uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

Trabajo de Fin de Grado

«Procesamiento de señales de encefalograma para la detección de ataques epilépticos.»

Autor
José Antonio Verde Jiménez

Tutor
José Daniel García Sánchez

Leganés, España,
8 de septiembre de 2025

Beware of bugs in the above code;
I have only proved it correct,
not tried it.

— **Donald E. Knuth**

RESUMEN

La epilepsia es una de las enfermedades cerebrales más comunes, la detección de ataques epilépticos a partir de señales de encefalograma puede tener un impacto positivo en la vida de los pacientes que la padecen. Pero para ello es necesario detectarlos a tiempo.

Esta tesis toma de base un estudio sobre de detección de ataques epilépticos utilizando patrones aumentados y las características estadísticas de la señal de encefalogama para implementarlo en un sistema de bajo consumo en tiempo real, además de diminuir el tiempo de entrenamiento del modelo.

Se hace un estudio sobre el impacto de distintas técnicas de programación, del paralelismo, de varios lenguajes de programación (Python 3, C++ y Ada) y de utilizar números codificados en punto fijo y en punto flotante en el tiempo de ejecución del programa en las distintas plataformas.

También se hace un estudio exhaustivo y riguroso de los algoritmos que forman el detector para demostrar formalmente mediante SPARK la ausencia de errores en tiempo de ejecución y demostrar las propiedades del propio programa.

Palabras clave: Señales de encefalograma • Epilepsia • Rendimiento • Tiempo real

ABSTRACT

Epilepsy is one of the most common brain-related diseases. Electroencephalography-based seizure detection may have a possitive impact in patients who suffer it. But for that to happen, it is essential to detect them on time.

This thesis builds upon a study about pattern augmented by features epileptic seizure detection, as a means to implement it in an energy-efficient real-time system. And to reduce training time.

Different programming techniques, parallelism, various programming languages (Python 3, C++ and Ada) and real number encoding (fixed types and floating types) are analysed. And their performance is measured in different platforms.

A thorough and strict study about the algorithms that make-up the detector is performed as well. By using SPARK it is possible to formally prove the absence of run-time errors and certain properties about the program itself.

Keywords: EEG Signals • Seizure detection • Performance • Real time

AGRADECIMIENTOS

Me gustaría dedicarle esta tesis a esas personas que me fueron guiando desde el inicio y, gracias a las cuales, hoy puedo presentar esta tesis. A mi tutor, José Daniel García Sánchez; y a los investigadores de la Universidad de Málaga con los que cooperamos, que escribieron el artículo en el cual se basa este trabajo: a María Ángeles González Navarro, a Felipe Muñoz López y especialmente a Rafael Asenjo Plaza, que descanse en paz. A vosotros os lo dedico y os agradezco de todo corazón todo el tiempo que me habéis dedicado, los recursos que me habéis dado y el interés por verlo completado. Muchas gracias.

Asimismo, agradezco a mis compañeros del laboratorio de ARCOS de la Universidad Carlos III de Madrid, por haber estado a mi lado y por haberme ayudado cuando lo necesitaba. A Santiago, por haberme ayudado con partes más técnicas del desarrollo, por haberse quedado conmigo horas y horas depurando el programa. Gracias a él funciona mi `CMakeLists.txt`. A Álvaro Guerrero y a Lucía por haberme ayudado con varias demostraciones y a depurar y discutir distintas soluciones para varios problemas. Y al resto de compañeros del laboratorio de ARCOS: a Diego, a Álvaro, a Daniel y a Elisa. Muchas gracias.

Igualmente, quiero dar las gracias a Luis Daniel, a Jorge Lázaro y a Eduardo Alarcón por sus contribuciones para preparar y mejorar la plantilla de este mismo documento; y agradecer a Diego Díaz Huertas por su revisión de las demostraciones, las convenciones y el estilo de los teoremas matemáticos. Muchas Gracias.

Finalmente, a esas personas que me han estado apoyando desde el principio, en mis peores y en mis mejores momentos, que siempre han estado ahí cuando más lo necesitaba. A vosotros os lo dirijo: a mi familia y a mis amigos. Muchas gracias.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS

CAPÍTULO 1

INTRODUCCIÓN

Este capítulo describe brevemente la motivación que existe detrás del proyecto (Sección 1.1), los objetivos principales del mismo (Sección 1.2) y se resume brevemente el contenido de cada uno de los capítulos (Sección 1.3). El código y la documentación de este proyecto están alojados en https://github.com/joseaverde/TFG .

1.1. Motivación

Este proyecto surge de la colaboración entre el grupo de investigación de modelos de programación paralela y compiladores de la Universidad de Málaga (PPMC) [1], que escribió un artículo sobre detección de ataques epilépticos utilizando patrones aumentados y características estadísticas [2], y la Universidad Carlos III de Madrid.

El estudio obtuvo resultados prometedores y el foco pasó a ser la viabilidad de ejecución del algoritmo de detección en un dispositivo empotrado de bajo consumo en tiempo real. La detección de estos ataques es computacionalmente costosa, pero si fuera factible que funcionase en tales dispositivos abriría las puertas al desarrollo de accesorios que monitoreen la actividad cerebral de un paciente en tiempo real.

Finalmente se quería estudiar cuál es el impacto del paralelismo en el tiempo de ejecución del programa de entrenamiento del modelo, pues consume mucha energía y tiempo ejecutarlo y cualquier mejora es necesaria; y también qué técnicas se podrían aplicar a los algoritmos para mejorar su rendimiento en máquinas de bajas características computacionales.

1.2. Objetivos

El objetivo principal del proyecto es hacer un estudio de distintas técnicas y algoritmos para la optimización de programas. Se ha utilizado un artículo redactado por investiga-

dores de la universidad de Málaga [2] como base para explorar en qué formas se puede mejorar el rendimiento del mismo.

Se ha trabajado en compañía de dicho equipo para intentar mejorar los tiempos de entrenamiento y para confirmar la viabilidad de ejecutar el algoritmo de detección en un sistema empotrado de bajo consumo. Se decidió también que el sistema estuviera escrito en C++, por su increíble rendimiento y su facilidad para ejecutarlo en sistemas empotrados.

A mitad del proyecto sugió un tercer objetivo tras determinar la viabilidad de utilizar números en punto fijo en vez de punto flotante para la representación y procesamiento de los datos. Había que verificar formalmente que no hubiera casos de desbordamiento en operaciones de punto fijo. Para ello se usó el lenguaje de programación Ada junto a SPARK, que usa probadores de teoremas por debajo, para verificar formalmente el trabajo.

- **O1**: Implementar un módulo de Python3 en C++ que disminuya el tiempo de entrenamiento del modelo para la detección de ataques epilépticos de la implementación de referencia [3].
- **O2**: Implementar un sistema de tiempo real en un dispositivo empotrado de bajo consumo que utilice el modelo generado para clasificar épocas de señal en «ataques epiléticos» y en «libres de ataques epilépticos».
- **O3**: Verificar formalmente utilizando un probador de teoremas interactivo que el programa se comporta como debe y no puede terminar de manera abrupta.

1.3. Estructura del documento

El documento contiene los siguientes capítulos:

- Capítulo 1 – *Introducción*, explica la motivación que llevó a realizar este proyecto, los objetivos del mismo y la propia estructura.
- Capítulo 2 – *Estado del arte*, un análisis del estado de la cuestión, qué temas son relevantes y qué estudios existen al respecto. Analiza estudios sobre detección de ataques epilépticos, demostraciones interactivas de teoremas y técnicas de programación.
- Capítulo 3 – *Análisis*, estudia y analiza los casos de uso del sistema, a partir de los cuales enumera y examina los requisitos funcionales y no funcionales del mismo. Finalmente se da el diagrama con la arquitectura del sistema.
- Capítulo 4 – *Diseño e implementación*, contiene el estudio de la solución final: define matemáticamente el problema, a base de la cual se justifican las decisiones de diseño del problema.
- Capítulo 5 – *Validación, verificación y evaluación*, efectúa una validación y una verificación del programa final, y hace una evaluación del rendimiento del mismo.
- Capítulo 6 – *Marco regulador*, estudia la legislación aplicable sobre el mismo y las licencias de las herramientas utilizadas y las dependencias con las que se enlaza la

solución, además se identifican los estándares técnicos relevantes para el desarrollo del proyecto.

- Capítulo 7 – *Entorno socio-económico y objetivos de desarrollo sostenible*, elabora un presupuesto para el proyecto, hace un análisis del entorno socio-económico y compara a qué objetivos de desarrollo sostenible se adhiere.
- Capítulo 8 – *Planifiación*, separa el proyecto en distintas tareas y elabora un cronograma indicando en qué momento empiezan y en qué momento terminan.
- Capítulo 9 – *Conclusiones y trabajo futuro*, concluye el proyecto, resume qué se hizo y estudia cómo se podría continuar y qué se podría mejorar.

CAPÍTULO 2

ESTADO DEL ARTE

Este capítulo presenta las distintas tecnologías que guardan una estrecha relación con el proyecto en cuestión: desde las técnicas para el procesamiento de señales de encefalograma, hasta cuestiones relacionadas con la parte de computación y entornos de ejecución.

2.1. Detección de ataques epilépticos

Según la Organización Mundial de la Salud, la epilepsia es una enfermedad crónica que afecta al cerebro y a gente de todas las edades. Se calcula que alrededor de 50 millones de personas en todo el mundo la padecen, lo que la posiciona como una de las afecciones neurológicas más comunes de todo el mundo [4]. El riesgo de muerte prematura es tres veces mayor a resto de la población [4].

Además los ataques epilépticos se pueden controlar, alrededor del 70% de la gente que sufre de epilepsia puede vivir sin experimentar ninguno con las medicación apropiada [4]. Una etiología documentada de la epilpsia o patrones anormales de encefalografía son los predictores más consitentes de epilepsia [4].

Una evaluación de detección de ataques epilépticos utilizando clasificadores de apredizaje automático explica que aplicar aprendizaje automático directamente sobre el conjunto de datos de encefalografía en bruto puede no producir patrones sensatos [5]. Por lo tanto, seleccionar las características estadísticas de la señal de encefalograma es crucial, entre ellas se encuentran distintas técnicas de transformación como: transformaciones de ondículas discretas (*discrete wavelet transformation* o DWT), transformaciones de ondículas continuas (*continuous wavelet transformation* o CWT), transformadas de Fourier (*Fourier transformation*, FT), transformaciones de conseno discretas (*discrete cosine transformation* o DCT), descomposición en valores singulares (*singular value decomposition* o SVD) entre otras [5].

2.1.1. PaFESD: *Patterns Augmented by Features Epileptic Seizure Detection*

O patrones aumentados por características de detección de ataques epilépticos, es un método para detección de ataques epilépticos a partir de las características estadísticas de encefalografía además de comparación de patrones. Las características principales son tres:

1. $f_1(E_i)$: la amplitud pico a pico (*pk-pk*) de la época, que es la diferencia entre el el valor máximo y mínimo. En este proyecto se refiere a ella como *max distance* o distancia máxima.
2. $f_2(E_i)$: la energía de la época como la varianza de la época. En este proyecto se la denomina *energy* o energía.
3. $f_3(E_i)$, $f_4(E_i)$, $f_5(E_i)$: es la potencia integrada de la banda espectral de la señal de encefalograma en las bandas, respectivamente: de 2.5 a 12 Hz ,cubriendo casi completamente las ondas cerebrales α, β y θ; de 12 a 18 Hz, banda baja de las ondas β; y de 18 a 35 Hz, bandas altas de las ondas β. Se computa como la integral de cada banda de la densidad espectral de potencia de la época a partir del método de Welch. A estas se las denomnina PSD_1, PSD_2 y PSD_3 en el resto del documento.

Finalmente se utiliza la distancia dada por el algoritmo de la deformación dinámica del tiempo (DTW) para comparar cada época de señal con varios patrones previamente seleccionados [2].

Se ha decidido utilizar este estudio como referencia y se ha colaborado con los investigadores originales que escribieron el artículo. La implementación de referencia está alojada en https://github.com/PPMC-DAC/PaFESD-Epileptic-Seizure-Detection. Entre otras razones, porque con tan solo un 22% de datos de entrenamiento consigue una sensibilidad del 99.6%, una especificidad del 100% y una precisión del 98.3%, mucho mejor que otros modelos que necesitan mucho más porcentaje de entrenamiento [2].

2.2. Demostración interactiva de teoremas

2.2.1. *Rocq*

Rocq conocido hasta marzo de 2025 como *Coq* [6], es un programa para la demostración interactiva de teoremas matemáticos desarrollado por el instituto francés de investigación en informática y automática (INRIA). Fue diseñado para desarrollar demostraciones matemáticas y para escribir especificaciones formales [7].

Rocq se ha utilizado para demostrar gran variedad de teoremas matemáticos, entre los más notables: el teorema de los cuatro colores [8], el algoritmo del castor para $\text{BB}(5) = 47176870$ [9] o el grupo fundamental de un círculo en teoría de tipos homotópicos [10].

2.2.2. *Lean*

Lean es a la vez un probador de teoremas y un lenguaje de programación, cuyo compilador está escrito en C++ y en el propio *Lean*, que permite escribir código correcto, mantenible y verificado formalmente [11]. Hoy en día su desarrollo está apoyado por la organización sin ánimo de lucro *Lean Focused Research Organization* (FRO) [12]. Un

ejemplo de cómo definir los números naturales (cero incluido) basándose en los axiomas de Peano en *Lean* se ve en la Listado 2.1:

Listado 2.1

Axiomas de Peanno en Lean

```Lean
inductive Natural : Type
| zero : Natural
| succ : Natural -> Natural
```

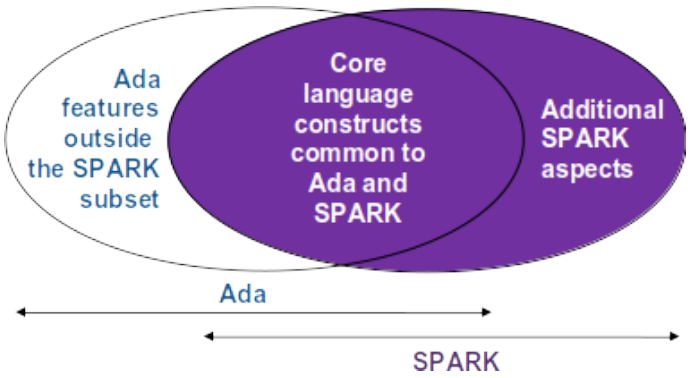
2.2.3. SPARK

SPARK 2014 es un lenguaje de programación definido formalmente y un conjunto de herramientas de verificación diseñadas específicamente para permitir el desarrollo de *software* de alta integridad. Con SPARK los desarrolladores pueden verificar formalmente las propiedades de su código tales como: flujo de información, ausencia de errores en tiempo de ejecución, corrección funcional y políticas de seguridad [13].

SPARK 2014 está basado en un subconjunto del lenguaje de programación Ada. Ada es particularmente apto para la verificación formal pues fue diseñado para desarrollo de *software* crítico, el cual usa de base. Ada 2012 introdujo el uso de aspectos que se pueden utilizar para denotar contratos en subrutinas [14]. Además SPARK 2014 también añade sus propios aspectos para extender la capacidad de análisis estático [15], como se muestra en la Figura 2.1

Figura 2.1

Diagrama de Venn que muestra la relación entre SPARK y Ada [15]



El juego de herramientes de *GNATprove* está basado en la colección de compiladores de GCC y utiliza por debajo distintos probadores de teoremas como: *Alt-Ergo*, *Colibri*, *cvc5* y *Z3* por defecto. También puede utilizar *Coq* 8.11 (actualmente conocido como Rocq a partir de la versión 9) para realizar demostraciones interactivas de teoremas [16].

Por ejemplo, el siguiente programa en SPARK (Listado 2.2), que también es compatible con Ada y puede compilarse utilizando un compilador de Ada, tiene varios problemas: que las funciones `Get` y `Put` pueden lanzar excepciones de entrada y salida, y que `X + Y` puede desbordar. Y SPARK es capaz de reconocerlos.

Listado 2.2

Ejemplo de programa válido en Ada y SPARK, pero en el que pueden saltar excepciones.

```
1  with Ada.Text_IO, Ada.Integer_Text_IO;                    SPARK
2  use Ada.Text_IO, Ada.Integer_Text_IO;
3  procedure Program with SPARK_Mode is
4     X, Y : Integer;
5  begin
6     Put ("Introduzca dos números: ");
7     Get (X);
8     Get (Y);
9
10    Put ("La suma de ");
11    Put (X, 1);
12    Put (" + ");
13    Put (Y, 1);
14    Put (" es ");
15    Put (X + Y, 1);
16    New_Line;
17 end Program;
```

Para solucionar el problema de las excepciones basta con añadir un bloque para tratar excepciones entrada y salida con `exception when others =>`. Para la suma es más complicado, pues es la entrada del usuario y este puede introducir cualquier valor. Una opción es utilizar un tipo más grande para almacenar el resultado; otra opción sería utilizar un subrango de valores válidos para `X` e `Y`; y otra opción podría ser identificar el desbordamiento o subdesbordamiento antes de que ocurra e imprimir un mensaje de error. Por ejemplo, el primer caso (utilizar un tipo más grande) se muestra en el Listado 2.3.

Listado 2.3

Posible solución para eliminar las excepciones del Listado 2.2. El `when others` solo captura las excepciones de entrada y salida, ninguna de tipo numérico

```
1  with Ada.Text_IO, Ada.Integer_Text_IO, Ada.Long_Integer_Text_IO;   SPARK
2  use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Long_Integer_Text_IO;
3  procedure Program with SPARK_Mode is
4     X, Y : Integer;
5     R : Long_Integer;
6  begin
7     Put ("Introduzca dos números: ");
8     Get (X);
9     Get (Y);
10
11    Put ("La suma de ");
12    Put (X, 1);
```

```
13     Put (" + ");
14     Put (Y, 1);
15     Put (" es ");
16     R := Long_Integer (X) + Long_Integer (Y);
17     Put (R, 1);
18     New_Line;
19  exception
20     when others =>
21        Put_Line ("Ha ocurrido un error");
22  end Program;
```

Así se puede garantizar que no va a surgir una excepción inesperada. No se trata de evitar que el programa no tenga errores, se trata de identificar los casos en los que pueda fallar y tratarlos adecuadamente.

Por ejemplo, la función división `"/"` en Ada se define implícitamente para el tipo entero (`Integer`). Sin embargo, hay uno o dos posibles puntos de fallo: el primero, y más obvio, es que la división entre cero no está definida; el segundo, y solamente en ciertos computadores que por ejemplo codifican los valores enteros en complemento a dos, es que la división entre -1 desborda, no está definida cuando el numerador es -2^{n-1} en un entero de n bits, pues $\frac{-2^{n-1}}{-1} = 2^{n-1}$ no se puede codificar en un entero de n bits. En SPARK y en Ada se puede definir dicha propiedad como se muestra en el Listado 2.4.

Listado 2.4

Precondiciones implícitas de la función división entera `"/"`.

```
1  function "/" (Left, Right : in Integer) return Integer with                SPARK
2     Pre => (Right /= 0 and then
3        (if Integer'First < -Integer'Last
4           and then Left = Integer'First then Right /= -1))
5        or else raise Constraint_Error;
```

Al predicado que se debe cumplir antes de poder llamar una subrutina se llama precondición y es el que llama la función el que debe comprobarlo. La postcondición es el predicado que se asegura que se cumple después de haber llamado a una subrutina, y es la propia subrutina la que debe comprobar que este sea cierto.

2.3. Técnicas de programación

2.3.1. Diseño por contrato

El diseño por contrato, término acuñado por Bertrand Meyer que está conectado con el diseño del lenguaje de programación *Eiffel* [17]. En contra de la llamada «programación defensiva», que obliga a los programadores a proteger todos y cada uno de los módulos para cualquier caso, que añade código redundante y que complica el código, propone la noción de contrato [17]. Un contrato se divide en dos partes: una precondición, que la

parte que ofrece el servicio espera que sea cierta; y una postcondición, las garantías que ofrece el servicio a la parte que lo utiliza [17].

Varios lenguajes de programación dan soporte nativo para contratos, entre ellos Eiffel (el primer lenguaje) [18], Ada desde la versión 2012 del lenguaje [14] y en C++ hay una propuesta para la versión 26 del lenguaje, pero es controvertida [19].

2.3.2. Rangos

La programación basada en rangos consiste en aplicar y combinar distintos operadores sobre secuencias y vistas de objetos. Desde C++ 20 están en C++, la biblioteca de rangos es una extensión y generalización de las bibliotecas de algoritmos e iteradores que las hace más potentes al hacerlas combinables y menos propensas a errores [20].

2.3.2.1. *range-v3*

Es la biblioteca original en la que se basa la biblioteca de rangos del estándar de C++, se llama *range-v3* y fue desarrollada por Eric Niebler [21]. Contiene más vistas y acciones que las que están estadarizadas en C++ 23 y es compatible con las del estándar. Una parte de las funciones estandarizadas todavía no han sido implementadas en los compiladores de C++ más relevantes [22], así que todavía sigue siendo útil. Un ejemplo de uso se puede ver en el Listado 2.5.

Listado 2.5

Ejemplo de uso de la biblioteca *range-v3* para rangos.

```C++
constexpr auto result = accumulate
                        ( views::ints(0)
                        | views::remove_if([](int i){ return i % 2 == 1; })
                        | views::transform([](int i){ return i * i; );
                        | views::take(3)
                        );

static_assert(result == 12);
```

2.3.2.2. *flux*

Una biblioteca que utiliza un modelo alternativo para trabajar con rangos y vistas es *flux*, cuya implementación está alojada en https://github.com/tcbrindle/flux . En vez de utilizar el operador *pipe* (|) como hace el la biblioteca de rangos estandarizada y *range-v3*, utiliza notación punto (.), como se puede ver en el Listado 2.6.

Listado 2.6

Ejemplo de uso de la biblioteca *flux* para rangos.

```C++
constexpr auto result = flux::ints()
                          .filter(flux::pred::even)
                          .map([](int i) { return i * 2; })
                          .take(3)
```

```
5                           .sum();
6
7 static_assert(result == 12);
```

2.3.3. Paralelismo

El paralelismo es un conjunto de técnicas que permite realizar varias tareas de manera simultánea para mejorar su eficiencia [23]. En aplicaciones hay básicamente dos tipos de paralelismo:

1. Paralelismo a nivel de datos (*data-level parallelism* o DLP): nace de la posibilidad de operar muchos datos al mismo tiempo [23].
2. Paralelismo a nivel de tarea (*task-level parallelism* o TLP): aparece porque distintos trabajos pueden operar de manera independiente [23].

A nivel de *hardware* se pueden explotar esos dos tipos de paralelismo en aplicaciones de cuatro maneras distintas:

1. Paralelismo a nivel de instrucción (*instruction-level parallelism* o ILP): que explota el paralelismo a nivel de datos modestamente con ayuda del compilador y utiliza ideas como el *pipeline* (cadena de montaje) y a veces utilizando ejecución especulativa [23].
2. Arquitecturas vectoriales (*vector architectures*), unidades de procesamiento gráfico (*graphic processor units* o GPU) y juegos de instrucciones multimedia (*multimedia instruction sets*): que explotan el paralelismo a nivel de datos al aplicar una única instrucción a una colección de datos en paralelo [23].
3. Paralelismo a nivel de hilo (*thread-level parallelism*): que explota tanto paralelismo a nivel de datos o a nivel de tarea en un modelo de *hardware* fuertemente acoplado que permite la interacción entre hilos paralelos [23].
4. Paralelismo a nivel de petición (*request-level parallelism*): que explota el paralelismo entre tareas fuertemente desacopladas especificadas por el programador o el sistema operativo [23].

El concepto de concurrencia y de paralelismo están relacionados, pero existe un matiz que los diferencia. La concurrencia signfica que varias acciones están ocurriendo en el mismo intervalo de tiempo, mientras que paralelo indica que están ocurriendo a la vez [24].

La ley de Ambdahl, formulada por Gene Amdahl, habla sobre la máxima mejora en el rendimiento que puede obtener un programa si una porción del sistema es mejorada. Por ejemplo, si se mejora todo el programa para que vaya el doble de rápido, el programa resultante irá el doble de rápido. Pero si solo duplicamos la velocidad del programa en dos quintas partes, el sistema mejora por 1.25. Viene dado por la fórmula:

$$f = \left(\frac{1}{(1-t) + \left(\frac{t}{m}\right)} \right)$$

Donde t es el la fracción del total del tiempo que tarda el sistema a mejorar, m es el factor con que se ha mejorado dicha parte y f es el factor de mejora máximo del sistema final [24].

2.3.3.1. Paralelismo y concurrencia en lenguajes de programación

Algunos lenguajes de programación vienen con construcciones para realizar concurrencia y a veces paralelismo de forma nativa y portable. En el caso de este proyecto solo interesa ver cómo lo hacen los lenguajes de programación que se han utilizado: Python 3, C++ y Ada.

2.3.3.1.1. En C++

En C++ existen las clases `std::thread` y su versión mejorada `std::jthread` que dan soporte para hilos en dicho lenguaje. `std::jthread` tiene el mismo comportamiento que `std::thread`, pero cuando se destruye el objeto además hace *join* (espera a que el hilo termine de ejecutar [25]) y puede ser cancelado o parado en ciertas condiciones [26]. El siguiente programa del Listado 2.7 computa la suma de los elementos de un vector de manera paralela.

Listado 2.7

Suma de los elementos de un vector de manera paralela en C++, utilizando `std::thread`. Hay que partir a mano el vector en varios trozos. Es tedioso y todavía podría mejorarse.

```cpp
#include <vector>
#include <thread>
#include <algorithm>

template <class InputIt, typename T>
T accumulate (InputIt first, InputIt last, T init, std::size_t thread_count) {
  std::vector<std::thread> thread_pool;
  std::vector<T> result_pool(init, thread_count);
  std::size_t const length = last - first;
  std::size_t const chunk = length / thread_count;
  std::size_t const rem = length % thread_count;
  for (std::size_t p = 0; p < thread_count; p++) {
    std::size_t first_index = p * chunk + std::min(p, rem);
    std::size_t last_index = (p + 1) * chunk + std::min(p + 1, rem);
    result_pool.push_back(init);
    thread_pool.emplace_back(std::move(std::thread{
      [&result_pool] (InputIt first, InputIt last, std::size_t index) {
        for (auto it = first; it != last; ++it) { result_pool[index] += *it; }
      },
      first + first_index, first + last_index, p}));
  }

  for (auto & thread : thread_pool) { thread.join(); }

```

```
25    T result = init;
26    for (auto const & x : result_pool) { result += x; }
27    return result;
28  }
```

Sin embargo, desde C++17 se puede utilizar las políticas de ejecución (*execution policy*), que permite ejecutar algoritmos de manera paralela [27], como se muestra en el Listado 2.8 y en el Listado 2.9.

Listado 2.8

Reimplementación del Listado 2.7, pero utilizando la política de ejecución paralela de C++17.

```
1 #include <execution>                                                    C++
2 #include <numeric>
3 #include <functional>
4
5 template <class InputIt, typename T>
6 T accumulate (InputIt first, InputIt last, T init) {
7   return std::reduce(std::execution::par, first, last, init, std::plus<T>{});
8 }
```

Listado 2.9

Bucle `for each` en C++, utilizando política de ejecución paralela [27].

```
1 int x = 0;                                                            C++
2 std::mutex m;
3 int a[] = {1, 2};
4 std::for_each(std::execution::par, std::begin(a), std::end(a), [&](int)
5 {
6   std::lock_guard<std::mutex> guard(m);
7   ++x;
8 });
```

2.3.3.1.2. En Ada

La ejecución de un programa en Ada consiste en la ejecución de una o más tareas. Cada tarea representa una actividad separable que procede independientemente y concurrentemente entre puntos en los que interactúa con otras tareas [28]. Una única tarea, en el contexto de una construcción paralela, puede representar múltiples hilos lógicos de control que pueden proceder en paralelo; en otros contextos, cada tarea representa un hilo lógico de control [28]. Véase el Listado 2.10.

Listado 2.10

Implementación de la suma paralela de los elementos de un vector utilizando tareas en Ada. Tiene la misma funcionalidad que el Listado 2.7.

```ada
-- Specification
generic
   type Element_Type is private;
   type Index_Type is (<>);
   type Array_Type is array (Index_Type range <>) of Element_Type;
   Initial_Value : in Element_Type;
   with function "+" (Left, Right : in Element_Type) return Element_Type is <>;
function Accumulate (
   Item       : in Array_Type;
   Task_Count : in Positive := 1)
   return Element_Type with
   Pure, Global => null;

-- Body
function Accumulate (
   Item       : in Array_Type;
   Task_Count : in Positive := 1)
   return Element_Type is
   type Count_Type is new Long_Integer;
   task type Worker is
      entry Start (Index : Index_Type; Count : in Count_Type);
      entry Get (Result : out Element_Type);
   end Worker;

   task body Worker is
      Result : Element_Type := Initial_Value;
      Index  : Index_Type;
      Count  : Count_Type;
   begin
      accept Start (Index : Index_Type; Count : in Count_Type) do
         Worker.Index := Index;
         Worker.Count := Count;
      end Start;

      for I in 1 .. Count loop
         Result := Result + Item (Index);
         Index := Index_Type'Succ (Index);
      end loop;

      accept Get (Result : out Element_Type) do
         Result := Worker.Result;
      end Get;
   end Worker;

   Workers : array (Count_Type range 1 .. Count_Type (Task_Count)) of Worker;
   Result : Element_Type := Initial_Value;
```

```
47    Temp   : Element_Type;
48    Length : constant Count_Type := Item'Length;
49    Chunk  : constant Count_Type := Length / Count_Type (Task_Count);
50    Rest   : constant Count_Type := Length rem Count_Type (Task_Count);
51
52 begin
53    for Partition in Workers'Range loop
54       Workers (Partition).Start (
55         Index => Index_Type'Val (Index_Type'Pos (Item'First)
56                                  + (Partition - 1) * Chunk
57                                  + Count_Type'Min (Rest, Partition - 1)),
58         Count => Chunk + Count_Type'Min (Rest, Partition)
59                        - Count_Type'Min (Rest, Partition - 1));
60    end loop;
61
62    for Worker of Workers loop
63       Worker.Get (Temp);
64       Result := Result + Temp;
65    end loop;
66
67    return Result;
68 end Accumulate;
```

Sin embargo, la versión 2022 del estándar de Ada introduce el atributo `'Parallel_Reduce` y el bloque de control de flujo `parallel do`, que permiten simplificar la solución de este problema [28]. Sin embargo, a día de hoy ningún compilador de Ada ha implementado esta parte del estándar. Por ejemplo, el equipo que más contribuye al *front-end* de GCC de Ada (GNAT) dice que de momento van a pausar el desarrollo para dar soporte al paralelismo en GNAT, pues hay que tener en cuenta gran variedad de tecnologías actuales y habría que hacer cambios profundos la interfaz del lenguaje [29]. En el Listado 2.11 y el Listado 2.12 se muestran ejemplos de cómo se codificaría.

Listado 2.11

Implementación del Listado 2.10 de sumar los elementos de un vector, pero utilizando el nuevo atributo `'Parallel_Reduce`.

Ada

```
1  -- Specification
2  generic
3     type Element_Type is private;
4     type Index_Type is (<>);
5     type Array_Type is array (Index_Type range <>) of Element_Type;
6     Initial_Value : in Element_Type;
7     with function "+" (Left, Right : in Element_Type) return Element_Type is <>;
8  function Accumulate (Item : in Array_Type) return Element_Type with
9     Pure, Global => null;
10
```

```
11 -- Body
12 function Accumulate (
13    Item : in Array_Type)
14    return Element_Type is (
15    Item'Parallel_Reduce ("+", Initial_Value));
```

Listado 2.12

Ejemplo de los bloques de código `parallel for` y `parallel do` de Ada 2022.

```ada
1  with Ada.Numerics.Elementary_Functions, Ada.Text_IO;
2  use Ada.Numerics.Elementary_Functions, Ada.Text_IO;
3  procedure Program is
4     Arr : array (Positive range 1 .. 10_000) of Float;
5  begin
6     parallel do
7        Put_Line ("¡Hola mundo!");
8     and
9        Put_Line ("Hello, world!");
10    and
11       Put_Line ("Bonjour, le monde !");
12    and
13       Put_Line ("世界よ、こんにちは! ");
14    end do;
15
16    parallel for I in Arr'Range loop
17       Arr (I) := Sin (Float (I - 1) / Float (Arr'Length));
18    end loop;
19 end Program;
```

2.3.3.1.3. Python3

En Python3 existen dos módulos para trabajar con concurrencia, el primero es `threading` que es similiar a `std::thread` en C++ y a las `task` de Ada [30], [31]; y otro específico para construcciones paralelas llamado `multiprocessing` que utiliza procesos para paralelizar algoritmos [32]. Obsérve el Listado 2.13.

Listado 2.13

Ejemplo de paralelismo en Python 3 con `multiprocessing`. Implementación de la suma paralela de los elementos de un vector como Listado 2.7

```python
1  import multiprocessing
2  from typing import Final
3
4  def split[T] (lst : list[T], cores : int) -> list[list[T]]:
5    result : list[list[T]] = []
6    length : Final[int]    = len(lst)
```

```
7     chunk  : Final[int]    = length // cores
8     rem    : Final[int]    = length % cores
9     for i in range(cores):
10      slice = lst[i * chunk + min(i, rem): (i+1) * chunk + min(i+1, rem)]
11      result.append((slice,))
12    return result
13
14  def parallel_sum[T] (lst : list[T], cores : int) -> int:
15    with multiprocessing.Pool(cores) as p:
16      return sum(p.starmap(sum, split(lst, cores), cores))
```

2.3.3.2. OpenMP

OpenMP es una especificación para un juego de directivas de compilación, rutinas de biblioteca y variables de entorno que se pueden utilizar para especificar paralelismo de alto nivel en programas escritos en Fortran, C y C++ [33].

OpenMP es una de las especificaciones para hacer paralelismo más utilizadas, solo en GitHub en 2023 el 45% de los repositorios lo utilizan [34].

Listado 2.14

Implementación del Listado 2.7 con OpenMP en C++.

```C++
1  #include <omp.h>
2
3  template <class InputIt, typename T>
4  T accumulate (InputIt first, InputIt last, T init) {
5    #pragma omp parallel for reduction(+:init)
6    for (auto it = first; it != last; ++it) { init += *it; }
7    return init;
8  }
```

2.3.3.3. oneTBB

Intel® oneAPI Threading Building Blocks, o también conocido como opeTBB, es una biblioteca flexible para mejorar el rendimiento que facilita añadir paralelismo a aplicaciones complejas en multitud de arquitecturas aceleradas [35].

TBB (*Threading Building Blocks*) es una solución para escribir programas paralelos en C++. TBB fue introducido en 2006, así que tiene soporte para compiladores previos a C++ 11, aunque características que se encuentran a partir de C++ 11 como soporte para funciones lambda hace TBB mucho más fácil de comprender y utilizar [24].

Listado 2.15

Implementación del Listado 2.7 con oneTBB en C++.

```C++
1  #include <functional>
2  #include <oneapi/tbb.h>
```

```
3
4   template <class InputIt, typename T>
5   T accumulate (InputIt first, InputIt last, T init) {
6     return tbb::parallel_reduce(
7       tbb::blocked_range(range.begin(), range.end()),
8       init,
9       [&] (auto const & r, T acc) -> T {
10        for (auto x : r) { acc += x; }
11        return acc;
12      },
13      std::plus<T>{});
14  }
```

CAPÍTULO 3

ANÁLISIS

En este capítulo se da una descripción general del problema (Sección 3.1), los casos de uso (Sección 3.2), los requisitos del mismo (Sección 3.3) y el análisis de los mismos (Sección 3.4). Finalmente, se resume la arquitectura del sistema (Sección 3.5).

3.1. Planteamiento del problema

El objetivo del proyecto es implementar el algoritmo para la detección de ataques epilépticos que utiliza los patrones y las características de una señal encefalograma para clasificar segmentos (épocas) de una señal desarrollado en la Universidad de Málaga [2], de forma que ejecute en tiempo real en sistemas empotrados de bajo consumo energético, además de optimizar el código preexistente para reducir el tiempo de entrenamiento.

El proceso se separa en dos fases: una primera fase de entrenamiento, que genera un *batch* (modelo) con patrones relevantes e intervalos de los valores de las características de la señal que define (distancia máxima, energía y densidad espectral de potencia) que optimiza la puntuación F_1 (F_1 *score*) del modelo; y una segunda fase, en la que un paciente lleva un dispositivo empotrado conectado a un sensor de señales de encefalograma, que hace uso del modelo generado previamente para clasificar la época actual de la señal en «zona de ataque epiléptico» (positivo) o «zona libre de ataque epiléptico» (negativo).

Para el desarrollo se ha definido que un *stride* (número de muestras por segundo) son 256 muestras, que una época son cinco *strides* o 1280 muestras y que el *batch* tiene a lo sumo tres patrones con que compararlo. Para considerar que el sistema ejecute en tiempo real (todo aquel sistema capaz de garantizar una respuesta antes de un tiempo límite) debe ser capaz de procesar en el peor de los casos una época por segundo.

Existe ya una implementación [3] que solamente entrena el modelo. Además de implementar el *software* para el sistema empotrado, también hay que mejorar el tiempo

de entrenamiento de dicha implementación, que comparte la función de detección con el dispositivo empotrado, pero este solo la usa para optimizar el modelo.

3.2. Casos de uso

De acuerdo con Craig Larman, *Unified Process* define el modelo de casos de uso dentro de la disciplina de requisitos e insiste en que los casos de uso son documentos textuales, no diagramas, y que el modelado de casos de uso es principalmente un acto de escribir texto, no de dibujar diagramas [36].

El mismo autor define «actor» como todo aquello con comportamiento, eso incluye el propio sistema cuando este hace uso de otros servicios o sistemas. Los actores no solo son roles que interpretan personas, también organizaciones, *software* y máquinas. Hay tres tipos de actores externos: actor principal, actor de apoyo (*supporting actor*) y actor entre bastidores (*offstage actor*). [36]

En este proyecto solo tiene sentido de hablar de dos actores: que son el paciente, que quiere detectar sus propios ataques epilépticos; y el doctor, que puede ser un médico o un investigador que se encarga de entrenar el modelo para posteriormente detectar ataques epilépticos.

Tabla 3.1

Plantilla de «Caso de uso»

Campo	Descripción
Nombre	Nombre del caso de uso, es una acción así que debe empezar con un verbo.
Alcance	A qué subsistema o subsistemas específicos afecta el mismo. En este proyecto se diferencia la fase de entrenamiento del dispositivo empotrado con el sistema en tiempo real. (*Dispositivo empotrado*, *Entrenamiento*, *Dispositivo empotrado y entrenamiento*)
Nivel	«Meta de usuario» o «Subfunción». La subfunción se diferencia de la meta de usuario en que es un paso intermedio. (*Meta de usuario*, *Subfunción*)
Actores principales	Los agentes que hacen uso de los servicios del sistema para alcanzar su objetivo.
Parte interesada	Son los actores a quienes les concierne y qué quieren.
Precondiciones	Qué debe ser cierto al inicio.
Postcondiciones	Qué se garantiza que es cierto al completar la operación.
Escenario de éxito principal	La descrición de los pasos de las situación más típica.
Extensiones	Escenarios alternativos y ramificaciones del escenario principal.
Requisitos especiales	Requisitos no funcionales relacionados.
Frecuencia con que ocurre	Cada cuánto tiempo se espera que se utilice el caso de uso.

Tabla 3.2

Caso de uso «Leer señal de encefalograma»

Campo	Descripción
Nombre	Leer señal de encefalograma
Alcance	Dispositivo empotrado
Nivel	Subfunción
Actores principales	
Parte interesada	• Paciente: Quiere que la señal de encefalograma para detectar un posible ataque epiléptico sea continua y no se detenga bajo ningún concepto. • Doctor: Quiere poder utilizar tanto señales reales como señales pregrabadas como señales sintéticas para hacer estudios y entrenar el modelo.

Campo	Descripción
Precondiciones	El sensor tiene alimentación y se ha activado por *software*.
Postcondiciones	El resultado es un valor real que está dentro de un rango válido.
Escenario de éxito principal	1. El sensor mide un valor analógico en bruto en un instante. 2. El sensor convierte dicho valor a otro con las unidades esperadas. 3. El sensor escribe el valor en la señal.

Extensiones	1. Desconexión del sensor
	  1. El sensor notifica al paciente
	  2. El sensor escribe el valor máximo del rango válido, para no detener el sistema ni dejar esperando al resto de actores.
	2. Si el valor después de convertirlo fuera mayor que el límite superior del rango válido.
	  1. El sensor escribe un *log*.
	  2. El sensor escribe el valor máximo del rango válido y continúa con la operación.
	3. Si el valor después de convertirlo fuera menor que el límite inferior del rango válido.
	  1. El sensor escribe un *log*.
	  2. El sensor escribe el valor mínimo del rango válido y continúa con la operación.

Requisitos especiales	• Debe leer 256 muestras por segundo. • Debe leer a un ritmo constante. • Los valores leídos deben estar en un rango válido.
Frecuencia con que ocurre	Continuo

Tabla 3.3

Caso de uso «Detectar ataque»

Campo	Descripción
Nombre	Detectar ataque
Alcance	Dispositivo empotrado y entrenamiento
Nivel	Meta de usuario
Actores principales	Paciente
Parte interesada	• Detector: Quiere clasificar las distintas épocas de la señal para entrenar el modelo o hacer estudios.. • Paciente: Quiere saber si está teniendo un ataque epiléptico.
Precondiciones	El modelo debe estar cargado, y el sensor debe estar activo.
Postcondiciones	Ninguna

Campo	Descripción
Escenario de éxito principal	1. El detector espera a tener una época de señal del sensor. 2. El detector pasa un filtro de paso bajo y paso alto a la época. 3. El detector decide que la época no es un artefacto (pestañeo, ...). 4. El detector computa las características de la señal (`max_distance`, ...). 5. El detector computa la distancia entre la época y cada uno de los patrones. 6. Retorna que es un ataque epiléptico.

Extensiones	1. En el paso 3., si se trata de un artefacto.
	  1. Como es un artefacto, no hace falta continuar, retorna que no es un ataque.
	2. En el paso 4. si para alguna de las características el valor cae fuera de los rangos definidos por el modelo (*batch*).
	  1. No es un ataque epiléptico, retorna que no lo es.
	3. En el paso 5. si para ninguno de los patrones la distancia es lo suficientemente pequeña (determinada por el modelo).
	  1. No es un ataque epiléptico, lo retorna.

Requisitos especiales	• Debe procesar una época y compararla hasta con tres patrones por segundo.
Frecuencia con que ocurre	Cotinua (una vez por segundo)

Tabla 3.4

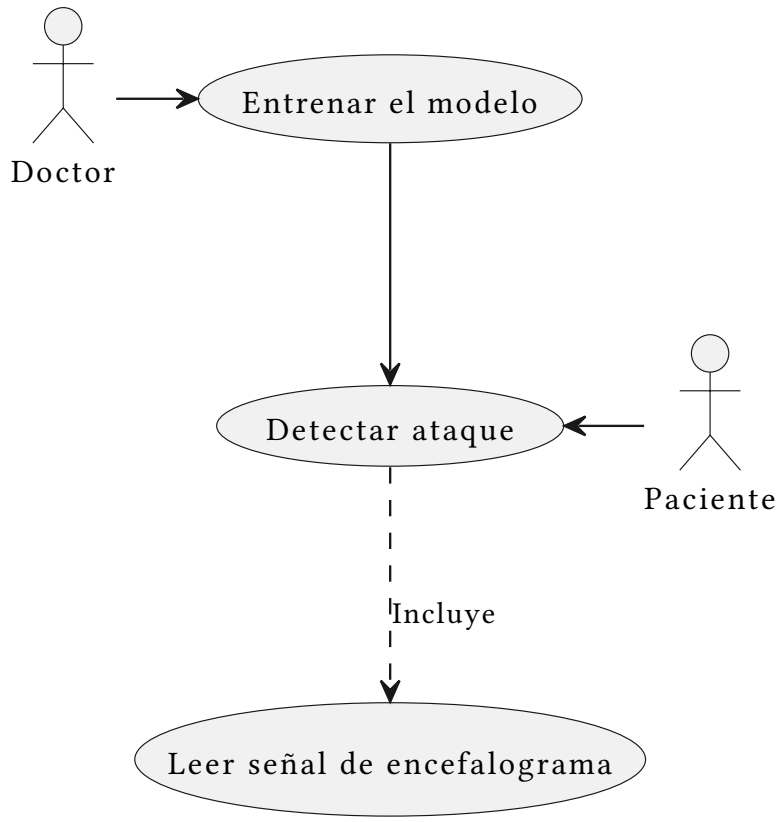
Caso de uso «Entrenar el modelo»

Campo	Descripción
Nombre	Entrenar el modelo
Alcance	Entrenamiento
Nivel	Meta de usuario
Actores principales	Doctor
Parte interesada	• Doctor: Quiere obtener el modelo y su eficacia. • Paciente: Quiere un modelo eficaz para detectar sus posibles ataques epilépticos.
Precondiciones	Una señal continua con los intervalos de ataque epiléptico etiquetados.
Postcondiciones	Un modelo válido con los rangos de características de la señal y los patrones de la propia señal que optimicen la puntuación F_1 del sistema. Además de la precisión, la sensibilidad y su puntuación F_1.

Campo	Descripción
Escenario de éxito principal	1. El doctor graba la señal de un paciente durante un periodo largo de tiempo. 2. El doctor etiqueta los intervalos de la propia señal en los que está ocurriendo un ataque. 3. El sensor pasa un filtro de paso bajo y paso alto a la señal. 4. El detector marca en la señal las secciones con artefactos. 5. El optimizador (externo) toma el detector y optimiza los parámetros del modelo para maximizar la puntuación F_1. 6. El optimizador retorna el modelo preparado en un archivo.
Extensiones	1. Puntuación F_1 baja. 1. El doctor estudiará por qué ocurre. 2. El doctor reconfigura el optimizador y vuelve a intentar generar el modelo.
Requisitos especiales	• La puntuación F_1 debe ser superior al 99%. • Debe haber una interfaz compatible con la implementación de referencia. • Los resultados deben asemejarse a los de la implementación de referencia.
Frecuencia con que ocurre	Escasa: el modelo se entrene una sola vez por paciente y tarda mucho en hacerlo

Figura 3.1

Modelo de casos de uso



3.3. Requisitos

En esta sección se provee la lista de requistios que se ha obtenido a partir de un análisis exhaustivo de los casos de uso, definidos previamente en la Sección 3.2. Los requisitos a continuación se clasifican en dos grandes categorías:

- **Requisitos funcionales**: se dice de aquellos requisitos que prescriben el comportamiento del sistema. Responden a la pregunta: *«¿qué debe hacer?»*.
- **Requisitos no funcionales**: se dice de aquellos requisitos que imponen restricciones sobre cómo debe implementarse el sistema. Responden a la pregunta: *«¿cómo debe hacerlo?»*.

Cada requisito se identifica de manera unequívoca con un identificador con el formato:

- **Requisitos funcionales**: `RF-XX`, donde `XX` es un valor numérico de dos cifras, que comienza en `01` y que crece monótonamente de uno en uno.
- **Requisitos no funcionales**: `RNF-XX`, donde `XX` es un valor numérico de dos cifras, que comienza en `01` y que crece monótonamente de uno en uno.

La Tabla 3.5 muestra el formato de los requisitos funcionales y la Tabla 3.11 de los no funcionales.

3.3.1. Requisitos funcionales

Tabla 3.5

Plantilla de «Requisito funcionales»

Campo	Descripción
Descripción	Descripción detallada del requisito
Necesidad	Prioridad del requisito para el usuario (*Alta*, *Media*, *Baja*)
Prioridad	Prioridad del requisito para el desarrollador (*Alta*, *Media*, *Baja*)
Estabilidad	Indica la variabilidad del requisito a lo largo del proceso de desarrollo. (*Constante*, *Inconstante*, *Inestable*)
Verificabilidad	Capacidad de probar la valided del requisito. (*Alta*, *Media*, *Baja*)
Fuente	Quién ha propuesto el requisito. (*Cliente*, *Analista*)
Orígenes	Caso de uso del que es origen este requisito.

Tabla 3.6

Requisito «RF-01»

Campo	Descripción
Descripción	Debe existir una interfaz con Python 3 (*binding*) en forma de módulo, que se integre con la base de código antigua y permita entrenar el modelo
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.7

Requisito «RF-02»

Campo	Descripción
Descripción	El sistema debe clasificar una época dada en ataque o no ataque.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Constante
Verificabilidad	Baja
Fuente	Cliente
Orígenes	Detectar ataque

Tabla 3.8

Requisito «RF-03»

Campo	Descripción
Descripción	El sistema debe notificar al paciente si está teniendo un ataque epiléptico.
Necesidad	Media
Prioridad	Alta
Estabilidad	Constante
Verificabilidad	Media
Fuente	Analista
Orígenes	Detectar ataque

Tabla 3.9

Requisito «RF-04»

Campo	Descripción
Descripción	El sistema debe leer señales de encefalograma de un lector de señales de encefalograma.
Necesidad	Baja
Prioridad	Baja
Estabilidad	Inconstante
Verificabilidad	Baja
Fuente	Analista
Orígenes	Leer señal de encefalograma

Tabla 3.10

Requisito «RF-05»

Campo	Descripción
Descripción	El sistema debe tener el mismo compotamiento que la implementación de referencia.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

3.3.2. Requisitos no funcionales

Tabla 3.11

Plantilla de «Requisito no funcional»

Campo	Descripción
Descripción	Descripción detallada del requisito
Necesidad	Prioridad del requisito para el usuario (*Alta*, *Media*, *Baja*)
Prioridad	Prioridad del requisito para el desarrollador (*Alta*, *Media*, *Baja*)
Estabilidad	Indica la variabilidad del requisito a lo largo del proceso de desarrollo. (*Constante*, *Inconstante*, *Inestable*)
Verificabilidad	Capacidad de probar la valided del requisito. (*Alta*, *Media*, *Baja*)
Fuente	Quién ha propuesto el requisito. (*Cliente*, *Analista*)
Orígenes	Caso de uso del que es origen este requisito.

Tabla 3.12

Requisito «RN-01»

Campo	Descripción
Descripción	El sensor de encefalograma en el dispositivo empotrado debe leer a razón de 256 muestras por segundo.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Inconstante
Verificabilidad	Media
Fuente	Cliente
Orígenes	Leer señal de encefalograma

Tabla 3.13

Requisito «RN-02»

Campo	Descripción
Descripción	El sensor debe leer muestras a un ritmo constante, es decir, el intervalo entre lecturas debe ser el mismo.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Inconstante
Verificabilidad	Media
Fuente	Analista
Orígenes	Leer señal de encefalograma

Tabla 3.14

Requisito «RN-03»

Campo	Descripción
Descripción	El detector de encefalograma deberá procesar como mínimo una época de 1280 muestras por segundo en el peor de los casos.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Inconstante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Detectar ataque

Tabla 3.15

Requisito «RN-04»

Campo	Descripción
Descripción	El sistema no puede terminar de forma abrupta bajo ningún concepto.
Necesidad	Baja
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Media
Fuente	Analista
Orígenes	Leer señal de encefalograma, Detectar ataque

Tabla 3.16

Requisito «RN-05»

Campo	Descripción
Descripción	La puntuación F_1 o F_1 *score* para un modelo entrenado debe ser superior al 99% y se calcula como: $F_1 = \frac{2 \cdot \mathrm{PRE} \cdot \mathrm{SEN}}{\mathrm{PRE} + \mathrm{SEN}}$, donde $\mathrm{SEN} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$ y $\mathrm{PRE} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$. Donde *TP* son los verdaderos positivos; *FN*, los falsos negativos; y *FP*, los falsos positivos.
Necesidad	Baja
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Baja
Fuente	Analista
Orígenes	Entrenar el modelo

Tabla 3.17

Requisito «RN-06»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 10^{-6} en la función `max_distance` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.18

Requisito «RN-07»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 10^{-6} en la función `energy` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.19

Requisito «RN-08»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 10^{-3} en la función `dtw` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.20

Requisito «RN-09»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 0.5 en la función `PSD 1` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.21

Requisito «RN-10»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 0.5 en la función `PSD 2` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.22

Requisito «RN-11»

Campo	Descripción
Descripción	Debe haber un error medio absoluto de máximo 0.5 en la función `PSD 3` con respecto a la implementación original en Python 3.
Necesidad	Media
Prioridad	Baja
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Entrenar el modelo

Tabla 3.23

Requisito «RN-12»

Campo	Descripción
Descripción	El sistema debe detectar más rápido un ataque epiléptico que la implementación de referencia, comparado en el mismo dispositivo.
Necesidad	Alta
Prioridad	Alta
Estabilidad	Constante
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Detectar ataque

Tabla 3.24

Requisito «RN-13»

Campo	Descripción
Descripción	El sistema debe funcionar en un dispositivo empotrado de bajo consumo. Preferiblemente con un procesador RISC-V como el ESP32C3 o el ESP32C6.
Necesidad	Alta
Prioridad	Media
Estabilidad	Inestable
Verificabilidad	Alta
Fuente	Cliente
Orígenes	Detectar ataque, Leer señal de encefalograma

3.4. Análisis de requisitos

Finalmente, las siguientes dos matrices de trazabilidad nos permiten ver la relación de cada requisito con cada caso de uso para determinar la cobertura y el grado de dependencia entre ambos. La primera (Tabla 3.25) relaciona los requisitos funcionales con los casos de uso, las segunda (Tabla 3.26) relaciona los requisitos no funcionales con los casos de uso. Se puede ver que todos los requisitos cubren todos los casos de uso, y no hay requisitos sin relación con ningún caso de uso.

Tabla 3.25

Matriz de Trazabilidad, Requisito funcionales vs Caso de uso

	Leer señal de encefalograma	Detectar ataque	Entrenar el modelo
RF-01			✓
RF-02		✓	
RF-03		✓	
RF-04	✓		
RF-05			✓

Tabla 3.26

Matriz de Trazabilidad, Requisito no funcional vs Caso de uso

	Leer señal de encefalograma	Detectar ataque	Entrenar el modelo
RN-01	✓		
RN-02	✓		
RN-03		✓	
RN-04	✓	✓	
RN-05			✓
RN-06			✓
RN-07			✓
RN-08			✓
RN-09			✓
RN-10			✓
RN-11			✓
RN-12		✓	
RN-13	✓	✓	

3.5. Arquitectura

La arquitectura se del sistema se puede separar en cuatro componentes principales:

- **Lector** (Tabla 3.31): que lee una señal de encefalograma de un sensor.
- **Detector** (Tabla 3.30): que utiliza un *batch* (o modelo) generado por el entrenador para clasificar la señal generada por el lector.
- **Entrenador** (Tabla 3.29): que genera un modelo a partir de los datos de un paciente.
- **Validador** (Tabla 3.28): que es utilizado por el entrenador para optimizar el modelo.

El diagrama UML se resume en la Figura 3.2, donde se pueden ver los distintos componentes y su relación entre ellos.

Tabla 3.27

Plantilla de «Componente»

Campo	Descripción
Nombre	La función del componente en el sistema.
Rol	La función del componente en el sistema.
Dependencias	Componentes que dependen de este.
Descripción	Explicación de la funcionalidad del componente.
Entrada	Datos de entrada del componente.
Salida	Datos de salida del componente.

Tabla 3.28

Componente «Validador»

Campo	Descripción
Nombre	Validador
Rol	Dado un modelo computa su puntuación F_1.
Dependencias	Detector
Descripción	Este componente lo que hace es ejecutar el modelo junto a un conjunto de prueba para obtener la puntuación F_1 que nos da cómo de bueno es el propio modelo.
Entrada	• Detección: Utiliza las detecciones generadas por el detector computar la puntuación F_1.
Salida	• Puntuación F_1 del modelo.

Tabla 3.29

Componente «Entrenador»

Campo	Descripción
Nombre	Entrenador
Rol	Entrenar el modelo para un paciente
Dependencias	Validador
Descripción	El componente genera múltiples modelos (*batches*) y los valida contra el validador, el cual utiliza para optimizar la puntuación F_1 del modelo resultante.
Entrada	• Los datos del paciente.
Salida	• Un *batch* o modelo.

Tabla 3.30

Componente «Detector»

Campo	Descripción
Nombre	Detector
Rol	Clasificar una época de señal.
Dependencias	Lector
Descripción	A partir de un modelo generado (*batch*), clasifica épocas de señal en dos categorías: «zona libre de ataque epiléptico» y en «zona con ataque epiléptico».
Entrada	• *Batch*: o modelo generado. • Señal: generada por el lector.
Salida	• Clasificación de cada época de la señal.

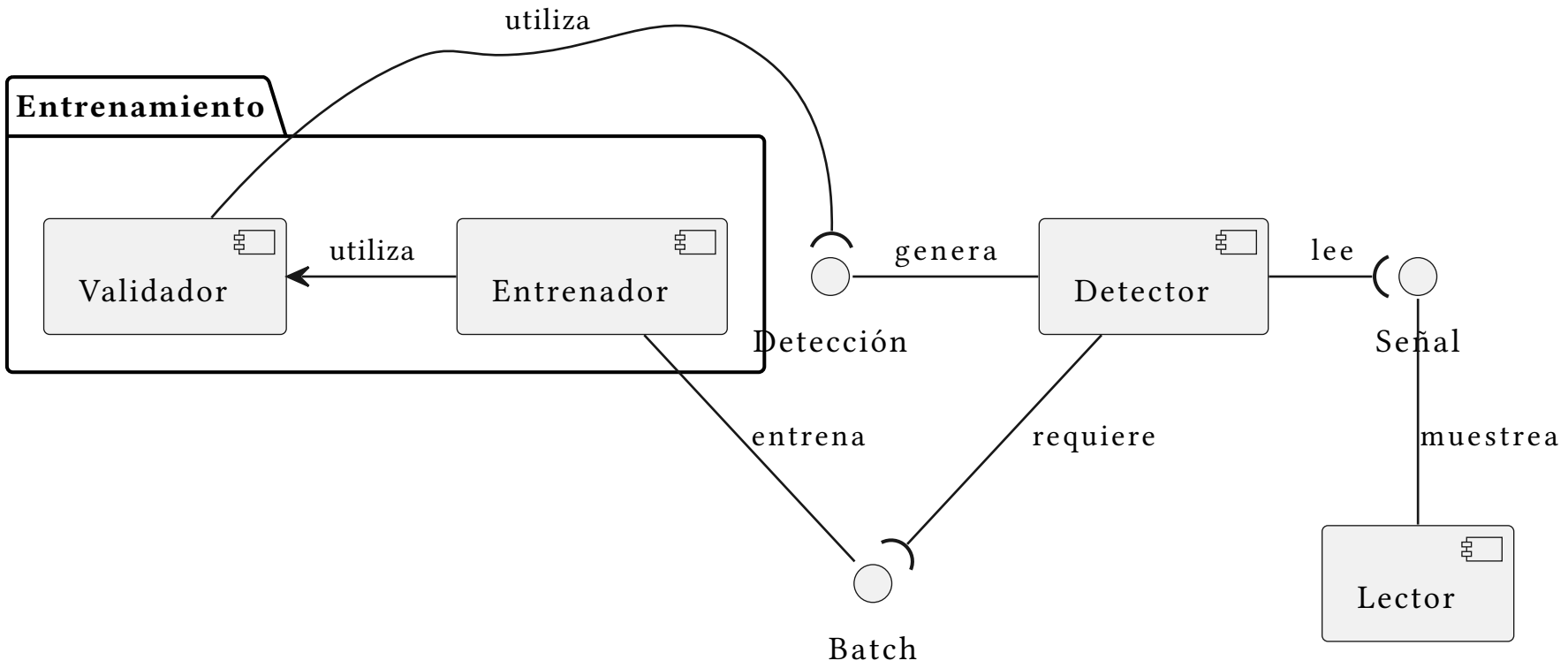
Tabla 3.31

Componente «Lector»

Campo	Descripción
Nombre	Lector
Rol	Leer muestras de encefalograma.
Dependencias	*Ninguna*
Descripción	Leer de manera continua, constante e ininterrumpida muestras de señal de encefalogramo de un sensor.
Entrada	*Ninguna*
Salida	• Señal leída.

Figura 3.2

Arquitectura



CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN

En este capítulo se explican las decisiones de diseño que se han tomado para implementar cada una de las partes que componen el algoritmo de detección de ataques epilépticos *Patterns Augmented by Features Epileptic Seizure Detection*. Se hace un análisis matemático de cada uno de los algoritmos que lo componen y se estudia cómo se puede mejorar y cómo se pueden solucionar los errores del mismo.

La parte de implementación se introduce con una lista de distintas convenciones matemáticas que se usan a lo largo del capítulo, se ofrece una descripción del algoritmo de manera global utilizando dichas convenciones, y termina con un análisis de todos los algoritmos que se consideran relevantes para el desarrollo del proyecto.

4.1. Estudio de la solución final

La implementación original del algoritmo de detección de ataques epilépticos que se analiza en este proyecto (PaFESD) [3] se escribió en Python 3.10. En este trabajo se analiza el impacto en el rendimiento al utilizar otros lenguajes de programación, compiladores, técnicas y herramientas, de cara a un análisis en el impacto energético y de seguridad del programa.

Que la implementación original estuviera escrita en Python dificulta bastante la compilación cruzada. Esto es especialmente difícil para dispositivos empotrados como la ESP32C3, que tiene pocos megabytes de memoria disponibles para almacenar el ejecutable, pues el tamaño de la suma de todas las dependencias superaba los cientos de megabytes. Además uno de los requisitos fundamentales del proyecto es que corriera en tiempo real. Por estas razones técnicas se ha optado por utilizar otros lenguajes de programación.

Los lenguajes usados y las representaciones de valores numéricos utilizadas son los siguientes: C++ 20 con punto flotante de simple precisión, C++ 20 con punto flotante de

doble precisión, Ada 2022 con punto flotante de simple precisión, Ada 2022 con punto flotante de doble precisión, SPARK 2014 con punto fijo y Python 3.10 con punto flotante.

Para la implementación solo se considera la parte del algoritmo de validación del artículo original; es decir, se supone que se ha pasado un filtro de paso bajo y algo a la señal de entrada, y que se han marcado previamente los segundos de la señal que contienen artefactos (pestañeos, errores de medición, etcétera).

4.2. Convenciones matemáticas

Las siguientes convenciones matemáticas solo se utilizan en este capítulo y permiten desambiguar algunos conjuntos como $\mathbb{N}$. Los conjuntos clásicos se denotan de la siguiente manera:

- $\mathbb{B} = \{\bot, \top\}$ es el conjunto de los valores lógicos falso ($\bot$) y verdadero ($\top$).
- $\mathbb{Z} = \{0, 1, -1, 2, -2, ...\}$ es el conjunto de los números enteros.
- $\mathbb{N} = \{0, 1, 2, ...\}$ es el conjunto de los números enteros no negativos.
- $\mathbb{N}^+ = \{1, 2, 3, ...\}$ es el conjunto de los números enteros positivos.
- $\mathbb{Q}$ es el conjunto de los números racionales.
- $\mathbb{R}$ es el conjunto de los números reales.
- $\mathbb{C}$ es el conjunto de los números complejos y $j = \sqrt{-1}$ es la unidad imaginaria.

Como la implementación en el computador se trabaja con números con un tamaño fijo de bits, a continuación se definen ciertos conjuntos interesantes para ello. Se supone que los números enteros y en punto fijo se codifican en complemento a dos:

- $\mathbb{I}_n = \mathbb{Z} \cap \left[-2^{n-1}, 2^{n-1} - 1\right], n \in \mathbb{Z}, n > 1$ es el conjunto de enteros de computador de n bits codificados en complemento a dos.
- $\mathbb{M}_n = \mathbb{Z}/2^n\mathbb{Z}$ es el anillo de enteros módulo 2^n.
- $\mathbb{X}_{b,f} = \left\{2^f x : x \in \mathbb{I}_b\right\}, f \in \mathbb{Z}$ es el conjunto de números racionales en punto fijo binario de b bits multiplicados por 2^f. Nótese que $\mathbb{X}_{b,0} = \mathbb{I}_b$. Se denota además el valor más pequeño no nulo en valor absoluto del conjunto como $\delta_{b,f} = 2^{-f}$.
- $\mathcal{F}_{32}$ es el conjunto de valores codificados en el estándar de coma flotante IEEE 754 de simple precisión.
- $\mathcal{F}_{64}$ es el conjunto de valores codificados en el estándar de coma flotante IEEE 754 de doble precisión.

Un vector $v \in S^n$ es una secuencia de n elementos de un conjunto S cualquiera: $v = (v_1, v_2, ..., v_n)$. Se denota por $v(i)$ o v_{i-1} el i-ésimo elemento del vector v; donde v_0 o $v(1)$ es el primer elemento y v_{n-1} o $v(n)$ es el último, excepto cuando se diga lo contrario. No están definidos los elementos $v(k), k \leq 0 \vee k > n$.

$$S^1 = S$$

$$S^n = S^{n-1} \times S$$

Además la notación S^+, inspirada por la clausura de Kleene, denota:

$$S^+ = \bigcup_{i \in \mathbb{N}^+} S^i$$

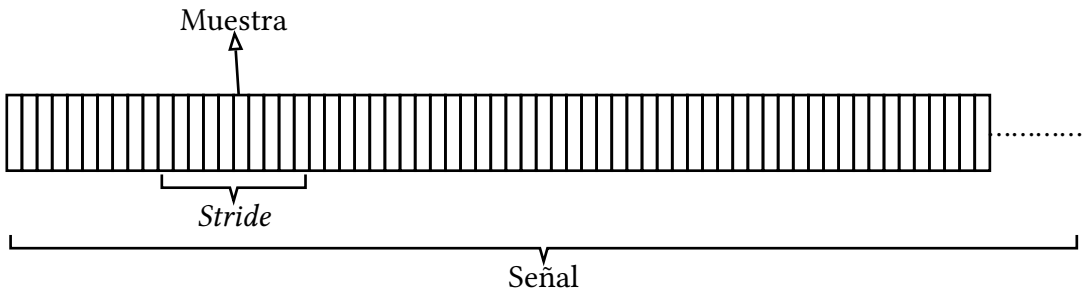
Además, si se se define $S^0 \neq \{\}$, se puede extender la notación de $S^* = S^0 \cup S^+$. Estas notaciones son útiles para definir vectores.

Una muestra $m \in \mathbb{R}$ es un valor que ha sido leido por un sensor de encefalograma en un instante de tiempo, el valor puede ser tanto negativo, como positivo, como nulo. Una señal $S_r(t), S : \mathbb{R} \to \mathbb{R}$ es una función que asocia para un sensor cualquiera r en un instante de tiempo t una muestra.

Un sensor r lee a una razón de $s \in \mathbb{N}^+$ muestras por segundo, a esa constante se denomina *stride*. Así que en vez de trabajar sobre una señal continua, se trabaja sobre una señal discreta como se muestra en la Figura 4.1.

Figura 4.1

Señales, *strides*, y muestras



A la hora de hacer el análisis se trabaja por épocas (*epochs*) en vez de *strides*. Si *stride* es una secuencia contigua de muestras, una época es una secuencia contigua de *strides*. Como referencia, se ha utilizado un *stride* de $s = 256$ muestras/s, épocas de 5 segundos (1280 muestras) y los valores de las muestras suelen estar en el rango -10000 a 10000.

4.2.1. Conversión entre tipos en punto fijo binario

Definición 4.2.1 ($\text{conv}_f(x)$)

Conversión de punto fijo: Dado un valor de punto fijo $x = 2^f k$ con $x \in \mathbb{X}_{b,f}$ y con $k \in \mathbb{I}_b$, se define la operación de conversión al $x' = k' 2^{f'} \in \mathbb{Q}$ que aproxima el valor x en supuesto punto fijo que está multiplicado por el factor $2^{f'}$.

Sea $x' = 2^{f'} k$ con $x' \in \mathbb{Q}$ y con $k' \in \mathbb{Z}$ el valor al que se quiere convertir x. Se denota como $\text{conv}_{f'}(x) = x'$ donde $\text{conv}_{f'} : \mathbb{X}_{b,n} \to \mathbb{Q}$ y donde:

$$k' = \begin{cases} \lfloor 2^{f-f'} k \rfloor, & \text{si } k \geq 0 \\ \lceil 2^{f-f'} k \rceil, & \text{si } k < 0 \end{cases}$$

Porque $2^f k = 2^{f'} \left(\frac{2^f}{2^{f'}} k \right)$ y k' aproxima dicho valor.

Nota

Como dice más adelante el Teorema 4.2.5, si se cumple que $b' \geq b + (f - f')$ se puede demostrar que si $x \in \mathbb{X}_{b,f}$ entonces se cumple que $\text{conv}_{f'}(x) \in \mathbb{X}_{b',f'}$. La función original no depende del número de bits del tipo de punto fijo de retorno, pues para las demostraciones es necesario trabajar sobre $\mathbb{Q}$ y luego ya se puede añadir la restricción de que pertenzca o no a $\mathbb{X}_{b',f'}$.

Lema 4.2.2

$\lfloor x \rfloor = n$ si y solo si $n \leq x < n + 1$

Corolario 4.2.2.1

Si $b > 0$ entonces, $a - b < b\left\lfloor \frac{a}{b} \right\rfloor \leq a$.

Demostración.

Si $\left\lfloor \frac{a}{b} \right\rfloor = x$ entonces de acuerdo con el Lema 4.2.2:

$$x \leq \frac{a}{b} < x + 1$$

$$bx \leq b\frac{a}{b} < b(x + 1)$$

$$bx \leq a < b(x + 1)$$

$$b\left\lfloor \frac{a}{b} \right\rfloor \leq a < b\left(\left\lfloor \frac{a}{b} \right\rfloor + 1\right)$$

$$b\left\lfloor \frac{a}{b} \right\rfloor \leq a < b\left\lfloor \frac{a}{b} \right\rfloor + b$$

$$a - b < b\left\lfloor \frac{a}{b} \right\rfloor$$

$$a - b < b\left\lfloor \frac{a}{b} \right\rfloor \leq a$$

□

Lema 4.2.3

$\lceil x \rceil = n$ si y solo si $n - 1 < x \leq n$

Corolario 4.2.3.1

Si $b > 0$ entonces, $a \leq b\left\lceil \frac{a}{b} \right\rceil < a + b$

Demostración.

Si $\left\lceil \frac{a}{b} \right\rceil = x$ entonces de acuerdo con el Lema 4.2.3:

$$x - 1 < \frac{a}{b} \leq x$$

$$b(x-1) < a \leq bx$$

$$b\left(\left\lceil \frac{a}{b} \right\rceil - 1\right) < a \leq b\left\lceil \frac{a}{b} \right\rceil$$

$$b\left\lceil \frac{a}{b} \right\rceil - b < a \Rightarrow b\left\lceil \frac{a}{b} \right\rceil < a + b$$

$$a \leq b\left\lceil \frac{a}{b} \right\rceil < a + b$$

□

Teorema 4.2.4

Dados $x \in \mathbb{X}_{b,f}$ y $x' = \text{conv}_{f'}(x) \in \mathbb{Q}$. Si $f \geq f'$ entonces, $x = x'$. Si $f < f'$ y $k \geq 0$ se cumple que $0 \leq x - x' < 2^{f'}$, pero si $k < 0$ se cumple que $0 \leq x' - x < 2^{f'}$.

Demostración.

Dados $x = 2^f k$ y $x' = 2^{f'} k'$, donde $x' = \text{conv}_{f'}(x)$. El valor del suelo o techo depende del valor de la diferencia $f - f'$.

- Si $f = f'$ y $k \geq 0$, entonces $k' = \left\lfloor 2^{f-f'} k \right\rfloor = \lfloor k \rfloor = k$, porque $k \in \mathbb{I}_b$. Luego $x = x'$ y $x - x' = 0$.
- Si $f = f'$ y $k < 0$, entonces $k' = \left\lceil 2^{f-f'} k \right\rceil = \lceil k \rceil = k$, porque $k \in \mathbb{I}_b$. Luego $x = x'$ y $x - x' = 0$.
- Si $f > f'$ y $k \geq 0$, entonces $k' = \left\lfloor 2^{f-f'} k \right\rfloor = 2^{f-f'} k$. Luego $x' = 2^{f'} k' = 2^{f'} 2^{f-f'} k = 2^f k = x$, por lo que $x - x' = 0$.
- Si $f > f'$ y $k < 0$, entonces $k' = \left\lceil 2^{f-f'} k \right\rceil = 2^{f-f'} k$. Luego (como para $f > f', k \geq 0$) $x - x' = 0$.
- Si $f < f'$ y $k \geq 0$, como $2^{f'} k' = 2^{f'} \left\lfloor \frac{2^f k}{2^{f'}} \right\rfloor$, de acuerdo con el Corolario 4.2.2.1:

$$2^f k - 2^{f'} < 2^{f'} \left\lfloor \frac{2^f k}{2^{f'}} \right\rfloor \leq 2^f k$$

$$x - 2^{f'} < x' \leq x$$

$$-2^{f'} < x' - x \leq 0$$

$$0 \leq x - x' < 2^{f'}$$

- Si $f < f'$ y $k < 0$, como $2^{f'} k' = 2^{f'} \left\lceil \frac{2^f k}{2^{f'}} \right\rceil$, de acuerdo con el Corolario 4.2.3.1:

$$2^f k \leq 2^{f'} \left\lfloor \frac{2^f k}{2^{f'}} \right\rfloor < 2^f k + 2^{f'}$$

$$x \leq x' < x + 2^{f'}$$

$$0 \leq x' - x < 2^{f'}$$

□

Del Teorema 4.2.4 se obtienen propiedades muy relevantes. En primer lugar, si se convierte de un punto fijo multiplicado por un coeficiente más grande a uno con uno más pequeño, no hay error y por tanto no se pierde información. Por ejemplo: $a = 1 \cdot 2^{-2} \in \mathbb{X}_{32,-2}$, al convertirlo a un punto fijo con un bit más de precisión en la parte fraccionaria como $\text{conv}_{32,-3}(a)$ se puede representar perfectamente como $2 \cdot 2^{-3}$, pues el numerador debe ser entero.

Cuando $f < f'$, es decir, hay más bits en la parte fraccionaria del tipo de origen que el de destino, hay un error (que está acotado) en el rango $\left[0, 2^{-f'}\right)$. De vuelta al ejemplo anterior, al convertirlo a uno con menos bits en el denominador como $\text{conv}_{32,-1}$, no hay forma de representar el valor $\frac{1}{4}$ con un numerador entero y el denominador 2. Así que la conversión da $0 \cdot 2^{-1}$, que está a como mucho 2^{-1} unidades del valor real.

Otras propiedades importantes son que el número de bits del tipo de punto fijo no influye en la conversión ni en el error. Además el error solamente depende del exponente f' del coeficiente del tipo al que se convierte, no depende de cuál era el exponente f del coeficiente del valor origen. Esto simplifica considerablemente el análisis del error.

Corolario 4.2.4.1

Dado $x \in \mathbb{X}_{b,f}$, su valor convertido $\text{conv}_{b,f}$ en valor absoluto no puede ser mayor que el valor absoluto de x. Es decir, si $x \geq 0$ entonces, $0 \leq \text{conv}_{b,f}(x) \leq x$; y si $x < 0$ entonces se cumple que su valor convertido también es un punto fijo: $0 \geq \text{conv}_{b,f}(x) \geq x$.

Teorema 4.2.5

Si $b' \geq b + (f - f')$ entonces, dado un $x \in \mathbb{X}_{b,f}$, se tiene que $\text{conv}_{b',f'}(x) \in \mathbb{X}_{b',f'}$

Demostración.

Dado el conjunto $\mathbb{X}_{b,f}$, el valor máximo del conjunto es $M = 2^f 2^{b-1} - \delta_{b,f}$ y el valor mínimo es $m = -2^f 2^{b-1}$. La conversión se hace a $\mathbb{X}_{b',f'}$:

$$m' = \text{conv}_{b',f'}(m) = \left\lceil -2^{f-f'} 2^{b-1} \right\rceil 2^{f'}$$

$$M' = \text{conv}_{b',f'}(M) = \left\lfloor 2^{f-f'} \left(2^{b-1} - \delta_{b,f}\right) \right\rfloor 2^{f'}$$

Para m', si $f \geq f'$ de acuerdo con el Teorema 4.2.4, $m' = m$, entonces solo se puede representar m' en $\mathbb{X}_{b',f'}$ si $f - f' + b - 1 \leq b' - 1$ pues el valor mínimo de dicho conjunto es $-2^{b'-1}$. Se necesita que si $f \geq f'$ entonces $b' \geq b + (f - f')$.

Si por el contrario $f < f'$, se deben encontrar los valores que satisfagan que $m' \geq -2^{b'-1}$. Por el Corolario 4.2.4.1 se sabe que $0 \geq m' \geq m$, luego el valor de la conversión no es positivo (así que el límite superior se puede ignorar) y no es menor que el valor original m. Luego la restricción debe ser que $m \leq m' \Rightarrow 2^{b-1}2^f \leq 2^{b'-1}2^{f'}$. Luego $b - 1 + f \leq b' - 1 + f' \Rightarrow b' \geq b + (f - f')$.

Para M', si $f \geq f'$, de acuerdo con el Teorema 4.2.4, $M' = M$, entonces solo se puede representar M' en $\mathbb{X}_{b',f'}$ si

$$\begin{aligned} M &= \left\lfloor 2^{f-f'}\left(2^{b-1} - \delta_{b,f}\right)\right\rfloor 2^{f'} \\ &= \left\lfloor 2^{f-f'+b-1} - 2^{f'}\right\rfloor 2^{f'} \\ &= 2^{f+b-1} - 2^{f'} \\ &\leq \left(2^{b'-1} - 1\right)2^{f'} \\ &= 2^{f'+b'-1} - 2^{f'} \end{aligned}$$

$$\Rightarrow f + b - 1 \leq f' + b' - 1 \Rightarrow b' \geq b + (f - f')$$

Finalmente, si $f < f'$, de acuerdo con el Corolario 4.2.4.1 $0 \leq M' \leq M$. Se trabaja con valores no negativos y, como se vio arriba, la condición más fuerte de que $b' \geq b + (f - f')$ se sigue manteniendo. □

4.2.2. Subconjunto uniforme de $\mathbb{X}_{b,1-b}$

Para los siguientes algoritmos resulta bastante útil trabajar con un punto fijo que esté en el rango $(-1, 1)$, pues se utilizan muchas multiplicaciones. Además, dado que hay un valor negativo más que valores positivos también añade complicaciones. Así que se define $\mathbb{U}_b$ como el subconjunto uniforme de $\mathbb{X}_{b,1-b}$ o como conjunto uniforme de b bits:

$$\mathbb{U}_b = \mathbb{X}_{b,1-b} \setminus \{-1\}$$

Lema 4.2.6

$|x| < 1, \forall x \in \mathbb{U}_b \forall b > 1$

Teorema 4.2.7

Dados $x = p2^{1-b}$ con $x \in \mathbb{U}_b$ e $y = q2^f$ con $y \in \mathbb{U}_{b'}$, se cumple que su producto $xy = p2^{1-b}q2^{1-b'}$ también es uniforme $xy \in \mathbb{U}_{b+b'-1}$.

Demostración.

Dados $x = p2^{1-b}$ con $x \in \mathbb{U}_b$ e $y = q2^f$ con $y \in \mathbb{U}_{b'}$. Por definición $p \in \mathbb{I}_b \setminus \{-2^{b-1}\}$ y $q \in \mathbb{I}_{b'} \setminus \left\{-2^{b'-1}\right\}$. O que es lo mismo $|p| < 2^{b-1}$ y $|q| < 2^{b'-1}$. Eso quiere decir que:

$$|pq| \leq \left(2^{b'-1} - 1\right)\left(2^{b-1} - 1\right) < 2^{b-1}2^{b'-1} = 2^{b+b'-2}$$

Luego $pq \in \mathbb{I}_{b+b'-1}$. Por definición $\mathbb{X}_{b+b'-1,2-b-b'}$ es el conjunto de la forma: $\left\{k2^{2-b-b'} : k \in \mathbb{I}_{b+b'-1}\right\}$. Ya hemos visto que $pq \in \mathbb{I}_{b+b'-1}$, luego

$$xy = pq2^{2-b-b'} \in \mathbb{X}_{b+b'-1,2-b-b'}$$

La única diferencia entre el conjunto $\mathbb{X}_{b+b'-1,2-b-b'}$ y el conjunto $\mathbb{U}_{b+b'-1}$ es que $-1 \in \mathbb{X}_{b+b'-1,2-b-b'}$ (cuando tiene la forma $-2^{b+b'-1-1}2^{2-b-b'} = -1$) y que $-1 \notin \mathbb{U}_{b+b'-1}$.

Suponiendo que existiera un $x \in \mathbb{U}_b$ y un $y \in \mathbb{U}_{b'}$ de forma que $xy = -1$, implicaría que $\exists p \in \mathbb{I}_b$, $\exists q \in \mathbb{I}_{b'}$ de forma que $pq = -2^{b+b'-2}$ para que $xy = pq2^{2-b-b'} = -2^{b+b'-2}2^{2-b-b'} = -1$. Sin embargo, $\nexists p \in \mathbb{I}_b, \nexists q \in \mathbb{I}_{b'}$ de forma que $pq = -2^{b+b'-2}$, porque contradice que $|pq| < 2^{b+b'-2}$. Luego $\nexists x \in \mathbb{U}_b, \nexists y \in \mathbb{U}_{b'}$ tales que $xy = -1$. Y por ende:

$$xy \in \mathbb{X}_{b+b'-1,2-b-b'} \setminus \{-1\}$$

$$xy \in \mathbb{U}_{b+b'-1}$$

□

Lema 4.2.8 (Producto de enteros de computador)

Dados dos enteros $x \in \mathbb{I}_b$ e $y \in \mathbb{I}_{b'}$. Su producto es un entero de $b + b' - 1$ bits: $xy \in \mathbb{I}_{b+b'-1}$.

Teorema 4.2.9 (Producto de punto fijo)

Dados $x = p2^f$ con $x \in \mathbb{X}_{b,f}$ e $y = q2^{f'}$ con $y \in \mathbb{X}_{b',f'}$, su producto $xy = pq2^{f+f'}$ es otro punto fijo: $xy \in \mathbb{X}_{b+b'-1,f+f'}$.

Demostración.

Dados dos conjuntos de punto fijo $\mathbb{X}_{b,f}$ y $\mathbb{X}_{b',f'}$. Por definición:

$$\mathbb{X}_{b,f} = \left\{k2^f : k \in \mathbb{I}_b\right\}$$

$$\mathbb{X}_{b',f'} = \left\{k2^{f'} : k \in \mathbb{I}_{b'}\right\}$$

Sea P el conjunto que contiene todos los posibles productos entre los elementos del primer conjunto por los elementos del segundo:

$$\begin{aligned}
P &= \left\{xy : x \in \mathbb{X}_{b,f}, y \in \mathbb{X}_{b',f'}\right\} \\
&= \left\{p2^f y : p \in \mathbb{I}_b, y \in \mathbb{X}_{b',f'}\right\} \\
&= \left\{p2^f q2^{f'} : p \in \mathbb{I}_b, q \in \mathbb{I}_{b'}\right\} \\
&= \left\{pq2^{f+f'} : p \in \mathbb{I}_b, q \in \mathbb{I}_{b'}\right\} \\
&\subseteq \left\{k2^{f+f'} : k \in \mathbb{I}_{b+b'-1}\right\} [k := pq, \text{ Lema 4.2.8}] \\
&= \mathbb{X}_{b+b'-1,f+f'}
\end{aligned}$$

Como $xy \in P \subseteq \mathbb{X}_{b+b'-1,f+f'}$, se puede afirmar que $xy \in \mathbb{X}_{b+b'-1,f+f'}$. □

Definición 4.2.10 (Conversión del producto)

Dados $x \in \mathbb{X}_{b,f}$ e $y \in \mathbb{X}_{b',f'}$, se denomina «conversión del producto de ambos en otro punto fijo» a la conversión ($\text{conv}_{b,f}$) del producto de x e y, y se denota como

$$x *_{b'',f''} y = \text{conv}_{b'',f''}(xy)$$

Por otro lado, si $x \in \mathbb{U}_b$ e $y \in \mathbb{U}_{b'}$, denotaremos

$$x *_{b''} y = \text{conv}_{b'',1-b''}(xy)$$

a la conversión del producto de ambos en otro punto fijo, pero uniforme.

Teorema 4.2.11 (Conversión del producto de uniformes es uniforme)

Si $x \in \mathbb{U}_b$ e $y \in \mathbb{U}_{b'}$ entonces, $x *_{b''} y \in \mathbb{U}_{b''}$.

Demostración.

Dados $x \in \mathbb{U}_b$ e $y \in \mathbb{U}_{b'}$. Sea su producto $z = xy$, que según el Teorema 4.2.7 se sabe que $z \in \mathbb{U}_{b+b'-1}$, se aplica el Teorema 4.2.5 con:

- $b := b + b' - 1$
- $f := 1 - (b + b' - 1) = 2 - b - b'$
- $b' := b''$
- $f' := 1 - b''$

Si es cierto que:

$$\begin{aligned}
b'' &\geq (b + b' - 1) + [(2 - b - b') - (1 - b'')] \\
&= b + b' - 1 + 2 - b - b' - 1 + b'' \\
&= b''
\end{aligned}$$

(que siempre es cierto), significaba que $z \in \mathbb{X}_{b+b'-1,2-b-b'}$, luego su conversión $\text{conv}_{b'',1-b''}(z) \in \mathbb{X}_{b'',1-b''}$, es decir, $x *_{b''} y \in \mathbb{X}_{b'',1-b''}$. Lo único que es necesario determinar para terminar de demostrar este teorema es que $\nexists x \in \mathbb{U}_b, \nexists y \in \mathbb{U}_{b'}$ tales que $x *_{b''} y = -1$.

Supongamos que sí $\exists x = p2^{1-b} \in \mathbb{U}_b$ y sí $\exists y = q2^{1-b'} \in \mathbb{U}_{b'}$, para los que $z = x *_{b''} y = -1$. Según la Definición 4.2.1, $z = k2^{1-b''}$, donde k es:

$$k = \begin{cases} \lfloor 2^{(2-b-b')-(1-b'')} pq \rfloor, & \text{si } pq \geq 0 \\ \lceil 2^{(2-b-b')-(1-b'')} pq \rceil, & \text{si } pq < 0 \end{cases}$$

Se busca una k para que $z = -1 = k2^{1-b''}$ implique que $k = -2^{b''-1} \in \mathbb{I}_{b''}$. Como $k < 0 \Rightarrow pq < 0$.

$$k = \left\lceil 2^{(2-b-b')-(1-b'')} pq \right\rceil = -2^{b''-1}$$

Como $|p| < 2^{b-1}$ y $|q| < 2^{b'-1}$, $|pq| < 2^{b+b'-2}$. Y como $pq < 0$ entonces $0 > pq > -2^{b+b'-2}$.-

$$\begin{aligned} 0 &> pq > -2^{b+b'-2} \\ 0 &> 2^{(2-b-b')-(1-b'')} pq > 2^{(2-b-b')-(1-b'')} \left(-2^{b+b'-2}\right) \\ 0 &> 2^{(2-b-b')-(1-b'')} pq > -2^{b''-1} \end{aligned}$$

Se sabe que $\left\lceil -2^{b''-1} \right\rceil = -2^{b''-1}$, porque $b'' > 1, b'' \in \mathbb{N}$. Según el Lema 4.2.3:

$$\left\lceil -2^{b''-1} \right\rceil = -2^{b''-1} \Leftrightarrow -2^{b''-1} - 1 < -2^{b''-1} \leq -2^{b''-1}$$

Como $-2^{b''-1} < 2^{(2-b-b')-(b''-1)} pq$, se concluye que $\left\lceil 2^{(2-b-b')-(b''-1)} pq \right\rceil > -2^{b''-1}$. Luego $\nexists x \in \mathbb{U}_b, \nexists y \in \mathbb{U}_{b'}, x *_{b''} y = -1$, y por tanto: $x *_{b''} y \in \mathbb{U}_{b''}$. □

4.2.3. Uniformización de un vector $\mathbb{X}^n_{b,f}, n \in \mathbb{N}^+$

Antes de definir en qué consiste uniformizar un vector, es necesario definir la división para el punto fijo. La división de dos números racionales: $x = \frac{a}{b}$ e $y = \frac{p}{q}$, $x, y \in \mathbb{Q}$ y $a, b, p, q \in \mathbb{Q}$, es el siguiente número racional:

$$\frac{x}{y} = \frac{\frac{a}{b}}{\frac{p}{q}} = \frac{aq}{bp} \in \mathbb{Q}$$

Trabajar con números en punto fijo es similar, dados dos números en punto fijo $x = p2^f, x \in \mathbb{X}_{b,f}$ e $y = q2^{f'} \in \mathbb{X}_{b',f'}$. Su cociente también es un número en punto fijo:

$$\frac{x}{y} = \frac{p2^f}{q2^{f'}} = \frac{p}{q} 2^{f-f'}$$

Sigue pareciendo un número en punto fijo multiplicado por $2^{f-f'}$, la única diferencia es que $\exists p \exists q, \frac{p}{q} \notin \mathbb{Z}$, lo cual complica bastante tratarlo como un punto fijo.

Definición 4.2.12 (División de punto fijo)

Dados dos números en punto fijo $x = p2^f \in \mathbb{X}_{b,f}$ e $y = q2^{f'} \in \mathbb{X}_{b,f'}$ con $\boldsymbol{y \neq 0}$, se define la división de punto fijo de x entre y y se denota como $x \div y$ a:

$$x \div y = \begin{cases} \left\lfloor \frac{p}{q} \right\rfloor 2^{f-f'} & \text{, si } \frac{p}{q} \geq 0 \\ \left\lceil \frac{p}{q} \right\rceil 2^{f-f'} & \text{, si } \frac{p}{q} < 0 \end{cases} \in \mathbb{Q}$$

Teorema 4.2.13

Dados dos números en punto fijo $x = p2^f \in \mathbb{X}_{b,f}$ e $y = q2^{f'} \in \mathbb{X}_{b,f'}$, entonces:

$$x \div y \in \begin{cases} \mathbb{X}_{b,(f-f')} & \text{, si } x \neq (2^{b-1}-1)\delta_{b,f} \text{ e } y \neq -\delta_{b',f'} \\ \mathbb{X}_{b+1,(f-f')} & \text{, si no} \end{cases}$$

Demostración.

Sean $x = p2^f \in \mathbb{X}_{b,f}$ e $y = q2^{f'} \in \mathbb{X}_{b',f'}$ con $y \neq 0$, por definición $p \in \mathbb{I}_b$ y $q \in \mathbb{I}_{b'}$. Sea $z = x \div y$.

- Si $\frac{p}{q} = 0$, entonces $x \div y = \left\lfloor \frac{p}{q} \right\rfloor 2^{f-f'} = 0$.
- Si $\frac{p}{q} > 0$, entonces $x \div y = \left\lfloor \frac{p}{q} \right\rfloor 2^{f-f'}$. Es preciso ver que $\left|\frac{a}{b}\right| < \frac{|a|}{|b|+1}, \forall |b| > 0, \forall a$, es decir, cuando aumenta el denominador en valor absoluto, el valor absoluto del cociente es menor. Lo que nos indica que la función $\left|\frac{a}{b}\right|$ se maximiza cuando $|b| = 1$ y decrece monótonamente a medida de que $|b|$ aumenta.
  - Si $0 < q \leq 2^{b'-1} - 1$ implica que $0 < p \leq 2^{b-1} - 1$. Luego, cuando $q = 1$, el cociente $\left\lfloor \frac{p}{q} \right\rfloor = \left\lfloor \frac{p}{1} \right\rfloor = p$, lo que da el primer límite superior $2^{b-1} - 1$. Pues cuando q crece, el cociente decrece monótonamente.
  - Si $-2^{b'-1} \leq q < 0$ implica que $-2^{b-1} \leq p < 0$. Cuando $q = -1$, el cociente $\left\lfloor \frac{p}{q} \right\rfloor = \lfloor -p \rfloor = -p$, es decir, igual que el caso anterior se obtiene un nuevo límite superior 2^{b-1}. Sin embargo, si $q \neq -1 \wedge p \neq -2^{b-1}$, el límite superior es distinto pues:
    - Si $q = -2$ y $p = -2^{b-1}$, $\left\lfloor \frac{p}{q} \right\rfloor = \left\lfloor \frac{-2^{b-1}}{-2} \right\rfloor = \lfloor 2^{b-2} \rfloor = 2^{b-2} \leq 2^{b-1} - 1$. Que maximiza la función.
    - Si $q = -1$ y $p = -2^{b-1} + 1$, $\left\lfloor \frac{p}{q} \right\rfloor = 2^{b-1} - 1 \leq 2^{b-1} - 1$. Que maximiza la función.

    Luego si $q = -1$ y $p = -2^{b-1}$ el límite superior es 2^{b-1}, si no, el límite superior es $2^{b-1} - 1$.

  Por lo que se concluye que si $q = -1$ y $p = -2^{b-1}$ entonces $x \div y \leq 2^{b-1}$, si no $x \div y \leq 2^{b-1} - 1$.
- Si $\frac{p}{q} < 0$, entonces $x \div y = \left\lfloor \frac{p}{q} \right\rfloor 2^{f-f'}$, se sigue cumpliendo que $\left|\frac{a}{b}\right| < \frac{|a|}{|b|+1}, \forall |b| > 0, \forall a$.
  - Si $-2^{b'-1} \leq q < 0$ implica que $0 < p \leq 2^{b-1} - 1$. La función $\frac{p}{q}$ se minimiza cuando $q = -1$ y $p = 2^{b-1} - 1$, porque $\left\lceil \frac{p}{q} \right\rceil = \left\lceil \frac{2^{b-1}-1}{-1} \right\rceil = -2^{b-1} + 1$, que es el primer límite inferior.

- Si $0 < q \leq 2^{b'-1} - 1$ implica que $-2^{b-1} \leq p < 0$. La función $\frac{p}{q}$ se minimiza cuando $q = 1$ y $p = -2^{b-1}$, porque $\left\lceil \frac{p}{q} \right\rceil = \left\lceil \frac{-2^{b-1}}{1} \right\rceil = -2^{b-1}$, que nos da el segundo límite inferior.

De esto se concluye que si $p \neq -2^{b-1}$ y $q \neq -1$, entonces $\left\lfloor \frac{p}{q} \right\rfloor \in \mathbb{I}_b, \frac{p}{q} \geq 0$ y $\left\lceil \frac{p}{q} \right\rceil \in \mathbb{I}_b, \frac{p}{q} < 0$. Y si $p = -2^{b-1}$ y $q = -1$, entonces $\left\lfloor \frac{p}{q} \right\rfloor \in \mathbb{I}_b \cup \{2^{b-1}\} \subseteq \mathbb{I}_{b+1}, \frac{p}{q} \geq 0$ y que $\left\lceil \frac{p}{q} \right\rceil \in \mathbb{I}_b, \frac{p}{q} < 0$.

Luego $x \div y \in \mathbb{X}_{b,(f-f')}$ si $x \neq (2^{b-1} - 1)\delta_{b,f}$ e $y \neq -\delta_{b',f'}$, si no $x \div y \in \mathbb{X}_{b+1,(f-f')}$. □

Como consecuencia de la Definición 4.2.12 siempre se pierde información cuando se divide por el redondeo dado por la función techo y la función suelo. Una forma de minimizar dicho impacto es multiplicar primero el numerador y luego dividir.

La idea es poder trabajar con valores escalados en un conjunto uniforme $\mathbb{U}_b$ sin perder precisión, a este proceso le vamos a denominar uniformización. En primer lugar, la de un valor:

Definición 4.2.14 (Uniformización de un valor)

La uniformización de un vector convierte un valor en punto fijo $x = p2^f \in \mathbb{X}_{b,f} \setminus \{-2^{b-1}2^f\}$ a otro conjunto uniforme $\mathbb{U}_b$ sin perder precisión, es decir, el numerador es el mismo, el denominador cambia para que sea uniforme. Se denota como $u = \text{unif}(x)$ y se define como:

$$\text{unif}_{b,f} : \mathbb{X}_{b,f} \to \mathbb{U}_b$$

$$\text{unif}_{b,f}(x) = p2^{1-b} = \frac{x}{2^{f+b-1}}$$

Definición 4.2.15 (Uniformización de un vector)

La uniformización de un vector convierte un vector $v \in \mathbb{X}^n_{b,f}$ a otro vector escalado $v' \in \mathbb{U}^n_b$ y se expande la definición de la función *unif* para vectores:

$$\text{unif}_{b,f} : \mathbb{X}^n_{b,f} \to \mathbb{U}^n_b$$

$$v' = \text{unif}_{b,f}(v)$$

$$v'(i) = \text{unif}_{b,f}(v(i)), \forall i = 1, 2, ...n$$

Teorema 4.2.16 (Uniformización es biyectiva)

La función $\text{unif}_{b,f}$ es biyectiva.

Demostración.

La uniformización es inyectiva porque, por contradicción, supongamos que no es inyectiva y que $\exists x = p2^f \in \mathbb{X}_{b,f} \setminus \{-2^{b-1}2^f\}, \exists y = q2^f \in \mathbb{X}_{b,f} \setminus \{-2^{b-1}2^f\}$ con $x \neq y$, tales que:

$$\begin{aligned} & \text{unif}_{b,f}(x) = \text{unif}_{b,f}(y) \\ \Rightarrow\ & \text{unif}_{b,f}(p2^f) = \text{unif}_{b,f}(q2^f) \\ \Rightarrow\ & p2^{1-b} = q2^{1-b} \\ \Rightarrow\ & p = q \end{aligned}$$

Se llega a una contradicción, así que es **inyectiva**.

Finalmente, es suprayectiva porque la imagen de la función es el conjunto:

$$\begin{aligned} & = \{\text{unif}_{b,f}(x) : x \in \mathbb{X}_{b,f} \setminus \{-2^{b-1}2^f\}\} \\ & = \{\text{unif}_{b,f}(p2^f) : x \in \mathbb{I}_b \setminus \{-2^{b-1}\}\} \quad \text{[por definición]} \\ & = \{p2^{1-b} : x \in \mathbb{I}_b \setminus \{-2^{b-1}\}\} \\ & = \{p2^{1-b} : x \in \mathbb{I}_b\} \setminus \{-2^{b-1}2^{1-b}\} \\ & = \{p2^{1-b} : x \in \mathbb{I}_b\} \setminus \{-1\} \\ & = \mathbb{U}_b \qquad \text{[por definición]} \end{aligned}$$

Que cubre todo el codominio $\mathbb{U}_b$, así que la función es **suprayectiva**.

Como la función es a la vez inyectiva y suprayectiva, se dice que la función es **biyectiva**.

□

Corolario 4.2.16.1

Como la función $\text{unif}_{b,f}$ es biyectiva (Teorema 4.2.16), existe una función inversa $\text{unif}_{b,f}^{-1}$ que también es biyectiva.

Definición 4.2.17 (Desuniformización de un valor)

La desuniformización es el proceso inverso a la uniformización del vector, pues la función uniformización es biyectiva (Teorema 4.2.16) y existe una función inversa que también es biyectiva (Corolario 4.2.16.1). Dado $u = \frac{p}{2^{1-b}} \in \mathbb{U}_b$, se denota la desuniformización como $x = \text{unif}_{b,f}^{-1}(u)$ y se define como:

$$\text{unif}_{b,f}^{-1} : \mathbb{U}_b \to \mathbb{X}_{b,f}$$

$$\text{unif}_{b,f}(u) = p2^f = u2^{f-b+1}$$

Definición 4.2.18 (Desuniformización de un vector)

Se extiende la definición de la desuniformización a vectores pues por razones similares al Teorema 4.2.16, la uniformización de un vector también es biyectiva y existe una función inversa. Dado un vector $v \in \mathbb{U}_b^n$ se denota la desuniformización como $v' = \text{unif}_{b,f}^{-1}(v)$ y se define como:

$$\text{unif}_{b,f}^{-1}(v) : \mathbb{U}_b^n \to \mathbb{X}_{b,f}^n$$

$$v' = \text{unif}_{b,f}^{-1}(v)$$

$$v'(i) = \text{unif}_{b,f}^{-1}(v(i)), \forall i = 1, 2, ..., n$$

Un caso específico de división que se ha utilizado a lo largo del proyecto es la división entre un número entero, cabe recordar que $\mathbb{X}_{b,0} = \mathbb{I}_b$ (Sección 4.2). Luego se obtiene el Corolario 4.2.18.1.

Corolario 4.2.18.1

Del Teorema 4.2.13 se deduce que dados $x = p2^f \in \mathbb{X}_{b,f}$ y $n \in \mathbb{I}_{b'} = \mathbb{X}_{b',0}$, la división de un número en punto fijo entre un número entero está en el mismo conjunto excepto el cuando $n = -1$ y $x = -2^{b-1}2^f$:

$$x \div n \in \mathbb{X}_{b,f} \text{ , si } \quad x \neq -2^{b-1}2^f \quad \text{y} \quad n \neq -1$$

4.2.4. Resumen del algoritmo

El algoritmo a implementar es el algoritmo 4 (*validation phase*) del artículo en que se basa el proyecto [2]. Para determinar si una época pertenece o no a un ataque epiléptico se computan lo que el artículo llama *features* (o características) que son cinco funciones matemáticas: `max_distance`, `energy`, `psd_1`, `psd_2` y `psd_3`. Si las características de una época están en ciertos rangos determinados por el modelo, la época no es un artefacto y la distancia utilizando el algoritmo de deformación dinámica del tiempo es lo suficientemente pequeña para alguno de los patrones: el modelo dice que la época puede tratarse de un ataque epiléptico.

El modelo (al que el artículo llama *batch*) es una 7-tupla:

$$B = \left(B_M, B_E, B_{P_1}, B_{P_2}, B_{P_3}, B_D, B_Q\right), B \in \mathcal{B}$$

$$B_M, B_E, B_{P_1}, B_{P_2}, B_{P_3} \in \mathbb{R} \times \mathbb{R}$$

$$B_D \in \mathbb{R}$$

$$B_Q = \left\{S_q : \mathbb{R} \to \mathbb{R}\right\}$$

- B_M, B_E, B_{P_1}, B_{P_2}, B_{P_3} son 2-tuplas de dos números reales $B_x = \left(B_{x_l}, B_{x_h}\right), B_{x_l} \leq B_{x_h}$ que indican el rango válido de los resultados

de evaluar las funciones `max_distance`, `energy`, `psd_1`, `psd_2` y `psd_3` respectivamente. Además, como se verá más adelante, los valores no negativos: $B_M, B_E, B_{P_1}, B_{P_2}, B_{P_3} \geq 0$.

- B_D es el valor máximo del resultado del algoritmo de deformación dinámica del tiempo que determina si una época puede ser una ataque epiléptico. Nótese que la función solo retorna números no negativos. Se puede decir que $B_D \geq 0$.
- B_Q es un conjunto de señales finitas que contienen los patrones a contrastar.

Así, se dice que una época tiene un ataque si cumple todas las condiciones mencionadas anteriormente:

$$\begin{aligned} \text{¿ataque?}(e, b) = & \quad \neg\ \text{¿artefacto?}(e) \\ & \wedge\ b_{M_l} \leq \text{max\_distance}(e) \leq b_{M_h} \\ & \wedge\ b_{E_l} \leq \text{energy}(e) \leq b_{M_h} \\ & \wedge\ b_{P_{1_l}} \leq \text{psd\_1}(e) \leq b_{M_{1_h}} \\ & \wedge\ b_{P_{2_l}} \leq \text{psd\_2}(e) \leq b_{M_{2_h}} \\ & \wedge\ b_{P_{3_l}} \leq \text{psd\_3}(e) \leq b_{M_{3_h}} \\ & \wedge\ \exists q \in b_Q : \text{dtw}(e, q) \leq b_M \cdot d_{\text{th}} \end{aligned}$$

$$\text{¿ataque?} : (\mathbb{R} \to \mathbb{R}) \to \mathcal{B} \to \mathbb{B}$$

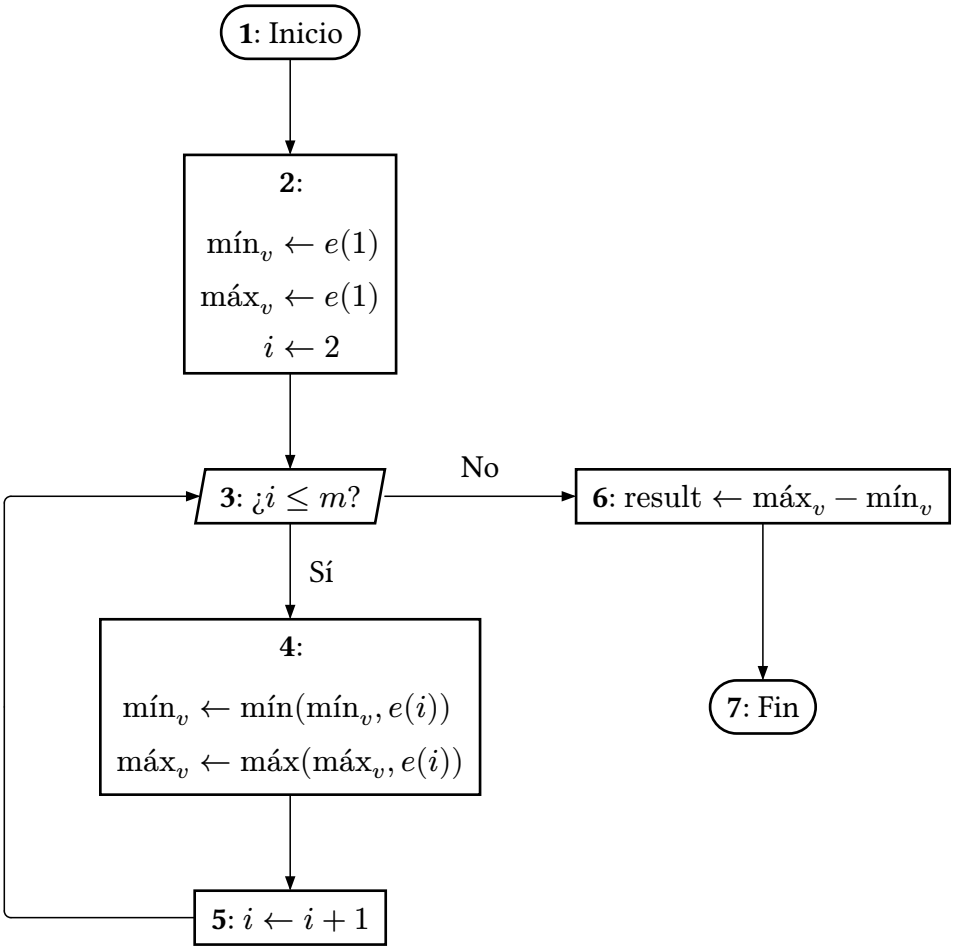
4.3. Algoritmos

4.3.1. *Max distance*

Max distance es sin duda uno de los algoritmos más sencillos. Retorna la distancia pico a pico de una señal, es decir, la diferencia entre el valor máximo y el valor mínimo de una época. Sea $e \in \mathbb{R}^m$ una época de m elementos, la entrada del algoritmo. Obsérvese el diagrama de flujo siguiente (Figura 4.2).

Figura 4.2

Algoritmo `max_distance`



4.3.1.1. Problemas

Los puntos de fallo suelen ser cuando se indexa o cuando se opera. Indexar fuera de la época no está definido; y si el resultado de una operación no se puede representar en el conjunto es un error.

Se observa posibles fallos en:

- Paso **2**: Indexar el primer elemento de la época $e(1)$. Este falla si $m = 0$. Es decir, si no hay ningún elemento en el vector. Eso no tiene mucho sentido, pero podría ocurrir, así que la primera restricción es que $m > 0$.

- Paso **5**: Si se utiliza un $i \in \mathbb{I}_b$ e $i = 2^{b-1} - 1$, entonces al añadir 1 desborda porque $2^{b-1} \notin \mathbb{I}$. Esto ocurre cuando $m = 2^{b-1} - 1$. Otra restricción debe ser que $m < 2^{b-1} - 1$.
- Paso **6**: Este paso solo da problemas trabajando con punto fijo. Por ejemplo, en un vector de $m = 2$ elementos que sea $e = \left(-\delta_{b,f}, 2^f\left(2^{b-1} - 1\right)\right)$. Al final del algoritmo $\text{máx}_v = 2^f\left(2^{b-1} - 1\right)$ y $\text{mín}_v = -\delta_{b,f}$, y el resultado $\text{máx}_v - \text{mín}_v = 2^f\left(2^{b-1} - 1\right) - \left(-\delta_{b,f}\right) = 2^f\left(2^{b-1}\right) \notin \mathbb{X}_{b,f}$.

La función en punto fijo depende, además de la época, en otros dos conjuntos: El de entrada $\mathbb{X}_{b_e,f_e}$ y el de salida $\mathbb{X}_{b_s,f_s}$. De manera que la época será un vector de m elementos del primer conjunto: $e \in \mathbb{X}^m_{b_e,f_e}$. Las variables $\text{mín}_v, \text{máx}_v \in \mathbb{X}_{b_e,f_e}$, también pertenecen al conjunto de entrada. $i \in \mathbb{I}_b$ para un $b > 1$ cualquiera.

4.3.1.2. Soluciones

Para simplificar el problema, se va a utilizar valores de entrada de 32 bits y la salida también será de 32 bits. Así que $b_e = b_s = 32$. Para solucionar el problema que existía en la resta se debe convertir primero los valores al tipo de retorno. De acuerdo con el Teorema 4.2.5 $b_s \geq b_e + (f_e - f_s) \Rightarrow 32 \geq 32 + (f_e - f_s) \Rightarrow f_s \geq f_e$ para que la conversión se pueda realizar.

Para que la resta se pueda realizar además es necesario que $f_e < f_s$. Pues el valor se maximiza cuando $\text{máx}_v = (2^{31} - 1)2^f_e$ y $\text{mín}_v = -2^{31}2^f_e$ entonces $\text{máx}_v - \text{mín}_v = 2^{32} - 1$. Nótese que se pierde información como consecuencia del Teorema 4.2.4, se minimiza la pérdida de información cuando más pequeño sea f_s. Así que $f_e + 1 = f_s$. Las **precondiciones** son:

- $m > 0$, el vector debe tener al menos un elemento.
- $m < \text{máx}\{\mathbb{I}_b\}$, la longitud del vector debe ser menor que el máximo del número entero que se utilice para indexar.
- $f_s = f_e + 1$, para poder operar y minimizar la pérdida de información.

Y las **postcondiciones** son:

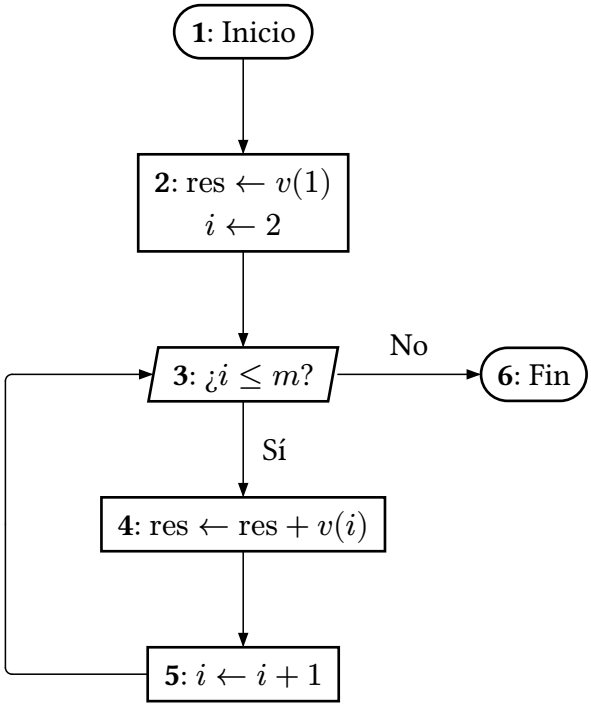
- El resultado es no negativo, pues $\text{máx}_v \geq \text{mín}_v$.

4.3.2. Acumulación

Varios algoritmos tienen algún paso que consiste en suma una secuencia de elementos. El algoritmo es similar al algoritmo *max distance* (Sección 4.3.1) en que es una reducción. Dado un vector $v \in \mathbb{R}^m$ el algoritmo acumuación computa: $\sum_{i=1}^{m} v(i)$, de acuerdo con el siguiente diagrama de flujo:

Figura 4.3

Algoritmo «acumulación»



4.3.2.1. Problemas

Se siguen viendo similitudes con *max distance*, y se ve que comparte sus dos primeros problemas Sección 4.3.1.1:

- En el paso **2**, $v(1)$ falla si $m = 0$.
- En el paso **5**, $i + 1$ falla si $i \in \mathbb{I}_b \wedge i + 1 \notin \mathbb{I}_b$, por ejemplo si $i \in \mathbb{I}_b$ y $m = 2^{b-1} - 1$
- Además en el paso **4**, se ve un problema similar al que tenía `max-distance` en su paso número **6**. Y es que la suma puede desbordar en punto fijo.

4.3.2.2. Análisis de punto fijo

Existen varias alternativas para solucionar este problema, la más sencilla es sin duda utilizar un tipo de punto fijo más grande para el resultado, consecuencia del Teorema 4.3.2.

> **Lema 4.3.1**
>
> Si $a \geq 2$ y $b \geq 2$, entonces $ab \geq a + b$

Teorema 4.3.2

Dados $i \in \mathbb{I}_B$ y $v(i) \in \mathbb{X}_{b,f} \forall i = 1, 2, ..., m$, entonces el resultado de $\sum_{k=1}^{i} v(k)$ está en $\mathbb{X}_{b+B-1,f}$ y está en el intervalo $\left[-i2^{b-1}2^f, i\left(2^{b-1}-1\right)2^f\right]$

Demostración.

Si $i \in \mathbb{I}_B$ y $v(i) \in \mathbb{X}_{b,f} \forall i = 1, 2, ..., m$. Sea la función de acumulación $g(v, m) = \sum_{i=1}^{m} v(i)$, como $\mathbb{X}_{b,f}$ y $\mathbb{I}_B$ son finitos, el codominio de g también es finito y tiene un valor máximo y mínimo que pertenece a $\mathbb{X}_{b+B-1,f}$

La función se maximiza cuando $v(i) = \left(2^{b-1}-1\right)2^f, \forall i = 1, 2, ..., m$, el resultado entonces sería:

$$\text{máx}\left\{\sum_{i=1}^{m} v(i)\right\} =$$

$$= \sum_{i=1}^{2^{B-1}-1} \left(2^{b-1}-1\right)2^f$$

$$= 2^f\left(2^{B-1}-1\right)\left(2^{b-1}-1\right)$$

$$= 2^f\left[2^{B+b-2} - 2^{B-1} - 2^{b-1} + 1\right]$$

Es necesario ver que:

$$\left(2^{b+B-2}+1\right)2^f > \left(2^{B-1}+2^{b-1}\right)2^f \Leftrightarrow 2^{b+B-2}+1 > 2^{B-1}+2^{b-1}$$

Hay que tener en cuenta que $b, B \in \mathbb{N}^+$. Los casos son los siguientes: Cuando $b \geq 2 \wedge B \geq 2$, como $2^{B-1} \geq 2$ y $2^{b-1} \geq 2$, de acuerdo con el Lema 4.3.1:

$$2^{b+B-2} \geq 2^{B-1}2^{b-1} \Rightarrow 2^{b+B-2}+1 > 2^{B-1}2^{b-1}$$

Por el contrario si $b = 1$, entonces $2^{b+B-2}+1 = 2^{B-1} \geq 1 + 2^{B-1}$. Igualmente con $B = 1$, entonces $2^{b+B-2}+1 = 2^{b-1} \geq 1 + 2^{b-1}$.

Entonces:

$$0 \leq \left(2^{b+B-2}+1\right)2^f - \left[2^{b-1}+2^{B-1}\right]2^f \leq \left(2^{B+b-2}-1\right)2^f$$

$$\text{máx}\left\{\sum_{i=1}^{m} v(i)\right\} = \left(2^{b+B-2}+1\right)2^f - \left[2^{b-1}+2^{B-1}\right]2^f \in \mathbb{X}_{B+b-1,f}$$

La función se minimiza cuando $v(i) = -2^{b-1}2^f, \forall i = 1, 2, ..., m$, el resultado entonces sería:

$$\text{mín}\left\{\sum_{i=1}^{m} v(i)\right\} =$$

$$= 2^f \sum_{i=1}^{2^{B-1}-1} -2^{b-1}$$

$$= 2^f (2^{B-1} - 1)(-2^{b-1})$$

$$= -2^{B+b-2}2^f + 2^{b-1}2^f$$

$$\left[2^{b-1}2^f \leq 2^{B+b-2}2^f \Leftrightarrow b-1 \leq B+b-2 \Leftrightarrow B \geq 1\right]$$

$$-2^{B+b-2}2^f \leq -2^{B+b-2}2^f + 2^{b-1}2^f \leq 0$$

$$\text{mín}\left\{\sum_{i=1}^{m} v(i)\right\} = -2^{B+b-2}2^f + 2^{b-1}2^f \in \mathbb{X}_{B+b-1,f}$$

□

Esto viene a decir que si se utiliza un punto fijo de b bits y el tamaño máximo se puede almacenar en una variable de tipo entero con signo de B bits, se necesitaría que el acumulador tuviera $b + B - 1$ bits para almacenar el resultado, con la misma f que el punto fijo de entrada.

4.3.2.3. SPARK

A la hora de explicar esto a SPARK hay que utilizar una técnica que utiliza un vector con las sumas parciales del vector de manera que $v'(i) = \sum_{j=1}^{i} v(i), i = 1, 2, ..., m$ [37]. Además, una condición adicional que se ha visto útil, es decir, que los valores de $v'(i)$ están acotados. Es decir,

$$v(i) \in [a, b], \forall i = 1, 2, ..., m \Rightarrow v'(i) = \sum_{j=1}^{i} v(i) \in [ai, bi], \forall i = 1, 2, ..., m$$

En SPARK se utilizan lo que se llaman funciones fantasma (*Ghost*), que son subrutinas que no generan código y que solamente son utilizadas por el probador de teoremas interno. En este caso se define la función `Generic_Accumulation` se declararía como dice el Listado 4.1.

Listado 4.1

Especificación de la función fantasma `Generic_Accumulation`

```
1  generic                                                    SPARK
2     type Fixed_Type is delta <>;
3     type Index_Type is range <>;
4     type Array_Type is array (Index_Type range <>) of Fixed_Type;
5     First : in Fixed_Type;
6     Last  : in Fixed_Type;
7  function Generic_Accumulation (
8     Item : in Array_Type)
9     return Array_Type with
```

```
10    Ghost    => True,
11    Global   => null,
12    Pre      => Item'Length > 0
13       and then (for all X of Item => X in First .. Last),
14    Post     => (
15       declare  Result renames Generic_Accumulation'Result;
16       begin    Result'First = Item'First
17       and then Result'Length = Item'Length
18       and then Result (Item'First) = Item (Item'First)
19       and then (for all I in Item'First + 1 .. Item'Last =>
20                    Result (I - 1) in Positive (I - Item'First) * First
21                                   .. Positive (I - Item'First) * Last
22                    and then Result (I) = Result (I - 1) + Item (I))
23       and then Result (Item'Last) in Item'Length * First
24                                    .. Item'Length * Last);
```

La línea 10 del Listado 4.1 (Listado 4.1-10) dice que es una función fantasma, que no genere código. En Listado 4.1-11 mencionamos que es una función que no modifica el estado global (en SPARK está prohibido si es una función, no si es un procedimiento). A continuación hay dos precondiciones:

- La longitud debe ser estrictamente positiva
- Todos los elementos deben estar acotados en el mismo rango: $v(i) \in [\text{first}, \text{last}] \forall v(i)$.

Finalmente, la postcondición dice que,

- La longitud del resultado es igual a la del parámetro. Y además que comienzan en el mismo índice (en Ada o SPARK se puede indexar a partir de cualquier valor).
- Todos los elementos están acotados como se veía al principio en la Sección 4.3.2.3.

Y se implementaría como se muestra en Listado 4.2.

Listado 4.2

Implementación de la función fantasma `Generic_Accumulation`

```
1  function Generic_Accumulation (                                    SPARK
2     Item : in Array_Type)
3     return Array_Type is
4     Result : Array_Type (Item'Range) := [others => 0.0];
5  begin
6     Result (Item'First) := Item (Item'First);
7     for Index in Item'First + 1 .. Item'Last loop
8        pragma Loop_Invariant (Result (Item'First) = Item (Item'First));
9        pragma Loop_Invariant (
10          (for all I in Item'First + 1 .. Index - 1 =>
11             Result (I - 1) in Positive (I - Item'First) * First
12                            .. Positive (I - Item'First) * Last
```

```
13             and then Result (I) = Result (I - 1) + Item (I)
14             and then Result (I) in Positive (I - Item'First + 1) * First
15                                 .. Positive (I - Item'First + 1) * Last));
16       Result (Index) := Result (Index - 1) + Item (Index);
17    end loop;
18    return Result;
19 end Generic_Accumulation;
```

Es preciso mencionar que es necesario especificar los predicados invariantes en el bucle (`Loop_Invariant`), es decir, qué propiedades no cambian de una iteración a la siguiente. En este caso estamos diciendo que primero todos los valores hasta el índice actual está acotados (Listado 4.2-11); que su valor es la suma del anterior resultado más el valor del vector en el índice actual (Listado 4.2-13); y que, finalmente, esa suma sigue estando acotada en un superconjunto (Listado 4.2-14). SPARK, no sabe si se están modificando los elementos anteriores o no.

La demostración en SPARK es similar a la inducción matemática: sea $i \in \mathbb{I}_B$ la variable de iteración, $v \in \mathbb{Z}^m_{b,f}$ el vector de entrada y $v' \in \mathbb{Z}^m_{b+B,f}$ el vector de salida, de modo que $v(i) \in [a, b], \forall i = 1, 2, ..., m$.

1. **Caso base**: $v(1) = v'(1)$ y $v(1) \in [a, b]$
2. **Hipótesis inductiva**: Para $v'(k), k > 1$ suponemos $v'(k) \in [ka, kb]$.
3. **Tesis inductiva**: Para $v'(k+1)$, si $v'(k) \in [ka, kb]$, como $v'(k+1) = v'(k) + v(k)$ y como $v(k) \in [a, b]$, entonces $v'(k) + v(k) \in [a(k+1), b(k+1)]$, por lo que $v'(k+1) \in [a(k+1), b(k+1)]$

La función fantasma no computa la acumulación como tal, debe ser otra función la que haga uso de ella para definir dichos rangos. Suponiendo que hay un tipo de punto fijo `Input_Type` ($\mathbb{X}_{b,f}$), un tipo para indexar `Index_Type` ($\mathbb{I}_B$), un tipo de resultado `Result_Type` ($\mathbb{X}_{b+B-1,f}$) y un tipo vector de elementos de tipo `Input_Type` que indexa con `Index_Type >= 1` (véase el Listado 4.3).

Listado 4.3

Ejemplo de uso de la función fantasma `Generic_Accumulation`

```
1  function Accumlate (Item : in Input_Type_Array) return Result_Type is      SPARK
2     subtype Constrained_Result is Result_Type
3        range Result_Type (Input_Type'First) .. Result_Type (Input_Type'Last)
4     type Result_Array is array (Index_Type range <>) of Result_Type;
5     function Acc_Sum is
6        new Lemmas.Generic_Accumulation (
7        Fixed_Type => Result_Type,
8        Index_Type => Index_Type,
9        Array_Type => Result_Array,
10       First      => Constrained_Result'First,
11       Last       => Constrained_Result'Last);
12    Mapped : constant Result_Array (Item'Range) :=
```

```
13          [for I in Item'Range => Constrained_Result (Item (I))] with
14          Ghost => True;
15       Result : Result_Type := 0.0;
16    begin
17       for Index in Item'Range loop
18          Result := Result + Constrained_Result (Item (Index));
19          pragma Loop_Invariant (Result = Acc_Sum (Mapped) (Index));
20       end loop;
21       return Result;
22    end Accumulate;
```

Se debe «instanciar» la función que `Generic_Accumulation`, es decir, es genérica (sirve para distintos tipos de tipos) y por lo tanto está parametrizada (Listado 4.3-5). Se define una constante fantasma con el vector de entrada, pero esta vez contiene los valores con el tipo del resultado. Finalmente, en Listado 4.3-19 dice que el resultado `Result` en la i-ésima posición tiene el mismo valor que el valor en la i-ésima posición de `Generic_Accumulation`.

4.3.3. Media

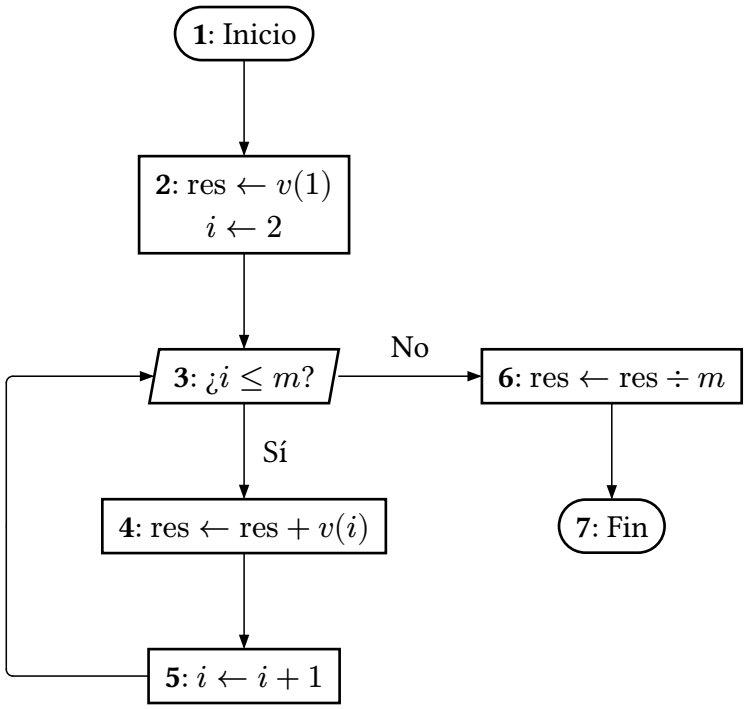
El algoritmo de la media es básicamente el de la acumulación, pero dividiendo entre el número de elementos:

$$\mu(v, m) = \sum_{i=1}^{m} \frac{v(i)}{m}$$

O en diagrama de flujo de la Figura 4.4.

Figura 4.4

Algoritmo de la media



4.3.3.1. Problemas

Se siguen manteniendo los problemas de la acumulación como se puede ver en la Sección 4.3.2.1. Pero en el caso de punto fijo el resultado, si $v(i) \in \mathbb{X}_{b,f}, \forall i = 1, 2, ..., m$, también pertenece al mismo conjunto que los valores de entrada $\mathbb{X}_{b,f}$, según la demostración del Teorema 4.3.2 (esta vez con m en vez del valor máximo):

$$\text{máx}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} = m\left(2^{b-1} - 1\right)2^{f}$$

$$\text{mín}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} = -m2^{b-1}2^{f}$$

Y por supuesto si se divide entre m cualquiera de los dos para hacer la media se obtiene el máximo y mínimo respectivamente del conjunto de los valores del vector:

$$\text{máx}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} \div m = \left(2^{b-1}-1\right)2^{f}$$

$$\text{mín}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} \div m = 2^{b-1}2^{f}$$

Sin embargo, nótese que esta condición no es cierta al trabajar con punto flotante por problemas de precisión. Es bien conocido que trabajando con un computado de punto flotante de 64 bits ($\mathcal{F}_{64}$), por ejemplo, la suma $0.1+0.2$ no da exactamente 0.3, da 0.30000000000000004; porque ni 0.1 ni 0.2 se pueden respresentar en binario con un número finito de dígitos binarios: $0.1_{10} = 0.00\overline{0011}_2$ y $0.2_{10} = 0.0\overline{0011}$. Luego no se puede hacer dicha suposición cuando se trabaja en cualquier $\mathcal{F}_b$.

4.3.3.2. Soluciones

Con punto flotante siempre se trabaja con cierto error, así que hay poco que se pueda hacer. Sin embargo, con punto fijo se puede aplicar los conceptos que se utilizan para implementar la acumulación y se ve que si

- $i \in \mathbb{I}_k$
- $v \in \mathbb{X}_{b,f}^{m}$

Entonces la variable para acumular *res* como dice Teorema 4.3.2 debe pertenecer a $\mathbb{X}_{b+k-1,f}$. La división del final es una división de punto fijo que pierde precisión pues se divide entre $m \in \mathbb{N}^{+}$ como dice el Teorema 4.2.13. Como $m > 1$, se sabe que:

$$\text{res} \div m \in \mathbb{X}_{b+k-1,f}$$

y que está acotado en $[-m2^{b-1}2^{f}, m\left(2^{b-1}-1\right]$, porque

$$\text{máx}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} = m\left(2^{b-1}-1\right)2^{f}$$

$$\text{mín}_{m=m}\left\{\sum_{i=1}^{m} v(i)\right\} = -m2^{b-1}2^{f}$$

Aplicando la Definición 4.2.12 con $m \in \mathbb{N}^{+}$ para

$$\left(m\left(2^{b-1}-1\right)2^{f}\right) \div m = \left\lfloor \frac{m2^{b-1}-1}{m} \right\rfloor 2^{f} = 2^{b-1}-1$$

$$\left(-m2^{b-1}2^{f}\right) \div m = \left\lceil \frac{-m2^{b-1}}{m} \right\rceil 2^{f} = -2^{b-1}$$

Luego $\text{res} \div m \in \left[-2^{b-1}, 2^{b-1}-1\right]$ y además $\text{res} \div m \in \mathbb{X}_{b,f}$

4.3.4. Varianza

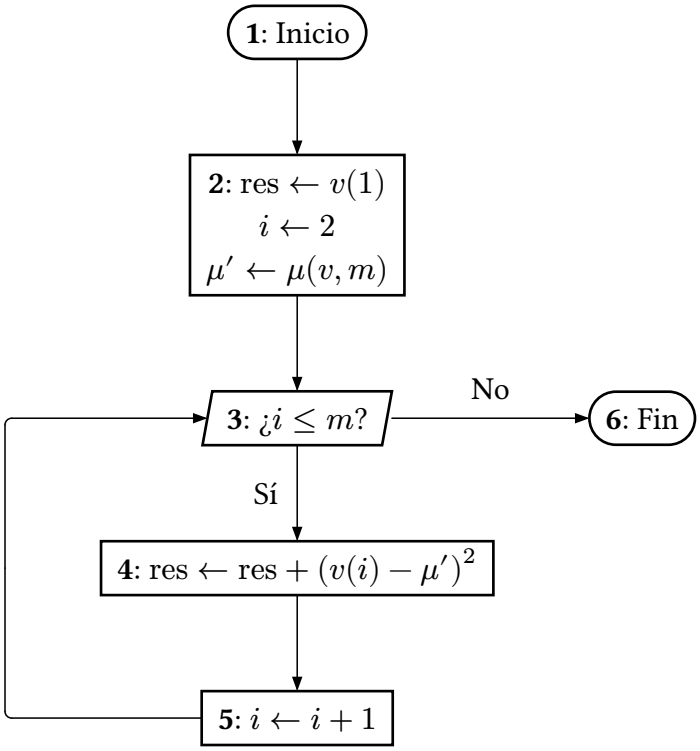
La varianza se define como:

$$\mathrm{Var}(v, m) = \sum_{i=1}^{m} (x - \mu(v, m))^2$$

donde $v \in \mathbb{R}^m$. Se parece mucho a la media, es una acumulación, su diagrama de flujo sigue siendo parecido como se puede ver en Figura 4.5.

Figura 4.5

Algoritmo de la varianza



4.3.4.1. Problemas

Para punto flotante sigue habiendo los mismos problemas que para la acumulación (Sección 4.3.2.1) en cuanto a indexado:

- El paso **2**, $v(1)$ falla si $m = 0$
- En el paso **5**, $v(i + 1)$ falla.

Sin contar los numerosos problemas que acarrea utilizar punto flotante:

- En el paso **3** puede subdesbordar $v(i) - \mu'$.
- En el paso **3**, además, puede desbordar el cuadrado $(v(i) - \mu')^2$
- Es más, en el paso **3**, la suma también puede desbordar: $\mathrm{res} + (v(i) - \mu')^2$.

4.3.4.2. Soluciones

Cuando los números son uniformes (Sección 4.2.2) se derivan propiedades interesantes. Por ejemplo, en el supuesto en que $v(i) - \mu'$ fuera uniforme, su cuadrado también lo será según el Teorema 4.2.11.

Además, como se vio en la función media (Sección 4.3.3.1), su resultado también está en el mismo conjunto que los elementos del vector:

$$\mu : \mathbb{X}_{b,f}^{m} \to \mathbb{N}^{+} \to \mathbb{X}_{b}$$

De esta manera si estamos trabajando en $\mathbb{U}_b$, también se sigue cumpliendo:

$$\mu : \mathbb{U}_{b}^{m} \to \mathbb{N}^{+} \to \mathbb{U}_{b}$$

Se decide entonces trabajar con $v \in \mathbb{U}_b^m$, $m > 0$ e $i \in \mathbb{I}_B$, entonces $\mu' \in \mathbb{U}_b$. El problema es que:

$$v(i) - \mu' \in (-2, 2)$$

Y entonces:

$$(v(i) - \mu')^2 \in (0, 4)$$

Para solucionarlo, antes de restar se dividen entre dos ambos operandos, para que:

$$v(i) \div 2 - \mu' \div 2 \in (-1, 1)$$

Y por tanto:

$$(v(i) \div 2 - \mu' \div 2)^2 \in (0, 1)$$

En este caso se va a definir la función «cuarto de la varianza», que computa la cuarta parte de la varianza y sin desuniformizar. Esto se debe a que varios algoritmos utilizan esta función y aprovechan que está dividida entre cuatro para aplicar optimizaciones. El diagrama de flujo se muestra en la Figura 4.6, del cual las variables son:

- $i \in \mathbb{I}_k$
- $v \in \mathbb{X}_{b,f}^{m}$
- $v' \in \mathbb{U}_{b}^{m}$ (Por Definición 4.2.15).
- res $\in \mathbb{U}_{b+k-1}$ (Por el Teorema 4.3.2)

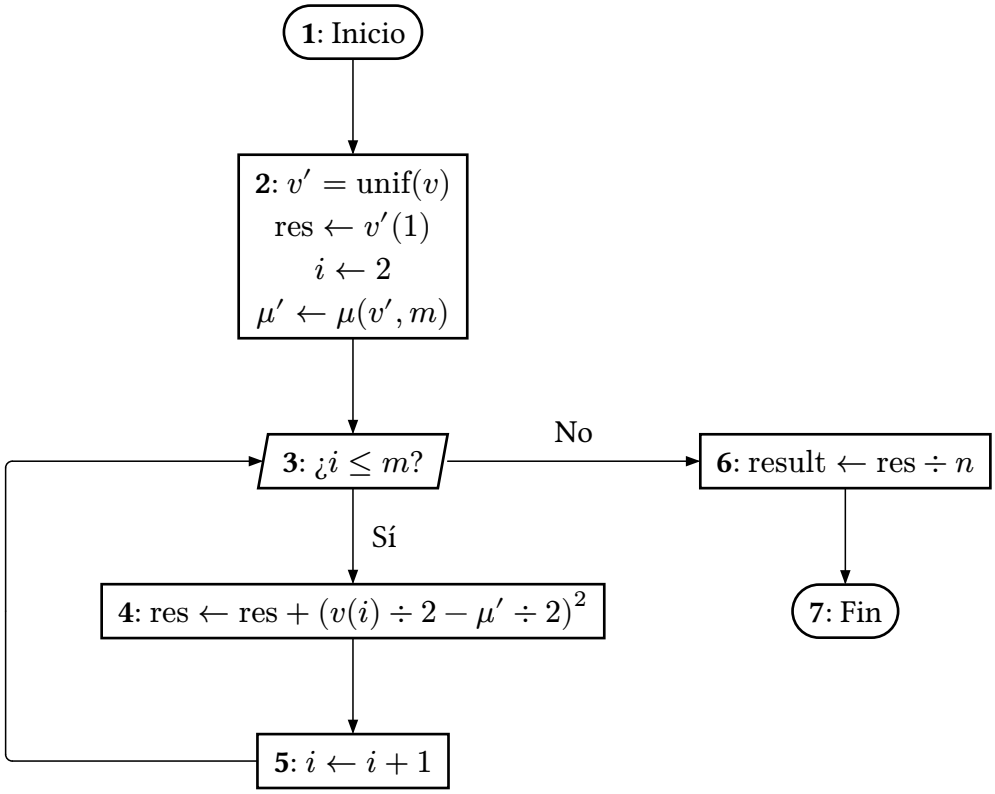
La variable *res* almacena lo que es el cuarto de la varianza. En el sexto paso del algoritmo se divide entre el número de elementos similar a como lo hace la media (Sección 4.3.3.2):

$$\text{res} \in [0, m(2^{b-1} - 1)2^{1-b}]$$
$$\text{res} \div m \in [0, (2^{b-1} - 1)2^{1-b}]$$

Como res $\div m \in [0, 1)$ es uniforme, puede convertirse a cualquier tipo en punto fijo binario uniforme.

Figura 4.6

Algoritmo de la cuarto de la varianza



4.3.5. *Energy*

La función *energy* la define la implementación de referencia como la propia varianza, que su vez se define en la Sección 4.3.4. En este caso se extiende el algoritmo del cuarto de la varianza para desuniformizar el resultado y multiplicarlo por cuatro. Nótese que con la implementación de punto flotante no hay problema porque no calcula el cuarto de la varianza, solo la de punto fijo.

En este caso no es necesario definir el diagrama de flujo, pues es una única expresión dada por partes:

$$q = \text{conv}_{b,1-b}(\text{quarter\_variance}(v)), v \in \mathbb{U}_b$$

q es el cuarto de la varianza, se ha convertido a un tipo en punto fijo uniforme de b bits, una operación que puede hacer que se pierda información, pero por razones prácticas se va a ignorar.

Este valor no es el real, además de estar dividido entre cuatro, está uniformizado **dos** veces. Porque en un punto del cuarto de la varianza se hizo el cuadrado del valor uniformizado:

$$(v(i) \div 2 - \mu' \div 2)^2 \in (-1, 1)$$

Por la Definición 4.2.14 el valor $v(i)$ que está uniformizado está implícitamente multiplicado por $\frac{1}{2^{f+b-1}}$, de igual manera la media μ' también está implícitamente multiplicada por el mismo valor. Suponiendo $\mu' = \frac{\mu''}{2^{f+b-1}}$ y que $v(i) = v'' \frac{i}{2^{f+b-1}}$, la expresión:

$$\left(v'' \frac{i}{2^{f+b-1}} \div 2 - \frac{\mu''}{2^{f+b-1}} \div 2\right)^2$$

$$\frac{(v''(i) \div 2 - \mu'' \div 2)^2}{(2^{f+b-1})^2}$$

Está uniformizada dos veces, así que hay que aplicar la desuniformización dos veces.

$$q'' = \text{unif}_{b,f}^{-1}\left(\text{unif}_{b,f}^{-1}(q)\right) \in \mathbb{X}_{b,f}$$

Finalmente, el resultado hay que multiplicarlo por cuatro, es decir, son necesarios dos bits addicionales para almacenar el valor. Se puede ver que como $4 \in \mathbb{X}_{3,0}$ y según el Teorema 4.2.5:

$$4q'' \in \mathbb{X}_{b+3-1,f+0} = \mathbb{X}_{b+2,f}$$

Eso quiere decir que si $v \in \mathbb{X}_{b,f}^m$, entonces el tipo del resultado debe ser convertible desde $\mathbb{X}_{b+2,f}$, pues:

$$\text{energy}(v) = 4q'' \in \mathbb{X}_{b+2,f}$$

Es decir, si $\mathbb{X}_{b',f'}$ es el tipo del resultado, según el Teorema 4.2.5, se debe cumplir que $b' \geq b + 2 + (f - f')$ para que $\text{conv}_{b',f'}(\text{energy}(v)) \in \mathbb{X}_{b',f'}$.

4.3.6. Regla de *Simpson*

4.3.6.1. Regla de Simpson para datos irregularmente espaciados

La regla de Simpson, en honor de Thomas Simpson, es un método para aproximar integrales; existe un caso específico para integrar datos irregularmente espaciados [38].

Sean el límite de integración inferior $a \in \mathbb{R}$ y el límite de integración superior $b \in \mathbb{R}$ con $b > a$. Dados N subintervalos de anchuras h_k, la regla de Simpson compuesta viene dada por la expresión:

$$\int_a^b f(x)dx \approx \varphi + \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} \frac{h_{2i} + h_{2i+1}}{6} \left[\left(2 - \frac{h_{2i+1}}{h_{2i}} \right) f_{2i} + \frac{(h_{2i} + h_{2i+1})^2}{h_{2i} h_{2i+1}} f_{2i+1} + \left(2 - \frac{h_{2i}}{h_{2i+1}} \right) f_{2i+2} \right]$$

donde

$$f_k = f\left(a + \sum_{i=0}^{k-1} h_i \right)$$

y donde φ depende de la paridad de N, si hay N intervalos, hay $N + 1$ anchuras,

$$\varphi = \begin{cases} \alpha f_N + \beta f_{N-1} - \eta f_{N-2} & \text{, si } \ N \equiv 1 (\text{mód } 2) \\ 0 & \text{, si no} \end{cases}$$

donde

$$\alpha = \frac{2h_{N-1}^2 + 3h_{N-1}h_{N-2}}{6(h_{N-2} + h_{N-1})}$$

$$\beta = \frac{h_{N-1}^2 + 3h_{N-1}h_{N-2}}{6h_{N-2}}$$

$$\eta = \frac{h_{N-1}^3}{6h_{N-2}(h_{N-2} + h_{N-1})}$$

4.3.6.2. Regla de Simpson para datos uniformemente espaciados

Los datos de entrada están igualmente espaciados, es decir, $h_k = h_j \forall k, j = 1, 2, \ldots N - 1$, por lo que la expresión se puede simplificar y por lo tanto se puede mejorar el tiempo de ejecución del algoritmo.

$$\int_a^b f(x)dx \approx \varphi + \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} \frac{h + h}{6} \left[\left(2 - \frac{h}{h} \right) f_{2i} + \frac{(h + h)^2}{hh} f_{2i+1} + \left(2 - \frac{h}{h} \right) f_{2i+2} \right]$$

$$\int_a^b f(x)dx \approx \varphi + \frac{h}{3} \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} [f_{2i} + 4f_{2i+1} + f_{2i+2}]$$

También se puede simplificar la expresión que da φ, porque

$$\alpha = \frac{2h^2 + 3hh}{6(h+h)} = \frac{5}{12}h$$
$$\beta = \frac{h^2 + 3hh}{6h} = \frac{2}{3}h$$
$$\eta = \frac{h^3}{6h(h+h)} = \frac{1}{12}h$$

luego

$$\varphi = \begin{cases} \frac{h}{12}(5f_N + 8f_{N-1} - 1f_{N-2}) & \text{, si } \ N \equiv 1(\text{mód } 2) \\ 0 & \text{, si no} \end{cases}$$

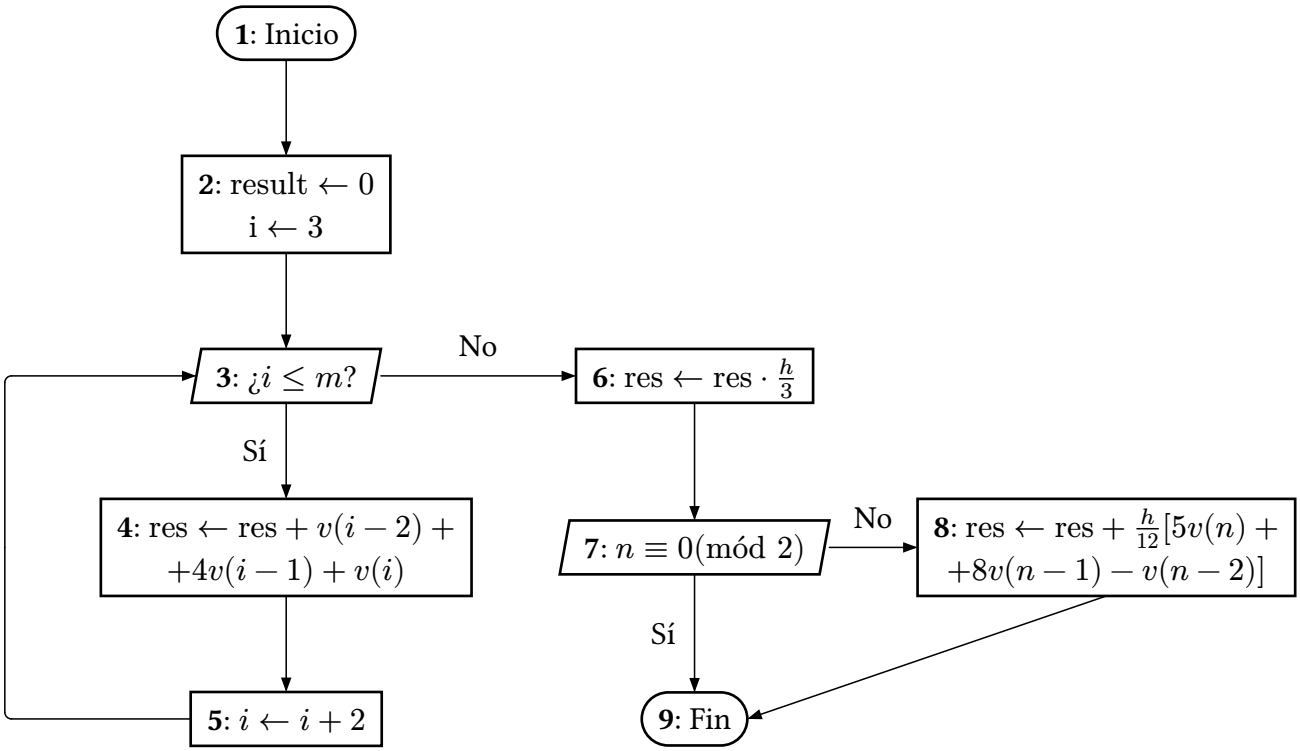
Además, los valores f_k vienen dados por el vector de valores uniformememente espaciados de entrada $v(k+1) = f_k$, $v \in \mathbb{R}^{N-1}$.

4.3.6.3. Algoritmo

El algoritmo es similar al de una acumulación. Dado un vector de m elementos $v \in \mathbb{Q}^m$ con elementos igualmente espaciados en $h \in \mathbb{Q}$ unidades, se deduce el algoritmo de la Figura 4.7. Nótese que el paso **7** comprueba que m es par en vez de impar, porque si hay m valores hay $m-1$ intervalos.

Figura 4.7

Regla de Simpson para datos uniformemente espaciados



4.3.6.4. Problemas

Nótese en la Figura 4.7 que en vez de indexar en el paso **4** con $v(i)$, $v(i+1)$ y $v(i+2)$, se empeiza a contar en 3 y se indexa en $v(i-2)$, $v(i-1)$ y $v(i)$ respectivamente. Esto se debe a que comprobar que no desborda en el límite superior del vector es más complicado que empezar después. Se pueden ver las propiedades del bucle utilizando inducción:

1. **Caso base**: Al inicio del bucle $i = 3$, y el indexado del vector está definido para $v(i-2) = v(1)$, $v(i-1) = v(2)$ y $v(i) = v(3)$. Porque además se sabe que $i \leq m$.
2. **Hipótesis inductiva**: En la n-ésima iteración, $i = 3 + 2n \leq m$ y los valores $v(i-2) = v(2n+1)$, $v(i-1) = v(2n+2)$ y $v(i) = v(2n+3)$ suponemos que están definidos.
3. **Tesis inductiva**: Para la $n+1$-ésima iteración, suponiendo que $i \leq m$ (si no, habría salido del bucle en la bifurcación del paso **3**), $i = 3 + 2(n+1) = 5 + 2n \leq m$. Como $i \leq m$, está definido $v(i) = v(5+2n)$, y por lo tanto están definidos $v(i-1) = v(4+2n)$ y $v(i-2) = v(3+2n)$, porque $0 < 3 + 2n < 4 + 2n < 5 + 2n \leq m$.

Los problemas están en los pasos:

- **Paso 5**: $i \in \mathbb{I}_B$ puede desbordar, eso hace necesario que $m \leq 2^{B-1} - 3$.
- **Paso 8**: Si $m < 3$, no se puede indexar el vector, eso hace necesario que $m \geq 3$.

Que como se ve se puede solucionar con la precondición $3 \leq m \leq 2^{B-1} - 3$.

4.3.6.5. Análisis de punto fijo

En punto fijo puede desbordar en el **paso 4** y en el **paso 8** de la Figura 4.7, hay que replantear el problema como una reducción utilizando la acumulación mostrada en la Sección 4.3.2, la cual sí está demostrada.

Lo primero es reescribir la ecuación en dos sumas, una de los elementos pares y otra de los elementos impares, ya que se puede ver que $2i \equiv 0 (\text{mód } 2)$, $2i + 2 \equiv 0 (\text{mód } 2)$ y $2i + 1 \not\equiv 0 (\text{mód } 2)$ de la siguiente manera.

$$
\begin{aligned}
\int_a^b f(x)dx &\approx \varphi + \frac{h}{3} \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} \left[f_{2i} + 4f_{2i+1} + f_{2i+2} \right] \\
&= \varphi + \frac{h}{3} \left[\sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i} + 4 \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i+1} + \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i+2} \right] \\
&= \varphi + \frac{h}{3} \left[\sum_{i=1}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i} + f_0 + 4 \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i+1} + \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} f_{2i} \right] \\
&= \varphi + \frac{h}{3} \left[\sum_{i=1}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i} + f_0 + 4 \sum_{i=0}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i+1} + \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor - 1} f_{2i} + f_{N'} \right]
\end{aligned}
$$

donde N' depende de la paridad de N, si N es par $N' = N$, si no $N' = N - 1$, la expresión que queda es la siguiente:

$$\int_a^b f(x)dx \approx \varphi + \frac{h}{3}\left[2\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} f_{2i} + 4\sum_{i=0}^{\lfloor\frac{N}{2}\rfloor-1} f_{2i+1} + f_0 + f_{N'}\right]$$

Para simplificar más adelante las demostraciones, es preciso que ambos sumatorios empiecen y terminen en el mismo valor:

$$\int_a^b f(x)dx \approx \varphi + \frac{h}{3}\left[2\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} f_{2i} + 4\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} f_{2i-1} + f_0 + f_{N'} + 4f_{N''}\right]$$

donde N'' depende de la paridad de N, si N es impar $N'' = N$, si no, $N' = N - 1$.

Cambiando los índices f_k por el vector $v \in \mathbb{X}_{b,f}, v(k+1) = f_k$ que indexa a partir de 1, $n = N + 1$, $n' = N' + 1$, $n'' = N'' + 1$:

$$\int_a^b f(x)dx \approx \varphi + \frac{h}{3}\left[2\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i+1) + 4\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i) + v(1) + v(n') + 4v(n'')\right]$$

Se puede ver que ambas sumas tienen la misma cantidad de sumandos, sea s el número de sumandos de ambas sumas que viene dado por la expresión:

$$s = \left\lfloor\frac{N}{2} - 1\right\rfloor - 1 + 1 = \left\lfloor\frac{n-1}{2} - 1\right\rfloor = \left\lfloor\frac{n-3}{2}\right\rfloor$$

Sea $o \in \mathbb{X}^s_{b,f}$ y $e \in \mathbb{X}^s_{b,f}$ los vectores que contienen los elementos en posiciones (contando desde 1) impares (excepto el primer y último elementos impares) y pares (excepto el último elemento par) de v respectivamente.

$$o(i) = v(2i+1), i = 1, 2, ..., \left\lfloor\frac{N}{2} - 1\right\rfloor$$

$$e(i) = v(2i), i = 1, 2, ..., \left\lfloor\frac{N}{2} - 1\right\rfloor$$

De acuerdo con el Teorema 4.3.2 como $s \in \mathbb{I}_B$, entonces $\sum_{i=1}^s o(i)$, $\sum_{i=1}^s e(i) \in \mathbb{X}_{B+b-1,f}$ y además $\sum_{i=1}^s o(i)$, $\sum_{i=1}^s e(i) \in \left[-s2^{b-1}2^f, s\left(2^{b-1}-1\right)2^f\right]$. Además $2\sum_{i=1}^s o(i) \in \mathbb{X}_{B+b,f}$ y $2\sum_{i=1}^s o(i) \in \left[-2s2^{b-1}2^f, 2s\left(2^{b-1}-1\right)2^f\right]$, según el Teorema 4.2.5 con $2 \in \mathbb{X}_{2,0}$; y de la misma manera $4\sum_{i=2}^s e(i) \in \mathbb{X}_{B+b+1,f}$ y $4\sum_{i=1}^s e(i) \in \left[-4s2^{b-1}2^f, 4s\left(2^{b-1}-1\right)2^f\right]$ por el mismo Teorema 4.2.5 con $4 \in \mathbb{X}_{3,0}$.

La suma de ambas sumas también está acotada:

$$2\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i+1) + 4\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i) = 2\sum_{i=1}^{s} o(i) + 4\sum_{i=1}^{s} e(i) \in \left[-6s2^{b-1}2^f, 6s\left(2^{b-1}-1\right)2^f\right]$$

Los sumandos que faltan son $v(1) + v(n') + 4v(n'')$, como $v \in \mathbb{X}^m_{b,f}$, se puede ver las suma de esos tres valores como si fuera $6x$ para algún $x \in \mathbb{X}_{b,f}$, como $6 \in \mathbb{X}_{3,0}$ de acuerdo con el Teorema 4.2.5:

$$v(1) + v(n') + 4v(n'') \in \left[-6 \cdot 2^{b-1}2^f, 6\left(2^{b-1}-1\right)2^f\right]$$

Luego la suma

$$2\sum_{i=1}^{s} o(i) + 4\sum_{i=1}^{s} e(i) + v(1) + v(n') + 4v(n'') \in \left[-6(s+1)2^{b-1}2^f, 6(s+1)\left(2^{b-1}-1\right)2^f\right]$$

también está acotada. Si se divide el resultado entre $3 \in \mathbb{X}_{2,0}$ también sigue estando acotado:

$$\begin{aligned}\sigma &= \left[2\sum_{i=1}^{s} o(i) + 4\sum_{i=1}^{s} e(i) + v(1) + v(n') + 4v(n'')\right] \div 3 \\ &\in \left[-2(s+1)2^{b-1}2^f, 2(s+1)\left(2^{b-1}-1\right)2^f\right]\end{aligned}$$

Queda por añadir φ, que también está acotada:

$$\varphi = \begin{cases} \frac{h}{12}(5v(n) + 8v(n-1) - v(n-2)) & \text{, si } \ n \equiv 0(\text{mód } 2) \\ 0 & \text{, si no} \end{cases}$$

El problema es que está multiplicada por h y dificulta el análisis del punto fijo. Así que es preferible trabajar con $\varphi' = \frac{\varphi}{h}$ y luego multiplicar por h:

$$\int_a^b f(x)dx \approx h\left\{\varphi' + \frac{1}{3}\left[2\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i+1) + 4\sum_{i=1}^{\lfloor\frac{N}{2}\rfloor-1} v(2i) + v(1) + v(n') + 4v(n'')\right]\right\}$$

donde

$$\varphi' = \begin{cases} \frac{1}{12}(5v(n) + 8v(n-1) - v(n-2)) & \text{, si } \ n \equiv 0(\text{mód } 2) \\ 0 & \text{, si no} \end{cases}$$

Por razones similares a la suma de los últimos sumandos, se puede ver que $(5v(n) + 8v(n-1) - v(n-2) \in \left[-14 \cdot 2^{b-1}2^f, 14\left(2^{b-1}-1\right)2^f\right]$. Y por tanto $(5v(n) + 8v(n-1) - v(n-2) \div 12 \in \left[-2 \cdot 2^{b-1}2^f, 2\left(2^{b-1}-1\right)2^f\right]$. Nótese que no se ha decidido utilizar $\frac{14}{12}$ como factor y se ha optado por 2, porque simplifica después añadir φ' al sumando. Además, obviamente $0 \in \left[-2 \cdot 2^{b-1}2^f, 2\left(2^{b-1}-1\right)2^f\right]$. De ahí:

$$\sigma + \varphi' \in \left[-2(s+2)2^{b-1}2^f, 2(s+2)\left(2^{b-1}-1\right)2^f\right]$$

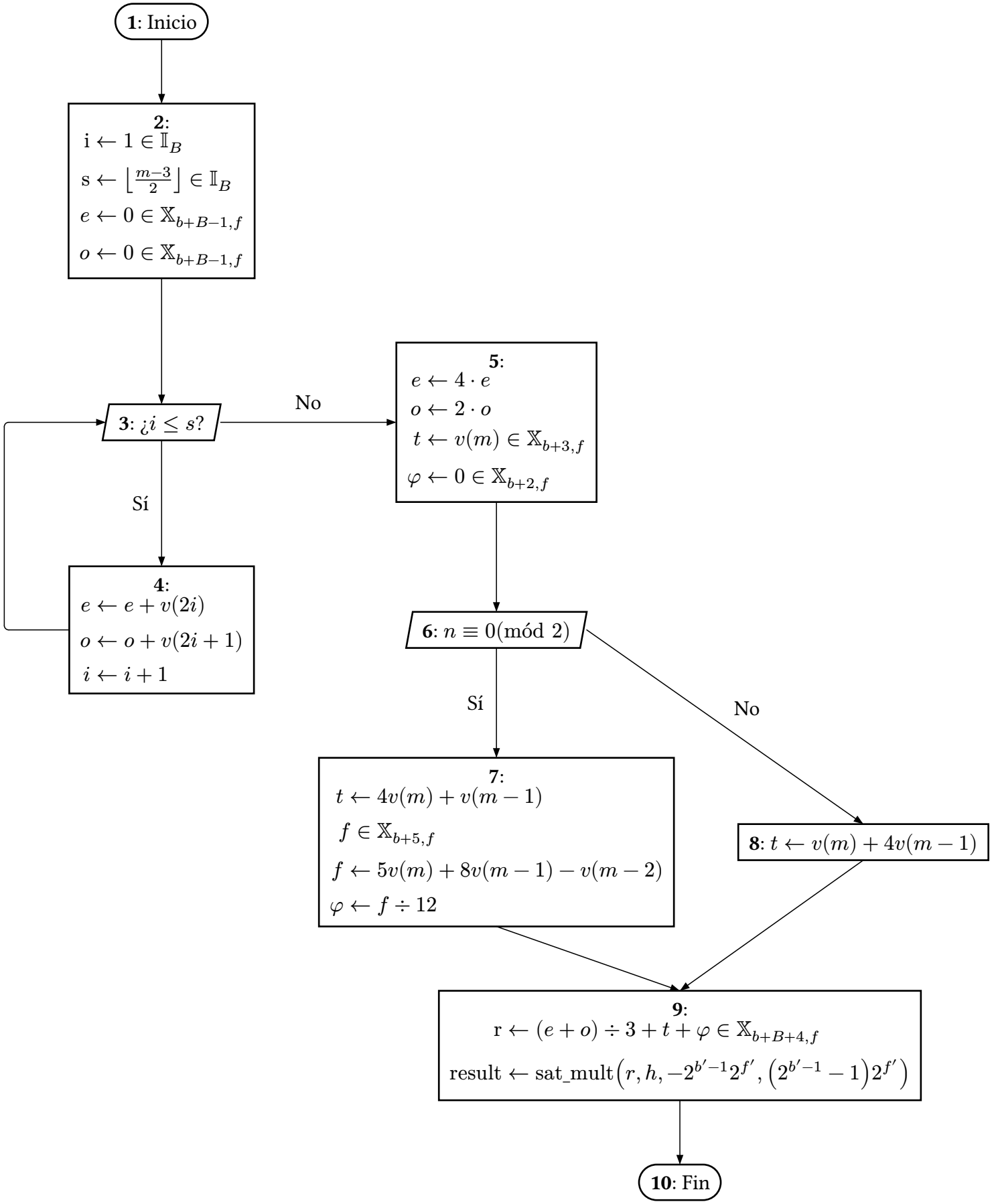
Lo único que faltaría sería multiplicar $\sigma + \varphi'$ por h. Sin embargo, el producto complica demasiado la demostración, así que se ha decidido utilizar un producto saturado, es decir:

$$\text{sat\_mult}(x, y, l, h) = \begin{cases} h & \text{, si } \ xy > h \\ l & \text{, si } \ xy < l \\ xy & \text{, si no} \end{cases}$$

Ya que en la práctica si el resultado de Simpson es demasiado grande, supera el valor del *batch* de la densidad espectral de potencia y ya clasificaría como «no ataque». El algoritmo definitivo se ve en la Figura 4.8.

Figura 4.8

Regla de Simpson para datos uniformemente espaciados para punto fijo



4.3.7. Transformada rápida de Fourier (FFT)

4.3.7.1. Transformada de Fourier discreta

Dado un vector $x \in \mathbb{C}^m$ de m números complejos $x_1, x_2, ..., x_m \in \mathbb{C}$, la transformada discreta de Fourier (DFT) se define como un vector $X \in \mathbb{C}^m$ de m números complejos $X_1, X_2, ..., X_m \in \mathbb{C}$ que vienen dados por la fórmula:

$$X_k = \sum_{n=0}^{m-1} x_{n+1} e^{\frac{-2\pi j}{m} kn}$$

donde $j = \sqrt{-1} \in \mathbb{C}$ es la unidad imaginaria.

4.3.7.2. Algoritmo de Cooley-Tukey – Transformada rápida de Fourier

Es un algoritmo del tipo «divide y vencerás» que se pude combinar el resultado de la transformada de Fourier del vector de los elementos en posiciones pares y los de posiciones impares. Sin embargo, solo funciona para vectores con m elementos donde $m = 2^k$ para algún $k \in \mathbb{N}$, es decir, potencias de dos. Se deduce de la siguiente forma:

$$\begin{aligned} X_k &= \sum_{n=0}^{m-1} x_{n+1} e^{\frac{-2\pi j}{m} kn} \\ &= \sum_{n=0}^{\frac{m}{2}-1} x_{2n} e^{\frac{-2\pi j}{m} k(2n)} + \sum_{n=0}^{\frac{m}{2}-1} x_{2n+1} e^{\frac{-2\pi j}{m} k(2n+1)} \\ &= \sum_{n=0}^{\frac{m}{2}-1} x_{2n} e^{\frac{-2\pi j}{m} k(2n)} + e^{\frac{-2\pi j}{m} k} \sum_{n=0}^{\frac{m}{2}-1} x_{2n+1} e^{\frac{-2\pi j}{m} k(2n)} = E_k + e^{\frac{-2\pi j}{m} k} O_k \end{aligned}$$

También se puede llegar a demostrar que:

$$X_{k+\frac{N}{2}} = E_k - e^{\frac{-2\pi j}{m} k} O_k$$

por la periodicidad de la exponencial compleja.

El algoritmo se puede describir de manera recursiva:

$$\mathrm{FFT}(v, m) = \begin{cases} \mathrm{DFT}(v) & \text{, si } \ m \not\equiv 0 (\text{mód } 2) \\ \begin{cases} v_k = & \mathrm{FFT}(\text{pares}(v), \frac{m}{2})(k) + e^{\frac{-2\pi j}{m} k} \ \mathrm{FFT}(\text{impares}(v), \frac{m}{2})(k) \\ v_{k+\frac{m}{2}} = & \mathrm{FFT}(\text{pares}(v), \frac{m}{2})(k) - e^{\frac{-2\pi j}{m} k} \ \mathrm{FFT}(\text{impares}(v), \frac{m}{2})(k) \end{cases} k < \frac{m}{2} & \text{, si no} \end{cases}$$

Donde pares(v) e impares(v) son dos funciones que retornan dos vectores con los elementos del vector v en las posiciones pares e impares resepectivamente.

4.3.7.3. Recursiva a iterativa

El algoritmo se define de manera recursiva, pero tiene inconvenientes. En primer lugar, cada vez que se llama a la función el tamaño de la pila de memoria del sistema incrementa pues necesita espacio adicional para almancenar los resultados parciales. Esto podría provocar que desborde la pila cuando recurra mucho, especialmente en un dispositivo empotrado.

El segundo problema es con SPARK, es complicado para el probador de teoremas hacer demostraciones de funciones recursivas. Y mucho menos con una función tan compleja como es la de la transformada de Fourier. Así que es preciso «iterativizar» el algoritmo.

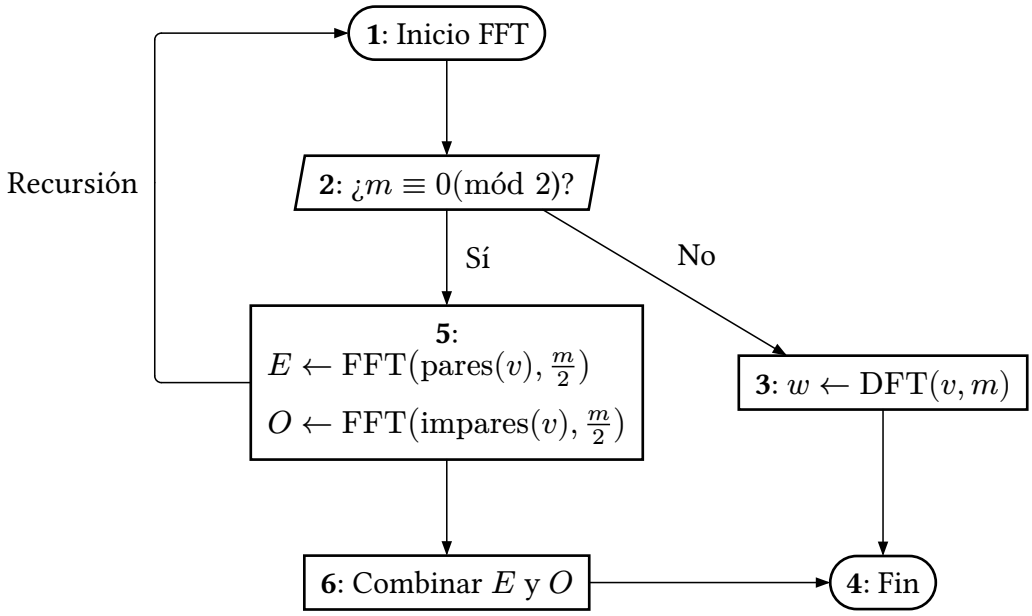
Para deducir cómo se podría implementar lo primero es definir los parámetros:

- $m = p2^q > 1$, es la longitud del vector de entrada y salida, donde $p \not\equiv 0(\text{mód } 2)$, $p \in \mathbb{N}^+$ y $q \in \mathbb{N}$.
- $v = (v_0, v_1, ..., v_{m-1}), v \in \mathbb{C}^m$ es el vector de entrada.
- $w = (w_0, w_1, ..., w_{m-1}), w \in \mathbb{C}^m$ es el vector de salida.

Para simular recursión utilizando iteración, primero se ejecuta el caso base y luego se ejecuta un bucle. Hasta ahora el algoritmo tiene la forma del diagrama de flujo de la Figura 4.9.

Figura 4.9

Diagrama de flujo de la transformada rápida de Fourier recursiva



En el paso recursivo se ve que el vector se divide en dos vectores de la mitad de tamaño, uno con los elementos en posiciones pares (empezando por cero) y posiciones impares. En cada paso recursivo siempre que sea múltiplo de dos, se divide en pares e impares.

- En la primera iteración se divide v en dos vectores v_0 y v_1: $v_{0_i} = v_{2i}$ y $v_{1_i} = v_{2i+1}$, es decir, del vector v los elementos en posiciones $i \equiv 0(\text{mód } 2)$ e $i \equiv 1(\text{mód } 2)$ respectivamente.
- En la segunda iteración (si siguen siendo múltiplos de dos) se dividen de nuevo ambos vectores v_0 y v_1 en pares e impares: de v_0 salen $v_{0,0}$, $v_{0,1}$ y de v_1 salen $v_{1,0}$ y $v_{1,1}$.

- $v_{0,0}$ y $v_{0,1}$ respecto a v_0 están en posiciones $i \equiv 0(\text{mód } 2)$ e $i \equiv 1(\text{mód } 2)$ respectivamente, pero en relación al vector v original están en posiciones $i \equiv 0(\text{mód } 2)$ e $i \equiv 2(\text{mód } 2)$ respectivamente.
- Del mismo $v_{1,0}$ y y $v_{1,1}$ respecto a v_1 están en posiciones $i \equiv 0(\text{mód } 2)$ e $i \equiv 1(\text{mód } 2)$ respectivamente, pero en relación al vector v original están en posiciones $i \equiv 1(\text{mód } 2)$ e $i \equiv 3(\text{mód } 2)$ respectivamente.

Se empieza a ver un patrón.

Para simplificar el problema, vamos a utilizar secuencias de índices. El vector w de m elementos tiene todos los elementos en los índices $w = (0, 1, ...m-1)$. En la segunda iteración se toman los índices en posiciones pares $w_0 = (0, 2, 4, ..., m-2)$; y los impares conjunto $w_1 = (1, 3, 5, ..., m-1)$, siempre y cuando su cardinalidad sea par: $|w| \equiv 0(\text{mód } 2)$.

Se denota el vector $w_{a_1,a_2,...,a_k}$ con $a_i \in \{0, 1\}, i = 1, 2, ..., k$ al vector que ha tomado los elementos de paridad a_k del vector $w_{a_1,a_2,...,a_{k-1}}$. El vector $w_{a_1,a_2,...,a_k}$ da la secuencia ordenada $\left(i : i \equiv \sum_{i=1}^{k} 2^{i-1} a_i (\text{mód } 2^k), i \in 0, 1, ...m-1\right)$.

Demostración.

- **Hipótesis de inducción**: En la k-ésima iteración si $m \equiv 0(\text{mód } 2^k)$, el vector $w_{a_1,a_2,...,a_k} = \left(i : i \equiv \sum_{i=1}^{k} 2^{i-1} a_i (\text{mód } 2^k), i \in \mathbb{Z}\right)$. Sea $f = \sum_{i=1}^{k} 2^{i-1} a_i$, entonces $w_{a_1,a_2,...,a_k} = \left(f, 2^k + f, 2 \cdot 2^k + f, 3 \cdot 2^k + f, ...\right)$.
- **Tesis inductiva**: La $k+1$-ésima iteración, puede ser uno de dos vectores dependiendo si se divide en pares o impares:
  - **Pares**: El vector $w_{a_1,a_2,...,a_k,0}$, con $a_{k+1} = 0$ tiene los valores en posiciones pares del vector $w_{a_1,a_2,...,a_k} = \left(f, 2^k + f, 2 \cdot 2^k + f, 3 \cdot 2^k + f, ...\right)$. Es decir, los elementos $\left(f, 2 \cdot 2^k + f, 4 \cdot 2^k + f, ...\right) = \left(f, 1 \cdot 2^{k+1} + f, 2 \cdot 2^{k+1} + f, ...\right)$, Que cumplen el predicado:

$$
\begin{aligned}
&\left(i : i \equiv f (\text{mód } 2^{k+1}), i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv \sum_{i=1}^{k} 2^{i-1} a_i (\text{mód } 2^{k+1}), i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv 2^k a_{k+1} + \sum_{i=1}^{k} 2^{i-1} a_i (\text{mód } 2^{k+1}), i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv \sum_{i=1}^{k+1} 2^{i-1} a_i (\text{mód } 2^{k+1}), i = 0, 1, ..., m-1\right)
\end{aligned}
$$

  - **Impares**: De manera similar el vector $w_{a_1,a_2,...,a_k,1}$, con $a_{k+1} = 1$ tiene los valores en posiciones impares del vector $w_{a_1,a_2,...,a_k} = \left(f, 2^k + f, 2 \cdot 2^k + f, 3 \cdot 2^k + f, 4 \cdot 2^k, ...\right)$. Es decir, los elementos $\left(1 \cdot 2^k + f, 3 \cdot 2^k + f, 5 \cdot 2^k + f, 7 \cdot 2^k, ...\right)$ $= \left(2^k + f, 1 \cdot 2^{k+1} + 2^k + f, 2 \cdot 2^{k+1} + 2^k + f, 3 \cdot 2^{k+1} + 2^k + f, ...\right)$. Que cumplen el predicado:

$$
\begin{aligned}
&\left(i : i \equiv 2^k + f \pmod{2^{k+1}}, i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv 2^k + \sum_{i=1}^{k} 2^{i-1} a_i \pmod{2^{k+1}}, i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv 2^k a_{k+1} + \sum_{i=1}^{k} 2^{i-1} a_i \pmod{2^{k+1}}, i = 0, 1, ..., m-1\right) \\
&= \left(i : i \equiv \sum_{i=1}^{k+1} 2^{i-1} a_i \pmod{2^{k+1}}, i = 0, 1, ..., m-1\right)
\end{aligned}
$$

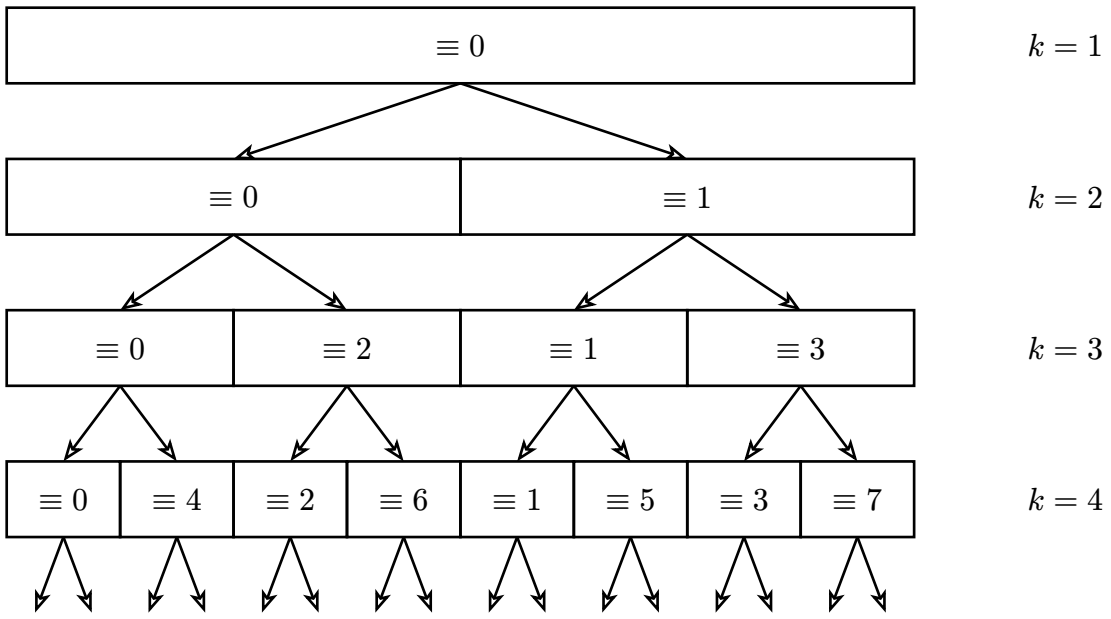
- **Caso base**: En la primera iteración cuando $k = 0$, $w = (0, 1, 2, 3, ..., m-1)$.

□

Esta relación se puede ver gráficamente como muestra la Figura 4.10. Donde las dos flechas que salen de cada cuadrado significa que dividide el vector en dos partes con los elementos en las posiciones $\equiv i \pmod{2^k}$ en la k-ésima iteración.

Figura 4.10

Relación entre el resto y la iteración en que está la transformada rápida de Fourier



El problema de esta aproximación es que para hacerlo de manera iterativa, se empieza abajo (en el caso base) y se va subiendo hacia arriba combinando los resultados. Supondría tener que cambiar los elementos de orden en el vector, pues con los elementos ordenados en los índices: $0, 1, 2, 3, 4, 5, ..., m-3, m-2, m-1$.

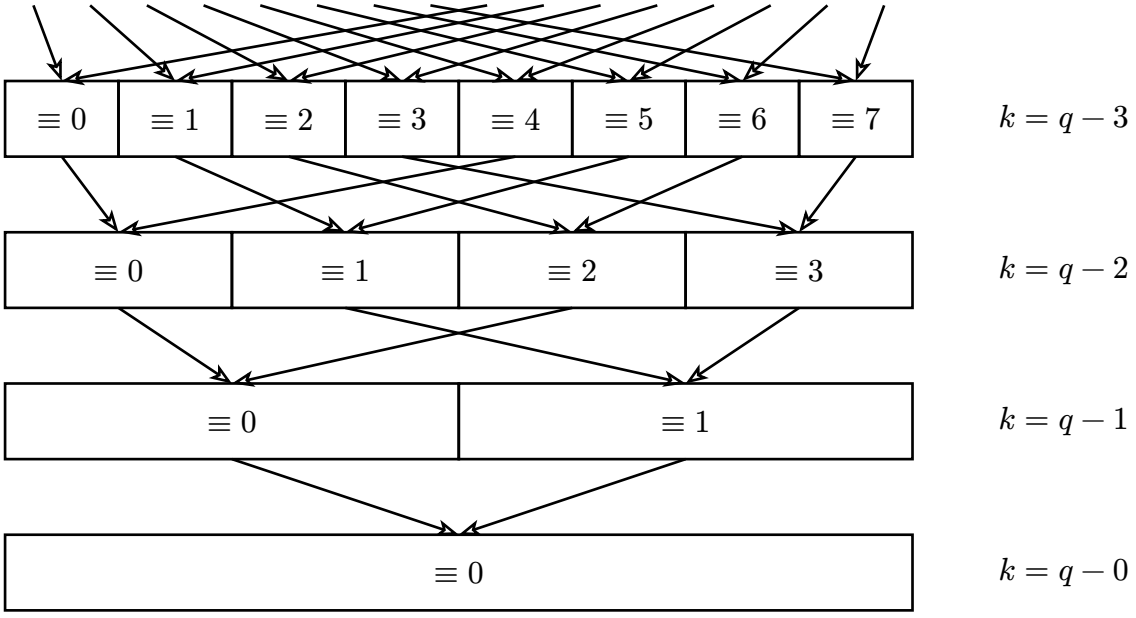
Se propone una alternativa que se puede ver en la Figura 4.11, donde las flechas indican que se combina la solución (utilizando $X_n = E_n + e^{\frac{-2\pi j}{2^k} n} O_n$) en el paso recursivo de la función original. Se puede ver que en la k-ésima iteración en el n-ésimo

bloque se combina la solución del: n-ésimo bloque de la $k-1$-ésima iteración con el $n+2^{k-1}$-ésimo bloque de la $k-1$-ésima iteración.

Esto se debe a que se definieron los índices previamente como $\left(i : i \equiv \sum_{i=1}^{k} 2^{i-1}a_i \left(\text{mód } 2^k\right), i \in 0, 1, ..., m-1\right)$ para el vector de índices $w_{a_1,a_2,...,a-k}$ en la k-ésima iteración. Si el siguiente era impar entonces tendría los elementos que sean $\equiv 2^k + f$ mód $\left(2^{k+1}\right)$ y si no eran los elementos que fueran $\equiv f\left(\text{mód } 2^{k+1}\right)$, donde $f = \sum_{i=1}^{k} 2^{i-1}a_i$.

Figura 4.11

Propuesta de índices para la transformada de Fourier



Se puede ver en la Figura 4.11 que en la k-ésima iteración, el n-ésimo bloque tiene el doble de elementos que en la iteración anterior, en este caso $p2^k$ elementos, donde p era la parte impar de la longitud del vector ($m = p2^q$), y se computa combinando las soluciones de los bloques n y $n + 2^{k-1}$.

Llamemos «bloque final» a este n-ésimo bloque que combina las soluciones del n-ésimo bloque de la iteración anterior («bloque izquierdo») y el $n + 2^{k-1}$-ésimo bloque de la iteración anterior («bloque derecho»):

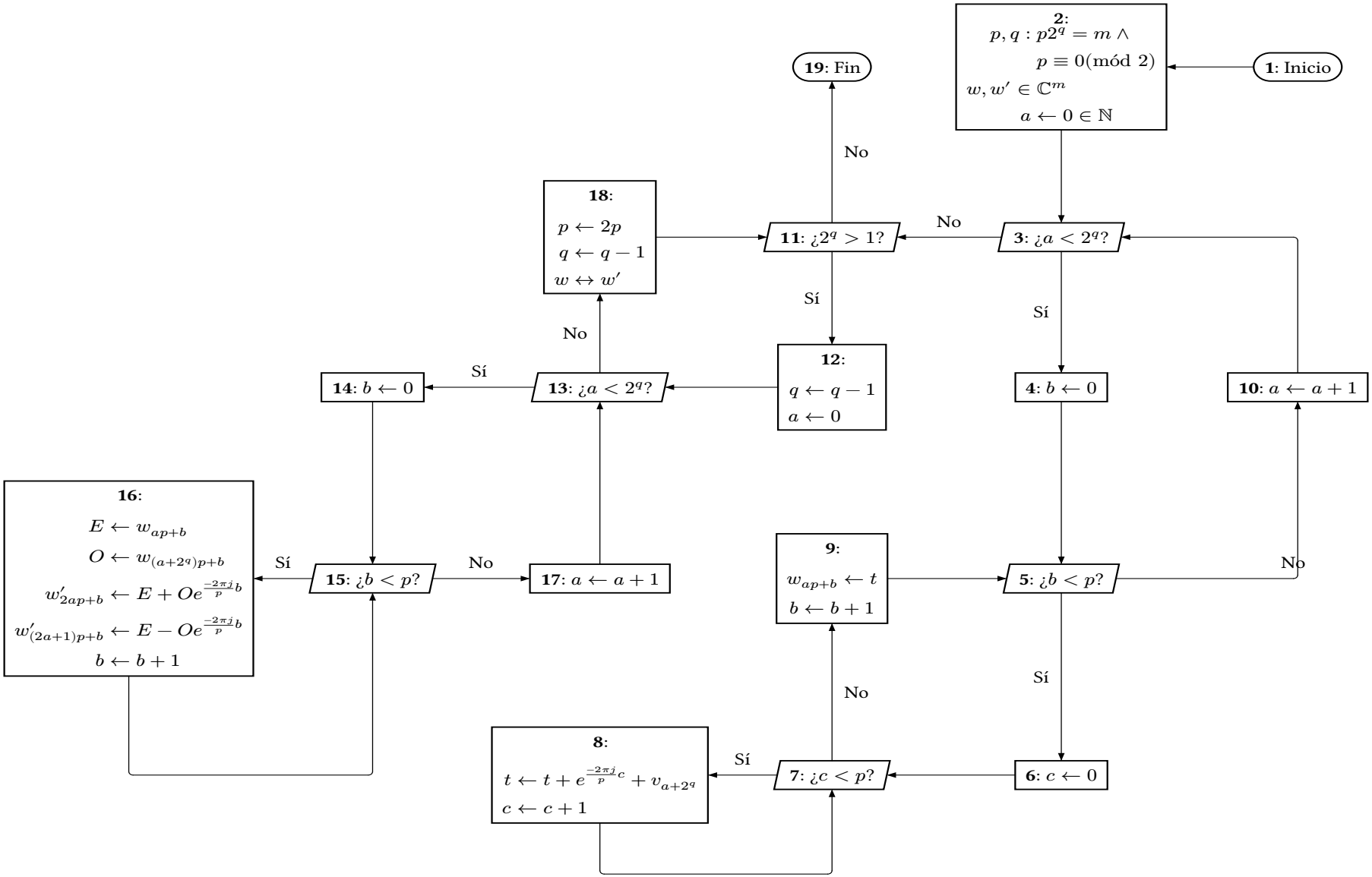
- El «bloque izquierdo» empieza en el índice $np2^{k-1}$ y tiene $p2^{k-1}$ elementos.
- El «bloque derecho» empieza en el índice $\left(n + 2^{k-1}\right)p2^{k-1}$ y tiene $p2^{k-1}$.
- El «bloque final» empieza en el índice $np2^k$ y tiene $p2^k$ elementos.

Esa es la clave para hacer que algoritmo deje de ser recursivo y sea iterativo. Véase el diagrama de flujo que muestra la Figura 4.12, para ver el resumen de cómo se podría implementar. Nótese que cuando aparece el subíndice w_i indexa desde cero $w_i = w(i+1)$.

Nótese que esto también es necesario para la transformada discreta de Fourier del caso base cuando $p > 1$, en ese caso los elementos están separados 2^q elementos.

Figura 4.12

Diagrama de flujo de la transformada rápida de Fourier iterativa



4.3.7.4. Consideraciones para punto fijo

Para punto fijo, el algoritmo desborda en el **paso 8** y en el **paso 16**, se puede demostrar que en el resto de pasos no hay errores, pero se sale del alcance de la memoria de este trabajo. Sin embargo existe un estudio del doctor P. Welch [39] que permite simplificar el desarrollo del algoritmo de la transformada de Fourier utilizando punto fijo.

Este algoritmo permite eliminar el desbordamiento en el **paso 16** del algoritmo que se muestra en la Figura 4.12 y sus técnicas se pueden aplicar para eliminar también el desbordamiento en el **paso 8**. Este algoritmo necesita que el vector esté uniformizada, es decir, con todos los elementos en el conjunto abierto $(-1, 1)$.

Teorema 4.3.3

Dados $x, y \in \mathbb{C}$, $|xy| = |x||y|$.

Demostración.

Dados $x = \mathrm{Re}\{x\} + \mathrm{Im}\{x\}j, x \in \mathbb{C}$ e $y = \mathrm{Re}\{y\} + \mathrm{Im}\{y\}j, y \in \mathbb{C}$, donde $j = \sqrt{-1}$ es la unidad imaginaria.

$$\begin{aligned} xy &= (\mathrm{Re}\{x\} + \mathrm{Im}\{x\}j)(\mathrm{Re}\{y\} + \mathrm{Im}\{y\}j) \\ &= (\mathrm{Re}\{x\}\ \mathrm{Re}\{y\} - \mathrm{Im}\{x\}\ \mathrm{Im}\{y\}) + (\mathrm{Re}\{x\}\ \mathrm{Im}\{y\} + \mathrm{Im}\{x\}\ \mathrm{Re}\{x\}j) \\ \mathrm{Re}\{xy\} &= (\mathrm{Re}\{x\}\ \mathrm{Re}\{y\} - \mathrm{Im}\{x\}\ \mathrm{Im}\{y\}) \\ \mathrm{Im}\{xy\} &= (\mathrm{Re}\{x\}\ \mathrm{Im}\{y\} + \mathrm{Im}\{x\}\ \mathrm{Re}\{y\}) \end{aligned}$$

$$|x| = \sqrt{\mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2}$$

$$|y| = \sqrt{\mathrm{Re}\{y\}^2 + \mathrm{Im}\{y\}^2}$$

$$|x||y| = \sqrt{(\mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2)(\mathrm{Re}\{y\}^2 + \mathrm{Im}\{y\}^2)}$$

$$\begin{aligned} |xy| &= \sqrt{\mathrm{Re}\{xy\}^2 + \mathrm{Im}\{xy\}^2} \\ &= \sqrt{(\mathrm{Re}\{x\}\ \mathrm{Re}\{y\} - \mathrm{Im}\{x\}\ \mathrm{Im}\{y\})^2 + (\mathrm{Re}\{x\}\ \mathrm{Im}\{y\} + \mathrm{Im}\{x\}\ \mathrm{Re}\{y\})^2} \\ &= \sqrt{\begin{array}{l}(\mathrm{Re}\{x\}^2\mathrm{Re}\{y\}^2 + \mathrm{Im}\{x\}^2\mathrm{Im}\{y\}^2 + -2\ \mathrm{Re}\{x\}\ \mathrm{Re}\{y\}\ \mathrm{Im}\{x\}\ \mathrm{Im}\{y\} \\ +\mathrm{Re}\{x\}^2\mathrm{Im}\{y\}^2 + \mathrm{Im}\{x\}^2\mathrm{Re}\{y\}^2 + 2\ \mathrm{Re}\{x\}\ \mathrm{Re}\{y\}\ \mathrm{Im}\{x\}\ \mathrm{Im}\{y\})\end{array}} \\ &= \sqrt{\mathrm{Re}\{x\}^2\mathrm{Re}\{y\}^2 + \mathrm{Im}\{x\}^2\mathrm{Im}\{y\}^2 + \mathrm{Re}\{x\}^2\mathrm{Im}\{y\}^2 + \mathrm{Im}\{x\}^2\mathrm{Re}\{y\}^2} \\ &= \sqrt{\mathrm{Re}\{x\}^2(\mathrm{Re}\{y\}^2 + \mathrm{Im}\{y\}^2) + \mathrm{Im}\{x\}^2(\mathrm{Re}\{y\}^2 + \mathrm{Im}\{y\}^2)} \\ &= \sqrt{\mathrm{Re}\{x\}^2|y|^2 + \mathrm{Im}\{x\}^2|y|^2} \\ &= \sqrt{|y|^2 + (\mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2)} \\ &= \sqrt{|y|^2 + |x|^2} \\ &= |x||y| \end{aligned}$$

□

Corolario 4.3.3.1

Del Teorema 4.3.3 se obtiene que, dados $x, y \in \mathbb{C}$, si $|x| < 1 \land |y| < 1$ entonces $|xy| < 1$.

Definición 4.3.4 (Fórmula de Euler)

$$e^{jx} = \cos(x) + j\sin(x)$$

$$e^{-jx} = \cos(x) - j\sin(x)$$

donde $j = \sqrt{-1}$.

Teorema 4.3.5 (Relación pitagórica)

$$\sin^2(\theta) + \cos^2(\theta) = 1$$

Lema 4.3.6 (Raíz de la unidad)

De la Definición 4.3.4 y el Teorema 4.3.5 se obtiene que:

$$\left|e^{jx}\right| = 1, \forall x \in \mathbb{R}$$

Teorema 4.3.7

Sea $x \in \mathbb{C}$ tal que $|x| < a$ entonces $|\mathrm{Re}\{x\}| < a \land |\mathrm{Im}\{x\}| < a$, $a \in \mathbb{R}, a \geq 1$.

Demostración.

Dado $x \in \mathbb{C}$, con $|x| < a$:

$$|x| = \sqrt{\mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2} < a$$
$$\Rightarrow \mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2 < a^2$$

Por contradicción si $|\mathrm{Re}\{x\}| \geq a$, entonces $\mathrm{Re}\{x\}^2 \geq a^2$ y como $\mathrm{Im}\{x\}^2 \geq 0$, se llega a una contradicción porque entonces $\mathrm{Re}\{x\}^2 + \mathrm{Im}\{x\}^2 \geq a^2$. Lo mismo aplica si $\mathrm{Im}\{x\} \geq a$, entonces es necesario que $|\mathrm{Re}\{x\}| < a$ y que $|\mathrm{Im}\{x\}| < a$. □

Teorema 4.3.8 (Desigualdad triangular)

Dados $x, y \in \mathbb{C}$, se cumple que $|x + y| \leq |x| + |y|$.

En el **paso 16** se computa:

$$E \pm Oe^{\frac{-2\pi j}{p}b}$$

Supongamos que se cumple antes de ejecutar el **paso 16** que $|E| < 1$ y que $|O| < 1$. Entonces por el Teorema 4.3.3 se puede ver que

$$\left|Oe^{\frac{-2\pi j}{p}b}\right| = |O|\left|e^{\frac{-2\pi j}{p}b}\right| = |O| < 1$$

Como $|E| < 1$, se puede ver, por la desigualdad triangular (Teorema 4.3.8), es cierto que:

$$\begin{aligned}\left|E \pm Oe^{\frac{-2\pi j}{p}b}\right| &\leq |E| + \left|Oe^{\frac{-2\pi j}{p}b}\right| \\ &\leq |E| + |O| \\ &< 2\end{aligned}$$

Es decir al final del **paso 16** el módulo del número complejo se duplica. Además si se aplica el Teorema 4.3.7, se ve que la parte real y parte imaginaria se duplican. Esta propiedad es muy interesante de cara a ver cómo crecen los valores en punto fijo y es lo que aprovecha P. Welch para dar tres posibles aproximaciones, de las cuales se ha decidido utilizar la más sencilla, porque si no, las demostraciones para el probador de teoremas resultarían muy complicadas.

Después del **paso 16**, se divide el número complejo entre dos, de esta manera se mantiene la invariante de que el módulo de E y O y por tanto de todos los elementos de w están en el conjunto abierto $(-1, 1)$. Así se pueden utilizar el conjunto uniforme $\mathbb{U}_b$, y al final se reescala el resultado.

4.3.8. *Welch*

El método de Welch, también conocido como el método de periodograma, se utiliza para estimar la densidad espectral y consiste en dividir la señal temporar en bloques sucesivos, formar el periodograma de cada bloque y hacer la media [40].

Se denota el m-ésimo marco al que se le ha aplicado una ventana w y que ha sido rellenado con ceros de la señal x como:

$$x_m(n) \triangleq w(n)x(n+mR), n = 0, 1, ..., M-1, m = 0, 1, ..., K-1$$

donde R es el solapamiento de la ventana, y sea K el número de marcos disponibles [40]. Y donde M es el tamaño del marco. Luego el periodograma del m-ésimo bloque viene dado por

$$P_{x_m,M}(\omega_k) = \frac{1}{M}\left|\mathrm{FFT}_{N,k}(x_m)\right|^2 \triangleq \frac{1}{M}\left|\sum_{n=0}^{N-1} x_m(n)e^{-j2\pi n\frac{k}{N}}\right|^2$$

el estimado de Welch para la densidad espectral viene dado por

$$\hat{S}_x^W(\omega_k) \triangleq \frac{1}{K}\sum_{m=0}^{K-1} P_{x_m,M}(\omega_k)$$

en otras palabras, es la media de los periodogramas a lo largo del tiempo [40].

En este caso la ventana que la implementación de referencia ha utilizado es la ventana de Hann. Para ventana de n elementos, se define la función de ventana de Hann $w_n(x)$ como:

$$w_n : \mathbb{N} \cap [0, n) \to \mathbb{R}$$

$$w_n(x) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi x}{n}\right)$$

En primer lugar, mencionar que la imagen de la función de ventana de Hann w_n es el conjunto $[0, 1]$, porque $\cos : \mathbb{R} \to [-1, 1]$ y luego $-\frac{1}{2}\cos(x) \in \left[-\frac{1}{2}, \frac{1}{2}\right], \forall x \in \mathbb{R}$. Así que no hay problemas al multiplicar por la ventana utilizando punto fijo.

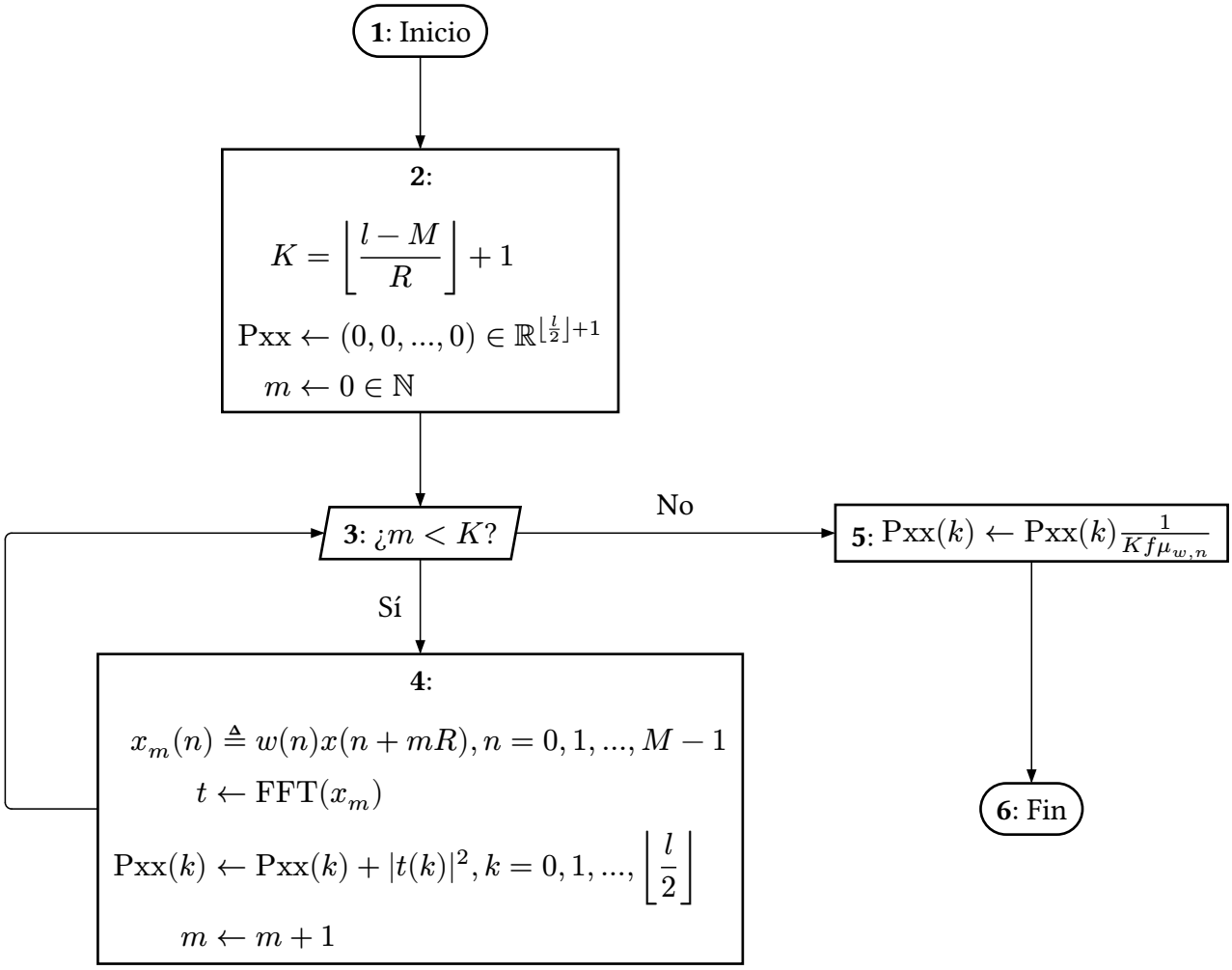
El algoritmo completo se encuentra en la Figura 4.13, dada una señal $x \in \mathbb{R}^l$ de $l \in \mathbb{N}^+$ elementos, una función ventana $w : \mathbb{N} \to \mathbb{R}$, un tamaño de marco $M \in \mathbb{N}^+, M < l$, un tamaño de solapamiento $R \in \mathbb{N}^+, R < M$ y la frecuencia de muestreo $f > 0$, $f \in \mathbb{R}$. Nótese que el vector x se indexa desde 0 en vez de 1 para este caso. Además se extiende la definición de $x(i)$ para cuando $i \geq l$, en cuyo caso $x(i) = 0, \forall i \geq l$.

En el paso final además se aplica un factor de normalización que se va a llamar $\mu_{w,n}$ que depende de la función de ventana y se calcula como:

$$\mu_{w,n} = \sum_{i=0}^{n} w(n)$$

Figura 4.13

Diagrama de flujo del método de Welch



4.3.8.1. Análisis de punto fijo

Los dos pasos más problemáticos son el de la suma del **paso 4** y la división del **paso 5**. Para solucionarlos se va a utilizar una técnica inspirada en la solución propuesta para la transformada rápida de Fourier de la Sección 4.3.7.4.

La señal que retorna la transformada de Fourier está uniformizada y está escalada, así que se puede suponer que el resultado de la transformada de Fourier es un vector de valores uniformes $\mathbb{U}_b \subset (-1, 1)$. Para no perder tanta precisión el tipo del vector *Pxx* será uno uniforme con el doble de bits, es decir $\mathbb{U}_{2b}$.

A partir de ahora se va a tener una variable de tipo entero $s \in \mathbb{I}_b$ que almacenará el valor de reescalado y que irá incrementando en cada iteración. Al final del algoritmo se reescalará multiplicando por 2^s y luego, como se explicará más adelante, desuniformizando dos veces. Para ello, la transformada de Fourier se define como:

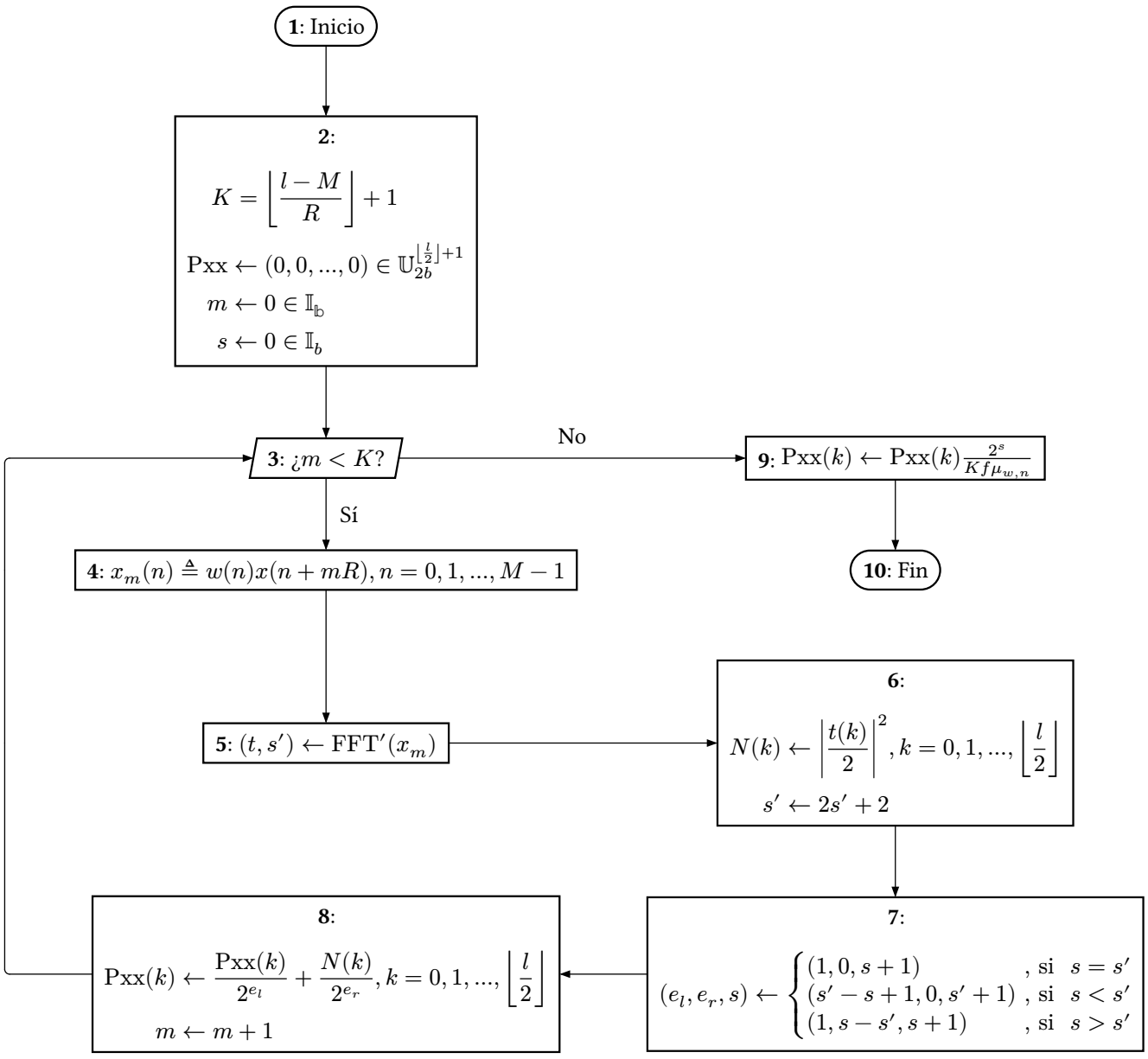
$$\text{FFT}' : \mathbb{U}_b^n \to (\mathbb{U}^n \cdot \mathbb{N})$$

$$(r, s) = \text{FFT}'(x)$$

Retorna dos valores r, que es el vector resultante de la transformada de Fourier, y s es el factor de escalado de la transformada de Fourier que dice que debe multiplicarse 2^s a cada uno de los elementos de r. Véase la Figura 4.14, con el algoritmo definitivo para punto fijo.

Figura 4.14

Diagrama de flujo del método de Welch



Es una aproximación equivalente, pero sin desbordamientos.

- **Paso 2**: $s \leftarrow 0 \in \mathbb{I}_b$, es el factor de escalado. La señal está uniformizada, pero todavía no está escalada. Nótese que al inicio del algoritmo se mantiene el predicado de que $\mathrm{Pxx}(i) \in \mathbb{U}_{2b} \subset [0,1), \forall i = 0,1,...,\lfloor \frac{l}{2} \rfloor$.
- **Paso 4**: No puede haber desbordamientos porque la señal está normalizada y al menos la ventana de Hann retorna un valor en el conjunto $[0,1]$, luego el producto también es uniforme.
- **Paso 5**: Se computa la transformada de Fourier, el resultado se almacena en la variable t y el factor de escalado, en la variable s'. La señal x_m está uniformizada, así que se puede llamar a FFT' sin problema, el resultado de dicha transformada está uniformizado y escalado entre un factor de $2^{s'}$
- **Paso 6**: En vez de calcular las normas al cuadrado de los números complejos del vector resultado de la transformada de Fourier t, se calcula un cuarto de la norma al cuadrado. Esto se debe a que $\mathrm{Re}\{t(k)\}, \mathrm{Im}\{t(k)\} \in (-1,1), \forall k = 0,1,...,\lfloor \frac{l}{2} \rfloor$, pero no sabemos si la norma es menor que uno o no, así que $|t(K)| \in [0,2), \forall k = 0,1,...,\lfloor \frac{l}{2} \rfloor$. Si se divide entre dos sus elementos la norma está en $[0,0.5)$.

  Estos valores de los cuartos de las normas al cuadrado se almancenan en el vector de las normas. El factor de escalado s' ha cambiado, como se hace el cuadrado el factor de escalado se cuadra, también se añade 2, porque se ha dividido entre 2^2 y luego hay que reescalarlo. Nótese que se podría obtener un mejor factor de escalado si en vez de utilizar el peor caso, se tomara el valor absouto máximo de las normas y se escalara de acuerdo con él.
- **Paso 7**: Los valores del vector *Pxx* en este punto están divididos entre el factor de escalado 2^s, a la hora de sumar el valor antiguo de *Pxx* con su norma N es necesario que tengan el mismo factor de escalado. En este paso se calculan el exponente del divisor de *Pxx* (e_l) y el de N (e_r), y se obtiene el factor de escalado del resultado:
  - Si $s = s'$, recordemos que $N(k) \in [0,0.5)$, pero que $\mathrm{Pxx}(k) \in [0,1)$. Así que $\mathrm{Pxx}(k)$ se debe dividir entre 2, por lo que $e_l = 1$ y e_r. El resultado estará en el rango $[0,1)$ y como se ha dividido $\mathrm{Pxx}(k)$ entre dos, el factor de escalado aumenta en uno.
  - Si $s < s'$, significa que $N(k)$ está dividido entre una potencia más grande de dos que $\mathrm{Pxx}(k)$, por lo que hay que dividir $\mathrm{Pxx}(k)$ entre la diferencia. El factor de escalado es el mayor de los dos $s' + 1$, porque $\mathrm{Pxx}(k)$ se divide entre $2^{s'-s} \cdot 2$, luego su factor de escalado será $s' - s + s + 1 = s' + 1$.
  - Si $s > s'$, es el caso contrario $\mathrm{Pxx}(k)$ se debe dividir entre dos para que esté en $[0,0.5)$, y, como está dividido entre una potencia más grande que $N(k)$, hay que dividir este último para que tenga el mismo factor de escalado.
- **Paso 8**: El valor $\frac{\mathrm{Pxx}(k)}{2^{e_i}} \in [0,0.5)$ y $\frac{N(k)}{2^{e_r}} \in [0,0.5)$, porque $e_i > 0$. Así que el nuevo valor de $\mathrm{Pxx}(k)$ estará en $[0,1)$ y por consiguiente se mantiene la invariante del bucle.

- **Paso 9**: Se aplica el factor de reescalado y se divide por los valores entre los que se dividía antes para obtener el resultado definitivo. Para este paso hay que asegurarse de que tipo del resultado pueda representarlo.

Hay que tener en cuenta que en el paso **paso 9**, el resultado está uniformizado dos veces, porque en el **paso 6** se hizo el cuadrado y por consiguiente el factor de uniformización se cuadró.

4.3.9. Densidad espectral de potencia (PSD)

Este algoritmo utiliza la función Welch (Sección 4.3.8) para computarlo. Se trata de integrar la estimación espectral de potencia utilizando el método de Simpson (Sección 4.3.6) en distintos rangos de señales. La densidad integral de potencia se utiliza como una de las características estadísticas para determinar si una época es un ataque epiléptico o no.

En el estudio se utilizaron los rangos de 2 a 12 Hz para la que llaman PSD_1, el rango de 12 a 18 Hz para la PSD_2 y el rango 18 a 35 Hz para la PSD_3. El rango del vector resultante del método de Welch que se debe integrar viene dado por la expresión:

$$(L, H) = \mathrm{rangos}(l, h) = \left(\left\lfloor l\frac{M}{s}\right\rfloor, \left\lfloor h\frac{M}{s}\right\rfloor\right)$$

Donde M era el tamaño del marco o la ventan del método de Welch (Sección 4.3.8) y s es la frecuencia de lectura de la señal (en este caso 256 muestras por segundo). Luego:

$$\mathrm{PSD}(x, l, h) = \mathrm{unif}^{-1}(\mathrm{unif}^{-1}(\mathrm{simpson}(\mathrm{welch}(x, f := s, M := M, R := M/2))))$$

Donde:

$$\mathrm{PSD}_1(x) = \mathrm{PSD}(x, 2, 12)$$

$$\mathrm{PSD}_2(x) = \mathrm{PSD}(x, 12, 18)$$

$$\mathrm{PSD}_3(x) = \mathrm{PSD}(x, 18, 35)$$

4.3.10. *Batch normalisation*

Es una técnica de normilización que se usa para mejorar el tiempo y la estabilidad de entrenamiento de redes de neuronas artificiales, pues recentra los valores alrededor del cero y los reescala, fue introducido por Sergey Ioffe and Christian Szegedy en 2015 [41].

Dada la media $\mu' = \mu(v, m) \in \mathbb{R}$ y la varianza $\sigma^2 = \mathrm{Var}(v, m)$ de un vector $v \in \mathbb{R}^m$. Se denota la normalización de *batch* como $v' = \mathrm{bnorm}(v, m)$ y se define para como:

$$v'(i) = \mathrm{bnorm}(v, m) = \frac{v(i) - \mu'}{\sqrt{\sigma^2 + \varepsilon}}$$

Donde $0 < \varepsilon < 1$ es un valor pequeño que se utiliza para dar estabilidad numérica. Como $\sigma^2 \geq 0$ y que $\varepsilon > 0$, $\sqrt{\sigma^2 + \varepsilon} > 0$ y por tanto nunca se divide entre cero.

4.3.10.1. Análisis de punto fijo

Solo se encuentran problemas cuando se trabaja con punto fijo, pues la operación del numerador puede desbordar, la suma dentro de la raíz cuadrada puede desbordar y la división puede hacer que desborde el resultado. También hay que trabajar con el cuarto de la varianza, valor que está normalizado dos veces (ver Sección 4.3.5) y al que se le va a dar como nombre

$$q = \mathrm{cuarto\_var}(v, m) = \mathrm{unif}_{b,f}\left(\mathrm{unif}_{b,f}\left(\frac{\sigma^2}{4}\right)\right)$$

La expresión queda como:

$$\begin{aligned} v'(i) = \mathrm{bnorm}(v, m) &= \frac{\mathrm{unif}_{b,f}(v)(i) - \mu\big(\mathrm{unif}_{b,f}(v, m)\big)}{\sqrt{4q + \varepsilon}} \\ &= \frac{\mathrm{unif}_{b,f}(v)(i) - \mu\big(\mathrm{unif}_{b,f}(v, m)\big)}{2} \frac{1}{\sqrt{q + \varepsilon}} \end{aligned}$$

Como el numerador está uniformizado y el denominador también lo está (al hacer la raíz cuadrada, del factor de uniformización $\left(2^{f+b-1}\right)^2$ da $\sqrt{\left(2^{f+b-1}\right)^2} = 2^{f+b-1}$, que es solo un factor de uniformización), el resultado es escalar y no está uniformizado (se cancelan los factores de uniformización).

Sea $\nu \in [0, 1)$,

$$\nu = \mathrm{unif}_{b,f}(v)(i) \div 2 - \mu\big(\mathrm{unif}_{b,f}(v, m)\big) \div 2$$

El denominador en vez de ser $\sqrt{q + \varepsilon}$, suma que puede desbordar, es preferible que sea $d \in (0, 1)$:

$$d = \begin{cases} \delta_{b,f} & \text{, si } \sqrt{q} = 0 \\ \sqrt{q} & \text{, si no} \end{cases}$$

El ε estaba para que el denominador no fuera cero, en el caso en el que la raíz cuadrada sea cero, el denominador se vuelve el valor más pequeño representable en el conjunto de punto fijo $\mathbb{X}_{b,f}$, que es $\delta_{b,f} = 2^f$.

En segundo lugar $q \in [0, 1)$ porque es uniforme como se ve al final de la Sección 4.3.4.1, eso quiere decir que $\sqrt{q} \in [0, 1)$. Además $d > 0$.

Puesto que por definición la normalización de *batch* se teóricamenete acerca los valores alrededor de cero. Para simplificar la demostración se ha decidido poner un rango finito para el resultado de las expresiones. Pues al dividir entre un denominador muy pequeño implicaría que el resultado aumenta.

Por razones prácticas se ha decidido que el rango sea $[-B, B]$ con $B = 16$, o que es lo mismo: $\exists b', f' : -16, 16 \in \mathbb{X}_{b',f'}$, por ejemplo: $\mathbb{X}_{32,-13}$ es válido para almacenar el resultado.

El valor del cociente $\frac{\nu}{d}$ siempre va crecer en valor absoluto, porque $d < 1$, para evitar el desbordamiento es preciso encontrar un predicado que preferiblemente no desborde:

- Si $\nu \geq 0$ entonces:

$$\begin{aligned} \frac{\nu}{d} &\geq 0 && [d > 0 \land \nu \geq 0] \\ \frac{\nu}{d} &\leq B && [\text{Postcondición}] \\ \nu &\leq Bd && [d > 0] \\ \frac{\nu}{B} &\leq d && [B > 0] \end{aligned}$$

  Es decir, si $\nu > 0 \land \frac{\nu}{B} \leq d$ entonces $\frac{\nu}{d} \leq B$.

- Si $\nu < 0$ entonces:

$$\begin{aligned} \frac{\nu}{d} &< 0 && [d > 0 \land \nu < 0] \\ \frac{\nu}{d} &\geq -B && [\text{Postcondición}] \\ \nu &\geq -Bd && [d > 0] \\ \frac{\nu}{-B} &\leq d && [-B < 0] \end{aligned}$$

  Es decir, si $\nu < 0 \land \frac{\nu}{-B} \leq d$ entonces $\frac{\nu}{d} \geq -B$.

El valor del resultado depende del signo y de los valores del numerador y del denominador:

$$v'(i) = \text{bnorm}(v, m) = \begin{cases} \frac{\nu}{d} & , \text{si } \nu \geq 0 \land \frac{\nu}{B} \leq d \\ B & , \text{si } \nu \geq 0 \land \frac{\nu}{B} > d \\ \frac{\nu}{d} & , \text{si } \nu < 0 \land \frac{\nu}{-B} \leq d \\ -B & , \text{si } \nu < 0 \land \frac{\nu}{B} > d \end{cases}$$

4.3.11. Deformación dinámica del tiempo (DTW)

Es un algoritmo para medir la similaridad entre dos secuencias temporales que difieren en la velocidad. El algoritmo está basado en la implementación de referencia [3].

Dadas dos señales de m elementos $u \in \mathbb{X}^m_{b'',f''}$ y $v \in \mathbb{X}^m_{b''',f'''}$. Sean $u', v' \in \mathbb{X}^m_{b,f}$ las señales *batch* normalizadas de acuerdo con la Sección 4.3.10 como $u' = \text{bnorm}(u, m)$ y $v' = \text{bnorm}(v, m)$. Sea $w \in \mathbb{N}^+, w < m$ la ventana de deformación (*warping window*). El algoritmo se resume en la Figura 4.15. Donde máx(...) es la función que retorna el valor máximo de una secuencia de valores

$$\begin{aligned} \text{máx}(x) &= x \\ \text{máx}(x, y, ...) &= \begin{cases} x & \text{, si } \; x > \text{máx}(y, ...) \\ \text{máx}(y, ...) & \text{, si no} \end{cases} \end{aligned}$$

mín(...) es la función que retorna el mínimo de una secuencia de valores

$$\begin{aligned} \text{mín}(x) &= x \\ \text{mín}(x, y, ...) &= \begin{cases} x & \text{, si } \; x < \text{mín}(y, ...) \\ \text{mín}(y, ...) & \text{, si no} \end{cases} \end{aligned}$$

$\text{dist}(a, b)$ es la función de distancia que se define como:

$$\text{dist}(a, b) = (a - b)^2$$

y M es un valor arbitrario lo suficientemente grande.

4.3.11.1. Problemas

Al igual que el resto de algoritmos, existen puntos de fallo, especialmente cuando se indexa. Sin embargo, en este caso algunas soluciones ya se han aplicado al algoritmo Figura 4.15 para simplificar la explicación.

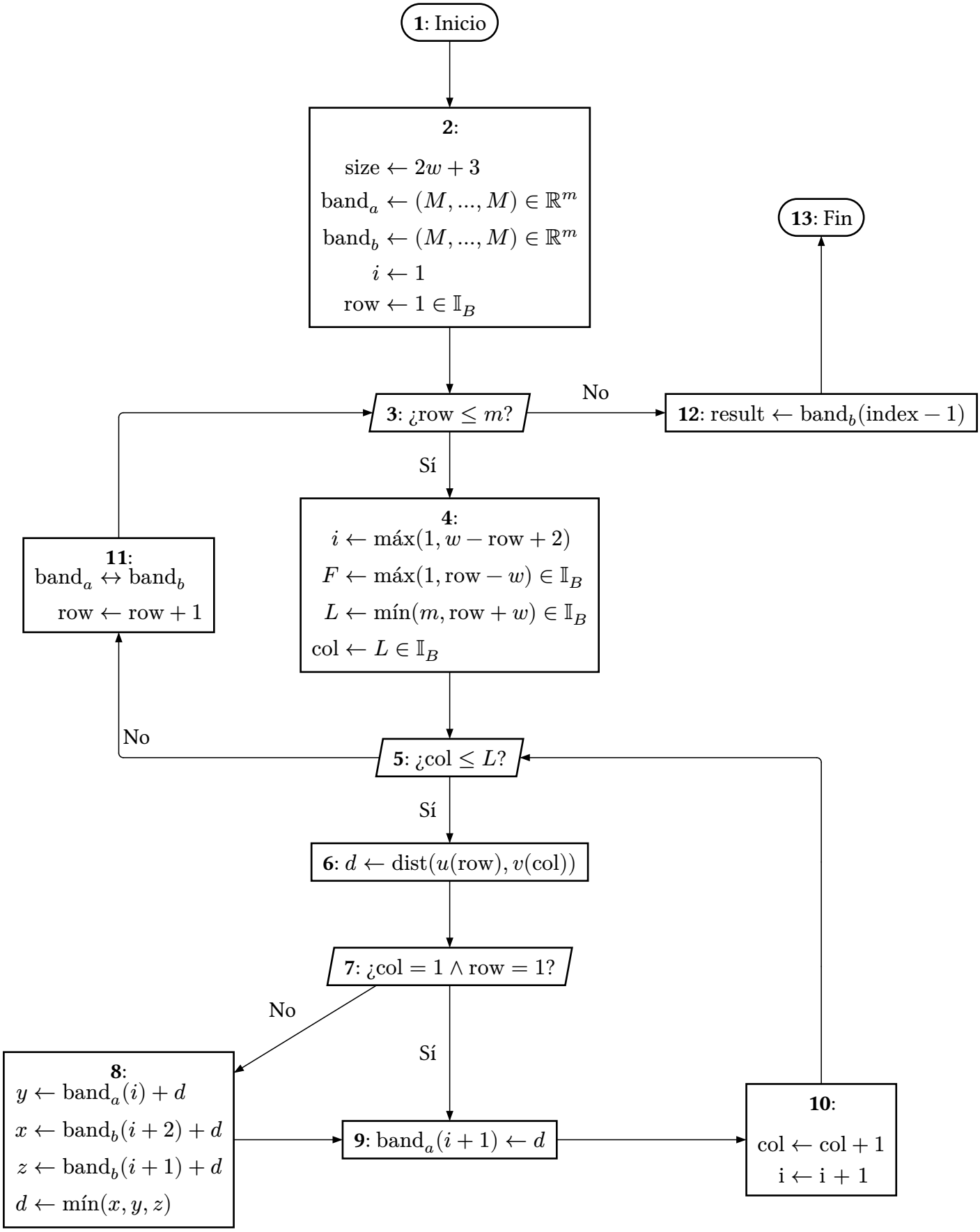
- **Paso 2**: La operación $2w + 3$ puede desbordar
- **Paso 4**: Las sumas pueden desbordar.
- **Paso 6**: Indexar los vectores $u(\text{row})$ y $v(\text{col})$ puede ser fuera de rango.
- **Paso 8**: Indexar puede fallar.
- **Paso 9**: Indexar en $\text{band}_a(i + 1)$ puede fallar.
- **Paso 10**: Incrementar cualquiera de los acumulador puede desbordar.
- **Paso 11**: Incrementar el acumulador puede desbordar.
- **Paso 12**: Indexar en $\text{band}_b(i + 1)$ puede fallar.

Además, para punto fijo hay problemas en:

- **Paso 6**: La función $\text{dist}(a, b)$ también puede desbordar.
- **Paso 8**: Cualquiera de las sumas puede y lo más seguro es que desborde, pues ambos vectores se inician con todos elementos a M, que es muy grande.

Figura 4.15

Algoritmo de la deformación dinámica del tiempo (DTW)



4.3.11.2. Soluciones

En primer lugar para que el **paso 2** no desborder en la suma $2w + 3$, basta con añadir la precondición:

$$\begin{aligned} & 2w + 3 \leq 2^{B-1} - 1 \\ \Rightarrow\ & 2w \leq 2^{B-1} - 4 \\ \Rightarrow\ & w \leq 2^{B-2} - 2 \end{aligned}$$

A continuación en el **paso 4**, pueden desbordar las operaciones: $(w - \text{row} + 2)$, $(\text{row} - w)$, $(\text{row} + w)$, eso añade más precondiciones, se sabe que $\text{row} \in [1, m]$:

$$\begin{aligned} & -2^{B-1} \leq w - \text{row} + 2 \leq 2^{B-1} - 1 \\ \Leftrightarrow\ & -2^{B-1} \leq (w+2) - \text{row} \leq 2^{B-1} - 1 && [1 \leq w \leq 2^{B-2} \Rightarrow 3 \leq (w+2) \leq 2^{B-2}] \\ \Leftrightarrow\ & -2^{B-1} \leq 3 - \text{row} \wedge 2^{B-2} - \text{row} \leq 2^{B-1} - 1 \\ \Leftrightarrow\ & -2^{B-1} \leq 3 - 2^{B-1} - 1 \wedge 2^{B-2} - 1 \leq 2^{B-1} - 1 && [1 \leq \text{row} \leq \text{m} \leq 2^{B-1} - 1] \end{aligned}$$

La expresión $(w - \text{row} + 2)$ no añade ninguna precondición nueva, porque siempre está en rango y siempre es cierto que $-2^{B-1} - 1 + 3 \leq -2^{B-1}$ y que $2^{B-2} - 1 \leq 2^{B-1} - 1$. De igual manera:

$$\begin{aligned} & -2^{B-1} \leq \text{row} - w \leq 2^{B-1} - 1 \\ \Leftrightarrow\ & -2^{B-1} \leq 1 - w \wedge 2^{B-1} - 1 - w \leq 2^{B-1} - 1 && [1 \leq \text{row} \leq m \leq 2^{B-1} - 1] \\ \Leftrightarrow\ & -2^{B-1} \leq 1 - 2^{B-2} - 2 \wedge 2^{B-1} - 1 - 1 \leq 2^{B-1} - 1 && [1 \leq w \leq 2^{B-2} - 2] \end{aligned}$$

Siempre es cierto y entonces $\text{row} - w$ no añade ninguna precondición. Y finalmente:

$$\begin{aligned} & -2^{B-1} \leq \text{row} + w \leq 2^{B-1} - 1 \\ \Leftrightarrow\ & \text{row} + w \leq 2^{B-1} - 1 && [\text{row} \geq 1 \wedge w \geq 1] \\ \Leftrightarrow\ & m + w \leq 2^{B-1} - 1 && [\text{row} \leq m] \end{aligned}$$

Que sí añade una precondición adicional, pues dicha operación puede desbordar, se deduce que:

$$m + w \leq 2^{B-1} - 1$$

El **paso 6** depende de cuál es el máximo y cuál es el mínimo del **paso 4**:

$$i = \begin{cases} 1 & , \text{si} \quad w - \text{row} + 2 \leq 1 \\ w - \text{row} + 2 & , \text{si} \quad w - \text{row} + 2 > 1 \end{cases}$$

$$F = \begin{cases} 1 & , \text{si} \quad \text{row} - w \leq 1 \\ \text{row} - w & , \text{si} \quad \text{row} - w > 1 \end{cases}$$

$$L = \begin{cases} m & , \text{si} \quad \text{row} + w \geq m \\ \text{row} + w & , \text{si} \quad \text{row} + w < m \end{cases}$$

Si se reordenan los las inecuaciones:

$$i = \begin{cases} 1 & , \text{si } \ \text{row} \geq w+1 \\ w - \text{row} + 2 & , \text{si } \ \text{row} < w+1 \end{cases}$$

$$F = \begin{cases} 1 & , \text{si } \ \text{row} \leq w+1 \\ \text{row} - w & , \text{si } \ \text{row} > w+1 \end{cases}$$

$$L = \begin{cases} m & , \text{si } \ \text{row} \geq m-w \\ \text{row} + w & , \text{si } \ \text{row} < m-w \end{cases}$$

Y si se combinan i y F en la misma expresión condicional:

$$(i, F) = \begin{cases} (w - \text{row} + 2, 1) & , \text{si } \ \text{row} < w+1 \\ (1, \quad 1) & , \text{si } \ \text{row} = w+1 \\ (1, \quad \text{row} - w) & , \text{si } \ \text{row} > w+1 \end{cases}$$

Si también se combina con L, añadiendo una **precondición adicional** $\boldsymbol{m - w > w + 1}$, se puede llegar a que

$$(i, F, L) = \begin{cases} (w - \text{row} + 2, 1, \quad \text{row} + w) & , \text{si } \ \text{row} < w+1 \\ (1, \quad 1, \quad \text{row} + w) & , \text{si } \ \text{row} = w+1 \\ (1, \quad \text{row} - w, \text{row} + w) & , \text{si } \ w+1 < \text{row} < m-w \\ (1, \quad \text{row} - w, m) & , \text{si } \ \text{row} \geq m-w \end{cases}$$

Conocer cuánto valen i, F y L en cada iteración nos permite identificar cuál es el dominio de *col* en el bucle de iteración más anidado, pues depende de los valores de L y de F:

$$L - F = \begin{cases} \text{row} + w - 1 & , \text{si } \ \text{row} < w+1 \\ \text{row} + w - 1 & , \text{si } \ \text{row} = w+1 \\ 2w & , \text{si } \ w+1 < \text{row} < m-w \\ m - \text{row} + w & , \text{si } \ \text{row} \geq m-w \end{cases}$$

Además se puede demostrar que $L - F \in [w, 2w]$, por casos:

Demostración.

- **row $< w + 1$:** Como $L - F = \text{row} + w - 1$ y como $1 \leq \text{row} < w \Rightarrow 1 + w \leq \text{row} + w < 2w \Rightarrow w \leq \text{row} + w - 1 < 2w - 1$.
- **row $= w + 1$:** Como $L - F = \text{row} + w - 1 = w + w + 1 - 1 = 2w \in [w, 2w]$
- **row $< m - w$:** Como $L - F = 2w \in [w, 2w]$.
- **row $\geq m - w$:** Como $L - F = m - \text{row} + w$ y como $m - w \leq \text{row} \leq m \Rightarrow 0 \leq m - \text{row} \leq w \Rightarrow w \leq m - \text{row} + w \leq 2w$

□

Otro valor que es necesario acotar es i y la expresión $i - F + L$, que es el valor que tendría i al final del bucle en el **paso 10**. Se puede demostrar por casos que $i \in [1..w]$ y que $i + L - F \in [w + 1, 2w + 1]$.

Demostración.

- **row $< w + 1$**:

1. $i = w - \text{row} + 2$, como $1 \leq \text{row} < w + 1 \Rightarrow -1 < w - \text{row} \leq w - 1$ $\Rightarrow 1 < w - \text{row} + 2 \leq w + 1$. $i \in [2, w + 1] \subset [1, w + 1]$
2. $i + L - F = \text{row} + w - 1 + w - \text{row} + 2 = 2w + 1 \in [w + 1, 2w + 1]$

- **row $= w + 1$**:
  1. $i = w - \text{row} + 2 = w - (w + 1) + 2 = 1 \in [1, w + 1]$.
  2. $i + L - F = 1 + \text{row} + w - 1 = 2w + 1 \in [w + 1, 2w + 1]$.
- **row $< m - w$**:
  1. $i = 1 \in [1, w + 1]$
  2. $i + L - F = 2w + 1 \in [w + 1, 2w + 1]$
- **row $\geq m - w$**:
  1. $i = 1 \in [1, w + 1]$
  2. $i + L - F = 1 + m - \text{row} + w$, como se demostró antes cuando row $\geq m - w$ entonces $w \leq L - F \leq 2w$ luego $1 + w \leq 1 + L - F \leq 2w + 1$ y $i + L - F \in [w + 1, 2w + 1]$.

□

De aquí se obtienen tres propiedades esenciales:

- El bucle itera $L - F + 1 \in [w + 1, 2w + 1]$ veces
- $i \in [1, 2w + 1]$ en cualquier punto del bucle
- col $\in [1, m]$, porque $F = \text{máx}(1, \text{row} - w)$ que es el índice inferior y se minimiza cuando $F = 1$; y $L = \text{mín}(m, \text{row} + w)$ que es el índice superior que maximiza cuando $L = m$.

El problema del **paso 6** se soluciona automáticamente porque row $\in [1, m]$ y col $\in [1, m]$, ya que u y v son vectores de m elementos.

El problema del **paso 9** se soluciona porque $i \in [1, 2w + 1]$ y las bandas son vectores de $2w + 3$ elementos e $i + 1 \in [2, 2w + 2]$. De igual manera se soluciona el **paso 8** y el **paso 12**.

El desbordamiento **paso 10** y el **paso 11** se pueden solucionar si se pone la restricción de que $m < 2^{B-1} - 1$.

Es preciso tener en cuenta que los valores en las bandas band_a y band_b son estrictamente positivos:

Demostración.

Se quiere demostrar que $\text{band}_a(k) \geq 0 \wedge \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$.

- **Paso 2**: Como $\text{band}_a(k) = M \wedge \text{band}_b(k) = M, \forall k \in 1, 2, ...m$ y como $M > 0$, se ve que es cierto.
- **Paso 3**: En la primera iteración es cierto, es necesario demostrar que para una iteración arbitraria seguirá siendo cierto suponiendo que es cierto en la iteración previa (inducción):
  - **Paso 4**: Irrelevante.

- **Paso 5**: En la primera iteración es cierto suponiendo que es cierto en el **paso 3**, es necesario demostrar que para una iteración arbitraria se mantiene la condición suponiendo que la iteración anterior es cierta (inducción):

  - **Paso 6**: $d \geq 0$, es no negativo.
  - **Paso 7**: $d \geq 0$
  - **Paso 8**: Como $d \geq 0$ y $\text{band}_a(k) = M \land \text{band}_b(k) = M, \forall k \in 1, 2, ...m$ entonces $y \geq 0$, $x \geq 0$ y $z \geq 0$ y por lo tanto $\text{mín}(x, y, z) \geq 0$.
  - **Paso 9**: $d \geq 0$ independientemente de si ha pasado por el **paso 8** y por lo tanto $\text{band}_a(i+1) \geq 0$.
  - **Paso 10**: Irrelevante.

  Si suponiendo que $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$, al final del bucle se sigue manteniendo la condición, como en la primera iteración también se cumple, por inducción se demuestra que el bucle más anidado mantiene la invariante de que: $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$.

- **Paso 11**: Intercambia el valor de band_a y band_b, se sigue manteniendo el predicado de que sean todos sus elementos positivos.

Como en la primera iteración es cierto que $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$ y para una iteración arbitraria suponiendo que es cierto $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$ se sigue manteniendo la invariante al final del bucle, por inducción se demuestra que $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$ es una invariante de los pasos 2 al 11 (todos incluidos).

□

Finalmente, queda solucionar los problemas de punto fijo. Para ello, en el **paso 8** se ha decidido utilizar suma saturada, que se define solo para números positivos pues en el como se ha visto por inducción una invariante del bucle es que $\text{band}_a(k) \geq 0 \land \text{band}_b(k) \geq 0, \forall k \in 1, 2, ...m$; y también se puede ver que $\text{dist}(a, b) \geq 0, \forall a \in \mathbb{R}, \forall b \in \mathbb{R}$:

$$\text{sat\_sum}_{b,f} : \mathbb{X}_{b,f} \cap [0, \infty) \to \mathbb{X}_{b,f} \cap [0, \infty) \to \mathbb{X}_{b,f} \cap [0, \infty)$$

$$\text{sat\_sum}_{b,f}(a, b) = \begin{cases} 2^{b-1} - 1 & , \text{ si } a > 2^{b-1} - 1 - b \\ a + b & , \text{ si no} \end{cases}$$

La función $\text{sat\_sum}_{b,f}(a, b)$ lo que hace si la suma $a + b \in \mathbb{X}_{b,f}$ entonces retorna $a + b$, si no retorna el valor más grande del conjunto que es $2^{b-1}2^f$. De esta manera es posible evitar desbordamientos. Se sustituyen las sumas del **paso 8** por sumas saturadas.

En el **paso 6**, para la función distancia se podría utilizar el producto saturado, o bien, como se ha hecho en este proyecto trabajar sobre un conjunto $\mathbb{X}_{b,f}$ de manera que el cuadrado del valor máximo y mínimo del conjunto del tipo del resultado de la función *batch normalisation* sea contenido en él.

CAPÍTULO 5

VERIFICACIÓN, VALIDACIÓN Y EVALUACIÓN

Este capítulo trata sobre la verificación y validación del *software* (Sección 5.1) , y finalmente se da una evaluación completa de los resultados obtenidos (Sección 5.2).

5.1. Verficiación y validación (*V&V*)

La verificación y validación (*V&V*) es el proceso de determinar si los requisitos para un sistema o componente están completos y son correctos, que el producto de cada fase de desarrollo cumple los requisitos o condiciones impuestas por la fase previa y que el sistema o componente finales se adhiere a los requisitos especificados [42].

A continuación una lista de tests realizados al proyecto. Estos describen cómo se han realizado, cuáles son los resultados y qué requisitos cubren. Al final de esta sección se ofrece una matriz de trazabilidad para confirmar que todos los requisitos están cubiertos.

Cabe mencionar que muchas de las propiedades del programa se pueden y han sido validadas formalmente con SPARK, que utiliza probadores de teoremas por debajo. Así que si el programa compila, implica que todas las propiedades del programa son correctas, aunque eso no significa que los resultados sean correctos, sino que es correcto el programa.

Puesto que como se ve en la Sección 3.4, todos los requisitos cubren todos los casos de uso. Solo basta con demostrar que todos los requisitos están cubiertos para validar y verificar el programa. La Tabla 5.1 muestra la plantilla que van a seguir los tests. La única prueba que no se puede realizar es la relacionada al requisito de la Tabla 3.16, pues no hay forma de asegurarse de que el modelo que va a la placa tiene esa mínima puntuación.

Tabla 5.1

Plantilla de «Test»

Campo	Descripción
Descripción	Descripción del *test*
Precondiciones	Condiciones que deben cumplirse antes de ejecutar la prueba.
Postcondiciones	Condiciones que deben cumplirse después de ejecutar la prueba.
Evaluación	Resultado de la prueba (*Correcta*, *Errónea*)
Orígenes	Requisito del que se deriva este *test*.

Tabla 5.2

Test «T-01»

Campo	Descripción
Descripción	La interfaz de Python 3 funciona.
Precondiciones	1. Existe un fichero `.so` en Linux, `.dll` en Windows, con el nombre del módulo `seizures`. 2. El comando `ldd` muestra que está enlazado con la biblioteca estándar de Python con misma versión que el intérprete en uso. 3. Desde la consola interactiva de Python 3, ejecutar `import seizures`
Postcondiciones	El intérprete de Python 3 no termina inesperadamente y se puede acceder a todas las funciones definidas desde C++, todas ellas con el mismo nombre que la implementación de referencia en Python 3.
Evaluación	Correcta
Orígenes	RF-01

Tabla 5.3

Test «T-02»

Campo	Descripción
Descripción	El detector identifica ataques.
Precondiciones	1. Se utiliza una señal sintética de 1280. 2. Se generan un *batch* específico para dicha época. Donde las características sean las características de las señal sintética y el patrón sea la propia señal. 3. Se ejecuta el algoritmo con el *batch* generado y la señal sintética.
Postcondiciones	La distancia utilizando el algortimo de deformación dinámica del tiempo debe dar una distancia cercana a 0. Y todas las características son las mismas que las del *batch*, así que están todas en rango. El detector debe notificar de que hay un ataque.
Evaluación	Correcta
Orígenes	RF-02, RF-03

Tabla 5.4

Test «T-03»

Campo	Descripción
Descripción	El detector identifica que no es un ataque por las características.
Precondiciones	1. Se utiliza una señal sintética de 1280. 2. Se genera un *batch* con patrón idéntico a la señal sintética. Los rangos de las características serán $[0, \infty)$. 3. Para cada una de las características se cambia el rango a $[0, 0)$ y a $[0, x)$, donde x es el valor computado de la característica de la señal sintética. Y se ejecuta el algoritmo con la misma señal sintética.
Postcondiciones	La distancia debe estar cerca de 0, pues es la misma señal. Para cada una de las características $[0, 0)$ es un rango vacío, así que como no está en el rango no debería notificar de que hay ataque. De la misma manera, cuando es $[0, x)$, tampoco debería notificar de que hay ataque porque $x \notin [0, x)$.
Evaluación	Correcta
Orígenes	RF-02, RF-03

Tabla 5.5

Test «T-04»

Campo	Descripción
Descripción	El detector identifica que no es atauqe por el patrón.
Precondiciones	1. Se genera un *batch* con todas las características en el rango $[0, \infty)$. Y como patrón una señal sintética sintética de 1280 muestras. 2. A continuación se genera una nueva señal sintética de 1280 muestras y se computa su deformación dinámica del tiempo (DTW) contra la señal sintética generada en el paso anterior (la del patrón). 3. Se utiliza el detector con la segunda señal sintética con dos rangos para el valor DTW del *batch*: $[0, 0)$ y $[0, x)$ donde x es el valor computado.
Postcondiciones	No hay ataque, así que no lo notifica. El caso para el intervalo vacío $[0, 0)$ siempre debe dar falso, y como el intervalo $[0, x)$ es abierto, nunca lo detecta como ataque.
Evaluación	Correcta
Orígenes	RF-02, RF-03

Tabla 5.6

Test «T-05»

Campo	Descripción
Descripción	El detector identifica ataques.
Precondiciones	1. Se generan n muestras sintéticas. 2. Se genera un *batch* con las n muestras sintéticas como patrones. Las características son los rangos $[0, x]$ donde x es el mayor valor computado para cada una de las características de cada una de las muestras sintéticas. 3. Se ejecuta el algoritmo para cada muestra sintética.
Postcondiciones	Notifica de que hay ataque en todas y cada una de ellas. La distancia de cada una de los patrones a sí mismo tiende a cero. Además no se descarta ningún patrón porque la característica máxima del rango, también es la mayor de dicha característica para cada uno de los patrones.
Evaluación	Correcta
Orígenes	RF-02, RF-03

Tabla 5.7

Test «T-06»

Campo	Descripción
Descripción	Leer de un sensor de encefalograma
Precondiciones	1. Conectar el sensor un pin de la placa.
Postcondiciones	Debe mostrar los valores leídos.
Evaluación	Correcta
Orígenes	RF-04

Tabla 5.8

Test «T-07»

Campo	Descripción
Descripción	El sensor lee a razón de 256 muestras por segundo
Precondiciones	1. El sensor está conectado. 2. Esperar a que pase un segundo 3. Comprobar cuántas muestras a leído y cada cuánto
Postcondiciones	El sensor debe haber leído un total de 256 muestras cada segundo, a una muestra por cada una ducentésima quincuagésima sexta parte de un segundo.
Evaluación	Correcta
Orígenes	RN-01, RN-02

Tabla 5.9

Test «T-08»

Campo	Descripción
Descripción	El sensor procesa como mínimo una época por segundo
Precondiciones	1. Generar una señal sintética de un $x \cdot 256, x \geq 5$ muestras. 2. Generar un *batch* con todas las características en el rango $[0, \infty)$, con tres patrones y con las distancias de la deformación dinámica del tiempo (DTW) en el rango $[0, 0)$. 3. Ejecutar el algoritmo de detección con dicha señal y dicho *batch*. 4. Comprobar cuánto tarda la época más lenta.
Postcondiciones	Es el peor caso, pues como todas las características tienen un rango $[0, \infty)$ tiene que computarlas todas, los patrones tiene que compararlos todos también porque ninguno está en rango (para que sea un ataque alguno debe estar en rango). El tiempo total debe ser menor de un segundo por época.
Evaluación	Correcta
Orígenes	RN-03, RN-12

Tabla 5.10

Test «T-09»

Campo	Descripción
Descripción	El programa no puede terminar de manera abrupta.
Precondiciones	1. El programa escrito en SPARK y Ada compila.
Postcondiciones	Si compila tenemos la certeza de que no puede saltar ninguna excepción en tiempo de ejecución, porque el probador de teoremas nos garantiza ausencia de excepciones en tiempo de ejecución.
Evaluación	Correcta
Orígenes	RN-04

Tabla 5.11

Test «T-10»

Campo	Descripción
Descripción	El error medio absoluto de `max_distance` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `max_distance` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `max_distance` es $5.25 \cdot 10^{-7} < 10^{-6}$.
Evaluación	Correcta
Orígenes	RN-06

Tabla 5.12

Test «T-11»

Campo	Descripción
Descripción	El error medio absoluto de `energy` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `energy` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `energy` es $3.21 \cdot 10^{-8} < 10^{-6}$.
Evaluación	Correcta
Orígenes	RN-07

Tabla 5.13

Test «T-12»

Campo	Descripción
Descripción	El error medio absoluto de `dtw` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `dtw` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `dtw` es $2.3 \cdot 10^{-4} < 10^{-3}$.
Evaluación	Correcta
Orígenes	RN-08

Tabla 5.14

Test «T-13»

Campo	Descripción
Descripción	El error medio absoluto de `PSD 1` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `PSD 1` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `PSD 1` es $0.34 < 0.5$.
Evaluación	Correcta
Orígenes	RN-09

Tabla 5.15

Test «T-14»

Campo	Descripción
Descripción	El error medio absoluto de `PSD 2` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `PSD 2` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `PSD 2` es $0.16 < 0.5$.
Evaluación	Correcta
Orígenes	RN-10

Tabla 5.16

Test «T-15»

Campo	Descripción
Descripción	El error medio absoluto de `PSD 3` es aceptable
Precondiciones	1. Para una señal sintética lo suficientemente larga. 2. Utilizar la versión de referencia para computar `PSD 3` para cada época.
Postcondiciones	De acuerdo con los resultado para uno de los pacientes el error de `PSD 3` es $0.42 < 0.5$.
Evaluación	Correcta
Orígenes	RN-11

Tabla 5.17

Matriz de Trazabilidad, Test vs Requisito funcionales

	RF-01	RF-02	RF-03	RF-04	RF-05
T-01	✓				
T-02		✓	✓		
T-03		✓	✓		
T-04		✓	✓		
T-05		✓	✓		
T-06				✓	
T-07					
T-08					
T-09					
T-10					
T-11					
T-12					
T-13					
T-14					
T-15					

Tabla 5.18

Matriz de Trazabilidad, Test vs Requisito no funcional

	RN-01	RN-02	RN-03	RN-04	RN-05	RN-06	RN-07	RN-08	RN-09	RN-10	RN-11	RN-12	RN-13
T-01													
T-02													
T-03													
T-04													
T-05													
T-06													
T-07	✓	✓											
T-08			✓									✓	
T-09				✓									
T-10						✓							
T-11							✓						
T-12								✓					
T-13									✓				
T-14										✓			
T-15											✓		

5.2. Evaluación

En esta sección se hace una evaluación del rendimiento del programa y un análisis del mismo. Se hace un análisis en orden cronológico y se indica las conclusiones a las que se llegó en cada paso.

5.2.1. Comparación en la validación

En un primer lugar, como se ve en la Sección 8 en la **tarea 3.3** se hicieron pruebas de rendimiento con el algoritmo después de optimizarlo y paralelizarlo. Las pruebas se realizan en dos dispositivos:

- *Slimbook*: Un portátil con un *12th Gen Intel i7-12700H (20) @ 4.600GHz* con 20 hilos lógicos.
- Servidor: Un servidor con un *Intel Xeon Gold 6326 (64) @ 3.500GHz* con 64 hilos lógicos.

Las comparación se realiza entre la de referencia, que está escrita en *Python 3* con alguna parte en C++ (la función `dtw`, deformación dinámica del tiempo); y con la que de este proyecto que está escrita en C++. Cabe destacar que ambas implementaciones están paralelizadas, la de Python 3 usa `multiprocessing` y la de C++ usa `OpenMP`.

Para la prueba se toma la señal de un paciente en bruto, en este caso el paciente CHB-MIT 6 [43] en el canal `C3-P3` con un *stride* de 256 muestras por segundo y épocas de 5 o *strides* o 1280 muestras por segundo. Que tiene un total de 61502976 muestras (66 horas, 44 minutos y 6 segundos de señal).

El campo **muestras** indica cuántas veces se ha medido cada una de las funciones, en este caso se le ha dado un tiempo máximo de ejecución a cada una de ellas de 180 segundos. Se da la media (μ) y la desviación típica (σ) de los tiempos de ejecución. La función llamada **subrutina** es la función combinada que computa *Simpson*, *PSD* y *max distance* a toda la señal. Véase la Tabla 5.19 y la Tabla 5.20:

Tabla 5.19

Slimbook: 20 hilos

Función	Muestras	Python 3		C++		*Speed-up*
		μ	σ	μ	σ	
Simpson	1159	0.1335 s	0.0109 s	0.0219 s	0.0018 s	$\times$ **6.1**
Welch	109	1.398 s	0.0323 s	0.2598 s	0.0049 s	$\times$ **5.4**
PSD	18	9.9002 s	0.1139 s	0.1596 s	0.0049 s	$\times$ **62**
Energy	172	1.0355 s	0.0298 s	0.0152 s	0.0024 s	$\times$ **68.1**
Max Distance	41	4.4274 s	0.0728 s	0.0304 s	0.0044 s	$\times$ **145.6**
Subrutina	12	15.1082 s	0.1109 s	0.2664 s	0.0398 s	$\times$ **56.7**

Tabla 5.20

Servidor: 64 hilos

Función	Muestras	Python 3		C++		*Speed-up*
		μ	σ	μ	σ	
Simpson	831	0.1845 s	0.0024 s	0.0321 s	0.0008 s	$\times$ **5.7**
Welch	75	2.0053 s	0.0048 s	0.4095 s	0.0022 s	$\times$ **4.9**
PSD	53	3.3504 s	0.2755 s	0.0704 s	0.0069 s	$\times$ **47.6**
Energy	56	3.2378 s	0.6921 s	0.0129 s	0.0021 s	$\times$ **251**
Max Distance	60	3.0195 s	0.7865 s	0.0148 s	0.0053 s	$\times$ **204**
Subrutina	21	8.5647 s	1.4619 s	0.0997 s	0.0192 s	$\times$ **85.9**

A partir de los resultados se ve que la versión de C++ es mucho más rápida que la versión de Python bajo las mismas condiciones (paralelismo con el mismo número de hilos). El incremento se ve menor en las funciones de *Simpson* y *Welch*, posiblemente porque ambas pertenecen a `SciPy` que utiliza `numpy` por debajo, que está escrito en C. La función *max distance* que está escrita en Python 3 nativo es la que mejor mejora ve como se observa en la Figura 5.1, en la Figura 5.2 y en la Figura 5.3.

Figura 5.1

Tiempo de ejecución de *Simpson* y *Welch*

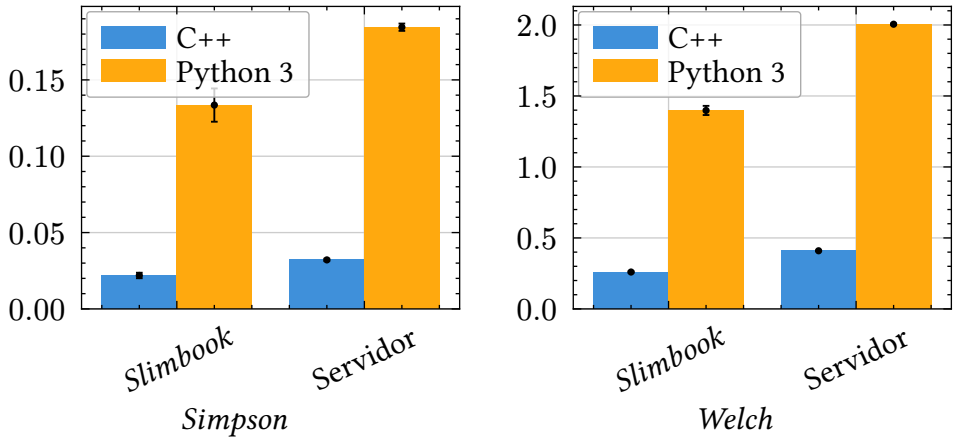


Figura 5.2

Tiempo de ejecución de *PSD* y *Energy*

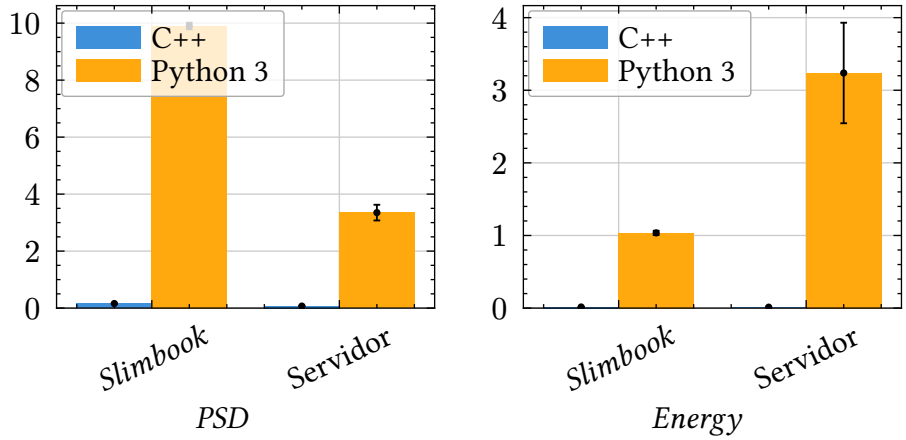
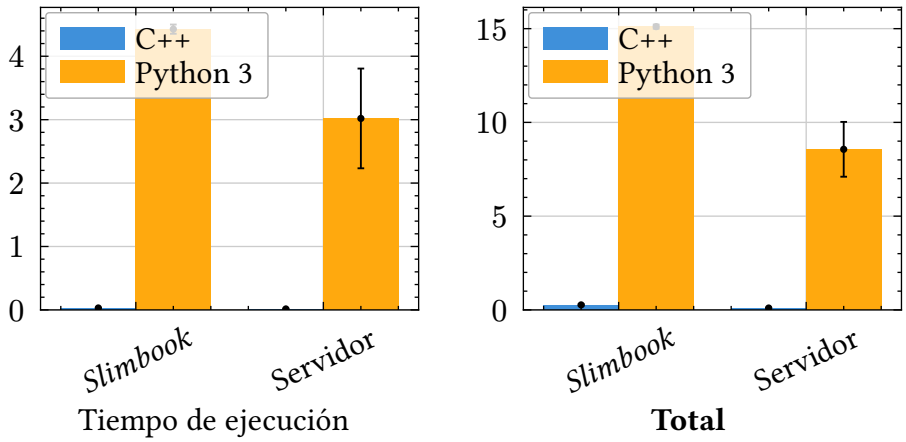


Figura 5.3

Tiempo de ejecución de *Max Distance* y total



5.2.2. Punto flotante en C++

En la **tarea 4.3** de planificación (Sección 8) se hizo las pruebas en el dispositivo empotrado con C++ y punto flotante. Donde *RPi 3* es la *Raspberry Pi 3*, *RPi 4* es la *Raspberry Pi 4* y ESP32C3 es un placa con procesador RISC-V de Espressif a 160MHz. Y se obtuvo los siguientes resultados (véase la Tabla 5.21):

Tabla 5.21

Épocas por segundo con tipo flotante IEEE de 32 bits en C++. Peor caso con 3 patrones

Lenguaje	Compilador	Máquina	épocas/s
C++	GCC 14	ESP32C3	1.64
C++	Clang 18	RPi 4 (32 bits)	470
C++	GCC 12	RPi 3 (64 bits)	217
C++	Clang 18	Slimbook	3036

Se analiza el peor caso, en el que el algoritmo tiene que computar todas las características y compararlo con tres patrones distintos. En la Tabla 5.21 se ve que en todas da tiempo real. Sin embargo, en la ESP32C3, que carece de unidad de cómputo para punto flotante (FPU), está bastante al límite. Como consecuencia se empezó a valorar la alternativa de usar punto fijo a punto flotante.

5.2.3. Punto flotante en Ada

C++ carece de punto fijo de manera nativa, así que se decidió escribir el algoritmo en Ada, que sí lo tiene. Aquí se analiza los resultados de las tareas **5.3**, **5.4** y **6.3** (Sección 8). Después de haberlo implementado en Ada, se obtuvieron los siguientes resultados que se muestran en la Tabla 5.22.

Tabla 5.22

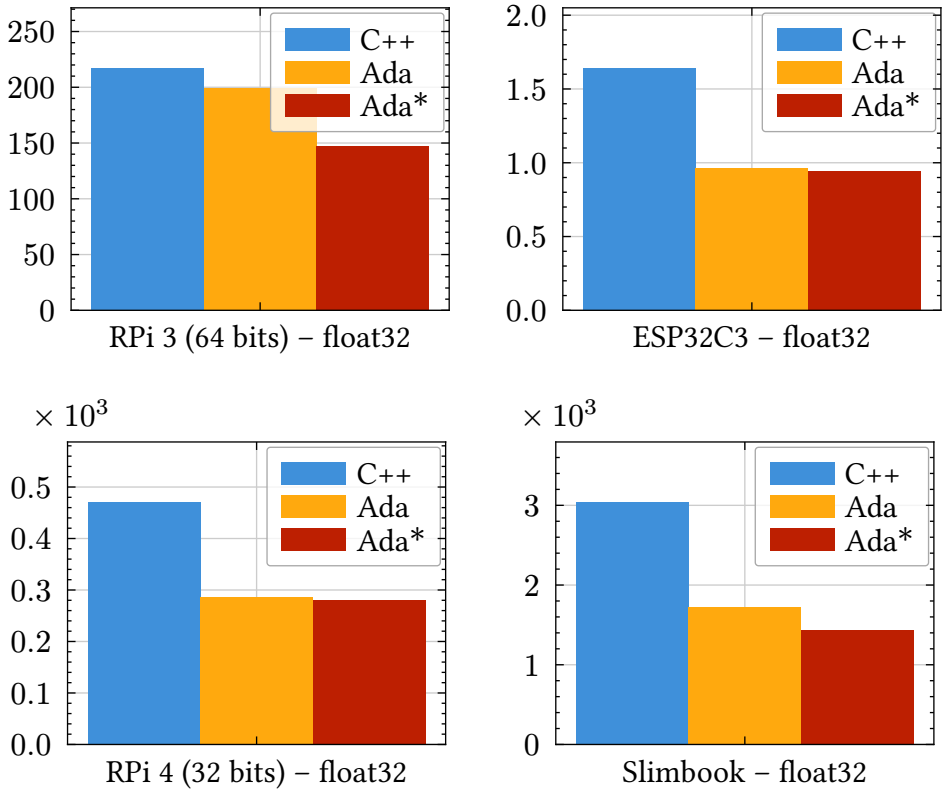
Épocas por segundo con tipo flotante IEEE de 32 bits en Ada. Peor caso con 3 patrones

Lenguaje	*Checks*	Compilador	Máquina	épocas/s
Ada	Desactivados	GNAT 14	ESP32C3	0.96
Ada	Activados	GNAT 14	ESP32C3	0.94
Ada	Desactivados	GNAT 10	RPi 4 (32 bits)	286
Ada	Activados	GNAT 10	RPi 4 (32 bits)	280
Ada	Desactivados	GNAT 12	RPi 3 (64 bits)	199
Ada	Activados	GNAT 12	RPi 3 (64 bits)	147
Ada	Desactivados	GNAT 14	Slimbook	1725
Ada	Activados	GNAT 14	Slimbook	1433

Se ve que no da en tiempo real por poco en la ESP32C3, y que procesa la mitad de épocas por segundo que C++ como se puede ver en la Figura 5.4. *Checks* indica si están activados o no las comprobaciones en tiempo de ejecución.

Figura 5.4

Número de épocas procesadas por segundo en C++, Ada y en Ada con comprobaciones en tiempo de ejecución activadas (Ada*)



5.2.4. Punto fijo en Ada (pruebas preliminares)

Este análisis sigue siendo parte de la **tarea 6.3** de la planificación. No se escribió en Ada porque se pensara que fuera a ir más rápido, sino para comprobar su viabilidad con punto fijo. Se probó con dos tipos de punto fijo, uno de 32 bits y otro de 64 bits, que pertenecen a $\mathbb{X}_{32,-8}$ y $\mathbb{X}_{64,-16}$ respectivamente (véase la Sección 4.2). Las pruebas se hicieron para la ESP32C3 y se compiló con GNAT 14 se resumen en la Tabla 5.23.

Tabla 5.23

Prueba preliminares de rendimiento de punto fijo en Ada.

Checks	Tipo	épocas/s
Activados	$\mathbb{X}_{32,-8}$	`Constraint_Error`
Activados	$\mathbb{X}_{64,-16}$	1.08
Desactivados	$\mathbb{X}_{32,-8}$	**21.04**
Desactivados	$\mathbb{X}_{64,-16}$	1.12

Con 64 bits el resultado es correcto, pero rinde peor que la versión de punto flotante de C++. Con pruebas activadas, la implementación con punto fijo de 32 bits termina abruptamente por un desbordamiento; pero con las pruebas desactivadas alcanza un total de 21.04 épocas por segundo.

Se llega a la conclusión de que a pesar de que no se puede traducir directamente a punto fijo, pues habría que estudiar cómo evitar el desbordamiento en todas las operaciones que pueden llegar a desbordar, la idea de convertirlo a punto fijo es **viable**. Incluso si un pequeño porcentaje del código utiliza punto fijo de 64 bits, el límite experimental nos dice que se puede mejorar enormemente.

Después de estudiar por qué la solución de 64 bits era más lenta, se descubrió que el 49.2% del tiempo total de ejecución lo pasaba dividiendo números en punto fijo. Así que dividir o multiplicar números de 64 bits de punto fijo en un procesador de 32 bits es muy costoso.

5.2.5. Punto fijo en SPARK

Finalmente fue demostrando poco a poco la ausencia de errores de programación utilizando SPARK junto a Ada. No dio tiempo a demostrar todas las funciones, pero pruebas unitarias y funcionales no llevan a error. Falta por demostrar formalmente `FFT` (*Fast Fourier Transform*) y `Welch`. Aun así, los resultados se pueden ver en la Tabla 5.24.

Tabla 5.24

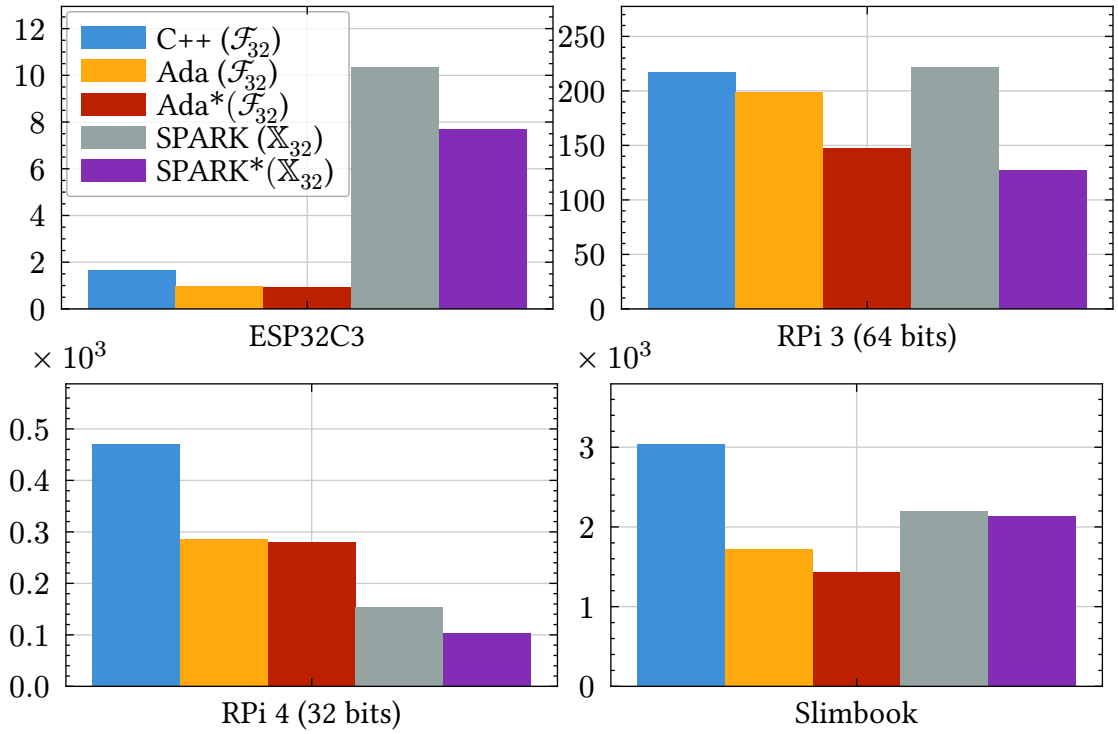
Pruebas finales de rendimiento de punto fijo en SPARK + Ada.

Compilador	Checks	Máquina	épocas/s
GNAT 14	Desactivados	ESP32C3	10.36
GNAT 14	Activados	ESP32C3	7.68
GNAT 10	Desactivados	RPi 4 (32 bits)	154
GNAT 10	Activados	RPi 4 (32 bits)	104
GNAT 12	Desactivados	RPi 3 (64 bits)	222
GNAT 12	Activados	RPi 3 (64 bits)	127
GNAT 14	Activados	Slimbook	2139
GNAT 14	Desactivados	Slimbook	2195

Se observa que para la ESP32C3 supera con creces el requisito de tiempo real de una época por segundo, con 10.36 épocas por segundo. El tiempo adicional permitiría o bien introducir código adicional para autoentrenamiento, o bien dormir el dispositivo empotrado para disminuir el consumo energético.

Figura 5.5

Comparación final de implementaciones. El eje de ordenadas indica el número de épocas por segundo. El asterisco indica que se ha compilado con comprobaciones en tiempo de ejecución activadas.



El uso de punto fijo únicamente tiene sentido en dispositivos que no tienen FPU (unidad de cómputo de punto flotante). A medida que las características del máquina mejoran, en específico las capacidades de punto flotante, disminuye el rendimiento del punto fijo. Además de tener unidades de cómputo específicas para procesar punto flotante, computadores actuales vectorizan fácilmente operaciones con punto flotante y no punto fijo.

Como se ve en la Figura 5.5, en la ESP32C3, que no tiene FPU, el número de épocas computadas por segundo sextuplica a las implementaciones con punto flotante. En la Raspberry Pi 3 los resultados son más parecidos, la diferencia entre la versión de C++ y la de SPARK con punto fijo es despreciable. Sin embargo, en la Raspberry Pi 4, la diferencia entre punto flotante y fijo es abismal. Curioso es el resultado para el Slimbook, pues punto fijo supera a punto flotante en Ada, pero no a C++ esto se puede deber a que las implementaciones no son exactamente iguales y el autor ha ido incluyendo optimizaciones.

El uso de punto fijo es complejo y hay que tener mucho cuidado al trabajar con él. Para aplicaciones modernas en dispositivos relativamente potentes, es mejor utilizar punto flotante. Sin embargo, para este proyecto se pidió un dispositivo de bajo consumo, así que punto fijo supera a punto flotante en este caso específico.

Otra conclusión pertinente que se puede sacar es el efecto de las comprobaciones en tiempo de ejecución, que hace en este caso Ada, y el impacto que tienen en el rendimiento global. Se ve que a medida que las características del computador aumentan el impacto parece disminuir. Posiblemente por la complejidad del *hardware*: predictores de saltos, cachés... Como se puede ver en la Tabla 5.25.

Tabla 5.25

Efecto en el rendimiento de activar las comprobaciones en tiempo de compilación en Ada. $\mu = 0.81, \sigma = 0.14$

Máquina	**Tipo**	**Compilador**	**Efecto**
ESP32C3	float32	GNAT 14	$\times$ 0.98
ESP32C3	fixed32	GNAT 14	$\times$ 0.74
RPi 4 (32 bits)	float32	GNAT 10	$\times$ 0.98
RPi 4 (32 bits)	fixed32	GNAT 10	$\times$ 0.68
RPi 3 (64 bits)	float32	GNAT 12	$\times$ 0.74
RPi 3 (64 bits)	fixed32	GNAT 12	$\times$ 0.57
Slimbook	float32	GNAT 14	$\times$ 0.83
Slimbook	fixed32	GNAT 14	$\times$ 0.97

Finalmente y como curiosidad, he aquí las estadísticas del probador del teoremas a punto de terminar el proyecto.

Tabla 5.26

Estadísticas del análisis de SPARK

SPARK Analysis results	**Total**	**Flow**	**Provers**	**Justified**	**Unproved**
Data Dependencies	41	41	.	.	.
Flow Dependencies	.	.	.	.	.
Initialization	34	34	.	.	.
Non-Aliasing	3	3	.	.	.
Run-time Checks	618	.	608	4	6
Assertions	125	.	123	1	1
Functional Contracts	68	.	57	8	3
LSP Verification	.	.	.	.	.
Termination	44	41	3 (CVC5)	.	.
Concurrency	.	.	.	.	.
Total	933	119 (13%)	791 (85%)	13 (1%)	10 (1%)

CAPÍTULO 6

MARCO REGULADOR

En este capítulo se hace un breve recorrido sobre la legislación aplicable al proyecto. En la Sección 6.1; se enumera y se describe resumidamente las licencias de las dependencias de los componentes del proyecto en la Sección 6.2; y finalmente se comenta qué estándares técnicos más importantes se han utilizado a lo largo del desarrollo del propio proyecto en la Sección 6.3.

6.1. Legislación aplicable

Pese a que se trata de datos de personas reales, estos datos están anonimizados y el *dataset* del que proviene es libre de uso [43], [44], [45], [46]. Las dependencias de enlazado del proyecto son todas de código abierto y son compatibles con la licencia que le ha puesto el autor a este proyecto: la **EUPL-1.2**.

6.2. Licencias de *software*

Dado que el programa depende de bibliotecas externas, es preciso definir primero las licencias y luego cómo depende cada *software* de otros. Se ofrece una breve descripción de cada una de ellas. Sin embargo, es preferible que el lector lea el texto oficial.

- **EUPL-1.2**: *European Union Public License* es la primera licencia libre de código abierto de la Unión Europea que tiene tiene validez en todos los idiomas oficiales de la misma [47]. Esta se puede ver en https://interoperable-europe.ec.europa.eu/collection/eupl/eupl-text-eupl-12 en las veintitrés lenguas oficiales. Es compatible con las licencias: GPLv2, GPLv3, AGPLv3, LGPLv2.1, LGPLv3, CC BY-SA 3.0, MIT, BSD, entre otras [48].

  A continuación un breve resumen de las características de la misma. Otorga derechos a: utilizar, reproducir, modificar, realizar obras derivadas, distribuir, prestar y alquilar, entre otras. El código fuente debe facilitarse de forma gratuita. Sin

embargo el licenciatario está obligado a: atribuir, que obras derivadas o distribución de la misma tenga una licencia de *copyleft* entre otras.

- **MIT** [49]: Es la licencia libre de código abierto del Instituto Tecnológico de Massachussetts. El *software* licenciado con la misma viene in garantía, pero no tiene restricciones en cuanto a: copiar, modificar, fusionar, publicar, distribuir, sublicenciar ni vender copias del mismo; siempre y cuando incluya copias de la propia licencia.
- **BSD-3-Clause** [50]: La licencia BSD (*Berkeley Software Distribution*) modificada de 3 cláusuras es una licencia libre de código abierto, que como su nombre indica, contiene tres cláusuras:
  1. La redistribución del código fuente debe mantener el aviso de *copyright* y el descargo de responsabilidad.
  2. La redistribución en formato binario debe mantener el aviso de *copyright* y el descargo de responabilidad en la documentación; y además otros materiales suministrados con la distribución.
  3. No se puede utilizar el nombre de los titulares de los derechos de autor para promocionar productos derivados sin su consentimiento.
- **Apache License v2.0** [51]: Es una licencia permisiva que requiere que se mantenga el aviso de *copyright* y la licencia. Contribuidores ceden automáticamente los derechos de la patente en sus contibuciones. Trabajos licenciados y modificaciones se pueden distribuir bajo otras condiciones.
- **BSL-1.0** [52]: *Boost Software License – Version 1.0* es una licencia permisiva que únicamente requiere que se mantenga el aviso de *copyright* y la licencia para la distribución del código fuente (no del binario). Se puede distribuir bajo otros términos modificaciones y trabajos licenciados.
- **GPLv3** [53]: *GNU Public License – Version 3* es una licencia de *copyleft* para *software* libre y de código abierto, que requiere a los usuarios de *software* licenciado bajo esta licencia a hacer disponible el código fuente completo bajo la misma licencia: la **GPLv3**. Se debe mantenere el aviso de *copyright* y la licencia. Además los contribuidores ceden los derechos sobre sus contribuciones automáticamente.
- **GPLv3 Runtime Library Exception** [54]: Tiene las mismas condiciones que la **GPLv3**, pero añade una excepción a *software* que enlace con la biblioteca, es decir, el *software* que se ha enlazado no con dicha biblioteca no tiene por qué cambiar su licencia.

El programa de referencia que se encuentra alojado en https://github.com/PPMC-DAC/PaFESD-Epileptic-Seizure-Detection tiene como licencia **MIT**.

Se diferencia entre las dependencias del programa escrito en C++ y el escrito en SPARK y Ada, como si fueran programas distintos. Las dependencias del proyecto son las siguientes:

- **C++**

- **Desarrollo**: los siguientes programas se han utilizado para desarrollar la aplicación, ninguno de ellos impone ningún tipo de restricción sobre la salida que generan. Es decir, el código compilado con GCC o clang en forma de objeto no tiene ninguna restricción, el único problema es a la hora de enlazar.
  - `conan`: Tiene como licencia **MIT** [55]. Se utiliza para gestionar las dependencias.
  - `CMake`: Tiene como licencia **BSD-3-Clause** [56]. Se utiliza junto a `conan` para gestionar el proceso de compilación y enlazar con las dependencias de manera correcta.
  - `GCC`: Tiene como licencia **GPLv3** [57]. Se utiliza para convertir el código fuente en código objeto (compilar) y posteriormente enlazar.
  - `clang`: Tiene como licencia **Apache License v2.0 with LLVM Exceptions**, que además de las condiciones de la licencia **Apache License v2.0**, añade una excepción: si al compilar el programa, porciones del código fuente de `clang` acabaran en dicho código objeto, este queda exenpto de las cláusuras 4(a), 4(b) y 4(d) de **Apache License v2.0** [58].
  - `benchmark/1.8.3`: Tiene la licencia **Apache-2.0** y se utiliza para hacer pruebas de rendimiento en el código [59]. Se utiliza para desarrollo y no va enlazado en el ejecutable final.
- **Bibliotecas**: las bibliotecas son colecciones de subrutinas con las que se enlaza el código fuente para obtener ciertas funcionalidades. En este caso, como las bibliotecas se empaquetan junto al ejecutable final, este tiene que adherirse a las restricciones de uso de sendas licencias.
  - `libstdc++`: Tiene la licencia **GPLv3 Runtime Library Exception**, implementa la biblioteca estándar de C++ y es necesaria a la hora de enlazar el programa. Tiene la excepción como biblioteca en tiempo de ejecución (*runtime library exception*), así que no hay problema al enlazar con ella [60].
  - `newlib` (Espressif): Esta solamente tiene sentido a la hora de enlazar el código fuente en los dispositivos empotrados de Espressif (ESP32C3, ESP32C6, ESP32S3) y tiene la licencia **BSD** [61].
  - `ms-gsl/4.0.0`: Tiene la licencia **MIT** [62]. Se utiliza para dar soporte a las *C++ Core Guidelines*.
  - `rapidcsv/8.80`: Tiene la licencia **BSD-3-Clause** [63]. Se utiliza para abrir archivos C.S.V.
  - `range-v3/0.12.0`: Tiene la licencia **BSL-1.0** [64]. Se utiliza para hacer operaciones con rangos más complejas que lo que permite la biblioteca estándar.
  - `onetbb/2021.12.0`: Está licencia bajo **Apache-2.0** [65]. Se utiliza para implementar ciertos algoritmos de forma paralela.
  - `pybind11/2.10.4`: Tiene la licencia **BSD-3-Clause** [66]. Se utiliza para crear el módulo para que Python3 pueda ejecutar funciones escritas en C++.

    - `boost/1.86.0`: Tiene la licencia **BSL-1.0** [67]. Se utiliza para lecturas y escrituras asíncronas de puertos seriales.
    - `PaFESD-Epileptic-Seizure-Detection`: Es el *software* de referencia y está bajo la licencia **MIT** [68]. De él solo se utiliza su implementación de la función de la deformación dinámica del tiempo.

- **Ada**
  - **Desarrollo**: Las siguientes herramientas se utilizaron únicamente para desarrollar la aplicación, no se enlaza con ninguna de ellas, así que no hay problemas de licencias.
    - `GNAT`: Es parte de la colección de GCC [57] y está bajo la misma licencia, la **GPLv3** [69], [70], [71]. Se utiliza para convertir código fuente de Ada a código objeto.
    - `gprbuild`: Bajo la licencia **GPLv3 Runtime Library Exception** [72]. Se utiliza de manera parecida a `cmake` para organizar la compilación y enlazado de múltiples archivos de código fuente de Ada, C y C++.
    - `alire`: Está bajo la **GPLv3** [73]. Es un gestor de paquetes similar a `conan` para gestionar dependencias y *toolchains* en Ada.
    - `spark2014`: Está bajo la **GPLv3** [74]. Contiene el programa `gnatprove` que permite correr distintos probadores de teoremas en el código de Ada para verificar propiedades del mismo. Además de una biblioteca con lemas y teoremas que no genera código, para ayudar al desarrollador.
  - **Bibliotecas**:
    - `libgnat`: Es la biblioteca que implementa la biblioteca estándar de Ada para GNAT en GCC, tiene la licencia **GPLv3 Runtime Library Exception** [69], [70], [71]; así que no hay ningún problema al enlazar con ella.
    - `newlib` (Espressif): La misma que en las bibliotecas de C++, tiene la licencia **BSD** [61]. `libgnat` necesita ciertos símbolos que exporta `newlib` para entrada/salida y para arrancar la placa.

6.3. Estándares técnicos

6.3.1. C++23 (ISO/IEC 14882:2024)

C++ es un lenguaje de programación estandarizado por ISO (*International Organization of Standarization*) y en el que está escrita la mayor parte del código fuente del proyecto. Una revisión del estándar se hace cada tres años, y recibe un nombre desde C++11 (primera revisión del que se considera C++ moderno), C++14, C++17, C++20 y C++23 (la última versión ratificada). Además ya se está preparando la siguiente revisión que se denominará en un principio C++26.

Es un lenguaje de programación de propósito general originalmente basado en el lenguaje de programación C, pero del que se ha ido alejando a lo largo de los años. Añade tipos de datos adicionales, clases, *templates* (plantillas, para genericidad), excepciones, espacios de nombres (*namespaces*), sobrecarga de operadores y de funciones, referencias y una biblioteca adicional de utilidades. [75]

6.3.2. Ada 2022 (ISO/IEC 8652:2023)

Ada es otro lenguaje de programación estandarizado por ISO, en el que se describe la forma y el significado de los programas escritos en dicho lenguaje, para promover la portabilidad de programas escritos en Ada a variedad de sistema de cómputo. Especifica además, una biblioteca estandarizada que debe ser suministrada junto a la distribución [28]. En el proyecto se utiliza para la parte de verificación formal.

CAPÍTULO 7

ENTRONO SOCIO-ECONÓMICO

7.1. Presupuesto

7.1.1. Recursos humanos

El proyecto ha sido desarrollado por un único desarrollador a lo largo de 34 semanas de acuerdo con la planificación (Sección 8), en las que trabajó de media unas 15 horas semanales. El salario de un investigador *junior* en una universidad de España suele comenzar en los 20000€ anuales [76]. Un año tiene 52 semanas, de las cuales 30 días naturales (o sea, unas 4 semanas) son de vacaciones en España. Se aproxima el salario por hora como:

$$20000\frac{€}{\text{año}} \cdot \frac{1 \text{ año}}{(52-4) \text{ semanas laborales}} \cdot \frac{1 \text{ semana laboral}}{5 \text{ días laboral}} \cdot \frac{1 \text{ día laboral}}{8 \text{ horas laborales}}$$

$$\approx 10.42\frac{€}{\text{hora laboral}}$$

Luego de 34 semanas, trabajando 3 horas cada día, da un total de 714 horas. Lo que da un total de 7439.88 € como se ve en la Tabla 7.1.

Tabla 7.1

Costes humanos

Horas totales	€/Hora	Coste total (€)
714	10.42	**7439.88**

7.1.2. Recursos materiales

En este caso solo se cuentan los recursos físicos como el *hardware*, todas las licencias que se usaron fueron de código abierto y gratuitas. Se utilizó: un portátil para hacer el entrenamiento y programar el código; y distintos tipos de dispositivos empotrados para hacer pruebas de rendimiento.

Tabla 7.2

Costes materiales

Producto	Coste	Vida útil	Tiempo de uso	Coste mensual	Coste amortizado
ESP32C3 (x3)	15.99€	30 meses	9 meses	0.53€	4.80€
ESP32S3	13.59€	30 meses	9 meses	0.45€	4.08€
ESP32C6	15.99€	30 meses	9 meses	0.53€	4.80€
Raspberry Pi 4	99.90€	36 meses	8 meses	2.78€	22.20€
Portátil	1600.00€	48 meses	10 meses	33.33€	333.33€
Total					**369.20€**

7.1.3. Costes indirectos

Los costes indirectos son aquellos que no pueden asociarte directamente a un producto o servicio en particular, pero que son necesarios para la operación general de la empresa [77]. Son costes indirectos: el alquier, la luz, el agua, el Internet etcétera. Véase la Tabla 7.3.

Tabla 7.3

Costes indirectos, para 9 meses

Concepto	Coste	Total
Luz	60€	540€
Internet	50€	450€
Agua	15€	135€
Alquiler	450€	4050€
Total		**5175.00€**

7.1.4. Coste total

El coste del proyecto se considera la suma de todos los costes. El importe final es el coste del proyecto más el beneficio industrial, que será del 16%, y el I.V.A. aplicable del 21%. Como se ve en la Tabla 7.4, el coste del proyecto asciende a los 12984.08€ y el importe total será de 18224.46€.

Tabla 7.4

Coste total

Concepto	Coste
Recursos humanos	7439.88 €
Recursos materiales	369.20 €
Costes indirectos	5175.00 €
Total del proyecto	**12984.08 €**
Beneficio industrial (16%)	2077.45€
IVA (21%)	2726.66€
Importe final	**18224.46 €**

7.2. Impacto socio-económico

El objetivo del proyecto no era crear un algoritmo de detección de ataques epilépticos original, sino utilizar uno ya existente, estudiarlo y optimizarlo. De hecho, el algoritmo utilizado es de clasificación, es decir, decide si está o no en un ataque epiléptico.

Sin embargo, desmostrar que es posible ejecutar un programa tan computacionalmente costoso como este en tiempo real en un dispositivo empotrado de bajas características permite indicar que esta clase de detectores son viables y además, que no tienen por qué consumir muchos recursos. Este estudio ha sido desarrollado en colaboración la Universidad de Málaga, donde se ideó el algoritmo original.

El otro objetivo del trabajo fue hacer un análisis comparativo entre distintas técnicas de programación, lenguajes de programación, arquitecturas... Y observar cómo mejora o empeora el rendimiento en distintas plataformas.

Utilizar un probador de teoremas junto al código permite, entre otras cosas, demostrar matemáticamente que el programa no puede fallar: todos los caminos que dan paso a un error están tratados y por ende el sistema no puede terminar de manera abrupta. Es un filosofía de programación que, pese a ser extremadamente costosa, para estos casos tan delicados como es la detección de ataques epilépticos da paz mental a los pacientes; pues saben que en ningún momento va a terminar de forma abrupta la aplicación.

7.3. Objetivos de desarrollo sostenible

Los objetivos de desarrollo sostenible son un conjunto de objetivos globales (17) para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible. Cada objetivo tiene metas específicas que deben alcanzarse en el año 2030 [78].

El primero de los objetivos con el contribuye es el tercero: «salud y bienestar». El proyecto se trata de una aplicación médica para detección de ataques epilépticos. Descubrir qué patrones de señales neuronales preceden una convulsión permite a los pacientes tomar medicación antes de que ocurra.

Esto lleva al décimo objetivo de desarrollo sostenible: «reducción de las desigualdades». La epilepsia es una enferemedad que en 2006 afectaba a seis de cada mil europeos. En muchos países, por ejemplo, está prohibido que conduzcan personas con epilepsia, exceptuando a aquellos que puedan controlar los posibles ataques [79]. Así pues, poder detectar el ataque antes de que ocurra permite que estas personas puedan reaccionar y, por consiguiente, conducir como el resto de ciudadanos.

Por último, en relación con el decimotercer objetivo («acción por el clima») y el séptimo objetivo («energía asequible y no contaminante»), una de las prioridades del proyecto es que el dispositivo final corra en tiempo real y con un consumo mínimo. El estudio de los algoritmos, sus optimizaciones y las técnicas de programación tienen como finalidad la de conseguir un sistema que sea eficiente además de eficaz.

CAPÍTULO 8

PLANIFICACIÓN

En este capítulo se describe la planificación del proyecto, el cual se ha reallizado en cascada y se divide en siete tareas distintas, y cada una de estas en otras subtareas. Se puede ver en la Figura 8.1 un diagrama de Gantt del proyecto con la fecha de inicio y finalización de cada una de las tareas.

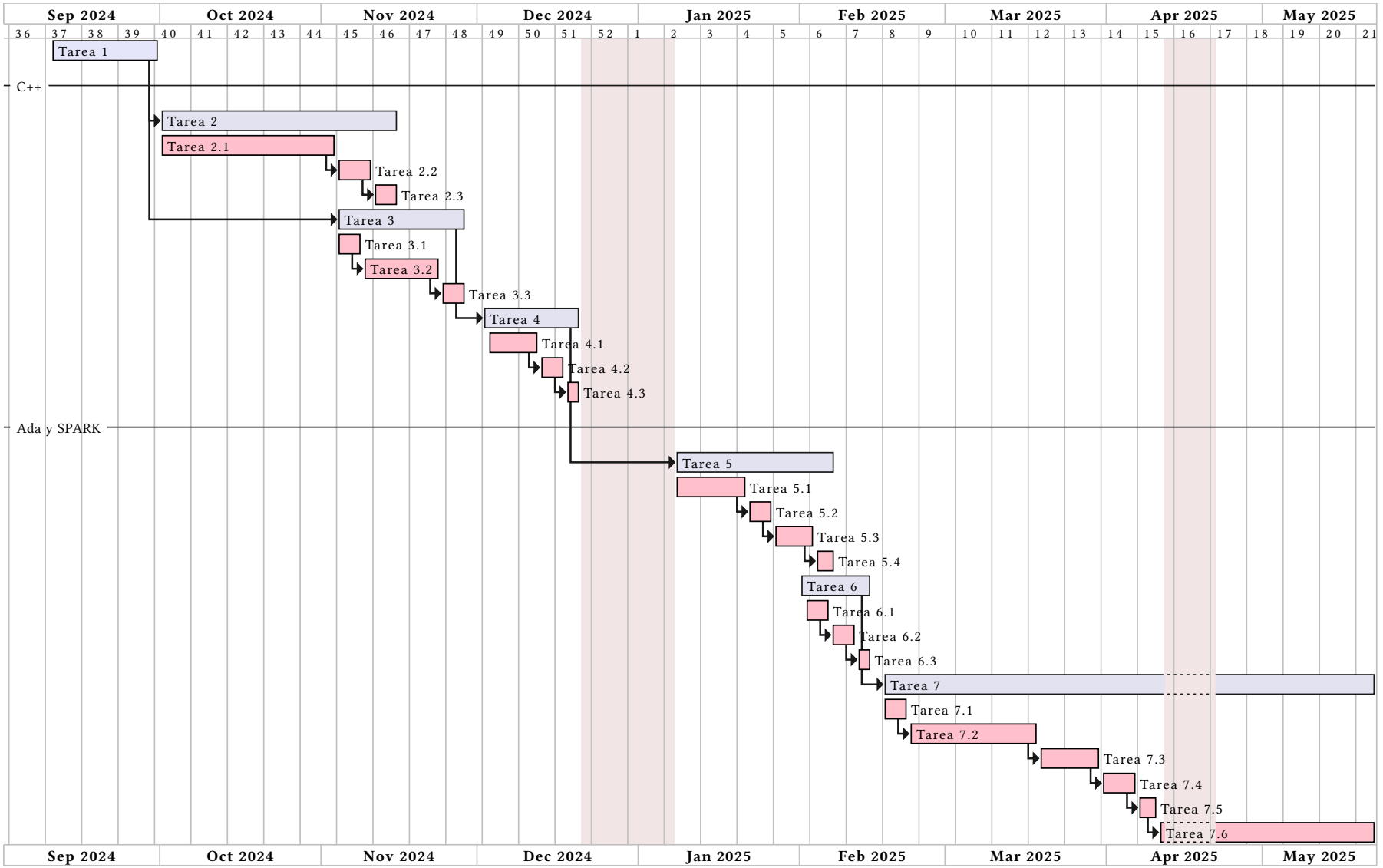
La planificación no cuenta el tiempo necesario para elaborar la memoria, única y exclusivamente de los pasos para realizar el trabajo desde principio a fin. En total fueron 34 semanas.

- **Tarea 1**: *Configuración y familiarización con el código original.*
  Esta tarea consiste en preparar el computador para el desarrollo del programa. Como este utiliza como base un trabajo preexistente, es necesario comprender: cómo funciona el código original, cómo obtiene los datos de los pacientes, cómo entrena modelos, dónde se puede mejorar. Finalmente, como el programa está escrito en C++, es necesario descargar el compilador y las herramientas de desarrollo.
  - **Tarea 1.1**: Instalación de Manjaro
  - **Tarea 1.2**: Instalación de herramientas de desarrollo.
  - **Tarea 1.3**: Familiarización con del repositorio original.
  - **Tarea 1.4**: Configuración del proyecto de C++ con `conan`.
- **Tarea 2**: *Implementación en C++.*
  Una vez que está el entorno preparado, se procede a hacer ingeniería inversa para identificar cómo funcionan las subrutinas del código original, a partir de las cuales se escribe el código optimizado en C++. Cuando este está terminado, se procede con un *binding* (interfaz entre dos lenguajes de programación) de C++ con Python 3, para que se pueda integrar en el programa original.
  - **Tarea 2.1**: Implementación de características de la señal.
  - **Tarea 2.2**: Implementación del módulo de Python 3.

  - **Tarea 2.3**: Pruebas y comparación de resultados entre ambas implementaciones.
- **Tarea 3**: *Análisis de rendimiento y paralelización del algoritmo.* Con el programa funcionando, se puede investigar cuáles son los cuellos de botella y qué partes del mismo se pueden mejorar. También se paraleliza para mejorar el tiempo de entrenamiento.
  - **Tarea 3.1**: Búsqueda de cuellos de botella
  - **Tarea 3.2**: Optimización y paralelización
  - **Tarea 3.3**: Pruebas de rendimiento
- **Tarea 4**: *Compilación cruzada de C++.* El programa de detección debe correr en algún dispositivo empotrado y se debe determinar si corre o no en tiempo real.
  - **Tarea 4.1**: Portar código y dependencias al sistema de Espressif para compilar para la ESP32C3, ESP32C6 y ESP32C6.
  - **Tarea 4.2**: Compilación cruzada a AArch64: Raspberry Pi3 y Raspberry Pi 4.
  - **Tarea 4.3**: Pruebas de rendimiento el empotrado
- **Tarea 5**: *Implementación en Ada.* Se reimplementan algunas funciones del algoritmo en Ada para probar si es viable utilizar punto fijo, como Ada tiene soporte para punto fijo de manera nativa. Se implementa inicialmente en punto flotante y luego se sustituye el tipo base de flotante a fijo para hacer pruebas.
  - **Tarea 5.1**: Implementación de funciones básicas.
  - **Tarea 5.2**: Paralelización del algoritmo de detección.
  - **Tarea 5.3**: Pruebas de rendimiento en punto flotante y comparación.
  - **Tarea 5.4**: Pruebas de rendimiento en punto fijo y comparación.
- **Tarea 6**: *Compilación cruzada de Ada.* Al igual que C++ es necesario hacer compilación cruzada en Ada para probar si funciona en tiempo real o no.
  - **Tarea 6.1**: Integración de `gprbuild` con el sistema de Espressif. y compilación cruzada del código de Ada como biblioteca.
  - **Tarea 6.2**: Compilación cruzada para Aarch64: Raspberry Pi 3 y Raspberry Pi 4.
  - **Tarea 6.3**: Pruebas de rendimiento.
- **Tarea 7**: *Implementación en SPARK.* Al determinar que es viable usar punto fijo, se procede a estudiar los casos de desbordamiento y subdesbordamiento del mismo para asegurar la ejecución correcta del mismo.
  - **Tarea 7.1**: Demostración de `maxdistance`.
  - **Tarea 7.2**: Demostración de `energy` e implementación y demostración de sus funciones auxiliares.
  - **Tarea 7.3**: Demostración de la función `dtw` e implementación y demostración de sus funciones auxiliares.
  - **Tarea 7.4**: Implementación y demostración de las funciones: seno, coseno y raíz cuadrada.
  - **Tarea 7.5**: Implementación y demostración de `simpson`.
  - **Tarea 7.6**: Implementación del resto de funciones.

Figura 8.1

Diagrama de Gantt del proyecto



CAPÍTULO 9

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se enuncian: en primer lugar, las conclusiones del proyecto (Sección 9.1); en segundo lugar, las conclusiones personales (Sección 9.2); y finalmente qué resta para el futuro (Sección 9.3).

9.1. Conclusiones del proyecto

Se han completado todos los objetivos, aunque el tercer objetivo solo parcialmente. Los objetivos se describieron en la Sección 1.2.

- **O1**: Se pudo implementar dicho módulo de Python 3 en C++ y como se observa en la Sección 5.2.1 los resultados son geniales. En un portátil se ha mejorado el tiempo 56.7 veces y en un servidor, que es donde va a acabar corriendo el entrenamiento, es 85.9 veces más rápido que la implementación anterior.

  Reducir el tiempo de entrenamiento permite que se pueda iterar más rápido y experimentar sin esperar tanto tiempo. Además está de acuerdo con los objetivos de desarrollo sostenible, menos tiempo de cómputo, menos energía consumida.

  Según el equipo de la Universidad de Málaga con el que se ha trabajado, los resultados son generiales y mejores de lo esperado. Así que, este primer objetivo se da por finalizado.

- **O2**: Se pudo hacer compilación cruzada de C++ a dispositivos empotrados de bajo consumo como la ESP32C3, ESP32C6 y ESP32S3 de Espressif, además de otras máquinas como la Raspberry Pi 3 y la Raspberry Pi 4. El programa ejecutaba en tiempo real como se puede ver en los resultados de la evaluación en la Sección 5.2.2, con un total de 1.64 épocas por segundo en el peor de los casos contrastando con tres patrones distintos en la ESP32C3.

Por lo que el objetivo se dio por completado. Sin embargo, el autor de este documento decidió experimentar cuánto se podría mejorar. Entre las razones, si computa más épocas por segundo, el sistema tiene más tiempo para descansar y así puede llegar a consumir menos energía. Se implementó en Ada con punto fijo y como se ve en la Sección 5.2.5, los resultados son muchos mejores con un total de 10.36 épocas por segundo en el peor de los casos contrastando con tres patrones distintos en la ESP32C3. De ahí también se origina el objetivo **O3**.

- **O3**: Puesto que punto fijo es muy sensible a errores de desbordamiento se hizo todo el estudio de la Sección 4.1 en el que, junto a un probador de teoremas, se pudo demostrar en la mayor parte del código la ausencia de errores en tiempo de ejecución. Sin embargo, dado que demostrar cada uno de los teoremas demora mucho tiempo, no se pudieron demostrar todos y cada uno de ellos. Aun así, puesto que todas las pruebas, a pesar de no haber demostrado completamente el programa entero, se da por completado el objetivo y se deja como trabajo futuro el demostrar el resto del algoritmo.

Por lo tanto, el trabajo se da por terminado.

Se concluye que C++ es *de facto* más rápido que Python 3 o Ada, y que es preferible para desarrollar sistemas en los que el rendimiento es indispensable. Además, hoy en día es preferible utilizar punto flontante en vez de punto fijo, excepto cuando se esté trabajando para dispositivos empotrados que no tengan FPU o para aplicaciones bancarias.

Ada junto a SPARK permiten demostrar gran variedad de propiedades del código y permite verificarlo formalmente con probadores de teoremas. Sin embargo, dado que trabajar con SPARK es tedioso y costoso no está indicado para cualquier proyecto, únicamente para aquellos en los que el fallo sea la diferencia entre la vida y la muerte. Es preferible utilizarlo para punto específicos del programa más sensibles a fallos. Ada sigue siendo un gran lenguaje de programación, aunque no sea tan rápido como C++, ya que permite describir propiedades del programa mediante contratos.

9.2. Conclusiones personales

Este ha sido un proyecto que me ha permitido utilizar gran variedad de tecnologías y combinarlas entre sí para descubrir qué es lo que funciona y qué no. He podido utilizar lenguajes de programación con los que estaba familiarizado como C++, Python 3 y Ada.

He aprendido mucho sobre compilación cruzada y he descubierto sobre todos los rinconcitos que tiene cada arquitectura y sobre cómo exprimirlos al máximo. He aplicado gran variedad de técnicas desde paralelismo, pasando por vectorización hasta metaprogramación y uso de la excelente biblioteca de rangos de C++.

Previamente he utilizado SPARK, pero esta es la primera vez que lo utilizo para un proyecto serio. He aplicado todo lo aprendido en lógica y matemáticas para demostrar propiedades de mi código fuente. Estoy muy orgulloso con los resultados y me gustaría de continuar con ello.

9.3. Trabajo futuro

Como se dijo previamente, todavía queda por demostrar utilizando SPARK los algoritmos de la transformada rápida de Fourier (`FFT`) y de `Welch`. Aunque sean más complicados de demostrar, existen artículos al respecto al menos para la transformada de Fourier [39] (que es el algoritmo que se ha implementado).

Después de discutirlo con los investigadores de la Universidad de Málaga, como trabajo futuro queda: por su parte, investigar cómo predecir en vez de clasificar ataques epilépticos; por nuestra parte, quedaría permitir que el modelo se reentrene en tiempo real. Para ello era necesario conseguir hasta el milisegundo y por eso este trabajo se ha centrado tanto en la parte de optimización del algoritmo.

BIBLIOGRAFÍA

[1] R. Asenjo, «PPMC – Parallel Programming Models and Compilers». Accedido: 28 de agosto de 2025. [En línea]. Disponible en: https://www.ac.uma.es/~asenjo/research/index

[2] F. Muñoz, R. Asenjo Plaza, y A. Navarro, «PaFESD: Patterns Augmented by Features Epileptic Seizure Detection», *IEEE Transactions on Biomedical Engineering*, vol. 72, n.º 1, pp. 137-151, 2025, doi: 10.1109/TBME.2024.3441090.

[3] R. A. y Felipe M. López, «PaFES: Patterns augmented by Features Epileptic Seizure Detection». GitHub, 2024.

[4] W. H. Organization, «Epilepsy». Accedido: 28 de agosto de 2025. [En línea]. Disponible en: https://www.who.int/news-room/fact-sheets/detail/epilepsy

[5] M. K. Siddiqui, R. Morales-Menendez, X. Huang, y N. Hussain, «A review of epileptic seizure detection using machine learning classifiers», *Brain informatics*, vol. 7, n.º 1, p. 5, 2020.

[6] I. N. de Recherche en Informatique et en Automatique, «Cambios en la versión 9.0 de Rocq». Accedido: 22 de julio de 2025. [En línea]. Disponible en: https://rocq-prover.org/doc/V9.0.0/refman/changes.html#version-9-0

[7] I. N. de Recherche en Informatique et en Automatique, «Rocq». Accedido: 22 de julio de 2025. [En línea]. Disponible en: https://rocq-prover.org/

[8] G. Gonthier y others, «Formal proof–the four-color theorem», *Notices of the AMS*, vol. 55, n.º 11, pp. 1382-1393, 2008.

[9] T. Stérin, «BB(5) = 47,176,870». Accedido: 22 de julio de 2025. [En línea]. Disponible en: https://discuss.bbchallenge.org/t/july-2nd-2024-we-have-proved-bb-5-47-176-870/237

[10] D. R. Licata y M. Shulman, «Calculating the Fundamental Group of the Circle in Homotopy Type Theory», en *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2013, pp. 223-232. doi: 10.1109/LICS.2013.28.

[11] T. L. F. R. Organization, «Lean». Accedido: 12 de agosto de 2025. [En línea]. Disponible en: https://lean-lang.org/

[12] T. L. F. R. Organization, «Lean Focused Research Organization». Accedido: 28 de agosto de 2025. [En línea]. Disponible en: https://lean-lang.org/fro

[13] J. W. McCormick y P. C. Chapin, *Building High Integrity Applications with SPARK*. Reino Unido: Cambridge University Press, 2015.

[14] A. S. Comitee, «6.1.1 Preconditions and Postconditions». [En línea]. Disponible en: http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-6-1-1.html

[15] AdaCore, «SPARK Overview». Accedido: 24 de julio de 2025. [En línea]. Disponible en: https://learn.adacore.com/courses/intro-to-spark/chapters/01_Overview.html

[16] AdaCore, «Alternative Provers». Accedido: 24 de julio de 2025. [En línea]. Disponible en: https://docs.adacore.com/spark2014-docs/html/ug/en/appendix/alternative_provers.html

[17] B. Meyer, «Applying'design by contract'», *Computer*, vol. 25, n.º 10, pp. 40-51, 2002.

[18] Eiffel, «Eiffel – Design by Contract and Assertions». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://www.eiffel.org/doc/solutions/Design_by_Contract_and_Assertions

[19] R. Regev, «Known pitfalls in C++26 Contracts – Ran Regev». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://www.youtube.com/watch?v=tzXu5KZGMJk

[20] cppreference, «Ranges library». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/ranges.html

[21] E. Niebler, «Range-v3 – User manual». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://ericniebler.github.io/range-v3/

[22] cppreference, «C++ compiler support». Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/compiler_support.html

[23] J. L. Hennessy y D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6.ª ed. 340 Pine Street, Sixth Floor, San Francisco, CA, United States: Morgan Kaufmann Publishers Inc, 2011.

[24] M. Voss, R. Asenjo, y J. Reinders, «Pro TBB – C++ Parallel Programming with Threading Building Blocks». [En línea]. Disponible en: https://link.springer.com/book/10.1007/978-1-4842-4398-5

[25] cppreference, «std::thread::join». Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/thread/thread/join.html

[26] cppreference, «std::jthread». Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/thread/jthread.html

[27] cppreference, «std::execution::sequenced_policy, std::execution::parallel_policy, std::execution::parallel_unsequenced_policy, std::execution::unsequenced_policy». Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/algorithm/execution_policy_tag_t.html

[28] «ISO/IEC 8652:2023: Information technology – Programming languages – Ada», 2023.

[29] A. Charlet, «Ada 202x support in GNAT». Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://blog.adacore.com/ada-202x-support-in-gnat

[30] P. S. Foundation, «threading – Thread-based parallelism». Accedido: 25 de agosto de 2025. [En línea]. Disponible en: https://docs.python.org/es/3.13/library/threading.html

[31] P. S. Foundation, «Python Documentation by Version». Accedido: 25 de agosto de 2025. [En línea]. Disponible en: https://www.python.org/doc/versions/

[32] P. S. Foundation, «multiprocessing – Process-based parallelism». Accedido: 25 de agosto de 2025. [En línea]. Disponible en: https://docs.python.org/es/3.13/library/threading.html

[33] OpenMP, «OpenMP API – General – What is OpenMP». Accedido: 28 de agosto de 2025. [En línea]. Disponible en: https://www.openmp.org/about/openmp-faq/#WhatIs

[34] T. Kadosh, N. Hasabnis, T. Mattson, Y. Pinter, y G. Oren, «Quantifying openmp: Statistical insights into usage and adoption», en *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1-7.

[35] Intel®, «Intel® oneAPI Threading Building Blocks – Scalable Parallel Programming at Your Fingertips». Accedido: 25 de agosto de 2025. [En línea]. Disponible en: http://intel.com/oneTBB

[36] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3.ª ed. One Lake Street, Upper Saddle River, NJ 07458, EE.UU.: Pearson Education, Inc, 2004.

[37] J. Kanig, «Taking on a Challenge in SPARK». Accedido: 26 de julio de 2025. [En línea]. Disponible en: https://blog.adacore.com/taking-on-a-challenge-in-spark

[38] N. Shklov, «Simpson's Rule for Unequally Spaced Ordinates», *The American Mathematical Monthly*, vol. 67, n.º 10, pp. 1022-1023, 1960, Accedido: 27 de agosto de 2025. [En línea]. Disponible en: http://www.jstor.org/stable/2309244

[39] P. Welch, «A fixed-point fast Fourier transform error analysis», *IEEE Transactions on Audio and Electroacoustics*, vol. 17, n.º 2, pp. 151-157, 2003.

[40] J. Smith, *Spectral Audio Signal Processing*. W3K Publishing, 2011. [En línea]. Disponible en: https://ccrma.stanford.edu/~jos/sasp/Welch_s_Method.html

[41] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», *CoRR*, 2015, [En línea]. Disponible en: http://arxiv.org/abs/1502.03167

[42] «Systems and software engineering – Vocabulary», 2017.

[43] G. J., «CHB-MIT Scalp EEG Database (version 1.0.0)», *PhysioNet*, 2010, doi: 10.13026/C2K01R.

[44] A. L. Goldberger *et al.*, «PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals», *circulation*, vol. 101, n.º 23, pp. e215-e220, 2000.

[45] A. H. Shoeb, «Application of machine learning to epileptic seizure onset detection and treatment», 2009, Accedido: 20 de agosto de 2025. [En línea]. Disponible en: http://hdl.handle.net/1721.1/54669

[46] O. D. Commons, «Open Data Commons Attribution License v1.0». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://physionet.org/content/chbmit/view-license/1.0.0/

[47] C. Europea, «EUPL». Accedido: 18 de agosto de 2025. [En línea]. Disponible en: https://interoperable-europe.ec.europa.eu/collection/eupl

[48] C. Europea, «EUPL». Accedido: 18 de agosto de 2025. [En línea]. Disponible en: https://interoperable-europe.ec.europa.eu/collection/eupl/matrix-eupl-compatible-open-source-licences

[49] I. T. de Massachussetts, «MIT». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://opensource.org/license/MIT

[50] R. de la Universidad de California, «BSD-3-Clause». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://opensource.org/license/bsd-3-clause

[51] A. S. Foundation, «Apache 2.0». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://www.apache.org/licenses/LICENSE-2.0

[52] Boost, «Boost Software License 1.0». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://boost.org.cpp.al/users/license.html

[53] GNU, Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://www.gnu.org/licenses/gpl-3.0.html

[54] GNU, Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://www.gnu.org/licenses/gcc-exception-3.1.html

[55] Conan.io, «Conan 2.0 – C/C++ Package Manager». Accedido: 17 de agosto de 2025. [En línea]. Disponible en: https://github.com/conan-io/conan

[56] K. Inc., «CMake». Accedido: 17 de agosto de 2025. [En línea]. Disponible en: https://github.com/Kitware/CMake

[57] GNU, «GCC, the GNU Compiler Collection». Accedido: 17 de agosto de 2025. [En línea]. Disponible en: https://gcc.gnu.org/

[58] L. project, «The LLVM Compiler Infrastructure». Accedido: 18 de agosto de 2025. [En línea]. Disponible en: https://github.com/llvm/llvm-project/

[59] Conan, «benchmark/1.8.3». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/benchmark?version=1.8.3

[60] GNU, «License». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html

[61] Espressif, «Copyrights and Licenses». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/COPYRIGHT.html

[62] Conan, «ms-gsl/4.0.0». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/ms-gsl?version=4.0.0

[63] Conan, «rapidcsv/8.80». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/rapidcsv?version=8.80

[64] Conan, «range-v3/0.12.0». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/range-v3?version=0.12.0

[65] Conan, «onetbb/2021.12.0». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/onetbb?version=2021.12.0

[66] Conan, «pybind11/2.10.4». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/pybind11?version=2.10.4

[67] Conan, «boost/1.86.0». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://conan.io/center/recipes/boost?version=1.86.0

[68] R. Asenjo y F. M. López, «PaFES: Patterns augmented by Features Epileptic Seizure Detection». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://github.com/PPMC-DAC/PaFESD-Epileptic-Seizure-Detection

[69] Alire, «GNAT Native». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://alire.ada.dev/crates/gnat_native

[70] Alire, «GNAT ARM ELF». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://alire.ada.dev/crates/gnat_arm_elf

[71] Alire, «GNAT RISC-V 64 ELF». Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://alire.ada.dev/crates/gnat_riscv64_elf

[72] AdaCore, Accedido: 20 de agosto de 2025. [En línea]. Disponible en: https://github.com/AdaCore/gprbuild

[73] Alire, «Alire». Accedido: 16 de agosto de 2025. [En línea]. Disponible en: https://alire.ada.dev/

[74] AdaCore, «SPARK 2014». Accedido: 17 de agosto de 2025. [En línea]. Disponible en: https://github.com/AdaCore/spark2014

[75] «ISO/IEC 14882:2024: Programming languages – C++», 2023.

[76] P. Carreño, «Esto es lo que gana un investigador científico en España en 2025», *elEconomista*, may 2025, Accedido: 10 de agosto de 2025. [En línea]. Disponible en: https://www.eleconomista.es/empleo/noticias/13370787/05/25/esto-es-lo-que-gana-un-investigador-cientifico-en-espana-en-2025.html

[77] E. de Postgrado de Economía y Finanzas, «Costes directos e indirectos: qué son, sus diferencias y ejemplos». Accedido: 24 de agosto de 2025. [En línea]. Disponible en: https://postgradoeconomia.com/costes-directos-e-indirectos-diferencias-ejemplos/

[78] U. C. I. de Madrid, «¿Qué son los ODS?». Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://www.uc3m.es/sostenibilidad/uc3m-con-ods/que-son

[79] C. Girona, «Conducir con epilepsia», *El País*, abr. 2006, Accedido: 8 de agosto de 2025. [En línea]. Disponible en: https://elpais.com/diario/2006/04/11/salud/1144706402_850215.html

GLOSARIO

AArch64 — Nombre que recibe la extensión de 64 bits del juego de instrucciones ARM

algoritmo — Serie de instrucciones que se aplican a unos datos de entrada para solucionar un problema.

asincronía — En informática, se refiere a eventos que ocurren independientemente al flujo principal del programa como señales o interrupciones.

biblioteca — En informática, colección de subrutinas.

bit — Unidad mínima de información.

byte — Grupo de bits, normalmente son octetos de 8 bits.

caché — En informática, estructura que almacena datos para acelerar consultas futuras a esos mismos datos.

codificación — En informática, representación de la información en un soporte.

copyleft — Técnica legal que permite dar libertades de uso y distribución de una obra bajo derechos de autor.

compilador — En informática, programa que traduce código fuente en un lenguaje de programación a código máquina, que entiende un computador.

corolario — En matemáticas, proposición que no necesitar ser demostrada que se deduce de un teorema demostrado previamente.

desbordar — En informática, se dice de estructura de datos o un valor numérico que rebasa el límite fijado.

discreto — En matemáticas, usualmente se refiere a procesos o a cantidades que se pueden representar con números enteros: …, −3, −2, −1, 0, 1, 2, …,

front-end — En informática, capa de presentación.

función — En matemáticas es una relación entre dos conjuntos. En informática es una subrutina.

indexar — En informática, acción de acceder a un elemento en una colección dado su índice.

instanciar — En informática, acción de crear una instancia de un objeto.

iteración — En informática, repetición de un segmento de código.

lema — En matemáticas, proposición que es preciso demostrar antes de probar un teorema.

lenguaje de programación — Lenguaje formal que permite describir el comportamiento de un programa de computador.

log — Del inglés, registro.

megabyte — 1 MiB = 1024 bytes

O.D.S. — Objetivos de desarrollo sostenible

procedimiento — En informática, subrutina.

punto fijo — En informática, valor numérico escalado por un factor implícito conocido.

punto flotante — En informática, valor numérico que codifica un factor en el propio dato.

recursivo — En informática, que se define consigo mismo en la propia definición.

rendimiento — Proporción entre producción y medios utilizados.

retornar — En informática, acción de devolver un valor de una subrutina.

RISC-V — Juego de instrucciones.

sistema empotrado — O sistema empotrado, es un sistema de computación que tiene una función dedicada dentro de un sistema más grande.

subdesbordar — Acción de desbordar por el límite inferior.

subrutina — En informática, es una porción de un programa que tiene una interfaz y comportamiento bien definidos y que se puede invocar múltiples veces.

teorema — Proposición demostrable lógicamente partiendo de axiomas ya demostrados.

tiempo real — En informática, se habla de un sistema de información que debe responder dentro de unos límites de tiempo.

toolchain — Del inglés *tool chain*, es una cadena de herramientas. En informática, se suele utilizar para hablar de compiladores.

APÉNDICE A

DECLARACIÓN DE USO DE IA GENERATIVA

DECLARACIÓN DE USO DE IA GENERATIVA EN EL TRABAJO DE FIN DE GRADO

El autor de esta tesis **no** ha usado ningún tipo de inteligencia artificial generativa de ningún tipo: ni durante el desarrollo del proyecto ni durante la redacción de este documento. No considera que sea útil.