CURSO BÁSICO DE PROGRAMACIÓN EN MATLAB®

Antonio Souto Iglesias
José Luis Bravo Trinidad
Alicia Cantón Pire
Leo Miguel González Gutiérrez

SEGUNDA EDICIÓN

Editorial Tébar

)00021651

034 - PG - 2016 - 513

MFN: 0000021651
005.1
.S68
G...
2014

UNIVERSIDAD TECNICA DEL NORTE
BIBLIOTECA
Via de adquisición: Compra
Documento No. 034 PG 2016 / 513
Fecha: 25-07-2016
Valor unitario: 30,00
Codigo de Barras: 055265
Anexos:

Curso básico de programación en MATLAB®

Antonio Souto Iglesias
José Luis Bravo Trinidad
Alicia Cantón Pire
Leo Miguel González Gutiérrez

Curso básico de programación en MATLAB®

Antonio Souto Iglesias

José Luis Bravo Trinidad

Alicia Cantón Pire

Leo Miguel González Gutiérrez

Datos de catalogación bibliográfica:
Curso básico de programación en MATLAB®
2ª edición
Antonio Souto Iglesias, José Luis Bravo Trinidad, Alicia Cantón Pire, Leo Miguel González Gutiérrez

EDITORIAL TÉBAR, S.L.
Madrid, año 2014
ISBN: 978-84-7360-520-5
Materias: 51, Matemáticas
Formato: 29,5 × 21 cm
Páginas: 230

MATLAB, Copyright 1984-2012, The MathWorks, Inc.

Todos los derechos reservados. Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución, comunicación pública y transformación de esta obra sin contar con la autorización expresa de Editorial Tébar. La infracción de estos derechos puede ser constitutiva de delito contra la propiedad intelectual (arts. 270 y siguientes del Código Penal).

Curso básico de programación en MATLAB®
2ª edición

Antonio Souto Iglesias, José Luis Bravo Trinidad, Alicia Cantón Pire, Leo Miguel González Gutiérrez
Editorial Tébar, S.L.
C/ del Toboso, 117
28019 Madrid (España)
Tel.: 91 550 02 60
Fax: 91 550 02 61
pedidos@editorialtebar.com
www.editorialtebar.com

ISBN: 978-84-7360-520-5
Depósito legal: M-706-2014
Foto de portada: Horacio Diez, horaciodiez.com
Imprime: Servicepoint

A mi padre, a quien tanto le hubiese
gustado hablar conmigo de estas
y de otras muchas cosas.

A. Souto Iglesias

A Dan, por el constante estímulo intelectual
que me proporciona su compañía.
A Chou, motor propulsor de este proyecto.

A. Cantón Pire

Índice general

Notación y abreviaturas 13

Prólogo 15

Introducción 21

1. MATLAB como calculadora 25

1.1. Introducción 25
1.2. Conceptos básicos 25
1.3. Manejo de vectores 30
1.4. Introducción al tratamiento de matrices 33
1.5. Resolución de sistemas lineales 37
1.6. Vectorización de operaciones 39
1.7. Creación de gráficas 42
1.8. Conjuntos de órdenes 43
1.9. MATLAB y números complejos 47
1.10. Matemáticas simbólicas con MATLAB 47

2. Funciones y Condicionales 49

2.1. General 49
2.2. MATLAB como un lenguaje de programación 49
2.3. Funciones y variables 50
2.4. Funciones con varios argumentos de entrada 54
2.5. Estructura de control condicional *if* 57
2.6. Estructura de control condicional *if-else* 60
2.7. Función que llama a otra función 61
2.8. Condicionales anidados 63
2.9. Variante *elseif* en el condicional 66
2.10. Depuración de códigos: *debugger* 68
2.11. Operadores lógicos 69
2.12. Operadores de comparación: *¿son iguales?* 72
2.13. Variables enteras y reales como argumentos 74

UNIVERSIDAD TÉCNICA DEL NORTE BIBLIOTECA

2.14. Variables contador y sumador 76
2.15. Función parte entera 78

3. Bucles **79**
3.1. General 79
3.2. Bucles 79
3.3. Bucles y relaciones de recurrencia 86
3.4. Bucles y condicionales 89
3.5. Bucles inconclusos: la sentencia *break* 92
3.6. Bucles en los que la condición no se refiere a un índice 95

4. Vectores **101**
4.1. General 101
4.2. Vectores como argumentos de funciones 101
4.3. Funciones que llaman a funciones con argumentos vectores 104
4.4. Cálculo de extremos 106
4.5. Cálculo de extremos utilizando un vector auxiliar 108
4.6. Cálculo de posición de extremos 110
4.7. Vectores y bucles anidados 113
4.8. Función que devuelve un vector definido a partir de escalares 116
4.9. Funciones que reciben y devuelven vectores 118
4.10. Construcción de vectores 122
4.11. Funciones con salidas múltiples 124
4.12. Vectores y polinomios 128
4.13. Evaluación de un polinomio 130

5. Entrada y salida con formato **135**
5.1. General 135
5.2. Entrada y salida con formato 136
5.3. Lectura y escritura de vectores 139
5.4. Ficheros 142
5.5. Diseño básico de GUIs 148
5.6. Proyectos de programación 153

6. Matrices **163**
6.1. General 163
6.2. Bucles for 163
6.3. Matrices como argumentos de funciones 165
6.4. Submatrices 171
6.5. Resolución de un sistema lineal mediante eliminación Gaussiana 173
6.6. Proyectos de programación 179

7. **Algoritmos** **185**
7.1. General 185
7.2. Algoritmos de búsqueda 185
7.3. Algoritmos de ordenación 193
7.4. Algoritmos geométricos 204
7.5. Proyectos de programación 208

Epílogo **215**

Referencias **217**

Apéndice: selección de ejercicios resueltos **219**

Índice alfabético **229**

Notación y abreviaturas

DNI:	para referirse al número del Documento Nacional de Identidad.
ECTS:	sistema de créditos europeos.
TIC:	tecnologías de la información y la comunicación.

Además de las abreviaturas mencionadas más arriba es necesario comentar los siguientes aspectos relativos a convenciones utilizadas:

1. Nos referiremos indistintamente a los programas como programas o como *scripts* ya que en MATLAB ambos conceptos son equivalentes.

2. Dado que seguiremos la formalidad de ubicar cada función en archivo cuyo nombre es el de la propia función más la extensión `.m` de MATLAB, a menudo abusaremos de la notación y nos referiremos indistintamente a la función y al archivo que la contiene. Algo similar pasa con los "scripts" y archivos que los contienen.

3. Utilizaremos la `fuente` para referirnos a nombres de funciones, *scripts*, comandos, para presentar los códigos y para referirnos en el texto a variables. Utilizaremos la $fuente$ para referirnos en el texto a algunas variables escalares específicas y a matrices. Utilizaremos la $\mathbf{fuente}$ para referirnos a vectores.

4. No se acentuará ninguna palabra en los comentarios incorporados a los propios códigos documentados en el texto.

5. Para simplificar la notación, se usará siempre como separador decimal el punto, tanto en el texto como en los códigos, dado que en estos es el obligatorio. Usaremos ocasionalmente la notación científica (por ejemplo, nos referiremos a 0.0005 como 5e-4).

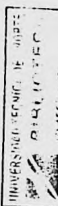
6. Cuando en un ejercicio se indica que se supone por ejemplo que los valores de entrada son diferentes entre sí, la función se codificará para que resuelva el problema planteado asumiendo que esa suposición es cierta y no necesita por tanto ser comprobada.

7. Se agradece envíen erratas detectadas a antonio.souto@upm.es.

Prólogo

Al escribir este libro hemos pretendido ofrecer una herramienta sobre la que articular un curso de introducción a la programación de ordenadores para titulaciones fuera del ámbito informático.

Para los profesionales cuyas titulaciones están directamente relacionadas con las tecnologías de la información y la comunicación (TIC), como los titulados en informática, ingeniería de telecomunicación, etc., la programación es una herramienta esencial de su trabajo y a ella dedican mucho esfuerzo durante los estudios. Sin embargo, para otros profesionales, la programación es un medio de solucionar problemas puntuales[1] como codificar alguna pequeña aplicación para resolver un problema específico, extender las funcionalidades de algún programa, subcontratar a alguna empresa que haga alguna de estas tareas, etc. También es importante aprender a programar para poder desarrollar estrategias para análisis de datos con otras herramientas informáticas.



Con esos otros profesionales nos referimos en particular a los titulados de ingeniería industrial, matemáticas, ciencias físicas y otras titulaciones en el ámbito de la ingeniería con menos alumnado como ingeniería civil, química, minas, energía, etc.... De hecho, el libro nace a partir de la asignatura de introducción a la programación del que los autores han sido o son responsables, en la titulación de ingeniería naval y oceánica de la Universidad Politécnica de Madrid.

Más recientemente, las TIC han entrado de lleno en titulaciones como ingeniería forestal, agrónoma, ciencias económicas, empresariales, biología, etc., incorporando transversalmente formación en aspectos específicos de programación aplicada a las mismas. La importancia global de estas titulaciones y los sectores económicos para los que se forman sus estudiantes es enorme dentro de nuestra economía. Para todos estos titulados es una ventaja competitiva el tener una formación básica en programación de ordenadores, y nos parece adecuado que esa empiece con un curso con una filosofía en la línea del aquí descrito y continúe de modo natural aprendiendo a escribir funciones para hojas de cálculo o bases de datos, con las diferentes posibilidades disponibles en el mercado.

Además, los cursos de programación brindan la oportunidad de un acercamiento directo a la computadora y a su funcionamiento, y si se hacen de un modo inteligente, al estudiante le

[1]Ello no es óbice para que haya un porcentaje significativo de titulados no informáticos que se dediquen profesionalmente a las TIC y en particular dentro de estas a la programación de aplicaciones.

quedan intuiciones fuertes y coherentes de lo que hay en las tripas de la máquina, y sobre todo de lo que un ordenador *puede y no puede hacer*. En otras asignaturas de estas titulaciones se manejan programas de aplicación, cuyo conocimiento y manejo con soltura, también son vitales -una hoja de cálculo, un procesador de textos, un programa de CAD, el sistema operativo, un gestor de proyectos, etc.- pero la programación contempla otros matices. Para nuestros titulados, la posibilidad de abstraer los elementos esenciales de un problema, analizando los escasos elementos de los que dispone para solucionarlo y establecer una metodología, estrategia y procedimiento para abordarlo, son elementos básicos de su trabajo. Precisamente en esto último consiste una forma de ver la programación de ordenadores. De hecho, como dice Soloway[2], en la vida diaria, todo el mundo está construyendo mecanismos y explicaciones para solucionar problemas y programar es llevar esto a su extremo de abstracción.

Normalmente, el contacto de estos estudiantes con la programación se limita a una asignatura en el primer curso. Por ello es esencial que el tiempo dedicado a programar se aproveche al máximo, de modo que el alumno tenga unos fundamentos claros de la programación. Por esto hemos escogido un lenguaje de programación de alto nivel, que permite que el esfuerzo se centre en la programación y no en la sintaxis. Con todo esto en mente, los objetivos concretos de este libro son los enumerados a continuación.

1. El libro esta concebido para que se pueda articular en torno a él un curso de introducción a la programación para titulaciones no informáticas, utilizando el lenguaje de comandos de MATLAB como lenguaje de referencia. Se pretende de este modo que, al final del curso, el estudiante haya asimilado los conceptos básicos de la programación estructurada y que además se sienta cómodo y seguro dentro del entorno MATLAB; esto le será útil en otros contextos, dado el amplio ámbito de aplicación del programa. La elección de MATLAB, aunque se justificará de modo detallado en la introducción a la primera parte del libro, se fundamenta en ese amplio ámbito de aplicación del programa y en su facilidad de uso.

2. Hemos pretendido que el libro proporcione material docente tanto teórico como práctico para que los estudiantes puedan seguir de modo efectivo el curso.

3. Pretendemos que sea interesante sobre todo para los estudiantes de titulaciones fuera del ámbito informático y para los colegas profesores de este tipo de materias. Para ello, los elementos que conforman la programación estructurada (estructuras de control, funciones, entrada y salida con formato, vectores y matrices, algoritmos, etc.) se introducen de modo progresivo y se vuelve sobre ellos continuamente. Al final, el alumno conocerá todos esos elementos y podrá usarlos con autonomía.

---

[2]Soloway, E., Learning to program = learning to construct mechanisms and explanations. Communications of the ACM, Volume 29, Number 9, September, 1986.

Junto a los objetivos que perseguimos, es importante resaltar lo que no hemos pretendido con el libro:

1. Aunque el estudiante se sienta más cómodo al final del curso en el entorno MATLAB, no hemos pretendido enseñar al estudiante a calcular y resolver problemas complejos utilizando las posibilidades de MATLAB. Nos hemos centrado en una serie de comandos e ideas canónicas como base para construir lo que es un curso de introducción a la programación.

2. Tampoco hemos pretendido escribir un curso de programación aprovechando las posibilidades específicas de MATLAB en ese sentido, dado que ello le quitaría generalidad. Así, hemos evitado el uso de operaciones explícitas entre vectores y matrices y la vectorización de operaciones en la parte del libro donde se explican los fundamentos de la programación estructurada (capítulos 2 a 4).

3. Finalmente, y dado que este libro es un curso de Introducción a la programación para no informáticos, nos hemos puesto límites en los conceptos a tratar. Creemos sin embargo que los conceptos tratados lo son en profundidad, con el objetivo de que las competencias adquiridas por los estudiantes les pudieran servir como base para retos más ambiciosos en el futuro.

Esperamos que el libro sea interesante sobre todo para los estudiantes de las titulaciones citadas al principio, para los colegas profesores de este tipo de materias, y para aquellos que se enfrentarán en breve a la redacción de nuevos planes de estudio incorporando a los mismos formación obligatoria en programación. Se ha tratado de detallar todo el planteamiento del curso, bajo el cual subyace la cita de Wittgenstein que aparece un poco más adelante: se introducen elementos de modo paulatino y se vuelve sobre ellos continuamente. Al final, el alumno conocerá todos esos elementos y podrá usarlos con autonomía.

El libro ha sido escrito en LaTeX, procesador de textos para Matemáticas desarrollado a partir de TeX, el cual debemos a Donald Knuth[3], una de las figuras más importantes en el mundo de la programación de ordenadores. Escribir en LaTeX es metodológicamente completamente

[3]Donald Knuth, matemático y profesor de Informática en la Universidad de Stanford, nació en Estados Unidos en 1938. Mientras realizaba su tesis doctoral en Matemáticas, la editorial Addison-Wesley le propuso escribir un libro de texto sobre compiladores que acabó siendo su obra más importante "El arte de programar ordenadores", compuesta por múltiples volúmenes, y sobre la cual sigue trabajando en la actualidad. Corrigiendo la re-edición de uno de los volúmenes, Knuth se dió cuenta que la calidad de la impresión y de los caracteres matemáticos dejaba mucho que desear. Decidió desarrollar un lenguaje específico para escribir textos y artículos matemáticos y científicos. Así nació TeX, con la intención de que pudiera ser usado directamente por los autores de los artículos, que fuera de acceso gratuito y que pudiera estar soportado en cualquier sistema operativo. Posteriormente han surgido extensiones de TeX, como LaTeX (que es una colección de comandos definidos de uso más simple que el TeX) y que es actualmente la principal herramienta para escribir en Matemáticas y otras disciplinas científicas. Knuth es un personaje muy interesante, que ha ido contra-corriente no solo cuando empezó con TeX. Por ejemplo, carece de correo electrónico y argumenta su postura con cierta lucidez, como se puede comprobar en este enlace (http://www-cs-faculty.stanford.edu/ knuth/email.html).

diferente a hacerlo en MS-WORD por ejemplo, dado que recuerda bastante a generar un programa que después hay que compilar para tener el PDF del libro. Sobre la idea inicial de Knuth, TEX ha evolucionado mucho en los últimos 30 años a partir del trabajo cooperativo de muchas personas en el mundo, desde la esfera del software libre, y se ha convertido en una aplicación fundamental para la generación de documentos científicos de alta calidad formal.

Para terminar, retomamos la maravillosa cita de Stephen Ball que aparece un poco más adelante y que junto a la de Wittgenstein y la de Vic Muniz que también encontramos ahí, inspiran todo este trabajo. En un momento que el que se coincide en resaltar la escasa motivación de los estudiantes, enriquecer la evaluación les proporciona mecanismos de enganche continuo con el curso, y les permite potenciar muchas facetas tomando como punto de partida aquellas en las que por diferentes razones ya son más válidos. Con este libro hemos pretendido proporcionar materiales que faciliten esa tarea y así los hemos utilizado en nuestras asignaturas[4]. Para nosotros, como docentes, mantener a los estudiantes en el sistema, comprometidos y progresando, es un motivo de satisfacción profesional y personal.

Antonio Souto Iglesias
Seixo y Madrid, 2012

[4]Souto-Iglesias, A., Bravo-Trinidad, J. L. (2008). Implementación ECTS en un curso de programación en Ingeniería. Revista de Educación, (346), 487-511.

....para mí una buena escuela sería una escuela distinta, una escuela que tuviese un principio según el cual todas sus normas estuviesen enfocadas a mantener a tantos estudiantes como sea posible durante el mayor tiempo dentro del sistema. Así, todo tendría que estar dirigido a hacer que los estudiantes participasen, que se sintiesen identificados con la escuela, que tuviesen la sensación de estar haciendo las cosas bien. Para mí una buena escuela es una escuela que mantiene a todos los alumnos trabajando, comprometidos y con la sensación de que no van a fracasar.

Stephen Ball.

In teaching you philosophy I'm like a guide showing you how to find your way round London. I have to take you through the city from north to south, from east to west, from Euston to the embankment and from Piccadilly to the Marble Arch. After I have taken you many journeys through the city, in all sorts of directions, we shall have passed through any given street a number of times - each time traversing the street as part of a different journey. At the end of this you will know London; you will be able to find your way about like a Londoner. Of course, a good guide will take you through the more important streets more often than he takes you down side streets; a bad guide will do the opposite. In philosophy I'm a rather bad guide.

L. Wittgenstein.

O cerebro não colhe ideias no cantaro do ocio. E sobretudo pela interação com o material, pelo trabalho, pelo esforço e, em ultima instância, pelo fracasso, que nos nutrimos nosso banco de ideias.

Vic Muniz.

Introducción

De modo resumido, podemos decir que programar es "enseñarle" a un ordenador cómo se resuelve un determinado problema. Para poder enseñar a alguien, son necesarias al menos dos cosas: tener un lenguaje común y conocer bien lo que se quiere enseñar. El lenguaje que "hablan" los ordenadores es muy simple, por lo que el proceso de aprenderlo será rápido. El principal problema es descomponer ideas y problemas complejos en otros más simples para que podamos desarrollar un algoritmo que permita resolverlos, y después programar esos algoritmos. Para facilitar el aprendizaje de este proceso hemos descompuesto el libro en una serie de capítulos con unidad temática y organizados con una estructura *concepto*$_1$-*uso*$_{1,1}$-*ejemplo*$_{1,1}$-*ejercicios propuestos, concepto*$_1$-*uso*$_{1,2}$-*ejemplo*$_{1,2}$-*ejercicios propuestos, ..., concepto*$_2$-*uso*$_{2,1}$-*ejemplo*$_{2,1}$-*ejercicios propuestos, etc.* Al final del libro se incluye una selección de ejercicios resueltos de entre los propuestos en las diferentes secciones.

Normalmente se trabajan uno o dos conceptos de programación en cada capítulo (funciones y condicionales, por ejemplo), los cuales se introducen primero formalmente y después mediante ejemplos sobre los que se proponen una serie de ejercicios que nos permitan asimilar y madurar las ideas explicadas. Por cada concepto introducido se trabajan cuatro o cinco ejemplos. Se pretende que en cada uno de estos ejemplos aparezca un uso habitual del concepto introducido. Se trata de asimilar bloques con un sentido concreto, construidos con los elementos básicos de programación.

A los códigos utilizados como ejemplo en el libro así como a las soluciones de una selección de ejercicios propuestos se puede acceder en la dirección:

http://canal.etsin.upm.es/matlab/

A menudo, en los cursos de programación, se introducen sólo los elementos de programación y se supone que el alumno aprenderá a integrarlos por sí mismo. Sin embargo, muchas aplicaciones de esos elementos son estándares en la programación (por ejemplo, el uso de contadores, un bucle "while" y un condicional para seleccionar determinados elementos de un vector, crear un vector mediante un contador, etc.) y una vez nos hemos familiarizado con ese uso conseguimos una mayor agilidad a la hora de programar.

El lenguaje elegido para la explicación de los elementos de la programación y para la implementación de ejemplos y ejercicios es el lenguaje de comandos de MATLAB (o su versión libre OCTAVE). MATLAB es un lenguaje de comandos desarrollado inicialmente en la década de

los 70 por Cleve Moler[5]. La elección de MATLAB se debió a varios motivos:

1. MATLAB es un entorno de cálculo que los estudiantes usarán a lo largo de la carrera y probablemente después en su vida profesional ya que dispone de herramientas específicas ("toolboxes") para muchos ámbitos. Aunque las competencias de manejo asociadas a esas herramientas específicas no se trabajan en este libro, el que el estudiante se sienta al final cómodo con el entorno MATLAB le permitirá si es necesario asimilar su funcionamiento con mucha mayor facilidad que si empezase de cero con el programa.

2. MATLAB es un lenguaje completo; tiene todos los elementos de un lenguaje de programación, con una sintaxis similar al C pero con la simplicidad del BASIC. Comprobamos en cursos anteriores que al utilizar un lenguaje como C, los alumnos dedicaban la mayor parte del tiempo a la corrección de errores de sintaxis y a la declaración de variables, reserva de memoria, etc., teniendo poco tiempo para comprender el funcionamiento de las estructuras de datos o de control del flujo del programa. En este sentido, el lenguaje MATLAB se acerca al pseudocódigo usado en algunos cursos de programación, pero con la ventaja de poder realmente ejecutar los códigos creados. La simplicidad de MATLAB a estos efectos es a veces causa de falta de rigor en la forma de abordar la programación. Así el hecho de que no haya tipado explícito de variables pudiendo la misma variable ser una matriz en una línea del código y un escalar un poco más abajo, o de MATLAB se ocupe del la reserva dinámica de memoria de modo automático, nos alejan de un lenguaje más potente como C. Sin embargo y como comentábamos más arriba, eso permite centrarse en el manejo de estructuras de control para resolver problemas y desarrollar estrategias, creemos que esencial al principio.

3. MATLAB es un lenguaje interpretable: MATLAB traduce durante la ejecución las diferentes sentencias al lenguaje primario y básico de la máquina. Se paga el precio de necesitar MATLAB para ejecutar nuestros códigos pero se recibe la recompensa de no tener que compilar y enlazar nuestros códigos para después ejecutarlos.

---

[5] Cleve Moler nació en Estados Unidos en 1939. Estudió matemáticas en Caltech ("California Institute of Technology") y leyó su tesis doctoral en Stanford en Análisis Numérico. Junto con otros autores creó LINPACK y EISPACK dos librerías de subrutinas de computación numérica para FORTRAN.

A finales de los 70, dando clases en la universidad de Nuevo México en análisis numérico y teoría de matrices, Moler quiso que sus alumnos utilizaran esas librerías sin necesidad de escribir complejos programas en Fortran, así que siguiendo un manual de Wirth, creó su propio lenguaje de programación: MATLAB ("Laboratorio de Matrices"). En esta primera versión de MATLAB el único tipo de dato eran matrices y solamente había 80 funciones incorporadas. No existían los "M-files", y si se quería añadir una función había que modificar el código fuente en Fortran y recompilar el programa completo. Aún así esta versión tenía la cualidad de que corría en todos los ordenadores que existían en la época. A comienzo de los 80 Moler conoció a Jack Little, un ingeniero del MIT ("Massachusetts Institute of Technology") que sabía cómo usar matrices en Teoría del Control, aplicación que incorporaron al programa. Juntos fundaron la compañía MathWork que dio salida comercial a MATLAB (su principal producto). Hoy en día MATLAB es usado por millones de ingenieros y científicos de todo el mundo.

4. MATLAB proporciona una interfaz que permite probar las funciones directamente sin necesidad de llamarlas desde un programa principal. Esto permite comprobar su funcionamiento de un modo sencillo e inmediato, y como comentamos más abajo, ha permitido una estructura del curso creemos que muy interesante para un curso de introducción a la programación para no informáticos.

5. Todo ello hace que sea muy sencillo empezar a generar códigos interesantes en MATLAB, algo a lo que se llega con mucho más esfuerzo en un lenguaje de programación más riguroso como C.

Los capítulos que componen este libro son los siguientes:

1. Tutorial de MATLAB.
2. Funciones y Condicionales.
3. Bucles.
4. Vectores.
5. Entrada y salida con formato.
6. Matrices.
7. Algoritmos en MATLAB.

El curso comienza con un tutorial en el que se usa MATLAB como una potente calculadora al principio para terminar dibujando curvas, y agrupando instrucciones en ficheros "script", lo que permite introducir ideas importantes para más adelante crear funciones. Además, se aprovecha este capítulo 1 para insistir en conceptos básicos de manejo de las herramientas del sistema operativo, sobre todo la creación y gestión de carpetas, y la ubicación precisa de las mismas en el sistema de archivos del usuario, bien en un disco local o en la unidad de red del aula de ordenadores donde se trabaja.

Una vez realizado este tutorial, hasta hace unos años empleábamos la organización curricular usual en un curso de programación (comenzando por la entrada y salida y el programa "Hola mundo"). Sin embargo, ahora usamos una estructura similar a la programación funcional, comenzando por el concepto de función y estudiando la entrada y salida casi al final. De este modo, en el capítulo 2 se trabaja al principio sobre el concepto de función, introduciendo inmediatamente la estructura de control condicional, la cual permite construir funciones más complejas e interesantes.

En el capítulo 3 se trabaja con bucles, posibilitando la repetición de operaciones. Es un capítulo de cierta complejidad, dado que no se utilizan todavía vectores ni matrices, donde los bucles surgen de modo natural. Aquí están los ejemplos más interesantes del libro, como el de

comprobar si un número natural es primo. Si el estudiante se siente cómodo al final de este capítulo, el resto del libro será para él una progresión sencilla. Si no se siente cómodo todavía, tendrá ocasión de cubrir los vacíos en los capítulos siguientes.

En el capítulo 4 se introducen los vectores, pues ya se tienen las herramientas para manejarlos. Dependiendo de la duración del curso, este podría ser un momento para terminar, en cursos de 40 o 45 horas, habiendo utilizado MATLAB básicamente como el lenguaje de base para aprender a programar. Para asignaturas más largas se puede continuar con los temas 5, dedicado a entrada y salida con formato (y ficheros), el 6, dedicado a matrices, y el 7, donde se trabajan algoritmos elementales de búsqueda, ordenación y geométricos. Además, en estos últimos temas, se trata de hacer uso de toda la potencialidad de MATLAB pues se pretende aprovechar esta parte del libro para adquirir competencias relevantes en ese sentido. Se incluyen además proyectos de programación de cierta complejidad que serán útiles para proporcionar a los estudiantes una visión global de todo lo aprendido.

No hemos pretendido escribir un compendio exhaustivo ni de programación ni de MATLAB. Debido a ello la selección de temas abordados no es inocente sino que subyace la idea de mostrar únicamente los aspectos considerados esenciales para aprender a programar. Creemos que en ese sentido el libro es una aportación interesante ya que apenas hay en la literatura cursos de introducción a la programación que se apoyen en MATLAB como lenguaje de referencia, y lo más parecido corresponde a cursos de MATLAB en los que en algún capítulo se tratan diferentes aspectos de la programación (ver comentarios sobre las referencias 2, 4 y 5 en la sección correspondiente en la página 217). En cuanto a libros en los que se utilice MATLAB como herramienta de cálculo y análisis, recomendamos el interesante trabajo de Guillem Borrell (referencia 3).

Capítulo 1

MATLAB como calculadora

1.1. Introducción

En este capítulo inicial presentamos un tutorial de MATLAB, una herramienta potentísima, casi estándar para cálculos en muchas ramas de la Ingeniería, y de uso razonablemente simple. Utilizaremos en este libro el intérprete de comandos de MATLAB para que el estudiante se introduzca en el apasionante mundo de la programación de ordenadores. Este tutorial, en el que haremos una descripción de los elementos básicos de MATLAB, tiene como objetivo que el estudiante se sienta cómodo con la interfaz del programa y aprenda a usarlo como si fuese una calculadora. Al final del tutorial será capaz de utilizar MATLAB como una versátil herramienta para realizar operaciones matemáticas escalares, vectoriales y matriciales complejas, manejando sus elementos de cálculo en línea de comandos y mediante "scripts"(programas de MATLAB) pero sin entrar en la programación de funciones, a lo cual nos dedicaremos en capítulos posteriores.

1.2. Conceptos básicos

Para arrancar MATLAB, se procede como con cualquier programa Windows, o sea, Inicio, Programas, MATLAB, y dependiendo de la versión que se tenga instalada, se ejecutará el programa correspondiente. El procedimiento se abrevia si se dispone de un icono de acceso directo en el escritorio. Una vez que hemos arrancado el programa, nos encontramos con algo similar a lo que se observa en la figura 1.1, con pequeñas variaciones dependiendo de la versión.

En el espacio que denotamos como ventana de comandos en dicha figura aparece el cursor con el símbolo ($>>$) o ($EDU >>$), indicando que se pueden introducir órdenes. De hecho, en este tutorial, cuando aparezca este símbolo, se tiene que introducir por teclado la orden que aparece escrita a la derecha del mismo. Podeis, de momento, cerrar las otras ventanas que aparecen en la pantalla, para quedaros simplemente con la ventana de comandos.

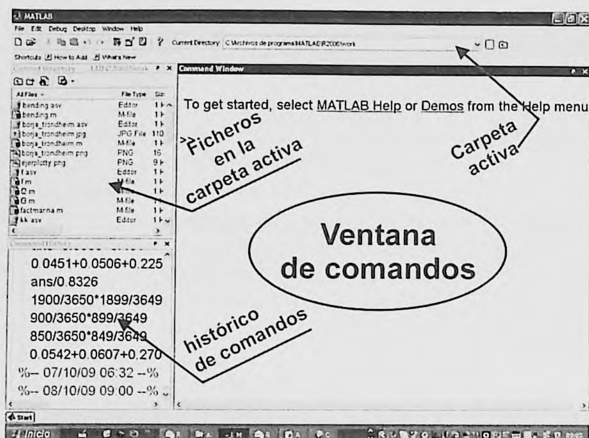


Figura 1.1: Espacio de trabajo de MATLAB

La utilización más básica de MATLAB es como calculadora[1]. Así, por ejemplo, para calcular $\cos(5)\cdot 2^{7.3}$, se debe introducir[2]:

```
>>cos(5)*2^7.3
ans =
   44.7013
```

Es importante resaltar que tanto para las variables como para las funciones MATLAB distingue entre mayúsculas y minúsculas. En este caso concreto el coseno se ha de invocar escribiendo cos en minúsculas.

MATLAB mantiene en memoria el último resultado. Caso de que ese cálculo no se asigne a ninguna variable, lo hace a una variable por defecto de nombre ans. Si queremos referirnos a ese resultado, lo haremos a través de la variable ans, y si no se asigna ese nuevo cálculo a ninguna variable, volverá a ser asignado a ans.

```
>>log(ans)
ans =
    3.8000
```

[1]Funcionando de este modo, es similar a una calculadora programable, aunque bastante más versátil.
[2]Los argumentos de las funciones trigonométricas siempre están en radianes.

En este momento cabría preguntarnos si tratamos con un logaritmo decimal o con uno neperiano (natural). Para saberlo, pedimos ayuda acerca del comando `log` utilizando:

```
>>help log
 LOG    Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also LOG2, LOG10, EXP, LOGM.
```

Aunque en la explicación que se obtiene al pedir `help` de las órdenes los comandos aparecen en mayúsculas, se deben usar en minúsculas. MATLAB dispone de un sistema de ayuda más sofisticado al cual se accede mediante la orden `doc`, tanto de modo genérico o haciendo referencia específica a un comando, como:

```
>>doc log
```

Por defecto, los resultados aparecen con 4 cifras decimales. Si se necesitara más precisión en los resultados, se puede utilizar la orden `format long` repitiendo los cálculos:

```
>>format long
```

Para recuperar una orden y ejecutarla otra vez o modificarla se usan la flechas arriba y abajo del cursor $\Uparrow$, $\Downarrow$. Presionemos $\Uparrow$ hasta recuperar la orden:

```
>>cos(5)*2^7.3
ans =
  44.70132670851334
```

Ejercicio 1.1 *Cambiar el formato para que otra vez se vean sólo cuatro decimales.*

Ejercicio 1.2 *Realizar la siguiente operación:* $2.7^{2.1} + \log_{10} 108.2$.

Ejercicio 1.3 *Pedir ayuda de la orden* `exp`.

Ejercicio 1.4 *Realizar la siguiente operación* $e^{2.7^{2.1}+\log_{10} 108.2}$.

El resultado del ejercicio 1.4 aparecerá como 2.3992e+004. La notación 2.3992e+004, significa $2.3992 \cdot 10^4$ o lo que es lo mismo 23992.

Si necesitamos referirnos a determinados cálculos, se asignan estos a variables y así se pueden recuperar después. El concepto de variable es crucial cuando se programa, como veremos durante todo el libro. Por ejemplo, podemos recuperar con $\Uparrow$ la orden `cos(5)*2^7.3` y asignar su valor a la variable x editando dicha orden.

```
>>x=cos(5)*2^7.3
x =
   44.7013
```

MATLAB buscará en la memoria RAM del ordenador un espacio para guardar esa variable a la que nos referiremos mediante la letra x.

Los nombres de las variables en MATLAB han de comenzar por una letra; además, no contendrán símbolos que no sean letras, números o el guión bajo (que está en la misma tecla que el signo menos). Ya hemos comentado que MATLAB distingue entre mayúsculas y minúsculas y podríamos tener simultáneamente por tanto en memoria una variable X mayúscula con un valor diferente.

Podemos referirnos a la variable recién creada x para utilizarla en otros cálculos:

```
>>y=log(x)
y =
    3.8000
```

Es muy importante señalar aquí que el símbolo $=$ en programación está vinculado a la idea de asignación. Se asigna a lo que hay a la izquierda del $=$ el valor de lo que hay a la derecha del mismo, una vez que se haya realizado la operación que hay a la derecha.

Ejercicio 1.5 *Realizar la siguiente operación:* $2.7^{2.1} + \log_{10} 108.2$ *y asignarla a la variable* x.

Ejercicio 1.6 *Realizar la siguiente operación:* $e^{2.7^{2.1}+\log_{10} 108.2}$ *y asignarla a la variable* t.

Para incidir en esta idea, podemos introducir los siguientes comandos.

```
>> x=3
x =
     3
>> x=2*x
x =
     6
```

El funcionamiento de esta última instrucción es la siguiente. MATLAB busca en la memoria RAM del ordenador el valor de la variable x, la multiplica por 2, y vuelve a colocar en ese lugar de la memoria, el de x, el valor recién calculado. Podemos repetir esta operación tantas veces como queramos con idéntico funcionamiento.

```
>> x=2*x
x =
    12
>> x=2*x
x =
```

```
    24
>> x=2*x
x =
    48
>> x=2*x
x =
    96
```

Si queremos saber cuánto vale una variable, no tenemos más que escribirla en la línea de comandos y pulsar `Enter`.

```
>>y
y =
   3.8000
```

MATLAB respeta la jerarquía de operaciones habitual en los cálculos aritméticos. Así, lo primero que se resuelve son las potencias, luego los productos y divisiones, y finalmente, las sumas y restas. Eso se puede apreciar en el resultado de la siguiente orden:

```
>> 4+3^2*5
ans =
    49
```

Esta jerarquía de operaciones se puede alterar por supuesto con la utilización adecuada de paréntesis.

```
>> (4+3)^2*5
ans =
   245
```

Ejercicio 1.7 *Empezando por $x = 100$ repetir la operación*

$$x = x - \frac{x^2 - 81}{2x}$$

hasta que se estabilice el cuarto decimal de x. ¿Qué relación hay entre el último x y 81?

Ejercicio 1.8 *Definir A como vuestro código postal. Empezando por $x = 100$ repetir la operación*

$$x = x - \frac{x^2 - A}{2x}$$

hasta que se estabilice el cuarto decimal. ¿A qué ha convergido la sucesión?[3]

[3]Las calculadoras obtienen la raíz cuadrada de un número mediante esta sucesión.

A veces es bueno apagar y encender la *calculadora* para borrar todo y empezar de nuevo. Esto se hace con la orden `clear all`. Hay que tener cuidado al utilizarla, ya que borra todas las variables que estén en la memoria sin pedir confirmación.

```
>>clear all
>>x
??? Undefined function or variable 'x'.
```

Ejercicio 1.9 *Preguntar el valor de A igual que acabamos de preguntar x. ¿Tiene sentido el resultado?*

La orden `clc` borra la pantalla.

1.3. Manejo de vectores

Podemos pensar un vector como una lista de valores que reflejan realidades de similar naturaleza; por ejemplo, la lista de DNIs de los estudiantes de una clase, o las tres coordenadas espaciales de un punto. Para crear y almacenar en memoria un vector **v** que tenga como componentes $v_1 = 0$, $v_2 = 2$, $v_3 = 4$, $v_4 = 6$ y $v_5 = 8$ podemos hacerlo componente a componente:

```
>>v(1)=0
v =
     0
>>v(2)=2
v =
     0     2
>>v(3)=4
v =
     0     2     4
>>v(4)=6
v =
     0     2     4     6
>>v(5)=8
v =
     0     2     4     6     8
```

Podemos construir el vector **v** editando directamente entre los corchetes sus componentes:

```
>>v = [0 2 4 6 8]
v =
     0     2     4     6     8
```

Se puede también definir este vector especificando su primer elemento, un incremento y el último elemento. MATLAB rellenará paso a paso sus componentes. Así, podemos definir el vector **v** como una secuencia que empieza en 0, avanza de 2 en 2 y que termina en el 8:

```
>> v = [0:2:8]
v =
      0     2     4     6     8
```

Podemos, para simplificar la sintaxis, prescindir de los corchetes al definir los vectores mediante incrementos.

```
>> v = 0:2:8
v =
      0     2     4     6     8
```

Ejercicio 1.10 *Si escribimos la siguiente orden, cuál será el resultado que aparecerá por pantalla:*

```
>> v = 0:2:9
```

Si ponemos ; (punto y coma) al final de una línea en la ventana de comandos, cuando pulsemos la tecla `Enter` (tecla de retorno de carro) para ejecutarla se ejecutará pero no mostrará el resultado en pantalla (se anula el eco en pantalla). Esto es muy útil algunas veces:

```
>> v = [0:2:8];
>> v
v =
      0     2     4     6     8
```

Es fácil acceder al contenido de una posición del vector, por ejemplo la primera:

```
>> v(1)
ans =
      0
```

O modificarla:

```
>> v(1)=-3;
>> v
v =
     -3     2     4     6     8
```

O hacer operaciones entre componentes, $v_2 \cdot v_5^3$:

```
>> v(2)*v(5)^3
ans =
        1024
```

Ejercicio 1.11 *Calcular la suma de los elementos de v, elemento a elemento.*

Para trasponer un vector o una matriz se usa el apóstrofo, que es el acento que está en la misma tecla que el signo de interrogación "?".

```
>> v'
ans =
      -3
       2
       4
       6
       8
```

Como hemos comentado, para recuperar una orden y ejecutarla otra vez o modificarla se usan la flechas arriba y abajo del cursor ⇑, ⇓. Presionemos ⇑ hasta recuperar la orden:

```
>> v(1)=-3;
```

Modifiquémosla para dejar el valor original

```
>> v(1)=0;
```

Al definir ese vector v de 5 componentes, en realidad lo que definimos es una matriz fila de cinco columnas, o sea, un matriz de 1×5. Esto se comprueba preguntando el tamaño de v con la sentencia `size`:

```
>>size(v)
ans =
      1      5
```

que nos indica que v tiene una fila y 5 columnas.

Ejercicio 1.12 *Definir un nuevo vector que sea el traspuesto de* v *y aplicar a ese vector el comando* `size`. *¿Es coherente el resultado?*

Ejercicio 1.13 *Pedir ayuda sobre la función* `norm` *y aplicarla al vector* v.

Podemos también eliminar una componente de un vector utilizando el operador `[]`. Por ejemplo:

```
>> v(5)=[]
v =
      0      2      4      6
```

y podemos recuperar el vector original haciendo:

```
>> v(5)=8
v =
      0      2      4      6      8
```

1.4. Introducción al tratamiento de matrices

Haremos una introducción a la definición y manipulación de matrices. Se supone que se ha seguido la sección anterior y que se dispone de los conocimientos básicos sobre la definición y manipulación de vectores usando MATLAB. La definición de una matriz es muy similar a la de un vector. Para definir una matriz, se puede hacer dando sus filas separadas por un punto y coma, poniendo cuidado en incluir un espacio en blanco para separar los elementos de cada fila:

```
>> A = [ 1 2 3; 3 4 5; 6 7 8]
A =
     1     2     3
     3     4     5
     6     7     8
```

o definirla directamente fila a fila, que es más intuitivo:

```
>> A = [ 1 2 3
         3 4 5
         6 7 8]
A =
     1     2     3
     3     4     5
     6     7     8
```

Se puede modificar alguno de los elementos de la matriz A accediendo a cualquiera de sus posiciones; por ejemplo:

```
>> A(2,2)=-9
A =
     1     2     3
     3    -9     5
     6     7     8
```

Recuperemos su valor original:

```
>> A(2,2)=4;
```

De igual modo, se la puede considerar como una fila de vectores columna:

```
>> B = [ [1 2 3]' [2 4 7]' [3 5 8]']
B =
     1     2     3
     2     4     5
     3     7     8
```

Recordamos que es importante incluir los espacios en blanco.

Ejercicio 1.14 *Sumar los elementos diagonales de la matriz A refiriéndonse a ellos elemento a elemento.*

Podemos sumar o restar matrices para tener otras matrices:

```
>>C=A+B
C =
     2     4     6
     5     8    10
     9    14    16
```

Ejercicio 1.15 *Definir la matriz $D = 2B - A$.*

También podemos multiplicarlas:

```
>>C=A*B
C =
    14    31    37
    26    57    69
    44    96   117
```

Ejercicio 1.16 *Definir la matriz $D = B - A \cdot B$.*

Ejercicio 1.17 *Definir la matriz $C = AA^t$.*

Podemos definir algunos tipos especiales de matrices, como por ejemplo una matriz de 3×3 que tenga todos sus elementos nulos.

```
>>I=zeros(3)
I =
     0     0     0
     0     0     0
     0     0     0
```

Podemos modificar sus elementos diagonales para tener la matriz identidad.

```
>>I(1,1)=1;
>>I(2,2)=1;
>>I(3,3)=1
I =
     1     0     0
     0     1     0
     0     0     1
```

Ejercicio 1.18 *Repetir el ejercicio 1.16 sacando factor común y utilizando la matriz identidad.*

Otra forma de definir la matriz identidad es a través de la función `diag`, que recibe un vector y lo convierte en diagonal de una matriz cuyos otros elementos son nulos.

```
>>J=diag([1 1 1])
J =
        1     0     0
        0     1     0
        0     0     1
```

Ejercicio 1.19 *Definir una matriz D diagonal cuyos elementos sean -2, 1, 0.2 y -0.7.*

Ejercicio 1.20 *Pedir ayuda de la función* `eye`, *y definir la matriz identidad de* 10×10.

Ejercicio 1.21 *Repetir el ejercicio 1.16 sacando factor común y utilizando la función* `eye`.

1.4.1. Definición de submatrices

La definición de "subvectores" o submatrices es muy fácil. Si `v` es

```
>> v = [0:2:8]
v =
        0     2     4     6     8
```

Podemos definir un vector e cuyas componentes sean las tres primeras componentes del vector `v` poniendo

```
>> e=v(1:1:3)
e =
        0     2     4
```

donde el primer uno entre paréntesis indica que nos referimos como primer elemento al primer elemento de `v`, el segundo número es el incremento de índices dentro de `v` y el último número marca el elemento final. Esta orden es equivalente a la siguiente:

```
>> e=v(1:3)
e =
        0     2     4
```

ya que cuando el incremento es la unidad no es necesario incluirlo al definir la secuencia pues es el incremento por defecto.

Ejercicio 1.22 *Deducir cuál va a ser el resultado de las dos órdenes siguientes:*

```
>> e=v(2:2:5)
>> e=v(1:3:5)
```

Como comentamos al principio, la notación usada por MATLAB sigue en lo posible la notación estándar de Álgebra Lineal. Es muy sencillo multiplicar matrices y vectores, teniendo cuidado de que las dimensiones sean las adecuadas.

```
>> A*v(1:3)
??? Error using == *
Inner matrix dimensions must agree.
>> A*v(1:3)'
ans =
    16
    28
    46
```

Es importante acostumbrase a ver ese mensaje de error. Una vez que se empieza a trabajar con vectores y matrices, es sencillo olvidar los tamaños de los objetos que se han ido creando.

Ejercicio 1.23 *Utilizando el comando* `size`*, razona sobre los problemas en lo que se refiere a dimensiones en la multiplicación anterior.*

Se pueden extraer columnas o filas de una matriz. Si queremos, por ejemplo, que C sea la tercera fila de la matriz A:

```
>> C=A(3,:)
C =
     6     7     8
```

O que C sea la segunda columna de la matriz B

```
>>C=B(:,2)
C =
     2
     4
     7
```

O bien que D sea la submatriz cuadrada de orden dos inferior derecha de la matriz A.

```
>> D=A(2:3,2:3)
D =
     4     5
     7     8
```

Ejercicio 1.24 *Definir una matriz de nombre $D1$ formada por la primera y tercera columnas de la matriz A.*

Una vez que se es capaz de crear y manipular una matriz, se pueden realizar muchas operaciones estándar. Por ejemplo, calcular su inversa. Hay que tener cuidado y no olvidar que las operaciones son cálculos numéricos realizados por ordenador. En el ejemplo, A no es una matriz regular, y sin embargo MATLAB devolverá su inversa ya que los errores de redondeo durante su cálculo convierten en *invertible* a dicha matriz.

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18
ans =
   1.0e+15 *
   -2.7022    4.5036   -1.8014
    5.4043   -9.0072    3.6029
   -2.7022    4.5036   -1.8014
```

Con la matriz B sí que es posible calcular su inversa:

```
>>inv(B)
ans =
   -3.0000    5.0000   -2.0000
   -1.0000   -1.0000    1.0000
    2.0000   -1.0000         0
```

Ejercicio 1.25 *Definir una matriz de nombre $B1$ como la inversa de B. Multiplicar B por $B1$ y razonar la coherencia del resultado.*

Hay que recordar que MATLAB distingue entre mayúsculas y minúsculas. Este puede ser el origen de algunas confusiones si se manejan algoritmos complejos.

```
>> inv(a)
??? Undefined function or variable a.
```

1.5. Resolución de sistemas lineales

También hay funciones para resolver sistemas lineales. Si $A\mathrm{x} = \mathrm{b}$ y queremos encontrar x, el modo más directo es simplemente invertir A, y luego premultiplicar por la inversa ambos lados. Aunque hay medios mucho más eficientes para resolver sistemas lineales de momento nos quedaremos con éste. Por ejemplo, el sistema lineal $B\mathrm{x} = \mathrm{v}$ con:

```
>>v = [1 3 5]'
v =
     1
     3
     5
>>B = [ [1 2 3]' [2 4 7]' [3 5 8]'];
```

se resuelve con:

```
>> x = inv(B)*v
x =
     2
     1
    -1
```

Para comprobar:

```
>> B*x
ans =
     1
     3
     5
```

Ejercicio 1.26 *Definir una matriz* $B2 = BB^t$.

Ejercicio 1.27 *Encontrar la solución del sistema lineal* $BB^t\mathbf{x} = \mathbf{v}$ *asignando esa solución al vector* $\mathbf{x}$.

Ejercicio 1.28 *Comprobar la solución obtenida realizando el cálculo* $BB^t\mathbf{x} - \mathbf{v}$.

Podemos crear una matriz aumentada a partir de B y del término independiente y reducirla hasta convertir el sistema en uno equivalente triangular, efectuando las necesarias transformaciones elementales de fila

```
>>BA=[B v]
BA =
     1     2     3     1
     2     4     5     3
     3     7     8     5
>>BA(2,:)=BA(2,:)-2*BA(1,:)
BA =
     1     2     3     1
     0     0    -1     1
     3     7     8     5
>>BA(3,:)=BA(3,:)-3*BA(1,:)
BA =
     1     2     3     1
     0     0    -1     1
     0     1    -1     2
```

La segunda fila tiene el elemento diagonal nulo, así que hay que realizar una permutación de filas, premultiplicando por la identidad permutada:

```
>>IP=[1 0 0;0 0 1;0 1 0];
>>BA=IP*BA
BA =
     1     2     3     1
     0     1    -1     2
     0     0    -1     1
```

Ahora ya es inmediato resolver este sistema por sustitución hacia atrás:

Ejercicio 1.29 *Definir una matriz H de 3×3 a partir de las tres primeras columnas de la matriz BA.*

Ejercicio 1.30 *Definir un vector* h *utilizando la última columna de BA.*

Ejercicio 1.31 *Definir el vector* z *tal que* $H\mathrm{z} = \mathrm{h}$. *¿Es coherente el resultado?*

Ejercicio 1.32 *Pedir ayuda de la función* `det` *utilizándola con la matriz B y de la función* `rank` *utilizándola con la matriz BA.*

Ejercicio 1.33 *Calcular el determinante de la matriz H.*

1.6. Vectorización de operaciones

Ejercicio 1.34 *Borra la memoria porque vamos a empezar operaciones nuevas re-utilizando nombres de variables ya usadas.*

Con MATLAB es sencillo crear vectores y matrices. La potencia de MATLAB nace de la facilidad con la que se pueden manipular estos vectores y matrices. Primero mostraremos cómo realizar algunas operaciones sencillas: sumar, restar y multiplicar. Luego las combinaremos para mostrar que se pueden realizar operaciones complejas a partir de estas operaciones simples sin mucho esfuerzo. Primero definiremos dos vectores, los cuales sumaremos y restaremos:

```
>> v = [1 2 3]'
v =
     1
     2
     3
>> b = [2 4 6]'
b =
     2
     4
     6
>> v+b
ans =
```

```
     3
     6
     9
>> v-b
ans =
    -1
    -2
    -3
```

Ahora multiplicaremos uno de esos vectores por un escalar cualquiera. Eso se traduce en un nuevo vector cuyas componentes, una a una, han sido afectadas por ese producto:

```
>> 2.4*v
ans =
    2.4000
    4.8000
    7.2000
```

Cuando se trata de sumar(restar) un escalar a un vector, MATLAB asume que esa suma se refiere a todos los elementos del vector.

```
>> 1.7+v
ans =
    2.7000
    3.7000
    4.7000
```

En la multiplicación de vectores y matrices, hay que recordar que MATLAB trata a los vectores (en este caso columna) como matrices de n filas (siendo n la dimensión del vector) y 1 columna y hay que resaltar también que MATLAB sigue de modo estricto las reglas del Álgebra Lineal. En el ejemplo anterior los vectores son por tanto matrices de 3×1, que no pueden ser por tanto multiplicadas directamente. Se debe recordar que en una multiplicación matricial, el número de columnas del primer operando debe ser igual al número de filas del segundo.

```
>> v*b
Error using == *
Inner matrix dimensions must agree.
>> v*b'
ans =
     2     4     6
     4     8    12
     6    12    18
>> v'*b
ans =
    28
```

MATLAB permite realizar las operaciones entre elementos de un vector o matriz de modo muy sencillo. Supongamos que queremos multiplicar, por ejemplo, cada elemento del vector **v** con su correspondiente elemento en el vector **b**. En otras palabras, supongamos que se quiere conocer $v(1) * b(1)$, $v(2) * b(2)$, y $v(3) * b(3)$. Sería estupendo poder usar directamente el símbolo "$*$" pues en realidad estamos haciendo una especie de multiplicación, pero como esta multiplicación tiene otro sentido, necesitamos algo diferente. Los programadores que crearon MATLAB decidieron usar el símbolo "$.*$" para realizar estas operaciones. De hecho, un punto delante de cualquier símbolo significa que las operaciones se realizan elemento a elemento.

```
>> v.*b
ans =
     2
     8
    18
>> v./b
ans =
    0.5000
    0.5000
    0.5000
```

Ejercicio 1.35 *Definir un vector* **w** *tal que sus componentes sean las de* **v** *al cubo.*

Una vez que hemos abierto la puerta a operaciones no lineales, ¿por qué no ir hasta el final? Si aplicamos una función matemática predefinida a un vector, MATLAB nos devolverá un vector del mismo tamaño en el que cada elemento se obtiene aplicando la función al elemento correspondiente del vector original:

```
>> sin(v)
ans =
    0.8415
    0.9093
    0.1411
>> log(v)
ans =
         0
    0.6931
    1.0986
```

Saber manejar hábilmente estas funciones vectoriales es una de las ventajas de MATLAB. De este modo, se pueden definir operaciones sencillas que se pueden realizar fácil y rápidamente. En el siguiente ejemplo, se define un vector muy grande y lo manipulamos de este modo tan sencillo.

```
>> x = [0:0.1:100]
x =
```

```
Columns 1 through 7
     0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
......................
Columns 995 through 1001
99.4000   99.5000   99.6000   99.7000   99.8000   99.9000 100.0000
>> y = sin(x).*x./(1+cos(x));
```

Usando este tratamiento vectorial, se pueden generar gráficos de modo muy sencillo. Damos una muestra de esto que luego completaremos:

```
>> plot(x,y)
```

Ejercicio 1.36 *Definir un vector* t *cuya primera componente sea* -4*, que tenga un incremento entre componentes de 0.05 y termine en el punto 1.*

Ejercicio 1.37 *Definir un vector* y *a partir de cada componente del vector* t *como:*

$$\mathrm{y} = 5e^{-\mathrm{t}^2} + \sin(10\mathrm{t})$$

Ejercicio 1.38 *Dibujar la curva* (t, y) *utilizando de modo adecuado la orden* `plot`.

1.7. Creación de gráficas

En esta sección presentamos con algo más de detalle los comandos para crear representaciones gráficas de funciones. Para mostrar el uso del comando `plot`, utilizaremos la función seno y su desarrollo en serie de Taylor[4] en torno al cero con términos cúbicos, $x - x^3/6$. Para dibujar la gráfica, seleccionamos el paso del vector de abscisas x y sus valores primero y último

```
>>h=0.1
>>xmin=-2;
>>xmax=2;
>>x=xmin:h:xmax;
>>yseno=sin(x);
>>ytaylor=x-x.^3/6;
```

[4]Brook Taylor nació en 1685 en Inglaterra en el seno de una influyente y adinerada familia. Estudió en Cambridge y cuando se graduó ya había escrito su primer artículo matemático de relevancia. Taylor participó activamente en las agrias disputas entre matemáticos británicos ("newtonianos") y matemáticos europeos ("leibnitzianos") sobre la adjudicación del descubrimiento del Cálculo Diferencial.

Aunque las aportaciones de Taylor a las matemáticas son profundas y variadas (entre otras, introdujo el cálculo en diferencias finitas, la integración por partes, desarrolló un método para encontrar soluciones singulares de ecuaciones diferenciales y sentó las bases de la geometría descriptiva y proyectiva) su resultado más conocido es el Teorema de Taylor, que permite el desarrollo de funciones en series polinómicas. Sin embargo, no fue Taylor el primero en obtenerlo; James Gregory, Newton, Leibniz, Johann Bernoulli, y de Moivre habían ya descubierto independientemente variantes del mismo (fuente: http://www-history.mcs.st-and.ac.uk/).

Tras esto, tenemos en los vectores yseno e ytaylor los valores reales y los valores aproximados obtenidos del desarrollo limitado. Para compararlos, dibujamos los valores exactos superpuestos con los aproximados marcados por puntos verdes 'o'.

El comando `plot` se utiliza para generar gráficas en MATLAB. Admite una gran variedad de argumentos. Aquí sólo utilizaremos el rango y el formato, y la posibilidad de representar dos curvas en la misma gráfica.

```
>>plot(x,yseno,'go',x,ytaylor);
```

La g se refiere al color verde (green), y la o significa que los puntos se van a marcar con un circulito. La tilde antes y después de *go* en el comando anterior es la que está en la tecla de la interrogación de cierre. Una vez en la ventana gráfica se puede entrar en modo Edición, y cambiar el aspecto del gráfico, activando el icono de la flecha y haciendo doble-click sobre el gráfico.

Ejercicio 1.39 *En la ventana en la que aparece la figura, seleccionar* `Edit, Copy Figure`. *Abrir un nuevo documento de* Ms-WORD *y pegar la figura en ese documento.*

UNIVERSIDAD TECNICA DE ...

También es buena idea representar la función error:

```
>>plot(x,abs(yseno-ytaylor),'mx');
```

Donde `abs` es la función valor absoluto.

Ejercicio 1.40 *Pedir ayuda de los comandos* `grid` *y* `plot`.

Ejercicio 1.41 *Dibujar la curva* $(\mathtt{t},\mathtt{y})$ *del ejercicio 1.37.*

Ejercicio 1.42 *Dibujar la curva* $(\mathtt{t},\mathtt{y})$ *del ejercicio 1.37 con cruces rojas y con una retícula incorporada.*

Ejercicio 1.43 *Hacer doble click sobre la curva tras seleccionar la herramienta flecha en el menú superior de la ventana de la gráfica. Cambiar la figura de color (violeta) y de grosor (a 6).*

También se puede copiar este gráfico al portapapeles desde la ventana del gráfico, para después pegarlo en un documento Word por ejemplo, como ya vimos en el ejercicio 1.39.

1.8. Conjuntos de órdenes

En esta sección explicaremos cómo reunir órdenes en ficheros ejecutables desde la línea de comandos de MATLAB. A estos ficheros se les suele llamar *scripts*. Ello permite realizar operaciones más complejas, y facilita sus repeticiones. Para empezar a trabajar sobre esta parte del tutorial, lo primero que haremos es ejecutar `clear all` para borrar las variables

activas.

Como ejemplo, consideramos el *script* correspondiente al dibujo de las gráficas de la sección 1.7. Primero hay que crear el fichero. El editor más conveniente es el que trae incorporado el propio MATLAB, aunque cualquier editor de texto es válido dado que la codificación de los archivos de comandos de MATLAB es el estándar ASCII como sucede habitualmente en los entornos de programación. El editor de MATLAB es muy simple y suficiente para este tipo de aplicaciones. A partir de la versión 5, viene incorporado al propio MATLAB y mejora de versión en versión. Los ficheros ejecutables de MATLAB, los *M-files*, deben tener la extensión ".m". En este ejemplo crearemos un fichero de nombre `senotaylor.m`. Para abrir el editor clickamos en (*File, New, M-file*) y debemos ir escribiendo y/o copiando-pegando los comandos necesarios.

Se debe tener en cuenta que cuando una sentencia comienza por %, es un comentario y no se va a ejecutar. Por tanto, en este ejemplo, no es necesario reproducir esas líneas. Es interesante que se pueden comentar o "descomentar" varias simultáneamente sin más que seleccionarlas en el editor, clickando con el botón derecho del ratón y eligiendo la opción correspondiente en el menú flotante que aparece.

```
% file: senotaylor.m. Seno y desarrollo del seno.
% El script genera tres vectores: x con las abscisas,
% yseno con el seno evaluado en esas abscisas e
% ytaylor con el desarrollo de Taylor hasta el
% termino cubico del seno en torno al cero.
xmin=-2;
xmax=2;
h = 0.1;
x=xmin:h:xmax;
yseno=sin(x);
ytaylor=x-x.^3/6;
plot(x,yseno,'rx',x,ytaylor);
shg;
```

Una vez que se hayan introducido las sentencias, se guarda el fichero en la carpeta que creamos conveniente. Ahora hay que informar a MATLAB de la ruta en la que se encuentra para que MATLAB lo encuentre. Esto se puede hacer de varias maneras, dependiendo de la versión de MATLAB que estemos usando. En las versiones 6.5 y superiores se puede hacer modificando la carpeta-directorio activo en la caja correspondiente (ver figura 1.1); para volver a activar la vista inicial de MATLAB se procede como se indica en la figura 1.2. En versiones previas, se puede indicar la ruta del archivo en el *path browser* con el icono correspondiente, o desde el menú *File* con la opción *Set Path*. Por defecto, si se guarda en la carpeta ..\matlab\bin, MATLAB lo encontrará[5].

[5]Si se utiliza MATLAB en el aula de ordenadores o laboratorio de una facultad o escuela, probablemente

Para ejecutar el *script* se ha de volver a la ventana de comandos, tecleando en la línea de comandos el nombre del fichero quitando $.m$. En este caso `senotaylor`.

```
>>senotaylor
```

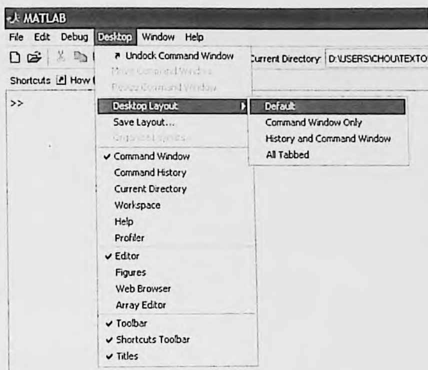


Figura 1.2: Recuperar vista por defecto del entorno MATLAB (ver figura 1.1)

Cuando tecleamos `senotaylor` en la línea de comandos, MATLAB buscará en las carpetas indicadas en el *path* un fichero llamado `senotaylor.m`. Una vez que lo encuentre lo leerá y ejecutará los comandos como si se hubiesen tecleado uno detrás de otro en la línea de comandos. Una vez ejecutada esta instrucción deberá aparecer una ventana con una gráfica como la de la figura 1.3.
Se puede ejecutar el programa otra vez pero con un paso h diferente, para lo cual simplemente hay que cambiar ese valor de h en el editor, guardar y volver a ejecutar.

Para que al final de la ejecución del aparezca siempre el gráfico como ventana activa se utiliza la orden `shg`. Usemos este comando como excusa para invocar el comando de petición de ayuda `doc` que es muy útil también por sus referencias cruzadas a otros comandos.

```
>> help shg

 SHG    Show graph window.
    SHG brings the current figure window forward.
```

el usuario no tenga permiso de escritura en esa carpeta y no pueda guardar ahí sus ficheros. En este caso, se pueden guardar en la carpeta que se desee que después se incorpora a la ruta de búsqueda (path), bien con el comando *path* o con el icono correspondiente.

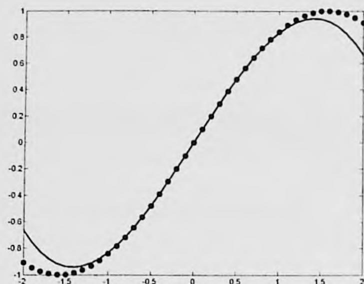


Figura 1.3: Gráfica correspondiente al ejemplo *senotaylor.m*

Ejercicio 1.44 *Estimar la dimensión que tienen que tener los vectores* x, yseno, $\mathrm{ytaylor}$ *y confirmar el resultado utilizando la orden* `size`.

Ejercicio 1.45 *Crear y ejecutar desde MATLAB un fichero que se llame* `CURVATY.m` *con una secuencia de comandos que realicen las operaciones siguientes:*

1. *Borrar todas las variables activas de la memoria.*

2. *Definir un vector* t *cuya primera componente sea* -4*, que tenga un incremento entre componentes de 0.05 y termine en el punto 1.*

3. *Definir un vector* y *a partir de cada componente del vector* t *recién definido como:*

$$\mathrm{y} = 5e^{-\mathrm{t}^2} + \sin(10\mathrm{t})$$

4. *Dibujar la curva* (t, y) *con cruces rojas y con una retícula incorporada.*

Ejercicio 1.46 *Crear y ejecutar desde MATLAB un fichero que se llame* `BAIP.m` *con la secuencia de comandos siguiente:*

```
v = [1 3 5]';
B = [ [1 2 3]' [2 4 7]' [3 5 8]'];
BA=[B v]
BA(2,:)=BA(2,:)-2*BA(1,:)
BA(3,:)=BA(3,:)-3*BA(1,:)
IP=[1 0 0;0 0 1;0 1 0];
BA=IP*BA
```

Ejercicio 1.47 *Pedir ayuda del comando* `pause` *e incorporarlo entre algunas líneas del ejercicio anterior para ver todos los pasos de la secuencia de comandos.*

1.9. MATLAB y números complejos

MATLAB entiende la aritmética compleja y es perfectamente posible trabajar con números complejos. Podemos multiplicar dos números complejos como:

```
>>(2+3i)*(3-7i)
ans =
  27.0000 - 5.0000i
```

O dividirlos como:

```
>>(2+3i)/(3-7i)
ans =
 -0.2586 + 0.3966i
```

1.10. Matemáticas simbólicas con MATLAB

MATLAB dispone de herramientas para cálculo simbólico. Para ello es necesario instalar el *Symbolic Math Toolbox*, que es una especie de versión reducida de Maple, un programa de cálculo simbólico muy conocido. Aquí podemos usar esta caja de herramientas para resolver integrales y calcular determinantes de modo simbólico entre otras cosas. Lo primero que tenemos que hacer es definir una variable como susceptible de ser utilizada en cálculo simbólico:

```
>>syms x
```

Ahora podemos definir una función que dependa de x y cuya integral queramos calcular:

```
>>f=cos(x)^2;
>>int(f)
ans=
1/2*cos(x)*sin(x)+1/2*x
```

Podemos también definir una matriz que dependa de x y de una nueva variable y:

```
>>syms y
>> A=[x y x-y
2 x^2 y
-x -y 0]
A =
[   x,    y, x-y]
[   2, x^2,    y]
[  -x,   -y,    0]
```

Y podemos calcular su determinante de modo simbólico:

```
>> det(A)
ans =
-2*y*x+2*y^2+x^4-x^3*y
```

Ejercicio 1.48 *Calcular de modo simbólico la inversa de la matriz A.*

Podemos evaluar este determinante para valores reales de x e y asignando valores a esas variables y utilizando después la orden `eval`:

```
>> x=2.41
x =
    2.4100
>> y=-3.2
y =
   -3.2000
>> eval(det(A))
ans =
  114.4301
```

En el momento en que hemos asignado valores a las variables, éstas dejan de ser símbolos. Si queremos que vuelvan a serlo tenemos que hacerlo de modo explícito

```
>>syms x
```

Ejercicio 1.49 *Definir una función f como e^{-x^2}.*

Ejercicio 1.50 *Pedir ayuda de la función* `diff` *y calcular la derivada de f. Evaluar esta derivada para $x = -3.327$.*

Ejercicio 1.51 *Pedir ayuda de la función* limit *y calcular el límite de f cuando $x \to \infty$.*

Capítulo 2

Funciones y Condicionales

2.1. General

Una vez que en el capítulo 1 hemos usado MATLAB como una calculadora potente pero al fin y al cabo una calculadora, ha llegado el momento de aprender a programar. En ese sentido, el capítulo 2, dedicado al concepto de función y la estructura de control condicional, y el capítulo 3, dedicado al estudio de los bucles, son los más importantes. Constituyen la base sobre la que se cimenta el resto del curso.

En el presente capítulo se tratan principalmente las ideas de funciones, variables, argumentos de las funciones, asignación y la primera estructura de control del libro: el condicional. También se realiza primeramente una contextualización de este curso desde la perspectiva de la historia de la programación y sus fundamentos.

2.2. MATLAB como un lenguaje de programación

Programación es la elaboración de las instrucciones necesarias para que el ordenador realice una serie de tareas. La estructura básica de un ordenador sigue hoy en día el modelo de Von Neumann[1] que consiste en una unidad de procesado y una memoria. La programación consiste

[1]Von Neumann nació en 1903 en Budapest (Hungría) con el nombre de János. Estudió Química en la Universidad de Berlín debido a que su familia se oponía a que se dedicara a una carrera como las Matemáticas sin perspectiva de reportarle beneficios económicos. Sin previos estudios matemáticos escribió su tesis doctoral en teoría de conjuntos. Impartió clases en las universidades de Berlín y Hamburgo y posteriormente realizó estudios postdoctorales en Göttingen con Hilbert. En 1930 se mudó a Princenton donde dió clases en la universidad durante tres años para posteriormente ser uno de los seis primeros matemáticos en el Instituto de Estudios Avanzados de Princeton. Durante la Segunda Guerra Mundial participó en el Proyecto Manhattan.

Las contribuciones de Von Neumann son muy variadas y profundas en matemáticas y se extienden hasta otros campos, principalmente la física. Desarrolló el formalismo matemático en el que se fundamenta la Mecánica Cuántica. Ha contribuido con importantes aportaciones en economía, teoría ergódica, teoría de juegos, teoría de conjuntos, etc... En los años 30, se interesó en el estudio de la turbulencia hidrodinámica. Para Von Nuemnan, la mejor manera de obtener intuición en los fenómenos no lineales que aparecen en las ecuaciones de la hidrodinámica era a través de los métodos numéricos. A finales de su vida se dedicó al

en diseñar las instrucciones adecuadas (que se almacenarán en la memoria) para que la unidad de procesado realice ciertas tareas utilizando los datos almacenados en la memoria. Un ejemplo de programación lo hemos visto en el capítulo 1 al estudiar los *scripts*.

Los lenguajes de programación han evolucionado mucho desde la aparición de los primeros computadores. En un principio, los programas se tenían que escribir directamente en el código interno del ordenador. Con el incremento de la potencia de los ordenadores y del tamaño de los programas que en ellos se utilizan, este sistema de programación dejó de usarse (salvo en ciertas ocasiones para escribir partes de programas que han de ser muy rápidos o realizar funciones muy específicas) y se comenzaron a usar lenguajes de programación. Un lenguaje de programación es un "idioma" en el que escribir programas más o menos universal (es decir, hasta cierto punto independiente del ordenador) que se entiende más fácilmente por las personas. Este lenguaje ha de traducirse después al lenguaje del ordenador, lo que hace de modo automático un programa llamado compilador.

Según el momento en el que se use el compilador se distinguen dos tipos de lenguajes. Cuando la compilación se hace del programa entero se habla de lenguajes compilados (por ejemplo C o Fortran) y cuando se va compilando cuando se ejecuta se denominan lenguajes interpretados (por ejemplo Basic o MATLAB).

En este libro haremos una introducción general a la programación, válida para los lenguajes de programación más utilizados. Hemos elegido MATLAB porque tiene una sintaxis más sencilla y flexible que los lenguajes usuales y además es interpretado, lo que nos permite dedicar mucho tiempo a entender las estructuras de programación y poco a depurar errores de sintaxis, manejar la entrada y salida, compilar los programas, etc. De hecho, muchos programadores explotan estas características de MATLAB y lo utilizan para probar sus algoritmos antes de codificarlos en lenguajes con sintaxis más compleja.

2.3. Funciones y variables

La organización habitual de un curso de programación supone que se comience por los típicos programas de entrada y salida, el programa "Hola mundo". Sin embargo, en este curso que ahora comienza nos hemos decidido por una estructura similar a la programación funcional, comenzando por el concepto de función y estudiando la entrada y salida ya con el curso muy avanzado. La ventaja de este enfoque es que los alumnos comienzan trabajando los conceptos fundamentales: funciones, estructuras de control y vectores. Como el aprendizaje

estudio de los autómatas celulares. Creó el primer automáta autorreplicante con lápiz y papel, sin ayuda del ordenador. Defendió el uso del *bit* como medida de la memoria de un ordenador y resolvió el problema de obtener respuestas fiables a partir de componentes no fiables del ordenador.

Von Neumann no fue un matemático antisocial encerrado en sus problemas. Le gustaban las fiestas y estaba bien relacionado socialmente, siendo respetado por políticos y hombres de negocios. Murió a causa de un cáncer en 1957 (fuente: http://www-history.mcs.st-and.ac.uk/).

es continuo, añadiendo sucesivamente nuevos conceptos y estructuras, conseguiremos que al final los alumnos estén más familiarizados con las estructuras esenciales de la programación. Esto es posible porque MATLAB proporciona una interfaz estándar para las funciones y desde la línea de comandos es posible ejecutarlas directamente.

El entender las cosas desde la perspectiva funcional proporciona la posibilidad de asimilar desde el principio conceptos claves en programación, como el de encapsulamiento de tareas, la división de tareas y su codificación en funciones, crucial cuando se abordan problemas grandes y se trabaja en equipo. También se trabajan ideas esenciales relativas a el diseño de funciones y a la reusabilidad de código ya escrito, visto ya como cajas negras que realizan determinadas tareas, y que servirá de base para construir nuevas funciones, etc...

Presentamos ahora la función `ud2_f1`, la más sencilla que veremos durante el curso. Tiene un argumento de entrada, x y un argumento de salida y. Lo único que se hace es calcular una expresión matemática sencilla y asignar ese valor a y.

```
% ud2_f1.m
% Una función sencilla. Un solo argumento
function y=ud2_f1(x)
y=x^2-log(x);
```

El símbolo = en programación es una asignación, no siendo por tanto simétrico; se asigna a lo que está a la izquierda del símbolo igual lo que haya a la derecha del mismo, una vez realizadas las operaciones que estén especificadas en esa parte derecha.

El nombre de la función, para evitar confusiones, debe coincidir con el nombre del archivo .m donde esta función se encuentra. Por tanto, como este primer ejemplo es la función `ud2_f1`, debemos guardarla en el archivo `ud2_f1.m`. Nombraremos a las funciones de cada capítulo con el prefijo `udX_` (por unidad didáctica) siendo `X` el número del capítulo.

La primera línea realmente ejecutable de cualquier función comienza siempre con la palabra reservada `function` lo cual es común a todas las funciones que veremos en el libro.

Las líneas iniciales de la función están precedidas del símbolo %. Eso significa que son comentarios que nosotros incluimos para documentar lo que hace la función y el significado de los argumentos de la misma. Además si pedimos ayuda de la función, aparecen esas líneas como explicación de la misma.

```
>> help ud2_f1
  ud2_f1.m
  Una función sencilla. Un solo argumento
```

Lo más interesante de este ejemplo es entender a partir cómo se pasan los argumentos desde la línea de comandos hacia las funciones. Para invocar a la función `ud2_f1` desde la línea de comandos, se puede hacer por ejemplo del siguiente modo:

```
>> ud2_f1(5)
```

Pulsando `Enter`, tecla de retorno de carro, obtendremos la siguiente respuesta:

```
ans =
   23.3906
```

Cuando se pulsa `Enter` se carga en memoria RAM la función `ud2_f1`, y se crea espacio en memoria para la variable x. En ese espacio se coloca el valor 5. Se crea espacio también para y. Las variables x e y se llaman variables locales de la función; el adjetivo locales procede de que viven en el espacio de memoria de la función. Una vez hecho esto, el ordenador ejecuta las sentencias de la función (en este caso una sola) de arriba hacia abajo. Durante esta ejecución se define la variable de salida y, en la cual al final de la misma está el resultado, 23.3906. Una vez terminada la ejecución se devuelve el control a la línea de comandos, se asigna en este caso el resultado a la variable por defecto ans y se borra de la memoria RAM la función `ud2_f1`[2].

Podemos invocar a la función `ud2_f1` ahora del siguiente modo:

```
>> t=5;
>> z=ud2_f1(t)
z =
   23.3906
```

En ese caso, tendremos 4 variables x, y, z y t. Las variables x e y son locales de la función y las variables z y t viven en el espacio de memoria asignado a la ventana de comandos. Cuando se llama a la función, x copia el valor de t y cuando se termina la ejecución de la función es z la que copia el valor de la calculada y antes de que esta desaparezca (ver figura 2.1).

La situación no cambia sustancialmente si invocamos a la función del siguiente modo:

```
>> x=5;
>> y=ud2_f1(x)
y =
   23.3906
```

Volvemos a tener 4 variables en memoria, x e y en el espacio de memoria de la ventana de comandos y x e y en el espacio de memoria de la función `ud2_f1` mientras ésta se ejecuta. De hecho, si cambiamos el valor de la x en el código de la función, la variable t del párrafo anterior y la variable x del espacio de memoria de la ventana de comandos no se ven afectadas. O sea que si cambiamos la función `ud2_f1.m` y la grabamos ahora como:

```
function y=ud2_f1(x)
x=2;
y=x^2-log(x);
```

[2] Este proceso no es exactamente así, pero ésta es la imagen más adecuada para un principiante en programación.

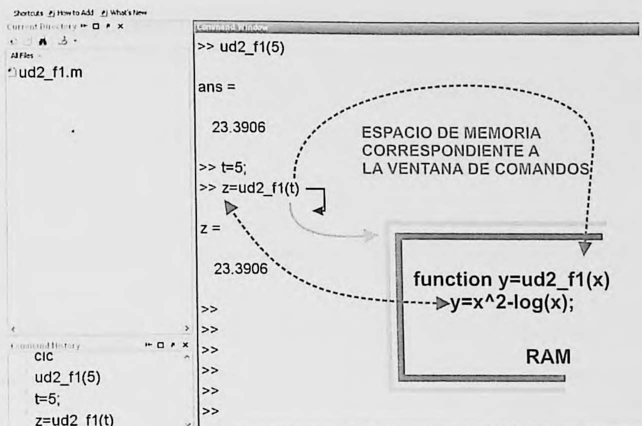


Figura 2.1: Ejemplo `ud2_f1`. Uso de la memoria RAM

Si la invocamos desde la línea de comandos del siguiente modo,

```
>> x=5;
>> y=ud2_f1(x)
y =
    3.3069
>> x
x =
     5
```

el resultado no será correcto, pero x tampoco habrá cambiado su valor. Ello es así porque la variable local x vive en la función. Al principio de la misma, copia el valor de la variable x del espacio de comandos, y aunque cambiemos la variable local en la función, la variable en el espacio de comandos no se ve afectada; están en mundos diferentes que sólo se comunican a través de la línea de argumentos. Hagamos ahora algunos ejercicios para consolidar estas ideas:

Ejercicio 2.1 *Crea una carpeta llamada* `ud2` *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 2*

Ejercicio 2.2 *Edita manualmente la función* `ud2_f1` *creando un archivo nuevo con el editor abriéndolo desde la ventana de comandos (con* File, New*), guárdala en tu carpeta de trabajo y ejecuta:* `>>ud2_f1(2.3)`

```
>>ud2_f1(0.1)
>>ud2_f1(0)
>>ud2_f1(-2.2)
```

¿Son correctos los resultados? ¿Qué errores o problemas da?

Ejercicio 2.3 *Crea una función que reciba el radio de un círculo y devuelva su área. MATLAB conoce el valor de π, pide ayuda sobre* pi *para usarlo.*

Ejercicio 2.4 *Prueba la función que has creado con un círculo de radio la unidad. Debería devolver 3.1416, aproximación del número π. Pruébala con 2; debería devolver 12.5664 (aproximación de 4π).*

2.4. Funciones con varios argumentos de entrada

El siguiente paso es construir funciones en las que haya más de un argumento de entrada. Tenemos así la siguiente, la cual calcula el área de un rectángulo. Si necesitamos más argumentos de entrada simplemente los colocamos uno tras otro separados por comas dentro de los paréntesis a la derecha del nombre de la función.

```
% ud2_farea
% primera función con más de un argumento.
% área del rectángulo de lados a y b
function area=ud2_farea(a,b)
area=a*b;
```

Para invocarla, se nos ocurren estas tres posibilidades, aprovechando lo explicado en la sección 2.3:

```
>> ud2_farea(2,3)
ans =
     6
>> x=3;
>> y=5;
>> ud2_farea(x,y)
ans =
    15
>> ud2_farea(x,4)
ans =
    12
```

En la primera, pasamos directamente dos números, los cuales son copiados por las variables locales a y b. En la segunda posibilidad pasamos dos variables correspondientes al espacio de memoria de la ventana de comandos, x e y, las cuales son copiadas igualmente por las variables locales de la función, a y b, a través de la lista de argumentos. Finalmente, en la

tercera, tenemos una combinación de las dos posibilidades anteriores.

En el ejemplo ud2_farea es posible cambiar de posición los argumentos argumentos de entrada sin que varíe el resultado de la función. Ello es así porque esos argumentos se relacionan entre sí únicamente a través de un producto, el cual es conmutativo. sin embargo, en general, esto no es así. En el siguiente ejemplo se trata de calcular el área de un polígono regular sabiendo el número de lados y el lado. En el cálculo de la apotema los dos argumentos no conmutan entre sí.

```
% ud2_fareapol.m
% Area de un polígono regular de n lados, y lado l
% Primeras variables propias de la rutina, P,a,
% Las variables no conmutan pues juegan distinto papel
function area=ud2_fareapol(l,n)
P=n*l; % perímetro
a=l/(2*tan(pi/n));
area=P*a/2;
```

Esta es la primera función en la que usamos variables estrictamente locales a la función, las cuales no aparecen en la lista de argumentos de la misma. Si invocamos esta función desde la ventana de comandos, y preguntamos después lo que valen esas variables locales, las cuales no pertenecen al espacio de memoria de la ventana de comandos tendremos el siguiente resultado:

```
>> ud2_fareapol(3,4)
ans =
    9.0000
>> ud2_fareapol(4,3)
ans =
    6.9282
>> P
??? Undefined function or variable 'P'.
>> a
??? Undefined function or variable 'a'.
>>
```

Es interesante la posibilidad que ofrece MATLAB de abreviar la llamada a una función; con escribir ud2_ y pulsar la tecla de tabulador MATLAB muestra todas las funciones y *scripts* que comienzan con esos caracteres y es más rápido invocarlos. Los ejercicios correspondientes a estos ejemplos son los siguientes:

Ejercicio 2.5 *Edita manualmente las funciones* ud2_farea *y* ud2_fareapol, *guárdalas en tu carpeta de trabajo y pruébalas desde la ventana de comandos de MATLAB.*

Ejercicio 2.6 *Crea una función que reciba la base b y la altura h de un triángulo y devuelva su área A.*

Ejercicio 2.7 *Crea una función que reciba la masa m y la velocidad v de un móvil y devuelva la energía cinética Ec.*

$$Ec = \frac{1}{2}mv^2$$

Ejercicio 2.8 *Crea una función que reciba dos alturas, h_1 y h_2 y una masa m y devuelva la energía potencial Ep perdida/ganada por dicha masa al caer/subir de h_1 a h_2. Se utilizarán unidades del sistema internacional.*

$$Ep = mg(h_1 - h_2), \quad g = 9.81$$

Ejercicio 2.9 *Consideramos la función* `ud2_fprueba(1,2,3)` *siguiente. Sin ejecutarla, calcula qué valor devolverá si invocamos desde MATLAB* `ud2_fprueba(1,2,3)`*?. Razónalo primero y compruébalo después editando la función y ejecutando esa orden. ¿Conmutan entre sí los argumentos de entrada?*

```
function d=ud2_fprueba(a,b,c)
b=c;
a=b;
d=a+b+c;
```

Ejercicio 2.10 *Crea una función que reciba los tres coeficientes a, b, c, de una ecuación de segundo grado ($ax^2 + bx + c = 0$) y devuelva la raíz*

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

Cuando se invoque la función, se elegirán los coeficientes para que la ecuación tenga raíces reales. Se recomienda usar una variable auxiliar D para definir el discriminante $b^2 - 4ac$. ¿Conmutan entre sí los argumentos de entrada?. Para comprobar si tu código es correcto, usa los coeficientes del polinomio $2x^2 + 5x - 3$, que tiene como raíces -3 y 0.5. ¿Por qué solo aparece una de las dos raíces al invocar la función?

Ejercicio 2.11 *Crea una función análoga para la otra raíz,*

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Ejercicio 2.12 *Codifica una función que reciba los 3 coeficientes (a, b, c) de un polinomio $p(x) = ax^2 + bx + c$ de segundo grado y un escalar T. La función devolverá la integral definida del polinomio p evaluada entre 0 y T, es decir*

$$\int_0^T p(x)dx = a\frac{T^3}{3} + b\frac{T^2}{2} + cT$$

Por ejemplo si el polinomio es el $3x^2 + 2x + 2$ y $T = 1$ el resultado debería ser 4.

Ejercicio 2.13 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.5. Estructura de control condicional if

Hasta ahora, todas las sentencias se han ejecutado de modo secuencial, una detrás de la otra. No disponemos todavía de herramientas que permitan controlar de algún modo la ejecución o realizar operaciones más complejas con la información de entrada a la función. La primera de esas herramientas y una de las más importantes es la estructura de control condicional y nos pone en la parrilla de salida del desarrollo de estrategias y algoritmos para resolver los problemas de los que un ordenador es capaz. Además, y no menos importante, se convierte en el primer mecanismo de provocación de vuestra capacidad de pensar y de articular un discurso complejo a partir de elementos mínimos, o sea, *PROGRAMAR*.

La estructura condicional aparece en los lenguajes de programación normalmente mediante la palabra reservada `if`. En MATLAB lo hace de ese modo, tal como se indica en las siguientes líneas.

```
function y=nombrefuncion(arg1,arg2,....)
....
if cond1 es cierta
    bloque1
end
....
if cond2 es cierta
    bloque2
end
....
```

Por `cond1` nos referimos a una condición lógica o combinación de ellas. Así, `cond1` puede ser que una variable sea mayor que un determinado valor, igual, mayor o igual (>=) etc. En caso de que la condición, en tiempo de ejecución, sea cierta, se ejecutarán las sentencias que hemos identificado como `bloque1` y que están entre la sentencia del `if` y la sentencia `end` que cierre dicho bloque.

Dentro de una misma función puede haber varias estructuras condicionales, en principio independientes entre sí, como mostramos también en el mismo esquema.

Es importante para que los códigos sean legibles tabular o indentar las instrucciones correspondientes a una estructura de control 3 o 4 espacios (serán 4 en nuestros ejemplos), como hemos hecho con `bloque1` y `bloque2` en el esquema anterior.

Uno de los ejemplos más sencillos que se pueden poner de esta estructura es el de una función que devuelva el mayor de dos números a, b supuestos distintos entre sí. Se puede abordar este problema de varias maneras. La más básica es mediante dos estructuras `if`, una de las cuales identifica el mayor de los valores caso de que a y la segunda identifica el mayor caso de que sea b. En este caso los dos bloques no son en realidad independientes, y veremos más adelante que esta no es la forma más natural de resolver este problema, aunque se considera adecuado ahora como ejemplo sencillo para introducir la estructura de control.

```
% ud2_fmayorab0
% primer uso del condicional if
% Devuelve el mayor de dos números a,b
% a,b se supondrán diferentes
function mayor=ud2_fmayorab0(a,b)
if a>b
    mayor=a;
end
%
if b>a
    mayor=b;
end
```

En el ejemplo anterior, los dos valores a comparar han de ser por hipótesis distintos. Es fácil comprobar que si los dos valores son iguales, en cuyo caso, habría que devolver cualquiera de ellos, la función ud2_fmayorab no va a ser capaz de tomar ninguno de ellos como mayor. Para evitar este pequeño inconveniente, se incluye una variante de esa función que utiliza una comprobación (>=) en uno de los casos con lo que si los dos valores son iguales tomará uno de ellos como mayor, que es lo más recomendable.

```
% ud2_fmayorab1
% primer uso del operador >=
% Devuelve el mayor de dos números a,b
function mayor=ud2_fmayorab1(a,b)
if a>=b
    mayor=a;
end
%
if b>a
    mayor=b;
end
```

En la tercera posibilidad se define la variable $mayor$ por defecto como a. Ahora se comprueba si b es mayor que a, y si eso es cierto se define la variable $mayor$ como b. Esta posibilidad es mejor porque ahorra el cálculo de un condicional.

```
% ud2_fmayorab2
% Devuelve el mayor de dos números a,b
function mayor=ud2_fmayorab2(a,b)
mayor=a;
if b>a
    mayor=b;
end
```

En la cuarta variante se juega con la variable a calcular, dándole primero el valor a. Después se compara b con esa variable, y si en la comparación gana b, se actualiza el valor de $mayor$.

Esta es la mejor de las posibilidades porque permitirá de modo sencillo la generalización del algoritmo para encontrar el mayor de una cantidad de números tan grande como queramos.

```
% ud2_fmayorab3
% Devuelve el mayor de dos números a,b
function mayor=ud2_fmayorab3(a,b)
mayor=a;
if b>mayor
    mayor=b;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.14 *Abre el editor de MATLAB y transcribe las funciones ejemplo estudiadas en esta sección:* ud2_fmayorab0, ud2_fmayorab1, ud2_fmayorab2 *y* ud2_fmayorab3. *Guárdalas en tu carpeta de trabajo y pruébalas.*

Ejercicio 2.15 *Crea una función que reciba un número r y devuelva el área del círculo de radio r si $r \geq 0$ y -1 en caso contrario. Por tanto, se definirá en suma una variable $area$ como*

$$area = \begin{cases} \pi \cdot r^2, & r \geq 0 \\ -1, & r < 0 \end{cases}$$

Ejercicio 2.16 *Crea una función que reciba un valor x y devuelva el valor y de la función definida a trozos:*

$$y = \begin{cases} x+1, & x < -1 \\ 1-x^2, & x \geq -1 \end{cases}$$

Ejercicio 2.17 *(Para valientes) Crea una función que reciba tres números a, b, c, que se supondrán diferentes entre si, y devuelva el mayor de los tres. Este es un ejercicio muy interesante, extensión de la última variante de este ejemplo,* ud2_fmayorab3.

Ejercicio 2.18 *Codifica una función que reciba tres valores supuestos diferentes a, b, c y devuelva el mayor de ellos elevado al menor. Por ejemplo, si los números son $a = 3$, $b = 4$ y $c = -1$, la función devolverá $4^{-1} = 0.25$*

Ejercicio 2.19 *Crea una función que reciba cinco números naturales distintos entre si y devuelva la media geométrica del mayor y del menor. La media geométrica de a y b se define como $\sqrt{a \cdot b}$. Por ejemplo, si los números son 2,3,7,4 y 5 entonces la función devuelve $\sqrt{2 \cdot 7} = \sqrt{14} = 3.7417$.*

Ejercicio 2.20 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.6. Estructura de control condicional if-else

En el caso de que queramos que se ejecuten determinadas sentencias cuando la condición sea falsa, deberemos complementar `if` con `else`. De este modo, si la condición es cierta se ejecutará el primer bloque y si es falsa el segundo, como se muestra en el siguiente esquema:

```
function y=nombrefuncion(arg1,arg2,....)
....
....
if cond es cierta
  bloque1
else
  bloque2
end
....
....
```

No hay que confundir esta estructura con dos bloques `if` independientes, uno a continuación del otro, que en principio, no tienen porque ser excluyentes entre si. El ejemplo para ilustrar la estructura `if-else` es otra vez el correspondiente a la función que calcula el mayor de dos números. Se comprueba si el primero es mayor que el segundo y si no es así, se toma el mayor como el segundo.

```
% ud2_felseab
% primer uso de la estructura if-else
% Devuelve el mayor de dos números a,b
function mayor=ud2_felseab(a,b)
if a>=b
    mayor=a;
else
    mayor=b;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.21 *Edita manualmente la función* `ud2_felseab`, *modificando alguna de las anteriores, guárdala en tu carpeta de trabajo y pruébala. Adivina cuál será el resultado si pasas como argumentos dos números iguales.*

Ejercicio 2.22 *Crea una función que reciba un número r y devuelva el área del círculo de radio r si $r \geq 0$ y -1 en caso contrario, utilizando la estructura* `if-else`.

Ejercicio 2.23 *Crea una función que reciba un valor x y devuelva, utilizando la estructura* `if-else`, *el valor y de la función definida a trozos:*

$$y = \begin{cases} x+1 & x < -1 \\ 1-x^2 & x \geq -1 \end{cases}$$

Ejercicio 2.24 *Codifica una función que reciba un número x y devuelva su valor absoluto (sin usar la función* `abs` *ni* `sqrt`, *sino mediante condicionales).*

Ejercicio 2.25 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.7. Función que llama a otra función

Una vez que con una función hemos resuelto un determinado problema y/o hemos agrupado una serie de tareas, es muy útil ver esa función como una caja negra que recibe unos argumentos de entrada y devuelve unos argumentos de salida (de momento uno sólo de salida) sin que tengamos que preocuparnos de cómo calcula/obtiene esos resultados. Vista de ese modo, es natural que sea a su vez llamada por otra función que eventualmente la necesita como parte de sus propios cálculos. La sintaxis de ello no ofrece ningún problema invocándose de modo análogo a como se invoca desde la línea de comandos. Como ejemplo de esta idea codificamos una función definida a trozos en la que el valor de uno de los trozos de la función se obtiene llamando a una función creada previamente, la `ud2_f1`.

```
% ud2_ftrozos
% primera funcion q llama a otra funcion
% Devuelve el valor de la funcion:
% f(x)=x si x<1
% f(x)=x^2-ln(x) si x>=1
function y=ud2_ftrozos(x)
if x<1
  y=x;
else
  y=ud2_f1(x);
end
```

El concepto de llamar a una función desde otra es muy poderoso y es la base tanto para resolver grandes problemas como para ser capaz de repartir la escritura de grandes códigos entre un equipo de programadores. Haremos uso abundante de esta técnica en el libro y si el estudiante es hábil conseguirá simplificar la resolución de muchos ejercicios si hace buen uso de funciones ejemplo estudiadas en clase y de funciones codificadas al resolver otros ejercicios. Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.26 *Edita manualmente la función* `ud2_ftrozos`, *guárdala en tu carpeta de trabajo y pruébala con los siguientes argumentos de entrada valorando si son correctos los resultados y encontrando el origen de posibles errores:*

```
>> ud2_ftrozos(2.3)
>> ud2_ftrozos(1.1)
>> ud2_ftrozos(0)
>> ud2_ftrozos(0.9)
```

Ejercicio 2.27 *Utilizando la función que calcula el área de un rectángulo (*`ud2_farea`*), crea una función que reciba los dos lados de la base y la altura de una pirámide de base rectangular y devuelva su volumen (el volumen de la pirámide es un tercio del área de la base por la altura). Por ejemplo la pirámide de Keops tiene una base cuadrada de 230.5 metros de lado y una altura de 146.6 metros. Por tanto, su volumen aproximado es de 2.5 millones de metros cúbicos.*

Ejercicio 2.28 *Usando las funciones de los ejercicios 2.10 y 2.11, crea una función que reciba los tres coeficientes A, B, C de un polinomio de segundo grado de raíces reales (se elegirán los coeficientes para que así sean) Ax^2+Bx+C y devuelva el producto de las mismas. Prueba con varios polinomios (el producto de las dos raíces ha de ser C/A).*

Ejercicio 2.29 *(Para los valientes) Codifica una función que reciba los tres coeficientes $a0$, $a1$, $a2$ de un polinomio de grado 2, $a_0+a_1x+a_2x^2$. Internamente calculará el discriminante $D=a_1^2-4a_0a_2$ de la ecuación $a_0+a_1x+a_2x^2=0$ para calcular sus raíces. En función del valor de dicho discriminante, cuando el polinomio tenga raíces reales distintas, devolverá el producto de sus raíces calculado invocando la función del ejercicio 2.28. Cuando las raíces tengan parte imaginaria o cuando sea una raíz doble, la función devolverá 0 en lugar del producto de las raíces.*

Ejercicio 2.30 *Crea una función que reciba tres valores x, y, $prec$ y devuelva 1 si la diferencia en valor absoluto entre x e y es estrictamente menor que $prec$ y 0 en caso contrario. Por ejemplo, si $x=0.87$, $y=0.83$ y $prec=0.05$, la función devolverá 1. Si $x=0.87$, $y=0.83$ y $prec=0.01$, la función devolverá 0. Para calcular el valor absoluto se utilizará la función del ejercicio 2.24.*

Ejercicio 2.31 *Codifica una función que reciba tres valores x, y y z (que se supondrán diferentes) y devuelva el mayor de ellos. Se podrá utilizar solo un bloque* `if-else` *y una llamada a la función* `ud2_fmayorab0`

Ejercicio 2.32 *Codifica una función que, llamando a* `ud2_fmayorab0` *,devuelva el máximo de cuatro valores supuestos distintos. Se utilizará la notación x, y, z, t, para esos valores y M para el resultado.*

Ejercicio 2.33 *(Para valientes) Repite 2.32 con seis valores. Trata de hacerlo con un único* `if-else` *y utilizando dos veces la función del apartado 2.31.*

Ejercicio 2.34 *(Para valientes) Repite 2.32 con siete valores. Trata de hacerlo con el menor número posible de* `if-else` *(ninguno).*

Ejercicio 2.35 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.8. Condicionales anidados

Las estructuras de control pueden contener dentro de sus bloques de instrucciones internos otras estructuras de control. Es muy habitual que dentro de un bucle haya condicionales (lo veremos específicamente en la sección 3.4), o que dentro del bloque de un condicional haya un bucle. También es habitual que dentro del bloque de un condicional haya otro condicional. Esto lo reflejamos en el siguiente esquema.

```
function y=nombrefuncion(arg1,arg2,....)
....
....
if cond es cierta
    bloque10
    if cond1 es cierta
        bloque11
    end
    bloque12
else
    bloque2
end
....
....
```

El nuevo condicional no tiene porque estar en el bloque correspondiente a que la condición sea cierta sino que podría estar en el segundo o podría haber condicionales tanto en el bloque del `if` como en el del `else`. Para mostrar el funcionamiento planteamos un ejemplo muy interesante consistente en la codificación de una función que devuelve el signo de una variable entera n. Si n es estrictamente negativa el signo se define como -1, si n es 0 el signo será 0, y si n es estrictamente positiva, el signo se define como +1. Por tanto, hay que gestionar tres posibilidades excluyentes, lo que abordaremos primero eliminando una de ellas frente a las otras dos y después las otras dos entre si.

```
% ud2_fsigno
% función que devuelve el signo de un número entero n
% -1 si es negativo, 0 si es 0, 1 si es positivo.
% primeros condicionales anidados.
function signo=ud2_fsigno(n)
if n<0
    signo=-1;
else
    if n>0
        signo=1;
    else
```

```
        signo=0;
    end
end
```

Hay otras variantes análogas dependiendo de cómo discriminemos los grupos. Es interesante reformular este ejemplo para introducir (un poco con calzador) una sentencia MATLAB importante, abs, la cual extrae el valor absoluto de un número:

```
>> abs(-2.5)
ans =
    2.5000
>> abs(3.1)
ans =
    3.1000
>> abs(0)
ans =
     0
```

El código en cuestión, ud2_fsignoabs (ver más abajo), detecta primero si un número es estrictamente negativo comprobando si su valor absoluto es estrictamente mayor que dicho numero, para a posteriori comprobar si es 0 o estrictamente positivo.

```
% ud2_fsignoabs
% Variante de ud2_fsigno utilizando
% la función propia de MATLAB abs
function signo=ud2_fsignoabs(n)
if n<abs(n)
    signo=-1;
else
    if n>0
        signo=1;
    else
        signo=0;
    end
end
```

En una tercera variante se hace uso del operador de comparación compuesto mayor o igual (>=).

```
% ud2_fsignocomp
% Variante de ud2_fsigno utilizando >=
function signo=ud2_fsignocomp(n)
if n>=0
    if n>0
```

```
        signo=1;
    else
        signo=0;
    end
else
    signo=-1;
end
```

Los ejercicios correspondientes a estos ejemplos son los siguientes:

Ejercicio 2.36 *Transcribe las funciones* `ud2_fsigno`, `ud2_fsignoabs` *y* `ud2_fsignocomp`. *Guárdalas en tu carpeta de trabajo y pruébalas desde la ventana de comandos de MATLAB.*

Ejercicio 2.37 *Crea una función que reciba un valor x y devuelva el valor y de la función definida a trozos:*

$$y = \begin{cases} \sin(x) & x < 0 \\ x & 0 \leq x < 1 \\ x^2 + \log(x) & x \geq 1 \end{cases}$$

Para comprobar el resultado, se tiene que si $x = -\pi/2$, entonces $y = -1$. Si $x = 0.5$, $y = 0.5$, y si $x = 2$, entonces $y = 4.6931$

Ejercicio 2.38 *¿Calcula esta función el mayor de tres números?*

```
function mayor=ud2_fquehace(a,b,c)
mayor=a;
if b>mayor
    mayor=b;
    if c>mayor
        mayor=c;
    end
end
```

Ejercicio 2.39 *Se trata de codificar una función que reciba una calificación y devuelva un variable* `clave` *que ha de valer 0 si la calificación es estrictamente menor que 5, 1 si la calificación es mayor o igual que 5 y menor que 7, 2 si la calificación es mayor o igual que 7 y menor que 9, 3 si la calificación es mayor o igual que 9 y menor o igual que 10 y y -1 si el argumento de entrada no está entre 0 y 10 (solución en apéndice).*

Ejercicio 2.40 *Codifica una función que devuelva el salario semanal de un trabajador en función del coste hora, c, de las horas que ha trabajado, h, y de un fijo de productividad, p, que se cobra si se trabajan más de 30 horas. Si se trabajan más de 40 horas, las horas por encima de esas 40 se pagan un 50 % más caras (horas extras). Por ejemplo, si $c = 10$, $p = 100$ y $h = 30$, el salario es 300. Con iguales coste hora y productividad pero $h = 35$, el salario es 450, y si $h = 45$, el salario es 575.*

Ejercicio 2.41 *Codifica la función*

$$f(x) = \text{máx}\{ud2_f1(x), 7\cos(6x)\}.$$

donde máx *significa máximo. Hay que tener en cuenta que en la definición de* `ud2_f1` *hay un logaritmo del argumento x implicado. Por tanto, cuando x sea menor o igual que cero, ese logaritmo no podrá ser evaluado y tomaremos como máximo la única función que está definida. Para comprobar el resultado, se tiene que si $x = -\pi/6$, entonces $y = -7$. Si $x = \pi/3$, $y = 7$, y si $x = 0.5$, entonces $y = 0.9431$*

Ejercicio 2.42 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.9. Variante elseif en el condicional

Una variante bastante útil del condicional es la que permite abrir el abanico de posibilidades de ejecución a no sólo el verdadero o falso referido a una determinada condición. La idea es que si una condición es cierta se ejecuten unas sentencias, pero si esta es falsa, se compruebe una segunda condición y si esta es cierta, se ejecuten el correspondiente segundo grupo de sentencias, pero si la segunda condición es falsa, se pase a una tercera y así sucesivamente. Finalmente, la orden terminará con una sentencia `else` cuyo bloque de instrucciones posterior se ejecutará si ninguna de las condiciones ha resultado cierta. Pretendemos reflejar esto con el siguiente esquema:

```
function y=nombrefuncion(arg1,arg2,....)
....
....
if cond1 es cierta
    bloque 1
elseif cond2 es cierta
    bloque 2
elseif cond3 es cierta
    bloque 3
elseif.....
    .....
elseif cond n-1 es cierta
    bloque n-1
else
    bloque n
```

```
end
....
....
```

Si la condición 1, `cond1`, es cierta, se ejecutará el bloque 1 de sentencias, y a posteriori se pasará directamente a las sentencias posteriores a la sentencia `end`. Si la condición 1 fuese falsa, se evaluaría la condición 2 y si está fuese cierta, se ejecutaría el bloque 2, pasando directamente a las sentencias posteriores a la sentencia `end`. Si ninguna de las condiciones fuese cierta, se pasaría directamente a la ejecución del bloque n.

Vemos esta estructura con el siguiente ejemplo. Se trata de construir una función que reciba una calificación (entre 0 y 10) y devuelva 0 si es suspenso (calificación estrictamente menor que 5), 1 si es aprobado (calificación mayor o igual que 5 y menor que 7), 2 si es notable (calificación mayor o igual que 7 y menor que 9) y 3 si es sobresaliente (calificación mayor o igual que 9). Si por error se introduce una calificación que no está entre 0 y 10, la función devolverá -1. Para codificar este ejemplo, lo primero que comprobamos es si la nota es menor que 0. Si no lo es, sólo comprobamos si es menor que 5, por que ya sabemos que va a ser mayor o igual que 0. Los demás casos son análogos.

```
% ud2_fnotas
% primera vez q usamos el elseif
function clave=ud2_fnotas(nota)
if nota<0
    clave=-1;
elseif nota<5
    clave=0;
elseif nota<7
    clave=1;
elseif nota<9
    clave=2;
elseif nota <=10
    clave=3;
else
    clave=-1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.43 *Edita manualmente la función* `ud2_fnotas`, *guárdala en tu carpeta de trabajo y pruébala desde la ventana de comandos de MATLAB.*

Ejercicio 2.44 *Crea una función que reciba un valor x y devuelva el valor y de la función definida a trozos:*

$$y = \begin{cases} \sin(x) & x < 0 \\ x & 0 \leq x < 1 \\ x^2 + \log(x) & x \geq 1 \end{cases}$$

Se utilizará la variante `elseif` *del condicional para realizar este ejercicio.*

Ejercicio 2.45 *Repite 2.40 utilizando la variante* `elseif` *del condicional.*

Ejercicio 2.46 *Codifica una función que reciba el salario anual bruto de un trabajador y calcule el impuesto de la renta de las personas físicas correspondiente a ese salario. Se supondrá que los primeros 9000€ están exentos. Hasta 17360€ al año se aplicará una retención del 24 %. Desde 17360 hasta 32360, la retención será del 28 %. Desde 32360, hasta los 52360€ anuales, p[illegible]aremos a Hacienda 37 %. Con ingresos anuales superiores a 52360€, el porcentaje se sitúa en el 4[illegible] %. Para probar la función, si el salario son 7000€, el impuesto será 0. Si el salario es 11000€, el [illegible]puesto son 480€. Si son 22000€, el impuesto es de 3305.6€. Si son 40000, el impuesto son 903[illegible] Si son 100000, el impuesto será de 34092€.*

Ejercicio 2.47 *Codifica la función*

$$f(x) = \text{máx}\{ud2_f1(x), 7\cos(6x)\}.$$

donde máx *significa máximo. En la definición de* `ud2_f1` *hay un logaritmo del argumento x [illegible]plicado. Por tanto, cuando x sea menor o igual [illegible], ese logaritmo no podrá ser evaluado y [illegible]naremos como máximo la única función [illegible] definida. Se utilizar[illegible] [illegible]riante* `els[illegible]f` *del [illegible]dicional para realizar este ejercicio.*

Ejercicio 2.48 *Imagina y especifica un problema susceptible de ser resuelto mediante el [illegible]nador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en [illegible]rticular, las de esta sección.*

2.10. Depuración de códigos: debugger

El ejemplo que acabamos de presentar es una excusa muy buena para introducir [illegible] uso de una herramienta muy poderosa de la que disponen los entornos de programación par[illegible] corregir errores. Es habitual que una función o un programa no realice correctamente los cálc[illegible]os para los que ha sido diseñado. Para detectar los fallos se puede seguir el flujo del mismo en tiempo de ejecución mediante el uso del depurador (*debugger*). Vamos a introducir esta idea con este ejemplo.

Para ello, nos ponemos en el editor, en la línea 4 de `ud2_fnotas.m`, y pinchamos en el icono señalado en la figura 2.2 para activar un punto de parada o *breakpoint*. Una vez hecho esto nos volvemos a la línea de comandos y ejecutamos a la función pasando 9.5 como argumento:

```
>> ud2_fnotas(9.5)
```

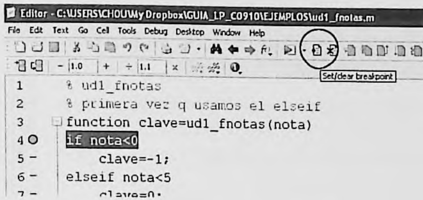


Figura 2.2: Activar *breakpoint*

Veremos que nos aparece la ventana del editor con un punto verde a la izquierda de la sentencia 4, la cual todavía no se ha ejecutado. Podemos ver aquí el valor de una determinada variable, para lo cual simplemente flotamos con el cursor del ratón encima de la misma, como en la figura 2.3. Es posible cambiar el valor de la variable en tiempo de ejecución sin más que asignar un nuevo valor a la misma en la ventana de comandos. Para seguir ejecutando podemos usar los iconos o pulsar F10. Veremos que vamos saltando los "elseif"hasta llegar a la línea 12, en la que la condición es cierta y entramos dentro del bloque correspondiente para asignar el valor 3 a la variable `clave`. Finalmente, si seguimos pulsando F10 terminaremos la función.

Es posible salirse de este modo pulsando el icono correspondiente (explorar los iconos en la barra), es posible también entrar dentro de funciones así como borrar todos los *breakpoints*.

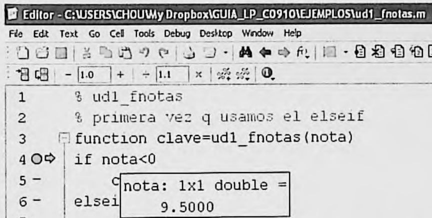


Figura 2.3: *Debugger*: comprobar valor de una variable

2.11. Operadores lógicos

Si queremos construir condiciones algo más interesantes, necesitaremos combinar condiciones elementales mediante los operadores lógicos habituales, estudiados a menudo durante la etapa de la educación secundaria. Necesitaremos saber escribir al menos los operadores Y lógico y el

Ó lógico. Si se pretende construir una condición que mezcle varias condiciones se pueden usar paréntesis para establecer qué operaciones lógicas se hacen primero, de modo similar a como sucede con las operaciones aritméticas.

Para el Y lógico se usa el símbolo reservado &&[3]. En el siguiente ejemplo vemos como se utiliza este operador para construir una función que devuelva el mayor de tres números supuestos distintos. Si uno de los números es mayor lo será porque es al mismo tiempo mayor que uno y mayor que otro.

```
%% operadores logicos.
function mayor=ud2_fmayor(a,b,c)
if a>b && a>c
    mayor=a;
end
if b>a && b>c
    mayor=b;
end
if c>a && c>b
    mayor=c;
end
```

Otro operador interesante es el Ó lógico (inclusivo). Para éste se usa el símbolo ||[4], que se obtiene pulsando 2 veces la combinación de teclas `AltGr` y la del 1. En los siguientes ejercicios veremos algún ejemplo de utilización de este operador.

Ejercicio 2.49 *Edita manualmente la función* `ud2_fmayor`*, guárdala en tu carpeta de trabajo y pruébala desde la ventana de comandos de MATLAB.*

Ejercicio 2.50 *Codifica una función que reciba tres valores a, b y c (que se supondrán diferentes) y devuelva una variable* `flag` *que ha de valer 1 si a es el mayor, 2 si b es el mayor, y 3 si lo es c. Por ejemplo, si $a = 2.3$, $b = 5.1$ y $c = -3.4$, la función devolverá 2.*

Ejercicio 2.51 *Codifica una función que reciba tres valores a, b y c devuelva el mayor si alguno de ellos es estrictamente positivo y el menor en caso contrario. Por ejemplo, si $a = 2.3$, $b = 5.1$ y $c = -3.4$, la función devolverá 5.1. Sin embargo, si si $a = -2.3$, $b = -2.1$ y $c = 0$, la función devolverá -2.3 (solución en apéndice).*

[3] Se puede usar también solamente &, pero hay ciertos matices que diferencian ambos operadores. Cuando se usa &&, caso de que la primera condición sea falsa, MATLAB ni siquiera evalúa la segunda dado que ya la sentencia lógica global va a ser falsa independientemente del valor que tome la segunda condición.

[4] Se puede usar también solamente |; sin embargo || es más eficiente dado que si la primera condición es cierta, MATLAB ni siquiera evalúa la segunda dado que ya la sentencia lógica global va a ser cierta independientemente del valor que tome la segunda condición.

Ejercicio 2.52 *Crea una función que reciba la longitud de los tres lados a, b, c de un triángulo y devuelva su área A obtenida mediante la fórmula de Herón. Caso de que los tres lados no correspondan a un triángulo la función devolverá -1. La fórmula de Herón es:*

$$A = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}; \quad p = \frac{a+b+c}{2}$$

Por ejemplo si los lados son 3/2, 5/3 y 17/6 la función devolverá que el área es 1, y si los lados son 1, 2, 4 devolverá -1.

Ejercicio 2.53 *Busca 3 valores a, b y c para los que la siguiente función no devuelva el mayor de los 3.*

```
function y=ud2_prueba2(a,b,c)
y=a+b+c;
if a>b || c>b
    y=y-b;
    if a>c
      y=y-c;
    else
      y=y-a;
    end
else
    y=b;
end
```

Ejercicio 2.54 *(Para valientes) ¿Qué errores hay en la siguiente función, la cual se supone que debería devolver la potencia de los dos segmentos definidos por las tres abscisas a, b, c? La potencia es el producto de las longitudes de los dos segmentos que determinan. No se sabe cuál es la relación de orden entre los valores pero si fuesen crecientes, la potencia sería $(b-a)(c-b)$. Trata de hacerlo primero en papel para encontrar algunos errores y luego prueba con el ordenador. Prueba con $(1,2,3)$, $(3,2,1)$, $(2,1,3)$, etc y con $(1,1,2)$, $(1,2,1)$, $(2,1,1)$ y $(1,1,1)$ (en estos cuatro últimos casos debería dar 0).*

```
function pot=ud2_fpotencia(a,b,c)
if a>b && a>c
  if b>c
    pot=(a-b)*(b-c)
  else
    pot=(a-c)*(c-b)
  end
end if b>a && b>c
  if a>c
    pot=(b-a)*(a-c)
```

UNIVERSIDAD TECNICA DEL NORTE BIBLIOTECA

```
  else
    pot=(b-c)*(c-a)
  end
else
  if a>b
    pot=(c-a)*(a-b)
  else
    pot=(c-b)*(b-a)
  end
end
```

Ejercicio 2.55 *(Para valientes) Codifica una función que reciba cuatro valores x, y, z, t supuestos todos diferentes entre sí, y devuelva el menor de los estrictamente positivos. Si no hay ninguno estrictamente positivo, la función devolverá 0. Por ejemplo, si $x = -2$, $y = 3$, $z = 7$, $t = -1$, la función devolverá 3.*

Ejercicio 2.56 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.12. Operadores de comparación: ¿son iguales?

La operación de comparar si dos variables son iguales tiene una sintaxis específica en MATLAB[5], colocando el símbolo = de modo duplicado, ==. En el siguiente ejemplo se muestra su uso para construir una función que indica si dos valores son iguales o no.

```
% ud2_figuales
% operador de comprobacion de igualdad
function flag=ud2_figuales(m,n)
if m==n
    flag=1;
else
    flag=0;
end
```

La negación lógica se escribe en MATLAB anteponiendo al operador correspondiente el símbolo ~, el cual se obtiene pulsando las teclas AltGr y 4 (también Alt + 126 con el teclado numérico desactivado). Por tanto, el operador para comprobar si dos valores son distintos será ~=. Se puede reescribir la función anterior utilizando esta posibilidad como:

[5] La comparación para comprobar si dos valores son iguales tiene más matices cuando alguna de las variables no es un número entero. En realidad el concepto matemático de igualdad entre números no enteros exige tener en cuenta un umbral para su diferencia, lo cual es más delicado de programar y no será tenido en cuenta en este texto, en el cual daremos por buena la comparación entre cualquier tipo de variable mediante el operador aquí explicado.

```
% ud2_figualesb
% operador de comprobacion "distinto de"
function flag=ud2_figualesb(m,n)
if m~=n
    flag=0;
else
    flag=1;
end
```

En los siguientes ejercicios veremos algún ejemplo de utilización de ambos operadores.

Ejercicio 2.57 *Edita manualmente la función* `ud2_figuales`, *guárdala en tu carpeta de trabajo y pruébala, desde la ventana de comandos de MATLAB.*

Ejercicio 2.58 *Codifica una función que reciba tres números a, b y c y devuelva una variable* `flag` *que ha de valer 1 si son iguales entre si, y 0 en caso contrario.*

Ejercicio 2.59 *Codifica una función que reciba tres números x, y, z y devuelva una variable* `flag` *que ha de valer 1 si $x \neq y$, 2 si $x = y$ y $y \neq z$ y 3 si los tres valores son iguales*

Ejercicio 2.60 *Codifica una función que reciba tres números a, b y c y devuelva una variable* `flag` *que ha de valer 2 si los tres son iguales entre si, 1 si dos de ellos son iguales entre si pero el otro es diferente, y 0 si los tres son distintos.*

Ejercicio 2.61 *Repetir el ejercicio 2.60 sin utilizar ni* `else`, *ni* `elseif`.

Ejercicio 2.62 *Codifica una función que reciba tres valores x, y, z supuestos todos diferentes entre si, y devuelva el del medio. Por ejemplo, si $x = -2$, $y = 3$, $z = 1$, la función devolverá 1.*

Ejercicio 2.63 *Codifica una función que reciba cuatro valores x, y, z, t supuestos todos diferentes entre si, y devuelva el segundo mayor. Por ejemplo, si $x = -2$, $y = 3$, $z = 7$, $t = 1$, la función devolverá 3.*

Ejercicio 2.64 *Codifica una función que reciba cinco valores x, y, z, t, s supuestos todos diferentes entre si, y devuelva el del medio. Por ejemplo, si $x = -2$, $y = 3$, $z = 7$, $t = 1$, $s = 9$, la función devolverá 3.*

Ejercicio 2.65 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.13. Variables enteras y reales como argumentos

Hasta ahora no hemos prestado atención alguna al tipo de variables (enteras o no enteras) que hemos utilizado, dado que eso no es en realidad necesario en MATLAB al no tener que especificar el tipo, como sucede en otros lenguajes como C por ejemplo. En el siguiente ejemplo conviven sin embargo ambos tipos como argumentos, lo cual es relevante a nivel conceptual, aunque no es necesario tenerlo en cuenta al programar; MATLAB se ocupa de ello. La función recibe los valores correspondientes a las dimensiones principales de una determinada figura elemental (rectángulo, triángulo, círculo, etc...) y recibe también un entero indicando el tipo de figura concreta a la que nos referimos de entre esa lista. La función devuelve el área de dicha figura.

Un aspecto interesante de este ejemplo, es que dependiendo del tipo de figura alguno de los argumentos puede no influir en el resultado de salida. Este es un buen ejemplo también para volver a probar las funcionalidades del "debugger"de MATLAB.

```
% ud2_fareafig
% primera funçion que combina variables esencialmente diferentes,
% una como tipo que funciona como un indice de especie (número
% entero) y otras (a,b) q funcionan como numeros reales.
% ud2_fareafig(tipo,a,b) devuelve el area de distintas figuras
% Si tipo=1, devuelve el area del rectangulo de lados a y b
% Si tipo=2, devuelve el area del circulo de radio a
% Si tipo=3, devuelve el area del triangulo de base a y altura b
% Si tipo=4, devuelve el area del cuadrado de lado a
% Si tipo=5, devuelve el area del triangulo equilatero de lado a
% Si no es ninguno de los tipos anteriores, devuelve -1
function area=ud2_fareafig(tipo,a,b)
if tipo==1
    area=a*b;
elseif tipo==2
    area=pi*a^2;
elseif tipo==3
    area=a*b/2;
elseif tipo==4
    area=a*a;
elseif tipo==5
    area=a*a*sqrt(3)/4;
else
    area=-1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.66 *Prueba la función* `ud2_fareafig`*, bien editándola manualmente o trayéndola de*

una carpeta determinada.

Ejercicio 2.67 *Modifica la función* `ud2_fareafig` *para que también calcule el área del rombo.*

Ejercicio 2.68 *Codifica* `ud2_fareafig` *sin la orden* `else` *ni la* `elseif`.

Ejercicio 2.69 *Codifica una función que reciba un parámetro* `signo`, *y los coeficientes* a, b *y* c *de un polinomio de segundo grado. Devolverá la raíz con el '+' en la fórmula si* `signo` *es* 1 *y la raíz con el '-' si* `signo` *es* $\neq 1$. *Para calcular estas raíces llamaremos a las funciones 2.10 y 2.11. Para comprobar si tu código es correcto, usa los coeficientes del polinomio* $2.34x^2 + 4.29x - 3.71$, *que tiene como raíces* -2.4741 *y* 0.6408.

Ejercicio 2.70 *Codifica una función que reciba los coeficientes de un polinomio de grado 2 y devuelva la suma de sus raíces, para lo cual usará del modo que corresponda la función del ejercicio 2.69. Para comprobar si tu código es correcto, usa los coeficientes del polinomio* $2.34x^2 + 4.29x - 3.71$, *que tiene como raíces* -2.4741 *y* 0.6408, *cuya suma es -1.8333 que será lo que devuelva la función.*

Ejercicio 2.71 *(Para valientes) Sabiendo que una pulgada son 2.54 cm, que un pie son 12 pulgadas, y que una yarda son 3 pies, se pide construir una función que reciba una cantidad, un número que indicará en qué sistema de medida está (0 para el sistema internacional (SI) y* $\neq 0$ *para el sistema inglés) y otro número que indicará en qué unidades está (1,2 o* $\neq 1, 2$ *según sea mm, cm o metros en SI y 1, 2 o* $\neq 1, 2$ *según sea pulgadas, pies o yardas en el sistema inglés). La función devolverá la magnitud convertida a metros si se ha recibido en el sistema inglés y convertida a pies si se ha recibido en el SI.*

Ejercicio 2.72 *Codifica una función que calcule la factura mensual de un teléfono móvil en euros, de acuerdo con los siguientes datos:*

- *Los argumentos de entrada serán:*
  - *Número de llamadas (N)*
  - *Minutos (min)*
  - *Número de mensajes (nsms)*
  - *Tarifa: 1, 2, 3 (idtf)*
- *El establecimiento de llamada es de 15 céntimos.*
- *El coste del mensaje es de 15 céntimos por mensaje.*
- *El coste de llamada para la tarifa 1 es 8 céntimos por minuto.*
- *El coste de llamada para la tarifa 2 es 6 céntimos por minuto.*
- *El coste de llamada para la tarifa 3 es 3 céntimos por minuto.*

- *la tarifa 1 tiene un consumo mínimo de 9€.*
- *la tarifa 2 tiene un consumo mínimo de 15€.*
- *la tarifa 3 tiene un consumo mínimo de 25€.*

Por ejemplo: si las entradas son 25, 180, 16, 2, la función devolverá 16.95€. Si las entradas son 10, 5, 3, 2 no se llega al consumo mínimo y la función devolverá por tanto 15€.

Ejercicio 2.73 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.14. Variables contador y sumador

Ya vimos que el operador = en MATLAB no tiene mucho que ver con lo que ese operador significa en Algebra. Su significado es de asignación. A la izquierda tendremos siempre una variable y a la derecha un expresión que puede combinar llamadas a funciones con operadores aritméticos y lógicos. El funcionamiento de la operación es que primero se evalúa la expresión a la derecha y el valor obtenido es asignado a la variable que está a la izquierda.

Esto abre la puerta para operaciones que no tienen sentido matemático pero sí en programación, como decir que $i = i + 1$. Lo que esto significa es que se evalúa la expresión $i + 1$ y se asigna su valor a i, con lo cual esta variable habrá incrementado su valor original en una unidad. Una variable como esta se llama un contador. A veces, hay otras variables en las que se acumula un determinado valor que no tiene por qué ser el mismo siempre. Nos referiremos a estas últimas como variables sumadoras. En el siguiente ejemplo aplicamos estos dos conceptos para encontrar la suma de los estrictamente positivos entre cuatro números.

```
%% ud2_fsumapos
%% primera variable sumadora (x=x+algo)
%% suma de los positivos entre 4 números.
function suma=ud2_fsumapos(a,b,c,d)
suma = 0;
if a>0
    suma=suma+a;
end
if b>0
    suma=suma+b;
end
if c>0
    suma=suma+c;
end
if d>0
```

```
    suma=suma+d;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.74 *Prueba la función* `ud2_fsumapos`

Ejercicio 2.75 *Codifica una función que reciba 5 números a, b, c, d, y e y devuelva cuántos son estrictamente positivos.*

Ejercicio 2.76 *Codifica una función que reciba 5 números x, y, z, t, y s y dos números a y b (con $a \neq b$ por hipótesis) y devuelva la suma de aquellos números de entre esos 5 que son mayores o iguales que el mayor de a y b, o menores o iguales que el menor de a y b. Por ejemplo, si $x = 7$, $y = 11$, $z = 3$, $t = -1$, y $s = 5$, y $a = 4$ y $b = 6$, la suma en cuestión es $7 + 11 + 3 + (-1) = 20$.*

Ejercicio 2.77 *Codifica una función que reciba 5 números x, y, z, t, y s y dos números a y b (con $a \neq b$ por hipótesis) y devuelva el producto de aquellos números de entre esos 5 que son mayores o iguales que el mayor de a y b, o menores o iguales que el menor de a y b. Por ejemplo, si $x = 7$, $y = 11$, $z = 3$, $t = -1$, y $s = 5$, y $a = 4$ y $b = 6$, el producto en cuestión es $7 \cdot 11 \cdot 3 \cdot (-1) = -231$.*

Ejercicio 2.78 *Codifica una función que reciba 5 números x, y, z, t, y s y un número a y devuelva la media aritmética de aquellos que son mayores que a. Si no hay ninguno, devolverá 0. Por ejemplo, si $x = 7$, $y = 11$, $z = 3$, $t = -1$, y $s = 5$, y $a = 4$, la media en cuestión es $(7+11+5)/3 = 7.6666$.*

Ejercicio 2.79 *Codifica una función que reciba 5 números x, y, z, t, y s y un número $a > 0$, por hipótesis, y devuelva la media geométrica de aquellos que son mayores que a. Si no hay ninguno, devolverá 0. Por ejemplo, si $x = 7$, $y = 11$, $z = 3$, $t = -1$, $s = 5$, $a = 4$, la media en cuestión es $(7 \cdot 11 \cdot 5)^{(1/3)} = 7.2748$ (solución en apéndice).*

Ejercicio 2.80 *Codifica una función que reciba 5 números a, b, c, d, y e y un número adicional positivo ref y devuelva la suma de aquellos valores que sean negativos pero que su valor absoluto sea mayor que ref. Si no hay ninguno, devolverá 0. Por ejemplo, si los números son, -7, 12, 4, -3, -6 y ref vale 3.7, la suma sería -7-6-=-13.*

Ejercicio 2.81 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

2.15. Función parte entera

Hay una función propia de MATLAB que se usa en muchos códigos y que conviene citar; es la función que extrae la parte entera, redondeando hacia $-\infty$, de un número. Así, la parte entera de 3.45 es 3, y la de -4.32 es -5. En el siguiente ejemplo se usa este operador, `floor`, para comprobar si un número es o no entero. El resultado es una variable entera de nombre `flag` que funciona como una variable lógica al recibir solamente valor 1 o 0 dependiendo de que x sea o no un número entero.

```
% ud2_fesentero
% comprueba si un numero x es entero (funcion floor)
function flag=ud2_fesentero(x)
if x==floor(x)
    flag=1;
else
    flag=0;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 2.82 *Prueba la función* `ud2_fesentero`.

Ejercicio 2.83 *Codifica una función que reciba 4 números a, b, c, d y devuelva la suma de los que entre ellos son enteros. Por ejemplo, si los números son 6.3, -4, 5.4, -7, la función devolverá -4-7=-11 (solución en apéndice).*

Ejercicio 2.84 *Codifica una función que reciba un número natural n y devuelva una variable* `flag` *que valga 0 si n es par y 1 si n es impar.*

Ejercicio 2.85 *(Para valientes) Codifica una función que reciba 4 números a, b, c, d y un número entero n y devuelva la suma de los que entre ellos son enteros más aquellos que no siendo enteros, su parte entera, redondeando hacia $-\infty$, sea múltiplo de n. Por ejemplo, si los números son 6.3, -4, 5.4, -7.2 y $n=2$, la función devolverá 6.3-4-7.2=-4.9 (solución en apéndice).*

Ejercicio 2.86 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

Capítulo 3

Bucles

3.1. General

Como ya se indicó al comienzo del capítulo 2, el capítulo 3 contiene junto con aquel la mayoría de las herramientas que un alumno va a tener que utilizar a lo largo de este curso, y para ello nos centraremos en la estructura de control más importante: el bucle. El bucle permite ejecutar de modo repetido bloques de instrucciones sin que estas tengan que aparecer codificadas de modo repetido. Los bucles junto con los condicionales representan la base de la programación estructurada.

Uno de los objetivos de este curso es que los estudiantes adquieran habilidades referidas a la depuración de códigos, cuando estos no funcionen o funcionen pero no proporcionen el resultado correcto. Aunque ya hemos introducido la herramienta de depuración en la sección 2.10, el capítulo que que ahora comienza es muy adecuado para profundizar en estos aspectos, dada la importancia que tiene la posición de determinadas sentencias dentro del código, las cuales afectan a veces no tanto al funcionamiento del mismo como a su funcionamiento correcto.

3.2. Bucles

3.2.1. General

En muchos problemas abordados desde una metodología científica se requiere repetir o iterar un mismo procedimiento. Por ejemplo, un ingeniero tendrá que ver como responde un sistema físico para diversos valores de una de las variables de las que depende. Esta es una tarea repetitiva y susceptible de ser automatizada mediante un bucle. Es por ello que todos los lenguajes de programación contienen la posibilidad de crear bucles que permitirán realizar una misma tarea repetidas veces. Los bucles se utilizan para ejecutar un bloque de instrucciones, conocidas como cuerpo del bucle, de forma reiterada sin tener que repetir varias veces el mismo código, lo cual podría ser imposible de codificar cuando el número de veces a repetir la tarea

se conoce en el tiempo de ejecución.

Durante este capítulo y gran parte del libro implementaremos los bucles mediante la utilización adecuada de la sentencia `while` y de su sentencia de cierre correspondiente `end`. Hay otra sintaxis posible para los bucles, utilizando la sentencia `for`, que veremos en el capítulo 6. El cuerpo de un bucle (sentencias entre `while` y `end`) se ejecutará mientras la condición que acompaña a la sentencia `while` sea cierta, de acuerdo con el siguiente esquema.

```
function y=nombrefuncion(arg1,arg2,....)
....
....
while cond sea cierta
    cuerpo del bucle
end
....
....
```

En los primeros bucles que utilizaremos se define una variable entera que controla mediante incrementos o decrementos la evolución del bucle. A esa variable entera se la conoce como índice del bucle e irá cambiando su valor a medida que se pasa una y otra vez sobre el cuerpo del bucle.

La gran ventaja de los bucles escritos mediante el uso de la sentencia `while` frente a otras formas de escribir un bucle (sentencia `for`) radica en que el índice del bucle es una variable más del programa y su valor es controlado en todo momento por el programador. Sin embargo, un eventual error de programación podría llevar a que el índice nunca llegase a cumplir la condición que pare la ejecución del bucle. Cuando en programación nos encontramos con un bucle en la que la condición de corte o ruptura no se llega a dar, nos referimos a él como bucle infinito. Este error conlleva que no vuelva a aparecer el símbolo típico >> que nos indica que la anterior ejecución ha finalizado. En estos casos MATLAB da al usuario la oportunidad de reaccionar y cortar el proceso pulsando "CTRL+C". Es posible que tras entrar en un bucle infinito MATLAB pierda su estabilidad e interrumpa su funcionamiento normal, algo que coloquialmente denominamos "cuelgue del programa".

3.2.2. Bucles con condición asociada a un índice

En el primer ejemplo de este capítulo se utiliza un bucle para calcular la suma de los números naturales entre 1 y un valor genérico n. Para ello utilizamos un índice dentro del bucle del modo ya explicado en la introducción de esta sección. Este índice i se inicializa a 1 y llegará hasta n que es el argumento de entrada de la función. La condición que requiere la sentencia `while` será cierta mientras el índice i sea menor o igual que n, en cuyo caso i se irá incrementando

en una unidad y la variable $suma$ se verá incrementada en el valor del propio índice i. De esta forma cuando el bucle se interrumpa por haber superado el índice el valor n, la variable $suma$ contendrá el valor resultante de la suma de los n primeros números naturales.

```
%ud3_fsuma
% Suma de todos los naturales entre 1 y n
% primer bucle
function suma=ud3_fsuma(n)
suma=0;
i=1;
while i<=n
    suma=suma+i;
    i=i+1;
end
```

Se puede introducir la sentencia `pause` dentro del bucle y eliminar el punto y coma al final de las sentencias en el cuerpo del bucle para comprobar la evolución de las variables en el mismo:

```
function suma=ud3_fsuma(n)
suma=0;
i=1;
while i<=n
    suma=suma+i
    i=i+1
    pause
end
```

Este tipo de técnicas permiten corregir errores asociados a un mal posicionamiento o inicialización de determinadas variables. Así, si cambiamos de orden las sentencias dentro del bucle, podemos comprobar por qué el resultado obtenido finalmente es incorrecto. Mediante los siguientes ejercicios se podrá afianzar el concepto de bucle.

Ejercicio 3.1 *Crea una carpeta llamada* `ud3` *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 3.*

Ejercicio 3.2 *Prueba la función* `ud3_fsuma`.

Ejercicio 3.3 *Codifica una función que reciba dos números naturales, m y n, con $m < n$ por hipótesis, y devuelva la suma de los naturales entre m y n, incluyendo en dicha suma a m y n. Por ejemplo, si $m = 3$ y $n = 5$ el resultado será $3 + 4 + 5 = 12$.*

Ejercicio 3.4 *Ídem sin usar bucles y llamando a la función* `ud3_fsuma`.

Ejercicio 3.5 *Codifica una función que reciba un número natural,* n *y devuelva la media de la función seno evaluada en los naturales entre 0 y* n, *ambos inclusive.*

$$\frac{\sin(0) + \sin(1) + \sin(2) + \ldots + \sin(n-1) + \sin(n)}{n+1}$$

Para vuestra comprobación, si $n = 2$, el resultado es 0.5836.

Ejercicio 3.6 *(Para valientes) Codifica una función que reciba dos números, reales a, b, con $a < b$, un número natural n, calcule $h = (b-a)/n$ y devuelva*

$$\frac{\sin(a) + \sin(a+h) + \sin(a+2h) + \ldots + \sin(a+(n-1)h) + \sin(b)}{n+1},$$

es decir, el valor medio aproximado del seno entre a y b, tomando $n+1$ puntos para aproximarlo. También se puede ver como la posición vertical del centro de gravedad de la curva $y = \sin(x)$ entre a y b si la consideramos representada por esos $n+1$ puntos. Para vuestra comprobación, si $a = 0$, $b = 2$ y $n = 2$, el resultado debería ser 0.5836.

Ejercicio 3.7 *(Para valientes) Codifica una función que reciba dos números reales a, b, con $a < b$, un número natural n, y calcule la posición vertical del centro de gravedad de la curva $y = \sin(x)$, considerada como una poligonal que se apoya en $n+1$ puntos de esa curva cuyas abscisas están equiespaciadas entre a y b (ambos inclusive). Hay que tener en cuenta que los diferentes tramos de la poligonal pueden tener por supuesto distinta longitud. Por ejemplo, si $a = 0$, $b = 1.5708$ y $n = 3$, los puntos considerados son 0, 0.5236, 1.0472 y 1.5708, cuyas ordenadas son 0, 0.5, 0.866 y 1. La coordenada y del centro de gravedad de estos puntos considerados de igual masa es 0.5893 (solución en apéndice).*

Ejercicio 3.8 *(Para valientes) Codifica una función que reciba dos números reales a, b, con $a < b$, un número natural n, y calcule el centro de gravedad del superficie definida por los rectángulos que tienen como base $h = (b-a)/n$ y como altura $\sin(x_i)$, con $x_i = a + i * h$, $0 \leq i < n$. Por ejemplo, si $a = 0$, $b = 1.5708$ y $n = 3$, los puntos considerados son 0, 0.5236 y 1.0472, cuyas ordenadas son 0, 0.5 y 0.866. La coordenada y del centro de gravedad de los rectángulos que tienen como altura esos puntos es 0.3660 (solución en apéndice). La solución analítica a este problema (0.3927) pasa por evaluar la integral siguiente, algo que podéis intentar utilizando las herramientas simbólicas de MATLAB, introducidas en la sección 1.10*

$$ycg = \frac{\int_0^{\pi/2} 0.5 \sin^2(x) dx}{\int_0^{\pi/2} \sin(x) dx}$$

Ejercicio 3.9 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.2.3. Bucles con incremento distinto de la unidad

En el anterior ejemplo hemos visto un caso donde el índice del bucle se incrementa en una unidad cada vez que se ejecuta el cuerpo del bucle. Con ello conseguimos que dicho índice recorra todos los naturales menores o iguales que uno dado y añadiéndolo a una segunda variable tenemos la suma de todos esos naturales. El segundo ejemplo que vamos a ver es una variante del caso anterior donde se desea que tan sólo se sumen los números pares. De nuevo vamos a utilizar el índice para ir incrementando el valor de la suma final. Sin embargo, el incremento del índice cada vez que se ejecute el cuerpo del bucle será de dos unidades en vez de una; de esta manera índice del bucle irá tomando y sumando exclusivamente valores pares. Dado que se trata de números pares el primer valor que debe tomar el índice i será 2.

```
% ud3_fsumapares
% Suma todos los numeros pares entre 1 y n
function suma=ud3_fsumapares(n)
suma=0;
i=2;
while i<=n
    suma=suma+i;
    i=i+2;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 3.10 *Prueba la función* `ud3_fsumapares`

Ejercicio 3.11 *Codifica una función que reciba un número natural, n, y devuelva la suma de los impares entre 1 y n inclusive. Por ejemplo, si $n = 7$, la suma de impares hasta n es 1+3+5+7=16*

Ejercicio 3.12 *Codifica una función que reciba un número natural, n, y devuelva la suma de los múltiplos de 3 entre 1 y n inclusive. Por ejemplo, si $n = 7$, la suma buscada es 9.*

Ejercicio 3.13 *(Para valientes) Codifica una función que reciba un número natural n y 3 números reales a, b, c. Se trata de calcular la media de los impares menores o iguales que n. Se calculará cuál de los tres valores - a, b o c - tiene una diferencia en valor absoluto con la media más pequeña. Se devolverá ese valor. Por ejemplo, si $n = 7$, la suma de los impares es $1 + 3 + 5 + 7 = 16$, y la media es $16/4 = 4$. Si $a = 7.1$, $b = 5.5$, $c = 2.3$, tenemos que $|7.1 - 4| = 3.1$, $|5.5 - 4| = 1.5$ y $|2.3 - 4| = 1.7$. Por tanto, el valor de los tres que está más cerca de la media es $b = 5.5$ y la función devolverá este valor.*

Ejercicio 3.14 *(Para valientes) Codifica una función que reciba un número natural n y devuelva una variable* `flag` *que valga 0 si* `n` *es par y* `1` *si* `n` *es impar. No se podrá usar la orden* `floor`, *ni llamar a ninguna función.*

Ejercicio 3.15 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.2.4. Bucles con otras operaciones

A diferencia de lo que ocurre en los dos ejemplos anteriores, en el correspondiente a esta sección la variable que se modifica dentro del bucle lo hace a través de productos en vez de sumas, con lo que su valor inicial ha de ser 1. El ejemplo que utilizamos es el de una función que calcula el factorial de un número n:

$$n! = n \cdot (n-1) \cdot (n-2) \ldots 3 \cdot 2$$

Otra variante interesante que introduce este ejemplo respecto a los anteriores es que el índice recorre valores decrecientes.

```
% ud3_ffactorial
% Calcula factorial de n
function fact=ud3_ffactorial(n)
fact=1; % factorial de 0 es 1 por definición.
i=n;
while i>=2
    fact=fact*i;
    i=i-1;
end
```

MATLAB dispone de su propio comando para calcular el factorial de un número; la orden es `factorial`, y la podéis usar cuando lo necesitéis a lo largo del curso.

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 3.16 *Prueba la función* `ud3_ffactorial`. *Compara los valores que obtienes con ella con los que obtienes usando la función propia de MATLAB* `factorial`.

Ejercicio 3.17 *¿Qué devuelve la siguiente función? Sin ejecutarla, trata de calcular qué devolvería* `ud3_fprueba(2,15,3)`. *Compruébalo con MATLAB.*

```
function y=ud3_fprueba(m,n,k)
y=1;
i=m;
while i<=n
    y=y*i
    i=i+k;
end
```

Ejercicio 3.18 *Codifica una función que reciba x y n con n natural y devuelva x^n, para lo cual no se podrá usar el operador ∧ sino que se ha de repetir la operación $x \cdot x \cdots x$ mediante un bucle el número de veces que sea necesario. Por ejemplo, si $x = 3$ y $n = 4$ el resultado ha de ser 81.*

Ejercicio 3.19 *Codifica una función que reciba x y n, y devuelva:*

$$\sum_{i=1}^{n} x^i$$

Para calcular cada uno de los términos de la suma, se llamará a la función del ejercicio 3.18 como corresponda. Por ejemplo, si $x = 3$ y $n = 4$, el resultado es $3^1 + 3^2 + 3^3 + 3^4 = 120$.

Ejercicio 3.20 *(Para valientes) Codifica una función que reciba x y n, y calcule:*

$$\sum_{i=1}^{n} x^i$$

No se podrá usar el operador $\wedge$ ni llamar a ninguna función y solo se podrá usar un bucle. Por ejemplo, si $x = 3$ y $n = 4$, el resultado es $3^1 + 3^2 + 3^3 + 3^4 = 120$.

Ejercicio 3.21 *Codifica una función que reciba 2 números naturales m y j siendo $j < m$, y devuelva, utilizando un solo bucle y sin llamar a ninguna función, el resultado de la siguiente operación:*

$$\frac{m!}{(m-j)!}$$

Por ejemplo si $m = 7$ y $j = 3$ el resultado es 210.

Ejercicio 3.22 *Codifica una función que reciba 2 números naturales n y m siendo $n < m$, y devuelva la suma*

$$\sum_{i=n}^{m} \binom{m}{i}$$

donde

$$\binom{m}{i} = \frac{m!}{i!(m-i)!}, \quad 0! = 1$$

Por ejemplo si $n = 2$ y $m = 4$ el resultado es 6+4+1=11.

Ejercicio 3.23 *¿Qué calcula la siguiente función? Sin ejecutarla, trata de deducir cuánto vale* `ud3_fguess(2,5)`. *Compruébalo con MATLAB.*

```
function suma=ud3_fguess(x,n)
suma=0;
i=1;
pot=x;
while i<=n
    suma=suma+pot;
    pot=pot*x;
    i=i+1;
end
```

Ejercicio 3.24 *(Para valientes) El número π, se puede obtener mediante la fórmula*

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + (-1)^i \frac{1}{2i+1} + \ldots\right)$$

Codifica una función que reciba un número natural n y devuelva la aproximación de π mediante los $n+1$ primeros sumandos de la expresión anterior (solución en apéndice). Por ejemplo, si $n = 3$, los sumandos y el resultado son:

$$4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}\right) = 2.8952$$

Ejercicio 3.25 *Crea una función que dado un natural n, devuelva el error que se comete al sustituir $\pi/4$ por la serie $\sum_{i=0}^{n}(-1)^i \frac{1}{2i+1}$, es decir*

$$error = \left|\pi/4 - \sum_{i=0}^{n}(-1)^i \frac{1}{2i+1}\right|$$

Así por ejemplo si $n = 4$, el error es

$$\left|\pi/4 - 1 + \frac{1}{3} - \frac{1}{5} + \frac{1}{7} - \frac{1}{9}\right| = 0.0495.$$

La alternancia de signo se debe conseguir sin utilizar ∧. No se podrá llamar a ninguna función.

Ejercicio 3.26 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.3. Bucles y relaciones de recurrencia

Otra aplicación interesante de los bucles consiste en la implementación de fórmulas recurrentes, muy usadas en cálculo numérico. En estos caso el índice suele jugar el papel de contador de modo que sólo sirve para avanzar en el bucle y no entra explícitamente en el cálculo de la variable que se vaya actualizando. En una fórmula recurrente tipo $x_{i+1} = f(x_i, x_{i-1}, ...)$ el valor de una variable en un paso $i+1$ depende de su valor en los pasos anteriores, o sea, i, $i-1$, $i-2, \ldots$. El índice del bucle se irá actualizando en el cuerpo del bucle, y a medida que se vaya ejecutando repetidamente dicho bloque de sentencias el valor de la variable con la que nos refiramos a x_{i+1} irá a su vez cambiando en cada uno de sus pasos. En el siguiente ejemplo se implementa la relación de recurrencia ya estudiada en la sección 1.2 del tutorial, y que como comentábamos allí, es en realidad el método de Newton[1] para resolver la ecuación $x^2 - 81 = 0$.

[1] Isaac Newton (1642-1727). Fue el creador de la física moderna, y de influencia por tanto decisiva en el desarrollo de la humanidad. Nació en una granja en Lincolnshire, al oeste de Inglaterra, el día de Navidad de

$$x_{i+1} = x_i - \frac{x_i^2 - 81}{2x_i}$$

```
% ud3_fnewton
% Recibe a y n y devuelve el termino n-esimo
% de la sucesion x_(i+1)=x_i-((x_i)^2-81)/(2*x_i), siendo
% a el primer termino, x_1=a
% Relacion entre sucesiones y bucles.
function x=ud3_fnewton(a,n)
x=a;
i=2;
while i<=n
    x=x-(x^2-81)/(2*x);
    i=i+1;
end
```

Cuando el valor de la variable en el paso $i+1$, x_{i+1}, depende exclusivamente del valor en el paso anterior x_i como en el ejemplo visto anteriormente, bastará una única variable x para el cálculo de la relación de recurrencia. La cuestión se complica en el caso de que el valor en el paso $i+1$ dependa tanto del valor en el paso i como de los valores en otros pasos anteriores a i. Es decir, hablamos de leyes de recurrencia de la forma general $x_{i+1} = f(x_i, x_{i-1}, x_{i-2}, \ldots)$. En estos casos habrá que hacer uso de variables auxiliares que nos permitan guardar los diferentes valores de x en los distintos pasos y actualizar su valor antes de que volvamos a aplicar la ley de recurrencia. Esto sucede por ejemplo en ejercicios relacionados con la sucesión de Fibonacci, como veremos más adelante.

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 3.27 *Prueba la función* `ud3_fnewton` *con algunos valores y comprueba que los resultados son correctos.*

Ejercicio 3.28 *Codifica una función que reciba un número natural n y devuelva el término n-ésimo de la sucesión*

$$x_{i+1} = \sqrt{2x_i}, \quad i \in \mathbb{N}, x_1 = 3.57$$

1642. Su padre había muerto dos meses antes y su madre pronto se volvió a casar dejando a Newton al cuidado de sus padres. Su infancia fue solitaria e influyó en su carácter introvertido y en la tendencia al secretismo que luego se mostró a lo largo de su vida, especialmente en la resistencia a publicar sus monumentales descubrimientos que guardó para sí mismo durante larguísimos periodos de tiempo.

En 1661 Newton dejó Lincolnshire para seguir sus estudios en Cambridge. El periodo 1661-1665 de sus estudios de grado fue irrelevante pero en 1665 regresó a pueblo natal huyendo de la peste que había obligado a cerrar las universidades. Allí, en la soledad del campo, se produjo un arrebato de creatividad incomparable, de dos años de duración, entre los 22 y los 24 años, en el que descubrió el cálculo diferencial, la composición de la luz blanca y la ley de gravitación universal.

Fue un soltero de gustos simples, muy sensible a las críticas que le producían amargos resentimientos y enfados. Como reacción a las críticas de Robert Hooke (el de la ley del muelle) a finales de 1670 escribió, en otro periodo de 18 meses de concentración increíble, su trabajo más importante, el *Principia Mathematica*, donde enunció las leyes de la dinámica (fuente: http://www-history.mcs.st-and.ac.uk/).

Si por ejemplo $n = 3$, $x_2 = \sqrt{2x_1} = 2.6721$, $x_3 = \sqrt{2x_2} = 2.3117$ *que es lo que devolverá la función. Prueba con valores de* n *más grande. ¿A qué valor parece converger la sucesión cuando* n *tiende a* ∞*?*

Ejercicio 3.29 *Codifica una función que reciba un número natural* n, *con* $n \geq 2$ *por hipótesis, y devuelva* $|x_n - x_{n-1}|$, *siendo* x_n *y* x_{n-1} *los términos* n *y* $n-1$ *de la sucesión del ejercicio 3.28. No se podrá llamar a ninguna función y solo se podrá usar un bucle. Por ejemplo, si* $n = 5$, *el resultado ha de ser 0.0765 (solución en apéndice).*

Ejercicio 3.30 *Codifica una función que reciba un número natural* n, *con* $n \geq 3$ *por hipótesis, y devuelva el* n*-ésimo término de la sucesión de Fibonacci*[2].

$$x_n = x_{n-1} + x_{n-2}, \quad x_1 = 1, \ x_2 = 1.$$

Por ejemplo, si $n = 6$, *los términos de la sucesión son 1, 1, 2, 3, 5, 8, y función devolverá 8 .*

Ejercicio 3.31 *Modifica el ejemplo 3.30 para que reciba un número natural* n, *con* $n \geq 3$ *por hipótesis, y dos números* x_1 *y* x_2 *y devuelva el término* n*-ésimo de la sucesión de Fibonacci iniciada por esos valores. Por ejemplo, si* $x_1 = -0.5$, $x_2 = 1$ *y* $n = 6$, *los términos de la sucesión son -0.5, 1, 0.5, 1.5, 2, 3.5 y función devolverá 3.5.*

Ejercicio 3.32 *Codifica una función que reciba un número natural* n *(con* $n \geq 4$ *por hipótesis), tres valores* x_1, x_2, *y* x_3, *y devuelva el* n*-ésimo término de la siguiente sucesión:*

$$x_n = -x_{n-1} + x_{n-2} + x_{n-3}.$$

Por ejemplo, si $x_1 = 4$, $x_2 = -1$, *y* $x_3 = -3$, *para* $n = 4$ *tendríamos:*

$$x_4 = -x_3 + x_2 + x_1 = 3 - 1 + 4 = 6,$$

---

[2]Leonardo Pisano nació en 1170 probablemente en Pisa, en el seno de la familia Bonacci, de ahí su sobrenombre Fibonacci, por el que es generalmente conocido. Su padre fue representante de los mercaderes de la República de Pisa en el norte de África, en lo que es hoy el noroeste de Argelia. Allí creció y se educó Fibonacci. En los numerosos viajes en los que acompañó a su padre, Fibonacci aprendió las ventajas de los sistemas matemáticos utilizados en otras culturas. Hacia 1200 regresó a Pisa y recopiló en diversos libros lo aprendido en sus viajes añadiendo además notables contribuciones propias.

Por ejemplo, en uno de sus libros introdujo el sistema decimal indo-arábigo, sistema de numeración posicional que usamos actualmente y extendió por Europa el uso del sistema árabe de numeración. En el mismo libro aparece la conocida sucesión de Fibonacci, 1, 1, 2, 3, 5, 8, 13, 21, 34,... (en la que cada término es suma de los dos anteriores) como solución al siguiente problema: *Un hombre aisla un par de conejos en un corral. ¿Cuántos pares de conejos hay al cabo de un año si se supone que cada par de conejos engendra un nuevo par que a partir del segundo mes engendra a su vez un nuevo par(ver fig. 3.1)?*

Sin embargo, la mayor aportación matemática de Fibonacci se encuentra en el área de la teoría de los números, en la que, entre otros resultados destaca el estudio de métodos matemáticos para encontrar triples pitagóricos, es decir, tres números naturales, m, n y k que verifican $m^2 + n^2 = k^2$ (fuente: http://www-history.mcs.st-and.ac.uk/).

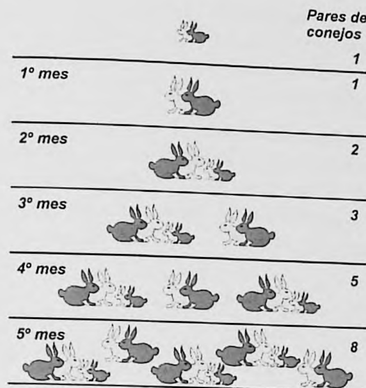


Figura 3.1: Sucesión de Fibonacci

y para $n = 5$ tendríamos:

$$x_5 = -x_4 + x_3 + x_2 = -6 - 3 - 1 = -10.$$

Ejercicio 3.33 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.4. Bucles y condicionales

En ocasiones es necesario combinar un bucle con un condicional para lograr distinguir, de entre un conjunto de potenciales candidatos, aquellos que cumplan una cierta condición. Esta situación implicaría que dentro del bucle *while-end* nos encontraríamos un condicional *if-end*. Las posibilidades de combinar bucles y condicionales son infinitas, y permiten abarcar una casuística de gran potencial para poder seleccionar, contar y/o calcular en conjuntos diversos.

Como ejemplo de lo dicho vamos a combinar un bucle y un condicional para contar los divisores de un número natural n. Para comprobar si un número es divisor de otro utilizaremos la función propia de MATLAB `mod`, que devuelve el resto de la división entera entre dos números. Así por ejemplo:

```
>> mod(14,6)
ans =
```

```
       2
>> mod(14,7)
ans =
       0
```

Para contar los divisores se recorrerán todos los potenciales candidatos a divisor, o sea los números naturales que van desde 1 a n. Se comprobará si cada uno de esos números verifica la condición, en cuyo caso, la variable de conteo `cont` se incrementa en una unidad. Si no se verifica la condición, dicha variable no se modifica.

```
% ud3_fcuentadiv
% ud3_fcuentadiv(n) cuenta los divisores de n
% primer uso de mod, y primer condicional dentro de un bucle.
function cont=ud3_fcuentadiv(n)
cont=0;
i=1;
while i<=n
    if mod(n,i)==0
        cont=cont+1;
    end
    i=i+1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 3.34 *Copia el ejemplo* `ud3_fcuentadiv` *a tu carpeta de trabajo. Pruébalo con algunos valores y comprueba que los resultados son correctos.*

Ejercicio 3.35 *Codifica una función que devuelva el número de divisores impares de un número natural n. Por ejemplo, si $n = 50$, la respuesta es 3, dado que 50 tiene como divisores a 1, 2, 5, 10, 25 y 50, de los cuales 3 son impares.*

Ejercicio 3.36 *Codifica una función que reciba un número natural n y devuelva una variable* `flag` *que ha de valer 0 si n es par y 1 si n es impar. Se usará la orden mod.*

Ejercicio 3.37 *Codifica una función que reciba un número natural n, con $n > 1$ por hipótesis, y devuelva el divisor máximo estrictamente menor que n. Por ejemplo, si $n = 12$, el resultado sería 6, mientras que si $n = 7$, el resultado sería 1.*

Ejercicio 3.38 *Codifica una función que reciba un número x, sume los números naturales estrictamente menores que x que son pares y no son múltiplos de 3 entre 1 y x y devuelva ese valor. Por ejemplo, si $x = 10.47$, la función devolverá 24, pues 2,4,8,10 son pares, no son múltiplos de 3, y son menores que 10.47.*

Ejercicio 3.39 *Codifica una función que reciba dos números naturales a y b y devuelva una variable* `flag` *que ha de valer* 1 *si a y b son una pareja de números* amigos, *y* 0 *en caso contrario. Se dice que dos números a y b son una pareja de números* amigos *si la suma de los divisores propios*[3] *de a dividida por a es igual a la suma de los divisores propios de b dividida por b.*

Por ejemplo, si $a = 30$ *y* $b = 140$ *entonces la función debería devolver* 1, *ya que la suma de los divisores propios de* 30 *es* $1 + 2 + 3 + 5 + 6 + 10 + 15 = 42$ *y la suma de los divisores propios de* 140 *es* $1 + 2 + 4 + 5 + 7 + 10 + 14 + 20 + +28 + 35 + 70 = 196$. *Como se tiene que* $42/30 = 1.4$ *es igual a* $196/140$, *se puede ver que* $(30, 140)$ *son una pareja de números amigos. También debería devolver* 1 *si* $a = 6$ *y* $b = 28$. *Por otro lado, si* $a = 8$ *y* $b = 12$ *la función debería devolver* 0 *ya que no son pareja de números amigos.*

Ejercicio 3.40 *Codifica una función que reciba un número natural n, recorra los valores*

$$\sin(1),\ \sin(2),\ \sin(3),\ \ldots,\ \sin(n),$$

y devuelva el número de ellos que son positivos. Por ejemplo, si $n = 5$, *hay 3 positivos.*

Ejercicio 3.41 *Codifica una función que reciba un número n, recorra los valores*

$$\sin(1),\ \sin(2),\ \sin(3),\ \ldots,\ \sin(n),$$

y devuelva el producto de los que son positivos. Por ejemplo, si $n = 5$, *el producto es 0.1080.*

Ejercicio 3.42 *La siguiente función recibe un número n y debería devolver el número de valores entre* 1 *y n para los que el seno es mayor que 0.5 menos el número de aquellos para los que es menor que -0.5. Corrígela para que funcione correctamente.*

```
function c=ud3_fprueba2(n)
i=1;
while i<=n
  if sin(i)>0.5
    c=c+1;
  else
    c=c-1;
  end
  i=1+1;
end
```

Ejercicio 3.43 *Codifica una función que reciba dos números naturales m, n y devuelva la cantidad de divisores comunes. Por ejemplo,* $m = 14$ *y* $n = 6$ *tienen dos divisores comunes, el 2 y el 1.*

[3] Los divisores propios de un números son los divisores del número sin incluir el propio número e incluyendo el 1.

Ejercicio 3.44 *Codifica una función que reciba dos números naturales m y n. La función devolverá el resultado de dividir el factorial de su suma por el número de divisores comunes. Por ejemplo, si $m = 4$ y $n = 6$, tienen 2 divisores comunes, el 1 y el 2. El factorial de la suma de 4 y 6 es el factorial de 10, que es 3628800. El resultado de dividir este valor por 2 es 1814400 que habría de ser el resultado de la función en este caso.*

Ejercicio 3.45 *(Para valientes) Codifica una función que reciba dos números naturales m y n, y devuelva su máximo común divisor mcd. Por ejemplo, si $m = 12$ y $n = 18$, $mcd = 6$ (solución en apéndice).*

Ejercicio 3.46 *(Para valientes) Codifica una función que reciba dos números naturales m y n, y devuelva su mínimo común múltiplo mcm. Por ejemplo, si $m = 12$ y $n = 18$, $mcm = 36$ (solución en apéndice).*

Ejercicio 3.47 *(Para valientes) Codifica una función que reciba 3 naturales m, n, p y devuelva una variable* `flag` *tal que* `flag=1` *si el mínimo de los máximos comunes divisores entre dos de estos números corresponde a la pareja m, n;* `flag=2` *si corresponde a la pareja m, p y* `flag=3` *si es n, p. Si por ejemplo $m = 20$, $n = 30$, $p = 15$,* `flag` *valdría 2.*

Ejercicio 3.48 *Repetir el ejercicio 3.47 pero sin llamar ahora a ninguna función, en particular a la del ejercicio 3.45.*

Ejercicio 3.49 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.5. Bucles inconclusos: la sentencia break

En algunas ocasiones puede ser necesario interrumpir la ejecución de un bucle porque en el cuerpo del mismo se haya dado una determinada situación que implica que no sea necesario seguir ejecutando dicho bucle. En estos casos se hace necesario tener la posibilidad de romper el bucle y salir del mismo. MATLAB dispone de la sentencia `break` para realizar dicha acción. Si el código llega hasta una instrucción `break`, automáticamente dará por finalizado el bucle `while` que contiene a dicho `break` y pasará a la instrucción posterior a la sentencia `end` que cierra dicho bucle.

La función correspondiente al ejemplo que se muestra a continuación devuelve un valor 1 o 0 dependiendo de que el número que introducimos como argumento sea o no primo[4]. Al igual

[4]En Matemáticas, no se considera que 1 sea un número primo para mantener la propiedad de que la descomposición de naturales como producto de primos es única. Así, 12=2*2*3, y esta descomposición es única. Si el 1 se considerase como primo, obviamente esta propiedad deja cumplirse dado que podemos multiplicar por uno tantas veces como queramos sin cambiar el resultado final del producto. En este libro, sin embargo, para hacer menos engorroso este ejemplo, hacemos notar que la función tal como está codificada considerará al 1 como primo

que en el caso anterior, el bucle cumple la función de recorrer todos los posibles candidatos que pueden cumplir una condición, en este caso ser divisor del argumento `numero`. Así, basta con que la condición en el cuerpo del bucle se cumpla una vez para que no haga falta continuar el bucle, y podamos asegurar que el número no es primo devolviendo por tanto un cero.

```
% ud3_fesprimo
% ud3_fesprimo(numero) devuelve 1 si numero es primo
% y 0 si no lo es
% orden break.
function primo=ud3_fesprimo(numero)
primo=1;
i=2;
while i<=sqrt(numero)
    if mod(numero,i)==0
        primo=0;
        break;
    end
    i=i+1;
end
```

Es interesante notar que cualquier número no primo tiene divisores menores o iguales que su raíz cuadrada. Si un número no los tiene es que es primo. Por eso el índice del bucle solo llega hasta la raíz del número analizado. Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 3.50 *Prueba la función* `ud3_fesprimo`.

Ejercicio 3.51 *Codifica una función que reciba un número natural n y devuelva su divisor de valor máximo, excluyendo al propio n. Por ejemplo, si $n = 27$, su divisor de valor máximo es 9. Para ello, se montará un bucle que arranque en $n - 1$ y vaya bajando hasta que encuentre un divisor de n, rompiendo el bucle en ese momento y devolviendo ese valor.*

Ejercicio 3.52 *Ídem más pequeño y distinto de 1. Si no hay ninguno, devolverá 0.*

Ejercicio 3.53 *Codifica una función que reciba dos enteros m, n, con $m < n$ por hipótesis, y devuelva el menor número entero r entre m y n (ambos inclusive) tal que $\sin(r) > 0$ (se usará* `break`*). Si no hubiese ninguno, la función devolverá 0. Por ejemplo, si $m = 16$ y $n = 27$ el resultado debe ser 19 ya que $\sin(16) = -0.2$, $\sin(17) = -0.9$, $\sin(18) = -0.7$, y $\sin(19) = 0.1$.*

Ejercicio 3.54 *Ídem sin* `break`.

Ejercicio 3.55 *Codifica una función que reciba un natural n y devuelva el mayor número entero positivo $m < n$ tal que $\sin(m) < 0$. (se usará* `break`*). Se supondrá que existe solución. Por ejemplo, si $n = 29$ el resultado ha de ser 25.*

Ejercicio 3.56 *Ídem sin* `break`.

Ejercicio 3.57 *Utilizando la función* `ud3_fcuentadiv`, *construye una función que haga lo mismo que* `ud3_fesprimo`.

Ejercicio 3.58 *Codifica una función que calcule el número de primos menores o iguales que un número natural n, llamando a* `ud3_fesprimo`. *Por ejemplo, si $n = 10$, los primos menores o iguales que n son 2,3,5 y 7. La función devolvería por tanto 4.*

Ejercicio 3.59 *Codifica una función que calcule la suma de de los primos menores o iguales que un número natural n. Por ejemplo, si $n = 10$, los primos menores o iguales que n son 2,3,5 y 7, cuya suma es 17, que es lo que debe devolver la función.*

Ejercicio 3.60 *Codifica una función que reciba dos naturales estrictamente positivos por hipótesis, m y n, con $m < n$ por hipótesis, y devuelva la suma de los números primos entre m y n, ambos inclusive. Por ejemplo si $m = 17$ y $n = 27$, los primos entre ambos son 17, 19 y 23, cuya suma es 59.*

Ejercicio 3.61 *Codifica una función que reciba dos naturales estrictamente positivos por hipótesis, m y n, con $m < n$ por hipótesis, y devuelva el producto de los números primos entre m y n. Por ejemplo si $m = 17$ y $n = 27$, los primos entre ambos son 17, 19 y 23, cuyo producto es 7429.*

Ejercicio 3.62 *Codifica una función que reciba dos naturales, m y n, con $m < n$ por hipótesis, y devuelva el mayor número primo entre m y n, ambos inclusive. (Para valientes) Hacerlo sin usar* `break`. *Si no hubiere ningún primo, la función devolverá 0. Por ejemplo si $m = 17$ y $n = 27$, los primos entre ambos son 17, 19 y 23 siendo el más grande el 23.*

Ejercicio 3.63 *Codifica una función que reciba dos naturales, m y n, con $m < n$ por hipótesis, y devuelva el menor número primo entre m y n, ambos inclusive. (Para valientes) Hacerlo sin usar* `break`. *Si no hubiere ningún primo, la función devolverá 0. Por ejemplo si $m = 17$ y $n = 27$, los primos entre ambos son 17, 19 y 23 siendo el más pequeño el 17.*

Ejercicio 3.64 *(Para valientes) Codifica una función que reciba un número num y devuelva una variable* `flag` *que ha de valer 1 si num descompone en producto de dos primos y 0 si no es así. Por ejemplo, para $num = 6 = 3 \cdot 2$ devolverá 1 y para $num = 7$ o $num = 12$ devolverá 0.*

Ejercicio 3.65 *(Para los más valientes) Este ejercicio está inspirado en la estupenda novela de Mark Haddon, The curious incident of the dog in the night-time, publicada en el año 2003. Codifica una función que reciba un número n y devuelva el mayor primo menor o igual que n terminado en 3 tal que el primo anterior termine en 1 y tal que la diferencia entre ambos sea 2. Por ejemplo, si $n = 55$, la respuesta sería 43. Si no hay ninguno, devolverá 0.*

Ejercicio 3.66 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

3.6. Bucles en los que la condición no se refiere a un índice

Como ya se dijo en la sección 3.2.2, la gran ventaja de los bucles escritos mediante el uso de la sintaxis `while` es que el índice del bucle es una variable más del programa. El bucle se realiza mientras la condición detrás del `while` siga siendo cierta y en los ejemplos estudiados hasta ahora esa condición ha estado relacionada con el valor del índice del bucle. Aunque hasta ahora hemos trabajado así, dicha condición no tiene por qué estar asociada al valor que tome un determinado índice sino que puede referirse a valores tomados por otras variables, funciones o combinaciones de ambas.

Un buen ejemplo para ilustrar esta idea es una función que cuente los dígitos de un número natural. Para ello dividiremos el número por 10 mientras el resultado sea mayor o igual que 1. El número de divisiones realizadas nos permite saber cuántos dígitos tiene el número.

```
% ud3_fdigitos.m
% Cuenta los digitos de n natural mayor que 0.
% Bucle con condicion no relativa a un indice.
function cont=ud3_fdigitos(n)
cont=0;
while n>=1
    n=n/10;
    cont=cont+1;
end
```

Otro ejemplo, un poco más complicado, es contar los elementos de la sucesión de Fibonacci menores que cierto número r. La sucesión estará iniciada por x_1 y x_2, supuestos estrictamente positivos y menores que r. A priori, no se sabe cuántos términos de la sucesión van a cumplir esa propiedad, así que no se puede crear un bucle con un índice de 1 a un determinado n con la seguridad de que vamos a recorrer todos los términos posibles menores que r (hay que tener en cuenta que x_1 y x_2 pueden ser números tan pequeños como queramos, por ejemplo, 0.0023 y 0.0001). La manera de resolver este problema es implementar esa condición directamente en el `while` del bucle. Es decir, el bucle estará corriendo mientras que los elementos de la sucesión sean menores que r. Concretamente:

```
% ud3_ffibonacci.m
% devuelve el numero de términos de la sucesion de
% Fibonacci menores que r
% (iniciada por x1, x2 positivos, con x1<r, x2<r)
function cont=ud3_ffibonacci(x1,x2,r)
x3=x1+x2;
cont=2;
while x3<r
    cont=cont+1;
```

```
    x1=x2;
    x2=x3;
    x3=x1+x2;
end
```

A continuación se presentan algunos ejercicios que inciden en la misma idea que el ejemplo expuesto:

Ejercicio 3.67 *Prueba las funciones* `ud3_fdigitos` *y* `ud3_ffibonacci` *con algunos valores y comprueba que los resultados son correctos.*

Ejercicio 3.68 *Codifica una función que reciba un número $num \geq 2$, y devuelva el primer elemento x_n de la sucesión de Fibonacci $x_n = x_{n-1} + x_{n-2}$, con $x_1 = 1 = x_2 = 1$, tal que x_n sea mayor o igual que num. Por ejemplo, si $num = 18$, la función devolverá 21, pues los términos de la sucesión de Fibonacci son 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., y el primero mayor o igual que 18 es 21.*

Ejercicio 3.69 *Escribe una función que reciba un número natural n, con $n \geq 2$ por hipótesis, y calcule la media aritmética de los elementos de la sucesión de Fibonacci (con $x_1 = 1$ y $x_2 = 1$) que son menores o iguales que n. Por ejemplo, si $n = 15$, los elementos de la sucesión de Fibonacci menores que 15 son 1, 1, 2, 3, 5, 8 y 13 y su media es*

$$(1 + 1 + 2 + 3 + 5 + 8 + 13)/7 = 33/7 = 4.7143$$

Ejercicio 3.70 *Crea una función que reciba un número natural n, con $n \geq 2$ por hipótesis, y devuelva el número de elementos de la sucesión de Fibonacci (con $x_1 = 1$ y $x_2 = 1$) que son primos y menores que n. Por ejemplo, si $n = 100$ la función debería devolver 5 ya que los elementos de la sucesión de Fibonacci menores que 100 son 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 y 89 y de ellos solamente 2, 3, 5, 13 y 89 son primos. La función devolverá por tanto 5.*

Ejercicio 3.71 *(Para valientes) Codifica una función que reciba tres números estrictamente positivos p, q y r, y devuelva una variable* `flag` *que ha de valer 1 si r pertenece a la sucesión de Fibonacci iniciada por p y q y 0 en caso contrario. O sea:*

$$x_n = x_{n-1} + x_{n-2}, \quad x_1 = p, \; x_2 = q$$

Por ejemplo, si $p = 0.2$, $q = 0.5$ y $r = 1.2$, la respuesta es 1, mientras que si $r = 1.5$, la respuesta ha de ser 0.

Ejercicio 3.72 *Codifica una función que reciba dos números naturales m y n, con $1 < m < n$ por hipótesis, y que cuente el número de veces que m divide a n. Si m no es divisor de n la función deberá devolver 0. Por ejemplo, si $m = 2$ y $n = 8$, el resultado es 3. Si $m = 2$ y $n = 9$, el resultado es 0.*

Ejercicio 3.73 *Codifica una función que reciba un número natural y devuelva el número de ceros que hay en él (así, por ejemplo, en 304 hay un cero, en 30 hay un cero y en 115 no hay ningún cero) (solución en apéndice).*

Ejercicio 3.74 *Dado un número natural n escribir una función que halle la media aritmética de los dígitos de n. Es decir, si $n = 204$ entonces la función ha de devolver $(2+0+4)/3 = 2$.*

Ejercicio 3.75 *(Para valientes) Construir una función que reciba dos naturales m, n, con m entre 0 y 9 ambos inclusive. La función devolverá el número de veces que aparece m entre los dígitos de n. Por ejemplo, si $n = 304$, y $m = 0$ el resultado será 1, pero si $m = 7$, el resultado será 0.*

Ejercicio 3.76 *Hay números de 5 dígitos del tipo $abbbb$ tales que cuando los elevas al cuadrado y les sumas o restas 1 queda un número que tiene 10 dígitos todos diferentes entre sí. Por ejemplo*

$$(85555)^2 - 1 = 7,319,658,024$$

$$(97777)^2 - 1 = 9,560,341,728$$

Codifica una función que reciba dos naturales a, b, con $0 \leq a, b \leq 9$, y compruebe si el número $abbbb$ es de ese tipo, devolviendo una variable `flag` *que valga 1 o 0 dependiendo de que lo sea o no.*

Ejercicio 3.77 *Considérese la serie del ejercicio 3.24 cuya suma es π. Es decir, las sumas parciales*

$$S_n = 4\left(1 - \frac{1}{3} + \frac{1}{5} + \cdots + (-1)^n \frac{1}{2n+1}\right)$$

son aproximaciones del número π que van mejorando a medida que n es más grande.

Escribe una función que dado un valor err devuelva el número $aprox$ que será la primera suma parcial S_n tal que aproxime a π con un error menor err es decir $|S_n - \pi| < err$. Por ejemplo, si $err = 0.15$, dado que los términos de la sucesión S_n son 4.0000, 2.6667, 2.8952, 3.3397, 2.9760, 3.2837, 3.0171, 3.2524, la función devolverá 3.2837, pues es el primer término para el que $|3.2837 - \pi| < 0.15$.

Ejercicio 3.78 *Codifica una función que reciba un valor real positivo $prec$, obtenga términos de la sucesión*

$$x_{i+1} = \sqrt{2x_i}, \quad x_1 = 8.32, \quad i \in \mathbb{N}$$

y devuelva el primer x_i para la que $|x_i - x_{i-1}| < prec$. Por ejemplo, en la siguiente tabla se muestran los valores de x_i para $i = 1, \ldots,$ y de los correspondientes $|x_i - x_{i-1}|$. Si la precisión es $prec = 0.05$ la primera vez que $|x_i - x_{i-1}| < 0.05$ es para $i = 7$, así que la función ha de devolver $x_7 = 2.0450$.

i	x_i	$\lvert x_i - x_{i-1}\rvert$
1	8.3200	
2	4.0792	4.2408
3	2.8563	1.2229
4	2.3901	0.4662
5	2.1864	0.2037
6	2.0911	0.0953
7	2.0450	0.0461
8	2.0224	0.0226
⋮	⋮	⋮

Ejercicio 3.79 *Codifica una función que reciba 3 valores reales positivos A, a, $prec$, obtenga términos de la sucesión*

$$x_{i+1} = x_i - \frac{x_i^2 - A}{2x_i}, \quad x_1 = a, \quad i \geq 1$$

y devuelva el primer x_i para la que $|x_i - x_{i-1}| < prec$. Por ejemplo, si $A = 225$, $a = 100$, $prec = 0.5$, en la siguiente tabla se muestran los valores de x_i para $i = 1, \ldots$, y de los correspondientes $|x_i - x_{i-1}|$. La primera vez que $|x_i - x_{i-1}| < 0.5$ es para $i = 6$, así que la función ha de devolver $x_6 = 15.0019$.

i	x_i	$\lvert x_i - x_{i-1}\rvert$
1	100	
2	51.1250	48.8750
3	27.7630	23.3620
4	17.9337	9.8293
5	15.2399	2.6937
6	15.0019	0.2381
⋮	⋮	⋮

Ejercicio 3.80 *Codifica una función que reciba 2 números naturales p y q y genere términos de la sucesión de Fibonacci iniciada por p y q. O sea:*

$$x_{i+2} = x_i + x_{i+1}, \quad x_1 = p, \quad x_2 = q, \quad i \geq 1$$

La función devolverá el primer múltiplo de $p + q$ (distinto de $p + q$). Por ejemplo, si $p = 3$, $q = 5$, la respuesta es 144. Otro ejemplo, si $p = 4$, $q = 15$, la respuesta es 118199.

Ejercicio 3.81 *Crea una función que reciba un número real x y un número natural m y devuelva el primer valor de k para el que suceda que $|\sin(x) - a_k(x)| < 10^{-m}$, con*

$$a_k(x) = \sum_{n=0}^{k} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Por ejemplo, si $x = 0.05$ *y* $m = 10$ *la función debería devolver* 3*, ya que los tres primeros términos de la sucesión verifican:*

$$|\sin(0.05) - a_1(0.05)| = 2.08307e-005 > 10^{-10}$$
$$|\sin(x) - a_2(0.05)| = 2.60401e-009 > 10^{-10}$$
$$|\sin(x) - a_3(0.05)| = 1.55005e-013 < 10^{-10}$$

Por tanto, a_3 *es el primer término cuya diferencia con* $\sin(x)$ *es menor que* 10^{-10}.

Ejercicio 3.82 *Codifica una función que reciba tres números naturales* p*,* q *y* r*. La función calculará términos de la sucesión de Fibonacci iniciada por* p *y* q *y comprobará si son múltiplos de* p*. En el momento en que hayamos encontrado* r *múltiplos de* p *la función terminará, devolviendo el término en que estemos de la sucesión de Fibonacci. No se tendrán en cuenta los valores* p *y* q*. Por ejemplo, si* $p = 3$*,* $q = 4$ *y* $r = 2$*, la sucesión de Fibonacci estaría formada por* $3, 4, 7, 11, 18, 29, 47, 76, 123, ...$ *Como* 18 *y* 123 *son los dos primeros múltiplos de* 3*, el valor a devolver sería* 123*. Si* $p = 2$*,* $q = 1$ *y* $r = 3$*, la función devolvería en este caso* 76*, pues los tres primeros múltiplos de* 2 *serían* $4, 18$ *y* 76*.*

Ejercicio 3.83 *Sea* $\{a_n\}$ *la sucesión de números naturales tal que* $a_1 > 0$*,* $a_2 > 0$ *y definida para* $n \geq 3$ *como*

$$a_n = \begin{cases} \frac{a_{n-1}}{2}, & \text{si } a_{n-1} \text{ es par,} \\ \\ na_{n-2}, & \text{si } a_{n-1} \text{ es impar.} \end{cases}$$

Crea una función que reciba tres números naturales p*,* q *y* m *(siendo* p *y* q *menores que* m*) y devuelva el primer término de la sucesión* $\{a_n\}$ *(con* $a_1 = p$ *y* $a_2 = q$*) que sea mayor que* m*. Por ejemplo, si* $p = 3$*,* $q = 5$ *y* $m = 15$ *entonces la función debería devolver* 20*.*

Ejercicio 3.84 *(Para valientes) El algoritmo de Euclides permite hallar el máximo común divisor de dos números naturales* m *y* n *dividiendo primero el mayor de los dos entre el otro y después recursivamente el divisor entre el resto. El último resto que no es cero es el máximo común divisor. (Si la primera división ya tuviera resto* 0*, entonces es que el máximo común divisor es el menor de los dos números). Por ejemplo, si* $m = 21$ *y* $n = 15$ *el máximo común divisor calculado mediante el algoritmo de Euclides es el siguiente:*

$$21 = 1 \cdot 15 + 6$$
$$15 = 2 \cdot 6 + 3$$
$$6 = 2 \cdot 3 + 0$$

El máximo común divisor de 21 *y* 15 *es* 3*, el último resto no nulo. Escribe una función que dados dos naturales, calcule el máximo común divisor mediante el algoritmo de Euclides.*

Ejercicio 3.85 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

Capítulo 4

Vectores

4.1. General

En el capítulo 4 se introduce un tipo de variable fundamental en la programación estructurada: los vectores o *arrays* de datos. No hay apenas órdenes nuevas en este capítulo y nos concentramos en desarrollar algoritmos más interesantes que los trabajados hasta ahora, que permiten resolver problemas más complejos y similares a los reales. Los vectores están muy relacionados con los bucles que hemos estudiado en el capítulo 3 pues son los bucles los que nos permiten recorrer los vectores. Se trata en este capítulo de que os expreséis con soltura en el lenguaje que ya habéis aprendido, y utilizamos los vectores como candidatos potenciales para codificar algoritmos interesantes y que utilizan todas las herramientas descritas en los capítulos anteriores. De este modo, este capítulo será sencillo y natural para los que habéis progresado correctamente hasta este punto. Servirá también para que los que han tenido más dificultades, puedan ver los temas anteriores desde una perspectiva diferente, la cual les permitirá comprenderlos mucho mejor.

4.2. Vectores como argumentos de funciones

Este ejemplo es la primera función en la que se usa un vector, y de lo que se trata es de calcular la media de sus elementos. Además, ese vector es en si mismo el argumento de entrada de la función, lo cual no requiere ninguna sintaxis específica.

Como vimos en el capítulo 1, podemos preguntar a un vector o a una matriz su tamaño. Para ello disponemos de la orden `length`. Esto no es en general así si usamos otro lenguaje de programación, ya que en principio, además del vector, habría que pasar como argumento un número natural correspondiente a su dimensión. En MATLAB no es necesario pero la primera línea de la función es una asignación de ese valor a una nueva variable n que nos va a permitir montar el bucle para movernos por el vector. Podéis, por ejemplo, ejecutar la siguiente sentencia, desde la línea de comandos para entender mejor el funcionamiento de la orden `length`. Nótese que los vectores van entre corchetes con los elementos del mismo

separados por un espacio y no por comas.

```
>> length([-5.2 3.7 12.98 1.1])
ans =
     4
```

Para obtener la media tenemos que sumar todos los elementos del vector. Para ello, inicializamos a 0 la variable suma y mediante un bucle la vamos incrementando con los elementos del vector. Finalmente dividimos suma por el número de elementos del vector (obtenidos mediante la función length, como hemos mencionado antes) para tener la media.

```
% ud4_fmedia.m
% ud4_fmedia(v) recibe un vector v y
% devuelve el valor medio de los elementos de v
% Con esta funcion se introduce el uso de vectores
function m=ud4_fmedia(v)
n=length(v);% n es la dimensión o número de
            % elementos del vector
suma=0;     % Inicializamos la suma a 0
i=1;
while i<=n
   suma=suma+v(i); % Sumamos el elemento i-esimo de v
   i=i+1;
end
m=suma/n; % Una vez sumados, hallamos la media
```

Un ejemplo de cómo invocar a esta función es el siguiente:

```
>> ud4_fmedia([-5.2 3.7 12.98 1.1])
ans =
    3.1450
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 4.1 *Crea una carpeta llamada* ud4 *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 4.*

Ejercicio 4.2 *Prueba la función* ud4_fmedia *para diferentes vectores. En particular, pruébala para el vector* $(-1.2, 4.3, 5.8)$

Ejercicio 4.3 *Crea una función que reciba un vector y devuelva la media de los cuadrados de los elementos del vector. ¿Coincidirá el resultado con la media al cuadrado?. Por ejemplo, si el vector es* $(5, 3, 4)$*, la media de sus cuadrados es 16.6667.*

Ejercicio 4.4 *Crea una función que reciba un vector (los elementos serán positivos por hipótesis) y devuelva la media geométrica de sus elementos. La media geométrica de* $\{a_1, \ldots, a_n\}$ *es* $\sqrt[n]{a_1 a_2 \ldots a_n}$. *Por ejemplo, si el vector es* $(7, 2, 1, 4)$*, la media geométrica es* $\sqrt[4]{7 \cdot 2 \cdot 1 \cdot 4} = 2.7356$.

Ejercicio 4.5 *La siguiente función recibe un vector* $\mathbf{v}$ *y una posición* pos *y devuelve la diferencia entre el primer elemento y el elemento que ocupa la posición* pos. *¿Es correcta? ¿Podrías hacerla más eficiente? (Pista: no es necesario usar un bucle ni un condicional)*

```
function dif=ud4_fdiferencia(v,pos)
n=length(v);
i=1;
while i<=n
    if i==pos
        dif=v(1)-v(i);
    end
    i=i+1;
end
```

UNIVERSIDAD TECNICA DEL NORTE BIBLIOTECA

Ejercicio 4.6 *(Muy importante) Crea una función que reciba un número* num *y un vector* $\mathbf{u}$ *y devuelva el número de veces que aparece ese número en el vector. Por ejemplo, si el vector es* $\mathbf{u} = (5, 9, 9, 7)$ *y el número es* $num = 9$*, la función devolverá 2, pero si* $\mathbf{u} = (15, 19, 12, 27)$ *la función devolverá 0.*

Ejercicio 4.7 *Crea una función que reciba un vector de números naturales y devuelva el número de elementos del vector que sean múltiplos de tres. Por ejemplo, si el vector es* $(5, 9, 12, 7)$*, la función devolverá 2.*

Ejercicio 4.8 *Crea una función que reciba un vector y calcule la media de los elementos estrictamente positivos del vector. En lugar de sumar todos los elementos, tendrás que comprobar si un elemento es positivo o no antes de sumarlo. No se podrá llamar a ninguna función. Si no hay ningún positivo, la función devolverá 0. Por ejemplo, si el vector es* $(5, -9, 12, 7)$*, la media de los positivos es* $(5 + 12 + 7)/3 = 8$ *(solución en apéndice).*

Ejercicio 4.9 *Crea una función que reciba un vector* $\mathbf{u}$ *y devuelva* 1 *si todos los elementos del vector son estrictamente positivos y* 0 *en caso contrario. Por ejemplo, si* $\mathbf{v} = (7, 4, 0, 2)$*, la función devolverá 0, y si* $\mathbf{u} = (7, 4, 7, 2)$*, la función devolverá 1 (solución en apéndice).*

Ejercicio 4.10 *Crea una función que reciba un vector de naturales* $\mathbf{u}$ *y un natural* d *y devuelva 1 si todos los elementos de* $\mathbf{u}$ *son divisibles por* d *y 0 en caso contrario. Utilizar la sentencia* break. *Por ejemplo, si el vector es* $\mathbf{u} = (5, 9, 12, 7)$ *y el valor* $d = 3$*, la función devolverá 0, pero si* $\mathbf{u} = (15, 9, 12, 27)$ *la función devolverá 1.*

Ejercicio 4.11 *Codifica una función que reciba un vector* $\mathbf{u}$, *y calcule si hay mayor número de componentes estrictamente positivas que de componentes estrictamente negativas. Si así fuere devolvería la media de las positivas. Si fuese del otro modo, devolvería la media de las negativas. Si hay igual número de elementos positivos y negativos o son todos cero, devolvería 0. Por ejemplo, si* $\mathbf{u} = (3, -6, -5, -11, 4)$, *hay más negativos que positivos y la función devolvería (-6-5-11)/3=-7.333*

Ejercicio 4.12 *Codifica una función que reciba un vector* $\mathbf{v}$ *de números naturales, un natural* n *y que devuelva* 1 *si el producto de las componentes del vector* $\mathbf{v}$ *es igual a* n, *y* 0 *en caso contrario. Por ejemplo si* $\mathbf{v} = (1, 2, 3)$ *y* $n = 6$ *la función debería devolver* 1 *ya que* $6 = 1 \cdot 2 \cdot 3$. *Sin embargo tomando el mismo vector* $\mathbf{v}$ *y* $n = 8$ *la función debería devolver* 0.

Ejercicio 4.13 *Crea una función que reciba un vector* $\mathbf{u}$ *de naturales y un natural* n *y devuelva 1 si la suma de las componentes de* $\mathbf{u}$ *que sean múltiplos de* n *es par y 0 en caso contrario. Por ejemplo, si* $\mathbf{u} = (4, 5, 6, 7)$ *y* $n = 2$, *la suma de los múltiplos de* n *en* $\mathbf{u}$ *es 10, par, y devolverá un 1.*

Ejercicio 4.14 *Escribe una función que reciba un vector* $\mathbf{v}$ *de elementos naturales y que devuelva una variable* `flag` *que ha de valer* -1 *si el número de elementos impares es mayor que el de elementos pares,* 0 *si son iguales y* 1 *si el número de elementos pares es mayor que el de impares. Por ejemplo si* $v = (4, 3, 24, 6)$ *la función devolverá* `flag=1`, *si* $v = (1, 3, 13, 24, 4)$ *la función devolverá* `flag=-1` *y si* $v = (1, 13, 7, 24, 4, 2)$ *la función devolverá* `flag=0`.

Ejercicio 4.15 *Crea una función que reciba un vector* $\mathbf{t}$ *y un número* x *y evalué en* x *el polinomio*

$$(x - t_1)(x - t_2) \dots (x - t_n),$$

donde n *es la dimensión del vector* $\mathbf{t}$. *Es decir, evalúa en* x *un polinomio cuyo coeficiente de grado máximo vale* 1 *y cuyas raíces son los elementos de* $\mathbf{t}$. *Por ejemplo, si* $\mathbf{t} = (1, 3)$ *y* $x = -0.5$, *el resultado es 5.25.*

Ejercicio 4.16 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.3. Funciones que llaman a funciones con argumentos vectores

Del mismo modo que en el ejemplo 2.7 estudiamos una función que llamaba a otra función, en este ejemplo vemos una función que recibe un vector y llama a otra función que recibe un vector, la que acabamos de estudiar, `ud4_fmedia`. La sintaxis no ofrece ninguna dificultad. En este ejemplo concretamente se calcula la varianza de un vector, que es una medida de la dispersión de los elementos del vector. Si los elementos del vector son similares, la varianza será pequeña y si son muy diferentes, la varianza será grande.

```
% ud4_fvarianza.m
% ud4_fvarianza(v) recibe un vector v y
% devuelve la varianza de los valores contenidos en el vector v
% Se llama a una función que recibe un vector (ud4_fmedia)
function var=ud4_fvarianza(v)
m=ud4_fmedia(v);
n=length(v);
suma2=0;
i=1;
while i<=n
    suma2=suma2+(v(i)-m)^2;
    i=i+1;
end
var=suma2/n;
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 4.17 *Prueba la función* `ud4_fvarianza` *para diferentes vectores.*

Ejercicio 4.18 *Crea una función que reciba un vector de números naturales y utilizando la función* `ud3_fesprimo`*, devuelva el número de elementos del vector que son primos. Por ejemplo, si* $\mathbf{v} = (9, 14, 11, 7, 12)$*, la función devolverá 2.*

Ejercicio 4.19 *Repite el ejercicio 4.18 sin utilizar* `if`*.*

Ejercicio 4.20 *Crea una función que reciba un vector de números naturales y utilizando la función* `ud3_fesprimo` *y sin utilizar* `if` *devuelva 1 si todos los elementos del vector son primos y 0 en caso contrario. Por ejemplo, si el vector es* $(3, 11, 16, 4, 7, 5, 12)$*, la función devolverá 0 y si es* $(11, 7, 5, 13)$*, devolverá 1.*

Ejercicio 4.21 *Codifica una función que reciba un vector de naturales* $\mathbf{u}$ *y devuelva el primer número primo que aparezca en el vector. Caso de que no haya ninguno, devolverá 0. Por ejemplo, si* $\mathbf{u} = (9, 14, 11, 7, 12)$*, la función devolverá 11.*

Ejercicio 4.22 *Crea una función que reciba un vector de números naturales positivos y devuelva el primer número NO primo que aparezca en el vector. En caso de que no haya ninguno, devolverá* 0*. Por ejemplo, si el vector es* $(3, 11, 16, 4, 7, 5, 12)$*, la función devolverá 16. El 1 se considerará primo.*

Ejercicio 4.23 *Crea una función que reciba un vector* $\mathbf{u}$ *y devuelva el primer valor menor que la media. Por ejemplo, si el vector es* $(12, 7, 5)$*, la media es 8 y el primer valor menor que la media es 7.*

Ejercicio 4.24 *Codifica una función que reciba un vector* $\mathbf{v}$ *de números naturales, un natural* n *y llamando a la función* `ud3_fcuentadiv` *devuelva 1 si hay algún elemento del vector* $\mathbf{v}$ *que tenga igual número de divisores que* n *(ni más ni menos) y 0 en caso contrario. Para ello, se barrerá el vector* $\mathbf{v}$ *y se verá si algún elemento tiene tantos divisores como* n*, rompiendo en ese caso el bucle y devolviendo 1. Así, si* $v = (7, 25, 21, 9)$ *y* $n = 15$*, devolvería 1 porque el 21 y el 15 tienen el mismo número de divisores, 4. Si* $v = (7, 25, 31, 9)$ *y* $n = 15$ *devolverá 0 porque ninguno de los elementos del vector* $\mathbf{v}$ *tiene 4 divisores.*

Ejercicio 4.25 *Escribe una función que reciba un vector de números naturales distintos y devuelva la suma de las posiciones donde se encuentran los números primos del vector, en caso de no haber ningún primo devolvería cero. Así por ejemplo si el vector es* $v = [2, 4, 14, 9, 3, 17]$*, los números primos están colocados en las posiciones 1,5 y 6, y por lo tanto debería devolver* $1 + 5 + 6 = 12$ *(solución en apéndice).*

Ejercicio 4.26 *Crea una función que reciba un vector de* $\mathbf{u}$ *números naturales y un numero* x *y que devuelva una variable* `flag=1` *si la media de los primos de* $\mathbf{u}$ *es estrictamente mayor que* x *y 0 en caso contrario. Por ejemplo, si el vector es* $u = (2, 4, 14, 9, 3, 13)$ *y* $x = 3.4$*, la media de los primos es 6, y por tanto la función devolverá un 1.*

Ejercicio 4.27 *Crea una función que reciba un vector de* $\mathbf{u}$ *números naturales y un natural* pos *menor que la dimensión de* $\mathbf{u}$*. La función devolverá una variable* `flag` *que ha de valer 1 si la media de los primos de* $\mathbf{u}$ *es estrictamente mayor que* $u(pos)$ *y 0 en caso contrario. Por ejemplo, si el vector es* $u = (2, 4, 14, 9, 3, 13)$ *y* $pos = 2$*, la media de los primos es 6, y por tanto la función devolverá un 1.*

Ejercicio 4.28 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.4. Cálculo de extremos

Los algoritmos de búsqueda de extremos son fundamentales en programación y el más básico de todos ellos es el de encontrar el máximo de una serie o vector. El algoritmo más sencillo consiste en suponer que el máximo es el primer elemento de la serie y se inicializa con ese valor una variable `maxi` que será la que almacene el máximo al final. A continuación se van barriendo los demás elementos y en el momento en que encontramos un valor mayor que el que tenemos como referencia, `maxi`, asignamos a `maxi` ese valor.

```
% ud4_fmaximo.m
% ud4_fmaximo(v) recibe un vector v y devuelve
% la mayor de sus componentes
% El algoritmo de busqueda de maximo es fundamental en el curso.
function maxi=ud4_fmaximo(v)
n=length(v);
maxi=v(1);
i=2;
while i<=n
    if v(i)>maxi
        maxi=v(i);
    end
    i=i+1;
end
```

Es importante, muy importante, entender todos los detalles de este ejemplo. MATLAB dispone de una función que realiza esta misma tarea, `max` (idem `min` para el mínimo), pero ello no es óbice para estudiar el algoritmo, el cual es fundamental. Los ejercicios correspondientes a este ejemplo son especialmente interesantes reflejando la importancia del algoritmo explicado.

Ejercicio 4.29 *Prueba la función* `ud4_fmaximo` *para diferentes vectores.*

Ejercicio 4.30 *Crea una función que reciba un vector y devuelva el mínimo de los elementos del vector.*

Ejercicio 4.31 *Crea una función que reciba un vector* $\mathbf{v}$*, cambie de signo todos los elementos del vector, calcule el máximo del nuevo vector y lo devuelva cambiado de signo. ¿Qué valor has obtenido?*

Ejercicio 4.32 *Crea una función que reciba un vector* $\mathbf{v}$*, y calcule la mayor diferencia* $v_{i+1} - v_i$*, entre dos elementos consecutivos del vector. Por ejemplo, si el vector es* $\mathbf{v} = (4, 7, -2, 9, 8)$*, el resultado sería 11.*

Ejercicio 4.33 *La siguiente función recibe un vector y debería devolver la mayor diferencia entre dos términos consecutivos (la diferencia de un elemento menos el anterior). Pruébala con los vectores* $(1,1,1)$*,* $(1,2,1)$*,* $(1,2,4)$ *y* $(4,2,1)$ *(debería devolver 0, 1, 2 y -1 respectivamente). Encuentra los errores.*

```
function y=ud4_fprueba3(v)
n=length(v);
y=0;
while i<=n
    if abs(v(i)-v(i-1))>y
        y=v(i)-(v(i)-1);
    end
end
```

Ejercicio 4.34 *¿Qué valor devuelve la siguiente función (*v *es un vector de números naturales)?*

```
function y=ud4_fprueba(v)
n=length(v);
i=1;
y=v(1);
while i<=n
  if ud3_fesprimo(i)==1 & v(i)>y
    y=v(i);
  end
  i=i+1;
end
```

Ejercicio 4.35 *Codifica una función que reciba un número natural n y devuelva el número estrictamente menor que n que tiene mayor número de divisores impares. Caso de que haya varios con el mismo número de divisores impares, devolverá el mayor. No se podrán usar vectores. Por ejemplo, si $n = 22$, la respuesta es 21, dado que tiene 4 divisores impares (21, 7, 3, 1). 15 también tiene 4 divisores impares (15, 5, 3, 1) pero es menor que 21.*

Ejercicio 4.36 *Codifica una función que reciba un número natural n y devuelva el número estrictamente menor que n que tiene mayor cantidad de divisores primos. Para ello se construirá primero una función auxiliar* `faux`*, que devuelva el número de divisores primos de uno dado (el número uno no se considerará primo), la cual se invocará desde la función pedida. Se llamará también del modo que se considere conveniente a la función* `ud3_fesprimo`*, ya estudiada. Caso de que haya varios números con la misma cantidad de divisores primos, se devolverá el menor. No se podrán usar vectores. Por ejemplo, si $n = 61$, la respuesta es 30, dado que tiene como factores primos el 2, el 3 y el 5 (solución en apéndice).*

Ejercicio 4.37 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.5. Cálculo de extremos utilizando un vector auxiliar

A menudo, utilizar un vector auxiliar para calcular los extremos facilita la escritura y clarifica un determinado código. Esto suele estar asociado a que busquemos no el extremo de nuestro vector original sino de otro obtenido a partir de este. Como ejemplo vamos a encontrar el mínimo de las diferencias en valor absoluto entre un elemento de un vector y su media, para lo cual crearemos un vector auxiliar que contenga dichas diferencias en valor absoluto, para posteriormente calcular su mínimo como se explicó en la sección 4.4.

```
% ud4_fvauxextremo.m
% Se crea vector auxiliar para calcular extremo.
% Mínima distancia a la media.
function dismin=ud4_fvauxextremo(v)
m=ud4_fmedia(v);
n=length(v);
% creo el vector auxiliar
i=1;
while i<=n
    vaux(i)=abs(v(i)-m);
    i=i+1;
end
% ya tengo el vector auxiliar
dismin=vaux(1);
i=2;
while i<=n
    if vaux(i)<dismin
        dismin=vaux(i);
    end
    i=i+1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 4.38 *Prueba la función* `ud4_fvauxextremo` *para diferentes vectores. Por ejemplo, si* $v = (3, 2, 1, 7)$, *la media vale 3.25 y la diferencia mínima es* $|3 - 3.25| = 0.25$.

Ejercicio 4.39 *Crea una función que reciba un vector* v *y devuelva el mínimo de las diferencias entre un elemento de* v *y la media de* v. *Por ejemplo, si* $v = (3, 2, 1, 7)$, *la media vale 3.25 y la diferencia mínima es 1-3.25=-2.25.*

Ejercicio 4.40 *Codifica una función que reciba un vector* v *de números naturales, y llamando a la función* `ud3_fcuentadiv` *construya un vector auxiliar* vaux *tal que la componente* i *de* vaux *será el número de divisores de la componente* i *de* v. *La función devolverá el máximo de* vaux. *Así, si* $v = (7, 25, 21, 9)$, *entonces* $vaux = (2, 3, 4, 3)$ *cuyo máximo es 4 que es el valor que devolverá la función (solución en apéndice).*

Ejercicio 4.41 *Codifica una función que reciba un vector* v *de números naturales, y llamando a la función* `ud3_fdigitos` *construya un vector auxiliar* vaux *tal la componente* i *de* vaux *será el número de dígitos de la componente* i *de* v. *La función devolverá el mínimo de* vaux. *Así, si* $v = (7023, 25, 21005, 939)$, *entonces* $vaux = (4, 2, 5, 3)$ *cuyo mínimo es 2 que es el valor que devolverá la función.*

Ejercicio 4.42 *Usando la función creada en el ejercicio 4.6, construir una función que reciba un vector y devuelva el número de veces que aparece el elemento que aparece más veces. Por ejemplo, para* $\mathbf{u} = (0, 3, 7, 3)$ *devolvería 2.*

Ejercicio 4.43 *Crea una función que reciba un vector* $\mathbf{x}$ *y devuelva la mínima distancia entre un elemento de* $\mathbf{x}$ *y los límites del intervalo* $[\overline{\mathbf{x}}-\sigma, \overline{\mathbf{x}}+\sigma]$*, siendo* $\overline{\mathbf{x}}$ *la media de* $\mathbf{x}$ *y* σ *la raíz de la varianza de* $\mathbf{x}$*. Para calcular la distancia se usará la orden* `abs`*. Por ejemplo, si* $\mathbf{x} = (3, 2, 1, 7)$*, la media vale 3.25 y* $\sigma = 2.2776$*. El intervalo es por tanto* $[0.9724, 5.5276]$*. Podemos construir un vector auxiliar que para cada elemento de* $\mathbf{x}$ *almacene el mínimo de la distancia de ese elemento a los extremos del intervalo. Ese vector para el ejemplo vale por tanto* $\mathbf{vaux} = (2.0276, 1.0276, 0.0276, 1.4724)$*, cuyo mínimo es 0.0276, que será el valor que devuelva la función (solución en apéndice).*

Ejercicio 4.44 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.6. Cálculo de posición de extremos

A veces no interesa tanto el valor del extremo como su posición en el vector. En esta función se juega con esa idea, de tal modo que se guarda la posición y su valor correspondiente cuando se supera el valor que ocupa la posición de referencia inicial (la primera) y luego se compara contra el valor de esa posición. Es un algoritmo muy interesante y esencialmente más complicado y difícil de entender que el anterior.

```
% ud4_fimax.m
% ud4_fimax(v) recibe un vector v y devuelve
% la posicion de la mayor de sus componentes
function imax=ud4_fimax(v)
n=length(v);
maxi=v(1); % Tomamos como maximo el primer elemento
imax=1; % y el índice del máximo será 1.
i=2;
while i<=n
  if v(i)>maxi % Si encontramos un elemento mayor
    maxi=v(i); % actualizamos el máximo
    imax=i; % y actualizamos el valor de su posicion
  end
  i=i+1;
end
```

Una variante interesante de ese ejemplo pasa por jugar solo con la posición

```
% ud4_fimaxb.m
% ud4_fimaxb (variante de ud4_fimax)
function imax=ud4_fimaxb(v)
n=length(v);
imax=1; % y el índice del máximo será 1.
i=2;
while i<=n
  if v(i)>v(imax) % Si encontramos un elemento mayor
    imax=i; % y actualizamos el valor de su posicion
  end
  i=i+1;
end
```

Los comandos de MATLAB `max` y `min` ya mencionados en la sección 4.4 permiten también calcular estos índices pero para ello es necesario saber operar con funciones con más de un argumento de salida, algo que se introducirá en la sección 4.11.

Los ejercicios correspondientes a este ejemplo son también bastante interesantes.

Ejercicio 4.45 *Prueba la función* `ud4_fimax.m` *para diferentes vectores.*

Ejercicio 4.46 *Determina qué valor devolverá la siguiente función si* $\mathrm{v} = (1, 3, 2, 5, 4)$*:*

```
function y=ud4_fprueba2(v)
n=length(v);
i=1;
y=1;
while i<=n
    if v(i)>y
        y=i;
    end
    i=i+1;
end
```

Ejercicio 4.47 *Crea una función que reciba un vector* v*, con todas las componentes diferentes entre si por hipótesis, y devuelva la posición del elemento de* v *que esté más cerca de la media. Por ejemplo, si* $\mathrm{v} = (3, 2, 1, 7)$*, la media vale 3.25 y el elemento más próximo a la media es 3, el cual ocupa la posición 1. La función devolverá por tanto 1 en este caso. Cambiad de orden los elementos en este ejemplo para probar el código y comprobar que funciona en todas las circunstancias.*

Ejercicio 4.48 *Crea una función que reciba un vector* v *y devuelva el valor del elemento de* v *que esté más cerca de la media. Por ejemplo, si* $\mathrm{v} = (3, 2, 1, 7)$*, la media vale 3.25 y el elemento más proximo a la media es 3.*

Ejercicio 4.49 *Crea una función que reciba un vector* $\mathbf{v}$ *y devuelva la posición del elemento en el que se dé el mínimo de las diferencias entre un elemento de* $\mathbf{v}$ *y la media de* $\mathbf{v}$. *Por ejemplo, si* $\mathbf{v} = (3, 2, 1, 7)$, *la media vale 3.25 y la diferencia mínima es 1-3.25=-2.25. El índice del elemento cuyo valor es 1 es 3, pues ocupa la tercera posición en el vector.*

Ejercicio 4.50 *(Para valientes) Usando la función creada en el ejercicio 4.6, construir una función que reciba un vector* $\mathbf{u}$ *y devuelva el elemento que aparece más veces. Se supone que siempre habrá uno que aparecerá más veces que todos los demás. Por ejemplo, para* $\mathbf{u} = (0, 3, 2, 3)$ *devolvería 3.*

Ejercicio 4.51 *(Para valientes) Usando la función creada en el ejercicio 4.6, construir una función que reciba un vector y devuelva la primera posición del elemento que aparece más veces. Por ejemplo, para* $(0, 1, 4, 1)$ *devolvería 2.*

Ejercicio 4.52 *Crea una función que reciba un vector de naturales* $\mathbf{v}$, *todos ellos estrictamente positivos por hipótesis, y devuelva el índice del primer número primo que aparezca en el vector. En caso de que no haya ninguno, devolverá* 0. *Por ejemplo, si* $\mathbf{v} = (4, 12, 7, 6, 13, 15)$, *la función devolverá 3.*

Ejercicio 4.53 *Repite el ejercicio 4.52, referido ahora al último elemento del vector.*

Ejercicio 4.54 *Crea una función que reciba un vector* $\mathbf{u}$ *de naturales, todos ellos estrictamente positivos por hipótesis, y devuelva la posición del mayor primo que aparezca en el vector. Se recomienda utilizar inteligentemente la función del ejercicio 4.52 para inicializar el valor de referencia. En caso de que no haya ninguno, devolverá* 0. *Por ejemplo, si* $\mathbf{u} = (20, 11, 23, 17)$, *la función deberá devolver 3.*

Ejercicio 4.55 *Crea una función que reciba un vector* $\mathbf{u}$ *de naturales, todos ellos estrictamente positivos por hipótesis, y devuelva la posición del menor primo que aparezca en el vector. En caso de que no haya ninguno, devolverá* 0. *Por ejemplo, si* $\mathbf{u} = (20, 23, 11, 17)$, *la función deberá devolver 3.*

Ejercicio 4.56 *Escribe una función que reciba un vector de números enteros distintos entre sí y devuelva el segundo número más pequeño. Por ejemplo, si el vector es* $[-7, 2, 0, 8]$ *la función debería devolver* 0 *(solución en apéndice).*

Ejercicio 4.57 *Repetir el ejercicio 4.56 calculando ahora el índice de dicho número. Por ejemplo, si el vector es* $[-7, 2, 0, 8]$ *la función debería devolver* 3.

Ejercicio 4.58 *(Para valientes) Usando la función creada en el apartado 4.6, construir una función que reciba un vector y devuelva la última posición del segundo elemento que aparece más veces. Por ejemplo, para* $[0\,1\,2\,1\,1\,0\,1]$ *devolvería 6.*

Ejercicio 4.59 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.7. Vectores y bucles anidados

A menudo, es necesario anidar bucles (meter un bucle dentro de otro) para resolver determinados problemas con vectores. Por ejemplo si tenemos dos vectores y queremos saber cuantos elementos tienen que sean iguales (supuestos todos distintos en cada uno de los vectores), tendremos que fijar un elemento del primer vector y barrer todos los del segundo mediante un bucle y así sucesivamente con el segundo, tercero, etc., lo que requiere otro bucle exterior a ese. Tener dos bucles exige que cada uno de ellos tenga su propia condición e índice.

```
% ud4_figuales
% numero de elementos iguales entre dos vectores. Se
% supone que todos los elementos en cada vector son distintos.
% primeros bucles anidados con vectores.
%
function iguales=ud4_figuales(u,v)
i=1;
iguales=0;
m=length(u);
n=length(v);
while i<=m
    j=1;
    while j<=n
       if u(i)==v(j)
          iguales=iguales+1;
       end
       j=j+1;
    end
    i=i+1;
end
```

Presentamos los ejercicios asociados a este ejemplo. Son muy interesantes y algunos exigen incluir rupturas de bucles mediante la orden `break`, lo que en el caso de dobles bucles puede implicar tener que romper tanto el bucle externo como el interno.

Ejercicio 4.60 *Prueba la función* `ud4_figuales`.

Ejercicio 4.61 *Codifica una función que reciba un vector* $\mathbf{u}$ *de* m *elementos naturales y calcule, utilizando un doble bucle, la siguiente cantidad:*

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{u(i)} j\right)$$

Este ejercicio se puede hacer utilizando una variable auxiliar para las sumas parciales

$$\sum_{j=1}^{u(i)} j$$

o directamente sumando todo mediante un doble bucle en una única variable. Como ejemplo, si $\mathbf{u} = (3,5,2)$, *la respuesta es* $(1+2+3)+(1+2+3+4+5)+(1+2)=24$

Ejercicio 4.62 *Codifica una función que reciba un vector* $\mathbf{u}$ *y devuelva una variable* `flag` *que ha de valer 1 si* $\mathbf{u}$ *tiene al menos dos elementos iguales y 0 en caso contrario. No se podrá llamar a ninguna función. Por ejemplo, si* $\mathbf{u} = (14, 5, 6, 5, 7)$*, la función devolverá 1, y si* $\mathbf{u} = (14, 5, 6, 8, 7)$ *devolverá 0.*

Ejercicio 4.63 *Codifica una función que dado un vector* $\mathbf{u}$ *y un número* r *y devuelva el número de parejas* $(u(i), u(j))$ *con* $i \neq j$ *de elementos del vector tales que* $u(i)/u(j) = r$*. Así por ejemplo si* $\mathbf{u} = (2, 5, 10, 9, 6, 12)$ *y* $r = 2$*, tenemos que* $10/5 = 2$ *y que* $12/6 = 2$*. Por tanto hay dos parejas y la función devolverá un 2. Se recomienda tener en cuenta con especial cuidado el caso donde alguno de los elementos de* $\mathbf{u}$ *fuese nulo, ya que habría que evitar dividir por cero.*

Ejercicio 4.64 *Codifica una función que reciba un vector* $\mathbf{v}$ *de naturales y devuelva* $\sum v_i!$*. No se podrá llamar a ninguna función y por tanto habrá que usar bucles anidados. Por ejemplo, si* $\mathbf{u} = (5, 3, 6)$*, la respuesta es* $120 + 6 + 720 = 846$*.*

Ejercicio 4.65 *Codifica una función que reciba 3 naturales,* m*,* n*,* p*, con* $m < n$ *y calcule:*

$$\sum_{i=m}^{n}\left(\sum_{j=1}^{p} i^j\right) = m + m^2 + \cdots + m^p + (m+1) + (m+1)^2 + \cdots + (m+1)^p + \cdots + n + n^2 + \cdots + n^p$$

Por ejemplo, si $m = 2$, $n = 4$ y $p = 3$, el resultado es

$$2 + 2^2 + 2^3 + 3 + 3^2 + 3^3 + 4 + 4^2 + 4^3 = 14 + 39 + 84 = 137$$

Otro ejemplo más. Si $m = 1$, $n = 5$, $p = 2$, el resultado será:

$$1 + 1^2 + 2 + 2^2 + 3 + 3^2 + 4 + 4^2 + 5 + 5^2 = 70$$

Ejercicio 4.66 *(Para valientes) Codifica una función que reciba un vector de números naturales y sin llamar a ninguna función, devuelva el número de elementos del vector que son primos. Por ejemplo, si* $\mathbf{u} = (30, 37, 14, 13)$*, la función devolvería 2.*

Ejercicio 4.67 *(Para valientes) Codifica una función que reciba un vector de números naturales y sin llamar a ninguna función, devuelva el primer elemento del vector que sea primo. Si no hay ninguno, que devuelva 0. Por ejemplo, si* $\mathbf{u} = (30, 37, 14, 13)$*, la función devolvería 37 (solución en apéndice).*

Ejercicio 4.68 *Codifica una función que calcule el número de primos menores o iguales que un número natural* n*, sin llamar a ninguna otra función. Por ejemplo, si* $n = 20$*, la función devolverá 8 ya que hay 8 primos menores o iguales que 20, que son 2, 3, 5, 7, 11, 13, 17 y 19.*

Ejercicio 4.69 *Codifica una función que calcule, sin llamar a ninguna otra función, la suma de de los primos menores o iguales que un número natural n. Por ejemplo, si $n = 20$, la función devolverá 77, que es la suma de los primos menores o iguales que 20, que son 2, 3, 5, 7, 11, 13, 17 y 19.*

Ejercicio 4.70 *Codifica una función que reciba dos naturales estrictamente positivos, m y n, con $m < n$ por hipótesis, y, sin llamar a ninguna función, devuelva la suma de los números primos entre m y n, ambos inclusive. Por ejemplo, si $m = 10$ y $n = 20$, la función devolverá 60, que es la suma de los primos entre 10 y 20, los cuales son 11, 13, 17 y 19.*

Ejercicio 4.71 *Codifica una función que reciba dos naturales estrictamente positivos, m y n, con $m < n$ y devuelva el producto de los números primos entre m y n (sin llamar a ninguna otra función). Por ejemplo, si $m = 10$ y $n = 20$, la función devolverá 46189, que es el producto de los primos entre 10 y 20, los cuales son 11, 13, 17 y 19.*

Ejercicio 4.72 *Construye una función que reciba dos naturales m y n con $m < n$ por hipótesis, y devuelva el mayor número primo entre m y n, ambos inclusive, o que devuelva 0 si no hubiera ningún primo entre m y n (incluyendo m y n). No se podrá usar* `ud3_fesprimo` *ni llamar a ninguna otra función ni ejemplo. Por ejemplo, si $m = 4$ y $n = 13$ la función devolverá 13, y si $m = 32$ y $n = 36$, la función devolverá 0.*

Ejercicio 4.73 *Codifica una función que reciba dos naturales, m y n, con $m < n$ por hipótesis, y devuelva el menor número primo entre m y n, ambos inclusive. No se podrá llamar a ninguna función. Si no hubiere ningún primo, la función devolverá 0. Por ejemplo, si $m = 10$ y $n = 20$, la función devolverá 11, que es el menor primo entre 10 y 20.*

Ejercicio 4.74 *Codifica una función que reciba un vector $\mathbf{u}$ de números naturales, todos ellos mayores que uno por hipótesis. Sin llamar a ninguna función, la función devolverá la media de los elementos primos de $\mathbf{u}$. Por ejemplo si $\mathbf{u} = (15, 11, 22, 8, 13, 2)$, los elementos primos son 11,13 y 2 (son 3), y su media será (11+13+2)/3=8.6666.*

Ejercicio 4.75 *Crea una función que reciba un vector $\mathbf{u}$ y devuelva 1 si la suma de dos cualesquiera de sus elementos es igual a un tercero y 0 en caso contrario. Por ejemplo, para $\mathbf{u} = (2, 7, 5, 13)$ devolvería 1 y para $\mathbf{u} = (-1, 2, 5, 13)$ devolvería 0 (solución en apéndice).*

Ejercicio 4.76 *(Para valientes) Codifica una función que reciba un vector $\mathbf{v}$ y devuelva el número de veces que aparece el elemento que más veces aparece. No se podrá llamar a ninguna función. Por ejemplo, para $\mathbf{v} = (0.3, 1.7, 3.2, 1.7)$ devolvería 2.*

Ejercicio 4.77 *(Para valientes) Codifica una función que reciba un vector* v *y devuelva el mayor de los repetidos. No se podrá llamar a ninguna otra función. Pista: Mediante un doble bucle se construirá un vector auxiliar que contenga un valor* `flag` *igual a 0/1 dependiendo de que ese elemento no esté o esté repetido en* v*. Después se buscará en ese vector la primera posición que sea 1. Sea i esa posición. Se tomará como referencia para la búsqueda de máximo el elemento $v(i)$. Se recorrerá* v *a partir de esa posición teniendo en cuenta al comparar con el máximo sólo aquellos elementos de* v *que estén repetidos, para lo cual aparecen como* `flag=1` *en el vector auxiliar. Por ejemplo, si* $\mathrm{v} = (0,7,3,7,15,3,3,2,0)$*, la función ha de devolver 7.*

Ejercicio 4.78 *Codifica una función que reciba un vector* v *y devuelva una variable* `flag` *que ha de valer 1 si* v *tiene exactamente dos elementos iguales y 0 en caso contrario. Por ejemplo, si* $\mathrm{v} = (2,3,4,3)$, `flag` *sería 1. Si* $\mathrm{v} = (2,3,3,3)$ *o* $\mathrm{v} = (2,3,3,2)$, `flag` *sería 0.*

Ejercicio 4.79 *Codifica una función que reciba un vector* v *de números naturales y devuelva el mayor de los máximos comunes divisores (mcd) entre 2 términos del vector que ocupen diferentes posiciones. Para ello, se fijará como referencia el mcd entre la primera y segunda componentes del vector o directamente la unidad. Después se montarán dos bucles anidados que vayan barriendo las demás combinaciones, teniendo cuidado de no hacer el mcd entre una componente y ella misma. Si el mcd fuere mayor que el que tengamos como referencia se sustituirá por este nuevo. Por ejemplo, si $v = (7,20,15,18)$, el mcd máximo entre dos términos cualesquiera es 5. No se podrá llamar a ninguna función (solución en apéndice, utilizando una función auxiliar y sin ella).*

Ejercicio 4.80 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.8. Función que devuelve un vector definido a partir de escalares

A menudo, la salida de nuestras funciones pueden ser también vectores. En MATLAB, la sintaxis no cambia con respecto a la de una variable escalar convencional. En este ejemplo, además ese vector se construye a partir de varios escalares, pues de lo que se trata es de construir un vector con los términos de la sucesión de Fibonacci, la cual ya estudiamos en el ejercicio 3.30.

```
% ud4_fibo
% ud4_fibo(x1,x2,n) devuelve los n primeros
% terminos  de la sucesion de Fibonacci
% x_1=x1, x_2=x2, x_n=x_(n-1)+x_(n-2)
% Por primera vez definimos una funcion que
% devuelve un vector, y lo construimos con
```

```
% argumentos de entrada escalares.
function v=ud4_fibo(x1,x2,n)
v(1)=x1;
v(2)=x2;
i=3;
while i<=n
    v(i)=v(i-1)+v(i-2);
    i=i+1;
end
```

Otro ejemplo interesante, es el de construir una subdivisión equidistante de un intervalo definido por sus dos extremos $xmin$, $xmax$, y por el número de tramos n. La función calculará el valor auxiliar

$$h = \frac{xmax - xmin}{n},$$

y devuelve un vector x de $n+1$ componentes que contiene los puntos entre $xmin$ y $xmax$, ambos inclusive, separados cada dos una distancia h. Así pues,

$$x_1 = xmin, \quad x_2 = xmin + h, \quad \ldots, \quad x_{n+1} = xmin + nh = xmax.$$

```
% ud4_fequidistante
function v=ud4_fequidistante(xmin,xmax,n)
h=(xmax-xmin)/n;
i=0;
while i<=n
    v(i+1)=xmin+i*h;
    i=i+1;
end
```

MATLAB dispone de una función propia que realiza exactamente esta tarea: `linspace`. Los ejercicios correspondientes a estos ejemplos son:

Ejercicio 4.81 *Prueba la función* `ud4_fibo`.

Ejercicio 4.82 *Codifica una función que reciba un número natural n y devuelva un vector* v *tal que sus componentes sean*

$$v_i = \left(1 + \frac{1}{i}\right)^i, \quad 1 \leq i \leq n$$

Llama desde MATLAB a la función. ¿A qué tienden las componentes del vector resultado?.

Ejercicio 4.83 *Codifica una función que reciba un número natural n y devuelva un vector formado por los n primeros términos de la sucesión*

$$x_1 = 2.456, \quad x_{i+1} = \sqrt{2x_i}, \quad i \in \mathbb{N}$$

Comprobar que converge a 2.

Ejercicio 4.84 *Implementar una función que reciba un número natural, por hipótesis, n y devuelva un vector formado por una sucesión en la que cada término se construirá a partir del anterior, siendo n el primero, de tal modo que mientras dicho término sea distinto a la unidad, si es par lo dividirá entre 2, y si es impar lo multiplicará por 3 y le sumará 1. Por ejemplo, si el número introducido es 7, el vector a devolver será: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1*

Ejercicio 4.85 *(Para valientes) Codifica una función que reciba un número natural y devuelva el vector formado por sus cifras. Así, si introducimos* 1423, *devolverá* [1 4 2 3].

Ejercicio 4.86 *(Para los más valientes) Criba de Eratóstenes. La manera más rápida de obtener todos los números primos menores que un número dado n es mediante la Criba de Eratóstenes. El procedimiento es el siguiente: se toman todos los números menores que n y se van eliminando de la lista los múltiplos de dos, los múltiplos de tres y así sucesivamente hasta llegar a los múltiplos de la raíz cuadrada de n. Crea una función que reciba un número n y devuelva todos los números primos menores que n mediante la Criba de Eratóstenes. Como lista de números, usaremos un vector* v *que tenga en la posición i el número i y para marcar el número como eliminado, pondremos un 0 en dicha posición. Este vector* v *compuesto al final de primos y ceros, será el que devolvamos como resultado de la criba y salida de la función (solución en apéndice).*

Ejercicio 4.87 *(Para los más valientes) Para mejorar el algoritmo del ejercicio 4.86, elimina los múltiplos de los números primos entre 2 y $\sqrt{n}$ en lugar de eliminar los múltiplos de todos los números entre 2 y $\sqrt{n}$. Nótese que según vamos aplicando la criba de Eratóstenes vamos obteniendo los números primos, así que podemos utilizar la información guardada en el vector para este caso.*

Ejercicio 4.88 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.9. Funciones que reciben y devuelven vectores

Si queremos que una función MATLAB nos devuelva una expresión o una función matemática evaluada en una serie de valores organizados como un vector, debemos tener la posibilidad de recibir un vector en una función MATLAB y devolver otro vector. La sintaxis no ofrece ningún problema, pero conceptualmente hay que saber moverse a través del vector de partida y generar otro. Ya hicimos algo parecido a esto cuando generamos un vector auxiliar para encontrar extremos de una lista, en la sección 4.5. Vamos a plantear primero un ejemplo sencillo en el que se trata simplemente de codificar una función que reciba un vector de abscisas x y devuelva un vector de ordenadas y tal que para cada abscisa x_i, la ordenada correspondiente y_i sea el máximo de las funciones seno y coseno evaluadas en x_i.

```
% ud4_fmaxsincos
% ud4_fmax_vect(x) recibe un vector x y devuelve el vector
% y tal que cada componente de y es el maximo de las
% funciones seno y coseno evaluadas en cada componente de x
% Primera función que recibe un vector y devuelve otro vector
%
function y=ud4_fmaxsincos(x)
n=length(x);
i=1;
while i<=n
    if sin(x(i))>cos(x(i))
        y(i)=sin(x(i));
    else
        y(i)=cos(x(i));
    end
    i=i+1;
end
```

Para probar esta función podemos ejecutar los siguientes comandos desde la ventana de comandos de MATLAB.

```
>>abscisas=0:0.01:8*pi;
>>ordenadas=ud4_fmaxsincos(abscisas);
>>plot(abscisas,ordenadas);
```

Otra posibilidad sencilla pero también interesante es el de tener una función que reciba un vector y uno o varios escalares y devuelva otro vector. Presentamos este ejemplo en el que se trata simplemente de codificar una función que reciba un vector x y un escalar a. La función devolverá un vector y, tal que la componente y_i de ese vector será 0 si la correspondiente x_i es menor que a y 1 en caso contrario.

```
% ud4_fmenormayor
% ud4_fmenormayor(x,a) recibe un vector x y devuelve el
% vector y tal que cada componente de y vale 0 o 1
% dependiendo de que la correspondiente de x sea menor
% o mayor o igual que a, respectivamente
%
function y=ud4_fmenormayor(x,a)
n=length(x);
i=1;
while i<=n
    if x(i)>=a
        y(i)=1;
    else
        y(i)=0;
    end
    i=i+1;
end
```

Presentamos un ejemplo más que tiene el interés de que llamamos a una función para ir calculando el valor de cada componente del vector resultado. Se trata de codificar una función que reciba un vector de naturales **u** y devolver un vector **vprimos** tal que cada componente de **vprimos** vale uno o cero en función de que la correspondiente componente de **u** sea o no prima.

```
% ud4_fvprimos
% ud4_fvprimos(u) recibe un vector de naturales u y devuelve
% un vector vprimos tal que cada componente de vprimos vale
% uno o cero dependiendo de que la componente correspodiente
% de u sea o no prima.

function vprimos=ud4_fvprimos(u)
n=length(u);
i=1;
while i<=n
    vprimos(i)=ud3_fesprimo(u(i));
    i=i+1;
end
```

La vectorización de operaciones explicada en el tutorial (ver capítulo 1) permite fácilmente obtener un vector a partir de otro, elemento a elemento, con operaciones relativamente complejas, y sin necesidad de utilizar un bucle. Sin embargo, para relaciones más complejas, que impliquen operaciones intermedias o comparaciones complicadas, esa vectorización de operaciones puede ser insuficiente para calcular un vector a partir de otro. En estos casos, será recorrer el vector original mediante un bucle lo que permitirá operar con cada elemento de dicho vector para obtener cada elemento del vector resultado. Un ejemplo de esto es la función que acabamos de ver, `ud4_fvprimos`, la cual no puede ser vectorizada. Sin embargo, la función `ud4_fmaxsincos` sí admite una vectorización que la convierte en una función de una única línea, en la que tomaremos la licencia de utilizar el comando propio de MATLAB `max`:

```
function y=ud4_fmaxsincos_vectorizado(x)
y= max(sin(x),cos(x));
```

En este tema no haremos uso sin embargo de esta técnica, para potenciar el uso de bucles en los ejercicios, dejando estas particularidades de MATLAB para temas sucesivos. Los ejercicios correspondientes a estos ejemplos son:

Ejercicio 4.89 *Prueba las funciones* `ud4_fmaxsincos`, `ud4_fmenormayor` *y* `ud4_fvprimos`.

Ejercicio 4.90 *Codifica una función que reciba un vector* **x** *y devuelva un vector* **y** *tal que* $y_i = x_i^2 - \log(x_i^2 + 1)$. *Cuando la ejecutes desde la línea de comandos, dibuja la curva resultado de modo análogo a como se dibujó la de la función* `ud4_fmaxsincos`, *en la parte teórica de esta sección.*

Ejercicio 4.91 *Crea una función similar a la función* `ud4_fmaxsincos` *que calcule el máximo entre la función del ejercicio 4.90 y el seno. Cuando la ejecutes desde la línea de comandos, dibuja la curva resultado de modo análogo a como se dibujó la de la función* `ud4_fmaxsincos`, *en la parte teórica de esta sección.*

Ejercicio 4.92 *Codifica una función que reciba un vector* $\mathbf{x}$ *y devuelva un vector* $\mathbf{y}$ *tal que para cada índice* i, y_i *sea la aproximación de* $\sin(x_i)$ *dada por el polinomio de Taylor de grado 7 del seno en el cero:*

$$y_i = x_i - \frac{x_i^3}{3!} + \frac{x_i^5}{5!} - \frac{x_i^7}{7!}.$$

Una vez que la hayas invocado desde la línea de comandos, dibuja con MATLAB el seno y su aproximación por la función anterior en $[-\pi/2, \pi/2]$, *con al menos 1000 puntos. Amplía el intervalo caso de que no aprecies ninguna diferencia entre las dos curvas. Por ejemplo, si introducimos el vector* $\mathbf{x} = (1.9, 2.5, 2.8)$ *y* $n = 7$, *el vector devuelto será* $\mathbf{y} = (0.9454, 0.5885, 0.3078)$. *El seno toma en esas mismas abscisas los valores* $(0.9463, 0.5985, 0.3350)$ *(solución en apéndice).*

Ejercicio 4.93 *Codifica una función que reciba un vector de naturales,* $\mathbf{u}$ *y devuelva el vector de los factoriales,* $(u_1!, u_2!, \ldots, u_n!)$. *Representa en una misma gráfica la función factorial entre* 1 *y* 10 *y la función exponencial entre* 1 *y* 10.

Ejercicio 4.94 *Crea una función que reciba un vector* $\mathbf{v}$ *y devuelva un vector con los elementos del vector* $\mathbf{v}$ *pero en orden inverso al original. Por ejemplo, si* $\mathbf{v} = (5, 7, 4, -1)$, *el vector resultado será* $(-1, 4, 7, 5)$.

Ejercicio 4.95 *Crea una función que reciba un vector* $\mathbf{x}$ *y dos valores* a *y* b, *con* $a < b$ *por hipótesis, y devuelva un vector* $\mathbf{y}$ *que es el original truncado entre* a *y* b *(si un valor está entre* a *y* b *no se toca, si es menor o igual que* a *se transforma en* a *y si es mayor o igual que* b *se transforma en* b. *Por ejemplo, si* $\mathbf{x} = (5, 7, 4, -1)$, $a = -0.2$ *y* $b = 5.5$, *el vector resultado será* $\mathbf{y} = (5, 5.5, 4, -0.2)$.

Ejercicio 4.96 *Crea una función que reciba un vector y devuelva los datos escalados entre cero y uno. Para ello determinaremos el valor mínimo del vector. Restamos ese valor al vector para que los datos comiencen en cero. Calculamos el máximo del nuevo vector y dividimos cada elemento del vector por dicho máximo. Al final tenemos que obtener un vector de números entre cero y uno, de modo que el menor sea cero y el mayor uno y las diferencias entre dos elementos del vector original y los correspondientes escalado sean proporcionales.*

Ejercicio 4.97 *(Para valientes) Modifica la función 4.96 para que reciba un vector* $\mathbf{v}$ *y dos valores* a *y* b, *con* $a < b$ *por hipótesis, y devuelva el vector* $\mathbf{v}$ *escalado entre* a *y* b.

Ejercicio 4.98 *Codifica una función que reciba un vector* $\mathbf{u}$ *de naturales y devuelva otro vector con los máximos comunes divisores de las parejas que forman el primer elemento del vector con el resto de los elementos del vector. Por ejemplo, si tenemos el vector* $(2, 4, 12, 3, 15)$, *los máximos comunes divisores de las parejas del 2 con el resto son* $mcd(2,4) = 2$ $mcd(2,12) = 2$ $mcd(2,3) = 1$ $mcd(2,15) = 1$, *por lo tanto la función devolvería un vector* $(2, 2, 1, 1)$.

Ejercicio 4.99 *Codifica una función que reciba un vector* $\mathbf{u}$ *y devuelva un vector* $\mathbf{v}$, *de forma que* $v(i)$ *represente el número de veces que aparece cada elemento* $u(i)$ *en el vector* $\mathbf{u}$. *Por ejemplo si* $\mathbf{u} = (2,4,2,3,3,4,2,2,3,4)$, *entonces el vector* $\mathbf{v} = (4,3,4,3,3,3,4,4,3,3)$ *dado que el* 2 *aparece 4 veces, el* 4 *aparece 3 veces y el* 3 *aparece 3 veces.*

Ejercicio 4.100 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.10. Construcción de vectores

A menudo surgen problemas en los que a partir de un vector es necesario construir un nuevo vector que no tiene la misma dimensión que el original. Además, al construir este nuevo vector a partir del original, a menudo no sabemos el tamaño o dimensión del vector resultante. La gestión de la memoria RAM en estos casos requeriría un cierto cuidado en muchos lenguajes de programación pero MATLAB lo maneja con mucha simplicidad. Así, a medida que vamos definiendo nuevas posiciones del vector, ese vector crece en tamaño y la memoria RAM que se necesita para él es gestionada de modo automático por MATLAB.

Para ilustrar esta idea planteamos un ejemplo muy sencillo pero también muy bonito, y que además sugiere infinidad de ejercicios muy interesantes, los cuales aparecerán a lo largo del resto del curso. Se trata de una función que recibe un vector y devuelve otro vector que contiene únicamente las componentes positivas del vector original, colocadas en el mismo orden en el que están en éste.

Para ir construyendo este segundo vector, de tal modo que cada componente vaya en la posición correcta, necesitamos un nuevo contador sólo de componentes positivas. A medida que ese contador se va incrementando, vamos colocando en cada posición el elemento correspondiente. El código completo de este ejemplo es el siguiente:

```
% ud4_fpositivas
% Recibe un vector v y devuelve otro vp con las componentes
% estrictamente positivas de v. Caso de que no haya ninguna
% devolvera 0. Es la primera funcion que recibe un
% vector y va construyendo otro de dimensión distinta
% que el original
function vp=ud4_fpositivas(v)
n=length(v);
ip=0;
vp(1)=0;
i=1;
while i<=n
    if v(i)>0
        ip=ip+1;
        vp(ip)=v(i);
    end
    i=i+1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 4.101 *Prueba* `ud4_fpositivas`.

Ejercicio 4.102 *Crea una función que reciba un vector* $\mathbf{u}$ *de números naturales, un natural* m *y devuelva un vector* $\mathbf{v}$ *que contenga aquellos elementos de* $\mathbf{u}$ *que sean divisores de* m. *Por ejemplo, si* $\mathbf{u} = (10, 7, 4, 5, 3)$ *y* $m = 40$, *el resultado sería* $\mathbf{v} = (10, 4, 5)$. *Si no hay ninguno, la función devolverá 0.*

Ejercicio 4.103 *Crea una función que reciba un número natural* n *y devuelva un vector con todos los términos de la sucesión de Fibonacci impares y menores que* n. *La sucesión de Fibonacci la iniciaremos con* $x_1 = 1$ *y* $x_2 = 1$. *Por ejemplo si* $n = 40$ *nos debería devolver un vector* $(1, 1, 3, 5, 13, 21)$.

Ejercicio 4.104 *Codifica una función que reciba un vector* $\mathbf{v}$ *de números naturales y un número natural* m *y devuelva un vector formado por aquellos elementos de* $\mathbf{v}$ *tal que su divisor máximo, excluyendo al propio número, sea mayor que* m. *La función ha de devolver cero si no hay ningún elemento de* $\mathbf{v}$ *con esa propiedad. Así, si* $v = (70, 15, 14, 11, 18)$, *y* $m = 6$, *el vector resultado será* $(70, 14, 18)$, *pero con el mismo* $\mathbf{v}$ *y* $m = 61$ *el resultado será* 0.

Ejercicio 4.105 *Crea una función que reciba un vector* $\mathbf{v}$, *cuyos elementos son números naturales estrictamente mayores que la unidad, por hipótesis, y que devuelva un vector con los primos que contiene. Si* $\mathbf{v}$ *no tiene ningún primo, entonces la función deberá devolver* 0. *Por ejemplo si* $\mathbf{v} = (12, 3, 6, 29, 11, 14)$ *la función deberá devolver* $(3, 29, 11)$, *pero si* $\mathbf{v} = (12, 4, 15, 8, 21)$ *entonces la función devolverá* 0 *(solución en apéndice).*

Ejercicio 4.106 *Repetir el ejercicio 4.105, sin llamar a ninguna función (solución en apéndice).*

Ejercicio 4.107 *Crea una función que reciba un vector* $\mathbf{u}$ *de números naturales y sin llamar a ninguna función, calcule un vector* **vprimos** *con los primos de* $\mathbf{u}$. *Una vez hecho esto, calculará la media de los elementos primos para lo cual llamará a su vez a la función* `ud4_fmedia` *pasándole* **vprimos**. *Esa media será el único resultado que devolverá la función. Caso de que no haya ningún primo, devolverá 0. Por ejemplo si* $\mathbf{u} = (15, 11, 22, 8, 13, 2)$, *los elementos primos son 11,13 y 2 (son 3), y su media será (11+13+2)/3=8.6666 (solución en apéndice).*

Ejercicio 4.108 *Codifica una función que reciba un vector* $\mathbf{u}$ *y devuelva un vector* $\mathbf{v}$ *del cual hayamos quitado el máximo y el mínimo de* $\mathbf{u}$. *Por ejemplo, si* $\mathbf{u} = (1, 7, -2, 4, 3)$, *entonces* $\mathbf{v} = (1, 4, 3)$ *(solución en apéndice).*

Ejercicio 4.109 *Codifica una función que reciba un vector* $\mathbf{v}$ *y cree un nuevo vector* $\mathbf{u}$ *en el que aparezca el vector original intercalando 1 si entre dos componentes consecutivas si la segunda es mayor que la primera, -1 si es menor y 0 si son iguales. Por ejemplo si* $\mathbf{v} = (1.3, 4.4, 4.4, 3.2, 7.3, 4.5, -2.3)$ *la función devolverá* $\mathbf{u} = (1.3, 1, 4.4, 0, 4.4, -1, 3.2, 1, 7.3, -1, 4.5, -1, -2.3)$.

Ejercicio 4.110 *Crea una función que reciba un número n (por hipótesis natural y mayor que la unidad) y devuelva un vector con los números primos divisores de n (incluyendo al propio n entre los divisores de n). Por ejemplo si $n = 12$ entonces la función devolverá $(2, 3)$, que son los primos que dividen a 12. Si $n = 7$ la función devolverá (7).*

Ejercicio 4.111 *Crea una función que reciba un número n (por hipótesis natural y mayor que la unidad) y devuelva un vector con los números primos divisores de n (incluyendo al propio n entre los divisores de n). No se podrá llamar a ninguna función. Por ejemplo si $n = 12$ entonces la función devolverá $(2, 3)$, que son los primos que dividen a 12. Si $n = 7$ la función devolverá (7).*

Ejercicio 4.112 *Codifica una función que reciba un vector $\mathbf{u}$ de naturales mayores que 1 por hipótesis, calcule la parte entera de la media aritmética de las componentes primas, y devuelva 1 si ese valor es primo y 0 en caso contrario. Por ejemplo, si $\mathbf{u} = (3, 6, 5, 11, 4)$, la media de los primos es (3+5+11)/3=6.333 cuya parte entera es 6, que no es primo. Por tanto, la función devolvería 0. La función no podrá llamar a ninguna otra función creada por el usuario ni a ningún ejemplo del libro.*

Ejercicio 4.113 *Codifica una función que reciba un número natural $n > 1$ y devuelva un vector cuyas componentes sean los primos que aparecen en la descomposición en factores primos de n, ordenados de menor a mayor. Por ejemplo si $n = 18$ entonces la solución será el vector $(2, 3, 3)$ ya que $18 = 2 \cdot 3 \cdot 3$. Si $n = 7$ entonces la solución será el vector (de una sola componente) 7*

Ejercicio 4.114 *Escribe una función que reciba un vector de números enteros $\mathbf{v}$ y devuelva otro vector $\mathbf{w}$ que solamente tenga las componentes de $\mathbf{v}$, sin que aparezcan repetidas, y en el orden en el que aparecen en $\mathbf{v}$. Por ejemplo si $\mathbf{v} = (-1, 2, -1, 3, 2)$ el vector $\mathbf{w}$ será $\mathbf{w} = (-1, 2, 3)$ (en $\mathbf{w}$ se omiten la tercera y quinta componentes de $\mathbf{v}$ porque están repetidas) o si $\mathbf{v} = (2, 2, 2, 5, 2)$ entonces $\mathbf{w} = (2, 5)$ (solución en apéndice).*

Ejercicio 4.115 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.11. Funciones con salidas múltiples

En esta sección se presentan funciones que tienen más de un argumento de salida. Es un tema que no plantea dificultad a los estudiantes y que permite hacer funciones más interesantes. En múltiples ocasiones es necesario obtener varios resultados en una misma función, para lo cual es necesario pasarlos todos como argumentos de salida. La sintaxis es muy sencilla, simplemente se agrupan todos los argumentos entre corchetes y separados por comas. El ejemplo más elemental que se nos ha ocurrido es el de resolver una ecuación de segundo grado, la cual

siempre tendrá 2 soluciones, sean reales, complejas o una raíz doble. MATLAB es capaz de trabajar directamente con complejos pero es un tema al que apenas hemos prestado atención y que escapa a los contenidos del curso. Por tanto, propondremos ejemplos de ecuaciones que tengan raíces reales.

```
% ud4_fe2grado.m
% devuelve las dos soluciones de ax^2+bx+c=0
function [x1,x2]=ud4_fe2grado(a,b,c)
x1=(-b-sqrt(b*b-4*a*c))/(2*a);
x2=(-b+sqrt(b*b-4*a*c))/(2*a);
```

Para invocar esta función desde la línea de comandos, lo haremos del siguiente modo, por ejemplo:

```
>> [r,s]=ud4_fe2grado(1,-3,2)
r =

     1

s =

     2
```

El resultado es que las variables r y s recogen los valores de las dos raíces, 1 y 2, de la ecuación $x^2 - 3x + 2 = 0$.

Ejercicio 4.116 *Copia* `ud4_fe2grado.m` *a tu carpeta de trabajo y pruébala.*

Ejercicio 4.117 *Crea una función que reciba los coeficientes de una ecuación de segundo grado y devuelva la suma y el producto de las soluciones. Para comprobar si tu código es correcto, usa los coeficientes del polinomio $2x^2 + 5x - 3$, que tiene como raíces -3 y 0.5, cuya suma es -2.5 y cuyo producto es -1.5.*

Ejercicio 4.118 *Modifica la función* `ud4_fe2grado` *para que si las raíces tienen parte imaginaria (el discriminante $b^2 - 4ac$ es negativo) devuelva $x1 = 0$, $x2 = 0$ y otra variable* `error=1`*. Si las raíces son reales la función devolverá las dos raíces en $x1$ y $x2$ y $error = 0$. Por ejemplo, si $a = 1$, $b = -5$ y $c = 6$, la función devolverá $x1 = 2$, $x2 = 3$,* `error=0`*. Si $a = 1$, $b = 3$ y $c = 6$, la función devolverá $x1 = 0$, $x2 = 0$,* `error=1` *(solución en apéndice).*

Ejercicio 4.119 *Crea una función que reciba los coeficientes de la ecuación $ax^4 + bx^2 + c = 0$. Resolverá primero la ecuación $ay^2 + by + c = 0$ haciendo el cambio $y = x^2$ y llamando para ello a* `ud4_fe2grado` *obteniendo y_1, y_2. Finalmente deshará el cambio de variable mediante $x = \pm\sqrt{y}$ para devolver las cuatro soluciones de la ecuación original $\pm\sqrt{y_1}$, $\pm\sqrt{y_2}$. Por ejemplo, si $a = 1$, $b = -5$ y $c = 6$, la función devolverá $-\sqrt{2}$, $\sqrt{2}$, $-\sqrt{3}$, $\sqrt{3}$ (solución en apéndice).*

Ejercicio 4.120 *Codificar una función que reciba dos números naturales n, m, y devuelva el máximo común divisor de esos dos números y su mínimo común múltiplo. Por ejemplo, si $n = 15$, $m = 20$, el máximo común divisor es 5 y el mínimo común múltiplo 60*

Ejercicio 4.121 *Crea una función que reciba un vector y devuelva el máximo y el mínimo de sus elementos.*

Ejercicio 4.122 *Crea una función que reciba un vector y devuelva la posición del máximo y del mínimo de sus elementos.*

Ejercicio 4.123 *Crea una función que reciba un vector y devuelva el máximo, el mínimo y sus posiciones.*

Ejercicio 4.124 *Codifica una función que reciba un vector de enteros* $\mathbf{v}$ *y un natural* n *y devuelva dos variables. La primera salida que sea el número de múltiplos de* n *que hay en* $\mathbf{v}$. *La función tendrá una segunda salida que será* 0 *si ese número es par y un* 1 *si es impar. Por ejemplo si* $\mathbf{v} = (3, 11, 16, 4, 7, 5, 12)$ *y* $n = 4$ *entonces la función devolverá* 3 *porque hay tres múltiplosL de* 4 *y la función devolverá también* 1 *porque* 3 *es impar. (2.5 p)*

Ejercicio 4.125 *Escribe una función que reciba un vector* $\mathbf{v}$ *de números naturales y devuelva dos salidas,* $[I, E]$. *La primera salida,* I, *ha de ser la posición del primer elemento impar de* $\mathbf{v}$. *La segunda salida,* E, *ha de ser el primer elemento impar de* $\mathbf{v}$. *Si no hay ningún elemento impar, ambas salidas han de ser* 0. *Por ejemplo, si* $\mathbf{v} = (2, 4, 16, 7, 21, 12)$ *la función ha de devolver* $I = 4$ *y* $E = 7$ *ya que la primera componente impar del vector es* 7 *y está en la posición* 4, *es decir,* $v_4 = 7$ *(solución en apéndice).*

Ejercicio 4.126 *Codifica una función que reciba un vector* $\mathbf{u}$ *de números naturales y devuelva una variable* `flag` *que ha de valer 1 si* $\mathbf{u}$ *tiene al menos dos elementos iguales y 0 en caso contrario. La función devolverá también el primer múltiplo de 3 del vector* $\mathbf{u}$ *(si no hay ninguno devolverá esta variable de salida valdrá 0). No se podrá llamar a ninguna función. Por ejemplo, si* $\mathbf{u} = (14, 5, 6, 5, 7, 12)$, *la función devolverá 1 y 6, y si* $\mathbf{u} = (14, 5, 6, 8, 7, 12)$ *devolverá 0 y 6.*

Ejercicio 4.127 *Crea una función que reciba un vector* $\mathbf{u}$ *y devuelva un vector* $\mathbf{up}$ *con las componentes estrictamente positivas y otro* $\mathbf{un}$ *con las estrictamente negativas de* $\mathbf{u}$ *respectivamente. Las componentes en los nuevos vectores irán colocadas en el mismo orden que en el vector original.*

Ejercicio 4.128 *Crea una función que reciba un vector* $\mathbf{u}$ *de números naturales y devuelva un vector con los números primos que contiene y otro con los no primos.*

Ejercicio 4.129 *Codifica una función que reciba dos vectores* $\mathbf{u}$, $\mathbf{v}$ *de igual dimensión y que tendrá 3 salidas. Primero, los comparará componente a componente y devolverá dos vectores* $\mathbf{umax}$, $\mathbf{vmax}$ *de tal modo que* $\mathbf{umax}$ *se vaya rellenando con los elementos de* $\mathbf{u}$ *que sean mayores que los elementos de* $\mathbf{v}$ *que ocupan la misma posición. Ídem con* $\mathbf{vmax}$ *pero al revés. La función devolverá también el valor mínimo de los dos vectores* $\mathbf{u}$, $\mathbf{v}$. *Por ejemplo, si* $\mathbf{u} = (4, 7, 8, 3, 5, -1)$ *y* $\mathbf{v} = (9, 5, 1, 3, 7, 2)$, *la función devolvería* $\mathbf{umax} = (7, 8)$ *y* $\mathbf{vmax} = (9, 7, 2)$. *El primer elemento de* $\mathbf{vmax}$ *es 9 porque* $u(1) < v(1)$. *El primer elemento de* $\mathbf{umax}$ *es 7 porque* $u(2) > v(2)$, *y así sucesivamente. El valor mínimo de los dos vectores, en este ejemplo sería -1, también será una salida de la función.*

Ejercicio 4.130 *Codifica una función que reciba un vector* $\mathbf{v}$ *y devuelva los índices de los elementos cuya diferencia en valor absoluto sea mínima. Se supondrá que el vector solamente tiene dos elementos que cumplan esa propiedad. Para obtener el valor absoluto se podrá usar la función* abs *de MATLAB. Por ejemplo, si* $v = (7, 14, 11, 15)$, *la función devolvería 2 y 4 pues son el segundo y cuarto elemento los que están más cerca.*

Ejercicio 4.131 *Codifica una función que reciba un vector* $\mathbf{v}$ *de números naturales y devuelva el mayor y menor de los números primos que contenga. Caso de que no contenga ningún primo, devolverá dos ceros. El 1 se considerará primo. Así, si* $v = (70, 11, 14, 19, 18, 5, 9)$, *el resultado será* $[5, 19]$. *Si* $v = (70, 11, 9)$, *el resultado será* $[11, 11]$. *Si* $v = (70, 12, 9)$, *el resultado será* $[0, 0]$.

Ejercicio 4.132 *Crea una función que reciba un vector* $\mathbf{u}$ *de números naturales y que devuelva su media aritmética* m, *un vector con los primos* $\mathbf{vprimos}$, *otro con los números pares* $\mathbf{vpares}$ *y otro con los números en posiciones impares* $\mathbf{vposimpar}$. *Si no no hay números primos o pares la función devolverá 0 en las variables correspondientes. No se podrá llamar a ninguna función. Por ejemplo, si* $\mathbf{u} = (7, 15, 20, 11, 38, 6)$, *el resultado será* $m = 16.1667$, $\mathbf{vprimos} = (7, 11)$, $\mathbf{vpares} = (20, 38, 6)$, $\mathbf{vposimpar} = (7, 20, 38)$.

Ejercicio 4.133 *Crea una función que reciba un vector* $\mathbf{v}$ *y devuelva un vector* $\mathbf{w}$ *y un número* $error$ *de modo que:*

1. *Si todos los elementos de* $\mathbf{v}$ *son estrictamente positivos, devolverá* $w(i) = \log(v(i))$, $1 \leq i \leq n$ *y* $error = 0$.

2. *Si algún elemento es menor o igual que cero, devolverá* $\mathbf{w} = \mathbf{v}$ *y* $error = 1$.

Ejercicio 4.134 *Crea una función que reciba un número natural* F *(mayor que 4 por hipótesis) y devuelva dos resultados,* `fibo` *y* `flag`. *La primera salida,* `fibo`, *ha de ser el término de la sucesión de Fibonacci iniciada por* $x_1 = 1$, $x_2 = 1$, *que está más cerca de* F. *La segunda salida* `flag` *ha de ser* 1 *si* `fibo` *es mayor que* F, *será* -1 *si* `fibo` *es menor que* F *y* `flag`$= 0$ *si son iguales. Caso de que haya dos valores a igual distancia, devolverá el menor. Por ejemplo si* $F = 32$, *como la sucesión de Fibonacci es* $1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$ *la función devolverá* `fibo`$= 34$ *(el término de la sucesión de Fibonacci más cercano a 32) y también devolverá* `flag`$= 1$ *(ya que* $34 > 32$*).*

Ejercicio 4.135 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.12. Vectores y polinomios

En las dos últimas secciones de este tema vamos a trabajar con polinomios. Esto las hace más cercanas a las aplicaciones prácticas con las que nos enfrentamos al implementar algoritmos para resolver problemas en muchos ámbitos de la ingeniería, teoría económica, optimización... Vincularemos polinomios con vectores y utilizaremos los algoritmos estudiados con vectores para resolver problemas relativos a polinomios.

Lo primero es conseguir identificar un polinomio de grado n con un vector de $n+1$ componentes y por tanto, dimensión $n+1$. Así, si tenemos el siguiente polinomio de grado n, la representación habitual que de él se hace en matemáticas sería:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

De este modo, este polinomio se identificaría de modo inmediato con el vector $\mathbf{a}$.

$$\mathbf{a} = (a_0, a_1, \cdots, a_{n-1}, a_n)$$

Sin embargo, no podemos trasladar esto directamente a MATLAB pues la indexación de los vectores siempre empieza por la componente 1. Ello invita a repensar polinomio y vector del siguiente modo:

$$a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1} + a_{n+1} x^n$$

$$\mathbf{a} = (a_1, a_2, \cdots, a_n, a_{n+1})$$

Por ejemplo, el polinomio $2.5 - 3.7x^2 + 0.5x^3$ se representa con el vector $\mathbf{a} = (2.5, 0.0, -3.7, 0.5)$. Como se puede observar su grado 3 pero la dimensión del vector asociado es 4.

El primer ejemplo de esta sección es el de una función que reciba un vector con los coeficientes de un polinomio y devuelva los coeficientes del polinomio derivada. Hay que fijarse que el polinomio derivada del polinomio

$$a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1} + a_{n+1} x^n$$

es

$$a_2 + 2 \cdot a_3 x + \cdots + (n-1)\, a_n x^{n-2} + n\, a_{n+1} x^{n-1}$$

Si identificamos este segundo polinomio con un vector $\mathbf{dp}$, tendremos que, por un lado el término constante de a desaparece y que

$$dp(i) = i\, a(i+1)$$

que es la idea principal detrás de la función.

```
% ud4_fderivapol
```

```
% recibe un vector con los coeficientes de un polinomio
% y devuelve su polinomio derivada.
function dp=ud4_fderivapol(a)
grado=length(a)-1;
i=1;
while i<=grado
    dp(i)=i*a(i+1);
    i=i+1;
end
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 4.136 *Prueba la función* `ud4_fderivapol`.

Ejercicio 4.137 *Crea una función que reciba un número natural n y devuelva el polinomio de Mc-Laurin de de grado n de la función e^x:*

$$e^x \sim 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Usa la orden `factorial` *para calcular los factoriales. Por ejemplo, si $n = 3$, la función devolverá el vector* $(1, 1, 1/2, 1/6)$

Ejercicio 4.138 *(Para valientes) Crea una función que reciba un número natural n y devuelva el vector formado por los coeficientes del polinomio de Taylor de grado n de la función seno en 0. Por ejemplo, si $n = 7$ la función devolverá el vector $(0, 1, 0, -1/3!, 0, 1/5!, 0, -1/7!)$, que se corresponde con el polinomio*

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

(solución en apéndice)

Ejercicio 4.139 *Crea una función que reciba un vector $\mathbf{p}$. El vector $\mathbf{p}$ es la representación vectorial de un polinomio $p(x)$, con el criterio seguido en el libro. La función devolverá la primitiva (integral) $IP(x)$ de $p(x)$, $\int p(x)dx$, con constante nula. Por ejemplo, si $\mathbf{p} = (3, -4, 1)$, se tiene que (haciendo nula la constante de integración)*

$$IP(x) = \int p(x)dx = \int (3 - 4x + x^2)dx = 3x - 2x^2 + \frac{x^3}{3}$$

El vector (IP) asociado al polinomio $IP(x)$ y salida de la función será por tanto:

$$\mathbf{IP} = (0.0000, 3.0000, -2.0000, 0.3333)$$

(solución en apéndice)

Ejercicio 4.140 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

4.13. Evaluación de un polinomio

La aplicación práctica de los polinomios exige habitualmente evaluarlos para determinados valores. La forma más directa de construir una función que evalúe un polinomio a en un punto x es:

```
% ud4_fevalua.m
% ud4_fevalua(a,x) recibe un vector a de dimensión grado+1
% y un escalar x y devuelve el polinomio
% a(grado+1)x^n+...+a(2)x+a(1) evaluado en x.
function y=ud4_fevalua(a,x)
grado=length(a)-1;
y=a(1);
i=2;
while i<=grado+1
    y=y+a(i)*x^(i-1);
    i=i+1;
end
```

en la cual simplemente vamos incrementando la variable y en el sumando $a_i x^{i-1}$, hasta que i sea $n+1$. Para invocar esta función desde la línea de comandos, evaluando el polinomio ejemplo utilizando en la sección anterior en $x = 2$ escribiremos:

```
>> ud4_fevalua([2.5 0.0 -3.7 0.5],2)
ans =
   -8.3000
```

Para comprobar, haremos las siguientes operaciones:

```
>> 2.5+0.0*2-3.7*2^2+0.5*2^3
ans =
   -8.3000
```

Podemos construir un pequeño *script* `ud4_spintapol.m` que construya una gráfica del polinomio anterior entre -1.5 y 7.5 por ejemplo, el cual ejecutamos directamente invocándolo desde la línea de comando como ya hicimos al manejar los primeros *scripts* en el tutorial del capítulo 1.

```
x=-1.5:0.01:7.5;
n=length(x);
y=0*x;
i=1;
while i<=n
    y(i)=ud4_fevalua([2.5 0.0 -3.7 0.5],x(i));
    i=i+1;
```

```
end
plot(x,y);
grid on;
shg;
```

MATLAB tiene una función específica para evaluar un polinomio. Se llama `polyval`. Se utiliza de modo similar a la aquí descrita aunque la indexación del vector que almacena los coeficientes del polinomio está invertida. Para más información al respecto, se recomienda pedir ayuda sobre la misma en la línea de comandos.

Los ejercicios correspondientes a este ejemplo son bastante interesantes:

Ejercicio 4.141 *Prueba la función* `ud4_fevalua`.

UNIVERSIDAD TECNICA DEL NORTE BIBLIOTECA

Ejercicio 4.142 *Para obtener un valor aproximado del seno de un ángulo α, podemos usar por ejemplo el polinomio de Taylor de grado 7 del seno en el cero.*

$$\alpha - \frac{\alpha^3}{3!} + \frac{\alpha^5}{5!} - \frac{\alpha^7}{7!}$$

Usando la función `ud4_fevalua`, *crea una función que reciba un valor de α y devuelva el valor del seno de α, calculado a partir de la fórmula anterior. Por ejemplo, si $\alpha = 1.9$, la función devolverá 0.9454 (solución en apéndice).*

Ejercicio 4.143 *Crea una función que reciba α y un número natural n y devuelva el valor del seno en α aproximándolo por el polinomio correspondiente a su desarrollo en serie de Taylor en 0 de grado n. Se hará un uso conveniente de la función* `ud4_fevalua`. *Por ejemplo, si $n = 7$ y $\alpha = 1.9$, la función devolverá 0.9454.*

Ejercicio 4.144 *Crea una función que reciba un vector* $\mathbf{pol}$ *que representa un polinomio, con la codificación estudiada en el libro, y que reciba un escalar x y devuelva el resultado de evaluar el polinomio derivada de* $\mathbf{pol}$ *en x, para lo cual se utilizará la función* `ud4_fevalua`. *Por ejemplo, si el polinomio es $-3 + 2x + x^2$, o sea $\mathbf{pol} = (-3, 2, 1)$, y $x = 5$, el polinomio derivada será $2 + 2x$, y evaluar este polinomio derivada en 5 daría $2 + 2 \cdot 5 = 12$ (solución en apéndice).*

Ejercicio 4.145 *Crea una función que reciba dos vectores, el primero de los cuales serán los coeficientes de un polinomio $p(x)$. La función devolverá 1 si en el segundo vector aparece una raíz de $p(x)$ y 0 en caso contrario. Utiliza* `ud4_fevalua` *para comprobar si un número es raíz del polinomio o no. Por ejemplo si $p(x) = 3 - 4x + x^2$ y $\mathbf{x} = (0, 0.5, 1, 1.5, 2, 2.5)$, la función devolverá 1 ya que $x_3 = 1$ es raíz del polinomio.*

Ejercicio 4.146 *Crea una función que reciba dos vectores,* p *y* x*. El primer vector,* p*, contendrá los coeficientes de un polinomio, $p(x)$. La función tendrá cinco salidas* `flag, xmax, fmax, xmin, fmin`*. La primera salida* `flag` *será* 1 *si algún elemento de* x *es raíz de $p(x)$ y* 0 *en caso contrario. La salida* `xmax` *es el elemento de* x *donde $p(x)$ tiene valor máximo, y* `fmax` *será ese valor máximo. Análogamente,* `xmin` *y* `ymin` *corresponden respectivamente, al elemento de* x *donde $p(x)$ tiene valor mínimo y ese valor. Utiliza la función* `ud4_fevalua` *del modo adecuado para evaluar el polinomio.*

Por ejemplo si $p(x) = 3 - 4x + x^2$ y $\mathrm{x} = (0, 0.5, 1, 1.5, 2, 2.5)$, la función devolverá $[1, 0, 3, 2, -1]$ ya que $x_3 = 1$ es raíz del polinomio; el valor máximo de $p(x)$ en x se alcanza en $x_1 = 0$ siendo este valor $p(0) = 3$. Por último, el valor mínimo de $p(x)$ en x se alcanza en $x_5 = 2$, siendo este valor $p(2) = -1$.

Ejercicio 4.147 *Crea una función que reciba un número natural n, un número real x y devuelva dos cosas. La primera salida será el polinomio de Mc-Laurin de grado n de la función e^x evaluado en x.*

$$e^x \sim 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Para evaluar el polinomio de Mc-Laurin, se construirá éste componente a componente y se utilizará del modo adecuado la función `ud4_fevalua`*. Por ejemplo, si $n = 3$ y $x = 0.5$, devolvería $1 + 0.5 + 0.5^2/2 + 0.5^3/6 = 1.6458$, que es bastante similar a $e^{0.5} = 1.6487$. La función devolverá como segunda salida la diferencia en valor absoluto entre e^x y el polinomio citado evaluado en x. Por tanto, con estos mismos valores ejemplo, devolvería también 0.0029.*

Ejercicio 4.148 *(Para valientes) Crea una función que reciba un vector* x*, un entero n y devuelva un vector* y *tal que para cada índice i, y_i sea la aproximación de $\sin(x_i)$ dada por un polinomio de Taylor en el 0 de grado n. Se usará la función 4.143. Por ejemplo, si introducimos el vector $\mathrm{x} = (1.9, 2.5, 2.8)$ y $n = 7$, el vector devuelto será el resultado de evaluar el polinomio*

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}.$$

en los puntos $(1.9, 2.5, 2.8)$, o sea devolverá el vector $\mathrm{y} = (0.9454, 0.5885, 0.3078)$. El seno toma en esas mismas abscisa los valores $(0.9463, 0.5985, 0.3350)$. Recuerda que en la unidad 3 hay una sección (3.2.4) dedicada al cálculo del factorial.

Ejercicio 4.149 *(Para valientes) Repetir el ejercicio 4.148 teniendo en cuenta que la evaluación del polinomio en cada valor del vector* x *se realizará llamando a la función* `ud4_fevalua` *(solución en apéndice).*

Ejercicio 4.150 *(Para valientes) Repetir el ejercicio 4.148 sin llamar a ninguna función.*

Ejercicio 4.151 *Codifica una función que reciba un vector* $\mathbf{pol}$ *que representa un polinomio, con la codificación estudiada en el libro, y que reciba un escalar* x*. La función calculará la variable* y *resultado de evaluar, utilizando la función* `ud4_fevalua` *el polinomio* $\mathbf{pol}$ *en* x*. Después construirá el vector polinomio derivada de* $\mathbf{pol}$*,* $\mathbf{dpol}$*, y evaluará dicho polinomio en* x*, para lo cual se utilizará nuevamente la función* `ud4_fevalua`*, denominando* dy *al resultado. Finalmente, se tomará la parte entera del valor absoluto de ambos valores,* y *y* dy *y la función devolverá 1 o 0 dependiendo de que éstas tengan o no algún divisor común.*

Por ejemplo, si el polinomio es $-3+2x+x^2$*, o sea* $\mathbf{pol}=(-3,2,1)$*, y* $x=-7.2$*, tendremos que* $y=-3-2\cdot7.2+7.2^2=34.44$*. El polinomio derivada será* $2+2x$*, y evaluar este polinomio derivada en* -7.2 *daría* $dy=2-2\cdot7.2=-12.4$*. 12 y 34 tienen algún divisor común y por tanto, la función devolverá 1.*

Sin embargo, si $x=-6.2$*,* $y=23.04$*,* $dy=-10.4$*; como 10 y 23 no tienen ningún divisor común la función devolverá 0.*

Ejercicio 4.152 *Codifica una función que reciba dos vectores* $\mathbf{u}$ *y* $\mathbf{v}$ *de numeros reales (distintos entre sí en cada vector) y devuelva el número de parejas* $(u(i),v(j))$ *tales que si evaluamos en cada uno de esos valores el polinomio* $p(x)=-0.6250x^4+7.5833x^3-31.8750x^2+52.9167x-27.0000$*, las imágenes difieran en menos que* 0.02*, es decir* $|p(u(i))-p(v(j))|<0.02$*. Por ejemplo, si tenemos los vectores* $\mathbf{u}=(1,1.125,2.75)$ *y* $\mathbf{v}=(2,2.263,3.71)$*, la función devolvería 3, ya que hay tres combinaciones que cumplen la condición citada. Esas combinaciones son:* $|p(1)-p(2.263)|=0.0065<0.02$*,* $|p(1.125)-p(2)|=0.0143<0.02$ *y* $|p(2.75)-p(3.71)|=0.0061<0.02$*; el resto de las combinaciones no cumplen.*

Ejercicio 4.153 *Crea una función que reciba un vector* $\mathbf{p}$ *y dos números* a *y* b*. El vector* $\mathbf{p}$ *es la representación vectorial de un polinomio* $p(x)$*, con el criterio seguido en el libro. La función devolverá* $\int_a^b p(x)dx$*. Recuérdese que la integral indefinida de un polinomio* $p(x)$ *es otro polinomio* $IP(x)=\int p(x)dx$ *y que la integral definida se calcula* $\int_a^b p(x)dx=IP(b)-IP(a)$*. Se utilizará la función* `ud4_fevalua` *para evaluar la integral. Por ejemplo si* $\mathbf{p}=(3,-4,1)$*,* $a=1$ *y* $b=3$*, se tiene que (obviando la constante de integración):*

$$IP(x)=\int p(x)dx=\int(3-4x+x^2)dx=3x-2x^2+\frac{x^3}{3}.$$

El vector (IP) *asociado al polinomio* $IP(x)$ *será por tanto:*

$$\mathbf{IP}=(0.0000,3.0000,-2.0000,0.3333)$$

Para calcular la integral definida:

$$\int_1^3(3-4x+x^2)dx=3x-2x^2+\frac{x^3}{3}\Big]_1^3=IP(3)-IP(1)=-1.3333$$

Ejercicio 4.154 *Crea una función que reciba un vector* q *con los coeficientes de un polinomio y dos números* a *y* b*, con* $a < b$ *por hipótesis. La función devolverá el valor de la integral del polinomio entre* a *y* b*. Devolverá también la pendiente de la recta tangente al polinomio* q *en el punto medio entre* a *y* b*, o sea, la derivada en el punto medio, para lo cual se llamará del modo adecuado a* `ud4_fderivapol` *y* `ud4_fevalua`*. Por ejemplo, si tenemos el polinomio* $2 + 3x + 4x^2$ *y* $a = 0$*,* $b = 2$*, el valor de la integral es 20.6667 y la pendiente de la recta tangente en 1 es 11.*

Ejercicio 4.155 *Crea una función que reciba un vector* p *y un número natural* $n > 1$ *y devuelva otro vector* v*. El contenido del vector* v *serán los términos impares de la sucesión de Fibonacci iniciada con* $x_1 = 1$ *y* $x_2 = 2$ *menores que* n *evaluados en el polinomio cuyos coeficientes vienen dados por el vector* p*. Es decir si el termino k-ésimo de la sucesión de Fibonacci* x_k *es impar y* $x_k < n$ *entonces evaluaré* $p_1 + p_2x_k + ... + p_nx_k^{n-1}$*. Por ejemplo si* $\mathrm{p} = (2, -4, 2, -3)$ *y* $n = 14$*, entonces los términos de la sucesión de Fibonacci son* $1, 2, 3, 5, 8, 13, 21, ...$ *de los cuales impares y menores que* 14 *tenemos* $1, 3, 5, 13$ *y por lo tanto* $\mathrm{v} = (p(1), p(3), p(5), p(13)) = (-3, -73, -343, -6303)$*. Se puede comprobar por ejemplo que* $2 - 4 \cdot 1 + 2 \cdot 1^2 - 3 \cdot 1^3 = 2 - 4 + 2 - 3 = -3$*. Para evaluar el polinomio* p *se ha de utilizar la función* `ud4_fevalua` *(solución en apéndice).*

Ejercicio 4.156 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

Capítulo 5

Entrada y salida con formato

5.1. General

En un curso convencional de introducción a la programación, lo primero que se aprende es a realizar entradas de valores por teclado y salida de resultados por pantalla con un formato determinado. El primer programa que se suele realizar en ese tipo de cursos consiste en conseguir que por pantalla aparezca un mensaje del tipo "Hola Mundo". En este curso de MATLAB hemos renunciado a ello porque creemos que es fundamental que se entienda que lo esencial para programar de modo estructurado es aprender a pensar de modo modular, planteando la resolución de problemas a través de su división en partes codificables como funciones independientes. De hecho, pensamos que con los capítulos 2 a 4 se cierra de este modo un gran bloque de introducción a la programación, el cual contiene las ideas fundamentales de la misma.

Hemos podido elegir esta estrategia porque el intérprete de comandos de MATLAB permite invocar directamente después dichas funciones y hace innecesaria la codificación de un programa principal que las invoque. En un lenguaje de programación convencional esto no es posible y es necesario codificar un programa principal que se comunique con el usuario mediante teclado, monitor y archivos e invoque también las funciones a utilizar. Por tanto, es importante entender los conceptos de entrada y salida con formato y lo que conllevan estos conceptos.

La idea es que nuestras funciones puedan interaccionar con el usuario a través de la pantalla, teclado y ratón de un modo más sofisticado a como lo han hecho hasta ahora y poder así entender la filosofía que subyace bajo los grandes programas comerciales. Para ello nos apoyaremos en el concepto de *script*, que ya vimos en la sección 1.8. Trataremos de insistir en la diferencia entre un *script* y una función, y los niveles que implican en la memoria RAM, algo un poco complicado en este curso inicial y sobre lo que ya trabajamos en la sección 2.3.

Además, entender las ideas de entrada y salida con formato nos permitirá dar un paso adicional imprescindible para avanzar en programación, que es el de hacer dicha entrada y salida a través de archivos de datos, a los cuales en MATLAB no podemos renunciar.

En suma, hablaremos en este capítulo de los conceptos de entrada y salida con formato. Aplicaremos estas ideas para aprender a gestionar archivos de datos de un modo sencillo. Finalmente, casi como curiosidad, mencionaremos los conceptos elementales que hay detrás de la construcción de una interfaz gráfica de usuario (GUI) de las que son habituales en los programas que se ejecutan en la forma de ventanas de Windows, o bajo alguna de las distribuciones de Linux que también lo permiten.

A partir de este tema se trata de hacer uso de toda la potencialidad de MATLAB pues se pretende aprovechar esta parte del curso para adquirir competencias relevantes en ese sentido. En los temas 1, 2 y 3 se había usado MATLAB con su sintaxis más canónica, simplemente como apoyo para enseñar los fundamentos de la programación estructurada de ordenadores. De ahora en adelante seremos un poco más heterodoxos y haremos uso de las multiples funcionalidades y atajos que ofrece la sintaxis del lenguaje de comandos de MATLAB.

5.2. Entrada y salida con formato

Cuando se elabora un programa de ordenador componiendo una serie de funciones que permiten resolver un determinado problema, normalmente hay que construir una forma de comunicación con el usuario que le permita usar ese programa de un modo sencillo. El intérprete de comandos de MATLAB se queda muy corto en ese sentido si lo comparamos por ejemplo con los entornos de ventanas que todos estamos acostumbrados a usar desde que apareció Windows.

El primer paso para realizar este tipo de entornos es entender lo que es una entrada con formato y una salida con formato. Hasta ahora si queríamos conocer el valor de una variable, simplemente retirábamos el punto y coma de la línea en que se define o cambia, y MATLAB la mostraba por pantalla. Sin embargo, si por ejemplo disponemos de una función que resuelve una ecuación de segundo grado y queremos construir una utilidad que resuelva ese problema y que además pregunte al usuario, de modo ordenado y con claridad, los tres coeficientes de la ecuación, podemos utilizar el comando `input`. La sintaxis de este comando, aplicada a preguntar uno de los coeficientes es la siguiente:

```
var=input('Dame el coeficiente de x2: ')
```

La comilla es la que está en la misma tecla que el símbolo de interrogación de cierre, ?. Cuando se ejecute esta sentencia, introduciendo 5, se obtiene, si se hace a través de la línea de comandos:

```
>>var=input('Dame el coeficiente de x2: ')
Dame el coeficiente de x2: 5
var =
     5
>>
```

y la variable `var` toma valor 5. Si ahora montamos un fichero .m con el siguiente conjunto de órdenes:

```
a=input('Dame el coeficiente de x2: ');
b=input('Dame el coeficiente de x: ');
c=input('Dame el término independiente: ');
[r1,r2]=ud4_fe2grado(a,b,c)
```

le llamamos `prueba.m` y ejecutamos `prueba` en la línea de comandos, tendremos:

```
>> prueba
Dame el coeficiente de x^2: 1
Dame el coeficiente de x: -3
Dame el término independiente: 2
r1 =

     1

r2 =

     2

>>
```

Lo que hemos construido no es una función, sino un *script*. Al introducir los diferentes valores (1,-3,2) habremos definido las variables a, b, c, las cuales pasamos a continuación a `ud4_fe2grado`, que calcula las dos raíces y las asigna a las variables $r1$ y $r2$. Como la instrucción de llamada no termina en punto y coma, MATLAB muestra (hace eco de) los resultados. El fichero `prueba.m` no es una función sino que se llama un programa o *script* MATLAB (ver sección 1.8).

Del mismo modo que se puede hacer entrada con formato, se puede hacer salida por pantalla[1] con formato. Para ello está la orden `fprintf`, cuya sintaxis es la siguiente:

```
fprintf('La primera raíz vale %g y la segunda %g\n',r1,r2);
```

Si ejecutamos esta sentencia en la línea de comandos, después de las anteriores, tendremos:

```
>> fprintf('La primera raíz vale %g y la segunda %g\n',r1,r2);
La primera raíz vale 1 y la segunda 2
```

Para cada variable que queramos imprimir necesitamos una "caja"que escribimos como `%g`. Al final de cada línea es conveniente incluir un retorno de carro, para que la siguiente vez que escribamos algo, aparezca en la línea siguiente. Eso se consigue añadiendo `\n` antes de la comilla de cierre del texto dentro de `fprintf`. Esta sintaxis es similar a la del lenguaje "C".

Si incorporamos la sentencia anterior al fichero `prueba.m` tendremos el ejemplo de referencia para esta sección: `ud5_se2grado`. Este es el primer programa o *script* en el que invocamos una de las funciones sobre las que hemos trabajado durante el curso. Nos referiremos a estos *scripts* como `udX_snombre.m` siendo X el número del capítulo y `nombre` el

[1] Nos referiremos a menudo con la expresión "imprimir" a mostrar por pantalla un determinado resultado, aunque obviamente no implique imprimir nada en papel.

identificador conceptual del *script* en cuestión. A este código le hemos añadido la sentencia inicial `clear all`, que borra la memoria RAM antes de empezar a ejecutar y que nos evitará más de un problema al trabajar con vectores, así como la sentencia `clc` que limpia la pantalla, ambas explicadas en el capítulo 1.

```
% ud5_se2grado.m
% Pide los coeficientes de una ecuación de segundo grado
% e imprime en pantalla las dos soluciones
% Primera vez que hacemos entrada por teclado y salida
% con formato
clear all;
clc;
% Pedimos los coeficientes
a=input('Coeficiente de x2: ');
b=input('Coeficiente de x: ');
c=input('Término independiente: ');
% Calculamos las dos raíces llamando a ud5_fe2grado.m
[r1,r2]=ud4_fe2grado(a,b,c);
% Imprimimos las raíces
fprintf('las raices son %g la primera y %g la segunda\n',r1,r2);
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 5.1 *Crea una carpeta llamada* `ud5` *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 5.*

Ejercicio 5.2 *Copia* `ud5_se2grado.m` *a tu carpeta de trabajo y pruébala.*

Ejercicio 5.3 *Crea un programa que pida los coeficientes de una ecuación de segundo grado e imprima la suma y el producto de las soluciones, obtenidas mediante la función del ejercicio 4.117. Para comprobar si tu código es correcto, usa el mismo ejemplo que en aquella.*

Ejercicio 5.4 *Crea un programa que pida los coeficientes de la ecuación* $ax^4 + bx^2 + c = 0$, *y devuelva sus cuatro soluciones, obtenidas mediante la función del ejercicio 4.119. Para comprobar si tu código es correcto, usa el mismo ejemplo que en aquella.*

Ejercicio 5.5 *Modifica* `ud5_se2grado` *para que cuando las raíces sean imaginarias, lo muestre en un mensaje y cuando no lo sean, imprima las dos raíces. Llamará de modo adecuado a la función 4.118. Para comprobar si tu código es correcto, usa el mismo ejemplo que en aquella.*

Ejercicio 5.6 *Crea un programa que pida los lados de un triángulo y diga qué tipo de triángulo es en función de la longitud de sus lados (equilátero, isósceles o escaleno). Caso de que no se verifique la desigualdad triangular imprimirá por pantalla que los datos no corresponden a un triángulo. Usa el siguiente formato (solución en apéndice):*

```
>> ud5_stipotriangulo
Primer lado: 3
Segundo lado: 4
Tercer lado: 5
Es un triángulo escaleno.
>> ud5_stipotriangulo
Primer lado: 1
Segundo lado: 1
Tercer lado: 5
Los valores introducidos no son los lados de un triángulo
```

Ejercicio 5.7 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

5.3. Lectura y escritura de vectores

Del mismo modo que hemos leído a través del teclado una serie de escalares podemos leer las componentes de un vector. Para ello utilizamos esquemas de este tipo:

```
n=input('Número de elementos: ');
i=1;
while i<=n
    fprintf('Dame el elemento %g\n',i);
    v(i)=input(' ');
    i=i+1;
end
```

El bucle nos permite recorrer con un índice los elementos del vector y pedir las componentes una a una. Aplicamos esta idea en el siguiente programa, el cual llama a la función `ud4_fpositivas`, antes estudiada (sección 4.10). Esta función construye el vector de los números positivos dentro de otro vector dado.

```
% ud5_spositivas.m
% Pide un número n, lee n números y llama
% a la función ud4_fpositivos para generar
% un vector vpos con los términos positivos de v
% Primer script que lee por teclado un vector.
clear all
% Pedimos un vector
n=input('Número de elementos a introducir: ');
i=1;
while i<=n
   fprintf('Dame la componente %g del vector\n',i);
   v(i)=input(' ');
```

```
   i=i+1;
end

% Calculamos un vector con los positivos
vpos=ud4_fpositivas(v)
```

Existe una forma más sencilla de introducir un vector a través de la orden `input`, en realidad la más obvia, escribiéndolo directamente entre corchetes.

```
>>v=input('Dame un vector: ');
Dame un vector: [2 3 4]
```

También es necesario que los programas puedan volcar por pantalla vectores con un determinado formato. Para ello podremos usar la función `fprintf` en bloques de este tipo:

```
n=length(v);
i=1;
while i<=n
   fprintf('%g ',v(i));
   i=i+1;
end
fprintf('\n');
```

La última instrucción se utiliza para imprimir una línea en blanco. El ejemplo en el que se recoge todo esto es el siguiente, el cual llama nuevamente a la función `ud4_fpositivas`.

```
% ud5_soutpos.m
% Pide un número n, lee n números y llama a la función
% ud4_fpositivos para generar un vector vpos con los
% términos positivos de v el cual imprime con formato por pantalla.
% Primer script que imprime por pantalla un
% vector con un formato determinado.
clear all
% Pedimos un vector
v=input('Dame un vector: ');
% Calculamos un vector con los positivos de v
vpos=ud4_fpositivas(v);
% Imprimimos los resultados
fprintf('Positivos: ');
np=length(vpos);
i=1;
while i<=np
   fprintf('%g ',vpos(i));
   i=i+1;
end
fprintf('\n');
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 5.8 *Prueba* `ud5_spositivas`.

Ejercicio 5.9 *Crea un programa que pida un vector, llame a la función del ejercicio 4.121 para obtener su mínimo y su máximo y los imprima en pantalla con un formato adecuado.*

Ejercicio 5.10 *Crea un programa que pida un vector, llame a la función del ejercicio 4.122 para obtener las posiciones de su mínimo y su máximo y los imprima en pantalla con un formato adecuado.*

Ejercicio 5.11 *Crea un programa que pida un vector e imprima en pantalla el máximo y el mínimo de sus elementos y las posiciones del máximo y del mínimo, para lo cual llamará a la función 4.123.*

Ejercicio 5.12 *Prueba* `ud5_soutpos`.

Ejercicio 5.13 *Crea un programa que pida un vector, llame a la función del ejercicio 4.127 para obtener dos vectores con las componentes positivas y negativas de aquel y los muestre en pantalla con un formato adecuado.*

Ejercicio 5.14 *Crea un programa que pida un número n y muestre en pantalla, con un formato adecuado, sus divisores primos, utilizando para calcularlos la función del ejercicio 4.110. Para comprobar si tu código es correcto, usa el mismo ejemplo que en aquella.*

Ejercicio 5.15 *Crea un programa que pida un vector de números naturales, llame a la función del ejercicio 4.128 para obtener dos vectores con los primos y no primos de aquel e imprima los vectores resultado con un formato adecuado.*

Ejercicio 5.16 *Crea una programa que pida un número n y n elementos de un vector. Si todos los elementos son positivos, imprimirá el vector dado por la función del ejercicio 4.133 y si alguno es negativo o cero, imprimirá un mensaje de error.*

Ejercicio 5.17 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

5.4. Ficheros

5.4.1. General

Los ficheros responden a necesidades esenciales en el uso de un ordenador. Permiten almacenar de modo permanente información en dispositivos como el disco duro o memorias USB. Cuando escribimos un documento en un procesador de textos como Ms-WORD por ejemplo, y guardamos ese documento en el disco duro, hemos creado un fichero. Aunque apaguemos el ordenador, cuando lo volvamos a encender podremos recuperar esa información.

En MATLAB hemos estado creando ficheros con el editor de funciones y *scripts*. Cada función y *script* la hemos guardado en un fichero con la extensión .m. También es posible crear en MATLAB un fichero con los valores de las variables activas en la ventana de comandos, aunque nosotros pretendemos generar ficheros que guarden no sólo las variables sino que lo hagan con un determinado formato y en ficheros fácilmente editables desde otros programas.

En la sección 1.8 ya mencionamos que el editor de MATLAB guarda las funciones como ficheros tipo ASCII[2]. En el disco duro en realidad lo que se guarda es una sucesión de unos y ceros y ASCII es un standard para transformar esos unos y ceros en un número limitado de caracteres alfanuméricos, básicamente las cifras del 0 al 9 y las letras del alfabeto inglés en forma mayúscula y minúscula. Son más o menos los símbolos que se pueden teclear directamente desde un teclado estándar de un ordenador. Por tanto, los ficheros que se llaman ASCII están todos codificados de esa manera y permiten guardar básicamente números y letras. Se podrían utilizar ficheros con esta codificación para guardar imágenes (y de hecho en algún caso se hace) pero ocuparían mucho más que un fichero con la codificación JPEG por ejemplo. Sin embargo sí que se usan por ejemplo para definir ficheros CAD de modo vectorial. Los estándares DXF de AUTOCAD o IGES de MICROSTATION, se apoyan en el estándar ASCII para guardar información CAD en forma de ficheros.

La gran ventaja que tienen los ficheros ASCII es que se pueden ver desde cualquier editor, y que además son fáciles de generar y de leer, lo que permite que sean usados como mecanismos de intercambio de la información. Por tanto, vamos a aprender a guardar las salidas de nuestros *scripts*como ficheros ASCII y vamos a aprender también a leer un archivo ASCII para que nos sirva de entrada de información para un *script*.

5.4.2. Escritura de ficheros

La sintaxis para volcar la información en los ficheros es muy similar a la estudiada en las secciones 5.2 y 5.3, que permitía volcar la información en el monitor con un determinado formato. En realidad es básicamente igual pero redirigiendo la salida a un archivo de datos

[2]American Standard Code for Information Interchange

que se encuentra en una unidad física como un disco duro o una memoria USB.

Aunque entraremos en detalle en esta sintaxis, existen atajos muy cómodos de utilizar y que pueden ser usados desde *scripts*o directamente desde la línea de comandos. Uno de ellos es la orden `save`, que permite guardar una o más variables en un archivo ASCII. Si la variable es `var` y el nombre del fichero es `nombrefichero`, la sintaxis es la siguiente:

```
>> save nombrefichero var -ascii
```

Si lo aplicamos por ejemplo a las siguientes sentencias, utilizando la función `ud4_fibo` de la sección 4.8:

```
>> suc=ud4_fibo(1,2,5)'
suc =
     1
     2
     3
     5
     8
>> save fibo.dat suc -ascii
```

tendremos un fichero de nombre `fibo.dat` en nuestra carpeta activa[3].

Se suele añadir a los ficheros que contienen datos la extensión `.dat`, aunque se puede elegir cualquier otra que sea más conveniente en cada caso. El fichero tendrá el siguiente aspecto cuando lo abramos con el editor de MATLAB o directamente lo arrastremos sobre la ventana del editor. Por defecto, las variables se vuelcan al fichero con un formato largo, de alta precisión.

```
1.0000000e+000
2.0000000e+000
3.0000000e+000
5.0000000e+000
8.0000000e+000
```

Otro modo directo de utilizar la orden `save` para escribir un vector o una matriz en un archivo de datos es escribir directamente

```
>> save suc
```

Aparecerá en la carpeta activa un fichero de nombre `suc.mat`. Es un tipo de archivo binario, con una codificación propia de MATLAB, diferente al ASCII, que minimiza el espacio en disco, pero que impide que pueda ser visualizado directamente con un editor ASCII. Se pueden guardar varias variables en el mismo archivo e inclusive todas las variables que estén activas en un determinado momento. Supongamos que queremos guardar todas esas variables en un archivo de nombre `filename.mat`. La orden sería:

[3]Hemos traspuesto la salida de *ud4_fibo* para que aparezca como un vector columna en el archivo de salida.

```
>> save('filename.mat');
```

Aparte de estos formatos genéricos para vectores y matrices, se puede escribir un archivo con un formato determinado. Lo primero que hay que hacer es utilizar la orden `fopen` para crear el archivo, usando la opción `w` como segundo argumento de dicha orden[4]; el primero es el nombre del archivo. La orden se asigna a una variable que permite direccionar el archivo y el archivo se guarda en la carpeta activa.

Para escribir sobre el archivo se usa la orden `fprintf` igual que en las secciones 5.2 y 5.3, pero añadiendo como primer argumento la variable que direcciona el fichero. Presentamos el siguiente ejemplo que escribe en el archivo `suc2.dat` la misma secuencia de datos previa pero mediante un formato determinado de tal forma que cada término de la sucesión se escribe en una línea diferente, precedido de la palabra `Término` y del índice correspondiente.

```
% ud5_sescribefich.m
% script para crear un fichero
clear all;
suc=ud4_fibo(1,2,5);
n=length(suc);
fichero=fopen('suc2.dat','w');
i=1;
while i<=n
    fprintf(fichero,'Término %g = %g\n',i,suc(i));
    i=i+1;
end
fclose(fichero);
```

Si ejecutamos `ud5_sescribefich` desde la línea de comandos, el archivo resultado es el siguiente:

```
Término 1 = 1
Término 2 = 2
Término 3 = 3
Término 4 = 5
Término 5 = 8
```

Como podemos comprobar, tiene el formato que nosotros queríamos especificar. El archivo `suc2.dat` podemos importarlo por ejemplo desde Ms-EXCEL y hacer una gráfica con sus valores.

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 5.18 *Prueba* `ud5_sescribefich`. *Modifícala para generar 20 términos de la sucesión y ábrelo después desde Ms-EXCEL, dibujando la gráfica de los mismos.*

Ejercicio 5.19 *Rehaz el ejemplo* `ud5_se2grado`, *de la sección 5.2, realizando la salida sobre un archivo de datos de nombre* `e2grado.dat`. *Ábrelo desde Ms-WORD (solución en apéndice).*

[4] w por "write"

Ejercicio 5.20 *Rehaz el ejemplo* `ud5_soutpos`*, de la sección 5.3, realizando la salida sobre un archivo de datos de nombre* `positivas.dat`*. Importa el archivo desde Ms-EXCEL y dibuja una gráfica con esos valores.*

Ejercicio 5.21 *Codifica un script que lea por teclado un vector* $\mathbf{u}$ *del modo estudiado en el libro, llame a la función* `ud4_fimax` *y devuelva el resultado por pantalla mediante un mensaje del tipo:*

```
El máximo del vector es XXX y ocupa la posición XXX en el vector.
```

Salvará además el vector original $\mathbf{u}$*, el máximo, y su posición, en un archivo tipo ASCII de nombre* `vector.dat`

Ejercicio 5.22 *Resuelve nuevamente el ejercicio 5.6 realizando la salida sobre un archivo de datos de nombre* `triangulos.dat`.

Ejercicio 5.23 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

5.4.3. Lectura de ficheros

Para leer archivos, existe un atajo similar al correspondiente a la escritura, utilizando la orden `load`. Esta orden es útil para leer archivos que contienen vectores o matrices bien organizadas. Por ejemplo podemos usarla para leer el archivo que previamente hemos creado, el `fibo.dat`.

```
>> load fibo.dat
>> fibo
fibo =
     1
     2
     3
     5
     8
```

Esa lectura genera una variable **fibo**, cuyo valor es el vector que hay dentro del archivo. Es interesante comentar que la orden `load` ignora aquellas líneas del archivo en cuestión que comienza con el símbolo de comentario de MATLAB, o sea %.

Una sintaxis alternativa de esta variable puede ser más conveniente en algunas ocasiones, asignando el contenido del archivo a una variable específica:

```
>> x=load('fibo.dat');
>> x
x =
```

```
1
2
3
5
8
```

`'fibo.dat'` es aquí una cadena de caracteres que indica a `load` el nombre de archivo a abrir y eventualmente la ruta en la que se encuentra. Aunque apenas trabajaremos con cadenas de caracteres en este libro es importante resaltar aquí que ese argumento de `load` puede ser por ejemplo un vector de nombres de ficheros obtenidos con la orden `dir` (se deja este tema a los lectores más interesados).

Otra posibilidad para la lectura de ficheros de datos es la herramienta de importación de archivos de MATLAB. Si pinchamos sobre el archivo `fibo.dat` en el espacio de la carpeta activa (ver figura 0.1.1), veremos que aparece una herramienta de importación de archivos (figura 5.1) que ofrece varias opciones, similares a las que nos ofrece el importador de archivos de Ms-EXCEL que hemos usado previamente, que permiten establecer el tipo de separador entre columnas y si hay que eliminar algunas líneas en el encabezado. No recomendamos esta herramienta pues exige intervención permanente del usuario para leer archivos, algo poco práctico en general.

Otra funcionalidad interesante es la que proporciona la orden `hdrload`, la cual no está en la distribución estándar de MATLAB, pero que se encuentra fácilmente online. Está pensada para leer archivos de datos como el siguiente, que incluye un encabezado, y al que llamaremos `fibocabecera.dat`.

```
Esto es el encabezado
Este archivo corresponde a un análisis de datos, bla,
bla, bla, bla...
  1.0000000e+000
  2.0000000e+000
  3.0000000e+000
  5.0000000e+000
  8.0000000e+000
```

Al leer el archivo usando `hdrload`, se discrimina el encabezado de la parte que tiene los datos. Tiene esta sintaxis:

```
>> [header, data] = hdrload('fibocabecera.dat')
```

Podeis ver el resultado, simplemente pidiendo a MATLAB que os muestre el valor de `header` y de `data`.

La última opción presentada es leer directamente los archivos con extensión `.mat`, el formato propio de MATLAB. Imaginemos que hay un archivo en el carpeta activa que se llame `suc.mat`. Leerlo y cargar su contenido en memoria es tan fácil como escribir:

```
>> load suc
```

Con las opciones previas podremos solucionar la mayor parte de los problemas. Existen métodos más generales para lectura de ficheros de datos pero no son especialmente sencillos en MATLAB, al apoyarse éste en la filosofía matricial del entorno.

Para construir una aplicación práctica trabajaremos sobre el ejemplo `ud5_soutpos.m`, de la sección 5.3. En ese ejemplo se trataba de extraer de una lista de números estructurada como un vector, aquellos que son positivos, mostrándolos por pantalla. Lo que haremos será obtener la lista inicial de un archivo de datos e imprimiremos el vector resultado en otro archivo de datos.

Para crear el archivo de datos inicial, simplemente lo editamos manualmente en Ms-EXCEL, para tener lo que aparece en la figura 5.2. Lo guardaremos con el nombre `numeros.dat`, y con formato de archivo de texto, desde el menú de guardar archivo de MS-EXCEL.

Para leer el archivo utilizaremos la orden `load`, y para guardar el resultado lo haremos directamente con la orden `save`.

```
% ud5_sleefich.m
% Lee n números de un fichero numeros.dat y llama
% a la función ud4_fpositivos para generar
% un vector vpos con los términos positivos de v
% el cual guarda en un fichero positivos.dat
clear all
load numeros.dat
vpos=ud4_fpositivas(numeros)';
save positivos.dat vpos -ascii
```

Si ejecutamos `ud5_seleefich` desde la línea de comandos, el archivo resultado es el siguiente:

```
7.0000000e+000
1.5000000e+001
1.2000000e+001
4.0000000e+000
4.0000000e+000
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 5.24 *Crea un programa que lea un vector* **u** *desde un archivo de datos* `vector.dat` *(el cual debes crear editándolo manualmente en Ms-EXCEL o cualquier otro editor ASCII), y calcule un vector* **up** *con las componentes estrictamente positivas y otro* **un** *con las estrictamente negativas del vector* **u** *respectivamente. Las componentes en los nuevos vectores irán colocadas en el mismo orden que en el vector original. El programa creará dos archivos de salida* `un.dat` *y* `up.dat` *con los vectores resultado, conteniendo las componentes positivas y las negativas, con un formato adecuado.*

Ejercicio 5.25 *Idem 4.128 relativo a números primos y no primos.*

Ejercicio 5.26 *Codifica un script que lea por teclado un vector* `p`, *conteniendo los coeficientes de un polinomio con la codificación estudiada en el libro. El script pedirá por teclado dos valores a y b, con $a < b$ por hipótesis y un número natural n. El script llamará a* `ud4_fequidistante` *para generar un vector* `x` *a partir de los valores a, b, n. El script hará ahora un bucle que le permitirá obtener un vector* `y` *resultado de evaluar el polinomio* `p` *en cada componente del vector* `x`, *para lo cual llamará del modo adecuado a* `ud4_fevalua`. *Finalmente el script dibujará una gráfica de la curva* `x,y` *y guardará ambos valores en un archivo ascii de nombre* `curva.dat`.

Ejercicio 5.27 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que la secuencia de operaciones que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

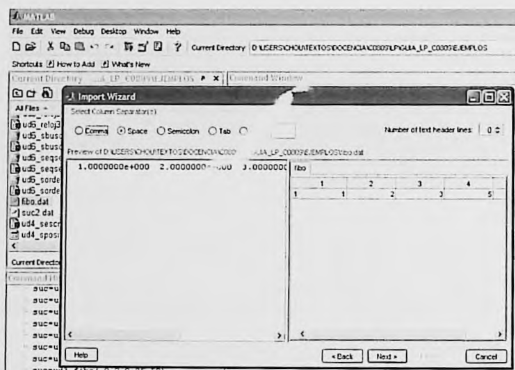


Figura 5.1: Herramienta de importación de archivos de datos

5.5. Diseño básico de GUIs

5.5.1. General

Es muy importante facilitar a los usuarios de los programas una interacción amigable con los mismos. Para ello se construyen las interfaces gráficas (GUIs por Graphics User Interfaces) con ventanas, menús, botones, etc. Uno de los atractivos más importantes de lenguajes como Visual BASIC o entornos como LabVIEW es la posibilidad de desarrollar de modo sencillo aplicaciones con interfaces gráficas poderosas. MATLAB no tiene esa potencia pero sí dispone de algunas herramientas que permiten elaborar interfaces gráficas sencillas desde las que invocar a *scripts*o directamente a programas externos. No estamos todavía preparados para tratar este tema

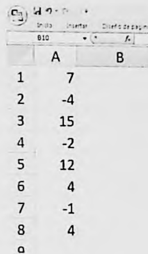


Figura 5.2: Edición del fichero numeros.dat

con profundidad en MATLAB pero si lo suficiente como para construir entornos como el la figura 5.3, correspondiente al ejemplo `ud5_wfibo`. Así, en la siguiente sección veremos una sencilla aplicación de estas herramientas para hacer una gráfica los elementos de la sucesión de Fibonacci obtenidos mediante el ejemplo `ud4_fibo` de la sección 4.8. No profundizaremos en este tema, y dejaremos al estudiante interesado el explorar las posibilidades que ofrecen este tipo de herramientas.

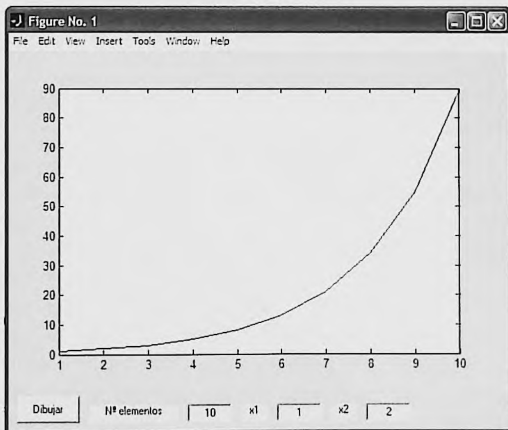


Figura 5.3: GUI para `ud4_fibo`

5.5.2. Ejemplo de GUI

Una interfaz gráfica de usuario (GUI) constará de un conjunto de ventanas y menús. Nos referiremos a cada una de esas ventanas como un formulario. Construiremos como ejemplo una GUI que constará únicamente de un formulario.

Básicamente en un formulario podremos poner los siguientes tipos de elementos.

- Botones. Al pulsarlos se invoca (se hace "callback") a alguna función o *script*.
- Etiquetas. Contienen texto informativo no modificable por el usuario de la GUI.
- Cajas. Permiten introducir valores para fijar el valor de determinadas variables.
- Gráficas. Podremos ubicar gráficas dentro de la ventana para representar gráficamente vectores o matrices.

Para crear cada uno de esos elementos dentro de una ventana existe el comando `uicontrol`, que asigna a un identificador el direccionamiento del elemento del tipo anterior que se haya creado mediante dicha orden. El primer elemento que crearemos será un botón, a cuyo identificador llamaremos `hboton`. La `h` procede de "handle" que es como se conocen este tipo de variables identificadoras en inglés. Cuando pulsemos el botón MATLAB ejecutará una secuencia de operaciones que hayamos guardado en un *script*. El botón va a tener el texto *Dibujar* dentro, y cuando lo pulsemos llamará al *script* `ud5_sfibo` que es quien realmente lanza el cálculo de los elementos de la sucesión, llamando (haciendo un `callback`) a la función `ud4_fibo`. El aspecto de la sentencia que crea el botón será el siguiente:

```
hboton=uicontrol('String','Dibujar','Position',[10 10 70 30],'Callback',...
      'ud5_sfibo');
```

Es interesante apreciar en esa línea la utilización de ... para romper una línea de código que continúa en la línea siguiente.

Los argumentos de `uicontrol` podemos ver que están organizados en parejas. En cada pareja tenemos el nombre de la propiedad y el valor que toma. en este caso:

- `String`, cadena de caracteres en inglés, es la propiedad que se refiere a lo que se puede leer en el elemento (*Dibujar* en este caso).
- `Position`, posición que ocupará el elemento. Para ubicar los elementos dentro de una ventana-formulario se utilizarán cuatro valores: x del extremo inferior izquierda del elemento e y del extremo inferior izquierda del elemento, longitud y altura del elemento. las coordenadas x, y se referirán al extremo inferior izquierda de dicha ventana que es el origen considerado para las mismas. Los valores de esas coordenadas son píxeles.
- `Callback`, *script* que será invocado al pulsar el botón.

Crearemos ahora las etiquetas con las que se indicará al usuario la variable que se está introduciendo en la caja adyacente y crearemos también dichas cajas. Veamos con un poco detalle la pareja (etiqueta, caja) correspondiente al primer término de la sucesión $x1$.

```
hx1txt =uicontrol('Style','text','String','x1','Position',[260 10 30 20]);
hx1    =uicontrol('Style','edit','String',1,'Position',[300 10 50 20]);
```

Las diferencias respecto al botón son que estos elementos no tienen un "callback"pero el segundo de ellos (la caja) es editable, o sea podemos cambiar el valor que está guardado en el elemento. Eso se traduce en que la propiedad `Style` toma el valor `edit`.

Finalmente, se define la posición de la gráfica en fracciones de la unidad con respecto al tamaño de la ventana, para que sea escalable con ésta.

```
subplot('position',[0.1 0.2 0.8 0.7])
```

En conjunto, este ejemplo queda así:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ud5_wfibo.m
% Interface para el programa ud5_sfibo
% Primera vez que usamos un interfaz con ventanas
clear all;
close all;
hboton=uicontrol('String','Dibujar','Position',[10 10 70 30],...
        'Callback','ud5_sfibo');
% Dibujamos las etiquetas y las cajas para leer valores
hntxt  =uicontrol('Style','text','String','N° de términos',...
        'Position',[90 10 100 20]);
hn     =uicontrol('Style','edit','String',10,'Position',[200 10 50 20]);
hx1txt =uicontrol('Style','text','String','x1','Position',[260 10 30 20]);
hx1    =uicontrol('Style','edit','String',1,'Position',[300 10 50 20]);
hx2txt =uicontrol('Style','text','String','x2','Position',[360 10 30 20]);
hx2    =uicontrol('Style','edit','String',2,'Position',[400 10 50 20]);
% Definimos el área de dibujo.
subplot('position',[0.1 0.2 0.8 0.7])
```

Las dos primeras sentencias de este *script* son `clear all` (ya estudiada) y `close all`. La sentencia `close all` se utiliza habitualmente en programas que crean gráficas o ventanas para que cierre todas las ventanas de este tipo abiertas y evite conflicto con las que abrirá el programa cuando se ejecute.

Ahora codificamos el *callback* `ud5_sfibo` que invocado en la sentencia `uicontrol` del *script* previo. Para ello usamos la sentencia `get`, que obtiene de cada `handle` una determinada propiedad, en este caso los valores de cada una de de las cajas. Esa propiedad es un texto, una cadena de caracteres. Durante este curso, no hemos prestado atención a las cadenas de caracteres pues son variables esencialmente diferentes a las numéricas y creemos menos esenciales para un curso introductorio. Para hacer la equivalencia entre una cadena de caracteres y una variable numérica se usa la función propia de MATLAB `str2num`, (de *string to number* en inglés):

```
x1=str2num(get(hx1,'String'));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ud5_sfibo.m
% callback de ud5_wfibo.m
n=str2num(get(hn,'String'));
x1=str2num(get(hx1,'String')); %
x2=str2num(get(hx2,'String'));
% Llamamos a ud4_fibo para calcular los términos
v=ud4_fibo(x1,x2,n);
% Dibujamos el vector v
plot(v);
```

Para ejecutar este ejemplo deberemos teclear `ud5_wfibo` desde la ventana de comandos.

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 5.28 *Copia los archivos* `ud5_sfibo.m`, `ud5_wfibo.m` *y* `ud4_fibo.m` *a tu carpeta de trabajo. Ejecuta en MATLAB* `ud5_wfibo`. *En particular, pruébalo con* $x_1 = 1$ *y* $x_2 = -0.6180$.

Ejercicio 5.29 *Modifica* `ud5_wfibo` *para que los huecos para los datos estén a la izquierda de la ventana y el botón debajo de esos huecos.*

Ejercicio 5.30 *Crea una función que reciba dos valores reales* a, b, *con* $a < b$ *y un número natural* n. *Se construirá dentro de la función un vector* $\mathbf{x}$ *de* $n+1$ *puntos equiespaciados entre* a *y* b, *incluyendo a ambos. La función devolverá el vector* $\mathbf{x}$ *y un vector* **yseno** *resultado evaluar la función seno en las componentes del vector* $\mathbf{x}$ *(solución en apéndice).*

Ejercicio 5.31 *Crea una GUI que llame a la función codificada en el ejercicio 5.30 y presente la gráfica correspondiente.*

Ejercicio 5.32 *Crea una función que reciba dos valores reales* a, b, *con* $a < b$ *y un número natural* n. *Se construirá dentro de la función un vector* $\mathbf{x}$ *de* $n+1$ *puntos equiespaciados entre* a *y* b, *incluyendo a ambos. La función devolverá ese vector* $\mathbf{x}$ *y dos vectores* **yseno** *e* **ytaylor**. **yseno** *será el resultado evaluar la función seno en las componentes de* $\mathbf{x}$. **ytaylor** *será tal que para cada índice* i, $ytaylor_i$ *sea la aproximación de* $\sin(x_i)$ *dada por:*

$$ytaylor_i = x_i - \frac{x_i^3}{3!} + \frac{x_i^5}{5!} - \frac{x_i^7}{7!}.$$

(solución en apéndice)

Ejercicio 5.33 *Crea una GUI que llame a la función codificada en el ejercicio 5.32 y genere la gráfica correspondiente a ambas funciones, superponiéndolas (solución en apéndice). Estudia la variante de hacerlo pidiendo ayuda sobre el comando* `hold`, *así como la posibilidad de incluir una leyenda con el comando* `legend`.

Ejercicio 5.34 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

5.6. Proyectos de programación

5.6.1. General

Estos proyectos pueden plantearse como exámenes o como trabajos para realizar individualmente o en equipo. Involucran una buena parte de las ideas estudiadas hasta este punto y precisan de la codificación de diferentes funciones, las cuales permiten finalmente disponer de una aplicación integral que resuelve un determinado problema. Tiene sentido plantearlos a partir de este capítulo pues ya se disponen de herramientas suficientes como para diseñar y construir auténticos programas. Estos proyectos se adaptan de modo natural a las metodologías que valoran el trabajo del estudiante a lo largo del curso, muy propias del cambio de paradigma ECTS.

5.6.2. Normas para la realización de los proyectos

Los proyectos se han de codificar cumpliendo las siguientes especificaciones generales:

1. Las funciones se ubicarán en ficheros independientes, cuyo nombre coincidirá con el de la función a falta por supuesto de la extensión .m.

2. Deberá estar escrito en MATLAB. No se permitirá usar funciones propias de MATLAB no estudiadas durante el curso, ni conceptos o estructuras correspondientes a capítulos que todavía no se hayan explicado.

3. Las variables que conceptualmente sean vectores o matrices serán codificadas como vectores o matrices. Aquellas variables que conceptualmente sean escalares serán codificadas como escalares.

4. Deberá constar de un programa ("script") principal, que servirá como soporte de todas las funciones que se vayan construyendo. La idea que hay detrás de esto es que esas funciones que habéis escrito pudiesen servir como partes de un programa más grande.

5. Cuando en el enunciado se exija la utilización de una función estudiada en el libro o incluida en el enunciado, ésta no podrá ser modificada. Deberá ser usada tal y como está escrita en los apuntes o en el enunciado.

6. El conjunto de variables de entrada y salida será el mínimo imprescindible.

7. Caso de que sea necesario por complicado o enrevesado, se explicará de modo preciso mediante breves comentarios el desarrollo de los diferentes procesos que realizan estas funciones, así como una descripción de sus entradas y salidas.

8. Salvo que se indique explícitamente en el enunciado, se podrán usar todos los ejemplos ya estudiados.

9. Salvo que se indique explícitamente en el enunciado, se podrán crear funciones auxiliares, si se considera que pueden simplificar el trabajo.

5.6.3. Medidas sobre la ruta de un velero "Volvo Ocean Race"

Se trata de construir un programa que reciba la ruta (proporcionada por el GPS) de un barco de la "Volvo Ocean Race" y analice los datos para obtener diversas medidas.

El GPS emite la posición del barco (longitud y latitud)[5] cada jornada. La ruta se recibirá como dos vectores **x** e **y**; el vector **x** contendrá la longitud del barco medida al comienzo de cada jornada y el vector **y** la latitud. Los últimos elementos serán la posición del barco al final de la última jornada (es decir, en la meta).

Como ejemplo para probar el programa, tomaremos (figura 5.4)

$$\mathbf{x} = (0.5, -0.1, -1.4, -0.8, -0.3, -1.3, -2.4),\ \mathbf{y} = (40.5, 39.8, 38.6, 38.2, 37.4, 36.9, 35.8).$$

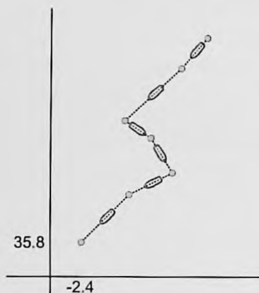


Figura 5.4: Trayectoria de la ruta ejemplo (proyecto 5.6.3).

1. Crea una función llamada `fdist` que reciba dos numeros naturales, i y f y dos vectores **x** e **y** y devuelva

$$\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}.$$

Es decir, la distancia entre la posición a la jornada i y la posición a la jornada f, suponiendo que las coordenadas fuesen cartesianas. En nuestro ejemplo, si $i = 1$ y $f = 5$, deberá devolver

[5]Supondremos que podemos asimilar la zona de estudio a un plano y que la longitud y latitud no son ángulos sino coordenadas cartesianas.

3.2016 (distancia del primer punto $(0.5, 40.5)$ al quinto $(-0.3, 37.4)$).

2. Crea una función llamada `fdorigen` que reciba los dos vectores, `x` e `y`, y devuelva un vector tal que su componente i-ésima sea la distancia al origen después de i jornadas. Sea **vdist** ese vector. Para calcular la distancia entre cada punto y el origen, utilizará la función `fdist`. En nuestro ejemplo debería devolver $(0.92, 2.69, 2.64, 3.2, 4.02, 5.52)$.

3. Crea una función llamada `fanalisis` que reciba los dos vectores `x` e `y` y devuelva en qué jornada la distancia al origen superó la distancia media al origen. Hazlo con el siguiente procedimiento:

   - Llama a `fdorigen` para obtener el vector de las distancias recorridas cada jornada.
   - Llama a `ud4_fmedia` para calcular la media de dicho vector.
   - Recorre el vector de distancias distancias **vdist**; cuando encuentres una que sea mayor que la media, guarda el índice y rompe el bucle. Ese es el valor que ha de devolver la función.

   Así, en nuestro ejemplo, la media de distancias es 3.1667, luego tenemos que devolver 4, número de jornada en la que la distancia al origen superó ese valor.

4. Crea un programa ("script") `principal.m` que pida por teclado el número de datos del GPS que se van a introducir, pida cada una de las latitudes, pida cada una de las longitudes, llame a la función `fanalisis` e imprima en qué jornada la distancia al origen superó la distancia media al origen.

5. Modifica la función `fanalisis` para que, además de lo anterior, devuelva en otra variable el número de posiciones "casi alineadas" que hay. Consideraremos que tres posiciones $i < j < k$ están "casi alineadas" cuando $d(i,k) > 0.999 \cdot (d(i,j) + d(j,k))$. Calcula las distancias entre las posiciones usando `fdist`. En el ejemplo considerado, debería devolver 2, que se corresponde a las posiciones 1-4-7 y 2-4-6 (ver figura 5.4).

6. Modifica el programa `principal.m` para que, además de todo lo anterior, devuelva el número de posiciones "casi alineadas".

7. Modifica la función `fanalisis` para que, además de todo lo anterior, devuelva el máximo número de jornadas consecutivas en las que la distancia al origen siempre aumentó.

   En el ejemplo serían 4 jornadas (3, 4, 5 y 6), pues entre la jornada 3 (distancia al origen 2.64) y la jornada 6 (distancia al origen 5.52) la distancia al origen siempre creció.

8. Modifica el programa `principal.m` para que, además de todo lo anterior, devuelva el máximo número de jornadas consecutivas en las que la distancia al origen siempre aumentó y dibuje la trayectoria del barco.

9. Rehacer el proyecto para que lea la información de entrada de un archivo de datos y para que imprima toda la información en un archivo de salida con el formato que se considere necesario.

10. Rehacer el proyecto creando una GUI que ejecute todo el programa, incorporando una llamada al archivo de datos en el que se encuentren los datos de entrada y presentando en la ventana correspondiente el gráfico del apartado 8.

11. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

5.6.4. Decodificación de un mensaje

Se trata de construir un programa que reciba un mensaje codificado y lo analice. El mensaje vendrá dado como un vector de números enteros con valores entre 1 y 27. Como ejemplo para probar el programa, tomaremos

$$\mathbf{mensaje} = (17, 11, 23, 12, 11, 15, 23, 23, 18, 15, 21, 13)$$

1. Crea una función llamada `frepite` que reciba un vector v y un número n devuelva el número de veces que aparece n en v. En nuestro ejemplo, si $n = 11$ debería devolver 2.

2. Crea una función `fanalisis` que reciba un vector y devuelva devuelva el número de primos que hay en el vector. Para ver si un número es primo se usará la función `ud3_fesprimo`. Así en nuestro ejemplo, debería devolver 7.

3. Modifica `fanalisis` para que, además de lo anterior, devuelva en otra variable el número de elementos que aparecen una única vez. Para ello recorre el vector y para cada elemento del vector cuenta uno cada vez que el número de veces (calculado con `frepite`) sea igual a 1. En nuestro ejemplo, el número de elementos que aparecen una única vez es 5 (17, 12, 18, 21 y 13).

4. Crea un programa `principal.m` que pida por teclado el número de elementos que tiene el mensaje, pida cada uno de los elementos, llame a la función `fanalisis` e imprima por pantalla el número de elementos que son primos y el número de elementos que aparecen una sola vez.

5. Modifica `fanalisis` para que, además de lo anterior, devuelva en otra variable el número primo que más veces se repite. Por ejemplo, puedes crear un vector con los números primos y calcular el que más se repite llamando para cada elemento a la función `frepite`. En el ejemplo devolverá 23. Pruébalo también con $(3, 4, 23, 4, 23, 4, 29)$, para el que también debería devolver 23.

6. Modifica el programa `principal.m` para que, además de todo lo anterior, imprima por pantalla el número primo que más veces se repite.

7. Crea una función `ftrozo`, que reciba dos vectores, uno v con el mensaje y otro w y devuelva 1 si w está incluido en v y 0 en caso contrario. Por ejemplo, si

$$\mathbf{v} = (17, 11, 23, 12, 11, 15, 23, 23, 18, 15, 21, 13), \ \mathbf{w} = (11, 23, 12, 11),$$

devolverá 1 y si

$$\mathbf{v} = (17, 11, 23, 12, 11, 15, 23, 23, 18, 15, 21, 13), \ \mathbf{w} = (17, 12, 23, 12, 11),$$

devolverá 0.

8. Modifica el programa `principal.m` para que, además de todo lo anterior, pida un segundo mensaje, llame a `ftrozo` e imprima por pantalla si el segundo mensaje está incluido en el inicial o no.

9. Rehacer el proyecto para que lea la información de entrada de un archivo de datos y para que imprima toda la información en un archivo de salida con el formato que se considere necesario.

10. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

5.6.5. Propiedades geométricas de una poligonal

Se tiene una ristra de $n+1$ puntos sumergidos en el espacio. Dichos puntos tomados consecutivamente forman una poligonal. Se pide construir una función de nombre `fpoligonal` que reciba las componentes de esos puntos como tres vectores $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, de $(n+1)$ componentes y que devuelva otros tres vectores con los puntos medios de los tramos de esa poligonal, y devuelva también el índice del tramo de mayor longitud y la longitud de dicho tramo así como las coordenadas del centro de gravedad de la poligonal. Para el cálculo del índice del máximo se utilizará la función `ud4_fimax`. Además `fpoligonal` llamará a la función `ud3_fesprimo` estudiada en el curso para averiguar si n es un número primo, devolviendo a su vez este resultado.

Se construirá también un programa `principal.m` que lea de un archivo la ristra de puntos, llame a `fpoligonal` y escriba en otro archivo los resultados.

Se pide aplicar el programa al siguiente ejemplo:

i	x_i	y_i	z_i
0	0	1	2
1	5	0.1	20
2	-10	-0.9	-2
3	20	0.9	2

El programa se codificará siguiendo la siguiente secuencia, y no se pasará hasta el siguiente paso de la secuencia hasta que no funcione el paso actual. La secuencia es la siguiente:

1. Creación del archivo de entrada, `NODOS.DAT` mediante el editor que elijas, correspondiente al ejemplo propuesto.

2. Implementación de un programa `principal.m` que lea el archivo de datos `NODOS.DAT`.

3. Implementación de la rutina en la que se calculan los vectores de puntos medios devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

4. Implementación en esa misma rutina de la comprobación de si n es o no número primo llamando a `ud3_fesprimo` y devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

5. Cálculo del índice del lado de longitud máxima utilizando la rutina `ud4_fimax`, y devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

6. Cálculo de la longitud de ese lado, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

7. Cálculo del centro de gravedad de la poligonal, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

8. Rehacer el proyecto creando una GUI que ejecute todo el programa, incorporando una llamada al archivo de datos en el que se encuentren los datos de entrada y presentando en la ventana correspondiente el gráfico de la poligonal, puntos medios, centro de gravedad, etc.

9. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

Para comprobación, el fichero de salida que se obtiene para la entrada ejemplo sugerida es:

```
xm[1]=2.500000
ym[1]=0.550000
zm[1]=11.000000
xm[2]=-2.500000
ym[2]=-0.400000
zm[2]=9.000000
xm[3]=5.000000
ym[3]=0.000000
zm[3]=0.000000
imax=3
L=30.318972
3 es primo (0 no, 1 si)=1
La coordenada z del ultimo punto es 2.000000
el cdg de la poligonal es 1.741004,-0.004910,5.888191
```

5.6.6. Polinomio a trozos

Se considera un polinomio a trozos de grado 2 y de n tramos. El enganche se produce en $n-1$ puntos distribuidos de modo uniforme entre dos valores reales A y B. Los cuatro primeros tramos de la figura 5.5 podrían corresponder al ejemplo que vamos a tener de referencia para probar el programa. Habrá puntos (tipo =-1) en los que el enganche se produce sin continuidad de la función a trozos, como el punto x_1 de la figura ejemplo, otros en los que la derivada no es continua pero la función sí (tipo =0), como en x_3, y otros en los q hay continuidad de la función y de la derivada

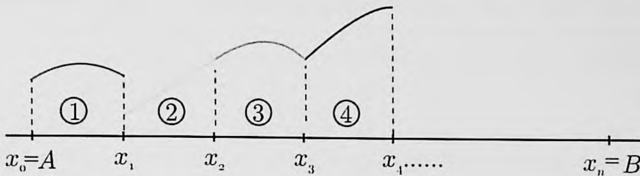


Figura 5.5: Proyecto 5.6.6, gráfica tipo del polinomio a trozos a estudiar

(tipo =1) como x_2.

Para escribir todos los tramos tenemos tres vectores de dimensión n, que llamaremos **a**, **b**, y **c**. De este modo, el polinomio correspondiente al tramo i se escribirá como:

$$a_i t^2 + b_i t + c_i, \quad t \in [x_{i-1}, x_i)$$

En el ejemplo que utilizaremos para probar el código, el primer tramo es el polinomio de segundo grado:

$$-t^2 + 0.5t + 0.9375, \quad t \in [0, 0.5)$$

El segundo tramo será, el polinomio

$$0t^2 + t + 0, \quad t \in [0.5, 1.0)$$

y así sucesivamente.

Si no hay continuidad entre el tramo i y el $i+1$ en el punto x_i tendremos que:

$$a_i x_i^2 + b_i x_i + c_i \neq a_{i+1} x_i^2 + b_{i+1} x_i + c_{i+1}$$

Si hay continuidad de la función pero no de la derivada tendremos que:

$$a_i x_i^2 + b_i x_i + c_i = a_{i+1} x_i^2 + b_{i+1} x_i + c_{i+1}$$

$$2a_i x_i + b_i \neq 2a_{i+1} x_i + b_{i+1}$$

Si hay continuidad de la función y de la derivada tendremos que:

$$a_i x_i^2 + b_i x_i + c_i = a_{i+1} x_i^2 + b_{i+1} x_i + c_{i+1}$$

$$2a_i x_i + b_i = 2a_{i+1} x_i + b_{i+1}$$

Cuando nos referimos a que dos valores r y s genéricos son iguales, es porque la diferencia entre ellos en valor absoluto es menor que una precisión $prec$ que se fijará como dato.

Vamos a trabajar sobre estas ideas, y vamos también a evaluar el polinomio a trozos para un valor

concreto $u \in (A, B)$. También lo vamos a evaluar para una partición equiespaciada de $m+1$ puntos definidos entre A y B que guardaremos en un vector t para después generar un fichero con formato MATLAB con las imágenes de las componentes de t que llamaremos y.

Se pide:

1. Codificar un programa principal PRINCIPAL.m que lea un archivo ENTRADA.DAT en el que se tengan todos los valores citados en el enunciado y que genere un archivo de salida SALIDA.DAT en el que se vuelquen todos esos datos.

2. Crear el archivo ENTRADA.DAT correspondiente al apartado anterior para el siguiente ejemplo, con gráficas similar a la figura 5.5:

$$A = 0, B = 2, n = 4, m = 100, prec = 0.001, u = 0.38$$

$$a_1 = -1, b_1 = 0.5, c_1 = 0.9375$$

$$a_2 = 0, b_2 = 1, c_2 = 0$$

$$a_3 = -2, b_3 = 5, c_3 = -2$$

$$a_4 = 1, b_4 = -2, c_4 = 1.75$$

3. Codificar una función de nombre UBICA.m, que indique el índice del tramo correspondiente a un valor cualquiera w. Por ejemplo, el tramo correspondiente a 0.38 es el 1.

4. Codificar una función de nombre EVALUA.m, que evalúe el polinomio a trozos en un valor cualquiera w, para lo cual llamará a la función UBICA, definirá de modo adecuado el polinomio correspondiente a ese tramo y llamará después a la función ud4_fevalua para calcular el valor en w.

5. Codificar una función FPOLTROZOS para que utilizando la función EVALUA.m, calcule el valor en u del polinomio a trozos. Para el ejemplo, el resultado tendría que ser 0.9831.

6. Modificar PRINCIPAL.m para que llame a FPOLTROZOS y escriba el resultado en un fichero SALIDA.DAT.

7. Modificar la función FPOLTROZOS para calcular primero el vector de abscisas t y utilizando la función EVALUA.m, calcule el vector de $m+1$ ordenadas y resultado de evaluar el polinomio a trozos en los $m+1$ puntos de t.

8. Modificar PRINCIPAL.m para que llame a esta función y cree un nuevo fichero con extensión .m, SALIDA.m, con el formato siguiente:

```
t(1)=0.000000;
y(1)=0.937500;
t(2)=0.020000;
y(2)=0.947100;
t(3)=0.040000;
```

```
y(3)=0.955900;
t(4)=0.060000;
....................
....................
t(101)=2.000000;
y(101)=1.750000;
plot(t,y)
shg
```

La figura 5.6 es la que se obtiene en MATLAB al ejecutar el script `SALIDA.m`.

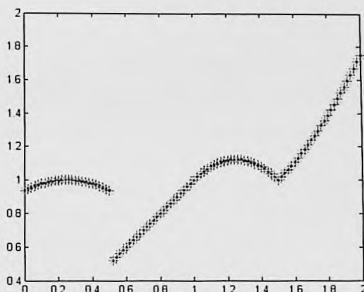


Figura 5.6: Figura obtenida para el ejemplo del proyecto 5.6.6

9. Modificar `FPOLTROZOS`, para que calcule un vector **vtipo** de $n-1$ componentes que indique si cada uno de los enganches es de tipo -1,0, o 1. En el ejemplo, el vector **vtipo** valdrá $(-1, 1, 0)$.

10. Modificar `PRINCIPAL.m` para que llame a `FPOLTROZOS` y escriba el resultado **vtipo** en el fichero `SALIDA.DAT`.

11. Modificar la función `FPOLTROZOS`, para que cuente cuantos elementos de **vtipo** son mayores o iguales que cero. Sea p ese número. La función devolverá si p es o no primo (0 si no es primo, 1 si es primo). En el ejemplo $p = 2$ que es primo.

12. Modificar `PRINCIPAL.m` para que llame a `FPOLTROZOS` y añada el resultado al fichero `SALIDA.DAT`.

13. Rehacer el proyecto creando una GUI que ejecute todo el programa, incorporando una llamada al archivo de datos en el que se encuentren los datos de entrada y presentando en la ventana correspondiente el gráfico del polinomio a trozos (figura 5.6).

14. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

UNIVERSIDAD TECNICA DEL NORTE

Capítulo 6

Matrices

6.1. General

En el capítulo que ahora comienza se generaliza el concepto de vectores o "arrays" de datos, ampliándolo a dos o más dimensiones. Al igual que sucedió en el capítulo dedicado a los vectores (4) tampoco aquí hay apenas órdenes nuevas y por lo tanto nos concentraremos en generalizar los algoritmos más interesantes, los cuales permiten resolver problemas más complejos que aquellos y similares a los que aparecen en diversos ámbitos profesionales. Al igual que los vectores, las matrices están muy relacionadas con los bucles que hemos estudiado en el capítulo 3. Veremos muy pronto que mediante bucles nos podremos mover a través de las nuevas estructuras matriciales. Vamos a estudiar de hecho un modo más directo de trabajar con los bucles más habituales, apoyándonos en la sentencia `for`. Dedicaremos la primera sección de este tema a los bucles de este tipo.

6.2. Bucles for

Hasta ahora hemos venido escribiendo los bucles de forma que el índice del bucle se ha ido incrementando explícitamente, finalizando en el momento que se cumplía una condición asociada a la sentencia `while`. La gran ventaja de los bucles escritos mediante el uso de la sentencia `while` frente a la otra forma de escribir un bucle (sentencia `for`) radica en que el índice del bucle es una variable más del programa y su valor es controlado en todo momento por el programador. En los bucles `for`, las iteraciones se realizan mientras un índice recorre una serie de valores, y este índice ni siquiera necesita ser actualizado dentro del bucle. Por tanto, la sintaxis es muy simple.

Las desventajas que tiene el uso de la sentencia `while` frente a `for` son que tiene una sintaxis más complicada y que por tanto requiere más esfuerzo codificar, lo cual implica también que es más fácil que el código tenga errores, "bugs". Además, el ciclo `while` es más lento que el `for`, aunque es más general dado que, como comentábamos más arriba, el índice del bucle es una variable como cualquier otra, sobre la que se tiene un absoluto control. Finalmente, la

condición de parada puede ser todo lo compleja que queramos y no referirse únicamente a un índice (ya vimos algunos de estos casos en la sección 3.6). En este capítulo vamos a introducir la sentencia `for` para realizar los bucles. Su estructura es muy sencilla:

```
....
....
for i=l:m:n
    cuerpo del bucle
end
....
....
```

Donde el índice i tomará valores desde l hasta n, inclusive, mediante saltos de m unidades. En los casos donde se desea que el índice se incremente de uno en uno se puede suprimir el valor intermedio. Está forma de definir una secuencia ya la vimos en el apartado 1.3 del tutorial de MATLAB.

Vamos a rehacer el ejemplo `ud3_fsuma` con el que introdujimos la sentencia `while` utilizando ahora `for` para implementar el bucle.

```
% ud6_fsuma.m
% Suma de todos los naturales entre 1 y n
% primer bucle for
function suma=ud6_fsuma(n)
suma=0;
for i=1:n
    suma=suma+i;
end
```

Se puede observar que no es necesario actualizar el valor del índice en el cuerpo del bucle dado que éste se actualiza directamente en su definición como secuencia (`i=1:n`). Además aunque cambiemos su valor en el cuerpo del bucle, cuando pasa otra vez por el principio del mismo, toma el siguiente valor que le corresponde en la secuencia. Esta es una diferencia fundamental respecto a los bucles `while` como ya comentamos más arriba. Hay también cierta diferencia en velocidad de ejecución entre ambos tipos de bucles para el mismo problema, aunque dejaremos la verificación empírica de este extremo para el capítulo 7.

Los ejercicios correspondientes a esta sección consisten básicamente en reescribir, si es posible, con esta nueva sintaxis, algunas de las funciones estrella del curso.

Ejercicio 6.1 *Crea una carpeta llamada* `ud6` *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 6.*

Ejercicio 6.2 *Prueba la función* `ud6_fsuma` *para diferentes valores de* `n`.

Ejercicio 6.3 *Codifica una función análoga a* `ud3_fcuentadiv` *pero utilizando un bucle tipo* `for`.

Ejercicio 6.4 *Codifica una función análoga a* `ud3_fesprimo` *pero utilizando un bucle tipo* `for` *(solución en apéndice).*

Ejercicio 6.5 *Codifica una función análoga a* `ud3_fdigitos` *pero utilizando un bucle tipo* `for`.

Ejercicio 6.6 *Codifica una función análoga a* `ud3_ffibonacci` *pero utilizando un bucle tipo* `for`.

Ejercicio 6.7 *Codifica una función análoga a* `ud4_fimax` *pero utilizando un bucle tipo* `for`.

Ejercicio 6.8 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

6.3. Matrices como argumentos de funciones

Este ejemplo muestra una primera función en la que se usa una matriz como argumento. Como sabemos, desde que hicimos el tutorial (capítulo 1), podemos preguntar a un vector o a una matriz su tamaño. Para ello disponemos de la orden `size`. Las matrices son estructuras de datos con dos dimensiones, filas y columnas. La función `size` nos devolverá por tanto 2 números que serán respectivamente el número de filas y el de columnas de la matriz. De hecho cualquier variable en MATLAB es en realidad una potencial matriz sin dimensiones fijas.

Dado que las matrices transportan sus dimensiones, suele suceder que la primera línea de una función que tenga una matriz como argumento de entrada sea la asignación de las dimensiones de la matriz a unas nuevas variables m y n que nos van a permitir montar los bucles para movernos por la matriz. En general las matrices no tienen por qué ser cuadradas ($m = n$) y por tanto debemos suponer que salvo que se indique lo contrario $m \neq n$.

Es muy importante entender que movernos por una matriz implica la necesidad de un doble bucle con un índice que recorrerá las filas mientras que otro índice recorrerá las columnas. Las diferentes combinaciones de ambos índices nos permitirán referirnos a los distintos elementos de la matriz.

En MATLAB las matrices se estiran y encogen sin ninguna limitación ni en el tamaño ni en las dimensiones. Si intentamos colocar en la fila 9 y la columna 4 de una matriz inexistente el número 1, tendremos el siguiente resultado:

```
>> foo(9,4)=1
foo =
0    0    0    0
0    0    0    0
```

```
0    0    0    0
0    0    0    0
0    0    0    0
0    0    0    0
0    0    0    0
0    0    0    0
0    0    0    1
```

Esta característica de MATLAB facilita la tarea de escribir los algoritmos pero en determinadas ocasiones puede dar lugar a que se bloquee un espacio grande de memoria sin necesidad.

En el siguiente ejemplo se desea obtener el valor medio de los elementos de una matriz, de modo análogo a como obtuvimos el de un vector mediante la función `ud4_fmedia`. Para ello tenemos que realizar una suma a la vez que vamos recorriendo todos los elementos de dicha matriz mediante un doble bucle. Lo primero, inicializamos a 0 la variable $suma$ y a continuación mediante un bucle de filas y otro de columnas vamos incrementando el valor de la suma parcial con los elementos recorridos de la matriz. Finalmente dividimos por el numero de elementos $n \cdot m$ para tener la media.

```
% ud6_fmedia.m
% ud6_fmedia(M) recibe una matriz M y
% devuelve el valor medio de sus elementos.
% Primera función con argumento de entrada una matriz
function media=ud6_fmedia(M)
[n,m]=size(M);  % n y m seran las dimensiones de la matriz M
suma=0; % Inicializamos la suma a 0
for i=1:n
    for j=1:m
        suma=suma+M(i,j); % Sumamos el elemento de Mij
    end
end
media=suma/(n*m); % Hallamos la media dividiendo entre n*m
```

Para probar esta función podemos introducir en la línea de comandos la siguiente orden:

```
>> ud6_fmedia([1 2 3; 4 5 6; 7 8 9])
ans =
     5
```

Otra forma de encontrar la media es utilizar el comando `mean` de MATLAB. Cuando tiene como argumento de entrada una matriz, `mean` devuelve la media de las columnas:

```
>> mean([1 2 3; 4 5 6; 7 8 9])
ans =
     4     5     6
```

Si la aplicamos a un vector, devuelve la media del vector:

```
>> mean([4 5 6])
ans =
      5
```

Por tanto, para obtener la media de la matriz utilizando el comando `mean`, tendremos que aplicarla dos veces, de este modo:

```
>> mean(mean([1 2 3; 4 5 6; 7 8 9]))
ans =
      5
```

Vamos a construir ahora una función un poco más interesante. Se trata de codificar una función que reciba una matriz de naturales A y devuelva otra de igual tamaño B en la que sean nulos todos los elementos no primos de la primera. Para ello, utilizamos un doble bucle para recorrer la matriz A, pasando cada elemento por la función `ud3_fesprimo`, sabiendo que esa función devuelve 1 para argumentos de entrada primos y 0 para no primos.

```
function B=ud6_fesprimo(A)
[m,n]=size(A);
B=zeros(m,n);
for i=1:m
    for j=1:n
        B(i,j)=ud3_fesprimo(A(i,j))*A(i,j);
    end
end
```

Por ejemplo, si la matriz es:

$$A = \begin{pmatrix} 23 & 6 & 9 \\ 13 & 14 & 11 \\ 12 & 14 & 2 \end{pmatrix}$$

el resultado será entonces:

$$B = \begin{pmatrix} 23 & 0 & 0 \\ 13 & 0 & 11 \\ 0 & 0 & 2 \end{pmatrix}$$

algo que podemos comprobar fácilmente haciendo una llamada a la función en cuestión por línea de comandos de MATLAB:

```
>> ud6_fesprimo([23 6 9; 13 14  11; 12 14  2])
ans =
    23     0     0
    13     0    11
     0     0     2
```

Dado que hemos trabajado con archivos ya en el capítulo 5, podemos aplicar lo aprendido allí ahora para cargar la matriz a analizar desde un fichero de datos mediante un *script* desde el que invoquemos la función que acabamos de codificar `ud6_fesprimo`. Para ello entramos en el editor de MATLAB, creamos un fichero nuevo, y escribimos estas tres líneas

```
23  6  9
13 14 11
12 14  2
```

y guardamos el archivo como `input.dat`.

Ahora creamos un nuevo archivo desde el editor de MATLAB, que será el *script* en cuestión, el cual salvamos con el nombre `ud6_sesprimo`:

```
% ud6_sesprimo
A=load('input.dat');
B=ud6_fesprimo(A);
save output.dat B -ascii
```

En la primera línea cargamos el contenido del archivo `input.dat` y lo asignamos a la variable A, la cual pasamos a la función `ud6_fesprimo`. La salida B la guardamos con formato ASCII en el archivo de salida `output.dat`.

Los ejercicios correspondientes a estos dos ejemplos son los siguientes:

Ejercicio 6.9 *Prueba la función* `ud6_fmedia` *para diferentes matrices.*

Ejercicio 6.10 *Crea una función que reciba una matriz de números enteros, por hipótesis, y devuelva otra matriz que contenga tan sólo los números pares de la anterior, siendo el resto 0.*

Ejercicio 6.11 *Crea una función que reciba una matriz de números naturales A y devuelva un vector formado por los elementos de la matriz que sean múltiplos de su traza (suma de los elementos de la diagonal principal). En caso de no haber ningún múltiplo devolverá un 0. Por ejemplo, si la matriz es:*

$$A = \begin{pmatrix} 2 & 6 & 1 \\ 1 & 1 & 12 \end{pmatrix}$$

su traza es 3, y el vector resultado sería $\mathbf{u} = (6, 12)$.

Ejercicio 6.12 *Crea una función que reciba una matriz y devuelva su traza (suma de los elementos de la diagonal principal) si es cuadrada y cero si no lo es. Devolverá también una variable* `flag` *que ha de valer 1 si la matriz es cuadrada y 0 si no lo es (solución en apéndice).*

Ejercicio 6.13 *Crea una función que reciba una matriz M y devuelva una variable* `flag` *que ha de valer 1 si la matriz es diagonal y 0 si no lo es. Una matriz diagonal ha de tener nulos todos los elementos que no estén en la diagonal principal (solución en apéndice).*

Ejercicio 6.14 *Crea una función que reciba una matriz de números naturales y devuelva 1 si hay algún elemento primo y 0 en caso contrario. No se podrá llamar a ninguna función.*

Ejercicio 6.15 *Crea una función que reciba una matriz y devuelva su traspuesta. Para ello no podrá usar el operador de trasposición estudiado en la sección 1.3, sino que habrá de hacerse con un doble bucle (solución en apéndice).*

Ejercicio 6.16 *Crea una función que reciba una matriz y devuelva un vector con la suma de los elementos de cada columna. Además de hacerlo mediante un bucle anidado, se debe también codificar una variante utilizando del modo adecuado la función propia de MATLAB* `sum`. *Por ejemplo, si la matriz es:*

$$A = \begin{pmatrix} 2 & 6 & 1 \\ 1 & 1 & 12 \end{pmatrix},$$

el vector suma buscado será $(3, 7, 13)$.

Ejercicio 6.17 *Crea una función que reciba una matriz y devuelva un vector con la suma de los elementos de cada fila, utilizando del modo adecuado la función* `sum`.

Ejercicio 6.18 *Crea una función que reciba una matriz y devuelva su máximo. Además de hacerlo mediante un bucle anidado, se debe probar también la utilización de la función* `max`, *ya introducida en la sección 4.4.*

Ejercicio 6.19 *Crea una función que reciba una matriz y devuelva el máximo de la suma de los elementos de cada columna.*

Ejercicio 6.20 *Codifica una función que reciba una matriz A y devuelva los índices de fila y columna del elemento que esté más próximo a su media. Por ejemplo, si la matriz es:*

$$A = \begin{pmatrix} 8 & 6 & 1 \\ 3 & 4 & 1 \\ -7 & 6 & -15 \\ 2 & 1 & 12 \end{pmatrix}$$

su media es 1.8333 y el elemento más cercano es 2, que está en la posición (4,1) que serán los valores que devuelva la función.

Ejercicio 6.21 *Crea una función que reciba una matriz A y un número a y devuelva los índices de fila y columna del mayor número que resulte al restar a a la matriz y luego tomar el valor absoluto a todos sus elementos.*

Ejercicio 6.22 *Crea una función que reciba una matriz de números naturales, por hipótesis, y devuelva los índices de fila y columna del menor número primo que aparezca en la matriz. Caso de que no haya ninguno, devolverá 0 para ambos valores (solución en apéndice).*

Ejercicio 6.23 *Crea una función que reciba una matriz A y devuelva otra matriz B donde cada fila sea igual a suma de ella misma con la fila inferior. Es decir, la fila 1 de B será la suma de de las filas 1 y 2 de A y así sucesivamente. A la última fila no le sumaremos nada. Por ejemplo, si:*

$$A = \begin{pmatrix} 20 & 6 & 9 \\ 13 & 14 & 12 \\ 12 & 14 & 24 \end{pmatrix},$$

el resultado será entonces:

$$B = \begin{pmatrix} 33 & 20 & 21 \\ 25 & 28 & 36 \\ 12 & 14 & 24 \end{pmatrix}.$$

Se creará también un script que cargue la matriz de un fichero, llame a la función y escriba el resultado sobre un archivo de salida. Se creará manualmente el fichero de entrada (solución en apéndice).

Ejercicio 6.24 *Crea una función que reciba una matriz M y devuelva una variable* `flag` *que ha de valer -1 si la matriz es triangular inferior, 1 si es triangular superior, 2 si es diagonal y 0 si no es ninguna de las tres cosas.*

Ejercicio 6.25 *Crea una función que reciba una matriz A y devuelva una variable* `flag` *que será 1 si hay dos elementos de A iguales y 0 en caso contrario (solución en apéndice).*

Ejercicio 6.26 *Crea una función que reciba una matriz A y devuelva una variable* `flag` *que será 1 si hay dos elementos de A cuya suma sea igual a un tercer elemento de A.*

Ejercicio 6.27 *Codifica una función que reciba dos matrices A y B y devuelva una variable* `flag` *que ha de valer 1 si las matrices se pueden multiplicar y 0 en caso contrario. La función devolverá como otra salida una variable C que ha de valer el producto de A y B si esto es posible y 0 en caso de que no sea posible. No se podrá llamar a ninguna función y ha de realizarse el producto mediante un triple bucle. Por ejemplo, si:*

$$A = \begin{pmatrix} 2 & 6 & 1 \\ 1 & 1 & 12 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 1 \\ 2 & -1 \end{pmatrix},$$

las matrices no se pueden multiplicar y la función devolverá `flag=0`, $C = 0$. *Si intercambiamos los valores de ambas matrices, sí se pueden multiplicar y la función devolverá* `flag=1` *y*

$$C = \begin{pmatrix} 9 & 25 & 16 \\ 3 & 11 & -10 \end{pmatrix}$$

(solución en apéndice). Repetir el ejercicio realizando la multiplicación mediante el producto de matrices de MATLAB.

Ejercicio 6.28 *Crea una función que reciba dos matrices A y B y devuelva 1 si hay al menos algún elemento de ambas matrices que es igual y cero en caso contrario. Por ejemplo, si:*

$$A = \begin{pmatrix} 2 & 6 & 11 \\ 3 & -4 & 12 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 7 \\ 12 & -3 \end{pmatrix}$$

la función devolverá 1.

Ejercicio 6.29 *(Para valientes) Crea una función que reciba una matriz y calcule la matriz que se obtiene al sumar a cada elemento sus cuatro vecinos. Es decir el elemento* a_{ij} *pasará a ser* $a'_{ij} = a_{ij} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1}$*. Como caso particular habrá que tratar a los elementos que estén en las filas y columnas primera y última ya que sólo tendrán 3 vecinos, y también los elementos situados en las esquinas que tendrán exclusivamente dos vecinos. Por ejemplo, si la matriz es*

$$A = \begin{pmatrix} 2 & 6 & 1 & -1 \\ 3 & 4 & 8 & 6 \\ -7 & 6 & -15 & -1 \\ 1 & 1 & 12 & 2 \end{pmatrix}$$

devolverá

$$\begin{pmatrix} 11 & 13 & 14 & 6 \\ 2 & 27 & 4 & 12 \\ 3 & -11 & 10 & -8 \\ -5 & 20 & 0 & 13 \end{pmatrix}$$

(solución en apéndice)

Ejercicio 6.30 *Sea A una matriz de 2 columnas y un número indeterminado de filas, conteniendo en la primera columna el DNI de un trabajador y en la segunda su salario. Se trata de codificar una función que reciba dicha matriz y devuelva otra matriz B con tantas filas como la de entrada y 8 columnas. Cada columna contendrá, respectivamente, el número de billetes mínimo, necesarios para efectuar la paga, siendo la primera columna el numero de billetes de 500 euros y sucesivamente, 200, 100, 50, 20, 10, 5, y euros sueltos. Así por ejemplo: si el salario de un trabajador es: 1876, debería devolver en su correspondiente fila: 3,1,1,1,1,0,1,1. La función debe comprobar si la matriz de entrada es del tamaño adecuado y si no lo fuera se debería escribir un mensaje y dar una matriz nula como resultado.*

Se escribirá además un archivo de resultados, de nombre `paga.txt` *con el DNI, salario y billetes y euros.*

Ejercicio 6.31 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

6.4. Submatrices

Entenderemos por submatriz un subconjunto de los elementos de una cierta matriz que guarda su misma ordenación de filas y columnas. En el siguiente ejemplo se forma una submatriz a partir de las filas pares de una matriz dada.

```
% ud6_fsubmatpares.m
```

```
% ud6_fsubmatpares(M) recibe una matriz M y
% devuelve una submatriz formada por las filas de índice par de M.
% Con esta funcion se introduce el concepto de submatriz en Matlab
function subM=ud6_fsubmatpares(M)
[n,m]=size(M);  % n y m seran las filas y columnas de M
subM=[];
j=0;
for i=2:2:n
    j=j+1;
    subM(j,:)=M(i,:);
end
```

Es interesante observar que definimos `subM` por defecto como una matriz vacía (ver sección 1.3) que vamos rellenando. En la función `ud6_fsubmatpares` el criterio de selección de filas es sencillo (filas pares). De hecho, es tan sencillo que obtener una submatriz `subM` a partir de otra En la función `ud6_fsubmatpares` el criterio de selección de filas es sencillo (filas pares). De hecho, es tan sencillo que obtener una submatriz `M` tomando solo sus filas pares se puede conseguir con la siguiente instrucción, utilizando la potencia de la notación matricial de MATLAB :

```
>> subM=M(2:2:end,:)
```

El comando `end` detecta el final de la matriz en ese índice, lo que supone una simplificación adicional a la escritura de código.

En general el criterio para construir submatrices no tiene porque ser tan sencillo y en algunos casos podría obligarnos a incluir un condicional en el interior de uno o de ambos bucles. El hecho de incluir un condicional dentro de un bucle ralentiza mucho la velocidad del proceso y tan sólo se debe hacer cuando no haya ninguna otra alternativa. Se proponen los siguientes ejercicios relativos a este ejemplo:

Ejercicio 6.32 *Prueba la función* `ud6_fsubmatpares` *para diferentes vectores.*

Ejercicio 6.33 *Crea una función que reciba una matriz de números naturales, por hipótesis, y devuelva una submatriz que contenga aquellas filas para las cuales la suma de sus elementos sea un número par. En caso de no haber ninguna fila que verifique esta propiedad, devolverá una matriz vacía.*

Ejercicio 6.34 *Crea una función que reciba una matriz de números naturales y devuelva una submatriz que contenga aquellas columnas para las cuales la suma de sus elementos sea un número primo. En caso de no haber ninguna la función devolverá una matriz vacía. Por ejemplo, si la matriz es:*

$$A = \begin{pmatrix} 2 & 6 & 7 \\ 4 & 5 & 12 \end{pmatrix}$$

el resultado será

$$B = \begin{pmatrix} 6 & 7 \\ 5 & 12 \end{pmatrix}$$

(solución en apéndice)

Ejercicio 6.35 *Crea una función que reciba una matriz A de números naturales y devuelva otra matriz B que contenga aquellas filas de A que no tengan ningún número primo.*

Ejercicio 6.36 *(Para valientes) Crea una función que reciba una matriz A y dos números naturales p y q y devuelva la submatriz que se obtiene al eliminar la fila p y la columna q de A. Se entiende que los números p y q deben ser menores o iguales que el número de filas y columnas de la matriz, respectivamente (solución en apéndice).*

Ejercicio 6.37 *Crea una función que reciba una matriz 3x3 y devuelva su determinante. Comprobar el resultado con la función* `det` *de MATLAB.*

Ejercicio 6.38 *(Para valientes) Crea una función que reciba una matriz 4x4 y calcule su determinate desarrollando por menores. Para ello utilizará las funciones de los ejercicios 6.36 y 6.37.*

Ejercicio 6.39 *(Para valientes) Crea una función que reciba una matriz A de n filas y m columnas, $n, m \geq 2$ y busque la submatriz de 2x2 de determinante máximo.*

Ejercicio 6.40 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

6.5. Resolución de un sistema lineal mediante eliminación Gaussiana

6.5.1. General

Resolver sistemas lineales se puede conseguir mediante métodos directos o mediante métodos iterativos. En los primeros, en un número finito de pasos, se llega a la solución exacta del sistema (salvo errores de redondeo). En los segundos se ha de iterar un esquema mientras algún indicador de error no sea inferior a una determinada cota. Una buena referencia para profundizar en este tema es la 7 (ver sección de referencias en la página 217).

Entre los métodos directos, el más popular para resolver sistemas lineales es la eliminación

gaussiana[1]. La idea básica del método, que ilustramos con un ejemplo, consiste en utilizar transformaciones elementales de filas para eliminar sucesivamente las variables empezando por la primera ecuación y la primera variable y continuando con el resto. De esta manera, tras $(n-1)$ eliminaciones se llega a un sistema equivalente al dado, de matriz triangular superior que se resuelve directamente por sustitución hacia atrás. Veamos cómo funciona con este sistema lineal:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 2 & 3 & 4 & 9 \\ -1 & 0 & -1 & -2 \end{array}\right)$$

Haciendo transformaciones elementales utilizando la primera fila (sumando a las filas 2 y 3 la 1 multiplicada por el factor correspondiente), hacemos cero todos los elementos de la primera columna excepto el diagonal:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & -1 & -2 & -3 \\ 0 & 2 & 2 & 4 \end{array}\right)$$

Fijándonos en el elemento diagonal de la segunda fila, hacemos cero todos los elementos de la segunda columna por debajo del diagonal.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 6 \\ 0 & -1 & -2 & -3 \\ 0 & 0 & -2 & -2 \end{array}\right)$$

[1] Carl Friedrich Gauss. Conocido como el *príncipe de las matemáticas*, fue el más grande matemático del siglo XIX y uno de los más importantes de todos los tiempos. Nació en Brunswick, en el norte de Alemania, en 1777. Fue un niño prodigio con una excepcional habilidad para la aritmética que mostró muy temprano. A los tres años le corrigió a su padre un error aritmético que tenía en las nóminas. A los diez años su maestro en la escuela pública propuso a los alumnos, para tenerlos ocupados un buen rato, que sumaran los 100 primeros números enteros. Casi de inmediato Gauss dejó el resultado correcto, 5050, sin ninguna operación adicional, en la mesa del maestro. Había sumado mentalmente la progresión aritmética $1+2+3+...+99+100$ tras observar que $100+1=99+2=98+3=...=101$. Ya que hay 50 parejas el resultado es $50\times 101=5050$.

Con 15 años desarrolló el método de los mínimos cuadrados, que aplicó después con gran éxito en Astronomía. Los astrónomos buscaban a finales del siglo XVIII un planeta nuevo entre las órbitas de Marte y Júpiter cuya existencia era sugerida por la ley de Bode (1772). El mayor de estos asteroides se descubrió en 1801 y se llamó Ceres. Se hicieron entonces unas pocas mediciones de su posición antes de que se alejara del Sol. ¿Cómo calcular su órbita con tan pocos datos? Gauss aceptó el desafío y utilizó el método de los mínimos cuadrados para determinar su órbita, y precisó a los astrónomos dónde debían enfocar sus telescopios para encontrar a Ceres, y allí estaba. Este logro le dio fama, un contrato de profesor de Astronomía y la dirección del nuevo observatorio de Göttingen.

En 1827 publicó su obra maestra, las "Disquisitiones generales circa superficies curvas" introduciendo las ideas de geometría intrínseca de superficies y demostrando el teorema "egregium". Murió en Göttingen en 1855.

Con esto tenemos ya el sistema triangular superior equivalente que resolvemos por sustitución hacia atrás comenzando por la última ecuación y acabando en la primera.

$$\begin{aligned} -2x_3 = -2 \quad &\Rightarrow \quad x_3 = 1 \\ -x_2 - 2x_3 = -3 \quad &\Rightarrow \quad x_2 = 1 \\ x_1 + 2x_2 + 3x_3 = 6 \quad &\Rightarrow \quad x_1 = 1 \end{aligned}$$

En la eliminación gaussiana tal y como la acabamos de exponer se asume que en la k-ésima eliminación el elemento que ocupa la posición (k, k) de la matriz del sistema (pivote) en ese momento es distinto de cero. A menudo no sucede así, por lo que se aconseja reordenar las ecuaciones e incluso los términos en cada una de ellas para lograr, por razones de estabilidad numérica, que el pivote sea el mayor posible en valor absoluto. Existen diferentes estrategias para la elección del pivote, según que la reordenación afecte sólo a las filas (estrategia de pivote parcial) o tanto a las filas como a las columnas (estrategia de pivote total). Implementaremos en esta sección la eliminación usando el pivote diagonal y dejaremos como ejercicio propuesto el algoritmo de eliminación con pivote parcial.

6.5.2. Resolución por sustitución hacia atrás

Empezamos con la resolución del sistema triangular por ser un algoritmo más sencillo que la triangularización. Un sistema triangular superior se resuelve de modo muy sencillo por sustitución hacia atrás, tal como se mostró en el ejemplo de la sección 6.5.1. Para ello, empezamos resolviendo la última ecuación, que tiene solo una incógnita. Con esta solución ya podemos resolver la penúltima, que tenía 2, y así sucesivamente. La codificación de este algoritmo la presentamos en la siguiente función:

```
% ud6_fbacksubstitution.m
% A matriz triangular superior
% b vector columna de igual dimension
% Devuelve x, solución del sistema lineal Ax=b
% obtenido por sustitución hacia atrás.
function x=ud6_fbacksubstitution(A,b)
[m,n]=size(A);
x=zeros(n,1); % x es vector columna aquí por
              % aparecer así formalmente en el sistema.
for k=n:-1:1
   suma=0;
   for j=k+1:n
       suma=suma+A(k,j)*x(j);
   end
   x(k)=(b(k)-suma)/A(k,k);
end
```

Se puede simplificar la escritura de esta función en la operación de modificar la fila i a partir de la k utilizando la notación matricial de MATLAB:

```
% ud6_fbacksubstitutionbis.m
% Variante de fbacksubstitution con notación matricial
function x=ud6_fbacksubstitutionbis(A,b)
[m,n]=size(A);
x=zeros(1,n); %x es aquí vector fila para poder
              % vectorizar su multiplicación con A
for k=n:-1:1
    suma=sum(A(k,k+1:n).*x(1,k+1:n));
    x(k)=(b(k)-suma)/A(k,k);
end
```

Dejamos los ejercicios complementarios de este apartado para ser realizados conjuntamente con los del apartado siguiente.

6.5.3. Obtención del sistema triangular superior

En el tutorial de MATLAB, en la sección 1.5, ya realizamos las operaciones correspondientes a la obtención de un sistema lineal triangular superior equivalente para el caso particular de una matriz de 3x3. Ahora vamos a generalizar ese algoritmo implementándolo en una función.

Lo más importante para implementar el algoritmo de eliminación gaussiana es la selección y tratamiento del pivote. Con el pivote, para cada fila posterior a la que estemos tomando como referencia se define el factor que permite, mediante la operación de filas, eliminar el elemento correspondiente. Se hace la misma operación para el término independiente.

Un aspecto interesante de este código es que dado que en realidad su objetivo es transformar los argumentos de entrada para conseguir que la matriz del sistema sea triangular superior, eso implica que los argumentos de entrada y salida de la función sean las mismas variables, algo que no había sucedido hasta ahora. La codificación en MATLAB la encontramos en la siguiente función:

```
% ud6_ftriangularsup.m
% A matriz cuadrada
% b vector columna de igual dimension
% Devuelve A y b (sistema lineal equivalente) tras
% haber realizado una serie de transformaciones
% elementales por filas para convertir A en
% triangular superior.
function [A,b]=ud6_ftriangularsup(A,b)
[m,n]=size(A);
for k=1:n-1
    pivote=A(k,k);
```

```
    for i=k+1:n
       factor=A(i,k)/pivote;
       b(i)=b(i)-b(k)*factor;
       for j=k:n
           A(i,j)=A(i,j)-A(k,j)*factor;
       end
    end
end
```

Se puede simplificar la escritura de esta función en la operación de modificar la fila i a partir de la k utilizando la notación matricial de MATLAB:

```
% ud6_ftriangularsupbis.m
% ud6_ftriangularsup modificada con notación MATLAB
function [A,b]=ud6_ftriangularsupbis(A,b)
[m,n]=size(A);
for k=1:n-1
    pivote=A(k,k);
    for i=k+1:n
       factor=A(i,k)/pivote;
       b(i)=b(i)-b(k)*factor;
       A(i,k:n)=A(i,k:n)-A(k,k:n)*factor;
    end
end
```

Para resolver un sistema lineal del ejemplo de la sección 6.5.1, utilizamos las siguientes órdenes desde la ventana de comandos de MATLAB:

```
>>[C,d]=ud6_ftriangularsup([1 2 3;2 3 4;-1 0 -1],[6 9 -2])
C =
      1     2     3
      0    -1    -2
      0     0    -2
d =
      6    -3    -2
>>x=ud6_fbacksubstitution(C,d)
x =
      1
      1
      1
```

Proponemos los siguientes ejercicios de esta sección y de la previa.

Ejercicio 6.41 *Prueba las funciones* `ud6_ftriangularsup` *y* `ud6_ftriangularsupbis` *para el ejemplo de la sección 6.5.1. Utiliza el "debugger" para comprobar que las operaciones realizadas se corresponden con el algoritmo presentado en dicha sección.*

Ejercicio 6.42 *Prueba* `ud6_fbacksubstitution` *y* `ud6_fbacksubstitutionbis` *para el ejemplo de la sección 6.5.1. Utiliza el "debugger" para comprobar que las operaciones realizadas se corresponden con el algoritmo presentado en dicha sección.*

Ejercicio 6.43 *Crea una función que reciba una matriz cuadrada regular A y un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, y devuelva la solución de ese sistema lineal, combinando las funciones anteriores.*

Ejercicio 6.44 *Crea una función que reciba una matriz cuadrada regular A y un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, y devuelva un sistema triangular inferior equivalente. Por ejemplo si*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ -1 & 0 & -1 \end{pmatrix}, \quad \mathrm{b} = \begin{pmatrix} 6 \\ 9 \\ -2 \end{pmatrix},$$

el resultado será:

$$A' = \begin{pmatrix} -2/3 & 0 & 0 \\ -2 & 3 & 0 \\ -1 & 0 & -1 \end{pmatrix}, \quad \mathrm{b}' = \begin{pmatrix} -2/3 \\ 1 \\ -2 \end{pmatrix}.$$

Ejercicio 6.45 *Crea una función que reciba una matriz cuadrada regular A y un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, y devuelva un sistema diagonal equivalente.*

Ejercicio 6.46 *(Para valientes) Crea una función que reciba una matriz cuadrada regular A y un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, y devuelva un sistema diagonal equivalente tal que la matriz sea la identidad y por tanto b sea la solución del sistema.*

Ejercicio 6.47 *(Para valientes) Utilizando el algoritmo del ejercicio 6.46, crea una función que reciba una matriz cuadrada regular A y devuelva su inversa. Hay que tener en cuenta que la inversa de una matriz cuadrada de orden n se puede obtener resolviendo n sistemas lineales en los que cada término independiente sea una columna de la matriz identidad.*

Ejercicio 6.48 *Crea una función que reciba una matriz cuadrada regular A, un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, dos naturales p, q, y devuelva un sistema equivalente tal que hayamos intercambiado las filas p y q del sistema.*

Ejercicio 6.49 *(Para los más valientes) Crea una función que reciba una matriz cuadrada regular A y un vector de igual dimensión b, los cuales definen el sistema lineal $A\mathrm{x} = \mathrm{b}$, y devuelva un sistema triangular superior equivalente en el cual se haya implementado la estrategia de selección de pivote parcial, eligiendo como fila para el pivote aquella cuyo elemento diagonal en valor absoluto sea máximo e intercambiando filas utilizando la función del ejercicio 6.48.*

Ejercicio 6.50 *Repetir el ejercicio 6.43 utilizando la función 6.49 para obtener el sistema triangular superior equivalente mediante estrategia de pivote parcial. Aplicarlo al ejemplo de la sección 6.5.1.*

Ejercicio 6.51 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

6.6. Proyectos de programación

6.6.1. General

Hemos utilizado proyectos similares en el capítulo anterior. Las normas y filosofía de los mismos se pueden encontrar en la sección 5.6.

6.6.2. Sistema lineal simétrico

Se pide codificar una función que reciba una matriz rectangular A de m filas y n columnas. Esta función calculará y devolverá el vector definido por la diagonal principal de la matriz AA^t.

Se pide también que esa función devuelva la solución del sistema lineal que tiene como matriz a AA^t y que tiene como término independiente la diagonal de dicha matriz tomada en orden inverso. Para resolver este sistema lineal se utilizarán de modo adecuado las funciones `ud6_ftriangularsup` y `ud6_fbacksubstitution`.

La función devolverá también el mínimo de los valores mayores que 2 del vector solución.

La función llamará también a `ud3_fesprimo` para saber si $m + n$ es o no primo, devolviendo el resultado al programa principal y volcando esa información sobre el archivo de resultados.

Además se codificará un programa que lea de un archivo la información necesaria para definir el problema, llame a la función que acabamos de codificar y escriba en otro archivo los resultados.

Se probará todo el problema para la matriz:

$$A = \begin{pmatrix} 2 & 3 & 1 \\ 1 & -2 & 0 \end{pmatrix}$$

generando los archivos de datos y resultados correspondientes a este ejemplo. La secuencia de operaciones a realizar es la siguiente:

1. Creación del archivo de entrada, `MATRIZ.DAT` mediante el editor que elijas, correspondiente al ejemplo propuesto.

2. Implementación de un programa que lea el archivo de datos `MATRIZ.DAT`, y genere un archivo de salida en el que esté solamente el valor m y el elemento a_{mn} de la matriz A.

3. Implementación de la función en la que se calcula la diagonal de AA^t devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

4. Implementación en esa misma función de la comprobación de si $m + n$ es o no número primo llamando a `ud3_fesprimo`, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

5. Planteamiento y resolución del sistema lineal propuesto, devolviendo a su vez este resultado al programa principal y volcando de esta información sobre el archivo de salida.

6. Cálculo del mínimo de los valores mayores que 2 del vector solución, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

7. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

6.6.3. Interpolación entre los elementos de una matriz

Para obtener un valor experimental de una función $f(x,y)$, se eligen n valores de x y m valores de y, se realizan $n \cdot m$ experimentos, uno por cada combinación de esos valores, y se guardan dichos valores en una matriz, F, de n filas y m columnas. Por ejemplo:

$$\mathbf{x} = \begin{pmatrix} 0.1 \\ 0.3 \\ 0.5 \\ 0.7 \\ 1 \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} 1 \\ 1.3 \\ 1.4 \\ 1.8 \end{pmatrix} \quad F = \begin{pmatrix} -3.6 & -1.1 & 3.2 & 4.1 \\ -4.4 & -2.2 & 2.6 & 3.3 \\ -4.7 & -2.4 & 2.3 & 2.7 \\ -5.1 & -2.7 & 2.1 & 2.6 \\ -5.2 & -3.0 & 1.9 & 2.3 \end{pmatrix}$$

El valor $F_{23} = 2.6$ correspondería a los valores $x = 0.3$ e $y = 1.3$.

Se pide construir una función que reciba n, m, los dos vectores $\mathbf{x}$ e $\mathbf{y}$ (ambos ordenados de menor a mayor por hipótesis), la matriz de valores asociada F y las coordenadas, $xeval$, $yeval$, de un punto. La función estimará el valor de f en $(xeval, yeval)$ mediante interpolación lineal, primero en el eje de abscisas y después en el de ordenadas.

Por ejemplo, para obtener el valor interpolado de f en $(0.6, 1.1)$, interpolamos entre los puntos $(0.5, 1)$ y $(0.7, 1)$, y entre los puntos $(0.5, 1.3)$ y $(0.7, 1.3)$, obteniendo los valores

$$f(0.6, 1.0) = F_{31} + (0.6 - 0.5)\frac{F_{41} - F_{31}}{0.7 - 0.5} = -4.9,$$

$$f(0.6, 1.3) = F_{32} + (0.6 - 0.5)\frac{F_{42} - F_{32}}{0.7 - 0.5} = -2.55$$

Luego, interpolamos entre los puntos $(0.6, 1)$ y $(0.6, 1.3)$, con los valores obtenidos anteriormente, para obtener

$$f(0.6, 1.1) = -4.9 + (1.1 - 1.0)\frac{-2.55 + 4.9}{1.3 - 1.0} = -4.1167$$

La función también devolverá también los puntos en los que se alcanza el máximo y mínimo de los valores de la matriz y un valor aproximado de la integral de volumen asociada a la misma.

Construir también el programa principal que lea de un archivo los datos de entrada y escriba sobre otro archivo los resultados después de llamar a la función.

El programa se codificará siguiendo la siguiente secuencia:

1. Creación del archivo de entrada `DATOS.DAT` mediante el editor que elijas, correspondiente al ejemplo propuesto.

2. Implementación de un programa `principal.m` que lea del archivo `DATOS.DAT` todos los

datos de entrada y genere un archivo de salida en el que estén todos los elementos del archivo de entrada.

3. Implementación de una función en la que se calculen los índices correspondientes al vértice inferior izquierdo del rectángulo en el que está el punto $(xeval, yeval)$. En el ejemplo propuesto, el vértice es $(0.5, 1.0)$ y los índices 3 y 1.

4. Cálculo en esta misma función del valor interpolado de f en $(xeval, yeval)$, devolviendo a su vez este resultado al programa principal.

5. Volcado de esta información sobre el archivo de salida (vértice + valor interpolado).

6. Cálculo en la función de los índices correspondiente a los elementos de F de valores mínimos y máximos, devolviendo a su vez este resultado al programa principal.

7. Volcado de esta información sobre el archivo de salida.

8. Crear una nueva función que aproxime la integral de la función mediante la suma de los volúmenes de los prismas definidos por cuatro vértices contiguos y como altura el valor de la función aproximada en el punto medio. Para obtener la altura, usar la función anterior.

9. Volcado de esta información sobre el archivo de salida.

10. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

6.6.4. Interpolación en la base de los monomios

Se pide construir una función que resuelva un problema de interpolación en la base de los monomios. Se trata de interpolar una nube de puntos $\{x_i, f_i\}$, $i = 1, n+1$ y la derivada en el segundo punto fp_2. La solución es un polinomio de grado $n+1$

$$\mathrm{a}(x) = \sum_{i=1}^{n+2} a_i x^{i-1}$$

La función recibirá como input el vector de abscisas de los nodos x, el valor de la función en esos nodos f y el valor de la derivada en el segundo nodo fp_2. Devolverá el vector de coeficientes a calculado al resolver el sistema lineal:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdot & x_1^{n+1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & x_{n+1} & x_{n+1}^2 & \cdot & x_{n+1}^{n+1} \\ 0 & 1 & 2x_2 & \cdot & (n+1)x_2^n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \cdot \\ a_{n+1} \\ a_{n+2} \end{pmatrix} = \begin{pmatrix} f_1 \\ \cdot \\ f_{n+1} \\ fp_2 \end{pmatrix}.$$

Para resolver este sistema lineal se utilizarán de modo adecuado las funciones `ud6_ftriangularsup` y `ud6_fbacksubstitution`.

La función llamará también a la función `ud4_fevalua` estudiada en el curso para evaluar el valor del polinomio resultado en un vector **xeval** que se obtendrá equiespaciando $m+1$ puntos entre a y b, ambos inclusive, con a, b, m entradas de la función.

Además se codificará un programa que lea de un archivo la información necesaria para definir el problema, llame a la función que acabamos de codificar y escriba en otro archivo los resultados.

Al escribir el archivo de salida se incluirán las instrucciones MATLAB que generen la gráfica del polinomio solución, definida entre a y b con $n+1$ puntos.

Se probará todo el problema para los vectores:

$$x = (-1, 0, 1)$$

$$f = (-1, 0, 1)$$

$$fp_2 = 0$$

entregando los archivos de datos y resultados correspondientes a este ejemplo. El rango para la gráficia será $[a, b] = [-2, 2]$, con 1001 puntos.

El programa se codificará siguiendo la siguiente secuencia:

1. Creación del archivo de entrada, `DATOS.DAT` mediante el editor que elijas, correspondiente al ejemplo propuesto(0.5p).

2. Implementación de un programa `principal.m` que lea del archivo `DATOS.DAT` todos los datos que definen el problema y genere un archivo de salida en el que estén todos los elementos del archivo de entrada.

3. Implementación de la función en la que se construye la matriz del sistema lineal y se calcule la suma de sus elementos, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

4. Resolver el sistema lineal propuesto, devolviendo a su vez este resultado al programa principal y volcando esta información sobre el archivo de salida.

5. Escribir, desde `principal.m`, con formato MATLAB, un fichero de extensión `.m` que dibuje el gráfico del polinomio entre $[a, b]$ con $m+1$ puntos equiespaciados, así como los puntos originales y un segmento que se apoye en el segundo punto y que tenga como pendiente fp_2.

6. Rehacer el proyecto creando una GUI que ejecute todo el programa, incorporando una llamada al archivo de datos en el que se encuentren los datos de entrada y presentando en la ventana correspondiente el gráfico del apartado 5.

7. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

6.6.5. Resolución de un sistema tridiagonal: método de Thomas

Cuando un sistema lineal

$$A\mathrm{x} = \mathrm{y}$$

es tridiagonal, el algoritmo de eliminación gaussiana se puede simplificar teniendo en cuenta la estructura particular de A. Este caso es importante porque se obtienen a menudo sistemas de este tipo en la resolución numérica de algunas ecuaciones diferenciales en derivadas parciales, o trabajando con splines cúbicos.

Adoptemos para los elementos no nulos de A la notación siguiente:

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & \cdots & \cdots & \cdots \\ a_2 & b_2 & c_2 & 0 & \cdots & \cdots & \cdots \\ 0 & a_3 & b_3 & c_3 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ \cdots & \cdots & \cdots & \cdots & 0 & a_n & b_n \end{bmatrix}$$

El algoritmo de Thomas es el siguiente:

1. Normalizamos la primera línea del sistema

$$\begin{bmatrix} 1 & \frac{c_1}{b_1} & 0 & \cdots & \cdots & \cdots & \cdots \\ a_2 & b_2 & c_2 & 0 & \cdots & \cdots & \cdots \\ 0 & a_3 & b_3 & c_3 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ \cdots & \cdots & \cdots & \cdots & 0 & a_n & b_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_i \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \frac{y_1}{b_1} \\ y_2 \\ y_3 \\ \cdots \\ y_i \\ \cdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

2. Reducimos la segunda fila después de restarle la primera multiplicada por a_2

$$\begin{bmatrix} 1 & \frac{c_1}{b_1} & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & \left(b_2 - a_2\frac{c_1}{b_1}\right) & c_2 & 0 & \cdots & \cdots & \cdots \\ 0 & a_3 & b_3 & c_3 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ \cdots & \cdots & \cdots & \cdots & 0 & a_n & b_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_i \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \frac{y_1}{b_1} \\ y_2 - a_2\frac{y_1}{b_1} \\ y_3 \\ \cdots \\ y_i \\ \cdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

3. Normalizamos la segunda fila.

$$\begin{bmatrix} 1 & \frac{c_1}{b_1} & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & \left(\frac{c_2}{b_2 - a_2\frac{c_1}{b_1}}\right) & 0 & \cdots & \cdots & \cdots \\ 0 & a_3 & b_3 & c_3 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ \cdots & \cdots & \cdots & \cdots & 0 & a_n & b_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_i \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \frac{y_1}{b_1} \\ \frac{y_2 - a_2\frac{y_1}{b_1}}{b_2 - a_2\frac{c_1}{b_1}} \\ y_3 \\ \cdots \\ y_i \\ \cdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

4. Reducimos la tercera fila después de restarle la segunda multiplicada por a_3, etc.

Continuando así para las n filas del sistema, se obtiene un sistema bidiagonal con diagonal unidad

$$\begin{bmatrix} 1 & \gamma_1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & \gamma_2 & 0 & \cdots & \cdots & \cdots \\ 0 & 0 & 1 & \gamma_3 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & 0 & 1 & \gamma_i & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 0 & 1 & \gamma_{n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_i \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \cdots \\ \beta_i \\ \cdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

cuya resolución es inmediata.

Se pide construir una función que implemente este método de Thomas para la reducción de un sistema tridiagonal. Se pide también construir un programa principal `principal.m` que recoja la información de entrada de un archivo de datos, llame a la función que resuelve el sistema lineal y vuelque la información sobre un archivo de salida. El archivo de entrada contendrá toda la información que el estudiante considere necesaria para definir el problema.

El ejemplo al que hay que aplicar todo el programa será el correspondiente al siguiente sistema lineal.

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 1 \end{bmatrix}$$

Capítulo 7

Algoritmos

7.1. General

El carácter de MATLAB como lenguaje de programación interpretable y no compilable, pero con códigos fácilmente depurables, lo convierte en candidato estupendo para el estudio y desarrollo de algoritmos de complejidad media. Aunque durante todo el libro hemos potenciado el desarrollo de algoritmos sencillos a medida que introducíamos nuevas ideas, comandos y estructuras de datos, en esta unidad trabajaremos específicamente algunos algoritmos clásicos. A esta lección es difícil llegar en un curso estándar de fundamentos de programación o similar, pero la incluimos por el interés que puede tener para aquellos grupos que hayan funcionado especialmente bien, y puedan disfrutar con este acercamiento a problemas más reales.

Nos centraremos en algoritmos que tienen un interés transversal para nuestros titulados, los cuales manejan habitualmente aplicaciones de las que esos algoritmos son parte esencial y utilizan estas funcionalidades de modo rutinario. Así, estudiaremos algoritmos de ordenación similares a los que forman el núcleo de gestores de bases de datos como Ms-ACCESS o de hojas de cálculo como Ms-EXCEL. También estudiaremos algunos algoritmos geométricos como los que son usados por sistemas CAD. La referencia bibliográfica más relevante para este tema es la correspondiente al texto de R. Sedgewick (ver ref. 8 en la sección de Referencias, pág. 218). El objetivo final de este capítulo es consolidar las estrategias de diseño de algoritmos para resolver problemas algo más complicados y cerrar de este modo este curso de introducción a la programación, utilizando MATLAB como lenguaje de referencia.

7.2. Algoritmos de búsqueda

7.2.1. General

Es muy habitual utilizar las funcionalidades de búsqueda que ofrecen los procesadores de textos, editores, hojas de cálculo o visualizadores de ficheros como "Acrobat". También es muy común utilizar buscadores web como "GOOGLE", o las utilidades de búsqueda en catálogos online.

Detrás de esas funcionalidades hay algoritmos de búsqueda muy poderosos, el conocimiento de cuyos fundamentos básicos puede ser de gran utilidad para la programación de determinadas aplicaciones.

Elegiremos un problema de referencia sencillo; se parte de un vector de números enteros ordenado de modo ascendente. Se quiere conocer si un determinado valor está incluido en ese vector o no. Plantearemos dos técnicas para resolver este problema y compararemos la eficiencia de las mismas.

7.2.2. Búsqueda secuencial

En este algoritmo, se recorre un vector desde el principio hasta el final comprobando si un determinado valor está en el vector. En el momento en que encontremos un valor que es igual al buscado terminamos. También terminamos si sobrepasamos ese valor buscado, porque dado que el vector está ordenado, eso significa que el elemento no pertenece al vector. Cuando en el capítulo 4 vimos los algoritmos de búsqueda de extremos de vectores, en realidad ya estábamos viendo técnicas de búsqueda secuencial. Presentamos ahora el código utilizado.

```
% v - vector de enteros ordenado ascendentemente
% e - valor que se trata de encontrar
% flag - vale 0 si e no está en v
%        vale 1 si e está en v
function flag=ud7_seqsearch(v,e)
n=length(v);
flag=0;
i=1;
while i<=n && v(i)<=e
    if e==v(i)
        flag=1;
        break;
    end
    i=i+1;
end
```

Es interesante resaltar que el operador && ejerce una función denominada "short-circuit", que en este contexto es importante. El operador evalúa la segunda condición sólo si la primera es cierta (ver sección 2.11). Si no fuese así, el índice i podría tomar valores que nos llevasen fuera del vector $\mathbf{v}$.

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 7.1 *Crea una carpeta llamada* `ud7` *en donde consideres oportuno. Esta será tu carpeta de trabajo para todos los ejercicios y ejemplos del capítulo 7.*

Ejercicio 7.2 *Copia la función* `ud7_seqsearch.m` *a tu carpeta de trabajo y pruébala.*

Ejercicio 7.3 *Modifica la función* `ud7_seqsearch.m` *cambiando del modo que corresponda el bucle* `while` *por* `for`.

Ejercicio 7.4 *Crea una función que reciba un vector de enteros* $\mathbf{v}$ *(no ordenado) y un elemento* e *y devuelva si el elemento está en el vector. La búsqueda secuencial es el algoritmo más básico cuando el vector donde estamos buscando no tiene ninguna estructura.*

Ejercicio 7.5 *Codifica una función que reciba dos vectores* $\mathbf{u}$ *y* $\mathbf{v}$ *ordenados de modo creciente, que carecen de elementos comunes y de dimensiones* m *y* n *que son en general distintas. La función devolverá otro* $\mathbf{w}$ *construido a partir de todas las componentes de* $\mathbf{u}$ *y* $\mathbf{v}$, *tales que el vector* $\mathbf{w}$ *también esté ordenado. No se podrá llamar a ninguna función. Por ejemplo, si* $\mathbf{u} = (2, 4, 5, 9, 10)$ *y* $\mathbf{v} = (1, 3, 6, 8)$, *el resultado será* $\mathbf{w} = (1, 2, 3, 4, 5, 6, 8, 9, 10)$. *Como comprobación adicional, si intercambiamos* $\mathbf{u}$ *y* $\mathbf{v}$ *en la llamada, el resultado tiene que ser el mismo.*

Ejercicio 7.6 *Crea una función que reciba un vector ordenado de modo ascendente por hipótesis* $\mathbf{v}$ *y un elemento* a *y devuelva la posición que ocuparía* a *en el vector, es decir, devuelva* 1 *si* a *es menor que todos los elementos de* $\mathbf{v}$, $n+1$, *donde* n *es el número de elementos de* $\mathbf{v}$, *si* a *es mayor que todos los elementos, un número* i *tal que* $v(i-1) \leq a \leq v(i)$ *en cualquier otro caso. Por ejemplo, si* $\mathbf{v} = (1, 3, 6, 8)$ *y* $a = 7.5$ *la función devolvería 4.*

Ejercicio 7.7 *Crea una función que reciba un vector ordenado de modo ascendente* $\mathbf{v}$, *un elemento* a, *dos posiciones* m *y* n, *con* $m < n$ *por hipótesis, y devuelva qué posición ocuparía* a *con respecto a los elementos* $v_m, \ldots, v_n$ *(*m *si es menor que todos,* $m+1$ *si es mayor que el primero y menor que los demás,* $m+2$ *si es mayor que los dos primeros y menor que los demás y así sucesivamente).*

Ejercicio 7.8 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.2.3. Búsqueda binaria

En el algoritmo de búsqueda binaria se aprovecha que el vector está ordenado para ubicar mediante subdivisiones consecutivas el elemento buscado. Para entenderlo, veamos un ejemplo: si tenemos la siguiente secuencia formada por 11 elementos:

$$2 \quad 5 \quad 8 \quad 16 \quad 17 \quad 18 \quad 19 \quad 20 \quad 23 \quad 28 \quad 29$$

y queremos ver si el 21 pertenece a la lista, las operaciones que se deducen del algoritmo son las siguientes:

1. Se define un rango de búsqueda (i, j). En la primera pasada será todo el vector, y por tanto tomamos $i = 1$, $j = 11$.

2. Se busca elemento central, o sea $(i + j)/2 = 6$, $v(6) = 18$

3. Se compara $v(6)$ con 21, y se llega a la conclusión de que 21 está en la segunda mitad de la lista, pues es mayor que $v(6)$. Se cambia i por $6 + 1 = 7$, y buscamos ahora en el rango $(7, 11)$

4. El elemento central es ahora $(i + j)/2 = (7 + 11)/2 = 9$, $v(9) = 23$

5. Se compara $v(9)$ con 21, y se llega a la conclusión de que 21 está en la primera mitad del tramo $(7, 11)$, pues es menor que $v(9)$. Se cambia j por $9 - 1 = 8$, y buscamos ahora en el tramo $(7, 8)$

6. El elemento central es ahora $(i + j)/2 = (7 + 8)/2 = 7.5$, cuya parte entera es 7, $v(7) = 19$

7. Se compara $v(7)$ con 21, y se llega a la conclusión de que 21 está en la segunda mitad del tramo $(7, 8)$, pues es mayor que $v(7)$. Se cambia i por $7 + 1 = 8$, y buscamos ahora en el tramo $(8, 8)$

8. El elemento central es ahora $(i + j)/2 = (8 + 8)/2 = 8$, $v(8) = 20$

9. Se compara $v(8)$ con 21, y se llega a la conclusión de que 21 está en la segunda mitad del tramo $(8, 8)$, pues es mayor que $v(8)$. Se cambia i por $8 + 1 = 9$, y al definir el nuevo tramo tenemos que es $(9, 8)$, o sea, $i > j$, en cuyo caso terminamos, porque el elemento no pertenece a la lista.

Si estuviésemos buscando el elemento 20, cambiarían las últimas tareas:

6. El elemento central es ahora $(i + j)/2 = (7 + 8)/2 = 7.5$, cuya parte entera es 7, $v(7) = 19$

7. Se compara $v(7)$ con 20, y se llega a la conclusión de que 20 está en la segunda mitad del tramo $(7, 8)$, pues es mayor que $v(7)$. Se cambia i por $7 + 1 = 8$, y buscamos ahora en el tramo $(8, 8)$

8. El elemento central es ahora $(i + j)/2 = (8 + 8)/2 = 8$, $v(8) = 20$

9. Se compara $v(8)$ con 20, y se llega a la conclusión de que son iguales. Por tanto, cambiamos a 1 la variable indicadora de pertenencia, rompemos el bucle de búsqueda y terminamos.

Mostramos ahora el código:

```
% v - vector de enteros ordenado
% e - valor que se trata de encontrar
% flag - vale 0 si e no está en v
%        vale 1 si e está en v
function flag=ud7_binsearch(v,e)
n=length(v);
flag=0;
i=1;
j=n;
while j>=i
    k=floor((i+j)/2);
    if e<v(k)
        j=k-1;
    elseif e>v(k)
        i=k+1;
    else
        flag=1;
        break;
    end
end
```

La idea de ir dividiendo el problema es muy poderosa. Es aplicable a la resolución de ecuaciones no lineales en una variable mediante el método de la bisección, como se plantea en el ejercicio 7.13. Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 7.9 *Copia la función* ud7_binsearch.m *a tu carpeta de trabajo y pruébala.*

Ejercicio 7.10 *Crea una función que reciba un vector ordenado de modo ascendente* v, *un número* x, *compruebe si* x *está en el vector, y devuelva 0 si no es así o el índice de la posición que ocupa en el vector si pertenece.*

Ejercicio 7.11 *Repite 7.6 utilizando la técnica de la búsqueda binaria (solución en apéndice).*

Ejercicio 7.12 *Repite 7.7 utilizando la técnica de la búsqueda binaria.*

Ejercicio 7.13 *Codifica una función que reciba un vector* $\mathbf{p}$ *representando un polinomio* p, *dos valores* a, b $a < b$, *tales que* $p(a) \cdot p(b) < 0$, *y devuelva alguna de las raíces del polinomio* p *en ese intervalo*[1]. *Para ello utilizará el método de la bisección. Calcularemos* $y = p((a+b)/2)$ *y transformaremos el intervalo inicial en uno mitad dependiendo del signo* y, *de modo análogo a como se hace en el esquema de búsqueda binaria. En el momento que* $\mathrm{abs}(p(y))$ *sea menor que una determinada precisión se parará. La precisión se tomará como la milésima parte del máximo en valor absoluto de* $p(b)$ *y* $p(b)$. *Se probará la función resultado con el polinomio* $-5 + x + 3x^3$, *con* $a = 1$ *y* $b = 1.5$.

[1]Al haber un cambio de signo, hay al menos una raíz

Ejercicio 7.14 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.2.4. Comparación

Es interesante comparar la velocidad de los dos algoritmos de búsqueda estudiados. Para ello, creamos un vector de naturales ordenado y otro con valores aleatorios que estén en el rango del primero. Se trata de buscar todos ellos en ese primer vector con los dos métodos.

Con un único elemento sería difícil apreciar la diferencia, a no ser que el vector fuese muy grande, y además en ese caso, puede haber factores azarosos, que se mitigan en gran medida al coger una serie grande de búsquedas diferentes. Introducimos con este *script* tres instrucciones nuevas: `tic`, `toc` y `rand`. Invitamos al estudiante a pedir ayuda sobre ellas a MATLAB pues las tres son útiles e interesantes.

```
% ud7_sbusqueda.m
clear all
n=input('Numero de elementos del vector: ');
% Generamos un vector ordenado de modo aleatorio
% sus elementos están entre 0 y 10*n;
v(1)=floor(rand(1)*10);
for i=2:n
    v(i)=v(i-1)+floor(rand(1)*10);
end
% Definimos m elementos entre 0 y v(n)
m=input('Numero de elementos a buscar: ');
e=floor(v(n)*rand(1,m));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% buscamos en v los elementos de e
tic;
for i=1:m
    ud7_seqsearch(v,e(i));
end
t1=toc;
%
tic;
for i=1:m
    ud7_binsearch(v,e(i));
end
t2=toc;
%
fprintf('Búsqueda secuencial: %g s\n',t1);
fprintf('Búsqueda binaria: %g s\n',t2);
```

Los ejercicios correspondientes a este ejemplo son los siguientes:

Ejercicio 7.15 *Copia el script* `ud7_sbusqueda.m` *a tu carpeta de trabajo y pruébalo con los siguientes valores, valorando los tiempos obtenidos:*

```
>> ud7_sbusqueda
Numero de elementos a ordenar: 2000
Numero de elementos a buscar: 7000
Búsqueda secuencial: 2.57591 s
Búsqueda binaria: 0.124875 s
```

Ejercicio 7.16 *Crea una función que reciba una matriz cuadrada A de números naturales de 10 filas con*

```
A=floor(50*rand(10,10))
```

y devuelva una submatriz formada por las filas de la 4 a la 9 y las columnas de la 5 a la 8.

Ejercicio 7.17 *Construye un script que te permita comparar la velocidad de los bucles* `while` *y* `for`.

UNIVERSIDAD TÉCNICA DEL NORTE BIBLIOTECA

Ejercicio 7.18 *Crea una función que reciba dos números naturales z y $n \geq 2$ y devuelva, si existen, dos números naturales x e y tales que*

$$x^n + y^n = z^n.$$

Caso de que no existan, devolverá $x = 0$ e $y = 0$. Si el problema está bien codificado, sólo encontrará alguna solución para $n = 2$. Que esta ecuación no tenga soluciones para $n > 2$ fue postulado como teorema por Pierre Fermat en el siglo XVII y demostrado en 1995 por Andrew Wiles[2]. Por ejemplo, si $z = 5$ y $n = 2$ entonces $x = 3$, $y = 4$.

Ejercicio 7.19 *Crea una función que reciba un número par n mayor que dos y devuelva dos números primos p y q tales que $p + q = n$, si existen, y si no, que devuelva $p = 0$ y $q = 0$. No está probado que siempre se pueda descomponer n de ese modo. A la suposición de que n se puede descomponer*

---

[2] Andrew Wiles nació en Inglaterra en 1953 y es bien conocido por ser el matemático que consiguió probar el último Teorema de Fermat, después de más de tres siglos de intentos por numerosos matemáticos ilustres.

Wiles se topó con este problema siendo niño curioseando un libro en una biblioteca. En aquel entonces ya le fascinó un problema que había dado tantos quebraderos de cabeza a eminentes matemáticos y que sin embargo tenía un enunciado tan simple. El último Teorema de Fermat (que más bien debería denominarse la última Conjetura de Fermat) afirma que la ecuación $x^n + y^n = z^n$ no tiene solución con x, y, z números naturales cuando $n \geq 3$.

Wiles estudió en Oxford e hizo su tesis doctoral en Cambridge, visitando después numerosas universidades y desarrollando su investigación en Teoría de Números, Geometría y Aritmética. A mediados de los 80 se descubrió que el último Teorema de Fermat se podía deducir de la demostración de una conjetura de Shimura y Taniyama, conjetura dentro del área de interés matemático de Wiles. En ese momento, Wiles abandonó sus proyectos y se encerró durante siete años en solitario a demostrar dicha conjetura. Su prueba contenía un error que subsanó un año más tarde. A pesar de su inmenso logro Wiles no fue galardonado con la Medalla Fields (el equivalente al premio Nobel de Matemáticas) ya que la prueba definitiva del Teorema de Fermat la presentó cuando ya tenía 41 años y tradicionalmente la Medalla Fields es otorgada a matemáticos de hasta 40 años (fuente: http://www-history.mcs.st-and.ac.uk/).

compruebo como el uso de la función permite intercambiar el valor de las variables x e y. Es importante apreciar que x, y en la línea de comandos no tienen nada que ver con las variables locales de la función x, y. Si ejecutamos del siguiente modo la función, utilizando variables adicionales no habrá cambiado su valor, como se puede comprobar:

```
>> x=3.2;
>> y=-5.7;
>> [p,q]=ud7_fswap(x,y)
p =
   -5.7000
q =
    3.2000
x =
    3.2000
y =
   -5.7000
```

Los ejercicios correspondientes a esta sección son muy interesantes. Una secuencia de los mismos nos lleva a un algoritmo de ordenación de un vector.

Ejercicio 7.24 *Copia la función* `ud7_fswap.m` *a tu carpeta de trabajo y pruébala.*

Ejercicio 7.25 *Usando la función* `ud7_fswap`*, construye una función que reciba dos números a y b y los devuelva en orden creciente.*

Ejercicio 7.26 *(Para valientes) Usando la función creada en el ejercicio 7.25, construye una función que reciba tres números y los devuelva en orden creciente, sin usar* `if`*. Idem con cuatro. Crea los programas correspondientes para gestionar la entrada y salida de datos de las dos funciones.*

Ejercicio 7.27 *Codifica una función que reciba un vector, intercambie cada dos elementos del vector para que estén en orden creciente usando la función construida en el ejercicio 7.25 y devuelva el vector. Es decir, cogerá los dos primeros elementos y los intercambiará para que estén en orden creciente. Después cogerá el segundo y el tercero y los intercambiará, y así sucesivamente. Por ejemplo, si introducimos el vector $(3, 1, 4, 2, 5)$, en sucesivos pasos iremos obteniendo $(1, 3, 4, 2, 5)$, $(1, 3, 4, 2, 5)$ y $(1, 3, 2, 4, 5)$.*

Ejercicio 7.28 *Crea un programa que pida un vector, llame a la función del ejercicio 7.27 e imprima el vector obtenido. ¿Cuál es el ultimo elemento del vector modificado?. Introduce el vector obtenido de nuevo en el programilla. ¿Qué ocurre ahora con los dos últimos elementos del vector modificado?.*

Ejercicio 7.29 *Crea una función que reciba un vector, repita el proceso del ejercicio 7.27 tantas veces como elementos tenga el vector y devuelva el vector.*

Ejercicio 7.30 *Crea un programa que pida un vector, llame a la función del ejercicio 7.29 y devuelva el vector modificado. ¿Cuál es este vector?*

Ejercicio 7.31 *Se pide crear función que decida cuál gana a la grande entre dos jugadas de mus. Para simular las jugadas, recibirá dos vectores* mano *y* postre *con los números de las cartas, todos naturales entre 1 a 7 y 10 a 12 por hipótesis. Hay que tener en cuenta que un 3 es equivalente a un 12 (cerdos), por lo que lo primero será transformar los tres en doces. Lo mismo sucede con los unos y los doses (pitos), por lo que habrá que transformarlos también.*

La mano ganará si la carta más alta es mayor. En caso de que sean iguales, se pasará a la siguiente carta y así sucesivamente. Si todas son iguales, será la mano el que gane.

La función devolverá una variable `flag` *que ha de valer 1 si gana la mano y 0 si gana el postre.*

Se creará también un script que cargue la matriz de un fichero, llame a la función y escriba el resultado sobre un archivo de salida con un formato adecuado. Se creará manualmente el fichero de entrada (solución en apéndice).

Por ejemplo, si $\text{mano} = (3, 6, 7, 1)$ *y* $\text{postre} = (7, 12, 7, 2)$, *ganará el postre pues ordenadas de mayor a menor, las dos primeras son iguales pero la tercera del postre (7) gana a la tercera de la mano (6).*

Ejercicio 7.32 *Crea una función que decida entre dos jugadas de chica de mus. Las reglas son similares a las de grande, salvo que primero se comparan la carta más pequeña de cada uno y gana el jugador cuya carta sea más pequeña. Si son iguales, se mira la siguiente más pequeña y así sucesivamente. En caso de ser iguales, gana la mano. Recuerda que en el mus un 3 es un rey a todos los efectos y un 2 es un as.*

Ejercicio 7.33 *(Para valientes) Crea una función que decida entre dos jugadas de pares de mus. Gana el que tenga más parejas (la mejor jugada son dos parejas, la siguiente un trío y la siguiente un par). En caso de que los dos jugadores tengan la misma jugada (por ejemplo, dobles parejas), gana aquel que tenga una pareja de cartas más altas. En caso de que ninguno de los jugadores tenga pareja, la función devolverá un 0.*

Ejercicio 7.34 *Crea una función que decida entre dos jugadas de juego de mus. Para ello se suman los valores de las cartas (las sotas, caballos y reyes valen 10 puntos). Gana el que sume 31, después 32, después 40, 39, 38, ..., 33. En caso de empate, gana la mano. Si uno no tiene juego (suman 30 o menos), gana el que lo tenga. Si ninguno tiene juego, la función devolverá 0.*

Ejercicio 7.35 *Crea un programa que pida dos jugadas de mus e imprima por pantalla quién gana la grande, quién la chica, quién los pares y quién el juego.*

Ejercicio 7.36 *(Para los más valientes) Vamos a crear una función que obtenga la mejor jugada de poker en una mano. Para simular la mano, recibirá un vector con cinco cartas. Deberá devolver un número según la mejor jugada que encuentre, así:*

0 Si no encuentra jugada.

1 Si encuentra una pareja.

2 Si encuentra un trío.

3 Si encuentra un full (un trío y una pareja).

4 Si encuentra una escalera.

5 Si encuentra poker (4 cartas iguales).

Ejercicio 7.37 *Crea un programa para gestionar la entrada y salida de la función 7.36.*

Ejercicio 7.38 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.3.3. Algoritmo de selección

El algoritmo de ordenación más sencillo e intuitivo es el de selección. Como queremos ordenar de menor a mayor, el primer elemento de nuestro vector resultado será el mínimo del vector original. Se trata de calcular la posición de ese valor y de intercambiarlo después con el primero. Una vez hecho esto, se busca el mínimo de entre todos los restantes y se intercambia con el segundo y así sucesivamente. Podemos ver como funcioina este algoritmo con el ejemplo de la figura 7.1(a).

Para codificar el algoritmo, escribimos primero una función, `ud7_fimin`, que nos devuelve la posición del mínimo entre dos posiciones de un vector. La función es similar a la ya estudiada en la sección 4.5, y es de hecho una búsqueda secuencial similar a la estudiada también en la sección 7.2.2. La función `ud7_fimin` permite convertir el proceso de ordenación por selección en un bucle en el que se llama a `ud7_fimin` en cada paso y se intercambian posiciones.

```
% ud7_fimin.m
% Recibe un vector v y dos naturales, m y n, con m<=n.
% Devuelve la posicion del minimo entre las
% componentes m y n del vector (m y n inclusive)
function imin=ud7_fimin(v,m,n)
imin=m;
for i=m+1:n
  if v(i)<v(imin)
    imin=i;
  end
end
```

```
%ud7_fseleccion.m
%Recibe un vector y lo devuelve ordenado de menor a mayor
function v=ud7_fseleccion(v)
n=length(v);
for i=1:n-1
  imin=ud7_fimin(v,i,n);
  [v(imin),v(i)]=ud7_fswap(v(imin),v(i));
end
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.39 *Copia* `ud7_fseleccion.m` *a tu carpeta activa y pruébalo.*

Ejercicio 7.40 *Codifica una función que reciba un vector* v, *con un número de componentes impar y todas diferentes entre si por hipótesis, y obtenga su mediana (valor que tiene tantos elementos de la matriz menores como mayores que él. Para obtener la mediana se ordenará* v *mediante un algoritmo de selección.*

Ejercicio 7.41 *Codifica una función que reciba una matriz A, un natural c y devuelva la matriz ordenada por filas teniendo como criterio que la columna c esté ordenada de modo ascendente. Por ejemplo, si*

$$A = \begin{pmatrix} 2 & 6 & 1 & -1 \\ 3 & 4 & 8 & 6 \\ -7 & 6 & -15 & -4 \\ 1 & 1 & 12 & 2 \end{pmatrix}, \quad c = 4$$

la función devolverá

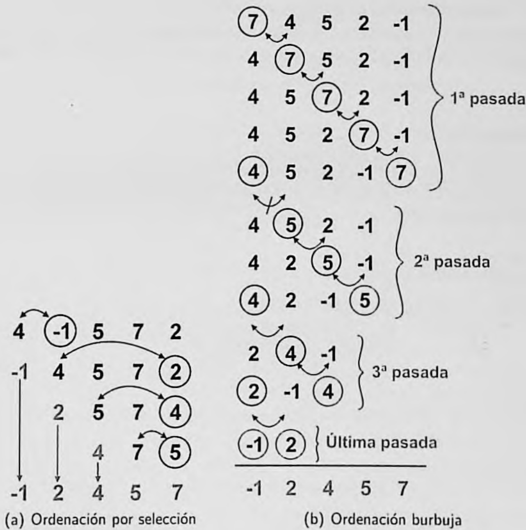
$$\begin{pmatrix} -7 & 6 & -15 & -4 \\ 2 & 6 & 1 & -1 \\ 1 & 1 & 12 & 2 \\ 3 & 4 & 8 & 6 \end{pmatrix}$$

No se podrá llamar en principio a ninguna función propia de MATLAB para ordenación como `sort` *o* `sortrows` *aunque se repetirá después el ejercicio haciendo uso adecuado de estas funcionalidades (solución en apéndice).*

Ejercicio 7.42 *Codifica una función que reciba una matriz A, un natural c y una variable* `flag` *y devuelva la matriz ordenada por filas teniendo como criterio que la columna c esté ordenada de modo descendente si* `flag=0` *y ascendente si* `flag=1`.

Ejercicio 7.43 *Codifica una función que reciba una matriz A, dos naturales $c1, c2$ y devuelva la matriz ordenada por filas teniendo como criterio que la columna $\max(c1, c2)$ esté ordenada de modo ascendente.*

```
        end
        if ordenado==1 % Si estaba ordenado, paramos
            break;
        end
        i=i+1;
end
```



(a) Ordenación por selección

(b) Ordenación burbuja

Figura 7.1: Algoritmos de ordenación

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.46 *Crea una función que reciba un vector y una variable llamada* `direccion`*. Si* `direccion` *vale 1, recorrerá sus elementos del primero al último, intercambiando cada dos consecutivos si el segundo es menor que el primero y devolviendo* 1 *o* 0 *en* `modificado` *según se haya modificado el vector. Si* `direccion` *vale* -1*, recorrerá sus elementos del último al primero, intercambiando cada dos consecutivos si el segundo es menor que el primero y devolviendo* 1 *o* 0 *en* `modificado` *según se haya modificado el vector. Si* `direccion` *toma cualquier otro valor, devolverá el vector sin modificar y* `modificado`*=-1.*

Ejercicio 7.47 *El algoritmo de la burbuja funciona muy mal cuando el vector está ordenado exactamente al revés de lo que se pretende conseguir. Para mejorarlo se utiliza el algoritmo de las Sacudidas: Se procede igual que con la burbuja, pero intercambiando la dirección de las pasadas en*

cada iteración. Crea una función que reciba un vector y lo devuelva ordenado según el algoritmo de las Sacudidas, usando para ello la función creada en el ejercicio 7.46.

Ejercicio 7.48 *(Para valientes) Crea una función que reciba un vector y lo devuelva ordenado según el algoritmo de las Sacudidas, sin usar ninguna función auxiliar distinta de* `ud7_fswap`.

Ejercicio 7.49 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.3.5. Comparación

Es interesante comparar las diferencias en lo que se refiere a velocidad de ambos métodos para ordenar un mismo vector, o vectores de determinadas características. Para ello, hemos escrito el programa *ud7_sordenacion*:

```
%ud7_sordenacion.m
clear all
n=input('Numero de elementos a ordenar: ');
v=rand(1,n); % Generamos un vector de modo aleatorio
             % Sus n componentes están entre 0 y 1.
%
tic; % Establecemos el tiempo inicial
w=ud7_fburbuja(v); % Ordenamos con el algoritmo de la burbuja
t1=toc; % Guardamos el tiempo que hemos tardado
tic;
w=ud7_fseleccion(v); % Ordenamos con el algoritmo de selección
t2=toc;
tic;
w=ud7_fburbuja(w); % Probamos ordenar un vector ya ordenado
t3=toc;
tic;
w=ud7_fseleccion(w); % Idem con el algoritmo de selección
t4=toc;
%
% Probemos a ordenar un vector ordenado de mayor a menor
w=-w; % Cambiamos el orden de w.
tic;
v=ud7_fburbuja(w);
t5=toc;
tic;
v=ud7_fseleccion(w); % Idem con el algoritmo de selección
t6=toc;
%
fprintf('Burbuja: %g s\n',t1);
fprintf('Seleccion: %g s\n',t2);
```

cada iteración. Crea una función que reciba un vector y lo devuelva ordenado según el algoritmo de las Sacudidas, usando para ello la función creada en el ejercicio 7.46.

Ejercicio 7.48 *(Para valientes) Crea una función que reciba un vector y lo devuelva ordenado según el algoritmo de las Sacudidas, sin usar ninguna función auxiliar distinta de* `ud7_fswap`.

Ejercicio 7.49 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.3.5. Comparación

Es interesante comparar las diferencias en lo que se refiere a velocidad de ambos métodos para ordenar un mismo vector, o vectores de determinadas características. Para ello, hemos escrito el programa *ud7_sordenacion*:

```
%ud7_sordenacion.m
clear all
n=input('Numero de elementos a ordenar: ');
v=rand(1,n); % Generamos un vector de modo aleatorio
             % Sus n componentes están entre 0 y 1.
%
tic; % Establecemos el tiempo inicial
w=ud7_fburbuja(v); % Ordenamos con el algoritmo de la burbuja
t1=toc; % Guardamos el tiempo que hemos tardado
tic;
w=ud7_fseleccion(v); % Ordenamos con el algoritmo de selección
t2=toc;
tic;
w=ud7_fburbuja(w); % Probamos ordenar un vector ya ordenado
t3=toc;
tic;
w=ud7_fseleccion(w); % Idem con el algoritmo de selección
t4=toc;
%
% Probemos a ordenar un vector ordenado de mayor a menor
w=-w; % Cambiamos el orden de w.
tic;
v=ud7_fburbuja(w);
t5=toc;
tic;
v=ud7_fseleccion(w); % Idem con el algoritmo de selección
t6=toc;
%
fprintf('Burbuja: %g s\n',t1);
fprintf('Seleccion: %g s\n',t2);
```

```
fprintf('Burbuja con vector ordenado: %g s\n',t3);
fprintf('Seleccion con vector ordenado: %g s\n',t4);
fprintf('Burbuja con vector ordenado al reves: %g s\n',t5);
fprintf('Idem seleccion: %g s\n',t6);
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.50 *Copia* `ud7_sordenacion.m` *a tu carpeta de trabajo y pruébalo. Valora los resultados.*

Ejercicio 7.51 *Incluye el algoritmo de ordenación por sacudidas en* `ud7_sordenacion` *y pruébalo de nuevo. ¿Qué mejoras encuentras respecto al algoritmo de la burbuja?*

Ejercicio 7.52 *Crea una función que reciba dos vectores,* $\mathbf{u}$ *y* $\mathbf{v}$*, el segundo de ellos de numeros naturales entre* 1 *y la dimensión del vector* $\mathbf{u}$*. La función devolverá el mínimo de los valores de los elementos de* $\mathbf{u}$ *indexados por los elementos del vector* $\mathbf{v}$*. Por ejemplo, si* $\mathbf{u} = (2.3, 4.2, 1.2, 0.1, 43.1)$ *y* $\mathbf{v} = (3, 1, 5)$*, devolverá* 1.2*, que es el mínimo de* u_3*,* u_1 *y* u_5*. Programa el algoritmo del siguiente modo:*

1. *Toma como máximo el elemento de* $\mathbf{u}$ *indicado por el primer elemento de* $\mathbf{v}$*, es decir* u_{v_1}*.*
2. *Recorre el resto de elementos indexados en* $\mathbf{v}$*. Para cada elemento* i *de* $\mathbf{v}$ *comprueba si el elemento* u_{v_i} *es mayor que el máximo hasta el momento; si es así, tómalo como nuevo máximo.*

Ejercicio 7.53 *Crea una función que reciba un vector* $\mathbf{v}$ *y una posición* i *y un elemento* a *y devuelva el vector* $\mathbf{v}$*, colocando* a *en la posición* i *y desplazando los elementos de* $\mathbf{v}$ *una posición a la derecha. Es decir, el vector* $\mathbf{w}$ *solución será:* $w(j) = v(j)$*, si* $j < i$*,* $w(i) = a$*,* $w(j) = v(j+1)$ *si* $j \geq i$

Ejercicio 7.54 *Crea una función que reciba un vector y lo ordene de modo ascendente utilizando el siguiente algoritmo:*

1. *Toma el primer elemento del vector y consideraremos ese elemento como un vector ordenado.*
2. *Tomamos el segundo elemento y con la función del apartado 7.7 obtenemos en qué posición iría.*
3. *Lo colocamos en dicha posición con la función del apartado 7.53.*
4. *Tomamos el tercer elemento, obtenemos la posición respecto a los dos primeros y lo colocamos ahí.*
5. *Consideramos los tres primeros, obtenemos la posición del cuarto y lo colocamos, y así sucesivamente. Este es el algoritmo de inserción.*

Ejercicio 7.55 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.3.6. Ordenación con vector auxiliar

Tal como hemos tratado hasta ahora el tema de la ordenación, el vector ordenado es diferente del original; ha sufrido una serie de cambios para verificar un determinado criterio de ordenación. A menudo, es más interesante que la información original no se modifique y generar información adicional que nos permita recorrer la original siguiendo un determinado criterio. Por ejemplo, si tenemos el siguiente vector $\mathbf{v} = (2.3, 6.2, 1.2, 0.1, 3.1)$, podemos generar uno auxiliar que nos permita recorrerlo de modo ascendente $\mathbf{u} = (4, 3, 1, 5, 2)$. Así que $u(1) = 4$ significa que el elemento más pequeño del vector $\mathbf{v}$ es el 4; que $u(2) = 3$ significa que el segundo elemento más pequeño del vector $\mathbf{v}$ es el 3, y así sucesivamente. A este tipo de técnicas está dedicada esta sección y en el ejemplo presentado se codifica una solución del ejemplo expuesto:

```
% ud7_fordenapos.m
% Recibe un vector v y devuelve un vector u con las
% posiciones de los elementos tal que si lo recorremos
% en ese orden iremos de menor a mayor
function u=ud7_fordenapos(v)
n=length(v);
u=1:n; %Creamos un vector con los indices de v
for i=1:n-1 % Posicion en la que colocaremos el mínimo
  imin=ud7_fimin(v,i,n); % Calculamos la posicion del
                         % mínimo desde la componente i
  [v(imin),v(i)]=ud7_fswap(v(i),v(imin)); % Colocamos el
        % mínimo en su posicion intercambiando v(imin) y v(i)
        % Hacemos lo mismo con los indices
  [u(imin),u(i)]=ud7_fswap(u(i),u(imin));
end
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.56 *Copia la función* `ud7_fordenapos.m` *a tu carpeta de trabajo y pruébala.*

Ejercicio 7.57 *Crea una función que reciba tres números enteros, $ganados$, $empatados$ y $perdidos$ y devuelva la puntuación calculada como $3 * ganados + empatados$.*

Ejercicio 7.58 *Crea una función que reciba tres vectores de n componentes,* **ganados**, **empatados** *y* **perdidos**, *de modo que la componente i-ésima de cada uno de esos vectores es el número de partidos ganados, empatados o perdidos por el equipo i. La función devolverá un vector que en la posición i-ésima tendrá la puntuación del equipo i-ésimo. Utiliza para calcularla la función del apartado anterior.*

Ejercicio 7.59 *Crea un programa que pida el número de equipos y para cada equipo pida el número de partidos ganados, empatados y perdidos, llame a la función creada en el apartado anterior e imprima por pantalla la puntuación de cada equipo. La entrada y salida de la función debe tener este aspecto:*

```
Introduce el número de equipos: 10
-Equipo 1-
Partidos ganados: 3
Partidos empatados: 4
Partidos perdidos: 2
-Equipo 2- ...
Las puntuaciones son:
Equipo 1: 13 puntos
Equipo 2: 7 puntos ...
```

Ejercicio 7.60 *Crea una función que reciba tres vectores de n componentes,* **ganados**, **empatados** *y* **perdidos**, *de modo que la componente i-ésima de cada uno de esos vectores es el número de partidos ganados, empatados o perdidos por el equipo i y devuelva la posición en la que quedaría cada equipo y los puntos. Para calcular la puntuación de cada equipo llamará a la función del apartado 7.58. Para calcular la posición, llamará a* `ud7_fordenapos`.

Ejercicio 7.61 *Crea un programa que pida el número de equipos y para cada equipo pida el número de partidos ganados, empatados y perdidos, llame a la función creada en el apartado anterior e imprima por pantalla la posición de cada equipo. La entrada y salida de la función debe tener este aspecto:*

```
Introduce el número de equipos: 10
-Equipo 1-
Partidos ganados: 3
Partidos empatados: 4
Partidos perdidos: 2
-Equipo 2- ...
Clasificación:
1° Equipo 5: 21 puntos
2° Equipo 7: 20 puntos ...
```

Ejercicio 7.62 *(Para valientes) Crea una función que haga lo mismo que* `ud7_fordenapos` *pero utilizando el algoritmo de la burbuja en lugar del de selección.*

Ejercicio 7.63 *Crea una función que reciba un vector* $\mathbf{v}$, *llame a* `ud7_fordenapos` *para calcular la posición de cada uno de sus elementos, que guardará en un vector* $\mathbf{u}$. *A continuación generará un vector* $\mathbf{w}$ *tal que* $w_i = v_{u_i}$. *La función devolverá el vector* $\mathbf{w}$. *¿Cómo es el vector* $\mathbf{w}$*?*

Ejercicio 7.64 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.4. Algoritmos geométricos

7.4.1. General

La presencia de la Geometría en un capítulo de algoritmos para programación no necesita mucha justificación. Siempre que manejemos gráficos, imágenes o planos en el ordenador, estaremos usando algoritmos que tienen que ver específicamente con los problemas geométricos tratados. El progresivo incremento de las posibilidades de los ordenadores en el manejo de gráficos ha supuesto una revolución en las técnicas de diseño arquitectónico, de diseño en Ingeniería, y de diseño gráfico en general. Esto hace que el estudio de estos algoritmos sea importante y ha dado lugar a una de las ramas más fructíferas de las matemáticas, la geometría computacional. Sin embargo, quizá la razón que los hace más interesantes es el cambio de perspectiva que exigen en el programador. Mientras que en los algoritmos estudiados hasta ahora, de algún modo, podíamos trasladar directamente al código nuestros esquemas mentales, en los problemas geométricos la exigencia de abstracción es mucho mayor. El problema de comprobar si un punto está dentro de un polígono tiene una solución trivial cuando uno ve un dibujo sobre un papel; conseguir diseñar y codificar un algoritmo para resolver este problema exige repensar los conceptos de punto y línea e identificar las operaciones elementales sobre estos conceptos que son necesarias para resolver el problema.

Esta sección está organizada en torno a dos ejemplos y a ejercicios que usan esos ejemplos como punto de partida para resolver problemas más complicados, como el de pertenencia de un punto a un polígono, o formación de un polígono conexo a partir de una nube de puntos.

7.4.2. Orientación de 3 puntos

El problema es caracterizar la posición relativa de tres puntos en el plano, en lo que respecta a si al avanzar desde el primero hasta el segundo y después al tercero, esto se realiza en el sentido de las agujas del reloj o en el sentido contrario. Si nos fijamos en la figura 7.2, podemos comprobar que el sentido de giro está asociado al crecimiento o decrecimiento relativo de las pendiente del segmento que une los dos primeros puntos y el punto segundo y tercero. Si la pendiente decrece, estamos girando en el sentido de las agujas del reloj, y si crece, en sentido contrario. Si nombramos las diferencias en las coordenadas como $dx1$, $dy1$, $dx2$ y $dy2$, la relación entre las pendientes pasa por comparar los valores:

$$\frac{dy1}{dx1} \quad \text{vs} \quad \frac{dy2}{dx2}$$

Para evitar divisores nulos, podemos pensar esta comparación del siguiente modo:

$$dy1 \cdot dx2 \quad \text{vs} \quad dy2 \cdot dx1$$

En realidad, es como si hubiésemos multiplicado las dos fracciones por $dx1 \cdot dx2$. En teoría habría que tener cuidado con el efecto del signo de este factor en la desigualdad entre las fracciones, pero observando cada posibilidad se llega a la conclusión de que el efecto no es tal. También, aunque son de cierto interés las diferentes posibilidades dependiendo de la posición del punto intermedio cuando los tres puntos están alineados, vamos a asignar valor 0 a los casos en los 3 puntos están alineados.

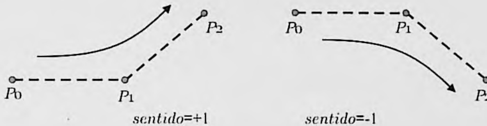


Figura 7.2: Orientación de 3 puntos

```
% p0         primer punto de la serie,
%            p0(1)=x del punto, p0(2)=y
% p1         segundo punto de la serie.
% p2         tercer punto de la serie.
% sentido = 0 los tres puntos estan alineados.
%         =-1 estan en el sentido de las agujas del reloj.
%         =+1 los puntos estan en el sentido contrario.
%
function sentido=ud7_reloj3(p0,p1,p2)
%
dx1=p1(1)-p0(1);
dx2=p2(1)-p1(1);
dy1=p1(2)-p0(2);
dy2=p2(2)-p1(2);
%
if dy1*dx2 > dx1*dy2
    sentido =-1;
elseif dy1*dx2 < dx1*dy2
    sentido=+1;
else
    sentido=0;
end
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.65 *Copia la función* `ud7_reloj3` *a tu carpeta de trabajo y pruébala con las siguientes entradas* `ud7_reloj3([0 0],[1 0],[0 1])` *y* `ud7_reloj3([0 0],[1 0],[0 -1])`, *decidiendo si los resultado son coherentes con lo que se pretende de la misma*

Ejercicio 7.66 *Codifica una función que reciba 4 puntos y devuelva -1 si están orientados en el sentido de las agujas del reloj, 1 si están en sentido contrario y 0 en cualquier otro caso.*

Ejercicio 7.67 *Codifica una función que reciba una matriz de 2 filas y n columnas. Cada columna representa las coordenadas x e y de un punto. La función devolverá -1 si los puntos están orientados en el sentido de las agujas del reloj, 1 si están en sentido contrario y 0 en cualquier otro caso.*

Ejercicio 7.68 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.4.3. Intersección de 2 segmentos

Discernir si dos segmentos en el plano se cortan o no es un problema clave en geometría computacional, dado que muchos otros problemas importantes se apoyan en la resolución de éste. Se podría plantear la solución de modo analítico, buscando la intersección de las rectas en las que se apoyan los segmentos, y comprobando después si esa intersección pertenece a cada uno de dichos segmentos. Sin embargo, hay una solución más interesante, la cual pasa por analizar el sentido de giro de los extremos de un segmento y uno de los puntos del otro. Si hay cambio de sentido de giro en todos los casos, es que los segmentos se cortan (fig. 7.3). En dicha figura, en la posibilidad de la izquierda, hay cambio de sentido en todos los casos (se cortan), mientras que en la de la derecha, para uno de los segmentos lo hay pero para el otro no (no se cortan). Otra vez, no tenemos en cuenta los casos de alineamiento que por cierto son el rompecabezas de los programadores de este tipo de aplicaciones.

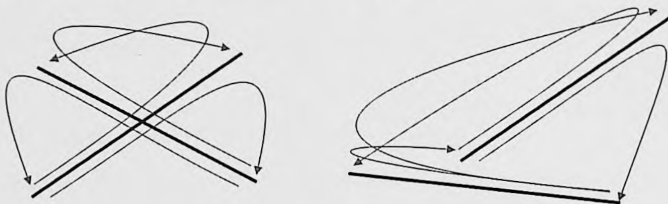


Figura 7.3: Intersección de 2 segmentos

```
% Esta funcion comprueba si dos segmentos se cortan o no.
% p1,p2 - extremos del primer segmento.
% q1,q2 - extremos del segundo segmento.
%
function corte=ud7_intersec(p1,p2,q1,q2)
%
corte=0;
test1=ud7_reloj3(p1,p2,q1)*ud7_reloj3(p1,p2,q2);
test2=ud7_reloj3(q1,q2,p1)*ud7_reloj3(q1,q2,p2);
```

```
if test1<0 && test2<0
    corte=1;
end
```

Los ejercicios correspondientes a esta sección son los siguientes:

Ejercicio 7.69 *Copia la función* `ud7_intersec.m` *a tu carpeta de trabajo y pruébala con las siguientes entradas, decidiendo si los resultado son coherentes con lo que se pretende de la misma:*

```
>> ud7_intersec([0 0],[2 0],[1 1],[1 -1])
>> ud7_intersec([0 0],[2 0],[1 -1],[1 -3])
```

Ejercicio 7.70 *Se define un polígono de n vértices como una matriz ordenada de 2 filas y n columnas. La columna i contiene las coordenadas x e y del vértice i. Se supone que los vértices están ordenados, de tal modo que el segmento que une los vértices i, $i+1$ es el lado del polígono que los une. Se pide codificar una función que reciba un polígono de este modo y devuelva el área que encierra el polígono*

Ejercicio 7.71 *Uno de los problemas clásicos es el de la inclusión de un punto en un polígono. Si tenemos un punto y construimos un segmento que vaya desde ese punto a cualquier infinito, este segmento cortará al polígono un número de veces o quizá ninguna. Se puede ver gráficamente(figura 7.4) que si el número de cortes es par es que el punto no pertenece y si el número de cortes es impar, es que sí pertenece. No se tendrá en cuenta que los cortes puedan producirse en los vértices del polígono, ni que el punto pertenezca al perímetro del polígono. Consideraremos el infinito como un punto que tenga la coordenada y del punto en cuestión, y su coordenada x sea 2 veces el máximo de las coordenadas x de los vértices del polígono. Se pide codificar una función que reciba un polígono y un punto y devuelva 0 o 1 dependiendo de que el punto no esté o sí esté incluido en el polígono.*

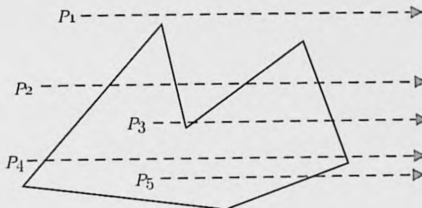


Figura 7.4: Inclusión de un punto en un polígono

Ejercicio 7.72 *Sea una matriz P de 2 filas n columnas que contiene puntos en el plano. Construir una función que reciba la matriz P y devuelva una matriz Q que contenga el polígono convexo asociado a P (ver figura 7.5).*

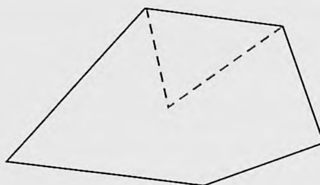


Figura 7.5: Polígono convexo asociado al de la figura 7.4

Ejercicio 7.73 *Imagina y especifica un problema susceptible de ser resuelto mediante el ordenador y tal que el algoritmo que lo resuelva utilice las ideas utilizadas hasta ahora en el libro, y en particular, las de esta sección.*

7.5. Proyectos de programación

7.5.1. General

Ya hemos utilizado estos proyectos en capítulos anteriores. Las normas de los mismos se pueden encontrar en la sección 5.6.2.

7.5.2. Cálculo de nóminas

Se trata de codificar un programa que calcule la nómina de los trabajadores de una empresa.

1. Crea un programa llamado `principal.m` que pida el número de trabajadores, n y un vector **v** con el número de horas que trabaja cada uno de ellos e imprima en pantalla el vector. Por ejemplo, si introducimos $n = 5$ y los números $[155\ 104\ 133\ 201\ 139]$, imprimirá:

```
v(1)=155
v(2)=104
v(3)=133
v(4)=201
v(5)=139
```

2. Codificar una función `Fnomina.m` que reciba un vector, $\mathbf{horas}$, y para cada elemento en el vector calcule la nómina del trabajador correspondiente aplicando las siguientes reglas:

   a) Cada hora trabajada se paga a 30 euros.

   b) Las horas extras (se consideran horas extras las que sobrepasen las 150) se pagan doble.

   La función devolverá los € totales que debe pagar la empresa a los trabajadores, es decir, la suma de todas las nóminas.

3. Modificar el programa `principal.m` para que, además de lo anterior, llame a la función `Fnomina.m`, le pase el vector con las horas trabajadas, reciba el montante final y lo imprima en pantalla. Recuerda que es conveniente comenzar los programas con `clear all`.

4. La empresa también tiene una lotería que funciona del siguiente modo: Para cada trabajador, genera un número al azar entre 1 y el número de trabajadores, y si ese número es primo, recibe una bonificación de 100 euros. Modificar la función `Fnomina` para que también incluya la lotería. Utilizar de modo adecuado la función `rand` y `ud3_fesprimo` para comprobar si es primo.

5. Modificar la función `Fnomina.m` para que también devuelva la nómina de cada trabajador, el número de trabajadores con horas extras y el número de trabajadores que han recibido lotería.

6. Modificar `principal.m` para que imprima por pantalla los datos devueltos por la función `Fnomina.m` del apartado anterior.

7. Modificar `Fnomina.m` para que, además de todo lo anterior, devuelva la posición en el vector del trabajador que más horas ha trabajado y del que menos.

8. Modificar `principal.m` para que también imprima en pantalla la posición del trabajador que más horas ha trabajado y del que menos.

9. Modificar `Fnomina.m` para que, además de todo lo anterior, reciba un vector **tramos** de dimensión m ordenado ascendentemente, el cual contendrá tramos horarios. `Fnomina.m` devolverá un vector $\mathbf{w}$ de dimensión $m+1$ con el número de trabajadores que están en cada intervalo determinado por el vector. Por ejemplo, si

```
v(1)=155
v(2)=104
v(3)=133
v(4)=201
v(5)=139
```

y el vector $\mathbf{tramos} = [100, 130, 150, 200]$, devolverá un vector $\mathbf{w} = [0, 1, 2, 1, 1]$ (es decir, 0 trabajadores con menos de 100 horas, 1 entre 100 y 130, 2 entre 130 y 150, etc.).

10. Modificar el programa principal para que también imprima $\mathbf{w}$.

11. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

7.5.3. Luz sobre una poligonal

Se trata de obtener la cantidad de luz que recibe una superficie, aunque para simplificar el problema se considera su versión en 2D, o sea, la cantidad de luz sobre una poligonal; la cual vendrá dada mediante una serie de segmentos obtenidos de unir consecutivamente sus vértices, $\mathbf{p}_i = (x_i, y_i)$, $i = 1 : n$ (ver figura 7.6). La dirección de la luz vendrá dada por un vector bidimensional, $\mathbf{v} = (v_x, v_y)$, en el que

supondremos $v_x > 0$ y $v_y < 0$, lo cual no es necesario comprobar. La intensidad de la fuente de luz en kW/m será I. Como ejemplo para probar el programa, consideramos la dirección $(2.1, -1.2)$, los puntos $(0, 1.2)$, $(0.6, 0.2)$, $(1.1, 2.2)$, $(1.6, 2.3)$ y $(1.7, -0.2)$, e $I = 0.5\, kW/m$.

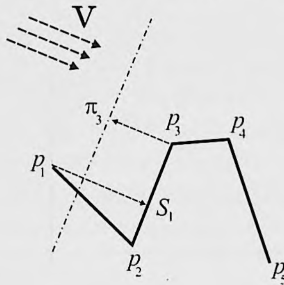


Figura 7.6: Ejemplo correspondiente al proyecto 7.5.3

El programa se codificará siguiendo la siguiente secuencia:

1. Creación del archivo de entrada `DATOS.DAT`, mediante el editor que elijas, correspondiente al ejemplo propuesto.

2. Implementación de un programa `principal.m` que lea del archivo `DATOS.DAT` todos los datos de entrada y genere un archivo de salida en el que estén todos los elementos del archivo de entrada.

3. Implementación de una función en la que se calcule la media M de las alturas de los puntos $\mathbf{p}_i$, $i = 1 : n$, usando para ello la función `ud4_fmedia`. En el ejemplo propuesto, la media es 1.14.

4. Cálculo en esta misma función de la cantidad total de luz que recibe la poligonal. Para cada punto $\mathbf{p}_i$, se calcularán su proyección π_i en la recta que pasa por $(0, 0)$ y perpendicular al rayo de luz, mediante la fórmula

$$\lambda_i = \frac{x_i + \frac{v_x}{-v_y} y_i}{-v_y + \frac{(v_x)^2}{-v_y}},$$

donde λ_i es el valor del parámetro tal que $(-\lambda_i\, v_y, \lambda_i\, v_x)$ es la proyección π_i (ver en la figura 7.6 la correspondiente interpretación geométrica). En el ejemplo, $\lambda_1 = 0.431$, $\lambda_2 = 0.195$, $\lambda_3 = 1.015$, $\lambda_4 = 1.154$, $\lambda_5 = 0.277$.

La cantidad total de luz (energía) que recibe la superficie (por segundo), se obtendrá mediante la fórmula

$$c = I(\lambda_{max} - \lambda_0)\|v\|,$$

donde λ_{max} es el valor máximo de los λ_i. En el ejemplo,

$$c = 0.5 \cdot (1.154 - 0.432) \cdot \sqrt{1.2^2 + 2.1^2} = 0.8731\, kW.$$

5. Se define la magnitud H como el cociente de c/M. H será el resultado de la función y se pasará como salida de la misma.

6. Llamar a la función desde el principal y volcar esta información (H) sobre el archivo de salida.

7. La cantidad de luz que recibe un segmento de poligonal, determinado por los puntos (x_i, y_i) e (x_{i+1}, y_{i+1}), se calculará mediante la fórmula

$$c_i = \begin{cases} I(\lambda_{i+1} - \lambda_i)\|v\|, & \lambda_{i+1} > \lambda_i, \\ 0, & \lambda_{i+1} <= \lambda_i. \end{cases}$$

Modificar la función para que también calcule y devuelva un vector que contenga la cantidad de luz que recibe cada segmento de poligonal.

8. Modificar la llamada a la función desde el principal para que reciba de la función el vector calculado en el punto anterior y volcar esa información en el archivo de salida.

9. Para cada punto $\mathbf{p_i}$, su sombra $\mathbf{S}_i$ en la poligonal se obtiene por el siguiente procedimiento: Recorremos los puntos a la derecha de (x_i, y_i) mientras el λ_j del punto considerado sea menor que λ_i. Cuando lleguemos al primer punto $\mathbf{p}_j$ con $\lambda_j \geq \lambda_i$, la sombra $\mathbf{S}_i$ será la intersección del segmento $(x_{j-1}, y_{j-1}) - (x_j, y_j)$ con la recta que pasa por (x_i, y_i) con dirección (v_x, v_y). En caso de que no haya ningún punto con $\lambda_j \geq \lambda_i$, se considerará como sombra de $\mathbf{p}_i$ el propio punto $\mathbf{p}_i$. En la figura 7.6, la sombra del punto $\mathbf{p_1}$ es $\mathbf{S}_1$. Nótese que, en este ejemplo, la sombra de cada uno de los demás puntos será el propio punto. Modificar la función para que obtenga la sombra de los puntos $\mathbf{p}_i$, $i = 1 : n$.

10. Modificar la llamada a la función para que reciba las coordenadas de las sombras y volcar esta información sobre en el archivo de salida.

11. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

7.5.4. Crecimiento de un montículo

Se trata de construir un programa que simule el crecimiento de un montículo de arena.

Para modelizar el montículo se calculará un vector de n componentes, $\mathbf{vH}$, que almacenará en cada posición los niveles de altura (números enteros) alcanzados por las partículas de arena depositadas allí. Así, si por ejemplo $vH(6) = 3$ significará que en la posición 6 hay 3 partículas en altura (ver figura7.7).

Las partículas se modelizarán como un vector de m componentes, $\mathbf{vP}$, conteniendo en cada una un número entre 1 y n que será la posición en la que caerá la partícula. Así, si $vP(1) = 2$, eso significa que la partícula 1 va al montículo 2 (ver figura7.7). Como ejemplo para probar el programa, tomamos $n = 7$, $m = 8$, y el siguiente vector de partículas, cuya disposición representamos en la figura 7.7.

$$\mathbf{vP} = [2\ 6\ 2\ 3\ 5\ 6\ 6\ 7]$$

Presentamos la puntuación de modo acumulado.

1. Construir una función `Facumula.m` que reciba un vector **vP**, el número de montones n y genere un vector **vH** de n componentes, todas ellas iguales a 0. Dará valor 0 a una variable entera T que será lo que devolverá la función.

2. Modificar `Facumula.m` para que simule la deposición de las partículas de **vP** en **vH**, obteniendo al final el vector **vH**, pero sin devolverlo como salida. Para ello, recorremos el vector de partículas **vP** y para cada elemento de dicho vector, $vP(i)$, sumamos una unidad en la posición de **vH** indicada por $vP(i)$. En el ejemplo propuesto, tomamos el primer elemento de **vP**, 2, y sumamos una unidad en la segunda posición de **vH**. Cogemos el segundo elemento de **vP**, 6, y sumamos una unidad a la posición 6 de **vH**, para tener de momento que $vH(6) = 1$. Se procederá de modo análogo con el resto de los elementos de $vP(i)$. Al concluir el proceso, tendremos, para el ejemplo propuesto, que:

$$\mathbf{vH} = [0\ 2\ 1\ 0\ 1\ 3\ 1].$$

   lo cual significa que en la primera posición hay 0 partículas, en la 2 hay 2 partículas, en la 3, 1, etc... como aparece en la figura 7.7.

3. Modificar `Facumula.m` para que, después de todo lo anterior, calcule y devuelva como salida el número de montones de arena distintos T(un montón de arena será una serie de posiciones consecutivas con altura no nula). Para ello, utilizaremos un contador al que asignaremos inicialmente el valor 1 si el primer elemento de **vH** es distinto de cero y 0 si el primer elemento es cero. Recorreremos el vector **vH** y cada vez que un elemento sea cero y el siguiente distinto de cero, sumaremos uno al contador. En el ejemplo, $\mathbf{vH} = [0\ 2\ 1\ 0\ 1\ 3\ 1]$, y tenemos dos montones.

4. Crea un programa llamado `principal.m` que pida un número m, un vector con m elementos indicando la posición a la que caen las partículas con m elementos y un número n. El programa llamará a `Facumula.m` e imprimirá en pantalla el número de montones. Si introducimos $m = 8$, el vector $[2\ 6\ 2\ 3\ 5\ 6\ 6\ 7]$ y $n = 7$, imprimirá un 2 pues hay 2 montones. Recuerda que es conveniente comenzar los programas con `clear all`.

5. Modificar `Facumula.m` para que devuelva también el vector **vH**.

6. Modificar `principal.m` para que además de lo anterior, imprima en pantalla este vector.

7. Modificar `Facumula.m` para que, después de depositar todas las partículas (es decir, calcular los valores de **vH**), calcule las alturas máximas y media alcanzadas. En el ejemplo considerado, la altura máxima es 3 y la media 1.1429.

8. Modificar `principal.m` para que además de lo anterior, imprima en pantalla la altura máxima y media y el número de montones.

9. Modificar `Facumula.m` para que en lugar de recibir el vector **vP**, reciba únicamente n y m y genere el vector **vP** aleatoriamente. Es decir, recorrerá con un bucle las posiciones de **vP** y a cada posición le asignará un valor aleatorio.

10. Modificar `principal.m` para que no pida el vector $\mathbf{vP}$.

11. Modificar `Facumula.m` para que también devuelva un vector con las posiciones inicial (primer elemento distinto de cero) y final (último elemento distinto de cero) de cada montón. En el ejemplo considerado, dicho vector será $[2\ 3\ 5\ 7]$.

12. Modificar `principal.m` para que, además, imprima el nuevo vector.

13. Cuando la altura en una posición es muy grande respecto a las vecinas, se caen partículas de arena de esa posición. Crear una función `Favalancha.m` que reciba el vector $\mathbf{vH}$ y para cada posición i de ese vector, si la altura de una de las posiciones vecinas ($i-1$ e $i+1$ salvo si estamos en un borde) es dos unidades menor, reste una unidad de la posición i y se la sume a esa posición vecina (si las dos son menores, se la suma a la $i-1$). `Favalancha.m` devolverá el vector $\mathbf{vH}$ modificado y una variable, `estado`, que valdrá 1 si ha habido alguna modificación y cero en caso contrario.

14. Modificar `Facumula.m` para que, después de todo lo anterior, llame a `Favalancha.m` con un bucle hasta que que devuelva `estado`=0.

15. Rehacer el proyecto utilizando toda la potencia de la notación matricial de MATLAB y de sus funciones propias.

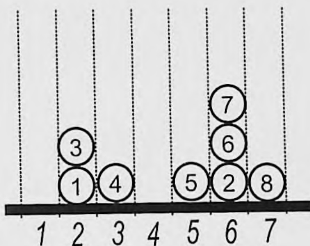
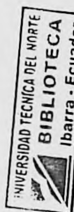


Figura 7.7: Configuración de partículas de arena correspondiente al ejemplo del proyecto 7.5.4

Epílogo

Para terminar el libro, recuperamos algunas ideas mencionadas en el prólogo sobre lo que no es este libro.

1. No hemos pretendido escribir un compendio de MATLAB. Hay mucha y buena literatura sobre las posibilidades enormes de MATLAB en los ámbitos de la ciencias y la ingeniería, a la cual poco hubiésemos podido aportar. Así, aunque el estudiante se sienta más cómodo al final del curso en el entorno MATLAB, no hemos pretendido enseñar al estudiante a calcular y resolver problemas complejos utilizando las posibilidades de MATLAB. Nos hemos centrado en una serie de comandos e ideas canónicas como base para construir lo que es un curso de introducción a la programación. Creemos que sin embargo el estudiante estará ahora bien preparado para empezar a explorar las posibilidades que tiene MATLAB en multitud de campos, a través de los *toolboxes* disponibles (estadística, *splines*, tratamiento de señales, procesado de imágenes, etc.) y a través de una utilización inteligente de la potente notación matricial del programa.


UNIVERSIDAD TECNICA DEL NORTE
BIBLIOTECA
Ibarra - Ecuador

2. No hemos pretendido escribir un curso de programación aprovechando las posibilidades específicas de MATLAB en ese sentido, dado que ello le quitaría generalidad. Así, en los primeros capítulos se han realizado todas las operaciones elemento a elemento para potenciar la familiarización con los bucles. Solo en los capítulos finales se ha hecho uso de esas potencialidades.

3. Finalmente, y dado que este libro es un curso de introducción a la programación para no informáticos, nos hemos puesto límites en los conceptos a tratar. No hemos tratado por ejemplo la utilización de cadenas de caracteres porque creemos que es un aspecto importante en programación pero que puede confundir en un curso introductorio. Tampoco hemos tratado la recursividad, ni la orientación a objetos, ni otros muchos aspectos. Creemos sin embargo que los conceptos tratados lo son con cierta profundidad, con el objetivo de que las competencias adquiridas por los estudiantes les pudieren servir como base para retos más ambiciosos en el futuro, ya orientados para trabajar en ámbitos específicos utilizando tanto los conceptos generales de programación como los relativos a las potencialidades de MATLAB.

Para terminar, si has llegado hasta aquí, es procedente que hagamos de abogados del diablo presentando algunos problemas que tiene MATLAB como lenguaje de programación. Lo

primero a comentar es que MATLAB no es realmente un lenguaje compilable y eso impide distribuir aplicaciones desarrolladas con MATLAB de un modo sencillo. Ello implica también que cuando el problema a resolver es muy exigente desde el punto de vista computacional (cálculos en mecánica de fluidos computacional por ejemplo), MATLAB no sea una opción viable. A esto se une el hecho de que MATLAB es un lenguaje propietario y si se quiere comercializar código escrito en MATLAB, el cliente debe haber adquirido previamente licencias del programa, cuyo coste es significativo. Hay que decir que en algunos casos la versión libre de MATLAB, OCTAVE, puede solucionar este inconveniente pero este no siempre es el caso.

Finalmente, MATLAB compite con lenguajes tan elegantes como C o C++ y tan potentes como Python, sin tener ni su rigor ni su consistencia ni su complejidad conceptual de cara al usuario. En nuestra opinión, que sea más simple hace que MATLAB sea una mejor opción para un curso de introducción a la programación para no informáticos. A ti te toca ahora encontrar sus límites.

Referencias

1. Título: Aprender Matlab-Octave en 25 horas

   Autor: Arias Marco, T, et al.

   Editorial: Tébar

   Año: 2011

   Este libro es un curso básico de MATLAB. Está pensado para adquirir un manejo elemental sobre la realización de operaciones matemáticas, uso de vectores, gráficas y definir pequeñas funciones.

2. Título: A Practical Introduction to Programming and Problem Solving.

   Autor: Attaway, S.

   Editorial: Elsevier.

   Año: 2009

   Este libro ofrece una guía práctica de introducción a la programación enfocado a MATLAB. Es bastante extenso, quizá demasiado, para curso de introducción a la programación estando además muy orientado a MATLAB.

3. Título: Introducción Informal a MATLAB y OCTAVE

   Autor: Borrell i Nogueras, G..

   Año: 2007.

   Este texto es una excelente introducción a MATLAB como herramienta para resolver problemas de Métodos Numéricos. Disponible online en http://forja.rediris.es/.

4. Título: MATLAB programming for engineers

   Autor: Chapman, S. J.

   Editorial: Brooks/Cole-Thomson Learning.

   Año: 2002.

   Es un libro similar al de Attaway, creemos que demasiado poco específico para un curso de introducción a la programación. Este libro es ahora mismo difícil de conseguir. En 2013 sacará una edición actualizada con título "MATLAB Programming with Applications for Engineers", Con Nelson Engineering.

5. Título: MATLAB. Una introducción con ejemplos prácticos.

   Autores: Gilat, A.

   Editorial: Reverté.

   Año: 2006.

   Buen texto introductorio a MATLAB y a la programación con MATLAB. No está organizado como el curso que se presenta aquí y creemos que es excesivo para un curso de introducción, pero es un buen libro.

6. Título: Aprenda MATLAB 7 como si estuviera en primero.

   Autores: García de Jalón, J., Rodríguez, J.I., Brazález, A.

   Editorial: Escuela Superior de Ingenieros Industriales, Universidad Politécnica de Madrid.

   Año: 2005

   Guía para aprender MATLAB paso a paso. Está pensada para estudiantes de primero y se puede seguir con facilidad. Aunque la parte de programación es demasiado escueta, es interesante tenerla para utilizarla como ayuda cuando utilicemos el MATLAB para otras asignaturas.

7. Título: Problemas de Cálculo Numérico para Ingenieros con Aplicaciones MATLAB

   Autor: Sánchez Sánchez, J.M., Souto Iglesias, A.

   Editorial: McGraw-Hill, Colección Schaum.

   Año: 2011

   Este texto es una referencia interesante sobre la aplicación de MATLAB a la resolución de problemas de Análisis Numérico.

8. Título: Algoritmos en C++

   Autor: Sedgewick, R.

   Editorial: Addison-Wesley Iberoamericana

   Año: 1995

   Libro de algoritmos que se ha usado como referencia en el capítulo correspondiente. No tiene ejercicios especialmente orientados a la parte de programación, que sean complementarios con los aquí planteados.

Apéndice
Selección de ejercicios resueltos

Aunque en general es positivo que el estudiante se enfrente a los ejercicios sin poder ver de antemano la solución, también puede ser a veces frustrante el no tener una solución para comparar en determinados casos. Por ello, hemos incorporado esta sección en la cual se han incluido las soluciones de una serie de ejercicios seleccionados por ser interesantes, algo difíciles o simplemente representativos.

2.39

```
function clave=fclaveifsanidados(x)
if x<0
    clave=-1;
else
    if x<5
        clave=0;
    else
        if x<7
            clave=1;
        else
            if x<9
                clave=2;
            else
                if x<=10
                    clave=3;
                else
                    clave=-1;
                end
            end
        end
    end
end
```

2.51

```
function y=fposmayormenor(a,b,c)
if a>0 || b>0 || c>0
    y=ud2_fmayor(a,b,c);
else
    y=-ud2_fmayor(-a,-b,-c);
end
```

2.79

```
function media=fmediamayoresa(x,y,z,t,s,a)
contador=0;
sumamayoresquea=0;
if x>a
    contador=contador+1;
    sumamayoresquea=sumamayoresquea+x;
end
if y>a
    contador=contador+1;
    sumamayoresquea=sumamayoresquea+y;
end
if z>a
    contador=contador+1;
    sumamayoresquea=sumamayoresquea+z;
end
if t>a
    contador=contador+1;
    sumamayoresquea=sumamayoresquea+t;
end
if s>a
    contador=contador+1;
    sumamayoresquea=sumamayoresquea+s;
end
media=sumamayoresquea/contador;
```

2.83

```
% dos posibilidades para esta función
function suma=fsumaenteros(a,b,c,d)
suma=0;
if ud2_fesentero(a)==1
    suma=suma+a;
end
if ud2_fesentero(b)==1
    suma=suma+b;
end
if ud2_fesentero(c)==1
    suma=suma+c;
end
if ud2_fesentero(d)==1
    suma=suma+d;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function suma=fsumaenteros(a,b,c,d)
suma=a*ud2_fesentero(a)+b*ud2_fesentero(b)...
     +c*ud2_fesentero(c)+d*ud2_fesentero(d);
```

2.85

```
function suma=fentero_o_multiplo(a,b,c,d,n)
suma=0;
if ud2_fesentero(a)==1||ud2_fesentero(floor(a)/n)==1
  suma=suma+a;
end
if ud2_fesentero(b)==1||ud2_fesentero(floor(b)/n)==1
  suma=suma+b;
end
if ud2_fesentero(c)==1||ud2_fesentero(floor(c)/n)==1
  suma=suma+c;
end
if ud2_fesentero(d)==1||ud2_fesentero(floor(d)/n)==1
  suma=suma+d;
end
```

3.7

```
function YCG=fycgpoligonal(a,b,n)
sumador=0;
longtotal=0;
h=(b-a)/n;
xant=a;
yant=sin(xant);
i=1;
while i<=n
    xsig=a+i*h;
    ysig=sin(xsig);
    longsegmento=sqrt((xsig-xant)^2+(ysig-yant)^2);
    ycgsegmento=(ysig+yant)/2;
    sumador=sumador+longsegmento*ycgsegmento;
    longtotal=longtotal+longsegmento;
    i=i+1;
    xant=xsig;
    yant=ysig;
end
YCG=sumador/longtotal;
```

3.8

```
function YCG=fycgrectangulos(a,b,n)
sumador=0;
areatotal=0;
h=(b-a)/n;
i=0;
while i<=n-1
    x=a+i*h;
    y=sin(x);
    arearect=y*h;
    ycgrect=y/2;
    sumador=sumador+arearect*ycgrect;
    areatotal=areatotal+arearect;
    i=i+1;
end
YCG=sumador/areatotal;
```

3.29

```
function dif=fdifx2x1(n)
x1=3.57;
i=2;
while i<=n
    x2=(2*x1)^0.5;
    dif=abs(x2-x1);
    x1=x2;
    i=i+1;
end
```

3.24

```
function mipi=fmipi(n)
suma=0;
i=0;
signo=1;
while i<=n
    suma=suma+signo/(2*i+1);
    signo=-signo;
    i=i+1;
end
mipi=suma*4;
```

3.45

```
function mcd=fmcd(m,n)
i=1;
while i<=m  && i<=n
    if mod(m,i)==0 && mod(n,i)==0
        mcd=i;
    end
    i=i+1;
end
```

3.46

```
function mcm=fmcm(m,n)
i=m*n;
while i>=1
    if mod(i,m)==0 && mod(i,n)==0
        mcm=i;
    end
    i=i-1;
end
```

3.73

```
function ceros=fceros(n)  %304
ceros=0;
while n>=10
    resto=mod(n,10);  % 0
    if resto==0
        ceros=ceros+1; %ceros=1
    end
    n=n-resto;% n=30-0 = 30
    n=n/10; % 3
end
```

4.8

```
function media=fmediapositivos(v)
n=length(v);
suma=0;
cont=0;
i=1;
while i<=n
    if v(i)>0
        suma=suma+v(i);
        cont=cont+1;
    end
    i=i+1;
end
if cont>0
    media=suma/cont;
else
    media=0;
end
```

4.9

```
function flag=ftodospos(u)
flag=1;
i=1;
while i<=length(u)
    if u(i)<=0
        flag=0;
        break;
    end
    i=i+1;
end
```

4.25

```
function sumaposprimos=fsumaposprimos(v)
n=length(v);
i=1;
sumaposprimos=0;
while i<=n
    if ud3_fesprimo(v(i))==1
        sumaposprimos=sumaposprimos+i;
    end
    i=i+1;
end
```

4.36

```
function num=fmayorndivprimos(n)
div=faux(2);
num=2;
i=3;
while i<n
    if div<faux(i)
        num=i;
        div=faux(i);
    end
    i=i+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function prim=faux(n)
i=2;
prim=0;
while i<=n
    if mod(n,i)==0
        prim=prim+ud3_fesprimo(i);
    end
    i=1+i;
end
```

4.40

```
function maximo=fvauxdivisores(v)
n=length(v);
for i=1:n
    vaux(i)=ud3_fcuentadiv(v(i));
end
maximo=ud4_fmaximo(vaux);
```

4.43

```
function dismin=fclosetoavgpmsigma(x) % [3 2 1 7]
m=ud4_fmedia(x);
sigma=sqrt(ud4_fvarianza(x));
n=length(x);
% creo el vector auxiliar
i=1;
while i<=n
    if abs(x(i)-(m-sigma))<abs(x(i)-(m+sigma));
        vaux(i)=abs(x(i)-(m-sigma));
    else
        vaux(i)=abs(x(i)-(m+sigma));
    end
```

```
    i=i+1;
end
% ya tengo el vector auxiliar. Calculo minimo.
dismin=vaux(1);
i=2;
while i<=n
    if vaux(i)<dismin
        dismin=vaux(i);
    end
    i=i+1;
end
```

4.56

```
function segundomini=fsegundomin(u)
n=length(u);
mini=u(1);
i=2;
while i<=n
    if u(i)<mini
        mini=u(i);
    end
    i=i+1;
end
% ya tenemos el mini
i=1;
while u(i)==mini
    i=i+1;
end
segundomini=u(i);
% ya tengo un candidato para comparar.
i=1;
while i<=n
    if u(i)<segundomini && u(i)>mini
        segundomini=u(i);
    end
    i=i+1;
end
```

4.67

```
function primo1=fprimo1(v) %v=[30 37 14 13]
n=length(v);
i=1;
while i<=n
    primo=1; % supongo q v(i) es primo
    j=2;
    while j<=sqrt(v(i))
        if mod(v(i),j)==0
            primo=0;
            break;
        end
        j=j+1;
    end
    % llegados aquí, si primo=1 es q v(i) es primo
    % y si primo=0 es q v(i) NO ES primo.
    if  primo==1
        primo1=v(i);
        break;
    end
    i=i+1;
end
```

4.75

```
function flag=ftripleloop(v)
n=length(v);
flag=0;
i=1;
while i<=n
    j=1;
    while j<=n
        k=1;
        while k<=n
            if v(i)+v(j)==v(k) && ...
                i~=j && i~=k && j~=k
                flag=1;
            end
            k=k+1;
        end
        j=j+1;
    end
    i=i+1;
end
```

4.79

```
% variante llamando a una función auxiliar
function mcdmax=fmaxmcd(v)
%mcd entre v(1)&v(2) para inicializar
mcdmax=fmcd(v(1),v(2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n=length(v);
i=1;
while i<=n
    j=i+1;
    while j<=n
        mcdij=fmcd(v(i),v(j));
        if mcdij>mcdmax
            mcdmax=mcdij;
        end
        j=j+1;
    end
    i=i+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% variante sin función auxiliar (triple bucle)
function mcd=fmaxmcd(v)
% mcd entre v(1) & v(2)
mcd=1;
i=2;
while i<=v(1) && i<=v(2)
    if mod(v(1),i)==0 && mod(v(2),i)==0
        mcd=i;
    end
    i=i+1;
end
%%%%%
n=length(v);
i=1;
while i<=n
    j=1;
    while j<=n
        if i~=j  % calculo el mcd de v(i) y v(j)
            k=2;
            mcdij=1;
            while k<=v(i) && k<=v(j)
                if mod(v(i),k)==0 && mod(v(j),k)==0
                    mcdij=k;
                end
                k=k+1;
            end
            if mcdij>mcd
                mcd=mcdij;
            end
        end
        j=j+1;
    end
    i=i+1;
end
```

4.86

```
function criba=fcriba_eratostenes(n)
i=1;
while i<=n
```

```
    criba(i)=i;
    i=i+1;
end
%
i=2;
while i<=sqrt(n)
    mj=i*2;
    criba(mj)=0;
    j=2;
    while mj<=n
        criba(mj)=0;
        j=j+1;
        mj=j*i;
    end
    i=i+1;
end
criba(1)=0; % 1 no es primo
```

4.92

```
function y=ftaylorseno7(x)
n=length(x);
i=1;
while i<=n
    y(i)=x(i)-x(i)^3/factorial(3)+...
          x(i)^5/factorial(5)-...
          x(i)^7/factorial(7);
    i=i+1;
end
```

4.105

```
function vprimos=fconstrprimos0(v)
n=length(v);
i=1;
ip=0;
vprimos=[0];
while i<=n
    if ud3_fesprimo(v(i))==1
        ip=ip+1;
        vprimos(ip)=v(i);
    end
    i=i+1;
end
```

4.106

```
function vprimos=fconstrprimos(u)
n=length(u);
vprimos=[0];
i=1;
ip=1;
while i<=n
    primo=1;
    j=2;
    while j<=sqrt(u(i))
        if mod(u(i),j)==0
            primo=0;
            break
        end
        j=j+1;
    end
    if primo==1
        vprimos(ip)=u(i);
        ip=ip+1;
    end
    i=i+1;
end
```

4.107

```
function media=fconstrmediaprimos(u)
n=length(u);
vprimos=[0];
i=1;
k=1;
while i<=n
    primo=1;
    j=2;
    while j<=sqrt(u(i))
        if mod(u(i),j)==0
            primo=0;
            break
        end
        j=j+1;
    end
    if primo==1
        vprimos(k)=u(i);
        k=k+1;
    end
    i=i+1;
end
media=ud4_fmedia(vprimos);
```

4.108

```
function v=fconstrsinminmax(u)
n=length(u);
mayor=max(u);
menor=min(u);
i=1;
ip=1;
while i<=n
    if u(i)~=mayor && u(i)~=menor
        v(ip)=u(i);
        ip=ip+1;
    end
    i=i+1;
end
```

4.114

```
function w=fconstrsinrepes(v)
n=length(v);
i=1;
j=1;
while i<=n
    cont=0;
    k=1;
    while k<i
        if v(i)==v(k)
            cont=1;
            break
        end
        k=k+1;
    end
    if cont==0
        w(j)=v(i);
        j=j+1;
    end
    i=i+1;
end
```

4.118

```
function [x1,x2,error]=f2salidasx1x2error(a,b,c)
D=b^2-4*a*c;
if D<0
    x1=0;
    x2=0;
    error=1;
else
    x1=(-b-sqrt(D))/(2*a);
    x2=(-b+sqrt(D))/(2*a);
    error=0;
end
```

4.119

```
function [x1,x2,x3,x4]=f2salidasx4(a,b,c)
[y1,y2]=ud4_fe2grado(a,b,c);
x1=sqrt(y1);
x2=-sqrt(y1);
x3=sqrt(y2);
x4=-sqrt(y2);
```

4.125

```
function [I,E]=f2salidasIE(v)
n=length(v);
i=1;
I=0;
E=0;
while i<=n
    if mod(v(i),2)~=0
        I=i;
        E=v(i);
        break
    end
    i=i+1;
end
```

4.138

```
function p=ftaylorseno(n)
i=1;
signo=1;
while i<=n+1
    p(i)=0;
    p(i+1)=signo/factorial(i);
    signo=-signo;
    i=i+2;
end
```

4.139

```
function IP=fintegrapol(p)
gradoI=length(p);
IP(1)=0;
i=2;
while i<=gradoI+1
    IP(i)=p(i-1)/(i-1);
    i=i+1;
end
```

4.142

```
function ytaylor=fseno7(alpha)
p=[0  1 0 -1/3/2 0 1/5/4/3/2 0 -1/7/6/5/4/3/2];
ytaylor=ud4_fevalua(p,alpha);
```

4.144

```
function y=fevaluaderivadax(pol,x)
dpol=ud4_fderivapol(pol);
y=ud4_fevalua(dpol,x);
```

4.149

```
function y=ftaylorsenodex(x,n)
signo=1;
i=2;
while i<=n+1;
    PT(i)=signo/factorial(i-1);
    signo=-signo;
    i=i+2;
end
n=length(x);
i=1;
while i<=n
    y(i)=ud4_fevalua(PT,x(i));
    i=i+1;
end
```

4.155

```
function v=fevafibopol(p,n) % n>1 por hipotesis
vaux(1)=1;
cont=1;
x1=1;
x2=2;
x3=x1+x2;
while x3<n
    if mod(x3,2)==1
        cont=cont+1;
        vaux(cont)=x3;
    end
    x1=x2;
    x2=x3;
    x3=x1+x2;
end
%%%%%%%%%%%%%%%%%
i=1;
while i<=length(vaux)
    v(i)=ud4_fevalua(p,vaux(i));
    i=i+1;
end
```

5.6

```
% stipotriangulo
a=input('dame el primer lado:');
b=input('dame el segundo lado:');
c=input('dame el tercer lado:');
if a+b>c && c+a>b && b+c>a
    if a==b && a==c
        fprintf('Es un triángulo equilátero\n');
    elseif b~=a && b~=c
        fprintf('Es un triángulo es escaleno\n');
    else
        fprintf('Es un triángulo isósceles\n');
    end
else
    fprintf('Los valores introducidos NO son los lados de un triángulo\n');
end
```

5.19

```
% se2grado_warchivo
clear all;
a=input('Coeficiente de x^2: ');
b=input('Coeficiente de x: ');
c=input('Término independiente: ');
[r1,r2]=ud4_fe2grado(a,b,c);
%
fichero=fopen('e2grado.dat','w');
fprintf(fichero,'Primera solucion:  %g\nSegunda solucion %g',r1,r2);
fclose(fichero);
```

5.30

```
function [x,ysin]=fxyseno(a,b,n)
dx=(b-a)/n;
x=a:dx:b;
ysin=sin(x);
```

5.32

```
function [x,ytaylor,yseno]=f_xyseno_xytaylor(a,b,n)
dx=(b-a)/n;
x=a:dx:b;
yseno=sin(x);
ytaylor=x-x.^3/factorial(3)+x.^5/factorial(5)-x.^7/factorial(7);
```

5.33

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% w_xyseno_xytaylor.m
% GUI para el script s_xyseno_xytaylor
% Primera vez que usamos un interfaz con ventanas
clear all;
close all;
hboton=uicontrol('String','Dibujar','Position',[10 10 70 30],'Callback','s_xyseno_xytaylor');
% Dibujamos las etiquetas y las cajas para leer valores
hntxt  =uicontrol('Style','text','String','Nº de tramos','Position',[90 10 100 20]);
hn     =uicontrol('Style','edit','String',100,'Position',[200 10 50 20]);
hx1txt =uicontrol('Style','text','String','a','Position',[260 10 30 20]);
hx1    =uicontrol('Style','edit','String',-3,'Position',[300 10 50 20]);
hx2txt =uicontrol('Style','text','String','b','Position',[360 10 30 20]);
hx2    =uicontrol('Style','edit','String',5,'Position',[400 10 50 20]);
% Definimos el área de dibujo.
subplot('position',[0.1 0.2 0.8 0.7]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% s_xyseno_xytaylor.m
% callback de w_xyseno_xytaylor.m
n=str2num(get(hn,'String'));
a=str2num(get(hx1,'String')); %
b=str2num(get(hx2,'String'));
% Llamamos a ud4_fibo para calcular los términos
[x,ytaylor,yseno]=f_xyseno_xytaylor(a,b,n);
% Dibujamos el vector v
plot(x,ytaylor,x,yseno);
```

6.4

```
function primo=fesprimo_for(numero)
primo=1;
for i=2:sqrt(numero)
    if mod(numero,i)==0
        primo=0;
        break;
    end
end
```

6.12

```
function [traza,flag]=ftraza(M)
traza=0;
[n,m]=size(M);
if n==m
    flag=1;
    for i=1:n
        traza=traza+M(i,i);
    end
else
    flag=0;
end
```

6.13

```
function flag=fdiagonal(M)
[n,m]=size(M);
flag=1;
for i=1:n
    for j=1:m
        if M(i,j)~=0 && i~=j
            flag=0;
            break
        end
    end
    if flag==0
        break;
    end
end
```

6.15

```
function T=ftraspuesta(M)
[m,n]=size(M);
T=zeros(n,m);
for i=1:n
    for j=1:m
        T(i,j)=M(j,i);
    end
end
```

6.22

```
function [indf,indc]=fmenorprimomatriz(M)
[n,m]=size(M);
indf=0;
indc=0;
% Buscamos un primo de referencia.
for i=1:n
  for j=1:n
    if ud3_fesprimo(M(i,j))==1
        indf=i;
        indc=j;
        break;
    end
  end
  if indf~=0
    break;
  end
end
% Ahora buscamos entre los primos el menor.
for i=1:n
  for j=1:m
    if M(i,j)<M(indf,indc) && ud3_fesprimo(M(i,j))==1
      indf=i;
      indc=j;
    end
  end
end
```

6.23

```
% input_sadd_filainferior.dat
20  6  9
13 14 12
12 14 24

clear all
A=load('input_sadd_filainferior.dat');
B=fadd_filainferior(A);
save output_sadd_filainferior.dat B -ascii

function B=fadd_filainferior(A)
[m,n]=size(A);
B=A;
for i=1:m-1
    B(i,:)=A(i,:)+A(i+1,:);
end

  3.3000000e+001  2.0000000e+001  2.1000000e+001
  2.5000000e+001  2.8000000e+001  3.6000000e+001
  1.2000000e+001  1.4000000e+001  2.4000000e+001
```

6.25

```
function flag=figuales_mat(M)
[m,n]=size(M);
flag=0;
for i=1:m
  for j=1:n
    for k=1:m
      for l=1:n
        if (M(i,j)==M(k,l))&&(i~=k||j~=l)
          flag=1;
          break
        end
      end
      if flag==1;
        break
      end
    end
    if flag==1;
      break
    end
  end
  if flag==1;
    break
  end
end
```

6.27

```
function [flag,C]=fprodmat(A,B)
[m,n]=size(A);
[p,q]=size(B);
C=zeros(m,q);
if n==p
    flag=1;
    for i=1:m
        for j=1:q
            aux=0;
            for k=1:n
                aux=aux+(A(i,k)*B(k,j));
```

```
            end
            C(i,j)=aux;
        end
    end
else
    flag=0;
    C=0;
end
```

6.29

```
function N=fvecinos(M)
[m,n]=size(M);
N=zeros(m,n);
for i=1:m
    idown=max(i-1,1);
    iup=min(i+1,m);
    for j=1:n
        aux=0;
        for ip=idown:iup
            if ip==i-1 || ip==i+1
                jdown=j;
                jup=j;
            else
                jdown=max(j-1,1);
                jup=min(j+1,n);
            end
            for jp=jdown:jup
                aux=aux+M(ip,jp);
            end
        end
        N(i,j)=aux;
    end
end
```

6.34

```
function B=fsumacolsprimo(A)
[m,n]=size(A);
j=0;
suma=sum(A);
for i=1:n
    if ud3_fesprimo(suma(i))==1
        j=j+1;
        B(:,j)=A(:,i);
    end
end
if j==0
    B=0;
end
```

6.36

```
function B=fquitafilapcolq(A,p,q)
B=A;
B(p,:)=[];
B(:,q)=[];
```

7.11

```
function pos=fbinsearch_ubica(v,a)
n=length(v);
pos=1;
i=1;
j=n;
while j>=i
    k=floor((i+j)/2);
    if a<v(k)
        pos=i;
        j=k-1;
    else
        pos=j+1;
        i=k+1;
    end
end
```

7.31

```
% input_musgrande.dat
3 6 7 1
7 12 7 2

clear all;
A=load('input_musgrande.dat');
mano=A(1,:);
postre=A(2,:);
flag=fmusgrande(mano,postre);
fichero=fopen('output_musgrande.dat','w');
if flag==1
    fprintf(fichero,'El ganador es la mano');
else
    fprintf(fichero,'El ganador es el postre');
end
fclose(fichero);

function flag=fmusgrande(mano,postre)
for i=1:4
    if mano(i)==3
        mano(i)=12;
    end
    if postre(i)==3
        postre(i)=12;
    end
end
v=ud7_fseleccion(mano);
u=ud7_fseleccion(postre);
flag=1;% De salida, siempre gana la mano
for i=4:-1:1
    if v(i)<u(i)
        flag=0;%gana el postre
        break;
    end
end
% ----------------------------------------

El ganador es el postre
```

7.41

```
function A=fordenamatriz(A,c)
[n,m]=size(A);
for i=1:n-1
  imin=ud7_fimin(A(:,c),i,n);
  [A(imin,:),A(i,:)]=ud7_fswap(A(imin,:),A(i,:));
end
```

Índice alfabético

abs, 43, 61, 64, 110
algoritmos
    búsqueda, 185
    búsqueda secuencial, 186
    geométricos, 204
    inserción, 201
    ordenación, 193
    sacudidas, 199
ASCII, 44, 142

búsqueda secuencial, 186
búsqueda, algoritmos, 185
Ball, S., 19
binaria, búsqueda, 187
binario, archivo, 143
break, 92, 113
break line, 150
breakpoint, 68, 69
bucle infinito, 80
bucles, 79
bug, 163
burbuja, algoritmo, 198

clc, 138
clear all, 30, 43, 138, 151
close all, 151
comandos
    abs, 43, 61, 64, 110
    break, 92, 113
    clc, 138
    clear all, 30, 43, 138, 151
    close all, 151
    det, 39, 173
    diff, 48
    dir, 146
    doc, 45
    else, 60
    elseif, 66
    end, 80, 172
    eval, 48
    factorial, 84, 129
    floor, 78
    fopen, 144
    for, 80, 163
    fprintf, 137
    get, 151
    hold, 152
    if, 57
    if-else, 60
    if-elseif, 66
    if-end, 57
    input, 136
    legend, 152
    length, 101
    linspace, 117
    load, 145, 147
    max, 107, 111, 120, 169
    mean, 166
    min, 107, 111
    mod, 89
    polyval, 131
    rand, 190, 209
    rank, 39
    save, 143, 147
    shg, 45
    size, 32, 165
    sort, 193
    sortrows, 193
    sqrt, 61
    str2num, 151
    sum, 169
    tic, 190
    toc, 190
    while, 80, 163
contadores, 76

debugger, 68
    breakpoint, 68, 69
    watch, 69
depurador, 68
det, 39, 173
diff, 48
dir, 146

EISPACK, 22
ejemplos, 21
ejercicios propuestos, 21
else, 60
elseif, 66
end, 57, 80, 172
eval, 48

factorial, 84, 129
Fermat, teorema de, 191
Fibonacci, 87, 88
    sucesión, 149
ficheros
    ASCII, 44, 142
    crear, 142
    escribir, 142
    leer, 145
floor, 78

fopen, 144
for, 80, 163
FORTRAN, 22
fprintf, 137, 144
function, 51

Gauss, Carl Friedrich, 174
geométricos, algoritmos, 204
get, 151
Goldbach, Christian, 192
GUI, 148
 botones, 150
 cajas, 150
 callback, 150
 etiquetas, 150
 formularios, 150
 get, 151
 gráficas, 150
 handle, 150
 str2num, 151
 uicontrol, 150

hold, 152

if, 57
if-else, 60
if-elseif, 66
if-end, 57
input, 136
inserción, algoritmo, 201

Knuth, Donald, 17

LabVIEW, 148
LaTeX, 17
legend, 152
length, 101
LINPACK, 22
linspace, 117
Little, Jack, 22
load, 145, 147

MATLAB
 cálculo simbólico, 47
 carpeta activa, 44
 importar archivos, 146
 ventana de comandos, 25
max, 107, 111, 120, 169
mean, 166
min, 107, 111
mod, 89
Moler, Cleve, 22
Ms-ACCESS, 185
Ms-EXCEL, 144, 146, 147, 185
Ms-WORD, 142, 144
Muniz, Vic, 19

Newton, Isaac, 86

operadores lógicos, 69
 Ó, 69
 comparación de igualdad, 72
 negación, 69
 Y, 69
ordenación, algoritmos, 193

pi, 54
polígono, punto, inclusión, algoritmo, 207
polyval, 131

rand, 190, 209
rank, 39
run-test, algoritmo, 198

sacudidas, algoritmo, 199
save, 143, 147
script, 135, 137
secuencial, búsqueda, 186
Sedgewick, R., 185
segmentos, intersección, algoritmo, 206
selección, algoritmo, 196
shg, 45
size, 32, 165
sort, 193
sortrows, 193
sqrt, 61
str2num, 151
string, 150
sum, 169
sumadores, 76

Taylor, Brook, 42
TeX, 17, 18
tic, 190
toc, 190
tres puntos, orientación, algoritmo, 204

uicontrol, 150

Visual BASIC, 148
Von Neumann, 49

while, 80, 163
Wiles, Andrew, 191
Wittgenstein, L., 19

CURSO BÁSICO DE PROGRAMACIÓN EN MATLAB®

Este libro está concebido para que se pueda articular en torno a él un curso de introducción a la programación estructurada para titulaciones no informáticas utilizando el lenguaje de comandos de MATLAB como lenguaje de referencia. Incorpora explicaciones teóricas, ejemplos, ejercicios propuestos y resueltos, proyectos, etc. El objetivo que se busca es que al final del curso el estudiante haya asimilado los conceptos básicos de la programación estructurada y que se sienta cómodo dentro del entorno MATLAB.

Pretende ser relevante para los estudiantes y docentes de titulaciones ajenas al ámbito estrictamente informático y para profesionales de diferentes ámbitos (ingeniería, economía, matemáticas, biología, etc.), que necesitan una formación elemental en programación. Los elementos que conforman la programación estructurada se explican de manera organizada para que el lector los asimile y aplique con facilidad.

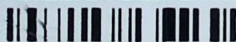
Antonio Souto Iglesias es doctor ingeniero naval por la Universidad Politécnica de Madrid (UPM) y profesor titular en la Escuela Técnica Superior de Ingenieros Navales (ETSIN) de la UPM. Ha coordinado las asignaturas de informática en las diferentes titulaciones de la ETSIN desde 1996 y desarrolla su labor investigadora sobre técnicas de simulación computacional en hidrodinámica en el grupo de investigación del canal de ensayos hidrodinámicos.

José Luis Bravo Trinidad es licenciado y doctor en matemáticas por la Universidad de Extremadura (UNEX) e ingeniero técnico en informática de sistemas por la UNED. Es profesor titular de matemática aplicada en la UNEX, donde imparte un curso de iniciación a MATLAB para profesores. Sus temas de investigación son ecuaciones diferenciales ordinarias, superficies de energía potencial en reacciones químicas y automatización agrícola con la empresa Agrobot.

Alicia Cantón Pire es licenciada y doctora en ciencias matemáticas por la Universidad Autónoma de Madrid (UAM). Ha realizado estancias de investigación e impartido docencia en la Universidad de Washington, en la Autónoma de Barcelona, y actualmente en la UPM. Su área de investigación en matemáticas se centra en la teoría geométrica de funciones y desde su incorporación a la ETSIN se ha interesado en el diseño geométrico asistido por ordenador.

Leo Miguel González Gutiérrez es doctor ingeniero industrial por el ICAI y licenciado en ciencias físicas por la UAM. La mayor parte de la docencia impartida ha sido en el entorno de la mecánica de fluidos. En cuanto a los temas de investigación, ha desarrollado diversos trabajos sobre mecánica de fluidos computacional y participado en proyectos con empresas de alto valor competitivo como Repsol o Airbus.

BIBLIOTECA UTN



055265

9 788473 605205

Editorial Tébar