo que demostró la existencia de partículas mediadoras de fuerza, conduciendonos poco a poco al desarrollo teórico del Modelo Estándar. El Modelo Estándar clasifica las partículas en dos categorías principales según su espín: fermiones y bosones. Los fermiones, con espín semientero, constituyen los bloques fundamentales de la materia, mientras que los bosones, con espín entero, actúan como mediadores de las fuerzas fundamentales.

Fermiones.

A las partículas del Modelo Estándar con espín $1/2$ (en unidades naturales) se les denomina fermiones y, como se mencionó, constituyen los bloques fundamentales de la materia. Los fermiones se clasifican en dos categorías: quarks y leptones, que se diferencian por las fuerzas bajo las cuales interactúan y sus cargas asociadas.

Existen seis quarks conocidos: up (u), down (d), charm (c), strange (s), top (t) y bottom (b). Cada uno representa un sabor (o flavor) diferente, y pueden transformarse entre sí mediante la interacción débil. Los quarks poseen carga de color, lo que les permite interactuar a través de la fuerza nuclear fuerte. Esta carga es análoga a la carga eléctrica, pero en lugar de tener valores positivos o negativos, adopta seis estados posibles, tres de los cuales llamaremos rojo, y azul, mientras que los tres restantes corresponden a los opuestos a los anteriores[20]. La interacción entre quarks mediada por gluones es descrita por la Cromodinámica Cuántica (QCD, por sus siglas en inglés)[27].

Generación	**Quarks**	**Masa (GeV)**	**Carga eléctrica (e)**
Primera	$\begin{pmatrix} u \\ d \end{pmatrix}$	$(2{,}16 \pm 0{,}07) \times 10^{-3}$ $(4{,}70 \pm 0{,}07) \times 10^{-3}$	$2/3$ $-1/3$
Segunda	$\begin{pmatrix} c \\ s \end{pmatrix}$	$1{,}273 \pm 0{,}005$ $(93{,}5 \pm 0{,}8) \times 10^{-3}$	$2/3$ $-1/3$
Tercera	$\begin{pmatrix} t \\ b \end{pmatrix}$	$172{,}57 \pm 0{,}9$ $4{,}183 \pm 0{,}007$	$2/3$ $-1/3$

Tabla 2.1: Quarks del Modelo Estándar organizados por generaciones. Los valores de masa corresponden a los reportados por PDG[28].

El segundo tipo de fermiones son los leptones. A diferencia de los quarks, estos no poseen carga de color y, por lo tanto, no interactúan mediante la fuerza nuclear fuerte. Los leptones se clasifican en seis partículas: tres leptones cargados—electrón (e), muón (μ) y tau (τ)—y tres neutrinos—neutrino del electrón (ν_e), neutrino del muón (ν_μ) y neutrino del tau (ν_τ). Los neutrinos carecen de carga eléctrica, mientras que los leptones cargados poseen carga eléctrica -1, en unidades de la carga elemental.

Generación	**Leptones**	**Masa (GeV)**	**Carga eléctrica (e)**
Primera	$\begin{pmatrix} e \\ \nu_e \end{pmatrix}$	$(5{,}11 \pm 0{,}00) \times 10^{-4}$ $< 8 \times 10^{-10}$	-1 0
Segunda	$\begin{pmatrix} \mu \\ \nu_\mu \end{pmatrix}$	$(1{,}057 \pm 0{,}000) \times 10^{-1}$ $< 8 \times 10^{-10}$	-1 0
Tercera	$\begin{pmatrix} \tau \\ \nu_\tau \end{pmatrix}$	$1{,}777 \pm 0{,}001$ $< 8 \times 10^{-10}$	-1 0

Tabla 2.2: Leptones del Modelo Estándar organizados por generaciones. Los valores de masa corresponden a los reportados por PDG[28].

Las tablas 2.1 y 2.2 presentan la masa y carga eléctrica de los fermiones del Modelo Están-

dar. Estas propiedades muestran un patrón que refleja una estructura ordenada: los fermiones se agrupan en tres generaciones, donde cada generación contiene fermiones con una masa mayor a sus homólogas en generaciones anteriores. Las cargas eléctricas se mantienen constantes dentro de cada tipo de fermión en todas las generaciones: los quarks poseen cargas de $^2/_3$ y $-^1/_3$, mientras que los leptones cargados tienen carga -1 y los neutrinos son eléctricamente neutros, dejando a la masa como la única propiedad que nos permite diferenciar entre los elementos de cada familia.

La materia ordinaria —es decir, todos los elementos de la tabla periódica y la química convencional— del universo observable está constituida exclusivamente por fermiones de la primera generación: quarks up y down (formando neutrones y protones) y electrones[22].

Bosones.

Los bosones en el Modelo Estándar son partículas con espín entero que se clasifican en dos categorías, bosones de norma y bosones escalares.

Los bosones de norma poseen espín $s = 1$ y actúan como mediadores de tres de las cuatro fuerzas fundamentales de la naturaleza. Entre ellos se encuentran: el fotón (γ), mediador de la fuerza electromagnética; los ocho gluones (g), responsables de la fuerza nuclear fuerte; y los bosones $W^{\pm}$ y Z, mediadores de la fuerza débil.

Bosón de Norma	**Masa**	**Carga eléctrica(e)**
Fotón	$0\,eV$	0
Gluon	$0\,eV$	0
$W^{\pm}$	$80{,}3692 \pm 0{,}0133\,GeV$	± 1
Z	$91{,}1880 \pm 0{,}0020\,GeV$	0

Tabla 2.3: Masa y carga eléctrica de los bosones de norma en el Modelo Estándar. La masa de los bosones $W^{\pm}$ y Z es la obtenida en mediciones experimentales[28].

Para el caso de los bosones escalares, con espín $s = 0$, el Modelo Estándar solo contempla el bosón de Higgs. Con una masa de $125{,}20 \pm 0{,}11\,GeV$ y eléctricamente neutro, es el bosón más masivo en este marco. Este bosón es responsable de la generación de las masas de las partículas elementales mediante el mecanismo de Higgs.

2.2. Principales problemas.

A pesar de que el Modelo Estándar describe con precisión una gran variedad de fenómenos observados experimentalmente, otros permanecen sin respuesta dentro de esta teoría. Además, el modelo presenta hasta 28 parámetros libres cuyo origen y valores específicos carecen de justificación teórica, introduciendo cierto grado de arbitrariedad en la teoría. Entre estos parámetros se incluyen:

- 6 masas de quarks,
- 6 masas de leptones,
- 3 constantes de acoplamiento (intensidad de las interacciones),
- 3 ángulos de mezcla y 1 fase de CP en el sector de quarks,
- 3 ángulos de mezcla y entre 1-3 fases de CP en el sector de leptones,
- 1 masa del bosón de Higgs,
- 1 acoplamiento cuártico en el sector escalar, y
- 1 ángulo de vacío de QCD.

En la sección 2.1.1 se describieron las tres generaciones de fermiones conocidas experimentalmente. Sin embargo, el Modelo Estándar no proporciona explicación para la existencia de estas generaciones adicionales más allá de la primera, ni explica el hecho de que solo existan precisamente estas tres generaciones y no un número diferente. Tampoco hay explicación a la masa de cada familia y su variación entre ellas.

Otros problemas significativos incluyen la violación de la simetría CP observada experimentalmente, la imposibilidad de incorporar la gravedad dentro del marco teórico del Modelo Estándar, los mecanismos responsables de la bariogénesis que expliquen la asimetría materia-antimateria del universo —los cuales requieren necesariamente la violación del número bariónico, la violación de las simetrías C y CP, y una ruptura del equilibrio térmico en el universo primitivo, conocidas como condiciones de Sakharov[32]— y, de particular relevancia para este trabajo, la naturaleza y composición de la materia oscura, cuya existencia está respaldada por un gran número de observaciones y que carece de candidatos dentro del contenido de partículas del Modelo Estándar.

2.3. Materia Oscura.

Diversas observaciones astronómicas y cosmológicas[10, 14, 19] apuntan hacia la existencia de un tipo de materia que no interactúa electromagnéticamente: las curvas de rotación de galaxias, el movimiento de galaxias dentro de cúmulos, la observación de lentes gravitacionales y el análisis del Fondo Cósmico de Microondas (CMB, por sus siglas en inglés) proporcionan evidencia convincente de que el universo contiene más materia de la que podemos observar directamente. Esta información llega a nosotros gracias a los fotones (luz, CMB, lentes) y neutrinos. El análisis de estas señales nos proporciona una panorama sólido de la estructura del universo. A estos se le ha sumado la detección de ondas gravitacionales, aunque no permiten inferir propiedades de materia oscura.

El análisis del CMB y los estudios de nucleosíntesis primordial del Big Bang descartan la posibilidad de que esta materia sea de naturaleza bariónica—es decir, partículas del SM como los protones y neutrones—, sugiriendo que debe estar compuesta por partículas aún no identificadas. Esta materia oscura constituye aproximadamente el 26 % de la *energía-masa* del universo[14], mientras que la materia bariónica representa menos del 5 %, planteando la pregunta fundamental de cómo una cantidad de materia tan grande no ha llegado a ser detectada durante casi un siglo de búsqueda experimental.

2.3.1. Propiedades.

Aunque no existe certeza sobre su naturaleza, el problema de la materia oscura se aborda comúnmente desde la hipótesis de que está compuesta por partículas elementales aún no descubiertas más allá del Modelo Estándar. Si bien no conocemos todas sus propiedades, las observaciones establecen restricciones claras: deben ser partículas que no interactúan de forma electromagnética puesto que no emiten ni absorben luz, interactúan gravitacionalmente, y presentan interacción nula o muy débil consigo mismas.

La materia oscura y sus candidatos se clasifican en tres categorías: materia oscura caliente (HDM, por sus siglas en inglés), materia oscura tibia (WDM, por sus siglas en inglés) y materia oscura fría (CDM, por sus siglas en inglés). Esta clasificación depende de si las partículas eran relativistas, semirrelativistas o no relativistas cuando se desacoplaron del equilibrio térmico[25].

La materia oscura fría (CDM), que es el enfoque central de este trabajo, consiste en partículas estables y no relativistas que desempeñan un papel fundamental en la formación

de estructuras cósmicas. Los modelos basados en CDM reproducen con mayor precisión las observaciones de distribución y concentración de materia[25]. Esto ha motivado el desarrollo de numerosos modelos teóricos y herramientas computacionales especializados en el análisis de candidatos de materia oscura fría.

2.3.2. Descubrimiento de la materia oscura.

La primera evidencia de materia no luminosa fue proporcionada por Fritz Zwicky en 1933 al estudiar el cúmulo de Coma. Utilizando el teorema del virial y asumiendo que el cúmulo había alcanzado un estado mecánicamente estacionario, Zwicky estimó que la densidad debía ser al menos 400 veces mayor a la estimada considerando únicamente la materia luminosa[40]. Aunque estas mediciones carecían de precisión suficiente para ser concluyentes, establecieron la primera sugerencia de la existencia de materia oscura.

La evidencia definitiva llegó en la década de 1970 con los trabajos de Vera Rubin y colaboradores sobre las curvas de rotación de galaxias espirales. En 1980, Rubin, Ford y Thonnard publicaron un estudio de 21 galaxias tipo Sc —una categoría dentro de la clasificación morfológica de Hubble que se caracteriza por poseer un bulbo central pequeño, brazos espirales muy sueltos y abiertos, y una gran abundancia de gas y polvo que favorece una elevada formación estelar—, demostrando que las anomalías en las velocidades de rotación en las regiones externas no eran casos aislados[31]. Sus curvas de rotación mostraban velocidades que permanecían constantes o continuaban creciendo incluso en las regiones más alejadas del centro galáctico, contradiciendo las predicciones hechas contemplando únicamente la materia visible en las galaxias. Este comportamiento proporcionó la primera evidencia sólida y ampliamente aceptada de la existencia de materia oscura.

Desde entonces, múltiples observaciones han reforzado esta conclusión: el análisis del fondo cósmico de microondas, estudios de lentes gravitacionales, la dinámica de los cúmulos de galaxias y simulaciones de formación de estructura[34] a gran escala convergen en la necesidad de la existencia de un tipo de materia oscura dominante en el universo. Estas observaciones han establecido restricciones precisas sobre las propiedades de materia oscura. Entre las más importantes tenemos la restricción del valor de la densidad reliquia a $\Omega_{DM}h^2 = 0{,}120 \pm 0{,}001$ hecha por la colaboración Planck en el 2018[2].

La determinación de su densidad reliquia ($\Omega_{DM}h^2$) se ha convertido en una tarea fundamental para validar modelos teóricos con candidatos de materia oscura y comparar sus predicciones con las restricciones observadas.

2.3.3. Límites y estrategias de detección.

Las evidencias observacionales confirman la existencia de materia oscura y establecen algunas de sus propiedades fundamentales, pero no revelan su naturaleza física específica. Esta limitación ha motivado el desarrollo de tres estrategias experimentales para su detección.

Los experimentos de detección directa buscan señales de materia oscura a través de la dispersión entre partículas de materia oscura y materia ordinaria. La detección indirecta rastrea en el espacio profundo indicios de decaimiento o aniquilación buscando excesos de rayos gamma, neutrinos, positrones u otras partículas. Finalmente, los experimentos en colisionadores como el LHC buscan producir partículas de materia oscura rastreando pérdidas de energía en colisiones.

Estas estrategias nos han permitido restringir el espacio de parámetros en modelos de materia oscura. La comparación entre las predicciones de $\Omega_{DM}h^2$ calculadas para modelos específicos y los límites experimentales se ha convertido en la mejor estrategia para identificar regiones viables del espacio de parámetros, motivando la necesidad de herramientas computacionales eficientes para estos cálculos.

2.4. Modelos más allá del modelo estándar para materia oscura.

La naturaleza de la materia oscura ha motivado el desarrollo de numerosos modelos teóricos que extienden el Modelo Estándar. Entre los candidatos más estudiados se encuentran las partículas masivas débilmente interactuantes (WIMPs)[30], y los axiones[17], propuestos originalmente como solución al problema de CP fuerte en QCD y que posteriormente se identificaron como candidatos viables de materia oscura.

Una alternativa más simple a estos dos candidatos consiste en extender el Modelo Estándar mediante la adición de campos escalares singletes bajo su grupo de simetría $SU(3) \times SU(2) \times U(1)$[36]. Estos modelos introducen uno o más campos escalares que no transforman bajo las simetrías de norma del Modelo Estándar, pero que pueden acoplarse al sector de Higgs a través de un portal de Higgs[29]. Esta extensión coloca al singlete escalar como un candidato de materia oscura fría.

El modelo del singlete escalar es de particular interés en este trabajo por algunas razones:

requiere la introducción de un número reducido de parámetros libres, permite calcular $\Omega_{DM}h^2$ con relativa facilidad, y puede ser fácilmente implementado en herramientas computacionales. Por estas razones, constituye un caso de estudio ideal para el desarrollo y validación de herramientas computacionales, siendo éste el enfoque del presente trabajo.

Capítulo 3: Fenomenología en física de partículas.

La fenomenología en física de partículas constituye el puente entre la teoría y el experimento. Esta busca determinar qué extensiones teóricas son viables mediante la comparación de sus predicciones con datos experimentales y restricciones observacionales establecidas.

Para un modelo en particular, el análisis fenomenológico requiere calcular observables físicas y compararlas con mediciones experimentales. Para modelos de materia oscura, entre las observables se encuentran la densidad reliquia ($\Omega_{DM}h^2$), y secciones eficaces de dispersión. La obtención de estas propiedades tiene como punto de partida la obtención de las reglas de Feynman del modelo, que describen todas las interacciones permitidas entre las partículas.

Tradicionalmente, la obtención de las reglas de Feynman y el cálculo de observables era un proceso manual, laborioso y propenso a errores. Para modelos con múltiples partículas y varios canales de interacción, el número de diagramas de Feynman que contribuyen a un proceso dado puede ser considerable, haciendo (casi) imposible el análisis puramente analítico. Esta complejidad computacional ha motivado el desarrollo de herramientas especializadas que automatizan diferentes etapas del análisis fenomenológico, desde la generación de reglas de Feynman hasta el cálculo numérico de observables cosmológicos y de colisionadores.

3.1. Herramientas computacionales para fenomenología.

El análisis fenomenológico moderno de modelos con materia oscura se apoya en un ecosistema de herramientas computacionales especializadas, cada una diseñada para abordar etapas específicas del proceso de cálculo. En este trabajo se emplean tres herramientas fundamentales que, en conjunto, permiten calcular observables como la densidad reliquia de materia oscura: FeynRules para la generación automática de reglas de Feynman, Cal-

cHEP para el cálculo de elementos de matriz y secciones eficaces, y micrOMEGAs para la evaluación numérica de la densidad reliquia. A continuación se describen estas herramientas y sus funciones específicas.

3.1.1. FeynRules.

FeynRules es un paquete de Mathematica diseñado para derivar automáticamente las reglas de Feynman de un modelo de física de partículas a partir de su lagrangiano[12]. El usuario proporciona la definición del modelo especificando el contenido de partículas, parámetros del modelo y la expresión del lagrangiano en notación matemática estándar. FeynRules procesa esta información y genera automáticamente todos los vértices de interacción del modelo.

Una característica fundamental de FeynRules es su capacidad de exportar las reglas de Feynman en múltiples formatos compatibles con diferentes generadores de eventos y calculadoras de procesos. Entre estos formatos se encuentra CalcHEP/CompHEP, FeynArts/FormCalc, y el formato UFO (Universal FeynRules Output)[16], este último diseñado específicamente para proporcionar un estándar universal de intercambio de información entre diferentes herramientas fenomenológicas. Para este trabajo, se utiliza la exportación en formato CalcHEP, que permite la integración directa con micrOMEGAs.

3.1.2. CalcHEP.

CalcHEP es un paquete computacional diseñado para el cálculo automático de elementos de matriz y secciones eficaces en física de partículas[9]. A partir de las reglas de Feynman de un modelo, CalcHEP genera todos los diagramas de Feynman que contribuyen a un proceso específico y calcula la amplitud de dispersión correspondiente a nivel árbol.

CalcHEP requiere como entrada cuatro archivos que describen el modelo: `varsN.mdl` conteniendo los parámetros del modelo, `funcN.mdl` con las restricciones y relaciones entre parámetros, `prtclsN.mdl` especificando el contenido de partículas y sus propiedades, y `lgrngn.mdl` con las reglas de Feynman. Estos archivos pueden ser generados manualmente o, más comúnmente, mediante la exportación desde FeynRules.

Una característica relevante de CalcHEP es su capacidad para generar código en C que implementa el cálculo de elementos de matriz para procesos específicos. Este código generado puede ser utilizado por otras herramientas, como micrOMEGAs, para realizar

cálculos numéricos que permiten la obtención de la densidad reliquia. Esta funcionalidad establece a CalcHEP como un componente intermedio esencial en la cadena computacional que conecta la definición del modelo con el cálculo de observables.

3.1.3. micrOMEGAs.

micrOMEGAs es un software especializado en el cálculo de observables para candidatos de materia oscura[7, 8]. Su función principal es resolver numéricamente la ecuación de Boltzmann que describe la evolución de la densidad numérica de partículas de materia oscura en el universo temprano, permitiendo calcular la densidad reliquia $\Omega_{DM}h^2$.

Para realizar estos cálculos, micrOMEGAs necesita conocer las interacciones del candidato de materia oscura con las partículas del Modelo Estándar y otras partículas nuevas del modelo. Esta información se obtiene mediante la integración de las librerías de CalcHEP que permiten calcular todas las secciones eficaces relevantes de aniquilación y co-aniquilación que contribuyen a la densidad reliquia.

Una ventaja significativa de micrOMEGAs es que resuelve la ecuación de evolución de la abundancia de materia oscura sin aproximaciones[8], considerando automáticamente todos los canales de aniquilación y co-aniquilación relevantes del modelo. Sin embargo, el uso de micrOMEGAs requiere que el modelo haya sido previamente procesado y exportado en el formato apropiado desde FeynRules, estableciendo una dependencia casi obligatoria entre estas tres herramientas.

3.2. Proceso tradicional de análisis y sus limitaciones.

El análisis fenomenológico de modelos de materia oscura mediante las herramientas descritas sigue un proceso secuencial bien establecido. El usuario debe primero implementar el modelo en FeynRules utilizando Mathematica, exportar las reglas de Feynman en formato CalcHEP, transferir manualmente estos archivos al directorio apropiado de micrOMEGAs, compilar el modelo dentro de micrOMEGAs, y finalmente ejecutar los cálculos para obtener $\Omega_{DM}h^2$ y otros observables. Este proceso presenta varias limitaciones significativas que impactan la eficiencia del análisis de cualquier modelo.

La primera limitación es la gestión manual de archivos entre diferentes plataformas. Cada herramienta requiere que el usuario gestione y copie archivos en directorios específicos. La

segunda es el tiempo de cómputo extendido puesto que para explorar el espacio de parámetros de un modelo, cada punto debe ser evaluado individualmente y de manera secuencial, desaprovechando las capacidades de procesamiento paralelo de los ordenadores modernos.

Una tercera limitación importante es la presentación fragmentada de resultados. Los datos numéricos generados por micrOMEGAs se despliegan en la terminal del ordenador, requiriendo procesamiento adicional para su visualización y análisis. El usuario debe escribir scripts separados para leer estos datos, procesarlos y generar gráficas que permitan identificar regiones viables del espacio de parámetros. Esto incrementa significativamente el tiempo total de análisis de modelos teóricos.

Las limitaciones anteriores motivaron el desarrollo de Amalgama, una herramienta que integra y automatiza este proceso completo, abordando específicamente cada una de estas problemáticas mediante la unificación del flujo del análisis de modelos con materia oscura a través de las herramientas anteriores en una sola interfaz, la implementación de cálculos paralelos, y la generación automática de visualizaciones de resultados.

Capítulo 4: Metodología y diseño de AMALGAMA

En los capítulos anteriores se presentaron los fundamentos teóricos de la materia oscura y las herramientas computacionales existentes para su análisis fenomenológico. Como se discutió en el Capítulo 3, el proceso tradicional de análisis presenta limitaciones significativas en términos de gestión manual de archivos, tiempos de cómputo y presentación de resultados.

Este capítulo describe el diseño e implementación de AMALGAMA, una herramienta desarrollada para automatizar e integrar el proceso completo de análisis de modelos de materia oscura. Se presenta la arquitectura general del programa, las decisiones de diseño fundamentales y la descripción de sus funciones principales. Los detalles técnicos de implementación y el código fuente completo se encuentran en el Apéndice B.

4.1. Descripción del programa.

AMALGAMA es un programa desarrollado en Python 3 que integra tres herramientas especializadas para automatizar el análisis de modelos con materia oscura: FEYNRULES (paquete de MATHEMATICA para generar reglas de Feynman), CALCHEP (que obtiene elementos de matriz), y MICROMEGAS (software para obtener observables). El objetivo principal de AMALGAMA es eliminar la gestión manual de archivos entre estas herramientas y optimizar el análisis de modelos reduciendo los tiempos de cómputo mediante la paralelización de procesos. Además, la arquitectura modular del programa facilita futuras actualizaciones, extensiones e incorporación de nuevas funcionalidades.

4.1.1. Objetivos de diseño.

El diseño de AMALGAMA planteó cumplir con varios objetivos específicos. En primer lugar, se buscó explotar la relación existente entre MICROMEGAS y FEYNRULES para crear

un flujo de trabajo continuo siguiendo la metodología habitual en el análisis de modelos. Para esto se desarrollaron interfaces de comunicación entre programas que operan en diferentes lenguajes y entornos de ejecución.

Segundo, simplificar la interacción del usuario con el sistema a través de una interfaz de línea de comandos clara e intuitiva. A diferencia del proceso tradicional que requiere de un conocimiento detallado de la sintaxis, formatos y protocolos de entrada y salida de datos, AMALGAMA implementa menús interactivos que guían al usuario a través de cada etapa del análisis.

Tercero, desarrollar una aplicación con una arquitectura modular que facilite la incorporación de nuevas funciones. El código de AMALGAMA separa cada función en módulos independientes para diversas tareas. Esto permite mantener y extender el programa sin afectar las demás funciones.

Cuarto, establecer un formato estandarizado para los datos de salida. Los resultados se almacenan en archivos de texto tabulados, facilitando proceso de análisis y de obtención de gráficas.

Quinto, reducir significativamente los tiempos de ejecución mediante la paralelización de tareas recurrentes independientes como el cálculo de la densidad reliquia, aprovechando los procesadores multinúcleo modernos.

Finalmente, proporcionar un instalador automatizado que gestione todas las dependencias necesarias para su funcionamiento: compiladores, librerías y los paquetes especializados.

El desarrollo de AMALGAMA siguió un modelo incremental, implementando primero las funcionalidades básicas y desarrollando cada una de ellas según las necesidades.

4.2. Arquitectura general.

AMALGAMA implementa una arquitectura modular organizada en cuatro componentes principales. La Figura 4.1 ilustra el flujo de trabajo general del programa y la interacción entre los diferentes módulos.

El primer componente es el **módulo de gestión de modelos**, responsable de localizar, importar y organizar los archivos que contienen la definición de modelos de física de partí-

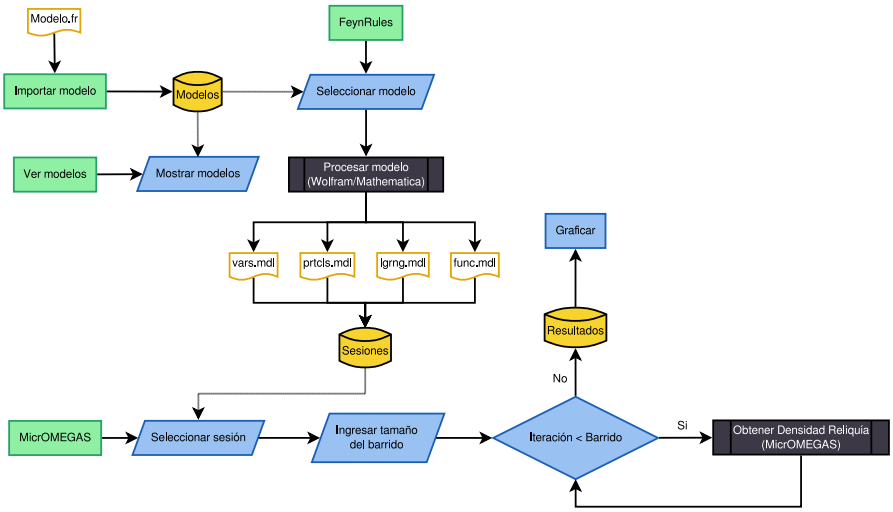


Figura 4.1: Diagrama de flujo de la funcionalidad general de Amalgama. En verde se indican las funciones principales desarrolladas en Python, y en gris oscuro la llamada a programas externos.

culas. Este módulo implementa un sistema de búsqueda que permite al usuario seleccionar modelos previamente definidos o importar nuevos desde ubicaciones arbitrarias en el sistema de archivos.

El segundo componente es la **interfaz con FeynRules** que automatiza la generación de reglas de Feynman a partir de la definición del lagrangiano del modelo. Este módulo ejecuta código de Mathematica desde un entorno gestionado en Python mediante la generación de scripts que configuran la ejecución de FeynRules con los parámetros apropiados y exportan los resultados en formato CalcHEP.

El tercer componente es la **interfaz con micrOMEGAs**, que gestiona la creación de proyectos, configuración de módulos de cálculo y compilación del código generado por CalcHEP. Este módulo automatiza tareas que tradicionalmente requieren múltiples pasos manuales como la verificación de archivos de entrada, creación de directorios de trabajo, selección de observables a calcular y compilación del ejecutable principal. En esta interfaz se incluye la evaluación de $\Omega_{DM}h^2$ a través de múltiples núcleos del procesador. Este módulo coordina la generación de conjuntos independientes de parámetros del modelo y la ejecución simultánea de micrOMEGAs para cada punto del espacio de parámetros.

El último componente es el **módulo de visualización**, que genera automáticamente gráficas de contorno a partir de los datos calculados mediante la creación de scripts de

MATHEMATICA que procesan los resultados numéricos y producen visualizaciones en formato PDF.

Estos cuatro componentes operan de manera secuencial pero cada uno está diseñado como módulo independiente, facilitando su mantenimiento y extensión futura. La comunicación entre módulos se realiza mediante archivos intermedios. Cada sesión de análisis genera un identificador único basado en la fecha y hora de ejecución, permitiendo mantener resultados de múltiples análisis sin conflictos.

4.3. Gestión de modelos

El análisis de modelos de materia oscura comienza con la definición del modelo teórico a estudiar. AMALGAMA implementa un sistema de gestión que permite importar, organizar y trabajar con archivos que contienen la descripción completa del contenido de partículas, parámetros y lagrangianos en la sintaxis requerida por FEYNRULES.

4.3.1. Estructura de almacenamiento

Los modelos utilizados por AMALGAMA se almacenan en el directorio `Models/` dentro de la carpeta principal del programa, facilitando la localización y administración de múltiples modelos. El programa reconoce archivos con extensión `.fr`, formato utilizado por FEYNRULES para la definición de modelos.

La arquitectura del sistema de gestión fue diseñada teniendo en mente la escalabilidad del software, permitiendo el uso archivos con diferentes extensiones para dar soporte a otros generadores de reglas de Feynman como SARAH y LANHEP.

4.3.2. Importación de modelos.

El módulo de importación permite al usuario integrar nuevos modelos desde una ubicación arbitraria dentro del equipo hacia AMALGAMA. Esta importación se realiza mediante la función `find_files` la cual toma el nombre del modelo y lo busca en todos los directorios dentro del ordenador y guarda las coincidencias en la lista `matches`.

```
1    def find_files(filename, search_path):
2        matches = []
3        for root, dirs, files in os.walk(search_path):
4            if filename in files:
```

```
            matches.append(os.path.join(root, filename))
    matches.append("Volver a buscar")
    matches.append("Salir")
    return matches
```

Código 4.1: Función `find_files` extraida del código de AMALGAMA.

El sistema implementa una búsqueda recursiva mediante `os.walk()` que recorre los directorios del sistema. Las coincidencias se presentan en un menú interactivo que incluye la opción de repetir la búsqueda o salir al menú principal.

Cuando un modelo es seleccionado el sistema hace una validación que verifica la existencia del archivo en el directorio de destino para evitar sobreescritura. Esto se muestra en el Código 4.2.

```
def move_files(filename, src_path, dst_path):
    if os.path.isfile(dst_path + "/" + filename) is True:
        print("\nEl archivo ya existe en " + dst_path)
        input("Presione Enter para salir...")
    else:
        shutil.copy2(src_path, dst_path, follow_symlinks=True)
        print("\nEl archivo fue movido con exito a " + dst_path)
        input("Presione Enter para salir...")
```

Código 4.2: Función `move_files` extraída del codigo de AMALGAMA

4.3.3. Visualización de modelos disponibles

Una vez importados, los modelos dentro de la carpeta `Models` pueden visualizarse gracias a la función `find_files_wth_ext()`. La lista de modelos se genera escaneando el directorio y filtrando por extensión, mostrando únicamente los modelos válidos disponibles para su análisis.

```
def find_files_wth_ext(search_path, ext):
    find_files = []
    for file in os.listdir(search_path):
        if file.endswith(".fr"):
            find_files.append(file)
    return find_files
```

Código 4.3: Función `find_files_wth_ext` extraída del codigo de AMALGAMA

4.4. Integración con FeynRules.

Una de las subrutinas principales dentro de Amalgama es la ejecución de FeynRules. Este es el primer paso para comenzar a estudiar un modelo más allá del Modelo Estándar y para ello se utiliza el paquete FeynRules para Mathematica. Este se encarga de calcular las reglas de Feynman en el espacio de momentos de un modelo QFT escrito previamente en un archivo `.fr`.

La integración de esta herramienta en Amalgama presentó varios desafíos técnicos que requirieron soluciones específicas para lograr la automatización completa del proceso.

4.4.1. Desafíos de automatización.

Para poder implementar este paquete en Amalgama hubo algunas consideraciones importantes que realizar. El principal desafío radica en que FeynRules es un paquete de Mathematica, mientras que Amalgama está escrito en Python. Esta incompatibilidad requirió implementar estrategias de comunicación entre ellos que permitiera controlar la ejecución de código escrito para Mathematica en Python de forma automática.

El segundo desafío esta relacionado con la detección de los lagrangianos dentro del archivo con el modelo. Los lagrangianos están escritos con la sintaxis de Mathematica y su selección requiere, habitualmente, la intervención directa del usuario a través de un script en Mathematica.

El tercer desafío concierne a la gestión de rutas y directorios. Mathematica debe conocer la ubicación exacta del paquete FeynRules dentro del sistema, la ubicación del modelo a procesar, y el directorio donde se almacenarán los datos. Estas rutas varían entre instalaciones y deben configurarse en cada ejecución.

4.4.2. Estrategia de solución.

Priorizar el rendimiento sobre lo visual condujo a la decisión de usar el kernel de Mathematica desde la terminal y no desde el programa en sí. Para que toda entrada de modelo pueda ser automatizada se creo un script plantilla el cual es ejecutado por Mathematica. La estructura base de este script se aprecia en el Código 4.4.

```
$FeynRulesPath = SetDirectory[]
<<FeynRules`
SetDirectory[ ]
LoadModel[]
FeynmanRules[]
(*WriteCHOutput[]*)
(*WriteFeynArtsOutput[]*)
(*WriteSHOutput[]*)
(*WriteLaTeXOutput[]*)
(*WriteUFO[]*)
(*WriteWOOutput[]*)
```

Código 4.4: Script `Plantilla_FeynRules_Script.m` en blanco utilizado para automatizar la generación de las reglas de Feynman.

AMALGAMA toma este script, realiza una copia y lo modifica escribiendo el directorio correcto donde se encuentra el modelo, el paquete FEYNRULES, el o los lagrangianos cuyas reglas de Feynman serán calculadas y establece el formato de salida.

Para que los lagrangianos puedan ser detectados automáticamente por AMALGAMA, es necesario declararlos dentro del archivo del modelo con marcadores específicos. Esta convención se implementa al final del archivo `.fr` de la siguiente forma:

```
(***Lagrangians***)
(*Lagrangian_01*)
(*Lagrangian_02*)
(***End***)
```

Código 4.5: Declaracion de los lagrangianos en un archivo genérico `model.fr` para su recuperación.

Puede apreciarse que las lineas aparecen como comentarios en la sintaxis de MATHEMATICA de forma que no alteran la lógica del programa. La clave en la detección de los Lagrangianos radica en la escritura de estos entre las lineas `(***Lagrangians***)` y `(***End***)` ya que es donde AMALGAMA busca. Es importante recalcar que todo lagrangiano debió ser previamente declarado de la forma habitual dentro del cuerpo principal del modelo.

La función `create_script` es la encargada de realizar la copia del script plantilla, editarlo y ejecutarlo. La función recibe los directorios necesarios como el directorio donde se alojan los modelos, la ubicación del paquete FEYNRULES y el directorio donde se guardarán los archivos, además de los lagrangianos y formato de salida elegido por el usuario. El script plantilla puede observarse en el Apéndice C.

Para el guardado de los archivos se crea una carpeta dentro del directorio `Sesiones` con el nombre `Sesion_ID` donde el identificador es generado a través del módulo `datetime`. Para trabajar con micrOMEGAs es necesario que el usuario seleccione la opción CalcHEP/CompHEP como formato de salida. La función completa que gestiona este proceso, incluyendo la creación de directorios y la invocación de Mathematica, se encuentra en el Apéndice B.

4.4.3. Ejecución y verificación.

Una vez generado el script personalizado, Amalgama invoca el kernel de Mathematica desde la terminal mediante el comando `math -script`. Esta ejecución captura toda la salida del proceso en tiempo real, permitiendo al programa detectar errores o advertencias generadas por FeynRules y mostrarlas al usuario.

FeynRules genera cinco archivos `.mdl` cuando el procesamiento es exitoso. Los principales archivos en el formato compatible con CalcHEP son: `vars.mdl` que contiene los parámetros del modelo, `func.mdl` con las restricciones y relaciones entre parámetros, `prtcls.mdl` especificando el contenido de partículas y sus propiedades, y `lgrng.mdl` con las reglas de Feynman. El quinto archivo, `extlib.mdl`, contiene definiciones de librerías externas si el modelo las requiere.

Si existe algún error en el archivo del modelo original, FeynRules puede generar alguno, ninguno, o archivos incompletos. Debido a ello, Amalgama implementa una verificación de la existencia de los cinco archivos para garantizar que el procesamiento fue correcto. Si cualquiera de ellos falta, el programa notifica al usuario que ocurrió un error durante la derivación de reglas de Feynman.

Los archivos generados se almacenan en un subdirectorio dentro de la sesión correspondiente. FeynRules agrega automáticamente el sufijo '`-CH`' al nombre de la carpeta para indicar que los archivos corresponden al formato compatible con CalcHEP. Esta convención de nomenclatura facilita la posterior localización de los archivos cuando el usuario ejecuta la rutina de micrOMEGAs, ya que Amalgama puede identificar automáticamente qué sesiones contienen modelos ya procesados y listos para análisis fenomenológico.

4.5. Integración con micrOMEGAs.

Otra de las funciones principales de Amalgama es la ejecución de micrOMEGAS para calcular propiedades de partículas masivas y estables de materia oscura, siendo la densidad reliquia la observable de mayor interés en este trabajo. Como muestra el diagrama de flujo de la Figura 4.1, micrOMEGAS se alimenta con cuatro archivos que contienen la información de un modelos con nueva física: `vars.mdl`, `prtcls.mdl`, `lgrng.mdl` y `func.mdl`. Estos cuatro archivos son previamente generados por FeynRules y Amalgama se encarga de la gestión automática de ellos.

4.5.1. Selección de sesión y verificación de archivos.

Lo primero que Amalgama realiza al iniciar la subrutina de micrOMEGAs es leer la carpeta `Sesiones` y mostrar todos los modelos ya procesados previamente por FeynRules. Esto permite al usuario seleccionar con cuál sesión desea trabajar mediante un menú interactivo.

Sin embargo, existe un problema: todas las exportaciones de FeynRules se guardan en la carpeta de sesión seleccionada, sin importar el o los formatos generados en la rutina anterior. Afortunadamente, FeynRules agrega un indicador al nombre de la carpeta donde se guardan los archivos exportados. La etiqueta para el modelo exportado en formato CalcHEP/CompHEP es '`-CH`' al final del nombre. Usando ese indicador, Amalgama puede determinar si en el directorio seleccionado existe dicha carpeta mediante la función `select_dir_wth_keyword`, cuya estructura es similar a la función `find_files_wth_ext` mostrada anteriormente en el Código 4.3.

Verificar la existencia de la carpeta `XXXX-CH` no es suficiente para proceder con el cálculo de propiedades de materia oscura. Es importante resaltar que FeynRules debe generar cinco archivos `.mdl`: los ya mencionados previamente. Debido a ello se implementó una segunda verificación que comprueba la existencia de los cinco archivos. A pesar de que micrOMEGAs solo necesita cuatro de esos cinco archivos, es necesario que existan los cinco para garantizar que el procesamiento de FeynRules fue correcto.

4.5.2. Creación de proyectos en micrOMEGAs

Después de haber verificado la existencia de los archivos necesarios, Amalgama crea un nuevo proyecto en micrOMEGAs ejecutando el script `newProject` incluido en la carpeta

raíz de micrOMEGAs. Esto genera todos los archivos y directorios necesarios para el análisis. Amalgama entonces copia los archivos `.mdl` dentro de la nueva carpeta `models` ubicada en la carpeta creada para el nuevo proyecto.

Para identificar los proyectos creados por Amalgama, el programa agrega el prefijo '`MO_`' al nombre del proyecto. El nombre completo sigue el formato `MO_Sesion_XXXXXXXXXXXXX`. Todo este proceso es ejecutado automáticamente por la función `create_project`, que retorna el directorio del nuevo proyecto para su uso en etapas posteriores.

La estructura del proyecto creado sigue la organización estándar de micrOMEGAs: un directorio principal que contiene los archivos de configuración (`main.c`, `main.cpp`, `data.par`, `Makefile`), un subdirectorio `work/` para archivos temporales y resultados, un subdirectorio `lib/` para librerías compiladas, y el subdirectorio `work/models/` que aloja los archivos `.mdl` copiados desde la sesión de FeynRules. Ver Figura 4.2.

4.5.3. Configuración de módulos de cálculo

micrOMEGAs es capaz de calcular diferentes propiedades del candidato de materia oscura. El cálculo de cada propiedad se lleva a cabo por diferentes módulos dentro de `main.c` y `main.cpp`. Para activar estas funciones dentro de micrOMEGAs basta con descomentar las líneas donde se definen las macros correspondientes. Los módulos disponibles incluyen:

`#define MASSES_INFO`	Muestra información sobre el espectro de masas.
`#define CONSTRAINTS`	Muestra $B_s\gamma$, $B_s \to \mu^+\mu^-$, etc.
`#define HIGGSBOUNDS`	Llama a HiggsBounds para restringir el sector de Higgs.
`#define HIGGSSIGNALS`	Llama a HiggsSignal para restringir el bosón de Higgs.
`#define SUPERISO`	Llama a SuperISO para calcular observables de sabor.
`#define LILITH`	Llama a LiLith para restringir el sector de Higgs.
`#define SMODELS`	Llama a SModelS para restringir el sector de nueva física.
`#define MONOJET`	Restringe el sector de nueva física usando el límite monojet del LHC.
`#define OMEGA`	Calcula la densidad reliquia Ωh^2.
`#define FREEZEIN`	Calcula la densidad relicta en el mecanismo de freeze-in.

`#define INDIRECT_DETECTION`	Señales de aniquilación de DM en el halo galáctico.
`#define LoopGAMMA`	Líneas de rayos gamma.
`#define RESET_FORMFACTORS`	Redefinición de los factores de forma y otros parámetros.
`#define CDM_NUCLEON`	Calcula amplitudes y secciones eficaces para colisiones DM-nucleón.
`#define CDM_NUCLEUS`	Calcula el número de eventos por 1kg·día, distribución de energía de retroceso para varios núcleos y compara con datos experimentales.
`#define NEUTRINO`	Calcula el flujo de neutrinos solares y el correspondiente flujo de muones.
`#define DECAYS`	Calcula anchos de decaimiento y fracciones de ramificación.
`#define CROSS_SECTIONS`	Calcula secciones eficaces.
`#define CLEAN`	Elimina archivos intermedios.

La elección manual de lo que el usuario requiera calcular implicaría que el mismo usuario deba intervenir directamente en el código de `main.c`. Para evitar esto se crearon las funciones `get_funct` y `set_funct`. La función `get_funct` se encarga de leer el archivo `main.c` para detectar automáticamente todos los módulos disponibles. Esta función fue diseñada para que AMALGAMA lea siempre el `main.c` generado por MICROMEGAS en cualquier proyecto, garantizando compatibilidad con futuras versiones que puedan añadir o eliminar módulos.

La función `set_funct` se encarga de descomentar las funciones que el usuario seleccione a través de un menú interactivo y, en su defecto, comentar aquellas que no sean seleccionadas pero que por defecto estén activas en MICROMEGAS.

El siguiente paso que AMALGAMA realiza es la compilación del `main.c` ejecutando el comando `make main=main.c` en el directorio del proyecto. Si todo es correcto y CALCHEP no detecta algún error (como una variable del modelo evaluada incorrectamente), AMALGAMA continuará a la siguiente etapa. En caso contrario, será imposible compilar el archivo y no se generará el ejecutable `main`. Para detectar este error, AMALGAMA verifica la existencia del archivo `main` después de la compilación y, de haber algún problema, muestra un mensaje indicando que no fue posible crearlo y sale de la rutina, permitiendo al usuario revisar los errores reportados por el compilador.

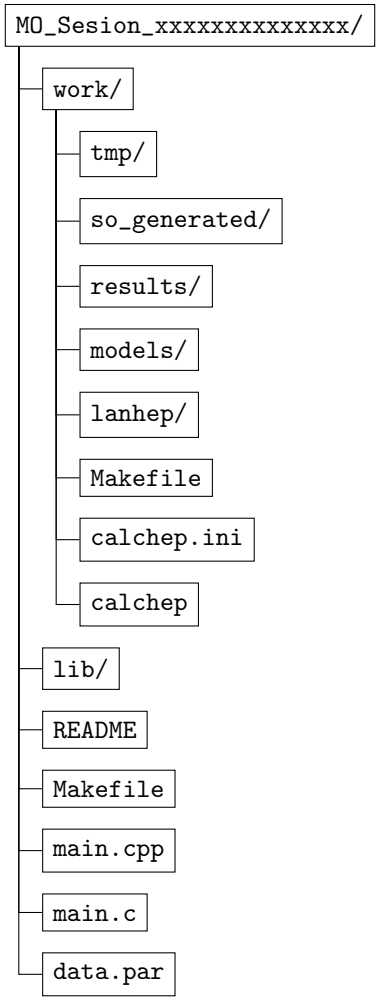


Figura 4.2: Directorio del proyecto creado por micrOMEGAs a través de `newProject`.

4.5.4. Paralelización del cálculo de observables

Aunque las etapas previas de automatización reducen significativamente el tiempo de preparación comparado con el proceso manual, la optimización más importante de Amalgama radica en esta etapa. micrOMEGAs ejecuta un análisis por cada conjunto de parámetros dados por el usuario de manera secuencial. Para realizar una exploración completa del espacio de parámetros sería necesario ejecutar el programa miles o millones de veces, resultando en tiempos de cómputo prohibitivamente largos. La paralelización de este proceso constituye la principal ventaja de Amalgama para el análisis fenomenológico eficiente.

La paralelización no es un proceso que se realice de forma nativa en Python. Esta se obtiene utilizando el módulo `concurrent.futures`. El proceso de calcular las propiedades como la densidad reliquia con parámetros dados es una tarea independiente entre sí; su salida

solo depende del archivo `data.par` que especifica los valores de los parámetros. Esta razón, junto con el número previsto de ejecuciones mínimas para lograr un buen análisis del modelo, hizo viable la paralelización del proceso.

Para comenzar este proceso es necesario conocer todos los parámetros que el modelo contiene, lo cual es posible utilizando la función `vars_Info` que extrae del archivo `vars.mdl` el nombre, valor y descripción de cada parámetro. Con esta información se muestra un menú de selección para que el usuario pueda elegir los parámetros que desea variar en el barrido del espacio de parámetros.

Para identificar diferentes ejecuciones de micrOMEGAs en un mismo proyecto, es necesario proporcionar un nombre para el archivo que contendrá los valores calculados. Inmediatamente después, el programa solicita al usuario que elija el tipo de edición de parámetros: dejar los valores establecidos en el modelo, ingresar un valor específico para cada parámetro, usar un valor aleatorio para cada parámetro o usar valores aleatorios en un rango para cada parámetro.

La cuarta opción implementa completamente las capacidades de paralelización del sistema. Cuando se selecciona esta modalidad, el usuario especifica el tamaño del barrido y los rangos de valores permitidos para cada parámetro. Esta configuración requiere la ejecución masiva de procesos en paralelo, justificando plenamente la arquitectura de cómputo distribuido implementada.

Previo a la paralelización es necesario establecer el número de núcleos del equipo que Amalgama podrá utilizar. Se decidió usar $n - 1$ núcleos con el objetivo de dejar un núcleo libre para otros procesos que el usuario pudiera estar ejecutando, evitando que el equipo se vea saturado. Establecer el número de núcleos se logró mediante la función `max(cpu_count()-1, 1)`.

Problema de generación de semillas aleatorias.

La primera problemática fue la generación de los parámetros aleatorios. Generarlos usando un solo núcleo no presentaba ningún problema; sin embargo, al paralelizar esta tarea, las semillas se repetían en los procesos debido a que el generador de números aleatorios se heredaba en cada proceso hijo, ocasionando que los valores de los parámetros se repitieran. Para solucionarlo se generan las diferentes semillas de forma aleatoria antes de la creación de los parámetros y del inicio de la paralelización.

Como el tamaño del barrido está pensado para ser mayor a 10,000 iteraciones, existía el riesgo de que alguna semilla pudiera repetirse. Para garantizar unicidad se implementó la función `seedsGen()` mostrada a continuación.

```
def seedsGen(size):
    seeds = set()
    while len(seeds) < size:
        seeds.add(random.randint(0,2**32-1))
    return list(seeds)
```

Código 4.6: Función para generar semillas únicas para la paralelización.

Esta función utiliza un set para almacenar las semillas, asegurando automáticamente que cada semilla guardada sea única. El rango de las semillas generadas va desde 0 hasta $2^{32} - 1$, asegurando que la semilla sea un número de 32 bits como máximo. El ciclo se repite hasta que el conjunto tenga el tamaño establecido. Esto impone una limitación técnica importante: no es posible hacer un barrido mayor a 2^{32} iteraciones, ya que las semillas comenzarían a repetirse y Amalgama entraría en un ciclo infinito.

4.5.5. Implementación de la paralelización.

Para iniciar la paralelización se crea una lista llamada `data` destinada a almacenar los parámetros y la salida de MICROMEGAS de cada ejecución. Posteriormente se generan las semillas usando la función mostrada anteriormente. AMALGAMA crea una tupla llamada `intsParallel` donde se almacena información compartida entre procesos: el directorio del proyecto, el nombre de los parámetros y sus rangos para el barrido.

```
#Iniciar la paraleización
data = []
seeds = seedsGen(rand_size)
intsParallel = (Project_Dir, varName, varRange)

with concurrent.futures.ProcessPoolExecutor(max_workers=num_cores) as executor:
    with tqdm(total=len(seeds)) as progress:
        futures = []
        for seed in seeds:
            future = executor.submit(parRand, *intsParallel, seed)
            future.add_done_callback(lambda p: progress.update())
            futures.append(future)

        for future in concurrent.futures.as_completed(futures):
            result = future.result()
            data.append(result)

with open(f'{resultFile}', 'a') as file:
```

```
19        file.write(tabulate(data,headers=header, tablefmt='plain'))
```

Código 4.7: Código principal de paralelización usando `ProcessPoolExecutor`.

La división de las tareas entre los $n-1$ núcleos del equipo comienza usando la clase `ProcessPoolExecutor`. La función que se reparte entre los núcleos es `parRand`, la cual crea archivos `temp_PID.par` donde se almacenan los valores de cada parámetro seleccionado en el rango establecido, usando el Identificador de Procesos (PID) como identificador único para cada archivo temporal.

Esta función también se encarga de ejecutar el archivo `main` compilado por micrOMEGAs con el archivo `.par` creado y, además, de capturar la salida de terminal con toda la información para extraer el valor de la densidad reliquia u otros observables seleccionadas. Los archivos temporales se eliminan automáticamente después de cada ejecución para evitar saturar el disco.

Adicionalmente, como un gesto para el usuario final, se implementó dentro de la paralelización una barra de progreso que muestra el avance del barrido de datos así como un tiempo estimado de finalización, gracias al módulo `tqdm`. Esto permite al usuario monitorear el progreso del cálculo y estimar cuánto tiempo falta para completar el análisis.

Los resultados se tabulan automáticamente incluyendo los valores de los parámetros seleccionados además de las observables calculadas (como $\Omega_\chi h^2$), preparando los datos para su uso en la siguiente etapa de visualización. La implementación completa de las funciones de paralelización se documenta en el Apéndice B.5.

4.6. Visualización de resultados.

La última etapa del análisis automatizado de modelos mas allá del Modelo Estándar en Amalgama consiste en la visualización de resultados. Como ya se ha mencionado, Amalgama considera principalmente el escenario del cálculo de la densidad reliquia. Es de particular interés conocer para cuáles de estos parámetros se cumple que $\Omega_\chi \leq \Omega_{DM}$. La última función que Amalgama contiene es la generación automática de gráficas, cuya lógica principal está contenida en el archivo `plotting_routine.py`.

4.6.1. Generación del script de MATHEMATICA.

AMALGAMA genera gráficas de contorno usando MATHEMATICA. Dada la simplicidad relativa del script requerido, se optó por generarlo dinámicamente en lugar de utilizar una plantilla predefinida como en el caso de FEYNRULES.

```
1    with open(plot_file_dir, 'w') as file:
2       file.write(f'data = Import["{data_file}", "Table"][[All,{{{x_column},{y_column},{
            contour_column}}}]];\n')
3       file.write('data = Rest[data];\n')
4       file.write('interpolData = Interpolation[data, InterpolationOrder -> 1];\n')
5       file.write(f'plot = ContourPlot[interpolData[x,y], {{x,{input('Especifica x_min: ')},{
            input("Especifica x_max: ")}}}, {{y,{input("Especifica y_min: ")},{input("
            Especifica y_max: ")}}}, Contours -> {{{input("Especifica la condición de contorno
            : ")}}}, ContourShading -> True, ColorFunction -> (Blend[{{White, Green, Gray}},
            #] &), ScalingFunctions -> {{"Linear", "Log"}}, PlotRange -> All, PlotLegends ->
            Automatic] \n')
6    file.write('Export["Plot.pdf",plot]\n')
```

Código 4.8: Código en PYTHON para la generación del script para la obtención de gráficas en MATHEMATICA.

El script primero importa los datos del archivo de resultados seleccionando únicamente las columnas especificadas por el usuario; segundo, elimina la primera fila que contiene los encabezados; tercero, crea una función de interpolación lineal de los datos; cuarto, genera el gráfico de contorno con la configuración especificada; y finalmente, exporta el resultado en formato PDF.

La función `ContourPlot` utiliza interpolación de primer orden para crear la superficie continua a partir de los puntos discretos calculados. Se configuró con `ColorFunction` para diferenciar visualmente las regiones que cumplen o no la condición de contorno, y `ScalingFunctions` para aplicar escala logarítmica cuando sea apropiado (típicamente para parámetros de acoplamiento que varían en varios órdenes de magnitud).

4.6.2. Ejecución y visualización.

Por último, Amalgama debe ejecutar el script directamente en terminal. El programa invoca el kernel de Mathematica mediante:

```
1    code = f'cd {os.path.dirname(data_file)} && math -script plotScript.m'
2    os.system(code)
```

Para facilitar la inspección inmediata de resultados, se implementó la visualización automática mediante la función `Popen` del módulo `subprocess`, que abre el archivo PDF generado

en el visor predeterminado del sistema.

La gráfica generada se guarda en la carpeta del proyecto dentro de micrOMEGAs del cual se tomó el archivo de resultados, manteniendo la organización estructurada de todos los componentes del análisis.

4.7. Sistema de instalación.

Para garantizar la practicidad y respetar la filosofía de fácil uso de Amalgama, se desarrolló un script de instalación escrito en bash. Dicho script se encuentra disponible en el repositorio de GitHub donde se encuentra alojado Amalgama. El script, llamado `Amalgama_Install.sh`, puede ser descargado directamente usando el siguiente comando directamente en la terminal:

```
wget https://raw.githubusercontent.com/HCR264/Amalgama/refs/heads/main/Install%20Script/
    Amalgama_Install.sh
```

Una vez descargado, se le deben otorgar permisos de ejecución para poder comenzar con la instalación:

```
chmod +x Amalgama_Install.sh
./Amalgama_Install.sh
```

4.7.1. Detección del sistema operativo.

Amalgama fue diseñado para funcionar en cualquier distribución de Linux siempre y cuando se tengan los paquetes necesarios. Este instalador funciona específicamente para Ubuntu/Debian, Fedora y Arch Linux, por lo que fue necesario incluir la detección del sistema operativo para garantizar la correcta instalación de cada uno de los paquetes a través de su gestor correspondiente: APT para Ubuntu y Debian, Pacman para Arch Linux y DNF para Fedora. La detección del sistema operativo se logra leyendo el archivo `/etc/os-release`.

```
if [ -f /etc/os-release ]; then
    . /etc/os-release
    DISTRO=$ID
    if [[ "$DISTRO" != "arch" && "$DISTRO" != "ubuntu" && "$DISTRO" != "fedora" && "
        $DISTRO" != "debian"]]; then
        printf "Este script solo funciona en Arch, Fedora, Ubuntu o Debian.\n"
        exit 1
    fi
else
    printf "No se pudo identificar el sistema operativo.\n"
    exit 1
fi
```

4.7.2. Gestión de dependencias.

Cada gestor de paquetes puede contener versiones y nombres distintos de los paquetes requeridos. El instalador define listas de paquetes específicas para cada distribución:

```
if [ "$DISTRO" == "ubuntu" || "$DISTRO" == "ubuntu"]; then
   paquetes_libres=(make gcc gfortran libx11-dev git python3 python3-tqdm python3-
       tabulate python3-numpy)
elif [ "$DISTRO" == "fedora" ]; then
   paquetes_libres=(gcc gcc-gfortran libX11-devel git python3 python3-tqdm python3-
       tabulate python3-numpy)
else
   paquetes_libres=(gcc gcc-fortran git python python-tqdm python-tabulate python-numpy)
fi
```

Para la verificación de los paquetes y su instalación se implementaron funciones que consultan si cada paquete ya se encuentra instalado a través de los comandos nativos de cada gestor:

```
is_installed() {
   case "$DISTRO" in
      "ubuntu")
         dpkg -l | grep -q "$1" &> /dev/null
         ;;
      "arch")
         pacman -Qs "$1" &> /dev/null
         ;;
      "fedora")
         dnf list installed "$1" &> /dev/null
      ;;
   esac
}
```

Aquellos paquetes que no se encuentren instalados se descargan automáticamente usando la función `install` que invoca el gestor de paquetes apropiado con privilegios de superusuario.

4.7.3. Instalación de componentes especializados.

Todos los paquetes de software libre son instalados automáticamente. Para MATHEMATICA, el instalador solo verifica si se encuentra instalado mediante:

```
is_MW_installed() {
   command -v "$1" &> /dev/null
}
```

De no encontrarse, se muestra un mensaje al usuario sugiriendo su instalación y la adquisición de una licencia, ya que MATHEMATICA es software privativo que no puede ser instalado automáticamente.

El script clona el repositorio de GitHub donde AMALGAMA se encuentra alojado. Dentro del repositorio se incluye FEYNRULES. La versión incluida corresponde a una adaptación de la versión más reciente disponible en GitHub que es compatible con versiones actuales de Mathematica. Durante las pruebas se encontraron algunos errores en la versión oficial que fueron corregidos cambiando `MatrixSymbol` a `FRMatrixSymbol` en todos los archivos dentro de FEYNRULES.

Para terminar con la instalación, el script descarga MICROMEGAS versión 6.2.3 desde el sitio oficial usando `wget`. El instalador compila automáticamente MICROMEGAS en el directorio apropiado, dejando el sistema completamente funcional para el uso inmediato de AMALGAMA. Todo el proceso de instalación está diseñado para ejecutarse con mínima intervención del usuario, solicitando únicamente confirmación para operaciones que requieren privilegios root.

Capítulo 5: Resultados

Este capítulo presenta la validación de Amalgama mediante su aplicación al Modelo Estándar más un singlete escalar, uno de los modelos más simples de materia oscura más allá del Modelo Estándar. La validación se realizó en dos aspectos complementarios: primero, verificando que los resultados físicos obtenidos son consistentes con publicaciones existentes; segundo, cuantificando las mejoras en tiempos de ejecución mediante el análisis de escalabilidad del programa en diferentes configuraciones de hardware.

La elección del modelo del singlete escalar como caso de prueba se justifica por varias razones: su simplicidad teórica permite verificar la correcta implementación del flujo de trabajo completo sin complejidades adicionales; existe literatura abundante con la cual comparar resultados; y el cálculo de la densidad reliquia es suficientemente demandante para evaluar las ventajas de la paralelización.

El capítulo inicia describiendo el proceso de instalación automatizada del programa, seguido de la metodología de validación empleada, la aplicación al modelo del singlete escalar, y finalmente el análisis detallado del desempeño computacional.

5.1. Instalación de Amalgama

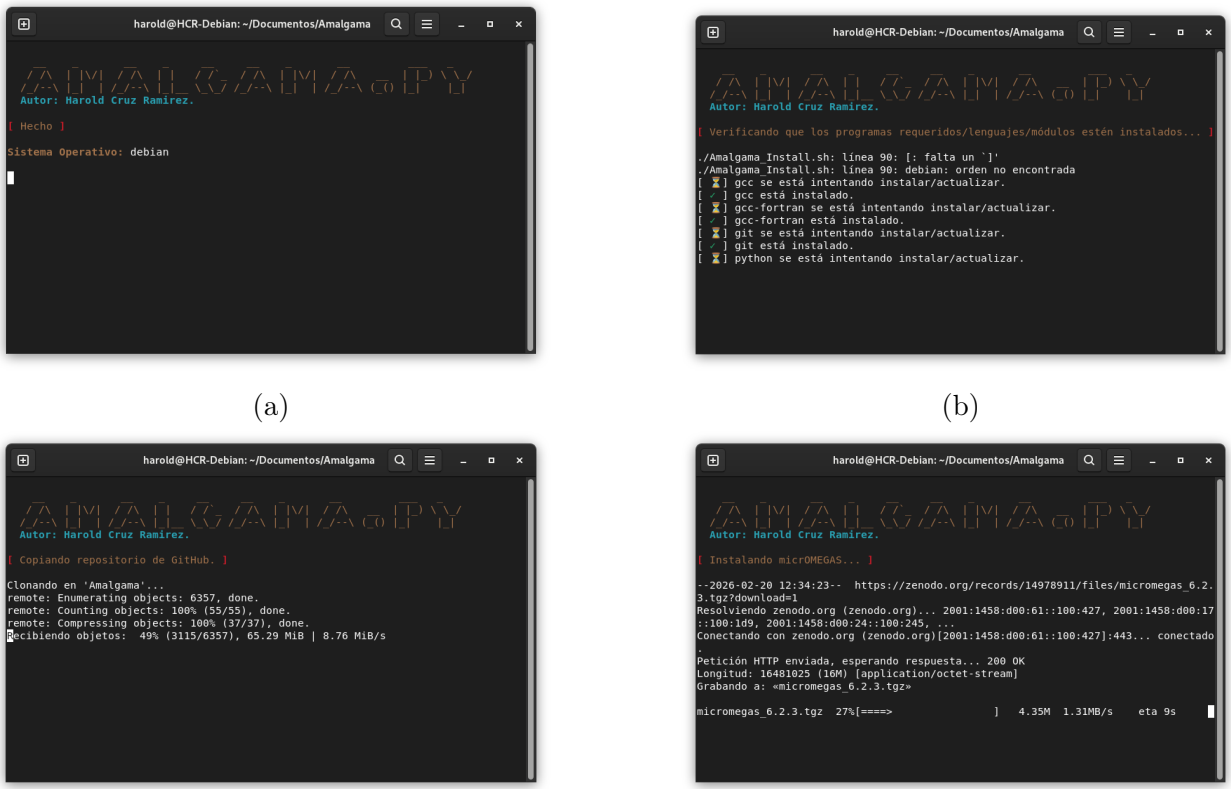
El script de instalación `Amalgama_Install.sh` fue probado exitosamente en cuatro distribuciones de Linux: Debian 13,Ubuntu 24.04 LTS, Fedora 42 y Arch Linux. El proceso de instalación se inicia descargando el script desde el repositorio de GitHub y otorgándole permisos de ejecución como se describió en la Sección 4.7.

La Figura 5.1 muestra las diferentes etapas del proceso de instalación. En la Figura 5.1a se aprecia la detección automática del sistema operativo y la verificación inicial de requisitos. El instalador identifica correctamente la distribución mediante la lectura del archivo `/etc/os-release` y selecciona el gestor de paquetes apropiado.

La Figura 5.1b muestra el proceso de verificación e instalación de dependencias. El script consulta qué paquetes ya están instalados en el sistema y procede a instalarlos y actualizarlos. Este proceso incluye compiladores (gcc, gfortran), librerías del sistema (libX11), herramientas de desarrollo (git, make), y módulos de Python (tqdm, tabulate, numpy).

Posterior a la instalación de dependencias, el instalador clona el proyecto de AMALGAMA desde su repositorio en GitHub. Este paso puede verse en la Figura 5.1c.

En la Figura 5.1d se observa la descarga y compilación de MICROMEGAS desde el sitio oficial. El instalador descarga la versión 6.2.3, descomprime el archivo y realiza la compilación inicial del código base.



(a) (b)

(c) (d)

Figura 5.1: Capturas del proceso de instalación de AMALGAMA en un ordenador con Debian 13.

5.2. Metodología de validación.

La validación de Amalgama se estableció mediante dos criterios complementarios:

Validación del modelo. Los resultados obtenidos deben ser consistentes con publicaciones previas del modelo analizado. Específicamente, las gráficas de contorno de densidad reliquia deben reproducir las regiones de parámetros viables identificadas en la literatura.

Validación de desempeño. El programa debe demostrar mejoras significativas en tiempos de ejecución comparado con el proceso tradicional secuencial. La paralelización debe escalar adecuadamente con el número de núcleos disponibles.

5.2.1. Especificaciones computacionales.

Para la validación se utilizó como equipo principal una laptop con las especificaciones mostradas en la Tabla 5.1.

Especificaciones del equipo.	
Dispositivo	Laptop
Modelo	Aspire A315-59 V1.27
Sistema Operativo	Arch Linux x86_64
Kernel	6.12.36-1-lts
CPU	Intel Core i5-1235U
Memoria	19704 MiB

Tabla 5.1: Especificaciones del equipo principal utilizado para la validación de Amalgama. El procesador Intel Core i5-1235U cuenta con 12 núcleos (4 de rendimiento + 8 de eficiencia).

Adicionalmente, se realizaron pruebas en dos equipos de escritorio con mayor número de núcleos para evaluar el comportamiento del programa en configuraciones de alto rendimiento.

5.3. Modelo Estándar más Singlete Escalar.

El modelo del singlete escalar extiende el Modelo Estándar mediante la adición de un campo escalar real masivo no cargado S. Este campo obedece una simetría Z_2 bajo la cual $S \to -S$, garantizando su estabilidad[15] y convirtiendolo en un candidato viable de materia oscura fría.

La densidad lagrangiana del modelo es:

$$\mathcal{L} = \mathcal{L}_{SM} + \frac{1}{2}\partial_\mu S \partial^\mu S - \frac{1}{2}\mu_s^2 S^2 - \frac{1}{2}\lambda_{HS} S^2 H^\dagger H - \frac{1}{2}\lambda_S S^4 \tag{5.1}$$

donde μ_S es la masa desnuda del singlete, λ_S es el auto-acoplamiento de S, λ_{HS} es la constante de acoplamiento con el Higgs, H el doblete de Higgs del Modelo Estándar, y $\mathcal{L}_{SM}$ es la lagrangiana del Modelo Estándar. Tras el rompimiento espontaneo de simetría se obtiene el termino de masa física $m_s^2 = \mu_s^2 + \frac{1}{2}v_{vev}^2$.

5.3.1. Implementación en AMALGAMA

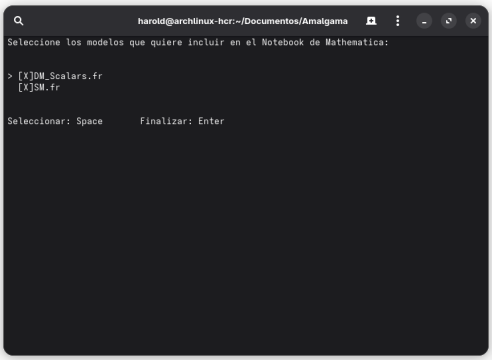
El modelo fue implementado en un archivo `.fr` siguiendo la sintaxis de FEYNRULES. Dentro del archivo se definió el campo escalar S con sus propiedades, la declaración de m_S, λ_{HS} y λ_S.

Se fijó el valor del auto-acoplamiento en $\lambda_S = 0{,}13$, siguiendo el análisis de Cline, Scott, Kainulainen y Weniger (2013). Los valores iniciales para m_S y λ_{HS} fueron establecidos en $150\,GeV$ y 0,5 respectivamente, pero estos serán modificados durante el barrido del espacio de parámetros. Los detalles del modelo pueden ser consultados en el Apéndice A.1.

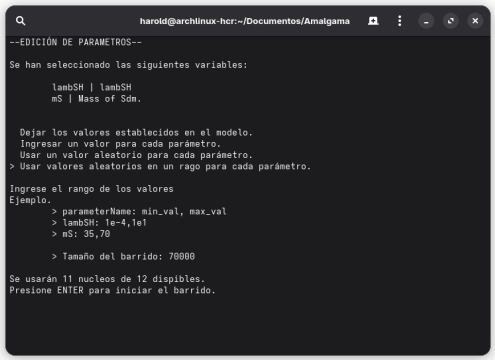
5.3.2. Ejecución del análisis.

La ejecución de AMALGAMA para este modelo siguió el flujo de trabajo completo ilustrado en el diagrama de la Figura 4.1. El proceso se documenta mediante las capturas de pantalla de la Figura 5.2.

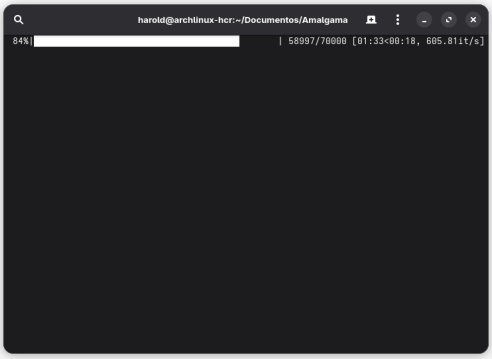
Se seleccionaron los archivos del Modelo Estándar (`SM.fr`) y el nuevo campo escalar (`DM_Scalars.fr`). AMALGAMA generó automáticamente el script de MATHEMATICA, lo ejecutó mediante FEYNRULES y verificó la correcta generación de los cinco archivos `.mdl` necesarios para MICROMEGAS. Se especificó un barrido aleatorio del espacio de parámetros con 70,000 iteraciones. El rango de masas se estableció entre $45\,GeV$ y $70\,GeV$, y el acoplamiento del portal entre 10^{-4} y 10. La elección de estos rangos permite comparar directamente con las gráficas publicadas por Cline, Scott, Kainulainen y Weniger (2013). El número de iteraciones fue seleccionado para lograr una densidad de puntos suficiente que permita la interpolación suave en la generación de contornos.



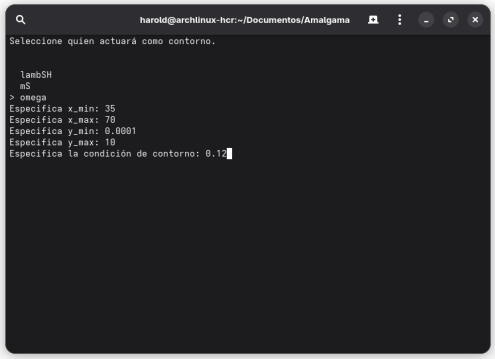
(a) Selección de los modelos.



(b) Fijar el rango de parámetros.



(c) Calculo en paralelo de $\Omega_{DM}h^2$.



(d) Establecer el dominio, rango y contorno.

Figura 5.2: Capturas de pantalla de la ejecución de AMALGAMA para el modelo de singlete escalar descrito en el Apéndice A.1.

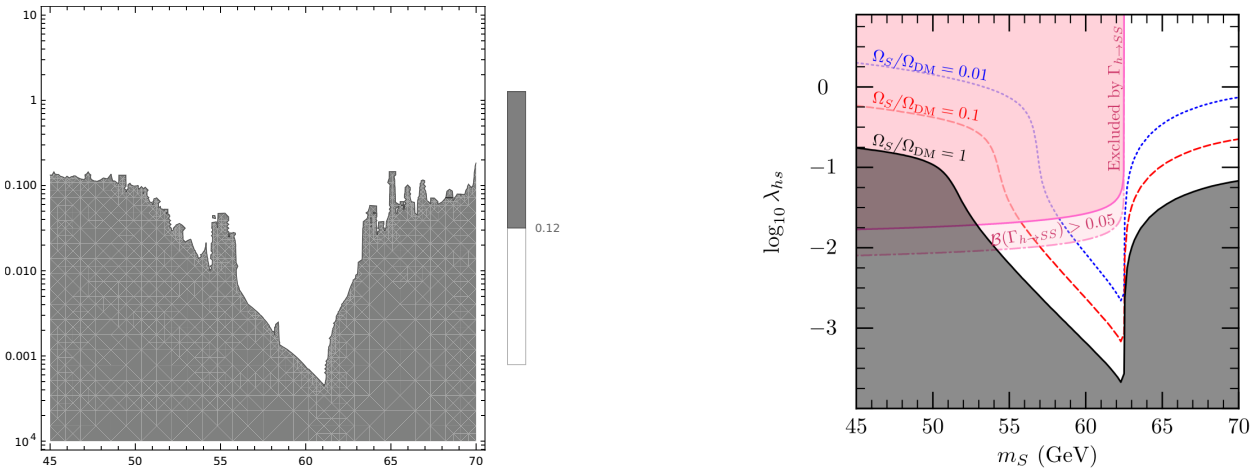
5.4. Resultados y comparación.

5.4.1. Gráfica de contorno obtenidas.

La Figura 5.3a muestra la gráfica de contorno generada por AMALGAMA. La región en gris representa combinaciones de parámetros (m_S, λ_{HS}) que producen una densidad reliquia mayor que el valor observado $\Omega_{DM}h^2 = 0{,}12$. La línea de contorno separa esta región de aquella donde $\Omega_S \leq \Omega_{DM}$, que corresponde a parámetros viables para el modelo.

La Figura 5.3b presenta la gráfica extraída de Cline, Scott, Kainulainen y Weniger (2013). La publicación incluye cinco regiones delimitadas por diferentes restricciones experimentales y teóricas. La región de particular interés para la validación es aquella separada por la condición

$\Omega_S = \Omega_{DM}$, ya que define los parámetros donde el singlete escalar puede constituir toda la materia oscura observada.



(a) (b)

Figura 5.3: Gráficas de contorno obtenidas por AMALGAMA (izquierda) y por Cline, Scott, Kainulainen y Weniger (2013) (derecha).

5.4.2. Validación de la implementación.

A pesar de las pequeñas discrepancias cuantitativas, la concordancia entre los resultados es excelente. Esto demuestra que el flujo de trabajo completo funciona correctamente, las reglas de Feynman fueron derivadas apropiadamente y la paralelización no interfiere con el calculo de las observables. Por lo tanto, se concluye que AMALGAMA implementa correctamente el análisis fenomenológico de modelos con materia oscura y produce resultados consistentes con la literatura establecida.

5.5. Análisis de desempeño computacional.

Para evaluar la funcionalidad y estabilidad de AMALGAMA se realizaron pruebas en dos configuraciones diferentes de hardware, mostradas en la Tabla 5.2. Los equipos presentan 12 y 48 núcleos respectivamente, con arquitecturas diferentes: Intel Xeon y AMD Threadripper.

5.5.1. Tiempos de ejecución.

La Tabla 5.2 muestra los resultados de dos pruebas más con diferentes tamaños de barrido. El Equipo 1 completó 1,000,000 iteraciones en 24 minutos y 25 segundos utilizando 11

Especificaciones del equipo.			
Dispositivo	PC de escritorio	PC de escritorio	
Modelo	Dell Precision Tower 7910	TRX40 AORUS PRO WIFI -CF	
Sistema Operativo	Ubuntu 22.04.5 LTS x86_64	Ubuntu 24.04.1 LTS x86_64	
Kernel	6.8.0-51-generic	6.8.0-49-generic	
CPU	Intel Xeon E5-2609 V3	AMD Ryzen Threadripper 3960X	
No. de núcleos	12	48	
Memoria	47.1 GB	62.6 GB	
Detalles de la prueba			
Tamaño de barrido	1 000 000	1 000 000	10 000 000
Tiempo del barrido	24:25	6:22	1:04:53

Tabla 5.2: Especificaciones de los dos equipos utilizados para el análisis de escalabilidad de Amalgama. Los equipos presentan 12 y 48 núcleos respectivamente, permitiendo evaluar el comportamiento del programa en diferentes configuraciones.

núcleos. El Equipo 2, con mayor capacidad de procesamiento paralelo, completó el mismo barrido en 6 minutos y 22 segundos usando 47 núcleos. Adicionalmente se realizo otra prueba con 10,000,000 iteraciones en el Equipo 2, completándose en 1 hora, 4 minutos y 53 segundos.

Comparado con el proceso tradicional donde las evaluaciones se realizarían secuencialmente, Amalgama reduce el tiempo de análisis de días a minutos en configuraciones típicas, habilitando exploraciones exhaustivas del espacio de parámetros que serían imprácticas de otra manera.

Capítulo 6: Conclusiones y trabajo a futuro.

6.1. Conclusiones.

Este trabajo desarrolló exitosamente AMALGAMA, un programa en PYTHON 3 que integra automáticamente FEYNRULES, CALCHEP y MICROMEGAS para el análisis fenomenológico de modelos de materia oscura. El programa resuelve las tres limitaciones principales identificadas en el proceso tradicional: elimina la gestión manual de archivos mediante interfaces automatizadas, estandariza la presentación de resultados en formatos tabulados y gráficos, y reduce drásticamente los tiempos de cómputo mediante paralelización eficiente.

La implementación de cálculos paralelos aprovechando procesadores multinúcleo modernos constituye la principal contribución técnica. El análisis de desempeño demostró eficiencias superiores al proceso habitual, procesando más de 2,600 evaluaciones por segundo en un equipo con 47 núcleos. Esta optimización reduce el tiempo de análisis que tomarían días en el proceso tradicional a ejecuciones de minutos, habilitando exploraciones exhaustivas del espacio de parámetros.

La validación mediante el modelo del singlete escalar confirmó que el programa produce resultados físicos consistentes con publicaciones establecidas. Las gráficas de densidad reliquia reproducen las características esperadas del modelo.

La arquitectura modular del programa facilita futuras extensiones, y el instalador automatizado compatible con las principales distribuciones de Linux reduce la complejidad de la compilación e instalación. AMALGAMA cumple los objetivos planteados, proporcionando una herramienta funcional, validada y extensible para el análisis de modelos de materia oscura.

6.2. Limitaciones identificadas.

Durante las pruebas más demandantes de AMALGAMA se identificaron dos limitaciones técnicas que requieren atención en versiones futuras del programa.

Para barridos superiores a $100,000$ iteraciones se observó un comportamiento anómalo en la generación de archivos temporales `data.par`. En algunos casos, los archivos contenían múltiples declaraciones de parámetros con diferentes valores, sugiriendo condiciones de carrera en el acceso concurrente al sistema de archivos. Este problema no estaba presente en las primeras implementaciones de la paralelización. La solución inicial al problema de generación de semillas aleatorias resolvió exitosamente la duplicación de parámetros, pero introdujo esta nueva complicación en la gestión de archivos temporales. El problema es intermitente y no afecta todos los barridos extensos, sugiriendo que está relacionado con la sincronización de procesos en condiciones específicas de carga del sistema. Es necesario revisar detalladamente la función `parRand` que gestiona la creación y eliminación de archivos temporales en el contexto de paralelización.

También se observó que el consumo de memoria RAM de AMALGAMA incrementa con el número de iteraciones del barrido. Esto se debe a que la implementación actual almacena todos los resultados en memoria (en la lista `data`) hasta que la paralelización finaliza, momento en el cual se escriben todos los datos al archivo de resultados. Esta estrategia funciona bien para barridos de hasta 1,000,000 de iteraciones en equipos con memoria suficiente, pero puede causar problemas en sistemas con memoria limitada o para barridos muy extensos ($> 10,000,000$ de iteraciones).

6.3. Trabajo a futuro.

6.3.1. Soluciones propuestas a limitaciones actuales.

Para el problema de gestión de archivos, se proponen dos estrategias. La primera consiste en implementar archivos temporales con nombres únicos mediante la combinación del PID del proceso, el timestamp de creación, y la semilla utilizada. Esto eliminaría cualquier posibilidad de conflicto de nombres incluso en condiciones de alta concurrencia. La segunda estrategia es utilizar el módulo `tempfile` de PYTHON, que garantiza la creación de archivos temporales únicos mediante primitivas del sistema operativo, eliminando la necesidad de gestión manual.

Para el problema de consumo de memoria, se contempla dividir la paralelización en bloques de 10,000 iteraciones. Cada bloque completa su ejecución, escribe sus resultados al archivo, libera la memoria, e inicia el siguiente bloque. Esto mantendría el consumo de memoria constante independientemente del tamaño total del barrido, al costo de un pequeño overhead adicional por la gestión de bloques. Una segunda estrategia consiste en implementar escritura incremental donde cada proceso escribe sus resultados inmediatamente al completar cada iteración, usando un lock para sincronizar el acceso al archivo de salida.

Ambas aproximaciones son viables y la elección entre ellas dependerá de pruebas de desempeño que cuantifiquen el overhead introducido por cada estrategia.

6.3.2. Extensiones planificadas del programa.

La arquitectura modular de AMALGAMA facilita varias extensiones planificadas como la inclusión de SARAH y LANHEP, ampliando el rango de modelos que pueden analizarse. La estructura actual del módulo de gestión de modelos ya contempla esta extensibilidad mediante funciones parametrizables por extensión de archivo.

Actualmente AMALGAMA se enfoca principalmente en la densidad reliquia, pero MICROMEGAS puede calcular múltiples observables (secciones eficaces de detección directa, espectros de aniquilación indirecta, etc.). Extender las funciones de los módulos de AMALGAMA para generar automáticamente gráficas de estos observables sería una adición valiosa.

La estructura modular actual permite incorporar estas extensiones sin modificar los componentes existentes, validando las decisiones de diseño tomadas durante el desarrollo inicial del programa.

Apéndice A: Modelos para FeynRules

En este apartado se encuentran los dos modelos análizados en este trabajo de tesis. El primero de ellos corresponde al modelo de un sigléte escalar y el segundo a el modelo de dos Higgs inértes (revisar el nombre).

A.1. Modelo Singléte Escalar

Esta extensión del modelo estandar propone un nuevo campo escalar, S, como candidato para materia oscura. El lagrangiano de este modelo se escribe como

$$\mathcal{L} = \mathcal{L}_{SM} + \frac{1}{2}\partial_\mu S \partial^\mu S - \frac{1}{2}\mu_s^2 S^2 - \frac{1}{2}\lambda_{HS} S^2 H^\dagger H - \frac{1}{2}\lambda_S S^4 \tag{A.1}$$

donde S es nuevo campo, λ_{HS} el acoplamiento con el Higgs, λ_S el autoacoplamiento.

En el código A.1 se muestra como se escribió el archivo `DS_Scalars.fr` utilizado en este trabajo. Es importante resaltar que en la declaración del campo escalar la masa utilizada fue directamente la masa física $m_S^2 = \mu_S^2 + \frac{1}{2}v_{vev}^2$.

```
M$ModelName = "Scalar_Dark_Model"

M$Information = {
   Authors      -> {"Harold Cruz Ramirez"},
   Version      -> "1.0",
   Date         -> "06. 11. 2024";
}

M$ClassesDescription = {
   S[9] == {
      ClassName          -> Sdm,
      SelfConjugate      -> True,
      Mass               -> {mS, 150},
      ParticleName       -> {"~Sdm"},
      Width              -> {wS, Internal},
      PDG                -> 1003,
      PropagatorLabel    -> {"Sx"},
      PropagatorType     -> ScalarDash
   }
```

```
20    }
21
22    M$Parameters = {
23       lambSH == {
24          Value                -> 0.5,
25          InteractionOrder     -> {QED, 2}
26       },
27
28       lambSx =={
29          value              -> 0.13,
30          InteractionOrder   -> {QED, 2}
31       }
32    }
33
34    (* Lagrangiano DM *)
35
36    LDM = 1/2 del[Sdm, mu].del[Sdm, mu] - 1/2 mS^2 Sdm^2 - 1/2 lambSH vev^2 Sdm^2 (H/vev +
             1/2 H*H/(vev*vev)) - 1/2 lambSx Sdm^4 ;
37
38    (***Lagrangians***)
39    (*LDM*)
40    (***End***)
```

Código A.1: Declaracion del modelo de singlete escalar en FeynRules.

Apéndice B: Códigos.

B.1. Amalgama.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue May 28 13:30:25 2024

@author: hcr
"""

#   MODULOS
import os
import shutil

from Modulos.feynrules_routine import feynrules_routine
from Modulos.micromegas_routine_v2 import main_micromegas
from Modulos.plotting_routine import main_plot
from Modulos.menu import clear_screen
from Modulos.menu import menu
from Modulos.files import get_dir
from Modulos.files import select_dir


# Verificar la existencia de un archivo
def exist_file(file_dir):
    return os.path.isfile(file_dir)


# Buscar archivos
def find_files(filename, search_path):
    matches = []
    for root, dirs, files in os.walk(search_path):
        if filename in files:
            matches.append(os.path.join(root, filename))
    matches.append("Volver a buscar")
    matches.append("Salir")
    return matches


# Mover archivos
def move_files(filename, src_path, dst_path):
```

```
40      if os.path.isfile(dst_path + "/" + filename) is True:
41          print("\nEl archivo ya existe en " + dst_path)
42          input("Presione Enter para salir...")
43      else:
44          shutil.copy2(src_path, dst_path, follow_symlinks=True)
45          print("\nEl archivo fue movido con exito a " + dst_path)
46          input("Presione Enter para salir...")
47
48
49  # Importar modelos a 'Modelos'
50  def import_models(dst_path):
51      while True:
52          os.system('clear')
53
54          filename = input("Ingrese el nombre del modelo: ")
55          search_path = "/home"
56
57          matches = find_files(filename, search_path)
58
59          if len(matches) == 2:
60              print("No se ha encontrado el archivo.")
61              option = input("¿Desea volver a intentarlo? (y/n): ")
62              if option == "n":
63                  break
64              else:
65                  continue
66
67          elif len(matches) == 3:
68              selected = menu(matches, f"Se encontró 1 archivo que coincide con el nombre en {
                    matches[0]}")
69              if matches[selected] == "Volver a buscar":
70                  continue
71              elif matches[selected] == "Salir":
72                  break
73              else:
74                  input(matches[selected])
75                  input(dst_path)
76                  move_files(filename, matches[selected], dst_path)
77                  break
78          else:
79              selected = menu(matches, f"Se encontraron {len(matches)-2} archivos que
                    coinciden con el nombre. Seleccione el archivo que desee importar:")
80              if matches[selected] == "Volver a buscar":
81                  continue
82              elif matches[selected] == "Salir":
83                  break
84              else:
85                  move_files(filename, matches[selected], dst_path)
86                  break
87
88
89  def find_files_wth_ext(search_path, ext):
90      find_files = []
91      for file in os.listdir(search_path):
92          if file.endswith(".fr"):
```

```
93              find_files.append(file)
94      return find_files
95
96
97  # MENUS
98  # Lista de los modelos
99  def model_list():
100     clear_screen()
101
102     model_list = find_files_wth_ext(ModelsDir, ".fr")
103     print("Modelos importados:\n")
104
105     prefix = "- "
106     for i, files in enumerate(model_list):
107         print(prefix + files)
108
109     input("\nPresiona Enter para salir...")
110
111
112 # Menu para manipular FeynRules
113 def FeynRulesMenu():
114     clear_screen()
115
116     menu_info = "CREACIÓN DE NOTEBOOK\n"
117     options = [""]
118
119
120 # Menu Principal
121 def main_menu():
122     menu_info = "Seleccione la acción que quiere realizar."
123     options = ["Importar un modelo", "Ver modelos", "FeynRules", "MicrOMEGAS", "Graficar", "
        Salir"]
124
125     while True:
126         selected = menu(options, menu_info)
127         if options[selected] == "Salir":
128             print("Saliendo...")
129             break
130         elif options[selected] == "Importar un modelo":
131             import_models(ModelsDir)
132         elif options[selected] == "Ver modelos":
133             model_list()
134         elif options[selected] == "FeynRules":
135             feynrules_routine(GlobalDir, ModelsDir, FeynRulesDir)
136         elif options[selected] == "MicrOMEGAS":
137             main_micromegas(GlobalDir)
138         elif options[selected] == 'Graficar':
139             main_plot(GlobalDir + '/micromegas_6.2.3')
140
141
142 def create_dir(src_dir, dir_name):
143     os.mkdir(src_dir + "/" + dir_name)
144
145
146 # MAIN
```

```
147 def main():
148     # VARIABLES GLOBALES
149     global GlobalDir
150     global FeynRulesDir
151     global ModelsDir
152     global Plantilla_FeynRules_NB
153     global Sesions_Dir
154 
155 
156     # OBTENER EL DIRECTORIO DE TRABAJO
157     GlobalDir = get_dir()
158 
159     # Definir las carpetas de trabajo
160     FeynRulesDir = GlobalDir + '/feynrules-current'
161     ModelsDir = GlobalDir + '/Modelos'
162     Plantilla_FeynRules_NB = GlobalDir + '/.Plantilla_FeynRules_Script'
163     Sesions_Dir = GlobalDir + '/Sesiones'
164 
165     main_menu()
166 if __name__ == "__main__":
167     main()
```

B.2. feynrules_routine.py

```
1  import os
2  import shutil
3  from datetime import datetime
4  from .menu import clear_screen
5  from .menu import multi_select_menu
6 
7 
8  def extract_lagrangians(model_path):
9      with open(model_path, 'r') as file:
10         lines = file.readlines()
11 
12     lagrangians_in_model = []
13     capturing = False
14 
15     for line in lines:
16         line = line.strip()
17         if line == "(***Lagrangians***)":
18             capturing = True
19             continue
20         elif line == "(***End***)":
21             capturing = False
22             break
23         if capturing:
24             cleaned_line = line.replace("(*", "").replace("*)", "").strip()
25             if cleaned_line:
26                 lagrangians_in_model.append(cleaned_line)
27     return lagrangians_in_model
28 
29 
```

```
30 def obtain_lagrangians(ModelsSelected, ModelsDir):
31     lagrangians = []
32     for models in ModelsSelected:
33         for lagrangian in extract_lagrangians(ModelsDir + '/' + models):
34             lagrangians.append(lagrangian)
35     return lagrangians
36 
37 
38 # EDITAR PLANTILLA
39 def create_script(Models, Lagrangians, Interfaces, Plantilla_FeynRules_Dir, FeynRulesDir,
       ModelsDir, Sesions_Dir):
40     global Current_Sesion_Dir
41     global Current_Script_Name
42     global Current_Sesion_Code
43     with open(Plantilla_FeynRules_Dir, 'r') as file:
44         lines = file.readlines()
45 
46 
47     Models_Str = ", ".join(f'"{m_item}"' for m_item in Models)
48     Lagrangians_Str = ", ".join(f'{l_item}' for l_item in Lagrangians)
49 
50     for i, line in enumerate(lines):
51         if '$FeynRulesPath = SetDirectory[]' in line:
52             lines[i] = f'$FeynRulesPath = SetDirectory["{FeynRulesDir}"]\n'
53         if 'SetDirectory[ ]' in line:
54             lines[i] = f'SetDirectory["{ModelsDir}"]\n'
55         if 'LoadModel[]' in line:
56             lines[i] = f'LoadModel[{Models_Str}]\n'
57         if 'FeynmanRules[]' in line:
58             lines[i] = f'FeynmanRules[{Lagrangians_Str}]\n'
59         if '(*WriteCHOutput[]*)' in line:
60             if 'CalcHep/CompHep' in Interfaces:
61                 lines[i] = f'WriteCHOutput[{Lagrangians_Str}]\n'
62         if '(*WriteFeynArtsOutput[]*)' in line:
63             if 'FeynArts/FormCalc' in Interfaces:
64                 lines[i] = f'WriteFeynArtsOutput[{Lagrangians_Str}]\n'
65         if '(*WriteSHOutput[]*)' in line:
66             if 'Sherpa' in Interfaces:
67                 lines[i] = f'WriteSHOutput[{Lagrangians_Str}]\n'
68         if '(*WriteLaTeXOutput[]*)' in line:
69             if 'TEX' in Interfaces:
70                 lines[i] = f'WriteLaTeXOutput[{Lagrangians_Str}]\n'
71         if '(*WriteUFO[]*)' in line:
72             if 'UFO' in Interfaces:
73                 lines[i] = f'WriteUFO[{Lagrangians_Str}]\n'
74         if '(*WriteWOOutput*)' in line:
75             if 'Whizard/Omega)' in Interfaces:
76                 lines[i] = f'WriteWOOutput[{Lagrangians_Str}]\n'
77 
78     code = datetime.now()
79     Current_Sesion_Code = code.strftime('%Y%m%d%H%M%S')
80 
81     Current_Sesion_Dir = Sesions_Dir + f'/Sesion_{Current_Sesion_Code}'
82     os.mkdir(Current_Sesion_Dir)
83     Current_Script_Name = f'FeynRules_{Current_Sesion_Code}.m'
```

```
84
85      with open(Current_Sesion_Dir + '/' + Current_Script_Name, 'w') as file:
86          file.writelines(lines)
87
88      run_mathematica_script()
89
90      move_dirs(ModelsDir)
91
92      clear_screen()
93
94      input(f'Finalizado. Los archivos se han guardado dentro de {Current_Sesion_Dir}')
95
96
97  def run_mathematica_script():
98      clear_screen()
99      print(f'Ejecutando {Current_Script_Name} ...\n')
100     code = f'math -script {Current_Sesion_Dir}/{Current_Script_Name}'
101     os.system(code)
102
103
104 def move_dirs(Dir):
105     clear_screen()
106     matches = []
107     for element in os.listdir(Dir):
108         element_dir = Dir + '/' + element
109         if os.path.isdir(element_dir):
110             matches.append(element_dir)
111
112     for dir in matches:
113         shutil.move(dir, Current_Sesion_Dir)
114
115
116 # FUNCIONES QUE ESTÁN EN OTRO ARCHIVO
117 def find_files_wth_ext(search_path, ext):
118     find_files = []
119     for file in os.listdir(search_path):
120         if file.endswith('.fr'):
121             find_files.append(file)
122     return find_files
123
124 # FUNCION PRINCIPAL
125 def feynrules_routine(GlobalDir, ModelsDir, FeynRulesDir):
126     Plantilla_FeynRules_Dir = GlobalDir + '/.Plantilla_FeynRules_Script/
        Plantilla_FeynRules_Script.m'
127     Sesions_Dir = GlobalDir + '/Sesiones'
128
129     Current_Sesion_Dir = ''
130     Current_Script_Name = ''
131
132     Models_List = find_files_wth_ext(ModelsDir, '.fr')
133     ModelSelected_Info = "Seleccione los modelos que quiere incluir en el Notebook de
        Mathematica:"
134     Selected_Models = multi_select_menu(Models_List, ModelSelected_Info)
135
136     Lagrangians_List = obtain_lagrangians(Selected_Models, ModelsDir)
```

```
137
138     Selected_Lagrangians_Menu_Info = "Seleccione los lagrangianos que quiere incluir en el
            Notebook de Mathematica:"
139     Selected_Lagrangians = multi_select_menu(Lagrangians_List,
            Selected_Lagrangians_Menu_Info)
140
141     Interfaces = ["CalcHep/CompHep", "FeynArts/FormCalc", "Sherpa", "TEX", "UFO", "Whizard/
            Omega"]
142     Selected_Interfaces_Menu_Info = "Seleccione las interfaces en los que deseé que
            FeynRules exporte:"
143     Selected_Interfaces =  multi_select_menu(Interfaces, Selected_Interfaces_Menu_Info)
144
145     create_script(Selected_Models, Selected_Lagrangians, Selected_Interfaces,
            Plantilla_FeynRules_Dir, FeynRulesDir, ModelsDir, Sesions_Dir)
146
147
148 def FeynRules_Rutine_main():
149
150     global GlobalDir
151     global ModelsDir
152     global FeynRulesDir
153     global Plantilla_FeynRules_Dir
154     global Selected_Interfaces
155     global Sesions_Dir
156     global Current_Sesion_Dir
157     global Current_Script_Name
158
159     Current_Sesion_Dir = ''
160     Current_Script_Name = ''
161
162     GlobalDir = '/home/harold/Documents/Tesis/Programa'
163     ModelsDir = GlobalDir + '/Modelos'
164     FeynRulesDir = GlobalDir + '/feynrules-current'
165     Plantilla_FeynRules_Dir = GlobalDir + '/.Plantilla_FeynRules_Script/
            Plantilla_FeynRules_Script.m'
166     Sesions_Dir = GlobalDir + '/Sesiones'
167
168     Models_List = find_files_wth_ext(ModelsDir, '.fr')
169     ModelSelected_Info = "Seleccione los modelos que quiere incluir en el Notebook de
            Mathematica:"
170     Selected_Models = multi_select_menu(Models_List, ModelSelected_Info)
171
172     Lagrangians_List = obtain_lagrangians(Selected_Models, ModelsDir)
173
174     Selected_Lagrangians_Menu_Info = "Seleccione los lagrangianos que quiere incluir en el
            Notebook de Mathematica:"
175     Selected_Lagrangians = multi_select_menu(Lagrangians_List,
            Selected_Lagrangians_Menu_Info)
176
177     Interfaces = ["CalcHep/CompHep", "FeynArts/FormCalc", "Sherpa", "TEX", "UFO", "Whizard/
            Omega"]
178     Selected_Interfaces_Menu_Info = "Seleccione las interfaces en los que deseé que
            FeynRules exporte:"
179     Selected_Interfaces = multi_select_menu(Interfaces, Selected_Interfaces_Menu_Info)
180
```

```
181     create_script(Selected_Models, Selected_Lagrangians, Selected_Interfaces,
            Plantilla_FeynRules_Dir, FeynRulesDir, ModelsDir, Sesions_Dir)
182
183
184 if __name__ == '__main__':
185     input("Esta es la prueba del main")
186     FeynRules_Rutine_main()
```

B.3. files.py

```
1  # -*- coding: utf-8 -*-
2
3  import os
4  from .menu import menu
5  from datetime import datetime
6  from shutil import move
7
8
9  # Obtener el directorio de trabajo
10 def get_dir():
11     dir = os.getcwd()
12     return dir
13
14
15 # Seleccionar un directorio
16 def select_dir(Search_Dir, Info):
17     Dirs = os.listdir(Search_Dir)
18     Dirs.sort()
19     Num_Dir_Selected = menu(Dirs, Info)
20     Dir_Selected = Dirs[Num_Dir_Selected]
21     return Dir_Selected
22
23
24 # Buscar un directorio con una keyword
25 def select_dir_wth_keyword(Search_Dir, Keyword):
26     find_dir = ""
27     for root, dirs, files in os.walk(Search_Dir):
28         for dir in dirs:
29             if Keyword.lower() in dir.lower():
30                 find_dir = dir
31     return find_dir
32
33 # Buscar un archivo con una keyword
34 def find_files_wth_keyword(Search_Dir, Keyword):
35     find_files = []
36     for root, dirs, files in os.walk(Search_Dir):
37         for file in files:
38             if Keyword.lower() in file.lower():
39                 find_files.append(os.path.join(root,file))
40     return find_files
41
42
43 # Encontrar archivos con extensión
```

```
def find_files_wth_ext(search_path, ext):
    find_files = []
    for file in os.listdir(search_path):
        if file.endswith(ext):
            find_files.append(file)
    return find_files


# Codigo de archivo
def code_name(key):
    code = datetime.now()
    sesion_name = key + '_Sesion_' + code.strftime('%Y%m%d%H%M%S')
    return sesion_name


# Mover archivos
def move_files(dir_src, dir_dst, file_name):
   input()

# Escribir script de Mathematica para graficar
def plotScript(dataDir, dataCols):
    with open(
        os.path.dirname(dataDir) + '/plotScript.m', 'w') as file:
        file.write(f'"data = Import[{dataDir}", "Table"][[All,{dataCols}]];\n')
        file.write('interpolData = Interpolation[data, InterpolationOrder -> 1];\n')
        file.write('plot =ContourPlot[interpolData[x, y], {x, xmin, xmax}, {y, logYmin,
            logYmax}, Contours -> {0.12}, ContourShading -> Automatic, ColorFunction -> "
            Rainbow", FrameLabel -> {"m_χ [GeV]", "log10(σ_eff)"}, PlotRange -> All,
            PlotPoints -> 100, Mesh -> None, Frame -> True, GridLines -> None]')
```

B.4. menu.py

```
import os
import sys
import tty
import termios


def clear_screen():
    os.system('clear')


def get_key():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(fd)
        ch = sys.stdin.read(1)
        if ch == '\x1b':
            ch += sys.stdin.read(2)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch
```

```
22
23
24 def print_multi_select_menu(options, selected_indices, current_index, menu_info):
25     clear_screen()
26     print(menu_info)
27     print('\n')
28     for i, option in enumerate(options):
29         preprefix = "> " if i == current_index else "  "
30         prefix = "[X]" if i in selected_indices else "[ ]"
31         print(f"{preprefix}{prefix}{option}")
32     print('\n')
33     print("Seleccionar: Space \t Finalizar: Enter")
34
35
36 def multi_select_menu(options, menu_info):
37     selected_indices = set()
38     current_index = 0
39
40     while True:
41         print_multi_select_menu(options, selected_indices, current_index, menu_info)
42         key = get_key()
43
44         if key == '\x1b[A':
45             current_index = (current_index - 1) % len(options)
46         elif key == '\x1b[B':
47             current_index = (current_index + 1) % len(options)
48         elif key == ' ':
49             if current_index in selected_indices:
50                 selected_indices.remove(current_index)
51             else:
52                 selected_indices.add(current_index)
53         elif key == '\r':
54             return [options[i] for i in selected_indices]
55
56
57 def print_menu(options, selected_index, menu_info):
58     clear_screen()
59     print(menu_info)
60     print('\n')
61     for i, option in enumerate(options):
62         prefix = "> " if i == selected_index else "  "
63         print(f"{prefix}{option}")
64
65
66 def menu(options, menu_info):
67     selected_index = 0
68     while True:
69         print_menu(options, selected_index, menu_info)
70         key = get_key()
71
72         if key == '\x1b[A':
73             selected_index = (selected_index - 1) % len(options)
74         elif key == '\x1b[B':
75             selected_index = (selected_index + 1) % len(options)
76         elif key == '\r':
```

```
77            return selected_index
```

B.5. micromegas_routine_v2.py

```
1  #Librerias
2  import os
3  import subprocess
4  import time
5  from shutil import copy
6  import re
7  from numpy import random
8  from tabulate import tabulate
9  from multiprocessing import cpu_count, Manager, Pool
10 from tqdm import tqdm
11
12 #Funciones propias
13 from .menu import clear_screen, multi_select_menu, menu
14 from .files import select_dir, select_dir_wth_keyword, find_files_wth_ext
15 from .mO_f import create_project, get_funct, set_funct, vars_Info, parNone, parManual,
       parRand, seedsGen
16 import concurrent.futures
17
18 #Posible libreria para paralelizar
19
20 menu_info_1 = 'Seleccione la sesión con la que quiere trabajar:'
21
22 info_sesion_dir_1 = '\nNo se encontró ningún directorio con los archivos requeridos para
       ejecutar micrOMEGAS.\n\nPresione ENTER para volver.'
23 info_sesion_dir_2 = '\nNo se encontraron los archivos necesarios para ejecutar micrOMEGAS.\n
       \nPresione ENTER para volver.'
24
25 menu_info_2 = 'Seleccione las funciones que desea que micrOMEGAS realice.'
26
27 info_make = '\n\n\nNo se ha podido crear el programa "main.c"'
28
29 info_var = 'Seleccione los parametros con los que desea trabajar.'
30
31 info_random_1 = '--EDICIÓN DE PARAMETROS--\n\nSe han seleccionado las siguientes variables:\
       n\n\t'
32 info_random_2 = ['Dejar los valores establecidos en el modelo.', 'Ingresar un valor para
       cada parámetro.', 'Usar un valor aleatorio para cada parámetro.', 'Usar valores
       aleatorios en un rago para cada parámetro.']
33 info_random_3 = 'Ingrese los nuevos valores.\n'
34 info_random_4 = '\nIngrese el rango de los valores\nEjemplo.\n\t> parameterName: min_val,
       max_val'
35 info_random_5 = '\n\t> Tamaño del barrido: '
36
37 error_random_1= '\nError. Ingrese datos válidos.'
38
39 def main_micromegas(Global_Dir):
40 #Definir Directorios principales
41     Sesions_Dir = Global_Dir + '/Sesiones'
42     Micromegas_Dir = Global_Dir + '/micromegas_6.2.3'
```

```
43
44 #Seleccionar sesion con la que se va a trabajar
45     Sesion_Dir = f'{Sesions_Dir}/{select_dir(Sesions_Dir, menu_info_1)}'
46     Select_Sesion_Dir = select_dir_wth_keyword(Sesion_Dir, 'CH')
47     if Select_Sesion_Dir == "":
48         input(info_sesion_dir_1)
49         return
50     CH_Dir = f'{Sesion_Dir}/{Select_Sesion_Dir}'
51     mdl_files = find_files_wth_ext(CH_Dir, '.mdl')
52     if len(mdl_files) != 5:
53         input(info_sesion_dir_2)
54         return
55 #Crear el proyecto de micromegas y copia los archivos necesarios
56     Project_Dir = create_project(Global_Dir, Micromegas_Dir, CH_Dir, mdl_files)
57
58
59 #Seleccionar las funciones que queremos que micrOMEGAS haga
60     all_functions = get_funct(f'{Project_Dir}/main.c')
61     set_funct(f'{Project_Dir}/main.c', multi_select_menu(all_functions,menu_info_2))
62     clear_screen()
63
64
65 #Ejecutar make
66     os.system(f'cd {Project_Dir} && make main=main.c')
67     clear_screen()
68     if not os.path.exists(f'{Project_Dir}/main'):
69         input(info_make)
70         return
71
72 #Obtener la información del archivo vars.mdl
73     varName, varVal, varCom, varOpt = vars_Info(f'{Project_Dir}/work/models/vars1.mdl')
74
75
76 #Elejir que variables se van a cambiar
77     varSelect = multi_select_menu(varOpt, info_var)
78     delIndex = [i for i, element in enumerate(varOpt) if element not in varSelect]
79     for index in sorted(delIndex, reverse=True):
80         varName.pop(index)
81         varVal.pop(index)
82         varCom.pop(index)
83         varOpt.pop(index)
84
85 #Elejir el nombre del archivo de resultados
86     resultName = input('Escribe un nombre para el archivo de resultados:')
87
88     #Elejir el tipo de edición de parametros
89     randomOpt = menu(info_random_2, f'{info_random_1}'+'\n\t'.join(varOpt))
90
91 #Escritura de parámetros
92     Results_Dir = f'{Project_Dir}/Results'
93     resultFile = f'{Results_Dir}/results_{resultName}.txt'
94     header = list(varName) + ['omega']
95     os.system(f'mkdir {Results_Dir}')
96     #Caso 1: Dejar los datos igual
97     if randomOpt==0:
```

```
98          data = parNone(Project_Dir, varName, varVal)
99          with open(f'{resultFile}', 'a') as file:
100             file.write(tabulate(data,headers=header, tablefmt='plain'))
101     #Caso 2: Editar los datos manualmente
102     if randomOpt==1:
103         print(info_random_3)
104         varVal = []
105         for name in varName:
106             varVal.append(float(input(f'> {name}: ')))
107         data = parManual(Project_Dir, varName, varVal)
108         with open(f'{resultFile}', 'a') as file:
109             file.write(tabulate(data, headers=header, tablefmt='plain'))
110     #Caso 3: Editar aleatoriamente una vez
111     if randomOpt==2:
112         varRange = []
113         print(info_random_4)
114         for name in varName:
115             while True:
116                 try:
117                     rmin, rmax = map(float, input(f'\t> {name}: ').split(','))
118                     varRange.append([rmin, rmax])
119                     break
120                 except ValueError:
121                     print(error_random_1)
122         data = parRand(Project_Dir, varName, varRange, seedsGen(1))
123         with open(f'{resultFile}', 'a') as file:
124             file.write(tabulate(data, headers=header, tablefmt='plain'))
125     #Caso 4: Hacer un barrido de parámetros aleatorios
126     if randomOpt==3:
127         varRange = []
128         print(info_random_4)
129         for name in varName:
130             while True:
131                 try:
132                     rmin, rmax = map(float, input(f'\t> {name}: ').split(','))
133                     varRange.append([rmin, rmax])
134                     break
135                 except ValueError:
136                     print(error_random_1)
137         while True:
138             try:
139                 rand_size = int(input(info_random_5))
140                 break
141             except ValueError:
142                 print(error_random_1)
143         num_cores = max(cpu_count()-1,1)
144         input(f'\nSe usarán {num_cores} nucleos de {max(cpu_count(),1)} dispibles.\nPresione
                ENTER para iniciar el barrido.')
145         clear_screen()
146
147         #Iniciar la paraleización
148         data = []
149         seeds = seedsGen(rand_size)
150         ints_parallel = (Project_Dir, varName, varRange)
151         with concurrent.futures.ProcessPoolExecutor(max_workers=num_cores) as executor:
```

```
152             with tqdm(total=len(seeds)) as progress:
153                 futures = []
154                 for seed in seeds:
155                     future = executor.submit(parRand, *ints_parallel, seed)
156                     future.add_done_callback(lambda p: progress.update())
157                     futures.append(future)
158 
159             #futures = [executor.submit(parRand, *ints_parallel, seed) for seed in seeds]
160                 for future in concurrent.futures.as_completed(futures):
161                     result = future.result()
162                     data.append(result)
163 
164 
165         #Inicia la paralelización
166         with open(f'{resultFile}', 'a') as file:
167             file.write(tabulate(data,headers=header, tablefmt='plain'))
```

B.6. mO_f.py

```
1  import os
2  import re
3  import subprocess
4  from multiprocessing import Pool, cpu_count, Manager
5  from shutil import copy
6  from tabulate import tabulate
7  from numpy import random
8 
9 
10 from .menu import clear_screen
11 from .files import code_name
12 
13 
14 def create_project(global_dir, dir, sesion_dir, files):
15     clear_screen()
16     print('Creando Proyecto...\n')
17     name = code_name('MO')
18     os.system(f'cd {dir} && ./newProject {name}')
19 
20     for file in files:
21         copy(f'{sesion_dir}/{file}', f'{dir}/{name}/work/models')
22 
23     copy(f'{global_dir}/Modulos/main.c', f'{dir}/{name}')
24 
25     return f'{dir}/{name}'
26 
27 
28 def get_funct(dir_file):
29     with open(dir_file, 'r') as file:
30         lines = file.readlines()
31     functions = []
32     for line in lines:
33         if 'define' in line:
34             function = line.split("define",1)[1].split()[0].strip()
```

```
            functions.append(function)
    functions.remove('SHOWPLOTS*/')
    functions.remove('CLEAN')
    functions.remove('d(HIGGSBOUNDS)')
    return functions

def set_funct(dir, functions):
    for function in functions:
        with open(dir, 'r') as file:
            lines = file.readlines()
        with open(dir, 'w') as file:
            for line in lines:
                if '#define' in line and function in line:
                    file.write(f'#define {function}\n')
                else:
                    file.write(line)


def vars_Info(varFile):
    names = []
    values = []
    comments = []
    options = []
    with open(varFile, 'r') as file:
        lines = file.readlines()

    for line in lines:
        option = ''
        elements = []
        if '|' in line:
            if '|>' in line:
                pass
            else:
                elements = re.split(r'[|]', line)
                elements = [element.strip() for element in elements]
                if len(elements) > 2:
                    elements[2] = ' '.join(elements[2].split())
                options.append(f'{elements[0]} | {elements[2]}')
                names.append(elements[0])
                values.append(elements[1])
                comments.append(elements[2] if len(elements) > 2 else '')
    return names, values, comments, options


def getOmega(data):
    omega = data.split('Omega=')[1].split('\n')[0].strip()
    return omega

def seedsGen(size):
    seeds = set()
    while len(seeds) < size:
        seeds.add(random.randint(0,2**32-1))
    return list(seeds)


```

```
90 def parNone(dir, name, value):
91     run_main = f'cd {dir} && ./main data.par'
92     resultData = subprocess.run([run_main], shell=True, capture_output=True, text=True).
           stdout
93     value.append(float(getOmega(resultData)))
94     return [value]
95 
96 def parManual(dir, name, value):
97     temp_name = f'temp_{os.getpid()}.par'
98     temp_data = f'{dir}/{temp_name}'
99     with open(temp_data, 'a') as file:
100        for i, val in enumerate(value):
101            file.writelines(f'{name[i]} {val}\n')
102    run_main = f'cd {dir} && ./main {temp_name}'
103    resultData = subprocess.run([run_main], shell=True, capture_output=True, text=True).
           stdout
104    value.append(float(getOmega(resultData)))
105    os.remove(temp_data)
106    return [value]
107 
108 def parRand(dir, names, valRange, seed):
109    value = []
110    temp_name = f'temp_{os.getpid()}.par'  # Usar índice para nombres únicos
111    temp_data = f'{dir}/{temp_name}'
112 
113    random.seed(seed)
114 
115    with open(temp_data, 'a') as file:
116        for i, name in enumerate(names):
117            value.append(random.uniform(valRange[i][0], valRange[i][1]))
118            file.writelines(f'{name} {value[i]}\n')
119 
120    run_main = f'cd {dir} && ./main {temp_name}'
121    resultData = subprocess.run([run_main], shell=True, capture_output=True, text=True).
           stdout
122    value.append(float(getOmega(resultData)))
123 
124    os.remove(temp_data)
125    return value
```

B.7. plotting_routine.py

```
1 import time
2 
3 from .files import find_files_wth_keyword, select_dir, plotScript
4 from .menu import menu, clear_screen
5 import os
6 import subprocess
7 import sys
8 
9 #Información de menús
10 menu_info_1 = 'Seleccione el archivo que desea graficar'
11 
```

```
def main_plot(micromegas_dir):
#Buscar los archivos results_XXXXXX.txt
    result_files = find_files_wth_keyword(micromegas_dir, 'results')
    data_file = result_files[menu(result_files,menu_info_1)]

# Detectar los encabezados del archivo
    with open(data_file) as file:
        info = file.readline().split()

# Establecer los ejes y el parámetro de contorno
    x_column = menu(info,'Seleccione quien actuará como eje "x".')+1
    y_column = menu(info,'Seleccione quien actuará como eje "y".')+1
    contour_column = menu(info, 'Seleccione quien actuará como contorno.')+1

# Establecer el directorio y nombre del script
    plot_file_dir = os.path.dirname(data_file) + '/plotScript.m'

# Crear el script y ejecutarlo
    with open(plot_file_dir, 'w') as file:
        file.write(f'data = Import["{data_file}", "Table"][[All,{{{x_column},{y_column},{
            contour_column}}}]];\n')
        file.write('data = Rest[data];\n')
        file.write('interpolData = Interpolation[data, InterpolationOrder -> 1];\n')
        file.write(f'plot = ContourPlot[interpolData[x,y], {{x,{input('Especifica x_min: ')
            },{input("Especifica x_max: ")}}}, {{y,{input("Especifica y_min: ")},{input("
            Especifica y_max: ")}}}, Contours -> {{{input("Especifica la condición de
            contorno: ")}}}, ContourShading -> True, ColorFunction -> (Blend[{{White, Green,
             Gray}}, #] &), ScalingFunctions -> {{"Linear", "Log"}}, PlotRange -> All,
            PlotLegends -> Automatic] \n')
        file.write('Export["Plot.pdf",plot]\n')

    clear_screen()
    print(f'Ejecutando {plot_file_dir} ...\n')
    code = f'cd {os.path.dirname(data_file)} && math -script plotScript.m'
    os.system(code)

# Mostrar gráfica
    print('Mostrando gráfica...')
    time.sleep(2)
    subprocess.Popen(['xdg-open', os.path.dirname(data_file) + '/Plot.pdf'])

    #time.sleep(1)
    input()
```

Apéndice C: Código auxiliar.

C.1. Plantilla_FeynRules_Script.m

```
$FeynRulesPath = SetDirectory[]
<<FeynRules`
SetDirectory[ ]
LoadModel[]
FeynmanRules[]
(*WriteCHOutput[]*)
(*WriteFeynArtsOutput[]*)
(*WriteSHOutput[]*)
(*WriteLaTeXOutput[]*)
(*WriteUFO[]*)
(*WriteWOOutput[]*)
```

Bibliografía

[1] Georges Aad, Tatevik Abajyan, Brad Abbott, Jalal Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012.

[2] N Aghanim et al. Planck 2018 results. vi. cosmological parameters. *Astron. Astrophys*, 641:A6, 2020.

[3] G. Alguero, G. Bélanger, F. Boudjema, S. Chakraborti, A. Goudelis, S. Kraml, A. Mjallal, and A. Pukhov. micromegas 6.0: N-component dark matter. *Computer Physics Communications*, 299:109133, June 2024.

[4] Adam Alloul, Neil D. Christensen, Céline Degrande, Claude Duhr, and Benjamin Fuks. Feynrules 2.0 — a complete toolbox for tree-level phenomenology. *Computer Physics Communications*, 185(8):2250–2300, August 2014.

[5] Alexandre Arbey and Farvah Mahmoudi. Relic density and future colliders: inverse problem (s). In *AIP Conference Proceedings*, volume 1241, pages 327–334. American Institute of Physics, 2010.

[6] Stefan Ask, Neil D. Christensen, Claude Duhr, Christophe Grojean, Stefan Hoeche, Konstantin Matchev, Olivier Mattelaer, Stephen Mrenna, Andreas Papaefstathiou, Myeonghun Park, Maxim Perelstein, and Peter Skands. From lagrangians to events: Computer tutorial at the mc4bsm-2012 workshop, 2012.

[7] G Belanger, F Boudjema, and A Pukhov. micromegas: a code for the calculation of dark matter properties in generic models of particle interaction. *arXiv preprint arXiv:1402.0787*, 2014.

[8] G Belanger, F Boudjema, A Pukhov, and A Semenov. Micromegas: a tool for dark matter studies. *arXiv preprint arXiv:1005.4133*, 2010.

[9] Alexander Belyaev, Neil D Christensen, and Alexander Pukhov. Calchep 3.4 for collider physics within and beyond the standard model. *Computer Physics Communications*, 184(7):1729–1769, 2013.

[10] Gianfranco Bertone. *Particle dark matter: observations, models and searches*. Cambridge University Press, 2010.

[11] Serguei Chatrchyan, Vardan Khachatryan, Albert M Sirunyan, Armen Tumasyan, Wolfgang Adam, Ernest Aguilo, Thomas Bergauer, Marko Dragicevic, Janos Erö, Christian Fabjan, et al. Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1):30–61, 2012.

[12] Neil D Christensen and Claude Duhr. Feynrules–feynman rules made easy. *Computer Physics Communications*, 180(9):1614–1641, 2009.

[13] Neil D. Christensen and Claude Duhr. Feynrules – feynman rules made easy. *Computer Physics Communications*, 180(9):1614–1641, September 2009.

[14] Marco Cirelli, Alessandro Strumia, and Jure Zupan. Dark matter. *arXiv preprint arXiv:2406.01705*, 2024.

[15] GAMBIT Collaboration:, Peter Athron, Csaba Balázs, Torsten Bringmann, Andy Buckley, Marcin Chrząszcz, Jan Conrad, Jonathan M Cornell, Lars A Dal, Joakim Edsjö, et al. Status of the scalar singlet dark matter model. *The European Physical Journal C*, 77(8):568, 2017.

[16] Celine Degrande, Claude Duhr, Benjamin Fuks, David Grellscheid, Olivier Mattelaer, and Thomas Reiter. Ufo–the universal feynrules output. *Computer Physics Communications*, 183(6):1201–1214, 2012.

[17] Leanne D Duffy and Karl van Bibber. Axions as dark matter particles. *New Journal of Physics*, 11(10):105008, oct 2009.

[18] Joakim Edsjö and Paolo Gondolo. Neutralino relic density including coannihilations. *Physical Review D*, 56(4):1879, 1997.

[19] Katherine Garrett and Gintaras Dūda. Dark matter: A primer. *Advances in Astronomy*, 2011(1):968283, 2011.

[20] Murray Gell-Mann. A schematic model of baryons and mesons. *Physics Letters B*, 8:1–4, 1964.

[21] Sheldon L Glashow. Partial-symmetries of weak interactions. *Nuclear physics*, 22(4):579–588, 1961.

[22] David Griffiths. *Introduction to elementary particles*. John Wiley & Sons, 2020.

[23] Gerard Jungman, Marc Kamionkowski, and Kim Griest. Supersymmetric dark matter. *Physics Reports*, 267(5-6):195–373, 1996.

[24] Edward Kolb. *The early universe*. CRC press, 2018.

[25] Paul Langacker. *The standard model and beyond*. Taylor & Francis, 2017.

[26] Cesare MG Lattes, Hugh Muirhead, Giuseppe PS Occhialini, and Cecil Frank Powell. Processes involving charged mesons. *Nature*, 159(4047):694–697, 1947.

[27] Chong-sa Lim, Toshiyuki Morii, and Shankar Nath Mukherjee. *The physics of the standard model and beyond*. World Scientific, 2004.

[28] S. Navas et al. Review of particle physics. *Phys. Rev. D*, 110(3):030001, 2024.

[29] Brian Patt and Frank Wilczek. Higgs-field portal into hidden sectors. *arXiv preprint hep-ph/0605188*, 2006.

[30] Leszek Roszkowski, Enrico Maria Sessolo, and Sebastian Trojanowski. Wimp dark matter candidates and searches—current status and future prospects. *Reports on Progress in Physics*, 81(6):066201, may 2018.

[31] Vera C Rubin and W Kent Ford Jr. Rotation of the andromeda nebula from a spectroscopic survey of emission regions. *Astrophysical Journal, vol. 159, p. 379*, 159:379, 1970.

[32] Andrei D Sakharov. Violation of cp-invariance, c-asymmetry, and baryon asymmetry of the universe. In *In The Intermissions... Collected Works on Research into the Essentials of Theoretical Physics in Russian Federal Nuclear Center, Arzamas-16*, pages 84–87. World Scientific, 1998.

[33] Abdus Salam and John Clive Ward. Weak and electromagnetic interactions. *Il Nuovo Cimento (1955-1965)*, 11(4):568–577, 1959.

[34] Robert H Sanders. *The dark matter problem: A historical perspective*. Cambridge University Press, 2010.

[35] A.V. Semenov. Lanhep—a package for the automatic generation of feynman rules in field theory. version 3.0. *Computer Physics Communications*, 180(3):431–454, March 2009.

[36] Vanda Silveira and A Zee. Scalar phantoms. *Physics Letters B*, 161(1-3):136–140, 1985.

[37] F. Staub. Sarah, 2012.

[38] Joseph John Thomson. Xl. cathode rays. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 44(269):293–316, 1897.

[39] Steven Weinberg. A model of leptons. *Physical review letters*, 19(21):1264, 1967.

[40] Fritz Zwicky. Republication of: The redshift of extragalactic nebulae. *General Relativity and Gravitation*, 41(1):207–224, 2009.