Trabajo Fin de Grado

Doble Grado en Ingeniería Informática y Matemáticas

Algoritmos matemáticos en R. Aplicaciones a la docencia y al cálculo científico

por

Carlos Armero Cantó

UNIVERSIDAD
COMPLUTENSE
MADRID

Facultad de Informática

Tutorizado por:

María Victoria López López

Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Junio 2019

Índice general

Resumen

Las asignaturas de cálculo científico son fundamentales en el plan de estudios de las diferentes universidades que imparten grados de ciencias. Dichas asignaturas, tradicionalmente, son impartidas mediante la enseñanza de algoritmos en el lenguaje de programación de MATLAB u otras plataformas de software privativo, que son muy potentes computacionalmente pero no son de libre distribución. R es un lenguaje de libre distribución que es muy popular en el ámbito del análisis estadístico pero que puede actuar perfectamente como lenguaje utilizado en una asignatura de cálculo científico en la universidad. En este trabajo se analizan algoritmos de cálculo científico desarrollados en R, así como funciones previamente definidas en el lenguaje. Dichos algoritmos se comparan con resultados obtenidos en lenguajes de cálculo simbólico. Además, este trabajo prueba la utilidad de R como lenguaje introductorio al mundo de la programación para jóvenes preuniversitarios (Educación Secundaria Obligatoria y Bachillerato). Para ello, se han realizado experiencias prácticas en la Semana de la Ciencia 2018 y en las jornadas "4ESO+Empresa", ambas celebradas en la facultad de informática de la Universidad Complutense de Madrid. Parte de estos resultados se han publicado en el congreso internacional INTED 2019, donde además se analiza el pensamiento crítico de los estudiantes.

Palabras clave

Algoritmos matemáticos, lenguaje de programación R, cálculo científico, pensamiento crítico, software libre.

Abstract

Scientific calculus subjects are really important in the curriculum of all universities which impart science degrees. Those subjects, traditionally, are taught through algorithms that are implemented in MATLAB or other proprietary software platforms, which have powerful computational tools but they are not free software. R is a very popular programming language in statistical analysis, and it could be the main language used in numerical analysis subjects without any problem. We analyse scientific calculus algorithms implemented in R and predefined language methods. These algorithms are compared with other scientific programming languages. In addition, we prove that R could be the main language when youngsters are introduced to programming world by first time (ESO and Bachillerato). For that purpose, we have carried out practical experiences with them in Semana de la Ciencia 2018 and "4ESO+Empresa", held at Universidad Complutense de Madrid. Some of those results have been published in the international congress INTED 2019, where we have also studied critical thinking of students.

Keywords

Mathematical algorithms, R programming language, scientific calculus, critical thinking, free software.

Capítulo 1

Introducción

1. Motivación

El cálculo científico es una rama de las matemáticas y la ingeniería que analiza el empleo de técnicas computacionales para la resolución de problemas [5,14,17,22]. Su relevancia es bastante elevada, ya que permite alcanzar resultados sobradamente cercanos a soluciones de problemas complejos mediante algoritmos de aproximación [11]. Además, se tiene en cuenta la importancia de la precisión de los mismos mediante los conocimientos que se tienen en la materia no solo para obtener resultados fiables, sino también para estimar el error cometido en el cálculo de la solución. Cabe destacar que los algoritmos de cálculo científico son en gran medida algoritmos transparentes, de fácil modificación y adaptables a las situaciones correspondientes según la necesidad del usuario.

Las matemáticas están presentes en todos los ámbitos de nuestras vidas de una manera u otra [1]. Además, las formas de trabajar en el ámbito matemático han vivido una convergencia con el mundo de la algoritmia, formando una simbiosis en la que ya no se conciben las matemáticas sin algoritmia. Las matemáticas ya no están en el papel, sino que están en el computador [23]. Vivimos en un mundo en el que la tecnología está a la orden del día. Desde los más jóvenes ya se crece en un ámbito puramente digital, estando la tecnología al alcance de la gran mayoría de la población.

En el mundo empresarial y académico se lleva a cabo un uso ingente de software de una gran diversidad y con objetivos muy amplios y diferentes. Muchas empresas pueden optar a multitud de licencias de cualquier software privativo sin ningún problema más allá del coste, pero en multitud de situaciones esto podría ser evitable mediante el uso de software de libre distribución [26]. En el ámbito académico es donde más relevancia adquiere el uso de software libre, ya que permite su acceso a todo el mundo independientemente de sus posibilidades.

2. Objetivos

Nuestro objetivo principal consiste en mostrar que R [20], de libre distribución, puede actuar perfectamente como hilo conductor para la exposición y el aprendizaje de una asignatura de cálculo científico completa y sin nada que envidiar a Matlab [9], que es el software utilizado por antonomasia para implementación de algoritmos de cálculo científico. Para ello, nuestra intención es mostrar que cualquier temática dentro del cálculo científico puede ser perfectamente impartida en R, además de analizar las diferencias computacionales de los algoritmos tanto en un lenguaje como en otro. Además, pretendemos probar que un lenguaje como R también puede servir para introducir a jóvenes preuniversitarios al mundo de la programación que tan importante es hoy en día y que creemos que se debería conocer antes de la universidad. También queremos conocer la percepción de los preuniversitarios acerca de sus destrezas y qué grado de pensamiento crítico aplican ante las diferentes soluciones que se pueden adoptar para obtener un resultado.

Para afrontar este trabajo hemos planteado una serie de objetivos específicos, los cuales son descritos a continuación:

- Estudiar los diferentes algoritmos de computación científica de interés en el ámbito académico.
- Estudiar las diferentes herramientas que pueden servir de apoyo para la elaboración de este trabajo y la implementación de los algoritmos, tanto de libre distribución como software privativo.
- Programar una serie de algoritmos de cálculo científico en R y realizar pruebas.
- Comparar los resultados entre distintos lenguajes desde un punto de vista computacional.
- Probar que R es un lenguaje perfectamente viable para el desarrollo de algoritmos de cálculo científico.
- Analizar el pensamiento crítico de los estudiantes preuniversitarios.
- Realizar un taller experimental con preuniversitarios para analizar sus destrezas y su percepción de la dificultad.
- Recoger los resultados obtenidos a lo largo de todo el trabajo para la elaboración de esta memoria y la publicación de un artículo en el congreso internacional INTED 2019.

3. Metodología

Para llevar a cabo la realización de este trabajo se ha seguido la metodología que se describe a continuación.

En primer lugar, llevé a cabo un estudio intenso de distintos materiales sobre cálculo científico. Entre ellos, son destacables diferentes libros mencionados en la bibliografía, así como algún manual utilizado en asignaturas universitarias de cálculo científico [12]. Cabe destacar que a lo largo del doble grado en matemáticas e ingeniería informática que he estudiado he trabajado en diferentes asignaturas ciertas nociones sobre cálculo científico. Dichas asignaturas son Métodos Numéricos y Análisis Numérico. Además, también otras asignaturas sirvieron como base para el aprendizaje de la programación, como Fundamentos de la Programación, Tecnología de la Programación o Métodos Algorítmicos para la Resolución de Problemas [7], entre otras. En cuanto al lenguaje de programación R, la asignatura de Modelos Estadísticos aportó las nociones básicas sobre la programación estadística con dicho lenguaje en la parte práctica de la asignatura.

Posteriormente, di los siguientes pasos con un nivel de detalle más elevado en los lenguajes de programación R y Matlab [19], así como a RStudio, donde pude aprender diferentes técnicas y sobre todo la sintaxis de los lenguajes. Además, mi directora y yo realizamos un estudio sobre posibles trabajos preliminares a la etapa universitaria, llevándolos a la práctica y obteniendo resultados bastante satisfactorios, ya que organizamos una experiencia con 148 estudiantes que nos permitió llevar a cabo un estudio sobre el pensamiento crítico en el ámbito preuniversitario y la percepción que tienen los jóvenes ante la programación mediante una serie de cuestionarios que acompañaron a las sesiones experimentales. A continuación, profundicé en el ámbito de la programación en R y procedí, finalmente, a implementar los distintos algoritmos de cálculo científico que se pueden observar a lo largo del trabajo, lo cual permitió poder realizar un análisis crítico comparativo entre R y Matlab.

Evidentemente no ha sido un proceso totalmente lineal en el que se haya aplicado un método en cascada en cuanto a la planificación y la metodología seguida, ya que se ha iterado en diferentes fases sobre el proceso de ejecución del trabajo, retornando a etapas anteriores cuando ha sido necesario. Dicho esto, un posible diagrama de flujo de una iteración de las etapas de las que consta el trabajo podría ser el que se aprecia en la Figura 1.1.

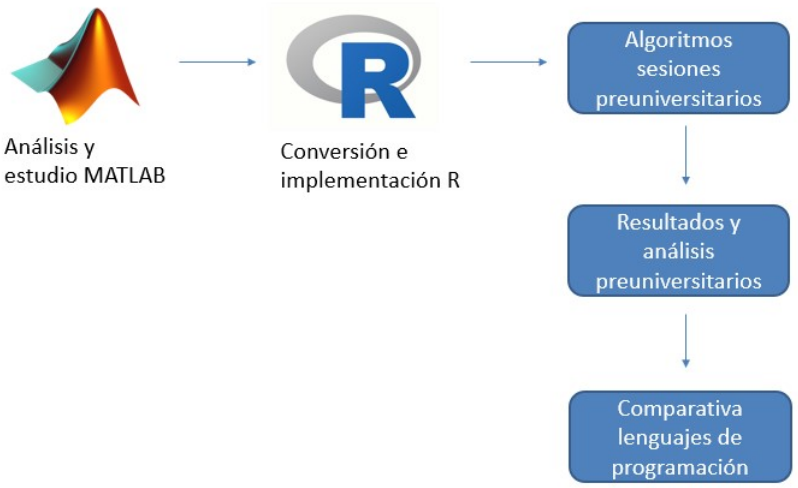


Figura 1.1: Diagrama del proceso de ejecución del trabajo

4. Estructura del trabajo

El trabajo se ha estructurado en dos bloques diferenciados. El primer bloque corresponde a los temas acerca de los algoritmos sobre cálculo científico. El segundo bloque habla sobre las primeras aproximaciones a la programación y la algoritmia en el ámbito preuniversitario.

En primer lugar, encontramos este primer tema introductorio donde exponemos la motivación, la metodología utilizada y la estructura general del trabajo. En el tema 2 desarrollamos el estado del arte. Entre los temas 3 y 7 se implementan y analizan los diferentes algoritmos de cálculo científico, entre los que se encuentran algoritmos para el cálculo de raíces de una función, algoritmos para la factorización de matrices, algoritmos para la resolución de sistemas de ecuaciones lineales, algoritmos de interpolación de funciones y algoritmos para la derivación e integración de funciones. Además, al final de cada uno de estos capítulos, se establece una comparativa entre los lenguajes de programación R y Matlab, donde podemos analizar tiempos de ejecución, utilidades y recursos de ambos lenguajes. Para finalizar, el tema 8 habla sobre la experiencia llevada a la práctica y los algoritmos utilizados con jóvenes preuniversitarios acerca de una primera toma de contacto con el mundo de la algoritmia y su relación con las matemáticas, de la cual se extraen una serie de resultados en un estudio acerca del nivel de percepción de la dificultad y el pensamiento crítico de los estudiantes. El tema 9 contiene las conclusiones obtenidas tras desarrollar el trabajo y las propuestas de trabajo futuro que podrían partir de este trabajo como base.

Capítulo 2

Estado del arte

El cálculo científico es una rama de las matemáticas y de la ingeniería muy importante tanto en el ámbito académico como en el profesional. El desarrollo de algoritmos de cálculo científico nos permite obtener aproximaciones de soluciones a diferentes problemas, aportando además la capacidad de estimar el error cometido en el cálculo de la solución. Las asignaturas de cálculo científico son frecuentemente impartidas en los grados universitarios de ciencias. El objetivo de estas asignaturas consiste en alcanzar los conocimientos suficientes acerca de los distintos mecanismos que se abordan en cualquier libro sobre la materia o, por ejemplo, en este trabajo. Tradicionalmente se ha asociado a la implementación de algoritmos de cálculo científico en este tipo de asignaturas a Matlab, pero lenguajes de libre distribución como R también pueden llevar a cabo esta tarea.

1. Lenguajes de programación científica y cálculo simbólico

Existen multitud de lenguajes de programación destinados al cálculo científico y al cálculo simbólico. Entre ellos, encontramos el lenguaje propio de Matlab, Maple o Mathematica, entre otros. Hoy en día es muy común el uso de este tipo de lenguajes en el ámbito académico, ya que presentan una potencia computacional elevada y una aplicación potencialmente útil desde el punto de vista industrial. Hoy en día, prácticamente todas las empresas grandes y medianas poseen departamentos o equipos completos destinados al desarrollo de diferentes tipos de software. El auge del Big Data, la ciencia de datos o el internet de las cosas han producido un enorme crecimiento en el interés de la industria por contratar servicios precisamente destinados a desarrollo y mantenimiento de software. Aunque posteriormente analizaremos que otros lenguajes como R pueden llevar a cabo prácticamente todos los propósitos que tengamos en el ámbito del cálculo científico, sigue siendo muy común la apuesta por parte de las empresas por lenguajes del estilo de Matlab. Como comentábamos anteriormente, este es uno de los motivos principales

por los que en el ámbito académico se sigue apostando fielmente por lenguajes de programación precisamente diseñados para la computación científica y el cálculo simbólico.

1.1. Matlab

Matlab (Figura 2.1) consiste en un sistema diseñado para la computación numérica que posee un lenguaje propio (cuando hagamos referencia a Matlab a lo largo del trabajo nos referiremos a su lenguaje de programación). La gestión de la plataforma, el desarrollo y sus actualizaciones son responsabilidad de la empresa MathWorks, que es la propietaria de Matlab. Cleve Moler fue el desarrollador inicial de Matlab, fundando además la empresa MathWorks en los años 80.

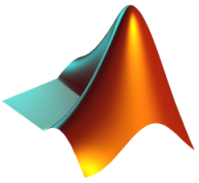


Figura 2.1: Logo de Matlab

Matlab sigue siendo de los lenguajes más utilizados para la implementación de algoritmos de cálculo científico, por no decir el más utilizado, disponiendo de multitud de librerías que ayudan a ello. Destaca su potencia computacional y su capacidad para trabajar de manera vectorizada y matricial. Se trata del lenguaje comúnmente más enseñado en las universidades para impartir asignaturas de cálculo científico y análisis numérico. Además, muchas empresas utilizan Matlab como herramienta principal para el desarrollo de software en ámbitos diversos como el análisis de datos o el desarrollo de modelos matemáticos, entre muchos otros. Este hecho permite que Matlab siga siendo siempre uno de los lenguajes más solicitados en el mercado de contratación de nuevos trabajadores.

1.2. Otros lenguajes de programación científica y cálculo simbólico

Maple (Figura 2.2) es un lenguaje interpretado de programación cuyas funcionalidades están plenamente orientadas hacia cálculo algebraico y álgebra computacional. Maple es utilizado en asignaturas de la universidad impartidas en grados de matemáticas, ya que unifica el álgebra con la computación de una manera directa. Sin ir más lejos, en la Universidad Complutense de Madrid se utiliza el lenguaje Maple para desarrollar algoritmos algebraicos en la asignatura de álgebra computacional. Dicha asignatura es impartida en el último curso del doble grado en matemáticas e ingeniería informática, además de en otros grados de la facultad de matemáticas.

Mathematica es un lenguaje de programación que puede ser utilizado para diferentes propósitos, aunque su función principal es la de desarrollar algoritmos relacionados con el álgebra

computacional. Mathematica posee funcionalidades bastante amplias en este aspecto, siendo una alternativa potente como lenguaje de cálculo simbólico, de la misma manera que lo era Maple.

Figura 2.2: Logo de Maple

SageMath es otra alternativa para el desarrollo de algoritmos de álgebra computacional, además de servir como motor para implementar funciones de cálculo matricial de una manera amigable, ya que presenta una interfaz cómoda para el usuario. SageMath también es una herramienta utilizada en la Universidad Complutense de Madrid. Más concretamente, es la herramienta que sirve para llevar a cabo los cálculos necesarios en algunas asignaturas de la facultad de matemáticas como geometría lineal, por ejemplo. SageMath hace uso de paquetes ya creados concretamente para otros lenguajes de programación, optimizando de esta manera el cálculo y el rendimiento de las operaciones.

Octave es un lenguaje de programación de software libre destinado a la computación científica. Octave es una alternativa fiable a Matlab, ya que está diseñado para el desarrollo de algoritmos de análisis numérico y presentando el beneficio de no ser software privativo.

2. El lenguaje R y RStudio

R es un lenguaje de programación de libre distribución normalmente asociado a tareas estadísticas, ya que dispone de una gran variedad de librerías sobre tratamiento estadístico, manejo de datos y visualización de los mismos. Su origen data de 1993 en la universidad de Auckland, Nueva Zelanda. R consiste en un proyecto colaborativo, beneficiándose de las aportaciones de la comunidad y promoviendo la colaboración de los propios usuarios al desarrollo y la evolución del lenguaje y la comunidad, como por ejemplo, mediante la publicación de librerías por parte de los propios usuarios. R es un lenguaje de programación basado en memoria, lo que provoca una mayor importancia de la eficiencia del algoritmo implementado para evitar la disminución del rendimiento. Además, R facilita la interacción del código con otros lenguajes de programación y con diferentes tipos de bases de datos. R es muy utilizado en universidades y empresas, ya que aporta al usuario una serie de características muy potentes en cuanto al tratamiento de volúmenes importantes de datos, visualización de los mismos mediante una amplia gama de gráficos disponibles y una gran cantidad de herramientas estadísticas. Las empresas hoy en día poseen grandes equipos de trabajadores cuyo trabajo está estrechamente relacionado con la minería de datos, por lo que R es uno de los lenguajes preferidos por multitud de compañías para llevar a

cabo su desempeño. Uno de nuestros objetivos es ampliar estas miras y establecer los límites de la capacidad de R en un horizonte más lejano.

Los críticos de R argumentan que este lenguaje no es capaz de procesar cantidades ingentes de datos, a diferencia de otros lenguajes. La realidad es que, en el caso de que este hecho fuera cierto, no sería problema ya que se pueden implementar métodos para que las cantidades de datos se dividan en particiones, evitando de esta manera el problema. Además, otros lenguajes como Matlab consumen muchos recursos del dispositivo en el que se está ejecutando, mientras que R es capaz de llevar a cabo muchas tareas de diversa naturaleza mediante el consumo de pocos recursos.

CRAN (Comprehensive R Archive Network) es una red de servidores FTP y servidores web que pone a disposición del usuario librerías y módulos adicionales de R y contiene las nuevas versiones de código y documentación.

Figura 2.3: Logos de R y RStudio

RStudio es la herramienta que hemos utilizado para codificar los algoritmos en R desarrollados en este trabajo, ya que nos facilita bastante la implementación de los mismos, permitiéndonos hacerlo de una manera más visual y clara. RStudio es la interfaz comercial de R, presentando una versión gratuita entre otras opciones de pago más avanzadas y con mayor número de utilidades disponibles. RStudio facilita la codificación, la depuración del código, la documentación y, en definitiva, la accesibilidad al entorno. En la Figura 2.3 observamos los logos de R y RStudio.

3. Otras herramientas relacionadas

Por otro lado, existen multitud de lenguajes de programación relacionados. Python es un lenguaje que posee gran presencia en el mercado empresarial sobre todo, aportando una gran versatilidad con funcionalidades muy diversas, desde el tratamiento de datos con librerías tan famosas como Pandas o Numpy hasta tareas de web scraping en departamentos de ciberseguridad. El uso de Pandas en la industria está muy extendido, ya que el tratamiento de datos es una de las funcionalidades más valoradas a la hora de contratar nuevos empleados en tareas de desarrollo de software. Python, gracias a paquetes como Pandas y Numpy (Figura 2.4), permite llevar a cabo la limpieza de datos, la unión, la mezcla y el tratamiento analítico de los mismos, permitiendo la conexión con otros lenguajes de programación, así como a diferentes tipos de bases de datos.

Figura 2.4: Logos de Pandas y Numpy

Para concluir, vamos a hablar del lenguaje de programación SAS (Figura 2.5). Se trata de un lenguaje orientado al tratamiento de datos y el desarrollo de algoritmos relacionados con el análisis de datos. SAS facilita el manejo de los datos de manera similar a lenguajes de manejo de bases de datos, como SQL. Además, SAS ofrece una gran cantidad de información estadística sobre las tablas de datos que maneja. SAS es una de las herramientas más demandadas por las empresas para nuevas contrataciones en el ámbito del análisis de datos y el desarrollo de modelos.

Figura 2.5: Logo de SAS

Capítulo 3

Algoritmos sobre cálculo de raíces de una función

En este capítulo vamos a estudiar una serie de algoritmos que tienen como objetivo el cálculo de las raíces de una función dada.

El cálculo de raíces de funciones tiene algunas aplicaciones curiosas. Una de ellas consiste en que la frecuencia de vibración de la cuerda de un instrumento musical es la raíz de un polinomio generado por las características de la cuerda. La optimización es otra de sus aplicaciones más conocidas, permitiendo la correcta administración y reparto de los recursos en cualquier ámbito, ya sea empresarial, industrial o doméstico [1]

A lo largo del tema propondremos diferentes métodos para el cálculo de raíces, razonando la obtención de los algoritmos y estableciendo los pasos, las condiciones y las fórmulas imprescindibles para llevar a cabo los mismos. Además, finalizaremos el tema estableciendo una comparativa entre los resultados de R y MATLAB.

1. Método de la bisección

El método de la bisección se trata con total seguridad del más sencillo de comprender para el cálculo de raíces de una función dada. Se basa en el teorema de Bolzano [3].

La idea general del método es la de ir acotando el intervalo solución a partir de un intervalo inicial dado hasta dar con la raíz. Dicho intervalo debe tener como extremos dos puntos a y b cuyas imágenes tengan signo contrario. Este método no es muy sofisticado y puede llegar a resultar algo lento, además de depender de la validez del intervalo inicial de entrada.

La acotación del intervalo en cada iteración se realiza evaluando el punto medio c del in-

tervalo. Si $f(c) = 0$ o $f(c)$ es menor que una cierta tolerancia dada, habremos obtenido la raíz buscada. Si $f(c)$ y $f(b)$ tienen signo contrario, utilizaremos c como nuevo extremo izquierdo del intervalo. Si, por el contrario, $f(c)$ y $f(a)$ tienen signo contrario, el punto medio c será el nuevo extremo derecho del intervalo de búsqueda. En la Figura 3.1 podemos observar el diagrama de flujo de este algoritmo.

El punto c en cada iteración lo obtenemos calculando el punto medio.

$$c = \frac{a+b}{2}$$

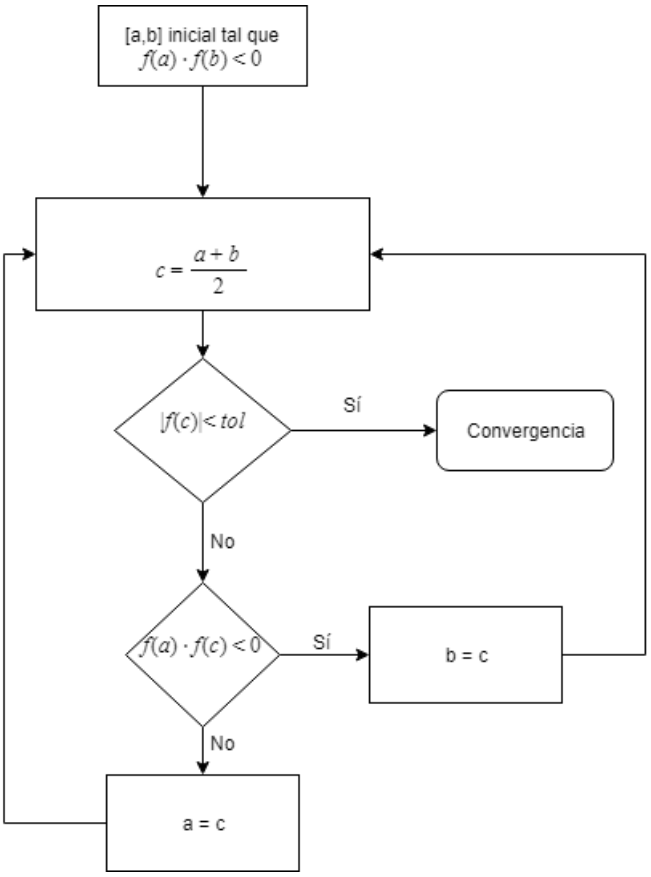


Figura 3.1: Diagrama de flujo de la bisección

En las Figuras 3.2 y 3.3, respectivamente, podemos observar implementaciones del método de la bisección de manera iterativa y recursiva.

```
biseccion <- function(a, b, tol, iters){
  c <- (a+b)/2
  it <- 1

  while (abs(f(c)) >= tol & (it < iters)) {
    if (f(c) < 0){
      a <- c
    }
    else{
      b <- c
    }
    c <- (a+b)/2
    it <- it + 1
  }

  if (abs(f(c))< tol){
    return (c)
  }
  else {
    return ("Iteraciones maximas superadas.")
  }
}
```

Figura 3.2: Algoritmo iterativo de la Bisección

```
biseccion <- function(a, b, tol, iters){
  tiempo_inicial <- Sys.time()
  c <- (a+b)/2
  if (abs(f(c))< tol){
    tiempo_final <- Sys.time()
    diferencia <- tiempo_final - tiempo_inicial
    print(diferencia)
    return (c)
  }
  else if (iters < 1){
    tiempo_final <- Sys.time()
    diferencia <- tiempo_final - tiempo_inicial
    print(diferencia)
    return ("Iteraciones maximas superadas.")
  }
  else if (f(c) < 0){
    return (biseccion(c, b, tol, iters-1))
  }
  else{
    return (biseccion(a, c, tol, iters-1))
  }
}
```

Figura 3.3: Algoritmo recursivo de la Bisección

2. Método de la interpolación lineal

El método de la interpolación lineal para el cálculo de raíces converge generalmente más rápido que el método de la bisección. La idea de acotar el intervalo de búsqueda es similar, eligiendo un nuevo punto c como nuevo extremo izquierdo o derecho del intervalo en cada iteración, siguiendo los mismos criterios que en el método de la bisección. La diferencia consiste en la forma de calcular este nuevo punto c.

El punto c será la intersección de la recta formada por los puntos $(a, f(a))$ y $(b, f(b))$ y el eje de abscisas. La ecuación de dicha recta se puede plantear como

$$\frac{x-b}{a-b} = \frac{y-f(b)}{f(a)-f(b)}$$

Tomando la intersección con el eje de abscisas y desarrollando la ecuación, obtenemos

$$0 = \frac{f(a)-f(b)}{a-b} \cdot (x-b) + f(b)$$

Despejando x obtenemos el punto c del intervalo que buscábamos.

$$c = b - f(b) \cdot \frac{b-a}{f(b)-f(a)}$$

En la Figura 3.4 podemos ver un diagrama de flujo del método.

En las Figuras 3.5 y 3.6 podemos observar la implementación iterativa y recursiva del método de la interpolación lineal.

3. Método de Newton-Raphson

El método de Newton-Raphson es un método iterativo en el que se parte de un punto inicial x_0 que tiene que ser lo suficientemente cercano a la raíz para que converja el método. En otras palabras, el método de Newton-Raphson no garantiza su convergencia global.

El método se obtiene partiendo del desarrollo en serie de Taylor de la función $f(x)$ de la que queremos obtener las raíces. El desarrollo en serie de Taylor para un entorno del punto x_n es el siguiente:

$$f(x) = f(x_n) + f'(x_n) \cdot (x - x_n) + \frac{f''(x_n)}{2!} \cdot (x - x_n)^2 + ...$$

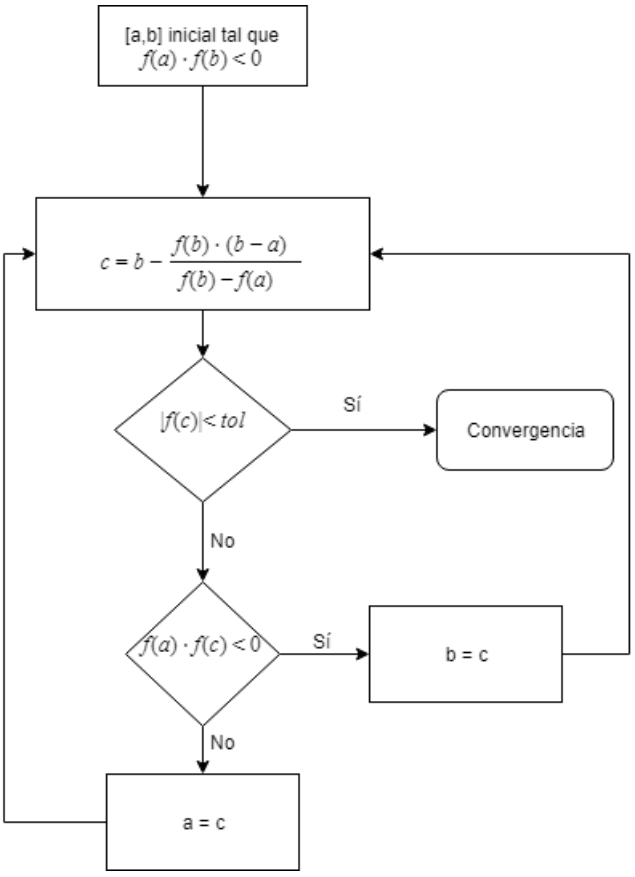


Figura 3.4: Diagrama de flujo de la interpolación lineal

Si truncamos el desarrollo a partir del segundo término, obtenemos:

$$f(x) = f(x_n) + f'(x_n) \cdot (x - x_n)$$

Como podemos apreciar, hemos obtenido la ecuación de la recta tangente a $f(x)$ en el punto x_n. Si ahora intersecamos esta recta tangente con el eje de abscisas y despejamos x, obtendremos el punto a estudiar en esta iteración.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

En la Figura 3.7 podemos apreciar el diagrama de flujo de este algoritmo.

Además, en las Figuras 3.8 y 3.9 presentamos las implementaciones de los algoritmos iterativo y recursivo del método de Newton-Raphson, respectivamente.

```
calcular_c <- function(a, b){

  return (b - (f(b)/(f(b) - f(a)))*(b-a))

}


interpolacion_lineal <- function(a, b, tol, iters){
  c <- calcular_c(a, b)
  it <- 1

  while (abs(f(c)) >= tol & it < iters){
    if (f(c) < 0){
      a <- c
    }
    else{
      b <- c
    }
    c <- calcular_c(a, b)
    it <- it + 1
  }

  if (abs(f(c))< tol){
    return (c)
  }
  else{
    return ("Iteraciones maximas superadas.")
  }
}
```

Figura 3.5: Algoritmo iterativo de la Interpolación lineal

```
calcular_c <- function(a, b){

  return (b - (f(b)/(f(b) - f(a)))*(b-a))

}


interpolacion_lineal <- function(a, b, tol, iters){
  c <- calcular_c(a, b)
  if (abs(f(c))< tol){
    return (c)
  }
  else if (iters < 1){
    return ("Iteraciones maximas superadas.")
  }
  else if (f(c) < 0){
    return (interpolacion_lineal(c, b, tol, iters-1))
  }
  else{
    return (interpolacion_lineal(a, c, tol, iters-1))
  }
}
```

Figura 3.6: Algoritmo recursivo de la Interpolación lineal

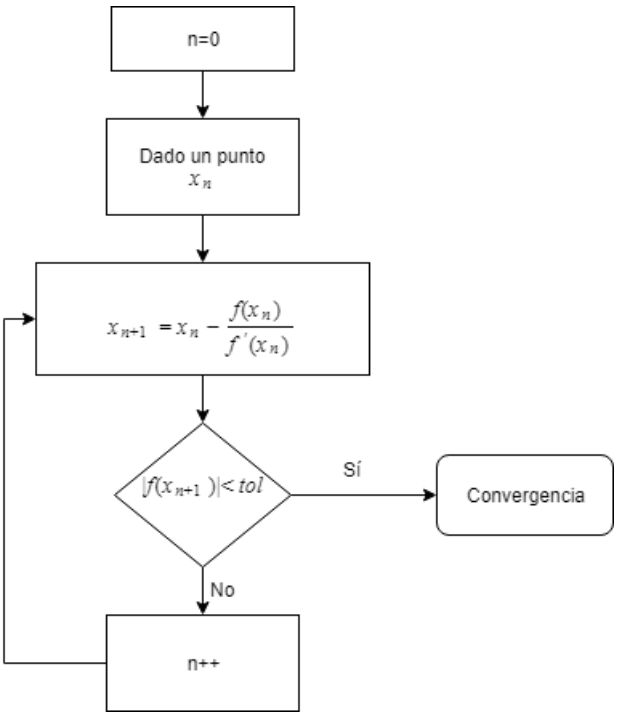


Figura 3.7: Diagrama del método de Newton-Raphson

```
calcular_x <- function(x0){
  return (x0 - f(x0)/deriv_f(x0))
}


newton <- function(x0, tol, iters){
  x <- calcular_x(x0)
  it <- 1

  while (abs(f(x)) >= tol & it < iters){
    x <- calcular_x(x)
    it <- it + 1
  }

  if (abs(f(x))< tol){
    return (x)
  }
  else{
    return ("Iteraciones maximas superadas.")
  }
}
```

Figura 3.8: Algoritmo iterativo del método de Newton-Raphson

```
calcular_x <- function(x0){
  return (x0 - f(x0)/deriv_f(x0))
}


newton <- function(x0, tol, iters){
  x <- calcular_x(x0)
  if (abs(f(x))< tol){
    return (x)
  }
  else if (iters < 1)
  {
    return ("Iteraciones maximas superadas.")
  }
  else{
    return (newton(x, tol, iters-1))
  }
}
```

Figura 3.9: Algoritmo recursivo del método de Newton-Raphson

4. Método de la secante

El método de la secante parte de las mismas premisas que el de Newton, obteniendo la misma ecuación para calcular el siguiente punto x_{n+1} a analizar:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

La diferencia principal es que en el método de la secante se calcula la derivada de la función como la pendiente de la recta secante que une dos puntos: el punto x_n que estamos analizando en esta iteración y el punto que analizamos en la iteración anterior, es decir, x_{n-1}. Esto es:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

El método de la secante consiste en una variación del método de Newton pero, además, posee muchas similitudes con el método de la interpolación lineal. La idea del algoritmo es parecida, ya que ambos hallan el siguiente punto a analizar trazando la recta secante a la función que une dos puntos obtenidos anteriormente y su intersección con el eje de abscisas. Las diferencias principales entre estos dos métodos consisten en que el método de la interpolación lineal necesitaba que la raíz buscada estuviese entre los dos puntos con los que se traza la recta secante, mientras que el método de la secante no lo necesita. Además, en el método de la secante se utilizan los dos puntos obtenidos en las dos iteraciones inmediatamente anteriores, mientras que en el método de la interpolación lineal se utilizan dos puntos anteriores basándose en las reglas del signo contrario, para mantener la idea de que la raíz buscada esté entre los dos puntos utilizados.

Este algoritmo presenta un orden de convergencia lineal mejor que el del método de Newton-

Raphson. Además, se trata de un método que no garantiza la convergencia si los valores de partida no son lo suficientemente buenos.

En la Figura 3.10 podremos analizar un diagrama del método. En las Figuras 3.11 y 3.12 observaremos las implementaciones iterativa y recursiva del método de la secante, respectivamente.

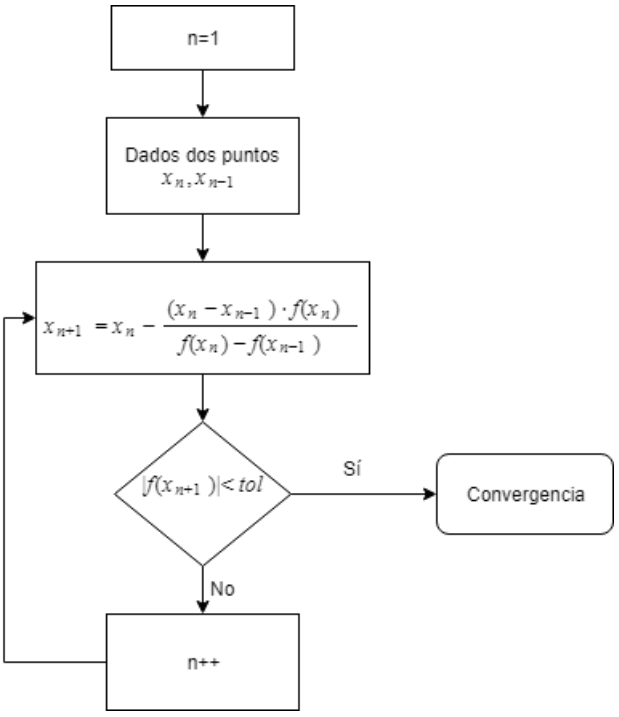


Figura 3.10: Diagrama del método de la secante

5. Método del punto fijo

El método del punto fijo es un metodo iterativo. El objetivo es encontrar una raíz de la función $f(x)$, luego se quiere encontrar el valor x que satisfaga $f(x) = 0$. Dicha función $f(x)$ puede tomarse como $g(x) - x$, siendo $g(x)$ una función auxiliar. Como $f(x) = 0$, sustituyendo, tenemos que $g(x) - x = 0$. Despejando, obtenemos la ecuación $g(x) = x$. El procedimiento parte de un valor inicial x_0, es decir, se calcula primero $g(x_0)$, que nos dará el valor x_1; posteriormente se calcula $g(x_1)$, que nos dará el valor x_2, y así sucesivamente hasta encontrar el punto fijo, es decir, aquel valor x_n que cumpla que $g(x_n) = x_n$. En su defecto, se propone como válida la solución que cumple que la diferencia entre los valores obtenidos en las dos últimas iteraciones del método es menor que cierta tolerancia permitida.

```
calcular_x <- function(x0, x1){
  return (x1 - ((x1-x0)*f(x1)/(f(x1)-f(x0))))
}


secante <- function(x0, x1, tol, iters){
  x <- calcular_x(x0, x1)
  it <- 1

  while (abs(f(x)) >= tol & it < iters) {
    x0 <- x1
    x1 <- x
    x <- calcular_x(x0, x1)
    it <- it + 1
  }
  if (abs(f(x))< tol){
    return (x)
  }
  else{
    return ("Iteraciones maximas superadas.")
  }
}
```

Figura 3.11: Algoritmo iterativo del método de la secante

```
calcular_x <- function(x0, x1){
  return (x1 - ((x1-x0)*f(x1)/(f(x1)-f(x0))))
}


secante <- function(x0, x1, tol, iters){
  x <- calcular_x(x0, x1)
  if (abs(f(x))< tol){
    return (x)
  }
  else if (iters < 1){
    return ("Iteraciones maximas superadas.")
  }
  else{
    return (secante(x1, x, tol, iters-1))
  }
}
```

Figura 3.12: Algoritmo recursivo del método de la secante

El método del punto fijo es un procedimiento bastante sencillo de programar para una función dada $f(x)$, como podemos observar en el código implementado en la Figura 3.13. El problema que presenta es que alcanza la convergencia sólo si se cumple una serie de condiciones. Dichas condiciones están recogidas en el teorema de la existencia y la unicidad del punto fijo [11].

```
punto_fijo <- function(x0, tol, iters){
  if (abs(g(x0) - x0) < tol){
    return (x0)
  }
  else if (iters < 1) {
    return ("Iteraciones maximas superadas.")
  }
  else{
    return (punto_fijo(g(x0), tol, iters-1))
  }
}
```

Figura 3.13: Algoritmo del punto fijo

6. Comparativa entre lenguajes de programación

Como hemos visto, diferentes algoritmos que tienen como objetivo el cálculo de raíces de una función pueden ser implementados en R. Además, hemos analizado algoritmos iterativos y recursivos. Nuestro objetivo en este momento es realizar una comparativa de tiempos de ejecución de los algoritmos entre R y Matlab para poder analizar si es viable el uso de R con estas intenciones, si las diferencias entre lenguajes diferentes son grandes o insignificantes y, en definitiva, si merece la pena el uso de R como sustituto de Matlab en el caso de no poder o querer hacer uso de Matlab por cualquier motivo.

Vamos a realizar un ejemplo de ejecución de todos los métodos para la función $f(x) = e^x + x - 2$. Establecemos un intervalo de partida para los métodos que lo necesitan, tomando el $(-6, 5)$. La tolerancia permitida para el error de la solución final es de 0.1, y permitimos un máximo de 100 iteraciones antes de mostrar un mensaje de fracaso. En la Tabla 3.1 podemos observar los tiempos de ejecución de cada uno de los métodos de manera comparativa entre R y Matlab.

En Matlab disponemos de funciones predefinidas para el cálculo directo de raíces de una función. Una de ellas, la más genérica y útil, es *fzero*, que recibe como parámetro la función a evaluar y el intervalo $[a, b]$ donde evaluar la raíz. Esta función requiere que en el intervalo que introducimos como parámetro se produzca un cambio de signo en la evaluación de la función, ya que no puede llevar a cabo el cálculo de la raíz sin esta característica. Además, la función *fzero* puede recibir como parámetro un valor cercano a la raíz buscada en lugar del intervalo $[a, b]$ que comentábamos anteriormente. Otra opción predefinida en Matlab es la función *roots*, que calcula las raíces de un polinomio. Recibe como parámetro un listado de elementos que actuan como coeficientes del polinomio del que se quieren obtener las raíces. En R existe la función ya definida *uniroot*, que recibe por parámetro la función a evaluar y el intervalo donde evaluarla. Implementa por debajo el método de la bisección.

Los tiempos de ejecución en este tipo de algoritmos son tan bajos que las diferencias son

insignificantes entre un lenguaje y otro. Se ha tomado el tiempo medio de diez ejecuciones de cada método para seguir un criterio establecido. Los resultados se pueden apreciar en la gráfica de la Figura 3.14. En ella vemos que los métodos de Matlab son ciertamente más rápidos, pero no de manera destacable en ninguno de ellos. Los tiempos son tan bajos que R es un lenguaje perfectamente válido para desarrollar algoritmos de este tipo. En definitiva, utilizar R para desarrollar algoritmos sobre el cálculo de raíces de una función no significa una disminución notable del rendimiento ni mucho menos ya que los tiempos son muy bajos tanto en un lenguaje como en otro.

Método	**R**	MATLAB
Bisección iterativo	0.016	0.0084
Bisección recursivo	0.0148	0.0045
Interpolación lineal iterativo	0.012	0.0092
Interpolación lineal recursivo	0.0104	0.0054
Newton-Raphson iterativo	0.0208	0.01012
Newton-Raphson recursivo	0.0156	0.0098
Secante iterativo	0.0177	0.0072
Secante recursivo	0.0169	0.0064
R uniroot	0.0156	-
MATLAB fzero	-	0.0089

Tabla 3.1: Tiempos de ejecución en métodos de cálculo de raíces

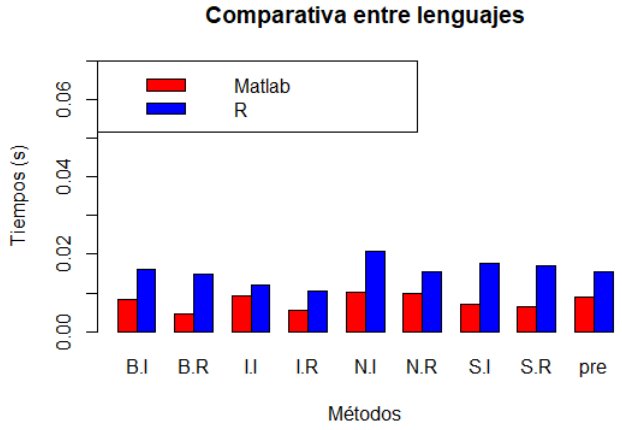


Figura 3.14: Comparativa de tiempos en cálculo de raíces

Capítulo 4

Algoritmos sobre factorización de matrices

La idea de este capítulo reside en el análisis y la implementación de diferentes formas de factorización de una matriz. Dichas factorizaciones consisten en la descomposición de una matriz como producto de diferentes matrices.

La factorización de matrices es muy recurrente a la hora de realizar operaciones con matrices, ya que dichas descomposiciones reducen notablemente la complejidad en determinadas situaciones. Entre otras aplicaciones, la factorización de matrices es muy útil para llevar a cabo el cálculo del determinante de una matriz, además de simplificar en algunos casos la resolución de sistemas de ecuaciones lineales, como veremos en el capítulo posterior. Además, la descomposición de matrices presenta otras aplicaciones interesantes en el campo de la inteligencia artificial y los algoritmos de machine learning, normalizando datos y reduciendo dimensiones de manera útil en métodos como el PCA [4].

Concretamente, vamos a estudiar las siguientes factorizaciones: LU, Cholesky, diagonalización, QR y SVD. Relacionaremos los desarrollos de dichas factorizaciones con algoritmos que las lleven a cabo.

1. Método de la factorización LU

La factorización LU consiste en la descomposición de una matriz A en el producto de una matriz triangular inferior L por una matriz triangular superior U [12].

$$A = LU$$

La matriz U tomará inicialmente el valor de la matriz A, mientras que la matriz L comenzará siendo la matriz identidad. Para el cálculo de las matrices L y U, se procederá a realizar la eliminación gaussiana sobre U, generando así una matriz triangular superior con U y una matriz triangular inferior en L. En la Figura 4.1 podemos observar una implementación de esta factorización.

La factorización LU tiene múltiples aplicaciones en el análisis numérico. Una de ellas es la simplificación del cálculo del determinante de una matriz, ya que $\det(A) = \det(L)\det(U)$, y el determinante de una matriz triangular se obtiene mediante el producto de los elementos de su diagonal. También se utiliza para el cálculo optimizado de matrices inversas. Otra aplicación consiste en el cálculo simplificado de ecuaciones matriciales. Siendo $A = LU$, la ecuación $Ax = b$ se puede escribir como $LUx = b$, y a su vez, $Ly = b$ con $Ux = b$.

La factorización LU presenta una serie de problemas para ciertos tipos de matrices. Por ejemplo, cuando la matriz a factorizar posee ceros en su diagonal se realizarán divisiones entre cero, lo cual supone un problema ya que hará que los cálculos resulten erróneos. Para ello, existe una variante de la descomposición LU denominada factorización $PA = LU$. La idea es similar, sólo que la factorización PA=LU utiliza una matriz P de permutaciones que representa la alteración adecuada del orden de las filas de A. El algoritmo comienza con la matriz P siendo la matriz identidad. La idea es que P intercambia sus filas de manera similar a U cuando lo necesita (como cuando hay ceros en la diagonal de U y queremos intercambiar sus filas para que no los haya).

Para ejecutar en R la factorización $PA = LU$, existe una función muy útil ya definida. Dicha función es lu(A). Utilizando el paquete Matrix y realizando la llamada expand(lu(a)), obtendremos una lista con las matrices L, U y P.

2. Método de la factorización de Cholesky

La factorización de Cholesky consiste en la descomposición de una matriz cuadrada, simétrica y definida positiva en el producto de una matriz triangular inferior por su matriz traspuesta [12].

Sea A una matriz que cumpla las condiciones que hemos mencionado previamente y sea L una matriz triangular inferior. La factorización de Cholesky de A sería $A = L \cdot L^T$.

Una variante de la factorización de Cholesky consiste en la descomposición $A = U^T \cdot U$, siendo U una matriz triangular superior.

```
lu <- function(A, L, U)
{
  n <- nrow(A)
  for (i in 1:n)
  {
    for(j in 1:n)
    {
      aux=0;
      if(i<=j)
      {
        for(k in 1:n)
        {
          if(k!=i)
          {
            aux=aux+L[i,k]*U[k,j]
          }

        }
        U[i,j]=A[i,j]-aux
      }
      else
      {
        for(k in 1:n)
        {
          if(k!=j)
          {
            aux=aux+L[i,k]*U[k,j]
          }
        }
        L[i,j]=(A[i,j]-aux)/U[j,j]
      }
    }
  }

return (list("L" = L, "U" = U))
}
```

Figura 4.1: Factorización LU

$$\begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & ... \\ ... & ... & ... & ... \\ a_{n1} & ... & ... & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & ... & 0 \\ l_{21} & l_{22} & ... & 0 \\ ... & ... & ... & ... \\ l_{n1} & ... & ... & l_{nn} \end{pmatrix} \cdot \begin{pmatrix} l_{11} & l_{21} & ... & l_{n1} \\ 0 & l_{22} & ... & l_{n2} \\ ... & ... & ... & ... \\ 0 & ... & ... & l_{nn} \end{pmatrix}$$

Para hallar los valores de la matriz L se puede ir analizando cómo se obtienen los primeros valores para, posteriormente, tratar de encontrar una generalización que sirva para todos los valores de L.

$$a_{11} = l_{11} \cdot l_{11}$$

$$a_{22} = l_{21} \cdot l_{21} + l_{22} \cdot l_{22}$$

$$a_{33} = l_{31} \cdot l_{31} + l_{32} \cdot l_{32} + l_{33} \cdot l_{33}$$

Luego los elementos de la diagonal de A se pueden generalizar como $a_{ii} = l_{ii}^2 + \sum_{k=1}^{i-1} l_{ik}^2$. Es decir, los elementos de la diagonal de L se obtienen de la siguiente forma:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

Analizando los elementos que no forman parte de la diagonal, observamos que:

$$a_{21} = l_{21} \cdot l_{11}$$

$$a_{31} = l_{31} \cdot l_{11}$$

$$a_{32} = l_{31} \cdot l_{21} + l_{32} \cdot l_{22}$$

La obtención de los elementos que de L que no pertenecen a la diagonal se puede generalizar de la siguiente manera:

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot l_{jk}}{l_{jj}}$$

Lo que podemos observar es que todos los elementos de la matriz triangular inferior L se pueden calcular mediante las fórmulas propuestas y que para poder aplicar dichas fórmulas se necesitan conocer previamente ciertos valores de la propia matriz L. Los elementos de la matriz L se calculan iterativamente por columnas. En la Figura 4.2 podemos encontrar la implementación de la factorización de Cholesky de una matriz A cuadrada, definida positiva y simétrica.

La factorización de Cholesky tiene diferentes aplicaciones, siendo la más conocida su uso en resolución de sistemas lineales $Ax = b$, cumpliendo A las premisas para poder aplicar la descomposición de Cholesky. $Ax = b$ se puede transformar mediante la factorización en $LL^T x = b$, obteniendo finalmente $L^T x = y$ y $Ly = b$.

3. Método de la diagonalización de matrices

Una matriz cuadrada A de dimensión nxn es diagonalizable si y solo si posee n autovectores linealmente independientes [11]. En ese caso, la matriz A se puede expresar como $A = PDP^{-1}$, siendo P una matriz invertible llamada matriz de paso y D una matriz diagonal. P está formada por los autovectores de A, mientras que la matriz diagonal D está formada por los autovalores de A como elementos en la diagonal.

$$A = PDP^{-1}$$

```
cholesky <- function(A, L)
{
  n <- nrow(A)
  for (k in 1:n){
    L[k,k] = A[k, k]
    i <- 1
    while (i <= (k-1)){
      L[k,k] = L[k,k] - (L[k,i])^2
      i <- i + 1
    }
    L[k,k] = sqrt(L[k,k]) #Elemento diagonal

    j <- k + 1
    while (j <= n){
      L[j,k] = A[j,k]
      i <- 1
      while (i <= (k-1)){
        L[j,k] = L[j,k] - L[j,i] * L[k,i]
        i <- i + 1
      }
      L[j,k] = L[j,k]/L[k,k]
      j <- j + 1
    }
  }
  return (L)
}
```

Figura 4.2: Factorización de Cholesky

$$D = P^{-1}AP$$

La diagonalización de matrices posee muchas aplicaciones en diferentes campos. La más conocida es la del cálculo de potencias de una matriz, ya que calcula potencias altas de una matriz reduciendo el coste computacional notablemente. Esto se debe a que, para calcular la potencia n-ésima de una matriz A, tan solo hará falta realizar el producto PD^nP^{-1}. Además, la n-ésima potencia de la matriz diagonal D se calcula realizando la n-ésima potencia de los elementos de la diagonal, lo cual es sencillo y poco costoso computacionalmente.

En R, la diagonalización de matrices se calcula rápidamente mediante una función ya definida denominada eigen(). Dicha función devuelve tanto los autovalores como los autovectores de una matriz cuadrada dada.

4. Método de la factorización QR

La factorización QR descompone una matriz A en el producto de una matriz ortogonal Q y una matriz triangular superior R [12].

$$A = QR$$

La factorización QR tiene aplicaciones sencillas como el cálculo del determinante, de manera similar al resto de factorizaciones. También puede utilizarse para simplificar modelos de regresión.

En la práctica, podemos calcular la factorización QR de diversas maneras. Una de ellas, la que mostramos en el algoritmo de la Figura 4.3, consiste en utilizar la ortogonalización de Gram-Schmidt modificada. Dicho procedimiento es similar al de la ortogonalización de Gram-Schmidt simple, pero con el añadido de ir asegurando la estabilidad de la solución para cualquier matriz inicial A de orden nxn. Dicha estabilidad se genera alterando los valores de la matriz A mediante una matriz auxiliar para que los vectores que forman sus columnas sean ortogonales a las columnas que se van generando de la matriz Q.

En R hay una función ya definida llamada qr(), la cual recibe una matriz y devuelve su descomposición QR.

```
qr <- function(A, Q, R)
{
  V <- A
  n <- nrow(A)
  cols <- ncol(A)
  for (i in 1:n){
    R[i, i] <- 0
    for (j in 1:n){
      R[i, i] = R[i,i] + (V[j,i])^2
    }
    R[i, i] <- sqrt(R[i, i])
    for (j in 1:n){
      Q[j,i] <- V[j,i]/R[i,i]
    }
    z <- i + 1
    while (z <= n){
      R[i, z] <- 0
      for (k in 1:cols){
        R[i,z] <- R[i,z] + Q[k,i] %*%V[k,z]
      }
      for (k in 1:cols){
        V[k,z] <- V[k,z] - R[i,z] %*%Q[k,i]
      }
      z <- z + 1
    }
  }
return (list("Q" = Q, "R" = R))
}
```

Figura 4.3: Factorización QR

5. Método de la factorización SVD

La factorización SVD consiste en la descomposición de una matriz A de orden mxn en el producto de una matriz ortogonal U de dimensión mxm, una matriz diagonal D de dimensiones mxn y la traspuesta de una matriz ortogonal de n filas y n columnas, V^T [12]. Existe un teorema

que asegura la existencia de la factorización SVD para cualquier matriz real A de orden mxn [4].

$$A = UDV^T$$

Los elementos de la diagonal de D son los valores singulares de la matriz y están ordenados de mayor a menor. Las columnas de las matrices U y V forman los vectores singulares de A por la izquierda y por la derecha, respectivamente.

La descomposición SVD tiene múltiples aplicaciones y se trata de una de las descomposiciones más conocidas y utilizadas. Una de sus aplicaciones es la compresión de imágenes. La descomposición SVD se utiliza para aportar mayor robustez numérica al algoritmo PCA (Análisis de Componentes Principales), utilizado mayoritariamente para compresión y normalización de datos. [4]

En R podemos utilizar una función propia llamada svd(A) para calcular la descomposición SVD de cierta matriz A.

6. Comparativa entre lenguajes de programación

Al igual que en el capítulo anterior y como haremos al final del resto de capítulos, vamos a establecer una comparativa entre los algoritmos de factorización de matrices planteados en R y los algoritmos implementados en Matlab. La intención, una vez más, es analizar si utilizar R con el objetivo de implementar este tipo de algoritmos de cálculo científico es viable, útil y se disponen de los recursos necesarios para llevarlo a cabo sin sufrir una disminución grave del rendimiento.

En cuanto a los algoritmos desarrollados en este capítulo, podemos encontrar su implementación en Matlab en funciones ya definidas como la función *lu*, la función *qr*, la función *svd*, la función *chol*. Todas ellas implementan el método que hace honor a su nombre (LU, QR, SVD y Cholesky, respectivamente). La función *eig* calcula los autovectores y los autovalores de la función, luego resulta una función muy útil en el proceso de diagonalización. Todas estas funciones implementadas en Matlab reciben como parámetro la matriz A que se pretende factorizar. En el caso de la diagonalización y la factorización SVD, utilizamos en R las funciones ya definidas *eigen* y *svd*, respectivamente.

A la hora de probar los tiempos de ejecución de cada uno de los algoritmos en los dos lenguajes se ha utilizado la misma matriz tanto para un lenguaje como para otro, variando la matriz solo entre algoritmos para que en cada caso el input sea adecuado a las condiciones iniciales de cada método. En la Tabla 4.1 tomamos los tiempos de ejecución resultantes de calcular la media de los tiempos de diez repeticiones de cada método.

Los tiempos de ejecución de los métodos son de nuevo muy bajos tanto en un lenguaje como en otro, por lo que R vuelve a ser perfectamente válido para la implementación de este tipo de algoritmos. Es cierto que aunque sean muy bajos en ambos lenguajes, Matlab presenta unos tiempos muy positivos, como podemos visualizar en la gráfica de la Figura 4.4. Esto ocurre sobre todo en los métodos en los que en Matlab hemos utilizado una función ya predefinida y en R no. Dichas funciones predefinidas probablemente contengan optimizaciones internas que desconocemos. Dicho esto, los tiempos de R son también muy buenos y en ningún caso se sobrepasa ningún límite en cuanto a rendimiento negativo. La conclusión es que a la hora de implementar algoritmos que tienen como objetivo la factorización de matrices podemos escoger perfectamente R en lugar de Matlab si así lo deseamos o lo necesitamos, ya que a nivel de rendimiento los tiempos no varían en gran medida y además hemos probado que en R se pueden implementar los diferentes algoritmos que hemos expuesto sin ningún problema, pudiendo echar mano a su vez de algunas funciones ya predefinidas en el lenguaje en vez de tener que implementar dichos algoritmos a mano si así lo deseamos.

Método	**R**	MATLAB
LU	0.03459	0.0023
Cholesky	0.03125	0.0021
Diagonalización	0.0156	0.004
QR	0.03641	0.0012
SVD	0.03124	0.0018

Tabla 4.1: Tiempos de ejecución en métodos de factorización de matrices

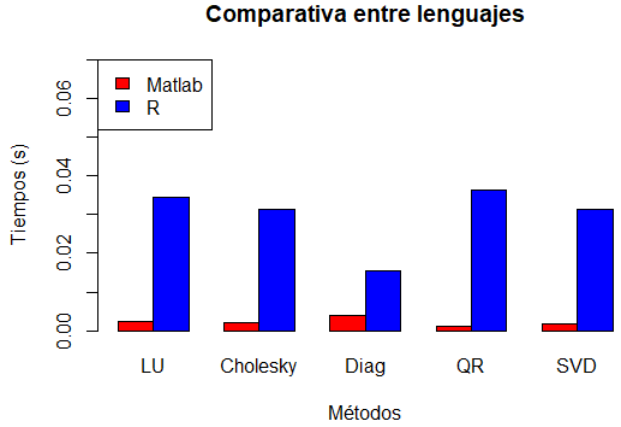


Figura 4.4: Comparativa de tiempos en factorización

Capítulo 5

Algoritmos sobre resolución de sistemas de ecuaciones lineales

En este capítulo se muestran diferentes métodos para la resolución de sistemas de ecuaciones lineales. Dichos métodos pueden dividirse entre métodos directos e iterativos. Dentro de los directos, algunos de los métodos que vamos a tratar tienen una estrecha relación con las factorizaciones de matrices vistas en el capítulo anterior. Además, analizaremos el método de Gauss-Jordan. En cuanto a los métodos iterativos, comentaremos e implementaremos los métodos de Jacobi y de Gauss-Seidel. Vamos a observar que los distintos algoritmos destinados a la resolución de sistemas de ecuaciones lineales pueden ser perfectamente implementados en R.

Las aplicaciones de los sistemas de ecuaciones lineales son evidentes en campos como la física, la ingeniería o las matemáticas. Multitud de leyes y enunciados científicos se fundamentan mediante sistemas de ecuaciones lineales, por lo que su relevancia queda fuera de toda duda.

1. Método de la factorización de matrices

Como vimos en el capítulo anterior, existen múltiples maneras útiles de factorizar una matriz A. También mencionamos que muchas de ellas se podían aplicar en la resolución de sistemas de ecuaciones lineales. Esto es así ya que estas factorizaciones descomponen una matriz como producto de otras matrices con características determinadas, como matrices diagonales, triangulares u ortogonales. A continuación vamos a ver cómo resolver sistemas de ecuaciones lineales cuando la matriz de coeficientes es diagonal o triangular, y posteriormente lo relacionaremos con distintas factorizaciones de matrices comentadas en el capítulo anterior.

Un sistema de ecuaciones lineales tiene la forma $Ax = b$, siendo A la matriz de coeficientes.

$$\begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & ... \\ ... & ... & ... & ... \\ a_{n1} & ... & ... & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{pmatrix}$$

Un sistema de ecuaciones lineales es un sistema triangular cuando la matriz A de coeficientes es una matriz triangular.

$$\begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ 0 & a_{22} & ... & ... \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{pmatrix}$$

Cuando es un sistema triangular superior como el que está descrito arriba, se pueden ir calculando las ecuaciones desde la última hasta la primera progresivamente, de tal manera que siempre vamos a tener una sola incógnita que despejar. En este caso, la primera ecuación sería $a_{nn}x_n = b_n$, de la cual averiguaríamos x_n. La siguiente, $a_{n-1n-1}x_{n-1} + a_{n-1n}x_n = b_{n-1}$, en la que x_n ya es conocida y obtendríamos x_{n-1}. Sucesivamente, calcularíamos x entero hasta x_1. El algoritmo que resuelve un sistema triangular superior de ecuaciones lineales lo podemos observar en la Figura 5.1. La ecuación para calcular cada x_i se puede generalizar como sigue:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} \cdot x_j}{a_{ii}}$$

En el caso de un sistema triangular inferior, el procedimiento es similar al anterior pero calculando las incógnitas progresivamente desde la primera ecuación hasta la última, en vez de desde la última hasta la primera. Comenzaríamos con $a_1x_1 = b_1$ y finalizaríamos calculando x_n. El algoritmo se implementa de manera análoga al algoritmo del sistema triangular superior (Figura 5.2).

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j}{a_{ii}}$$

Una vez analizado cómo resolver con R los sistemas triangulares, vamos a estudiar cómo encajan las factorizaciones de matrices vistas en el capítulo anterior a la hora de resolver sistemas de ecuaciones lineales.

1.1. LU

Sea $Ax = b$ un sistema de ecuaciones lineales con A matriz de coeficientes (ya no tiene por qué ser triangular, evidentemente). Utilizando la factorización $PA = LU$ vista, podemos multiplicar el sistema de ecuaciones lineales por la matriz P de permutaciones a ambos lados de la igualdad. Quedaría $PAx = Pb$. Por la factorización $PA = LU$, el sistema sería $LUx = Pb$. Dicho sistema se podría dividir en dos subsistemas; uno con $Ux = y$, y otro $Ly = Pb$. El primer sistema se podría resolver utilizando la misma técnica vista en el algoritmo de resolución de sistemas triangulares superiores (Figura 5.1), ya que U sería una matriz de coeficientes triangular superior. Posteriormente, $Ly = Pb$ se podría resolver mediante el algoritmo de resolución de sistemas triangulares inferiores de ecuaciones visto en 5.2, ya que L sería una matriz de coeficientes triangular inferior.

1.2. Cholesky

Otro procedimiento para resolver sistemas de ecuaciones lineales sería utilizar la factorización de Cholesky, también implementada en el capítulo anterior. Dicha factorización descomponía una matriz A en LL^T, siendo L una matriz triangular inferior. El sistema $Ax = b$ puede descomponerse como $LL^Tx = b$ utilizando la factorización de Cholesky para la matriz de coeficientes A. De nuevo, el sistema se puede descomponer en dos partes; $L^Tx = y$, y $Ly = b$. Ambos se pueden resolver con los algoritmos comentados para sistemas triangulares, ya que sus matrices de coeficientes son triangular superior e inferior, respectivamente.

1.3. QR

La factorización QR descompone la matriz de coeficientes A como producto de una matriz ortogonal Q y una matriz triangular superior R. El sistema $Ax = b$, tras aplicar a A la factorización QR, queda como $QRx = b$. Por ser Q ortogonal, si multiplicamos cada lado de la igualdad por la izquierda por Q^T, obtenemos $Rx = Q^Tb$, ya que $Q^TQ = I$. Y $Rx = Q^Tb$ ya se puede calcular de manera sencilla como hemos comentado previamente, ya que R es una matriz triangular superior.

1.4. SVD

La factorización SVD descompone una matriz A como $A = UDV^T$, siendo U y V matrices ortogonales y D una matriz diagonal. El sistema $Ax = b$, tras aplicar la descomposición SVD, queda $UDV^Tx = b$. Multiplicando por U^T cada lado de la igualdad por la izquierda, utilizando que U es una matriz ortogonal (es decir, $U^TU = I$), obtenemos $DV^Tx = U^Tb$. Dicho sistema

se puede descomponer en dos partes; la primera, como $V^T x = y$ y la segunda como $Dy = U^T b$. La segunda se calcula de manera trivial, ya que en los sistemas cuya matriz de coeficientes es diagonal se puede obtener el vector solución y despejando directamente $y_1, y_2, ..., y_n$ de cada ecuación. Una vez obtenido y, basta tomar $V^T x = y$ y despejar x para hallar la solución deseada. Si multiplicamos V por la izquierda de cada lado de la igualdad, aprovechando que V es ortogonal, habremos obtenido que $x = Vy$, luego habríamos calculado la solución final.

```
sistema_triangular <- function(A, x, b){
  n <- nrow(A)
 for (i in n:1)
 {
   x[i, 1] <- b[i, 1]
    j <- n
   while (j >= (i + 1))
   {
     x[i, 1] <- x[i, 1] - A[i, j] * x[j, 1]
     j <- j - 1
   }
   x[i, 1] <- x[i, 1]/ A[i, i]
 }
return (x)
}
```

Figura 5.1: Sistema triangular superior

```
sistema_triangular <- function(A, x, b){
  n <- nrow(A)
 for (i in 1:n)
 {
   x[i, 1] <- b[i, 1]
    j <- 1
   while (j <= (i - 1))
   {
     x[i, 1] <- x[i, 1] - A[i, j] * x[j, 1]
     j <- j + 1
   }
   x[i, 1] <- x[i, 1]/ A[i, i]
 }
return (x)
}
```

Figura 5.2: Sistema triangular inferior

2. Métodos de la eliminación de Gauss y Gauss Jordan

2.1. Eliminación de Gauss

El método de eliminación de Gauss consiste en realizar operaciones entre las filas de la matriz ampliada (matriz formada por los coeficientes más los términos independientes de las ecuaciones como última columna) de tal manera que la matriz de coeficientes finalice siendo triangular. Una

vez sea triangular, podremos aplicar uno de los algoritmos vistos en el apartado anterior para cálculo de sistemas triangulares de ecuaciones. La implementación del método de eliminación de Gauss puede observarse en la Figura 5.3.

```
eliminacion_gauss <- function(A){
  n <- nrow(A)
  for (i in 1:(n-1))
  {
    for (j in (i+1):n)
    {
      fact <- A[j,i]/A[i,i]
      for (k in 1:n)
      {
        A[j,k] = A[j,k] - fact*A[i,k]
      }
    }
  }
return (A)
}
```

Figura 5.3: Eliminación de Gauss

El método de eliminación de Gauss puede implementarse con una mejora bastante relevante, como se puede ver en la Figura 5.4. Dicha mejora consiste en reordenar las filas de la matriz de coeficientes según el elemento mayor de cada columna por debajo de la diagonal. Este cambio afectará positivamente a la hora de trabajar con matrices que posean ceros en la diagonal o elementos cercanos al cero.

```
eliminacion_gauss <- function(A){
  n <- nrow(A)
  for (j in 1:(n-1))
  {
    maximo <- abs(A[j,j])
    for (i in j:n)
    {
      if (abs(A[i,j]) > maximo){
        fila_max <- i
        maximo <- abs(A[i,j])
      }
    }

    if (fila_max != j){
      for (k in 1:n){
        o <- A[j, k]
        A[j, k] <- A[fila_max, k]
        A[fila_max, k] <- o
      }
    }
    for (i in (j+1):n){
      fact <- A[i,j]/A[j,j]
      for (k in 1:n){
        A[i,k] <- A[i,k] - A[j, k]*fact
      }
    }
  }
return (A)
}
```

Figura 5.4: Eliminación de Gauss con pivoteo de filas

2.2. Gauss-Jordan

Una variante del método de eliminación de Gauss es el método de Gauss-Jordan. Este método consiste en continuar la eliminación de Gauss cuando la matriz de coeficientes ya es triangular

y eliminar el resto de elementos de la matriz de coeficientes que no pertenecen a la diagonal mediante más operaciones entre filas. Dichas operaciones también afectan a los términos independientes de las ecuaciones, como vimos con el método de eliminación de Gauss. El objetivo es transformar la matriz de coeficientes en una matriz diagonal y que la resolución del sistema de ecuaciones sea trivial, despejando cada x_i de manera directa de su ecuación correspondiente. El problema del método de Gauss-Jordan es que, a pesar de facilitar un poco más la resolución del sistema de ecuaciones, su complejidad es más elevada que la del método de eliminación de Gauss simple. Como el método de eliminación de Gauss genera una matriz triangular de coeficientes, matriz con la que sabemos calcular la solución del sistema mediante los algoritmos vistos para sistemas triangulares del apartado anterior, no merece la pena asumir ese coste computacional añadido para generar una matriz diagonal de coeficientes. De todas formas, una implementación en R del método de Gauss-Jordan se puede visualizar en la Figura 5.5, donde se realiza inicialmente una llamada a la función que lleva a cabo la eliminación de Gauss sobre la matriz inicial ampliada. Dicha función, como hemos visto, está implementada en las Figuras 5.3 y 5.4 sin pivoteo de filas y con pivoteo, respectivamente. .

La implementación de los métodos vistos en este subapartado requieren una fila auxiliar que actúa como ecuación con todos los coeficientes y el término independiente nulos. Esta ecuación auxiliar sirve para el correcto funcionamiento del algoritmo.

```
gauss_jordan<- function(A)
{
  n <- nrow(A)-1
  W<- eliminacion_gauss(A)
  for (j in n:2)
  {
    for (i in (j-1):1)
    {
      fact <- W[i,j]/W[j,j]
      for (k in 1:(n+1))
      {
        W[i,k]=W[i,k]-W[j,k]*fact
      }
    }
  }
  return(W)
}
```

Figura 5.5: Método de Gauss Jordan

3. Métodos iterativos

Un método iterativo calcula progresivamente posibles soluciones utilizando en cada iteración el valor generado en la iteración anterior, hasta que una de las soluciones posibles generadas cumple ciertas condiciones que ratifican la validez de dicho valor como solución del sistema,

siendo realmente una aproximación a la solución real del sistema. Los métodos iterativos normalmente son computacionalmente más eficientes que los métodos no iterativos y son muy útiles cuando el cálculo de un sistema mediante los algoritmos no iterativos es relativamente complejo por volumetría, por ejemplo.

3.1. Jacobi

El método de Jacobi asegura la convergencia cuando la matriz A de coeficientes es de diagonal estrictamente dominante. Cuando A no es una matriz diagonal estrictamente dominante no se asegura la convergencia pero tampoco la descarta. Otra posibilidad para analizar la convergencia del método consiste en calcular el radio espectral de la matriz A. Si el radio espectral es menor que 1, entonces el método converge. Si no lo es, no converge. [11]

El proceso de cálculo iterativo del método de Jacobi es el siguiente:

Sea $Ax = b$. La matriz A de coeficientes se puede descomponer como $D + L + U$, siendo D una matriz diagonal, L una matriz triangular inferior y U una matriz triangular superior. Evidentemente, D está formada por los elementos de la diagonal de A, L por los elementos por debajo de la diagonal de A y U por los elementos de encima de la diagonal de A. El sistema queda $(D + L + U)x = b$. Despejando, obtenemos $Dx = b - (L + U)x$. Finalmente, $x = D^{-1}(b - (L + U)x)$. Esta última ecuación es la utilizada como ecuación de recurrencia del método de Jacobi, generando x_{n+1} a partir del valor x_n tal y como hemos visto.

$$x_{n+1} = D^{-1}(b - (L + U)x_n)$$

El algoritmo del método de Jacobi se implementa de manera muy sencilla como podemos observar en la Figura 5.6. El motivo es que, de la ecuación de recurrencia, todos los elementos (excepto los x_i) son siempre los mismos. Por lo tanto, solo hace falta calcularlos una vez antes de comenzar las iteraciones. R posee funciones específicas para el tratamiento de matrices que nos permiten calcular las matrices D, L y U de manera directa. Además, utilizamos la norma de una matriz y una tolerancia dada para tomar la decisión de validez de la solución calculada.

Con el objetivo de añadir una posible mejora en cuanto a la convergencia del algoritmo, existe un método llamado método de relajación de los algoritmos iterativos que consiste en generar la solución de una iteración como la media ponderada de la solución generada en la iteración anterior y la solución que se hubiera generado en la iteración actual de haber utilizado el método iterativo tradicional. A la solución generada en la iteración anterior se la multiplica por $(1 - \alpha)$, y a la solución que hubiera salido utilizando el método de Jacobi tradicional se la multiplica por α, siendo α el denominado factor de relajación. Nótese que si α tiene valor igual a 1, entonces estamos hablando del método de Jacobi tradicional.

$$x_{n+1} = \alpha \cdot x'_{n+1} + (1 - \alpha) \cdot x_n$$

```
jacobi <- function(A, b, x0, tol, it_max){
  D <- diag(diag(A))
  U[lower.tri(U,diag=TRUE)] <- 0
  L[upper.tri(L,diag=TRUE)] <- 0
  D_inv <- solve(D)
  D_inv_prod_b <- D_inv %*%b
  D_inv_prod_LU <- D_inv %*%(L + U)

  x1 <- D_inv_prod_b - D_inv_prod_LU %*%x0
  it <- 1
  while ((norm(x1 - x0) > tol) & (it < it_max))
  {
    x0 <- x1
    x1 <- D_inv_prod_b - D_inv_prod_LU %*%x1
    it <- it + 1
  }
  if (it >= it_max)
  {
    return ("Iteraciones maximas superadas.")
  }
  else{
    return (x1)
  }
}
```

Figura 5.6: Método de Jacobi

3.2. Gauss-Seidel

El método de Gauss-Seidel, igual que el de Jacobi, es un método iterativo de resolución de sistemas de ecuaciones lineales. Dicho método supone una ligera mejora respecto al método de Jacobi, reduciendo el número de iteraciones necesario para hallar una solución al sistema que respete la tolerancia de error permitida. Basa la generación de soluciones en cada nueva iteración no solo en el valor generado en la iteración inmediatamente anterior, sino que también hace uso de los valores de las componentes ya generadas en la iteración presente, optimizando así el proceso de generación de la nueva solución. El método de Gauss Seidel converge si y sólo si el radio espectral de la matriz de coeficientes del sistema es menor que 1. Además, una condición suficiente (aunque no necesaria) es que el método converge si la matriz de coeficientes es de diagonal estrictamente dominante. [11]

El algoritmo de Gauss-Seidel se puede implementar también de manera relativamente sencilla gracias a las funciones predefinidas en R para tratamiento de matrices. La implementación del método la podemos observar en la Figura 5.7. El procedimiento se basa en realizar la misma descomposición de la matriz A que ya se realizaba en el método de Jacobi, es decir, descomponer A como suma de una matriz con su parte diagonal, otra con su parte triangular superior y otra

con su parte triangular inferior (D, U y L, respectivamente). La diferencia es que en el método de Gauss Seidel, a diferencia del método de Jacobi, la ecuación se despeja de la siguiente manera:

$$x_{n+1} = (D + L)^{-1}(b - Ux_n)$$

Al igual que con el método de Jacobi, el método de Gauss Seidel también puede implementarse con una pequeña mejora en cuanto convergencia asociada al factor α de relajación ya comentado previamente. Esta mejora consiste nuevamente en calcular la nueva solución de una iteración como media ponderada entre la solución que resultaría del método de Gauss Seidel estándar y la solución que se generó en la iteración anterior. La ecuación de recurrencia quedaría de la siguiente manera, siendo $x^{'}_{n+1}$ la solución que se generaría en la presente iteración mediante el método de Gauss Seidel estándar:

$$x_{n+1} = \alpha \cdot x^{'}_{n+1} + (1 - \alpha) \cdot x_n$$

```
gauss_seidel <- function(A, b, x0, tol, it_max){
  D <- diag(diag(A))
  U[lower.tri(U,diag=TRUE)] <- 0
  L[upper.tri(L,diag=TRUE)] <- 0
  D_L <- D + L
  D_L_inv <- solve(D_L)
  D_L_inv_prod_b <- D_L_inv %*%b
  D_L_inv_prod_U <- D_L_inv %*%U


  x1 <- D_L_inv_prod_b - D_L_inv_prod_U %*%x0
  it <- 1
  while ((norm(x1 - x0) > tol) & (it < it_max))
  {
    x0 <- x1
    x1 <- D_L_inv_prod_b - D_L_inv_prod_U %*%x1
    it <- it + 1
  }
  if (it >= it_max)
  {
    return ("Iteraciones maximas superadas.")
  }
  else{
    return (x1)
  }
}
```

Figura 5.7: Método de Gauss Seidel

4. Comparativa entre lenguajes de programación

Vamos a analizar la comparativa entre los tiempos de ejecución de los algoritmos en R y los implementados en Matlab. En primer lugar, los algoritmos de la primera sección hacen uso de las

factorizaciones de matrices analizadas en el capítulo anterior, donde ya se realizó la comparación.

En la Tabla 5.1 observamos los tiempos de ejecución de los algoritmos implementados tanto en R como en Matlab. El cálculo se ha obtenido mediante la ejecución de diez repeticiones de cada método y tomando la media de todas ellas. En los métodos no iterativos (Eliminación de Gauss y Gauss Jordan) es donde se aprecia una diferencia mayor entre el rendimiento de Matlab y R, mientras que en los métodos iterativos (Jacobi y Gauss Seidel) los tiempos de R tienden a acercarse a los tiempos de Matlab. A pesar de las diferencias en los métodos no iterativos, los tiempos de R nunca sobrepasan las 4 centésimas, lo cual nos hace pensar que R no traspasa ningún límite negativo y que es un lenguaje perfectamente válido para la implementación de este tipo de algoritmos. Una vez más, al igual que en los capítulos previos, observamos que los tiempos son lo suficientemente pequeños como para que el rendimiento no suponga un factor diferencial a la hora de decidir utilizar R o Matlab para este tipo de algoritmos. Además, observamos que R sigue sin tener nada que envidiar a Matlab en cuanto a las utilidades para desarrollar este tipo de algoritmos, ya que hemos sido capaces de codificar sin ningún problema todos los algoritmos para la resolución de sistemas de ecuaciones lineales que nos habíamos planteado. Se puede apreciar una comparativa método a método en la gráfica de la Figura 5.8.

Método	**R**	MATLAB
Eliminación de Gauss	0.0301	0.0049
Gauss Jordan	0.03267	0.0069
Jacobi	0.01325	0.0057
Gauss Seidel	0.00844	0.0024

Tabla 5.1: Tiempos de ejecución en métodos de resolución de sistemas de ecuaciones lineales

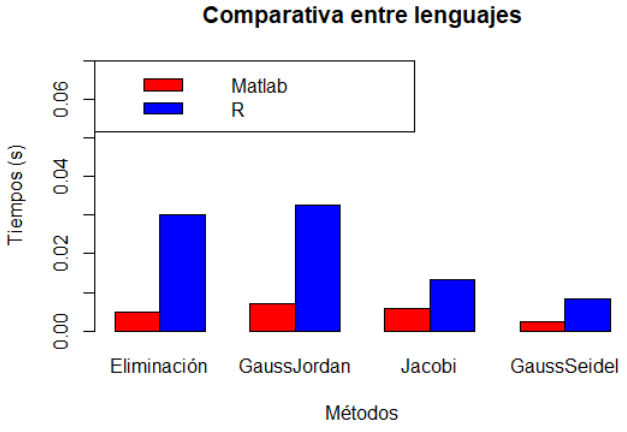


Figura 5.8: Comparativa de tiempos en sistemas de ecuaciones lineales

Capítulo 6

Algoritmos sobre interpolación de funciones

En este capítulo vamos a proponer diferentes maneras de interpolar funciones de manera polinómica, es decir, de aproximar funciones mediante polinomios. Conocidos ciertos puntos de una función, la interpolación trata de generar una función polinómica que sirva para hallar nuevos puntos de la función en cuestión. La interpolación de funciones facilita el tratamiento de los datos. Como aspecto negativo, implica cierto error cometido ya que realmente se está tomando una aproximación de la función real. Los diferentes métodos de interpolación de funciones que vamos a analizar en este capítulo son el polinomio de Taylor, el método de Vandermonde que nos proporcionará un sistema de ecuaciones lineales , el polinomio de interpolación de Lagrange, el método de las diferencias divididas (o método de Newton) y el método de Hermite.

1. Método del polinomio de Taylor

Sea f una función definida en $\mathbb{R}$ y n veces diferenciable en un entorno del punto x_0. Su polinomio de Taylor es el siguiente:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + ... + f^{(n)}(x_0)\frac{(x - x_0)^n}{n!}$$

En este caso el error que se comete al interpolar la función f mediante el polinomio de Taylor es el término que le falta al polinomio de Taylor para ser igual a la expansión en serie de Taylor. Es decir, el error es $f^{(n+1)}(m)\frac{(x-m)^{n+1}}{(n+1)!}$, donde m es un valor en el entorno de x_0.

En la Figura 6.1 está implementada la manera en la que se calcula la imagen de una función dado un punto x y dados los datos necesarios para simplemente poder aplicar el polinomio de

Taylor de orden dos ($f(x_0)$, $f'(x_0)$ y $f''(x_0)$). En la Figura 6.2 implementamos el cálculo de la imagen de una función dado el valor de x, la función y el centro x_0. Utilizamos el polinomio de Taylor como vía de cálculo, hallando manualmente las derivadas primera y segunda de la función.

```
taylor <- function(x0, f_x0, f_prim_x0, f_seg_x0, x){
 return (f_x0 + f_prim_x0*(x-x0) + (1/2)*f_seg_x0*(x-x0)*(x-x0))
}
```

Figura 6.1: Cálculo de la imagen dados los factores (mediante el polinomio de Taylor)

```
f <- function(x){
  return (x^4 + exp(x) - x - 2)
}

deriv_f<- function(x0, times_){
  fun <- expression(x^4 + exp(x) - x - 2)
  for (i in 1:times_){
    deriv <- D(fun, "x")
  }
  x <- x0
  return (eval(deriv))
}

taylor <- function(x0, x){
 return (f(x0) + deriv_f(x0, 1)*(x-x0) +
           (1/2)*deriv_f(x0, 2)*(x-x0)*(x-x0))
```

Figura 6.2: Cálculo de la imagen dada una función (mediante el polinomio de Taylor)

2. Método del sistema de ecuaciones lineales

Vamos a comentar un método sencillo de obtener para la interpolación de una función conocidos ciertos puntos pero que no tiene demasiada relevancia en el ámbito computacional comparado con el resto de métodos de interpolación que veremos a lo largo de este capítulo.

Supongamos que conocemos un conjunto de puntos (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) que pertenecen a la función f que se pretende interpolar. Sea $y = c_n x^n + c_{n-1} x^{n-1} + ... + c_2 x^2 + c_1 x + c_0$ el polinomio final que se desea obtener. Los puntos que conocemos de la función han de verificar la ecuación de dicho polinomio. Este hecho nos genera n ecuaciones que forman un sistema con n incógnitas, siendo las incógnitas los coeficientes del polinomio que pretendemos generar. Resolviendo dicho sistema, ya obtendríamos el polinomio interpolador. Las ecuaciones se plantearían tomando como matriz de coeficientes una matriz de Vandermonde, como podemos ver en la representación matricial que planteamos a continuación:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^n \end{pmatrix}$$

El sistema de ecuaciones lineales planteado podría resolverse con alguna de las técnicas vistas en el capítulo 5.

3. Método del polinomio interpolador de Lagrange

El método de Lagrange es otra forma de interpolación de una funcion dado un cierto número de puntos de la misma.

Supongamos conocidos n+1 puntos que pertenecen a la función (x_0, y_0), (x_1, y_1), (x_2, y_2), ... ,(x_n, y_n). El polinomio interpolador de Lagrange se obtiene de la siguiente manera:

$$L(x) = \sum_{k=0}^{n} y_k l_k$$

y_k son las ordenadas de los puntos que conocemos previamente, mientras que l_k se construye así:

$$l_k = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

Si desarrollamos el productorio, obtenemos:

$$l_k = \frac{(x - x_0)(x - x_1)(x - x_2)...(x - x_{k-1})(x - x_{k+1})...(x - x_n)}{(x_k - x_0)(x_k - x_1)(x_k - x_2)...(x_k - x_{k-1})(x_k - x_{k+1})...(x_k - x_n)}$$

La interpolación de Lagrange presenta una serie de inconvenientes respecto a otros métodos. Uno de ellos es la necesidad de recalcular desde el principio el polinomio interpolador si se altera el número de datos. Por otro lado, su coste es alto cuando se posee un número alto de puntos a interpolar, ya que cuantos más puntos mayor es el grado del polinomio interpolador de Lagrange. Además, la tendencia de que existan grandes oscilaciones entre dos nodos aumenta.

En la Figura 6.3 podemos observar la implementación de un algoritmo que calcula la coordenada y de un punto dada su coordenada x utilizando el polinomio de interpolación de Lagrange. El algoritmo recibe como parámetro una serie de puntos que pertenece a la función a interpolar en los vectores de coordenadas x e y.

```
lagrange <- function(x, y, x0){
 y0 = 0
 for (k in 1:length(x)){
   lk = 1
   for (i in 1:length(x)){
     if (i!= k){
       lk = lk * (x0 - x[i]) / (x[k] - x[i])
     }
   }
   y0 = y0 + lk*y[k]
 }
 return (y0)
}
```

Figura 6.3: Polinomio de interpolación de Lagrange

4. Método de las diferencias divididas

El método de diferencias divididas utiliza el cálculo de las diferencias divididas entre los puntos que se conocen de la función para el cálculo de los coeficientes del polinomio de interpolación.

La diferencia dividida de un solo punto x_i es $f[x_k] = y_k$ La diferencia dividida entre dos puntos consecutivos x_i y x_{i+1} se calcula de la siguiente manera:

$$f[x_i, x_{i+1}] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Aumentando el número de puntos, las diferencias divididas se calculan como sigue:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

$$f[x_i, x_{i+1}, ..., x_{i+j}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+j}] - f[x_i, x_{i+1}, ..., x_{i+j-1}]}{x_{i+j} - x_i}$$

El polinomio interpolador que se pretende obtener posee la forma:

$$Ip(x) = c_0 + c_1(x - x_0) + c_2(x - x_1)(x - x_0) + ... + c_n(x - x_{n-1})(x - x_{n-2}) \cdots (x - x_0)$$

Si evaluamos los puntos que conocemos de la función en el polinomio $Ip(x)$, obtenemos un sistema de ecuaciones tal que:

$$(x_0, y_0) \rightarrow y_0 = c_0$$

$$(x_1, y_1) \rightarrow y_1 = c_0 + c_1(x_1 - x_0)$$

$$(x_2, y_2) \rightarrow y_2 = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_1)(x_2 - x_0)$$

$$...$$

$$(x_n, y_n) \rightarrow y_n = c_0 + c_1(x_n - x_0) + c_2(x_n - x_1)(x_n - x_0) + ... + c_n(x_n - x_{n-1})(x_n - x_{n-2}) \cdots (x_n - x_0)$$

Dicho sistema de ecuaciones se puede resolver utilizando las diferencias divididas comentadas anteriormente, calculando cada coeficiente como se muestra a continuación:

$$c_0 = f[x_0] = y_0$$

$$c_1 = f[x_0, x_1]$$

$$c_2 = f[x_0, x_1, x_2]$$

$$...$$

$$c_n = f[x_0, x_1, ..., x_n]$$

El método de diferencias divididas resulta más ventajoso respecto a los anteriores métodos vistos. En el caso de que se añada un nuevo punto al conjunto de puntos conocidos de la función, nos servirán los cálculos realizados hasta ese momento y no tendremos que comenzar desde un principio a calcular el polinomio interpolador.

En la Figura 6.4 se encuentra la implementación del método de diferencias divididas [25], devolviendo la correspondiente coordenada y dada una x y una serie de puntos iniciales que pertenecen a la función a interpolar. Este algoritmo calcula los coeficientes del polinomio interpolador generado mediante las diferencias divididas y adecuadamente va construyendo el polinomio interpolador, multiplicando cada coeficiente por todos los $(x - x_i)$ correspondientes según lo explicado en esta sección.

```
diferencias_divididas <- function(x, y, x0) {
  require(rSymPy)
  c <- y
  f <- as.character(round(c[1], 5))
  fi <- ''

  for (i in 2:length(x)) {
    for (j in i:length(x)) {
      c[j] <- (c[j] - y[j-1]) / (x[j] - x[j-i+1])
    }
    fi <- paste(fi, '*(x - ', x[i-1], ')', sep = '', collapse = '')

    f <- paste(f, ' + ', round(c[i], 5), fi, sep = '', collapse = '')
    y <- c
  }

  x <- Var('x')
  sympy(paste('e = ', f, collapse = '', sep = ''))
  approx <- sympy(paste('e.subs(x, ', as.character(x0), ')', sep = '',
                        collapse = ''))
  print(f)
  return(as.numeric(approx))
}
```

Figura 6.4: Interpolación por diferencias divididas

5. Método de Hermite

El método de interpolación de Hermite calcula un polinomio de interpolación de manera similar al visto en la sección anterior, ya que se utilizan las diferencias divididas entre puntos conocidos de la función para el cálculo de los coeficientes del polinomio. La principal diferencia es que el método de interpolación de Hermite no solo hace uso de las imágenes de los puntos conocidos de la función, sino que además utiliza las derivadas de la función en los puntos conocidos de antemano.

El polinomio interpolador de Hermite se puede calcular conocidas las derivadas sucesivas de la función en los puntos conocidos, pero vamos a centrarnos en el caso en el que se trabaja solo con la primera derivada. Es necesario añadir un concepto con el que no se trabajaba en el método de las diferencias divididas de Newton, que consiste en el cálculo de las diferencias divididas entre un punto x_i y x_i.

$$f[x_i, x_i] = \lim_{x \to x_i} \frac{f(x) - f(x_i)}{x - x_i} = f'(x_i)$$

El polinomio interpolador de Hermite conocidos n+1 puntos x_0, x_1, x_2,...,x_n, sus imágenes y sus primeras derivadas, se describe a continuación:

$$Ip(x) = f[x_0] + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 + f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) +$$

$$+... + f[x_0, x_0, x_1, x_1, ..., x_n, x_n](x - x_0)^2(x - x_1)^2 \cdots (x - x_n)$$

El método de interpolación de Hermite posee la ventaja de que el polinomio de interpolación coincide en su primera derivada en los puntos utilizados con la primera derivada de la función que se trata de interpolar. También presenta ciertas desventajas, como la necesidad de conocer la primera derivada de los puntos conocidos inicialmente, además de generar un polinomio interpolador de grado muy alto comparado con el número de puntos que se manejan.

Para implementar la interpolación de Hermite podemos utilizar la función ya definida en R denominada *pchip*, que recibe como parámetros las coordenadas x e y de los puntos iniciales que interpolan la función y el valor de x_0, cuya ordenada pretendemos obtener.

6. Comparativa entre lenguajes de programación

Queremos realizar una comparación entre los métodos implementados en R para la interpolación de funciones y sus correspondientes implementaciones equivalentes en Matlab. También pretendemos comentar las funciones predefinidas en Matlab que tienen como objetivo la interpolación de funciones.

Hemos expuesto en R diferentes algoritmos relacionados con la interpolación de funciones. Entre ellos, encontramos códigos relacionados con el polinomio de Taylor, la matriz de Vandermonde resultante con el método del sistema de ecuaciones lineales, el polinomio interpolador de Lagrange, el método de las diferencias divididas de Newton y el método de Hermite. En cuanto al método del sistema de ecuaciones lineales, hemos comentado que se puede resolver mediante uno de los métodos analizados en el capítulo previo. Para el método de Hermite, hemos utilizado una función ya definida en R llamada *pchip*, que recibe como parámetro los puntos conocidos de la función y las coordenadas x de los puntos que se pretende interpolar.

En cuanto a Matlab, existen varias funciones predefinidas en el lenguaje bastante útiles para la interpolación de funciones. Una de ellas es *pchip*, que actúa igual que su función homónima en R. Otra opción es la función *polyfit*, que ajusta una curva polinómica recibiendo por parámetro una serie de puntos y el grado del polinomio que se quiere interpolar, devolviendo los coeficientes del polinomio interpolado. Además, por ejemplo, dispone de una función para el cálculo de la serie de Taylor (*taylor*).

De nuevo, se comparan resultados de tiempos de ejecución tomando como valor el resultado de aplicar la media a los tiempos de diez ejecuciones de cada método. Además, se toman similares datos de entrada para un mismo algoritmo en los dos lenguajes. En la Tabla 6.1 podemos observar estos resultados. En la Figura 6.5 apreciamos la comparativa, viendo que en el método de las diferencias divididas es donde la diferencia es más notable respecto a un lenguaje u

otro, mientras que en los otros tres lenguajes los tiempos de R se asemejan a los de Matlab. El método de diferencias divididas que hemos propuesto es mejorable en cuanto a rendimiento, pero hay que tener en cuenta que su implementación contiene una preparación concreta de los datos para generar un output con unas características estéticas concretas, lo cual probablemente ha empeorado su rendimiento final. En los métodos de Lagrange y Taylor los tiempos son muy similares y no hay gran diferencia entre R y Matlab, mientras que sorprende que las funciones ya definidas *pchip* difieran de esta manera entre R y Matlab, aunque en ningún caso los tiempos sobrepasan ningún límite. Una vez más, podemos concluir que el tiempo de ejecución no es un factor determinante a la hora de escoger qué lenguaje utilizar para desarrollar algoritmos de cálculo científico, y más en concreto, para implementar algoritmos relacionados con la interpolación de funciones polinómicas. R vuelve a disponer de las herramientas necesarias para llevar a cabo un nuevo tipo de métodos de cálculo científico, contando tanto con funciones predefinidas como con las utilidades necesarias para desarrollar los métodos que precisemos. Aunque Matlab sea un lenguaje precisamente diseñado para esta serie de labores, R puede actuar como lenguaje conductor sin ningún tipo de problema.

Método	**R**	MATLAB
Taylor	0.01125	0.0091
Lagrange	0.0156	0.0102
Diferencias divididas	0.0553	0.0114
Hermite	0.019	0.0041

Tabla 6.1: Tiempos de ejecución en métodos de interpolación de funciones

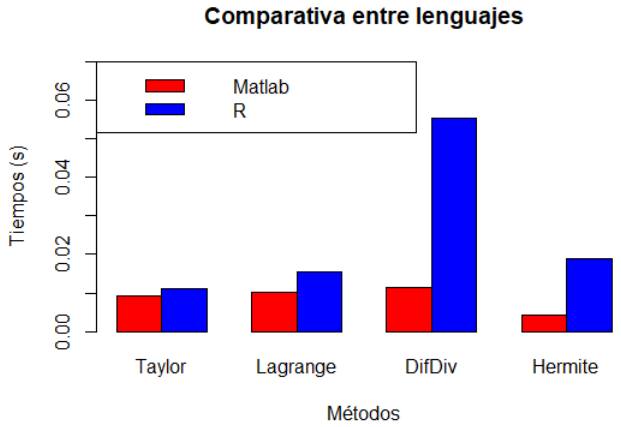


Figura 6.5: Comparativa de tiempos en interpolación

Capítulo 7

Algoritmos sobre derivación e integración

En este capítulo vamos a analizar diferentes maneras de obtener aproximaciones a las derivadas e integrales de una función cuando son conocidos ciertos puntos de dichas funciones.

La derivación tiene diversas aplicaciones en multitud de campos. En la física, la derivación presenta aplicaciones en termodinámica y cinemática, entre muchos otros. Está presente en la química, en la biología e incluso en la arquitectura para optimizar y diseñar, además de en la economía para optimizar diversas funciones y realizar predicciones de evolución. La integración es una herramienta utilizada en muchos campos también, pero su relevancia es fundamental en ámbitos como la ingeniería o la física.

1. Derivación

La derivación numérica sirve para calcular aproximaciones a la derivada de una función cuando se conocen ciertos puntos de la misma. Puede resultar útil ya que aporta una solución al problema de derivación de una función cuando la misma no es conocida, o simplemente resta complejidad al cálculo de ciertas derivaciones.

1.1. Método diferencias centrales

Sean (x_i, y_i) y (x_j, y_j) dos puntos pertenecientes a la función f que se pretende derivar y la cual no conocemos. Si pretendemos obtener la derivada de f en x_k, siendo x_k el punto medio entre x_i y x_j, podemos utilizar una fórmula derivada del desarrollo de Taylor como podemos ver a continuación. La distancia entre x_k y las abscisas de los dos puntos conocidos es de h.

Por un lado, tomamos:

$$f(x_k + h) = f(x_k) + hf'(x_k)$$

Por otro lado, cogemos:

$$f(x_k - h) = f(x_k) - hf'(x_k)$$

Restando ambas ecuaciones, obtenemos:

$$f(x_k + h) - f(x_k - h) = 2hf'(x_k)$$

$$f'(x_k) = \frac{f(x_k + h) - f(x_k - h)}{2h}$$

$$f'(x_k) = \frac{f(x_j) - f(x_i)}{2h}$$

En la Figura 7.1 observamos la implementación del método.

```
cent <- function(x0, x1, x2, f_x0, f_x2){
  h <- abs(x1-x0)
  if (abs(x2-x1) != h){
    return ('No son nodos equiespaciados.')
  }
 return ((f_x2 - f_x0)/(2*h))
}
```

Figura 7.1: Diferencias centrales

1.2. Método diferencias progresivas

Partiendo del mismo razonamiento que en el método anterior y utilizando el desarrollo de Taylor conocidos dos puntos de la función (x_i, y_i) y (x_j, y_j) con distancia h entre sus abscisas, tenemos:

$$f(x_i + h) = f(x_i) + hf'(x_i)$$

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$$f'(x_i) = \frac{f(x_j) - f(x_i)}{h}$$

En la Figura 7.2 observamos su aplicación.

```
progre <- function(x0, x1, f_x0, f_x1){
  h <- abs(x1-x0)
 return ((f_x1 - f_x0)/(h))
}
```

Figura 7.2: Diferencias progresivas

1.3. Método diferencias regresivas

De igual manera, conocidos (x_i, y_i) y (x_j, y_j) con distancia h entre sus abscisas, se puede partir del desarrollo de Taylor y obtener la siguiente fórmula:

$$f(x_j - h) = f(x_j) - hf'(x_j)$$

$$f'(x_j) = \frac{f(x_j) - f(x_j - h)}{h}$$

$$f'(x_j) = \frac{f(x_j) - f(x_i)}{h}$$

En la Figura 7.3 se muestra la fórmula implementada.

```
regre <- function(x_1, x0, f_x_1, f_x0){
  h <- abs(x_1-x0)
 return ((f_x0 - f_x_1)/(h))
}
```

Figura 7.3: Diferencias regresivas

1.4. Derivadas sucesivas

Los métodos vistos anteriormente se pueden utilizar para desarrollar las fórmulas de las derivadas sucesivas de una función. Por ejemplo, para calcular la segunda derivada de f se puede partir del método de diferencias progresivas para ir obteniendo la fórmula deseada. Conocidos tres puntos con abscisas x_j, x_{j+1} y x_{j+2} obtenemos el siguiente resultado:

$$f''(x_j) = \frac{f'(x_{j+1}) - f'(x_j)}{h}$$

$$f''(x_j) = \frac{\frac{f(x_{j+2})-f(x_{j+1})}{h} - \frac{f(x_{j+1})-f(x_j)}{h}}{h}$$

$$f''(x_j) = \frac{f(x_{j+2}) - 2f(x_{j+1}) + f(x_j)}{h^2}$$

De manera similar, se pueden obtener fórmulas para derivadas de mayor grado partiendo de los métodos vistos.

En la Figura 7.4 podemos observar la fórmula implementada para la segunda derivada de una función.

```
regre <- function(x_2, x_1, x0, x1, x2, f_x_2, f_x_1, f_x1, f_x2){
  h <- abs(x1-x0)
  if (abs(x2-x1) != h | abs(x0-x_1) != h | abs(x_1-x_2) != h)
  {
    return('Nodos no equiespaciados!')
  }
 return ((f_x2-f_x1-f_x_1+f_x_2)/(2*h*h))
}
```

Figura 7.4: Segunda derivada

1.5. Método de derivación por el polinomio interpolador de Lagrange

Otra manera de obtener una aproximación a la derivación de una función f es mediante la obtención de su polinomio de interpolación y su posterior derivación. En el tema anterior estudiamos diferentes vías de interpolación de una función. Por ejemplo, tomando la interpolación de Lagrange para aproximar una función f conocidos n+1 puntos, obtenemos el siguiente razonamiento:

La interpolación de Lagrange se obtenía como se muestra a continuación:

$$L(x) = \sum_{k=0}^{n} y_k l_k$$

Siendo l_k:

$$l_k = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

Por lo tanto, para obtener la primera derivada de la función bastaría con calcularla de la siguiente manera:

$$L(x) = \sum_{k=0}^{n} y_k l'_k$$

La implementación de este método es similar al visto en el capítulo anterior acerca de la interpolación de Lagrange. La única diferencia sería la de derivar el factor l_k tras calcular su valor.

2. Integración

La integración numérica nos permite aproximar el valor de la integral de una función. Los métodos de integración numérica resultan útiles en diferentes situaciones y contextos, ya que a veces no se conoce la función a integrar y simplemente se conocen ciertos puntos de la función, o sí se conoce la función pero el cálculo de la primitiva resulta excesivamente complejo.

Una forma de obtener la integral de una función es la de proceder mediante la integración del polinomio de interpolación de la misma. De esta idea se derivan multitud de métodos que analizaremos a lo largo de esta sección.

2.1. Método del rectángulo

Sean (x_0, y_0) y (x_1, y_1) puntos conocidos de la función f a integrar, el método del rectángulo pretende calcular la integral definida entre x_0 y x_1 mediante el cálculo del área del rectángulo que tiene por base $[x_0, x_1]$ y como altura la imagen de uno de los extremos de la base, y_0 o y_1.

La integral quedaría de la siguiente manera, si se toma la imagen de x_0 o de x_1, respectivamente, siendo $h = x_1 - x_0$ la distancia entre los extremos del intervalo de integración:

$$\int_{x_0}^{x_1} f(x)dx = h \cdot y_0$$

$$\int_{x_0}^{x_1} f(x)dx = h \cdot y_1$$

En la Figura 7.5 observamos la implementación del método del rectángulo utilizando la imagen del punto de menor abscisa.

```
rectangulo <- function(x0, x1, f_x0){
  h <- abs(x1-x0)
 return (h*f_x0)
}
```

Figura 7.5: Método del rectángulo

2.2. Método del punto medio

El método del punto medio persigue la misma idea que el método del rectángulo que hemos visto anteriormente, pero con una diferencia. El método del punto medio calcula el área del rectángulo que tiene como base la misma que el método del rectángulo, es decir, la formada por los extremos del intervalo a integrar, y que tiene como altura la imagen del punto medio del intervalo. En este caso, si x_1 es el punto medio de los extremos de integración x_0 y x_2, entonces la integral se calcularía como el producto de $h = x_2 - x_0$ y la imagen de x_1, es decir, y_1. La integral quedaría de la siguiente manera:

$$\int_{x_0}^{x_2} f(x)dx = h \cdot y_1$$

La fórmula implementada se puede ver en la Figura 7.6.

```
punto_medio <- function(x0, x2, f_x1){
  h <- abs(x2-x0)
 return (h*f_x1)
}
```

Figura 7.6: Método del punto medio

2.3. Método del trapecio

El método del trapecio se basa en la idea de aproximar la integral de una función en un intervalo conocidos los extremos del mismo. Sean (x_0, y_0) y (x_1, y_1) puntos conocidos de la función f, la integral de $f(x)$ se puede calcular de la siguiente forma, siendo $h = x_1 - x_0$ la distancia entre los extremos del intervalo de integración:

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2} \cdot (y_0 + y_1)$$

Observamos el método del trapecio en la Figura 7.7.

```
trapecio <- function(x0, x1, f_x0, f_x1){
  h <- abs(x1-x0)
 return ((h/2)*(f_x0+f_x1))
}
```

Figura 7.7: Método del trapecio

Si son conocidos n+1 puntos de la función (x_0, y_0), (x_1, y_1), ... , (x_n, y_n), podemos obtener una aproximación a la integral de la función f mediante la aplicación del método del trapecio a cada uno de los subintervalos formados por cada pareja de puntos conocidos. La fórmula para la aplicación del método quedaría así:

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{2} \cdot (y_0 + y_1) + \frac{h}{2} \cdot (y_1 + y_2) + ... + \frac{h}{2} \cdot (y_{n-1} + y_n)$$

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{2} \cdot (y_0 + 2y_1 + 2y_2 + ... + y_n)$$

El resultado de este método es la obtención de un área encerrada sobre el eje de abscisas equivalente a la de un trapecio en cada subintervalo, obteniendo de esta manera una aproximación a la integral real de la función.

En la Figura 7.8 observamos el método del trapecio compuesto implementado.

```
trapecio_comp <- function(x0, xn, y){
  n <- length(y)
  h <- abs(xn-x0)/(n-1)
  tot <- 0
  for (i in 1:n){
    if (i == 1 | i == n){
      tot = tot + y[i]
    }
    else{
      tot = tot + 2*y[i]
    }
  }
 return ((h/2)*tot)
}
```

Figura 7.8: Método del trapecio compuesto

2.4. Método de Simpson

La fórmula de Simpson es un método de cálculo de una aproximación de la integral de una función mediante el uso de tres puntos de la misma, uno más que con el método del trapecio. Sean (x_0, y_0), (x_1, y_1) y (x_2, y_2) puntos de la función f, la aproximación de la integral de f

mediante el método de Simpson se realiza de la siguiente manera:

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3} \cdot (y_0 + 4y_1 + y_2)$$

La implementación del método de Simpson se encuentra en la Figura 7.9.

```
simpson <- function(x0, x1, x2, f_x0, f_x1, f_x2){
  h <- abs(x1-x0)
 return ((h/3)*(f_x0+4*f_x1+f_x2))
}
```

Figura 7.9: Método de Simpson

Además, igual que con la regla del trapecio, podemos aplicar el método de Simpson cuando se conocen n+1 puntos a cada terna de puntos formando subintervalos. La fórmula quedaría de la siguiente forma, tomando el final de cada subintervalo como comienzo del siguiente:

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{3} \cdot (y_0 + 4y_1 + y_2) + \frac{h}{3} \cdot (y_2 + 4y_3 + y_4) + ... + \frac{h}{3} \cdot (y_{n-2} + 4y_{n-1} + y_n)$$

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{3} \cdot (y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + ... + 4y_{n-1} + y_n)$$

Podemos observar la implementación del método de Simpson compuesto en la Figura 7.10.

```
simpson_comp <- function(x0, xn, y){
  n <- length(y)
  h <- abs(xn-x0)/(n-1)
  tot <- 0
  for (i in 1:n){
    if (i == 1 | i == n){
      tot = tot + y[i]
    }
    else{
      if (i %% 2 == 0){
        tot = tot + 4*y[i]
      }
      else{
        tot = tot + 2*y[i]
      }
    }
  }
 return ((h/3)*tot)
}
```

Figura 7.10: Método de Simpson compuesto

Una variante del método de Simpson es el método de Simpson 3/8, que utiliza cuatro puntos

de la función para obtener la aproximación de la integral. La fórmula del método de Simpson 3/8 es la siguiente:

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8} \cdot (y_0 + 3y_1 + 3y_2 + y_3)$$

La variante conocida como método de Simpson 3/8 está implementada en la Figura 7.11.

```
simpson_tresoct <- function(x0, x1, x2, x3, f_x0, f_x1, f_x2, f_x3){
  h <- abs(x1-x0)
 return (((h*3)/8)*(f_x0+3*f_x1+3*f_x2+f_x3))
}
```

Figura 7.11: Método de Simpson 3/8

A su vez, el método de Simpson 3/8 también se puede utilizar para, conocidos n+1 puntos, dividir una integral en subintervalos e integrar en ellos, como hemos observado en los métodos anteriores. La integral quedaría como se observa a continuación:

$$\int_{x_0}^{x_n} f(x)dx = \frac{3h}{8}\cdot(y_0+3y_1+3y_2+y_3)+\frac{3h}{8}\cdot(y_3+3y_4+3y_5+y_6)+...+\frac{3h}{8}\cdot(y_{n-3}+3y_{n-2}+3y_{n-1}+y_n)$$

$$\int_{x_0}^{x_n} f(x)dx = \frac{3h}{8} \cdot (y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + 2y_6 + ... + 3y_{n-1} + y_n)$$

Encontramos el método de Simpson 3/8 compuesto en la Figura 7.12.

```
simpson_tresoct_comp <- function(x0, xn, y){
  n <- length(y)
  h <- abs(xn-x0)/(n-1)
  tot <- 0
  for (i in 1:n){
    if (i == 1 | i == n){
      tot = tot + y[i]
    }
    else{
      if (i %%3 == 1){
        tot = tot + 2*y[i]
      }
      else{
        tot = tot + 3*y[i]
      }
    }
  }
 return ((3*h/8)*tot)
}
```

Figura 7.12: Método de Simpson 3/8 compuesto

3. Comparativa entre lenguajes de programación

Por último, vamos a comparar los tiempos de ejecución en R y en Matlab de los diferentes métodos de derivación e integración numérica vistos en este capítulo.

En cuanto a la derivación numérica, hemos expuesto una serie de mecanismos para su cálculo. El método de la derivación por el polinomio interpolador de Lagrange hace uso del algoritmo desarrollado en el capítulo anterior acerca del cálculo del polinomio interpolador de Lagrange. El resto de métodos consisten en un conjunto de operaciones matemáticas lo suficientemente directas como para que sus tiempos de ejecución sean ínfimos. Con los algoritmos de integración numérica ocurre algo similar, ya que la obtención de las integrales en los diferentes métodos expuestos supone un coste computacional lo suficientemente bajo como para que no se aprecie diferencia significativa con los tiempos de ejecución de Matlab. Al igual que ocurre con los métodos vistos de derivación numérica, los algoritmos de integración numérica consisten en una serie de operaciones matemáticas muy detalladas que nos permiten obtener el resultado de una manera veloz. No todos los métodos son totalmente directos, ya que también hemos implementado los métodos compuestos del trapecio, Simpson y Simpson 3/8. En la Tabla 7.1 podemos observar una comparación entre R y Matlab acerca de los tiempos de ejecución de dichos algoritmos, observando como resultado el tiempo medio de diez ejecuciones de cada método. En la Figura 7.13 podemos apreciar la comparativa método a método entre lenguajes, donde nos podemos dar cuenta de que en ciertos algoritmos Matlab es mejor que R aunque R no tenga malos tiempos en ninguno de ellos. Además, en los métodos que consisten en operaciones sencillas y relativamente directas, los tiempos entre lenguajes tienden a igualarse. Podemos apreciar mayor diferencia en los métodos compuestos del trapecio, Simpson y Simpson 3/8, en los que Matlab obtiene un mejor rendimiento que R. En el resto de métodos no compuestos los tiempos tienden a igualarse. De nuevo, utilizamos inputs similares para un mismo algoritmo en los dos lenguajes, con intención de no obtener diferencias en cuanto al tiempo de ejecución por motivo de recibir un input diferente. Los resultados nos vuelven a mostrar lo que ya hemos podido observar en capítulos anteriores. R es una herramienta capacitada para implementar los diferentes tipos de algoritmos de cálculo científico que nos propongamos, sin suponer un problema en cuanto a utilidades o a tiempos de ejecución.

Método	**R**	MATLAB
Diferencias regresivas	0.0032	0.0021
Diferencias centrales	0.001	0.001
Diferencias progresivas	0.0071	0.0017
Rectángulo	0.001	0.001
Punto medio	0.001	0.001
Trapecio	0.008	0.0012
Trapecio compuesto	0.027	0.0015
Simpson	0.001	0.001
Simpson compuesto	0.021	0.0018
Simpson 3/8	0.0019	0.001
Simpson 3/8 compuesto	0.0292	0.0021

Tabla 7.1: Tiempos de ejecución en métodos de derivación e integración numérica

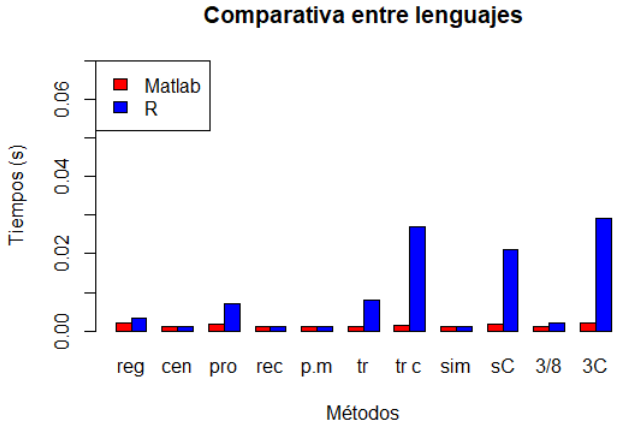


Figura 7.13: Comparativa de tiempos en derivación e integración

Capítulo 8

Análisis del pensamiento crítico en los estudiantes preuniversitarios a través de una experiencia práctica

Además de utilizar R para asignaturas de cálculo científico en la universidad, también puede ser muy útil para introducir la programación en el instituto a jóvenes en los cursos comprendidos entre tercero de la Educación Secundaria Obligatoria y segundo de Bachillerato. El mundo está cambiando de manera muy rápida y la educación de los más jóvenes no debe ser excepción ante esta revolución tecnológica [2, 6, 8, 10, 13, 16, 18, 24]. En este capítulo vamos a exponer una serie de ponencias impartidas a estudiantes de distintos institutos enmarcadas en la Semana de la Ciencia 2018 y las jornadas de "4ESO+Empresa", llevadas a cabo en la facultad de informática de la Universidad Complutense de Madrid. De este trabajo se han podido extraer unos resultados que nos permiten analizar el nivel de pensamiento crítico [15, 21] y el grado de percepción de los estudiantes sobre este tema, gracias a una serie de encuestas que rellenaron los estudiantes en el momento de las exposiciones. Además, analizaremos el trabajo realizado al respecto que nos permitió publicar un artículo relacionado con el mundo de la educación y las nuevas tecnologías en el congreso internacional INTED 2019.

1. Preparación y ejecución de la experiencia con los estudiantes

En la Semana de la Ciencia 2018 organizada por la Universidad Complutense de Madrid realizamos unas sesiones destinadas a institutos tituladas '¡Algoritmos matemáticos: Vaya historia!'. A la cita acudieron 4 institutos de Madrid con un total de 148 jóvenes, teniendo que dejar fuera a varios institutos que solicitaron su presencia por aforo completo. Los centros educativos

que acudieron a la cita fueron los siguientes: IES Beatriz Galindo, IES Las Musas y colegio El Porvenir. Además, en las jornadas "4ESO+Empresa", acudieron 27 jóvenes a participar en sesiones similares a las impartidas en la Semana de la Ciencia.

El objetivo de estas sesiones consistía en introducir a los jóvenes al mundo de la programación mediante un lenguaje de libre distribución y uso sencillo como R, introduciendo una serie de algoritmos de menor a mayor complejidad, fomentando el pensamiento crítico de los estudiantes al exponer diferentes vías de complejidad variable para resolver un mismo problema. Además, se enmarcaron dichos algoritmos en un contexto histórico y con relación con algunos de los matemáticos más famosos de la historia, como Gauss, Fibonacci o Pitágoras, entre otros. Hay que tener en cuenta que para la gran mayoría, esta era su primera toma de contacto con la programación, por lo que se tuvo que ahondar con tranquilidad en los algoritmos explicados.

Las sesiones tenían dos modalidades diferentes. Una modalidad destinada a estudiantes de tercero y cuarto de la ESO, y otra para los estudiantes de Bachillerato. En la primera modalidad se hizo más hincapié en algoritmos iterativos, mientras que para los estudiantes de Bachillerato se introdujo el concepto de recursión.

1.1. Sesión iterativos

En la sesión de algoritmos iterativos, en primer lugar, se comenzó exponiendo la fórmula de la serie aritmética de Gauss. La idea de este apartado era la de mostrar que las cosas se pueden pensar desde diferentes puntos de vista para alcanzar un mismo objetivo, obteniendo diferentes soluciones con diferente complejidad. Para calcular la suma de los 100 primeros números naturales se expuso la idea de Gauss, que mejoraba notablemente la idea inicial de sumar los 100 números uno tras otro. La fórmula de la serie aritmética de Gauss es la siguiente:

$$s = n \cdot \frac{a_1 + a_n}{2}$$

Posteriormente, se fueron explicando una serie de algoritmos mientras los alumnos los plasmaban en sus ordenadores y probaban ejemplos. En primer lugar, se expuso el concepto de terna pitagórica: (a, b, h) verificando que $h^2 = a^2 + b^2$. El primer algoritmo que se explicó consistía en una función que, dados tres elementos por parámetro, devuelve si forman una terna pitagórica o no (Figura 8.1).

Posteriormente, se introdujo el concepto de bucle y se comenzó a ahondar en la cuestión de la complejidad. El siguiente código expone cómo obtener las ternas pitagóricas que pueden generarse con los primeros 100 números naturales mediante tres bucles anidados (Figura 8.2). Dicho código es muy ineficiente ya que realiza multitud de iteraciones innecesarias, por lo que a continuación se propusieron soluciones directas a la necesidad de obtener ternas pitagóricas.

```
forman_terna <- function(a, b, h) {
  if(a*a + b*b == h*h) {
    print('FORMAN TERNA')
   }
  else {
    print('NO FORMAN TERNA')
  }
}
```

Figura 8.1: Forman terna

```
for (a in 1:100){
  for(b in 1:100){
    for(h in 1:100){
      if (h==sqrt(a*a + b*b)){
        terna <- c(a, b, h)
        print(terna)
      }
    }
  }
}
```

Figura 8.2: Fuerza bruta

El método babilonio aparece en el libro X de los Elementos de Euclides y sirve para obtener una terna pitagórica dados dos elementos m y n. Se trata de un método directo y resulta muy útil para evitar iteraciones innecesarias como en el algoritmo anterior. Se trató de transmitir a los estudiantes la importancia de este tipo de métodos. Las ecuaciones del método son las que observamos en la Figura 8.3, obteniendo tres elementos a, b y h tal que forman una terna pitagórica. En la Figura 8.4 observamos el método pitagórico, método directo de cálculo de ternas pitagóricas introducido por Pitágoras y que devuelve una terna pitágorica dado un solo valor de entrada k.

Además, se expuso cómo utilizar un método directo para obtener tantas ternas pitagóricas como pretendamos. Este resultó el punto clave de la sesión ya que los estudiantes pudieron apreciar que para sacar múltiples ternas pitagóricas bastaba con iterar tantas veces como ternas pitagóricas se quisieran obtener, al contrario que con el algoritmo de los bucles anidados. Dicha función está implementada en al Figura 8.5.

```
a <- m*m - n*n
b <- 2*m*n
h <- m*m + n*n
```

Figura 8.3: Método babilonio

```
  a <- k*k - 1
  b <- 2*k
  h <- k*k + 1
```

Figura 8.4: Método pitagórico

```
for (k in valores_k){
  a <- k*k - 1
  b <- 2*k
  h <- k*k + 1

  terna <- c(a, b, h)
  print(terna)
}
```

Figura 8.5: Método pitagórico. Múltiples ternas

Para concluir la sesión, se introdujo una curiosa relación entre las ternas pitagóricas y la sucesión de Fibonacci. Esta relación es otro método directo para calcular una terna pitagórica, solo que esta vez los elementos de entrada son cuatro elementos consecutivos de la sucesión de Fibonacci. La relación se puede observar en la Figura 8.6, siendo $v1$, $v2$, $v3$ y $v4$ cuatro elementos consecutivos de la sucesión de Fibonacci y a, b y h los elementos que forman una terna pitagórica.

```
chequea_pitagoras <- function(v1, v2, v3, v4) {
  a = v1*v4
  b = 2*v2*v3
  h = v2*v2 + v3*v3

  print(paste0("Para: ", v1, " ", v2, " ", v3, " ", v4))

  if (a*a + b*b == h*h){
    print(" se cumple ")

  }
  else{
    print(" no se cumple ")

  }
}

for (i in 1:(length(sucesion_fibo)-3)){
  v1 <- sucesion_fibo[c(i)]
  v2 <- sucesion_fibo[c(i+1)]
  v3 <- sucesion_fibo[c(i+2)]
  v4 <- sucesion_fibo[c(i+3)]

  chequea_pitagoras(v1, v2, v3, v4)
}
```

Figura 8.6: Relación entre ternas pitagóricas y Fibonacci

1.2. Sesión recursivos

En la segunda sesión, se trató de realizar una comparativa mediante algoritmos iterativos y recursivos con conceptos conocidos y suficientemente relevantes como la función factorial y la sucesión de Fibonacci. La idea es que los estudiantes comprendiesen tanto la vía iterativa como la recursiva, que probasen los algoritmos con sus propios ejemplos y que comparasen tiempos de ejecución, valorando que existen soluciones más óptimas que otras para resolver un mismo problema.

En primer lugar, se expuso la función factorial de manera recursiva, además de la iterativa. Se pueden observar las implementaciones en las Figuras 8.7 y 8.8. En una función así, la diferencia entre utilizar el algoritmo iterativo y el algoritmo recursivo es insignificante, y sirve como punto de partida para mostrar que con los cálculos de la sucesión de Fibonacci no ocurre lo mismo.

Para la sucesión de Fibonacci, se comenzó proponiendo una solución recursiva que utiliza dos llamadas recursivas. Se puede observar en la Figura 8.9. Esta solución no es óptima y así se expuso, además de servir como buen ejemplo para que los estudiantes entendiesen la recursión de una manera muy directa.

Además, en la Figura 8.10 está implementado un algoritmo iterativo para el cálculo de la sucesión de Fibonacci. Posteriormente, se propuso un algoritmo para el cálculo recursivo de la sucesión de Fibonacci mediante una sola llamada recursiva (Figura 8.11).

Para finalizar, se expuso una solución de nivel más elevado que utiliza recurrencia para calcular de manera directa términos de la sucesión de Fibonacci.

```
fact_recur <- function(n){
  if (n==0) return (1)
  else return (n*fact_recur(n-1))
}
```

Figura 8.7: Factorial recursivo

```
fact_iter <- function(n){
  total <- 1
  for (num in n:1){
    total <- total * num
  }
  return(total)
```

Figura 8.8: Factorial iterativo

```
fib<-function (n){
  if ((n==0)|(n==1)) return(n)
  else return(fib(n-1)+fib(n-2))
}
```

Figura 8.9: Sucesión de Fibonacci con dos llamadas recursivas

```
fib_iter<-function(n){
  f1<-0;
  f2<-1;
  aux<-f1+f2
  while (n>1){f1<-f2;f2<-aux;aux<-f1+f2;n<-n-1}
  return(f2)
}
```

Figura 8.10: Sucesión de Fibonacci mediante método iterativo

```
fib_final<- function(n, f1, f2){
  if (n==1) return(f2)

  else return(fib_final(n-1,f2, f1+f2))

}
```

Figura 8.11: Sucesión de Fibonacci con una llamada recursiva

2. Resultados obtenidos

El trabajo realizado en las sesiones expuestas en el apartado anterior sirvió como base para elaborar una publicación que relaciona el mundo de la educación con la era digital que estamos viviendo. Al término de las sesiones, los estudiantes rellenaron unos cuestionarios donde respondían diferentes preguntas en las que se buscaba evaluar su nivel de pensamiento crítico, su atracción por el mundo tecnológico y su predisposición por encaminar su futuro en el ámbito científico, permitiendo realizar filtros por edad y sexo. Para el artículo publicado se han utilizado las respuestas obtenidas en las sesiones de la Semana de la Ciencia 2018, ya que el congreso tuvo lugar antes de que se impartiesen las jornadas "4ESO+Empresa". A pesar de ello, los resultados de estas jornadas también son útiles y nos aportan cierta información que analizaremos posteriormente. Cabe destacar que la Semana de la Ciencia fue una actividad organizada para clases completas de instituto, es decir, no hubo voluntariedad en la asistencia de los participantes, sino que fue una decisión de sus profesores o responsables. No sucedió lo mismo con las jornadas de "4ESO+Empresa", ya que a esta experiencia acudieron jóvenes de manera voluntaria e independiente. Dichas jornadas consistieron en tres días consecutivos de actividades para jóvenes que realizaron una especia de estancia, donde pudieron conocer gente nueva con inquietudes similares a las suyas y participar de manera voluntaria en actividades relacionadas con sus intereses. Este

hecho puede ser interesante de conocer a la hora de analizar las respuestas con cierta perspectiva. El género y las edades de los participantes se pueden observar en la Tabla 8.1

		Semana de la Ciencia	4ESO+Empresa
Sexo	**Chicas**	57.7 %	44.4 %
	Chicos	42.3 %	55.6 %
Curso	**3ESO**	43 %	0 %
	4ESO	34.9 %	37 %
	Bachillerato	22.1 %	63 %

Tabla 8.1: Sexo y edad de los participantes

El artículo ha sido publicado en el congreso internacional INTED 2019, y sus objetivos principales son los siguientes:

- Analizar el pensamiento crítico de los estudiantes, el cual pretendimos fomentar en las sesiones de la Semana de la Ciencia.
- Buscar diferencias significativas en cuanto a la atracción por el mundo de la programación, analizando la brecha de género y la edad.
- Analizar la consecución de los objetivos de las sesiones por los alumnos y su visión acerca de la dificultad de las mismas, observando su percepción sobre la dificultad de los ejercicios y compararla con la dificultad real.

En las encuentas que se realizaron a los estudiantes se preguntaron las siguientes cosas:

- Curso y género.
- Dificultad de cada una de las partes del taller.
- Objetivos cumplidos en cada una de las partes del taller.
- Dificultad general del taller.
- ¿Te ha gustado el taller?
- Gusto por las matemáticas y/o la programación.
- Para entender matemáticas, ¿es importante la imaginación?
- Para programar un algoritmo, ¿es importante la imaginación?
- ¿Crees que podrías aprender a programar algoritmos?

Entre los estudiantes, encontramos un 57.7 % de chicas y un 42.3 % de chicos. De ellos, un 92.6 % considera que la experiencia ha sido satisfactoria y la ha disfrutado. Además, un 85 % completó las actividades propuestas a lo largo de la sesión. En cuanto a las diferencias entre género, no hemos encontrado diferencias notables entre chicas y chicos, ya que de manera general han conseguido sus objetivos y han percibido la dificultad de la actividad en medidas similares. Además, el gusto por las matemáticas y la programación tampoco nota diferencia entre chicos y chicas, lo cual nos hace reflexionar sobre la brecha de género en el mundo tecnológico y la importancia de fomentar también las carreras profesionales tecnológicas entre el género femenino. La única diferencia que hemos podido hallar entre chicas y chicos es la concepción que se tiene sobre qué es lo más importante a la hora de aprender a programar, ya que un alto porcentaje de chicas piensa que es más relevante la imaginación, mientras que los chicos tienden más a pensar que la algoritmia consiste en seguir una serie de pasos definidos.

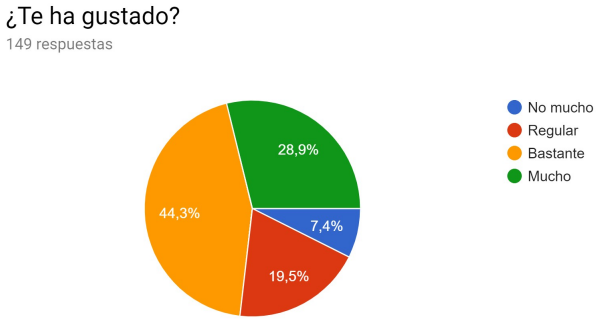


Figura 8.12: Satisfacción con la sesión

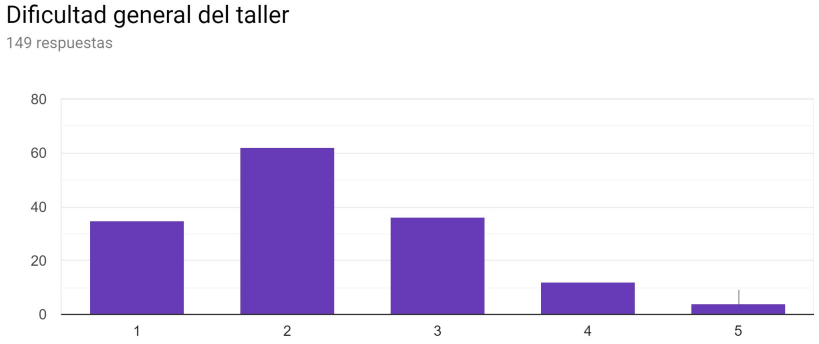


Figura 8.13: Dificultad de la sesión

En cuanto a la edad sí podemos observar que los más mayores tienden a tener un grado de percepción de la dificultad bastante más acertado y encaminado que los más jóvenes, que

tienden a percibir las cosas de una manera quizás más ingenua. En cuanto a la consecución de objetivos, no se han encontrado diferencias reseñables en ningún ámbito.

El artículo completo publicado en el congreso INTED 2019 lo podemos ver en el Apéndice C.

En cuanto a las jornadas "4ESO+Empresa", hemos obtenido 27 respuestas que nos permiten extraer ciertas conclusiones acerca de la percepción de los participantes sobre las sesiones realizadas, su gusto por las matemáticas y la computación y sus intenciones de futuro. De los participantes, un 55 % son chicos mientras que un 45 % son chicas. Un 66.7 % de los participantes opina que RStudio es un software sencillo de manejar. En cuanto al gusto por las matemáticas, un 70.3 % de los participantes siente afinidad hacia las matemáticas, mientras que el 29.6 % de los encuestados no tiene ningún gusto por ellas. Un dato interesante es que casi el 90 % de los participantes opina que para entender las matemáticas y trabajar con ellas es muy importante la imaginación, mientras que tan solo un porcentaje cercano al 60 % cree que la imaginación es importante para el desarrollo de algoritmos. Es decir, un 40 % de los encuestados creen que la programación simplemente consiste en el seguimiento de unos pasos preestablecidos. Este dato nos aporta un claro ejemplo de que el pensamiento crítico de los preuniversitarios no está muy desarrollado en ciertas ocasiones, ya que no se centran en razonar una solución óptima para un resultado sino que es suficiente con obtener una respuesta a un problema mediante unos pasos a seguir.

En las Figuras 8.12, 8.13, 8.14, 8.15 y 8.16 podemos visualizar algunos gráficos con resultados de las respuestas de los participantes en las sesiones.

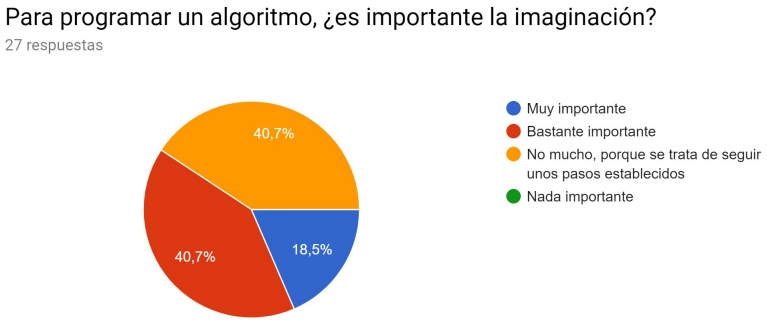


Figura 8.14: Imaginación en la algoritmia

Para enmarcar estas respuestas y su utilidad, cabe destacar que alrededor del 85 % de los participantes en estas sesiones tiene la intención de estudiar un grado de ciencias o ingeniería en la universidad.

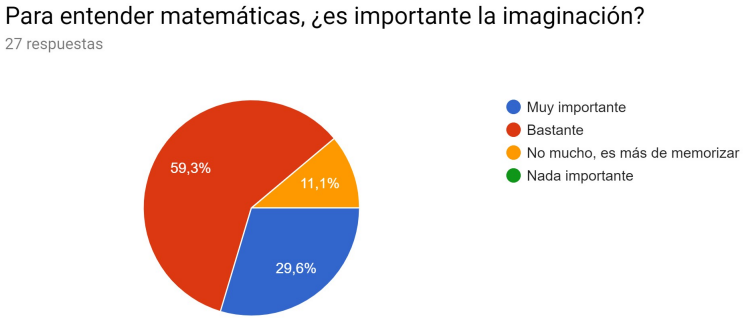


Figura 8.15: Imaginación en las matemáticas

Figura 8.16: Planes de futuro

Capítulo 9

Conclusiones y trabajo futuro

El objetivo principal de este trabajo consistía en ofrecer una alternativa razonable a Matlab para su aplicación en el aula en asignaturas de cálculo científico. Como conclusión principal, hemos probado que R es un lenguaje de programación plenamente capacitado para el desarrollo de algoritmos de cálculo científico. Aunque R es un lenguaje tradicionalmente asociado a tareas estadísticas y analítica de datos, hemos visto que también se puede utilizar para implementar los diferentes métodos de análisis numérico que nos propongamos. Además, no solo es capaz de llevar a cabo los diferentes métodos de programación científica, sino que además lo hace sin influir de manera significativa en el rendimiento de los programas. Como hemos analizado al final de cada capítulo, los tiempos de ejecución de los métodos implementados en R no distan mucho de los tiempos obtenidos mediante el uso de un lenguaje diseñado originalmente para este tipo de objetivos, como es el caso del lenguaje propio de Matlab. El propio lenguaje de Matlab debe tener optimizaciones internas que permiten que los algoritmos más sofisticados mejoren sus prestaciones, pero cuando los algoritmos consisten en operaciones relativamente directas, los tiempos de ejecución entre lenguajes tienden a igualarse.

Hemos probado que R es un lenguaje de software libre totalmente válido para la implementación de algoritmos de cálculo científico, proponiendo de esta manera una alternativa fiable a los lenguajes tradicionalmente asociados a este tipo de tareas.

Por otro lado, también hemos sido capaces de utilizar R para la implementación de los algoritmos utilizados en las experiencias prácticas con preuniversitarios. En esta parte del trabajo hemos sido capaces de analizar la percepción de los jóvenes preuniversitarios sobre sus propias destrezas y acerca del mundo de las matemáticas y de la computación. Además, hemos podido estudiar su percepción de la dificultad y su capacidad de pensamiento crítico. Esta parte del trabajo ha resultado muy gratificante por diferentes motivos. En primer lugar, hemos sido capaces de estudiar las posibles diferencias entre géneros sobre la afinidad hacia las ciencias y la algoritmia, además de analizar las miras de futuro de los estudiantes. Los resultados han sido

los esperados realmente, ya que no existe un sesgo real en cuanto a la afinidad de los estudiantes preuniversitarios por las matemáticas y la programación. Este sesgo está presente en la sociedad y provoca de manera directa o indirecta que muchas chicas no lleguen a estudiar carreras científicas, mientras que el entorno facilita el acceso a los chicos. Estudios como los que hemos realizado en este trabajo pueden ayudar a que la sociedad se quite la venda de los ojos y se comience a fomentar en mayor medida el acceso de las mujeres a las carreras científicas, aportando la visibilidad suficiente por parte del entorno. Por otro lado, ha resultado muy interesante compartir este tipo de sesiones prácticas con jóvenes que aportan su visión sincera sobre su percepción y ayudarlos a vivir un primer encuentro con el mundo de la programación, algo que muchos hemos echado en falta en nuestra etapa preuniversitaria.

En el ámbito personal, este trabajo ha sido una experiencia muy satisfactoria para mí por varios motivos. Por un lado, me ha permitido aprender a programar en R y mejorar mi nivel de Matlab. Además, mi capacidad crítica en cuanto a los lenguajes de programación ha aumentado considerablemente. Por otro lado, las experiencias prácticas con preuniversitarios me han permitido colaborar en la publicación de mi primer artículo en un congreso internacional, con las implicaciones motivacionales y profesionales que ello conlleva.

Una posible aplicación de este trabajo en un proyecto futuro es la creación de un paquete completo en R de cálculo científico. A pesar de la existencia de funciones ya definidas en R que implementan ciertos métodos de cálculo científico, sería útil la creación de una librería que ofrezca la implementación de todos los algoritmos analizados en este trabajo. Este paquete facilitaría mucho el trabajo a todas aquellas personas que por motivos académicos o profesionales necesitan llevar a cabo el desarrollo de métodos de cálculo científico en un lenguaje de programación relacionado con el software libre. Además, también podría ser útil para aquellos desarrolladores que simplemente utilizan R para diversas funcionalidades, que aunque no estén a priori estrechamente relacionadas con el cálculo científico, sí pueden echar mano en algún momento de algún método concreto.

Bibliografía

[1] Aurentz, J. (2017). "Polinomios para construir edificios seguros". *Café y teoremas. El País*. [En línea] https://elpais.com/elpais/2017/09/22/ciencia/1506069453_894253.html [último acceso el 02/05/2019].

[2] Bailin, S. "Critical thinking and science education", *Science & Education*, vol. 11, no. 4, pp. 361-375, 2002.

[3] Bartle, R. y Sherbert, D. *Introducción al Análisis Matemático de una Variable*. Ed. Limusa-Willey, 2010.

[4] Bishop, C. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Burden, R. *Análisis Numérico*. Ediciones Paraninfo, 2002.

[6] Cañas, J.M., Martin, A., Perdices, E. et al "Academic framework for teaching robot programming at university", *Revista Iberoamericana de Automática e Informática Industrial*, vol. 15, no. 4, pp. 404-415, 2018.

[7] Cormen T., Leiserson C., Rivest R. et al. *Introduction To Algorithms*. McGraw-Hill, 1990.

[8] EDISON (2017). http://edison-project.eu/edison/edison-data-science-framework-edsf (último acceso enero 2019).

[9] Gil, M. *Introducción rápida a Matlab y Simulink para ciencia e ingeniería*. Diaz de Santos, 2003.

[10] Huda, M. , Maseleno, A., Atmotiyoso, P. et al, "Big Data emerging technology: insights into innovative environment for online learning resources". *International Journal of Emerging Technologies in Learning* (iJET), vol. 13, no. 1, pp. 23-36, 2018.

[11] Infante del Río, J. y Rey Cabezas J. *Métodos Numéricos. Teoría, problemas y prácticas con MATLAB*. 5ª edición. Editorial Pirámide, 2018.

[12] Jiménez, J. *Laboratorio de Computación Científica*. Universidad Complutense de Madrid, 2017.

[13] Kelley, T. R. y Knowles, J. G. “A conceptual framework for integrated STEM education”, *International Journal of STEM Education*, pp.3:11, 2016.

[14] Kincaid, D. y Cheney, W. *Análisis Numérico: las Matemáticas del Cálculo Científico*. Addison-Wesley Iberoamericana, 1994.

[15] Kitchin, R. “Thinking critically about and researching algorithms”, *Information, Communication & Society* vol. 20, no. 1, pp. 14-29, 2017.

[16] Korkmaz, O., Recep, C. y Özden M.Y. “A validity and reliability study of the Computational Thinking Scales (CTS)”, *Computers in Human Behavior* vol. 72, pp. 558-569, 2017

[17] Martín, I. y Pérez, V. *Cálculo numérico para computación en ciencia e ingeniería*. Editorial Síntesis, 1998.

[18] Martín-H, J.A., Santos, M., López, V. et al, “A software tool for the automatic tracking and segmentation of student’s profiles”, *6th Int. Technology, Education and Development Conference*, INTED, pp. 1511-1517, 2012.

[19] Mathews, J. H. y Fink, K. D. *Métodos Numéricos con MATLAB*. 3ª edición. Editorial Prentice Hall, 2004.

[20] Matloff, N. *The art of R programming. A Tour of Statistical Software Design.* No Starch Press, 2011.

[21] Quinn, V. *Critical thinking in young minds*, Routledge, 2018.

[22] Quintela, P. *Métodos Numéricos en Ingeniería*. Editorial Tórculo, 2001.

[23] Rivera, M. y Arrieta, J. "La algoritmia; una práctica social de las comunidades de ingenieros en sistemas computacionales."*Acta Latinoamericana de Matemática Educativa*, (pp. 456-460), 2007.

[24] Santos, M. y Farias G. “Laboratorios virtuales y remotos de procesamiento de señales”, *Revista Iberoamericana de Automática e Informática Industrial*, vol. 7, no. 1, pp. 91-100, 2010.

[25] Schlegel, A. (2017). "Divided Differences Method of Polynomial Interpolation". *R-bloggers*. [En línea] https://www.r-bloggers.com/divided-differences-method-of-polynomial-interpolation [último acceso el 02/05/2019].

[26] Stallman, R. *Software libre para una sociedad libre*. Traficantes de Sueños, 2004.

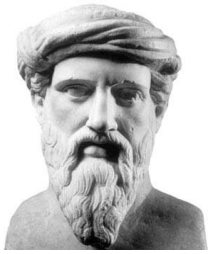
Anexo A

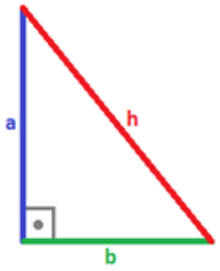
Fichas didácticas utilizadas en las jornadas con preuniversitarios

SEMANA DE LA CIENCIA 2018
Ficha 1 Algoritmos Iterativos

El objetivo de esta ficha es mostrar la **PROGRAMACIÓN ITERATIVA**

AÑO 569aC: Nace Pitágoras, matemático y filósofo griego que fundó la escuela pitagórica: una hermandad religiosa y filosófica con un gran sentido de la matemática en la vida cotidiana.





Todos conocemos el teorema de Pitágoras. Hoy sabemos que los babilonios ya lo conocían pues usaban las ternas que hoy se conocen como pitagóricas.

$$h^2 = a^2 + b^2$$

Una terna (a, b, h) es pitagórica si cumple la ecuación de Pitágoras.

Por ejemplo, (3, 4, 5) es una terna pitagórica, ya que $3^2 + 4^2 = 5^2$.

Otros ejemplos son: (5, 12, 13), (7, 24, 25), (8, 15, 17).

AÑO 300aC: En el **Libro X de los Elementos de Euclides** aparece un método para calcular ternas pitagóricas a partir de dos números m y n:

$$\begin{cases} a = m^2 - n^2 \\ b = 2mn \\ h = m^2 + n^2 \end{cases}$$

Este método fue utilizado por los babilonios alrededor del año 1900 a. C.

Más fácil aún. La siguiente terna se calcula a partir de un solo número k. Este método se conoce como método pitagórico.

$$\begin{cases} a = k^2 - 1 \\ b = 2k \\ h = k^2 + 1 \end{cases}$$

AÑO 1170: Nace Leonardo de Pisa, conocido como Fibonacci y descubre su famosa sucesión:

$$f(n) = \begin{cases} 0 \; si \; n = 0 \\ 1 \; si \; n = 1 \\ f(n-1) + f(n-2) \; si \; n \geq 2 \end{cases}$$

Sucesión de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

A partir de **cuatro términos consecutivos** de la sucesión de Fibonacci: v1, v2, v3, v4 se puede obtener una terna pitagórica. Veamos cómo:

Primer cateto: Calcular el producto de los extremos: v1 × v4

Segundo cateto: Calcular el doble del producto de los dos términos del medio: 2 × v2 × v3

Hipotenusa: Calcular la suma de los cuadrados de los términos del medio: $v2^2 + v3^2$

Por ejemplo, (3, 5, 8, 13) produce la terna (39, 80, 89)

SEMANA DE LA CIENCIA 2018
Ficha 2. Algoritmos Recursivos

El objetivo de esta ficha es mostrar la programación de **algoritmos recursivos**.

Una de las funciones recursivas más famosas es factorial:

$$fact(n) = \begin{cases} 1 & si\ n = 0 \\ n * fact(n-1) & si\ n \geq 1 \end{cases} \quad \begin{cases} 0! = 1 \\ n! = n * (n-1) * (n-2) * \ldots * 2 * 1 \end{cases}$$

Factorial es útil en la composición de permutaciones, por ejemplo. El cálculo es sencillo y su veremos que el código en un lenguaje de programación es casi cortar y pegar la fórmula matemática.

Otra función recursiva un poquito más difícil es la que hemos mostrado en la presentación. La sucesión de Fibonacci:

$$fib(n) = \begin{cases} 1 & si\ n = 1\ o\ 2 \\ fib(n-1) + fib(n-2) & si\ n \geq 3 \end{cases} \qquad Ecuación\ original$$

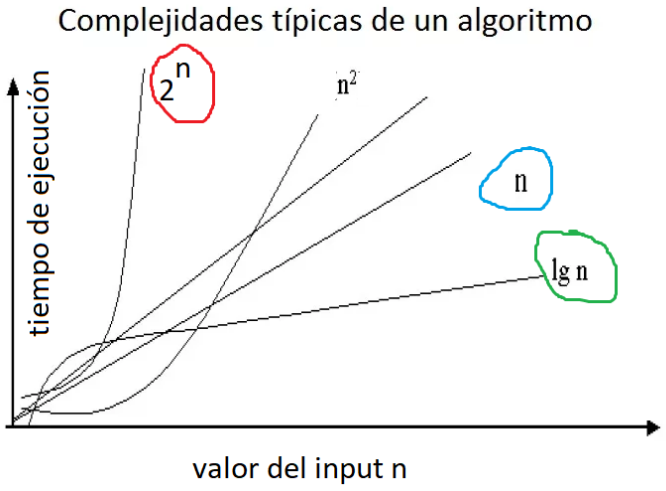
Sucesión de Fibonacci: **1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …**

Una primera implementación puede ser precisamente la fórmula matemática con recursión doble, pero veremos que es muy poco eficiente, pues ni siguiera un buen ordenador será capaz de calcular el Fibonacci de 50, por ejemplo.

Con ayuda de las matemáticas y de la imaginación vamos a implementar otra solución que calcula cualquier Fibonacci en tiempo record, lo que llamamos 'tiempo real'.

$$fib\_final(n, f1, f2) = \begin{cases} f2 & si\ n = 1 \\ fib\_final(n-1, f2, f1+f2)\ si\ n \geq 2 \end{cases} \qquad Ecuación\ generalizada$$

Para calcular números de Fibonacci con esta función hacemos: *fib_final(n, 0,1)*



En la práctica implementaremos varios algoritmos con diferentes complejidades como se marca en esta gráfica.

Rojo: El tiempo de ejecución es tan alto que no podríamos vivir para conocer el resultado.

Azul: El tiempo de ejecución es proporcional al número que actúa como input.

Verde: El tiempo de ejecución será el logaritmo del número actúa como input.

Anexo B

Códigos y documentación utilizados en las jornadas con preuniversitarios

Iterativos.R

Victoria

Mon Nov 12 23:12:21 2018

```
# APRENDER A PROGRAMAR CON R

# HAY DOS VENTANAS: CONSOLA Y EDITOR.
# EN EL EDITOR ESCRIBIREMOS LAS INSTRUCCIONES QUE EL ORDENADOR DEBE EJECUTAR
# EN LA CONSOLA COMPROBAREMOS LOS RESULTADOS

# TRES PASOS FUNDAMENTALES PARA PROGRAMAR

# 1.-COPIAR EN LA VENTANA DEL EDITOR, TODO EXACTAMENTE, EXCEPTO COMENTARIOS (#)
# 2.-SELECCIONAR EL CoDIGO A EJECUTAR,
# 3.-CON EL BOTON DERECHO DEL RAToN SELECCIONAR EJECUTAR
#

### PRIMERA PARTE ###
#EN PRIMER LUGAR VAMOS A ESCOGER TRES NuMEROS Y LOS VAMOS A ALMACENAR EN VARIABLES D
EL ORDENADOR:
a<- 34
b<- 12
h<- 66
#(SELECCIONAR Y EJECUTAR LAS TRES INSTRUCIONES ANTERIORES)
# PARA SABER SI UNA TERNA ES PITAGoRICA PODEMOS COMPROBARLO HACIENDO LA CUENTA
# COMPARANDO LOS RESULTADOS:
a*a+b*b
```

```
## [1] 1300
```

```
h*h
```

```
## [1] 4356
```

```
#(SELECCIONAR Y EJECUTAR LAS DOS INSTRUCIONES ANTERIORES)
# EN LA CONSOLA VEMOS QUE LOS RESULTADOS SON DIFERENTES, ENTONCES 34, 12 Y 66 NO SO
N TERNA PITAGÃRICA

# ES MaS EFICIENTE CREAR UNA FUNCIoN QUE NOS DIGA CUANDO UNA TERNA ES O NO PITAGORIC
A
# ESTO LO HACEMOS ASi: (COPIAR EL CoDIGO EXACTAMENTE EN LA VENTANA DEL EDITOR)

terna <-function (a, b, h){
  if (a*a+b*b==h*h) {print('si forman una terna pitagorica')}
  else {print(' no forman terna pitagorica')}
}
#(SELECCIONAR Y EJECUTAR LAS INSTRUCIONES ANTERIORES)
# AHORA PODREMOS INVOCAR A LA FUNCION CON UN CASO CONCRETO
terna(3,4,5)
```

```
## [1] "si forman una terna pitagorica"
```

```
# OTROS EJEMPLOS
terna(12, 34, 66)
```

```
## [1] " no forman terna pitagorica"
```

```
terna(40, 9, 41)
```

```
## [1] "si forman una terna pitagorica"
```

```
### SEGUNDA PARTE ###
## QUEREMOS CREAR MUCHAS TERNAS PITAGoRICAS. HAY VARIAS FORMAS.
## LA PRIMERA FORMA ES MUY COMPLICADA E INEFICIENTE, PERO VAMOS A VERLA:
# (ESTA NO ES NECESARIO COPIARLA)
t<-0
for (i in 1:20){
  for (j in 1:20) {
    for (k in 1:20){
      if (i*i+j*j==k*k){
        t<-t+1;
        print (c(i,j,k))}
    }
  }
}
```

```
## [1] 3 4 5
## [1] 4 3 5
## [1]  5 12 13
## [1]  6  8 10
## [1]  8  6 10
## [1]  8 15 17
## [1]  9 12 15
## [1] 12  5 13
## [1] 12  9 15
## [1] 12 16 20
## [1] 15  8 17
## [1] 16 12 20
```

```
# DE LAS 8000 TERNAS POSIBLES, SOLO 12 SON PITAGoRICAS.
# ESTE MeTODO INEFICIENTE DE PROGRAMACIoN SE DENOMINA 'FUERZA BRUTA'

t
```

```
## [1] 12
```

```
## ECUACIONES DE EUCLIDES: METEMOS DOS NuMEROS CUALESQUIERA m Y n
## (COPIAR EL CODIGO A CONTINUACIoN)
m<- 56
n<- 34
## Y CALCULAMOS LOS VALORES DE EUCLIDES:
a<-m*m-n*n
b<-2*m*n
h<-m*m+n*n
print(c(a,b,h))
```

```
## [1] 1980 3808 4292
```

```
## AHORA COMPROBAMOS SI HA FUNCIONADO USANDO 'terna'
terna(a,b,h)
```

```
## [1] "si forman una terna pitagorica"
```

```
# PROBAD PARA OTROS VALORES DE m Y n
# ¿QUe OCURRE SI m ES MENOR QUE n?
```

```
## ECUACIONES DE PITAGORAS: METEMOS UN NuMEROS CUALQUIERA k
## (COPIAR  Y EJECUTAR EL CODIGO A CONTINUACIoN)
k<- 87

## Y CALCULAMOS LOS VALORES DE PITAGORAS:
a<-k*k-1
b<-2*k
h<-k*k+1
print(c(a,b,h))
```

```
## [1] 7568  174 7570
```

```
## AHORA COMPROBAMOS SI HA FUNCIONADO USANDO 'terna'
terna(a,b,h)
```

```
## [1] "si forman una terna pitagorica"
```

```
# PROBAD PARA OTROS VALORES DE k
# ¿QUe OCURRE SI k ES MENOR QUE 0?
```

```
### SI NO TENEMOS CUIDADO CON EL INPUT, LOS PROGRAMAS PUEDEN FUNCIONAR MAL.
### LAS CONDICIONES QUE DEBEN CUMPLIR LOS PROGRAMAS SE LLAMAN 'PRECONDICIONES'


### TERCERA PARTE ###
## AHORA QUEREMOS CALCULAR 100 TERNAS PITAGoRICAS CON EL MeTODO ANTERIOR, PERO SIN T
ANTAS
## COMPLICACIONES COMO VIMOS EN EL CASO DE LOS BUCLES ANIDADOS.
## CREAMOS UNA ITERACIoN FOR QUE RECORRE UN RANGO DE VALORES PARA K
## PARA CADA VALOR DE K SE CREA UNA NUEVA TERNA:
## (COPIAR  Y EJECUTAR EL CODIGO A CONTINUACIoN)
for (k in 2:10) {
  a<-k*k-1
  b<-2*k
  h<-k*k+1
  print(c(a,b,h))
}
```

```
## [1] 3 4 5
## [1]  8  6 10
## [1] 15  8 17
## [1] 24 10 26
## [1] 35 12 37
## [1] 48 14 50
## [1] 63 16 65
## [1] 80 18 82
## [1]  99  20 101
```

```
## PROBAD EL CoDIGO CAMBIANDO 10 POR 100, OBSERVAD LOS RESULTADOS. SON DIFERENTES A
LOS DEL ALGORITMO POR FUERZA BRUTA, YA QUE
## AHORA NO HAY PERMUTACIONES, POR ESO LOS NuMEROS SON MaS GRANDES


### CUARTA PARTE ###
### RELACION DE LAS TERNAS PITAGoRICAS CON LOS NuMEROS DE FIBONACCI.
# EN PRIMER LUGAR PONEMOS UNOS CUANTOS NuMEROS DE FIBONACCI EN UN VECTOR
fib<-c(0,1,1,2,3,5,8,13,21,34,55,89)
# ESCOGEMOS CUATRO DE ELLOS SEGUIDOS Y LOS METEMOS EN LAS VARIABLES V1 - V4
v1<-5; v2<-8; v3<-13; v4<-21;
# CON ESTOS NuMEROS CALCULAMOS LOS VALORES DE LA TERNA:
a<- v1*v4
b<-2*v2*v3
h<-v2*v2+v3*v3
# COMPROBAMOS QUE FORMAN TERNA
terna(a,b,h)
```

```
## [1] "si forman una terna pitagorica"
```

```
## CALCULAR OTRAS TERNAS A PARTIR DE SECUENCIAS DE FIBONACCI


### FIN DEL TALLER DE ALGORITMOS ITERATIVOS.
```

Recursivos.R

Victoria

Mon Nov 12 23:20:37 2018

```
# APRENDER A PROGRAMAR CON R

# HAY DOS VENTANAS: CONSOLA Y EDITOR.
# EN EL EDITOR ESCRIBIREMOS LAS INSTRUCCIONES QUE EL ORDENADOR DEBE EJECUTAR
# EN LA CONSOLA COMPROBAREMOS LOS RESULTADOS

# TRES PASOS FUNDAMENTALES PARA PROGRAMAR

# 1.-COPIAR EN LA VENTANA DEL EDITOR, TODO EXACTAMENTE, EXCEPTO COMENTARIOS (#)
# 2.-SELECCIONAR EL CoDIGO A EJECUTAR (INSTRUCCIONES QUE DAMOS AL ORDENADOR)
# 3.-CON EL BOTON DERECHO DEL RAToN SELECCIONAR EJECUTAR
#

### PRIMERA PARTE ###-----------------------------------------------------------
----------
#EN PRIMER LUGAR VAMOS A ASIGNAR EL NuMERO 8 A UNA VARIABLE n:
n<-8
#(SELECCIONAR Y EJECUTAR LA INSTRUCIoN ANTERIOR)

# IMPLEMENTAR LA FUNCIoN FACTORIAL USANDO LA ECUACIoN MATEMaTICA

fact_recur <- function(n){
  if (n==0) return (1)
  else return (n*fact_recur(n-1))
}
# COMPROBAMOS SU FUNCIONAMIENTO REALIZANDO 'INSTANCIAS' CON UN INPUT CONCRETO, EL Nu
MERO 5
print(fact_recur(5))
```

```
## [1] 120
```

```
# TAMBIeN PODEMOS REALIZAR UNA INSTANCIA CON LA VARIABLE n
print(fact_recur(n))
```

```
## [1] 40320
```

```
# TAMBIeN PODEMOS IMPLEMENTAR FACTORIAL MEDIANTE UN PROCESO ITERATIVO, SERiA ASi:

fact_iter <- function(n){
  total <- 1
  for (num in n:1){
    total <- total * num
  }
  return(total)
}

# PODEMOS COMPROBAR QUE FUNCIONA IGUAL QUE LA RECURSIVA MEDIANTE LAS INSTANCIAS
print(fact_iter(5))
```

```
## [1] 120
```

```
print(fact_iter(n))
```

```
## [1] 40320
```

```
# ESTAS VERSIONES DE FACTORIAL SON DISTINTAS PERO AMBAS TIENEN UN TIEMPO DE EJECUCIo
N LINEAL,
# ESO SIGNIFICA QUE AUNQUE EL INPUT SEA GRANDE NO TENDREMOS QUE ESPERAR,
# A VECES DECIMOS QUE SE EJECUTA EN 'TIEMPO REAL' POR ESTA RAZoN.
## EN ESTE TALLER QUEREMOS MOSTRAR QUE PROGRAMAR RECURSIVAMENTE ES SENCILLO Y EFICIE
NTE
## SIEMPRE Y CUANDO SE REALICE CORRECTAMENTE.

### SEGUNDA PARTE ###-----------------------------------------------------------
----------

## LA SEGUNDA FUNCION RECURSIVA QUE VAMOS A VER ES LA FUNCION DE FIBONACCI
## VEREMOS VARIAS VERSIONES CON EL FIN DE COMPARAR Y ESCOGER UNA  VERSION EFICIENTE

## VERSION DOBLE RECURSIVA. ESTA VERSION ES EXACTAMENTE LA FORMULA ORIGNIAL MATEMATI
CA
fib<-function (n){
  if ((n==0)|(n==1)) return(n)
  else return(fib(n-1)+fib(n-2))
}
print(fib(5))
```

```
## [1] 5
```

```
# HAGAMOS UNAS PRUEBAS, ESCOGE COMO INPUT TU EDAD,
print(fib(16))
```

```
## [1] 987
```

```
#Y POR ULTIMO LA EDAD DE TU PADRE O TU MADRE, PERO ANTES Y DESPUES TOMAREMOS
# NOTA DE LA HORA DEL SISTEMA, ASI PODREMOS SABER CUANTO TIEMPO SE TARDA EN REALIZA
R EL CALCULO
# ¿QUE OCURRE? (pulsar ESC para parar una ejecucion)

t1<-Sys.time()
print(fib(35))
```

```
## [1] 9227465
```

```
t2<-Sys.time()
print(t2-t1)
```

```
## Time difference of 19.7669 secs
```

```
# CUANTO MAS GRANDE ES EL INPUT MUCHO MAS GRANDE ES EL TIEMPO DE EJECUCION, DE HECH
O,
# TIENE UN CRECIMIENTO EXPONENCIAL ESO SIGNIFICA QUE PARA CALCULAR FIBONACCI DE N
# NECESITAMOS REALIZAR UNAS 2^N OPERACIONES, O LO QUE ES LO MISMO, FIBONACCI DE 5 NE
CESITA 32 OPERACIONES, P
# PERO FIBONACCI DE 50 NECESITA 2 ELEVADO A 50 OPERACIONES, Â¿SABES CUANTO ES ESO?
# LO CALCULAMOS:
operaciones<- 2^50
# EXACTAMENTE 1123900000000000000 OPERACIONES, POR ESO HEMOS TENIDO QUE ESPERAR UN
RATITO


### TERCERA PARTE ###-----------------------------------------------------------------
----------
# PODEMOS IMPLEMENTAR FIBONACCI DE UNA FORMA MAS RAPIDA MEDIANTE EL SIGUIENTE ALGORI
TMO:
# ESTA VEZ ES UN ALGORITMO ITERATIVO:
fib_iter<-function(n){
  f1<-0;
  f2<-1;
  aux<-f1+f2
  while (n>1){f1<-f2;f2<-aux;aux<-f1+f2;n<-n-1}
  return(f2)
}
fib_iter(8)
```

```
## [1] 21
```

```
# ¿QUE TE PARECE ESTE CODIGO? Â¿SENCILLO O COMPLICADO?
# ¿ES MAS RAPIDO QUE EL ANTERIOR? CALCULA EL NUMERO DE FIBONACCI DE LA EDAD DE UNO D
E
# TUS ABUELOS:
fib_iter(87) # por ejemplo.
```

```
## [1] 6.798916e+17
```

```
# EN ESTA TERCERA PARTE VAMOS A IMPLEMENTAR FIBONACCI RECUSIVO REDUCIENDO
# LAS DOS LLAMADAS RECURSIVAS INICIALES A UNA SOLA. ESTA ES UNA TECNICA DE PROGRAMAC
ION
# QUE SE CONOCE COMO GENERALIZACION EL PROBLEMA O RECURSION FINAL.

fib_final<- function(n, f1, f2){
  if (n==1) return(f2)

  else return(fib_final(n-1,f2, f1+f2))

}
# LO PROBAMOS PARA UN PAR DE CASOS:
fib_final(5,0,1)
```

```
## [1] 5
```

```
# COPIAR ESTE CODIGO Y LO EJECUTAMOS CON LA SUMA DE LA EDAD DE NUESTROS ABUELOS
fib_final(187,0,1) # por ejemplo.
```

```
## [1] 5.385223e+38
```

```
# PODEMOS USAR SYS.TIME PARA SABER CUAL TARDA MAS:
t1<-Sys.time()
fib(32)
```

```
## [1] 2178309
```

```
t2<-Sys.time()
fib_iter(32)
```

```
## [1] 2178309
```

```
t3<-Sys.time()
fib_final(32,0,1)
```

```
## [1] 2178309
```

```
t4<-Sys.time()

t2-t1
```

```
## Time difference of 4.644287 secs
```

```
t3-t2
```

```
## Time difference of 0 secs
```

```
t4-t3
```

```
## Time difference of 0.01561713 secs
```

```
# TRAS EJECUTAR ESTE CODIGO, ¿PODEIS DECIR CUANTO TIEMPO TARDA CADA UNO DE LOS ALGOR
ITMOS?
```

Anexo C

Congreso internacional INTED 2019

CRITICAL THINKING IN SCHOLAR STUDENTS IN THE BIG DATA ERA

Carlos Armero[1], Victoria López[1], Matilde Santos[1]

[1]Computer Science Faculty, Dept. Computer Architecture and Automatic Control, University Complutense of Madrid (SPAIN)

Abstract

In this paper we study the perception and capabilities of the students from the critical thinking point of view. The work consists of two parts. First, we have synthesized the concerns about the evolution of education presented by the experts on this topic in some international forums. Secondly, we have carried out some preliminary experiments to evaluate the critical capacity of a group of students.

Keywords: Critical thinking, education, STEM, sciences, gender

1 INTRODUCTION

Critical thinking in young students is a concern in the educational field. Moreover, it must be one of the goals of science education [1]. There are several forums in which this concern has been highlighted. For example, in the 2nd Workshop on Data Science in Education, Health and Society held in Madrid in April 2018 or in the second edition of the EDISON European Project for Building the Data Science Profession [2], which also was held in Madrid in March 2017. In these congresses, representatives of the United Kingdom, Ireland, Germany, France, Spain, and a long etcetera were agreed on the importance of improving the critical thinking in the students. Here we also want to contribute to these reflections on how student manage data as part of the Social Big Data project, which goal is to analyze the social change based on Big Data and in which several universities are working together. The new concept of Big Data in recent ICT (information and communication technology) domain extends the promising research direction on online learning and Big Data integration through promising content that can be tailored for each student based on the context and Internet behaviour of users in online learning [3].

In the last years, education was mainly focused on the ability to retain and repeat information. Today, learning is moving toward developing skills that will prepare young people to navigate the real world outside of and after school. In addition, the need for qualified candidates in STEM (Science, Technology, Engineering and Mathematics) industries is growing everyday as the world becomes more digitally reliant. Many organizations have noticed this important need and are working to encourage interest in a STEM education among today's youth. Studies indicate this could have a large impact on the future of business [4].

Nowadays, students are under a steady barrage of information, particularly from online sources, friends, parents and media, and it quickly becomes evident that they need to learn how to evaluate what they see and hear every day so they can identify false ideas and look beyond superficial appearances. That is our main concern, and in this work we want to study the perception and capabilities of the students from this point of view.

This work consists of two parts. First of all, we have synthesized the concerns about the evolution of education presented by the experts in the forums indicated above. Secondly, we have carried out a first experiment with which we try to evaluate the critical capacity of a group of students.

The experiment consists in teaching the students different strategies to solve a problem. Students should note the difficulty and enthusiasm they find in the application of the different strategies.

We find that the use of computational tools that are so close to students' background help motivate them [5], [6]. Indeed, more and more aspects of our everyday lives are being mediated, augmented, produced and regulated by software-enabled technologies. Software is fundamentally composed of algorithms [7]. That is why we try to develop and extend the critical thinking about algorithm and computer programs or applications that are usually run carelessly.

For the application of the experiment mathematical problems of simple approach but of complex resolution have been used. The different strategies are implemented in a programming language (R)

with methodologies that range from brute force to optimal final recursion. In the digital age individuals are expected to have the computational thinking skill, and at what degree they develop these skills is important [8], [9].

The structure of the paper is as follows. Section 2 is devoted to present some of the main conclusions drawn in international events on critical thinking. In section 3, experiments carried out with students show how the development of critical thinking is a big challenge and how they have shown some skills. The paper ends with the conclusions and future work.

2 REFLEXIONS ON CRITICAL THINKING AND BIG DATA

There are several forums where educators who want to incorporate critical thinking for students into curriculum development can participate. We attended EDISON (2017), and from the different workshops that had place we collected the following reflections to take into account:

1. Data Science is an emerging field of science, which requires a multi-disciplinary approach and should be built with the strong link to Big Data and data driven technologies. But so far, there are gaps in knowledge and competences of the future Data Scientist graduates for their smooth integration in the real working environment (both in industry and academia). Computer and math students (STEM) will not be able to meet the demand for data science professionals;

2. For business related occupations there are important competences that are essential for the Data Scientist students in order to be successful at work. They have to be able to discover new relations and provide actionable insight into available data, and have ability to formulate good research questions, hypothesis and evaluate them based on collected data. Nevertheless, students hardly understand the problems they are asked to solve because the proposed cases are usually far from the training they had;

3. Employers do not find the expert they are looking for. It is surprising to see the amount of requirements and skills that are asked in some jobs for data scientist that are not included, at least at European level, in degree or master's programs or specific training;

4. A society capable of working and understanding the sources of information with which it coexists is required. Therefore, data science and critical thinking should begin to be known from the school stages, as well as programming.

5. The mission of the university and the academy is to create critical thinking.

Moreover, for the correct functioning and development of artificial intelligence, the guarantee of an ethical, moral and critical level is paramount.

The era of big (and open) data has provided many advantages: mobility, access to information and quality of life. Advances in education and society are clear examples. Nevertheless, it is very common for people (especially young people) to accept without reading the conditions of use of the applications they install on their devices. It is also common for people to ignore the consequences of the use of applications, especially the consequences that may appear over time.

The algorithms that provide the mentioned advantages are not completely open but people trust both the algorithms and the data that feeds them. Both, data and algorithms, can be manipulated, present errors or cause automations unsuspected by the user. Even more, most of the people are not able to understand the code of an algorithm.

In this sense, we agree with Quinn statement, “theory that is rooted in practice is a better thing, even as theory. But good practice without theory is blind and good theory without practice is sterile” [10]. So, we have to introduce new words and concept to help students to understand their thinking process.

Because of this, critical thinking is a value that must be educated from an early age, especially for the future STEM employees.

3 EXPERIMENTS

In this second part of our work we present an experiment where we have tried to measure the capacity of critical thinking of some students through a practice with mathematical algorithms. The final goal is

to draw some conclusions regarding possible differences between age or gender when solving computational problems that require creativeness and critical thinking.

The experiment took place during “Science Week”, which is an event organised by all the Universities in Spain. In our case, we applied it at the University Complutense of Madrid. We gave an introduction to the world of algorithms with some interesting mathematical exercises for students. These algorithms had been proposed by distinguished mathematicians such as Fibonacci, Pitagoras, Euclides and Gauss. Besides, we try to relate them. Our goal was to attract students’ attention explaining curious aspects and then teach them different ways to think and solve problems.

The experiment is divided into four parts. Each part is formed by different algorithms that are presented from lower to higher difficulty. But not only the analytical complexity increases but for each part we present some different solutions to the same problem. That is, at the beginning we present some easy to understand but not efficient algorithms and more efficient solutions at the end. This way, the students grasp the benefits of thinking deeper to get a more efficient solution. The purpose is to show students that the easiest solutions are not always the best ones, and also to show them that the fact that an algorithm that works may be not good enough. These ideas can help them to improve critical thinking and change their way of thinking regarding a computational solution.

There were 149 attendees. The 149 students come from 3 different schools of the Community of Madrid, Spain. Their age range was between 14 and 18 years old, precisely the pre-university courses. There were 57.7% female participation and 42.3% male participation. Out of these, 92.6% considered that the activity was satisfactory and more than 85% successfully completed all the exercises proposed. Therefore, we consider that the students have participated collaboratively and that they represent an acceptable sample for drawing conclusions regarding the understanding and scope of the proposed objectives.

Different task oriented to understanding mathematical formulas have been carried out according to the educational level, in three sessions: 3ESO, 4ESO and Bach (13-14, 15-16, 17-18 years respectively). The different strategies are implemented in the form of 4 exercises of ascending difficulty. We have measured ratios of goals reached within the different strategies for solving mathematical problems with the help of computer programming, and we have also studied some differences between girls and boys, such as preferences or critical thinking, to analyse if gender gives us relevant information.

First of all, we can see in Figure 1 that most of the students considered all of the exercises easy to understand. As expected, the students find more difficult the last exercises. The shortage of difficulty observed may be due to the teacher's explanations in the classroom. It must be considered that only with written explanations or through the new video tutorial formats, the difficulty can be substantially increased. In any case, the data in Figure 1 are representative for the differences observed between the different exercises.

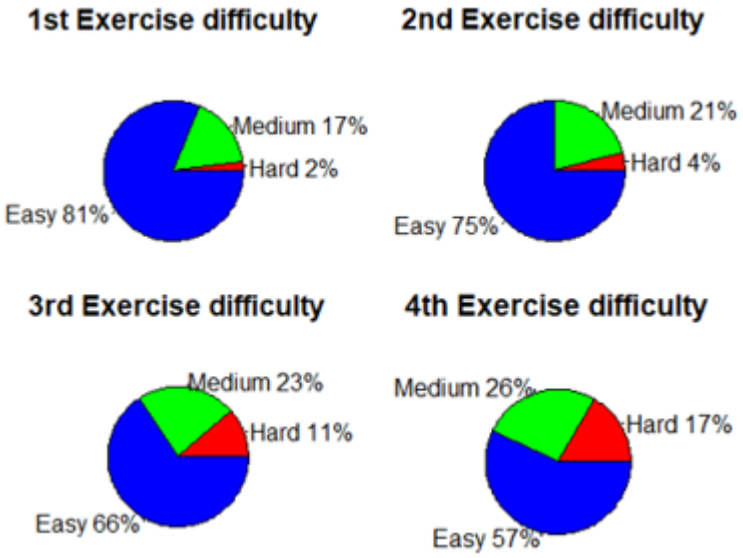


Figure 1: Difficulty of all exercises.

General difficulty filtered by age is represented in Figure 2 which shows a clear difference between high school students and the rest. There is a strong inverse relationship between the level of maturity of the students and the level of difficulty found in the tests. Filtering by gender, we find no big differences between boys and girls, as we see in Figure 3.

Figure 2: General difficulty filtered by age

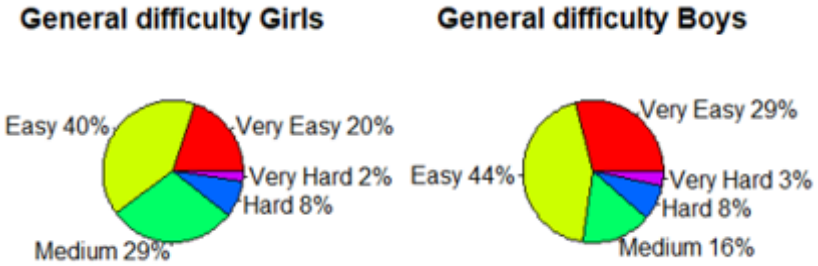


Figure 3: General difficulty filtered by gender

In Figure 4 we can see the differences between the difficulty as perceived by the students (average) and a measure of the real difficulty we have assigned to each part. It is possible to see that High School students are more realistic than younger students, so age could be a significant factor to measure the perception of reality and to get a higher critical thinking.

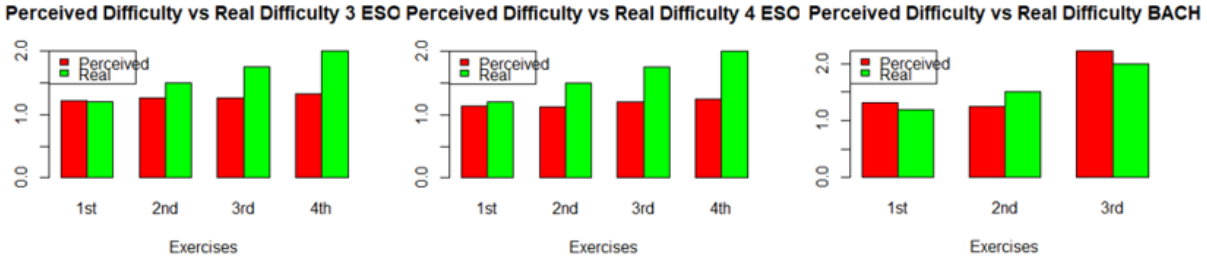


Figure 4: Perceived difficulty vs Real difficulty filtered by age

Most of the students think that they have achieved the goals in almost all the exercises. In this case, we cannot find significant differences between age or gender. We can see the averages of achieved goals filtered by age in Figure 5, and averages filtered by gender in Figure 6.

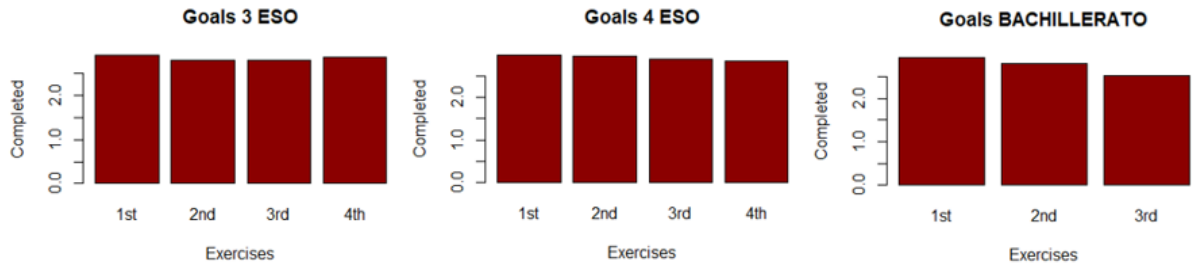


Figure 5: Achieved goals by exercise (filtered by age).

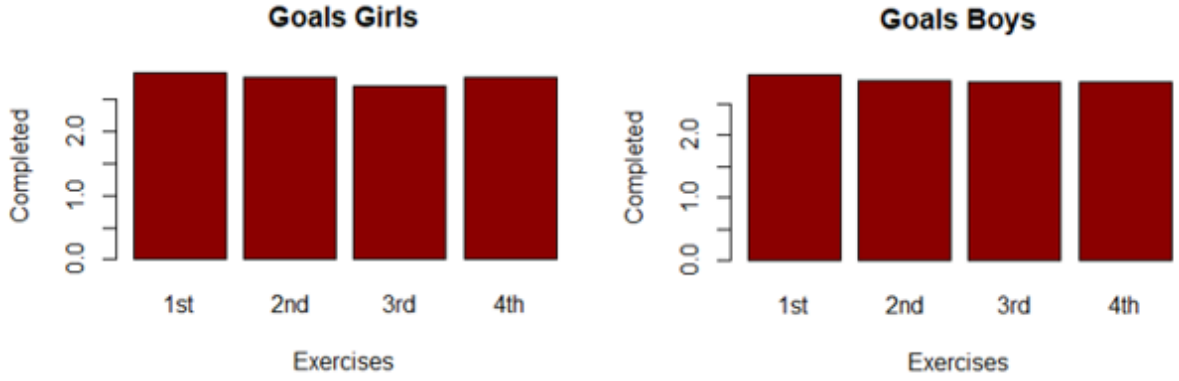


Figure 6: Achieved goals by exercise (filtered by gender).

As we can see in Figure 7, 81% of boys and 67% of girls have enjoyed the technological experience, so the gender difference is not very relevant. Girls think that imagination is important in algorithms more than boys, who believe that programming is just to deal with some defined steps (Figure 8).

Figure 7: Have students enjoyed the experience?

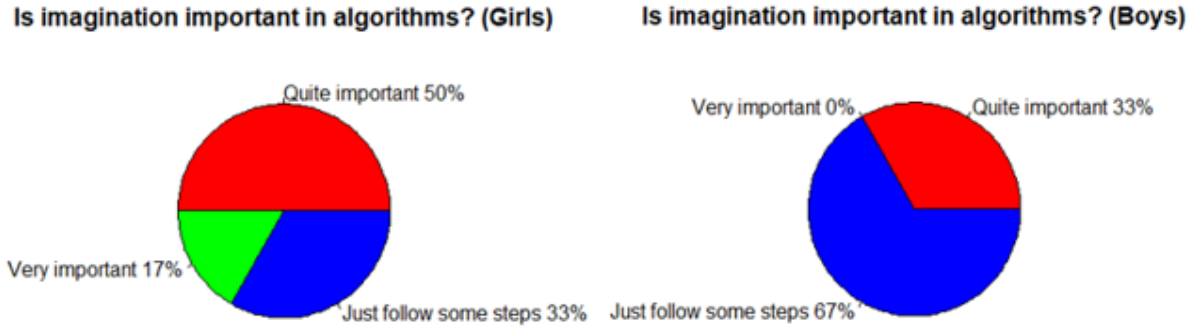


Figure 8: Is imagination important?

In Figure 9 we can see that both genders like programming in a similar way, so that there is not a prevailing one over the other. We can say that professional orientation in computer science and software skills do not depend on gender.

Figure 9: Do you like programming?

The results of the study show no differences about the capability of girls against boys in critical thinking. It is also proved that at scholar age the students are well skilled to develop a correct critical thinking.

4 CONCLUSIONS

Some of the conclusions drawn from the results are that the study shows no difference about the capability of girls against boys in critical thinking. Indeed, regarding programming in particular and computer science professional orientation in general, it does not depend on gender.

Also it is proved that at scholar age the students are well skilled to develop a correct critical thinking and Schools and University should foster it. We think that in the STEM (Science, Technology, Engineering and Mathematics) education field, this skill has to be included.

ACKNOWLEDGEMENTS

This paper is supported thanks to the project Social Big Data – CM (http://socialbigdata.transyt-projects.com/).

REFERENCES

[1] S. Bailin, "Critical thinking and science education", *Science & Education*, vol. 11, no. 4, pp. 361-375, 2002.

[2] EDISON (2017). http://edison-project.eu/edison/edison-data-science-framework-edsf (last access Jan2019).

[3] M. Huda, A. Maseleno, P. Atmotiyoso, M. Siregar, R. Ahmad, K. Jasmi, N. Muhama,"Big Data emerging technology: insights into innovative environment for online learning resources". *International Journal of Emerging Technologies in Learning (iJET)*, vol. 13, no. 1, pp. 23-36, 2018.

[4] T. R. Kelley and J. G. Knowles, "A conceptual framework for integrated STEM education", *International Journal of STEM Education*, pp.3:11, 2016.

[5] M. Santos and G. Farias Castro. "Laboratorios virtuales y remotos de procesamiento de señales", *Revista Iberoamericana de Automática e Informática Industrial*, vol. 7, no. 1, pp. 91-100, 2010.

[6] J.M. Cañas, A. Martin, E. Perdices, F. Rivas, R. Calvo, R. "Academic framework for teaching robot programming at university", *Revista Iberoamericana de Automática e Informática Industrial*, vol. 15, no. 4, pp. 404-415, 2018.

[7] R. Kitchin. "Thinking critically about and researching algorithms", *Information, Communication & Society* vol. 20, no. 1, pp. 14-29, 2017.

[8] O. Korkmaz, C. Recep, and M.Y. Özden, "A validity and reliability study of the Computational Thinking Scales (CTS)", *Computers in Human Behavior* vol. 72, pp. 558-569, 2017

[9] J.A. Martín-H, M. Santos, V. López, A. Vargas, "A software tool for the automatic tracking and segmentation of student's profiles", *6th Int. Technology, Education and Development Conference, INTED*, pp. 1511-1517, 2012.

[10] V. Quinn, "Critical thinking in young minds", *Routledge*, 2018.