Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Sistematización del análisis de desempeño y precisión para simulaciones de física de altas energías

Tesis de Licenciatura en Ciencias de la Computación

Alejandro Rubén Mignanelli

Director: Dr. Rodrigo Castro

Codirector: Dr. Lucio Santi

Buenos Aires, Argentina - 2024

RESUMEN

La simulación computacional es una parte fundamental en el dominio de la Física de Altas Energías (FAE). Esta intenta reproducir el movimiento de una partícula a través de una geometría de detectores, los cuales son representados mediante volúmenes de diversos materiales y formas, que imitan de forma precisa su estructura física y (por su propia naturaleza) producen discontinuidades al momento del cruce entre ellos.

Geant4 es la principal herramienta de simulación a nivel mundial para FAE, utilizada por experimentos de gran escala como los realizados en el *Large Hadron Collider* (LHC) de CERN. Los algoritmos que implementa Geant4 se basan en métodos numéricos clásicos, que discretizan el tiempo y deben interrumpir su flujo para manejar las discontinuidades que presenta la simulación.

Esta tesis se enmarca en un conjunto de trabajos que buscan proponer una solución alternativa que maneje de manera eficiente las discontinuidades con el fin de disminuir los tiempos de simulación. La familia de métodos *Quantized State System* (QSS) discretizan el estado del sistema, en vez de el tiempo, mitigando este problema mediante la resolución eficiente de funciones polinomiales simples.

En trabajos previos se consiguió producir un integrador QSS autónomo embebido directamente en Geant4, y se probó con éxito en modelados reales de detectores como el *Compact Muon Solenoid* (CMS), donde se pudo ofrecer una mejora de rendimiento en algunas situaciones, especialmente cuando el experimento produce muchos cruces geométricos.

En la presente tesis se expandirá el universo de pruebas de este integrador y se proporcionará una herramienta automática de comparación exhaustiva de experimentos entre integradores, con el fin de poder analizar una mayor cantidad de escenarios de manera rápida, ordenada, comparable, disminuyendo considerablemente las probabilidades de error humano. Disponer de esta herramienta permitió aportar por primera vez los métodos QSS a la versión oficial productiva de Geant4, lo que representa un hito sobresaliente para esta línea de investigación.

Palabras clave: Quantized State Systems, Simulación de Partículas, Física de Altas Energías, Geant4

Índice general

Índice de figuras

Índice de cuadros

1. INTRODUCCIÓN

La simulación computacional es una parte fundamental en el dominio de la Física de Altas Energías (FAE), ya que se propone reproducir el movimiento de una partícula a través de una geometría de detectores, los cuales son representados mediante volúmenes adyacentes de diversos materiales y formas, imitando de manera precisa su estructura física. Entre las aplicaciones más importantes de estas simulaciones se encuentra la de impulsar el diseño y optimización de los detectores, tratando así de mejorar su rendimiento. Esto se debe a que su diseño debe permitir un adecuado balance entre costos y rendimiento físico. Con el objetivo de obtener estos balances, se realizan baterías de simulaciones computacionales instanciadas de diferentes maneras.

Por lo antes dicho, la fase de experimentación computacional ha tomado gran relevancia a la hora de obtener financiamiento para los experimentos de FAE. No es de extrañar que ante la necesidad de mayor precisión y variedad de experimentos, se vuelva imprescindible incrementar la eficiencia de los simuladores. A modo de ejemplo ilustrativo, una mejora del 1 % en el módulo de simulación de Geant4 [14] (uno de los más utilizados para los experimentos FAE, y protagonista de este trabajo) puede generar ahorros del orden de los 80.000 USD anuales [15] para el proyecto del detector de partículas Compact Muon Solenoid (CMS) [11] en el Large Hadron Collider (LHC) [12] en el CERN [25].

1.1. Motivación

Los movimientos en el espacio tridimensional de las partículas subatómicas (protones, piones, neutrones, muones, etc.) se resuelven a través de Ecuaciones Diferenciales Ordinarias (EDOs), y los algoritmos que implementa Geant4 para solucionarlas se basan exclusivamente en métodos numéricos clásicos, basados en alguna forma de discretización del tiempo [9], sobre todo aquellos de la familia de algoritmos Runge-Kutta (RK) [2].

Sin embargo, el transporte de partículas en geometrías de detectores tiende a tener que resolver discontinuidades producidas por cruces geométricos de éstas mientras se desplazan por la geometría que modela el detector, y los métodos clásicos no están preparados para lidiar con estas situaciones de manera sencilla. Para resolverlas, interrumpen la integración

dando lugar a procedimientos iterativos que permitan detectar el tiempo y los valores de las variables de estado en el instante de cada discontinuidad. Esto resulta computacionalmente costoso y ha sido identificado como un buen punto de partida para tratar de mejorar la eficiencia de Geant4 en ciertos escenarios.

La familia de métodos numéricos Quantized State System (QSS) [5] discretiza las variables de estado de un sistema de EDOs, en lugar de discretizar el tiempo permitiendo, en teoría, resolver los problemas producidos por los cruces geométricos de manera más eficiente. QSS resuelven las EDOs utilizando aproximaciones a eventos discretos de los modelos continuos subyacentes [9, 4], y resuelven las discontinuidades más eficientemente utilizando funciones polinomiales de cruce por cero (que son tratadas como eventos discretos para los cuales los métodos están preparados por su propia definición) [7].

En trabajos anteriores se ha comprobado la potencialidad de QSS para simular modelos típicos FAE, y se lo ha probado en algunos escenarios con un integrador 2.2.1 embebido directamente en el módulo de transporte de partículas de Geant4 [22, 23].

Sin embargo, para poder determinar la conveniencia de QSS por sobre los métodos actuales, se requiere estudiar su uso en diversos escenarios, con distintas configuraciones y poder comparar diversos aspectos de estos experimentos. Por ello, surge la necesidad de tener un sistema de comparación que nos permita abordar esta experimentación de manera automática y ordenada.

1.2. Propuesta

En esta tesis nos proponemos crear un sistema capaz de comparar integradores entre sí. También se ampliará el universo de pruebas existente con el integrador QSS, y se utilizará el sistema creado para compararlos con algunos que utiliza hoy en día Geant4. Para cumplir con esto, realizaremos las siguientes acciones:

- Actualizaremos el código existente de Geant4 de G4 10.5 a la versión G4 11.0.0-ref-02.
- Se añadirá el código necesario para extraer datos que permitan una correcta comparación en términos de calidad de la solución, y velocidad de cómputo.
- Crearemos un sistema de comparación entre integradores que nos permitan obtener

gráficas y tablas comparativas.

- Ejercitaremos baterías de tests provistos por Geant4 para probar a QSS en distintas situaciones, comparando sus resultados frente a los generados con los métodos Runge-Kutta (RK) y Dormand-Prince (DOPRI) [1].

- Utilizaremos QSS en una aplicación realista de un detector principal de CERN (ATLAS) que utilice a Geant4 (toolkit FullSimLight [21]) y compararemos sus resultados con los generados con los métodos RK y DOPRI.

2. CONCEPTOS PRELIMINARES

En este capítulo introduciremos los conceptos fundamentales sobre los cuáles se desarrolla la tesis. Para esto, comenzaremos una breve explicación sobre resolución numérica de sistemas continuos, enseñando las diferencias más importantes entre los métodos clásicos basados en discretización temporal y los métodos basados en la discretización de estados como QSS. Luego, realizaremos una introducción a Geant4 y una breve explicación de FullSimLight, un software de gran relevancia que usa Geant4, y que utilizaremos en el presente escrito. Finalmente, contaremos el trabajo previo relacionado a esta tesis, como antecedente de la misma. Cabe aclarar que secciones de este capítulo se inspiraron en el trabajo [22]

2.1. Resolución numérica de sistemas continuos

Los sistemas continuos se caracterizan por tener variables de estado que van cambiando de manera ininterrumpida con el tiempo, y cuyo comportamiento puede modelarse mediante ecuaciones diferenciales, tanto ordinarias (EDOs) como parciales (EDPs). Aunque en algunos casos muy específicos es posible resolver estos sistemas de forma exacta, la gran mayoría no permite obtener soluciones analíticas, por lo que se recurre a métodos de integración numérica para simularlos.

En lo que sigue consideraremos un sistema autónomo de EDOs en la forma de la Ecuación 2.1, donde $\mathbf{x}(t)$ y $\mathbf{u}(t)$ representan el vector de estados y el vector de entradas, respectivamente. Este último consiste en variables independientes para las que no hay derivadas en el sistema.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.1}$$

La componente $x_i(t)$ del vector de estados representa la i-ésima trayectoria del sistema en función del tiempo. Mientras no haya discontinuidades en $f_i(\mathbf{x}, \mathbf{u})$, dicha función será también continua. Tales discontinuidades, en caso de que estuvieran presentes, deben ser manipuladas cuidadosa y eficientemente a lo largo de la simulación con el objeto de

preservar la precisión de los resultados.

2.1.1. Métodos de Integración Basados en la Discretización Temporal

Los algoritmos clásicos de integración numérica están basados en la discretización temporal. Dados los valores presentes y pasados de las variables de estado y sus derivadas, los métodos explícitos estiman los valores que adoptarán los estados en el siguiente paso de integración (o *paso*) h. Al no haber discontinuidades en 2.1, cada una de las trayectorias $x_i(t)$ es una función continua en el tiempo y, como tal, se puede aproximar en cualquier instante de tiempo t^* con cualquier precisión requerida utilizando la expansión en serie de Taylor:

$$x_i(t^* + h) = x_i(t^*) + \frac{dx_i(t^*)}{dt} \cdot h + \frac{d^2x_i(t^*)}{dt^2} \cdot \frac{h^2}{2!} + ... \tag{2.2}$$

Si se reemplaza con 2.1, se obtiene:

$$x_i(t^* + h) = x_i(t^*) + f_i(t^*) \cdot h + \frac{df_i(t^*)}{dt} \cdot \frac{h^2}{2!} + ... \tag{2.3}$$

Los métodos de integración típicamente varían no sólo en la forma de discretizar el tiempo (elección de h), sino también en cómo aproximan las derivadas de f y el número de términos en la serie de Taylor que se consideran en la aproximación.

La precisión con la que son aproximadas las derivadas de orden superior debe coincidir con la cantidad considerada de términos de la serie. En otras palabras, si se toman $n + 1$ términos, la precisión en la aproximación de la segunda derivada $\frac{d^2x_i(t^*)}{dt^2} = \frac{df_i(t^*)}{dt}$ debe ser de orden $n - 2$ dado que este factor se multiplica por h^2. De manera similar, la precisión de la tercera derivada debe ser de orden $n - 3$ dado que se multiplica por h^3. De este modo, la aproximación es correcta hasta h^n. A este valor de n se lo conoce como el *orden de aproximación* del método.

Mientras mayor es el orden de un método, más precisa es la estimación de $x_i(t^* + h)$. En consecuencia, al usar métodos de orden mayor, se puede integrar utilizando pasos h más grandes, ahorrando pasos de cómputo. Por otro lado, al usar pasos cada vez más chicos, los términos de orden superior de la serie de Taylor decrecen cada vez más rápido y la serie de Taylor puede truncarse antes. Es decir, el costo computacional de cada paso está muy ligado al orden del método en uso; los algoritmos de orden alto son mucho más

costosos, pero tienen un paso mucho mayor, necesitando menos de estos para terminar la simulación, por lo que, al fin y al cabo, se debe encontrar una solución de compromiso entre ambos factores.

Sin embargo, cuando intentamos utilizar estos métodos en sistemas discontinuos, como el caso de la geometría de detectores1.1, se suele tener que introducir soluciones ad-hoc para localizar y procesar discontinuidades. Esto es necesario para evitar integrar a través de una discontinuidad, lo que podría provocar resultados incorrectos. Pero si bien se puede evadir el problema de la continuidad, los algoritmos iterativos utilizados suelen ser computacionalmente demandantes, lo cuál en modelos con una gran cantidad de discontinuidades, podría causar notorias penalidades de desempeño.

Métodos Runge-Kutta

La familia de métodos Runge-Kutta [2] es probablemente la más utilizada dentro de los algoritmos de simulación de paso único (i.e., donde si se tiene información sobre un paso $\mathbf{x}_k = \mathbf{x}(t^*)$, entonces se puede calcular el próximo paso $\mathbf{x}_{k+1} = \mathbf{x}(t^* + h)$). Estos métodos computan las derivadas a través de una secuencia de etapas que evalúan el lado derecho de la Ecuación 2.1 a partir de distintos coeficientes que configuran cada método particular. Una de las variantes más utilizadas es el método Runge-Kutta de cuarto orden, conformado por cuatro etapas. Otro método que también pertenece a la familia Runge-Kutta es el método Dormand-Prince [1]. Este involucra seis evaluaciones de la función f para calcular soluciones de cuarto y quinto orden. Con esta última se determina qué tan grande es el error de la solución de cuarto orden, para poder ajustar el h con el que se va a mover. Por esto es que Dormand-Prince es de paso *adaptativo*.

2.1.2. Métodos de Integración Basados en la Discretización de Estados y Quantized State System (QSS)

Finalizando la década de los '90, Bernard Zeigler propone discretizar el estado del sistema en vez del tiempo, dejándolo fluir de forma continua [3]. Siguiendo esta línea de pensamiento y corrigiendo algunas falencias, Ernesto Kofman desarrolla en 2001 un algoritmo de primer orden bautizado *Sistemas de Estados Cuantificados* (en inglés Quantized State Systems, QSS) [5, 8], para el cual ofrece las pruebas necesarias de estabilidad y

convergencia que caracterizan a los métodos numéricos.

Formalmente, el método QSS de primer orden (QSS1) aproxima la EDO 2.1 por:

$$\dot{\mathbf{x}}(t) = f(\mathbf{q}(t), \mathbf{u}(t)) \tag{2.4}$$

en donde $\mathbf{q}(t)$ es un vector que contiene las versiones cuantificadas de las variables de estado $\mathbf{x}(t)$, mientras que $\mathbf{u}(t)$ representan variables de entrada. Cada variable de estado cuantificada $q_i(t)$ sigue una trayectoria constante de a tramos separados por un ***quantum*** ΔQ_i y está relacionada con la variable de estado $x_i(t)$por la **función de cuantificación**:

$$q_i(t) = \begin{cases} \lfloor x_i(t_0)/\Delta Q_i \rfloor \Delta Q_i, & \text{si } t = t_0 \\ x_i(t), & \text{si } |q_i(t^-) - x_i(t)| \geq \Delta Q_i \\ q_i(t^-), & \text{en caso contrario.} \end{cases} \tag{2.5}$$

donde $q_i(t^-)$ es el límite por izquierda de $q_i(t)$. De esta se puede observar que la diferencia entre $q_i(t)$ y $x_i(t)$ nunca puede superar ΔQ_i. Como se puede ver en la Figura 2.1, la función de cuantificación $\mathbf{q}(t)$ en los métodos QSS introduce un efecto de histéresis, propiedad necesaria para evitar modelos *ilegítimos* y es uno de los añadidos propuesto por Kofman que lo diferencian de trabajos anteriores.

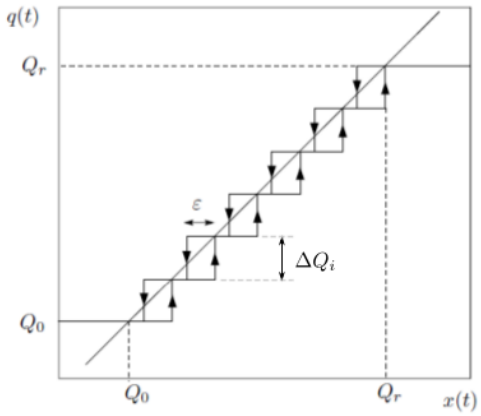

Fig. 2.1: Función de cuantificación con histéresis. El mapeo de $x(t)$ a $q(t)$ es asimétrico dependiendo la dirección de cambio de $x(t)$.

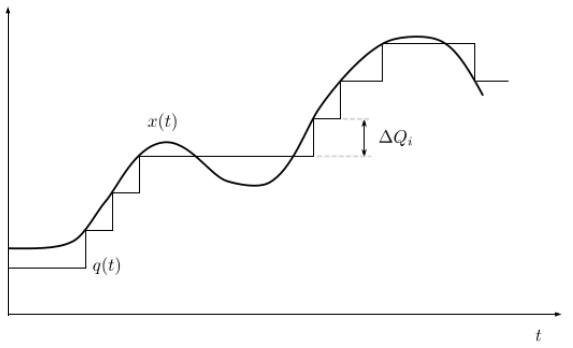

Fig. 2.2: Trayectoria para una variable $x(t)$ y su versión cuantificada $q(t)$ para un ΔQ dado.

La variable de estado cuantificado $q_i(t)$ sólo cambia cuando difiere de $x_i(t)$ en un valor

mayor al quantum ΔQ_i y el nuevo valor que toma en ese caso difiere del anterior en exactamente ΔQ_i. De esta manera, el quantum toma el papel de la tolerancia en los algoritmos clásicos de paso variable. En estas situaciones, el control sobre el largo del paso busca satisfacer una tolerancia de error relativa. En QSS esto se obtiene mediante un quantum relativo (ΔQRel) y un quantum absoluto (ΔQMin) cuyos valores son suministrados por el usuario según la precisión que desee siguiendo la siguiente ecuación:

$$\Delta \mathrm{Q}_i = \mathrm{máx}\left(\Delta\mathrm{QRel} \cdot |x_i|, \Delta\mathrm{QMin}\right)$$

Cada uno de estos cambios corresponde a un *paso de integración*. Entre esos pasos, en QSS1, los estados cuantificados $\mathbf{q}(t)$ siguen trayectorias constantes *por tramos*, como se puede observar en la Figura 2.2. Como las derivadas $\dot{\mathbf{x}}(t)$ son funciones de los estados cuantificados, también son constantes por tramos. Como consecuencia, la variable de estado $\mathbf{x}(t)$ será lineal por tramos.

Cuando QSS realiza un *paso* de integración en t_c, cambiando el valor de una variable cuantificada según $q_i(t_c) = q_i(t_c^-) \pm \Delta Q$, este cambio puede afectar algunas derivadas de otras variables de estado (en particular aquellas que dependan de $q_i(t)$), que deberán ser actualizadas en forma acorde. Es decir, cada *paso* de integración involucra únicamente a una variable de estado y su correspondiente variable cuantificada, generando la necesidad de reevaluar sólo aquellas derivadas que dependen explícitamente de dicha variable. Este hecho es especialmente útil al simular *sistemas ralos*[1]. Más aún, si una variable de estado no cambia su valor significativamente (cambio mayor al quantum), no provocará ningún *paso* de integración ni reevaluación alguna sobre otras variables que dependan de ella.

Entre los aspectos más importantes de QSS para este trabajo se encuentra la forma en que se manipulan las discontinuidades. Todas las trayectorias son aproximaciones polinomiales (más adelante veremos que dependiendo del orden del método pueden ser constantes, lineales o parabólicas). Esto tiene como consecuencia que detectar una discontinuidad (en rigor, activar un *evento de estado*) sea equivalente a encontrar el cruce por cero de dichas aproximaciones polinomiales. Para polinomios hasta grado 3 esto equivale a resolver una ecuación cúbica, lo cual puede ser resuelto en forma muy eficiente **sin itera-**

[1] Denominamos sistemas ralos a aquellos en donde la actualización de una variable afecta a un pequeño subconjunto de estados. El término surge porque, al escribir la matriz de incidencia de cada estado sobre las derivadas del resto, dicha matriz será rala.

ciones. Una vez que el algoritmo detecta la discontinuidad, ésta es tratada como cualquier otro *paso* de integración por QSS, ya que el método consiste justamente en secuencias de transiciones discontinuas entre un valor de estado y el subsiguiente. Por esto decimos que los métodos QSS manejan naturalmente las discontinuidades, siendo muy eficientes para simular sistemas que las presentan [7].

El método QSS1 que presentamos hasta aquí es un método de primer orden. Por lo tanto, para mantener un error pequeño en la simulación será necesario efectuar un número grande de *pasos*. Para reducir la cantidad de pasos manteniendo el error, se desarrollaron métodos QSS de orden superior: de segundo orden (QSS2 [6]) y de tercer orden (QSS3 [10]). En general, en estos métodos de órdenes superiores una variable de estado cuantificada $q(t)$ es una aproximación polinomial por tramos de $x(t)$ de la siguiente forma:

$$x(t) = \sum_{k=0}^{n} a_k \cdot (t - t_x)^k \tag{2.6}$$

$$q(t) = \sum_{k=0}^{n-1} a_k \cdot (t - t_q)^k \tag{2.7}$$

donde n es el orden del método. Esto es, en cualquier instante t, la variable de estado $x(t)$ es una suma finita de monomios hasta orden n en $\Delta t = (t - t_x)$. Los coeficientes de dichos monomios son los a_k, en donde el intervalo temporal se cuenta a partir de un $t_x \leq t$ que denota el comienzo del segmento polinómico. De manera acorde, $q(t)$ aproxima a $x(t)$ con un polinomio de un orden menor $(n - 1)$.

Estos conceptos los podemos visualizar en la Figura 2.3, que presenta una simulación QSS2 de la posición en el eje $\hat{x}$ de una partícula cargada en un campo magnético constante. En la Figura 2.3(a) podemos ver la trayectoria de la variable de estado $x(t)$ y la de su correspondiente estado cuantificado $q(t)$, compuestas por tramos cuadráticos y lineales respectivamente. Cada uno de los puntos destacados en la curva marca los límites de secciones polinomiales adyacentes. Secciones como (1) afectan los coeficientes de la variable de estado $x(t)$. Estas ocurren debido a una reacción a una actualización originada en otra variable de estado del sistema que afecta el cómputo de la derivada $\dot{x}(t)$. Por otro lado, secciones como (2) ocurren cuando se alcanza el quantum ΔQ (el máximo desvío entre $q(t)$ y $x(t)$). En estas situaciones, los coeficientes de $q(t)$ se recomputan cuantificando la variable de estado $x(t)$.

La diferencia entre $q(t)$ y $x(t)$ es el error $e(t)$ introducido por el método. Lo podemos ver en la Figura 2.3b. Se determina a través de los parámetros de precisión proporcionados por el usuario, ΔQRel y ΔQMin (que se pueden observar en la Figura 2.3c). Si nos focalizamos en la sección que comienza en el tiempo t_k, se denota que tanto $q(t)$ como $x(t)$ evolucionan hasta que la diferencia entre ellos alcanza el quantum ΔQ. Es ahora que $q(t)$ se actualiza cuantificando $x(t)$, dando lugar a una nueva sección polinomial en el gráfico. Este cambio se propaga al sistema evaluando aquellas variables de estado cuyo lado derecho depende de esta variable. Los coeficientes de las aproximaciones polinomiales de $q(t)$ y $x(t)$ pueden observarse en las Figuras 2.3d, 2.3e y 2.3f.

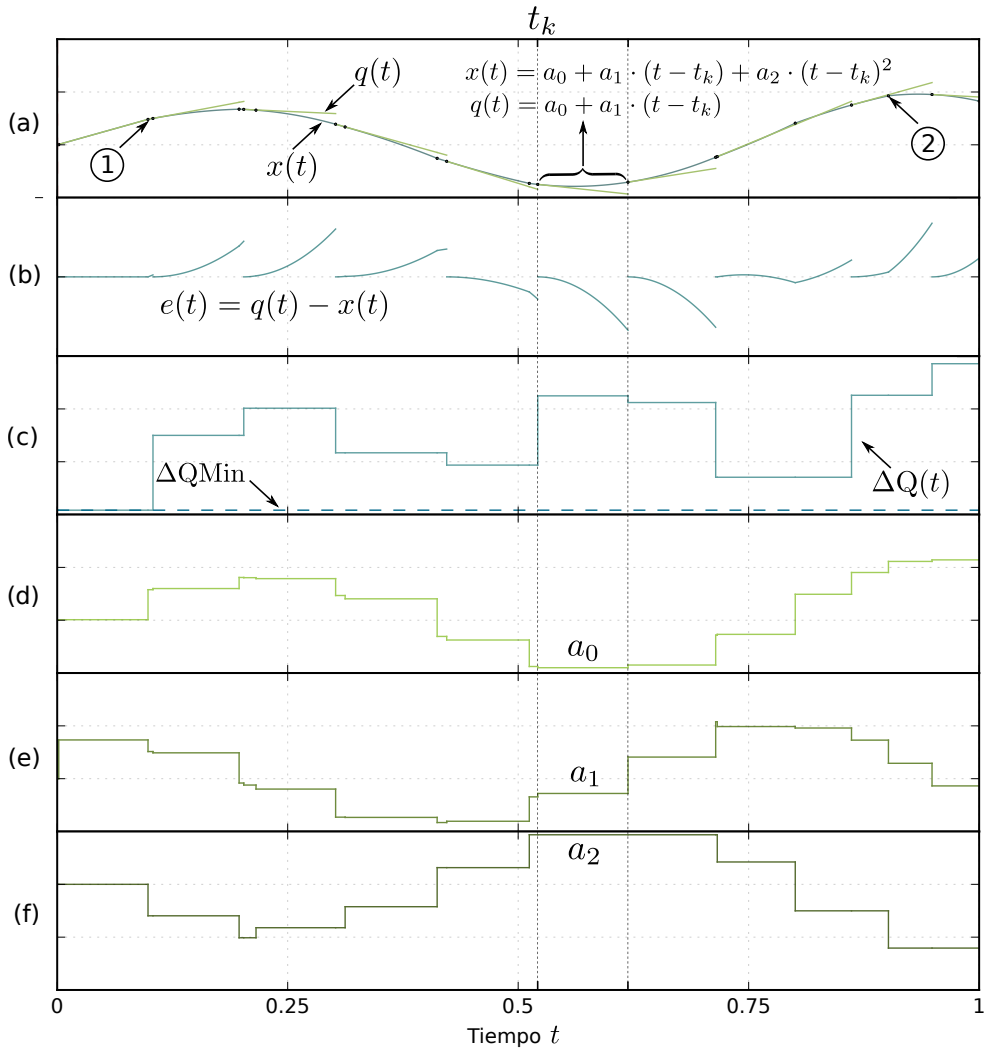

Fig. 2.3: Ejemplo ilustrativo de una simulación basada en QSS2.

En 2.1 se puede ver el algoritmo que detalla el funcionamiento de QSS2 con todo lo

discutido anteriormente y las acciones tomadas en cada paso de intergación.

Algoritmo 2.1: QSS2.

```
1  while(t < t_f) // simular hasta tiempo final tf
2    t = min(t_j) // avanzar tiempo de simulación
3    i = argmin(t_j) // el i–ésimo estado cuantificado cambia primero
4    // tiempo transcurrido desde la última actualización de xi
5    e = t − t_i^x
6    x_i = x_i + ẋ_i · e + 0,5 · ẍ_i · e^2 // actualiza el i–ésimo estado
7    ẋ_i = ẋ_i + ẍ_i · e // actualiza la derivada del i–ésimo estado
8    // actualizar el i–ésimo estado cuantificado
9    q_i = x_i
10   q̇_i = ẋ_i
11   // calcular el tiempo del próximo cambio en el i–ésimo estado cuantificado
12   t_i = min(τ > t) subject to |q_i(τ) − x_i(τ)| = ΔQ_i
13   for each j ∈ [1,n] such that ẋ_j depends on q_i
14     e = t − t_j^x // tiempo transcurrido desde la última actualización de xj
15     // actualiza el j–ésimo estado y su derivada
16     x_j = x_j + ẋ_i · e + 0,5 · ẍ_j · e^2
17     ẋ_j = f_j(q(t),t) // recalcular la derivada del estado
18     ẍ_j = ḟ_j(q(t),t) // recalcular la derivada segunda del estado
19     //recalcula tiempo del próximo cambio en el j–ésimo estado cuantificado
20     t_j = min(τ > t) subject to |q_j(τ) − x_j(τ)| = ΔQ_j
21     if j ≠ i then t_j^x = t // última actualización de xj
22   end for
23   t_i^x = t // última actualización de xi
24 end while
```

2.2. Herramientas de Simulación

En esta sección se describiran las herramientas de simulación que se utilizarán en este trabajo.

2.2.1. Geant4

Geant4 es la plataforma computacional más utilizada para simulaciones en FAE. Proporciona un conjunto completo de herramientas para todas las áreas de simulación de

detectores, focalizando principalmente en la interacción entre partículas y materia. Entre sus funcionalidades caben destacar:

- Crear un modelo de geometría con formas y materiales,
- Localizar puntos y navegar trayectorias en ese modelo,
- Aplicar los efectos de las interacciones físicas y generar partículas secundarias,

Geant4 fue diseñado para ofrecer flexibilidad en cuanto a los procesos físicos que permite simular, permitiendo además la manipulación de geometrías complejas(por ejemplo, la geometría de ATLAS, proyecto del que hablaremos en próximas secciones). El software se desarrolló originalmente en el CERN y está implementado en el lenguaje C++, utilizando programación orientada a objetos.

La Figura 2.4 presenta un diagrama de alto nivel de Geant4 donde destacamos dos de sus componentes más importantes: **Geometry**, encargado de describir una estructura geométrica y propagar partículas a través de ella, y **Processes**, que tiene como fin implementar los modelos de interacciones físicas.

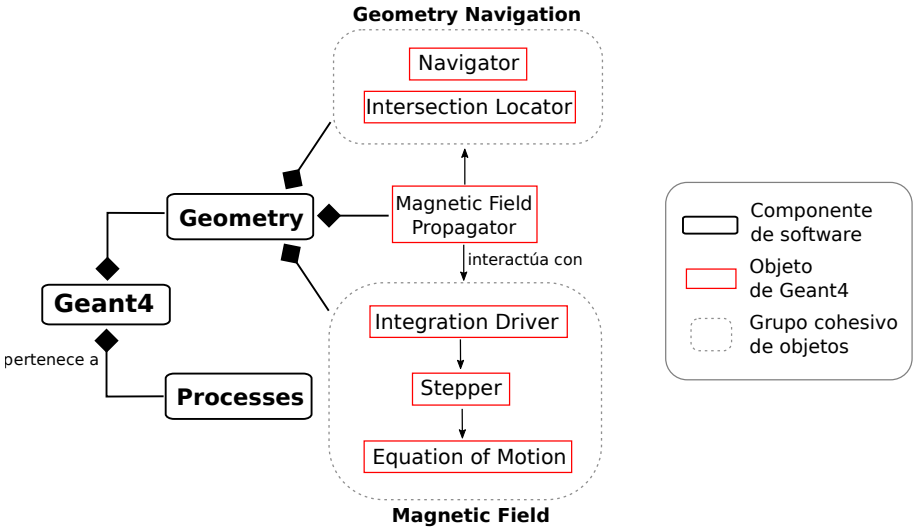

Fig. 2.4: Estructura de alto nivel de Geant4.

Una simulación Geant4 consiste en una serie de *eventos*, la unidad básica de simulación. Un evento se compone de uno o más *tracks*, que representan el estado de una partícula en un momento particular a lo largo de su trayectoria (cada una de estas instancias contiene cantidades físicas como la energía y el momento). Cuando comienza un nuevo evento, se

generan los *tracks primarios* y se envían a una pila de simulación. Los *tracks* se sacan de la pila de a uno por vez y se simulan. Las interacciones físicas pueden generar nuevos *tracks secundarios*, que también se introducen en la pila y se simulan en serie (las partículas no interactúan entre sí). Un evento finaliza cuando la pila se vacía.

Las trayectorias de las partículas se calculan a través de un complejo algoritmo de *propagación de partículas* [16] con varios parámetros de precisión que puede proporcionar el usuario: ϵ (controla el error relativo en la posición y el momento), $\Delta Chord$ (distancia máxima permitida de segmentos lineales a la trayectoria curva) y $\Delta Intersection$ (impone restricciones de precisión al cálculo de cruces de límites de volumen). El objeto *magnetic field propagator* (MFP) es el encargado de propagar partículas cargadas en un campo magnético mediante este algoritmo. Una trayectoria se compone de *pasos* que avanzan la partícula a una distancia dada (que el usuario puede limitar a un tamaño máximo fijo a través del parámetro *stepMax*). Cada uno de estos *pasos* se calculan mediante *integradores*, que proporcionan implementaciones personalizadas de algoritmos de integración numérica basados en Runge-Kutta para resolver las ecuaciones de movimiento subyacentes. El MFP no interactúa directamente con los integradores sino con un *driver de integración* que expone una interfaz de integración común para diferentes tipos de métodos numéricos.

Un *paso* puede terminar antes de cubrir su longitud en caso de alcanzar el límite de un volumen. Cuando Geant4 detecta tales situaciones, ejecuta un algoritmo iterativo para calcular el punto de intersección dentro de ciertas restricciones de precisión. Esto se implementa mediante el objeto *intersection locator*. De esta forma, la rutina principal de *propagación de partículas* se puede descomponer en el *cálculo de trayectorias* y *detección de cruces geométricos*, como se ilustra en la Figura 2.5. En este trabajo, nos enfocamos en el componente de *propagación de partículas* de las simulaciones Geant4 completas, de extremo a extremo, dejando fuera del alcance otros aspectos como la evaluación de los procesos físicos.

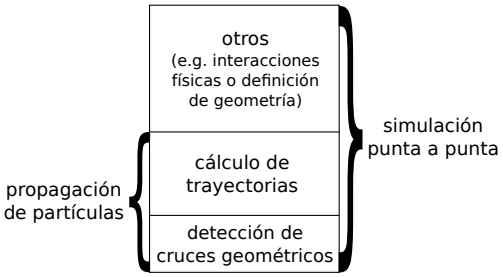

Fig. 2.5: Componentes principales de una simulación de Geant4.

Todos los aspectos configurables por el usuario de Geant4 (Longitud máxima del paso, cantidad de hilos utilizados, partículas lanzadas, campos magnéticos, etc) se manejan desde las macros. Estos son archivos de texto plano con una instrucción por línea que se ejecutan de manera secuencial y tienen la forma $instruction\ param_1\ param_2\ ...\ param_n$. Un ejemplo de macro se puede observar en la figura 2.6.

```
1 # Macro file for example B2
2 #
3 # To be run preferably in batch, without graphics:
4 # % exampleB2[a,b] run2.mac
5 #
6
7 # QSS Params
8 /QSS/selectStepper TemplatedDoPri
9 /QSS/dQMin 1
10 /QSS/dQRel 1
11
12 /run/numberOfThreads 1
13 /run/initialize
14 /run/verbose 1
15 /globalField/setValue 0.2 0 0 tesla
16 /run/beamOn 500
```

Fig. 2.6: Ejemplo de una macro de Geant4.

Desde la versión 10.4, lanzada en diciembre de 2017, el *integrador* predeterminado en Geant4 está basado en una implementación ad-hoc del método Dormand-Prince (DOPRI745). En las versiones anteriores, el integrador predeterminado se basaba en una implementación ad-hoc del método Runge-Kutta de cuarto orden (RK4).

2.2.2. FullSimLight

El experimento ATLAS (abreviación de "*A Toroidal LHC Apparatus*") es el detector más grande en el LHC (2.7), que es el acelerador de partículas más grande y de mayor energía que existe, además de la máquina más grande construida por el ser humano en el mundo. Investiga una amplia gama de la física, desde el bosón de Higgs hasta dimensiones extra y partículas que podrían formar la materia oscura. El framework Athena es la infraestructura principal de software utilizada para la experimentación computacional de ATLAS, y se usa para realizar simulaciones, reconstrucciones de trayectorias de partículas y análisis de propiedades físicas.

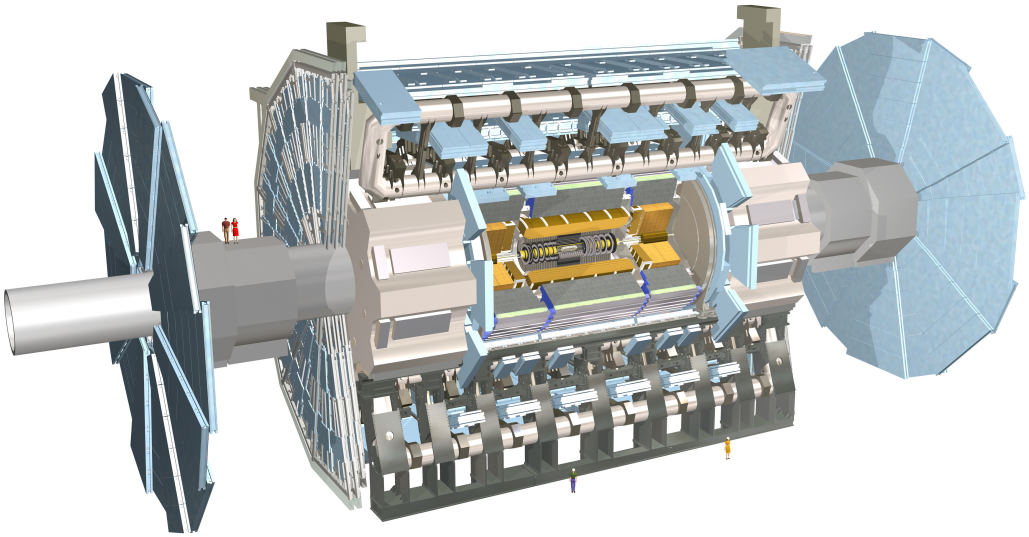

Fig. 2.7: Imagen computarizada del detector ATLAS. Al pie de la imagen se pueden observar personas, para entender el tamaño que tiene el detector.

Si bien tener un framework común en un proyecto permite armar un esqueleto de aplicación desde el cuál los desarrolladores pueden montarse para programar distintas funcionalidades manteniendo una cierta cohesión brindada por el framework, tener una

única pieza de software que se encargue de realizar todo el trabajo lleva a la sobrecomplejidad de este programa, complicando algunos flujos que podrían ser simplificados teniendo varias piezas que se encarguen de resolver distintas secciones de un problema. Es por esto que se crea FullSimLight (FSL) [21], una pieza de software independiente a ATLAS, encargada de desacoplar el proceso de simulación de la infraestructura del experimento.

2.3. Trabajo Relacionado

En el trabajo de tesis de Nicolás Ponieman [13] se abordó un estudio de factibilidad de la aplicación de métodos QSS a problemas de simulación en física de altas energías. En el trabajo se modeló una partícula cargada sometida a un campo magnético constante, describiendo trayectorias circulares y cruzando planos insertados a lo largo de una geometría bidimensional. Al comparar los modelos generados contra los de Geant4, se observó que QSS puede ofrecer importantes ventajas en situaciones de alta frecuencia de cruces geométricos, alcanzando mejoras de hasta seis veces en el tiempo de simulación. Estos resultados también se publicaron en [17].

A raíz de lo anterior, surgió la necesidad de proveer implementaciones de QSS en el contexto de Geant4, de modo de poder realizar comparaciones de desempeño más justas y al mismo tiempo permitiendo abordar una paleta más amplia de problemas. En primera instancia, se propuso una interfaz de software que permite una interacción directa entre Geant4 y QSS Solver: GQLink. Este mecanismo permitió conectar el motor de simulación de QSS Solver con Geant4 de manera tal de delegar las responsabilidades de integración numérica a los métodos QSS subyacentes.

En [18] se mostró que esta estrategia es metodológicamente válida, permitiendo simular correctamente no sólo casos de estudio introductorios como en [17] sino además aplicaciones realistas y complejas como simulaciones del detector CMS. No obstante, el caso particular de CMS demostró ser un desafío modesto desde el punto de vista de la frecuencia de cruces de volúmenes, es decir, no es una aplicación en donde la estrategia de co-simulación con QSS mejore significativamente los tiempos de simulación. Esto motivó abordar una nueva etapa en donde se realizaron optimizaciones de software variadas sobre GQLink, tanto algorítmicas como de bajo nivel [19]. Se realizaron nuevos estudios de desempeño en dos escenarios complementarios: una partícula cargada describiendo trayectorias helicoidales

en un reticulado 3D de cubos, y el modelo del detector CMS. Los resultados obtenidos para este último caso mostraron mejoras considerables respecto de [20], obteniendo ganancias moderadas respecto del integrador RK4 de Geant4.

Luego, en [22], se pudo probar que teniendo un integrador QSS autónomo embebido directamente en el módulo de transporte de partículas de Geant4 se podían mejorar los tiempos obtenidos por GQLink, e inclusive en los escenarios ensayados, bajo ciertos porcentajes de cruces geométricos, superar los tiempos de RK4 y DOPRI.

Cabe destacar que en [18] se pudo comprobar empíricamente en un par de aplicaciones FAE que QSS2 es el mecanismo más eficiente para tratar el problema, y es por esta razón, que en la presente tesis se trabajará con esta variación, y no con QSS1 ni QSS3, por ejemplo.

3. MIGRACIÓN A GEANT4 11 Y CAMBIOS INTRODUCIDOS

Como parte de este nuevo proceso de experimentación, el primer paso a realizar fue llevar el código de QSS a la última versión de Geant4 disponible (G4 11.0.0-ref-02). Por otro lado, se agregó la recolección de datos, para permitir una correcta comparación entre dos integradores.

Al momento de iniciar la tesis, se contaba con la versión 10.5, la cuál inmediatamente se tuvo que migrar a la versión 10.7.2. Esta migración obligó a realizar cambios más bien estructurales, dado que las relaciones entre las clases sufrieron modificaciones. Luego de esto, se trabajó con distintos experimentos que se enunciarán más adelante, hasta tener que realizar una nueva y final migración hacia la versión 11. Ésta no tuvo diferencias destacables que afectaran a QSS, pero fue luego de ella que se añadió la recolección de datos.

Cabe destacar que estas actualizaciones, y los análisis que se verán en otros capítulos de esta tesis llevaron a que el 08/12/2023, para el release 11.2.0-ref-02 , **QSS se introdujera por primera vez como integrador de Geant4 productivo, marcando un hito en esta investigación** 2.3.

En la figura 3.1 se puede observar la estructura de alto nivel de Geant4 luego de la incorporación de QSS.

Los cambios introducidos en esta migración fueron los siguientes.

3.1. Cambios estructurales

3.1.1. Los integradores deben conocer su driver

Los integradores de Geant4 (incluido el de QSS) sólo funcionan con un *interpolating driver*, es por eso que se agrega el método *build_driver* a G4MagIntegratorStepper (y sus subclases) para que se instancie y sea conocido el driver correspondiente en cada caso. También se agrega el método isQSS a la clase G4ChordFinder para que pueda llamar al *build_driver* correspondiente.

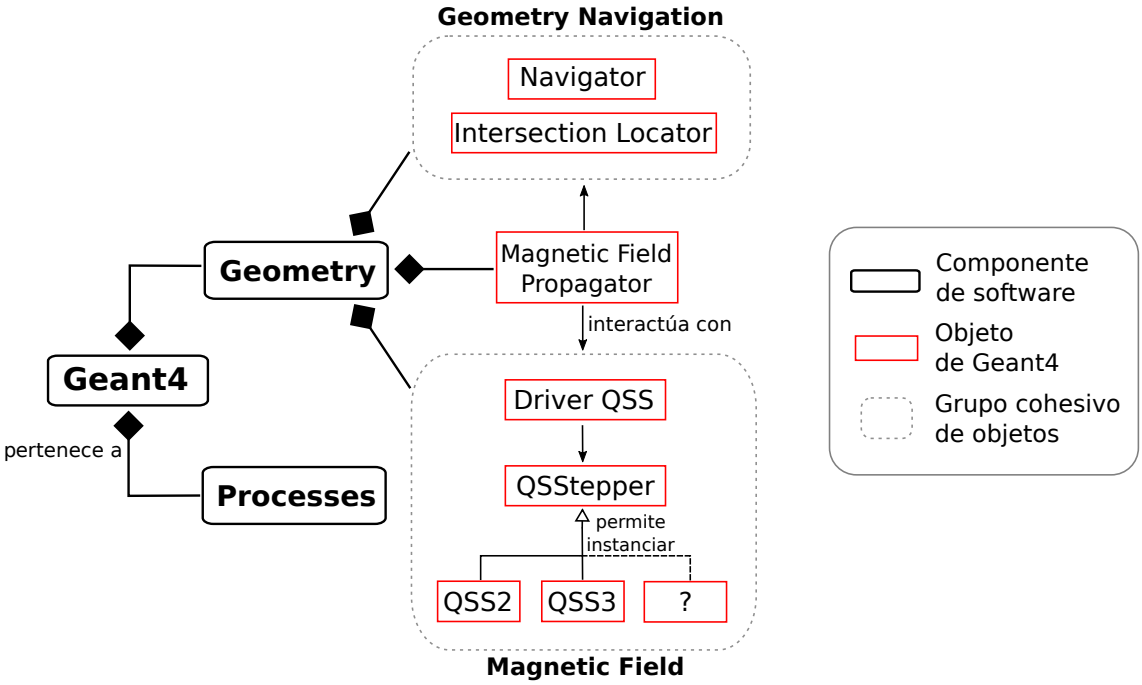

Fig. 3.1: Estructura de alto nivel de Geant4 luego de la incorporación de QSS

3.1.2. El método onComputeStep debe tener un parámetro de clase G4Track

En la nueva versión, el método onComputeStep debe llevar un parámetro que le permita utilizar información del track actual. Éste se agrega en toda la jerarquía de clases de G4VIntegrationDriver, y en G4ChordFinder.

3.1.3. Cambios para la recolección de datos

- Se crea la clase QSSMessenger que se encarga de recolectar de las macros 2.2.1 parámetros necesarios para el correcto funcionamiento de QSS. Esto permite, por ejemplo, la cómoda parametrización de los valores de dQMin y dQRel que utilizará el integrador QSS, permitiendo hacer barridos de parámetros sin mayores dificultades, para así intentar obtener una configuración óptima. Se añade también la capacidad de modificar el integrador utilizado vía macro, para no tener que recompilar el programa cada vez que se quiera cambiar el integrador.

- Se agrega más información estadística a recolectar a lo largo de una corrida para QSS como por ejemplo:

  - Cantidad de steps realizados
  - Cantidad de substeps realizados
  - Substeps promedio por step
  - Tiempo
  - Cantidad de Cambios por dQRel en variables de los ejes x, y, z
  - Cantidad de Cambios por dQRel en velocidad de los ejes x, y, z

  Entre otros. Muchos de estos datos permiten entender el escenario, y pueden explicar una diferencia de tiempos con otro integrador que no sea QSS.

- Se agregan valores por *default* para dQMin y dQRel para QSS, por si no se informaran por macro.
- Se permite *multithreading* con las características anteriormente enunciadas.
- Se crea la clase ParticleMovingLog que se encarga de guardar posición, tiempo y longitud recorrida de una partícula a través de un track. Luego al finalizar la ejecución del programa, se devuelve un .csv con la información que guardó esta clase. Esta acción solo se realiza si la macro de entrada tiene la instrucción generateVTKS.

Cabe destacar que durante el proceso de experimentación, se observó que en algunos experimentos, cuando el primer step era muy grande, ocurría un comportamiento no deseado donde QSS se quedaba calculando durante mucho tiempo, y por alguna razón que se desconoce, se salía de los límites de la geometría. Por esto, se agregó el parámetro trialProposedStepModifier que se encarga de reducir este primer step y es configurable vía macro.

4. SISTEMA AUTOMÁTICO DE COMPARACIÓN DE INTEGRADORES

Si bien la necesidad final es reducir los tiempos de cómputo del módulo de simulación de Geant4, no sería aceptable conseguirlo en desmedro de una reducción significativa de la calidad de la solución. Por otro lado, solo con tener parámetros de tiempo y calidad no alcanza, ya que una calidad mala con un dQMin y dQRel altos en QSS es esperable, tanto como una buena performance de tiempos en un experimento con muchos cruces geométricos.

Esta tesis propone estandarizar un sistema de comparación de integradores y demostrar su utilidad en diversos escenarios. Con este fin, se crearon una serie de *scripts* que al ejecutarse, obtienen gráficos y tablas que permiten explicar y entender los experimentos que fueron utilizados para crearlos.

El funcionamiento de estos *scripts* es el siguiente.

1. Se crean las macros con las que se van a correr los experimentos y se elige una que va a ser usada como referencia para ciertas mediciones de calidad y tiempo (cabe aclarar que cada macro corre un solo integrador a la vez).

2. Por cada macro creada se corre un experimento n veces y se extrae información. La información más relevante se introduce en un archivo llamado **RelevanStats.csv**. Aparte, se produce otro archivo por cada macro con información sobre tiempo, posición y longitud recorrida de una partícula, llamado **PML** (por *Particle Moving Log*). El mismo se tomará como base para determinar diferencias y a partir de ellas las mediciones de calidad. También, se producen una serie de gráficos con toda la información recolectada.

3. Los PMLs generalmente tienen una base temporal diferente, por lo que para poder compararlos, se interpolan las posiciones y tiempos de cada uno con el de referencia. Una vez hecho esto, se obtienen los MSEs y se produce un gráfico 3D comparativo entre ambas trayectorias de la partícula. (El MSE es el error cuadrático medio, en inglés Mean squared Error, y evalúa la precisión entre una estimación y aquello que

se busca estimar 6.1)

Los *scripts* que se corren y el orden son:

- `macroGenParamSweep.py`
- `experimentRunner.py`
- `ExperimentCriticalInformation.py`

4.1. macroGenParamSweep

El objetivo de este script será crear las macros 2.2.1 correspondientes a nuestro experimento.

4.1.1. Parámetros de entrada

Parámetro	Obligatorio	Descripción
`param`	Sí	Este parámetro puede repetirse, y siempre tendrá la forma "*clave* $valor_1$ ... $valor_n$" con $n \in N$. Cada repetición representa las variaciones que se le desea realizar a la macro de referencia.
`paramCombination`	Sí	Este puede tomar solo los valores *cartesian* u *ordered*. Este representa las combinaciones de `param` que se realizarán.
`macroTemplateFilename`	Sí	Archivo que contiene una macro de referencia.
`outputFolder`	Sí	Directorio donde se guardarán las macros creadas.

Tab. 4.2: Parámetros de entrada de macroGenParamSweep.py

Algoritmo 4.1: Ejemplo de uso

```
1 python macroGenParamSweep.py \
```

```
2   --param stepper QSS2 QSS2 QSS2 \
3   --param dQMin 0.01 0.001 0.0001 \
4   --param dQRel 0.1 0.01 0.001 \
5   --param trialProposedStepModifier 0.0006 0.0006 0.0006 \
6   --param numberOfThreads 1 1 1 \
7   --paramCombination ordered \
8   --macroTemplateFilename exampleB2a_withMagField.in \
9   --outputFolder tmpMacros
```

4.1.2. Explicación

El *script* tomará una macro de referencia, que funcionará como un *template* y desde la cuál se desea crear distintas macros para realizar experimentos, ubicándolos en un directorio en particular. Las secciones de esta macro de referencia que deseen ser modificadas, deberán tener la forma $\${sectionToInsert}$, con $sectionToInsert$ una palabra cualquiera que luego deberá aparecer como clave en algún `param` al invocar el script. Un ejemplo de macro de referencia se puede encontrar en la figura 4.1. Ésta corresponde con el ejemplo introducido anteriormente 4.1. Luego, las macros que se generarán dependerán del `paramCombination` utilizado. En caso de usarse *cartesian*, se creará una macro por cada combinación posible de valores de las claves de los `param`. En cambio, si se utiliza *ordered*, se deben enviar **siempre** la misma cantidad de valores para cada clave de los `param`, y se creará una macro por cada índice de los valores de las claves con la combinación de valores de dicho índice.

El diagrama de clases se puede observar en la figura 4.2 . `macroGenParamSweep.py` se encarga de juntar todos los `param` de entrada y cambiarlos a una estructura en forma de una lista de tuplas `clave:valores`. Siguiendo el ejemplo introducido en 4.1, la estructura sería:

```
[stepper:[QSS2,QSS2,QSS2], dQMin:[0.01, 0.001, 0.0001],
dQRel:[0.1, 0.01, 0.001], trialProposedStepModifier:[0.0006, 0.0006, 0.0006],
numberOfThreads:[1, 1, 1]]
```

Luego con esta estructura y el resto de los parámetros, el script crea un objeto `paramSweeper` y le envía a este el mensaje `sweep`. Este mensaje se encarga de tomar la lista de tuplas `clave:valores` y transformalo en una lista de lista de tuplas `clave:valor`.

```
1 # Macro file for example B2
2 #
3 # To be run preferably in batch, without graphics:
4 # % exampleB2[a,b] run2.mac
5 #
6
7 # QSS Params
8 /QSS/selectStepper ${stepper}
9 /QSS/dQMin ${dQMin}
10 /QSS/dQRel ${dQRel}
11 /QSS/trialProposedStepModifier ${trialProposedStepModifier}
12
13 /run/numberOfThreads ${numberOfThreads}
14 /run/initialize
15 /run/verbose 1
16 /globalField/setValue 0.2 0 0 tesla
17 /run/beamOn 500
```

Fig. 4.1: Ejemplo de una macroTemplate para el script macroGenParamSweep, donde se resaltaron las claves utilizadas e indicadas en el ejemplo 4.1

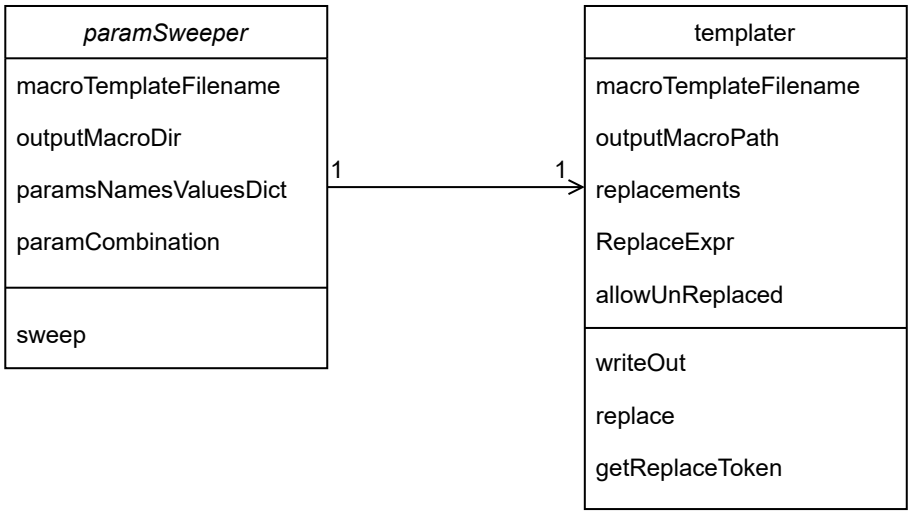

Fig. 4.2: Diagrama de clases del script macroGenParamSweep

En el ejemplo 4.1 , la transformación sería la siguiente:

```
Entrada: [stepper:[QSS2,QSS2,QSS2], dQMin:[0.01, 0.001, 0.0001],
dQRel:[0.1, 0.01, 0.001], trialProposedStepModifier:[0.0006, 0.0006, 0.0006],
numberOfThreads:[1, 1, 1]]
Salida: [[stepper:QSS2, stepper:QSS2, stepper:QSS2],
[dQMin:0.01, dQMin:0.001, dQMin:0.0001],
[trialProposedStepModifier:0.0006, trialProposedStepModifier:0.0006,
trialProposedStepModifier:0.0006],
[numberOfThreads:1, numberOfThreads:1, numberOfThreads:1]]
```

Luego de esto se crea una lista de N-tuplas con N la cantidad de claves, y tomando de base la transformación, si `paramCombination` es:

- *ordered*, resultará obligatorio que todas las claves tengan la misma cantidad de elementos(caso contrario se producirá un error) y la lista respetará que cada elemento de la tupla es el i-ésimo elemento de una de las listas que representan claves. No puede haber repetidos ni se pueda llegar por permutaciones a un elemento ya existente. Ejemplo [[a:1, a:2],[b:3, b:4], [c:5,c:6]] termina en [(a:1,b:3,c:5),(a:2,b:4,c:6)]

- *cartesian*, la lista contendrá todas las combinaciones posibles de tuplas `clave:valor` tal que haya un solo representante de cada clave, no haya repetidos ni se pueda llegar por permutaciones a un elemento ya existente. Ejemplo [[a:1, a:2],[b:3, b:4], [c:5]] termina en [(a:1,b:3,c:5),(a:1,b:4,c:5),(a:2,b:3,c:5),(a:2,b:4,c:5)] .

Luego, se crea un objeto `templater` pasándole la lista de N-tuplas, la macro de referencia y una lista de nombres para cada uno de los elementos de la N-tupla. Una vez construído, se le envía al nuevo objeto el mensaje `writeOut`. Al recibir el mensaje, el `templater`, para cada elemento de la N-tupla, reemplaza las claves por los valores en una copia de la macro de referencia, y coloca los archivos con las macros resultantes en el directorio indicado anteriormente con el nombre correspondiente según la lista.

4.2. experimentRunner

El objetivo de este *script* será correr experimentos, obtener los datos crudos, y presentarlos de forma ordenada, de manera tal que sea sencillo realizar un análisis con el cuál podamos entender que ocurrió en cada experimento, además de compararlos entre sí. Esta presentación puede ser en archivos de texto plano en formato *json* u ordenado en forma de grilla, tablas en formato csv, histogramas o gráficos de curvas.

Específicamente, lo que se obtendrá al correr este *script* será:

- un archivo `experiment.data` que nos indica qué corridas pertenecen a qué macros

- un archivo `stats.out` con todos los datos con los que se confeccionaron los gráficos en el directorio `imgs`

- un directorio `imgs` con gráficos que cambiarán dependiendo si estamos usando QSS como integrador o no. Si no estamos usando QSS, habrá un gráfico de curvas y un

histograma por cada una de estas características:

- intersecciones
- cantidad de threads
- tiempo Real
- tiempo de sistema
- tiempo de usuario
- total de steps

En caso de estar usando QSS, habrá un gráfico de curvas y un histograma por cada una de las características antes mencionadas, y las que siguen a continuación:

- dQMin
- dQRel
- tiempo de integración
- tiempo de integración promedio por *step*
- tiempo de integración promedio por *substep*
- cantidad de *tracks*
- tiempo de *reset*
- tiempo de reset promedio por *step*
- tiempo de reset promedio por *substep*
- cantidad de *substeps*
- cantidad de *steps* en *track* 1
- cantidad de *substeps* en *track* 1
- cambios de la variable VX por dQMin
- cambios de la variable VX por dQRel
- máximo error de la variable VX
- cambios de la variable VY por dQMin
- cambios de la variable VY por dQRel

- máximo error de la variable VY
- cambios de la variable VZ por dQMin
- cambios de la variable VZ por dQRel
- máximo error de la variable VZ
- cambios de la variable X por dQMin
- cambios de la variable X por dQRel
- máximo error de la variable X
- cambios de la variable Y por dQMin
- cambios de la variable Y por dQRel
- máximo error de la variable Y
- cambios de la variable Z por dQMin
- cambios de la variable Z por dQRel
- máximo error de la variable Z

En todos los casos, los ejes de los gráficos son la característica correspondiente y el número de corrida.

- un archivo `RelevantStats.csv` que contiene una tabla con un resumen de los datos más relevantes recogidos. Un ejemplo puede observarse en la figura 4.3

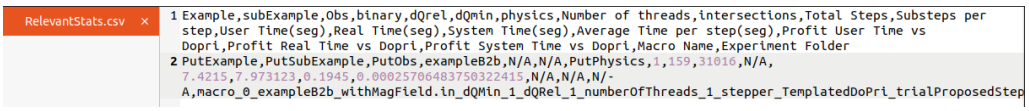

Fig. 4.3: Ejemplo del archivo RelevantStats.csv sobre un experimento que usaba como integrador a DOPRI. El archivo tiene como contenido información sobre una sola macro, donde la primer línea representa el encabezado, mientras que la segunda es información recolectada por el script. Esta se encuetra incompleta, como lo marcan los elementos que empiezan con el prefijo "put", y deberá completarse en el próximo script

También habrá un subdirectorio por cada macro corrida como muestra la figura 4.4. En cada uno de ellos habrá:

- directorios cuyo nombre es el número de corrida qué representan y qué contienen:

- la macro que corrieron
- un archivo `run.out` que devuelve Geant4 al correr un experimento con toda la información de este. Si el archivo supera los 2 MB, estará comprimido.
- un archivo `stats.out` con todos los datos recolectados del `run.out`

- un archivo `stats.out` que recolecta toda la información de todos los archivos `stats.out` de los directorios de las corridas.
- un directorio `imgs` con los mismos gráficos y con los mismos criterios que en el directorio `imgs` previamente descripto.
- un directorio llamado `aSampleRun` que contiene una corrida extra, que posee información sobre la calidad del experimento y que no se utiliza como dato para realizar los gráficos antes mencionados, como muestra la figura 4.5. Dentro de este se encuentra:
  - la macro que se ejecutó
  - un archivo `run.out` que devuelve Geant4 al correr un experimento con toda su información. Si el archivo supera los 2 MB, estará comprimido.
  - un archivo `stats.out` con todos los datos recolectados del `run.out` en formato *json*, más legible.
  - una carpeta llamada `PML` que contiene un archivo con la posición de la partícula del *track* primario a través del tiempo.

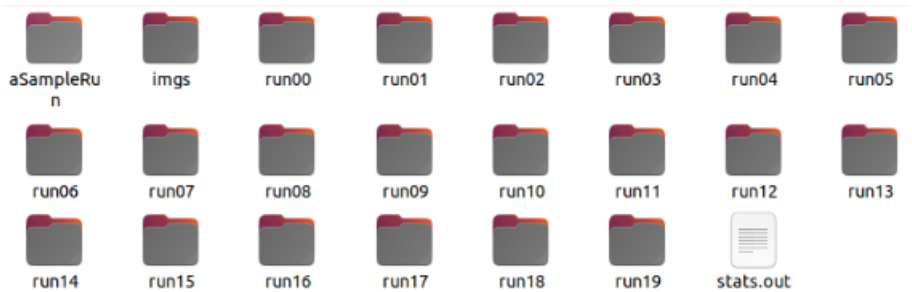

Fig. 4.4: Ejemplo de subdirectorios y archivos creados por macro corrida.

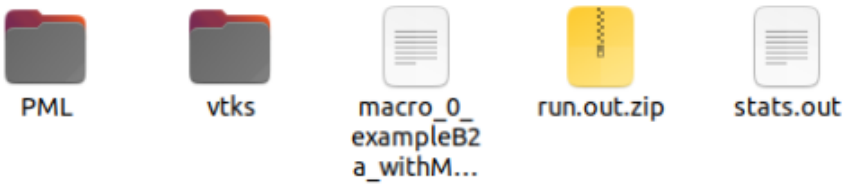

Fig. 4.5: Ejemplo de subdirectorios y archivos creados dentro de la carpeta `aSampleRun`.

4.2.1. Parámetros de entrada

Parámetro	Obligatorio	Descripción
`experimentName`	Sí	Nombre del directorio donde se guardarán los resultados. **NO** puede existir otro igual en la ruta donde se corre el *script*.
`geometryPath`	No	Geometría del experimento. Para algunos experimentos es necesaria la geometría donde se corren.
`runArgsMode`	No	Puede contener $B5like$, $B4like$ o FSL. Dependiendo el binario que se use, el comando para ejecutarlo puede variar, teniendo este parámetro la función de determinar que comando se ejecuta. Por *default*, toma B5like.
`stringForParser`	No	Dependiendo del Binario que se use, el run.out puede ser distinto. Este parámetro permite identificar el punto a partir del cuál se debe empezar a parsear los resultados en el `run.out`. Por *default* tiene "G4 kernel has come to Quit state([\S\s]+)".
`noQSSOutput`	No	Si se encuentra este parámetro, se le indica al script que este experimento no usa el integrador QSS, y no se intenta de parsear datos particulares del método (por ejemplo, dQMin).
`noPlots`	No	Si este parámetro está presente, no se realiza ninguna gráfica.
`noPerTrackStats`	No	Si este parámetro está, no se intenta de obtener los datos de *steps* y *substeps* por cada *track*.
`binary`	Sí	Binario que se debe correr.
`macroSource`	Sí	Directorio donde se encuentran las macros que se deben correr.
`numberOfRuns`	Sí	Indica la cantidad de corridas por macro.
`noSampleRun`	No	No crea la corrida *Sample* en cada macro y no devuelve información sobre la calidad del experimento.

Tab. 4.4: Parámetros de entrada de experimentRunner.py

Algoritmo 4.2: Ejemplo de uso

```
1 python3 experimentRunner.py \
2  --experimentName exampleB2BParaViewQSS \
3  --binary /home/Tesis/g4-Qss-Cern-MyBranch/B2b-build/exampleB2b \
4  --macroSource /home/Tesis/Geant4-qss/experimentos/tmpMacrosB2BParaViewQSS \
5  --numberOfRuns 1
```

4.2.2. Explicación

El diagrama de clases de `experimentRunner.py`, se puede observar en la figura 4.6

Lo primero que hace el *script* luego de tomar todos los parámetros de entrada es crear un objeto `runConfig` con el cuál creará otro objeto `multiMacroExperiment`. `runConfig`, al crearse, verificará inmediatamente que las variables de ambiente de Geant4 hayan sido seteadas,y en caso contrario devolverá un error. `multiMacroExperiment` por otra parte, se armará una lista de `singleMacroExperiment` con cada una de las macros que tiene en `macroFiles`, y las guardará en su atributo `runs`. Luego, se le enviará el mensaje `run` a `multiMacroExperiment`, seguido del mensaje `parseResults`.

Mensaje run al multiMacroExperiment

Cada objeto `singleMacroExperiment` al ser creado, controla la existencia del parámetro `noSampleRun` y de no existir inserta un objeto `experimentRun` en el atributo `sampleRun`. La macro utilizada para realizar este nuevo experimento tiene una pequeña diferencia con respecto al resto, y es la inclusión de la instrucción `generateVTKS` a la macro. Luego, se crean tantos `experimentRun` como indique `numberOfRuns`. Todos ellos se insertan en el atributo `runs`. Así es que cuando el objeto `multiMacroExperiment` recibe el mensaje `run`, este le envía otro mensaje `run` a cada uno de los elementos que tiene en su atributo `runs`, y cada `singleMacroExperiment` que lo recibe, le manda otro mensaje `run` a su `sampleRun` si tiene, y a todos los `experimentRun` que tiene en `runs`.

La clase `ExperimentRun` inicializa un atributo de clase `id` en 0 y el mensaje `getRunId` se encarga de obtenerlo, devolverlo e incrementarlo en 1. Es así que cada `experimentRun` al inicializarse, tiene su propio `runId` y `runName` asociado, siendo el nombre $runXX$ con XX un número de al menos dos dígitos que representa el runId. En caso de que el `experimentRun` represente el sample run, el `runId` será -1 y el `runName` “aSampleRun”. Al

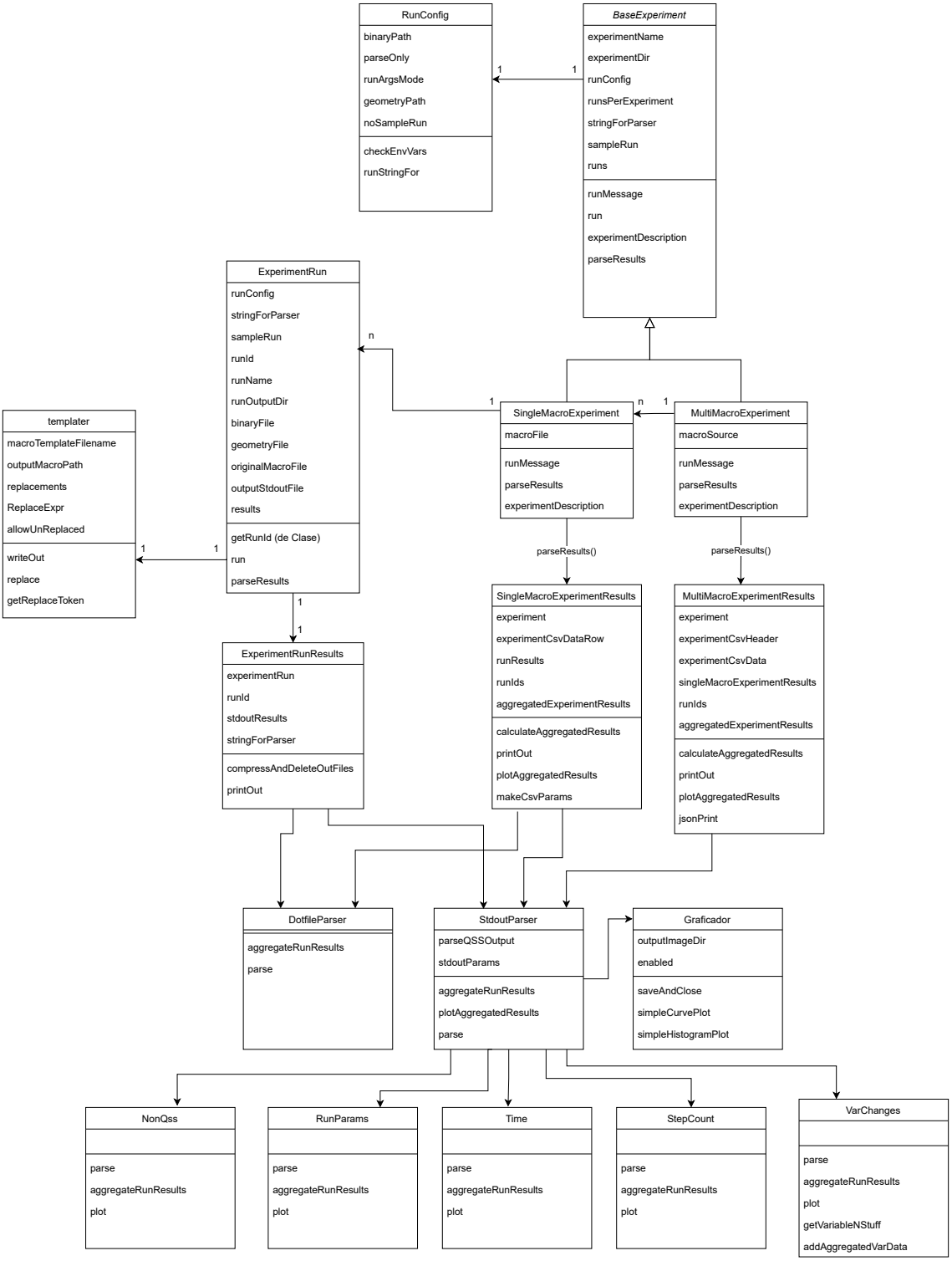

Fig. 4.6: Diagrama de clases del script experimentRunner

recibir el mensaje `run`, este objeto crea un directorio cuyo nombre será el atributo `runName`, deja allí la macro del experimento y ejecuta el binario. Para hacerlo, `runArgsMode` le indicará que parámetros recibe el binario. La salida de la ejecución se guardará en un archivo `run.out` .

Mensaje parseResults al multiMacroExperiment

Este mensaje crea un objeto `multiMacroExperimentResults` que contiene al `multiMacroExperiment`. Este nuevo objeto, le manda el mensaje `parseResults` a cada elemento del atributo `runs` de `multiMacroExperiment`, lo que produce que se cree un objeto `singleMacroExperimentResults` por cada `singleMacroExperiment` que haya en `runs` y que lo contenga. Cada `singleMacroExperimentResults` se guarda en el atributo `singleMacroExperimentRuns`. Cada `singleMacroExperimentResults` creado, le envía el mensaje `parseResults` a los `experimentRun` que tiene en el atributo `runs` de `singleMacroExperiment`, creándose así un objeto `experimentRunResults` por cada uno, y guardándose todos en el atributo `runResults`. Cada `experimentRunResults` se encargará de parsear la información del `run.out` y armará un json `stats.out` con la información recolectada. Luego, si el `run.out` pesa más de 2 MB, se comprime. Si no está activada la opción `noSampleRun` en el script, se crea una carpeta `PML` con un archivo como el que muestra la figura 4.7.

```
X Y Z LocalTime TrackLength
0.0000000000 0.0000000000 -90.0000000000 0.0000000000 0.0000000000
0.0000000000 0.0224434452 -75.0000000000 0.0500346866 15.0000000000
-0.0000265370 0.0225886139 -74.9327484065 0.0502590146 15.0672523358
-0.0000264834 0.0225903433 -74.9323924107 0.0502602021 15.0676083357
0.0001241688 0.0231770545 -74.8348595574 0.0505855435 15.1651445435
0.0001216947 0.0231931436 -74.8324470273 0.0505935911 15.1675571290
-0.1471191664 -0.0682205364 -72.4939039574 0.0584153035 17.5145245459
-0.1558560846 -0.0708362470 -72.3675650518 0.0588378228 17.6411970792
-0.2674558837 -0.0888692618 -71.0158780483 0.0633623155 18.9983101717
-0.2741101387 -0.0898179193 -70.9433139087 0.0636054004 19.0711865061
-0.3116082110 -0.0913863301 -70.6132109767 0.0647136030 19.4034554880
-0.4591881628 -0.0621372992 -69.5291282135 0.0683644203 20.4984216112
-0.5870372913 0.1617571641 -67.9633417489 0.0736149130 22.0971527725
```

Fig. 4.7: Ejemplo de un archivo PML

Por otro lado, `singleMacroExperimentResults` toma todos los `stat.out` que crearon los `experimentRunResults` y arma su propio `stats.out` con la información recolectada. Con todos estos datos, se arman los gráficos mencionados en 4.2 . Con la información considerada más relevante, se arma una fila para la tabla `relevant.stats` que creará

`multiMacroExperimentResults`. La información más relevante se puede ver en el ejemplo en la figura 4.3.

`multiMacroExperimentResults` toma todos los `stats.out` de los `singleMacroExperimentResults` y crea su propio `stats.out` que servirá para crear los gráficos de manera análoga a como los crea `singleMacroExperimentResults`. También, toma todas las filas que crearon los `singleMacroExperimentResults`, un encabezado que crea el mismo, y arma el archivo `RelevantStats.csv`. También se encarga de crear un archivo `experiment.data` indicando cuál es el mapeo entre corridas y macros, como lo muestra la figura 4.8 .

```
1 [0, 19]
  macro_0_exampleB4_withMagField_singleBeam.in_dQMin_0.01_dQRel_0.1_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0006_
2 [20, 39]
  macro_14_exampleB4_withMagField_singleBeam.in_dQMin_0.0001_dQRel_0.001_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0006_
3 [40, 59]
  macro_21_exampleB4_withMagField_singleBeam.in_dQMin_0.00001_dQRel_0.0001_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0006_
4 [60, 79]
  macro_28_exampleB4_withMagField_singleBeam.in_dQMin_0.000001_dQRel_0.00001_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0006
5 [80, 99]
  macro_35_exampleB4_withMagField_singleBeam.in_dQMin_0.0000001_dQRel_0.000001_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0
6 [100, 119]
  macro_7_exampleB4_withMagField_singleBeam.in_dQMin_0.001_dQRel_0.01_numberOfThreads_1_stepper_QSS2_trialProposedStepModifier_0.0006_
```

Fig. 4.8: Ejemplo de un archivo experiment.data

4.3. ExperimentCriticalInformation

El objetivo de este *script* será reunir la información más relevante de los experimentos y exponerla de manera ordenada y práctica, en una tabla y varios gráficos. Es así que, dado un conjunto de `PMLs` y de `CSVs` con la recolección de la información más importante de los experimentos, e indicando cuál es el experimento que se tomará como referencia para evaluar la calidad y la velocidad del resto, este *script* se encargará de crear una tabla resumen de todos los experimentos y una carpeta con información sobre la calidad de cada uno.

La tabla contendrá los siguientes headers:

- Example. Es el nombre del experimento. Por ejemplo: B2
- subExample. De corresponder, sirve para identificar las variaciones particulares de un experimento. Por ejemplo: el experimento B2 tiene dos variaciones, B2A, B2B, en este campo iría la A o la B según corresponda.
- stepperName. Es el nombre del integrador que se utilizó para el experimento. Por ejemplo, QSS2

- binary. Es el nombre del binario del experimento. Por ejemplo, exampleB2a.
- dQRel. Este campo representa el valor de dQRel y solo tiene sentido para QSS, en otro caso vendrá vacío.
- dQMin. Este campo representa el valor de dQMin y solo tiene sentido para QSS, en otro caso vendrá vacío.
- physics. El nombre de la física de Geant4 utilizada para este experimento. Esto representa todas las partículas que serán usadas en una simulación. Por ejemplo: FTFP_BERT.
- number Of Threads. representa la cantidad de hilos que se utilizaron en el experimento.
- intersections. Representa la cantidad de intersecciones geométricas que ocurrieron en el experimento. 2.2.1
- Total Steps. Representa la cantidad de *steps* que se crearon en este experimento.
- substeps per step. Representa la cantidad de *substeps* por *step* en promedio que ocurren. Este concepto solo tiene sentido para QSS, en otro caso, el valor en este campo será vacío.
- User time
- Real time
- System time
- Average Time per step.
- SpeedUp User time VS RefStepper
- SpeedUp Real time VS RefStepper
- SpeedUp System time VS RefStepper
- MSEx
- MSEy

- MSEz
- MSETrackLength .
- Indice de Desempeno M en la coordenada x
- Indice de Desempeno M en la coordenada y
- Indice de Desempeno M en la coordenada z
- Indice de Desempeno MSE en la coordenada x
- Indice de Desempeno MSE en la coordenada y
- Indice de Desempeno MSE en la coordenada z

Se creará una carpeta comparando la trayectoria de cada experimento con la trayectoria del experimento de referencia. El usuario podrá elegir el nombre de cada PML, y la carpeta se llamará `nombrePMLVsNombrePMLReferencia`. En cada una de estas carpetas se encontrarán los siguientes gráficos:

- Trayectoria del PML de referencia
- Trayectoria del otro PML
- Posición en x a través del tiempo de ambos PMLs
- Posición en y a través del tiempo de ambos PMLs
- Posición en z a través del tiempo de ambos PMLs
- TrackLength a través del tiempo de ambos PMLs
- Posición en x interpolada a través del tiempo de ambos PMLs
- Posición en y interpolada a través del tiempo de ambos PMLs
- Posición en z interpolada a través del tiempo de ambos PMLs
- TrackLength interpolado a través del tiempo de ambos PMLs
- Error en x a través del tiempo de ambos PMLs
- Error en y a través del tiempo de ambos PMLs
- Error en z a través del tiempo de ambos PMLs

4.3.1. Parámetros de entrada

Parámetro	Obligatorio	Descripción
`tabNam`	Sí	Nombre que se le dará a la tabla de información general de los experimentos.
`outputFormat`	No	Formato en que se puede exportar la tabla. Por el momento solo se acepta en excell, pero en un futuro se podrán ampliar los formatos permitidos.
`physics`	Sí	Físicas que se usaron en estos experimentos.
`csvRef`	No	`RelevantStats.csv` del experimento de referencia. El nombre RelevantStats puede haber sido cambiado, pero el contenido debe ser el mismo que devuelve `experimentRunner.py`.
`pmlRef`	No	Archivo PML del experimento de referencia.
`namePmlRef`	No	Nombre representativo para el PML de referencia. Este nombre aparecerá en las carpetas creadas, y en los gráficos para representar al PML.
`exNameRef`	No	Nombre del experimento de referencia.
`subExNameRef`	No	Nombre de la variación del experimento de referencia.
`stepperNameRef`	No	Nombre del integrador utilizado en el PML de referencia.
`csvsCom`	Sí	Lista de `RelevantStats.csv` del resto de los experimentos. El nombre RelevantStats de cada uno puede haber sido cambiado, pero el contenido debe ser el mismo que devuelve `experimentRunner.py`.
`pmlsCom`	Sí	Lista de archivos PML del resto de los experimentos.
`namesPmlsCom`	Sí	Lista de nombres representativos para el resto de los PMLs. Este nombre aparecerá en las carpetas creadas, y en los gráficos para representar al PML.
`exNamesCom`	Sí	Lista de nombres del resto de los experimentos.
`subExNamesCom`	Sí	Lista de nombres de las variaciones del resto de los experimentos.
`stepperNamesCom`	Sí	Lista de nombres de los integradores utilizados en el resto de los PMLs.

Algoritmo 4.3: Ejemplo de uso

```
python3 ExperimentCriticalInformation.py \
 --tabNam ResultadosCern_AmsEcal \
 --outputFormat EXCEL \
 --physics FTFP_BERT \
 --csvRef AmsEcal/RelevantStats_AmsEcal_DopriT.csv \
 --exNameRef AmsEcal --subExNameRef N/A --stepperNameRef DopriT \
 --pmlRef EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_TempDopri.txt \
 --namePmlRef DopriT \
 --csvsCom AmsEcal/RelevantStats_AmsEcal_DopriNoT.csv \
 AmsEcal/RelevantStats_AmsEcal_RK4.csv \
 AmsEcal/RelevantStats_AmsEcal_QSS2.csv \
 --exNamesCom AmsEcal AmsEcal AmsEcal --subExNamesCom N/A N/A N/A \
 --stepperNamesCom DopriNoT RK4 QSS2 \
 --pmlsCom EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal\_DopriNoT.txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_RK4.txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.01_dQRel_0.1.txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.0001_dQRel_0.001.txt
     \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.00001_dQRel_0.0001.
     txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.000001_dQRel_0.00001.
     txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.0000001_dQRel_0
     .000001.txt \
 EjemploDeRutaPMLSCERN/AmsEcal/PML_AmsEcal_Qss_dQMin_0.001_dQRel_0.01.txt \
 --namesPmlsCom DopriNoT RK4 \
 Qss_dQMin_0.01_dQRel_0.1 Qss_dQMin_0.0001_dQRel_0.001 \
 Qss_dQMin_0.00001_dQRel_0.0001 Qss_dQMin_0.000001_dQRel_0.00001\
 Qss_dQMin_0.0000001_dQRel_0.000001 Qss_dQMin_0.001_dQRel_0.01
```

4.3.2. Explicación

Las listas de parámetros `csvsCom`, `exNamesCom`, `subExNamesCom` y `stepperNamesCom` deben tener la misma longitud entre sí. De igual manera ocurre con `pmlsCom` y `namesPmlsCom` entre si. Caso contrario, se tirará un error indicando la situación. Luego de esta validación, se chequeará la existencia de un `pmlRef`. En caso de no existir, el *script* no presentará

ninguna información sobre la calidad de los experimentos, y simplemente fusionará los `relevantStats` en una tabla común. Esta fusión completará ciertas columnas, como por ejemplo `Example` o `subExample`, con la información proporcionada por los parámetros correspondientes, además de calcular los speed-ups de los distintos tiempos teniendo como referencia `csvRef`.

En caso de existir `pmlRef`, además del comportamiento antes descripto, se añade un análisis de la calidad de los experimentos en referencia a `pmlRef`. Para esto, por cada elemento de `pmlsCom`, se crea una nueva serie temporal con `pmlRef` cuyo contenido es el resultado de la unión de las series temporales de ambos PML. Luego se realiza una interpolación de ambos PMLs con la nueva serie temporal y con esa información calculan los MSEs de las posiciones en x, y, z y el trackLength. Toda la información de cada experimento ubicada en los PMLs y su versión interpolada se guardará en un objeto de la clase `QualityExperimentInformation` 4.9, existiendo uno de estos por cada `pmlCom`, y uno por el `pmlRef`. Se guardan todos los MSEs para luego colocarlos en la tabla, y también se realizarán los gráficos comunes e interpolados, enviándole el mensaje `plot` al objeto `PMLGraphics`. Este objeto a su vez, también realizará con la misma biblioteca un gráfico 3D de la trayectoria de la partícula, y calculará con las posiciones interpoladas de cada experimento, comparadas con las del experimento de referencia los errores en cada punto de cada coordenada, produciendo un gráfico de errores a través del tiempo por experimento. Toda esta información, se guardará en un directorio cuyo nombre es la concatenación del elemento actual de `namesPmlsCom` con el string "VS" y `namePmlRef`. Además, la función `calcularIndiceDesempeno` (que implementa la Ec. 6.3 explicada más adelante) calculará los índices de desempeño η_{max} y η_{MSE} para cada coordenada, y los reflejará en la tabla (utilizando como entrada el error absoluto máximo o el MSE).

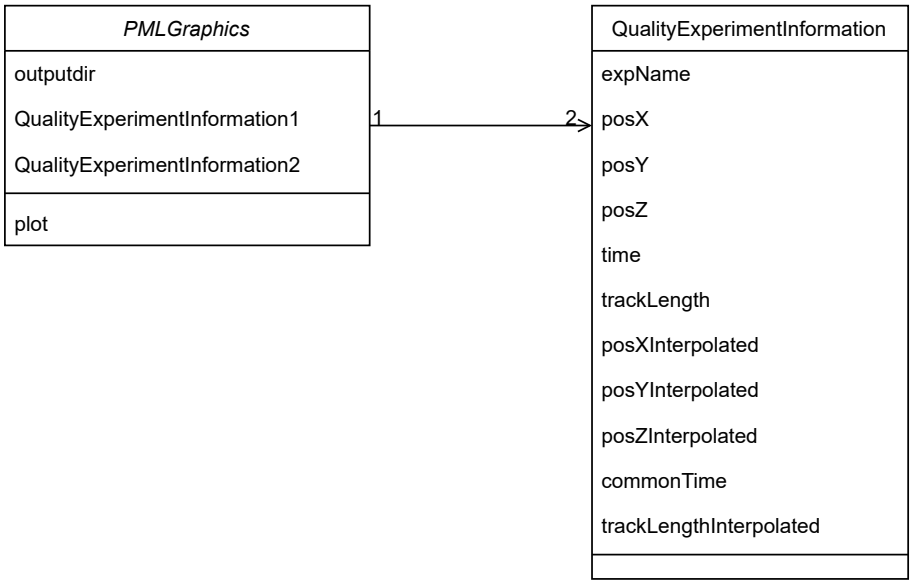

Fig. 4.9: Diagrama de clases del script ExperimentCriticalInformation

5. MARCO EXPERIMENTAL

Como se dijo en la introducción, el objetivo de esta tesis es crear una herramienta de comparación de integradores, y utilizarla para ampliar la experimentación de QSS en Geant4. El primer objetivo está cumplido y explicado en la sección 4. Para cumplir el segundo, utilizaremos algunos de los tests que se utilizan en el CERN para corroborar que al introducir nuevas funcionalidades o mejorar existentes en un release, no se introducen comportamientos no deseados. Estos tests se pueden ver en la siguiente página de Geant4 [24]. A continuación, se detallarán los tests a ejecutarse y la metodología de experimentación, seguido por las especificaciones técnicas de la plataforma de hardware empleada.

5.1. Casos de estudio

Los tests del CERN para Geant4, se dividen en tres categorías principales.

- Basics
- Extended
- Advanced

Cabe destacar que no todos los tests existentes son aplicables a QSS, puesto que algunos por su concepción, no utilizan campos magnéticos, que son necesarios para que se utilice la versión actual del stepper de QSS para Geant4 (ya que por el momento se basa en la Ecuación de Lorentz) o cualquier otro integrador. Aquellos que no lo sean, no serán tomados en cuenta.

5.1.1. Ejemplos tipo Basics

Estos tests representan los casos de uso más básicos que un usuario de una aplicación que use Geant4 puede utilizar. Para cada uno de ellos se dará una breve explicación, y se mostrará la geometría utilizada.

Example B2

Este ejemplo simula un experimento simplificado de blanco fijo. La geometría consta de un blanco seguido por seis cámaras de tamaños crecientes. Estas cámaras se encuentran en una región llamada "tracker region". El material del blanco y las cámaras pueden ser modificados vía macro 2.2.1. La figura 5.1 representa la geometría del *example*.

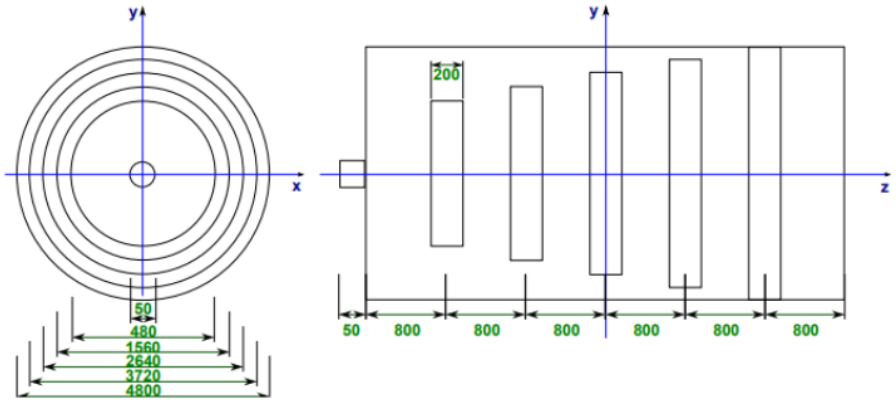

Fig. 5.1: Geometría del ejemplo B2

Example B4

Este ejemplo simula una configuración sencilla de un calorímetro de muestreo. El calorímetro es una caja formada por un número determinado de capas. Una capa consta de una placa absorbente y de una abertura de detección. La capa se repite. Cuatro aspectos definen la geometría del calorímetro.

- El grosor de la placa absorbente
- El grosor de la abertura de derección
- La cantidad de capas
- El tamaño transversal del calorímetro

La figura 5.2 representa la geometría del *example*.

Example B5

El ejemplo B5 implementa un espectrómetro de doble brazo con cámaras de hilos, hodoscopios y calorímetros. El espectrómetro consta de dos brazos detectores. Por uno

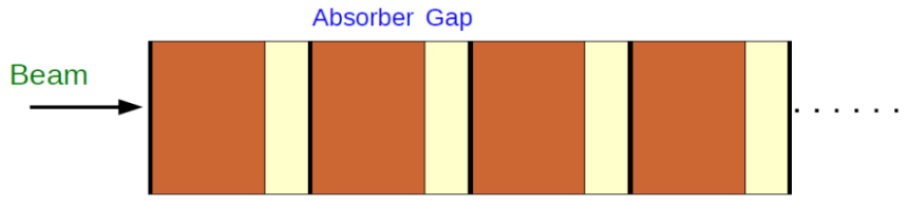

Fig. 5.2: Geometría del ejemplo B4

de ellos ingresa la partícula, y es desviado hacia el otro brazo por un campo magnético centrado en el punto de giro del espectrómetro.

El primer brazo consiste en una caja llena de aire que contiene:

- 1 hodoscopio (15 tiras verticales de centelleadores de plástico)
- 1 cámara de deriva (5 capas horizontales de gas Argón con un "hilo virtual" en el centro de cada capa)

El segundo brazo consiste en otra caja llena de aire que contiene:

- 1 hodoscopio (25 tiras verticales de centelleadores de plástico)
- 1 cámara de deriva (5 capas horizontales de gas Argón con un "hilo virtual" en el centro de cada capa)
- 1 calorímetro electromagnético (caja subdividida a lo largo de los ejes x,y,z en celdas de yoduro de cesio)
- 1 calorímetro hadrónico (caja subdividida a lo largo de los ejes x,y,z en celdas de plomo, con una capa de centelleador plástico colocada en el centro de cada celda)

La región del campo electromagnético contiene un cilindro lleno de aire con el campo dentro de él.

La figura 5.3 representa la geometría del example.

5.1.2. Ejemplos tipo Extended

Estos tests representan casos de usos específicos de algunos detectores reales. Los *extended* se dividen en más de veinte tipos de experimentos, de los cuales solo nos interesan

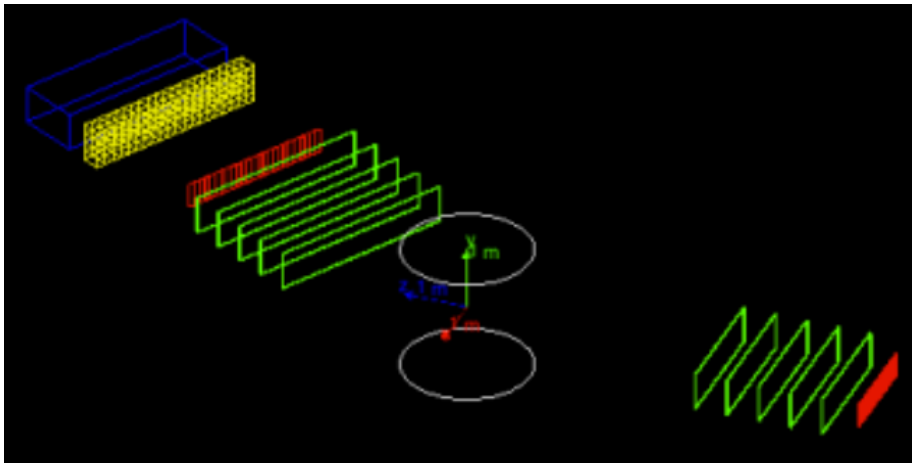

Fig. 5.3: Geometría del ejemplo B5

los "Field". Hay seis experimentos *fields* de los cuáles solo tomaremos el primero y el tercero, que son los que tienen campos magnéticos. Todos estos tienen la misma geometría, un cilindro que contiene un absorbente al cuál se le puede cambiar el material, el grosor y el tamaño transversal. A éste se le puede agregar un campo magnético. La figura 5.4 muestra la geometría del example.

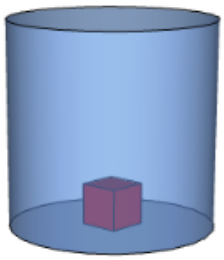

Fig. 5.4: Geometría de ejemplos extended

5.1.3. Ejemplos tipo Advanced

Corresponden a aplicaciones reales que utilizan Geant4. Hay una amplia variedad, y muchos requieren la instalación de herramientas aparte de Geant4, por lo que por simplicidad, tomaremos a AmsEcal como representante, ya que no requiere ningún componente extra.

5.1.4. Aplicación FulSimLight para el detector ATLAS

Sobre FSL se habló en la sección 2.2.2. Se utilizará como prueba una macro de ejemplo que viene con el software al descargarse el repositorio.

5.2. Metodología de experimentación

En todos los casos, no existe una solución analítica, por lo que el objetivo será comparar QSS contra el integrador default de Geant4, DOPRI, y el anterior default, RK4. Para QSS siempre habrá un barrido de parámetros para poder deducir cuál es la mejor configuración en cada caso. Este barrido siempre será igual, y contendrá las siguientes configuraciones:

- dQRel 0.1 y dQMin 0.01
- dQRel 0.01 y dQMin 0.001
- dQRel 0.001 y dQMin 0.0001
- dQRel 0.0001 y dQMin 0.00001
- dQRel 0.00001 y dQMin 0.000001
- dQRel 0.000001 y dQMin 0.0000001

La comparación será bajo los siguientes criterios.

- Criterio Temporal: se calcularán los tiempos de ejecución de cada test con cada posible integrador, y se los comparará. Para tener mayor información, y poder explicar los tiempos observados, se incluirán otras características en la comparación, como por ejemplo el número de *steps* producido, la cantidad de intersecciones encontradas, etc.
- Criterio de Calidad: se tomará la simulación obtenida con DOPRI como el objetivo, y se medirá el error obtenido con QSS. Para esto, se tomará una serie temporal por cada integrador, y se realizará una comparación entre éstas. Para ello se utilizarán distintas métricas que serán explicadas en el próximo capítulo.

Se realizarán veinte corridas de cada experimento, y los resultados contendrán el promedio de las mismas.

5.3. Especificaciones técnicas

Para concluir el capítulo, describiremos la plataforma de hardware empleada durante la fase de experimentación de nuestro trabajo.

Todas las simulaciones realizadas en la presente tesis se ejecutaron en máquinas en el CERN. Éstas son un Intel S2600KP con sistema operativos CentOS 7.9.2009 con 64 CPUs Intel(R) Xeon(R) CPU E5-2683 v4, con una frecuencia mínima de 1200MHz y máxima de 3000MHz. Geant4 se compiló en su versión 11, con gcc 9.2.0 y utilizando el flag O2, habilitado por defecto en compilaciones de Geant4 en modo release.

6. RESULTADOS DE EXPERIMENTACIÓN Y ANÁLISIS

En el presente capítulo mostraremos los resultados obtenidos para los casos de estudio que explicamos en el capítulo 5. Para esto, primero explicaremos algunas mediciones que se utilizaron para analizar los resultados. Posteriormente mostraremos estos casos, y finalmente tomaremos uno de los experimentos como ejemplo para analizar en profundidad, y mostrar las capacidades de la herramienta creada y explicada en el capítulo 4. Cabe aclarar que todos los gráficos y tablas presentes en este capítulo fueron generadas por el sistema automático de comparación de integradores.

6.1. Mediciones a utilizar

A continuación se presentarán 3 mediciones que se utilizarán para analizar los resultados de la experimentación.

6.1.1. MSE

El MSE es el error cuadrático medio (en inglés *Mean squared Error*), y evalúa la precisión entre una estimación y aquello que se busca estimar. La fórmula 6.1 representa esta medición.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.1}$$

donde:

- n es el número total de observaciones.
- y_i representa los valores reales u observados (En este caso serán los correspondientes a DOPRI).
- $\hat{y}_i$ representa los valores predichos por el modelo.

Se considerará 10^{-3} como un MSE aceptable.

6.1.2. SpeedUp

El *speedUp* representa la relación porcentual entre la performance de tiempo de ejecución de un *stepper* en relación a otro *stepper* de referencia. Un valor positivo representará una mejora de tiempos mientras que uno negativo indicará un aumento del tiempo de ejecución. Si llamamos $stepperTime$ al tiempo de ejecución de un experimento con un *stepper*, $refStepperTime$ al tiempo de ejecución del mismo experimento con un *stepper* de referencia, el speedUp del *stepper* vs. el *stepper* de referencia está dado por la fórmula 6.2.

$$SPEEDUP = 1 - \frac{stepperTime}{refStepperTime} \tag{6.2}$$

6.1.3. Indice de desempeño

En la tesis [23], se propuso una medición llamada "Índice de desempeño", que crea una relación entre el error absoluto y el tiempo de simulación. Se tomará de base la idea de este índice, para generalizarlo respecto al error, Es decir, en esta tesis se llamará "Índice de desempeño" a la relación que viene dada por la fórmula 6.3, donde t_{sim} es el tiempo real (*wall clock*) que toma una simulación y E el error en alguna de las coordenadas. Esta medición se puede aplicar en el resto de las coordenadas de forma análoga.

$$\theta = \frac{1}{t_{sim} \cdot E} \tag{6.3}$$

Se utilizarán dos índices de desempeño, θ_{MSE} que utiliza $E = MSE$ y θ_{MAX} que utiliza $E = max(|E|)$ (error absoluto máximo).

6.2. Resultados para ejemplos tipo Basics

6.2.1. Resultados Example B2

En la tabla 6.1 se observan los resultados para el Example B2. Si bien los MSE de QSS son aceptables, los tiempos son en todos los casos al menos un 22 % más lento que el experimento de referencia realizado con DOPRI. Además, se puede observar que RK4 es ligeramente más rápido que DOPRI, y órdenes de magnitud más pequeño en términos de calidad que QSS con las cotas de error más restrictivas.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			34,625		0	0	0	0
DopriNoT			33,919	2,04 %	0	0	0	0
RK4			33,954	1,94 %	0	0	1,820E-22	4,545E-22
QSS2	1,00E-01	1,00E-02	43,483	-25,58 %	3,030E-02	3,053E+00	6,025E-03	2,296E-08
QSS2	1,00E-02	1,00E-03	42,567	-22,94 %	3,033E-02	8,663E-02	6,011E-03	2,296E-08
QSS2	1,00E-03	1,00E-04	42,728	-23,40 %	3,034E-02	3,298E-03	6,009E-03	2,296E-08
QSS2	1,00E-04	1,00E-05	50,058	-44,57 %	3,034E-02	2,958E-03	4,995E-03	2,296E-08
QSS2	1,00E-05	1,00E-06	61,834	-78,58 %	1,077E-06	8,835E-06	3,297E-05	6,497E-12
QSS2	1,00E-06	1,00E-07	108,282	-212,73 %	2,700E-08	2,095E-07	6,829E-07	2,001E-13

Fig. 6.1: Tiempos y MSE Experimento B2A.

6.2.2. Resultados Example B4

En la tabla 6.2 se observan los resultados para el Example B4. Los MSE de QSS presentan una gran similitud con los de DOPRI, con tiempos muy similares, aunque siempre más lentos. Por otro lado, RK4 también tiene un tiempo similar, y cuenta con MSEs varios órdenes de magnitud más pequeños.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			5,249		0	0	0	0
DopriNoT			5,252	-0,05 %	0	0	0	0
RK4			5,259	-0,19 %	0	2,414E-21	0	0
QSS2	1,00E-01	1,00E-02	5,250	-0,01 %	4,459E-06	4,450E-04	3,797E-06	3,600E-05
QSS2	1,00E-02	1,00E-03	5,296	-0,89 %	4,526E-06	2,955E-05	4,672E-06	3,606E-05
QSS2	1,00E-03	1,00E-04	5,260	-0,19 %	1,123E-06	1,328E-06	1,127E-07	1,053E-06
QSS2	1,00E-04	1,00E-05	5,255	-0,10 %	5,152E-08	1,413E-07	1,571E-08	5,923E-09
QSS2	1,00E-05	1,00E-06	5,296	-0,88 %	5,251E-10	1,338E-09	8,486E-10	1,893E-09
QSS2	1,00E-06	1,00E-07	5,340	-1,72 %	7,932E-12	1,042E-11	2,205E-11	5,825E-11

Fig. 6.2: Tiempos y MSE Experimento B4C.

6.2.3. Resultados Example B5

En la tabla 6.3 se observan los resultados para el Example B5. Algunas configuraciones de QSS tienen mejores tiempos que DOPRI. Destaca la configuración dQRel 1E-05, dQMin 1E-06, ya que tiene MSEs de al menos 1E-05 en todas las posiciones con incremento de velocidad del 5,28 %. RK4 sigue teniendo similitud mucho más marcada con DOPRI que QSS, pero los tiempos en este caso son mayores.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			52,201		0	0	0	0
DopriNoT			52,305	-0,20 %	0	0	0	0
RK4			53,420	-2,33 %	9,308E-19	0	1,320E-19	3,938E-20
QSS2	1,00E-01	1,00E-02	41,154	21,16 %	3,335E+01	2,668E-02	1,823E+02	5,398E-13
QSS2	1,00E-02	1,00E-03	52,514	-0,60 %	1,951E+00	6,278E-03	1,288E+01	1,786E-13
QSS2	1,00E-03	1,00E-04	51,118	2,08 %	4,689E-02	3,326E-06	1,312E-01	7,825E-15
QSS2	1,00E-04	1,00E-05	53,392	-2,28 %	7,030E-04	2,929E-06	1,724E-03	1,143E-14
QSS2	1,00E-05	1,00E-06	49,445	5,28 %	7,636E-06	3,697E-12	1,811E-05	7,833E-17
QSS2	1,00E-06	1,00E-07	52,418	-0,42 %	7,825E-08	9,224E-14	1,877E-07	8,895E-17

Fig. 6.3: Tiempos y MSE Experimento B5.

6.3. Resultados para ejemplos tipo Extended

6.3.1. Resultados field01

En la tabla 6.4 se observan los resultados para el ejemplo F01. La primer precisión aceptable de QSS recién se encuentra en dQRel 1E-04, dQMin 1E-05 . con un tiempo un 2,48 % mayor. RK4 en este caso es ligeramente más lento que DOPRI y nuevamente más parecido.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			4,476		0	0	0	0
DopriNoT			4,477	-0,04 %	0	0	0	0
RK4			4,478	-0,06 %	1,052E-12	1,233E-12	3,064E-12	0
QSS2	1,00E-01	1,00E-02	4,599	-2,76 %	9,158E+01	9,440E+01	1,723E+02	0
QSS2	1,00E-02	1,00E-03	4,586	-2,47 %	2,856E+00	1,254E+00	3,235E-01	0
QSS2	1,00E-03	1,00E-04	4,599	-2,75 %	8,159E-02	3,010E-02	1,473E-04	0
QSS2	1,00E-04	1,00E-05	4,586	-2,48 %	1,257E-03	4,638E-04	6,679E-08	0
QSS2	1,00E-05	1,00E-06	4,628	-3,41 %	1,506E-05	5,589E-06	6,253E-11	0
QSS2	1,00E-06	1,00E-07	4,679	-4,54 %	1,666E-07	6,237E-08	1,485E-12	0

Fig. 6.4: Tiempos y MSE Experimento F01.

6.3.2. Resultados field03

En la tabla 6.5 se observan los resultados para el ejemplo F03. En este ejemplo, QSS resulta muy desfavorecido. Salvo en dQRel 1E-06, dQMin 1E-07, en ninguna otra configuración tiene una precisión aceptable, y en la que sí lo tiene, es un casi una 450 % más lento. Sin embargo, RK4 tampoco tiene una gran similitud, sobre todo observando el MSE de la posición x, aunque cuente con un tiempo ligeramente mejor.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			6,961		0	0	0	0
DopriNoT			8,248	-18,49 %	0	0	0	0
RK4			6,895	0,95 %	2,083E+03	1,942E-01	9,342E-01	9,947E-11
QSS2	1,00E-01	1,00E-02	4,997	28,22 %	8,910E+05	1,876E+03	8,004E+02	9,947E-11
QSS2	1,00E-02	1,00E-03	10,645	-52,92 %	1,889E+05	2,007E+02	2,684E+02	9,947E-11
QSS2	1,00E-03	1,00E-04	9,239	-32,72 %	7,441E+04	1,062E+01	1,102E+01	0
QSS2	1,00E-04	1,00E-05	10,359	-48,81 %	5,194E+03	4,030E-01	1,202E+00	9,947E-11
QSS2	1,00E-05	1,00E-06	17,207	-147,19 %	1,640E+00	7,201E-04	1,443E-03	0
QSS2	1,00E-06	1,00E-07	38,226	-449,14 %	1,885E-02	4,883E-06	1,040E-05	0

Fig. 6.5: Tiempos y MSE Experimento F03.

6.4. Resultados para ejemplos tipo Advanced

6.4.1. Resultados AmsEcal

En la tabla 6.6 se observan los resultados para el experimento AmsEcal. Los tiempos de QSS son siempre al menos un 45 % más lentos, mientras que RK4 no solo es ligeramente más rápido que DOPRI, sino también mucho más parecido que QSS.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			157,520		0	0	0	0
DopriNoT			155,961	0,99 %	0	0	0	0
RK4			156,240	0,81 %	6,793E-18	4,546E-20	2,747E-20	6,862E-18
QSS2	1,00E-01	1,00E-02	229,522	-45,71 %	1,464E-03	2,331E-01	6,547E-03	3,327E-08
QSS2	1,00E-02	1,00E-03	232,247	-47,44 %	1,497E-03	2,354E-01	6,776E-03	2,103E-08
QSS2	1,00E-03	1,00E-04	237,173	-50,57 %	3,570E-07	4,773E-06	4,118E-05	2,922E-10
QSS2	1,00E-04	1,00E-05	242,314	-53,83 %	3,214E-07	4,589E-06	4,091E-05	3,204E-10
QSS2	1,00E-05	1,00E-06	263,223	-67,10 %	3,228E-07	4,723E-06	4,165E-05	3,334E-10
QSS2	1,00E-06	1,00E-07	314,474	-99,64 %	3,337E-07	4,894E-06	4,330E-05	3,465E-10

Fig. 6.6: Tiempos y MSE Experimento AmsEcal.

6.5. Resultados FSL

En la tabla 6.7 se observan los resultados para el experimento FSL. QSS tiene una mala calidad hasta dQRel 1E-05, dQMin 1E-06, con tiempo mayores en hasta un 4,40 %. Por otro lado RK4 si bien es ligeramente más lento que DOPRI, es mucho más parecido que QSS.

6.6. Análisis en profundidad caso field03 (aka F03)

Dividiremos este análisis en dos. Mostraremos los resultados generales orientados a QSS por un lado, tratando de explicar la diferencia de tiempos observada, mientras que por otro lado explicaremos la baja precisión de RK4 observados en la tabla 6.5.

Stepper Name	dQrel	dQmin	Real Time(seg)	SpeedUp Real Time vs DopriT	MSE x	MSE y	MSE z	MSE trackLength
DopriT			3746,566		0	0	0	0
DopriNoT			3781,672	-0,94 %	1,971E+05	1,461E+05	5,377E+05	3,964E+05
RK4			3778,891	-0,86 %	7,970E+04	1,500E+05	6,580E+05	1,908E+01
QSS2	1,00E-01	1,00E-02	3776,481	-0,80 %	4,804E+02	2,464E+03	6,131E+02	5,576E+03
QSS2	1,00E-02	1,00E-03	3775,574	-0,77 %	4,654E+03	1,244E+03	5,615E+02	1,698E+03
QSS2	1,00E-03	1,00E-04	3766,781	-0,54 %	1,854E+03	1,790E+03	1,263E+01	3,486E+00
QSS2	1,00E-04	1,00E-05	3796,805	-1,34 %	1,104E+02	1,835E+03	8,877E+00	9,231E+02
QSS2	1,00E-05	1,00E-06	3840,472	-2,51 %	8,959E-07	8,944E-08	4,892E-08	1,897E-07
QSS2	1,00E-06	1,00E-07	3911,562	-4,40 %	5,727E-07	2,089E-07	2,126E-07	1,162E-07

Fig. 6.7: Tiempos y MSE Experimento FSL.

6.6.1. Resultados generales y Diferencia de tiempos

En la tabla 6.5 se puede observar cómo las precisiones de QSS son de muy mala calidad hasta llegar a dQRel 1E-06, dQMin 1E-07. Tomando por ejemplo dQRel 1E-01, dQMin 1E-02, se denota cómo tanto el MSE_x, como MSE_y, como MSE_z tienen valores muy altos. Esto se condice con los gráficos 6.8 , 6.9 y 6.10 respectivamente.

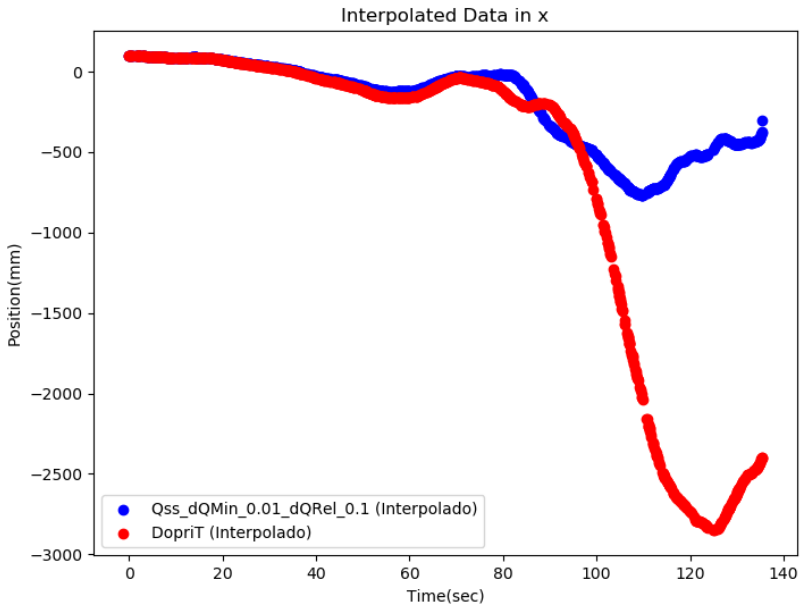

Fig. 6.8: Posición en X a través del tiempo, Experimento F03, QSS VS DOPRI. dQRel 1E-01, dQMin 1E-02.

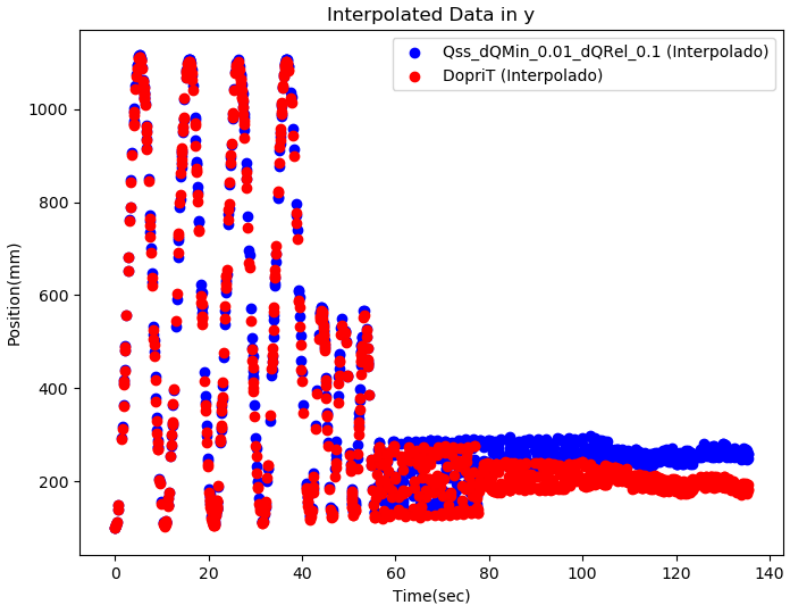

Fig. 6.9: Posición en Y a través del tiempo, Experimento F03, QSS VS DOPRI. dQRel 1E-01, dQMin 1E-02

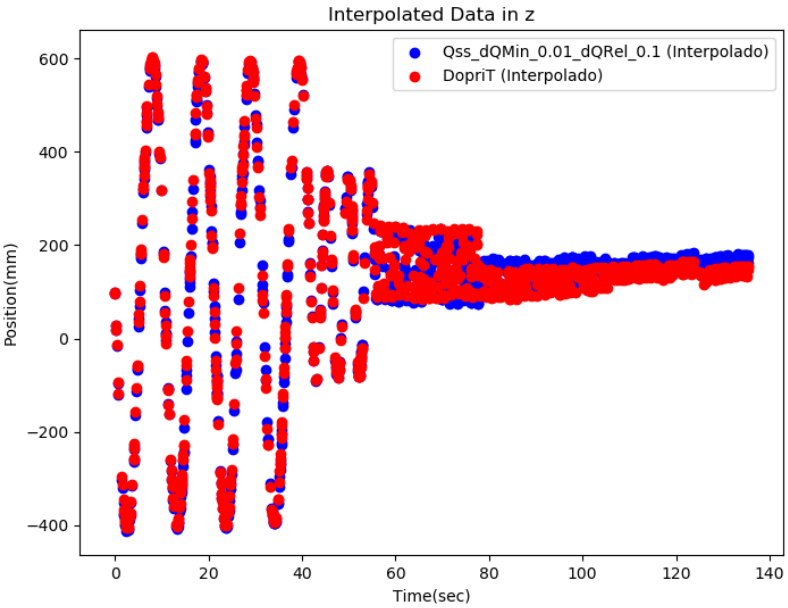

Fig. 6.10: Posición en Z a través del tiempo,Experimento F03, QSS VS DOPRI. dQRel 1E-01, dQMin 1E-02

Si quisiéramos comparar el recorrido completo de la partícula a través del tiempo, podríamos hacerlo viendo las figuras 6.11 y 6.12, las cuáles claramente son muy diferentes.

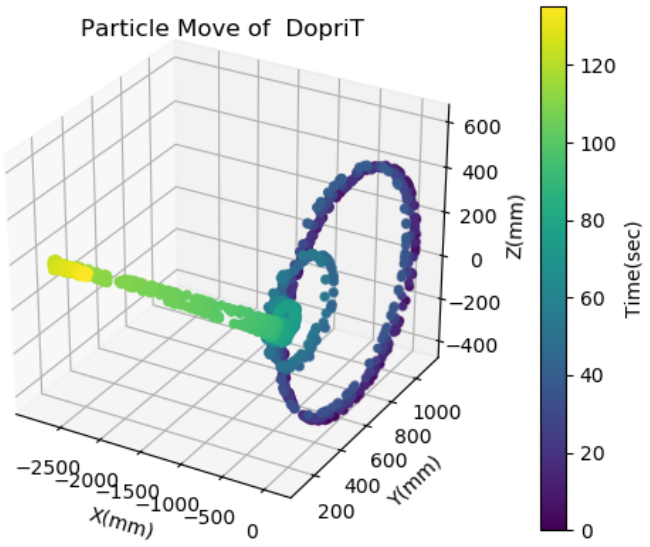

Fig. 6.11: Posición de la partícula a través del tiempo, DOPRI

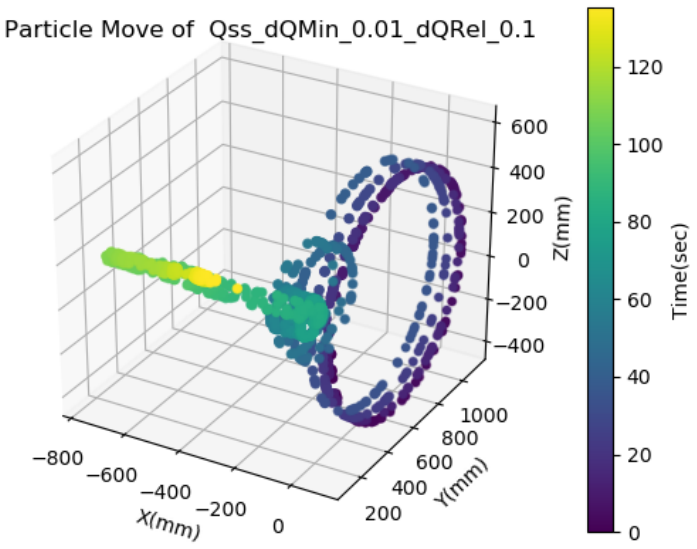

Fig. 6.12: Posición de la partícula a través del tiempo, QSS dQRel 1E-01, dQMin 1E-02

En cambio, si comparamos DOPRI con una mejor precisión de QSS como dQRel 1E-06, dQMin 1E-07, podemos ver cómo cambia la posición en X, Y, Z a través del tiempo, no siendo distinguible ninguna diferencia en las figuras 6.13 , 6.14 y 6.15 respectivamente.

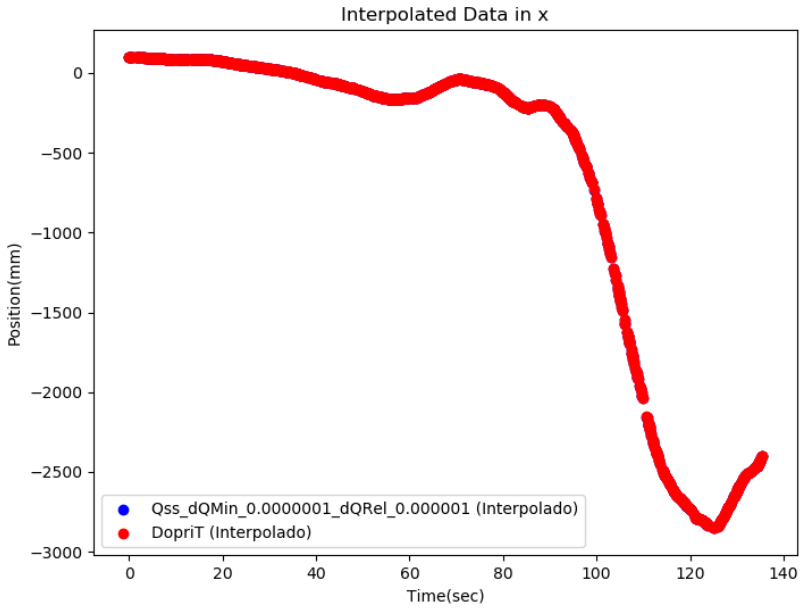

Fig. 6.13: Posición en X a través del tiempo, Experimento F03, QSS VS DOPRI. dQRel 1E-06, dQMin 1E-07.

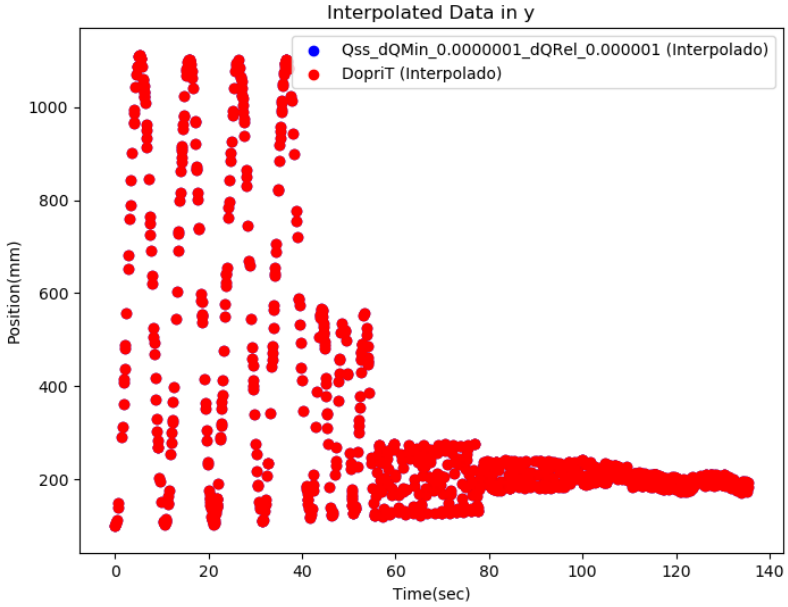

Fig. 6.14: Posición en Y a través del tiempo, Experimento F03, QSS VS DOPRI. dQRel 1E-06, dQMin 1E-07

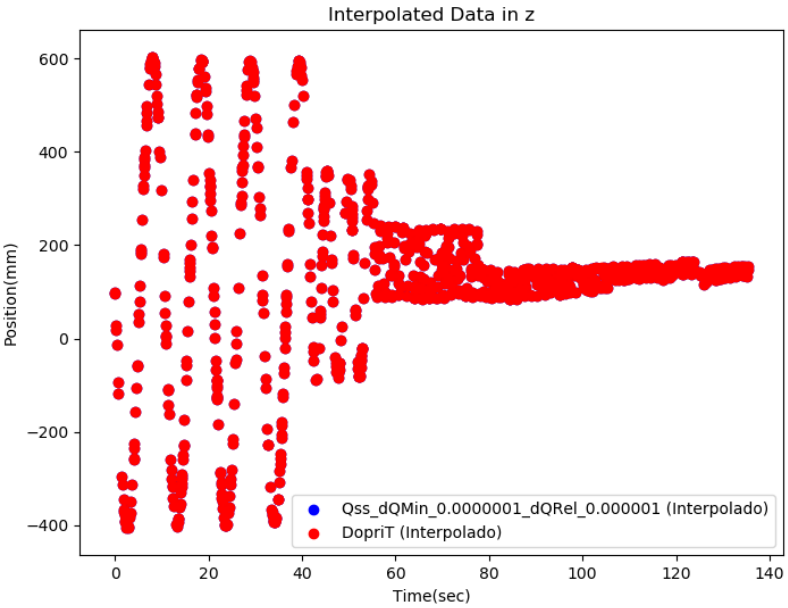

Fig. 6.15: Posición en Z a través del tiempo,Experimento F03, QSS VS DOPRI. dQRel 1E-06, dQMin 1E-07

También, si comparamos la trayectoria de QSS con dQRel 1E-06, dQMin 1E-07 en la figura 6.16 con la trayectoria de DOPRI en la figura 6.11, podemos notar que son muy parecidas.

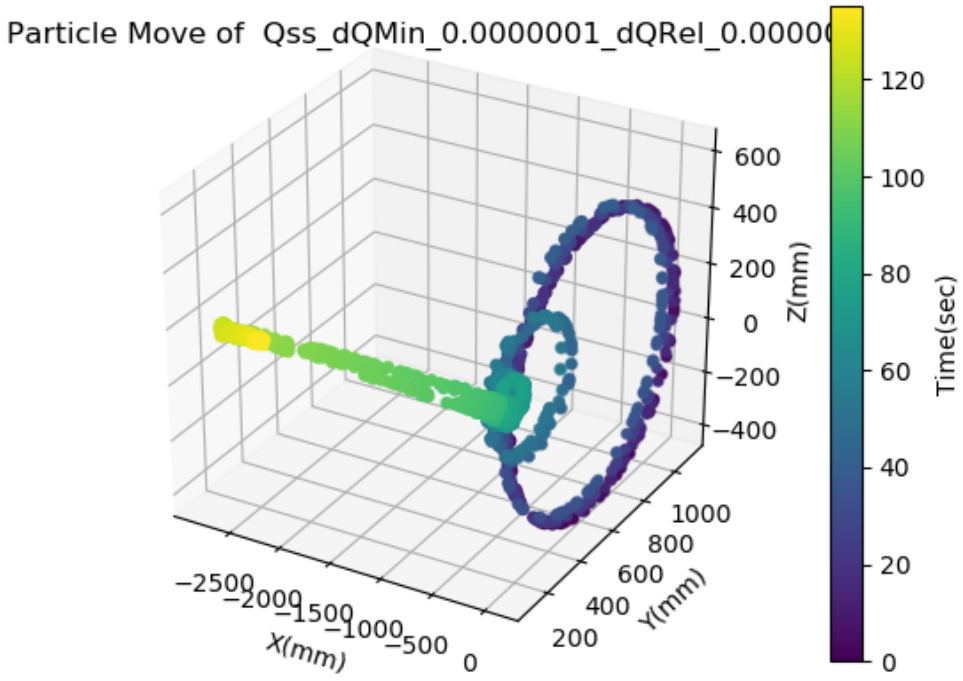

Fig. 6.16: Posición de la partícula a través del tiempo, QSS dQRel 1E-06, dQMin 1E-07

Sin embargo, cuando encontramos una configuración de QSS con una buena precisión, esta es casi un 450 % más lenta.

En el gráfico 6.17 se muestra que en los experimentos realizados, los tiempos obtenidos parecieran ser constantes, por lo que descartamos que haya algún problema de la muestra. Por otro lado, en el gráfico 6.18 podemos ver cómo en ningún experimento hubo cruces geométricos, que es la principal aplicación de los métodos QSS, y una posible explicación de la diferencia de tiempos. Podemos observar en el gráfico 6.19 la cantidad de *steps* por experimento, en el 6.20 la cantidad de *substeps* por experimento, y en 6.21 la cantidad de *substeps* por *steps* que hay en cada experimento. En ellos se muestra cómo a medida que crece dQMin y dQRel, la cantidad de *substeps* es significativamente mayor, explicando la

mayor demora en los tiempos de cómputo de configuraciones con errores menores.

En la tabla 6.22 hay un resumen de la información que otorgaron los gráficos anteriores, entre otros datos y permite ver más claramente por ejemplo, cómo por cada configuración con errores más restrictivos, la cantidad de *substeps* por *step* se triplica aproximadamente.

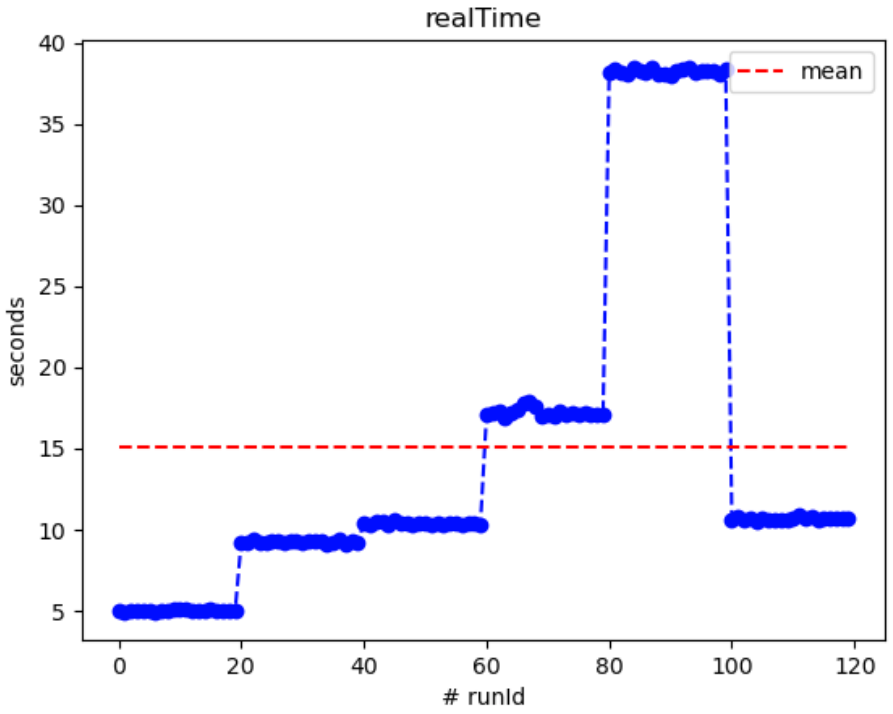

Fig. 6.17: Gráfico de tiempos de QSS por cada experimento, para el caso F03. Los runId del 0 al 20 representan la configuración dQRel 0.1 y dQMin 0.01; Los runId del 20 al 40 representan la configuración dQRel 0.001 y dQMin 0.0001; Los runId del 40 al 60 representan la configuración dQRel 0.0001 y dQMin 0.00001; Los runId del 60 al 80 representan la configuración dQRel 0.00001 y dQMin 0.000001; Los runId del 80 al 100 representan la configuración dQRel 0.000001 y dQMin 0.0000001; Los runId del 100 al 120 representan la configuración dQRel 0.01 y dQMin 0.001;

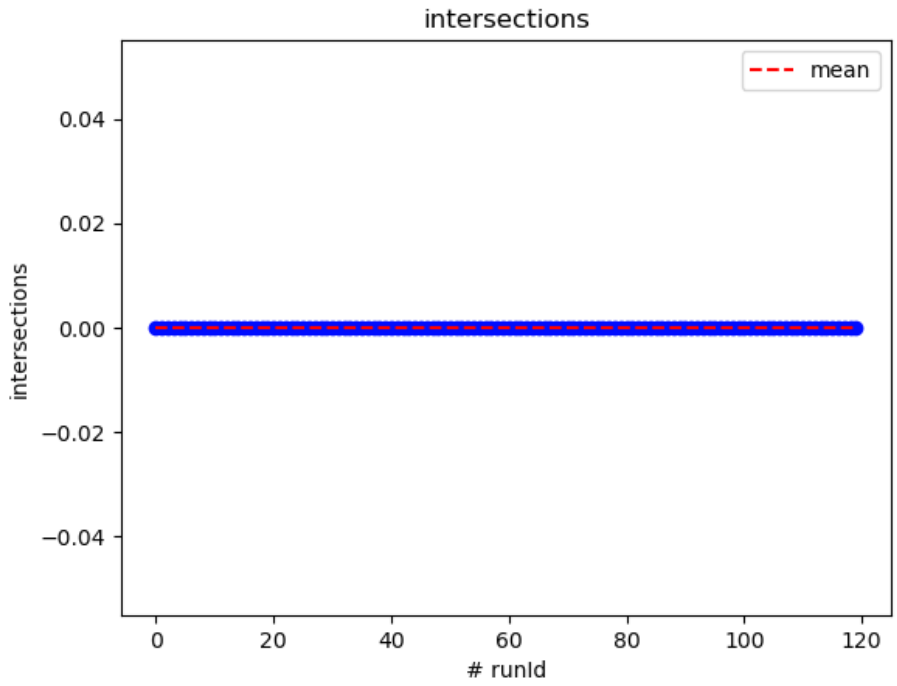

Fig. 6.18: Gráfico de intersecciones de QSS por cada experimento, para el caso F03. Los runId del 0 al 20 representan la configuración dQRel 0.1 y dQMin 0.01; Los runId del 20 al 40 representan la configuración dQRel 0.001 y dQMin 0.0001; Los runId del 40 al 60 representan la configuración dQRel 0.0001 y dQMin 0.00001; Los runId del 60 al 80 representan la configuración dQRel 0.00001 y dQMin 0.000001; Los runId del 80 al 100 representan la configuración dQRel 0.000001 y dQMin 0.0000001; Los runId del 100 al 120 representan la configuración dQRel 0.01 y dQMin 0.001;

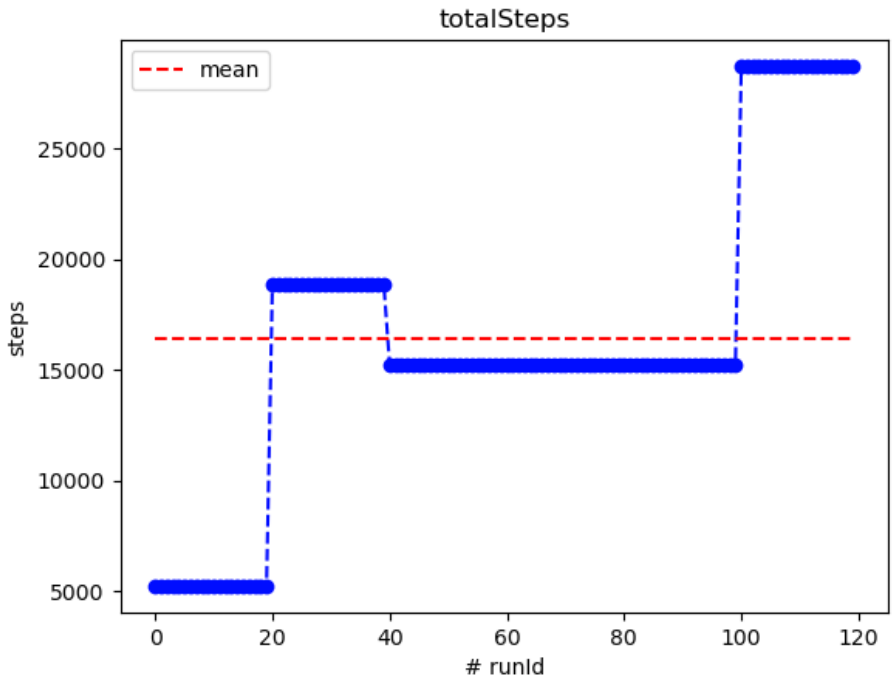

Fig. 6.19: Gráfico de cantidad total de steps de QSS por cada experimento, para el caso F03. Los runId del 0 al 20 representan la configuración dQRel 0.1 y dQMin 0.01; Los runId del 20 al 40 representan la configuración dQRel 0.001 y dQMin 0.0001; Los runId del 40 al 60 representan la configuración dQRel 0.0001 y dQMin 0.00001; Los runId del 60 al 80 representan la configuración dQRel 0.00001 y dQMin 0.000001; Los runId del 80 al 100 representan la configuración dQRel 0.000001 y dQMin 0.0000001; Los runId del 100 al 120 representan la configuración dQRel 0.01 y dQMin 0.001;

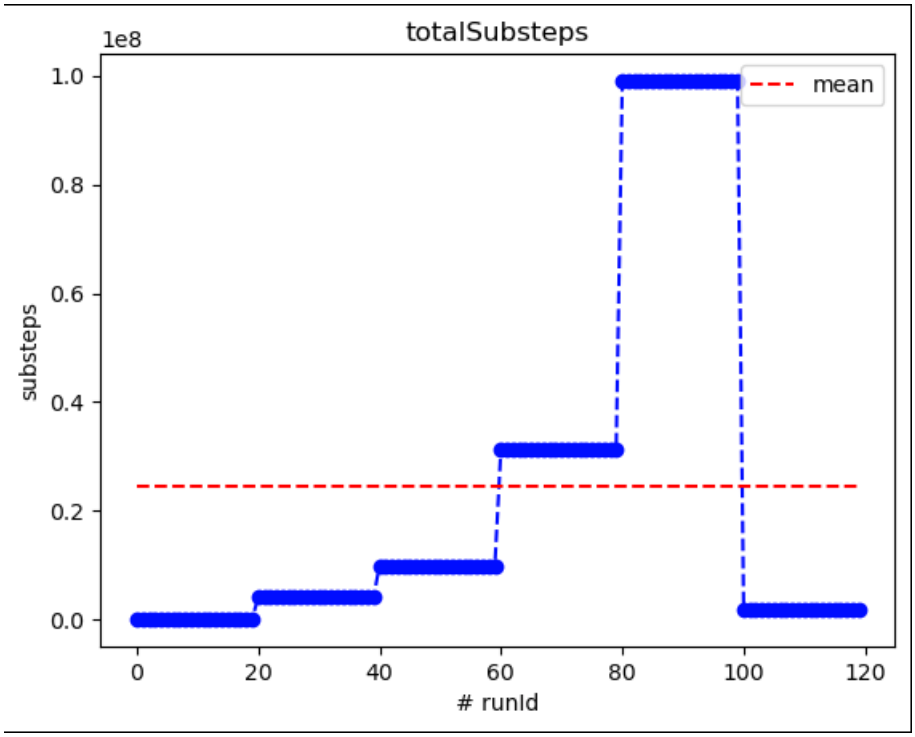

Fig. 6.20: Gráfico de cantidad total de substeps de QSS por cada experimento, para el caso F03. Los runId del 0 al 20 representan la configuración dQRel 0.1 y dQMin 0.01; Los runId del 20 al 40 representan la configuración dQRel 0.001 y dQMin 0.0001; Los runId del 40 al 60 representan la configuración dQRel 0.0001 y dQMin 0.00001; Los runId del 60 al 80 representan la configuración dQRel 0.00001 y dQMin 0.000001; Los runId del 80 al 100 representan la configuración dQRel 0.000001 y dQMin 0.0000001; Los runId del 100 al 120 representan la configuración dQRel 0.01 y dQMin 0.001;

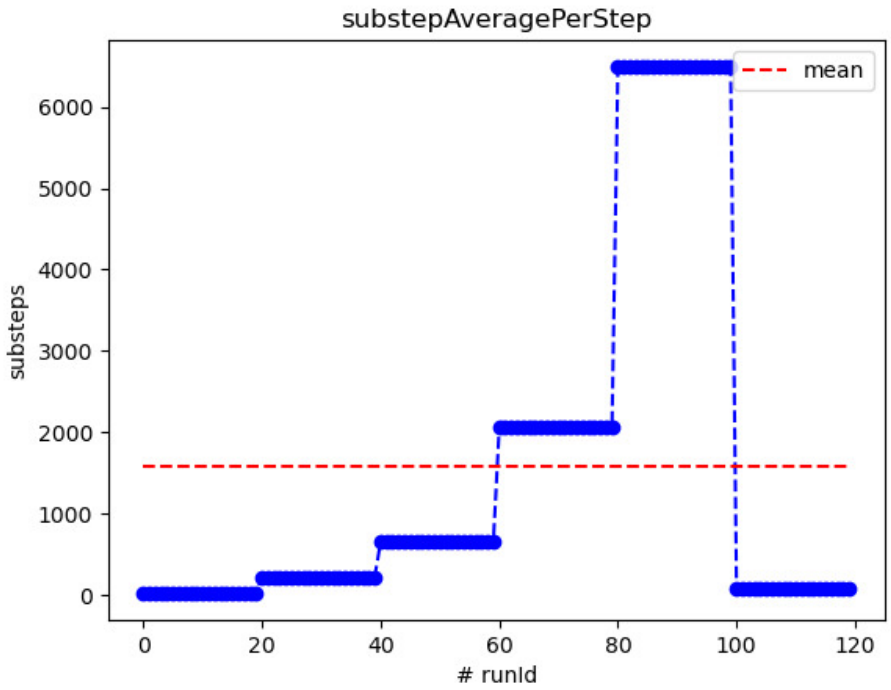

Fig. 6.21: Promedio de cantidad de substeps por step en F03 utilizando QSS. Los runId del 0 al 20 representan la configuración dQRel 0.1 y dQMin 0.01; Los runId del 20 al 40 representan la configuración dQRel 0.001 y dQMin 0.0001; Los runId del 40 al 60 representan la configuración dQRel 0.0001 y dQMin 0.00001; Los runId del 60 al 80 representan la configuración dQRel 0.00001 y dQMin 0.000001; Los runId del 80 al 100 representan la configuración dQRel 0.000001 y dQMin 0.0000001; Los runId del 100 al 120 representan la configuración dQRel 0.01 y dQMin 0.001;

Stepper Name	dQrel	dQmin	Number of threads	intersections	Total Steps	Substeps per step	Real Time (seg)	Average Time per step(seg)	SpeedUp Real Time vs DopriT
DopriT			1	0	15250		6,961	4,565E-04	
DopriNoT			1	0	23046		8,248	3,579E-04	-18,49 %
RK4			1	0	15250		6,895	4,521E-04	0,95 %
QSS2	1,00E-01	1,00E-02	1	0	5215	18,77	4,997	9,581E-04	28,22 %
QSS2	1,00E-02	1,00E-03	1	0	28737	67,74	10,645	3,704E-04	-52,92 %
QSS2	1,00E-03	1,00E-04	1	0	18888	218,08	9,239	4,891E-04	-32,72 %
QSS2	1,00E-04	1,00E-05	1	0	15250	650,61	10,359	6,793E-04	-48,81 %
QSS2	1,00E-05	1,00E-06	1	0	15250	2055,45	17,207	1,128E-03	-147,19 %
QSS2	1,00E-06	1,00E-07	1	0	15250	6495,88	38,226	2,507E-03	-449,14 %

Fig. 6.22: Características y Tiempos F03

6.6.2. Baja precisión RK4

Como se observa en la tabla 6.5, si bien RK4 tiene mejor tiempo que DOPRI, el MSE en la coordenada X es demasiado alto. No solo eso, sino que si observamos la tabla 6.23, se puede observar que QSS con dQRel 1E-05, dQMin 1E-06 y dQRel 1E-06, dQMin 1E-07 tiene mejor θ_{MSE} y θ_{MAX} que RK4 para todas las coordenadas. Esto llama la atención

puesto que en el resto de los experimentos no se observa este comportamiento. Si vemos los gráficos 6.24 , 6.25 y 6.26 podemos observar como RK4 tiene una buena precisión hasta el último punto del final. Se puede observar este comportamiento si comparamos las imágenes 6.27 y 6.11 que muestran toda la trayectoria de la partícula para RK4 y DOPRI, y se puede observar cómo ambas son muy parecidas, salvo por un punto al final en donde termina RK4.

Si bien no podemos explicar el por qué de este punto, la herramienta creada en la presente tesis nos permite identificar rápidamente este comportamiento indeseado, y una vez identificado, se podría proceder a intentar encontrar la causa del mismo.

Stepper Name	dQrel	dQmin	Theta Max en X	Theta Max en Y	Theta Max en Z	Theta MSE en X	Theta MSE en Y	Theta MSE en Z
DopriT			NA	NA	NA	NA	NA	NA
DopriNoT			inf	inf	inf	inf	inf	inf
RK4			1,004E-04	1,041E-02	4,743E-03	6,962E-05	7,469E-01	1,552E-01
QSS2	1,00E-01	1,00E-02	8,340E-05	1,742E-03	1,822E-03	2,246E-07	1,067E-04	2,501E-04
QSS2	1,00E-02	1,00E-03	1,351E-05	1,800E-03	2,296E-03	4,974E-07	4,680E-04	3,501E-04
QSS2	1,00E-03	1,00E-04	9,159E-05	7,765E-03	1,036E-02	1,455E-06	1,019E-02	9,826E-03
QSS2	1,00E-04	1,00E-05	4,279E-05	5,442E-03	2,967E-03	1,858E-05	2,395E-01	8,028E-02
QSS2	1,00E-05	1,00E-06	1,988E-02	6,386E-01	5,455E-01	3,544E-02	8,070E+01	4,028E+01
QSS2	1,00E-06	1,00E-07	6,032E-02	3,408E+00	3,001E+00	1,388E+00	5,358E+03	2,515E+03

Fig. 6.23: Indices de Desempeño F03

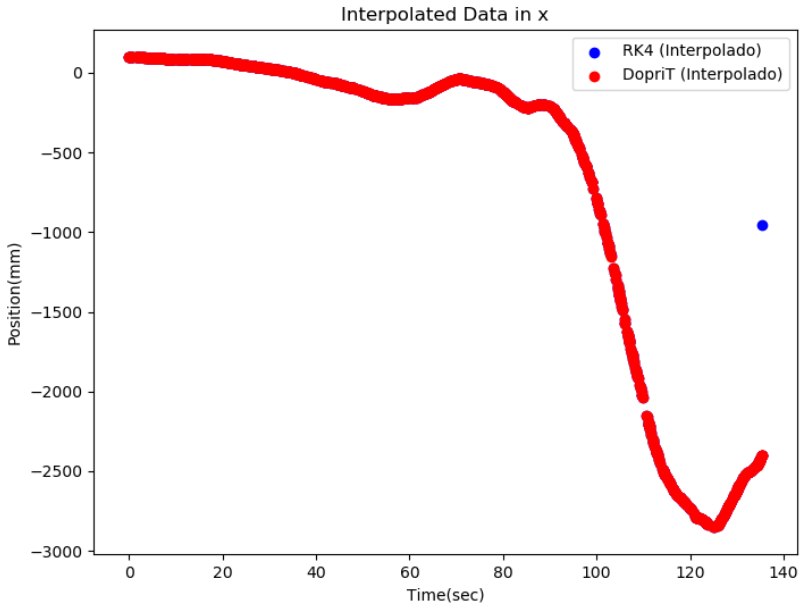

Fig. 6.24: Posición en X a través del tiempo, Experimento F03, RK4 VS DOPRI.

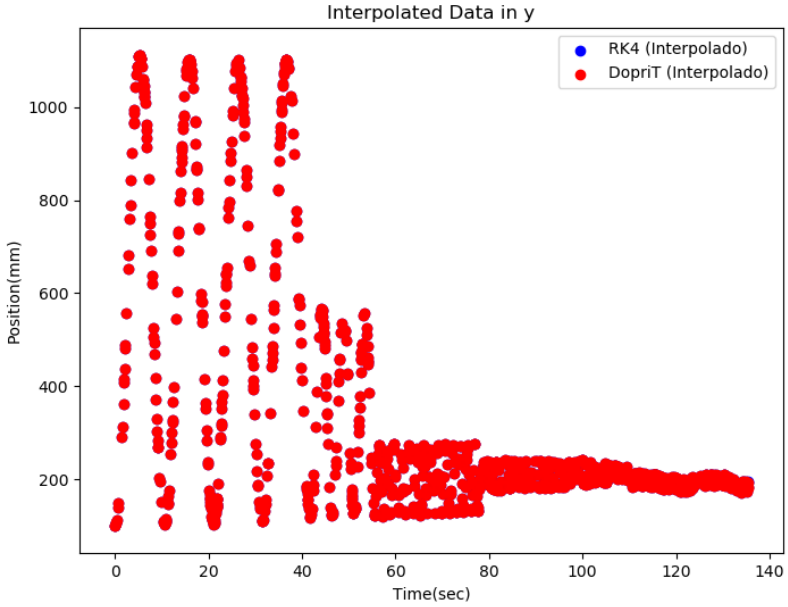

Fig. 6.25: Posición en Y a través del tiempo, Experimento F03, RK4 VS DOPRI.

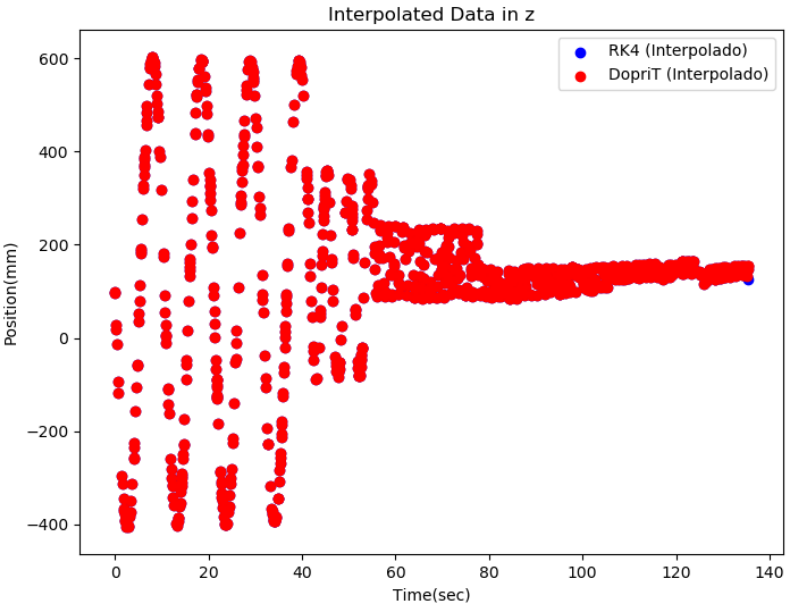

Fig. 6.26: Posición en Z a través del tiempo, Experimento F03, RK4 VS DOPRI.

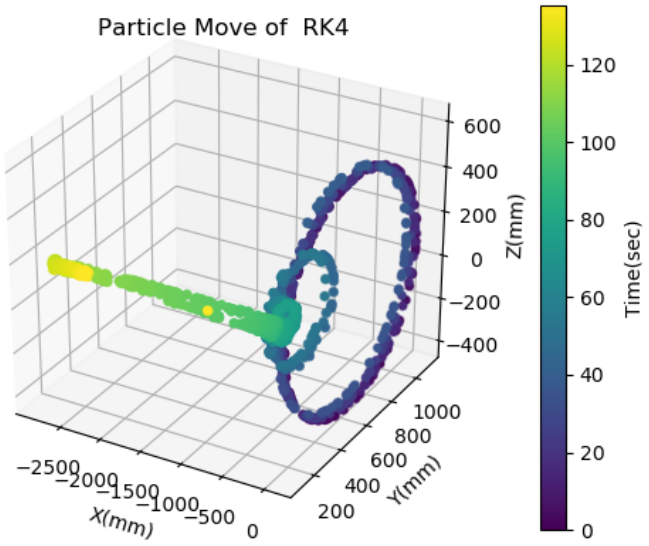

Fig. 6.27: Posición de la partícula a través del tiempo, RK4

7. CONCLUSIONES Y PRÓXIMOS PASOS

La mayoría de los objetivos propuestos se alcanzaron a lo largo del trabajo de la presente tesis. Se lograron añadir siete ejemplos de detectores de partículas reconocidos por el CERN como pruebas relevantes para la comprensión de las simulaciones de trayectorias de partículas. Seis de ellos pertenecen a los tests estándar de Geant4 y el séptimo es FSL, un ejemplo de una aplicación real de gran importancia para la simulación del experimento ATLAS del CERN.

También se contribuyó con una herramienta, junto con su metodología correspondiente, para sistematizar la comparación de la eficiencia y la precisión entre integradores numéricos.

Esto facilitó enormemente la tarea ineludible de analizar un gran conjunto de datos crudos, ya que presenta la información de forma ordenada, consistente y replicable, y potencia las capacidades de análisis comparativos (una actividad crucial en la disciplina).

Las nuevas tablas y gráficos demostraron permitir analizar en cuestión de minutos el desempeño de diversos integradores en un experimento dado, así como compararlos con experimentos similares.

Se logró un diseño extensible de la herramienta capaz de incorporar fácilmente nuevos experimentos, nuevos integradores y nuevas métricas de comparación. El mecanismo propuesto para la interpolación de series temporales que no comparten una misma marca de tiempo resultó eficaz para la construcción de métricas de error.

También se pudo obtener un gráfico en 3D con la trayectoria de movimiento de una partícula comparando varios métodos de integración, sin tener que recurrir a herramientas externas (como, por ejemplo, Paraview).

Finalmente, la herramienta se desarrolló de manera independiente de QSS, es decir, que sirve para la comparación de otros posibles integradores que se desarrollen en el futuro.

En el terreno práctico, disponer de esta herramienta permitió cumplir con los plazos de desarrollo de la nueva versión estable de Geant4, llegando a aportar por primera vez los métodos QSS en la versión pública oficial productiva del simulador (formando parte de la versión 11.2.0-ref-02). Esto supuso un hito para esta línea de investigación y para el

Laboratorio de Simulación de Eventos Discretos.

7.1. Próximos Pasos

De esta tésis se desprenden varias aristas de investigación futuras. Entre las más importantes podemos mencionar:

- Optimización del *stepper* QSS. De los resultados de esta tésis se desprende que solo en uno de los siete ejemplos de estudio QSS tiene un mejor desempeño que DOPRI. Más aún, en el ejemplo F03 se pudo observar cómo en condiciones claramente desfavorables para QSS, el tiempo de ejecución no es aceptable.
- Hallar una configuración de QSS óptima. Es necesario encontrar una configuración de *dQRel* y *dQMin* óptimas para QSS, y que se pueda utilizar por defecto, para que los usuarios de Geant4 puedan utilizarla, ahora que QSS pertenece oficialmente al conjunto de *steppers* utilizados en Geant4.
- Extender QSS para que acepte otros campos además del magnético. Sería interesante, por ejemplo, poder usar QSS en campos eléctricos.
- Extender el sistema automático de comparación de integradores para poder generar gráficas de *performance* comparativas entre experimentos.

BIBLIOGRAFíA

[1] John R Dormand y Peter J Prince. «A family of embedded Runge-Kutta formulae». En: *Journal of computational and applied mathematics* 6.1 (1980), págs. 19-26.

[2] J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. 1.ª ed. New York: Wiley-Interscience, 1987. ISBN: 0-471-91046-5.

[3] Bernard P Zeigler y Jong Sik Lee. «Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment». En: *Aerospace/Defense Sensing and Controls*. International Society for Optics y Photonics. 1998, págs. 49-58.

[4] E. Kofman y S. Junco. «Quantized State Systems. A Devs Approach For Continuous System Simulation». En: *Transactions of Scs* 18.3 (2001), págs. 123-132.

[5] Ernesto Kofman y Sergio Junco. «Quantized-state systems: a DEVS Approach for continuous system simulation». En: *Transactions of the Society for Modeling and Simulation International* 18.3 (2001), págs. 123-132.

[6] Ernesto Kofman. «A second-order approximation for DEVS simulation of continuous systems». En: *Simulation* 78.2 (2002), págs. 76-89.

[7] Ernesto Kofman. «Discrete event simulation of hybrid systems». En: *SIAM Journal on Scientific Computing* 25.5 (2004), págs. 1771-1797.

[8] François E Cellier y Ernesto Kofman. *Continuous system simulation*. Springer Science & Business Media, 2006.

[9] Francois E. Cellier y Ernesto Kofman. *Continuous System Simulation*. 1.ª ed. Berlin: Springer-Verlag, 2006.

[10] Ernesto Kofman. «A third order discrete event method for continuous system simulation». En: *Latin American applied research* 36.2 (2006), págs. 101-108.

[11] Roman Adolphi et al. «The CMS experiment at the CERN LHC». En: *Journal of Instrumentation* 3.08 (ago. de 2008), S08004-S08004.

[12] Lyndon Evans y Philip Bryant. «LHC Machine». En: *Journal of Instrumentation* 3.08 (ago. de 2008), S08001-S08001.

[13] Nicolás Bruno Ponieman. «Simulación por cuantificación de estados para modelos de trayectoria de partículas. Estudio comparativo respecto de Geant4 para aplicaciones en física de altas energías.» En: (2015).

[14] J. Allison et al. «Recent Developments In Geant4». En: *Nuclear Instruments and Methods In Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 835 (2016), págs. 186-225.

[15] D. Elvira. «Impact of Detector Simulation In Particle Physics Collider Experiments». En: *Physics Reports* 695 (2017), pág. 81. ISSN: 0370-1573.

[16] L. Santi et al. «Application of State Quantization-Based Methods In Hep Particle Transport Simulation». En: *Journal of Physics: Conference Series* 898.4 (2017), pág. 042049.

[17] L. Santi et al. «Application of State Quantization-Based Methods In Hep Particle Transport Simulation». En: *Journal of Physics: Conference Series* 898.4 (2017), pág. 042049.

[18] Lucio Santi et al. «Application of State Quantization-Based Methods in HEP Particle Transport Simulation». En: *Journal of Physics: Conference Series* 898.4 (2017), pág. 42049. ISSN: 17426596. DOI: 10.1088/1742-6596/898/4/042049. URL: http://stacks.iop.org/1742-6596/898/i=4/a=042049.

[19] L. Santi y R. Castro. «A Co-simulation Technique for Efficient Particle Tracking Using Hybrid Numerical Methods with Application in High Energy Physics». En: *Proceedings of the 2018 Winter Simulation Conference*. WSC '18. Gothenburg, Sweden: IEEE Press, 2018, págs. 1322-1333.

[20] L. Santi et al. «GQLink: an implementation of Quantized State Systems (QSS) methods in Geant4». En: *Journal of Physics: Conference Series* 1085 (sep. de 2018), pág. 052015.

[21] Marilena Bandieramonte, Riccardo Maria Bianchi y Joseph Boudreau. «FullSimLight: ATLAS standalone Geant4 simulation». En: (2020).

[22] Lucas Rossi. «Simulación por Cuantización de Estados en Física de Altas Energías». En: (2021).

[23] Lucio Santi. «Nuevos Métodos Híbridos para Modelado y Simulación Eficiente de Partículas en Geometrías 3D». En: (2021).

[24] Geant4 Collaboration. *Geant4 Examples Documentation*. 2024. URL: `https://geant4-userdoc.web.cern.ch/Doxygen/examples_doc/html/index.html` (visitado 18-10-2024).

[25] CERN. Accedido el 29/01/2024. URL: `https://home.cern`.