Problemas de programación estructurada para Físicas y Matemáticas

José Carlos Ciria, Ángel R. Francés, Jorge Lloret y María Antonia Zapata

febrero de 2020

Preámbulo

¿Cómo se determina la órbita de un satélite artificial, en función del uso que se le quiera dar? ¿Cuál es la trayectoria de los rayos de luz cuando observamos espejismos? ¿Cómo se diseña una escala musical de modo que las combinaciones de notas (los acordes) sean agradables al oído y, a la vez, el compositor virtuoso pueda cambiar fácilmente de tonalidad? ¿Cómo se distribuyen los colores del arco iris? ¿Cuántas horas de luz tiene un día determinado, según la latitud? ¿Cómo puedo transmitir un mensaje de modo que nadie sea capaz de entenderlo aunque lo intercepte? ¿Cómo cambiaría la configuración del Parlamento español si se asignaran los escaños mediante un algoritmo alternativo a la regla d'Hont? ¿y si hubiera una circunscripción única? Desde la implantación del Grado en Física en nuestra Facultad venimos planteando estas y otras preguntas a los estudiantes de la asignatura "Informática" (código 26904).

Enfrentado a problemas cuya dificultad trasciende la de los ejercicios habituales, el estudiante tiene la oportunidad de afianzar las nociones presentadas en clase y de adquirir madurez como programador. Los problemas propuestos, pues, deben satisfacer los siguientes requisitos:

- estar estructurados según el temario propuesto. Evidentemente, deben estar estrechamente vinculados al desarrollo de la asignatura.
- ser lo suficientemente complicados para suponer un desafío. Afrontar problemas no triviales obliga a un esfuerzo de análisis y modelización, competencias esenciales en un programador.
- ser asequibles para un estudiante de primer curso.
- como valor añadido sería deseable que su resolución estimulara la curiosidad del estudiante y le ayudara a entender mejor y profundizar en conceptos vistos en otras asignaturas de Física y Matemáticas.

En este documento recogemos los problemas (cinco inéditos por curso) que hemos ido elaborando desde la primera edición de la asignatura, en el curso académico 2010-2011.

Los problemas se resuelven en equipos de tres personas, tutorizados por un profesor de prácticas. El plazo concedido para la resolución de cada problema oscila entre dos y tres semanas. El tutor realiza un seguimiento de la evolución del equipo a lo largo del curso. Orienta al equipo a petición de los estudiantes (que tienen todo tipo de facilidades para plantear sus dificultades, dudas y problemas), y a través de la retroalimentación que sigue a la entrega de cada solución.

Un aspecto importante, en el que el tutor insiste de forma recurrente, es la validación del código. En este documento se proponen juegos de pruebas para cada problema, de modo que el estudiante pueda hacer una primera validación informal de su programa. Insistimos en que, aunque el programa dé resultados correctos en esos casos, eso no implica necesariamente que está bien (puede ser que se haya tenido buena suerte al hacer las pruebas). Estas pueden ayudar a localizar errores, pero sólo se puede tener la certeza de que un programa es correcto analizándolo formalmente. Técnicas como los invariantes de bucle o la función de cota, explicadas y discutidas en clase, quedan fuera del alcance de este documento, si bien deben ser tenidas en cuenta en cualquier curso de introducción a la programación.

Los problemas se organizan siguiendo la estructura de la asignatura (sentencias estructuradas: condicionales y bucles; subalgoritmos; tipos de datos; ficheros). Algunos de ellos son independientes de los demás. Otros se articulan en un único proyecto, como los descritos a continuación:

- A lo largo del curso 2015-2016 el objetivo global perseguido fue familiarizar al estudiante con la criptografía RSA, permitiéndole encriptar y desencriptar mensajes usando sus propias claves pública y privada. El *proyecto RSA* se completa con dos apéndices. En el Apéndice A presentamos el sistema criptográfico RSA. Su belleza reside en su sencillez: se basa en propiedades matemáticas fundamentales de Teoría de Números; no hemos podido resistir la tentación de dedicar un segundo apéndice (Apéndice B) a sus fundamentos matemáticos.

- En el curso 2017-2018 se abordó cuantitativamente el **problema de los dos cuerpos** (*proyecto 2C*). Aplicando el método de bisección a la ley de Kepler se calculó, con precisión arbitraria, la posición relativa de dos planetas en un instante de tiempo dado.

- A lo largo del curso 2018-2019 se introdujo a los estudiantes en los problemas de optimización. Mediante un ejemplo simplificado (la gestión de repartos de un **servicio de transporte**), el *proyecto ST* ilustra el impacto que dichos problemas tienen en los ámbitos económico y de gestión de recursos.

- Las tareas 1.9, 2.9, 3.7 y 5.9 permiten estudiar la dinámica del oscilador armónico y del péndulo simple. Se complementan con un apéndice introductorio sobre uno de los aspectos más fascinantes de la dinámica de este último: el comportamiento caótico en sistemas deterministas.

Algunos problemas requieren materiales de apoyo (librerías, ficheros con ejemplos...). Dichos materiales están disponibles en este curso ADD.

La labor de recopilación y formateado de estos materiales se llevó a cabo dentro de los proyectos PIIDUZ_15_227, y PRAUZ_18_032, correspondientes al Programa de Incentivación de la Innovación Docente en la Universidad de Zaragoza.

Índice general

Capítulo 1

Sentencias condicionales

1.1. Cálculo de órbitas

1.1.1. Introducción

En esta tarea consideraremos un satélite que gira en torno a la Tierra. El objetivo será calcular el radio y el periodo de su órbita en función del uso que se quiera dar al satélite.

Empecemos recordando que la dinámica del satélite viene descrita por las leyes de Kepler (que, a su vez, se derivan de las de Newton). En particular, el satélite describe una órbita elíptica; la relación entre el semieje mayor de la elipse, a, y el periodo de la órbita, T, viene dada por:

$$a^3 = \frac{G \cdot M_T}{4\pi^2} T^2 \tag{1.1}$$

donde G es la constante de gravitación universal y M_T la masa de la Tierra [1]. Por simplicidad, a lo largo de esta tarea supondremos órbitas circulares.

Clasificaremos las órbitas de los satélites en función de la utilidad de los mismos: emisores de señales de televisión, emisores de señales para GPS o destinados a la toma de imágenes.

Los satélites que emiten señales de televisión deben mantener constante su posición relativa respecto de cualquier punto de la superficie del planeta. Ello permite que, una vez que se orienta una antena parabólica, ésta permanece apuntando al satélite continuamente (¡de otro modo, habría que reorientar continuamente las antenas!). Por tanto, deben ser geoestacionarios: orbitan en torno al Ecuador, con el mismo periodo de rotación que la Tierra.

Un receptor GPS recibe señales de varios de los satélites GPS que orbitan en torno a Tierra, emitiendo señales de radio. El receptor determina la distancia que le separa de cada satélite midiendo el tiempo que le cuesta llegar a su señal. Comparando varias distancias, la posición se determina por triangulación. Los satélites utilizados para el Sistema de Posicionamiento Global (GPS) no siguen órbitas geoestacionarias. La razón básica es que a esa distancia se requerirían emisores de radio muy potentes. Por otro lado, sus órbitas no pueden estar demasiado cerca de la Tierra, ya que entonces pasarían la mayor parte del tiempo por debajo de la línea del horizonte, ocultos a los receptores GPS.

[1] La expresión 1.1 es fácil de obtener si la órbita es circular: basta igualar las expresiones de las fuerza gravitatoria y centrípeta. Para el caso general de órbitas elípticas, recomendamos la lectura de [17].

Para la toma de imágenes es necesario tener en cuenta una característica esencial de un dispositivo óptico (cámara fotográfica, ojo, telescopio...): su poder de resolución, es decir su capacidad para distinguir entre dos puntos muy próximos. El límite de dicho poder viene impuesto por la difracción, que es consecuencia de la naturaleza ondulatoria de la luz. Supongamos un dispositivo de diámetro D con el que queremos observar puntos que se encuentran a una distancia d; según el criterio de Rayleigh, es posible distinguir entre dos puntos que distan entre sí una distancia Δ, donde

$$\Delta \geq 1{,}22\ \frac{\lambda \cdot d}{D}\ , \tag{1.2}$$

siendo λ la longitud de onda de la luz (para la luz visible, $\lambda \approx 5 \cdot 10^{-7}$ m) [2]. En nuestro caso, $d = h$, la altura del satélite sobre la superficie de la Tierra.

Cuanto mejor sea el poder de resolución deseado (cuanto más pequeños sean los detalles que queremos captar) más cerca tiene que estar el sistema del objeto. Si, por ejemplo, queremos hacer el seguimiento de un huracán es suficiente con resolver objetos de 1 km. de tamaño. Suponiendo que el diámetro del sistema óptico $D = 1$ m, se obtiene $d \approx 10^3/(1{,}22 \cdot 5 \cdot 10^{-7}) = 1{,}64 \cdot 10^9$ m $= 1{,}64 \cdot 10^6$ km. Si, en cambio, necesitamos distinguir puntos separados un decímetro, la distancia máxima a que debe estar el sistema sería $d \approx 10^{-1}/(1{,}22 \cdot 5 \cdot 10^{-7}) = 1{,}64 \cdot 10^5$ m $= 164$ km.

A los interesados en profundizar más sobre el tema les recomendamos la lectura de [24].

1.1.2. Descripción de la tarea

Diseña e implementa un algoritmo para el cálculo de la órbita circular de un satélite.

El programa solicita al usuario que introduzca un carácter, indicando qué finalidad quiere darle a su satélite:

- Si el usuario pulsa `'t'` o `'T'`: interesa un satélite para emisión de TV. El programa informa al usuario (por pantalla) que el satélite es geoestacionario. A partir del periodo de su órbita (24 horas) el programa calcula el radio (en km) y lo muestra por pantalla.
- Si el usuario pulsa `'i'` o `'I'`, pretende usar el satélite para tomar imágenes de la Tierra. El programa pregunta cuál es la resolución deseada para las imágenes (en metros); a partir de ella, calcula la altura máxima h a la que el satélite debe estar sobre la superficie de la Tierra y el radio r de su órbita ($r = R_T + h$, donde R_T es el radio de la Tierra). Para ello supón que el diámetro del dispositivo óptico D es un metro. A continuación analiza cuál de los siguientes tres casos se da:
  - Si el radio máximo r es superior al de una órbita geoestacionaria, el programa sugiere al usuario que utilice un satélite geoestacionario, ya disponible.
  - Si la altura máxima h es inferior a 120 km, el programa informa al usuario que una órbita tan próxima a la superficie de la Tierra es desaconsejable.
  - Si no se da ninguno de los supuestos anteriores, el programa informa al usuario del radio r y del periodo de la órbita del satélite.

[2] En realidad, el criterio de Rayleigh establece que $\text{seno}(\theta) \geq 1{,}22\ \frac{\lambda}{D}$, donde θ es el ángulo subtendido por los puntos que queremos distinguir. En la fórmula 1.2 hemos hecho la aproximación $\text{seno}(\theta) \simeq \theta \simeq \Delta/d$, válida para ángulos muy pequeños. Puede encontrarse un tratamiento más detallado en [33]

- Si el usuario pulsa `'g'` o `'G'`, desea utilizar el satélite para emitir señales GPS. En ese caso, el satélite debe describir una órbita MEO (Medium Earth Orbit), con altura h entre 10000 y 20000 km. El programa informa al usuario de los radios mínimo y máximo de la órbita, junto con sus periodos correspondientes.
- Si el usuario pulsa cualquier otro carácter, el programa escribe un mensaje de error ("Opcion no valida") y termina.

1.1.3. Ejemplos de ejecución

```
Utilidad del satelite:
     Para un emisor de TV, introduce 't' o 'T'
     Para tomar imagenes, introduce 'i' o 'I'
     Para un satelite GPS, introduce 'g' o 'G'


 Introduce la utilidad deseada: T

 Necesitas una orbita geoestacionaria, de radio = 42226.9 km
```

```
Utilidad del satelite:
     Para un emisor de TV, introduce 't' o 'T'
     Para tomar imagenes, introduce 'i' o 'I'
     Para un satelite GPS, introduce 'g' o 'G'


 Introduce la utilidad deseada: I
 Introduce la resolucion deseada (en metros): 0.075

 Altura = 125000 m.
 Radio: 6495 km  y periodo 1.44776 horas = 5211.93 segundos
```

```
Utilidad del satelite:
     Para un emisor de TV, introduce 't' o 'T'
     Para tomar imagenes, introduce 'i' o 'I'
     Para un satelite GPS, introduce 'g' o 'G'


 Introduce la utilidad deseada: g

 Orbita MEO:
       radio minimo: 16370 km  y periodo 5.79296 horas = 20854.6 segundos
       radio maximo: 26370 km  y periodo 11.8438 horas = 42637.8 segundos
```

1.1.4. Ayudas

En tus cálculos, utiliza los siguientes valores:
$G = 6{,}67 \cdot 10^{-11}$ m^3 kg^{-1} s^{-2} (constante de gravitación universal).
$M_T = 5{,}97 \cdot 10^{24}$ kg (masa de la Tierra)
$R_T = 6{,}37 \cdot 10^{6}$ m (radio de la Tierra).

1.2. Modelo atómico de Bohr

1.2.1. Introducción

Para explicar los espectros de emisión y absorción discretos observados en gases, Niels Bohr propuso en 1913 un modelo atómico que, partiendo de la Física clásica, incorporaba postulados de cuantización, inéditos hasta entonces. Su modelo se basa en cuatro postulados:

1. Los electrones se mueven en órbitas circulares en torno al núcleo, sujetos a la ley de Coulomb, según las leyes de la mecánica clásica:

   $$K\frac{Zq_e^2}{r^2} = \frac{m_e v^2}{r}$$

   donde K es la constante de Coulomb en el vacío, Z el número atómico, q_e y m_e respectivamente la carga y la masa reducida del electrón, v la velocidad y r el radio de la órbita.

2. Sólo son posibles aquellas órbitas cuyo momento angular sea un múltiplo entero de la constante reducida de Planck:

   $$L = n\hbar$$

   donde $\hbar = \frac{h}{2\pi}$ y h es la constante de Planck.

   Por tanto, el radio y la energía de la órbita n-ésima vienen dados por

   $$r_n = n^2 r_1$$

   $$E_n = \frac{E_1}{n^2}$$

   con

   $$r_1 = \frac{h^2}{4\pi^2 K Z q_e^2 m_e} \tag{1.3}$$

   y

   $$E_1 = -\frac{2\pi^2 m_e K^2 Z^2 q_e^4}{h^2} \tag{1.4}$$

3. Al moverse en una órbita permitida el electrón, pese a experimentar una aceleración continua, no emite radiación electromagnética. En las órbitas permitidas, la energía del electrón permanece constante.

4. Se emite energía electromagnética cuando un electrón, inicialmente en una órbita de energía E_i, cambia su movimiento de manera discontinua saltando a otra órbita de energía E_f. La energía y la longitud de onda de la radiación emitida son:

   $$\delta E = E_1\left(\frac{1}{n_i^2} - \frac{1}{n_f^2}\right) \tag{1.5}$$

$$\lambda = \frac{hc}{|\delta E|} \tag{1.6}$$

siendo h la constante de Planck y c la velocidad de la luz en el vacío.

1.2.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que haga las siguientes tareas:

1. Calcule el radio y la energía de la primera órbita del átomo de Hidrógeno (fórmulas 1.3 y 1.4, con Z = 1), y los muestre por pantalla.
2. Solicite al usuario qué caso desea considerar: (a) el salto de un electrón entre dos órbitas cualesquiera o bien (b) el salto correspondiente a la primera línea de una serie del espectro. El usuario elegirá una opción u otra tecleando uno de los caracteres 'a' o 'b', respectivamente.

   - Si el usuario elige una opción distinta de 'a' o 'b', el programa mostrará un mensaje de error y terminará.
   - Si el usuario escoge la opción (a), el programa le solicitará que introduzca los valores de los niveles inicial y final, n_i y n_f.
   - Si el usuario escoge la opción (b), el programa le solicitará que escoja qué serie le interesa. Las opciones son: Lyman (L), Balmer (B), Paschen (P), Brackett (R) y Pfund (F). El usuario elegirá una opción u otra tecleando uno de los caracteres 'L', 'B', 'P', 'R' o 'F', respectivamente. La línea corresponde al salto entre los niveles $n_i = n_f + 1$ y n_f, donde n_f es:
     - para la serie de Lyman, n_f =1
     - para la serie de Balmer, n_f =2
     - para la serie de Paschen, n_f =3
     - para la serie de Brackett, n_f =4
     - para la serie de Pfund, n_f =5

     Si el usuario elige una opción distinta, el programa mostrará por pantalla un mensaje de error, indicando que considera por defecto la serie de Lyman ($n_i = 2$, $n_f = 1$)

     Una vez conocidos los niveles inicial y final, el programa calculará la energía y la longitud de onda asociadas al salto, usando las fórmulas 1.5 y 1.6, y los mostrará en pantalla.

1.2.3. Ejemplos de ejecución

```
Radio primera orbita: 5.29628e-11 m
Energia primera orbita: -2.1774e-18 J

(a) Salto entre dos orbitas
(b) Primera linea de una serie del espectro
--- Escoja opcion: b

Escoja serie:
(L) Lyman
```

```
(B) Balmer
(P) Paschen
(R) Brackett
(F) Pfund
--- Escoja opcion: B

Energia emitida: 3.02417e-19 J
Longitud de onda: 6.56866e-07 m
```

1.2.4. Ayudas

Valida tu programa comparando sus resultados con los valores reales, que puedes encontrar en cualquier libro de Física o en la red.

Constantes

Constante de Coulomb: $K = 8,987 \cdot 10^9\,\mathrm{Nm^2C^{-2}}$
Masa reducida del electrón: $m_e = 9,104 \cdot 10^{-31}\,\mathrm{kg}$
Carga elemental: $q_e = 1,602 \cdot 10^{-19}\,\mathrm{kg}$
Constante de Planck: $h = 6,626 \cdot 10^{-34}\,\mathrm{Js}$
Velocidad de la luz en el vacío $c = 2,998 \cdot 10^8\,\mathrm{ms^{-1}}$

1.3. Cálculo de volúmenes y densidades

1.3.1. Introducción

Existen diversas maneras de obtener el volumen de un sólido:

- Si tiene forma geométrica, aplicando la fórmula correspondiente:
  - El volumen de un ortoedro de lados a_1, a_2 y a_3 es $V = a_1 a_2 a_3$.
  - El volumen de un cilindro de radio r y altura h es $V = \pi r^2 h$.
  - El volumen de una esfera de radio r es $V = \frac{4}{3}\pi r^3$.
- Aplicando el principio de Arquímedes, según el cual el volumen de un cuerpo (expresado en litros) es la diferencia entre la masa del cuerpo y su masa equivalente al sumergirlo en agua (expresados en kg). Para ello hay que tener en cuenta que (1) el *peso equivalente* de un cuerpo sumergido en un fluido es la resultante del peso y el empuje ejercido por el fluido y (2) la *masa equivalente* se define como el peso equivalente dividido por g (la intensidad del campo gravitatorio).

1.3.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que calcule el volumen y la densidad de un cuerpo.

El algoritmo empieza solicitando al usuario que introduzca la masa del cuerpo, expresada en kg. A continuación, le solicita que especifique el método para obtener el volumen: aplicando una fórmula geométrica o utilizando el principio de Arquímedes. En el primer caso, el usuario tecleará 'G' o 'g'; en el segundo, 'A' o 'a'.

- Si el usuario ha elegido la primera opción, el algoritmo le solicita que especifique la forma del cuerpo: ortoedro ('o'), esfera ('e') o cilindro ('c') y sus medidas (los tres lados en el primer caso, el radio en el segundo, el radio de la base y la altura en el tercero). Las medidas se suponen expresadas en metros.
- Si el usuario ha optado por seguir el método de Arquímedes, el algoritmo le solicita que introduzca la masa equivalente del cuerpo sumergido en agua (expresada en kg).
- Si el usuario ha tecleado un valor no válido para el método o la forma del cuerpo, el programa muestra un mensaje de error y termina. Si los datos son correctos, el programa calcula el volumen y la densidad y los escribe en la pantalla.

1.3.3. Ejemplos de ejecución

```
Introduce la masa del cuerpo (expresada en Kg):  10
Introduce el metodo para obtener el volumen:
   'G' o 'g': Formula geometrica
   'A' o 'a': Aplicando Arquimedes
metodo> z

El metodo introducido no es correcto
```

```
Introduce la masa del cuerpo (expresada en Kg):  10
Introduce el metodo para obtener el volumen:
   'G' o 'g': Formula geometrica
   'A' o 'a': Aplicando Arquimedes
metodo> g

Indica la forma del cuerpo:
   'O' u 'o': ortoedro
   'E' o 'e': esfera
   'C' o 'c': cilindro
forma> z

La forma geometrica no esta reconocida por el programa
```

```
Introduce la masa del cuerpo (expresada en Kg):  10
Introduce el metodo para obtener el volumen:
   'G' o 'g': Formula geometrica
   'A' o 'a': Aplicando Arquimedes
metodo> g

Indica la forma del cuerpo:
   'O' u 'o': ortoedro
   'E' o 'e': esfera
   'C' o 'c': cilindro
forma> o

Introduce las longitudes de los lados del ortoedro (en m): 2 3 4

El volumen es 24 m3
La densidad es 0.416667 Kg/m3
```

```
Introduce la masa del cuerpo (expresada en Kg):  5
Introduce el metodo para obtener el volumen:
   'G' o 'g': Formula geometrica
   'A' o 'a': Aplicando Arquimedes
metodo> A

Introduce la masa equivalente en agua (en kg): 3.06

El volumen es 0.00194 m3
La densidad es 2577.32 Kg/m3
```

1.4. Trabajo con densidades

1.4.1. Introducción

La *densidad* de un cuerpo se define como el cociente de su *masa* por su *volumen*. Conocidos cualesquiera dos de estos valores, el tercero se calcula mediante una simple división o un producto. Pero para expresar correctamente el resultado, además de conocer los valores cuantitativos, hay que saber en qué unidades físicas están expresados los datos.

1.4.2. Descripción de la tarea

Dada la masa de una cierta cantidad de líquido se desea calcular el volumen que ocupa. El usuario podrá elegir de qué líquido se trata y en qué unidades físicas se indica la masa. El resultado se expresará siempre en litros.

El programa empieza pidiendo al usuario que especifique con qué sustancia desea trabajar. Los valores posibles son: agua ('a' o 'A'), gasolina ('g' o 'G'), keroseno ('k' o 'K') y mercurio ('m' o 'M'). A continuación, el usuario debe introducir el valor de la masa, indicando su cantidad y unidades. Las unidades posibles son: libra ('l' o 'L'), onza ('o' o 'O') y kilo ('k' o 'K'). Conocida la densidad de la sustancia, el programa calcula el volumen correspondiente a la masa indicada y lo muestra por pantalla (expresado en litros). Si el usuario introduce un valor no válido para la sustancia o las unidades de masa, el programa mostrará en la pantalla un mensaje de error.

Puedes utilizar los datos de las siguientes tablas:

Densidades:

Sustancia	Densidad (kg/l)
Agua	1,0
Gasolina	0,68
Keroseno	0,8
Mercurio	13,6

Unidades de masa:

1 libra = 0,454 kg

1 onza = $28{,}3 \cdot 10^{-3}$ kg

1.4.3. Ejemplos de ejecución

Comprueba si tu programa hace los cálculos esperados al menos en los ejemplos siguientes.

```
Indica la sustancia con la que vamos a trabajar:
   agua:     teclea 'a' o 'A'
   gasolina: teclea 'g' o 'G'
   keroseno: teclea 'k' o 'K'
   mercurio: teclea 'm' o 'M'
sustancia> z

Sustancia no valida
```

```
Indica la sustancia con la que vamos a trabajar:
   agua:     teclea 'a' o 'A'
   gasolina: teclea 'g' o 'G'
   keroseno: teclea 'k' o 'K'
   mercurio: teclea 'm' o 'M'
```

```
sustancia> M

Indica la masa (cantidad y unidades de medida):
   kilo:    teclea 'k' o 'K'
   libra:   teclea 'l' o 'L'
   onza:    teclea 'o' o 'O'
cantidad> 33
unidad> z

Unidades de masa no validas
```

```
Indica la sustancia con la que vamos a trabajar:
   agua:     teclea 'a' o 'A'
   gasolina: teclea 'g' o 'G'
   keroseno: teclea 'k' o 'K'
   mercurio: teclea 'm' o 'M'
sustancia> M

Indica la masa (cantidad y unidades de medida):
   kilo:    teclea 'k' o 'K'
   libra:   teclea 'l' o 'L'
   onza:    teclea 'o' o 'O'
cantidad> 1.5
unidad> l

Volumen: 0.0500735 litros
```

```
Indica la sustancia con la que vamos a trabajar:
   agua:     teclea 'a' o 'A'
   gasolina: teclea 'g' o 'G'
   keroseno: teclea 'k' o 'K'
   mercurio: teclea 'm' o 'M'
sustancia> K

Indica la masa (cantidad y unidades de medida):
   kilo:    teclea 'k' o 'K'
   libra:   teclea 'l' o 'L'
   onza:    teclea 'o' o 'O'
cantidad> 22300
unidad> k

Volumen: 27875 litros
```

1.4.4. Para saber más...

Cuando se indica el valor de una magnitud es necesario especificar, junto con la cantidad, la unidad de medida. Sin este último dato la información está incompleta, lo cual puede dar lugar a situaciones de todo tipo.

Esto es lo que ocurrió el 23 de julio de 1983 en el vuelo 143 de Royal Canada Air de Montreal a Edmonton, vía Ottawa. El avión tenía que repostar. El cálculo que hicieron los técnicos de tierra fue: "Para el vuelo hacen falta 22 300 kg de queroseno. En el depósito

del avión ya hay 7682 litros. Para pasar de masa a volumen hay que dividir por 1,76. Por tanto necesitamos repostar $22\,300/1{,}76 - 7682 = 4988$ litros".

El razonamiento era correcto, salvo un detalle: la densidad del queroseno es, en efecto, de 1,76 libras/litro o, lo que es lo mismo, 1,76 **libras/litro** $\times$ 0,454 **kg/libra** = 0,8 **kg/litro** (en aquella época, en Canadá se estaba pasando del Sistema Imperial al Sistema Internacional, y confusiones de este tipo no eran infrecuentes). El cálculo correcto era:

$$\frac{22\,300 \text{ kg}}{0{,}8 \text{ kg/litro}} - 7682 \text{ litros} = 27\,875 \text{ litros} - 7682 \text{ litros} = 20\,193 \text{ litros}.$$

¡¡Un valor cuatro veces mayor que el calculado por los técnicos!!

El epílogo forma ya parte de la historia de la aviación: cuando el avión, un Boeing 767 con 61 pasajeros y 5 tripulantes, llevaba algo más de una hora de vuelo, a unos 12 500 metros de altura, se quedó sin combustible y todos sus motores se detuvieron. Puedes encontrar más información sobre el incidente buscando "Gimli Glider" en Internet.

Otro ejemplo lo tenemos en la historia de la sonda espacial Mars Climate Orbiter, que se destruyó "debido a un error de navegación, consistente en que el equipo de control en la Tierra hacía uso del Sistema Anglosajón de Unidades para calcular los parámetros de inserción y envió los datos a la nave, que realizaba los cálculos con el sistema métrico decimal" [23].

1.5. Proyecto RSA: codificación-decodificación

1.5.1. Introducción

El primer paso para encriptar un mensaje es codificarlo: transformarlo en un número entero. Ese proceso recibe el nombre de codificación. Formalmente hablando, la codificación es un proceso que transforma datos de un sistema de representación en datos de otro sistema. La información contenida en ambos datos (origen y resultante) debe ser la misma.

Hemos visto en clase cómo en C/C++ se codifican datos de distinto tipo (entero con/sin signo, real, carácter): se transforman en (son representados por) series de dígitos binarios (números representados en el sistema binario).

El objetivo de esta tarea es familiarizarnos con el proceso de codificación. Por simplicidad, nos restringiremos a la representación de caracteres mediante números enteros.

1.5.2. Descripción de la tarea

Queremos codificar las letras mayúsculas y minúsculas dando lugar a códigos numéricos, así como decodificar los códigos numéricos dando lugar a las correspondientes letras. Excluimos los caracteres 'ñ', 'Ñ' y las vocales acentuadas.

Cada letra mayúscula se codifica de acuerdo con la siguiente regla: La letra 'A' se codifica con el número 0, la letra 'B' con el número 1 y así sucesivamente hasta la letra 'Z' que se codifica con el número 25. Respecto a las minúsculas la letra 'a' se codifica con el número 26, la letra 'b' con el número 27 y así sucesivamente hasta la letra 'z' que se codifica con el número 51. La codificación completa se muestra en el Cuadro 1.1.

Diseña y programa en C/C++ un algoritmo que tenga el siguiente comportamiento:

Empieza pidiendo al usuario que introduzca una opción: 'c' o 'C' para codificar, 'd' o 'D' para decodificar.

Si el usuario decide codificar, a continuación se le pide que introduzca un carácter. El programa escribe un entero en pantalla (el código asociado al carácter según el Cuadro 1.1).

Si el usuario decide decodificar, a continuación se le pide que introduzca un entero entre 0 y 51. El programa escribirá en pantalla un carácter (el correspondiente al código introducido, según el Cuadro 1.1).

Si el usuario introduce una opción no válida, un carácter o un entero fuera de los valores establecidos en el Cuadro 1.1, el programa escribirá un mensaje de error en pantalla.

Cuadro 1.1: Codificación.

Carácter	Código	Carácter	Código	Carácter	Código	Carácter	Código
A	0	N	13	a	26	n	39
B	1	O	14	b	27	o	40
C	2	P	15	c	28	p	41
D	3	Q	16	d	29	q	42
E	4	R	17	e	30	r	43
F	5	S	18	f	31	s	44
G	6	T	19	g	32	t	45
H	7	U	20	h	33	u	46
I	8	V	21	i	34	v	47
J	9	W	22	j	35	w	48
K	10	X	23	k	36	x	49
L	11	Y	24	l	37	y	50
M	12	Z	25	m	38	z	51

1.5.3. Ejemplos de ejecución

```
Indica que quieres hacer
    para codificar: tecla 'c' o 'C'
    para decodificar: tecla 'd' o 'D'

Introduce tu opcion: c
Introduce el caracter: A

El codigo de 'A' es: 0
```

```
Indica que quieres hacer
    para codificar: tecla 'c' o 'C'
    para decodificar: tecla 'd' o 'D'

Introduce tu opcion: d
Introduce el codigo: 0

El caracter decodificado es: 'A'
```

```
Indica que quieres hacer
    para codificar: tecla 'c' o 'C'
    para decodificar: tecla 'd' o 'D'

Introduce tu opcion: X

Operacion no valida
```

```
Indica que quieres hacer
    para codificar: tecla 'c' o 'C'
    para decodificar: tecla 'd' o 'D'

Introduce tu opcion: c
Introduce el caracter: $

Caracter no valido
```

```
Indica que quieres hacer
    para codificar: tecla 'c' o 'C'
    para decodificar: tecla 'd' o 'D'

Introduce tu opcion: D
Introduce el codigo: 100

Codigo no valido
```

1.5.4. Ayudas

En C/C++ un carácter se interpreta como un entero, cuyo valor viene dado por la tabla ASCII. Los códigos de las letras mayúsculas tienen valores correlativos (‘B’ = ‘A’ + 1, ‘C’ = ‘A’ + 2 ...), igual que los de las minúsculas (‘b’ = ‘a’ + 1, ‘c’ = ‘a’ + 2 ...).

Te aconsejamos que uses una sentencia `switch` para elegir la operación (codificación o decodificación) que se va a realizar.

1.6. Unidades astronómicas

1.6.1. Introducción

Desde la antigüedad el hombre ha estado interesado en conocer cosas sobre el universo que le rodea. Hemos querido saber acerca de la Luna, el Sol o las estrellas.

Aristarco de Samos (310 a. C.- 230 a. C.) fue la primera persona que trató de medir la distancia de la Tierra al Sol. Sus mediciones fueron relativas y concluyó que el Sol estaba 20 veces más lejos que la Luna, cuando en realidad está 400 veces más lejos.

Eratóstenes (Cirene, 276 a. C. - Alejandría, 194 a. C.) se hizo famoso por calcular el radio de la Tierra. El resultado fue que la circunferencia polar de la Tierra era de 252.000 estadios, es decir, 33.918 kilómetros. Suponiendo que Eratóstenes usase el estadio ático-italiano de 184,8 m, que era el que solía utilizarse por los griegos de Alejandría en aquella época, el error cometido sería de 6.192 kilómetros.

En 1673 Cassini, sobre la base de datos de Kepler, determinó que la distancia de la Tierra al Sol era de 136 millones de kilómetros, por lo que sólo cometió un error del 7 % sobre la distancia real.

Para medir distancias en Astronomía, se han usado varias unidades: Unidad astronómica, año luz, parsec y kilómetro. La unidad astronómica (abreviada ua, au, UA o AU) es una unidad de longitud igual por definición a $1{,}496 * 10^8$ kilómetros y que equivale aproximadamente a la distancia media entre el planeta Tierra y el Sol. Un año luz es la distancia que recorre la luz en un año y es igual a 365,25*24*3600*300000 km. El párse o parsec (abreviada pc) es una unidad de longitud cuyo nombre se deriva del inglés parallax of one arc second (paralaje de un segundo de arco o arcosegundo). En sentido estricto párse se define como la distancia a la que una unidad astronómica subtiende un ángulo de un segundo de arco (1"). En otras palabras, una estrella dista un párse si su paralaje es igual a 1 segundo de arco. De la definición resulta que 1 párse = 206265 ua.

Algunas distancias entre la Tierra y objetos del Universo se muestran en la Tabla 1.2. La Voyager 1 es una sonda espacial lanzada el 5 de septiembre de 1977 desde Cabo Cañaveral, Florida. Permanece operacional actualmente, prosiguiendo su misión extendida que es localizar y estudiar los límites del sistema solar. Es el objeto hecho por el ser humano que se encuentra más alejado de la Tierra, a una distancia en junio de 2016 de $2,02 \cdot 10^{10}$ kilómetros. Próxima Centauri es la estrella más cercana al sistema solar. La Vía Láctea es la galaxia donde se encuentra el sistema solar. Su diámetro medio se estima en unos 100.000 años luz y se calcula que contiene entre 200.000 millones y 400.000 millones de estrellas. La galaxia de Andrómeda tiene un diámetro de 220.000 años luz y contiene aproximadamante un billón de estrellas. Es el objeto visible a simple vista más lejano de la Tierra.

Puedes encontrar información completa sobre este tema en [34].

1.6.2. Descripción de la actividad

En esta actividad vas a hacer dos tipos de cálculo. En el primero, dada una unidad de distancia, que puede ser unidad astronómica, kilómetro, párse o año luz, y su correspondiente distancia, calculará su equivalencia en el resto de las unidades. En el segundo cálculo, el usuario elige un objeto cuya distancia a la tierra quiere conocer y el programa

Cuadro 1.2: Objetos y sus distancias a la Tierra

Objeto	Distancia a la Tierra
Voyager 1	20.200.000.000 km
Sol	1 unidad astronómica
Próxima Centauri	4,2420 años luz
Centro de la Vía Láctea	8.500 parsecs
Galaxia Andrómeda	700.000 parsecs

mostrará esa distancia en kilómetros, unidades astronómicas, años luz y pársecs. Para ello, diseña y programa en C/C++ un algoritmo que haga las siguientes tareas:

1. Solicite al usuario qué caso desea considerar. Los valores posibles son:
   - 'c' o 'C' para una calculadora de equivalencia de distancias
   - 'd' o 'D' para saber la distancia de un objeto a la Tierra.
2. Si el usuario elige una opción distinta de 'c' o 'd' o 'C' o 'D', el programa mostrará un mensaje de error y terminará.
3. Si el usuario escoge la opción 'c' o 'C', el programa le solicitará que introduzca las unidades y, a continuación, la distancia y calculará su equivalencia en el resto de las unidades. Los valores que indican cada una de las unidades son: 'u' unidad astronómica, 'k' kilómetro, 'p' pársec, 'a' año luz. Si el usuario elige una opción distinta, el programa mostrará por pantalla un mensaje indicando que considera que las unidades son parsecs.
4. Si el usuario escoge la opción 'd', el programa le solicitará que introduzca el objeto cuya distancia a la Tierra quiere conocer y, a continuación, mostrará esa distancia en kilómetros, unidades astronómicas, años luz y pársecs. Los objetos que el usuario puede elegir son: 'v' Voyager 1, 's' Sol, 'p' Proxima Centauri, 'l' Centro de la vía láctea, 'a' Galaxia de Andrómeda. Si el usuario elige una opción distinta, el programa considera que el objeto seleccionado es la sonda Voyager 1. El punto de partida para esos cálculos son los datos incluidos en la Tabla 1.2.

Se valorará especialmente que el programa no tenga código repetido.

1.6.3. Ejemplos de ejecución

Algunos ejemplos de ejecución del programa son:

```
Dime que quieres hacer:
(c) Calculadora, (d) Distancia de un objeto a la Tierra
Opcion> z
Error en la eleccion
```

```
Dime que quieres hacer:
(c) Calculadora, (d) Distancia de un objeto a la Tierra
Opcion> d
Dime el objeto cuya distancia a la Tierra quieres conocer:
```

```
(v) Voyager 1, (s) Sol, (p) Proxima Centauri, (l) Centro de la via lactea,
(a) Galaxia de Andromeda:
Opcion> v
La distancia de la Tierra a la Voyager 1 es de:
2.02e+010 km
135.027 uas
0.000654177 parsecs
0.00213366 agnos luz
```

```
Dime que quieres hacer:
(c) Calculadora, (d) Distancia de un objeto a la Tierra
Opcion> d
Dime el objeto cuya distancia a la Tierra quieres conocer:
(v) Voyager 1, (s) Sol, (p) Proxima Centauri, (l) Centro de la via lactea,
(a) Galaxia de Andromeda:
Opcion> x
La distancia de la Tierra a la Voyager 1 es de:
2.02e+010 km
135.027 uas
0.000654177 parsecs
0.00213366 agnos luz
```

```
Dime que quieres hacer:
(c) Calculadora, (d) Distancia de un objeto a la Tierra
Opcion> c
Dime las unidades:
(u) Unidad astronomica, (k) Kilometros, (p) parsec,  (a) agno luz
Opcion> u
Dime la distancia: 3
4.488e+008 kms
1.45444e-005 parsecs
4.74376e-005 agnos luz
```

```
Dime que quieres hacer:
(c) Calculadora, (d) Distancia de un objeto a la Tierra
Opcion> c
Dime las unidades:
(u) Unidad astronomica, (k) Kilometro, (p) parsec,  (a) agno luz
Opcion> x
Dime la distancia:3
Consideramos que las unidades son parsecs
618795 uas
9.26354e+013 kms
9.78474 agnos luz
```

1.7. Proyecto 2C: Las leyes de Kepler

Una fuerza $\vec{F}$ que actúa sobre un cuerpo se dice *central* si, en todo momento, es paralela al vector de posición $\vec{r}$ de dicho cuerpo. Dos ejemplos notables de fuerzas centrales son la gravitatoria y la electrostática.

Los resultados observacionales de Kepler, posteriormente demostrados por Newton, nos dicen que las órbitas que describen los planetas, sometidos a la fuerza gravitatoria del Sol, son elípticas, ocupando la estrella uno de los focos de la elipse. Ésta es la primera ley de Kepler. La segunda asegura que el área barrida por el vector Sol–planeta es proporcional al tiempo que el planeta emplea en recorrer el arco de la elipse que limita dicha área; mientras que la tercera ley de Kepler dice que el cuadrado del periodo del planeta, el tiempo que éste necesita para recorrer toda su órbita, es proporcional al cubo del semieje mayor de la elipse (uno de los parámetros que caracterizan su forma).

Estas leyes pueden generalizarse a fuerzas centrales cuyo módulo sea inversamente proporcional al cuadrado de la distancia al centro. Las trayectorias que describen los cuerpos sometidos a estas fuerzas son elipses o hipérbolas. En esta tarea usaremos estos resultados para diseñar un programa que calcule diversos parámetros que caracterizan la dinámica de estos cuerpos.

1.7.1. Contexto del problema

Una cónica es una curva definida por la intersección de un cono circular y un plano. Su forma depende de la relación entre el ángulo de conicidad y el formado por la normal al plano y el eje del cono. No obstante, utilizaremos una definición equivalente de las elipses y las hipérbolas que es más conveniente para nuestros propósitos.

Elipses

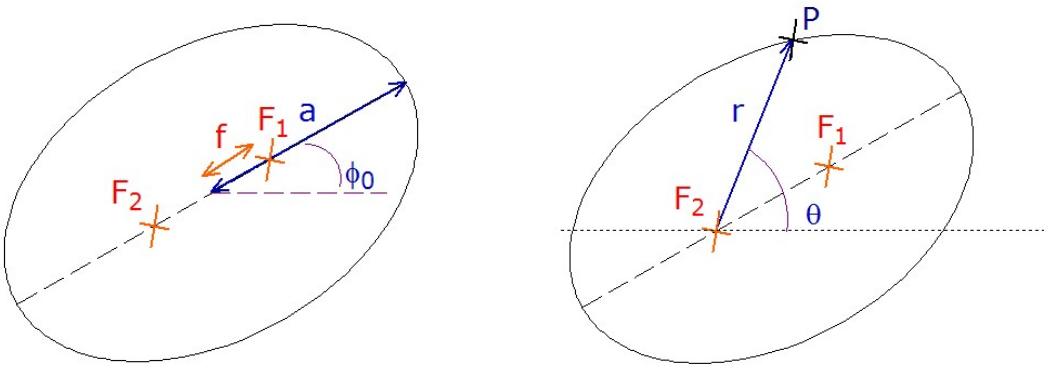
Una *elipse* es el lugar geométrico de los puntos del plano tales que la suma de sus distancias a otros dos puntos fijados del plano, llamados *focos*, es constante. Una circunferencia es un caso particular de elipse, en la que los dos focos son el mismo punto.

La forma de una elipse depende de dos parámetros: la longitud de su *semieje mayor*, a, y su *excentricidad*, e. El eje mayor de la elipse es el segmento con extremos en puntos de la elipse que pasa por los focos. Su longitud, $2a$, es el diámetro de la elipse. La excentricidad se define como el cociente f/a, donde $2f$ es la distancia entre los focos (ver la figura 1.1a). Observa que, dado que los focos se encuentran en el interior de la elipse, $0 < e < 1$; en el caso de una cicunferencia, $e = 0$.

En coordenadas polares, tomando como referencias uno de los focos y cualquier semirecta con origen en él, la ecuación de una elipse es

$$r = \frac{a \cdot (1 - e^2)}{1 - e \cdot \cos(\theta - \phi_0)} \ , \tag{1.7}$$

donde $r = \|\vec{r}\|$ es la distancia de un punto de la elipse al foco, y θ y ϕ_0 son los ángulos que forman con la semirrecta de referencia el vector de posición $\vec{r}$ de dicho punto y el eje mayor de la elipse, respectivamente.



(a) (b)

Figura 1.1: Elipse de focos F_1 y F_2. En (a) se muestran el semieje mayor de longitud a, la distancia focal (la semidistancia entre los focos, f) y el ángulo del eje respecto de la horizontal, ϕ_0. En (b) se presentan las coordenadas polares r y θ de un punto de la elipse.

Hipérbolas

De forma análoga a las elipses, una *hipérbola* se puede definir como el lugar geométrico de los puntos del plano tales que el valor absoluto de la diferencia de sus distancias a dos puntos fijados, llamados focos, es constante. Las hipérbolas tienen dos ramas, tal como se muestra en la figura 1.2a.

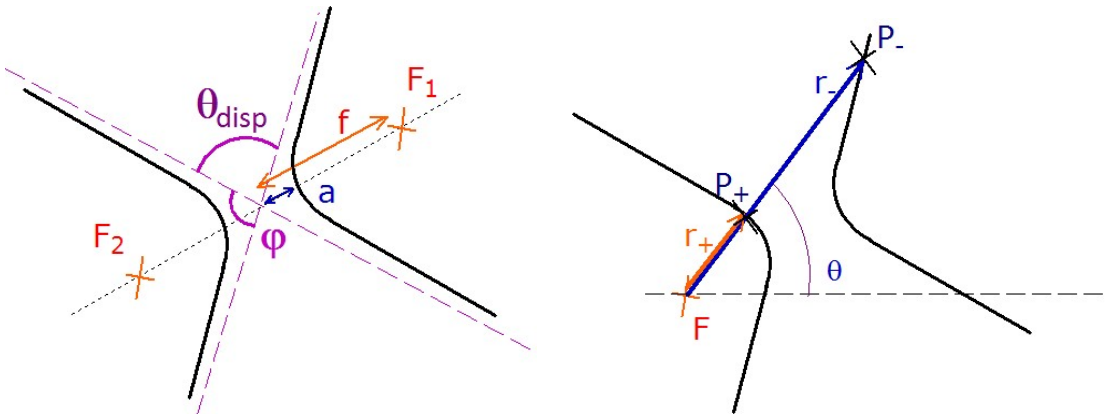
Las hipérbolas están caracterizadas por la longitud, a, de su semieje real y su excentricidad, e. El eje real es el segmento que tiene sus extremos en puntos de la hipérbola y está contenido en la recta que une los dos focos. Su longitud, $2a$, es la distancia entre las dos ramas de la hipérbola. Como en el caso de la elipse, la excentricidad e se define como f/a, donde $2f$ es la distancia entre los focos. Obsérvese que, en este caso, $e > 1$, ya que la distancia focal f es mayor que a.

La ecuación de la hipérbola en coordenadas polares, tomando como origen uno de los focos, es

$$r = \frac{a \cdot (e^2 - 1)}{e \cdot \cos(\theta - \phi_0) \pm 1} \ , \tag{1.8}$$

donde ϕ_0 es el ángulo que forma el eje real de la hipérbola con el eje de coordenadas polares. Los signos $+$ y $-$ del denominador corresponden cada uno a una rama de la hipérbola: en la figura 1.2a, a las ramas izquierda y derecha, respectivamente.

Dado que la distancia r es siempre positiva, la ecuación 1.8 sólo es válida para ángulos θ que hacen positivo el denominador. Obsérvese también que las ramas de la hipérbola tienden asintóticamente a dos rectas, cuyos ángulos con el eje de coordenadas polares son los que anulan el denominador de la ecuación. El ángulo entre estas dos rectas es $\varphi = 2 \arccos(1/e)$. Finalmente, maximizando el denominador de la ecuación 1.8, se puede calcular la distancia mínima de cada rama al foco tomado como origen de coordenadas: $a(e-1)$ para la rama izquierda y $a(e+1)$ para la rama derecha. En la ecuación 1.14 se da una interpretación física de estos dos valores, y en 1.15 del ángulo suplementario de φ.



(a) (b)

Figura 1.2: Hipérbola de focos F_1 y F_2. En (a) se muestran la semidistancia a entre las dos ramas y la distancia focal f junto con el ángulo φ formado por las asíntotas de la hipérbola. En (b) se ilustra cómo a ciertos ángulos θ le corresponden dos puntos, cada uno situado en una rama de la hipérbola.

Fuerzas centrales

Consideremos una partícula sometida a una fuerza del tipo $\vec{F} = \frac{K}{r^2}\hat{r}$, donde K es una constante, r la distancia de la partícula al origen de coordenadas y $\hat{r}$ el vector unitario paralelo al vector de posición de la partícula. Si $K > 0$ la fuerza es repulsiva; si $K < 0$, atractiva.

Dado que la fuerza es central, paralela al vector de posición, el momento angular de la partícula, $\vec{L} = \mu \cdot \vec{r} \times \vec{v}$, es constante, donde μ y $\vec{v}$ son la masa y la velocidad de la partícula, respectivamente, y $\times$ denota el producto vectorial. En particular, tanto el sentido de $\vec{L}$ como su módulo L son constantes y, por tanto, los vectores $\vec{r}$ y $\vec{v}$ se mantienen siempre dentro del plano perpendicular a $\vec{L}$ que, sin pérdida de generalidad, supondremos es XY. Bajo este supuesto, si $\vec{r} = (x, y, 0)$ y $\vec{v} = (v_x, v_y, 0)$, el módulo del momento angular está dado por la expresión siguiente

$$L = \mu \left| x \cdot v_y - y \cdot v_x \right| \ , \tag{1.9}$$

mientras que en coordenadas polares se puede calcular como

$$L = \mu r^2 \frac{d\theta}{dt} \ , \tag{1.10}$$

donde θ es el ángulo formado por el vector de posición de la partícula y la horizontal.

Además, la fuerza $\vec{F}$ es conservativa (el trabajo de la fuerza a lo largo de cualquier trayectoria depende únicamente de los puntos inicial y final). Por tanto, a partir de $\vec{F}$ podemos definir una energía potencial $V = K/r$, de modo que la energía total se conserva:

$$E = \frac{1}{2}\mu v^2 + \frac{K}{r} \ . \tag{1.11}$$

A partir de los principios de conservación del momento angular y la energía (ecuaciones 1.10 y 1.11), se pueden deducir los siguientes resultados, para el caso $L \neq 0$, que generalizan las leyes de Kepler en dos sentidos. Por un lado, son aplicables a cualquier

fuerza de tipo $\vec{F} = \frac{K}{r^2}\hat{r}$ (no sólo a la gravitatoria). Por otro, exploran todos los tipos de trayectoria posibles (elipses e hipérbolas).

1. La trayectoria de la partícula es una elipse o hipérbola con los siguientes parámetros:

$$e = \sqrt{1 + \frac{2EL^2}{\mu K^2}} \qquad \text{y} \qquad a = \frac{|K|}{2|E|} \tag{1.12}$$

   En el libro *Mecánica Clásica* [17], T.W.B. Kibble da una elegante demostración de este resultado. Dado que queda fuera del alcance de este curso no la razonaremos, limitándonos a enunciar la conclusión: si $E > 0$, entonces $e > 1$ y la trayectoria es hiperbólica; mientras que si $E < 0$, entonces $e < 1$ y la trayectoria es elíptica.

2. El vector de posición de la partícula barre áreas iguales en tiempos iguales. En particular, el área barrida en el intervalo Δt es $\frac{L}{2\mu}\Delta t$.

   En efecto, sea dA el área barrida por el vector posición de la partícula en un intervalo de tiempo pequeño, dt. El área barrida, dA, puede aproximarse por la de un triángulo de base r y altura $r\,d\theta$. Por tanto, dado que el módulo del momento angular es constante, usando la expresión 1.10 se obtiene que la siguiente también lo es

$$\frac{dA}{dt} = \frac{1}{2}r^2\frac{d\theta}{dt} = \frac{L}{2\mu}\ .$$

   En consecuencia, el área barrida en un intervalo $[t, t+\Delta t]$ es

$$\int_t^{t+\Delta t} dA = \int_t^{t+\Delta t} \frac{dA}{dt}dt = \frac{L}{2\mu}\Delta t\ .$$

   Esto es, el área barrida es independiente del instante inicial t: sólo depende del valor del tiempo transcurrido Δt.

3. Si la trayectoria es elíptica, entonces el movimiento es periódico. Más aún, el periodo T es independiente de la excentricidad:

$$T = 2\pi\sqrt{\frac{\mu}{|K|}}\,a^{3/2}\ . \tag{1.13}$$

   Este resultado es consecuencia inmediata del anterior. Para comprobarlo basta recordar que el periodo T es el tiempo que necesita la particula para recorrer completamente la elipse. Entonces, teniendo en cuenta que el área de una elipse es $\pi a^2\sqrt{1-e^2}$, tenemos que

$$\pi \cdot a^2 \cdot \sqrt{1-e^2} = \int_0^T \frac{dA}{dt}dt = \frac{L}{2\mu}T\ ,$$

   bastando usar las ecuaciones 1.12 para obtener 1.13.

Finalmente, si la trayectoria de la partícula es hiperbólica, tienen interés dos parámetros: la distancia mínima de ésta al origen, $d_{\text{mín}}$, y el ángulo de dispersión, θ_{disp}, que está formado por los vectores velocidad inicial y final de la partícula (ver la figura 1.2a). Estos valores quedan determinados por las ecuaciones siguientes.

$$d_{\text{mín}} = \begin{cases} a(e-1), \text{ si } K < 0 \text{ (potencial atractivo)} \\ a(e+1), \text{ si } K > 0 \text{ (potencial repulsivo)} \end{cases} \tag{1.14}$$

$$\theta_{disp} = \pi\ -\ 2\ \arccos(1/e) \tag{1.15}$$

1.7.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que calcule la dinámica de una partícula sometida a una fuerza central, de módulo inversamente proporcional al cuadrado de la distancia al origen.

El algoritmo empieza solicitando al usuario que identifique el tipo de fuerza:

- `'K'` o `'k'`: una fuerza genérica.
- `'G'` o `'g'`: un campo gravitatorio debido a una partícula central y fija de masa M.
- `'C'` o `'c'`: un campo electrostático debido a una partícula central y fija de carga Q.

Si el usuario introduce una opción no válida, el programa muestra un mensaje de error y termina.

En el caso `'K'` (`'k'`) el programa solicita al usuario el valor de la constante K.

En el caso `'G'` (`'g'`) se solicitará el valor de la masa central, M.

En el caso `'C'` (`'c'`) se solicitan los valores de la carga central, Q, y de la partícula, q.

A continuación, en todos los casos, el usuario deberá introducir el valor de la masa de la partícula, μ, así como su posición y velocidad iniciales, (x, y) y (v_x, v_y).

Finalmente, se mostrarán por pantalla los valores de K, r_0, v_0, E y L (constante del potencial, distancia inicial al origen, módulo de la velocidad inicial, energía total de la partícula, módulo del momento angular), y un mensaje indicando el tipo de órbita (elíptica o hiperbólica) junto con sus parámetros, a y e. Además se mostrarán el periodo de la órbita, si ésta es elíptica, o su distancia mínima al origen y el ángulo de dispersión en el caso hiperbólico.

1.7.3. Ejemplos de ejecución

El primer ejemplo es meramente académico. Con datos sencillos, fáciles de comprobar, ilustra el caso de una partícula cuya velocidad le permite vencer una fuerza atractiva y escapar al infinito:

```
Este programa simula una particula sometida a:
         Potencial generico ('K' o 'k')
         Campo gravitatorio ('G' o 'g')
         Campo electrostatico ('C' o 'c')
Introduzca opcion: k
Introduzca la constante K (en N m^2): -1
Introduzca los parametros de la particula:
         Masa (en Kg): 3
         Posicion inicial (x, y) (en m): 0 1
         Velocidad inicial (vx, vy) (en m/s): 2 0

K = -1 N m^2
Informacion sobre la particula:
Distancia inicial al origen: 1 m
Velocidad inicial: 2 m/s
Energia: 5 J
Momento angular = 6 kg m^2 /s

Trayectoria hiperbolica:
```

```
        Semieje real: 0.1 m
        Excentricidad: 11
        Distancia minima al origen: 1 m
        Angulo de dispersion: 0.18207 radianes
```

El siguiente ejemplo corresponde al sistema Sol-Tierra:

```
Este programa simula una particula sometida a:
         Potencial generico ('K' o 'k')
         Campo gravitatorio ('G' o 'g')
         Campo electrostatico ('C' o 'c')
Introduzca opcion: g
Introduzca la masa central (en Kg): 1.988E30
Introduzca los parametros de la particula:
         Masa (en Kg): 5.974E24
         Posicion inicial (x, y) (en m): 1.314E11 7.588E10
         Velocidad inicial (vx, vy) (en m/s): -1.4464E4 2.555E4

K = -7.92635e+044 N m^2
Informacion sobre la particula:
Distancia inicial al origen: 1.51736e+011 m
Velocidad inicial: 29360 m/s
Energia: -2.64896e+033 J
Momento angular = 2.6613e+040 kg m^2 /s

Trayectoria eliptica:
        Semieje mayor: 1.49612e+011 m
         Excentricidad: 0.0165768
         Periodo: 3.15666e+007  s
```

El siguiente ejemplo modeliza un par deuterio–tritio en un plasma:

```
Este programa simula una particula sometida a:
         Potencial generico ('K' o 'k')
         Campo gravitatorio ('G' o 'g')
         Campo electrostatico ('C' o 'c')
Introduzca opcion: c
Introduzca la carga central (en C): 1.602E-19
Introduzca la carga de la particula (en C): 1.602E-19
Introduzca los parametros de la particula:
         Masa (en Kg): 2E-27
         Posicion inicial (x, y) (en m): 1 1E-10
         Velocidad inicial (vx, vy) (en m/s): -2E6 0

K = 2.30651e-028 N m^2
Informacion sobre la particula:
Distancia inicial al origen: 1 m
Velocidad inicial: 2e+006 m/s
Energia: 4e-015 J
Momento angular = 4e-031 kg m^2 /s

Trayectoria hiperbolica:
        Semieje real: 2.88314e-014 m
```

```
Excentricidad: 3468.44
Distancia minima al origen: 1.00029e-010 m
Angulo de dispersion: 5.7265e-04 radianes
```

1.7.4. Ayudas

Recuerda que la fuerza gravitatoria es $-G\frac{M\mu}{r^2}\hat{r}$, donde $G = 6{,}67408 \cdot 10^{-11}$ $\mathrm{Nm^2kg^{-2}}$. Por tanto, en el caso gravitatorio $K = -GM\mu$.

La fuerza de Coulomb es $\frac{1}{4\pi\epsilon_0}\frac{Qq}{r^2}\hat{r}$, donde $\epsilon_0 = 8{,}8544 \cdot 10^{-12}$ $\mathrm{N^{-1}m^{-2}C^2}$. Por tanto, en ese caso $K = \frac{Qq}{4\pi\epsilon_0}$

En el lenguaje de programación C/C++, el valor absoluto, la raíz cuadrada y la potencia (respectivamente, $|x|, \sqrt{x}, b^x$) pueden calcularse utilizando las funciones `fabs(x)`, `sqrt(x)` y `pow(b, x)`, declaradas en el fichero de cabecera `<math.h>`. En este fichero de cabecera también se define la constante `M_PI` que contiene el valor del real π.

Cuadro 1.3: Distancia entre dos puntos

Distancia	Cálculo
Taxi	$\sum_{i=1}^{n} \|x_i - y_i\|$
Euclídea	$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
Máximo	$\max_{i=1}^{n} \|x_i - y_i\|$

1.8. Proyecto ST: Distancia entre dos puntos

1.8.1. Introducción

Para medir distancias, se han usado varias métricas como la distancia taxi, la distancia euclídea o la distancia del máximo. La forma de calcular cada una de esas distancias en dimensión $n >= 2$ para los puntos $(x_1, x_2, \ldots x_n)$ e $(y_1, y_2, \ldots y_n)$ se recoge en la Tabla 1.3.

1.8.2. Descripción de la actividad

En esta actividad trabajaremos en dimensión n=2. El programa pide al usuario las coordenadas de dos puntos, las unidades de medida usadas, que pueden ser metros, kilómetros o millas así como la distancia que se va a utilizar para medir la distancia entre esos dos puntos. La distancia puede ser: distancia taxi, distancia euclidea o distancia del máximo.

El resultado es la distancia entre los dos puntos calculada con la distancia elegida y expresada en metros.

Para ello, diseña y programa en C/C++ un algoritmo que haga las siguientes tareas:

1. Solicite al usuario las unidades que va a utilizar. Los valores posibles son:
   - ‘k’ o ‘K’ para una distancia en kilómetros
   - ‘a’ o ‘A’ para una distancia en millas
   - ‘m’ o ‘M’ para una distancia en metros.
2. Si el usuario elige una opción distinta de ‘k’ o ‘K’ o ‘a’ o ‘A’ o ‘m’ o ‘M’, muestra un mensaje al usuario indicando que no es una opción válida y termina.
3. En otro caso, solicita la distancia que va a usar Los valores posibles son:
   - ‘t’ para la distancia taxi
   - ‘e’ para la distancia euclídea
   - ‘m’ para la distancia del máximo
4. Si el usuario elige una opción distinta de ‘t’ o ‘e’ o ‘m’, muestra un mensaje al usuario indicando que no es una opción válida y termina.
5. En otro caso, pide al usuario que introduzca las coordenadas de dos puntos y, a continuación, calcula la distancia entre esos dos puntos, la convierte a metros y la muestra en pantalla.

Se valorará especialmente que el programa no tenga código repetido.

1.8.3. Ejemplos de ejecución

Algunos ejemplos de ejecución del programa son:

```
Calculadora de distancias
Dame las unidades: z
Opción no válida
```

```
Calculadora de distancias
Dame las unidades: m
Dame la metrica: z
Opción no válida
```

```
Calculadora de distancias
Dame las unidades: m
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 3 3
La distancia es 2 metros.
```

1.9. El oscilador armónico

El oscilador armónico es un modelo ubicuo en Física: modeliza el comportamiento de muchos sistemas en torno a su estado de equilibrio. En esta actividad realizaremos algunos cálculos sencillos utilizando únicamente el conocimiento adquirido en Bachillerato sobre este modelo.

1.9.1. Introducción

La ecuación del oscilador es:

$$m\,\ddot{x} = -k\,x \tag{1.16}$$

donde $\ddot{x}$ es la aceleración, m la masa y k la constante de recuperación. Haciendo $\alpha = k/m$, cuya dimensión es $tiempo^{-2}$, esta ecuación se transforma en:

$$\ddot{x} = -\alpha\,x \tag{1.17}$$

Conocemos la solución de la ecuación 1.16: puesto que la fuerza es conservativa, la energía del oscilador es constante y sólo depende de su posición y velocidad iniciales $(x_0, \dot{x}_0)$:

$$E = \frac{1}{2}m\,\dot{x}^2 + \frac{1}{2}\,k\,x^2 \tag{1.18}$$

Por su parte, x es una función sinusoidal. Su frecuencia de oscilación y periodo son, respectivamente,

$$\omega = \sqrt{\frac{k}{m}} = \alpha^{1/2} \;\;,\;\; T = \frac{2\pi}{\omega} = 2\pi\alpha^{-1/2} \tag{1.19}$$

1.9.2. Descripción de la actividad

El programa empieza pidiendo al usuario que introduzca la posición y velocidad iniciales (x_0, $\dot{x}_0$), así como la masa m y la constante de recuperación k. Para cada magnitud, el usuario introducirá un real seguido de un carácter, que indica en qué unidades se mide:

- La posición, x_0, en metros (`'m'`), decímetros (`'d'`) o centímetros (`'c'`).
- La velocidad inicial, $\dot{x}_0$, en metros por segundo (`'m'`), decímetros por segundo (`'d'`) o centímetros por segundo (`'c'`).
- La masa, m, en kilogramos (`'k'` o `'K'`) o en gramos (`'g'` o `'G'`).
- La constante de recuperación, k, en Newtons/metro (`'N'`), dinas/metro (`'d'`) o kilopondios/metro (`'k'`). Recordemos que 1 Newton = 10^5 dinas y 1 kilopondio = 9,8 Newton.

Si la introducción de datos es correcta, el programa calculará el parámetro $\alpha = k/m$, utilizando s^{-2} como unidad de medida, así como la energía en Julios, y la frecuencia y el periodo en s^{-1} y s, respectivamente.

Si el usuario introduce una unidad de medida no válida, el programa mostrará un mensaje advirtiéndolo y terminará inmediatamente. Respecto de los valores numéricos, la masa y la constante de recuperación k deben ser mayores que 0. El programa continuará solicitando datos de entrada aún cuando los valores para estas dos magnitudes no sean válidos, pero se comprobará que ambas son positivas antes de calcular los valores de salida, escribiendo un mensaje de error en caso contrario.

1.9.3. Ejemplos de ejecución

```
Posicion inicial: 10 m
Velocidad inicial: 5 m
Masa: 1 k
Constante de recuperacion: 2 N

alpha = 2 s^-2
w = 1.41421 s^-1
T = 4.44288 s
E = 112.5 J
```

```
Posicion inicial: 5 k
Unidad de medida no valida para la posicion
```

```
Posicion inicial: 10 d
Velocidad inicial: 5 c
Masa: -5 g
Constante de recuperacion: 2 k

La masa debe tener un valor positivo
```

1.9.4. Ayudas

El fichero de cabecera `<math.h>` (o `<cmath>`, si usamos el compilador de C++) contiene la declaración de las funciones matemáticas usuales; en particular de `sqrt` y `pow`, para el cálculo de la raíz cuadrada y la potencia, respectivamente. Concretamente, si `x`, `b` y `e` son expresiones de tipo `double`, entonces `sqrt(x)` es una expresión de tipo `double` cuyo valor es la raíz cuadrada de `x`, mientras que la expresión, también de tipo `double`, `pow(b,e)` tiene el valor b^e. Sin embargo, no es obligatorio que este fichero de cabecera contenga una definición para el número π; si la hay suele llamarse `M_PI`. En todo caso, siempre podemos definir una constante con el valor de π en nuestros programas, por ejemplo usando el arcocoseno de -1 (`acos(-1)`) o multiplicando por 4 el arcotangente de 1 (`4*atan(1)`).

La razón por la que puede o no existir predefinida una constante con el valor de π es que el último estándar de los lenguajes C y C++ no lo exige como una obligación para los implementadores. Esto es, cada constructor de un compilador para estos lenguajes es libre de añadirla o no.

1.9.5. Redireccionamiento de flujos

Teclear todos los datos cada vez que se ejecuta un programa puede ser tedioso, y una potencial fuente de errores. Si es necesario introducir muchas veces los mismos datos de entrada, como ocurre durante la fase de pruebas del código, es muy fácil equivocarnos. Una posible solución es redirigir los flujos de entrada y salida.

Un flujo (*stream*) es cualquier fuente de entrada o destino de la salida de un programa. Por defecto, los flujos de entrada y salida estándares son el teclado y la pantalla. Además, un tercer flujo, el error estándar también se escribe por defecto en la pantalla. Pero es posible redireccionarlos, haciendo, por ejemplo, que un programa lea los datos de un

fichero y escriba sus resultados en otro, mientras que los mensajes de error aún podrían escribirse en un tercero. Para ello sigue los pasos siguientes, que se ilustran en la Figura 1.4 (suponemos que hemos construido y compilado el programa, que su ejecutable se llama `01Tarea.exe`, y que se encuentra en la carpeta `D:\tareas`):

1. Crea un fichero de texto (usando, por ejemplo, el bloc de notas) y escribe allí los datos de entrada del programa (en nuestro ejemplo, el fichero se llama `datosEntrada.txt`), tal como se muestra en la figura 1.3.
2. Lanza el intérprete de comandos (`cmd` en `Windows`) .
3. Navega hasta la carpeta donde se almacena el ejecutable: `cd: D:\tareas`. Si es necesario, primero cambia de unidad escribiendo `D:`
4. Ejecuta `01Tarea.exe <datosEntrada.txt`. Si el ejecutable y/o el fichero con los datos de entrada se encuentran en un directorio distinto del que se considera en `cmd`, es preciso indicar la ruta completa de los ficheros.

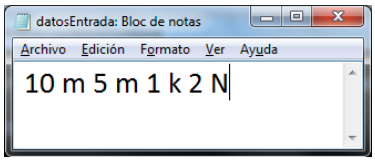


Figura 1.3: Ejemplo de fichero con datos de entrada

Del mismo modo, puedes redirigir los flujos de salida y error para que el programa escriba en un fichero, en vez de en pantalla:

```
01Tarea.exe <datosEntrada.txt >salida.txt 2>error.txt
```

Para comprobar la diferencia entre los canales de salida y error estándar deberás modificar el código de tu programa, escribiendo los mensajes de error en el objeto `cerr`. Más explícitamente, la sentencia

```
cerr << expresion;
```

escribe en el canal de error el valor de la expresión.

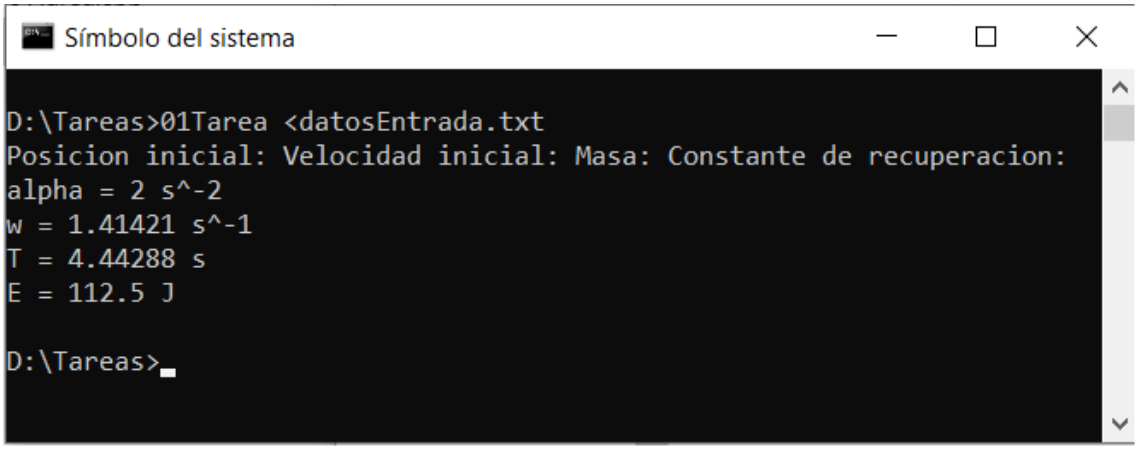


Figura 1.4: Ejecución desde el intérprete de comandos, leyendo los datos de un fichero

Capítulo 2

Bucles

2.1. Caída libre

2.1.1. Introducción

En esta tarea calcularemos la dinámica de un cuerpo en caída libre.

Sobre el cuerpo actúan dos fuerzas: la gravitatoria ($-mg\ \widehat{k}$) y el rozamiento del aire ($bv^2\ \widehat{k}$), donde b es el coeficiente de rozamiento, v la velocidad y $\widehat{k}$ la normal al suelo. La fuerza resultante está dirigida hacia abajo; el módulo de la aceleración es:

$$a = \frac{dv}{dt} = g\ -\ b/m \cdot v^2 \tag{2.1}$$

Suponemos que inicialmente la velocidad vertical del cuerpo es cero. Se acelera, y conforme aumenta su velocidad lo hace la fuerza de rozamiento, hasta que finalmente la fuerza de rozamiento y la gravitatoria se equilibran. A partir de ese momento, su velocidad se mantiene constante.

2.1.2. Descripción de la tarea

Diseña e implementa un algoritmo para resolver el siguiente problema: Un paracaidista se lanza desde un avión a una altura h_0. Cuando está a mitad de altura, abre el paracaídas. Dadas la altura inicial (h_0) y la masa del paracaidista (m), introducidas por teclado, calcular:

- El tiempo que tarda en abrir el paracaídas, y qué velocidad lleva en el momento de abrirlo.
- El tiempo que le cuesta llegar a tierra, y qué velocidad lleva en ese momento.
- El trabajo total hecho por la fuerza gravitatoria y por la resultante de las fuerzas (gravitatoria + rozamiento) en esos tiempos.

2.1.3. Ejemplos de ejecución

```
Introduzca la altura inicial (en metros): 300
Introduzca la masa (en kg): 100

Altura = 150 m. :
         t = 5.88424s, v = -45.4788 m/s, Wg = 147000 J,  Wt = 103416 J
Altura = 0 m. :
         t = 11.316s,  v = -23.831 m/s,  Wg = 294000 J,  Wt = 28395.8 J
```

2.1.4. Ayudas

Para calcular la velocidad v en un momento dado, podemos integrar la aceleración: $v(t) = v_0 + \int_0^t a(t)dt$. A su vez, al integrar la velocidad obtenemos el espacio recorrido: $y(t) = y_0 + \int_0^t v(t)dt$. El trabajo de una fuerza viene dado por la expresión: $W = \int_0^t F(y)dy$. La ecuación 2.1 puede resolverse de forma analítica, pero por el momento no tenemos conocimientos suficientes para ello. Por ello sugerimos recurrir a técnicas de integración numérica (en particular, fórmulas abiertas de Newton-Cotes), tal como se describen en el apéndice C.

Para ello discretizamos el tiempo: consideramos instantes $t_n = n \cdot \Delta t$ y descomponemos el tiempo en intervalos $[t_n, t_{n+1}]$ de longitud Δt. En cada intervalo $[t_n, t_{n+1}]$ aproximamos las integrales de v y a por la superficie de rectángulos, de base Δt y alturas respectivamente $v(t_n)$ y $a(t_n)$. De este modo calculamos:

$$y(t_{n+1}) = y(t_n) + v(t_n) \cdot \Delta t \ , \tag{2.2}$$

$$W(t_{n+1}) = W(t_n) + m \cdot a(t_n) \cdot v(t_n) \cdot \Delta t \ , \tag{2.3}$$

$$v(t_{n+1}) = v(t_n) + a(t_n) \cdot \Delta t \ . \tag{2.4}$$

El trabajo de una fuerza viene dado por la expresión: $W = \int_0^t F(y)dy$. Para intervalos de tiempo muy pequeños, en que el módulo de la fuerza se puede suponer prácticamente constante, la anterior integral se puede discretizar de modo análogo a los ejemplos vistos previamente.

Valores de las constantes

Para que la aproximación discreta sea válida, toma un intervalo $\Delta = 10^{-3}$ s.
Puedes utilizar los siguientes valores para las constantes:
$g = 9{,}8\ \text{ms}^{-1}$
b_0 (coeficiente de rozamiento con el paracaídas cerrado) = 0,25 kg/m
b_p (coeficiente de rozamiento con el paracaídas abierto) = 1,75 kg/m

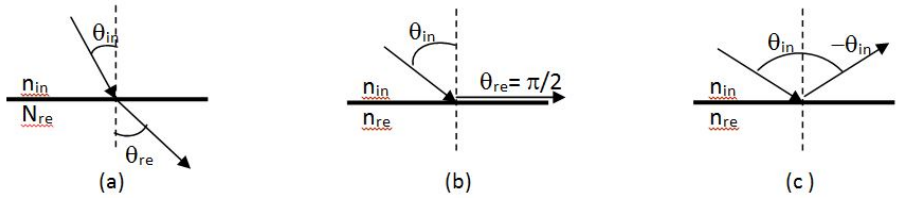


Figura 2.1: Ángulo de incidencia menor (a) igual (b) o superior (c) al ángulo crítico

2.2. Refracción de luz: espejismos

2.2.1. Introducción

En esta tarea estudiaremos la trayectoria de un rayo de luz que atraviesa una zona de la atmósfera con gradiente de temperatura. En particular, supondremos condiciones de inversión térmica y un gradiente vertical constante (la temperatura aumenta linealmente con la altura). En particular, estudiaremos bajo qué condiciones un rayo que parte de la superficie de la Tierra se curva, regresando a la superficie.

Ley de Snell: refracción y reflexión total

Sean dos medios, con índices de refracción n_{in} y n_{re} respectivamente. Cuando un rayo de luz procedente del primero incide sobre la superficie de separación entre ambos cambia de dirección, según la ley de Snell:

$$n_{in} \sin \theta_{in} = n_{re} \sin \theta_{in} \tag{2.5}$$

donde θ_{in} y θ_{re} son respectivamente los ángulos de incidencia y de refracción, y están tomados respecto de la normal a la superficie de separación entre los dos medios.

Si ocurre que sin $\theta_{in} = (n_{re}/n_{in})$, entonces $\sin \theta_{re} = 1$ y, por tanto $\theta_{re} = (\pi/2)$: el rayo sale refractado paralelo a la superficie (Figura 2.1(b)). En este caso, $\theta_{in} = \theta_c$ es llamado ángulo crítico. Para ángulos de incidencia $\theta_{in} > \theta_c$ no hay refracción ($\sin \theta_{re}$ no puede ser mayor que 1), y se produce una reflexión total y el rayo sigue una trayectoria de ángulo $-\theta_{in}$ (Figura 2.1(c)).

Espejismos

Cuando un rayo de luz atraviesa capas de aire con distintas temperaturas, y por tanto con distintos índices de refracción, sufre sucesivas refracciones y se curva. Eventualmente puede incidir sobre la superficie de separación entre dos capas con un ángulo mayor al crítico. Esta es la explicación de por qué en los días calurosos el asfalto parece encharcado: los rayos procedentes de un objeto (el cielo, un árbol...) se curvan, describiendo una trayectoria convexa (Figura 2.2(a)). La imagen virtual del objeto (la que se obtiene tomando los rayos tal como inciden en nuestro dispositivo óptico y prolongándolos en línea recta) se forma bajo el suelo: el efecto es el mismo que si el objeto se reflejara en un charco. Cuando se da el fenómeno de inversión térmica (el suelo está más frío que el aire por encima de él), los rayos procedentes de objetos a ras de suelo se curvan describiendo una trayectoria cóncava. En este caso se habla de espejismo superior. La imagen virtual del objeto se forma a una cierta altura sobre el suelo. Si se dan las condiciones adecuadas, es

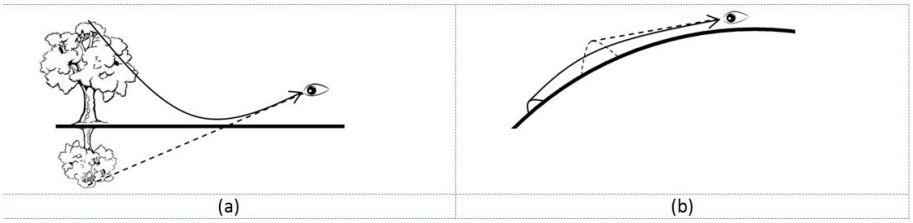


Figura 2.2: Espejismos inferior y superior

posible ver objetos que están más allá de la línea del horizonte (como se muestra en la Figura 2.2(b)).

Nuestro modelo

En esta tarea supondremos una situación de inversión térmica: la temperatura es mínima a ras de suelo, y se incrementa conforme ganamos en altura.

Supondremos que el gradiente de temperatura es constante y que a partir de una altura $h_{máx}$ la temperatura permanece constante:

$$T(y) = \begin{cases} T_0 + y \cdot (T_{máx} - T_0)/h_{máx}, \text{ si } y \leq h_{máx}, \\ T_{máx}, \text{ si } y \geq h_{máx}. \end{cases} \tag{2.6}$$

El índice de refracción del aire a temperatura T, dada en grados centígrados, es:

$$n(y) = 1{,}0002926 - 10^{-6} \cdot T(y). \tag{2.7}$$

2.2.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que tenga el siguiente comportamiento: Empieza leyendo de teclado los parámetros de nuestro modelo (altura máxima $h_{máx}$, temperatura T_0 del aire en el suelo -altura 0-, y temperatura máxima $T_{máx}$).

El algoritmo calcula el ángulo límite a partir del cual el rayo se refleja y vuelve a tierra: $n(0) \cdot \text{seno}(\theta_{lim}) = n(h_{máx})$.

A continuación el programa pedirá al usuario que introduzca el ángulo θ_0 con que parte del suelo un rayo de luz. Si el rayo se refleja y vuelve a tierra ($\theta_0 \geq \theta_{lim}$), el programa muestra por pantalla un mensaje indicando la altura máxima alcanzada por el rayo antes de la reflexión total. En caso contrario, el programa muestra por pantalla un mensaje indicando la distancia horizontal recorrida por el rayo al llegar a la altura máxima.

Tras esto el programa preguntará al usuario si desea conocer la trayectoria de un nuevo rayo: si la respuesta es 'S' o 's' se volverá a solicitar un nuevo ángulo, en caso contrario el programa terminará.

2.2.3. Ejemplos de ejecución

```
Introduzca datos:
   T0: -5
   TMax: 10
   Altura maxima: 20
```

```
Angulo maximo: 1.56532 radianes


Introduzca un angulo (en radianes): 1.565
        El rayo alcanza la altura maxima tras recorrer 5197.73 m.


Desea otro angulo?:  s
Introduzca un angulo (en radianes): 1.5653
        El rayo alcanza la altura maxima tras recorrer 6706.19 m.


Desea otro angulo?:  s
Introduzca un angulo (en radianes): 1.5653
        El rayo alcanza la altura maxima tras recorrer 6706.19 m.

Desea otro angulo?:  s
Introduzca un angulo (en radianes): 1.56531
        El rayo alcanza la altura maxima tras recorrer 6876.2 m.


Desea otro angulo?:  s
Introduzca un angulo (en radianes): 1.56532
        El rayo se refleja a una altura yMax = 19.9934 m.

Desea otro angulo?:  s
Introduzca un angulo (en radianes): 1.57
        El rayo se refleja a una altura yMax = 0.422758 m.

Desea otro angulo?:  n
```

2.2.4. Ayudas

En caso de reflexión total, la altura máxima alcanzada por el rayo es el valor $y_{máx}$ para el cual $n(0) \cdot \text{seno}\,\theta_0 = n(y_{máx})$. Los valores de $n(0)$, $n(y)$ y $T(y)$ se calculan mediante las expresiones 2.6 y 2.7, mientras que θ_0 es el ángulo inicial, tecleado por el usuario.

Cálculo de la distancia horizontal recorrida

La distancia horizontal viene dada por la integral:

$$x = \int_{y=0}^{h_{máx}} \tan\,\theta(y)\;\; dy, \tag{2.8}$$

donde el ángulo $\theta(y)$ se obtiene a partir de la ley de Snell:

$$\theta(y) = \text{arcoseno}\,\left(\frac{n(0)\cdot \text{seno}\,\theta(0)}{n(y)}\right), \tag{2.9}$$

y el índice de refracción a altura y, $n(y)$, se calcula a partir de las ecuaciones 2.6 y 2.7. Se trata, pues, de calcular la integral de una función ($\tan\,\theta(y)$), cuyo valor es conocido en

todo el intervalo de integración. Puede calcularse numéricamente, utilizando una fórmula cerrada de Newton-Cotes , tal como se explica en el Apéndice C(expresión C.2):

$$x = \int_{y=0}^{h_{máx}} \tan\theta(y)\ \ dy \rightarrow \sum_{n=0} \frac{\tan\theta(\Delta\cdot(n+1)) + \tan\theta(\Delta\cdot n)}{2}\cdot\Delta\ , \tag{2.10}$$

donde $\Delta\cdot(n+1) \leq h_{máx}$. Puedes tomar intervalos $\Delta = 10^{-3}$ m.

2.3. Péndulo simple

2.3.1. Introducción

El objetivo de esta tarea calcular el periodo de oscilación T de un péndulo simple dados su longitud (l) y ángulo inicial (Θ_0). Además, nos interesa comparar el periodo T obtenido con el que resulta en el caso de pequeñas oscilaciones $T_0 = 2\pi\sqrt{l/g}$, siendo $g = 9{,}8\ \mathrm{m/s^2}$ la intensidad del campo gravitatorio.

El péndulo simple

Sea un péndulo que inicialmente forma un ángulo Θ_0 con la vertical. Sometido a la fuerza de la gravedad, experimenta una serie de oscilaciones entre valores extremos de Θ (ver figura 2.3).

Si suponemos despreciable el rozamiento con el aire, y aplicando el principio de conservación de la energía, sabemos que el movimiento es periódico.

La dinámica de un péndulo de longitud l viene dada por la ecuación

$$\alpha = \frac{\mathrm{d}^2\Theta}{\mathrm{d}t^2} = -a\sin(\Theta)\ , \tag{2.11}$$

donde $a = g/l$ es el cociente entre la intensidad del campo gravitatorio y la longitud del péndulo.

Para simular la dinámica del péndulo realizaremos una doble integración: integraremos la aceleración angular $\left(\alpha = \frac{\mathrm{d}^2\Theta}{\mathrm{d}t^2}\right)$ para obtener la velocidad angular $\left(\omega = \frac{\mathrm{d}\Theta}{\mathrm{d}t}\right)$, e integraremos nuevamente la velocidad angular para obtener Θ.

2.3.2. Descripción de la tarea

La tarea consiste en realizar un programa que, tras solicitar por teclado los valores de la longitud, l, y del ángulo inicial Θ_0 de un péndulo simple (expresados, respectivamente, en metros y radianes), muestre por pantalla el valor del periodo (T) en segundos y el cociente T/T_0, siendo $T_0 = 2\pi\sqrt{l/g}$ el periodo que resulta en el caso de pequeñas oscilaciones. Además el programa debe mostrar las incertidumbres de cada uno de los dos cálculos.

Por simplicidad, suponemos que la velocidad inicial es cero ($\omega_0 = 0$). Para calcular el periodo del péndulo seguiremos su dinámica hasta que el ángulo de oscilación, Θ, alcance un valor extremo: el tiempo transcurrido hasta ese momento será la mitad del periodo.

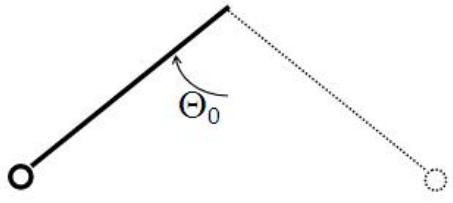


Figura 2.3: Péndulo que inicialmente forma un ángulo Θ_0 con la vertical.

2.3.3. Ejemplos de ejecución

```
Introduce la longitud del pendulo (en metros):  10
Introduce el angulo inicial (en radianes): 0.01

El periodo es 6.34709 +- 0.000101015 segundos
T/T0 = 1.00002 +- 1.59155e-005
```

```
Introduce la longitud del pendulo (en metros):  10
Introduce el angulo inicial (en radianes): -2

El periodo es 8.43508 +- 0.000101015 segundos
T/T0 = 1.32899 +- 1.59155e-005
```

2.3.4. Ayudas

Integración numérica de la ecuación 2.11

La ecuación diferencial 2.11 es no lineal, y su solución analítica es muy complicada. Por ello, la resolveremos numéricamente. Aproximaremos las integrales de la aceleración y la velocidad por sumatorios finitos (en particular, fórmulas abiertas de Newton-Cotes), tal como se describen en el apéndice C.

Observa que para calcular la velocidad angular ω en un momento dado, podemos integrar la aceleración angular: $\omega(t) = \omega_0 + \int_0^t \alpha(t)dt$. A su vez, al integrar la velocidad obtenemos el espacio recorrido: $\Theta(t) = \Theta_0 + \int_0^t \omega(t)dt$.

Para ello discretizamos el tiempo: consideramos instantes $t_n = n \cdot \Delta t$ y descomponemos el tiempo en intervalos $[t_n, t_{n+1}]$ de longitud Δt. En cada intervalo $[t_n, t_{n+1}]$ aproximamos las integrales de ω y α por la superficie de rectángulos, de base Δt y alturas respectivamente $\omega(t_n)$ y $\alpha(t_n)$. De este modo calculamos:

$$\Theta(t_{n+1}) = \Theta(t_n) + \omega(t_n) \cdot \Delta t \ , \tag{2.12}$$

$$\omega(t_{n+1}) = \omega(t_n) + \alpha(t_n) \cdot \Delta t \ . \tag{2.13}$$

Para que la aproximación del cálculo de la integral como un sumatorio sea aceptable, los intervalos de tiempo deben ser lo suficientemente pequeños en relación con el tiempo característico de evolución del sistema. Te sugerimos el valor $\Delta t = 10^{-4}\sqrt{l/g}$.

Extremos del intervalo de integración

La dinámica del péndulo empieza en $t = 0$. Nos detendremos cuando Θ alcance su primer valor extremo (su primer máximo o su primer mínimo).

Los extremos de Θ ocurren cuando $\omega = \frac{\mathrm{d}\Theta}{\mathrm{d}t} = 0$. Debido a la discretización del tiempo y a los límites en la precisión de los cálculos, es prácticamente imposible que se dé el caso $\omega_n = 0$.

Pero si en un intervalo ocurre que ω_{n-1} y ω_n tienen distinto signo, dado que ω debe ser continua, por el teorema de Bolzano (ver el apéndice D) podemos deducir que ω se hace cero en algún punto del intervalo.

En resumen, identificaremos que ha alcanzado un valor extremo, y finalizaremos la integración, cuando se dé alguno de los siguientes casos:

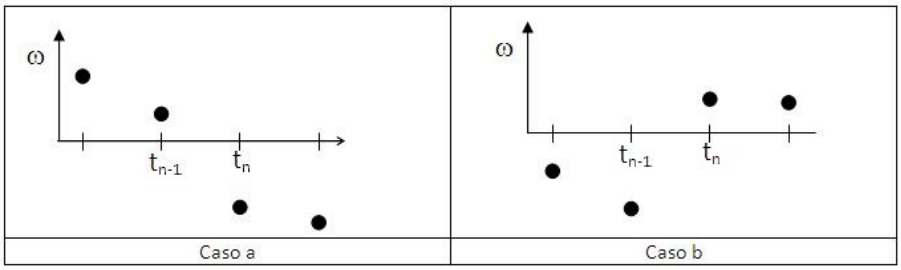


Figura 2.4: Posibles formas de que se anule la velocidad angular.

Cuadro 2.1: Periodo de oscilación 'exacto' del péndulo simple para diversas amplitudes de oscilación.

Θ	T/T_0	Θ	T/T_0	Θ	T/T_0	Θ	T/T_0
0°	1,0000	20°	1,0076	40°	1,0313	100°	1,2322
2°	1,0001	22°	1,0093	44°	1,0381	110°	1,2953
4°	1,0003	24°	1,0111	48°	1,0457	120°	1,3729
6°	1,0007	26°	1,0131	52°	1,0541	130°	1,4693
8°	1,0012	28°	1,0152	56°	1,0632	140°	1,5945
10°	1,0019	30°	1,0174	60°	1,0732	150°	1,7737
12°	1,0027	32°	1,0199	70°	1,1021	160°	2,0075
14°	1,0038	34°	1,0225	80°	1,1375	170°	2,4393
16°	1,0049	36°	1,0253	90°	1,1804	180°	∞
18°	1,0062	38°	1,0282				

- $\omega_{n-1} < 0$ y $\omega_n > 0$ (ver figura 2.4, caso a).
- $\omega_{n-1} > 0$ y $\omega_n < 0$ (ver figura 2.4, caso b).
- $\omega_n = 0$ (caso extremadamente improbable a menos que $\Theta_0 \in \{0, \pm\pi\}$).

En los tres casos, el extremo de Θ o curre en $t_n - \Delta t/2$ con una incerticumbre $\Delta t/2$. Por tanto, $T = 2t_n - \Delta t$, su incertidumbre es $\sigma_T = \Delta t$ y la incertidumbre de T/T_0 es σ_T/T_0.

Dado que necesitamos comparar los valores de ω en dos momentos distintos, es necesario:

- Que el programa ejecute, como mínimo, una iteración.
- Guardar, para cada intervalo, el valor de la velocidad del intervalo anterior (en la iteración n-ésima, además de calcular ω_n, debemos recordar el valor de ω_{n-1}).

Validación del programa

Para validar el programa, es facil encontrar en la web tablas con los valores del periodo para distintos ángulos iniciales (ver, por ejemplo, [26] de la que hemos extraído la tabla 2.1).

2.3.5. Para saber más...

Tienes una excelente clase magistral sobre el péndulo (y más cosas) en [35]

2.4. El arco iris

2.4.1. introducción

En esta práctica resolveremos el siguiente problema: si observamos el arco iris desde un ángulo determinado ¿qué color se ve?

El arco iris se produce por una combinación de refracciones y reflexiones de la luz (ver figura 2.5). Consideremos un haz de rayos horizontales que incide sobre una gota de agua. Parte de ellos se refracta al entrar en la gota (1), se refleja en la superficie interior de la misma (2) y vuelve a refractarse al salir de la gota (3), formando un ángulo Θ con la horizontal. Ese es el ángulo con el que el rayo llega al ojo del observador.

Este ángulo Θ de salida de los rayos de luz varía dentro de un cierto intervalo $\Theta \in [0, \Theta_{\text{máx}}]$, donde $\Theta_{\text{máx}}$, llamado *ángulo de desviación máxima*, está dado por

$$\Theta_{\text{máx}} = 4 \arcsin\left(\frac{\sin \alpha_M}{n}\right) - 2\alpha_M \ , \tag{2.14}$$

siendo n el índice de refracción del agua y

$$\alpha_M = \arcsin\sqrt{\frac{4-n^2}{3}} \ . \tag{2.15}$$

El índice de refracción del agua varía con la longitud de onda de la luz. Podemos aproximar esa dependencia mediante la siguiente expresión [11]:

$$n(\lambda) = 1{,}43 - 3{,}78 \cdot 10^{-4}\lambda + 5{,}08 \cdot 10^{-7}\lambda^2 - 2{,}41 \cdot 10^{-10}\lambda^3 \tag{2.16}$$

donde la longitud de onda, λ, viene expresada en nanómetros. En consecuencia, distintos colores tienen índices de refracción distintos y, por tanto, distintos ángulos de desviación máxima $\Theta_{\text{máx}}(\lambda)$. Por esta razón, y debido a que la luz se concentra en torno a este ángulo, correspondiente a su longitud de onda, los colores se separan al salir de la gota, dando lugar al arco iris. Como referencia sobre el tema, puedes consultar el libro "Física para la Ciencia y la Tecnología", de P. Mosca y G. Tipler [33].

Dada una longitud de onda, λ, es sencillo calcular su ángulo de desviación máxima: se calcula el índice de refracción $n(\lambda)$, a partir de él $\alpha_M(\lambda)$ y, finalmente, $\Theta_{\text{máx}}(\lambda)$, aplicando las fórmulas 2.16, 2.15 y 2.14, respectivamente. A modo de ejemplo, calculamos los valores de $n(\lambda), \alpha_M(\lambda)$ y $\Theta_{\text{máx}}(\lambda)$ para los extremos del intervalo de longitudes de onda de la luz visible: λ_{uv} (luz ultravioleta) y λ_{ir} (luz infrarroja).

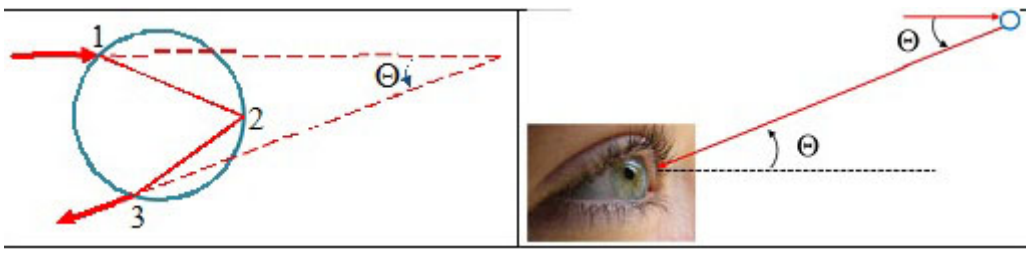


Figura 2.5: Trayectoria de un rayo de luz. El ángulo de salida del rayo, Θ, es el ángulo con el que llega al ojo del observador.

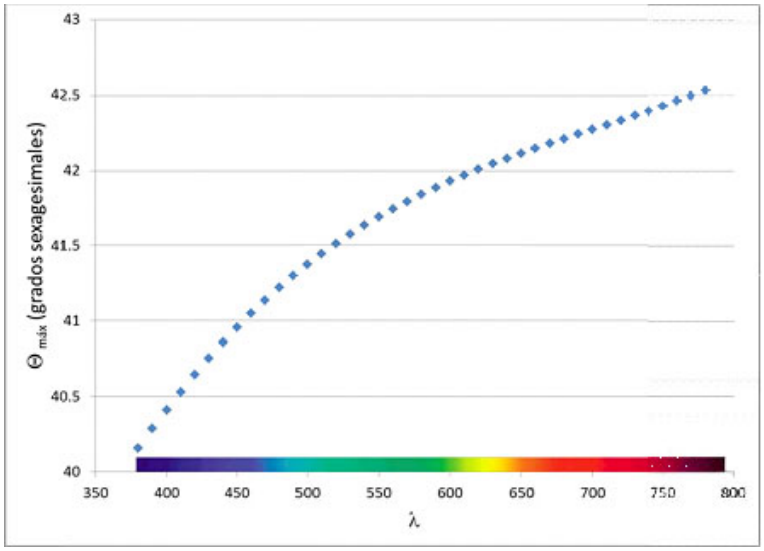


Figura 2.6: Ángulo de desviación máxima, $\Theta_{\text{máx}}(\lambda)$, para las longitudes de onda del espectro visible. En el eje de abcisas se muestra el color correspondiente a cada longitud de onda λ.

λ	$n(\lambda)$	$\alpha_M(\lambda)$ (radianes)	$\Theta_{\text{máx}}(\lambda)$ (gr. sexagesimales)
$\lambda_{\text{uv}} = 380$ nm	1,3465	1,0233	$\Theta_{\text{uv}} = 40{,}156°$
$\lambda_{\text{ir}} = 780$ nm	1,3299	1,0401	$\Theta_{\text{ir}} = 42{,}537°$

En la figura 2.6 se muestra el valor de $\Theta_{\text{máx}}(\lambda)$ para algunas longitudes de onda del espctro visible. Observa que, por ejemplo, en el intervalo $[42{,}17°, \Theta_{\text{ir}}]$ sólo se obtiene luz roja ($42{,}17°$ es mayor que el ángulo de desviación máxima de todos los colores, salvo el rojo): ese es el color que observaremos si miramos en esa dirección. Si miramos con un ángulo de $40{,}5°$ nuestro ojo recibe luz compuesta por todos los colores, pero la proporción del violeta es mayor (recuerda que la luz de un color se concentra en torno a su ángulo de desviación máxima $\Theta_{\text{máx}}$). Por encima de Θ_{ir} no hay luz (ese ángulo es mayor que $\Theta_{\text{máx}}$ para todas las longitudes de onda del espectro visible). Para ángulos menores que Θ_{uv} (por tanto, menores que $\Theta_{\text{máx}}(\lambda)$ para todas las longitudes de onda del espectro visible) obtenemos una combinación de todos los colores sin que destaque ninguno: veremos luz blanca tenue. Por ello el interior del arco iris es más luminoso que el exterior.

Así, el problema que planteábamos al principio se reduce a recorrer el camino inverso: **dado un ángulo Θ, ¿cuál es la longitud de onda λ para la cual $\Theta = \Theta_{\text{máx}}(\lambda)$?** En otras palabras, se trata de invertir la función $\Theta_{\text{máx}}(\lambda)$. Observa que, en el intervalo considerado, $\Theta_{\text{máx}}$ es una función de λ estrictamente creciente, y por tanto se puede invertir.

2.4.2. Descripción de la tarea

El programa empieza solicitando al usuario la precisión, Δ, con la que quiere obtener sus resultados. A continuación, el programa solicita al usuario que teclee el valor de un ángulo, Θ, en grados sexagesimales.

- Si $\Theta < \Theta_{\text{uv}} = 40{,}156°$ (menor que $\Theta_{\text{máx}}$ para todas las longitudes de onda del

espectro visible), en esa dirección se obtiene una combinación de todos los colores sin que destaque ninguno. El programa escribe en la pantalla "luz blanca tenue".

- Si $\Theta > \Theta_{\text{ir}} = 42{,}537°$ (mayor que $\Theta_{\text{máx}}$ para todas las longitudes de onda del espectro visible), en esa dirección no se obtienen rayos. El programa escribe en pantalla "no hay luz".
- Si $\Theta \in [\Theta_{\text{uv}}, \Theta_{\text{ir}}]$, sabemos que $\lambda \in [\lambda_{\text{uv}}, \lambda_{\text{ir}}]$. Más aún, como $\Theta_{\text{máx}}(\lambda)$ es una función estrictamente creciente en el intervalo $[\lambda_{\text{uv}}, \lambda_{\text{ir}}]$, podemos utilizar el *método de bisección* que se describe en el Apéndice D para calcular su inversa. Concretamente, debes realizar los siguientes pasos:

  1. Declara tres variables, $\lambda_{\text{mín}}$, $\lambda_{\text{máx}}$ y λ_{media}. Inicializa las dos primeras con los valores $\lambda_{\text{mín}} \leftarrow \lambda_{\text{uv}}$ y $\lambda_{\text{máx}} \leftarrow \lambda_{\text{ir}}$.
  2. Expresa el ángulo Θ en radianes.
  3. Inicializa $\lambda_{\text{media}} \leftarrow (\lambda_{\text{mín}} + \lambda_{\text{máx}})/2$ y, usando las ecuaciones 2.14 a 2.16, calcula $\Theta_{\text{máx}}(\lambda_{\text{media}})$.
  4. Redefine el intervalo de búsqueda $[\lambda_{\text{mín}}, \lambda_{\text{máx}}]$ como se indica a continuación:
     - Si $\Theta < \Theta_{\text{máx}}(\lambda_{\text{media}})$, sabemos que $\lambda < \lambda_{\text{media}}$; por tanto, λ está en el intervalo $[\lambda_{\text{mín}}, \lambda_{\text{media}}]$ y debemos ajustar $\lambda_{\text{máx}} \leftarrow \lambda_{\text{media}}$.
     - Si $\Theta \geq \Theta_{\text{máx}}(\lambda_{\text{media}})$, sabemos que $\lambda \geq \lambda_{\text{media}}$; por tanto, λ está en el intervalo $[\lambda_{\text{media}}, \lambda_{\text{máx}}]$ y debemos ajustar $\lambda_{\text{mín}} \leftarrow \lambda_{\text{media}}$.
  5. Los pasos (3) y (4) se repiten hasta que el valor de λ esté acotado con la precisión deseada: $\lambda_{\text{máx}} - \lambda_{\text{mín}} < \Delta$.

  El programa escribe en la pantalla el valor estimado $(\lambda_{\text{máx}} + \lambda_{\text{mín}})/2$, y solicita al usuario un nuevo ángulo.

El programa finaliza cuando el usuario introduce un ángulo negativo.

Para el cálculo del seno, el arcoseno y la raíz cuadrada puedes utilizar las funciones `sin`, `asin` y `sqrt`, respectivamente, que están declaradas en el fichero de cabecera `math.h`. El argumento de la función `sin` es un ángulo expresado en radianes, mientras que la función `asin` devuelve su resultado también en radianes.

2.4.3. Ejemplos de ejecución

Comprueba si tu programa hace los cálculos esperados al menos en los ejemplos siguientes.

```
Con que precision desea los resultados? 1E-5

Introduzca un ángulo (grados sexagesimales): -1
```

```
Con que precision desea los resultados? 1E-5

Introduzca un ángulo (grados sexagesimales): 40
   Se observa luz blanca tenue

Introduzca un ángulo (grados sexagesimales): 43
   No hay luz en esa direccion
```

```
Introduzca un ángulo (grados sexagesimales): 40.5
   lambda = 407.236 nm

Introduzca un ángulo (grados sexagesimales): 41
   lambda = 454.669 nm

Introduzca un ángulo (grados sexagesimales): 41.5
   lambda = 518.169 nm

Introduzca un ángulo (grados sexagesimales): 42
   lambda = 617.727 nm

Introduzca un ángulo (grados sexagesimales): 42.5
   lambda = 769.881 nm

Introduzca un ángulo (grados sexagesimales): -1
```

Cuadro 2.2: Disposición de los caracteres

A	B	C	D	E	F	G	H	I
Z								J
Y								K
X								L
W								M
V	U	T	S	R	Q	P	O	N

2.5. Proyecto RSA: Algoritmos de sustitución y transposición

2.5.1. Introducción

La transposición y la sustitución son dos técnicas tradicionales en criptografía. Básicamente, la transposición consiste en reordenar los caracteres de un texto; la sustitución, en reemplazar cada carácter por otro predefinido. En esta tarea diseñaremos y programaremos un algoritmo sencillo de cada tipo.

2.5.2. Descripción de la tarea

Algoritmo de sustitución

Para este algoritmo consideraremos mensajes formados exclusivamente por letras mayúsculas, exluida la 'Ñ', sin signos ortográficos. Un mensaje termina con un punto ('.'). Ese punto no es un carácter del mensaje: se limita a indicar que hemos llegado al final, y no se le aplica el algoritmo de sustitución.

Imagina que los caracteres de la 'A' a la 'Z' están dispuestos a lo largo de una línea cerrada, como se muestra en la Tabla 2.2. Para cifrar un texto necesitaremos una clave, que es un número entero n. Cada carácter del texto será reemplazado por el que se obtiene partiendo de él y desplazándonos n posiciones en el sentido de las agujas del reloj (si n es positivo), o en el sentido opuesto (si n es negativo).

En la Tabla 2.3 mostramos varios ejemplos de sustituciones de caracteres para distintos valores de n.

Cuadro 2.3: Ejemplos de sustitución

Carácter inicial	n	Carácter transformado
A	5	F
A	26	A
A	52	A
A	57	F
A	-2	Y
A	-26	A
A	-52	A
A	-54	Y
I	5	N
N	21	I
N	-5	I

Cuadro 2.4: Ejemplos de transposición

Mensaje inicial	Mensaje cifrado
.	.
A.	A.
AB.	BA.
ABC.	BAC.
ABCDEF.	BADCFE.
ABCDEFG.	BADCFEG.

Algoritmo de transposición

Para este algoritmo consideraremos mensajes formados por caracteres cualesquiera, excluidos tanto el punto ('.') como los caracteres "en blanco" (espacio en blanco, tabulador, salto de línea). Un mensaje termina con un punto ('.'). Ese punto no es un carácter del mensaje: se limita a indicar que hemos llegado al final, y no se le aplica el algoritmo de transposición.

Dado un texto terminado en un punto tomaremos sus caracteres por parejas e intercambiaremos el orden de cada uno dentro de la pareja. La Tabla 2.4 muestra el resultado del algoritmo.

Comportamiento del programa

El programa solicitará al usuario una opción ('s' o 'S' si desea aplicar el algoritmo de sustitución, 't' o 'T' si desea transposición). A continuación el usuario introducirá un texto (en el caso de transposición) o un entero seguido de un texto (en el caso de sustitución). El texto terminará en un punto ('.'). El programa devolverá el texto encriptado según la opción escogida por el usuario.

El proceso anterior se repetirá hasta que, al solicitarle una opción, el usuario introduzca un valor distinto de 's', 'S', 't' o 'T'.

2.5.3. Ejemplos de ejecución

```
Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: t
Introduzca su texto (acabado en un punto): Euler-Fermat.

uEel-reFmrta.


Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: T
Introduzca su texto (acabado en un punto): Euler.

uEelr.


Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: s
Introduzca un entero y su texto (acabado en un punto): 26 ABECEDARIO.

ABECEDARIO.


Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: s
Introduzca un entero y su texto (acabado en un punto): -5 ABECEDARIO.

VWZXZYVMDJ.


Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: S
```

```
Introduzca un entero y su texto (acabado en un punto): 57 VWZXZYVMDJ.

ABECEDARIO.


Que desea hacer ?
   transposicion: pulse 't' o 'T'
   sustitucion: pulse 's' o 'S'
   salir: pulse cualquier otra tecla

Introduzca su opcion: x
```

2.5.4. Ayudas

Para el algoritmo de sustitución, puedes codificar cada carácter mediante un entero ('A' $\rightarrow$ 0, 'Z' $\rightarrow$ 25). Súmale a ese código el número n. Observa la Tabla 2.3 ¿ Qué operación hay que hacer para transformar 0 + 26 en 0, 0 + 52 en 0, 0 + 57 en 5, 0 - 2 en 24, 0 - 26 en 0, 0 - 52 en 0, 0 - 54 en 24 ?

2.5.5. Para saber más... (congruencia)

En esta tarea hemos considerado que 0 + 26 y 0 + 52 eran equivalentes a 0, 0 + 27 y 0 + 53 a 1... Esta situación no es ajena a nuestra vida cotidiana, ocurre al contar las horas. Dentro de 24 o de 48 horas será la misma hora que ahora, y dentro de 25 o 49 una hora más. La noción matemática que formaliza esa equivalencia se llama congruencia:

Dados los enteros a, b, n, con $n > 0$, decimos que a es congruente con b módulo n, y escribimos

$$a \equiv b \ (mod\ n)$$

si n es divisor de $(a - b)$. Llamamos al número n módulo de la congruencia.

En nuestra tarea hemos hecho uso de que 0 + 27 y 0 + 53 son congruentes con 1 módulo 26 ($27 \equiv 1 \pmod{26}$, $53 \equiv 1 \pmod{26}$); al contar las horas, 12 + 25 es congruente con 13 módulo 24 ($37 \equiv 13 \pmod{24}$).

2.6. Métodos de Montecarlo

2.6.1. Introducción

Los métodos de Monte Carlo (o experimentos de Monte Carlo) son algoritmos computacionales que se basan en el muestreo aleatorio repetido para obtener resultados numéricos. En problemas relacionados con la física, los métodos de Montecarlo son muy útiles para simular sistemas con muchos grados de libertad como fluidos, materiales desordenados o estructuras celulares como el modelo celular de Potts.

En esta práctica, presentamos la aplicación del método de Montecarlo al problema de las Agujas de Buffon. Georges Louis Leclerc, conde de Buffon (Montbard, Borgoña, 7 de septiembre de 1707 - París, 16 de abril de 1788) fue un naturalista, botánico, matemático, biólogo, cosmólogo y escritor francés que pretendió compendiar todo el saber humano sobre el mundo natural en su obra en 44 volúmenes Histoire naturelle. Planteó el problema de las agujas de Buffon en 1733 y lo presentó resuelto por él mismo en 1757. El enunciado del problema es como sigue:

Supongamos que en el suelo hemos colocado tiras paralelas de madera, cada una con la misma anchura y dejamos caer una aguja sobre el suelo. ¿Cuál es la probabilidad de que la aguja caiga entre dos tiras?

Equivalentemente, si dibujamos líneas verticales paralelas cuya separación coincide con la anchura de la tira de madera, el problema se podría reformular preguntando ¿Cuál es la probabilidad de que la aguja caiga sobre una línea vertical?

Si ese resultado matemático no se conoce, como era el caso cuando Buffon planteó su problema, se puede calcular el valor de la probablidad experimentalmente mediante el método de Montecarlo, simulando una gran cantidad de lanzamientos de agujas y contando el número de lanzamientos que cruzan una línea vertical. Entonces la probabilidad calculada experimentalmente es:

$$probabilidadExperimental = (\#AgujasCruzanLíneaVertical)/(\#AgujasLanzadas)$$

Veamos un ejemplo en la Figura 2.7 donde la longitud de cada aguja es exactamente igual a la separación entre dos líneas verticales. En el experimento se han lanzado 7 agujas de las cuales 5 han caído sobre una línea vertical. Por tanto, la probabilidad experimental buscada es:

$$probabilidadExperimental = (\#AgujasCruzanLíneaVertical)/(\#AgujasLanzadas) = 5/7$$

Posteriormente, el problema se resolvió matemáticamente. Si la longitud de cada aguja es $longitudAguja$ y la separación entre líneas verticales es $separacion$, siendo $longitudAguja <= separacion$, se puede demostrar matemáticamente que la probabilidad buscada p es

$$p = 2\, longitudAguja/(\pi\, separacion)$$

De paso, podemos obtener una aproximación del valor de π. Para ello, en la solución matemática, sustituimos la probabilidad matemática por la probabilidad experimental $probabilidadExperimental = 2\, longitudAguja/(\pi\, separacion)$.

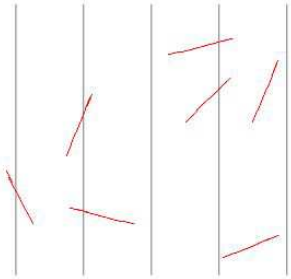


Figura 2.7: Ejemplo de lanzamiento de agujas

y despejando π

$\pi = 2\, longitudAguja/(probabilidadExperimental\ separacion)$

En los ejemplos que te mostramos posteriormente, verás que las aproximaciones de π que obtenemos no son excesivamente buenas. Por ejemplo, con 16 millones de lanzamientos sólo obtenemos dos dígitos de precisión. Sin embargo, en el siglo XVI, el matemático aleman Ludolph van Ceulen (28 de enero de 1540, Hildesheim - 31 de diciembre de 1610, Leiden) ya encontró las primeras 35 cifras de π e incluso las mandó inscribir en su tumba.

Puedes encontrar información completa sobre este tema en [31].

2.6.2. Descripción de la tarea

En esta tarea, vas a simular varios experimentos del problema de las agujas de Buffon y en cada experimento duplicarás el número de lanzamientos de agujas respecto al experimento anterior. Diseña y programa en C/C++ un algoritmo que haga las siguientes tareas:

1. Solicite al usuario los datos de entrada. Éstos son:
   - `longitudAguja` la longitud de la aguja
   - `separacion` la separación entre líneas verticales
2. Si el usuario elige una longitud superior a la separación entre líneas verticales, repetirá la solicitud hasta que la longitud sea igual o inferior a la separación entre líneas verticales
3. Pedirá cuántas agujas se lanzan en el primer experimento. Si el número introducido es negativo, repetirá la solicitud hasta que éste sea positivo. También pedirá cuántos experimentos se hacen, entendiendo que en cada nueva repetición se duplicará el número de agujas lanzadas. Por ejemplo, si el número inicial de agujas es de 1.000.000 y se hacen cinco experimentos, el número de agujas que se lanzará en cada uno de los cinco experimentos es: 1.000.000, 2.000.000, 4.000.000, 8.000.000, 16.000.000
4. Para cada experimento, simulará el lanzamiento de cada una de las agujas que lo componen, decidirá si la aguja ha caído sobre una línea vertical y actualizará el contador que indica el número de veces que una aguja ha cruzado la línea. Para modelizar matemáticamente cada lanzamiento, sin pérdida de generalidad podemos suponer que el extremo izquierdo de la aguja está entre 0 y *separacion*. En esa situación, el ángulo que forma la aguja con la horizontal está comprendido entre $-\pi/2$ y $\pi/2$. Así, para simular el lanzamiento de una aguja, deberemos generar aleatoriamente dos números:

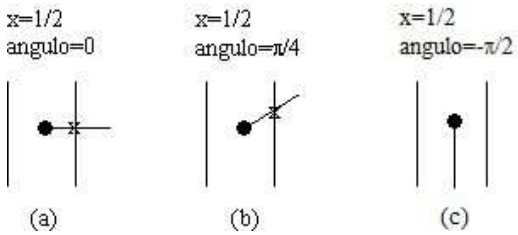


Figura 2.8: Ejemplo de posiciones de varias agujas

- *xIzquierda*. La coordenada x del extremo izquierdo de la aguja, que está comprendida entre 0 y *separacion* ambos incluidos
- *angulo*. El ángulo que forma la aguja con la horizontal, que está comprendido entre $-\pi/2$ y $\pi/2$ ambos incluidos

Una aguja corta la línea vertical si $xIzquierda = 0$ o bien si corta la línea $x = separacion$. Para determinar si el corte con la línea $x = separacion$ se produce, comprobamos si la coordenada x del extremo derecho de la aguja (obtenida a partir de $xIzquierda$ y de *angulo*) es mayor o igual que *separacion*. Obsérvese que en el caso de que $xDerecha = separacion$ consideramos que la aguja corta la línea $x = separacion$ aunque realmente se apoya en ella sin llegar a cortarla.

En la Figura 2.8, observamos varios ejemplos suponiendo que separacion= 1 y que longitudAguja=1. El extremo izquierdo de la aguja se señala con un círculo negro. En cada ejemplo, las líneas verticales son las líneas x=0 y x=1. Para cada aguja, se muestran los valores correspondientes de la coordenada x del extremo izquierdo de la aguja y del ángulo. Cuando la aguja corta a la línea x=1, se indica el punto de corte con un aspa sobre la línea.

5. Una vez finalizado el experimento, calculará el valor de la *probabilidadExperimental*
6. Como resultado, mostrará, para cada experimento, cuatro datos:
   - La probabilidad calculada matemáticamente.
   - La probabilidad calculada con el experimento de las agujas de Buffon.
   - La diferencia entre la probabilidad calculada matemáticamente y la calculada con el experimento de las agujas de Buffon
   - Una aproximación al valor de pi obtenida con el experimento de las agujas de Buffon
7. El programa pregunta al usuario si desea realizar otra tanda de experimentos. Si la respuesta es 's', volverá a empezar solicitando de nuevo todos los datos de entrada. Si la respuesta es 'n', el programa terminará

2.6.3. Ejemplos de ejecución

```
Dime la distancia entre líneas verticales: 1
Dime la longitud de la aguja: 1
Dime el numero total de experimentos: 5
Dime el numero de lanzamientos del primer experimento: 1000000
```

```
Experimento: 1
Numero de intentos: 1000000
Probabilidad matematica: 0.63662
Probabilidad experimental: 0.637168
Diferencia entre ambas: -0.000547707
Aproximacion de pi: 3.13889

Experimento: 2
Numero de intentos: 2000000
Probabilidad matematica: 0.63662
Probabilidad experimental: 0.636747
Diferencia entre ambas: -0.000127196
Aproximacion de pi: 3.14096

Experimento: 3
Numero de intentos: 4000000
Probabilidad matematica: 0.63662
Probabilidad experimental: 0.637107
Diferencia entre ambas: -0.000486493
Aproximacion de pi: 3.13919

Experimento: 4
Numero de intentos: 8000000
Probabilidad matematica: 0.63662
Probabilidad experimental: 0.636434
Diferencia entre ambas: 0.000186563
Aproximacion de pi: 3.14251

Experimento: 5
Numero de intentos: 16000000
Probabilidad matematica: 0.63662
Probabilidad experimental: 0.636744
Diferencia entre ambas: -0.000123978
Aproximacion de pi: 3.14098

Desea repetir la tanda de experimentos?n
```

2.6.4. Ayudas

Para generar números aleatorios, podemos usar la función rand(), que cada vez que se invoca devuelve un número entero pseudo-aleatorio en el intervalo $[0, RAND\_MAX]$. Para utilizar rand() y $RAND\_MAX$ deberás incluir en tu programa el fichero de cabecera <stdlib.h>.

Así

```
double x = (1.0 * rand())/RAND_MAX;
```

genera un número aleatorio entre 0 y 1 y

```
double angulo= (pi * rand()/RAND_MAX)-(pi/2.0);
```

genera un número aleatorio entre $-\pi/2$ y $\pi/2$

2.7. Proyecto 2C: Dinámica de una órbita elíptica (i)

En la tarea 1.7 caracterizamos la órbita de una partícula sometida a una fuerza central de módulo K/r^2 a partir de la energía y el módulo del momento angular. Recordemos que, en el caso de que la órbita resulte ser elíptica, la posición r de la partícula queda determinada por el ángulo θ mediante la expresión en coordenadas polares

$$r = \frac{a \cdot (1 - e^2)}{1 - e \cdot \cos(\theta - \phi_0)} \ . \tag{2.17}$$

En esta tarea iniciaremos el estudio cuantitativo de la dinámica de estas soluciones elípticas. Concretamente, dada una órbita caracterizada por su excentricidad, e, y el ángulo, ϕ_0, que su semieje mayor forma con el eje de coordenadas polares, calcularemos el tiempo que tarda la partícula en moverse de una posición θ_0 a otra $\theta_F \geq \theta_0$.

2.7.1. Contexto del problema

De la conservación del momento angular se sigue que el tiempo transcurrido entre dos instantes t_0 y t_F es directamente proporcional al área barrida por el vector de posición de la partícula en ese intervalo de tiempo (la parte sombreada de la figura 2.9). Ello nos permite estudiar la dinámica de la partícula: si los ángulos θ_0 y θ_F determinan su posición en los instantes t_0 y t_F, sabiendo que el área de la elipse es $\pi\, a^2 \sqrt{1 - e^2}$, tenemos que

$$\frac{t_F - t_0}{T} = \frac{\int_{\theta_0}^{\theta_F} dA}{\pi\, a^2 \sqrt{1 - e^2}} = \frac{1/2 \int_{\theta_0}^{\theta_F} r^2 d\theta}{\pi\, a^2 \sqrt{1 - e^2}} \ ,$$

donde T es el periodo de la órbita. Sustituyendo r por la expresión (2.17) se obtiene:

$$\frac{t_F - t_0}{T} = \frac{1}{2\pi} \int_{\theta_0}^{\theta_F} \frac{(1 - e^2)^{3/2}}{(1 - e \cdot \cos(\theta - \phi_0))^2} d\theta \ . \tag{2.18}$$

Así, el cociente $(t_F - t_0)/T$ es una función de la excentricidad e y del ángulo ϕ_0, pero es independiente del semieje mayor a.

La principal dificultad de esta tarea reside en resolver la integral de la expresión (2.18). En situaciones similares, ante la incapacidad o imposibilidad de dar una solución analítica a un problema, se recurre a métodos numéricos. En nuestro caso utilizaremos el método de Newton–Cotes[1] para calcular numéricamente una solución aproximada de la ecuación (2.18).

2.7.2. Descripción de la tarea

Diseña y programa en C/C++ un programa que tenga el comportamiento descrito a continuación.

El programa comenzará solicitando al usuario que introduzca por teclado los parámetros de la elipse relevantes para nuestro problema (e y ϕ_0). A continuación le solicitará un número entero positivo, k.

[1] A menudo ocurre que alguien con más competencias matemáticas sí es capaz de resolver el problema. Este es nuestro caso: en la tarea 3.6 daremos una solución analítica para la integral de la ecuación (2.18).

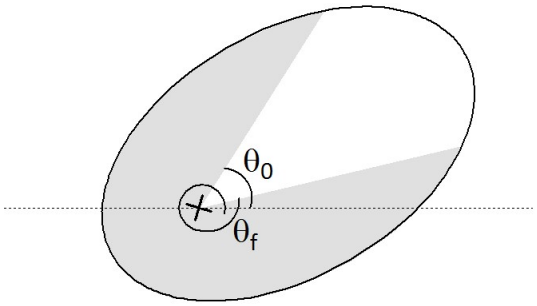


Figura 2.9: Área barrida por el vector posición entre θ_0 y θ_F.

Supondremos que la partícula se encuentra inicialmente en la posición determinada por el ángulo $\theta_0 = \phi_0$. Entonces, el programa mostrará, para cada ángulo

$$\theta_i \in \{\phi_0 + 2i\pi/k \mid 1 \leq i \leq k\} = \{\phi_0 + 2\pi/k, \phi_0 + 4\pi/k, \phi_0 + 6\pi/k, \ldots, \phi_0 + 2\pi\} \ ,$$

el instante de tiempo t_i para el cual la partícula alcanza la posición θ_i. Los tiempos t_i deben expresarse en términos del periodo T.

Tras la salida de estos resultados, el programa invitará al usuario a introducir un nuevo valor de k y repetirá el proceso, salvo que haya proporcionado un entero menor o igual que 0, en cuyo caso el programa terminará.

2.7.3. Ejemplos de ejecución

```
Introduzca los parametros de la orbita:
         Excentricidad: 0.5
         phi0 (en radianes): 0

Cuantos intervalos desea considerar? 4
Angulo   Tiempo
1.5708    0.40225 T
3.14159  0.5 T
4.71239  0.597751 T
6.28319  1 T

Cuantos intervalos desea considerar? 8
Angulo   Tiempo
0.785398    0.273478 T
1.5708        0.40225 T
2.35619      0.461283 T
3.14159      0.5 T
3.92699      0.538717 T
4.71239      0.597751 T
5.49779      0.726523 T
6.28319       1 T
```

```
Cuantos intervalos desea considerar? -1
```

```
Introduzca los parametros de la orbita:
         Excentricidad: 0.7
         phi0 (en radianes): 1

Cuantos intervalos desea considerar? 6
Angulo   Tiempo
2.0472  0.405782 T
3.0944  0.4753 T
4.14159 0.5 T
5.18879 0.5247 T
6.23599 0.594218 T
7.28319 1 T

Cuantos intervalos desea considerar? 0
```

Observa que, de acuerdo con la segunda ley de Kepler, a la partícula le cuesta bastante más tiempo cubrir los $\pi/2$ radianes que hay entre ϕ_0 y $\phi_0 + \pi/2$ que los que hay entre $\phi_0 + \pi/2$ y $\phi_0 + \pi$.

2.7.4. Ayudas

De acuerdo con la expresión C.3 del apéndice C, podemos calcular un valor aproximado de la integral de la ecuación (2.18) sustituyéndola por un sumatorio finito:

$$\begin{aligned}\frac{t_F - t_0}{T} &= \frac{(1-e^2)^{3/2}}{2\pi}\sum_{i=0}^{n-1}\frac{f(\theta_{i+1}) + f(\theta_i)}{2}\Delta\theta \\ &= \frac{(1-e^2)^{3/2}\Delta\theta}{2\pi}\left(\frac{f(\theta_0) + f(\theta_F)}{2} + \sum_{i=1}^{n-1} f(\theta_i)\right) ,\end{aligned}$$

donde

$$f(\theta) = \frac{1}{(1 - e\cdot\cos(\theta - \phi_0))^2} ,$$

$\Delta\theta = (\theta_F - \theta_0)/n$ y $\theta_{i+1} = \theta_i + \Delta\theta$, para $0 \le i < n$. Puedes tomar $n = 10^6$.

El coseno de un ángulo puede calcularse usando la función `cos(x)`, declarada en el fichero de cabecera `<math.h>`, cuyo argumento debe ser un ángulo expresado en radianes.

2.8. Proyecto ST: Longitud de un camino

2.8.1. Introducción

Vamos a medir la longitud de un camino usando alguna de las métricas que hemos usado en la tarea anterior: la distancia taxi, la distancia euclídea o la distancia del máximo.

2.8.2. Descripción de la actividad

En esta actividad trabajaremos en dimensión n=2. El programa pide al usuario las unidades de medida usadas, que pueden ser metros, kilómetros o millas así como la distancia que se va a utilizar para medir la longitud del camino. La distancia puede ser: distancia taxi, distancia euclidea o distancia del máximo. Finalmente, el programa pide las coordenadas de los puntos de un camino. No admitiremos que un nuevo punto del camino esté en la recta que une los dos puntos anteriores del camino. El final del camino se indica con el punto $(0, 0)$ y este punto no pertenece al camino

El resultado es la longitud del camino calculada con la distancia elegida y expresada en metros.

Para ello, diseña y programa en C/C++ un algoritmo que haga las siguientes tareas:

1. Solicite al usuario las unidades que va a utilizar. Los valores posibles son:
   - 'k' o 'K' para una distancia en kilómetros
   - 'a' o 'A' para una distancia en millas
   - 'm' o 'M' para una distancia en metros
   - 'z' para terminar
2. Si el usuario elige una opción distinta de 'k' o 'K' o 'a' o 'A' o 'm' o 'M', vuelve a solicitar el dato excepto si el valor introducido es la letra 'z' en cuyo caso termina.
3. Si no ha terminado, solicita la distancia que va a usar. Los valores posibles son:
   - 't' para la distancia taxi
   - 'e' para la distancia euclídea
   - 'm' para la distancia del máximo
   - 'z' para terminar
4. Si el usuario elige una opción distinta de 't' o 'e' o 'm', vuelve a solicitar el dato excepto si el valor introducido es la letra 'z' en cuyo caso el programa termina.
5. Si el programa no ha terminado, pide al usuario que introduzca las coordenadas de cada punto y, a continuación, calcula la distancia entre cada dos puntos y acumula ese valor a la longitud del camino. A partir del tercer punto del camino, no se permite que el nuevo punto esté en la recta que une los dos puntos anteriores. Si esto sucede, el programa debe pedir un nuevo punto.
6. El programa finaliza cuando el usuario introduce el punto $(0, 0)$. Este punto NO pertenece al camino. A continuación, muestra la longitud del camino.

En la Tabla 2.5 recogemos algunos ejemplos de ejecución del programa.
Se valorará especialmente que el programa no tenga código repetido.

Cuadro 2.5: Ejemplos de puntos

Antepenúltimo punto	Penúltimo punto	Nuevo punto	¿Se admite el nuevo punto?
$(1, 1)$	$(3, 3)$	$(5, 6)$	Sí, no está en la recta de los anteriores puntos
$(1, 1)$	$(3, 3)$	$(7, 7)$	No, está en la recta de los anteriores puntos
$(1, 1)$	$(3, 3)$	$(3, 3)$	No, está en la recta de los anteriores puntos
$(1, 1)$	$(3, 3)$	$(2, 2)$	No, está en la recta de los anteriores puntos
$(1, 1)$	$(3, 3)$	$(1, 1)$	No, está en la recta de los anteriores puntos
$(1, 1)$	$(3, 3)$	$(-1, -1)$	No, está en la recta de los anteriores puntos

2.8.3. Ejemplos de ejecución

Algunos ejemplos de ejecución del programa son:

```
Calculadora de longitudes de caminos
Dame las unidades: z
Programa terminado
```

```
Calculadora de longitudes de caminos
Dame las unidades: h
Opción no  válida
Dame las unidades: z
Programa terminado
```

```
Calculadora de longitudes de caminos
Dame las unidades: m
Dame la metrica: z
Programa terminado
```

```
Calculadora de longitudes de caminos
Dame las unidades: k
Dame la metrica: m
Dame el primer punto: 0 0
La longitud es 0 metros.
```

```
Calculadora de longitudes de caminos
Dame las unidades: m
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 0 0
La longitud es 0 metros.
```

```
Calculadora de longitudes de caminos
Dame las unidades: m
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 3 3
Dame el siguiente punto: 0 0
La longitud es 2 metros.
```

```
Calculadora de longitudes de caminos
Dame las unidades: k
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 3 3
Dame el siguiente punto: 5 7
Dame el siguiente punto: 0 0
La longitud es 6000 metros.
```

```
Calculadora de longitudes de caminos
Dame las unidades: k
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 3 3
Dame el siguiente punto: 2 2
Punto no valido
Dame el siguiente punto: 5 7
Dame el siguiente punto: 0 0
La longitud es 6000 metros.
```

```
Calculadora de longitudes de caminos
Dame las unidades: k
Dame la metrica: m
Dame el primer punto: 1 1
Dame el segundo punto: 5  6
Dame el siguiente punto: 12 30
Dame el siguiente punto: 3 3
Dame el siguiente punto: 5 5
Dame el siguiente punto: 0 0
La distancia es 58000 metros.
```

2.8.4. Ayuda

Para saber si el punto (z_1, z_2) está en la misma recta que los puntos (x_1, x_2) e (y_1, y_2), determina si los tres puntos están alineados.

Para ello, la ecuación de la recta que une (x_1, x_2) e (y_1, y_2) es

$$(t_1, t_2) = (x_1, x_2) + \lambda((y_1 - x_1, y_2 - x_2)$$

Sustituyendo el punto (z_1, z_2), obtenemos

$$(z_1, z_2) = (x_1, x_2) + \lambda((y_1 - x_1, y_2 - x_2)$$

y despejando

$$(z_1 - x_1, z_2 - x_2) = \lambda((y_1 - x_1, y_2 - x_2)$$

De la fórmula anterior, para que los tres puntos estén alineados, los vectores $(z_1 - x_1, z_2 - x_2)$ y $(y_1 - x_1, y_2 - x_2)$ deben ser linealmente dependientes, es decir, el rango de la matriz

$$\begin{pmatrix} z_1 - x_1 & z_2 - x_2 \\ y_1 - x_1 & y_2 - x_2 \end{pmatrix}$$

debe ser igual 1. Esto sucede si y sólo si el siguiente determinante es igual a cero

$$\begin{vmatrix} z_1 - x_1 & z_2 - x_2 \\ y_1 - x_1 & y_2 - x_2 \end{vmatrix}$$

2.9. El oscilador armónico: solución numérica

En esta tarea resolveremos numéricamente la ecuación diferencial del oscilador armónico, considerando el amortiguamiento y una fuerza externa periódica:

$$\ddot{x} = -\alpha\, x \,-\, \beta\, \dot{x} + \Gamma\, \cos(\omega_d\, t) \tag{2.19}$$

Las dimensiones de las distintas magnitudes son: $[\alpha] = \mathrm{T}^{-2}$, $[\beta] = \mathrm{T}^{-1}$, $[\Gamma] = \mathrm{L} \cdot \mathrm{T}^{-2}$, $[\omega_d] = \mathrm{T}^{-1}$.

2.9.1. Introducción

Integración numérica: método de Euler

Reescribimos la ecuación 2.19 como un sistema de dos ecuaciones diferenciales de primer orden acopladas:

$$\begin{aligned} \frac{dx}{dt} &= v \\ \frac{dv}{dt} &= a(x, v, t; \alpha, \beta, \Gamma, \omega_d) \end{aligned} \tag{2.20}$$

donde a, la aceleración, es la expresión del segundo miembro de 2.19. Observamos que la aceleración es una función del tiempo, del estado del oscilador (x, $\dot{x}$) y de los parámetros del mismo ($\alpha, \beta, \Gamma, \omega_d$).

De la definición de derivada se tiene:

$$\begin{aligned} x(t + \Delta t) &= x(t) \,+\, \int_t^{t+\Delta t} \dot{x}(t')\, dt' \\ \dot{x}(t + \Delta t) &= \dot{x}(t) \,+\, \int_t^{t+\Delta t} \ddot{x}(t')\, dt' \end{aligned} \tag{2.21}$$

El significado de las expresiones previas nos es de sobra conocido: la posición se obtiene integrando la velocidad, que a su vez se obtiene integrando la aceleración.

Si en las ecuaciones 2.21 se toma un valor de Δt muy pequeño, podemos hacer la aproximación de que la función integrada permanece prácticamente constante en el intervalo de integración, y por tanto $\int_t^{t+\Delta t} \dot{x}(t')\, dt' \approx \dot{x}(t)\, \Delta t$ y $\int_t^{t+\Delta t} \ddot{x}(t')\, dt' \approx \ddot{x}(t)\, \Delta t$. Dicho de otro modo: conocidas la posición y la velocidad en un instante t, si Δt es lo suficientemente pequeño podemos calcular $x(t + \Delta t)$ y $v(t + \Delta t)$ haciendo el desarrollo en serie de Taylor de x y v en t y truncando la serie en el término Δt.

Además, discretizaremos el tiempo: consideraremos sólo instantes $t_n = n\Delta t$. Denominaremos $x_n = x(t_n)$, $\dot{x_n} = \dot{x}(t_n)$ y $\ddot{x_n} = \ddot{x}(t_n)$. Las ecuaciones 2.21 quedan:

$$\begin{aligned} x_{n+1} &= x_n \,+\, \dot{x}_n\, \Delta t \\ \dot{x}_{n+1} &= \dot{x}_n \,+\, a(x_n, \dot{x}_n, t; \alpha, \beta, \Gamma, \omega_d)\, \Delta t \end{aligned} \tag{2.22}$$

Con las consideraciones anteriores, describiremos la dinámica del oscilador mediante un proceso iterativo. Al inicio de la iteración n-ésima conocemos la posición y la velocidad en t_n, x_n y $\dot{x}_n$. Calculamos la aceleración mediante la expresión 2.19 y, siguiendo las expresiones 2.22, obtenemos la posición y la velocidad en el instante inmediatamente posterior, t_{n+1}.

Tiempos relevantes

En esta tarea existen tres magnitudes temporales relevantes:

- Δt. Su valor debe ser una solución de compromiso entre precisión y eficiencia: cuanto más pequeño sea, tanto mejor será la aproximación presentada en la subsección 2.9.1 (sustituir una integral por una suma finita), y más precisos serán los resultados. Por otro lado, a menor Δt más iteraciones deberemos realizar para cubrir el intervalo de tiempo en que deseamos estudiar la dinámica.

- Con qué frecuencia se mostrará al usuario el estado del oscilador, t_{cout}.

- Cuánto tiempo cubrirá la simulación, t_{simul}.

En el guión os propondremos valores adecuados para estas magnitudes.

En concreto, tomaremos como referencia un **tiempo característico del sistema** T:

- En ausencia de término forzante ($\Gamma = 0$), $T = 2\pi\alpha^{-1/2}$ (el periodo de oscilación natural).

- Si $\Gamma \neq 0$, $T = 2\pi/\omega_D$ (el periodo del término forzante).

Δt, t_{simul} y t_{cout} se calcularán a partir de sendos factores, que serán datos de entrada del programa:

$$\begin{aligned} \Delta t &= \text{factor}_{\Delta} \cdot T, \\ t_{cout} &= \text{factor}_{\text{cout}} \cdot T, , \\ t_{simul} &= \text{factor}_{\text{tSimul}} \cdot T. \end{aligned} \tag{2.23}$$

2.9.2. Descripción de la tarea

Diseña y escribe un programa que pida al usuario los factores factor_{Δ}, $\text{factor}_{\text{cout}}$ y $\text{factor}_{\text{tSimul}}$, la posición y velocidad iniciales (x_0, $\dot{x}_0$) y los parámetros de la ecuación 2.19 (α, β, Γ y ω_d).

Por simplicidad, suponemos que todas las magnitudes se dan en unidades del Sistema Internacional (la longitud, en metros; el tiempo, en segundos; la velocidad, en $m\,s^{-1}$).

Para que la simulación tenga sentido, α, factor_{Δ}, $\text{factor}_{\text{cout}}$ y $\text{factor}_{\text{tSimul}}$ deben ser positivos, y β no negativo. De no ser así, el programa mostrará un mensaje de error y finalizará.

A continuación, se calcularán los valores de T, Δt, t_{simul} y t_{cout} tal como se indica en la subsección 2.9.1.

El programa calculará la dinámica del oscilador hasta $t = t_{simul}$ siguiendo el método de Euler, tal como se ha explicado en la subsección 2.9.1 (ecuaciones 2.22). A intervalos regulares t_{cout} mostrará por pantalla los valores de t, x, $\dot{x}$, así como $\dot{x}^2 + \alpha x^2$ (la energía dividida por $m/2$).

Se mostrarán también:

- En el caso de oscilador sin pérdidas ($\beta = \Gamma = 0$), la solución exacta (ecuación 2.25).

- En el caso del oscilador amortiguado ($\beta \neq 0, \Gamma = 0$), $e^{-\beta \cdot t/2} \sqrt{x_0^2 + \frac{\dot{x}_0^2}{\alpha}}$ (la envolvente de la solución en el límite $\beta \to 0$) y $x\, e^{\beta \cdot t/2}$ (el producto de la solución numérica por el factor $e^{\beta \cdot t/2}$); el análisis de esta última curva hace patente la diferencia entre los amortiguamientos débil, crítico y fuerte.
- En el caso del oscilador forzado ($\Gamma \neq 0$), la solución estacionaria (ecuación 2.30).

2.9.3. Ayuda

Con la suposición de que t_{cout} es un múltiplo de Δt, el programa escribirá una línea en pantalla cada `round (`$t_{cout}/\Delta t$`)` iteraciones.

2.9.4. Ejemplos de ejecución

El fichero `datosEntrada_OA.txt`, descargable desde Moodle, contiene los datos de entrada de los ejemplos discutidos en esta sección.

Al ejecutar nuestro programa reproducimos resultados ya estudiados en Física:

2.9.5. Oscilador armónico sin pérdidas ($\beta = \Gamma = 0$)

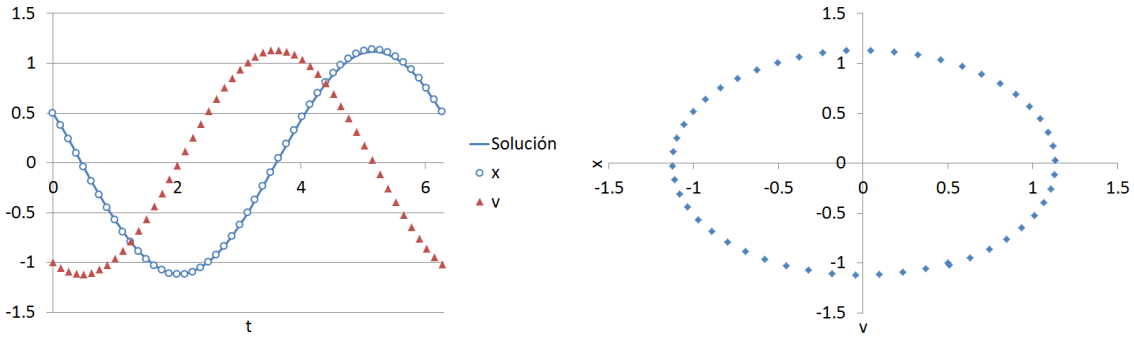
Para los datos de entrada:

```
0.001 0.02 2          0.5 -1          1   0 0 0
```

las primeras líneas devueltas por el programa son:

```
t x v E solucionSinusoidal
0 0.5 -1 1.25 0.5
0.125664 0.370872 -1.0552 1.25099 0.370724
0.251327 0.235791 -1.09379 1.25198 0.235602
0.376991 0.0968839 -1.11516 1.25296 0.0967637
0.502655 -0.0436618 -1.11895 1.25395 -0.0436003
```

El resultado se puede ver en la Figura 2.10, donde se muestra la dinámica del oscilador mediante desde dos perspectivas: la evolución de x y $\dot{x}$ en función de t, y la trayectoria en el espacio de fases (cuyas coordenadas son $x, \dot{x}$). Como la energía es constante, en el espacio de fases el oscilador describe una elipse de semiejes $\sqrt{2E/m}$ y $\sqrt{2E/k}$, donde E, m y k son la energía, la masa y la constante de recuperación.



(a) x (círculos), $\dot{x}$ (rombos) frente a la solución $x = x_M \,\text{seno}\,(\omega_0\, t + \varphi_0)$ (línea continua) (b) Trayectoria del oscilador en el espacio de fases

Figura 2.10: x y $\dot{x}$ frente a t (izquierda) y $\dot{x}$ vs. x (derecha). Los datos de la simulación son: t_Δ = 0.0001 T, t_{tSimul}= 2 T, t_{cout}= 0.02 T, x_0= 0.5 , $\dot{x}_0$ = -1 , α = 1 , β = 0 , Γ= 0 , ω_D= 0.

2.9.6. Oscilador amortiguado ($\beta \neq 0$, $\Gamma = 0$)

El término $-\beta\dot{x}$, opuesto a la velocidad, modeliza la fricción y frena al oscilador. Éste va perdiendo energía hasta llegar finalmente al reposo.

Más concretamente, $x(t) = e^{-\beta \cdot t/2} g(t)$: el factor dominante es una exponencial que decrece con el tiempo. La forma de $g(t)$ depende de la relación entre los parámetros α y β:

- Si $\beta < 2\sqrt{\alpha}$ (amortiguamiento débil), $g(t)$ es una función sinusoidal.
- Si $\beta = 2\sqrt{\alpha}$ (amortiguamiento crítico), $g(t)$ es una función lineal.
- Si $\beta > 2\sqrt{\alpha}$ (oscilador sobreamortiguado), $g(t)$ es una combinación de funciones exponenciales de t.

Esos casos se ilustran en las figuras 2.11a a 2.11f.

2.9.7. Oscilaciones forzadas ($\Gamma \neq 0$)

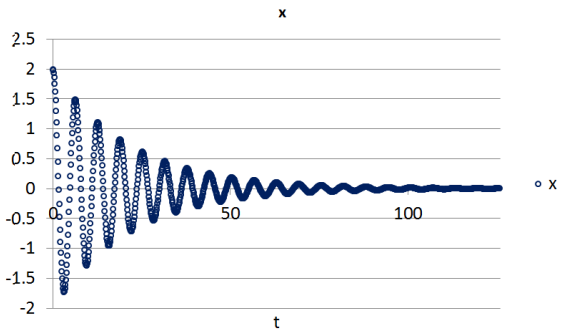
Si $\Gamma \neq 0$ el sistema, tras un tiempo transitorio, acaba oscilando a la frecuencia que le impone el término forzante, ω_D, con una amplitud

$$x_M = \frac{\Gamma}{\sqrt{\beta^2\,\omega_D^2 + \left(\alpha - \omega_D^2\right)^2}} \tag{2.24}$$

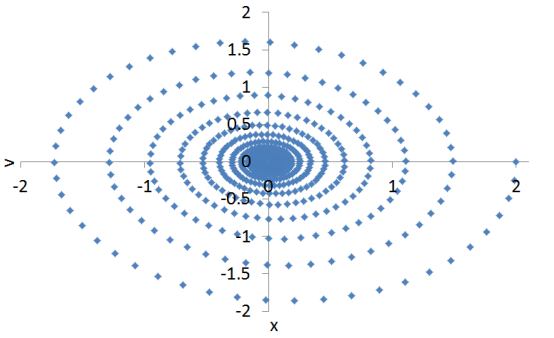
Si la frecuencia del término forzante es igual a la propia del oscilador ($\omega_0 \equiv \sqrt{\alpha} = \omega_D$), tiene lugar el fenómeno llamado *resonancia*. En todo momento, la fuerza *empuja* al oscilador en el sentido adecuado, y la transferencia de potencia es máxima. De no ser por el amortiguamiento, la amplitud de oscilación crecería indefinidamente. La resonancia se ilustra en las simulaciones mostradas en la figura 2.12: tras un tiempo transitorio (en torno a 80 s en 2.12a y 120 s en 2.12b), el sistema oscila con un periodo $T = 2\pi/\sqrt{\alpha} = 2\pi$ y la amplitud dada por 2.24.

Si $\omega_0 \neq \omega_D$, el péndulo inicialmente tiende a oscilar a su frecuencia propia; con el tiempo, acaba oscilando a la que le impone el término forzante. La figura 2.13 muestra esta situación, con $T_0 = 2\pi\alpha^{-1/2} = 2\pi$ y $T_D = 2\pi/\omega_D = \pi$. Inicialmente, la dinámica tiene un periodo T_0: en $t \in [0, 6\pi]$ se producen tres oscilaciones (tal como tiende a hacer espontáneamente el oscilador). Pero en cada periodo x tiene una estructura fina con un mínimo local y dos máximos.

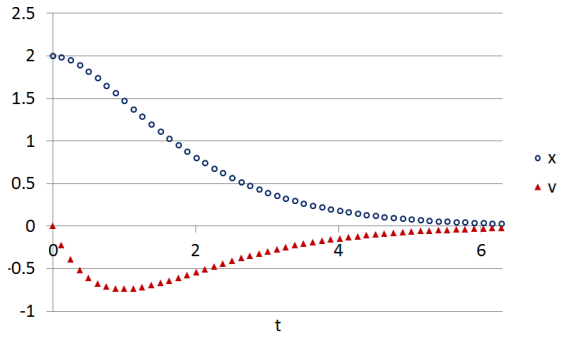
Conforme avanza el tiempo, los mínimos locales y los absolutos tienden a hacerse más próximos, hasta igualarse. En ese momento ya nos encontramos en el régimen estacionario, con un periodo T_D (entre 900 s y $900 + 6\pi$ s tienen lugar 6 oscilaciones completas).



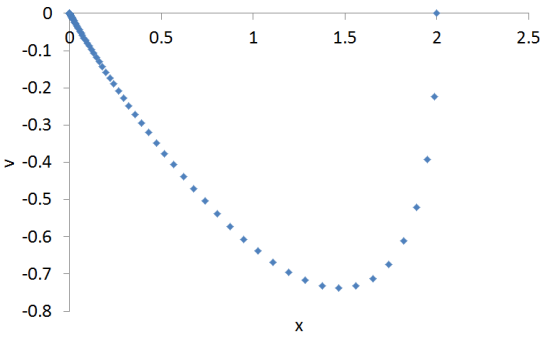
(a) Oscilador débilmente amortiguado ($\beta = 0{,}1$)



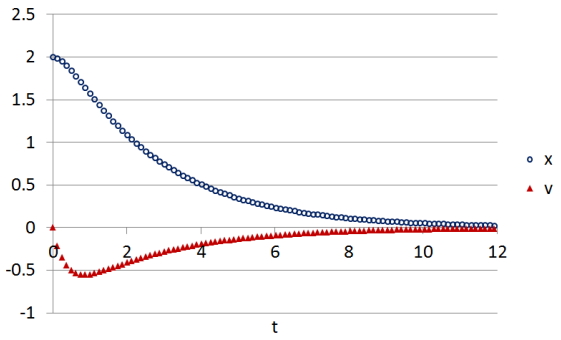
(b) Oscilador débilmente amortiguado ($\beta = 0{,}1$), espacio de fases



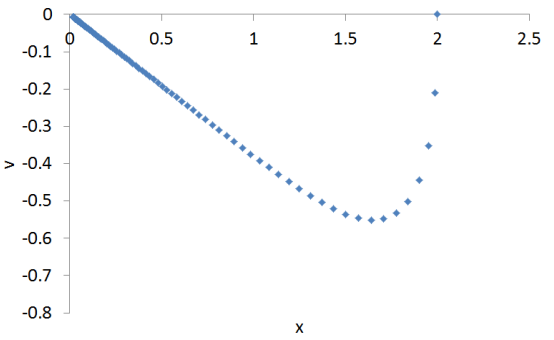
(c) Amortiguamiento crítico($\beta = 2$)



(d) Amortiguamiento crítico($\beta = 2$), espacio de fases

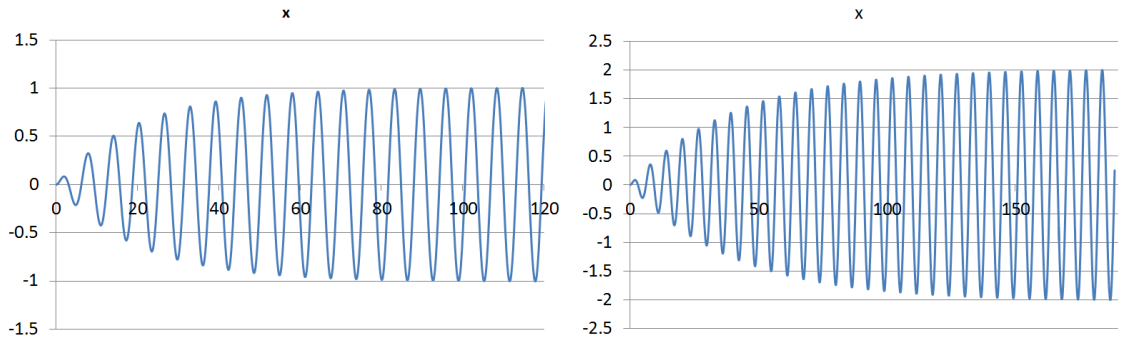


(e) Oscilador sobreamortiguado ($\beta = 3$)



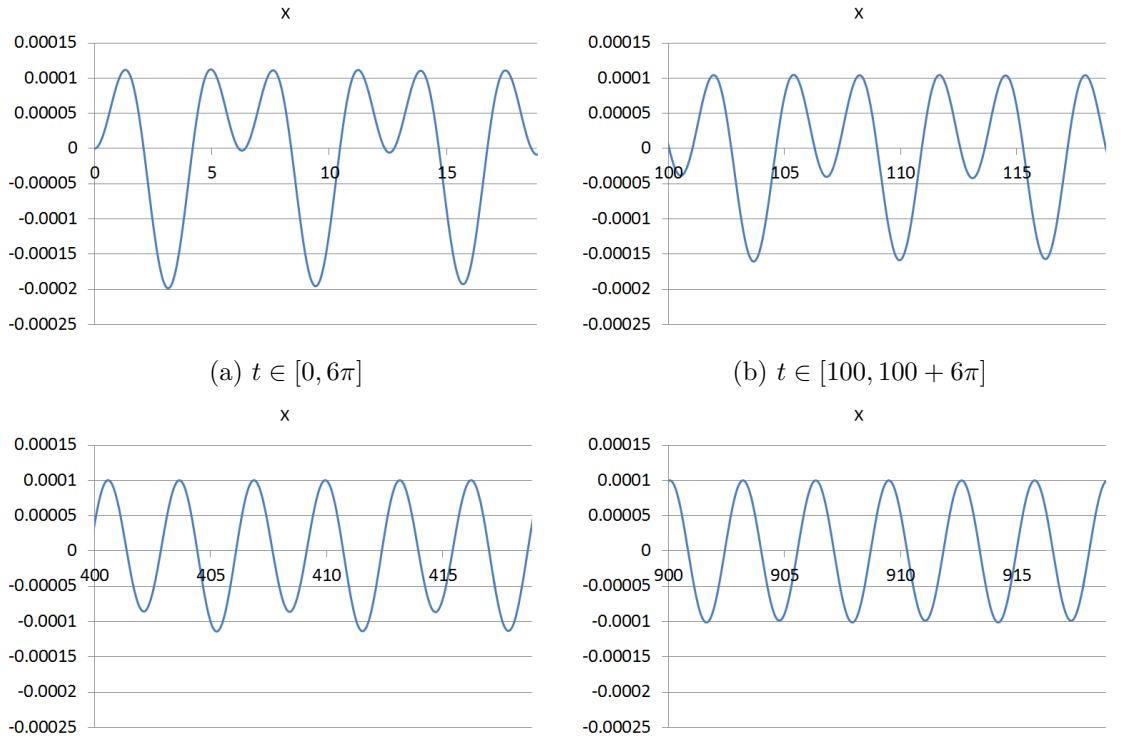
(f) Oscilador sobreamortiguado ($\beta = 3$), espacio de fases

Figura 2.11: Oscilador amortiguado sin término forzante($\Gamma = 0$). En ambos casos, $t_\Delta = 0.0001$ T, t_{cout}= 0.02 T, x_0= 2 , $\dot{x}_0 = 0$, $\alpha = 1$, Γ= 0



(a) $\beta = 0.1$ (b) $\beta = 0.05$

Figura 2.12: Resonancia. Se muestra el valor de x (línea continua) en función del tiempo. En todos los casos, $t_\Delta = 0.0001$ T, $t_{tSimul}= 30$ T, $t_{cout}= 0.02$ T , $x_0= 0$, $\dot{x}_0 = 0$, $\alpha = 1$, $\Gamma= 0.1$, $\omega_D= 1$.



(a) $t \in [0, 6\pi]$ (b) $t \in [100, 100 + 6\pi]$

(c) $t \in [400, 400 + 6\pi]$ (d) $t \in [900, 900 + 6\pi]$, ya en el régimen estacionario

Figura 2.13: Influencia del término forzante. Se muestra el valor de x (línea continua) en función del tiempo. Los datos de la simulación son: $t_\Delta = 0.0001$ T, $t_{tSimul}= 500$ T, $t_{cout}= 0.02$ T , $x_0= 0$, $\dot{x}_0 = 0$, $\alpha = 1$, $\beta = 0.01$, $\Gamma= 0.0003$, $\omega_D= 2$

2.9.8. Solución del oscilador armónico

Podemos distinguir distintos casos, caracterizados por los valores de los coeficientes α, β, Γ.

2.9.9. Oscilador armónico sin pérdidas ($\beta = \Gamma = 0$)

La energía del oscilador E es constante, ya que no hay disipación ($\beta = 0$) ni aporte ($\Gamma = 0$). Independientemente del estado inicial $(x_0, \dot{x}_0)$, pasado un tiempo el oscilador vuelve a encontrarse en la misma posición y con la misma velocidad: el movimiento es periódico.

Más concretamente, x es una función sinusoidal, cuya frecuencia es la raíz cuadrada de α y cuyos coeficientes vienen determinados por las condiciones iniciales:

$$x = x_M \,\text{seno}\,(\omega_0\, t + \varphi_0)\,, \quad \text{con} \tag{2.25}$$
$$\omega_0 = \sqrt{\alpha}\ ,\ \tan(\varphi_0) = \frac{\sqrt{\alpha}\, x_0}{\dot{x}_0}\ ,\ x_M = \sqrt{x_0^2 + \frac{\dot{x}_0^2}{\alpha}}$$

Hacemos notar que si $x_M = 0$, el ángulo de fase está indefinido; en ese caso, le podemos asignar un valor arbitrario $\varphi_0 = 0$. En general, φ_0 se calcula como el arco tangente de $\frac{\sqrt{\alpha}\, x_0}{\dot{x}_0}$.

Dado que la función `atan` en C/C++ devuelve un ángulo en el intervalo $[-\pi/2, \pi/2]$, si $\dot{x}_0 < 0$ es necesario añadir π al valor devuelto: $\varphi_0 = \text{atan}\left(\frac{\sqrt{\alpha}\, x_0}{\dot{x}_0}\right) + \pi$.

2.9.10. Oscilador armónico amortiguado ($\beta \neq 0, \Gamma = 0$)

Debido a la fricción, modelizada por el término $-\beta\dot{x}$, el oscilador va perdiendo energía hasta llegar finalmente al reposo.

Más concretamente,

$$x(t) = e^{-\beta \cdot t/2} g(t) \tag{2.26}$$

En función de la relación entre los parámetros α y β se identifican dos regímenes (subamortiguado y sobreamortiguado), separados por una transición (amortiguamiento crítico):

- Si $\beta < 2\sqrt{\alpha}$ (amortiguamiento débil), $g(t)$ es una función sinusoidal:

$$g(t) = x_M \,\text{seno}\,(\omega\, t + \varphi_0)\,, \quad \text{con} \tag{2.27}$$
$$\omega = \sqrt{\alpha - \beta^2/4}\ ,\ \tan(\varphi_0) = \frac{\omega x_0}{\dot{x}_0 + \beta x_0/2}\ ,\ x_M = \sqrt{\left(\frac{\dot{x}_0 + \beta\ x_0/2}{\omega}\right)^2 + x_0^2}$$

  Hacemos notar que si $x_M = 0$, el ángulo de fase está indefinido; en ese caso, le podemos asignar un valor arbitrario $\varphi_0 = 0$. En general, φ_0 se calcula como el arco tangente de $\frac{\omega x_0}{\dot{x}_0 + \beta x_0/2}$.

  Dado que la función `atan` en C/C++ devuelve un ángulo en el intervalo $[-\pi/2, \pi/2]$, para calcular φ_0 es necesario añadir π al valor devuelto si $\dot{x}_0 + \beta\, x_0/2 < 0$.

- Si $\beta = 2\sqrt{\alpha}$ (amortiguamiento crítico), $g(t)$ es una función lineal.

$$g(t) = A + Bt, \quad \text{con} \tag{2.28}$$
$$A = x_0\ ,\ B = \dot{x}_0 + \beta\, x_0/2$$

- Si $\beta > 2\sqrt{\alpha}$ (socilador sobreamortiguado), $g(t)$ es una combinación de funciones exponenciales de t.

$$g(t) = Ae^{\Omega t} + Be^{-\Omega t}, \quad \text{con} \tag{2.29}$$
$$\Omega = \sqrt{\beta^2/4 - \alpha}\,, \quad A = 0{,}5\left(x_0 + \frac{\dot{x}_0 + \beta x_0/2}{\Omega}\right)\,, \quad B = 0{,}5\left(x_0 - \frac{\dot{x}_0 + \beta x_0/2}{\Omega}\right)$$

2.9.11. Oscilaciones forzadas ($\Gamma \neq 0$)

El oscilador, tras un tiempo transitorio, acaba oscilando a la frecuencia que le impone el término forzante, ω_D. La amplitud depende de la relación entre la frecuencia natural de oscilación ($\sqrt{\alpha}$) y ω_D.

Si la frecuencia del término forzante es igual a la propia del oscilador ($\omega_0 \equiv \sqrt{\alpha} = \omega_D$), tiene lugar el fenómeno llamado *resonancia*: en todo momento, la fuerza *empuja* al oscilador en el sentido adecuado, y la transferencia de potencia es máxima. De no ser por el amortiguamiento, la amplitud de oscilación crecería indefinidamente.

La solución es la suma de dos términos:

- El término estacionario, dado por las ecuaciones siguientes:

$$x = x_M \,\text{seno}\,(\omega_D\, t + \varphi_0)\,, \quad \text{con} \tag{2.30}$$
$$\varphi_0 = \arctan\left(\frac{\alpha - \omega_D^2}{\beta\omega_D}\right)\,, \quad x_M = \frac{\Gamma}{\sqrt{\beta^2\,\omega_D^2 + \left(\alpha - \omega_D^2\right)^2}}$$

- La solución a la ecuación del oscilador amortiguado ($\Gamma = 0$), con los mismos valores de α y β, para las condiciones iniciales ($x_0 - x_M \,\text{seno}\,\varphi_0$, $\dot{x}_0 - x_M\,\omega_d \cos\varphi_0$).

Para tiempos suficientemente largos, el segundo término tiende a cero y sólo queda el estacionario.

Capítulo 3

Subalgoritmos

3.1. Modelización de sistemas biológicos: morfogénesis

3.1.1. Introducción

El artículo de Turing

En 1952 Alan Turing publicó "The Chemical Basis of Morphogenesis", un artículo pionero en la modelización de sistemas biológicos con ayuda del ordenador.

Los seres vivos, en general, no somos isótropos: pese a que en las primeras fases del desarrollo embrionario tenemos simetría esférica, en general desarrollamos formas o patrones como extremidades, tentáculos, formas espirales... En muchas especies (gatos, cebras, leopardos...) la piel no tiene un aspecto homogéneo, sino que su aspecto presenta patrones (manchas o rayas características).

Turing se pregunta por el mecanismo que hace que un ser vivo, que inicialmente tiene un aspecto isótropo y homogéneo, evolucione y deje de serlo. Aventura una idea: ese estado inicial es inestable, y basta una mínima perturbación para que su simetría se rompa.

Propone un modelo en el que considera dos sustancias químicas X e Y, a las que llama morfogenes, cuya evolución se debe a dos tipos de procesos:

- Reacciones químicas que ocurren en el interior de las células (creación, destrucción, transformación de uno en otro).
- Difusión por ósmosis entre una célula y sus vecinas.

En su artículo no especifica la naturaleza de los morfogenes (podrían ser proteínas, hormonas, pigmentos de la piel...).

El sistema partiría de un estado inicial de equilibrio, en que las concentraciones de X e Y son constantes en el tiempo y homogéneas (iguales en todas las células). Si el equilibrio es estable y se produce una pequeña variación en alguna de esas concentraciones, el sistema evolucionará rápidamente para regresar al estado de equilibrio inicial. Pero si el equilibrio es inestable, cualquier fluctuación estadística haría que el sistema evolucionara hacia un estado completamente distinto, que no seguiría la simetría inicial. En esta actividad reproduciremos una de las simulaciones del artículo de Turing. El artículo consideraba una fila de células; nosotros generalizaremos el modelo considerando un array bidimensional de células.

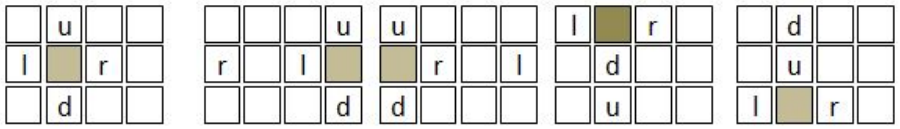


Figura 3.1: Ejemplo de array de 3×4 células

El modelo

Consideramos una formación rectangular de células, cada una de las cuales contiene dos morfogenes: X e Y . Para representar la concentración de estos morfogenes utilizaremos dos arrays de reales, $X = (X_{ij})$, $Y = (Y_{ij})$, donde $0 \leq i < m$ y $0 \leq j < n$, que variarán con el tiempo de acuerdo a las ecuaciones siguientes:

$$\frac{\Delta X_{i,j}}{\Delta t} = f(X_{i,j}, Y_{i,j}) + \mu * (X_{iUp,j} + X_{iDown,j} + X_{i,jRight} + X_{i,jLeft} - 4 * X_{i,j}) \quad (3.1)$$

$$\frac{\Delta Y_{i,j}}{\Delta t} = g(X_{i,j}, Y_{i,j}) + \nu * (Y_{iUp,j} + Y_{iDown,j} + Y_{i,jRight} + Y_{i,jLeft} - 4 * Y_{i,j}) \quad (3.2)$$

donde

$$f(x, y) = (1/320)(-7x^2 - 50xy + 57) \quad (3.3)$$

$$g(x, y) = (1/320)(7x^2 + 50xy - 2y - 55) \quad (3.4)$$

dan cuenta de las reacciones químicas internas de la célula en las que los morfogenes se crean, destruyen y transforman uno en otro, mientras que los términos con los factores constantes μ y ν (llamados coeficientes de difusión) modelizan la difusión de cada morfogen entre una célula y sus cuatro vecinas (las que están, respectivamente, encima y debajo de ella, a su derecha y a su izquierda).

Turing normalizó el tiempo: todos los coeficientes de las ecuaciones se han ajustado para $\delta t = 1$.

Condiciones periódicas de contorno Al igual que Turing, consideraremos que la formación de células tiene condiciones periódicas de contorno: a la derecha de la última columna está la primera; a la izquierda de la primera columna está la última; encima de la primera fila está la última fila, debajo de la última fila está la primera. En el dibujo de la Figura 3.1 consideramos un array de 3x4 células; para la célula sombreada indicamos sus vecinas de arriba (u), abajo (d), derecha (r) e izquierda (l):

Es decir,

- Encima de la fila i está la fila $(i + m - 1) MOD\, m$
- Debajo de la fila i está la fila $(i + 1) MOD\, m$
- A la derecha de la columna j está la columna $(j + 1) MOD\, n$
- A la izquierda de la columna j está la columna $(j + n - 1) MOD\, n$

Equilibrio, estabilidad y homogeneidad y estabilidad El sistema alcanza un estado de equilibrio cuando las derivadas temporales (ecuaciones 3.1 y 3.2) son nulas ($\Delta X_{i,j} = 0 = \Delta Y_{i,j} \forall i, j$): las concentraciones $X_{i,j}$ e $Y_{i,j}$ son constantes en el tiempo. De las ecuaciones 3.1 y 3.2 se deduce inmediatamente que el estado definido por los valores $X_{i,j} = 1 = Y_{i,j} \forall i, j$ es de equilibrio.

Un estado (de equilibrio) es inestable cuando una mínima perturbación (en nuestro caso, una variación en los valores de las concentraciones de los morfogenes) hace que el sistema evolucione a un estado distinto. Por el contrario, el estado (de equilibrio) es estable si tras sufrir una variación evoluciona regresando al mismo estado.

Finalmente, un estado de nuestro modelo es homogéneo si las concentraciones de los morfogenes X e Y son iguales en todas las células; esto es, si $X_{i,j} = k_x$ y $Y_{i,j} = k_y \forall i, j$, donde $k_x, k_y \in \mathbb{R}^+$ son constantes. Claramente, el estado definido por los valores $X_{i,j} = 1 = Y_{i,j} \forall i, j$ es homogéneo. Un estado es inhomogéneo si no es homogéneo.

Caracterización de los estados Al realizar una simulación de la evolución de nuestro modelo será inevitable cometer errores de redondeo relacionados con la precisión de la representación de los números reales. Por ello, consideraremos que el modelo se encuentra en un estado de equilibrio si la última variación que ha sufrido respecto del estado anterior es prácticamente nula; por ejemplo, si

$$\Sigma_{i,j} |\Delta X_{i,j}| + |\Delta Y_{i,j}| < \epsilon \tag{3.5}$$

para un real ϵ suficientemente pequeño. Análogamente, consideraremos que un estado es homogéneo si existen constantes $k_x, k_y \in \mathbb{R}^+$ tales que

$$\begin{aligned} \left|\bar{X} - k_x\right| &< \delta \\ \left|\bar{Y} - k_y\right| &< \delta \end{aligned} \tag{3.6}$$

para un real $\delta > 0$ y la varianza de las componentes de las matrices X e Y son suficientemente pequeñas; esto es, si

$$\begin{aligned} \sigma_x^2 &< \delta \\ \sigma_y^2 &< \delta \end{aligned} \tag{3.7}$$

Recuerda que:

$$\sigma_x^2 = \overline{X^2} - \bar{X}^2 \tag{3.8}$$

donde $\bar{X}$ y $\overline{X^2}$ son, respectivamente, la media de las componentes de X y la media de los cuadrados de las componentes de X (ver el apartado Para saber más al final de esta sección):

$$\overline{X^2} = \frac{1}{m * n} \Sigma_{i,j} X_{i,j}^2 \tag{3.9}$$

$$\bar{X} = \frac{1}{m * n} \Sigma_{i,j} X_{i,j} \tag{3.10}$$

3.1.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo para estudiar si, en una formación rectangular de $m \times n$ células, $m, n < 100$, el estado de equilibrio homogéneo definido por una concentración de los morfogenes X e Y igual a 1 en todas ellas es estable o inestable para diversos valores de los coeficientes de difusión μ y ν.

El algoritmo solicitará al usuario los valores m, n, μ y ν.

El estado del sistema en un instante de tiempo está representado por dos arrays de reales, X e Y, de tamaño mxn, que contienen la concentración de los morfogenes en las células. Para simular que se ha producido una alteración aleatoria en el estado inicial, se inicializarán estos arrays con valores aleatorios próximos a uno (ver subalgoritmo **inicializa**).

Seguidamente (ver subalgoritmo **evoluciona**) dejaremos que el sistema evolucione realizando una serie de iteraciones. En cada una de ellas se calcularán las matrices ΔX y ΔY, que caracterizan la variación del sistema respecto al tiempo según las ecuaciones 3.1 y 3.2, en las que tomaremos $\Delta t = 1$, y actualizaremos las matrices X e Y ($X \leftarrow X + \Delta X$, $Y \leftarrow Y + \Delta Y$). Hay que tener en cuenta que una concentración NO PUEDE SER NEGATIVA. Por tanto, si en la simulación se obtuviera que para alguna célula $\Delta X_{ij} < -X_{ij}$, reasignaremos el valor de ΔX_{ij} para que la concentración actualizada X_{ij} sea 0, y procedemos de modo análogo con Y. El bucle finaliza si el sistema alcanza de nuevo un estado de equilibrio (ver ecuación 3.5) o bien tras realizar un número máximo de iteraciones MAX_ITER, en cuyo caso no alcanza el equilibrio. En cualquier caso, el programa mostrará los valores de los coeficientes μ y ν, si se ha llegado o no a un estado de equilibrio, los valores de las medias y varianzas de las concentraciones de los morfogenes $(\bar{X}, \bar{Y}, \sigma_X^2, \sigma_Y^2)$ y si, por tanto, el estado inicial es o no estable (ver ecuaciones 3.6 y 3.7 tomando $k_x = 1 = k_y$).

Tras esto, el programa preguntará al usuario si desea realizar una nueva simulación con otros valores de los coeficientes μ y ν.

Se puede suponer que los valores MAX_ITER, ϵ y δ son constantes (por ejemplo, 10^5, 10^{-8} y 10^{-6}, respectivamente). El programa debe contener, al menos, los subalgoritmos siguientes:

f(x, y) y **g(x,y)** según las ecuaciones 3.3 y 3.4

inicializa: inicializa las componentes de las m primeras filas y n primeras columnas de un array, $m, n \leq 100$, con valores aleatorios de la forma $1 + x_{ij}$, donde $|x_{ij}| \leq 10^{-3}$ (ver la ayuda sobre generación de números aleatorios).

analizaArray: dado un array de $m \times n$ componentes, $m, n \leq 100$, devuelve el valor medio y la varianza de sus componentes.

iteracion: dados los coeficientes de difusión, μ y ν, y los arrays X e Y de tamaño $m \times n$, $m, n \leq 100$, que definen el estado del sistema, los actualiza conforme a las ecuaciones 3.1 y 3.2. Además determina si el estado es de equilibrio (ecuación 3.5). Recuerda que la concentración de los morfogenes en una célula no puede ser negativa.

evoluciona: dados los coeficientes de difusión μ y ν, determina si un estado inicial próximo al equilibrio homogéneo dado por $X_{i,j} = 1 = Y_{i,j}$ evolucionará o no a un estado de equilibrio. En cualquier caso el subalgoritmo devuelve, además, las medias y varianzas $(\bar{X}, \bar{Y}, \sigma_X^2, \sigma_Y^2)$ de las concentraciones de los morfogenes en el estado final que se alcance.

3.1.3. Ejemplos de ejecución

En la figura siguiente mostramos los resultados para algunos pares (μ, ν):

```
Array de 20x20 celulas

mu   nu    Equ   xMedio   varianzaX   yMedio  varianzaY   Estable
0.05 0.02  si     0.82388  0.11895      1.4797  0.32679        no
0.05 0.021 si     0.83967  0.12259      1.4511  0.32362        no
0.05 0.022 si     0.85459  0.10914      1.3865  0.25606        no
0.05 0.023 si     0.85702  0.12642      1.4015  0.27278        no
0.05 0.024 si     0.87412  0.10631      1.3317  0.20999        no
0.05 0.025 si     1        8.8818e-016  1       4.4409e-016    si
0.05 0.026 si     1        1.5543e-016  1      -7.7716e-016    si
0.05 0.027 si     1        4.4409e-016  1       1.3323e-015    si
0.05 0.028 si     1       -2.2204e-016  1       3.3307e-016    si
0.05 0.029 si     1       -6.6613e-016  1      -5.5511e-016    si
0.05 0.03   si     1       -2.2204e-016  1       6.6613e-016    si
```

3.1.4. Ayuda

Inicialización de las concentraciones: generación de números aleatorios

Sugerimos la siguiente función. En ella utilizamos la función rand(), que cada vez que se invoca devuelve un número entero pseudo-aleatorio en el intervalo $[0, RAND\_MAX]$. Para utilizar rand() y $RAND\_MAX$ deberás incluir en tu programa el fichero de cabecera <stdlib.h>.

```
double aleatorio(double delta)
{

    // crea un número real aleatorio r entre 0 y 1
    double r = (1.0 * rand())/$RAND\_MAX$;

   // lo transforma en otro número aleatorio entre -1 y 1
    r = 2*r-1;

// lo transforma en otro número aleatorio entre -delta y delta, y lo devuelve
    return delta*r;

}
```

Conviene inicializar el generador de números aleatorios utilizando la función srand. De este modo, cada vez que ejecutes el programa se obtendrá una secuencia distinta de números. Para ello, al principio del programa principal incluye la sentencia:

srand (time(NULL));

El generador de números aleatorios tomará como semilla un valor dependiente de la fecha y hora en que se invoca. Para utilizar la función time deberás incluir el fichero de cabecera <time.h>.

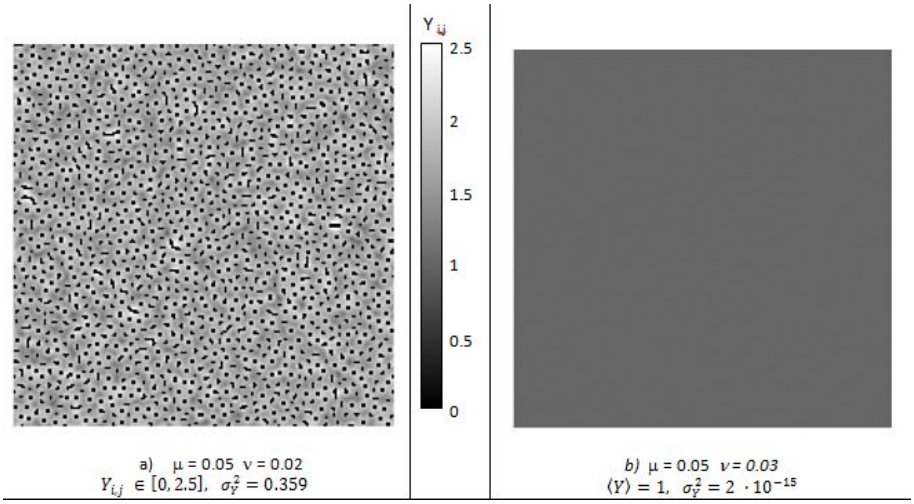


Figura 3.2: Concentración de Y en los estados finales de equilibrio para un array de 200×200 células.

3.1.5. Para saber más

Caracterización del estado final Nos interesa distinguir entre dos tipos de estado de equilibrio: homogéneo (todas las células tienen exactamente la misma concentración de X, Y) e inhomogéneo (las concentraciones de X, Y varían de célula a célula). En el caso del equilibrio inhomogéneo, deseamos una medida de esa inhomogeneidad (los valores de la concentración en las distintas células, ¿están muy dispersos o se parecen mucho al valor medio?).

Hacemos notar que no es viable mostrar por pantalla todos los valores $X_{i,j}$, $Y_{i,j}$ (para un array 100x100 estaríamos hablando de . . . ¡ 20000 valores!).

Una forma de saber cómo es el estado de equilibrio es mostrar gráficamente la concentración en todas las células, como se ve en la Figura 3.2. En la Figura 3.2(a) el morfogen Y se concentra en algunas células, mientras que está ausente de otras; en la Figura 3.2(b), todas las células tienen la misma concentración de Y.

Más adelante aprenderemos a guardar los resultados de un programa en un fichero, y a representarlos gráficamente. Mientras tanto, nos interesa medir de algún modo en qué caso estamos. Para caracterizar el estado de equilibrio proponemos utilizar la función varianza, definida como:

$$\sigma_X^2 = \frac{1}{m*n}\Sigma_{i,j}(X_{i,j} - \bar{X})^2 \tag{3.11}$$

Intuitivamente: si todas las células tienen la misma concentración de X (y por tanto $X_{i,j} = \bar{X}$), entonces $\sigma_X^2 = 0$. Cuanto mayor sea la dispersión de los valores de X, mayor es σ_X^2.

No es complicado ver que de la definición anterior se deduce la expresión siguiente, más sencilla de computar:

$$\begin{aligned}\sigma_X^2 = \frac{1}{m*n}\Sigma_{i,j}(X_{i,j} - \bar{X})^2 = \frac{1}{m*n}\Sigma_{i,j}(X_{i,j} - \bar{X} + \bar{X}^2 - 2X_{i,j}\bar{X})^2 = \\ \frac{1}{m*n}\left(\Sigma_{i,j}X_{i,j}^2\right) - 2\bar{X} + \bar{X}^2 - 2\bar{X}\frac{1}{m*n}\Sigma_{i,j}X_{i,j} = \overline{X^2} - \bar{X}^2\end{aligned} \tag{3.12}$$

En este último paso hemos utilizado las definiciones

$$\overline{X^2} = \frac{1}{m*n}\Sigma_{i,j}X_{i,j}^2 \tag{3.13}$$

$$\bar{X} = \frac{1}{m*n}\Sigma_{i,j}X_{i,j} \tag{3.14}$$

¿Es realista el modelo de Turing?

Turing era perfectamente consciente de las limitaciones de su modelo. En su artículo lo presenta como "manejable desde el punto de vista matemático, aunque inusual desde el punto de vista biológico". Más aún, .ˢᵉte modelo es la simplificación de una idealización, y por tanto una falsificación". El objetivo no era explicar el comportamiento de un sistema realista, sino apuntar a procesos que rompen la simetría inicial de un sistema.

En Física encontraremos igualmente modelos muy idealizados, pero que de algún modo capturan lo esencial de un fenómeno (por ejemplo, el modelo de Ising que modeliza el comportamiento de los materiales ferromagnéticos). Por otro lado, la rotura de simetría aparece en muchos modelos físicos (por ejemplo, es un mecanismo frecuente en Física de partículas).

Y, sin embargo, sí que existen en la Naturaleza sistemas que siguen el mecanismo propuesto por Turing. En [22] se identifican dos sustancias que se comportan como los morfogenes de Turing: las moléculas, WNT y DKK, que regulan el crecimiento de pelo en ratones.

3.2. Frecuencias de ondas: la escala musical

3.2.1. Introducción

Cada nota musical, como cualquier onda, se caracteriza por una frecuencia. Por ejemplo, la frecuencia de afinación estándar es, según la ISO 16:1975, de 440 Hz (440 vibraciones por segundo), que corresponde a la primera nota 'La' a la derecha del 'Do' central del piano.

El intervalo entre dos notas es la distancia entre ellas. A efectos de esta práctica, definiremos el *intervalo* entre dos notas como el cociente entre sus frecuencias. Un *semitono* es el intervalo entre dos notas consecutivas (correspondientes a dos teclas consecutivas del piano, o a dos trastes seguidos de la guitarra). Una *octava* es el intervalo entre dos notas cuyas frecuencias tienen una relación $2:1$. En nuestro sistema musical, una octava está formada por 12 semitonos: en una octava hay doce notas distintas, que podemos caracterizar mediante un número entero i, según se muestra a continuación.

Nota	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si	Do
i	0	1	2	3	4	5	6	7	8	9	10	11	12

Dentro de una octava, definimos *frecuencia relativa* de una nota al cociente entre su frecuencia y la del 'Do'.

A la hora de afinar un instrumento, hay que tener en cuenta dos intereses que entran en conflicto:

- Sería deseable que todos los semitonos fueran iguales: eso permitiría que una melodía siguiera siendo reconocible al cambiar de tonalidad (esto es, que una melodía, al desplazar todas las notas la misma cantidad de semitonos, siguiera sonando igual). Dado que una octava consta de 12 semitonos, un semitono valdría $\sqrt[12]{2} = 2^{1/12}$. En este principio se basa la escala *temperada*. La *frecuencia relativa* de la nota i en la escala *temperada* es $f_{\mathrm{RT}}(i) = 2^{i/12}$.
- Si el cociente de frecuencias de dos notas es un número racional sencillo, el oído percibe la combinación de ambas notas como algo agradable. Sería deseable que los semitonos fueran números racionales (cocientes n/d con n y d enteros positivos y lo más pequeños posible). En este principio se basa la *escala diatónica*. Las *frecuencias relativas* en la escala *diatónica*, $f_{\mathrm{RD}}(i)$, son números racionales sencillos. Como contrapartida, no todos los semitonos son iguales.

En las siguientes tablas mostramos las frecuencias relativas de las distintas notas de una octava en las escalas temperada y diatónica:

Nota	Do	Do#	Re	Re#	Mi	Fa	Fa#
i	0	1	2	3	4	5	6
$f_{\mathrm{RT}}(i) = 2^{i/12}$	1	1,059	1,122	1,189	1,260	1,335	1,414
$f_{\mathrm{RD}}(i)$	1	16/15	9/8	6/5	5/4	4/3	45/32

Nota	Sol	Sol#	La	La#	Si	Do	
i	7	8	9	10	11	12	
$f_{\mathrm{RT}}(i) = 2^{i/12}$	1,498	1,587	1,682	1,782	1,888	2	
$f_{\mathrm{RD}}(i)$	3/2	8/5	5/3	16/9	15/8	2	

3.2.2. Descripción de la tarea

El objetivo último de esta tarea es diseñar una escala musical que satisfaga los dos requisitos citados previamente. Se trata de asignar a cada nota una frecuencia relativa que venga dada por el número racional más sencillo posible que se aproxime razonablemente a la de la escala temperada, $f_{\mathrm{RT}}(i) = 2^{i/12}$. Como resultado (no sorprendente, por otro lado) obtendremos una escala que, a excepción de la nota 'Fa#', coincide con la diatónica.

Diseña y programa en C/C++ los subalgoritmos que se indican a continuación.

fRT

Dado un entero i que representa una nota, devuelve su frecuencia relativa según la escala temperada, $f_{\mathrm{RT}}(i) = 2^{i/12}$.

racionalAproximado

Dado un número real $f \geq 1$, devuelve dos enteros, n y d, que se calculan tal como se indica a continuación.

El subalgoritmo busca el racional $q = n/d$ de menor denominador, con $d \leq 100$, que se aproxima a f con una precisión de al menos un 1 %. Es decir, que satisfaga la condición:

$$\left|\frac{q}{f} - 1\right| < 10^{-2}$$

Observación: Hay que tener en cuenta que gracias a que $f \geq 1$ siempre puede encontrarse el número q buscado con $d \leq 100$.

frecuencias

Dado un entero no negativo i, que caracteriza una nota musical, devuelve dos enteros, n y d, y dos reales, f_{A} y f_{T}.

- n y d son el numerador y denominador del número racional n/d más sencillo que se aproxime, con un 1 % de precisión, a $f_{\mathrm{RT}}(i)$, es decir, son los valores devueltos por el subalgoritmo `fRT`.
- f_{A} y f_{T} son las frecuencias absolutas de la nota, expresadas en hertzios, según nuestra aproximación y la escala temperada, sabiendo que la frecuencia de la nota 'La' es 440 Hz. Es decir, para $i = 9$ se tiene que $f_{\mathrm{A}} = f_{\mathrm{T}} = 440$.

El subalgoritmo `frecuencias` debe utilizar los subalgoritmos `fRT` y `racionalAproximado`.

Programa Principal

Utilizando los subalgoritmos anteriores muestra para cada nota:

- la frecuencia absoluta (en hertzios) correspondiente a la escala temperada y a nuestra escala aproximada
- la frecuencia relativa temperada y la obtenida en nuestra aproximación.

3.2.3. Ejemplo de ejecución

```
Nota    fT (Hz)       fA (Hz)       fRT           fRA
0       261.626       264           1             1/1 = 1
1       277.183       281.6         1.05946       16/15 = 1.06667
2       293.665       297           1.12246       9/8 = 1.125
3       311.127       316.8         1.18921       6/5 = 1.2
4       329.628       330           1.25992       5/4 = 1.25
5       349.228       352           1.33484       4/3 = 1.33333
6       369.994       374           1.41421       17/12 = 1.41667
7       391.995       396           1.49831       3/2 = 1.5
8       415.305       422.4         1.5874        8/5 = 1.6
9       440           440           1.68179       5/3 = 1.66667
10      466.164       469.333       1.7818        16/9 = 1.77778
11      493.883       495           1.88775       15/8 = 1.875
12      523.251       528           2             2/1 = 2
```

3.2.4. Para saber más...

Se puede obtener más información sobre acústica en general y escalas sonoras en los libros [3] y [25]. Para obtener más información sobre ondas sonoras en el Universo, se puede consultar el libro [21].

Hay diversas páginas web con información sobre ondas de sonido. A modo de ejemplo, se pueden consultar las páginas 'Ondas de sonido en torno al agujero negro de Perseo' [5] y 'Acústica del big-bang: sonidos de un universo recién nacido' [15].

3.3. Buscando un camino

3.3.1. Introducción

En geometría, la línea de longitud mínima contenida en una superficie entre dos puntos de la misma se llama *geodésica*. Las líneas geodésicas en un plano son, obviamente, las líneas rectas, mientras que en una esfera son los arcos de círculos máximos (los que se obtienen al cortar la esfera con un plano que pasa por su centro).

Estos caminos más cortos no son siempre los más deseables. Por ejemplo, al diseñar una travesía de montaña preferiremos un camino un poco más largo, siempre que tenga un desnivel menor. Este es el problema que proponemos resolver en esta práctica.

Supongamos que el paisaje de montaña que queremos atravesar está modelizado por la superficie

$$z(x,y) = \cos\left(\frac{3}{2}\pi x\right) + \frac{\cos^2(\pi y)}{1+x^2} + y \ , \tag{3.15}$$

donde $z(x,y)$ es la altura asociada al punto (x,y) del plano.

En la figura 3.3 puedes ver una representación de este relieve. Observa que existe un 'pico' en el punto de coordenadas $(0,0)$, dos 'collados' en $(0,0{,}5)$ y $(0,-0{,}5)$ y cuatro 'pozos' cuyos fondos están, aproximadamente, en los puntos $(\pm 0{,}75, \pm 0{,}5)$. El objetivo es encontrar un camino con desnivel mínimo, que evite estos obstáculos, entre dos puntos de la superficie correspondientes a las coordenadas $(x_0,0)$ y $(x_f,0)$ del plano.

Para simplificar la tarea consideraremos únicamente caminos en los que la coordenada y sea una función polinomial de x. Concretamente, tomaremos

$$y = (x-x_0)(x-x_f)(a_0 + a_1 x + a_2 x^2) \ ,$$

lo que asegura que todos los caminos pasan por los puntos $(x_0,0)$ y $(x_f,0)$. Inicialmente los coeficientes a_0, a_1 y a_2 valdrán 0; esto es, estaremos tomando la línea recta entre los puntos inicial y final de nuestra ruta. En cada iteración modificaremos aleatoriamente los valores de los coeficientes, dando por bueno el camino que definen si tiene menos desnivel acumulado que el actual. El programa terminará tras realizar un cierto número de iteraciones sin mejorar la solución actual.

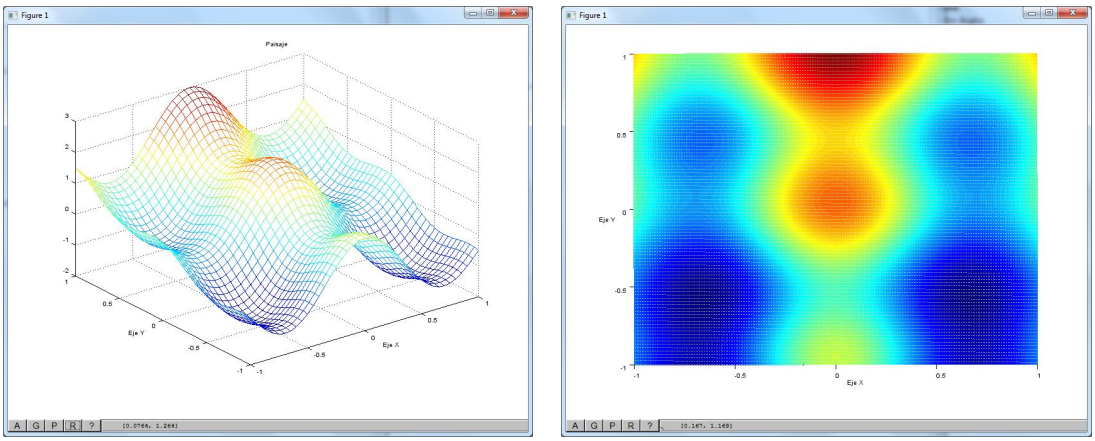


Figura 3.3: Representación de la superficie con ecuación 3.15. A la derecha se muestra su proyección sobre el plano horizontal: cuanto más vivo es el color, mayor es la altura $z(x,y)$.

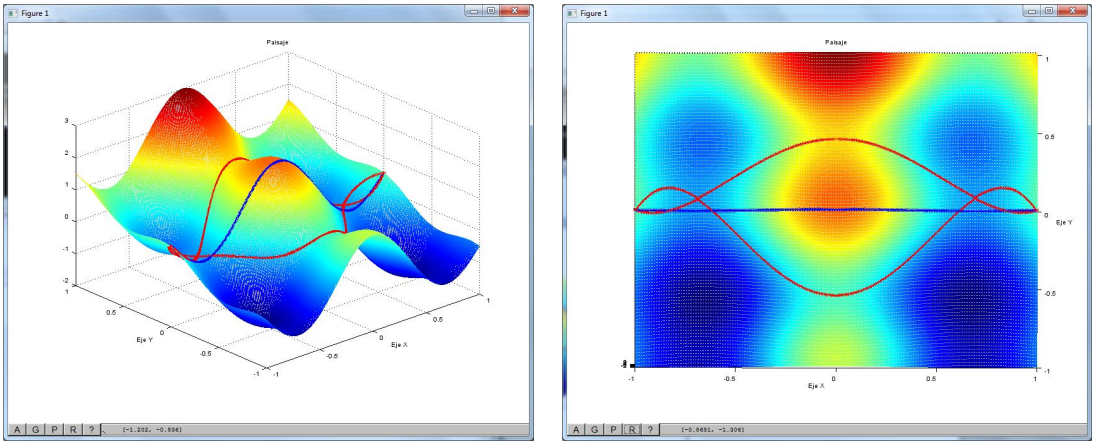


Figura 3.4: Trayectorias que conectan los puntos $(-1, 0)$ y $(1, 0)$ y sus proyecciones en el plano XY. La azul es la trayectoria inicial considerada en el algoritmo; las rojas son los dos posibles resultados.

Para calcular tanto el desnivel acumulado como la longitud de estos caminos, consideraremos $n+1$ puntos $x_i \in [x_0, x_f]$ homogéneamente espaciados y aproximaremos el camino por la línea poligonal formada por los segmentos de extremos (x_i, y_i, z_i) y $(x_{i+1}, y_{i+1}, z_{i+1})$. Así, tomaremos la suma de las longitudes de estos segmentos como una aproximación de la longitud del camino y la suma de los valores absolutos $|z_i - z_{i+1}|$ como una aproximación de su desnivel acumulado.

Crítica del método propuesto

El método propuesto sigue un algoritmo secillo, que es relativamente fácil de programar, y, por supuesto, en general encontrará, como solución, un camino mejor que la línea recta inicial. Sin embargo, esta solución sólo es óptima localmente: en el mejor de los casos, el camino encontrado es mejor que cualquier otro próximo a él, pero no podemos garantizar que la solución sea la mejor posible. Si $x_0 = -1$ y $x_f = 1$ hay dos caminos finales posibles, mostrados en la figura 3.4, y tenemos la misma probabilidad de que el programa acabe en cualquiera de ellos. Cada uno pasa por un collado, pero uno tiene menos desnivel acumulado. Ello se debe a que nuestro algoritmo es avaricioso (*greedy algorithm*): sólo da por buenas modificaciones que mejoren la solución actual (que hagan disminuir el desnivel acumulado). Supongamos, por ejemplo, que partimos de una trayectoria próxima al collado situado en el punto de coordenadas $(0, 0{,}5)$; para acabar encontrando el otro collado (que al tener menos altura arroja un desnivel acumulado menor) sería necesario antes poder remontar el pico situado en $(0, 0)$. Nuestra solución no permite hacer eso.

Por otro lado, es necesario cuantificar cómo de pequeñas deben ser las sucesivas modificaciones de la trayectoria inicial. Si sólo son posibles modificaciones infinitesimales es necesario un alto número de iteraciones para mejorarla sustancialmente. Si permitimos modificaciones de cualquier magnitud, la aplastante mayoría de ellas dará lugar a trayectorias disparatadas (como ir de Estós a Viadós pasando por Berlín) y la probabilidad de encontrar una buena solución será despreciable.

3.3.2. Descripción de la tarea

Diseña y programa en C/C++ los siguientes subalgoritmos.

numeroAleatorio

Dado un real, Δ, devuelve un número real aleatorio en el intervalo $[-\Delta, \Delta]$. (Consulta la sección de Ayuda de la Tarea 3.1).

calculaY

Dados un punto x, los coeficientes a_0, a_1, a_2 y las abcisas x_0, x_f, devuelve el valor del polinomio $(x - x_0)(x - x_f)(a_0 + a_1 x + a_2 x^2)$.

altura

Dadas las coordenadas x, y de un punto, devuelve su altura $z = \cos\left(\frac{3}{2}\pi x\right) + \frac{\cos^2(\pi y)}{1+x^2} + y$ en la superficie que estamos considerando.

analizaCamino

Dados cinco reales, a_0, a_1, a_2 (que parametrizan una trayectoria), x_0, x_f (las abcisas de los puntos inicial y final del camino) y un entero n, calcula y devuelve al algoritmo que lo invoca la *longitud* y el *desnivel acumulado* de esa trayectoria.

Para calcular la longitud y el desnivel acumulado de la trayectoria, la aproximaremos como una sucesión de n segmentos, caracterizados por un entero $i \in [0, n-1]$. Los extremos del segmento i-ésimo son (x_i, y_i, z_i) y $(x_{i+1}, y_{i+1}, z_{i+1})$, donde

- $x_i = x_0 + i \cdot \Delta x$, con $\Delta x = (x_f - x_0)/n$.
- y_i es el valor de y correspondiente a x_i; para calcularlo se utiliza el subalgoritmo `calculaY`.
- z_i es la `altura` correspondiente al punto de coordenadas (x_i, y_i).

Observa que el extremo final de un segmento coincide con el inicial del siguiente.

La longitud del segmento i-ésimo es

$$\Delta l_i = \sqrt{(\Delta x)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}$$

y su desnivel

$$\Delta d_i = |z_{i+1} - z_i| \ .$$

Por tanto, la longitud de la trayectoria y su desnivel acumulado son $\sum_{i=0}^{n-1} \Delta l_i$ y $\sum_{i=0}^{n-1} \Delta d_i$, respectivamente.

optimizaCamino

Dados seis reales, a_0, a_1, a_2 (que parametrizan una trayectoria), x_0, x_f (las abcisas de los puntos inicial y final del camino), Δ (que acota el intervalo de números aleatorios a considerar) y dos enteros, n (el número de segmentos que aproximan una trayectoria) y m (el número máximo de iteraciones sin mejora de la solución actual), modifica la trayectoria de acuerdo al siguiente algoritmo:

i) Los coeficientes a_0, a_1 y a_2 caracterizarán en todo momento la *mejor trayectoria*, t, encontrada hasta el momento. Inicialmente t está definida por los valores de entrada de estos parámetros.

ii) Genera una nueva trayectoria, t^*, caracterizada por coeficientes $a_i^* = a_i + \delta_i$, $0 \leq i \leq 2$, que se obtienen sumando un número aleatorio distinto $\delta_i \in [-\Delta, \Delta]$ a cada uno de los coeficientes, a_i, de la mejor trayectoria t.

iii) Si el desnivel acumulado de t^* es menor que el de t, t^* es una trayectoria mejor, por lo que hay que cambiar los coeficientes: $a_i \leftarrow a_i^*$. En caso contrario, el intento ha sido fallido y los coeficientes a_i no cambian de valor.

iv) Los pasos (ii) y (iii) se repiten hasta haber realizado m intentos fallidos consecutivos sin mejorar la trayectoria.

El subalgoritmo devuelve los valores finales de a_0, a_1, a_2, así como la longitud y el desnivel acumulado de la trayectoria final.

Escribe un programa principal que empiece solicitando al usuario los valores de Δ, las abcisas de los puntos inicial y final de los caminos, el número de segmentos que se usarán para aproximarlos y el número máximo de intentos fallidos consecutivos sin mejora de la trayectoria que se permiten. A continuación ejecutará 10 iteraciones. En cada una de ellas los coeficientes a_0, a_1, a_2 se inicializan a cero, se invoca al subalgoritmo `optimizaCamino` y se muestran en pantalla sus resultados.

3.3.3. Ejemplos de ejecución

```
Introduzca el valor de delta: 0.1
Introduzca los valores inicial y final de x: -1 1
Introduzca el numero de intervalos: 200
Introduzca el numero maximo de intentos fallidos 1000

L= 5.71636   desnivel= 5.06453   a0= 0.448291    a1= 0.00216315    a2= -0.626702
L= 4.41532   desnivel= 3.06283   a0= -0.545648   a1= 0.00156102    a2= 1.45637
L= 4.47168   desnivel= 3.06828   a0= -0.54551    a1= 0.000542619   a2= 1.45623
L= 4.42479   desnivel= 3.06158   a0= -0.545154   a1= 0.00215979    a2= 1.45154
L= 4.43272   desnivel= 3.05985   a0= -0.545392   a1= 0.00166845    a2= 1.4553
L= 4.41076   desnivel= 3.06412   a0= -0.545144   a1= 0.00254219    a2= 1.45287
L= 4.46303   desnivel= 3.06426   a0= -0.54537    a1= 0.00155431    a2= 1.45572
L= 5.71675   desnivel= 5.06606   a0= 0.448463    a1= 0.00286752    a2= -0.625725
L= 4.43213   desnivel= 3.05951   a0= -0.545324   a1= 0.00214637    a2= 1.4541
L= 4.44514   desnivel= 3.05875   a0= -0.54543    a1= 0.000177007   a2= 1.45502
```

Como puedes observar, el algoritmo opta básicamente por dos grupos de caminos:

- uno agrupa caminos un poco más largos (en torno a 5,72) y con un desnivel acumulado mayor (en torno a 5,06). Son caminos que pasan por el collado más alto $(0, 0{,}5)$.
- el otro agrupa caminos algo más cortos (en torno a 4,43) y con un desnivel acumulado menor (en torno a 3,07). Son caminos que pasan por el collado menos alto $(0, -0{,}5)$.

En la figura 3.4 se muestra, en rojo, un camino típico de cada uno de estos dos grupos.

3.4. Proyecto RSA: Algoritmos básicos

3.4.1. Introducción

En esta tarea nos familiarizaremos con los algoritmos básicos del sistema criptográfico RSA, que permite de modo sencillo encriptar mensajes, y hace que desencriptarlos sea prácticamente inviable. Antes de la descripción de la tarea presentaremos las ideas principales del sistema RSA. En los apéndices A y B se proponen, respectivamente, una introdución más detallada al sistema y los resultados matemáticos en que se basa.

Compo aplicación, en esta tarea desencriptaremos mensajes (en este caso, códigos numéricos).

El escenario

Imaginemos que dos personas, Alicia y Belén, se comunican entre sí utilizando la criptografía RSA. Para ello utilizan una clave pública (un par de enteros positivos, n, e). Sabemos, además, que n es producto de dos primos distintos (pero no sabemos cuáles son). Esa clave es de dominio público: todo el mundo la conoce. Belén desea comunicarle a Alicia un número secreto M (el PIN de su tarjeta de crédito, la combinación del candado de su taquilla...). Para encriptar M realiza la siguiente operación: $M_{encriptado} = M^e \,\%\, n$ (observa que en la operación sólo se usa la clave pública: todo el mundo puede encriptar mensajes para enviárselos a Alicia). Para desencriptar $M_{encriptado}$ y obtener M es necesario disponer de la clave privada d, que (supuestamente) sólo conoce Alicia.

Nosotros hemos interceptado el mensaje encriptado, $M_{encriptado}$, y queremos averiguar cuál es el número secreto de Belén, M. Para ello necesitaremos encontrar la clave privada d. El algoritmo de desencriptación consta de dos pasos:

i) Dada la clave pública (n, e), hay que hallar la clave privada d. Para ello es preciso:

   a) Calcular la descomposición factorial de n: obtener los dos primos, p, q, cuyo producto vale n.

   b) Calcular la función de Euler de n: $\varphi(n) = (p-1)\cdot(q-1)$.

   c) Calcular el inverso de e módulo $\varphi(n)$: el número d tal que $e \cdot d \,\%\, \varphi(n) = 1$. Ese inverso, d, es la clave privada.

ii) Se calcula $(M_{encriptado})^d \,\%n$. El valor obtenido es, precisamente, M.

El requisito de eficiencia

A lo largo de la tarea nos familiarizaremos con el esquema RSA y con sus principales algoritmos. A la hora de diseñar estos hay que tener en cuenta que deben ser eficientes. Aunque en esta tarea nos limitaremos a trabajar con enteros "pequeños" (de menos de 10 cifras), el espacio en memoria y el tiempo requerido por los algoritmos deben seguir siendo razonables cuando trabajemos con enteros de varios cientos de dígitos (cosa que haremos en las siguientes tareas): Se debe poder seguir encriptando mensajes fácilmente aun en ese caso.

Para comprender mejor la necesidad de hacer un uso eficiente de nuestros recursos (memoria y tiempo de CPU), haremos un cálculo rápido. Uno de los algoritmos que diseñaremos es **potenciaModulo** que, dados tres enteros, b, e y n, devuelve el valor $b^e \,\%\, n$.

En las próximas tareas trabajaremos con enteros b, e, n del orden de 10^{200} (tendrán, por tanto, 200 dígitos).

Una solución naïf consiste en calcular b^e siguiendo el algoritmo $b^e = \prod_{i=1}^{e} b = b \cdot b \cdot b ... \cdot b$ (multiplicar b e veces) y, a continuación, calcular el resto $b^e \,\%\, n$. Veremos enseguida que esa solución es inviable (necesitaremos, pues, algoritmos más eficientes).

- ¿Cuántas operaciones es necesario realizar para calcular la potencia b^e?
  El algoritmo naïf consiste en calcular $b \cdot b \cdot b ... \cdot b$ e veces: requiere 10^{200} multiplicaciones.

- ¿Cómo es de grande ese resultado, b^e? Es decir, ¿cuántos dígitos tiene?
  b^e es del orden de $(10^{200})^{10^{200}} = 10^{2 \cdot 10^{202}}$. En otras palabras, se trata de un número de $2 \cdot 10^{202}$ dígitos.

Para entender la magnitud de esos números es útil recurrir a referencias astronómicas. Desde el Big Bang han transcurrido unos 14000 millones de años, es decir, unos $5 \cdot 10^{17}$ segundos. En verano de 2015 se publicó que el presidente Obama había impulsado la construcción del que será el ordenador más potente del mundo [1]: está previsto que entre en funcionamiento en 2022, y podrá realizar 10^{18} operaciones por segundo [1]. Si el superordenador de Obama hubiera comenzado a trabajar en el mismo instante del Big Bang, le habría dado tiempo de hacer $5 \cdot 10^{17}$ segundos $\times$ 10^{18} operaciones/segundo $= 5 \cdot 10^{35}$ operaciones, una fracción irrisoria de las 10^{200} que requiere el algoritmo naïf.

El radio del Universo observable es de unos $4{,}4 \cdot 10^{26}$ metros. El radio de un átomo de hidrógeno es de unos $5 \cdot 10^{-11}$ metros (radio de Bohr), de modo que en el Universo observable cabrían unos $(4{,}4 \cdot 10^{26} / 5 \cdot 10^{-11})^3 \simeq 10^{111}$ átomos de hidrógeno densamente empaquetados. Es decir, si en nuestra RAM cada dígito ocupara el espacio de un átomo, el número más grande que cabría en una RAM del tamaño del universo visible tendría del orden de 10^{111} dígitos. El número b^e calculado por nuestro algoritmo naïf no cabría en una RAM del tamaño del Universo observable.

¿Dónde reside la seguridad del esquema RSA?

Somos capaces de satisfacer este requisito de eficiencia en todos los subalgoritmos... excepto en el que sirve para calcular la descomposición factorial de un entero. No se conoce un algoritmo capaz de calcular esa factorización de forma eficiente, y precisamente en ello reside la fortaleza de la criptografía RSA: si n es lo suficientemente grande, factorizarlo requeriría un tiempo prohibitivo.

3.4.2. Descripción de la tarea

Diseña y programa en C/C++ los siguientes subalgoritmos:

esPrimo

Dado un entero $n > 1$, determina si es primo o no.

[1] La potencia de cálculo prevista es de 10^9 Gflops: la máquina podrá realizar 10^{18} operaciones de coma flotante por segundo. Para tener una aproximación del orden de magnitud del tiempo necesario, podemos suponer que cada una de las multiplicaciones requeridas para calcular b^e es comparable con una operación de coma flotante.

potenciaModulo

Dados tres enteros, b, e y n, con $e \geq 0$ y $n > 0$, devuelve el valor $b^e \,\%\, n$. Utilizando la exponenciación binaria y las propiedades del resto, $b^e \,\%\, n$ se calcula del siguiente modo:

$$b^e \,\%\, n = \begin{cases} 1, & \text{si e } = 0, \\ \Big((b^{e/2} \,\%\, n) \cdot (b^{e/2} \,\%\, n) \Big) \,\%\, n, & \text{si e es par,} \\ (\, b \cdot (b^{e-1} \,\%\, n) \,) \,\%\, n, & \text{si e es impar} \end{cases}$$

Sugerencia: escribe primero la función **potencia** que calcula la potencia b^e siguiendo el algoritmo de exponenciación binaria (viene descrito en la Sección *Ayudas*). Comprueba que funciona correctamente, y a continuación complétalo incluyendo el cálculo de los restos.

euclidesExtendido

Dados dos enteros positivos, n y m, devuelve su máximo común divisor y otros dos enteros, a y b, tales que

$$\text{mcd}(n, m) = a \cdot n + b \cdot m.$$

El subalgoritmo viene descrito en la Sección *Ayuda*.

inversoModulo

Sean dos enteros positivos, n y m. El inverso de m módulo n es un entero $b \in [0, n-1]$ t.q.

$$b \cdot m \;\equiv\; 1 \,(\text{mod } n) \tag{3.16}$$

$$b \cdot m \equiv 1 \,(\text{mod}\, n) \tag{3.17}$$

Es decir: $b \cdot m - 1$ es múltiplo de n. Si el máximo común divisor de n y m es 1, ese inverso existe y es único.

Dados dos enteros positivos, n y m, cuyo máximo común divisor es 1, el subalgoritmo **inversoModulo** devuelve el inverso de m módulo n. Para ello utiliza el subalgoritmo extendido de Euclides para calcular a y b tales que:

$$mcd(n, m) = 1 = a \cdot n + b \cdot m$$

Es inmediato que b satisface la condición (3.16). Sabemos, además, que $|b| \leq n$. Si $n > b \geq 0$, el subalgoritmo ya ha terminado. Si $b < 0$, es necesario sumarle n para obtener un valor en el intervalo [0, n-1]. En el caso $b = n$, tomaremos $b = 0$. El subalgoritmo presupone que n y m son correctos: no es necesario que compruebe que son positivos ni coprimos.

descompone

Dado un entero positivo n devuelve dos enteros, p y q, donde p es el menor entero (mayor que 1) que divide a n, y $q = n/p$. A continuación mostramos los resultados del algoritmo para diversos valores de n:

n	p	q
15	3	5
17	17	1
175	5	35

Observa que si n es primo, entonces $p = n$ y $q = 1$.

fEuler

Dado un entero positivo n devuelve un booleano b y un entero f (la función de Euler de n, $\varphi(n)$).

Para simplificar la tarea, sólo calcularemos $\varphi(n)$ en dos situaciones: si n es primo, $\varphi(n) = n - 1$; si es producto de dos primos distintos $n = p \cdot q$, $\varphi(n) = (p-1) \cdot (q-1)$. En ambas situaciones, el booleano devuelto será *true*.

Si no se da ninguna de las dos situaciones anteriores, el booleano devuelto será $false$; el valor del entero f es irrelevante.

calculaClavePrivada

Dados dos enteros n, e, este subalgoritmo determina, en primer lugar, si el par (n, e) es aceptable como clave pública. En caso afirmativo, calcula su clave privada correspondiente. Da como resultados dos enteros: *codigo* y d: *codigo* indica si el par (n, e) es aceptable y, en caso contrario, la causa de que no lo sea; d es la clave privada (si la clave pública no es aceptable, el valor de d es irrelevante).

La clave privada d es el inverso de e módulo $\varphi(n)$.

Un par (n, e) es aceptable como clave pública si satisface:

- n y e son positivos. Si la condición se incumple, el subalgoritmo devuelve $codigo = 1$.
- $n < \sqrt{\text{INT\_MAX}}$ (INT_MAX es el máximo valor entero que podemos representar con el tipo *int*; su valor está definido en el fichero <limits.h>). Esta comprobación es necesaria para garantizar que en los cálculos que realicemos no se produzcan salidas de rango. Si la condición se incumple, el subalgoritmo devuelve $codigo = 2$.
- sabemos calcular $\varphi(n)$ (n es un número primo o el producto de dos primos distintos). Si la condición se incumple, el subalgoritmo devuelve $codigo = 3$.
- $\varphi(n)$ y e son coprimos (su máximo común divisor es 1). Si la condición se incumple, el subalgoritmo devuelve $codigo = 4$.

Si el par es aceptable, el subalgoritmo devuelve $codigo = 0$.

interpretaCodigo

Dado un entero, *codigo*, el subalgoritmo muestra por pantalla su interpretación (tal como se describe en el subalgoritmo anterior, e ilustramos en los siguientes ejemplos).

Programa principal

Escribe un programa en C/C++ que, dados una clave pública (n, e) y un mensaje encriptado $M_{encriptado}$, desencripte $M_{encriptado}$. Para ello, el programa debe seguir los siguientes pasos:

El programa empieza solicitando al usuario que introduzca su clave pública (n, e) una y otra vez hasta que se introduzca una clave aceptable. A continuación mostrará por pantalla la clave privada d. Después solicita al usuario que introduzca un mensaje encriptado (un entero $M_{encriptado}$), asegurándose de que éste sea menor que n. Se mostrará en pantalla el mensaje desencriptado $M = (M_{encriptado})^d \,\%\, n$. A continuación, como validación,

vuelve a encriptar M (calcula $M^e \,\%\, n$). El valor obtenido debe coincidir con el mensaje encriptado; de no ser así, hay un error en el programa.

El programa permite desencriptar varios mensajes, utilizando la misma clave pública. Termina cuando el usuario teclea un valor negativo como mensaje.

3.4.3. Ejemplos de ejecución

En el siguiente ejemplo desencriptaremos dos mensajes, encriptados usando la clave pública ($n = 41989, e = 13$). El primero es el valor que Sheldon Cooper nunca utilizaría como contraseña; su valor encriptado es 10201. El segundo, el (supuesto) año de fundación de nuestra institución; su valor encriptado es 32998.

```
Introduzca una clave publica valida:
         n: -44
         e: 13
        Clave publica no valida: n y e deben ser positivos.

Introduzca una clave publica valida:
         n: 1234567891
         e: 17
        Clave publica no valida: n debe ser menor que 46341

Introduzca una clave publica valida:
         n: 121
         e: 17
        Clave publica no valida: n debe ser primo
        o producto de dos primos distintos.

Introduzca una clave publica valida:
         n: 41981
         e: 18
        Clave publica no valida: fi(n) y e deben ser coprimos.

Introduzca una clave publica valida:
         n: 41989
         e: 13
Clave publica valida.


El inverso de e modulo fi es 6397

Introduzca un mensaje encriptado (mEncr) menor que n: 99999
mEncr debe ser menor que n, intente de nuevo: 9999999
mEncr debe ser menor que n, intente de nuevo: 10201
El mensaje desencriptado (mEncr^d %n) es 1234
Comprobacion: m^e % n = 10201  OK

Introduzca un mensaje encriptado (menor que n): 32998
El mensaje desencriptado (mEncr^d %n) es 1542
Comprobacion: m^e % n = 32998  OK

Introduzca un mensaje encriptado (menor que n): -1
```

3.4.4. Ayudas

potencia

Dados dos enteros, b y e, con $e \geq 0$, calcula b^e siguiendo el algoritmo de exponenciación binaria (indicado a continuación) y lo devuelve:

$$b^e = \begin{cases} 1, & \text{si e } = 0, \\ b^{e/2} \cdot b^{e/2}, & \text{si e es par,} \\ b \cdot b^{e-1}, & \text{si e es impar} \end{cases}$$

Algoritmo extendido de Euclides

PROCEDIMIENTO euclidesExtendido (n p.d, m p.d. SON ENTEROS POSITIVOS,
sol p.d.r. ES ENTERO POSITIVO),
a p.r., b p.r. SON ENTEROS)
Dados dos enteros positivos, n y m, devuelve su máximo común divisor, sol,
así como dos enteros tales que $mcd(n, m) = sol = a \cdot n + b \cdot m$
VARIABLES a_{aux}, b_{aux} SON ENTEROS
CUERPO

```
SI n%m == 0
    sol ← m
    a ← 0
    b ← 1
SI NO
    euclidesExtendido (m, n%m, sol, a_aux, b_aux)
    a ← b_aux
    b ← a_aux − n/m · b_aux
FIN SI
```

EXPLICACIÓN

En el algoritmo de Euclides se van obteniendo pares (n_i, m_i) sucesivamente más sencillos ($n_{i+1} = m_i$, $m_{i+1} = n_i \,\%m_i$), hasta llegar a un par (n_f, m_f) cuyo máximo común divisor es trivial (m_f).

Para este par es inmediato encontrar enteros a_f, b_f que cumplan la condición deseada: $a_f = 0, b_f = 1$. En efecto, $mcd(n_f, m_f) = m_f = 0 \cdot n_f + 1 \cdot m_f$.

A continuación se trata de desandar lo andado: dados dos enteros a_{i+1}, b_{i+1} que satisfacen

$$mcd(n_{i+1}, m_{i+1}) = a_{i+1} \cdot n_{i+1} + b_{i+1} \cdot m_{i+1},$$

¿Podemos encontrar a_i, b_i t.q. $mcd(n_i, m_i) = a_i \cdot n_i + b_i \cdot m_i$? basta recordar que $m_{i+1} = n_i \,\%m_i = n_i - n_i/m_i \cdot m_i$, para reagrupar los términos del miembro derecho de la igualdad:

$$a_{i+1} \cdot n_{i+1} + b_{i+1} \cdot m_{i+1} = b_{i+1} \cdot n_i + (a_{i+1} - n_i/m_i \cdot b_{i+1}) \cdot m_i$$

Por tanto, $a_i = b_{i+1}$ y $b_i = a_{i+1} - n_i/m_i \cdot b_{i+1}$.

Obsérvese que a lo largo de todo el proceso m_i y n_i son positivos, lo que puede probarse por inducción. En efecto, $m_0 \equiv m$ y $n_0 \equiv n$ son positivos por hipótesis. Si $m_i, n_i > 0$ es obvio que $n_{i+1} (= m_i)$ lo es; dado que m_{i+1} se calcula si m_i no es divisor de n_i, su valor $(n_i \,\%m_i)$ sigue siendo positivo.

Una propiedad interesante del algoritmo es que, a lo largo de todo el proceso, $|a_i| \leq m_i$, $|b_i| \leq n_i$, lo cual puede demostrarse por inducción. La inducción empieza en el último paso: $a_f = 0 < m_f$ y $b_f = 1 \leq n_f$. Hagamos la hipótesis de que $a_{i+1} \leq m_{i+1}$ y $b_{i+1} \leq n_{i+1}$; ¿ La condición se sigue cumpliendo para a_i y b_i ? Comprobémoslo: $|a_i| = |b_{i+1}| \leq n_{i+1} = m_i$. Por otro lado, $|b_i| \leq |a_{i+1}| + n_i/m_i \cdot |b_{i+1}| \leq m_{i+1} + n_i/m_i \cdot n_{i+1} = n_i \,\%m_i + n_i/m_i \cdot m_i = n_i$.

En particular, el valor final de b satisface que $|b| \leq n$: $-n \leq b \leq n$.

EJEMPLO

mcd (1791, 1272) =				
mcd (1272, 519) =				
mcd (519, 234) =				
mcd (234, 51) =				
mcd (51, 30) =				
mcd (30, 21) =				
mcd (21, 9) =				
mcd (9, 3) =	$0 \cdot 9$	+	$1 \cdot 3 =$	
mcd (21, 9) =	$1 \cdot 21$	+	$(-2) \cdot 9 =$	
mcd (30, 21) =	$(-2) \cdot 30$	+	$3 \cdot 21 =$	
mcd (51, 30) =	$3 \cdot 51$	+	$(-5) \cdot 30 =$	
mcd (234, 51) =	$(-5) \cdot 234$	+	$23 \cdot 51 =$	
mcd (519, 234) =	$23 \cdot 519$	+	$(-51) \cdot 234 =$	
mcd (1272, 519) =	$(-51) \cdot 1272$	+	$125 \cdot 519 =$	
mcd (1791, 1272) =	$125 \cdot 1791$	+	$(-176) \cdot 1272$	

Observa que los signos de a_i y b_i cambian alternativamente en cada iteración.

3.4.5. Para profundizar: el requisito de eficiencia

Puede demostrarse por inducción que el algoritmo de Euclides necesita un máximo de $2 \cdot \log_2 n$ pasos para calcular el máximo común divisor de (n, m), siendo $n > m$. Asimismo, el subalgoritmo **potencia** necesita entre $\log_2 e$ y $2 \cdot \log_2 e$ multiplicaciones para calcular b^e.

Hagamos cuentas: si n y m son del orden de 10^{200} (tienen 200 dígitos), se necesita un máximo de $2 \cdot 200 \cdot \log_2 10 \simeq 1330$ pasos para calcular su máximo común divisor. Además, el cálculo de b^n requiere, como máximo, $2 \cdot 200 \cdot \log_2 10 \simeq 1330$ multiplicaciones.

En el subalgoritmo **potenciaModulo** tomamos el resto entre n de los sucesivos valores calculados. Por tanto, en cada paso multiplicamos dos números que son menores o iguales que n: el valor más grande obtenido es n^2. Si $n \sim 10^{200}$, los números que manejemos tendrán, como máximo, 400 dígitos.

3.5. Refactorización

3.5.1. Introducción

En las tareas que hemos elaborado hasta ahora has resuelto un problema desde cero: Conociendo las especificaciones has llegado a escribir un programa que resuelve el problema propuesto.

Sin embargo, la tarea de programador no es siempre así. En muchas ocasiones, tendrás que revisar código que han hecho otras personas o has hecho tú mismo para adaptarlo a nuevas necesidades o para incorporar nuevas especificaciones o simplemente para mejorar la calidad del código. Se llama refactorización al proceso de generar un nuevo código a partir de un código ya existente.

En esta práctica te proponemos la tarea de refactorización del programa de las agujas de Buffon. Al resolver ese problema, como todavía no conocíamos los subalgoritmos, todo el código está contenido en un programa principal. Tu tarea consiste en reelaborar el código de forma que el nuevo código esté compuesto de subalgoritmos. De este modo, comprobarás que el código obtenido es mucho más legible que el código original.

3.5.2. Descripción de la actividad

Reelabora el código del programa de las agujas de Buffon de forma que contenga los siguientes subalgoritmos brevemente descritos aquí:

1. Subalgoritmo que, dado un mensaje y una cota inferior, muestra el mensaje y solicita un dato real al usuario. El mensaje informa al usuario del dato que tiene que introducir y de los límites que tiene que respetar. El subalgoritmo termina cuando el dato introducido es superior a la cota inferior y ese dato es devuelto por el subalgoritmo
2. El mismo subalgoritmo anterior pero solicitando un entero
3. Subalgoritmo que dados dos números reales, devuelve un número real generado de manera aleatoria y comprendido entre esos dos números reales
4. Subalgoritmo que, dado el número de agujas, la longitud de cada aguja y la separación, devuelve la probabilidad estimada y la aproximación a π

Para hacer esta tarea, usa el fichero `agujasDeBuffon.cpp` que está disponible en Moodle.

3.5.3. Ayudas

Para llevar a cabo la refactorización del código, puedes seguir estos pasos:

- Identifica fragmentos de código del programa original que pueden formar parte de cada subalgoritmo propuesto en el apartado anterior. Para ello, debes detectar fragmentos de código que realicen la tarea descrita para esos subalgoritmos.
- Utiliza el código que has identificado para construir el subalgoritmo correspondiente. Para ello deberás determinar la cabecera del subalgoritmo teniendo en cuenta la descripción de los subalgoritmos que hemos indicado en el apartado anterior
- En el programa principal, deja como comentario el código que has encontrado en el primer paso. Escribe en el punto donde comenzaba ese código la llamada al subalgoritmo que has escrito para reemplazar a ese código.

- Comprueba que el nuevo código funciona correctamente. Una vez hecho esto, ya puedes eliminar el código que hemos dejado entre comentarios.
- Repite los pasos anteriores con el resto del código.

3.6. Proyecto 2C: Dinámica de una órbita elíptica (ii)

En esta tarea avanzaremos en el estudio cuantitativo de la dinámica de una partícula que, sometida a una fuerza central de módulo K/r^2, describe una órbita elíptica. Concretamente realizaremos el cálculo inverso al de la tarea 2.7: dado el ángulo θ_0 que determina, en coordenadas polares, la posición de la partícula en un instante inicial t_0, calcularemos su posición θ_f en otro instante de tiempo t_f.

3.6.1. Contexto del problema

Recordemos que la ecuación de una elipse, en coordenadas polares, es la siguiente:

$$r = \frac{a \cdot (1 - e^2)}{1 - e \cdot \cos(\theta - \phi_0)} \ , \tag{3.18}$$

donde a es la longitud del semieje mayor de la elipse, e es su excentricidad y ϕ_0 el ángulo que el semieje mayor de la elipse forma con el eje de coordenadas polares.

Por otro lado, la segunda ley de Kepler nos dice que el área $\frac{1}{2}\int_{\theta_0}^{\theta_f} r^2 d\theta$ barrida por el vector de posición de la partícula entre dos ángulos, θ_0 y θ_f, es proporcional al tiempo $t_f - t_0$ que la partícula necesita para moverse de una posición a la otra. Así, puesto que el tiempo necesario para recorrer una órbita completa es el periodo, T, de la misma, y el área barrida es entonces el de la elipse, $\pi a^2\sqrt{1-e^2}$, tenemos que

$$\frac{1}{\pi a^2\sqrt{1-e^2}} \int_{\theta_0}^{\theta_f} \frac{r^2}{2} d\theta = \frac{(1-e^2)^{3/2}}{2\pi} \int_{\theta_0}^{\theta_f} \frac{d\theta}{(1 - e \cdot \cos(\theta - \phi_0))^2} = \frac{t_f - t_0}{T} \ . \tag{3.19}$$

Si escribimos

$$f(\varphi) = \frac{(1-e^2)^{3/2}}{2\pi} \int_{\phi_0}^{\varphi} \frac{d\theta}{(1 - e\cos(\theta - \phi_0))^2} \ , \tag{3.20}$$

la ecuación integral anterior, en la que θ_f es la incógnita, se puede expresar de forma más sencilla como

$$f(\theta_f) = f(\theta_0) + \frac{t_f - t_0}{T} \ . \tag{3.21}$$

Así, conocidos los parámetros e y ϕ_0 que definen la elipse, el periodo T de la órbita, los instantes de tiempo inicial, t_0, y final, t_f, y la posición de la partícula, θ_0, en el instante inicial, nuestro objetivo es calcular la inversa de la función f.

Para ello, en lugar de realizar una integración numérica, como en la tarea anterior, usaremos la siguiente expresión analítica de la función f.

$$f(\varphi) = \frac{1}{2\pi}\left[\frac{(1-e^2)^{3/2}\sin\theta\cos\theta}{(1 - e\cdot\cos\theta)^2} + \frac{2n+1}{2}\pi - (-1)^n \cdot \left(\xi + \frac{\sin(2\xi)}{2}\right)\right] \ , \tag{3.22}$$

donde $\theta = \varphi - \phi_0$, $n = \lfloor \theta/\pi \rfloor$ es el número de semivueltas correspondientes al ángulo θ (el mayor entero que es menor o igual que θ/π) y

$$\xi = \arcsin\left(\frac{\cos\theta - e}{1 - e\cdot\cos\theta}\right) \ .$$

Obsévese que la función f es estrictamente creciente. Este hecho crucial nos va a permitir encontrar una aproximación de θ_f con un error ε tan pequeño como se desee. Procederemos del modo siguiente.

En primer lugar calcularemos el entero $m = \lfloor (t_f - t_0)/T \rfloor$; esto es, el único $m \in \mathbb{Z}$ tal que $m \leq \frac{t_f - t_0}{T} < m + 1$. Así, puesto que el área barrida por el vector de posición entre los ángulos θ_0 y $2\pi m + \theta_0$ es igual a m órbitas y el tiempo que la partícula emplea en ello es mT, de la ecuación (3.19) se sigue que $\theta_f \in [\theta_{\text{inf}}, \theta_{\text{sup}}]$, donde $\theta_{\text{inf}} = 2\pi m + \theta_0$ y $\theta_{\text{sup}} = 2\pi(m+1) + \theta_0$. En otras palabras, para alcanzar la posición θ_f desde θ_0 es preciso dar m vueltas completas a la órbita y recorrer, pero sin completar, parte de una vuelta más.

Una vez encontrada esta primera acotación calculamos el ángulo $\theta_{\text{med}} = (\theta_{\text{inf}} + \theta_{\text{sup}})/2$ y, con la ayuda de la expresión (3.22), el valor de $f(\theta_{\text{med}})$. Puesto que f es creciente, si $f(\theta_{\text{med}}) < f(\theta_f)$ entonces $\theta_f \in [\theta_{\text{med}}, \theta_{\text{sup}}]$, mientras que si $f(\theta_{\text{med}}) > f(\theta_f)$ entonces $\theta_f \in [\theta_{\text{inf}}, \theta_{\text{med}}]$. Así, basta reemplazar por θ_{med} uno de los extremos del intervalo, según convenga, y repetir el proceso hasta que la longitud del intervalo sea menor que la precisión requerida; esto es, hasta que $\theta_{\text{sup}} - \theta_{\text{inf}} < \varepsilon$. Terminado este proceso el valor de θ_f es $(\theta_{\text{inf}} + \theta_{\text{sup}})/2$ con un error de ε.

Este método, conocido con el nombre de *método de bisección*, es una aplicación directa del Teorema de Bolzano, tal y como se describe en el Apéndice D.

3.6.2. Descripción de la tarea

Diseña y programa en C/C++ los subalgoritmos que se indican a continuación. Puedes añadir todos los subalgoritmos auxiliares que necesites.

integralExacta

Dadas la excentricidad, e, y la inclinación, ϕ_0, de una elipse y un águlo φ, calcula el valor de $f(\varphi)$ usando la expresión (3.22).

A continuación mostramos algunos ejemplos de ejecución del subalgoritmo:

e	ϕ_0	φ	$f(\varphi)$
0.95	1	1	0
0.95	2	5	0.499819
0.95	2	0	-0.498153
0.6	1.5	3	0.422786

dinamica

Dados la excentricidad, inclinación y periodo (e, ϕ_0 y T) de una órbita elíptica, un intervalo de tiempo $[t_0, t_f]$, la posición θ_0 de la partícula en t_0, y un valor $\varepsilon > 0$, calcula la posición de la partícula en el instante t_f con precisión ε.

La tabla siguiente muestra los valores de θ_f devueltos por el subalgoritmo para diversos valores de entrada, tomando en todos ellos $\varepsilon = 10^{-6}$:

e	ϕ_0	T	t_0	θ_0	t_f	θ_f
0.95	1	100	0	1	50	4.141593
0.95	1.5	100	10	-0.5	-25	-4.701583

coordenadas

Dados el semieje mayor, excentricidad e inclinación (a, e, ϕ_0) de una órbita elíptica y una posición, θ, devuelve su distancia r al origen (ecuación 3.18), así como sus coordenadas cartesianas: $x = r\cos\theta$, $y = r\sin\theta$.

A continuación mostramos algunos resultados devueltos por el subalgoritmo:

a	e	ϕ_0	θ	r	x	y
5	0.95	1	1	9.75	5.26795	8.20434
3	0.6	2	4	1.53638	-1.00425	-1.16274

Programa principal

El programa solicita al usuario que introduzca los parámetros de una órbita elíptica (semieje mayor, excentricidad, inclinación y periodo: a, e, ϕ_0 y T), así como la posición, θ_0, de la partícula en el instante $t_0 = 0$.

Como respuesta el programa presenta una tabla en la pantalla en la que se muestran las posiciones de la partícula entre los instantes $-T$ y T a intervalos de tiempo $0{,}1 \cdot T$. Los tiempos se indicarán tanto en función de T como en segundos. Las posiciones se mostrarán en coordenadas polares (r, θ) y cartesianas (x, y). Para el cálculo de la posición se desea una precisión $\varepsilon = 10^{-9}$.

3.6.3. Ejemplos de ejecución

```
Introduzca los parametros de la orbita:
         Semieje mayor (en m): 5
         Excentricidad: 0.95
         Inclinacion phi0 (en radianes): 1
         Periodo T (en segundos): 1000

Posicion de en t=0 (en radianes): 1
t(T)       t(s)       r     theta          x          y
     -1     -1000  9.75     -5.28319   5.26795     8.20434
   -0.9      -900  9.50136  -5.23069   4.70706     8.25345
   -0.8      -800  8.72872  -5.17215   3.87324     7.82232
   -0.7      -700  7.33283  -5.09664   2.74881     6.79812
   -0.6      -600  5.0356   -4.96796   1.27298     4.87205
   -0.5      -500  0.25     -2.14159  -0.135076   -0.210368
   -0.4      -400  5.0356    0.684772  3.90039     3.185
   -0.3      -300  7.33283   0.813454  5.0376      5.32851
   -0.2      -200  8.72872   0.888966  5.50098     6.77716
   -0.1      -100  9.50136   0.94751   5.54601     7.71477
      0         0  9.75      1         5.26795     8.20434
    0.1       100  9.50136   1.05249   4.70706     8.25345
    0.2       200  8.72872   1.11103   3.87324     7.82232
    0.3       300  7.33283   1.18655   2.74881     6.79812
    0.4       400  5.0356    1.31523   1.27298     4.87205
    0.5       500  0.25      4.14159  -0.135076   -0.210368
    0.6       600  5.0356    6.96796   3.90039     3.185
    0.7       700  7.33283   7.09664   5.0376      5.32851
    0.8       800  8.72872   7.17215   5.50098     6.77716
```

```
    0.9       900  9.50136   7.23069   5.54601     7.71477
      1      1000  9.75      7.28319   5.26795     8.20434
```

Los resultados anteriores se muestran gráficamente en la figura 3.5. Observa que los puntos, tomados a intervalos regulares de tiempo, se concentran en torno al apoastro (punto más alejado del centro de atracción). Esto es compatible con la segunda ley de Kepler: al alejarse del foco atractivo, la velocidad angular es menor.

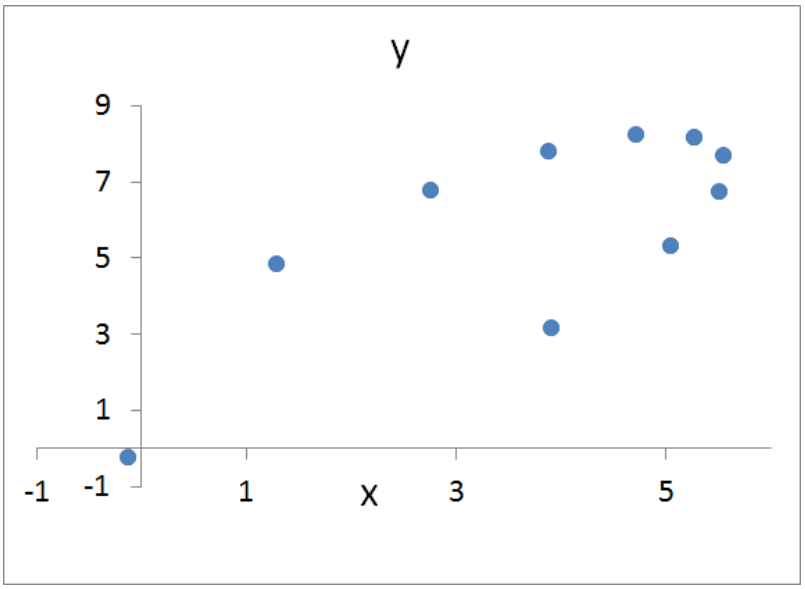


Figura 3.5: Posiciones de la partícula a intervalos regulares de tiempo.

3.6.4. Ayudas

La función con prototipo `double floor(double x)`, declarada en el fichero de cabecera `<math.h>`, devuelve el valor de $\lfloor x \rfloor$.

3.7. El péndulo simple: solución numérica (i)

El péndulo simple es una partícula puntual de masa m suspendida de un punto fijo mediante un hilo rígido de longitud l y masa nula. En esta tarea resolveremos numéricamente su ecuación diferencial, sin considerar por el momento fricción ni fuerza externa:

$$\ddot{\theta} = -\alpha \,\text{seno}(\theta), \tag{3.23}$$

donde α tiene dimensión de tiempo^{-2} (observa que $\alpha = g/l$, donde g es la aceleración debida a la gravedad y l la longitud del péndulo).

La diferencia entre el péndulo y el oscilador armónico es que en la aceleración el término lineal $-\alpha\, x$ se sustituye por una función no lineal: el seno. En consecuencia, la ecuación diferencial es más difícil de resolver pero, a cambio, la dinámica resultante es más compleja e interesante. Ya sabemos que en el límite de pequeñas oscilaciones $\text{seno}(\theta) \approx \theta$, y reobtenemos la ecuación del oscilador. En esta tarea empezaremos a explorar cómo cambia la dinámica cuando no estamos en ese límite.

3.7.1. Introducción

Análisis de la dinámica del péndulo sin pérdida

La energía del péndulo viene dada por la expresión $E = 1/2\, ml^2\dot{\theta}^2 + mgl\,(1 - \cos\theta)$. Según su valor, podemos distinguir tres situaciones:

i) $E < 2mgl$ (equivalentemente, $\frac{\dot{\theta}^2}{2\alpha} - \cos\theta < 1$).

El péndulo no tiene energía suficiente para voltearse, y describe un movimiento periódico cuya amplitud máxima viene dada por la conservación de la energía:

$$\theta_M = \arccos\left(cos\,(\theta_0) - \frac{\dot{\theta}_0^2}{2\,\alpha}\right) \tag{3.24}$$

ii) $E = 2mgl$ (equivalentemente, $\frac{\dot{\theta}^2}{2\alpha} - \cos\theta = 1$).

El péndulo tiene la energía justa necesaria para llegar a la posición vertical ($\theta = \pi$) con velocidad cero, y quedarse en ese estado. Ese estado de equilibrio inestable se alcanza en el límite $t \to \infty$.

iii) $E > 2mgl$ (equivalentemente, $\frac{\dot{\theta}^2}{2\alpha} - \cos\theta > 1$).

El péndulo tiene energía suficiente para voltearse. Si proyectamos el ángulo θ en el intervalo $[-\pi, \pi)$ (es decir, si sustituimos θ por θ', donde $\theta' = \theta + 2n\pi$ y $\theta' \in [-\pi, \pi)$), debido a la conservación de la energía θ' es una función periódica .

En los casos de movimiento periódico (casos i, iii) el cálculo del periodo, T, es más complicado que en el caso del oscilador armónico. Requiere resolver integrales elípticas, que quedan fuera del alcance de este curso. Nos limitaremos a presentar sus soluciones en la sección 3.7.3.

El método de Runge-Kutta

En la tarea 2.9 resolvimos la ecuación diferencial de segundo grado 2.19 reescribiéndola como un sistema de dos ecuaciones de primer grado acopladas. Resolvimos cada una de estas por el método de Euler, que esencialmente consiste en discretizar el tiempo en intervalos de longitud Δt y truncar el desarrollo en serie de Taylor en el término en Δt.

En esta tarea, los valores con los que trabajábamos ($\Delta t = 10^{-4}T$) no van a dar resultados con la precisión deseada. Una posible solución es tomar valores aún menores de Δt, lo que ralentizaría notablemente la ejecución del programa.

Hay una alternativa más eficiente: tomar más términos en el desarrollo en serie de Taylor. Esta es la estrategia que seguiremos en esta tarea. Seguiremos el método de Runge-Kutta de orden 4, descrito en la sección 3.7.3 (ecuación 3.32), que equivale a truncar la serie de Taylor en el término en Δt^4.

3.7.2. Descripción de la tarea

Diseña y programa en C/C++ los siguientes subalgoritmos:

validaAngulo

Dados un ángulo θ y un carácter, que especifica en qué unidades se da (`'r'` para radianes, `'s'` para grados sexagesimales) devuelve un booleano y un real. Si las unidades son correctas (`'r'` o `'s'`), el booleano vale `true`, y el real es el valor de θ en radianes. Si no, el booleano vale `false` y no se modifica el valor de θ.

fase

Dados dos reales, s y r, devuelve $\pi/2$ si $s \geq 1$, $-\pi/2$ si $s \leq -1$ y en cualquier otro caso un ángulo φ_0 tal que $\sin\varphi_0 = s$ y $\cos\varphi_0$ tenga el mismo signo que r.

Ten en cuenta que la función `asin` en C/C++ devuelve un ángulo en el intervalo $[-\pi/2, \pi/2]$

normaliza

Dado un ángulo θ, lo transforma en otro θ' tal que $\theta - \theta' = 2n\pi$, $\theta' \in [-\pi, \pi)$:

$$\theta' = \theta - 2n\pi \ , \quad \text{donde } n = \left\lfloor \frac{\theta + \pi}{2\pi} \right\rfloor . \tag{3.25}$$

Para calcular $\lfloor x \rfloor$, el mayor entero menor o igual que x, se puede usar la función matemática `floor`.

analisis

Dados el valor de α, el ángulo y la velocidad inciales (θ_0 y $\dot\theta_0$), y la precisión deseada (ϵ), devuelve dos reales, θ_M y τ, y un entero k que caracteriza el tipo de solución. θ_M es el ángulo límite (el ángulo máximo de oscilación en los casos periódicos, $\pm\pi$ si el péndulo acaba en el estado de equilibrio inestable) y τ un tiempo significativo del péndulo:

- En caso de movimiento periódico ($E \neq 2mgl$) (ver sección 3.7.1, casos i y iii), τ es el periodo de oscilación (ver ecuaciones 3.28 a 3.30).

- En el caso $E = 2mgl$, τ es el tiempo que le cuesta al péndulo llegar a $\theta = 0{,}999\,\theta_M$ (ver ecuación 3.27).

Caso	θ_M	τ	k
$E < 2mgl$	θ_M(ecuación 3,24)	T(ecuaciones 3,28 a 3,30)	-1
$E = 2mgl\,,\ \dot\theta_0 \geq 0$	π	$t(\theta = 0{,}999\,\pi)$(ecuación 3,27)	0
$E = 2mgl\,,\ \dot\theta_0 < 0$	$-\pi$	$t(\theta = -0{,}999\,\pi)$(ecuación 3,27)	0
$E > 2mgl$	π	T(ecuaciones 3,28 a 3,30)	1

A continuación mostramos los valores devueltos en función de distintos juegos de argumentos, para $\alpha = 1$, $\epsilon = 10^{-7}$:

θ_0	$\dot\theta_0$	k	θ_M	τ
0,001	0	-1	0,001	6,28319
0,5	0	-1	0,5	6,38279
1	0	-1	1	6,69998
1,5	0	-1	1,5	7,30086
2	0	-1	2	8,34975
2,5	0	-1	2,5	10,3232
3	0	-1	3	16,1555
0	2	0	3,14159	7,14932
0	-2	0	$-3{,}14159$	7,14932
0,001	2	1	3,14159	17,9744
0,01	2	1	3,14159	13,3692
0,1	2	1	3,14159	8,76067
1	2	1	3,14159	4,12642

RungeKutta

Dados α, el ángulo y la velocidad del péndulo en un instante de tiempo t ($\theta(t)$ y $\dot\theta(t)$) y los valores t y Δt, devuelve el ángulo y la velocidad del péndulo en el instante $t + \delta t$ ($\theta(t + \Delta t)$ y $\dot\theta(t + \Delta t)$) usando las expresiones 3.32 (sección 3.7.3).

programa principal

Recibe del flujo de entrada los factores factor_Δ, $\text{factor}_{\text{cout}}$ y $\text{factor}_{\text{tSimul}}$ (cuyo significado es el mismo que en la tarea 2.9), el ángulo y velocidad iniciales (θ_0, $\dot\theta_0$) y el parámetro del péndulo (α). El ángulo y la velocidad se leerán como un real y un carácter indicando sus unidades (`'r'` para radianes, `'s'` para grados sexagesimales). Para el resto de factores no se indicarán unidades: los factores temporales son adimensionales y suponemos que α se mide en s^{-2}.

Si α, factor_Δ, $\text{factor}_{\text{cout}}$ y $\text{factor}_{\text{tSimul}}$ no son positivos, o si las unidades del ángulo o de la velocidad angular son incorrectas, el programa mostrará un mensaje de error y finalizará.

A continuación se calcularán el ángulo máximo y el tiempo característico del péndulo, τ (subalgoritmo **analisis**) y, a partir de ellos, Δt, t_{simul} y t_{cout}: $\Delta t = \text{factor}_\Delta \cdot \tau$, $t_{cout} = \text{factor}_{\text{cout}} \cdot \tau$, $t_{simul} = \text{factor}_{\text{tSimul}} \cdot \tau$. Sugerimos calcular τ con un error relativo máximo $\epsilon = 10^{-7}$.

El programa calculará la dinámica del péndulo hasta $t = t_{simul}$ siguiendo el método de Runge-Kutta. A intervalos regulares t_{cout} enviará al flujo de salida los valores de t, θ, $\dot\theta$.

Además, en el caso de que la energía del péndulo sea menor que $2mgl$, se mostrará el valor de la función

$$\theta_{\text{seno}} = \theta_M \,\text{seno}\left(\frac{2\pi}{T}\, t \;+\; \varphi_0\right), \tag{3.26}$$

de modo que el valor de θ_{seno} en t=0 sea θ_0, y su velocidad angular en t=0 tenga el mismo signo que $\dot{\theta}_0$ (utiliza el subalgoritmo **fase**); θ_M es el ángulo máximo del péndulo. De esta forma podemos visualizar cómo cambia de forma la solución θ respecto del comportamiento del oscilador armónico.

Los valores de todos los ángulos se indicarán en el intervalo $[-\pi, \pi)$.

3.7.3. Ayudas

El periodo del péndulo

En el caso $E = 2mgl$ (equivalentemente, $\frac{\dot{\theta}^2}{2\alpha} - \cos\theta = 1$), la solución de la ecuación 3.23 es

$$t(\theta) = \pm\frac{1}{\sqrt{\alpha}} \ln\left[C \tan\left(\frac{\theta}{4} + \frac{\pi}{4}\right)\right] \quad \text{con} \quad C = 1/\tan\left(\frac{\theta_0}{4} + \frac{\pi}{4}\right) \tag{3.27}$$

El signo es el mismo que el de la velocidad inicial, $\dot{\theta}_0$.

En los demás casos el péndulo exhibe un comportamiento periódico, cuyo periodo T viene dado por la siguiente serie[4]

$$T = T_0 \sum_{n=0}^{\infty} a_n \tag{3.28}$$

donde:

$$a_0 = 1, \quad a_n = \left(\frac{2n-1}{2n} r\right)^2 a_{n-1} \, \forall n > 0 \tag{3.29}$$

Los valores de T_0 y r dependenden del caso en que estemos:

$$\text{Si } \frac{\dot{\theta}^2}{2\alpha} - \cos\theta < 1 : \; T_0 = \frac{2\pi}{\sqrt{\alpha}}, \quad r = \text{seno}\left(\frac{\theta_M}{2}\right), \text{ con } \theta_M = \arccos\left(cos\left(\theta_0\right) - \frac{\dot{\theta}_0^2}{2\,\alpha}\right)$$

$$\text{Si } \frac{\dot{\theta}^2}{2\alpha} - \cos\theta > 1 : \; T_0 = \frac{\pi}{A}, \quad r = \frac{\sqrt{\alpha}}{A}, \quad \text{con} \quad A = \frac{\sqrt{\dot{\theta_0}^2 + 2\alpha(1 - \cos\theta_0)}}{2} \tag{3.30}$$

Para que el error relativo al calcular T sea menor que un cierto valor ϵ, podemos truncar la serie anterior en el término a_N, donde

$$N \geq \frac{\log\left(\epsilon\left(1 - r^2\right)\right)}{2\,\log r} \tag{3.31}$$

Observa que para valores pequeños de la energía el periodo tiende a la expresión con que estamos familiarizados: $\lim_{E\to 0} T = 2\pi\sqrt{l/g}$.

El método de Runge-Kutta de orden 4

El método de Euler, que seguimos en la tarea 2.9, calcula el valor de una magnitud x en un tiempo $t + \Delta t$ conocidos su valor y su velocidad (su derivada respecto de t) en el instante t. Consiste en truncar en el término Δt el desarrollo en serie de Taylor en torno a $x(t)$. Si lo aplicáramos a la presente tarea, dados el ángulo y la velocidad angular en un instante de tiempo $(\theta_n,\ w_n)$ calcularíamos sus valores en el instante inmediatamente posterior así:

$$\begin{aligned} \theta_{n+1} &= \theta_n + w_n\,\Delta t \\ w_{n+1} &= w_n + a_n\,\Delta t \end{aligned}$$

donde la aceleración angular viene dada por $a_n = -\alpha\,\text{seno}\,(\theta_n t)$.

Sin embargo, en esta tarea seguiremos una aproximación más sofisticada y eficiente: los métodos de Runge-Kuta, que calculan el valor de x a partir de la media ponderada de las estimaciones de x y su velocidad en varios puntos del intervalo $[t, t + \Delta t]$. El método de orden n equivale a tomar los términos del desarrollo de Taylor hasta $(\Delta t)^n$.

En concreto, usaremos el método de Runge-Kutta de orden 4. Calcularemos $\theta(t+\Delta t)$ a partir de la media ponderada de su derivada (la velocidad angular, w) en t y de estimaciones de w en $t + \Delta t/2$ y $t + \Delta t$. Análogamente, el valor de $w(t + \Delta t)$ se calcula a partir de la media ponderada de su derivada (la aceleración angular, a) en t y de estimaciones de a en $t + \Delta t/2$ y $t + \Delta t$.

En esta sección nos limitaremos a presentar las expresiones que nos permiten implementar el método, sin justificar cómo se deducen. En particular, conocidos θ_n y $w_n = \dot\theta_n$, estimamos θ_{n+1} y $w_{n+1} = \dot\theta_{n+1}$ del siguiente modo:

$$\begin{aligned} \delta\theta_A &= w_n\,\Delta t \\ \delta w_A &= -\alpha\,\text{seno}\,(\theta_n)\ \Delta t \end{aligned}$$

$$\begin{aligned} \delta\theta_B &= \left(w_n + \frac{\delta w_A}{2}\right)\ \Delta t \\ \delta w_B &= -\alpha\,\text{seno}\left(\theta_n + \frac{\delta\theta_A}{2}\right)\ \Delta t \end{aligned}$$

$$\begin{aligned} \delta\theta_C &= \left(w_n + \frac{\delta w_B}{2}\right)\ \Delta t \\ \delta w_C &= -\alpha\,\text{seno}\left(\theta_n + \frac{\delta\theta_B}{2}\right)\ \Delta t \end{aligned}$$

$$\begin{aligned} \delta\theta_D &= (w_n + \delta w_C)\ \Delta t \\ \delta w_D &= -\alpha\,\text{seno}\,(\theta_n + \delta\theta_C)\ \Delta t \end{aligned}$$

$$\begin{aligned} \theta_{n+1} &= \theta_n + \frac{\delta\theta_A}{6} + \frac{\delta\theta_B}{3} + \frac{\delta\theta_C}{3} + \frac{\delta\theta_D}{6} \\ w_{n+1} &= w_n + \frac{\delta w_A}{6} + \frac{\delta w_B}{3} + \frac{\delta w_C}{3} + \frac{\delta w_D}{6} \end{aligned} \tag{3.32}$$

Se puede encontrar una excelente introducción al método de Runge-Kutta, y un análisis de sus ventajas e inconvenientes, en [27].

3.7.4. Ejemplos de ejecución

El fichero `datosEntrada_Pendulo.txt`, descargable desde Moodle, contiene los datos de entrada de los ejemplos discutidos en esta sección.

Para los datos de entrada:

```
0.0001  0.02    1   90 s 0 r  1
```

las primeras filas devueltas por el programa son:

```
t  angulo   velocidad   funcionSinusoidal
0 1.5708 0 1.5708
0.148326 1.5598 -0.148324 1.55841
0.296652 1.5268 -0.296595 1.52145
0.444978 1.47183 -0.444542 1.46049
0.593304 1.39497 -0.591471 1.3765
```

Los resultados se muestran en la Figura 3.6. La aproximación de pequeñas oscilaciones no es válida (90^0 no es un ángulo pequeño): el péndulo oscila con un periodo algo mayor que $T_0 = 2\pi$; aun así, θ todavía se puede aproximar razonablemente bien mediante una función sinusoidal, salvo en torno a sus extremos. La trayectoria descrita por el péndulo en el espacio de fases se puede aproximar mediante una elipse.

Para los datos de entrada:

```
0.0001  0.005    1   0 r 1.999999895801368 r  1
```

los resultados se muestran en la Figura 3.7. El comportamiento es totalmente distinto del observado en el caso de pequeñas oscilaciones. El periodo es ahora $6T_0 = 12\pi$. Además, θ ya no puede aproximarse como una función sinusoidal del tiempo: tiende a la forma de una onda cuadrada. La trayectoria descrita por el péndulo en el espacio de fases es claramente distinta de una elipse.

Consideremos ahora el caso:

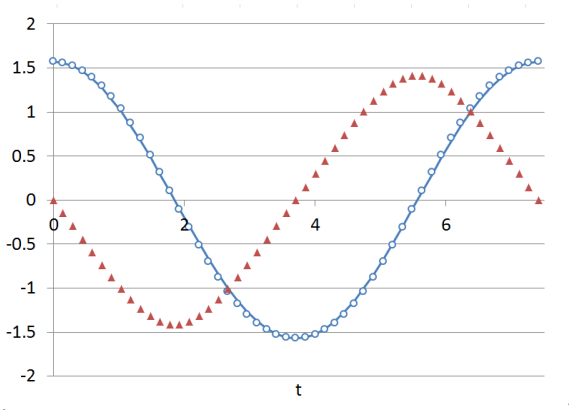
```
0.0001  0.02    1   0 r -2 r  1
```

El péndulo tiene la energía justa para llegar a la posición de equilibrio inestable $\theta = -\pi$ con velocidad 0, y permanecer allí indefinidamente. Los resultados se muestran en la Figura 3.8.

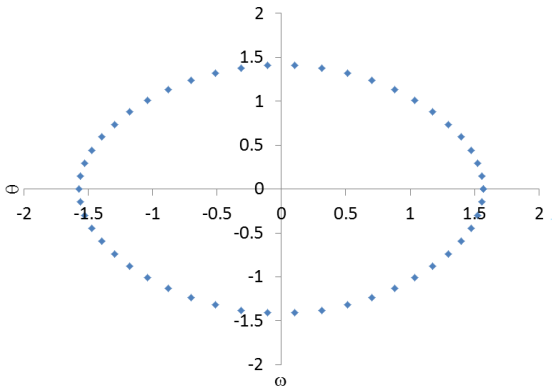
En el caso:

```
0.0001  0.02    1   0 r 2.01 r  1
```

el péndulo tiene energía suficiente para voltearse. Los resultados se muestran en la Figura 3.9.

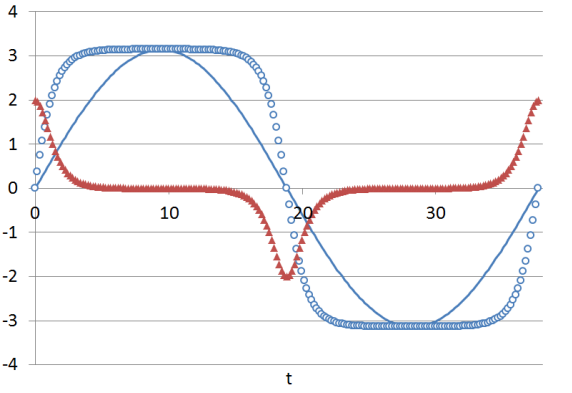


(a) θ (círculos) y $\dot{\theta}$ (triángulos) frente a la función $\theta = \theta_M \,\text{seno}\left(\frac{2\pi}{T}\, t + \varphi_0\right)$ (línea continua)

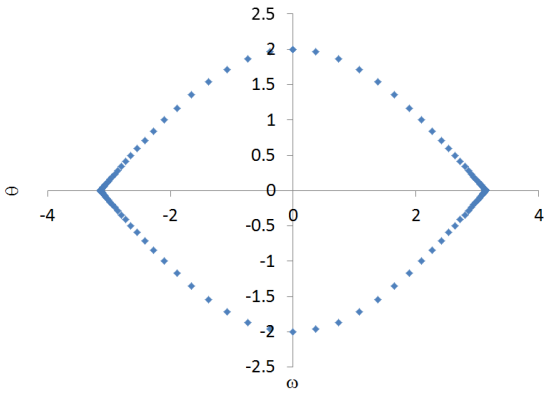


(b) Espacio de fases

Figura 3.6: $\theta_0 = 90^0$, $\dot{\theta}_0 = 0\ \text{rad}\,\text{s}^{-1}$. θ y $\dot{\theta}$ frente a t (izquierda) y $\dot{\theta}$ vs. θ (derecha).

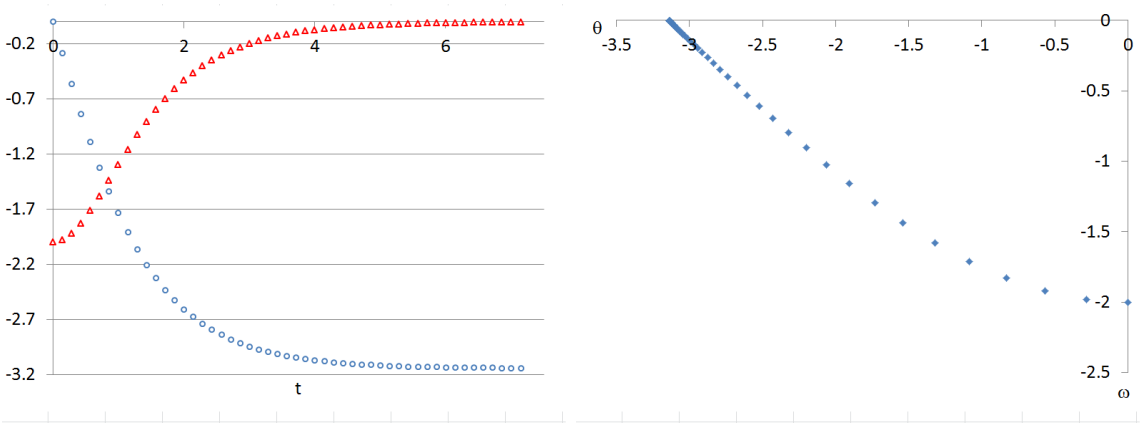


(a) θ (círculos) y $\dot{\theta}$ (triángulos) frente a la función $\theta = \theta_M \,\text{seno}\left(\frac{2\pi}{T}\, t + \varphi_0\right)$ (línea continua)



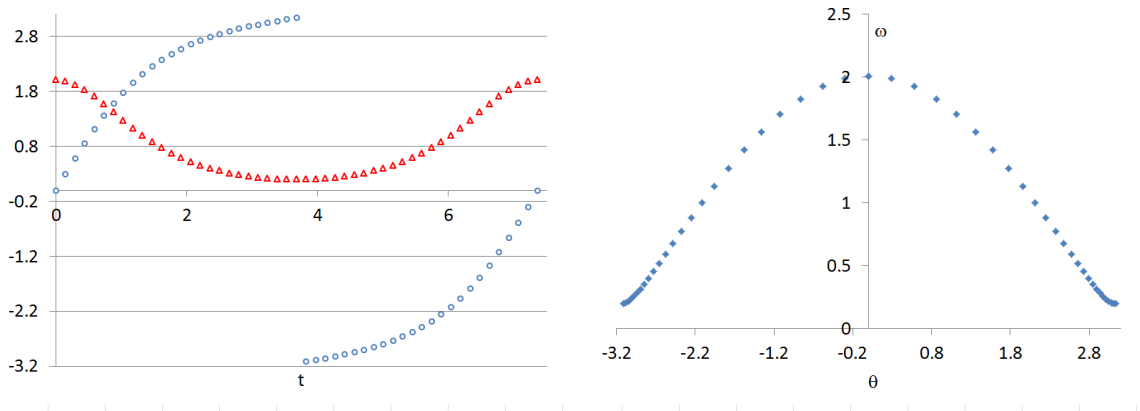
(b) Espacio de fases

Figura 3.7: $\theta_0 = 0$ rad, $\dot{\theta}_0 = 1.999999895801368\ \text{rad}\,\text{s}^{-1}$, $\theta_{\text{máx}} \sim 179{,}96^0$. θ y $\dot{\theta}$ frente a t (izquierda) y $\dot{\theta}$ vs. θ (derecha).



(a) θ (círculos) y $\dot{\theta}$ (triángulos) . (b) Espacio de fases

Figura 3.8: $\theta_0 = 0$ rad, $\dot{\theta}_0 =$ - 2 rad s^{-1}, $\theta_{\text{límite}} = -\pi$. θ y $\dot{\theta}$ frente a t (izquierda) y $\dot{\theta}$ vs. θ (derecha).



(a) θ (círculos) y $\dot{\theta}$ (triángulos) . (b) Espacio de fases

Figura 3.9: $\theta_0 = 0$ rad, $\dot{\theta}_0 = 2.01$ rad s^{-1}. θ y $\dot{\theta}$ frente a t (izquierda) y $\dot{\theta}$ vs. θ (derecha).

Capítulo 4

Tipos de datos

4.1. Horas de luz

4.1.1. Introducción

El objetivo de la siguiente actividad es calcular las horas de sol en un punto sobre la Tierra, conocida su latitud, el día del solsticio de invierno. Para ello haremos uso de nuestros conocimientos en álgebra matricial.

El eje de giro de la Tierra está inclinado respecto del plano de su órbita en torno al Sol. Esta inclinación da lugar a la sucesión de estaciones a lo largo del año (al variar la inclinación con que los rayos llegan a cada punto de la superficie de la Tierra). El ángulo de inclinación, llamado ángulo de la eclíptica, es $\varepsilon = 23^0\ 27'$ (el cálculo ya fue hecho, con una aproximación muy buena, por Eratóstenes en el siglo III a. C.).

Consideremos la Tierra el día del solsticio de invierno, y tomemos como eje X la recta que une el Sol y la Tierra: los rayos del sol tienen dirección (1,0,0). Sea un punto de latitud γ. La figura 4.1.a) muestra la situación a mediodía, momento en que los rayos del Sol inciden más directamente sobre ese punto, en el que hemos dibujado la normal a la superficie de la Tierra. 12 horas más tarde la Tierra habrá girado media vuelta en torno a su eje y la normal a la superficie en nuestro punto será como se indica en la figura 4.1.b). En este momento es de noche, por lo que en algún momento intermedio habrá tenido lugar el crepúsculo; exactamente cuando el vector normal en el punto que nos interesa y la dirección de los rayos solares forman un ángulo de $\pi/2$ o, equivalentemente, cuando su producto escalar es 0.

Representemos el producto escalar de la normal con el sentido de los rayos del sol en función del tiempo, donde hemos tomado como origen de tiempos ($t_0 = 0$) el mediodía, tal como se muestra en la Figura 4.2. En nuestro punto anochece (dejan de llegar los rayos del sol) cuando la normal y los rayos forman un ángulo de 90^0 o, equivalentemente, cuando su producto escalar vale cero. Esto ocurre en un tiempo $t_{crepúsculo}$. La duración del día es, pues, $2 \cdot t_{crepúsculo}$.

4.1.2. Descripción de la tarea

Diseña los siguientes dominios y subalgoritmos e impleméntalos en C/C++. No reinventes la rueda: siempre que sea posible, reutiliza un subalgoritmo que hayas diseñado y programado previamente:

Figura 4.1: Posición relativa Tierra-Sol al mediodía (a) y a medianoche (b)

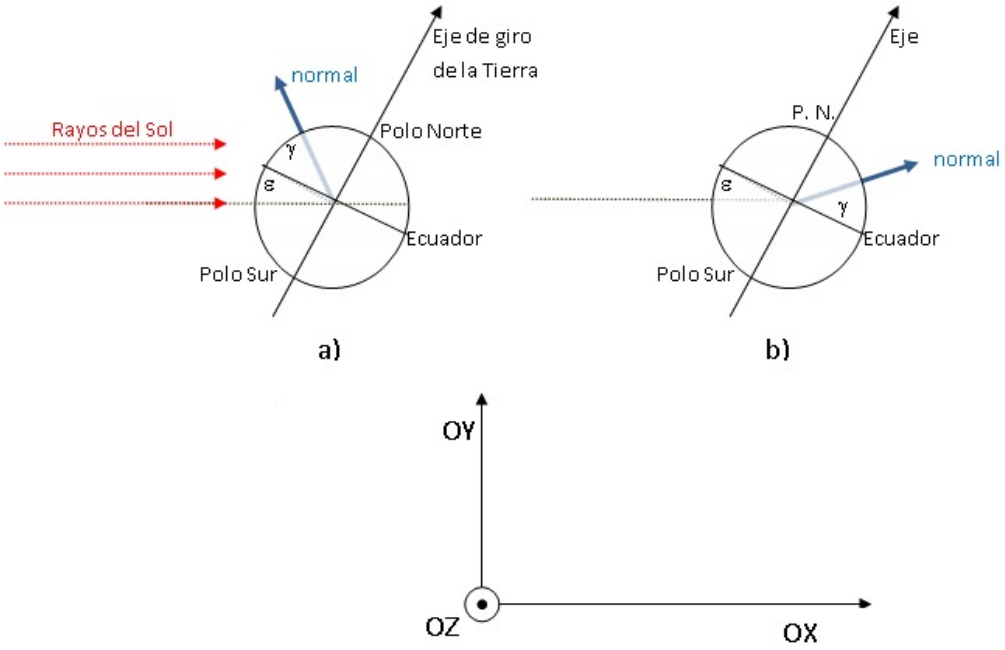
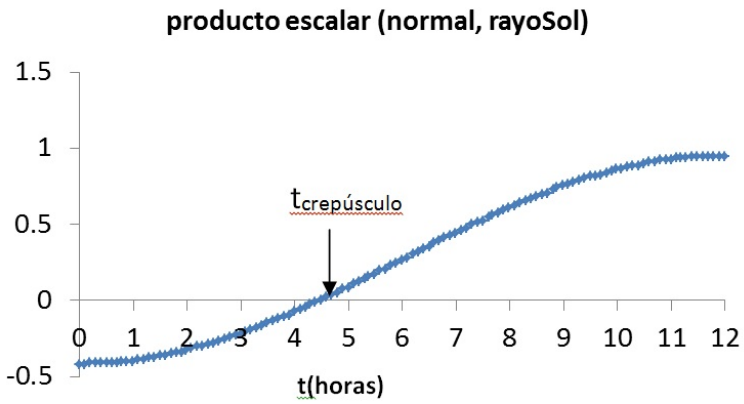


Figura 4.2: Producto escalar entre la normal a la superficie de la Tierra y la dirección de los rayos del sol.




```
DOMINIO AnguloSexagesimal = REGISTRO
    grad, min SON enteros;
    seg ES real;
FIN;
// Representa ángulos en notación sexagesimal
//(grados, minutos, segundos,
// donde minutos, segundos < 60).
```

```
DOMINIO Vector3D = REGISTRO
    x, y, z SON reales;
FIN;
```

sexa2Rad

Dado un dato del dominio *AnguloSexagesimal* que representa un ángulo, α, devuelve un número real (la representación del ángulo en radianes).

construyeAngulo

Dados dos enteros, *grados* y *minutos*, y un real, *segundos*, devuelve un dato del dominio *AnguloSexagesimal* que representa el ángulo con esos valores.

productoMatricial

Dadas dos matrices 3×3, calcula y devuelve su producto.

productoMatrizVector

Dados una matriz 3×3 y un dato del tipo Vector3D, M y $\overrightarrow{u}$, calcula y devuelve el vector $M\overrightarrow{u}$.

productoEscalar

Dados dos datos del tipo Vector3D, $\overrightarrow{u}$ y $\overrightarrow{v}$, devuelve su producto escalar:

$$< \overrightarrow{u}, \overrightarrow{v} >= u_x \cdot v_x \ + \ u_y \cdot v_y \ + \ u_z \cdot v_z \tag{4.1}$$

modulo

Dado un dato del tipo Vector3D, $\overrightarrow{v}$, devuelve su módulo: $|\overrightarrow{v}| = \sqrt{< \overrightarrow{v}, \overrightarrow{v} >}$

productoVectorial

Dados dos datos del tipo Vector3D, $\overrightarrow{v}$ y $\overrightarrow{w}$, devuelve su producto vectorial, $\overrightarrow{v} \times \overrightarrow{w}$

nuevaBase

Dado un dato de tipo Vector3D con módulo 1, $\hat{u}$, el subalgoritmo devuelve dos vectores, $\hat{v}$ y $\hat{w}$, tales que ($\hat{u}$, $\hat{v}$, $\hat{w}$) son una base ortonormal de $\mathbb{R}^3$.

Puedes generarlos mediante las siguientes operaciones:

$$\hat{v} = \frac{\hat{u} \times \hat{i}}{|\hat{u} \times \hat{i}|}, \hat{w} = \hat{u} \times \hat{v} \ , \tag{4.2}$$

donde $\hat{i} = (1, 0, 0)$.

matrizDeVectores

Dados tres datos del tipo Vector3D, ($\overrightarrow{u}$, $\overrightarrow{v}$, $\overrightarrow{w}$), devuelve una matriz 3×3, cuyas filas son las componentes de los vectores.

A modo de ejemplo, si los vectores son: $\overrightarrow{u} = \{1, 2, 3\}$, $\overrightarrow{v} = \{4, 5, 6\}$, $\overrightarrow{u} = \{7, 8, 9\}$, la matriz devuelta es:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

rotacionEjeX

Dado un real, θ, que representa un ángulo (en radianes), devuelve la matriz $R_x(\theta)$, que representa una rotación en 3D, en torno al eje $\overrightarrow{OX}$ y con ángulo θ:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\text{seno}(\theta) \\ 0 & \text{seno}(\theta) & \cos(\theta) \end{pmatrix}$$

rotacionEjeAngulo

Dados un vector unitario en $\mathbb{R}^3$, $\hat{u}$, y un ángulo expresado en radianes, θ, devuelve $R_{\hat{u}}(\theta)$, la matriz que representa una rotación de ángulo θ en torno a la dirección dada por $\hat{u}$.

Ayuda: empieza creando una base ortonormal en $\mathbb{R}^3$ que contenga al vector $\hat{u}$, ($\hat{u}$, $\hat{v}$, $\hat{w}$). La matriz de rotación $R_{\hat{u}}(\theta)$ viene dada por la expresión

$$R_{\hat{u}}(\theta) = T^t R_x(\theta) T \tag{4.3}$$

donde

$$T = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$

Puedes validar el subalgoritmo comprobando que:

- $R_{\hat{u}}(\theta)\hat{u} = \hat{u}$
- $R_{\hat{u}}(\theta)\hat{v} \perp \hat{u}$
- $R_{\hat{u}}(\theta)\hat{w} \perp \hat{u}$
- $R_{\hat{u}}(\theta)\hat{v} \perp R_{\hat{u}}(\theta)\hat{w}$
- $< R_{\hat{u}}(\theta)\hat{v}, \hat{v} >= \cos(\theta)$
- $< R_{\hat{u}}(\theta)\hat{w}, \hat{w} >= \cos(\theta)$

orientacionNormalSol

Dados tres vectores unitarios, ($\hat{n}$, $\hat{e}$, $\hat{i}$) y un número real t que representa un tiempo (en horas), hace rotar el vector $\hat{n}$ un ángulo $\omega \cdot t$ en torno a $\hat{e}$ ($\omega = \pi/12$ *radianes* h^{-1}), y devuelve el producto escalar del vector resultante por el vector $\hat{i}$:

$$< R_{\hat{e}}(\omega \cdot t)\hat{n},\ \hat{i} > \tag{4.4}$$

tCrepusculo

Dados tres vectores unitarios, $\hat{n}, \hat{e}, \hat{i}$ y dos reales, t_0, t_f, de los que además se sabe que $< R_{\hat{e}}(\omega \cdot t_0)\ \hat{n},\ \hat{i} >< 0$ y $< R_{\hat{e}}(\omega \cdot t_f)\ \hat{n},\ \hat{i} >> 0$.

El subalgoritmo devuelve un real, el tiempo $t_{crep} \in [t_0, t_f]$ pasado el cual el vector $\hat{n}$ ha girado hasta formar un ángulo de $\pi/2$ radianes con el vector $\hat{i}$. Esto es: el tiempo para el cual $< R_{\hat{e}}(\omega \cdot t_{crep})\ \hat{n},\ \hat{i} >= 0$.

Ayuda: Utiliza el teorema de Bolzano (Apéndice D). Deseamos conocer el resultado con una precisión de 10^{-3} segundos.

horasSol

Dados dos ángulos, ϵ y γ, que representan respectivamente el ángulo de la eclíptica de la Tiera y la latitud de un punto, expresados en notación sexagesimal (grados, minutos, segundos), devuelve un número real (las horas que dura el día en un punto de latitud γ durante el solsticio de invierno).

Ayuda: El algoritmo calcula los vectores unitarios correspondientes al eje de giro de la Tierra y a la normal a su superficie a mediodía en un punto de latitud ϵ:

$$\hat{e} = \begin{pmatrix} \cos(\pi/2 - \epsilon) \\ \text{seno}(\pi/2 - \epsilon) \\ 0 \end{pmatrix} \quad \hat{n} = \begin{pmatrix} -\cos(\epsilon + \gamma) \\ \text{seno}(\epsilon + \gamma) \\ 0 \end{pmatrix}$$

A continuación, busca el tiempo t_{crep} transcurrido el cual la normal se hace perpendicular a $\hat{i} = (1, 0, 0)$. El algoritmo devuelve como resultado $2 \cdot t_{crep}$.

4.1.3. Ejemplos de ejecución

Puedes poner a prueba tu programa calculando la duración del día el solsticio de invierno en distintas ciudades del mundo:

```
Ciudad    Latitud           Horas de sol
 La Habana  (22º 58')      10.59 horas (10 h 35 m 15.43 s)
 Zaragoza   (41º 39')       8.97 horas ( 8 h 58 m 27.13 s)
 Reykjavik  (64º 3')        3.59 horas ( 3 h 35 m 39.53 s)
```

4.1.4. Ayudas

Es interesante comparar estos resultados con los valores reales[1]. La siguiente tabla muestra, para distintas ciudades del mundo, las horas de salida y puesta del sol, la duración

[1] http://www.timeanddate.fasterreader.eu/pages/es/sunrise-calc-es.html
http://www.fomento.gob.es/salidapuestasol/2016/Zaragoza-2016.txt

del día y la diferencia entre la duración real y la obtenida con nuestro programa:

Ciudad	salida del sol	puesta del sol	duración del día	diferencia
La Habana	07 : 07	17 : 49	10 horas 42 minutos	7 minutos
Zaragoza	08 : 27	17 : 36	9 horas 9 minutos	11 minutos
Reikiavik	11 : 22	15 : 30	4 horas 8 minutos	33 minutos

Los tiempos reales son mayores que los calculados en nuestro programa. La razón es la dispersión de la luz en la atmósfera, que hace que en un punto amanezca (se empiece a ver el sol) antes de que los rayos del sol lleguen directamente a él y anochezca (deje de verse) después de que dejen de hacerlo. Cuanto más lejos está un punto del trópico de Capricornio mayor es el espesor de la capa de aire que la luz debe atravesar, y más se acentúa el efecto de la dispersión.

4.2. Método de Gauss-Jordan

4.2.1. Introducción

En esta actividad realizaremos una implementación, **usando exclusivamente el lenguaje de programación C**, del método de diagonalización de Gauss–Jordan, con el objetivo de calcular el determinante de una matriz y, si éste es no nulo, su inversa. Las matrices que usaremos estarán definidas sobre el cuerpo de los números racionales, lo que nos permitirá, realizando computación entera, evitar los problemas de precisión relacionados con la representación de los números reales en el ordenador. Los objetivos principales en esta actividad, que serán especialmente tenidos en cuenta para su evaluación, son:

- El uso de registros
- El uso de punteros para realizar la simulación del paso por referencia
- El uso de las funciones de entrada de C: `printf` y `scanf`.

No son objetivos de esta práctica el uso de punteros en lugar de arrays ni la aritmética de punteros.

Representación de $\mathbb{Q}$

Como es usual, representaremos un número racional $q = \frac{n}{d} \in \mathbb{Q}$ mediante un par de números enteros $(n, d) \in \mathbb{Z} \times \mathbb{Z}^*$, donde $\mathbb{Z}^* = \mathbb{Z} \setminus \{0\}$, n es el *numerador* de q, y $d \neq 0$ su *denominador*. Obviamente, cualquier número racional q puede ser representado por una cantidad numerable de estos pares. Más precisamente, los pares (n, d) y (n', d') representan al mismo elemento de $\mathbb{Q}$ si y sólo si $n \cdot d' = n' \cdot d$. Nosotros utilizaremos uno concreto (n, d), llamado *representante canónico* o *fracción irreducible*, que satisface las siguientes condiciones:

- $d > 0$.
- n y d son primos entre sí; en particular, si $\frac{n}{d} \in \mathbb{Z}$ entonces $d = 1$.
- Si $n = 0$ entonces $d = 1$; esto es, el representante canónico de $0 \in \mathbb{Q}$ es el par $(0, 1)$.

Durante la ejecución de un subalgoritmo podemos usar cualquier par de $\mathbb{Z} \times \mathbb{Z}^*$ para representar a un elemento de $\mathbb{Q}$ pero, al finalizar, cualquier resultado deberá estar expresado mediante su representante canónico.

Para representar los números racionales utilizaremos las siguiente estructura de datos:

```
struct __racional {
    int n, d;
};
typedef struct __racional Racional;
```

Y podremos utilizar los siguientes algoritmos:

```
// Entrada: un racional p
// Salida:  un booleano, representado por los enteros 0 (false) y
//          1 (true) en C. Devuelve true si p=0 y false en otro caso.
int esCeroR(Racional p) {
    return !p.n;
}
```

```
// Entrada: no hay
// Salida:  devuelve el racional que representa al entero 1.
Racional unidadR() {
    Racional res = {1,1};
    return res;
}

// Entrada: no hay
// Salida:  devuelve el racional que representa al entero 0.
Racional ceroR() {
    Racional res = {0,1};
    return res;
}

// Entrada: dos enteros arbitrarios no nulos
// Salida:  el máximo común divisor de los valores absolutos de
//          los enteros de entrada.
// Utiliza: la función     int abs(int)    de la librería <stdlib.h>
//      que devuelve, como entero, el valor absoluto de su parámetro
int mcdEuclides(int m, int n) {
    int aux;
    m = abs(m); n=abs(n);
    if (m<n) { aux = m; m = n; n = aux; }
    aux = m%n;
    while (aux!=0) {
        m = n; n = aux; aux = m%n;
    }
    return n;
}
```

4.2.2. Descripción de la tarea

En esta sección, describimos las dos tareas que debes completar. Por un lado, completamos la funcionalidad de los números racionales y, por otro lado, diseñamos e implementamos los algoritmos para el método de Gauss-Jordan.

Funcionalidad de los números racionales

Para completar la funcionalidad de esta representación de los números racionales se deben diseñar e implementar los subalgoritmos siguientes:

simplificaR: Dado un dato de tipo `Racional` lo modifica para que esté en forma canónica.

leeR: Lee de teclado el numerador y denominador de un racional, asegurando que éste es no nulo, y devuelve un dato de tipo `Racional` en forma canónica.

escribeR: Escribe en pantalla el numerador y denominador de un `Racional` en el formato `+nnnnn/ddddd`. Esto es, el numerador se escribirá siempre con signo, y se dedicarán al menos cinco espacios para los dígitos del numerador y del denominador. Ejemplo,

el racional $\frac{321}{101}$ será visualizado en pantalla como `+321/101`, quedando a su izquierda y derecha dos espacios en blanco ya que sólo se han escrito tres dígitos para el numerador y el denominador.

restaR: Dados dos números racionales p y q devuelve su diferencia $p - q$ en forma canónica.

multiplicaR: Dados dos números racionales p y q devuelve su producto $p \cdot q$ en forma canónica.

divideR: Dados dos números racionales p y q devuelve su cociente p/q en forma canónica. Si $q = 0$ el resultado está indefinido.

Método de Gauss-Jordan

A continuación, recordamos el método de Gauss–Jordan y describimos el objetivo principal de esta actividad. Sólo consideraremos matrices de los espacios $E_{n\times n}(\mathbb{Q})$, donde $n \leq 10$; esto es, matrices cuadradas sobre el cuerpo de los racionales de tamaño $n \times n$. Deberás diseñar e implementar el siguiente subalgoritmo:

GaussJordan: Dada una matriz $A \in E_{n\times n}(\mathbb{Q})$ de racionales calcula su determinante. Además, si éste es no nulo calcula A^{-1}. En otro caso, si el determinante es 0, devuelve la forma escalonada reducida de la matriz de entrada. La matriz dato no debe ser modificada en ningún caso.

Para ello diseña e implementa los siguientes subalgoritmos adicionales:

copiarMatriz: dada una matriz de racionales de tamaño $n \times n$, construye y devuelve una copia.

matrizIdentidad: dado un entero $n \leq 10$, construye y devuelve la matriz identidad de tamaño n, $I_n \in E_{n\times n}(\mathbb{Q})$.

buscarEltoNoNuloEnColumna: Dada una matriz $A \in E_{n\times n}(\mathbb{Q})$, un índice de fila, $0 \leq \text{fila} < n$, y un índice de columna, $0 \leq \text{col} < n$, calcula el menor índice $i \geq \text{fila}$ tal que $A_{i,\text{col}} \neq 0$. Si $A_{i,\text{col}} = 0$ para todo índice $\text{fila} \leq i < n$ el subalgoritmo devuelve el entero -1.

buscarEltoNoNuloEnFila: Dada una matriz $A \in E_{n\times n}(\mathbb{Q})$ y un índice de fila, $0 \leq \text{fila} < n$, calcula el menor entero $0 \leq j < n$ tal que $A_{\text{fila},j} \neq 0$. Si $A_{\text{fila},j} = 0$ para todo índice $0 \leq j < n$ el subalgoritmo devuelve el entero -1.

restaFilas: Dada una matriz $A \in E_{n\times n}(\mathbb{Q})$, un racional $\lambda \in \mathbb{Q}$ y dos índices de fila, $0 \leq \text{f1}, \text{f2} < n$, modifica la matriz A restando a cada elemento de la fila f1 el correspondiente elemento de la fila f2 multiplicado por λ; esto es, el valor del elemento $A_{\text{f1},j}$ pasará a ser $A_{\text{f1},j} - \lambda A_{\text{f2},j}$, para cada $0 \leq j < n$.

Diseño del método de `GaussJordan`

1. Creamos una copia B de la matriz A y asignamos en una variable C la matriz identidad I_n. En la copia B realizaremos las trasformaciones elementales de filas necesarias para obtener su forma escalonada o reducirla a la matriz identidad. Haciendo simultáneamente las mismas transformaciones sobre C obtendremos A^{-1} en el segundo caso.
2. Usaremos una variable `filaPivote`, que inicializaremos a 0, para indicar la siguiente fila en la que buscamos un pivote.
3. Para cada columna de B realizaremos las tareas siguientes:

   a) Buscamos el primer elemento no nulo de la columna a partir de la fila `filaPivote`.
   b) Si no se encuentra, la matriz no será invertible y su determinante será 0, pero debemos continuar hasta encontrar su forma escalonada. Para ello basta pasar a la siguiente columna de la matriz sin incrementar el índice de la fila donde debe buscarse el siguiente pivote (`filaPivote`).
   c) En caso contrario, si se encuentra un pivote en dicha columna, la fila en la que se encuentra debe ser intercambiada con la fila `filaPivote`, para después anular todos los elementos de la columna que quedan bajo el pivote usando el subalgoritmo `restaFilas`, haciendo las mismas transformaciones en la matriz C. Como en la fila `filaPivote` ya hemos encontrado un pivote, esta variable debe ser incrementada antes de pasar a la siguiente columna.

   Nótese que, en este paso, no reducimos a 1 el valor de los pivotes.

4. Terminado el paso (3) la matriz B es triangular y su determinante es el producto de los elementos que hay en su diagonal. entonces $|A| = (-1)^k |B|$, donde k es el número de permutaciones de filas que hemos realizado en (3).
5. Si el determinante que hemos calculado es 0, el resultado es la matriz escalonada B en la que todavía hay que reducir a 1 el valor de los pivotes. Para ello localizaremos en cada fila la posición del pivote, usando el subalgoritmo `buscarEltoNoNuloEnFila`, y dividiremos todos los elementos de dicha fila por el valor del pivote.
6. Si el determinante es no nulo, entonces la matriz es invertible. El resultado será la matriz C después de realizar sobre ella las mismas transformaciones que sean necesarias para transformar B en la matriz identidad. Para ello realizaremos las siguientes operaciones para cada fila f de B comenzando por la última:

   a) Dividiremos todos los elementos de la fila, tanto en B como en C, por el elemento de B que, en esa fila, está en la diagonal.
   b) Para cada fila por encima de ésta anularemos los elementos de la columna f en B usando el subalgoritmo `restaFilas`, y aplicaremos la misma transformación a las filas de C.

Existen varias variantes del algoritmo que conducen al mismo resultado. En particular, en el paso (3) se puede reducir a 1 el valor de todos los pivotes, pero entonces es necesario calcular simultáneamente el valor del determinante.

El programa principal

El programa principal deberá limitarse a solicitar que el usuario introduzca datos desde el teclado y mostrar los resultados en pantalla después de llamar al subalgoritmo `GaussJordan`.

4.2.3. Ayuda

En esta sección mostramos, para dos matrices, los resultados intermedios que deberían obtenerse tras cada iteración de los bucles indicados en los pasos (3) y (5) ó (6), según corresponda, del algoritmo descrito en la sección anterior.

Supongamos que la matriz de entrada es

$$A = \begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +1/1 & -3/1 \\ +4/1 & +3/1 & -2/1 \end{pmatrix}$$

Esta matriz es invertible, por lo que el resultado será el valor final de la matriz C.

B	C	
$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +1/1 & -3/1 \\ +4/1 & +3/1 & -2/1 \end{pmatrix}$	$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ +0/1 & +0/1 & +1/1 \end{pmatrix}$	Inicio
		Ejecución paso (3)
$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +1/1 & -3/1 \\ +0/1 & -5/1 & -14/1 \end{pmatrix}$	$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ -4/1 & +0/1 & +1/1 \end{pmatrix}$	Fin tratamiento columna 0
$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +1/1 & -3/1 \\ +0/1 & +0/1 & -29/1 \end{pmatrix}$	$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ -4/1 & +5/1 & +1/1 \end{pmatrix}$	Fin tratamiento columna 1
$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +1/1 & -3/1 \\ +0/1 & +0/1 & -29/1 \end{pmatrix}$	$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ -4/1 & +5/1 & +1/1 \end{pmatrix}$	Fin tratamiento columna 2

B	C	
Determinante: -29;		Ejecución paso (6)
$\begin{pmatrix} +1/1 & +2/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ +0/1 & +0/1 & +1/1 \end{pmatrix}$	$\begin{pmatrix} +17/29 & +15/29 & +3/29 \\ +12/29 & +14/29 & -3/29 \\ +4/29 & -5/29 & -1/29 \end{pmatrix}$	Fin tratamiento fila 2
$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ +0/1 & +0/1 & +1/1 \end{pmatrix}$	$\begin{pmatrix} -7/29 & -13/29 & +9/29 \\ +12/29 & +14/29 & -3/29 \\ +4/29 & -5/29 & -1/29 \end{pmatrix}$	Fin tratamiento fila 1
$\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ +0/1 & +0/1 & +1/1 \end{pmatrix}$	$\begin{pmatrix} -7/29 & -13/29 & +9/29 \\ +12/29 & +14/29 & -3/29 \\ +4/29 & -5/29 & -1/29 \end{pmatrix}$	Fin tratamiento fila 0

Supongamos, ahora, que la matriz de entrada es $A = \begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +0/1 & -3/1 \\ +4/1 & +8/1 & -2/1 \end{pmatrix}$.

Puesto que la segunda columna es igual a la primera multiplicada por dos su determinante es 0 y la matriz resultado deberá ser el valor final de B.

$$\underset{B}{\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +0/1 & -3/1 \\ +4/1 & +8/1 & -2/1 \end{pmatrix}} \quad \underset{C}{\begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ +0/1 & +0/1 & +1/1 \end{pmatrix}}$$ Inicio

Ejecución paso (3)

$$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +0/1 & -3/1 \\ +0/1 & +0/1 & -14/1 \end{pmatrix} \quad \begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ -4/1 & +0/1 & +1/1 \end{pmatrix}$$ Fin tratamiento columna 0

$$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +0/1 & -3/1 \\ +0/1 & +0/1 & +0/1 \end{pmatrix} \quad \begin{pmatrix} +1/1 & +0/1 & +0/1 \\ +0/1 & +1/1 & +0/1 \\ -4/1 & -14/3 & +1/1 \end{pmatrix}$$ Fin tratamiento columna 1

Al tratar la fila de índice 2 no se encuentra pivote, por lo que no se hacen nuevas transformaciones en las matrices y el paso (3) termina. Ahora el determinante es 0 y se ejecuta el paso (5), en el que sólo se transforma la matriz B. Al tratar, en la primera iteración, la fila de índice 0 no se hace ninguna transformación, pues el pivote ya tiene valor 1. En la siguiente iteración, los elementos de la fila de índice 1 se dividen por $-3/1$ obteniendo la siguiente matriz:

$$\begin{pmatrix} +1/1 & +2/1 & +3/1 \\ +0/1 & +0/1 & +1/1 \\ +0/1 & +0/1 & +0/1 \end{pmatrix}$$

Y ya no se realiza ninguna iteración más, ya que no hay pivote en la fila de índice 2.

4.3. Modelo cuántico del átomo de Hidrógeno

4.3.1. Introducción

En esta actividad estudiaremos el modelo del átomo de hidrógeno dado por la física cuántica (sin consideraciones relativistas). Compararemos algunas de sus previsiones con las del modelo atómico de Bohr [10]. La solución detallada de la ecuación de Schrödinger puede encontrarse en [9].

Modelo atómico de Bohr

Según el modelo atómico de Bohr, el electrón del átomo de hidrógeno describe una órbita circular en torno al núcleo. Sólo es posible un conjunto discreto de órbitas, que se caracterizan mediante un entero positivo n.

La órbita n-ésima tiene radio $r_n = n^2 a_0$ y la energía potencial del electrón es:

$$V_n = -\frac{1}{n^2}\frac{Kq_e^2}{a_0} \qquad \text{donde } a_0 = \frac{\hbar^2}{K\mu_e q_e^2} \ .$$

Hay que tener en cuenta que a_0, $\hbar$, K, μ_e y q_e son, respectivamente, el radio de Bohr, la constante de Planck reducida, la constante de Coulomb, la masa reducida y la carga del electrón.

De este modo, en la órbita n-ésima el radio y la energía potencial vienen dados por:

$$\frac{r_n}{a_0} = n^2 \qquad \frac{a_0}{Kq_e^2}V_n = -\frac{1}{n^2} \ .$$

Modelo cuántico

El estado del electrón viene dado por una función de onda $\Psi_{n,l}(r,\theta,\phi)$ que se obtiene al resolver la ecuación de Schrödinger. La función de onda está caracterizada por dos números cuánticos: n (entero positivo), que caracteriza el nivel energético, y l (en el rango $0 \le l < n$), que caracteriza el momento angular del electrón.

A partir de $\Psi_{n,l}(r,\theta,\phi)$ se calcula la densidad radial de probabilidad del electrón, $P_{n,l}(r)\mathrm{d}r$, definida como la probabilidad de encontrar al electrón a una distancia del núcleo entre r y $r + \mathrm{d}r$:

$$P_{n,l}(r)\mathrm{d}r = C_{n,l}e^{-\rho}\rho^{2l+2}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho \ ,$$

donde $\rho = \frac{2r}{na_0}$, $C_{n,l} = \frac{(n-l-1)!}{2n(n+l)!}$, a_0 es el radio de Bohr y $L_{n-l-1}^{2l+1}(\rho)$ es un polinomio de Laguerre generalizado.

El valor esperado de la posición radial del electrón, $< r >$, se calcula:

$$< r >= \int_0^\infty rP_{n,l}(r)\mathrm{d}r = \frac{na_0}{2}C_{n,l}\int_0^\infty e^{-\rho}\rho^{2l+3}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho$$

y el valor esperado de la energía potencial:

$$\langle V(r)\rangle = \int_0^\infty \frac{-Kq_e^2}{r}P_{n,l}(r)\mathrm{d}r = -\frac{2Kq_e^2}{na_0}C_{n,l}\int_0^\infty e^{-\rho}\rho^{2l+1}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho \ .$$

4.3.2. Descripción de la tarea

Para llevar a cabo la tarea, en primer lugar, crea el registro `Polinomio`, para representar polinomios de grado menor o igual a 99, según la especificación:

```
Dominio Polinomio = registro
    grado ES ENTERO
    coeficiente ES ARRAY DE 100 REALES
Fin
```

Después, diseña y programa en C/C++ los subalgoritmos que se indican a continuación. Si lo estimas necesario, puedes añadir los subalgoritmos extra que consideres necesarios.

factorial

Dado un número entero $n \geq 0$, devuelve $n!$.

combinatorio

Dados dos enteros, m y n, devuelve el número combinatorio $\binom{m}{n}$ y lo calcula de forma recursiva teniendo en cuenta que:

$$\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n} .$$

Cnl

Dados dos enteros, n y l, devuelve el número real $\frac{(n-l-1)!}{2n(n+l)!}$.

laguerre

Dados dos números enteros, a y b, devuelve el polinomio generalizado de Laguerre L_a^b: un polinomio de grado a cuyo coeficiente i-ésimo es $(-1)^i \frac{1}{i!}\binom{a+b}{a-i}$.

escribePolinomio

Dado un polinomio p muestra sus coeficientes por pantalla. A modo de ejemplo, este subalgoritmo mostraría el polinomio $p(x) = 1 - x^2 + 3x^5$ como:

```
1  0  -1  0  0  3
```

Hay que tener en cuenta que este subalgoritmo no devuelve ningún resultado al algoritmo que lo invoca.

multiplicaPolinomios

Dados dos polinomios, p y q, devuelve su producto: $r(x) = p(x) * q(x)$.

multiplicaPxn

Dados un polinomio $p(x)$ y un entero $n \geq 0$, devuelve el polinomio $q(x) = x^n p(x)$.

integral

Dado un polinomio p, devuelve el valor de la integral $\int_0^\infty p(x)e^{-x}\mathrm{d}x$.
Ayuda: recuerda que $\int_0^\infty ax^n e^{-x}\mathrm{d}x = an!$ (este resultado se puede demostar por inducción, integrando por partes).

resultados

Dados dos enteros, n y l (con $0 \leq l < n$), calcula y devuelve los siguientes tres resultados para un electrón que se encuentre en el estado (n, l):

- la probabilidad de que el electrón se halle a una distancia del núcleo atómico entre 0 e infinito. En otras palabras, calcula:

$$C_{n,l} \int_0^\infty e^{-\rho}\rho^{2l+2}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho \ .$$

  Nota: Por supuesto, esta integral debe dar como resultado 1. Nos sirve para validar los subalgoritmos anteriores.
- el valor esperado de r (la distancia al núcleo), normalizado al radio de Bohr. En otras palabras, calcula:

$$\frac{< r >}{a_0} = \frac{n}{2} C_{n,l} \int_0^\infty e^{-\rho}\rho^{2l+3}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho \ .$$

- el valor esperado de la energía potencial del electrón, normalizado en unidades de $\frac{Kq_e^2}{a_0}$. En otras palabras, calcula:

$$\frac{a_0}{Kq_e^2}\langle V(r)\rangle = -\frac{2}{n} C_{n,l} \int_0^\infty e^{-\rho}\rho^{2l+1}\left[L_{n-l-1}^{2l+1}(\rho)\right]^2 \mathrm{d}\rho \ .$$

Programa Principal

Programa que, utilizando los subalgoritmos anteriores, muestra por pantalla los valores de la probabilidad total, y los valores esperados de r y $V(r)$ para todos los estados posibles del átomo de H con $n \leq 5$.

4.3.3. Ejemplo de ejecución

```
n  l     Prob      <r>      <V>

1  0     1         1.5      -1

2  0     1         6        -0.25
2  1     1         5        -0.25

3  0     1         13.5     -0.111111
```

```
3  1     1        12.5    -0.111111
3  2     1        10.5    -0.111111

4  0     1        24      -0.0625
4  1     1        23      -0.0625
4  2     1        21      -0.0625
4  3     1        18      -0.0625

5  0     1        37.5    -0.04
5  1     1        36.5    -0.04
5  2     1        34.5    -0.04
5  3     1        31.5    -0.04
5  4     1        27.5    -0.04
```

4.4. Dígitos de control: los códigos de cuenta cliente (ccc) y el código IBAN

4.4.1. Introducción

El uso de dígitos (o caracteres) de control (*check digit*) es un mecanismo utilizado principalmente para detectar errores al teclear o en la transmisión de un dato. Un primer ejemplo es la letra del DNI. Para determinar esta letra basta calcular el resto de la división entera por 23 (el resto módulo 23) del número del DNI. Así obtenemos un entero mayor o igual que cero y estrictamente menor que 23, al que se asocia una letra de acuerdo a la tabla siguiente.

T	R	W	A	G	M	Y	F	P	D	X	
0	1	2	3	4	5	6	7	8	9	10	
B	N	J	Z	S	Q	V	H	L	C	K	E
11	12	13	14	15	16	17	18	19	20	21	22

Por ejemplo, si el número de tu DNI es `12345678` entonces la letra que te corresponde es la 'Z'.

El código de cuenta cliente (ccc) y el International Bank Account Number (IBAN)

En España, los números de las cuentas bancarias, y de otros productos financieros, se han venido identificando mediante un número de 20 dígitos, llamado "código cuenta cliente", abreviadamente **ccc**. De estos 20 dígitos, los cuatro primeros identifican a la entidad bancaria, los cuatro siguientes a la sucursal y sólo los 10 últimos corresponden realmente al número de la cuenta. Los dos dígitos restantes, el noveno y décimo empezando por la izquierda, son dígitos de control (**DC**), que se pueden calcular a partir de los restantes. Puedes obtener más información sobre el código ccc en Internet, por ejemplo en [8].

A partir del 1 de enero de 2014, para facilitar las transacciones intracomunitarias, los países de la Unión Europea han adoptado el código **IBAN** para identificar las cuentas inequívocamente en todo el territorio (ver [16]). El código IBAN está formado por dos caracteres, que identifican al país, conforme al estándar ISO 3166-1 alpha-2 ("ES" para España), seguidos de dos nuevos dígitos de control y, finalmente, el código de cuenta de acuerdo a la legislación del país: el ccc en España. Por tanto, el código IBAN tiene una longitud variable, pero fija dentro de cada país.

Los dígitos de control de los códigos **ccc** e **IBAN** se pueden calcular con los algoritmos que se describen a continuación.

Dígitos de control del código ccc El *segundo dígito* se construye a partir del número de cuenta (últimos 10 dígitos) de la forma siguiente:

1. Sea $x = \sum_{i=0}^{9}(d_i 2^i \bmod 11)$, donde d_i es el valor del dígito i-ésimo del número de cuenta (d_0 es el primer dígito, el más significativo, y d_9 el último, el situado más a la derecha) y 'mod' es el resto de la división entera.
2. Calcula el valor $11 - (x \bmod 11)$. Si es menor que 10, ese valor es el dígito de control, pero si es 10 u 11 entonces el dígito de control es 1 ó 0, respectivamente.

El *primer dígito de control* se construye con el mismo algoritmo aplicado al número formado por dos ceros iniciales seguidos por el código de la entidad y, a continuación, el de la sucursal.

Ejemplo. Suponed que el código de la entidad es `1234`, el de la sucursal `5678` y el número de cuenta es `1234567890`. El primer dígito de control será 0 y el segundo 6, que se obtienen al aplicar el algoritmo anterior a las secuencias de dígitos `0012345678` y `1234567890`. Observad que los posibles ceros a la izquierda son importantes, ya que modifican la posición de los restantes dígitos y, por lo tanto, la potencia de dos por la que se deben multiplicar. En consecuencia, el código ccc calculado será `1234 5678 06 1234567890`.

Dígitos de control del código IBAN

1. Coloca detrás del código de cuenta (el que sea en función del país), las dos letras que identifican al país y añade dos ceros.
2. Reemplaza cada letra (el código de cuenta puede contener letras en algunos países) por los dígitos de un número de acuerdo a la regla A = 10, B = 11,..., Z = 35. En estos códigos no hay letras con tildes ni eñes.
3. En el paso anterior has obtenido un número bastante grande (en este caso puedes ignorar los ceros a la izquierda si existen). Calcula su resto módulo 97 y resta ese valor de 98. Los dígitos de control son los del resultado, añadiendo a la izquierda un '0' si es menor que 10.

Ejemplo. Para el código ccc que hemos obtenido antes, y suponiendo que corresponde a una cuenta de un banco español, en el primer paso construimos la cadena de caracteres "`123456780612345678 90ES00`". En el segundo sustituimos la 'E' por "14" y la 'S' por "28" para obtener "`12345678061234567890142800`". El resto módulo 97 de este número es 30. Así, los dígitos de control serán $98 - 30 = 68$, y el código completo, que se suele escribir en grupos de cuatro caracteres: `ES68 1234 5678 0612 3456 7890`.

Ejemplo. ([16]) Supongamos que el número de cuenta "`WEST12345698765432`" corresponde a un banco del Reino Unido, que se identifica por las letras "`GB`". En el primer paso obtendríamos la cadena "`WEST12345698765432GB00`". Al reemplazar todas las letras que aparecen por los correspondientes números se obtiene "`32142829123456987654321611 00`". Si consideramos su contenido como un entero, obtendremos 16 al calcular su resto módulo 97, por lo que los dos dígitos de control son $98 - 16 = 82$ y el código IBAN correspondiente es `GB82 WEST 1234 5698 7654 32`.

Cálculo del resto de la división entera de un número muy grande representado por una cadena de caracteres

Para calcular los dígitos de control del código IBAN es necesario operar con números de unos 30 dígitos, que no se pueden representar con los tipos enteros estándar (`int` ó `unsigned int`) de C/C++. Afortunadamente, el algoritmo de la división, que aprendimos en la escuela, nos ofrece una forma sencilla para calcular el resto por la división entera de un número muy grande si el divisor lo podemos representar con el tipo `int`, como es el caso de 97. La idea es la siguiente.

Consideramos como entradas una cadena de caracteres D que representa al dividendo (por tanto sólo contiene dígitos de '0' a '9', y un entero, el divisor `d`.

1. Inicializa una variable entera r con el valor 0.
2. Añade por la derecha a r dígitos de D tomados por su izquierda hasta que r tenga 9 dígitos o bien no queden dígitos de D por añadir. Por ejemplo, si $4 = 31$ y queremos 'añadirle' por la derecha el dígito '7', tras hacer esta operación r deberá contener el entero 317 (esto es, el valor inicial de r por 10 más el valor numérico del dígito).
3. Almacena en r el resto de la división entera $r \bmod d$.
4. Repite los pasos (2) y (3) mientras queden dígitos de D por tratar. Cuando termines r contiene el resto $D \bmod r$.

Ejemplo. Supongamos $D =$ "`321428291234569876543216 1100`" y $d = 97$. Inicializamos $r = 0$. Tras el segundo paso obtenemos $r = 321428291$, cuyo resto por 97 es 70. Así, $r = 70$ tras esta primera iteración. Como quedan dígitos de D por tratar, repetimos los pasos anteriores obteniendo (2) $r = 702345698$, (3) $r = 29$. Realizamos una nueva iteración: (2) $r = 297654321$, (3) $r = 24$. Y terminamos con una última: (2) $r = 2461100$, (3) $r = 16$, que es el resultado.

4.4.2. Descripción de la tarea

Escribe, en C/C++, subalgoritmos que realicen las siguientes operaciones. Te recomendamos que escribas las funciones en el orden en que aparecen a continuación y que, una vez escritas, las pruebes con diferentes datos de entrada antes de continuar con la siguiente.

escribeIBAN

Dada una cadena de caracteres, la escribe en pantalla del siguiente modo: escribe grupos de cuatro caracteres, dejando un blanco entre cada par de grupos.

Ejemplo. Si la entrada del subalgoritmo es "`ABCDEFGHIJKLM`" muestra en pantalla

```
ABCD EFGH IJKL M
```

resto

Recibe como entrada una cadena (dividendo), que contiene la representación decimal de un entero no negativo, y un entero (divisor). Devuelve el resto de la división entera del dividendo por el el divisor.

dcCCCaux

Recibe como entrada una cadena de caracteres de longitud arbitraria cuyos elementos son todos dígitos. Su salida es el entero $\sum_{i=0}^{n-1}(d_i 2^i \bmod 11)$, donde n es la longitud de la cadena y d_i el valor como entero del i-ésimo carácter (un dígito) de la cadena. Si la cadena no contiene dígitos el resultado está indefinido.

dcCCC

Recibe tres cadenas que representan una entidad bancaria, una sucursal y un número de cuenta (las tres contienen únicamente dígitos y se supone que tienen longitudes 4, 4 y 10, respectivamente). Devuelve una cadena de dos caracteres, que son los dígitos de control de la cuenta ccc representada por las cadenas de entrada.

dcIBAN

Tiene como entradas dos cadenas. La primera tiene longitud 2, y supondremos que es el código de un país según el estándar ISO 3166-1 alpha-2 (por ejemplo "ES"). La segunda, que puede tener cualquier longitud y sólo contiene letras mayúsculas (excluidas la 'Ñ' y las vocales con tilde) y dígitos, se supone que representa un número de cuenta válido en dicho país. Devuelve como resultado una cadena de dos caracteres: los dígitos de control del código IBAN correspondiente a esa cuenta en ese país.

Escribe un programa que solicite al usuario los códigos de su banco, sucursal y el número de cuenta, y escriba en pantalla el código ccc correspondiente. A continuación, el programa mostrará el códgio IBAN de dicha cuenta, suponiendo que es de un banco español. Finalmente, el programa solicitará al usuario un código de país y una cadena de caracteres, que suponemos es un código de cuenta en ese país, y mostrará el código IBAN de esta cuenta.

4.4.3. Ejemplos de ejecución

```
Introduce codigo banco: 1234
Introduce codigo sucursal: 5678
Introduce codigo cuenta: 1234567890

ccc: 1234 5678 06 1234567890
IBAN: ES68 1234 5678 0612 3456 7890

Introduce codigo pais: GB
Introduce codigo cuenta: WEST12345698765432

IBAN: GB82 WEST 1234 5698 7654 32
```

4.4.4. Ayudas

En el apéndice E se recuerda cómo se representan las cadenas de caracteres en el lenguaje de programación C. En particular pueden resultar útiles las funciones `strlen`, `strcat` y `strcpy`, definidas en el fichero de cabecera `string.h`, que allí se explican. También podéis utilizar las dos funciones siguientes (cópiadlas en vuestro código):

```
int toInt(char digito) {
   return digito - '0';
}
```

Descripción: Devuelve el valor decimal de un dígito
Entradas:
digito (pvalor): un carácter ASCII entre `'0'` y `'9'`.
Salidas:
Devuelve: Un entero, el valor decimal de `digito`.
Excepciones:
a. El resultado está indefinido si el parámetro no es un dígito.

```
char toChar(int n) {
   return (char)(n + '0');
}
```

Descripción: Devuelve el digito (un carácter) cuyo valor numérico es `n`.
Entradas:
n (pvalor): un entero entre 0 y 9.
Salidas:
Devuelve: Un carácter, el dígito cuyo valor es `n`.
Excepciones:
a. El resultado está indefinido si $n < 0$ o $n > 9$.

Finalmente, en el apéndice E también se recuerda la aritmética de punteros en el contexto de un array, que podéis utilizar para encontrar una solución alternativa a la tarea usando punteros. Nuestra recomendación es que resolváis la tarea sin punteros y, si queréis profundizar, la resolváis después usando punteros.

4.5. Proyecto RSA: Algoritmos con enteros arbitrariamente grandes

4.5.1. Introducción

En esta tarea extenderemos las anteriores (1.5, 2.5 y 3.4), superando sus limitaciones:

- En la tarea 3.4 los mensajes eran números enteros. En esta podremos encriptar, además, mensajes de texto. Para ello bastará codificar el mensaje (transformar un texto en un número entero, al que podemos aplicar todas las transformaciones ya vistas). Los algoritmos de codificación que hemos visto en las Tareas 1.5 y 2.5 se complican si tratamos de ampliar el juego de caracteres utilizado para redactar mensajes (si, además de letras mayúsculas y minúsculas, añadimos dígitos, signos de puntuación...).

  En definitiva, necesitamos un mecanismo sencillo que nos permita codificar juegos arbitrarios de caracteres.

- Queremos poder encriptar mensajes de decenas, e incluso centenares, de caracteres (al codificarlos se transformarán en enteros de decenas o centenares de dígitos). Los tipos de datos elementales en C/C++ (`int`, `float` , `double` ...) no permiten representar números tan grandes. Para verlo, basta un cálculo sencillo: una variable de tipo `int` o `float` tiene asignados 32 bits (con nuestro compilador); con 32 bits son posibles 2^{32} valores distintos: sólo podemos representar $2^{32} \simeq 4 \cdot 10^9$ números enteros (o reales). Por su parte, el tipo `double` tiene asignados 64 bits: admite $2^{64} \simeq 2 \cdot 10^{19}$ valores posibles. Un carácter tiene del orden de 100 valores posibles (letras mayúsculas y minúsculas, dígitos, signos de puntuación...); por tanto, existen del orden de $100^5 = 10^{10}$ mensajes posibles con sólo cinco caracteres, y $100^{10} = 10^{20}$ mensajes posibles con diez. Mensajes distintos deben codificarse mediante números distintos. Si queremos codificar mensajes de cinco o más caracteres, los tipos `int` o `float` se quedan cortos. Por su parte, el tipo `double` permitiría encriptar mensajes de 10 caracteres como máximo.

  Necesitamos, pues, un nuevo tipo de datos que permita representar enteros arbitrariamente grandes (con varios centenares de dígitos).

Podemos utilizar los arrays para resolver ambos problemas. Por un lado, almacenaremos en un array la información que permitirá codificar y decodificar cada carácter.

Además, representaremos los números enteros mediante arrays de caracteres (cada uno de los cuales representará un dígito). De este modo podremos representar enteros arbitrariamente grandes.

Nuestro desafío: A new type of cipher...

La presentación de la criptografía RSA al gran público tuvo lugar en 1977, gracias al gran divulgador Martin Gardner. Gardner publicó en *Scientific American* un artículo titulado *"A new kind of cipher that would take millions of years to break"*, donde presentaba las ideas de Rivest, Shamir y Addleman y proponía un desafío: desencriptar un mensaje que había sido encriptado usando la clave pública (n=RSA_129, e=9007), ofreciendo una recompensa de 100 dólares. RSA_129, un entero de 129 dígitos, es uno de los pocos números que tiene nombre propio.

En realidad, no se requirieron millones de años. En 1994, un equipo coordinado de 600 programadores usando 1600 máquinas a lo largo de seis meses de trabajo consiguieron encontrar los factores primos de RSA_129. Conociéndolos, desencriptar el mensaje era cuestión de segundos.

En esta tarea te proponemos que desencriptes dos mensajes, ambos encriptados con la clave (n=RSA_129, e=9007). El primero es el histórico mensaje con el que Gardner desafió a sus lectores; el segundo corresponde a una celebérrima aparición de la criptografía en la literatura.

4.5.2. Descripción de la tarea

Descarga el fichero tareaRSA.zip y descomprímelo. La carpeta tareaRSA contiene un proyecto con tres ficheros: `main.cpp`, `aritmeticaBasica.cpp` y `arimeticaBasica.h`.

`arimeticaBasica.cpp` contiene funciones auxiliares que realizan operaciones elementales con cadenas de caracteres que representan números enteros (suma, resta, multiplicación, división, comparación). NO NECESITAS ABRIR ESE FICHERO. Todo lo que necesitas conocer son los prototipos de esas funciones, que se encuentran en el fichero `aritmeticaBasica.h`. Asegúrate de leer y entender toda la documentación de aritmeticaBasica.h.

En el fichero main.cpp encontrarás unas cuantas constantes (el array `codificacion`, el número RSA_129 junto con sus factores primos, p_129 y q_129). En la función principal aparecen los dos mensajes que debes desencriptar.

Codificaremos/decodificaremos los caracteres del mensaje siguiendo la tabla 4.1.

Esta información queda recogida en el array `codificacion`:

```
"\1ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzÑñáéíóúÁÉÍÓÚÜü"
"¡¿?!@#,;.:-_$&/()=’\"0123456789\n\t "
```

A cada carácter le corresponde un código entero (la posición que ocupa dentro del array). Así, al carácter `'A'` le corresponde el 1 (ya que `codificacion[1] = 'A'`), al carácter `'a'` le corresponde el 27 (`codificacion[27] = 'a'`), al carácter `'0'` el 87 (`codificacion[87] = '0'`), y al espacio en blanco `' '` el 99 (`codificacion[99] = ' '`).

Observa que en la tabla no consideramos el código `00`; en el array, la componente `codificacion[0]` corresponde al carácter `'\1'`, que no utilizaremos en nuestros mensajes.

Cuadro 4.1: Codificación.

Código	Carácter	Código	Carácter	Código	Carácter	Código	Carácter
01	A	27	a	53	Ñ	79	$
02	B	28	b	54	ñ	80	&
03	C	29	c	55	á	81	/
04	D	30	d	56	é	82	(
05	E	31	e	57	í	83	)
06	F	32	f	58	ó	84	=
07	G	33	g	59	ú	85	'
08	H	34	h	60	Á	86	"
09	I	35	i	61	É	87	0
10	J	36	j	62	Í	88	1
11	K	37	k	63	Ó	89	2
12	L	38	l	64	Ú	90	3
13	M	39	m	65	Ü	91	4
14	N	40	n	66	ü	92	5
15	O	41	o	67	¡	93	6
16	P	42	p	68	¿	94	7
17	Q	43	q	69	?	95	8
18	R	44	r	70	!	96	9
19	S	45	s	71	@	97	\n
20	T	46	t	72	#	98	\t
21	U	47	u	73	,	99	' ' (espacio en blanco)
22	V	48	v	74	;		
23	W	49	w	75	.		
24	X	50	x	76	:		
25	Y	51	y	77	-		
26	Z	52	z	78	_		

Diseña y programa en C/C++ los siguientes subalgoritmos:

esPar

Dada una cadena de caracteres que representa un entero positivo, indica si es par o no. Recuerda: para decidir si un número es par, basta mirar su última cifra.

A modo de ejemplo, el entero representado por `"123"` es impar, mientras que el representado por `"120"` es par.

decodifica

Dada una cadena de caracteres, que representa un entero no negativo usando dos dígitos, devuelve su carácter asociado.

El subalgoritmo empieza transformando la cadena en el entero que representa, i. Si éste es mayor que la longitud del array `codificacion`, devuelve el carácter `' '` (espacio en blanco). Si no, devuelve el carácter que ocupa la i-ésima posición en `codificacion`.

A continuación mostramos algunos ejemplos de cadenas junto con su carácter decodificado:

Cadena	carácter descodificado
`"99"`	`' '`
`"05"`	`'E'`
`"47"`	`'u'`
`"44"`	`'r'`
`"31"`	`'e'`
`"37"`	`'k'`
`"27"`	`'a'`
`"70"`	`'!'`

decodificaMensaje

Dada una cadena de caracteres que representa un entero no negativo, `textoCodificado`, devuelve otra, `texto`, obtenida del modo que se indica a continuación. Si `textoCodificado` tiene un número impar de caracteres, se le añade el carácter `'0'` al inicio. A continuación se toman los caracteres de `textoCodificado` por parejas, y se concatenan los caracteres que las decodifican.

A continuación mostramos algunos ejemplos de cadenas de caracteres junto con el resultado de decodificarlas.

`textoCodificado`	`texto`
`"547443137279970"`	`"Eureka !"`
`"2627442733415227"`	`"Zaragoza"`
`"946993545992738389907443131379946419939317 0"`	`"It is all Greek to me!"`

euclidesExtendido

Dadas dos cadenas que representan sendos enteros positivos, n y m, devuelve tres cadenas que representan su máximo común divisor y otros dos enteros, a y b, tales que

$$mcd(n, m) = a \cdot n + b \cdot m.$$

Ejemplos:

n	m	mcd	a	b
"237"	"53"	"1"	"17"	"-76"
"1791"	"1273"	"1"	"-376"	"529"
"519"	"234"	"3"	"23"	"-51"

inversoModulo

Dadas dos cadenas que representan sendos enteros positivos, n y m, devuelve una cadena que representa el inverso de m módulo n.

Ejemplos:

n	m	inverso
"237"	"53"	"161"
"1791"	"1273"	"529"

potenciaModulo

Dadas tres cadenas de caracteres que representan sendos enteros positivos, b, e, n, devuelve una cadena que representa el entero $b^e \,\%\, n$.

Ejemplos:

b	e	n	$b^e \;\%\; n$
"123"	"235"	"413"	"361"
"999"	"13"	"987"	"852"

desencripta

Este subalgoritmo toma como entradas tres cadenas: n, d, M_{encrip}, y se interpretan del siguiente modo: (n, d) es la clave privada, y M_{encrip} un mensaje encriptado. Devuelve una cadena de caracteres, $Texto$ (el mensaje desencriptado). Sigue el algoritmo indicado a continuación:

i) Se calcula $M = (M_{encriptado})^d \,\%n$

ii) Se decodifica M, sustituyendo cada pareja de dígitos por su letra correspondiente.

Ejemplo:

n	d	$M_{encriptado}$	Texto
"276121"	"9855"	"169111"	"ABC"

Programa principal

Sigue los pasos indicados en la función main:

- Calcula la función de Euler de RSA_129, $\varphi(RSA\_129)$. Dado que ya conoces los factores primos de RSA_129,

$$\varphi(RSA\_129) = (p\_129 - 1) \cdot (q\_129 - 1).$$

- Calcula el inverso de e módulo $\varphi(n)$: el número d tal que

$$e \cdot d \,\%\, \varphi(RSA\_129) = 1.$$

  La clave privada es (n, d) (la componente e de la clave pública es 9007, ya está definida al inicio de la función principal).

- Utilizando esa clave privada, desencripta los dos mensajes que proponemos.

4.5.3. Ayudas

Sobre la longitud de los arrays utilizados

Un punto extremadamente delicado de esta tarea es determinar el tamaño de los arrays con que se va a trabajar. Para simplificar la tarea, vamos a fijar un tamaño estándar para todos los arrays auxiliares que declares en los subalgoritmos. Dado que la clave pública es $(RSA\_129, 9007)$, el valor máximo posible de los enteros con que vamos a trabajar es RSA_129^2, que tiene 258 dígitos. Un array que contenga ese valor necesita 259 caracteres (258 para los dígitos y un carácter extra para el fin de cadena, `'\0'`). Por ello, te sugerimos que todos los arrays auxiliares que necesites declarar en los subalgoritmos tengan como tamaño `longMax = 259`.

Al codificar un texto T de l caracteres obtenemos un entero M de $2 \cdot l$ caracteres (recuerda que cada carácter se codifica mediante un par de dígitos): $M < 10^{2l}$. Recuerda que los mensajes M deben ser menores que RSA_129 (que es del orden de 10^{128}). Por ello, el número l de caracteres de un texto debe satisfacer $l \leq 64$. La variable `texto` tiene un tamaño máximo de 65 caracteres (64 para el texto y un carácter extra para el final de cadena).

subalgoritmos 'decodifica' y 'decodificaMensaje'

La principal dificultad de estos subalgoritmos es que trabajan con varios tipos de datos distintos: cadenas de caracteres (`"35"`), caracteres (`'3'`, `'5'`, `'\0'`) y números enteros (3, 5, 35). Ten presente cómo se transforman expresiones de un tipo a otro. Identifica en todo momento cuál es el tipo de dato que utilizas en tus expresiones.

Funciones auxiliares

Lee atentamente la documentación relativa a los subalgoritmos para la aritmética de enteros, representados por cadenas (fichero `aritmeticaBasica.h`). Allí encontrarás información sobre las funciones del fichero `aritmeticaBasica.cpp` que te puede ser útil: descripción de la función, parámetros, restricciones y ejemplos.

Funciones declaradas en el fichero `string.h`

Puedes utilizar las funciones `strlen`, `strcpy`, `strcat`, descritas en el apéndice E

Punteros constantes y punteros a constante

Al inicio del fichero *main.cpp* se encuentran expresiones como
`const char * const RSA_129`.
El primer const indica que RSA_129 es un puntero a constante; el segundo, que RSA_129 es un puntero constante. Para entender la diferencia entre ambos conceptos, considera el siguiente ejemplo:

```
void ejemploPunteroConstante()
{
   char textoA[] = "Hola", textoB[]="Adios";
   const char * pt1 = textoA;
   cout << pt1 << endl;
   pt1 = textoB;
   cout << pt1 << endl;


   char * const pt2 = textoA;
   cout << pt2 << endl;
   pt2[0] = 'I';
   cout << pt2 << endl;
}
```

Al ejecutar la función anterior, se muestra por pantalla:

```
Hola
Adios
Hola
Iola
```

- **const** `char * pt1` : `pt1` es un puntero a constante: no es posible cambiar el contenido de la cadena a la que apunta; por tanto, la sentencia

  `pt1[0]='I';`

  sería ilegal. Es posible cambiar la dirección a la que apunta (hacer que apunte a otra dirección), como se muestra en el ejemplo.

- `char *` **const** `pt2` : `pt2` es un puntero constante: no es posible hacer que apunte a ninguna otra dirección; por tanto, la sentencia

  `pt2 = textoB;`

  sería ilegal. Permite modificar el contenido de la cadena a la que apunta.

Cuadro 4.2: Cuotas de los métodos de resto mayor

Cuota Hare	m/n
Cuota Droop	1+(m/(n+1))
Cuota Imperiali	m/(n+2)

4.6. Leyes electorales

4.6.1. Introducción

Existen diversos sistemas para la asignación de escaños en los procesos de elección de representantes en asambleas o parlamentos, entre los que destacan los métodos de promedio mayor y los métodos de resto mayor. En la primera categoría figuran los siguientes métodos: D'Hont, Sainte-Laguë puro y Sainte-Laguë modificado. En la segunda categoría, el cociente Hare, el cociente Droop y el cociente Imperiali. En España, se usa la ley d'Hont.

El método d'Hont es un método de reparto de promedio mayor, creado por el jurista belga Victor d'Hont en 1878. El número de votos recibidos por cada lista se divide sucesivamente por los primeros números enteros, desde 1 hasta el número total de escaños a repartir. La asignación de escaños se hace ordenando los cocientes de mayor a menor y asignando un escaño a la lista correspondiente hasta que los escaños se agoten. Más concretamente, sea s_i el número de escaños que ya se han asignado a la lista l_i. Entonces, el siguiente escaño se asigna a lista tal que el cociente entero $v_i/(s_i + 1)$ es mayor, donde v_i es el número de votos que ha recibido la lista l_i. Antes de comenzar el proceso no se ha asignado ningún escaño a las listas y el proceso termina cuando se han asignado todos los escaños.

El método Sainte-Laguë puro es un método de reparto de promedio mayor, creado por el matemático francés André Sainte-Laguë (1882-1950). El número de votos recibidos por cada lista se divide sucesivamente por los primeros enteros impares, desde 1 hasta el doble de escaños a repartir más uno. La asignación de escaños se hace ordenando los cocientes de mayor a menor y asignando un escaño a la lista correspondiente hasta que los escaños se agoten. El método Sainte-Laguë modificado es igual al método puro pero para asignar el primer escaño se dividen los votos recibidos por cada partido por 1,4.

En los métodos de resto mayor, se utiliza la división entera del número de votos de cada lista entre un entero, denominado cuota, que representa el número de votos requeridos para obtener un escaño. En primer lugar se asigna a cada lista un número de escaños igual al cociente entero de dicha división. Este reparto dejará normalmente algunos escaños sin asignar. Entonces se ordenan las listas en función de sus restos y los partidos con mayores restos obtienen un escaño extra cada uno, hasta repartir todos los escaños.

La diferencia entre los métodos está en la cuota utilizada. En la Tabla 4.2 puede verse cada una de esas cuotas, siendo m el número de votos y n el número de escaños. Las divisiones que aparecen en cada una de las cuotas se hacen como divisiones reales y luego se redondean al entero más próximo.

En la Figura 4.3, mostramos la asignación de los siete primeros escaños según la ley D'Hont a los resultados en las elecciones generales de España de 2015 en la provincia de Madrid. El mayor cociente está en negrita e indica el partido al que se le asigna el escaño.

En la Figura 4.4, mostramos la asignación de 21 escaños según el cociente Hare a un ejemplo con un millón de votos extraído de Wikipedia. El cociente Hare es 1000000/21=47.619.

Partido 0	Partido 1	Partido 2	Partido 3	Partido 4	Partido 5	Se asigna el escaño al partido
1203837	750477	676839	643158	189237	43103	0
601918,5	**750477**	676839	643158	189237	43103	1
601918,5	375238,5	**676839**	643158	189237	43103	2
601918,5	375238,5	338419,5	**643158**	189237	43103	3
601918,5	375238,5	338419,5	321579	189237	43103	0
401279	375238,5	338419,5	321579	189237	43103	0
300959,25	**375238,5**	338419,5	321579	189237	43103	1

Figura 4.3: Ejemplo de aplicación de la ley d'Hont

Partido	Partido 0	Partido 1	Partido 2	Partido 3	Partido 4	Partido 5	Partido 6	Total escaños
Votos por partido	391.000	311.000	184.000	73.000	27.000	12.000	2.000	
Cociente entero	8	6	3	1	0	0	0	
Escaños por cociente	8	6	3	1	0	0	0	18
Resto entero	10.048	25.286	**41.143**	**25.381**	**27.000**	12.000	2.000	
Escaños por resto	0	0	1	1	1	0	0	3
Total de escaños	8	6	4	2	1	0	0	21

Figura 4.4: Ejemplo de aplicación del cociente Hare

Puedes encontrar información completa sobre este tema en [6].

4.6.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que para una circunscripción muestre el reparto de escaños de acuerdo con todos los métodos de reparto que hemos descrito en esta práctica.

Estructuras y subalgoritmos

Para resolver esta tarea, usa la siguiente estructura:

```
struct circunscripcion{
  char nombre[50];
  int numeroPartidos;
  int numeroEscanos;
  int votosPartidos[20];};
```

En la estructura `circunscripcion`, los campos son: el nombre de la circunscripción donde se han recogido los resultados, el número de partidos de que se dispone de resultados, el número total de escaños de la circunscripción y el número de votos por partido (hasta un máximo de 20 partidos). Algunos subalgoritmos que puedes usar son los siguientes (puedes añadir otros si lo crees necesario):

maximoVector

Dado un vector de reales, calcula la posición de la componente, entre las n primeras componentes del vector, que tiene el máximo valor

sumaVector

Dado un vector de enteros, calcula la suma de las n primeras componentes

muestraResultado

Dado el nombre de un método, el número de escaños por partido obtenidos por la aplicación del método y el número de partidos, muestra cuántos escaños tiene cada partido. Por ejemplo, para el caso del método d'Hont en la circunscripción de Madrid en las elecciones generales de 2015 que puedes ver en la Figura 4.3, el resultado de este subalgoritmo es:

```
D'Hont 13 8 7 6 2 0
```

calculaNMayores

Dado un vector de enteros no negativos, el número de sus componentes m y un número n, $n < m$, devuelve las posiciones en las que se encuentran los n valores mayores del vector.

Para escribir este algoritmo, puedes proceder del siguiente modo. Busca el mayor valor del vector, anota su posición en un vector `pos` de posiciones y cambia ese valor mayor a -1. Repite este proceso n veces anotando cada posición del máximo en una nueva componente del vector.

Por ejemplo, si $m = 7$, $n = 3$ y `v={1, 9, 200, 8, 90, 20, 100}` el subalgoritmo procede como sigue:

Paso 1. Encontramos el máximo de v, anotamos su posición y cambiamos el máximo a -1. Los vectores `v` y `pos` quedan como sigue:

```
v={1, 9, -1, 8, 90, 20, 100} pos={2}
```

Paso 2. Encontramos el máximo de v, anotamos su posición y cambiamos el máximo a -1. Los vectores `v` y `pos` quedan como sigue:

```
v={1, 9, -1, 8, 90, 20, -1} pos={2, 6}
```

Paso 3. Encontramos el máximo de v, anotamos su posición y cambiamos el máximo a -1. Los vectores `v` y `pos` quedan como sigue:

```
v={1, 9, -1, 8, -1, 20, -1} pos={2, 6, 4}
```

El resultado es el vector `pos={2, 6, 4}`.

calculaEscanosCircunscripcionPromedio

Dado un dato de tipo `circunscripcion`, el primer divisor y el factor para calcular los siguientes divisores, calcula el reparto de escaños para cada partido aplicando un método de promedio. Por ejemplo, para el método d'Hont, el primer divisor es el 1 y el factor para calcular los siguientes divisores es 1. Para el método Sainte-Laguë puro, el primer divisor es el 1 y el factor para calcular los siguientes divisores es 2. Para el método Sainte-Laguë modificado, el primer divisor es el 1.4 y el factor para calcular los siguientes divisores es 2.

calculaEscanosCircunscripcionRestoMayor

Dado un dato de tipo `circunscripcion` y un número entero que indica la cuota de un método de reparto de escaños de tipo resto mayor, calcula el reparto de escaños para cada partido aplicando ese método.

4.6.3. Ejemplos de ejecución

Para la circunscripción ejemplo de los resultados de las elecciones de 2015 en la provincia de Madrid, el reparto de escaños es:

```
Los escaños por partido en la circunscripción de Madrid son:
D'Hont      13   8   7   6   2   0
Saint puro  12   8   7   7   2   0
Saint modif 12   8   7   7   2   0
Hare        12   8   7   7   2   0
Imperiali   13   8   7   6   2   0
Droop       12   8   7   7   2   0
```

4.6.4. Ayudas

Puedes usar el fichero `reparto.cpp` que encontrarás en Moodle. En ese fichero, aparece como ejemplo los resultados en las elecciones de 2015 en la provincia de Madrid de la Figura 4.3. Además, para tener otra circunscripción de prueba, construye una estructura de tipo `circunscripcion` con los datos de la Figura 4.4 y comprueba que el resultado que obtienes con tu programa para el método Hare coincide con el que se muestra en esa Figura. Usa para esta nueva estructura el mismo nombre de la variable `circunscripcion` que para la estructura de Madrid. De esta forma, cuando quieras probar la nueva estructura, comenta la estructura de Madrid tal y como puedes ver a continuación.

```
//struct circunscripcion q={"Madrid",6,36,
//            {1203837,750477,676839,643158,189237,43103}};

struct circunscripcion q={"Ejemplo wikipedia",...};
```

Si quieres probar de nuevo la estructura de Madrid, quítale el comentario y sitúalo en la estructura del ejemplo de Wikipedia como puedes ver a continuación.

```
struct circunscripcion q={"Madrid",6,36,
          {1203837,750477,676839,643158,189237,43103}};
//struct circunscripcion q={"Ejemplo wikipedia",...};
```

Observa que este programa no solicita ningún dato de entrada al usuario. Sus datos de entrada provienen de la estructura `circunscripcion` de Madrid o de la estructura `circunscripcion` del ejemplo de Wikipedia.

Para redondear un número real al entero más próximo, puedes usar la función `round` de C++. Por ejemplo, `x=round(1.9)` guarda en la variable `x` el entero más próximo a 1.9, es decir, 2.

4.7. Proyecto 2C: Distancias en un sistema planetario (i)

En las tareas 2.7 y 3.6 nos hemos dotado de los medios para estudiar la dinámica de un objeto que, sometido a una fuerza central, describe una órbita elíptica. Más precisamente, somos capaces de calcular la posición angular del objeto en un instante de tiempo dado y, también, de resolver el problema inverso. Utilizaremos aquí estas herramientas para calcular la mayor aproximación entre planetas en un sistema planetario simulado: una simplificación del nuestro, en la que supondremos que las órbitas de todos los planetas se encuentran en el mismo plano. Con algo de esfuerzo, el trabajo que realicemos se puede adaptar a la realidad, lo que requiere considerar diversos sistemas de referencia, uno por planeta, y las correspondientes transformaciones de coordenadas entre ellos. El lector interesado puede consultar el libro *Spherical Astronomy* [14] de R.M. Green.

4.7.1. Contexto del problema

Para estudiar la dinámica del sistema planetario debemos conocer, para cada planeta, la órbita que sigue, el tiempo que tarda en recorrerla (el periodo), así como su posición en un instante de tiempo determinado. Para ello proponemos utilizar las siguientes estructuras de datos.

```
struct elipse {
    double a, e, phi0;
};
struct fecha {
   int year, dias;
};
struct sistema {
    struct planeta objeto[MAXPLA];
    int nobjetos;
};
```

```
struct planeta {
    char nombre[100];
    struct elipse orbita;
    double periodo, theta0;
    struct fecha t0;
};
```

Suponemos que todos los planetas orbitan en el mismo plano alrededor de su estrella, que ocupa uno de los focos de las órbitas elípticas que describen. Tomando la posición de la estrella como origen de coordenadas, basta considerar cualquier semirrecta como eje de coordenadas polares para disponer de un sistema de referencia común a todos los planetas.

Con estas consideraciones, la estructura `elipse` modeliza los parámetros que caracterizan una elipse en ese sistema de coordenadas: la longitud del semieje mayor, a, la excentricidad, e, y la inclinación, ϕ_0, del semieje mayor respecto del eje de coordenadas polares.

La estructura `planeta` contiene la información necesaria sobre un planeta: su nombre (una cadena de caracteres), la órbita que describe, su posición angular θ_0 en un instante de tiempo t_0 y el periodo T de la órbita.

El tiempo está modelizado por la estructura `fecha`. Su campo `year` especifica un año de referencia, mientras que el entero `dias` indica el número de días transcurridos desde el comienzo de dicho año. Por ejemplo, el par $(2017, 0)$ representa el día 01-01-2017, y $(2017, -1)$ el día 31-12-2016. Puesto que 2016 es un año bisiesto, obsévese que el par $(2017, -1)$ representa el mismo día que $(2016, 365)$.

Finalmente, la estructura `sistema` permite modelizar un sistema que tiene un número, `nobjetos`, de planetas que es menor o igual que una constante `MAXPLA` fijada. Los datos de los planetas se almacenan en las `nobjetos` primeras componentes del array `objeto`, quedando las restantes indefinidas.

Todas las longitudes estarán expresadas en unidades astronómicas (u.a.), los ángulos en radianes y el periodo de las órbitas en días. Una unidad astronómica es la distancia media entre la Tierra y el Sol.

4.7.2. Descripción de la tarea

Diseña y escribe en el lenguaje de programación C, utilizando únicamente el paso por referencia de C++ si se precisa, subalgoritmos que realicen las operaciones siguientes.

mostrarTiempo

Dado un dato de tipo `fecha`, muestra en pantalla su valor en el formato `dd/mm/yyyy`. Tras escribir en pantalla este texto, el subalgoritmo NO deberá escribir ningún salto de línea.

Ayudas: Para facilitar el diseño de este subalgoritmo, se proporciona el subalgoritmo `fechaEstandar` y otros que éste utiliza como auxiliares. Él código y su descripción se incluyen en la sección 4.7.4.

dinamica

Dado un dato de tipo `planeta`, p, y una cantidad Δt de tiempo expresada en días, calcula, con precisión ε, la posición angular del planeta en el instante $t_0 + \Delta t$. Recordad que t_0 es el instante de tiempo en el que conocemos que la posición del planeta es θ_0.

Ayudas: En la medida de lo posible reutiliza los subalgoritmos `integralExacta` y `dinamica` realizados en la tarea 3.6. Hay dos opciones: reescribirlos adaptándolos a las estructuras de datos utilizadas aquí o, simplemente, invocarlos con argumentos adecuados. En la sección 4.7.4 se incluye una versión de estos subalgoritmos.

coordenadas

Dado un dato de tipo `elipse` y la posición angular θ de un punto de dicha elipse calcula su posición (x, y) en coordenadas cartesianas. El resultado debe devolverse como un array de dos componentes reales.

Ayudas: Se sugiere reutilizar el subalgoritmo `coordenadas` desarrollado en la tarea 3.6 y del que también se proporciona una versión en la sección 4.7.4.

distanciaV

Dados un entero n y dos arrays de números reales con n componentes, que representan sendos puntos del espacio euclídeo n-dimensional, calcular la distancia euclídea entre ellos; esto es, el módulo del vector que dichos puntos definen.

distancia

Dados dos planetas, de los que se conocen sus posiciones en el mismo instante de tiempo t_0, y una cantidad Δt de tiempo expresada en días, calcular, con precisión ε, la distancia que hay entre ellos en el instante de tiempo $t_0 + \Delta t$.

Programa principal

Calcular, entre los días 01/01/2017 y 01/01/2117, la fecha en la que se producirá el máximo acercamiento de la Tierra a los planetas Marte, Mercurio y Venus, así como cuál será entonces la distancia, medida en unidades astronómicas, entre ellos. El cálculo de la posición de los planetas debe realizarse con una precisión de 10^{-9} radianes. La salida de resultados en pantalla debe parecerse lo más posible, tanto en valores como en formato, a la que se sugiere en la sección 4.7.3.

Puedes utilizar la información siguiente para inicializar variables adecuadas en el programa. El ángulo θ_0 es la posición angular en la que se encuentra cada planeta el día 01/01/2017.

Tierra:
$a = 1$, $e = 0{,}01671123$, $\phi_0 = 0{,}225971$, periodo: 365,256 días, $\theta_0 = -4{,}52357$.

Mercurio:
$a = 0{,}387098$, $e = 0{,}20563069$, $\phi_0 = 2{,}9266$, periodo: 87,9672 días, $\theta_0 = -1{,}23456$.

Venus:
$a = 0{,}723327$, $e = 0{,}00677323$, $\phi_0 = 3{,}866479$, periodo: 224,701 días, $\theta_0 = -2{,}34567$.

Marte:
$a = 1{,}52368$, $e = 0{,}093315$, $\phi_0 = 0{,}2886$, periodo: 686,971 días, $\theta_0 = -0{,}588455$.

4.7.3. Ejemplos de ejecución

```
Efemerides: acercamiento maximo de la Tierra a:

Planeta     Distancia minima  Fecha
  Mercurio   0.51794552 u.a.  13/03/2075
     Venus   0.25569592 u.a.  02/04/2021
     Marte   0.39856919 u.a.  02/04/2065
```

4.7.4. Ayudas

A continuación se incluye el código y descripción del subalgoritmo `fechaEstandar`, que se puede invocar desde `mostrarFecha`, así como de los subalgoritmos auxiliares que el primero necesita.

```
/** Determina si un año es bisiesto en el calendario Gregoriano.

    Parámetros:
        y - el año
    Devuelve:
        verdad si y sólo si el año y es bisiesto
*/
bool esBisiesto(int y) {
```

```
    return y%400==0 || (y%4==0 && y%100!=0);
}

/** Calcula el número de días de un mes en un año dado.

    Parámetros:
        m - el mes
        y - el año
    Devuelve:
        el número de días que tiene el mes m en el año y
*/
int nDiasMes(int m, int y) {
    switch(m) {
    case 2:
        if (esBisiesto(y)) return 29;
        else return 28;
    case 4: case 6: case 9: case 11:
        return 30;
    default:
        return 31;
    }
}

/** Calcula el número de días de un año.

    Parámetros:
        y - el año
    Devuelve:
        el número de días del año y.
*/
int nDiasYear(int y) {
    int dias;
    if (esBisiesto(y)) dias = 366;
    else dias = 365;
    return dias;
}

/** Normaliza una Fecha. Dada una Fecha la modifica para que,
    representando la misma información, tras la modificación el número de
    días sea mayor o igual que 0 y esctrictamente menor que el número de
    días del año de referencia.

    Parámetros (dato y resultado):
        t - una Fecha que, al finalizar, estará normalizada
*/
void normalizaFecha(struct fecha &t) {
    while(t.dias<0) {
```

```
        t.year--;
        t.dias += nDiasYear(t.year);
    }
    int aux = nDiasYear(t.year);
    while(t.dias>=aux) {
        t.dias -= aux;
        t.year++;
        aux = nDiasYear(t.year);
    }
}

/** Calcula el día, el mes y el año que un dato de tipo Fecha representa.

    Parámetros (dato):
        t - una Fecha
    Parámetros (resultado):
        d - el día representado por t
        m - el mes representado por t
        y - el año representado por t
*/
void fechaEstandar(struct fecha t,
                   int &d, int &mes, int &y) {
    normalizaFecha(t);
    y = t.year;
    mes = 1;                          //Cálculo del mes
    int aux = nDiasMes(mes, t.year);
    while(t.dias>=aux) {
        t.dias -= aux;
        mes++;
        aux = nDiasMes(mes, t.year);
    }
    d = t.dias+1;                     //Cálculo del dia
}
```

Finalmente, presentamos una versión de algunos subalgoritmos desarrollados en la tarea 3.6 que se pueden modificar o simplemente invocar en ésta.

```
double integralExactaT03(double e, double phi0, double phi) {
    double theta = phi - phi0;
    int n = (int)floor(theta/M_PI);
    double gamma = 1 - e*cos(theta);
    double xi = asin((cos(theta)-e)/gamma);
    double res = pow(1-e*e, 1.5)*sin(theta)*cos(theta)/gamma/gamma;
    res += (n + 0.5)*M_PI;
    if (n%2==0) { res -= xi + sin(2*xi)/2;
    } else { res += xi + sin(2*xi)/2;
    }
    return res/(2*M_PI);
```

```
}

double dinamicaT03(double e, double phi0, double periodo,
             double t0, double tf, double theta0, double epsilon) {
    const double taux = (tf-t0)/periodo;
    const double fthetaf = integralExactaT03(e, phi0, theta0) + taux;
    const double n = floor(taux);
    double inf = 2*n*M_PI + theta0,
           sup = inf + 2*M_PI;

    while(sup-inf>epsilon) {
        double med = (inf+sup)/2;
        double fmed = integralExactaT03(e, phi0, med);
        if (fmed<fthetaf) inf = med;
        else if (fmed>fthetaf) sup = med;
        else return med;
    }
    return (inf+sup)/2;
}

void coordenadasT03(double a, double e, double phi0, double theta,
                    double &r, double &x, double &y) {
    r = a*(1-e*e)/(1-e*cos(theta-phi0));
    x = r*cos(theta);
    y = r*sin(theta);
}
```

4.8. Proyecto ST: Servicio de transporte

En esta tarea planteamos un problema de optimización: planificar las entregas de una serie de mercancías de modo que se minimice el tiempo medio que el cliente debe esperar hasta recibir su paquete.

4.8.1. Introducción

El problema

Suponemos un servicio de transporte, dirigido a clientes que desean enviar paquetes de un sitio a otro. Disponemos de una única camioneta que, a lo largo de una jornada, debe realizar una serie de repartos preacordados. Para cada reparto conocemos sus puntos de origen y destino: dónde se recoge y entrega la mercancía.

Al inicio de cada jornada disponemos del listado completo de repartos. Los clientes saben qué día serán atentidos, pero no la hora. Al inicio de la jornada laboral, los paquetes deben estar listos para ser repartidos.

Con objeto de simplificar el problema, vamos a suponer que nuestra camioneta tiene un funcionamiento peculiar: después de cada reparto necesita regresar a su base (todos los servicios empiezan y terminan en la base).

Nuestro problema consiste en planificar en qué orden deben realizarse los repartos del modo más eficiente posible. ¿Cómo medimos la *eficiencia* de la planificación? Nuestra referencia será la satisfacción del cliente: queremos minimizar el tiempo medio que transcurre desde que el paquete está listo (desde el inicio de la jornada) hasta que llega a su destino. Pongamos un ejemplo para comprender este punto. Imaginemos dos repartos: reparto A, con origen en la plaza San Francisco y destino en el Paraninfo; reparto B, con origen en la Plaza de España y destino en Estadilla (a 140 km). La camioneta necesita un minuto para llegar a la plaza San Francisco, dos para ir desde allí al Paraninfo y otros dos para regresar a su base. La plaza de España está a un minuto de la base y a hora y media de Estadilla.

Una solución intuitiva

Es inmediato encontrar la solución óptima: primero se hace el servicio A y después el B. El tiempo de entrega para el primer cliente es de 3 minutos; el del siguiente, una hora y 36 minutos (5 minutos para el primer reparto, 1 minuto más para la recogida y hora y media de viaje). El tiempo medio de reparto es (3+96)/2 = 49.5 minutos. La otra opción es mucho peor: si el cliente A queda relegado al segundo lugar, tiene que esperar más de tres horas (!!) para que su paquete recorra apenas un kilómetro.

4.8.2. Formalización

Para abordar el problema formalmente debemos formularlo con claridad, sin opción a ambigüedades. Para ello necesitamos dos definiciones:

Definición 4.8.1. Tiempo de servicio Dado un servicio s, definimos su tiempo de servicio como la suma:

$$t_s = t_{s;ida} + t_{s;transporte} + t_{s;regreso} \tag{4.5}$$

donde $t_{s;ida}$ es el tiempo desde la base hasta el punto de recogida, $t_{s;transporte}$ es el tiempo desde el punto de recogida hasta el de entrega, $t_{s;regreso}$ es el tiempo desde el punto de entrega hasta la base.

Para aclarar qué entendemos por **tiempo medio de entrega** , tomemos como ejemplo una secuencia de tres servicios. El primer envío llega a su destino en $t_{s_0} - t_{s_0;regreso}$; el segundo, en $t_{s_0} + t_{s_1} - t_{s_1;regreso}$ (debe esperar a que finalice el primero), el tercero en $t_{s_0} + t_{s_1} + t_{s_2} - t_{s_2;regreso}$. La suma de tiempos que habrán esperado los clientes para la realización de sus servicios es: $(t_{s_0} - t_{s_0;regreso}) + (t_{s_0} + t_{s_1} - t_{s_1;regreso}) + (t_{s_0} + t_{s_1} + t_{s_2} - t_{s_2;regreso})$ $= 3 \cdot t_{s_0} + 2 \cdot t_{s_1} + t_{s_2} - t_{s_0;regreso} - t_{s_1;regreso} - t_{s_2;regreso}$

Definición 4.8.2. Dada una secuencia de n servicios $\{s_i\}$, el **tiempo medio de entrega** es

$$\frac{\sum_{i=0}^{i<n}(n-i)t_{s_i} - \sum_{i=0}^{i<n} t_{s_i;regreso}}{n} \tag{4.6}$$

Análogamente, el **tiempo medio de servicio** es:

$$\frac{\sum_{i=0}^{i<n}(n-i)t_{s_i}}{n} \tag{4.7}$$

Observamos que, dado un conjunto de servicios, la diferencia entre ambos tiempos es constante: es independiente de en qué orden se realicen los repartos.

La solución óptima viene dada por el siguiente teorema, que enunciamos sin demostración:

Teorema 4.8.3. *Dada una secuencia de n servicios $\{s_i\}$, el tiempo medio de entrega y el tiempo medio de servicio se minimizan cuando los servicios están ordenados crecientemente según su tiempo de reparto: La planificación es óptima cuando* $i < j \Leftrightarrow t_i < t_j$.

Un último comentario. Si suponemos que la velocidad es constante, distancia recorrida y tiempo empleado son directamente proporcionales. En ese caso, la solución óptima equivale a ordenar los servicios, de forma creciente, según su distancia.

Observemos que la solución óptima coincide con la que hemos intuido sin esfuerzo. No es casualidad: el problema está diseñado para que ocurra eso. En particular, esa *coincidencia* se debe al requisito (artificioso y forzado) de obligar a la camioneta a regresar a la base al terminar cada reparto. En futuras tareas plantearemos un escenario más realista (y muchísimo más difícil de resolver).

4.8.3. Tarea

Para representar los distintos servicios de una jornada usaremos las siguientes estructuras:

```
struct punto {
  double x, y;
};
typedef struct punto Punto;
// Punto en R^2.

struct reparto{
```

```
  char identificador;
  Punto origen, destino;
};
typedef struct reparto Reparto;

struct jornada{
    int nServicios;
    Reparto servicio[100];
};
typedef struct jornada Jornada;
```

Diseña y programa en C/C++ los siguientes subalgoritmos:

distanciaReparto

Dado un registro de tipo Reparto, calcula la distancia taxi requerida para realizarlo: la correspondiente al trayecto base - origen - destino - base. Tomamos como base el punto (0, 0).

distanciaMediaJornada

Dado un registro de tipo *Jornada*, calcula su distancia media de servicio:

$$\frac{\sum_{i=0}^{i<n}(n-i)d_{s_i}}{n} \tag{4.8}$$

donde d_{s_i} es la distancia total (ida+trayecto+vuelta) asociada al servicio s_i.

muestraJornada

Dado un registro de tipo *Jornada* muestra por pantalla, el listado de sus servicios (identificador, origen, destino y distancia), así como la distancia media de servicio.

optimiza

Dado un registro de tipo *Jornada*, ordena sus servicios de forma creciente según sus distancias correspondientes.

programa principal

Descarga el fichero *TareaJornada.cpp*, que incluye la información relativa a una jornada de reparto. Muestra por pantalla el listado de servicios (identificador, origen, destino, distancia de reparto) así como la distancia media de la jornada, antes y después de optimizar sus repartos.

4.8.4. Ejemplo de ejecución

```
Antes de ordenar:
a: origen = (3, 2)       destino =  (-3, 2)      Distancia = 16
b: origen = (3, 2)       destino =  (3, -2)      Distancia = 14
c: origen = (41, 5)      destino =  (-77, -1600)         Distancia = 3446
```

```
d: origen = (-81, -29)   destino =  (-68, -75)   Distancia = 312
e: origen = (-77, 72)    destino =  (67, 48)     Distancia = 432
f: origen = (31, -69)    destino =  (-94, -11)   Distancia = 388
g: origen = (-640, -47)  destino =  (78, 15)     Distancia = 1560
h: origen = (56, -75)    destino =  (-4, 70)     Distancia = 410
i: origen = (-83, 95)    destino =  (-56, -70)   Distancia = 496
j: origen = (14, 6)      destino =  (87, -7)     Distancia = 200
Distancia media: 4323.2


Tras ordenar:
b: origen = (3, 2)       destino =  (3, -2)      Distancia = 14
a: origen = (3, 2)       destino =  (-3, 2)      Distancia = 16
j: origen = (14, 6)      destino =  (87, -7)     Distancia = 200
d: origen = (-81, -29)   destino =  (-68, -75)   Distancia = 312
f: origen = (31, -69)    destino =  (-94, -11)   Distancia = 388
h: origen = (56, -75)    destino =  (-4, 70)     Distancia = 410
e: origen = (-77, 72)    destino =  (67, 48)     Distancia = 432
i: origen = (-83, 95)    destino =  (-56, -70)   Distancia = 496
g: origen = (-640, -47)  destino =  (78, 15)     Distancia = 1560
c: origen = (41, 5)      destino =  (-77, -1600)         Distancia = 3446
Distancia media: 1822.8
```

Capítulo 5

Ficheros

5.1. Carga sometida a un campo magnético

5.1.1. Introducción

En esta tarea calcularemos la dinámica de sistema de partículas con cargas q_i y masas m_i sometidas a un campo magnético $\vec{B}$. Por simplicidad supondremos que $\vec{B}$ es constante y uniforme. El objetivo de la tarea es adquirir destreza en el uso de tipos de datos, el manejo de ficheros y la resolución numérica de integrales en varias dimensiones. Conocer a priori cuál es el resultado nos servirá para validar nuestros algoritmos.

Recordemos brevemente cómo se describe la dinámica de una partícula sometida a $\vec{B}$. Para ello, descomponemos la velocidad de la partícula en dos componentes: una paralela al campo y otra en el plano perpendicular al campo: $\vec{v} = \overrightarrow{v_{||}} + \overrightarrow{v_{\perp}}$. Entonces:

- Dado que la fuerza de Lorentz $\vec{F}$ es perpendicular a la velocidad de la partícula, no ejerce trabajo. Por tanto, la energía cinética de la partícula se mantiene constante: $v_{\perp}^2 + v_{||}^2 = \text{cte}$.

- En la dirección del campo la fuerza es nula ($\overrightarrow{F_{||}} = 0$). Por tanto, en esa dirección la aceleración es nula y $\overrightarrow{v_{||}}$ es constante (en módulo y sentido).

- En el plano perpendicular a $\vec{B}$ el movimiento es circular, con radio $r = \frac{mv_{\perp}}{|q|B}$ y periodo $T = \frac{2\pi m}{|q|B}$.

En esta tarea calcularemos la dinámica usando la segunda ecuación de Newton. A partir de la fuerza obtendremos la aceleración e, integrando ésta, la velocidad y la posición de cada partícula a lo largo del tiempo. Los datos de entrada (campo magnético y descripción del sistema de partículas en un instante dado) y los de salida (posición y velocidad de las partículas a lo largo del tiempo) se almacenan en sendos ficheros de texto.

5.1.2. Descripción de la tarea

Diseña los siguientes dominios y subalgoritmos e impleméntalos en C/C++. No reinventes la rueda: siempre que sea posible, reutiliza un subalgoritmo que hayas diseñado y programado previamente.

Modelizaremos una partícula dando su carga (q), masa (m), posición (r) y velocidad (v). Modelizaremos un sistema como una colección de partículas (como máximo, 100). La componente nParticulas registra cuántas partículas tiene exactamente el sistema:

```
DOMINIO Vector3D = REGISTRO
    x, y, z SON reales;
FIN;

DOMINIO Particula = REGISTRO
    q, m SON reales
    r, v SON  Vector3D
FIN

DOMINIO Sistema = REGISTRO
    p  ES Particula[100]
    nParticulas ES ENTERO
FIN
```

La componente nParticulas registra cuántas partículas tiene exactamente el sistema.

sumaVectores

Dados dos datos del tipo Vector3D, $\overrightarrow{v}$ y $\overrightarrow{w}$, devuelve su suma: $\overrightarrow{v} + \overrightarrow{w}$

productoVectorial

Dados dos datos del tipo Vector3D, $\overrightarrow{v}$ y $\overrightarrow{w}$, devuelve su producto vectorial, $\overrightarrow{v} \times \overrightarrow{w}$

productoEscalarVector

Dados un real y un dato del tipo Vector3D, x y $\overrightarrow{v}$, devuelve el producto $x \cdot \overrightarrow{v}$

fuerzaLorentz

Dados un vector3D (que representa un campo magnético constante) y una partícula, $\overrightarrow{B}$ y p, devuelve la fuerza de Lorentz ejercida por $\overrightarrow{B}$ sobre p:

$$\overrightarrow{F}_{Lorentz} = q \cdot \overrightarrow{v} \times \overrightarrow{B} \tag{5.1}$$

dinamicaParticula

Dados un vector3D $\vec{B}$ (que representa un campo magnético constante), una partícula en un tiempo determinado t_0 (p) y dos reales, Δt y δt, devuelve la partícula p con su posición y velocidad un tiempo $t_0 + \Delta t$.

Ayuda: La velocidad y la posición en un tiempo t se obtienen, respectivamente, integrando la aceleración y la velocidad:
$\overrightarrow{v}(t_0 + \Delta t) = \overrightarrow{v}(t_0) + \int_{t_0}^{t_0+\Delta t} \overrightarrow{a}(t')dt'$
$\overrightarrow{r}(t_0 + \Delta t) = \overrightarrow{r}(t_0) + \int_{t_0}^{t_0+\Delta t} \overrightarrow{v}(t')dt'$

donde $\overrightarrow{a}(t') = \overrightarrow{F}(t')/m$ es la fuerza de Lorentz dividida por la masa. Sin pérdida de generalidad, tomaremos $t_0 = 0$. Una forma posible de calcular numéricamente esas

integrales es sustituirlas por sumatorios finitos, considerando intervalos muy pequeños de tiempo:

$$\begin{aligned} \overrightarrow{v}(\Delta t) &= \overrightarrow{v}(0) + q/m \sum_{l=0}^{n-1} \overrightarrow{v}(l \cdot \delta t) \times \overrightarrow{B} \cdot \delta t \\ \overrightarrow{r}(\Delta t) &= \overrightarrow{r}(0) + \sum_{l=0}^{n-1} \overrightarrow{v}(l \cdot \delta t) \cdot \delta t \end{aligned} \tag{5.2}$$

donde $n = \Delta t/\delta t$.

dinamicaSistema

Dados un campo magnético $\vec{B}$, un sistema de partículas en un tiempo determinado $t_0 = 0$, el nombre de un fichero y tres reales, t_{final}, Δt y δt, calcula la dinámica del sistema desde $t = t_0$ hasta $t = t_{final}$. En el fichero de salida se almacenará la información sobre dicha dinámica, tomado en intervalos Δt. Cada línea de dicho fichero tendrá la siguiente estructura: el valor de t, la posición $\vec{r}$ y el módulo de la velocidad para todas las partículas del sistema en ese tiempo.

A modo de ejemplo, el fichero de salida para un sistema de dos partículas, con $t_{final} = 1$ y $\Delta t = 0{,}2$, es del tipo:

```
0     0         0       0         1.4142   0        0        0        1.4142
0.2   0.1997   -0.0100   0.2001   1.4142   0.1987   0.0199   0.2001   1.4142
0.4   0.3975   -0.0399   0.4001   1.4142   0.3895   0.0790   0.4001   1.4142
0.6   0.5912   -0.0894   0.6002   1.4142   0.5648   0.1747   0.6002   1.4142
0.8   0.7791   -0.1580   0.8002   1.4142   0.7176   0.3034   0.8002   1.4142
1     0.9591   -0.2450   1.0003   1.4142   0.8417   0.4599   1.0003   1.4142
```

Por ejemplo, en $t = 0{,}2$ s, la posición de la primera partícula es $\overrightarrow{r_1} = (0{,}199724, -0{,}00999107, 0{,}200057)$, y su velocidad $v_1 = \sqrt{2}\ ms^{-1}$; la posición de la segunda partícula es $\overrightarrow{r_2} = (0{,}198727, 0{,}0199324, 0{,}200057)$, y su velocidad $v_2 = \sqrt{2}\ ms^{-1}$.

leeFichero

Dada una cadena de caracteres (que representa el nombre de un fichero de texto con información sobre el campo magnético y el sistema de partículas en un instante dado), lo lee y devuelve un booleano (*true*, en caso de haber abierto el fichero con éxito, *false* en caso contrario). Si ha conseguido abrir el fichero devuelve, además, el campo $\vec{B}$ y el sistema de partículas guardado en el fichero.

La estructura del fichero es la siguiente: En la primera línea se guardan las tres componentes del campo $\vec{B}$. En cada una de las siguientes líneas se guardan la masa, carga, posición y velocidad de una partícula en un tiempo t_0, como se muestra en el siguiente ejemplo:

```
0 0 1
9.109E-31   -1.602E-19   0   0   0   1.E7   0   0
9.109E-31    1.602E-19   0   0   0   1.E7   0   0
```

donde el campo magnético tiene un tesla, y está dirigido a lo largo del eje z. El sistema consta de dos partículas. Ambas tienen la misma masa ($9{,}109 \cdot 10^{-31}$ kg.), sus cargas son de signos opuestos y tienen el mismo valor absoluto ($1{,}602 \cdot 10^{-19}$ C), sus posiciones iniciales coinciden ($\vec{r_0} = (0, 0, 0)$) (las distancias vienen dadas en metros), así como sus velocidades iniciales ($10^7\ ms^{-1}$ en la dirección x).

Programa Principal

Un programa en C/C++, incluyendo las directivas del preprocesador y la definición de tipos y funciones que consideres adecuado.

El programa debe solicitar al usuario los nombres de los dos ficheros de texto, el de entrada y el de salida, así como un tiempo t_{final} y un intervalo Δt.

El fichero de entrada tiene información sobre el sistema en un tiempo $t_0 = 0$ (a priori no sabemos de cuántas partículas consta el sistema). El programa intenta abrir el fichero de entrada. Si no lo consigue, muestra un mensaje de error y finaliza. Si lo consigue, muestra el contenido del fichero por pantalla, y continúa.

Calcula un tiempo característico $T = \frac{2\pi m}{|q|B}$ y el tramo $\delta t = 10^{-5} \cdot T$. ($q$ y m son la carga y la masa de la primera partícula del sistema; B, el módulo del campo magnético).

En el fichero de salida se registrará la dinámica del sistema desde $t_0 = 0$ hasta t_{final}, tomado en intervalos Δt.

5.1.3. Ejemplos de ejecución

Puedes poner a prueba tu programa con distintos ficheros de entrada. Por ejemplo, supongamos el fichero 'ParElectronPositron.txt', con el siguiente contenido:

```
0 0 1
9.109E-31 -1.602E-19 0 0 0 1.E7 0 0
9.109E-31  1.602E-19 0 0 0 1.E7 0 0
```

El sistema está formado por un par electrón-positrón que se crea en el origen de coordenadas; inicialmente, ambas partículas se mueven paralelas al eje X, a una velocidad de $10^7\ ms^{-1}$. La ejecución del programa es:

```
Fichero donde leer los datos: ParElectronPositron.txt
Fichero donde guardar el resultado: salida.txt
Tiempo final? 3E-11
Con que intervalo de tiempo quieres guardar resultados? 3E-12
Campo= (0, 0, 1)
El sistema tiene 2 particulas:
         particula 0: m= 9.109e-031 q=-1.602e-019
                      r0=(0, 0, 0)  v0=(1e+007, 0, 0)
         particula 1: m= 9.109e-031 q=1.602e-019
                      r0=(0, 0, 0)  v0=(1e+007, 0, 0)
T= 3e-011 s;  intervalo = 3e-012 s;  deltaT = 3.57263e-016 s
```

Observa en el fichero de salida que la velocidad se mantiene constante (salvo errores debidos a la precisión de los cálculos). Es posible mostrar gráficamente la trayectoria de las partículas, como se muestra en la Figura 5.1

Consideremos ahora el fichero de entrada 'He.txt':

Figura 5.1: Trayectoria de las partículas del fichero 'ParElectronPositron.txt.txt'

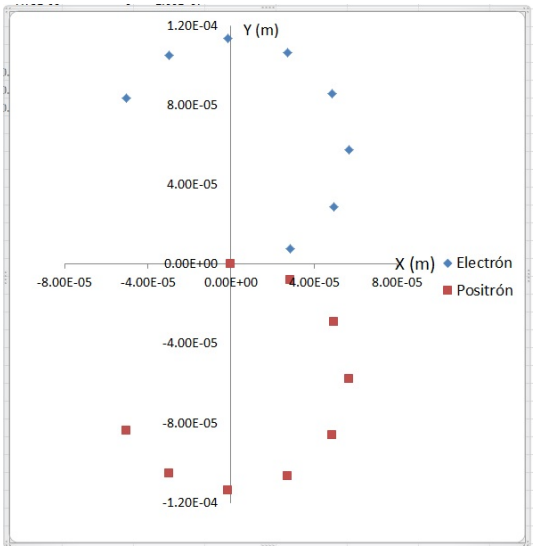
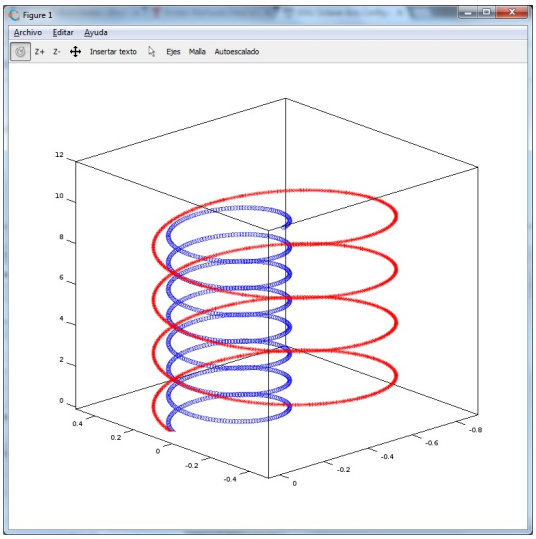


Figura 5.2: Trayectoria de las partículas del fichero 'He.txt'




```
0 0 1
6.64E-27  3.204E-19 0 0 0 1.E7 0 1.E7
6.64E-27  1.602E-19 0 0 0 1.E7 0 1.E7
```

¿Cuáles son las partículas representadas en el fichero?

Al ejecutar el programa con parámetros indicados a continuación:

```
Fichero donde leer los datos: He.txt
Fichero donde guardar el resultado: salidaHe.txt
Tiempo final? 1E-6
Con que intervalo de tiempo quieres guardar resultados? 1E-9
Campo= (0, 0, 1)
El sistema tiene 2 particulas:
```

```
         particula 0: m= 6.64e-027 q=3.204e-019
                      r0=(0, 0, 0)  v0=(1e+007, 0, 1e+007)
         particula 1: m= 6.64e-027 q=1.602e-019
                      r0=(0, 0, 0)  v0=(1e+007, 0, 1e+007)
T= 1e-006 s;  intervalo = 1e-009 s;  deltaT = 1.30213e-012 s
```

Se obtiene un fichero, cuyos datos se representan en la figura 5.2

Observa que la carga de la primera partícula es el doble que la de la segunda: su radio y su periodo son la mitad.

5.2. Método de Gauss-Jordan

5.2.1. Introducción

El método de Gauss-Jordan para la diagonalización de matrices puede ser aplicado 'a mano' de forma relativamente efectiva en matrices pequeñas. Su automatización, como la que desarrollamos en la Sección 4.2, se hace imprescindible cuando el tamaño de las matrices crece. Pero entonces, el elevado número de sus componentes hace que su introducción por teclado se convierte en una tarea penosa. En esta actividad completaremos la anterior añadiendo subalgoritmos para leer desde ficheros, tanto de texto como binarios, las matrices que deseamos tratar, así como para escribir los resultados también en estos tipos de ficheros.

Formato de los ficheros de entrada

Si el fichero es de texto, su primera línea contiene un número entero n, $0 < n \leq 20$: el número de filas y columnas de la matriz. En las siguientes líneas del fichero deberá haber, al menos n^2, números racionales en el formato <numerador>/<denominador>que se muestra en el ejemplo siguiente.

```
3
  1/1  2/1  3/1
  0/1  1/1  -3/1
  4/1  3/1  -2/1
```

No se exige que cada línea del fichero contenga exactamente los racionales que forman una fila de la matriz, y tampoco se exige que el carácter separador de un numerador y su correspondiente denominador sea '/', pudiendo ser cualquier otro. No obstante, cada número racional deberá estar separado del siguiente bien por un espacio, bien un tabulador o bien por un salto de línea. Recuérdese que cuando se intenta leer un número, entero o real, con la función fscanf, ésta ignora los blancos, tabuladores y saltos de línea que le preceden.

Si el fichero es binario, entonces el primer dato que contiene es un número entero (el tamaño de la matriz) y, a continuación, debe encontrase el código binario de, al menos, n^2 registros de tipo Racional.

Los racionales almacenados en estos ficheros no se encuentran, necesariamente, en forma canónica.

5.2.2. Descripción de la actividad

Diseña e implementa en C un programa que empiece solicitando al usuario el nombre del fichero, donde se almacena la matriz que se desea tratar, y su tipo (si es de texto o binario).

- El programa debe leer la matriz guardada en el fichero, que está formada por los primeros n^2 racionales almacenados en el fichero, donde n es el entero que figura en él como primer valor.

    1. Si la lectura fracasa, debido a que no hay suficientes números racionales o, simplemente, porque no se puede abrir el fichero, el programa mostrará un mensaje de error y terminará.

2. Si la lectura se realiza con éxito, el programa mostrará por pantalla la matriz que acaba de leer y le aplicará el subalgoritmo GaussJordan, creado en la actividad 4.2. A continuación el programa mostrará por pantalla el determinante de X y la matriz Y resultado de GaussJordan (esto es, Y es la inversa de X o bien una matriz escalonada equivalente).

- Finalmente, si el fichero es de texto el programa le añadirá, sea cual sea su contenido previo, las componentes de la matriz, escribiendo los números racionales en el formato +nnnnn/ddddd descrito en el subalgoritmo escribeR de la actividad 4.2, a razón de una fila por línea.

Diseña e implementa los siguientes subalgoritmos, que puedes utilizar en tu programa: **leeMatrizDeFichero**: dados una cadena de caracteres (que representa el nombre de un fichero con el formato descrito anteriormente) y un carácter (que expresa el tipo de fichero: de texto [t] o binario [b]), el subalgoritmo intenta leer del fichero una matriz de racionales.

El subalgoritmo devuelve, además de la matriz y su tamaño, un entero que indica si la lectura se ha realizado con éxito [1] ó si ha fracasado [0]. Como se ha descrito anteriormente, la lectura fracasa si no es posible abrir el fichero con el nombre indicado, si el carácter de tipo de fichero tiene un valor distinto de 't' o 'b', o si el fichero contiene menos de $n \times n$ números racionales, donde n es el primer dato almacenado en el fichero. Si la lectura fracasa, la matriz resultado y su tamaño quedan indefinidos.

escribeMatrizEnFichero: dados una matriz de $n \times n$ números racionales A, una cadena de caracteres que representa el nombre de un fichero, un carácter que indica su tipo ([t] texto, [b] binario), y un segundo carácter que indica el modo de acceso al fichero ([s] sobreescribir, [a] añadir) el subalgoritmo almacena en este fichero la matriz A.

Según el fichero sea de texto o binario, la matriz se guardará con el formato indicado en la sección formato de los ficheros de entrada.

El subalgoritmo devuelve un booleano (implementado como un entero) que indica si la operación se ha realizado o no con éxito. La operación fracasa si no es posible abrir el fichero con el nombre indicado, o si los caracteres de tipo de fichero o de modo de acceso tienen valores incorrectos.

5.2.3. Ejemplos de ejecución

Al ejecutar el programa, se obtienen pantallas similares a las siguientes:

```
Escribe el nombre del fichero de entrada: C:/Datos/entr.txt
Especifica el tipo de fichero: t
Fichero C:/Datos/entr.txt no encontrado
```

```
Escribe el nombre del fichero de entrada: C:/Datos/entrada.txt
Especifica el tipo de fichero: t

Matriz 3x3

La matriz leida es:
  1/1  2/1  3/1
  0/1  1/1  -3/1
  4/1  3/1  -2/1
```

```
El determinante de la matriz leida es: -29/1

La matriz obtenida es:
  -7/29  -13/29  9/29
 12/29   14/29 -3/29
  4/29    -5/29  -1/29
```

```
Escribe el nombre del fichero de entrada: C:/Datos/entrada.dat
Especifica el tipo de fichero: b

Matriz 3x3

La matriz leida es:
  1/1  2/1  3/1
  0/1  1/1  -3/1
  4/1  3/1  -2/1

El determinante de la matriz leida es: -29/1

La matriz obtenida es:
  -7/29  -13/29  9/29
 12/29   14/29 -3/29
  4/29    -5/29  -1/29
```

5.2.4. Ayuda

Para leer, en el programa principal, el nombre del fichero aconsejamos utilizar la función `char *gets(char *cadena)`, definida en el fichero `<stdio.h>`.

Para validar el programa, puedes crear un fichero de texto con cualquier editor (como el bloc de notas) y escribir allí la matriz a tratar. Utilizando los subalgoritmos anteriores, puedes leer la matriz del fichero de texto y guardarla en uno binario. De este modo puedes validar intensivamente los subalgoritmos `leeMatrizDeFichero` (lectura de ficheros de texto y binarios) y `escribeMatrizEnFichero` (escritura en binario y texto). Esta validación es opcional, sirve para que compruebes que tus subalgoritmos funcionan y no forma parte de la actividad propuesta (no debes entregarla).

Puedes escribir subalgoritmos especializados que sólo lean (o escriban) de un fichero de texto o de un fichero binario, de forma que `leeMatrizDeFichero` (`escribeMatrizEnFichero`, respectivamente) prácticamente puede limitarse a llamar a uno de ellos para realizar su tarea.

Para determinar si en el fichero binario hay números racionales puedes utilizar las funciones `fseek` y `ftell`.

5.3. Espectro γ del isótopo Cobalto-60 (^{60}Co)

5.3.1. Introducción

En esta actividad analizaremos el espectro de emisión de fotones (espectro γ) de un isótopo radiactivo. En concreto, el isótopo Cobalto-60 ($^{60}_{27}$Co) se desintegra a estados excitados del Niquel-60 ($^{60}_{28}$Ni), emitiendo un electrón. Estos estados tienen una semivida muy corta (del orden de picosegundos ps), y se desexcitan emitiendo fotones. Estos fotones son registrados por un detector, con lo que se obtiene un espectro que posteriormente se puede analizar.

A modo de ejemplo, el espectro γ sobre el que vamos a trabajar (ver figura 5.3) se ha obtenido utilizando un paquete de software RESTsoft [28] basado en C++ y ROOT [29], que usa el software GEANT4 [13] (para simular el paso y la interacción de partículas a través de la materia), recrea la microfísica de detectores gaseosos y analiza los datos. Agradecemos a Gloria Luzón, del Área de Física Atómica, Molecular y Nuclear, su colaboración para el uso de dichos paquetes.

En esta tarea vamos a suponer que los datos del espectro que se desea analizar vienen dados mediante un fichero binario con la siguiente estructura:

- Una cadena de 10 caracteres indicando en qué unidades de energía se dan los datos.
- Un real, ΔE, que indica el intervalo de energías con que se recogen los datos.
- Una colección de registros (E, f_E), donde f_E es el número de fotones detectados (frecuencia) en un intervalo de energías $[E - \Delta E, E + \Delta E)$.

Los registros que contenga el fichero estarán ordenados en orden creciente de energías y los valores de las energías, ΔE y E, se habrán almacenado como `float` (con 4 bytes).

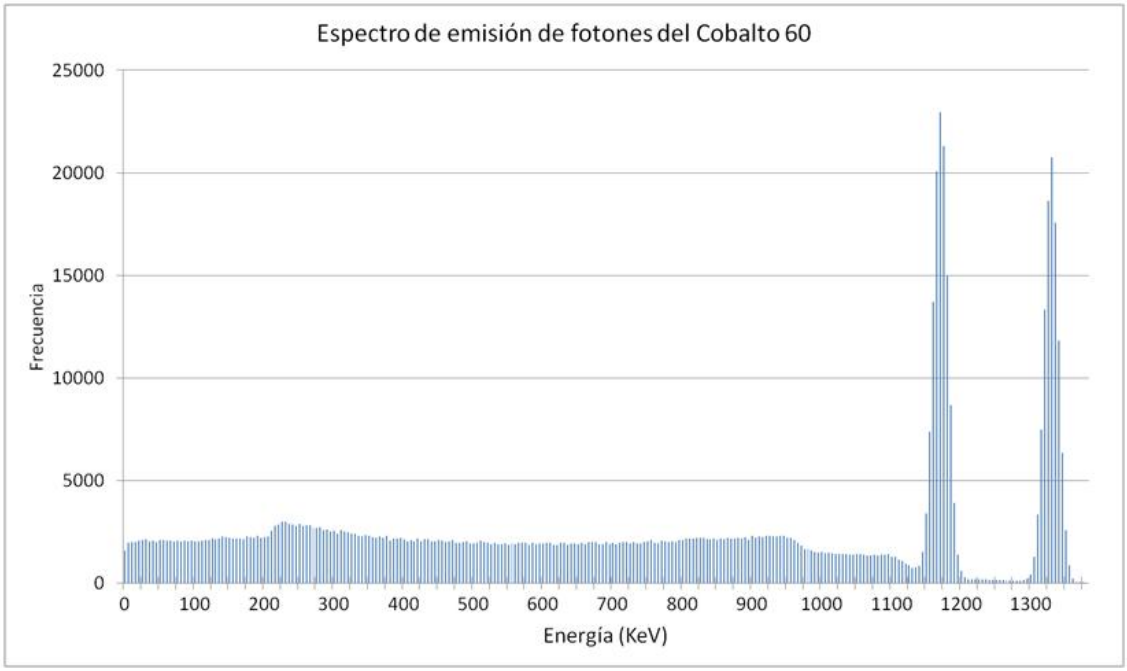


Figura 5.3: Espectro γ del Cobalto-60 (^{60}Co).

Dada una colección de registros (E, f_E), se pueden calcular los siguientes valores:

- Valor medio de la energía de los fotones: $\overline{E} = \frac{\sum f_E E}{\sum f_E}$
- Valor medio del cuadrado de la energía: $\overline{E^2} = \frac{\sum f_E E^2}{\sum f_E}$
- Desviación estándar de los datos: $\sigma_E = \sqrt{\overline{(E^2)} - (\overline{E})^2}$

donde los sumatorios recorren todos los registros de la colección.

5.3.2. Descripción de la tarea

Para llevar a cabo la tarea, en primer lugar, crea el dominio `Medida` con la siguiente especificación:

```
Dominio Medida = registro
    energ ES real
    frec ES entero
Fin
```

Después, diseña y programa en C/C++ los subalgoritmos que se indican a continuación.

analizaFichero

Este subalgoritmo toma como entradas una cadena de caracteres (el nombre de un fichero binario como el descrito anteriormente) y dos reales, $E_{\text{mín}}$ y $E_{\text{máx}}$. El subalgoritmo devuelve como resultados: un booleano, una cadena de 10 caracteres, tres reales (ΔE, $\overline{E}$, σ_E) y un entero n:

- El booleano será `true` si se ha podido abrir con éxito el fichero con los datos, `false` en caso contrario.
- La cadena indica en qué unidades se han tomado las medidas (es el primero de los datos almacenados en el fichero).
- El primer real, ΔE, indica el intervalo de energías con que se han recogido los datos (es el segundo de los datos almacenados en el fichero).
- Los reales $\overline{E}$ y σ_E son, respectivamente, el valor medio y la desviación estándar de la energía de las medidas del fichero cuyas energías se encuentran en el intervalo [$E_{\text{mín}}$, $E_{\text{máx}}$).
- El entero n es el número de medidas dentro de ese rango de energías.

Ten en cuenta que, a priori, no sabemos cuántos registros contiene el fichero. Podría incluso estar vacío.

guardaResultados

Este subalgoritmo tiene como entradas una cadena de caracteres (el nombre de un fichero de texto), cuatro reales y un entero. El subalgoritmo abre el fichero y añade, al final del mismo, una línea con esos cinco números seguida de un salto de línea; a continuación, cierra el fichero.

Programa Principal

El programa llevará a cabo las siguientes acciones:

1. Solicitará al usuario que teclee el nombre de un fichero binario con los datos de entrada. En caso de no poder abrir este fichero, el programa mostrará un mensaje de error y finalizará.

   Ayuda: para leer de teclado el nombre de un fichero se puede utilizar la función `gets()`.
2. En caso de que se pueda abrir el fichero de entrada, el programa solicitará al usuario el nombre del fichero de texto donde se guardarán los resultados, así como cuántos intervalos de energía se desean estudiar.
3. El programa creará el fichero de salida, que inicialmente contendrá únicamente una línea con los nombres de los valores que se desea guardar, seguida de un salto de línea y cerrará el fichero. En este momento, el contenido del fichero será del estilo:

   ```
   EMin    EMax    <E>             desviacion      nMedidas
   ```

4. A continuación, el programa irá solicitando al usuario los intervalos $[E_{\text{mín}}, E_{\text{máx}})$ que se desean analizar, y guardará los resultados en el fichero de salida.

Utiliza el programa para tratar los datos del fichero binario con el espectro del ^{60}Co que aparece en la figura 5.3 y analizar los dos picos del espectro, localizados en los intervalos de energía [1130, 1250) y [1250, 1400) (unidades expresadas en kilo-electrónvoltio KeV).

5.3.3. Ejemplos de ejecución

```
Fichero de entrada: D:\datos.dat

No se ha podido abrir el fichero D:\datos.dat
```

```
Fichero de entrada: D:\espectro_0.dat
Fichero de salida: D:\analisis.txt

 Numero de intervalos a estudiar: 1

   Introduzca los extremos del intervalo (en KeV): 0 10000
     No se han encontrado registros en el fichero
```

```
Fichero de entrada: D:\espectro.dat
Fichero de salida: D:\analisisEspectro.txt

 Numero de intervalos a estudiar: 2

   Introduzca los extremos del intervalo (en KeV): 1130 1250
     Unidades: KeU, intervalo = 2.5 KeV
     Intervalo [1130, 1250): numeroMedidas = 24 <E> = 1173.09 +- 12.931 KeV
```

```
   Introduzca los extremos del intervalo (en KeV): 1250 1400
     Unidades: KeU, intervalo = 2.5 KeV
     Intervalo [1250, 1400): numeroMedidas = 27 <E> = 1331.09 +- 12.1492 KeV
```

En este último caso, el fichero de salida tendrá el aspecto:

```
EMin    EMax    <E>             desviacion      nMedidas
1130    1250    1173.09         12.931          24
1250    1400    1331.09         12.1492         27
```

5.4. Análisis de ondas sonoras

5.4.1. Introducción

Disponemos de una serie de ficheros binarios, cada uno de los cuales almacena información sobre una nota ejecutada por un instrumento (para generar esos ficheros hemos procesado sendos ficheros de sonido en formato .wav).

Se desea realizar un análisis espectral de la información almacenada en esos ficheros. Como resultado de esta tarea, para cada sonido identificaremos su frecuencia fundamental (esto es, la nota) y sus armónicos (que definen el timbre del instrumento).

DTF: la Transformada Discreta de Fourier

Consideremos una magnitud física que evoluciona en el tiempo, $p(t)$, de la que hemos tomado medidas a intervalos regulares de tiempo. Sea Δ el intervalo de tiempo entre medida y medida. Esto es, disponemos de una secuencia de n medidas $(p_0, p_1, \ldots, p_{n-1})$, donde $p_m = p(m\Delta)$ y $m \in [0, n-1]$.

La *Transformada Discreta de Fourier (DFT)* de la secuencia $(p_0, p_1, \ldots, p_{n-1})$ es otra secuencia de n valores $(F_0, F_1, \ldots, F_{n-1})$ dados por:

$$F_k = \sum_{m=0}^{n-1} p_m e^{i2\pi km/n}, \text{para } k \in [0, n-1] \ . \tag{5.3}$$

Es decir, hemos obtenido n números (F_k) a partir de otros n (p_m). En general, podemos considerar que tanto los números iniciales como los finales son complejos. La transformada tiene las siguientes propiedades:

i) Las secuencias (p_m) y (F_k) contienen la misma información: del mismo modo que hemos obtenido (F_k) a partir de (p_m), es posible obtener (p_m) a partir de (F_k) mediante la inversa de la DFT:

$$p_m = \frac{1}{n} \sum_{k=0}^{n-1} F_k e^{-i2\pi km/n}, \text{para } m \in [0, n-1] \ .$$

ii) Si las medidas p_m son reales, entonces F_0 es un número real y $F_k = \overline{F}_{n-k}$, el complejo conjugado de F_{n-k}. (Recuerda que el *conjugado* de un número complejo $z = a + ib$ es $\overline{z} = a - ib$.)

iii) Combinando (i) y (ii) se obtiene que, si las medidas p_m son todas reales, entonces

$$p_m = \sum_{k=0}^{n/2} A_k \operatorname{sin}\left(\frac{2\pi km}{n} + \varphi_k\right), \text{ para } m \in [0, n-1] \ ,$$

donde $A_k = \frac{c_k}{n} |F_k|$, con $|F_k|$ el módulo de F_k y c_k es una constante definida por

$$c_k = \begin{cases} 1 \ , \text{ si } k = 0 \\ 1 \ , \text{ si } k = n/2 \text{ con } n \text{ par} \\ 2 \ , \text{ en el resto de casos} \end{cases}$$

O, equivalentemente, $p(m\Delta) = \sum_{k=0}^{n/2} A_k \operatorname{sin}(2\pi f_k m\Delta + \varphi_k)$, donde $f_k = \frac{k}{n\Delta}$.

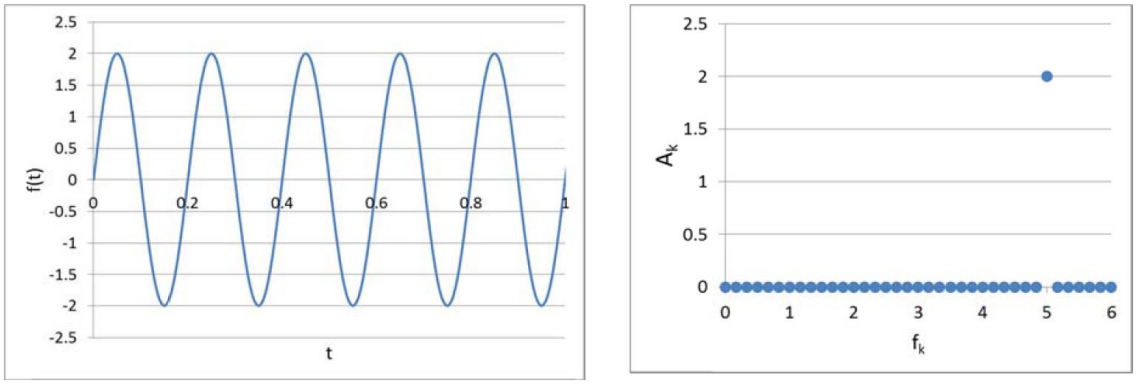


Figura 5.4: Representaciones equivalentes de la onda $f(t) = 2\sin(2\pi \cdot 5t)$

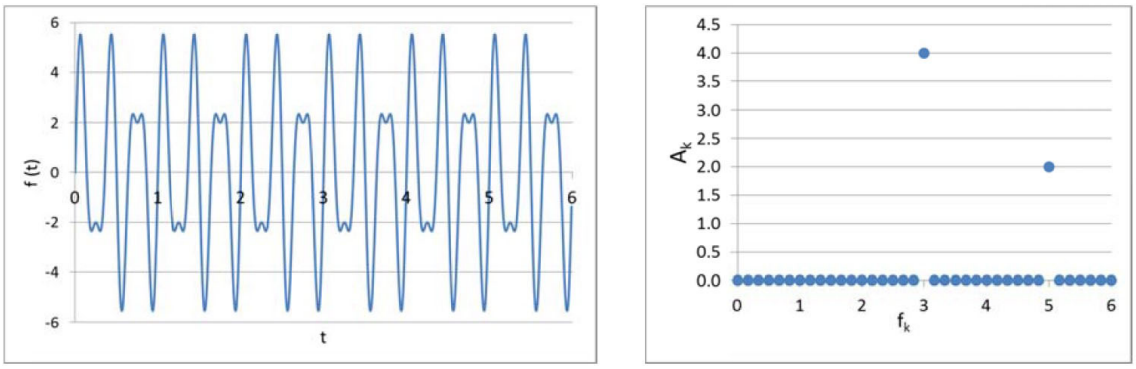


Figura 5.5: Representaciones equivalentes de la onda $f(t) = 2\sin(2\pi \cdot 5t) + 4\sin(2\pi \cdot 3t)$

Esta última expresión nos permite dar una visión intuitiva de la DFT: es posible reconstruir nuestras medidas mediante una combiandción lineal de funciones armónicas, cada una de las cuales tiene una amplitud A_k y una frecuencia f_k. La DFT nos proporciona los coeficientes de dicha combinación.

Para ilustrar las ideas previas, en las figuras 5.4 y 5.5, mostramos una doble representación de las ondas $f(t) = 2\sin(2\pi \cdot 5t)$ y $g(t) = f(t) + 4\sin(2\pi \cdot 3t)$: a la izquierda se presenta la representación usual de la onda frente al tiempo, mientras que a la derecha se representa la amplitud A_k frente a la frecuencia f_k.

En esta tarea analizaremos ondas sonoras. Además de las amplitudes A_k mostraremos una magnitud física más significativa: el nivel de intensidad de la onda, expresado en decibelios.

Nivel de intensidad del sonido

Como cualquier otra onda, las ondas sonoras transmiten energía. La *intensidad* del sonido se define como la potencia media por unidad de superficie perpendicular a la dirección de propagación de la onda sonora. Y el *nivel de intensidad* se define como:

$$\beta = 10 \log_{10} \left(\frac{I}{I_{\text{ref}}} \right)$$

donde I es la intensidad de la onda sonora e $I_{\text{ref}} = 10^{-12}$ Wm^{-2} es el umbral de audición (la intensidad mínima que debe tener un sonido para ser audible), y se mide en *decibelios*.

Supongamos una onda armónica, cuya presión venga dada por $A_k \sin(2\pi f_k t)$. Dado que la potencia media es proporcional al cuadrado de la amplitud de la onda de presión, podemos escribir:

$$\beta = 20 \log_{10} \left(\frac{A_k}{p_{\text{ref}}} \right) ,$$

donde $p_{\text{ref}} = 2 \cdot 10^{-5}$ pascales (ese valor se obtiene a partir de I_{ref}, la densidad del aire y de la velocidad de propagación del sonido).

Se puede encontrar más información sobre este punto, por ejemplo, en el libro "Física para la Ciencia y la Tecnología", de G. Mosca y P.A. Tipler [33].

5.4.2. Descripción de la tarea

En esta tarea trabajaremos con *ficheros binarios*, en los que todos los números reales se han codificado con el tipo `float`. La estructura de los ficheros es la siguiente:

- En primer lugar, un número real Δ, el intervalo de tiempo con el que se han tomado un cierto número de medidas.
- A continuación, una secuencia de medidas p_n, cada una de las cuales es un número real que representa la presión de una onda sonora en un instante $n\Delta$.

Comienza creando un nuevo tipo de datos para operar con números complejos. La representación puede ser la que se sugiere más abajo. Después, diseña y programa subalgoritmos para:

a. crear números complejos a partir de sus campos (parte real e imaginaria).
b. sumar números complejos
c. multiplicar un número real por un complejo, y
d. calcular el módulo de un complejo.

```
struct complejo {
    float real, imag;
};
typedef struct complejo Complejo;
```

Finalmente, para procesar los ficheros que contienen datos de sonidos, diseña y programa los subalgoritmos siguientes:

nMedidas

Dada una cadena de caracteres, el nombre de un fichero binario con información de un sonido, que tiene la estructura indicada al principio de esta sección, devuelve el número de medidas almacenadas en él. Recuerda que el primer valor del fichero no es una medida; por el contrario es el intervalo de tiempo Δ con el que se han tomado éstas.

DFT

Recibe, como entradas, dos cadenas de caracteres y un entero n. Una de las cadenas es el nombre de un fichero binario con la estructura indicada al principio de esta sección; la otra es el nombre de un fichero de texto en el que se guardarán los resultados.

El subalgoritmo abrirá el fichero binario y calculará las $n/2+1$ componentes de la DFT de las n primeras medias $(p_0, \ldots, p_{n-1})$ contenidas en el fichero según la fórmula:

$$F_k = \sum_{m=0}^{\mathtt{n}-1} p_m e^{i2\pi km/\mathtt{n}}, k \in [0, n/2] \ .$$

Recuerda que el primer valor del fichero es Δ y no corresponde a una medida.

En el fichero de salida el programa escribirá, en cada línea, los valores f_k, Real(F_k), Imag(F_k), A_k y β_k, para cada $k \in [0, n/2]$, donde:

- $f_k = \frac{k}{n\Delta}$ es una frecuencia
- F_k es la k-ésima componente de la DFT
- $A_k = \frac{c_k}{\mathtt{n}} |F_k|$, donde $|F_k|$ es el módulo de F_k y c_k toma los valores:

$$c_k = \begin{cases} 1 \text{ , si } k = 0 \\ 1 \text{ , si } k = n/2 \text{ con } n \text{ par} \\ 2 \text{ , en el resto de casos} \end{cases}$$

- $\beta_k = 20 \log_{10}(A_k/p_{\text{ref}})$ es el nivel de intesidad de la onda sonora asociado a la frecuencia f_k, medido en decibelios. Recuerda que $p_{\text{ref}} = 2 \cdot 10^{-5}$ pascales.

El subalgoritmo devolverá un booleano: `true` si se ha realizado con éxito la transformada, `false` en caso contrario (si no se ha podido abrir alguno de los ficheros, o si el de entrada contiene menos de n medidas).

Escribe un programa en C/C++ que solicite al usuario los nombres de un fichero de entrada y de salida junto con un entero n. El programa invocará al subalgoritmo DFT. Si éste no finaliza correctamente se mostrará en pantalla un mensaje informativo y se dará opción al usuario a introducir nuevos valores.

5.4.3. Ejemplos de ejecución

En la plataforma Moodle hemos colgado varios fichero para que compruebes tu programa. El más sencillo, "ficheroPruebaFourier.dat", contiene los siguientes valores: $\Delta = 1, p = (0, 1, 2, 3)$.

A continuación mostramos un ejemplo de la ejecución del programa:

```
Fichero de entrada: ficheroPruebaFourier.dat
Fichero de salida: resultado.txt
Numero de medidas que desea procesar: 10

El fichero ficheroPruebaFourier.dat solo tiene 4 medidas

Fichero de entrada: ficheroPruebaFourier.dat
Fichero de salida: resultado.txt
Numero de medidas que desea procesar: 4
```

Y el contenido del fichero de salida es el siguiente

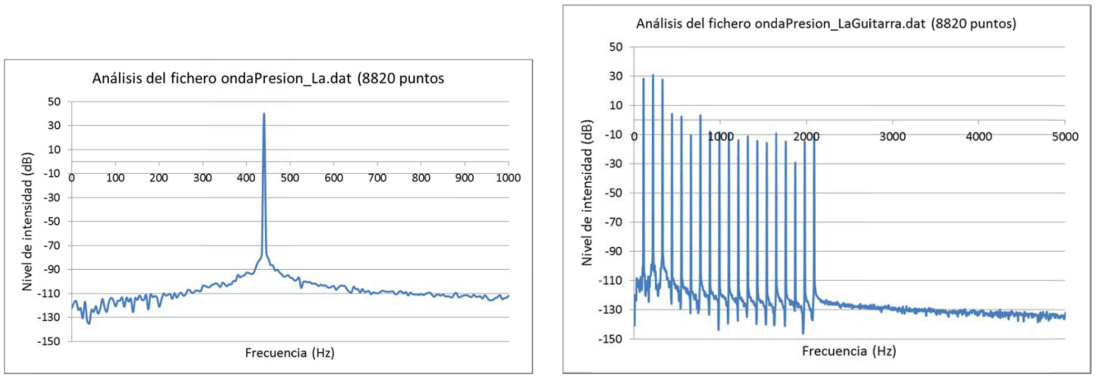


Figura 5.6: Representación de la superficie con ecuación 3.15. A la derecha se muestra su proyección sobre el plano horizontal: cuanto más vivo es el color, mayor es la altura $z(x, y)$.

```
0       6       0               1.5     97.5012
0.25    -2      -2              1.41421 96.9897
0.5     -2      7.34764e-016    0.5     87.9588
```

Los ficheros "onda1.dat" y "onda2.dat" contienen los valores $\Delta = 0{,}01$ y, a continuación, mil medidas de las ondas onda1 y onda2 de las figuras 5.4 y 5.5, respectivamente.

Los ficheros "ondaPresion*.dat" contienen información de ondas de presión de distintos sonidos, interpretados por varios instrumentos. En la figura 5.6 se muestran los resultados del análisis, utilizando 8820 medidas, de los ficheros "ondaPresion_La.dat" (izquierda) y "ondaPresion_LaGuitarra.dat" (derecha). Ambos se han obtenido a partir de sendos ficheros ".wav": el primero registra una onda sonora de frecuencia 440 Hz, mientras que el segundo registra la nota obtenida al pulsar la quinta cuerda de una guitarra. Observa que, en este último, la nota consta de una frecuencia fundamental de 110 Hz y varios armónicos. La frecuencia fundamental define la nota (La, en este caso). La cantidad y el nivel de intensidad de los armónicos determinan el timbre del instrumento.

5.4.4. Para saber más...

Al realizar esta práctica puede que te hayan surgido algunas preguntas. En esta sección damos respuesta a dos de ellas.

P) Al principio hemos tomado una muestra discreta de una función que es continua. ¿No estamos perdiendo información?

No forzosamente. El teorema del muestreo, de Nyquist–Shanon, establece que una función continua $h(t)$ cuya anchura de banda esté limitada a frecuencias menores que f_c, es decir: $H(f) = 0$ para toda frecuencia $|f| > f_c$, está completamente determinada por un conjunto discreto de valores h_m si $\Delta < 1/2f_c$.

El enunciado no parece especialmente impresionante. Sin embargo, sus consecuencias son muy relevantes. Tomemos por ejemplo una onda sonora. La frecuencia máxima que es capaz de percibir el oído humano es $f_c = 20$ kHz. Si tomamos muestras discretas de la onda a intervalos periódicos $\Delta < 40000$, a partir de esas muestras discretas es posible reconstruir la onda inicial de modo que el oído no note la diferencia. Precisamente, en los ficheros .wav la velocidad de muestreo es de ese orden (44100).

P) He estado comprobando el funcionamiento del subalgoritmo DFT usando "ondaPresion.dat" como fichero de entrada, y distintos valores de n. Para valores pequeños, el programa funciona razonablemente bien. He leído que el fichero tiene 65536 *medidas. He probado a ejecutar el programa con ese valor de n, pero parece que se queda bloqueado... ¿por qué?*

¿Cuántas operaciones hay que hacer para calcular cada valor de p_k? De la fórmula 5.3 se sigue que necesitamos:

- Calcular n coeficientes complejos $e^{i2\pi km/n}$
- Realizar n multiplicaciones de complejos.
- Realizar n sumas de complejos

Esto es, el cálculo de cada coeficiente p_k implica del orden de n operaciones. Como tenemos n coeficientes por calcular, el número total de operaciones es del orden de n^2, $O(n^2)$. Puedes comprobarlo midiendo el tiempo de ejecución del programa para distintos valores de n. En mi ordenador obtengo los siguientes tiempos:

n	Tiempo
1000	0,120 segundos
2000	0,427 segundos
3000	0,934 segundos
4000	1,649 segundos
8000	6,700 segundos

Observa que al duplicar el número de medidas el tiempo se multiplica por 4; al triplicar n, por 9;... Por tanto, con 65536 medidas el programa debería ser, arpoximadamente, $65{,}54^2 = 4925$ veces más lento que para 1000 medidas... casi 10 minutos de ejecución.

Existe un algoritmo muchísimo más eficiente para calcular la DFT, llamado (de modo nada sorprendente) *Transformada Rápida de Fourier (FFT)*, que necesita sólo $O(n \log_2 n)$ operaciones. Pero este curso no podremos verlo.

5.5. Proyecto RSA: ficheros

5.5.1. Introducción

En esta tarea desarrollaremos los tres algoritmos básicos de que consta el sistema RSA: creación de claves, encriptación y desencriptación. A partir de un fichero de texto que contiene una pareja de números primos p y q generaremos dos ficheros, uno con una clave pública y otro con la clave privada correspondiente. A partir de dos ficheros (uno con una clave pública y otro con un texto), generaremos un fichero con el mensaje encriptado. A partir de dos ficheros (uno con una clave privada y otro con un mensaje encriptado), generaremos un fichero con el texto obtenido al desencriptar el mensaje.

5.5.2. Descripción de la tarea

Diseña y programa en C/C++ los siguientes subalgoritmos:

codifica

Dado un carácter `c`, devuelve una cadena que representa su código entero asociado usando dos dígitos.

El subalgoritmo recorre el array `codificacion` en busca del carácter `c`. Si lo halla, toma el índice de la componente de `codificacion` tal que `codificacion[indice]` = `c`, lo transforma en un string usando dos dígitos y lo devuelve. Si tras recorrer todo el array no encuentra el carácter buscado, devuelve `"00"`.

A continuación mostramos algunos ejemplos de caracteres junto con su código asociado.

c	índice en `codificacion`	Código asociado
`' '`	0	`"99"`
`'E'`	5	`"05"`
`'u'`	47	`"47"`
`'r'`	44	`"44"`
`'e'`	31	`"31"`
`'k'`	37	`"37"`
`'a'`	27	`"27"`
`'!'`	68	`"70"`

codificaTexto

Dada una cadena de caracteres, `texto`, devuelve otra, `textoCodificado`, obtenida al concatenar las cadenas de caracteres que codifican cada carácter de `texto`.

A continuación mostramos algunos ejemplos de cadenas de caracteres junto con su código asociado.

`texto`	`textoCodificado`
`"Eureka !"`	`"0547443137279970"`
`"Zaragoza"`	`"2627442733415227"`
`"It is all Greek to me!"`	`"094699354599273838990744313137994641993931 70"`

encripta

Este subalgoritmo toma como entradas tres cadenas: $n, e, Texto$, que se interpretan del siguiente modo: (n, e) es la clave pública, y $Texto$ un texto que se desea encriptar. Devuelve una cadena de caracteres, M_{encrip} (el mensaje encriptado). Sigue el algoritmo indicado a continuación:

i) Se codifica $Texto$, generando una cadena M.

ii) Se calcula $M_{encriptado} = M^e \,\%n$

Ejemplo:

n	e	Texto	$M_{encriptado}$
`"276121"`	`"307"`	`"ABC"`	`"169111"`

leeEnteros

Dada una cadena de caracteres, $nombrFich$ (que representa el nombre de un fichero de texto que contiene dos enteros), devuelve un booleano y dos cadenas de caracteres. Si la lectura se ha realizado con éxito (se ha conseguido abrir el fichero, y se han leído con éxito dos cadenas de caracteres), el booleano es `true`, y las cadenas corresponden a los enteros alamacenados en el fichero. En caso contrario, el booleano es `false`.

Sugerencia: Para leer el fichero $nombrFichClave$ utiliza la función `fscanf`.

creaClaves

Dadas tres cadenas de caracteres, $nombrFichPrimos$, $nombrFichClavePubl$, $nombrFichClavePriv$ (que representan los nombres de tres ficheros de texto: el primero contiene dos primos p y q, el segundo una clave pública y el tercero una clave privada generadas a partir de p y q), devuelve un booleano b. El subalgoritmo realiza los siguientes pasos:

- Lee el contenido del fichero $nombrFichPrimos$. Este consta de dos números primos p y q. Si hay algún problema en la lectura, el subalgoritmo devuelve $b \leftarrow false$ y finaliza. En caso contrario, prosigue: A partir de p y q construirá los números $n = p \cdot q$ y $\varphi(n) = (p-1) \cdot (q-1)$.
- Genera un número e coprimo con $\varphi(n)$. Para ello parte de un número impar cualquiera (por ejemplo, `"12345"`) y lo va incrementando en 2 unidades cada vez (prueba con `"12347"`, `"12349"`...), hasta encontrar un número cuyo máximo común divisor con $\varphi(n)$ sea 1.
- Encuentra el inverso de e módulo n, d.
- Guarda el par n, e en el fichero $nombrFichClavePubl$.
- Guarda el par n, d en el fichero $nombrFichClavePriv$.

A modo de ejemplo, el contenido del fichero `parPrimosPrueba` es

```
135277
8368249
```

Con el algoritmo indicado, el contenido del fichero *nombrFichClavePubl* sería

```
1132031619973
12347
```

y el de *nombrFichClavePriv*:

```
1132031619973
187127171027
```

Si el subalgoritmo finaliza correctamente, el valor b devuelto es `true`. En caso contrario (si no se ha conseguido leer *nombrFichPrimos* o escribir en *nombrFichClavePubl*, *nombrFichClavePriv*, devuelve `false`.

Sugerencia: Para leer el fichero *nombrFichPrimos* utiliza la función `fscanf`.

encriptaFichero

Dadas tres cadenas de caracteres, $nFClavPubl$, $nFTexto$, $nFMEncr$ (que representan los nombres de tres ficheros de texto: el primero contiene la clave pública, el segundo el texto que deseamos encriptar, el tercero el mensaje encriptado), devuelve un booleano b. El subalgoritmo realiza los siguientes pasos:

- Lee la clave pública (n, e) almacenada en el fichero $nFClavPubl$.
- Determina `nCar`, el tamaño del array en que se almacenarán los caracteres leídos del fichero $nFTexto$. El resultado es:

  ```
  nCar = (strlen(n)+1)/2
  ```

  Ese valor garantiza que el número obtenido al codificar el texto leído, M, sea menor que n (la primera componente de la clave). Declara el array `char texto[nCar]`, que se usará para ir leyendo el fichero.
- Lee el fichero $nFTexto$: Lee `nCar-1` caracteres cada vez, guardándolos en el array `texto`; encripta `texto` utilizando la clave pública (n, e) y guarda el mensaje encriptado en una línea distinta del fichero $nFMEncr$.

El booleano devuelto es `false` si ha habido algún problema al leer el fichero $nFClavPubl$, o en la apertura de los ficheros $nFTexto$ y $nFMEncr$, `true` en caso contrario.

Sugerencia: para leer `nCar-1` caracteres cada vez del fichero $nFTexto$, utiliza la función `fgets (texto, nCar, f)`, donde `texto` es el array utilizado para leer, y `f` el fichero que estamos leyendo. La función `fgets` se describe en la sección *Ayudas*.

A modo de ejemplo, el contenido del fichero `TextoBreve` es

```
How does it feel
to be on your own?
```

Al encriptarlo con la clave pública del ejemplo del subalgoritmo **creaClaves**, el contenido del fichero $nFMEncr$ será

```
358538330536
524096427642
425513812072
848759050252
613394893713
262410278417
516591581121
```

desencriptaFichero

Dadas tres cadenas de caracteres, $nFMEncr$, $nFClavPriv$, $nFTexto$(que representan los nombres de tres ficheros de texto: el primero contiene un mensaje encriptado, el segundo una clave privada, en el tercero se almacenará el mensaje desencriptado), devuelve un booleano b. El subalgoritmo realiza los siguientes pasos:

- Lee la clave privada (n, d) almacenada en el fichero $nFClavPriv$.
- Lee, uno a uno, los *strings* almacenados en $nFMEncr$; desencripta cada uno, utilizando la clave privada (n, d) y guarda el texto desencriptado en el fichero $nFTexto$.

El booleano devuelto es `false` si ha habido algún problema al leer el fichero $nFClavPriv$, o en la apertura de los ficheros $nFTexto$ y $nFMEncr$, `true` en caso contrario.

Sugerencia: para leer el fichero $nFMencr$, utiliza la función `fscanf`.

A modo de ejemplo, al desencriptar el fichero obtenido en el apartado anterior usando la clave privada del ejemplo del subalgoritmo **creaClaves** se obtendría un fichero con el mismo contenido que `TextoBreve`.

Programa Principal

El programa empezará soliciando al usuario que introduzca tres cadenas de caracteres: el nombre de un fichero que contiene dos primos ($nombrFichPrimos$), y los nombres de los ficheros donde se almacenarán, respectivamente, las claves pública y privada ($nombrFichClavePubl$, $nombrFichClavePriv$). A continuación, lee el contenido del primer fichero, genera las claves y las guarda en los otros dos.

Después, solicita al usuario que introduzca otras dos cadenas: los nombres de sendos ficheros de texto ($nFTexto$ y $nFMEncr$). $nFTexto$ contiene un texto que se desea encriptar. Usando la clave pública de $nombrFichClavePubl$, encripta el contenido del primer fichero ($nFTexto$) y guarda el resultado en $nFMEncr$.

Después, solicita al usuario que introduzca otras dos cadenas: los nombres de sendos ficheros de texto ($nFMencr2$ y $nFTexto2$). $nFMencr2$ contiene un mensaje encriptado. Usando la clave privada de $nombrFichClavePriv$, desencripta el contenido del primer fichero ($nFMencr2$) y guarda el resultado en $nFTexto2$.

Si en algún momento detecta un error al manejar un fichero, el programa muestra un mensaje por pantalla y finaliza.

5.5.3. Ayudas

El valor por defecto para los arrays auxiliares de la tarea anterior, `longMax`, sigue siendo válido en ésta.

Ficheros de prueba

La carpeta comprimida `ficherosPrueba` contiene varios ficheros de prueba:

- `parPrimos60`, `parPrimos50`,`parPrimos40`, `parPrimosPrueba`, que contienen sendas parejas de números primos de 60, 50, 40 y 6/7 dígitos respectivamente.
- `Texto` y `TextoBreve`, que contienen textos para encriptar.
- `Texto_Encriptado_ClaveB`, que contiene un mensaje encriptado, y `clavePrivadaB`, que contiene la clave privada que permite desencriptarlo.

Ejemplo de ejecución

En este ejemplo se encripta un fichero, y se desencripta el mensaje encriptado. El fichero resultante (`TextoFinal`) debe tener el mismo contenido que el inicial (`Texto`).

```
Fichero con p, q:           D:\prueba\parPrimos40.txt
Fichero con claves publicas:D:\prueba\fichClavePublica.txt
Fichero con claves privadas:D:\prueba\fichClavePrivada.txt

Fichero que encriptar:              D:\prueba\Texto.txt
Fichero donde guardar el resultado: D:\prueba\Texto_Encriptado.txt

Fichero que desencriptar:           D:\prueba\Texto_Encriptado.txt
Fichero donde guardar el resultado: D:\prueba\Texto_Final.txt
```

Funciones auxiliares

```
char * fgets ( char * str, int num, FILE * flujo );
```

Descripción

Lee caracteres de un `flujo` y las almacena en `flujo`, hasta que lleva leídos `num`-1 caracteres o encontrar un carácter de salto de línea o de fin de fichero (lo que ocurra primero). En los dos últimos casos, el carácter de fin de línea o de fichero se incorpora también a `str`. `flujo` puede ser un fichero; si se desea leer de teclado, se asigna a `flujo` el valor `stdin`. Tiene la ventaja de especificar el número máximo de caracteres leídos, lo que permite evitar el desbordamiento de `str`.

Parámetros

- *str*: puntero a un array de caracteres donde se copia la cadena leída.
- *num*: número máximo de caracteres copiados en *str* (incluyendo el carácter de fin de cadena).
- *flujo*: Puntero a la estructura que identifica el flujo de entrada.

Devuelve

Si se ejecuta con éxito, devuelve el valor del puntero `str`. Si no, devuelve un puntero nulo.

Ejemplo

```
void pruebaFgets()
{
    int nCar = 7;
    char nombrFich[100], texto[nCar];
    printf ("Introduzca el nombre de un fichero: ");
    scanf ("%s", nombrFich);
    printf ("\nEl contenido del fichero es:\n");
    FILE *f = fopen (nombrFich, "r");
    if ( f == NULL)
        printf ("Error al abrir el fichero %s\n", f);
    else
    {
        fgets (texto, nCar, f);
        while (!feof (f))
        {
            printf ("[%s]", texto);
            fgets (texto, nCar, f);
        }
        fclose (f);
    }
}
```

Al ejecutar `ejemploFgets()` se muestra por pantalla:

```
Introduzca el nombre de un fichero: D:\prueba\TextoBreve.txt

El contenido del fichero es:
[How do][es it ][feel
][to be ][on you][r own?][
]
```

5.6. Leyes electorales

5.6.1. Introducción

Disponemos de un fichero de datos que contiene información sobre circunscripciones electorales. Cada línea del fichero está formada por el nombre de la circunscripción, su número de escaños, su número de partidos y una lista de parejas de nombre de partidos y votos recibidos en la circunscripción. Así, la línea

```
Huesca 3 4 PP 42351 PSOE 29866 Podemos 22251 Cs 17906
```

informa de que en la provincia de Huesca disponemos de 3 escaños y hay cuatro partidos de los que se indica su nombre y número de votos obtenidos.

Vamos a trabajar con el supuesto de distrito único. Esto significa que para repartir los escaños se suman los votos obtenidos por cada uno de los partidos en todas las circunscripciones contenidas en el fichero y entre ellos se reparte un número de escaños igual a la suma de los escaños de todas las circunscripciones. Vuestra tarea consiste en mostrar el reparto de escaños con el método D'Hont y con el método Hare.

5.6.2. Descripción de la tarea

Diseña y programa en C/C++ un algoritmo que dado un fichero de texto que contiene información de varias circunscripciones muestre el reparto de escaños bajo el supuesto de distrito único y de acuerdo con los métodos d'Hont y Hare. En el reparto debes mostrar los partidos ordenados de mayor a menor número de escaños asignados. Los pasos a seguir son:

1. Lee el nombre de dos ficheros de texto, uno de entrada y otro de salida
2. Extrae del fichero de entrada el total de escaños, el total de partidos distintos así como parejas formadas por un nombre de partido y el total de votos conseguidos por el partido
3. Ordena los partidos y sus votos en orden decreciente de número de votos. De este modo, cuando hagamos el reparto del número de escaños por partido, éstos aparecerán ordenados en sentido decreciente
4. Asigna escaños a cada partido usando los métodos d'Hont y Hare
5. Guarda el resultado en un fichero de salida

5.6.3. Estructuras y subalgoritmos

Para resolver esta tarea, usa la siguiente estructura:

```
struct circunscripcionUnica{
  int numeroPartidos;
  int numeroEscanos;
  char nombrePartidos[TMAX][LMAX];
  int votosPartidos[TMAX];
};
```

En la estructura `circunscripcionUnica`, los campos son: el número de partidos de que se dispone de resultados, el número total de escaños a repartir, el nombre de los partidos,

cuya longitud máxima es `LMAX-1`, y el número de votos por partido. El número máximo de partidos es `TMAX`.

Como ejemplo, usamos el fichero `datos.txt`, que corresponde a los resultados en las tres provincias aragonesas en las elecciones generales de 2016 y cuyo contenido mostramos a continuación:

```
Huesca 3 4 PSOE 29866 Podemos 22251 Cs 17906 PP 42351
Zaragoza 7 5 PACMA 5032 PP 178555 PSOE 124849 Podemos 103358 Cs 85884
Teruel 3 4 PP 30837 PSOE 19638 Podemos 12442 Cs 9820
```

Algunos subalgoritmos que puedes usar son los siguientes (puedes añadir otros si lo crees necesario):

anadePartido

Dada una estructura de tipo `circunscripcionUnica`, un nombre de partido que no existe en la estructura y un número de votos, añade el partido y el número de votos después de la última componente ocupada de los campos `nombrePartidos` y `votosPartidos` respectivamente.

perteneceA

Dada una estructura de tipo `circunscripcionUnica` y un nombre de partido devuelve un booleano y un entero. El booleano es `true` si el nombre del partido aparece en el campo `nombrePartidos` de la estructura y en ese caso el entero es la posición del vector de nombres de partidos en la que aparece el nombre del partido. El booleano es `false` si el nombre del partido no aparece en la estructura de tipo `circunscripcionUnica` y en ese caso el entero es -1.

Para comparar cadenas de caracteres o para copiar una cadena en una variable de tipo cadena, puedes usar las funciones de cadena que encontrarás en el apartado E

extraePartidosyVotos

Dado un nombre de un fichero, devuelve una estructura de tipo `circunscripcionUnica` y un booleano. El booleano es `true` si el proceso se ha completado con éxito y en ese caso la estructura de tipo `circunscripcionUnica` contiene el número total de partidos distintos encontrados, el número total de escaños, los partidos y los votos totales obtenidos por cada uno de ellos extraídos del fichero de entrada. El booleano es `false` si el proceso no se ha completado con éxito y en ese caso la estructura queda indefinida.

Así, para el fichero `datos.txt` del ejemplo, la estructura de tipo `circunscripcionUnica` devuelta es

```
{5, 13, {"PSOE", "Podemos","Cs", "PP", "PACMA"},
    {174353, 138051, 113610, 251743, 5032}}
```

Es decir el total de partidos es 5 y el total de escaños es 13.

Puedes usar aquí los subalgoritmos `perteneceA` y `anadePartido` anteriores.

ordenaPartidosPorVotos

Dada una estructura de tipo `circunscripcionUnica` ordena sus partidos y sus votos en orden decreciente de número de votos

Así, para la estructura

```
{5, 13, {"PSOE", "Podemos","Cs", "PP", "PACMA"},
    {174353, 138051, 113610, 251743, 5032}}
```

el resultado de la ordenación es

```
{5, 13, {"PP", "PSOE", "Podemos", "Cs", "PACMA"},
    {251743, 174353, 138051, 113610, 5032}}
```

calculaEscanosCircunscripcionPromedio.

calculaEscanosCircunscripcionRestoMayor.

Los has usado en la tarea anterior y ahora tienes que usarlos en una versión ligeramente modificada cuya entrada incluye una estructura de tipo `circunscripcionUnica`. La nueva versión de estos subalgoritmos la tienes disponible en Moodle

guardaResultado

Dado un fichero previamente abierto, el nombre de un método de reparto de escaños, una estructura de tipo `circunscripcionUnica` y el reparto de escaños resultado de aplicar el método a esa estructura, guarda en el fichero el reparto de escaños.

Por ejemplo, para el fichero `datos.txt`, las líneas guardadas en el fichero resultado para el método d'Hont son:

```
D'Hont
PP PSOE Podemos Cs   PACMA
5   3      3    2     0
```

5.6.4. Ejemplos de ejecución

Para el fichero de texto `datos.txt`, el resultado de la ejecución del programa que se guarda en el fichero de salida es:

```
D'Hont
PP PSOE Podemos Cs   PACMA
5   3      3    2     0
Hare
PP PSOE Podemos Cs   PACMA
5   3      3    2     0
```

Para el fichero de texto `resultadosElectorales.txt`, el resultado de la ejecución del programa que se guarda en el fichero de salida es:

```
D'Hont
PP PSOE PODEMOS C's ECP ERC-CATSÍ CDC EAJ-PNV EH-Bildu CCa-PNC
126  87    63    38  13   10       7     4      2      0
Hare
PP PSOE PODEMOS C's ECP ERC-CATSÍ CDC EAJ-PNV EH-Bildu CCa-PNC
125  86    62    38  13   10       8     5      2      1
```

5.6.5. Ayudas

Ficheros auxiliares

En Moodle encontraréis los siguientes ficheros

1. El fichero `datos.txt`.
2. El fichero `resultadosElectorales.txt` que contiene los resultados de las elecciones de junio de 2016 en España
3. El fichero `reparto.cpp` que contiene, entre otros, una versión de los subalgoritmos
   - `calculaEscanosCircunscripcionPromedio`
   - `calculaEscanosCircunscripcionRestoMayor`

   cuya entrada es una estructura de tipo `circunscripcionUnica`.

5.6.6. Comprobación de la validez de los subalgoritmos

Un subalgoritmo no está terminado cuando hemos escrito su código sino que debemos comprobar, con diferentes juegos de prueba, si el subalgoritmo funciona correctamente.

Para ayudarte en esta tarea cuando escribas subalgoritmos como `anadePartido`, `perteneceA` y `extraePartidosyVotos`, hemos incluido en el fichero `reparto.cpp` el subalgoritmo `muestraCircunscripcionUnica`. Este subalgoritmo muestra el contenido completo de una estructura de tipo `circunscripcionUnica`.

Así, por ejemplo, cuando hayas programado `extraePartidosyVotos`, puedes probar si funciona correctamente del siguiente modo. Ejecutas el subalgoritmo `extraePartidosyVotos` tomando como fichero de entrada el fichero `datos.txt`. Muestras con el subalgoritmo `muestraCircunscripcionUnica` el contenido de la estructura de tipo `circunscripcionUnica` obtenida. Si el resultado coincide con el siguiente

```
{5, 13, {"PSOE", "Podemos","Cs", "PP", "PACMA"},
    {174353, 138051, 113610, 251743, 5032}}
```

estamos en el buen camino aunque no podemos garantizar que nuestro subalgoritmo es correcto. Te recomendamos que te construyas nuevos juegos de pruebas y compruebes con ellos si tu subalgoritmo `extraePartidosyVotos` funciona correctamente. Repite este proceso de verificación con todos tus subalgoritmos.

5.7. Proyecto 2C: Distancias en un sistema planetario (ii)

Con frecuencia, los datos, tanto de entrada como de salida, que un programa procesa se almacenan en estructuras de información persistentes. Esto permite disponer de los datos en nuevas ejecuciones del mismo programa, sin pedir al usuario que los vuelve a introducir, y también los hace accesibles a otros programas o herramientas (interoperabilidad).

Utilizando las funciones que hemos desarrollado en la tarea 4.7, en ésta generaremos y almacenaremos en un fichero las posiciones que ocupan los planetas de un sistema en un gran número de instantes de tiempo. Podríamos utilizar otras herramientas para, a partir de esta información, obtener gráficas con las órbitas de los planetas o la distancia relativa entre ellos.

5.7.1. Descripción de la tarea

En el fichero binario "planetas.dat" se ha almacenado información de cuatro planetas (Tierra, Mercurio, Venus y Marte, en este orden). Concretamente, se han almacenado utilizando la estructura `struct planeta` que ya usamos en la tarea 4.7 y recordamos a continuación.

```
struct elipse {
    double a, e, phi0;
};
struct fecha {
   int year, dias;
};
struct sistema {
    struct planeta objeto[MAXPLA];
    int nobjetos;
};
```

```
struct planeta {
    char nombre[100];
    struct elipse orbita;
    double periodo, theta0;
    struct fecha t0;
};
```

Diseña y escribe, en el lenguaje de programación C, un programa que genere y almacene en un fichero de texto las coordenadas cartesianas de la posición que ocupa cada uno de los planetas de un sistema en cada uno de los días desde el 01/01/2017 al 31/12/2027. Los datos de los planetas del sistema se leerán de un fichero binario, como "planetas.dat", en el que se han almacenado los datos de un número desconocido de planetas, pero que es menor que 20.

El programa comenzará solicitando al usuario que introduzca los nombres de los ficheros de entrada y de salida. Si alguno de ellos no puede abrise, el programa terminará inmediatamente después de mostrar un mensaje de error en la pantalla que incluirá el nombre del fichero que causó el problema. Si los dos ficheros pueden abrirse, el programa almacenará en una variable del tipo `struct sistema` los datos de los planetas. A continuación escribirá en cada línea del fichero de texto la siguiente información:

```
dd mm aaaa   posx1 posy1   posx2 posy2 ... posxn posyn
```

donde `dd`, `mm` y `aaaa` es la fecha (día, mes y año) y `posxi`, `posyi` son las coordenadas de la posición que ocupa el i-ésimo planeta del sistema en esa fecha (recuerda que en nuestro sistema planetario simplificado las órbitas de todos los planetas se encuentran en

el mismo plano). Estas coordenadas deben escribirse con una precisión de tres decimales y con un ancho de campo de cinco posiciones, incluidas las posiciones necesarias para el punto decimal y el signo, si es necesario. En la siguiente sección se muestran las primeras y últimas líneas del fichero de salida. El programa también mostrará en pantalla el nombre de los planetas que han sido procesados.

Además de las estructuras de datos, en este programa puedes utilizar los subalgoritmos `fechaEstandar`, `dinamica` y `coordenadas` de la tarea 4.7. Cópialos en tu código, junto con las funciones que éstas utilicen, antes de comenzar el desarrollo del programa.

5.7.2. Ejemplos de ejecución

```
Introduce el nombre del fichero de entrada: miplaneta.dat
No se ha podido abrir el fichero 'miplaneta.dat'
```

```
Introduce el nombre del fichero de entrada: planetas.dat
Introduce el nombre del fichero de salida: salida.txt

Planetas identificados:
     Tierra
   Mercurio
      Venus
      Marte

Proceso terminado...
Los datos estan disponibles en el fichero 'salida.txt'
```

```
01 01 2017   -0.188  0.983    0.110 -0.316   -0.509 -0.520    1.336 -0.892
02 01 2017   -0.205  0.979    0.138 -0.300   -0.495 -0.534    1.344 -0.881
03 01 2017   -0.221  0.975    0.164 -0.281   -0.480 -0.548    1.352 -0.871
.........
29 12 2027   -0.123  0.994   -0.462  0.061   -0.726  0.036    0.033 -1.474
30 12 2027   -0.140  0.991   -0.464  0.039   -0.727  0.016    0.048 -1.475
31 12 2027   -0.157  0.988   -0.464  0.016   -0.727 -0.004    0.062 -1.475
```

5.7.3. *Octave:* procesando ficheros de texto

Podemos utilizar la herramienta *Octave* para generar fácilmente gráficas a partir de datos numéricos que se hayan almacenado en forma de tabla en un fichero, como el que produce el programa propuesto en esta tarea. Para ello, el primer paso es crear una matriz con la información del fichero. Supongamos que el fichero se llama "datos.txt"; entonces basta escribir el comando

```
datos = load("datos.txt");
```

para obtener la matriz `datos` cuyas filas y columnas contienen los datos del fichero. Recordad que en *Octave* el índice de la primera fila y de la primera columna de una matriz es el entero 1.

Puede ser útil obtener submatrices a partir de ésta. Para ello podemos utilizar *rangos* u otras matrices para indicar las colecciones de índices que queremos seleccionar. Un rango

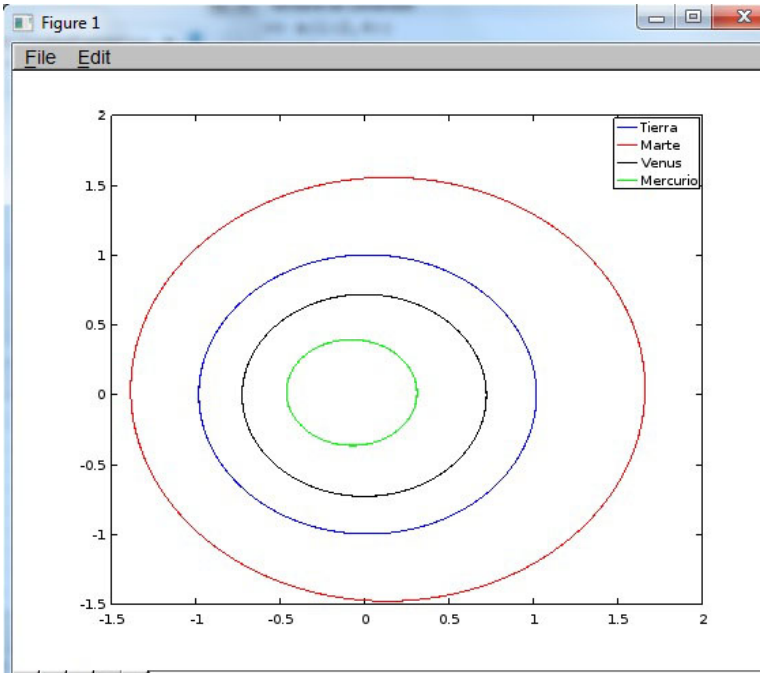


Figura 5.7: Gráfico de Octave

se especifica indicando dos valores enteros separados por el símbolo ':'. Por ejemplo, `3:6` es el rango formado por los enteros $3, 4, 5, 6$. Entonces, el comando

```
B = A(1:2,3:6);
```

crea la matriz `B` con las dos primeras filas de `A` y, de ellas, sólo las columnas 3 a 6. Si omitimos el índice inicial y final en el rango, `A(1:2,:)`, entonces seleccionamos todas las columnas (o filas). Finalmente, si necesitamos seleccionar índices no consecutivos (por ejemplo, las columnas 4 y 6, pero no la 5) utilizaríamos una matriz de índices (`B = A(1:2,[4 6])`).

En nuestro caso estamos interesados en seleccionar las posiciones de cada planeta. Puesto que las tres primeras columnas representan la fecha, bastará ejecutar el comando

```
tierra = datos(:,4:5);
```

para seleccionar la posición de la Tierra en todos los días considerados.

Para obtener la gráfica de una función usaremos el comando `plot`. En su forma más sencilla podemos escribir `plot(x, y)`, donde `x` e `y` son vectores unidimensionales, esto es, formados por una sola fila o una sola columna, de la misma longitud. La gráfica que se obtendrá mostrará las ordenadas `y` frente a las abcisas `x`. Si se omite el vector de abcisas se tomará como tal el rango de los enteros desde 1 hasta la longitud de `y`. Por ejemplo,

```
plot(tierra(:,1), tierra(:,2));
```

mostrará en una nueva ventana una gráfica con la órbita de la Tierra.

Si se desea mostrar la gráfica de varias funciones se puede usar la siguiente sintaxis

```
plot(x1, y1, x2, y2, ... xn, yn);
```

Y, si además, deseamos que cada curva aparezca con un color distinto, detrás del vector de abcisas de cada función se puede indicar, como un par de parámetros adicional, la palabra

"color" y una letra que seleccione el color deseado. Entre los posibles disponemos de: "k"–black, "g"–green, "y"–yellow, "b"–blue, "r"–red. Finalmente, con el comando `legend` se puede añadir una leyenda al gráfico que indique a qué función corresponde cada color.

Cada vez que ejecutamos un comando `plot` se elimina el gráfico anterior. Para añadir una nueva función a un gráfico debemos ejecutar el comando `hold on` antes de ejecutar un nuevo `plot`

Suponiendo que en las matrices `tierra`, `marte`, `mercurio` y `venus` disponemos de las posiciones de los correspondientes planetas, la secuencia de comandos

```
plot(tierra(:,1), tierra(:,2), "color", "b"); hold on;
plot(marte(:,1), marte(:,2), "color", "r"); hold on;
plot(venus(:,1), venus(:,2), "color", "k"); hold on;
plot(mercurio(:,1), mercurio(:,2), "color", "g"); hold on;
legend("Tierra", "Marte", "Venus", "Mercurio");
```

producirán el gráfico que se muestra en la figura 5.7.

¿Podrías generar, únicamente utilizando *Octave*, un gráfico que mostrara la distancia al Sol de los cuatro planetas entre los días 01/01/2017 y 31/12/2018? Recuerda que el Sol está en el origen de coordenadas.

5.8. Proyecto ST: Servicio de transporte (revisitado)

En esta tarea proponemos una versión más realista del problema abordado en la tarea 4.8. Al inicio de la jornada laboral la camioneta parte de su base - el punto (0,0)-, pero una vez finalizado un reparto pasa directamente al siguiente sin tener que regresar a la base. Esto es: para el primer reparto, el punto de partida es la base; para todos los demás, el punto de destino del reparto anterior; el punto final de cada reparto es su punto de destino.

Seguimos queriendo encontrar la organización de repartos que minimice el tiempo medio de entrega (equivalentemente, la distancia media por reparto). A diferencia de la tarea 4.8, el problema se ha vuelto extraordinariamente difícil: no tenemos el equivalente al teorema 4.8.3 que nos indique cuál es la solución óptima.

Optaremos por una solución de compromiso: en esta tarea desarrollarás un algoritmo análogo al de la tarea 4.8: seguiremos un proceso iterativo donde en cada paso escogeremos, de entre todos los repartos que aún no hayan sido seleccionados, el más corto. Para evaluar la bondad del resultado, te facilitaremos un subalgoritmo que da la solución óptima: void optimiza (Jornada &).

5.8.1. Introducción

El problema

Redefinimos tiempo medio de servicio y tiempo medio de entrega:

Definición 5.8.1. Tiempo de servicio Dado un servicio s y un punto de partida P, definimos su tiempo de servicio como la suma:

$$\mathrm{t_s} = \mathrm{t_{s;ida}} + \mathrm{t_{s;transporte}} \tag{5.4}$$

donde $\mathrm{t_{s;ida}}$ es el tiempo desde P hasta el punto de recogida, y $\mathrm{t_{s;transporte}}$ es el tiempo desde el punto de recogida hasta el de entrega.

Definición 5.8.2. Dada una secuencia de n servicios $\{s_i\}$, el **tiempo medio de entrega** es

$$\frac{\sum_{\mathrm{i=0}}^{\mathrm{i<n}}(\mathrm{n}-\mathrm{i})\mathrm{t_{s_i}}}{\mathrm{n}} \tag{5.5}$$

El enunciado de nuestro problema sigue siendo: **dadas una serie de tareas, queremos planificarlas de modo que el tiempo medio de entrega sea el menor posible**.

Algoritmos voraces

Al igual que en la tarea 4.8, nos enfrentamos al siguiente problema: una vez realizado un servicio, ¿cuál debería ser el siguiente? En aquélla seguimos una táctica cortoplacista: de todos los servicios pendientes, escogimos el más prometedor, el que requería menos tiempo: el que nos daba mayor beneficio inmediato. Este tipo de algoritmos se denomina *voraz* (*greedy*).

Pero ahora no podemos garantizar que esta táctica nos permita organizar los repartos de modo que el tiempo medio de espera sea el menor posible. Quizá habría sido mejor considerar otras alternativas (perder un poquito más de tiempo en el siguiente reparto, si

con ello pudiéramos recuperarlo al cabo de unos cuantos servicios). ¿Cuál es la solución óptima? Para estar seguros, deberíamos considerar todas las posibles formas de realizar los servicios. A modo de ejemplo, existen 10! = 3628800 formas distintas de ordenar 10 servicios. Con apenas 20 servicios tendríamos más de dos trillones de permutaciones posibles. Imaginemos la complejidad de gestionar una red de plataformas logísticas con millones de envíos diarios. En este punto, el problema deja de ser meramente teórico: además del desafío intelectual que supone, sus implicaciones económicas pasan a ser de primer orden.

Optaremos por una solución de compromiso. Seguiremos siendo cortoplacistas: **en cada momento escogeremos, de los repartos pendientes, el que tenga un tiempo de entrega mínimo**. No tenemos la certeza de encontrar una solución óptima, pero confiamos en que la hallada será razonablemente aceptable.

5.8.2. Tarea

Para representar los distintos servicios de una jornada usaremos las siguientes estructuras:

```
# define MAXSERV 100

struct punto {
  double x, y;
};
typedef struct punto Punto;
// Punto en R^2.

struct reparto{
  char identificador;
  Punto origen, destino;
};
typedef struct reparto Reparto;

struct jornada{
    int nServicios;
    Reparto servicio[MAXSERV];
};
typedef struct jornada Jornada;
```

Diseña y programa en C/C++ los siguientes subalgoritmos:

leeRepartos

Dado el nombre de un fichero binario que contiene una serie de registros de tipo *Reparto*, devuelve un booleano y un registro de tipo *Jornada*. Se sabe que el fichero no contiene más de MAXSERV repartos. Si se ha conseguido abrir con éxito el fichero, el booleano vale true, y el registro contiene toda la información almacenada en el fichero. Si no, el booleano vale *false* y el valor del registro es irrelevante.

distanciaReparto

Dados un punto de partida en R^2 y un registro de tipo Reparto, calcula la distancia taxi requerida para realizarlo, empezando en el punto de partida y finalizando en el punto de destino: la correspondiente al trayecto punto de partida - origen - destino.

distanciaMediaJornada

Dado un registro de tipo *Jornada*, calcula su distancia media de servicio:

$$\frac{\sum_{\mathrm{i}=0}^{\mathrm{i}<\mathrm{n}}(\mathrm{n}-\mathrm{i})\mathrm{d}_{\mathrm{s_i}}}{\mathrm{n}} \tag{5.6}$$

donde d_{s_i} es la distancia de reparto asociada al servicio s_i según el subalgoritmo **distanciaReparto**. Para el primer servicio, el punto de partida es (0,0); para los demás, el punto de partida es el de destino del servicio anterior.

ordena

Dado un registro de tipo *Jornada*, ordena sus servicios de forma creciente según sus distancias de reparto correspondientes. El primer servicio es aquél cuya distancia de reparto, partiendo de (0,0), es mínima. Una vez ordenados los primeros i servicios, el servicio (i+1)-ésimo se escoge de entre los $nServicios - i$ restantes: aquél cuya distancia de reparto, partiendo del punto de destino del servicio i-ésimo, es mínima.

guardaJornada

Dados un registro de tipo *Jornada*, y un fichero de texto[1], guarda en éste el listado de repartos: (identificador, origen, destino y distancia de reparto), así como la distancia media de servicio.

programa principal

Solicita al usuario los nombres de un fichero binario que almacena los repartos de una jornada y de un fichero de texto donde guardar los resultados del programa (la información sobre la jornada: los servicios ordenados y la distancia media). En caso de poder abrir ambos ficheros con éxito, lee los repartos y guarda en el fichero de texto:

- La jornada, tal como se ha leido del fichero binario.
- La jornada resultante de ordenar sus repartos con el subalgoritmo **ordena**.
- La jornada resultante de ordenar los repartos con el subalgoritmo void optimiza (Jornada &), incluido en el fichero *planificaJornada.cpp*, que calcula la planificación óptima.

En caso de no poder abrir alguno de los ficheros, el programa muestra un mensaje de error y termina.

[1] Por *fichero* entendemos *un puntero a una estructura de tipo FILE*: un objeto *f* de tipo *FILE **

5.8.3. Ayudas

Para comprobar tus subalgoritmos, descarga los ficheros Jornada.dat y planificaJornada.cpp. Jornada.dat incluye la información relativa a una jornada de reparto. El fichero planificaJornada.cpp contiene el subalgoritmo void optimiza (Jornada &), que calcula la planificación óptima de la jornada. Con los datos del fichero Jornada.dat, el contenido del fichero de salida es:

```
Jornada leida:
a: origen = (3, 2)  destino =  (-3, 2)   Distancia = 11
b: origen = (3, 2)  destino =  (3, -2)   Distancia = 10
c: origen = (41, 5)  destino =  (-77, -1600)   Distancia = 1768
d: origen = (-81, -29)  destino =  (-68, -75)   Distancia = 1634
e: origen = (-77, 72)  destino =  (67, 48)   Distancia = 324
f: origen = (31, -69)  destino =  (-94, -11)   Distancia = 336
g: origen = (-640, -47)  destino =  (78, 15)   Distancia = 1362
h: origen = (56, -75)  destino =  (-4, 70)   Distancia = 317
i: origen = (-83, 95)  destino =  (-56, -70)   Distancia = 296
j: origen = (14, 6)  destino =  (87, -7)   Distancia = 232
Distancia media: 3662.9

Tras ordenar:
b: origen = (3, 2)  destino =  (3, -2)   Distancia = 9
a: origen = (3, 2)  destino =  (-3, 2)   Distancia = 10
j: origen = (14, 6)  destino =  (87, -7)   Distancia = 107
d: origen = (-81, -29)  destino =  (-68, -75)   Distancia = 249
f: origen = (31, -69)  destino =  (-94, -11)   Distancia = 288
e: origen = (-77, 72)  destino =  (67, 48)   Distancia = 268
h: origen = (56, -75)  destino =  (-4, 70)   Distancia = 339
i: origen = (-83, 95)  destino =  (-56, -70)   Distancia = 296
g: origen = (-640, -47)  destino =  (78, 15)   Distancia = 1387
c: origen = (41, 5)  destino =  (-77, -1600)   Distancia = 1770
Distancia media: 1263.5

Tras optimizar:
b: origen = (3, 2)  destino =  (3, -2)   Distancia = 9
a: origen = (3, 2)  destino =  (-3, 2)   Distancia = 10
j: origen = (14, 6)  destino =  (87, -7)   Distancia = 107
f: origen = (31, -69)  destino =  (-94, -11)   Distancia = 301
d: origen = (-81, -29)  destino =  (-68, -75)   Distancia = 90
h: origen = (56, -75)  destino =  (-4, 70)   Distancia = 329
e: origen = (-77, 72)  destino =  (67, 48)   Distancia = 243
i: origen = (-83, 95)  destino =  (-56, -70)   Distancia = 389
g: origen = (-640, -47)  destino =  (78, 15)   Distancia = 1387
c: origen = (41, 5)  destino =  (-77, -1600)   Distancia = 1770
Distancia media: 1201.1
```

Observa que, en este caso, el algoritmo voraz que has programado rebaja la distancia media a aproximadamente la tercera parte de la inicial. Esta distancia resulta ser un 5 % mayor de la óptima.

5.9. El péndulo simple: solución numérica (ii)

En esta tarea estudiaremos la dinámica del péndulo, incluyendo términos que modelizan la fricción y una fuerza externa periódica:

$$\ddot{\theta} = -\alpha\,\text{seno}(\theta) \;-\; \beta\,\dot{\theta} \;+\; \Gamma\cos(\omega_D t), \tag{5.7}$$

Nos interesa el comportamiento estacionario del péndulo, una vez terminada la fase transitoria. En particular, prestaremos especial atención a la periodicidad de las soluciones. Vimos que el oscilador armónico, en el caso forzado, acababa oscilando a la misma frecuencia que la fuerza externa; en esta tarea compararemos el periodo del péndulo con el de dicha fuerza.

5.9.1. Introducción

Sección de Poincaré

En esta tarea entenderemos la sección de Poincaré como el conjunto de puntos de la órbita del péndulo en el espacio de fases tomados a intervalos regulares T:

$$\left\{ \Big(\theta(t_{stat}+nT), \dot{\theta}(t_{stat}+nT) \Big) \right\}$$

donde n es un entero, T un tiempo característico y t_{stat} un tiempo lo suficientemente grande como para garantizar que el péndulo ha alcanzado el régimen estacionario (si $\Gamma \neq 0$ o $\beta = 0$) o ha llegado al reposo (si $\Gamma = 0$ y $\beta \neq 0$). Es como si observáramos el péndulo con una luz estroboscópica que lanza un destello cada T segundos.

Si la órbita es periódica y T es su periodo, la sección de Poincaré consta de un único punto (cada vez que miramos vemos al péndulo en la misma posición, y con la misma velocidad). En general, para una órbita de periodo nT la sección de Poincaré consta de n puntos. Si la órbita no es periódica, o si el cociente entre su periodo y T no es un número racional, la sección de Poincaré consta de infinitos puntos.

En esta tarea, en ausencia de término forzante ($\Gamma = 0$) tomaremos como T el periodo natural de oscilación del péndulo sin fricción, ya calculado en la tarea 3.7. La sección de Poincaré consta de un único punto. En efecto, si $\beta = 0$, el péndulo oscila con periodo T: si lo observamos cada T segundos lo vemos siempre en el mismo estado; si $\beta \neq 0$, el péndulo ha alcanzado el reposo, y permanece para siempre en ese estado.

Si $\Gamma \neq 0$, T será el periodo de la fuerza externa. Es posible demostrar que si la sección de Poincaré consta de n puntos, el periodo del péndulo es nT [2].

El fichero `dinamicaPendulo.cpp`

Descarga el fichero dinamicaPendulo.cpp, que contiene las declaraciones de los siguientes tipos de datos:

```
struct parametros{
    double alpha, beta, Gamma, omega;
};
```

[2] Estrictamente hablando, el periodo del péndulo es $\frac{n}{m}T$, donde m es un entero coprimo con n. Comprobaremos visualmente que $m = 1$.

```
typedef struct parametros Parametros;
//Parámetros del péndulo

struct estado {
    double theta, w;
};
typedef struct estado Estado;
//El estado del péndulo queda definido
//por los valores del ángulo y la velocidad angular.
//Suponemos que vienen expresados,respectivamente, en radianes y radianes/segundo

struct datosSimulacion{
    double factorDeltaT, factorTCout, factorTStat, factorTSimul;
    Estado estadoInicial;
    Parametros param;
};
typedef struct datosSimulacion DatosSimulacion;
//Una simulación se define por su estado inicial (ángulo y velocidad),
//sus parámetros y los factores temporales.
```

El fichero contiene igualmente tres subalgoritmos ya desarrollados (los dos primeros quedaron resueltos en la tarea 3.7):

- `double normaliza (double theta);` que, dado un ángulo, devuelve otro equivalente módulo 2π en el intervalo $[-\pi, \pi)$.
- `int analisis (double alpha, double theta, double w, double epsilon, double & thetaM, double &T);` que, dados el coeficiente alpha, un ángulo, una velocidad angular y un valor para la precisión (epsilon), devuelve:
  - Un entero que caracteriza el tipo de solución.
  - Por referencia: el ángulo límite y un T característico (en caso de movimiento periódico, el periodo).
- `Estado Runge_Kutta (Parametros param, Estado e, double t, double DeltaT);` que, dados un conjunto de parámetros, el estado del péndulo en el instante t, dos reales(el propio t y DeltaT), devuelve el estado del péndulo en el instante t + DeltaT.

5.9.2. Descripción de la tarea

Completa el fichero `dinamicaPendulo.cpp` añadiendo los subalgoritmos y el programa principal descritos a continuación:

distancia

Dados dos estados, $e_A = (\theta_A, w_A)$ y $e_B = (\theta_B, w_B)$, devuelve la distancia euclídea entre ellos medida en el espacio de fases:

$$\sqrt{(\theta_A - \theta_B)^2 + (w_A - w_B)^2} \tag{5.8}$$

donde la diferencia de ángulos $(\theta_A - \theta_B)$ debe estar normalizada en el intervalo $[-\pi, \pi)$.

aceleracion

Dados los parámetros del péndulo, su estado en un instante t, $(\theta(t), w(t))$, y el tiempo t, devuelve la aceleración angular (miembro derecho de 5.7).

Para que pueda ser invocado por la función `Runge_Kutta`, el algoritmo debe tener la siguiente cabecera:

```
double aceleracion (Parametros param,Estado e, double t)
```

evolucion

Dados dos reales t_0 y Δt, un entero n, los parámetros del péndulo y el estado del mismo en el instante t_0, devuelve un real, $t = t_0 + n\Delta t$, y el estado del péndulo en t.

Idea: itera el método de Runge-Kutta entre t_0 y $t_0 + n\Delta t$ con intervalos Δt.

Programa principal

Escribe un programa que empiece creando un fichero de texto, `resumen.txt`.

A continuación, pedirá al usuario el nombre de un fichero binario donde están almacenados una serie de ejemplares de `DatosSimulacion`. Los valores guardados en este fichero han sido procesadas de modo que las relaciones entre ellas vienen dadas por números enteros:

- $factor_{simul} - factor_{stat}$ es un número entero, n_T.
- Existen enteros n_{stat}, n_{out}, n_Δ tales que:
  - $n_{stat} = factor_{stat}/factor_\Delta$,
  - $n_{cout} = 1/factor_{cout}$ y
  - $n_\Delta = factor_{cout}/factor_\Delta$

No es necesario que compruebes que las relaciones anteriores se cumplen: puedes darlo por supuesto.

Para cada uno de los ejemplares de `DatosSimulacion` almacenados en el fichero binario, el algoritmo deberá seguir los pasos que se indican a continuación.

Al leer cada ejemplar se le asignará como identificador un número entero correlativo (empezando en 0), `id`, y se calculará T del siguiente modo:

- En ausencia de término forzante ($\Gamma = 0$), T es el periodo del péndulo sin fricción (calculado por lafunción **analisis**, desarrollado en la tarea 3.7 y **ya incluida** en el fichero `dinamicaPendulo.cpp`; sugerimos tomar una precisión $\epsilon = 10^{-7}$).
- Si $\Gamma \neq 0$, T es el periodo del término forzante ($T = 2\pi/\omega_D$).

A partir de T se calcularán: $\Delta t = factor_{deltaT} * T$, $t_{stat} = factor_{stat} * T$, $t_{cOut} = factor_{cout} * T$ y $t_{simul} = factor_{simul} * T$.

Se mostrarán por pantalla: el identificador, los factores temporales: (factor_Δ, $\text{factor}_{\text{cout}}$, $\text{factor}_{\text{stat}}$, $\text{factor}_{\text{tSimul}}$), los parámetros $(\alpha, \beta, \Gamma, w_D)$, el estado inicial (θ_0, w_0) y el valor de T.

Se crearán los ficheros `outEstados_<id>.txt` y `outPoincare_<id>.txt` donde `<id>` es el identificador de los datos de la simulación, expresado con dos dígitos (ver la Sección **Ayudas**).

Se simulará la dinámica del péndulo desde $t = 0$ hasta $t = t_{stat}$. Denominaremos e_{stat} al estado del péndulo en $t = t_{stat}$.

A partir de t_{stat}, a intervalos regulares se añadirá a los dos ficheros una línea con los valores del tiempo t y el estado $(\theta, \dot{\theta})$. Se añadirá una línea a `outEstados_<id>.txt` cada t_{cout} segundos, y a `outPoincare_<id>.txt` cada T segundos. En ambos ficheros el valor del ángulo θ se dará en el intervalo $[-\pi, \pi)$. Observa que las filas de `outPoincare_<id>.txt` son un subconjunto de las de `outEstados_<id>.txt`: en el primero se escribe una por cada n_{cout} del segundo.

Cada T segundos se comparará el estado del péndulo con el que tenía en t_{stat}. El proceso finalizará cuando el programa detecte que el péndulo ha vuelto al estado e_{stat} ($e_t = e_{stat}$, con $t = t_{stat} + nT$ siendo n un entero positivo), o bien al llegar a t_{simul}. Supondremos que $e_t = e_{stat}$ si la distancia entre ambos estados en el espacio de fases es menor que 10^{-5} (usa el subalgoritmo **distancia**).

Entonces se añade al fichero `resumen.txt` una línea, mostrando:

- el identificador `<id>`.
- los siguientes datos de la simulación: los parámetros $(\alpha, \beta, \Gamma, w_D)$ y el estado inicial (θ_0, w_0).
- el número de puntos distintos de la sección de Poincaré para esos datos: el número de intervalos de T segundos transcurridos desde t_{stat} hasta el final de la simulación (equivalentemente, el número de filas del fichero `outPoincare_<id>.txt`).
- la distancia entre e_t y e_{stat}

A continuación se repite el proceso para el siguiente ejemplar de `DatosSimulacion`.

Si no se consiguen abrir el fichero `resumen.txt` o el que contiene los ejemplares de `DatosSimulacion`, el programa mostrará un mensaje de error por pantalla y finalizará. Si para algún ejemplar de `DatosSimulacion` no se consigue abrir alguno de los ficheros `outEstados_<id>.txt` o `outPoincare_<id>.txt`, el programa pasará a analizar el siguiente ejemplar. Todos los ficheros que se abran deben cerrarse antes de finalizar el programa.

5.9.3. Ayudas

Para construir los nombres de los ficheros `outEstados_<id>.txt` puedes usar la función `sprintf`, declarada en el fichero de cabecera `string.h`. Por ejemplo, dado un array de caracteres `nombre` al ejecutar

```
sprintf(nombre, "outEstados_%02d.txt", 3)
```

a `nombre` se le asigna el valor `"outEstados_03.txt"`.

5.9.4. Ejemplos de ejecución

Para el fichero `datosEntrada_PenduloCompleto.dat`, el programa muestra por pantalla:

```
Introduzca el fichero de entrada: datosTarea04.dat

Simulacion 00:
        Estado inicial: (0.0001, 0.003)          Parametros: (39.4784, 0, 0, 0)
        Factores temporales (0.0001 0.02 0 10)   T = 1 s
        Abriendo ficheros outEstados_00.txt y outPoincare_00.txt

Simulacion 01:
        Estado inicial: (2, 6.77457)     Parametros: (39.4784, 0, 0, 0)
        Factores temporales (0.0001 0.02 0 10)   T = 3 s
        Abriendo ficheros outEstados_01.txt y outPoincare_01.txt

Simulacion 02:
        Estado inicial: (0, 0)   Parametros: (1, 0.5, 0.9, 0.666667)
        Factores temporales (0.0001 0.002 5000 25000)    T = 9.42478 s
        Abriendo ficheros outEstados_02.txt y outPoincare_02.txt

Simulacion 03:
        Estado inicial: (0, 0)   Parametros: (1, 0.5, 1.07, 0.666667)
        Factores temporales (0.0001 0.002 5000 25000)    T = 9.42478 s
        Abriendo ficheros outEstados_03.txt y outPoincare_03.txt

Simulacion 04:
        Estado inicial: (0, 0)   Parametros: (1, 0.5, 1.19, 0.666667)
        Factores temporales (0.0001 0.1 5000 10000)      T = 9.42478 s
        Abriendo ficheros outEstados_04.txt y outPoincare_04.txt

Simulacion 05:
        Estado inicial: (0, 0)   Parametros: (1, 0.5, 1.3, 0.666667)
        Factores temporales (0.0001 0.002 5000 25000)    T = 9.42478 s
        Abriendo ficheros outEstados_05.txt y outPoincare_05.txt

Simulacion 06:
        Estado inicial: (0, 0)   Parametros: (1, 0.5, 1.46, 0.666667)
        Factores temporales (0.0001 0.002 5000 25000)    T = 9.42478 s
        Abriendo ficheros outEstados_06.txt y outPoincare_06.txt
```

El contenido del fichero `resumen.txt`, es:

id	*alpha*	*beta*	*Gamma*	*omega*	*theta*0	*w*0	*n*	*Distancia*
0	39,4784	0	0	0	0,0001	0,003	1	$1{,}91166e-017$
1	39,4784	0	0	0	2	6,77457	1	$4{,}04769e-010$
2	1	0,5	0,9	0,666667	0	0	1	$5{,}29407e-009$
3	1	0,5	1,07	0,666667	0	0	2	$1{,}03359e-008$
4	1	0,5	1,19	0,666667	0	0	5000	2,41195
5	1	0,5	1,3	0,666667	0	0	1	$1{,}64305e-009$
6	1	0,5	1,46	0,666667	0	0	4	$1{,}13972e-008$

Los contenidos de los ficheros `outPoincare_00.txt` y `outPoincare_01.txt` son, respectivamente

```
t   theta   w
1   0.0001   0.003
```

y

```
t   theta   w
3   2   6.77457
```

Los contenidos de los ficheros `outEstados_00.txt` y `outEstados_01.txt` son, respectivamente:

```
t   theta   w
0   0.0001   0.003
0.02   0.000159054   0.00289759
0.04   0.000215599   0.00274949
0.06   0.000268744   0.00255803
0.08   0.000317651   0.00232623
0.1   0.000361548   0.00205773
 . . .
1   0.0001   0.003
```

```
t   theta   w
0   2   6.77457
0.06   2.34627   4.84546
0.12   2.59095   3.38619
0.18   2.76079   2.33487
0.24   2.87731   1.59278
0.3   2.95631   1.07097
 . . .
3   2   6.77457
```

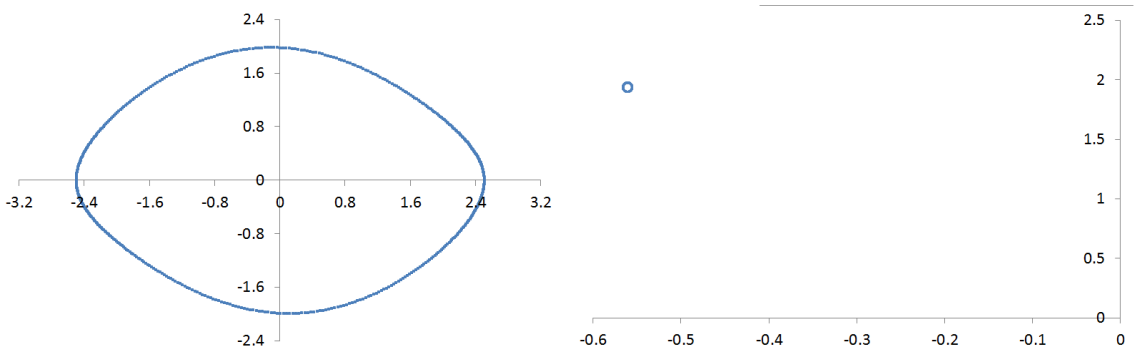
Los dos primeros ejemplares corresponden al caso, ya estudiado, de un péndulo sin fricción ni aporte de energía, con $\alpha = 4\pi^2$. Tal como era de esperar, $n = 1$: la dinámica es periódica con periodo T.

Analicemos los casos 2 al 6, estos ya con fricción y fuerza externa periódica.

En todos ellos, $\theta_0 = 0$ rad , $\dot{\theta}_0 = 0$ rad/s, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega_D = 2/3s^{-1}$: sólo cambia el valor de Γ. El contenido de los ficheros de salida se muestra gráficamente en las Figuras 5.8 a 5.12. Para los valores $\Gamma = 0.9, 1.07, 1.3, 1.46$ (expresado en s^{-2}) observamos

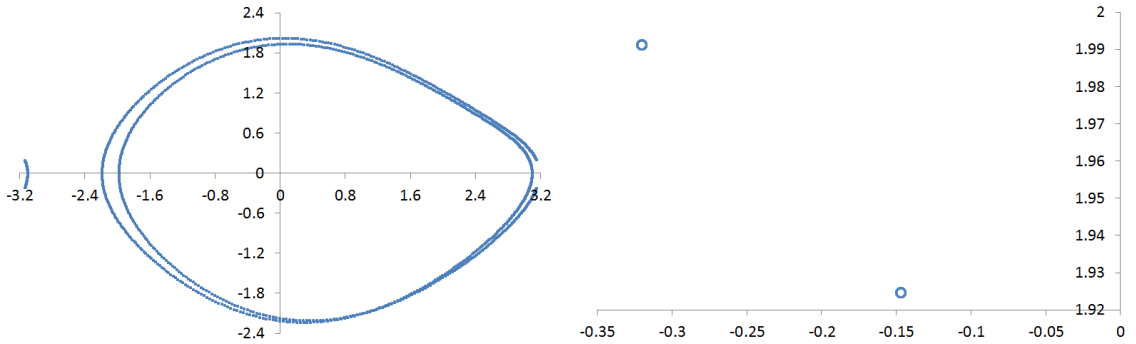
dinámicas periódicas, con periodos iguales a T, $2T$, T (de nuevo) y $4T$, donde $T = 2\pi/\omega_D$ es el periodo del término forzante. La periodicidad se aprecia claramente en las secciones de Poincaré (con uno, dos, uno y cuatro puntos, respectivamente). Los atractores (las órbitas a las que acaba convergiendo la dinámica del péndulo) son ciclos límite.

Para $\Gamma = 1{,}19s^{-2}$ la dinámica es caótica, con un atractor extraño.



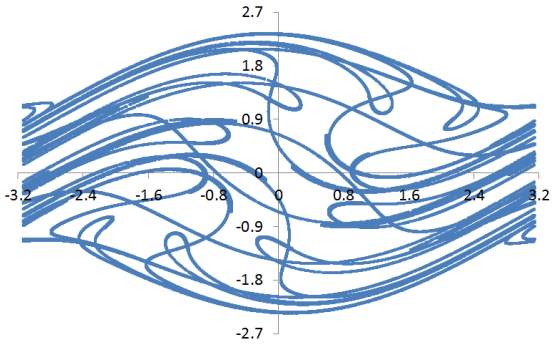
(a) $\Gamma = 0{,}9s^{-2}$, órbita en el espacio de fases (b) $\Gamma = 0{,}9s^{-2}$, sección de Poincaré

Figura 5.8: Dinámica en el espacio de fases (izquierda) y sección de Poincaré (derecha). $\Gamma = 0{,}9s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. El péndulo describe una órbita periódica en el espacio de fases cuyo periodo es $T = 2\pi/\omega$; por tanto, la sección de Poincaré muestra un único punto.

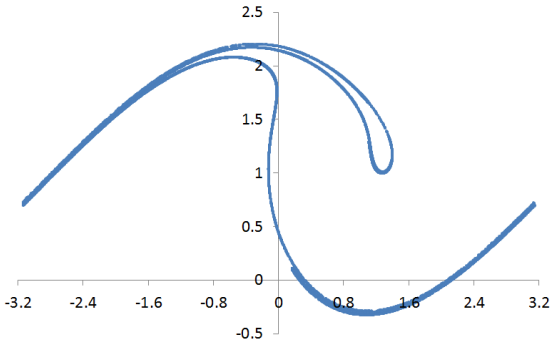


(a) $\Gamma = 1{,}07s^{-2}$, órbita en el espacio de fases (b) $\Gamma = 1{,}07s^{-2}$, sección de Poincaré

Figura 5.9: Dinámica en el espacio de fases (izquierda) y sección de Poincaré (derecha). $\Gamma = 1{,}07s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. El péndulo describe una órbita periódica en el espacio de fases cuyo periodo es el doble del del término forzante; por tanto, la sección de Poincaré muestra dos puntos.

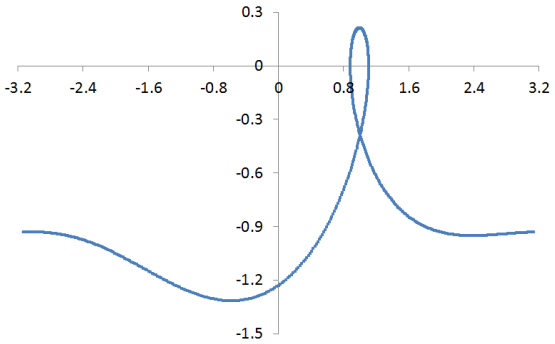


(a) $\Gamma = 1{,}19s^{-2}$, órbita en el espacio de fases

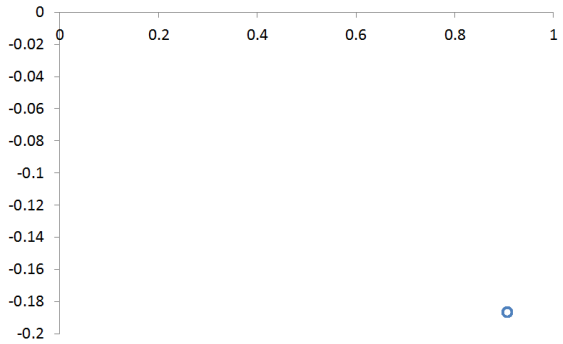


(b) $\Gamma = 1{,}19s^{-2}$, sección de Poincaré

Figura 5.10: Dinámica en el espacio de fases (izquierda) y sección de Poincaré (derecha). $\Gamma = 1{,}19s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. El péndulo describe una órbita caótica, con un atractor extraño.

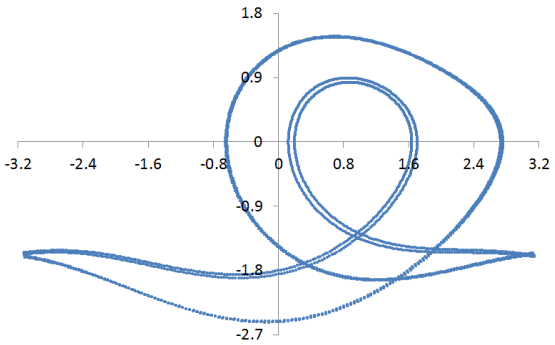


(a) $\Gamma = 1{,}3s^{-2}$, órbita en el espacio de fases

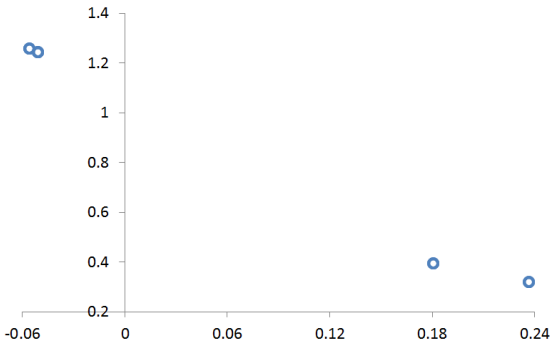


(b) $\Gamma = 1{,}3s^{-2}$, sección de Poincaré

Figura 5.11: Dinámica en el espacio de fases (izquierda) y sección de Poincaré (derecha). $\Gamma = 1{,}3s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. El péndulo describe una órbita periódica en el espacio de fases cuyo periodo vuelve a ser el del término forzante; por tanto, la sección de Poincaré muestra un único punto.



(a) $\Gamma = 1{,}46s^{-2}$, órbita en el espacio de fases



(b) $\Gamma = 1{,}46s^{-2}$, sección de Poincaré

Figura 5.12: Dinámica en el espacio de fases (izquierda) y sección de Poincaré (derecha). $\Gamma = 1{,}46s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. El péndulo describe una órbita periódica en el espacio de fases cuyo periodo es cuatro veces el del término forzante; por tanto, la sección de Poincaré muestra cuatro puntos.

Apéndice A

Criptografía RSA

A.1. Introducción

La necesidad de transmitir mensajes secretos es tan antigua como la Humanidad. Ha sido crucial en las relaciones entre países, en conspiraciones, en el desarrollo de guerras... A lo largo de la Historia se han dado distintas soluciones.

Se puede confiar el mensaje a un mensajero de absoluta lealtad, por ejemplo. Pero el mensajero puede ser interceptado (y torturado, o sobornado).

Otra opción es manipular el mensaje de modo que sea incomprensible para quien no esté en posesión de un secreto (emisor y receptor comparten una clave, un algoritmo...); esta opción ha sido abundantemente ilustrada en la literatura y el cine. Pero esta solución tampoco está exenta de problemas. Cuantas más personas conozcan el secreto más fácil es que alguna sea capturada, con lo que el secreto quede comprometido (pensemos, por ejemplo, en un ejército todas cuyas compañías y submarinos tengan copia del libro de claves). Por otra parte, el algoritmo puede ser descubierto.

El desarrollo tecnológico ha hecho el problema más acuciante: los ordenadores son cada vez más potentes, descifrar en fracciones de segundo mensajes que a Sherlock Holmes (en “La aventura de los bailarines") o William Legrand (en “El escarabajo de oro") les llevaron días, y dejar en evidencia al ordenador que Turing usó en Bletchley Park. Además, las comunicaciones electrónicas permiten transacciones entre cualquier par de puntos del globo, que es necesario asegurar.

Las fuerzas que hacen que el problema sea difícil de resolver son:

- Cualquiera debe poder enviar mensajes a quien esté dispuesto a recibirlos, aunque nunca hayan tenido contacto previo. Un potencial receptor no puede prever quiénes (ni cuántos) desearán comunicarse con él, ni cuándo intentarán hacerlo.
- Los mensajes deben ser confidenciales: sólo emisor y receptor deben poder entenderlos.
- El medio a través del cual se transmiten los mensajes es inseguro (un hacker puede interceptar nuestras comunicaciones).
- Emisor y receptor no pueden compartir claves y algoritmos secretos.

En 1977, Rivest, Shamir y Adleman presentaron un sistema criptográfico basado en la teoría de números que satisfacía los anteriores requisitos y, al menos en conjetura, es

seguro. El sistema se resume en los tres algoritmos que presentamos a continuación. Al describirlos personificaremos los distintos roles que intervienen mediante tres personajes. Siguiendo la convención habitual, sus nombres empiezan por las tres primeras letras del alfabeto:

- Alicia, que desea recibir mensajes.
- Bárbara, que desea comunicarse con Alicia de forma confidencial.
- Carlos, un espía que ha interceptado un mensaje no dirigido a él y quiere descifrarlo.

Como veremos a continuación, para encriptar un texto lo codificaremos (lo transformaremos en un número entero) y realizaremos operaciones matemáticas con dicho número. El resultado (un número entero) será el mensaje encriptado. Para desencriptar un mensaje realizaremos las operaciones inversas y decodificaremos el entero resultante. A lo largo de este apéndice usaremos la tabla de codificación 4.1, presentada en la tarea 4.5.

A.2. Fundamentos matemáticos (resumen para ejecutivos)

Para seguir el resto de la sección necesitamos dos definiciones y un teorema. Se puede encontrar un desarrollo riguroso de los fundamentos matemáticos de la criptografía RSA en el apéndice B

Definición A.2.1. Función de Euler
Dado un entero positivo n la función de Euler ($\varphi(n)$) es la cantidad de enteros positivos menores que n y coprimos con él.

- Si n es un número primo, entonces $\varphi(n) = (n-1)$.
- Si n es producto de dos primos, p y q, entonces $\varphi(n) = (p-1)\cdot(q-1)$.

Para más información, ver la definición B.2.1 y el teorema B.2.5, que muestra cómo se calcula en general.

Definición A.2.2. Inverso modular Dados dos enteros, n y e, el inverso modular de e módulo n es el número d que satisface: $e \cdot d \equiv 1\ (mod\ n)$.

Teorema A.2.3. *Sean los enteros no negativos M, e, d y n tales que:*

- *n es un producto de primos no repetidos ($n = p_1 \cdot p_2 \cdot ... \cdot p_n$, donde $p_i \neq p_j$ si $i \neq j$)*
- $0 \leq M < n$
- $mcd(e, \varphi(n)) = 1$
- *d es el inverso de e módulo $\varphi(n)$: $e \cdot d \equiv 1\ (mod\ \varphi(n))$.*

Entonces

$$M = (M_e)^d\ \%\ n, \text{donde } M_e = M^e\ \%n \tag{A.1}$$

El lector interesado puede encontrar la demostración del teorema anterior en el apéndice B.

Cuadro A.1: Ejemplo 1 (Creación de claves)

i: Generar dos primos distintos, p y q	
$p_1 =$	51854389
$q_1 =$	90863153
ii: Calcular n y $\varphi(n)$	
$n_1 = p_1 \cdot q_1 =$	4711653281428517
$\varphi(n_1) = (p_1 - 1) \cdot (q_1 - 1) =$	4711653138710976
iii: Encontrar un entero e t.q. $mcd(e, \varphi(n)) = 1$	
$e_1 =$	1307
iv: Calcular el inverso de e módulo $\varphi(n)$	
$d_1 =$	4556640831928595

A.3. Algoritmos

A.3.1. Creación de clave privada

Alicia da los siguientes pasos:

Algoritmo A.3.1*:* Creación de claves

i) Generar dos primos distintos, p y q.

ii) Calcular el producto $n = p \cdot q$ y la función de Euler $\varphi(n) = (p-1) \cdot (q-1)$.

iii) Encontrar un número e t.q. $mcd(e, \varphi(n)) = 1$.

iv) Calcular el inverso modular de e módulo $\varphi(n)$: d t.q. $ed \equiv 1\ (mod \varphi(n))$.

El proceso se ejemplifica en la tabla A.1.

El par (n, e) es la clave pública: Alicia la pone en conocimiento de todo el mundo. Guarda en secreto el valor de d (no comparte su secreto con nadie). p y q ya han cumplido su misión: puede borrarlos de su memoria.

Alicia también hace pública una codificación (en nuestro ejemplo, la tabla 4.1).

A.3.2. Construcción de un mensaje

Bárbara desea enviarle un mensaje condidencial a Alicia.

Algoritmo A.3.2*:* Encriptación de un mensaje.

Dado un mensaje, T:

i) Codificación: Se transforma T en un número entero, M. Para ello se sustituye cada carácter por una pareja de dígitos según la tabla de codificación publicada por Alicia (en nuestro caso, la tabla 4.1: $'A' \rightarrow$ "01", $'B' \rightarrow$ "02" ...).

ii) Encriptación: Se calcula: $M_{encriptado} \equiv M^e \, \%n$

El proceso se ejemplifica en la tabla A.2.

Bárbara le envía $M_{encriptado}$ a Alicia.

Cuadro A.2: Ejemplo 2 (Encriptación de un mensaje)

Texto (T):	Bazinga!
Clave pública:	(n_1, e_1) (igual que en la tabla A.1)
i: Codificación	
Mensaje (M):	0227523540332770
ii: Encriptación	
$M_{encript} = M^{e_1} \%n_1$:	263663526763178

Cuadro A.3: Ejemplo 3 (Desencriptación de un mensaje)

Mensaje encriptado ($M_{encript}$):	263663526763178
Clave pública:	(n_1, e_1) (igual que en la tabla A.1)
i: Encontrar el inverso de e módulo $\varphi(n)$	
$d_1 =$	4556640831928595
ii: Desencriptación	
$M = M_{encript}^{d_1} \%n_1$:	227523540332770
iii: Asegurar que M tiene un número par de dígitos	
M:	0227523540332770
iv: Decodificación	
T:	Bazinga!

A.3.3. Descifrado de un mensaje

Alicia recibe el mensaje de Bárbara. El teorema A.2.3 indica cómo desencriptar el mensaje $M_{encript}$ y recuperar M. Alicia sigue los pasos indicados a continuación:
Algoritmo A.3.3*:* Desencriptación de un mensaje

i) Dada la clave pública (n, e) se encuentra el inverso de e módulo $\varphi(n)$: el número d t.q. $e \cdot d \equiv 1\ mod(\varphi(n))$.

ii) Se calcula $M = (M_{encriptado})^d \%n$

iii) Si M tiene un número impar de dígitos, se añade un 0 al inicio.

iv) Se decodifica M, sustituyendo cada pareja de dígitos por su letra correspondiente según la tabla de codificación publicada por Alicia (en nuestro caso, la tabla 4.1: "01" $\rightarrow'$ A', "02" $\rightarrow'$ B' ...).

Observa que el paso i) es inmediato porque Alicia conoce el valor de d. El proceso se ejemplifica en la tabla A.3.

Supongamos ahora que Carlos intercepta el mensaje de la tabla A.4. Sólo conoce la clave pública (n, e), pero ignora el valor de d. Para calcularlo debe dar los pasos mostrados en el ejemplo A.4.

Cuadro A.4: Ejemplo 4 (Desencriptación de un mensaje)

Mensaje encriptado ($M_{encript}$):	801336179
Clave pública:	$(n, e) = (1373606407, 3137)$
i: Encontrar el inverso de e módulo $\varphi(n)$	
Descomponer n	$n = 32911 \cdot 41737$
Calcular $\varphi(n)$	$\varphi(n) = 32910 \cdot 41736 = 1373531760$
Calcular d	$d = 136608833$
ii: Desencriptación	
$M = M_{encript}^{d} \,\%n$:	547383144
iii: Asegurar que M tiene un número par de dígitos	
M:	0547383144
iv: Decodificación	
T:	Euler

A.4. ¿Cómo de difícil es descifrar un mensaje?

No hay misterios: sabemos exactamente qué algoritmo seguir para desencriptar cualquier mensaje RSA basta seguir el algoritmo A.3.3. ¿Dónde está la seguridad? ¿Qué nos impide desencriptar cualquier mensaje, como acabamos de hacer?

A.4.1. Descomposición factorial: crecimiento exponencial

En el ejemplo de la tabla A.4, Carlos sólo conoce la clave pública (n, e). Para calcular d necesita hallar antes la descomposición factorial de n, y ese es precisamente el punto más delicado. La robustez de la criptografía RSA se basa en que no se conoce ningún algoritmo eficiente para descomponer un número entero como un producto de primos. A fecha de hoy, el algoritmo más eficiente es la criba general del cuerpo de números (CGCN) [20]. Siguiéndolo, el número de operaciones necesarias para descomponer un número n de n_{Dig} dígitos es del orden de:

$$nOper(nDig) = exp\left((64/9 \cdot ln(10) \cdot n_{Dig})^{1/3} \cdot ln\,(ln(10) \cdot n_{Dig})^{2/3}\right) \qquad \text{(A.2)}$$

Observa que el número de operaciones (y el tiempo necesario para realizarlas) aumentan exponencialmente con el tamaño de n. Para hacernos una idea del impacto del tamaño de n, en la tabla A.5 mostramos el tiempo (normalizado) en función del número de dígitos de n:

Para tener idea cuantitativa del tiempo requerido, tomemos como ejemplo los números RSA-640, RSA-200 y RSA-678. Los dos primeros fueron factorizados en 2005, el tercero en 2009 usando clusters de ordenadores. En la tabla A.6 mostramos sus características, así como el tiempo que le hubiera costado factorizarlos a un ordenador con un microprocesador 2.2 GHz AMD Opteron mononúcleo:

¿Cuántos dígitos debe tener un número n para ser inexpugnable al ordenador más potente del mundo? Para contestar a esta pregunta haremos una estimación rápida, partiendo de los datos de la tabla A.6. Si no nos preocupa la exactitud del resultado (y nos es suficiente con una estimación del orden de magnitud) podemos caracterizar la potencia de

Cuadro A.5: Complejidad de la descomposición factorial de un entero de n_{Dig} dígitos siguiendo el algoritmo CGCN. El tiempo está normalizado: $tiempo(nDig) = nOper(nDig)/nOper(100)$. La tabla se lee: descomponer un número de 200 dígitos cuesta casi 600 000 veces más que descomponer uno de 100...

n_{Dig}	tiempo
100	1
193	$2{,}74 \cdot 10^5$
200	$5{,}96 \cdot 10^5$
232	$1{,}72 \cdot 10^7$
300	$9{,}57 \cdot 10^9$
400	$2{,}55 \cdot 10^{13}$
500	$2{,}22 \cdot 10^{16}$
1000	$2{,}12 \cdot 10^{27}$

Cuadro A.6: Hitos: descomposición factorial de algunos enteros RSA. Indicamos su nombre, referencia (URL donde se publicaron los detalles de su factorización), su tamaño (número de dígitos decimales y de bits) y el tiempo equivalente (el tiempo que le hubiera costado factorizarlo a un solo ordenador con un microprocesador 2.2 GHz AMD Opteron mononúcleo). Al pasar de 193 dígitos a 200 (añadiendo sólo 7 dígitos) el tiempo se duplica, y al pasar a 232 se multiplica por 60. Observa que los resultados son compatibles con las estimaciones de la tabla A.5

Nombre	referencia	dígitos	bits	tiempo
RSA-640	[12]	193	640	400 meses
RSA-200	[18]	200	663	75 años
RSA-768	[19]	232	768	2 000 años

cálculo de un ordenador indicando el número de operaciones de coma flotante que puede realizar por segundo (flops). Podemos estimar la velocidad de cálculo de un microprocesador 2.2 GHz AMD Opteron mononúcleo en 1 Gflops. En verano de 2015 se publicó que el presidente Obama impulsó la construcción del que será el ordenador más potente del mundo [1]: está previsto que entre en funcionamiento en 2022, y tendrá una potencia de cálculo de 10^9 Gflops: será mil millones de veces más rápido que nuestro viejo Opteron, y podrá factorizar el número RSA-768 en apenas $2 \cdot 10^{-6}$ años, es decir medio minuto. Consultando la tabla A.5, le costaría $1{,}69 \cdot 10^4$ segundos (algo menos de 5 horas) factorizar un entero de 300 dígitos, $4{,}33 \cdot 10^7$ segundos (17 meses) uno de 400 dígitos, $3{,}64 \cdot 10^{10}$ segundos (1 200 años) uno de 500 dígitos y $3{,}05 \cdot 10^{21}$ segundos uno de 1000 dígitos (como referencia, considera que desde el Big-Bang han transcurrido unos 14000 millones de años, es decir unos $5 \cdot 10^{17}$ segundos.

A.4.2. ¿Es obligatorio factorizar? La conjetura RSA

Puesto que el problema es factorizar, ¿es posible saltarnos ese paso? En su artículo, Rivest, Shamir y Adleman analizan distintas alternativas: calcular $\varphi(n)$ sin necesidad de factorizar n, o calcular d sin factorizar n y sin calcular $\varphi(n)$. Plantean algunas vías de ataque, pero todas ellas acaban siendo tan costosas como factorizar n.

Rivest, Shamir y Adleman basan su apuesta en una conjetura: *"any way of breaking our scheme must be as difficult as factoring. We have not been able to prove this conjecture, however".*

Quizá nunca llegue a demostrarse esta conjetura, y sigamos usando el método RSA sin tener la certeza absoluta sobre su seguridad. Quizá algún día se desarrolle un algoritmo eficiente para factorizar enteros, o la conjetura sea false y se encuentre un atajo para calcular d; en ese caso la criptografía RSA se desmoronaría como un castillo de arena. Una cosa está clara: con la matemática y la algoritmia de que disponemos, la criptografía RSA es inexpugnable al ataque de ordenadores tal como los conocemos.

A.4.3. Nuestro desafío

Te planteamos como desafío descifrar los mensajes de los ejemplos de las tablas A.7 a A.10. En los tres primeros ejemplos, las claves públicas son vulnerables: explotando su debilidad podrás descifrar los mensajes en cuestión de segundos. Creemos que en el último ejemplo la clave es robusta. No creemos que te sea posible descifrarlo.

Cuadro A.7: Ejemplo 5 (Desencriptación de un mensaje)

Clave pública	
n:	114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541
e:	907
Mensaje encriptado	
	101337642927733264264040317727185607274301270023925947112102688941632053584462741656682655652939532215700895029228671220906105 80

Cuadro A.8: Ejemplo 6 (Desencriptación de un mensaje)

Clave pública
(n, e) igual que en la tabla A.7
Mensaje encriptado
110741506991085698632623352618702756701440679176048816718027983313888027574595562276758834039334045276484886417666311070045141 14

Cuadro A.9: Ejemplo 7 (Desencriptación de un mensaje)

Clave pública	
n:	386545236867608795336764655384382194821920631863758898280117281763386175676700568300479232780401415804538417324791023088762884285780470921703573413893056830371292258984581381751521167921358103052941990568303712922589845813817515211679213581030529419 9
e:	162917
Mensaje encriptado	
	192502279401861536241723859528035256620705892490346900135077952260065024111493725119478206385990467332883616362919995265968676535566240005077448779758881428243347104744917446680306749210666468522234 09

Cuadro A.10: Ejemplo 8 (Desencriptación de un mensaje)

Clave pública	
n:	59541586125591083416986904060907398041654956453374
	51502682854315505566469815709 31
e:	12347
Mensaje encriptado	
	56996833687802024450597996100479976756730613807741
	62627691202057548831062716408 71

A.4.4. Potencias de 10: órdenes de magnitud

Para comprender la magnitud de los números con que vamos a trabajar necesitamos referencias astronómicas. Desde el Big Bang han transcurrido unos 14000 millones de años, es decir unos $5 \cdot 10^{17}$ segundos. Dado que la luz viaja a unos $3 \cdot 10^8$ ms^{-1}, el radio del Universo visible es de unos $1{,}5 \cdot 10^{26}$ metros. El radio de un átomo de hidrógeno es de unos $5 \cdot 10^{-11}$ metros (radio de Bohr), de modo que en el Universo visible cabrían unos $(1{,}5 \cdot 10^{26}/5 \cdot 10^{-11})^3 \simeq 3 \cdot 10^{109}$ átomos de hidrógeno densamente empaquetados.

Los mensajes encriptados de los ejemplos de las tablas A.7 a A.9 tienen 200 dígitos. n tiene 200 dígitos (es del orden de 10^{200}), el tamaño de $\varphi(n)$ es de ese mismo orden y e tiene 10 dígitos, por lo que d tendrá como mínimo unos 190 dígitos (es del orden de 10^{190}).

Si el superordenador de Obama [1] hubiera comenzado a trabajar en el mismo instante del Big Bang, le habría dado tiempo de hacer $5{\cdot}10^{17}$ segundos$\times 10^{18}$ flops/segundo $= 5{\cdot}10^{35}$ operaciones. ¿Cuántas operaciones te harán falta para calcular M^d_{encr}?

Por otro lado, M^d_{encr} es del orden de $(10^{200})^{10^{190}} = 10^{2\cdot 10^{192}}$. En otras palabras, se trata de un número de $2 \cdot 10^{192}$ dígitos. Aunque cada dígito ocupara el espacio de un átomo, M^d_{encr} no cabría en el Universo visible.

Apéndice B

Fundamentos teóricos del sistema criptográfico RSA

En este apéndice se recuerdan los resultados elementales de Teoría de Números que constituyen la base matemática del sistema criptográfico de clave asimétrica *RSA*, descrito originalmente en el artículo de Rivest, Shamir y Adleman [30]. La mayor parte de este material puede encontrarse en cualquier libro de texto sobre la materia, por ejemplo en [2].

B.1. El teorema fundamental de la Aritmética

Con objeto de hacer autocontenida esta revisión, comenzamos recordando algunas nociones y resultados básicos relacionados con la divisibilidad en el anillo $\mathbb{Z}$ de los números enteros, que en última instancia nos permitirán enunciar y probar el Teorema Fundamental de la Aritmética (B.1.11).

Teorema B.1.1. *Para cada par de enteros m y n, con $n \neq 0$, existen otros dos enteros q y r, únicos, tales que $m = q \cdot n + r$ y $0 \leq r < |n|$. Los enteros m, n, q y r suelen ser llamados* dividendo, divisor, cociente *y* resto*, respectivamente.*

Demostración. Supongamos, en primer lugar, que $n > 0$. Para demostrar la existencia basta tomar como q el mayor entero tal que qn es menor que m; esto es, tal que $qn \leq m < (q+1)n$. Entonces, restando qn de los tres miembros de esta desigualdad se obtiene $0 \leq m - qn < n$, por lo que basta definir $r = m - qn$.

Para probar la unicidad, supongamos que existen enteros q_1, r_1, q_2 y r_2 tales que

$$\begin{aligned} m &= q_1 \cdot n + r_1 \quad \text{y} \quad 0 \leq r_1 < n \\ m &= q_2 \cdot n + r_2 \quad \text{y} \quad 0 \leq r_2 < n \end{aligned}$$

Entonces, $q_1 n + r_1 = q_2 n + r_2$, de donde se deduce que $n\,|q_1 - q_2| = |r_1 - r_2|$. Y como $0 \leq r_1, r_2 < n$, también se tiene que $0 \leq |r_1 - r_2| < n$. Así, $0 \leq n\,|q_1 - q_2| < n$ y, como n es mayor que 0, dividiendo tenemos que $0 \leq |q_1 - q_2| < 1$. Finalmente, como q_1 y q_2 son enteros, necesariamente $0 = |q_1 - q_2|$, esto es, $q_1 = q_2$, de donde también se sigue que $r_1 = r_2$.

Si $n < 0$ entonces $-n > 0$, por lo que existen q' y r únicos tales que $m = q'(-n) + r = (-q')n + r$ con $0 \leq r < -n = |n|$, bastando tomar $q = -q'$. ∎

Nota B.1.2. Los lenguajes de programación proporcionan operadores para calcular el cociente y el resto de la división entre números enteros, como los operdores "`/`" y "`%`" de los lenguajes C/C++ y Java. No obstante, por razones de implementación, estos operadores no devuelven los valores sugeridos en el teorema anterior cuando el dividendo o el divisor son negativos. Concretamente, si m y n son el dividendo y el divisor, en los lenguajes mecionados su cociente q es un entero cuyo valor absoluto es igual al cociente $|m| \,/\, |n|$, mientras que el resto es un entero r tal que $0 \leq |r| < |n|$, cuyo signo se ajusta en función del cociente q para que se tenga la igualdad $m = qn + r$.

Por ejemplo, las expresiones `m/n` y `m%n` se evalúan como indica la tabla siguiente:

	`m` $= 7$		`m` $= -7$	
	`n` $= 3$	`n` $= -3$	`n` $= 3$	`n` $= -3$
`m/n`	2	-2	-2	2
`m%n`	1	1	-1	-1

Definición B.1.3 (Divisibilidad)**.** Se dice que un entero m *divide* a, o *es divisor* de, otro entero n, y se escribe $m \mid n$, si existe $a \in \mathbb{Z}$ tal que $am = n$; esto es, si el resto de la división de n por m es cero.

De entre las propiedades que se pueden deducir directamente de la definición anterior, cuyas demostraciones, por su sencillez, proponemos como ejercicio, destacamos las siguientes:

1. Todo entero m satisface que $1 \mid m$ y $m \mid 0$ (1 es divisor de todo entero y todo entero es divisor de cero).
2. Si a es divisor de m y de n, entonces también es divisor de cualquier combinación lineal $bm + cn$, donde b, c son números enteros cualesquiera.
3. La divisibilidad es transitiva: si $a \mid b$ y $b \mid c$, entonces $a \mid c$.
4. Si $d \mid n$, con $n \neq 0$ entonces $d \leq |n|$.

La última de estas propiedades indica que el conjunto de los divisores de todo entero no nulo está acotado, lo que permite introducir la noción de máximo común divisor.

Definición B.1.4. El *máximo común divisor* de dos enteros m y n, no simultáneamente nulos, es el mayor entero $d = \mathrm{mcd}(m, n)$ que divide tanto a m como a n.

Nótese que si $m \neq 0$ entonces $\mathrm{mcd}(m, 0) = |m|$. El primero de los siguientes resultados, que resultarán de gran utilidad más adelante, proporciona un método, conocido como *Algoritmo de Euclides*, para calcular el máximo común divisor de dos enteros no nulos.

Proposición B.1.5. *Sean m, n dos enteros no nulos. Entonces,* $\mathrm{mcd}(m, n) = \mathrm{mcd}(n, r)$*, donde $0 \leq r < |n|$ es el resto de la división de m por n.*

Demostración. Sea $q \in \mathbb{Z}$ el cociente de la división de m por n. Dado que $m = q \cdot n + r$, es inmediato que todo divisor común de n y de r lo es también de m. Y recíprocamente, puesto que $r = m - qn$, todo divisor común de m y de n lo es también de r. Por tanto, el conjunto de divisores comunes de m y n coincide con el de divisores comunes de n y r. Así, los máximos de ambos conjuntos, $\mathrm{mcd}(m, n)$ y $\mathrm{mcd}(n, r)$, también coinciden. ∎

Teorema B.1.6 (Identidad de Bézout)**.** *Sean m, n dos enteros no simultáneamente nulos. Entonces, existen otros dos enteros a, b tales que* $\mathrm{mcd}(m, n) = a \cdot n + b \cdot m$.

Demostración. Sea S el subconjunto de los enteros que se pueden expresar como una combinación lineal de m y n; esto es, $S = \{xm + yn \mid x, y \in \mathbb{Z}\}$. Probaremos que $\text{mcd}(m, n) = d$, donde $d = x_d m + y_d n$ es el menor elemento de S que es estrictamente mayor que 0.

En efecto, por el Teorema B.1.1, existen $c, r \in \mathbb{Z}$ tales $m = c \cdot d + r$, donde $0 \leq r < d$. Entonces, $r = m - cd = (1 - cx_d)m - y_d n \in S$, de donde se deduce que $r = 0$ y, por tanto, que d divide a m. Análogamente se demuestra que $d \mid n$. Finalmente, supongamos que d' divide a m y n. Entonces d' también divide $d = x_d m + y_d n$, de donde se sigue $d' \leq d$. ■

Teorema B.1.7 (Lema de Euclides (1.5 en [2]))**.** *Si $a \mid bc$ y $\text{mcd}(a, b) = 1$, entonces $a \mid c$.*

Demostración. Dado que $\text{mcd}(a, b) = 1$, por el Teorema B.1.6 sabemos que existen $\alpha, \beta \in \mathbb{Z}$ tales que $1 = \alpha \cdot a + \beta \cdot b$. Al multiplicar ambos miembros por c, tenemos $c = \alpha \cdot c \cdot a + \beta \cdot b \cdot c$. Esto es, c es suma de dos términos cada uno de los cuales es múltiplo de a, y por tanto c es múltiplo de a. ■

Corolario B.1.8. *Sean $b, c, n \in \mathbb{Z}$ tres enteros. Entonces, $\text{mcd}(b, n) = 1 = \text{mcd}(c, n)$ si y sólo si $\text{mcd}(bc, n) = 1$.*

Demostración. Sea $k = \text{mcd}(b, n)$. Entonces, por definición, $k \mid n$ y $k \mid b$, de donde se deduce que que también $k \mid bc$. Por tanto, $\text{mcd}(b, n) = 1$ si $\text{mcd}(bc, n) = 1$.

Recíprocamente, sean $a = \text{mcd}(bc, n)$ y $\alpha = \text{mcd}(a, b)$. Así, $\alpha \mid b$ y $\alpha \mid a$, y como $a \mid n$, también α divide a n, de donde se deduce que $\alpha = 1 = \text{mcd}(b, n)$. Entonces, $1 = \text{mcd}(a, b)$, y el Lema de Euclides (B.1.7) implica que $a \mid c$. Dado que el máximo divisor común de c y n es 1, se sigue que $a = 1$. ■

Los números primos han reclamado la antención de los mátematicos desde la antigüedad. Su estudio es una parte importante de la Teoría de Números y se encuentran presentes en numerosos resultados y conjeturas, en particular son fundamentales en el sistema criptográfico RSA. A continuación recordamos su definición y algunas de sus propiedades.

Definición B.1.9. Un entero p *es primo* si $p > 1$ y sus únicos divisores positivos son 1 y p. Si $n > 1$ no es primo, decimos que es un *número compuesto.*

Dos enteros n y m se dicen *coprimos* (*primos entre sí* o *primos relativos*) si $\text{mcd}(n, m) = 1$.

Proposición B.1.10.

I) *Si p es primo y no divide a un entero n, entonces $\text{mcd}(p^k, n) = 1$ para todo $k \geq 1$.*

II) *Si un número primo p divide a un producto $a_1 \cdot a_2 \cdot \ldots \cdot a_n$, entonces divide por lo menos a uno de los factores a_i.*

III) *Si un entero x es múltiplo de n primos distintos $p_1, p_2, \ldots, p_n$, entonces x es múltiplo de su producto; es decir, si $p_i \mid x$, para cada $1 \leq i \leq n$, y $p_i \neq p_j$ si $i \neq j$, entonces $(\prod_{i=1}^{n} p_i) \mid x$.*

Demostración.

i) Sea $\alpha = \text{mcd}(p, n)$. Por definición, $\alpha \mid p$ y, dado que p es primo, α sólo puede tener dos valores: 1 o p. Dado que p no es divisor de n, mientras que α sí lo es, el único valor posible de α es 1. Así, $\text{mcd}(p, n) = 1$, de donde se sigue, por el Corolario B.1.8, que $\text{mcd}(p^k, n) = 1$ para todo $k > 1$, suponiendo, por hipótesis de inducción, que $\text{mcd}(p^{k-1}, n) = 1$.

ii) Realizaremos la demostración por inducción sobre el número, n, de factores. Si el producto tiene un solo factor ($n = 1$) entonces $p \mid a_1$ y el resultado es obvio. Supongamos que $n > 1$ y p divide al producto $a_1 \cdot \ldots \cdot a_n$. El resultado se tiene trivialmente si $p \mid a_n$. En otro caso, si $p \nmid a_n$, entonces $\text{mcd}(p, a_n) = 1$ por el apartado anterior, y del Teorema B.1.7 se deduce que $p \mid (a_1 \cdot \ldots \cdot a_{n-1})$, de donde se sigue que p divide a alguno de estos factores por hipótesis de inducción.

iii) El resultado es trivial si el número de factores primos es uno. Usaremos un argumento inductivo para $n > 1$. Supongamos que x es múltiplo de n números primos p_i distintos. En particular $p_n \mid x$, por lo que existe un entero a tal que $x = a \cdot p_n$. Así, $p_i \mid ap_n$ y, puesto que $p_i \neq p_n$ son primos, $\text{mcd}(p_i, p_n) = 1$, de donde se deduce, por el Lema de Euclides B.1.7, que $p_i \mid a$ para todo $1 \leq i < n$. Entonces, a es múltiplo del producto $p_1 \cdot \ldots \cdot p_{n-1}$ por hipótesis de inducción, de donde se sigue el resultado. ■

Teorema B.1.11 (Teorema Fundamental de la Aritmética)**.** *Todo entero $n > 1$ se puede descomponer como un producto de factores primos de forma única; esto es, existen $p_1, \ldots, p_k$ primos distintos y $e_1, \ldots, e_k$ enteros positivos tales que $n = \prod_{i=1}^{k} p_i^{e_i}$.*

Demostración. Probaremos la existencia, es decir, que todo entero es igual a un prodcuto de factores primos, por inducción sobre el entero n. Para $n = 2$ la afirmación anterior es trivial, pues n es primo. Supongamos, por hipótesis de inducción, que todo entero estrictamente menor que n se puede descomponer como un producto de factores primos, y comprobemos que esta afirmación también es cierta para n. Si n fuera primo no hay nada que probar. En otro caso, existe un entero a que divide a n, con $1 < a < n$; esto es, existe otro entero b tal que $ab = n$. Por construcción, $a < n$ y $b < n$, por lo que ambos se pueden descomponer como producto de factores primos $a = a_1 \cdot \ldots \cdot a_r$ y $b = b_1 \cdot \ldots \cdot b_s$, por lo que $n = a_1 \cdot \ldots \cdot a_r \cdot b_1 \cdot \ldots \cdot b_s$ es también un producto de primos.

Para probar la unicidad, supongamos que $n = p_1^{e_1} \cdot \ldots \cdot p_r^{e_r} = q_1^{d_1} \cdot \ldots \cdot q_s^{d_s}$, donde $p_i < p_j$ y $q_i < q_j$ si $i < j$. Puesto que $p_1 \mid q_1^{d_1} \cdot \ldots \cdot q_s^{d_s}$ y tanto p_1 como los q_j son primos, de la Proposición B.1.10(ii) se deduce que $p_1 = q_j$ para algún $1 \leq j \leq s$; y, análogamente, se concluye que $q_1 = p_i$ para algún $1 \leq i \leq r$. Así $p_1 = q_1$, puesto que $p_1 \leq p_i = q_1$ y $q_1 \leq q_j = p_1$. Ahora comprobaremos que $e_1 = d_1$. En otro caso, podemos suponer, sin pérdida de generalidad, que $e_1 < d_1$. Entonces $m = p_2^{e_2} \cdot \ldots \cdot p_r^{e_r} = q_1^{d_1 - e_1} \cdot \ldots \cdot q_s^{d_s}$, de donde se deduce, por el argumento anterior, que $p_2 = q_1$, lo cual es una contradicción. Así, $k = p_2^{e_2} \cdot \ldots \cdot p_r^{e_r} = q_2^{d_2} \cdot \ldots \cdot q_s^{d_s}$, de donde se sigue que $r = s$ usando el mismo razonamiento junto con un sencillo argumento inductivo sobre el número r de factores primos. ■

Terminamos esta sección de fundamentos recordando la definición de la relación de congruencia y algunas de sus propiedades, cuya demostración se deja como ejercicio al lector interesado.

Definición B.1.12 (Congruencia)**.** Dados tres enteros a, b y n, con $n > 0$, decimos que a *es congruente con* b *módulo* n, y escribimos $a \equiv b \pmod{n}$, si n es divisor de $(a - b)$. Llamamos al número n *módulo de la congruencia.*

Proposición B.1.13.

i) *$a \equiv b \pmod{n}$ si y sólo si $r_a = r_b$, donde $0 \leq r_x < n$ es el resto de la división de $x \in \{a, b\}$ por n. En particular, $a \equiv r_a \pmod{n}$ para todo entero $a \in \mathbb{Z}$.*

ii) *Para todo entero $n > 0$, la congruencia módulo n es una relación de equivalencia:*

- *es reflexiva: $a \equiv a \pmod{n}$ para todo $a \in \mathbb{Z}$;*
- *es simétrica: si $a \equiv b \pmod{n}$ entonces $b \equiv a \pmod{n}$; y*
- *es transitiva: si $a \equiv b \pmod{n}$ y $b \equiv c \pmod{n}$ entonces $a \equiv c \pmod{n}$.*

iii) *Si $a \equiv \alpha \pmod{n}$ y $b \equiv \beta \pmod{n}$ entonces $ab \equiv \alpha\beta \pmod{n}$. En particular, $a^x \equiv r_a^x \pmod{n}$, para todo $x \in \mathbb{Z}$.*

Nota B.1.14. En el lenguaje de programación C se dice que el operador "`%`" devuelve el resto de la división de dos datos de tipo entero. Sin embargo, en la Nota B.1.2 hemos señalado que la expresión `m%n` devuelve un entero menor o igual que cero si el dividendo es negativo. Esta discrepancia no invalida la afirmación anterior: si, conforme a lo establecido en el Teorema B.1.1, q y r son el cociente y el resto, respectivamente, de la división de `m` por `n` es fácil comprobar que `m%n` $\equiv r \pmod{\texttt{n}}$; de hecho $r =$ `((m%n) + n)% n` para todo par de enteros `m` y `n` $\neq 0$.

B.2. La función φ de Euler y el sistema RSA

La base del sistema criptográfico RSA es la resolución de una ecuación lineal de congruencia en la que juega un papel fundamental la función φ de Euler. A continuación presentamos la definición de esta función y algunos resultados que nos permitirán calcular su valor para enteros arbitrarios.

Notación. Dado un conjunto C, denotaremos por $\sharp C$ al número de sus elementos (el *cardinal de* C).

Definición B.2.1. *La función φ de Euler* asigna a cada entero $n > 0$ el número de enteros positivos que son coprimos y menores o iguales que él; esto es:

$$\varphi(n) = \sharp\{m \in \mathbb{Z} \mid 1 \leq m \leq n \ \texttt{y} \ \ \mathrm{mcd}(m, n) = 1\} \ .$$

Es inmediato comprobar, a partir de la definición, que $\varphi(1) = 1$. El siguiente resultado también es inmediato.

Proposición B.2.2. *Un entero $n > 1$ es primo si y sólo si $\varphi(n) = n - 1$.*

Teorema B.2.3. *Si p es primo y $k > 0$ entonces $\varphi(p^k) = p^{k-1}(p - 1)$.*

Demostración. Basta restar de la cantidad $p^k - 1$, de enteros estrictamente menores que p^k, el número de ellos que no son coprimos con él. Para calcular este número, supongamos que $1 \leq n < p^k$. Entonces, por la Proposición B.1.10(i), $\mathrm{mcd}(n, p^k) > 1$ si y sólo si $n = ap$ para algún entero $1 \leq a < p^{k-1}$. Puesto que hay exactamente $p^{k-1} - 1$ de estos enteros, se tiene que $\varphi(p^k) = (p^k - 1) - (p^{k-1} - 1) = p^{k-1}(p - 1)$. ∎

Teorema B.2.4. *Si* $\mathrm{mcd}(m,n)=1$ *entonces* $\varphi(mn)=\varphi(m)\varphi(n)$.

Demostración. Si $m=1$ o $n=1$ el resultado es obvio ya que $\varphi(1)=1$. Supongamos pues que $m>1$ y $n>1$. Para cada entero $1\leq z<mn$ existen enteros q y r, con $0\leq r<m$, tales que $z=qm+r$, y sabemos, por la Proposición B.1.5, que $\mathrm{mcd}(z,m)=\mathrm{mcd}(r,m)$. Por tanto, dado $0\leq r<m$, todos los elementos del conjunto $C_r=\{z=qm+r\mid 0\leq q<n\}$ son coprimos con m si r también lo es, mientras que si r no es coprimo con m entonces ningún elemento de C_r será coprimo con él. Así, por el Corolario B.1.8, bastará probar que hay exactamente $\varphi(n)$ elementos coprimos con n en cada uno de los $\varphi(m)$ conjuntos C_r tales que $\mathrm{mcd}(m,r)=1$. Para ello, consideremos dos elementos distintos $z_1,z_2\in C_r$ y notemos que, al ser m y n coprimos, el Lema de Euclides implica que n no divide a su diferencia z_1-z_2. Por tanto, cada uno de los n elementos de C_r es congruente con un resto $0\leq s<n$ distinto, de donde se sigue que exactamente $\varphi(n)$ de ellos son coprimos con n, usando, otra vez, la Proposición B.1.5. ∎

A partir de estos resultados y del Teorema Fundamental de la Aritmética, no es difícil calcular el valor de $\varphi(n)$ usando un argumento inductivo sobre el número de factores primos del entero n.

Teorema B.2.5. *Para todo entero* $n\geq 1$, $\varphi(n)=n\prod_{p|n}\left(1-\frac{1}{p}\right)$, *donde* p *recorre el conjunto de factores primos de* n.

Dados tres enteros, $a,b,n\in\mathbb{Z}$, con $n>0$, una ecuación lineal de congruencia es una expresión de la forma $a\cdot x\equiv b\,(\mathrm{mod}\,n)$. En general, no siempre existe solución para este tipo de ecuaciones, y si existe no tiene por qué ser única. El siguiente resultado proporciona una condición suficiente para asegurar la existencia y unicidad (módulo n) de soluciones para estas ecuaciones.

Teorema B.2.6. *Sean* a *y* n *dos enteros positivos tales que* $\mathrm{mcd}(a,n)=1$. *Entonces:*

I) $a\cdot b\equiv a\cdot c\,(\mathrm{mod}\,n)$ *si y sólo si* $b\equiv c\,(\mathrm{mod}\,n)$.

II) *La ecuación* $ax\equiv b\,(\mathrm{mod}\,n)$ *tiene exactamente una solución (módulo* n*). Esto es, para todo* b *existe un único* $x_b\in\mathbb{Z}$ *tal que* $0\leq x_b<n$ *y* $ax_b\equiv b\,(\mathrm{mod}\,n)$. *En particular,*

III) *Existe un único entero* $0<d<n$, *llamado* inverso de a módulo n *o, simplemente,* inverso modular de a, *tal que* $ad\equiv 1\,(\mathrm{mod}\,n)$.

Demostración.

I) $ab\equiv ac\,(\mathrm{mod}\,n)$ implica que $a(b-c)$ es múltiplo de n. Dado que $\mathrm{mcd}(a,n)=1$, del Lema de Euclides B.1.7 se sigue que $b-c$ es múltiplo de n y, por tanto, que $b\equiv c\,(\mathrm{mod}\,n)$. Y, recíprocamente, si $b\equiv c\,(\mathrm{mod}\,n)$ entonces $b-c$ y, por tanto, $a(b-c)$ son múltiplos de n.

II) Dado que $\mathrm{mcd}(a,n)=1$, por la Identidad de Bézout B.1.6 existen enteros α,β tales que $1=\alpha\cdot a+\beta\cdot n$. Multiplicando ambos miembros por b se tiene que $b=\alpha\cdot b\cdot a+\beta\cdot b\cdot n$, de donde se sigue que $\alpha\cdot b$ es una solución para la ecuación de congruencia. Entonces, por la Proposición B.1.13(iii), basta tomar x_b como el resto de la división de αb por n para encontrar una solución menor que n.

Para probar la unicidad (módulo n) de esta solución, supongamos que $0\leq x_2<n$ es un entero tal que $ax_2\equiv b\,(\mathrm{mod}\,n)$. Como también $0\leq x_b<n$, entonces $|x_b-x_2|<n$, y del apartado anterior se sigue que $|x_b-x_2|$ es múltiplo de n, de donde se deduce que $x_b=x_2$. ∎

Estamos ya en condiciones de presentar el teorema de Euler–Fermat, central a la hora de comprender la criptografía RSA.

Teorema B.2.7 (Teorema de Euler-Fermat (5.17 en [2]))**.** *Sean a y n dos enteros positivos tales que* $\mathrm{mcd}(a,n)=1$. *Entonces:*

$$a^{\varphi(n)} \equiv 1 \,(\mathrm{mod}\, n)\ . \tag{B.1}$$

Demostración. Sea $A=\{a_1,a_2,\ldots,a_{\varphi(n)}\}$ el conjunto de todos los enteros no negativos menores que n y coprimos con él, y notemos que $a_i \equiv a_j \,(\mathrm{mod}\, n)$ si y sólo si $i=j$. Fijado un elemento $a\in A$, definimos el conjunto $B=\{b_1,b_2,\ldots,b_{\varphi(n)}\}$, donde $b_i=a\cdot a_i$.

En primer lugar, probaremos que cada elemento de B es congruente módulo n con un elemento distinto de A. Por la Proposición B.1.13 tenemos que $b_i \equiv r_i \,(\mathrm{mod}\, n)$, donde $0\le r_i<n$ es el resto de la división de b_i por n. Entonces, por la Proposición B.1.5 y el Corolario B.1.8 se tiene que $\mathrm{mcd}(r_i,n)=\mathrm{mcd}(b_i,n)=\mathrm{mcd}(aa_i,n)=1$, de donde se deduce que $r_i\in A$. Supongamos que $b_i=aa_i$ y $b_j=aa_j$ son congruentes con el mismo elemento de A. Dado que la congruencia es transitiva (ver la Proposición B.1.13), tenemos que $b_i\equiv b_j \,(\mathrm{mod}\, n)$ y, puesto que $\mathrm{mcd}(a,n)=1$, por B.2.6 $a_i\equiv a_j \,(\mathrm{mod}\, n)$, pero eso ocurre si y sólo si $i=j$.

Tras estas observaciones, la Proposición B.1.13(iii) nos permite establecer una relación entre los productos de todos los elementos de B y de A. Concretamente, se tiene que $\prod_{i=1}^{\varphi(n)} b_i \equiv \prod_{i=1}^{\varphi(n)} a_i \,(\mathrm{mod}\, n)$. Entonces, dado que $\prod_{i=1}^{\varphi(n)} b_i = \prod_{i=1}^{\varphi(n)} (a\cdot a_i) = a^{\varphi(n)}\cdot \prod_{i=1}^{\varphi(n)} a_i$, tenemos que

$$a^{\varphi(n)} \cdot \prod_{i=1}^{\varphi(n)} a_i \equiv 1 \cdot \prod_{i=1}^{\varphi(n)} a_i \,(\mathrm{mod}\, n)\ .$$

Entonces, ya que $\mathrm{mcd}(\prod_{i=1}^{\varphi(n)} a_i, n)=1$ por el Corolario B.1.8, obtenemos la Ecuación B.1 aplicando el Teorema B.2.6(i) a esta última expresión. ∎

El resultado siguiente es una consecuencia inmediata de este teorema.

Corolario B.2.8 (5.18 en [2])**.** *Sea p un número primo. Para todo entero a coprimo con p, $a^{p-1}\equiv 1 \,(\mathrm{mod}\, p)$.*

Utilizando el Corolario B.2.8 se obtiene el siguiente resultado en el que se basa la criptografía RSA:

Teorema B.2.9. *Sean M, e, d y n enteros no negativos tales que $0\le M<n$,* $\mathrm{mcd}(e,\varphi(n))=1$ *y $ed\equiv 1 \,(\mathrm{mod}\, \varphi(n))$. Si n es un producto de primos distintos, entonces*

$$(M^e)^d \equiv M \,(\mathrm{mod}\, n)\ . \tag{B.2}$$

Demostración. En primer lugar probaremos que $(M^e)^d = M^{ed} \equiv M \,(\mathrm{mod}\, p)$ para todo factor primo p de n. Si $\mathrm{mcd}(M,p)\neq 1$, por la Proposición B.1.10(i), M y, por tanto, M^{ed} son múltiplos de p; así $M^{ed}\equiv M \,(\mathrm{mod}\, p)$. En otro caso, si $\mathrm{mcd}(M,p)=1$, por el Corolario B.2.8 sabemos que $M^{p-1}\equiv 1 \,(\mathrm{mod}\, p)$. Por otro lado, como $ed\equiv 1 \,(\mathrm{mod}\, \varphi(n))$, sabemos que $ed-1$ es mútiplo de $\varphi(n)$ que, a su vez, es múltiplo de $p-1$, pues n es un producto de primos distintos (ver la Proposición B.2.2 y el Teorema B.2.4). Así, $ed=k(p-1)+1$ para algún entero $k\ge 1$, de donde se sigue que $M^{ed}=M^{k(p-1)}\cdot M=$

$\left(\prod_{i=1}^{k} M^{p-1}\right) \cdot M \equiv M \,(\mathrm{mod}\, p)$, usando k veces la Proposición B.2.6(i) para obtener la congruencia.

De este modo, $(M^e)^d - M$ es múltiplo de todos los factores primos primos de n y, por la Proposición B.1.10, también lo es de su producto n, como queríamos demostrar. ■

Apéndice C

Métodos de integración numérica: fórmulas de Newton-Cotes

Para resolver algunas tareas será necesario integrar funciones. Por ejemplo, en los problemas de dinámica, la posición es la integral de la velocidad, que es a su vez la integral de la aceleración. Nuestro problema es calcular la integral

$$I(b) = I(a) + \int_a^b f(x)dx \tag{C.1}$$

en casos donde no se dispone de una solución analítica. Puede ocurrir que f sea tan complicada que integrarla analíticamente esté fuera de nuestro alcance; peor aún, puede ocurrir que sólo conozcamos los valores de f para un conjunto discreto de puntos.

En estos casos podemos recurrir a métodos de integración numérica. Algunos textos utilizan el término *"cuadratura"* para referirse a integrales numéricas. Se trata de un término arcaico, que alude al método de integrar una función f tratando de construir un cuadrado con la misma superficie que la definida bajo f.

Los métodos de integración numérica más conocidos son las llamadas fórmulas de Newton-Cotes. En este apéndice haremos una breve introducción (remitimos a los interesados en profundizar en el tema a [7]). La idea es descomponer $[a, b]$ en subintervalos $[x_n, x_{n+1}]$, con $0 \leq n < F$, $x_0 = a$ y $x_F = b$, y en cada intervalo aproximar f mediante una función polinómica, fácil de integrar. Habitualmente tomaremos todos los subintervalos de la misma longitud ($x_n = n \cdot \Delta$, donde Δ es constante), Las claves del éxito del método son:

- Encontrar un polinomio que se aproxime a f en cada subintervalo. En este curso prevalecerá el requisito de sencillez: usaremos polinomios sencillos (de grado 0 o 1).
- Determinar la anchura de cada subintervalo. Intuitivamente, cuanto más pequeño sea $[x_n, x_{n+1}]$ menor será el error cometido en la aproximación (mejor será el resultado) pero, por otro lado, más subintervalos tendremos que considerar (se requerirá más tiempo de ejecución). La solución ideal pasa por encontrar un punto de equilibrio entre precisión y coste.

Un aspecto esencial es el cálculo de errores: determinar una cota superior al error cometido al sustituir la integral por una suma discreta de términos. Un cálculo riguroso está fuera del alcance de nuestro curso. Una posibilidad es utilizar la regla heurística de ir reduciendo

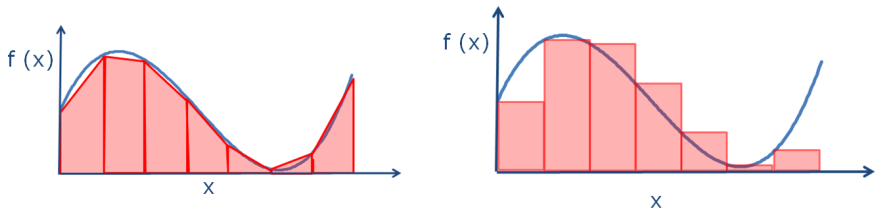


Figura C.1: Dos técnicas de integración numérica. La figura de la izquierda ilustra la regla del trapecio. En la figura de la derecha, aproximamos la integral de f por la suma de las áreas de una serie de rectángulos.

la longitud de los subintervalos hasta que el resultado se estabilice. En las tareas de este libro propondremos el valor de dicha longitud cada vez que se requiera.

Distinguiremos entre dos tipos de casos. Cuando se conoce el valor de f en los extremos de cada subintervalo de integración hablamos de fórmulas cerradas de Newon-Cotes; en caso contrario, hablamos de fórmulas abiertas.

Fórmulas cerradas de Newton-Cotes

La fórmula más sencilla es la *regla del trapecio*. Consiste en sustituir f en cada subintervalo $[x_n, x_{n+1}]$ por el polinomio de primer grado $f(x_n) + \frac{f(x_{n+1})-f(x_n)}{x_{n+1}-x_n} \cdot x$. De este modo, la integral se aproxima mediante la suma de las áreas de una sucesión de trapecios:

$$I(b) = I(a) + \int_a^b f(x)dx \to I(a) + \sum_{n=0}^{F-1} \frac{f(x_{n+1}) + f(x_n)}{2} \cdot (x_{n+1} - x_n) \; . \tag{C.2}$$

Cuando, la longitud de los subintervalos es constante, esto es, si para todo $0 \leq n < F$, $\Delta = x_{n+1} - x_n$, entonces la expresión C.2 se puede simplificar como

$$I(b) = I(a) + \int_a^b f(x)dx \approx I(a) + \frac{\Delta}{2}(f(a) + f(b)) + \Delta \sum_{n=1}^{F-1} f(x_n) \; . \tag{C.3}$$

La Figura C.1 (izquierda) muestra gráficamente la regla del trapecio: la superficie bajo la función (azul) se aproxima mediante la suma de superficies de los trapecios (área sombreada).

Supongamos, por ejemplo, que deseamos de integrar una función del tipo:

$$\int_0^h \frac{1}{\sqrt{g^2(x) - 1}} \, dx \; , con \tag{C.4}$$

$$g(x) = \alpha - \beta \cdot x \; , \tag{C.5}$$

donde α, β son constantes. Hacemos notar que la forma analítica de la integral es conocida: tras hacer el cambio de variable $z = \alpha - \beta \cdot x$ el resultado es:

$$\frac{1}{\beta} \cdot ln(z + \sqrt{z^2 - 1})|_{\alpha-\beta \cdot h}^{\alpha} \; . \tag{C.6}$$

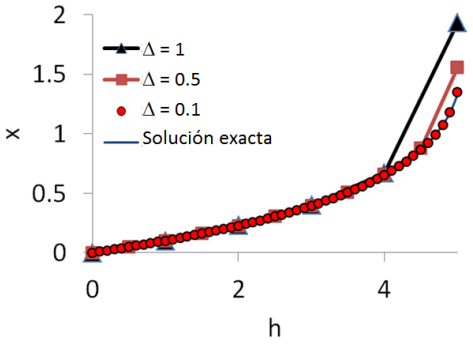


Figura C.2: Integral C.4 para los valores $\alpha = 11, \beta = 2$, realizada mediante la regla del trapecio con intervalos de longitud constante $[y_n, y_{n+1}] = \Delta$. Se comparan las integrales tomando $\Delta = 1$ (▲), $\Delta = 0{,}5$ (■), $\Delta = 0{,}1$ (•), y la solución exacta dada por la ecuación C.6 (línea continua). Nota que cuanto menor es la longitud de los subintervalos, Δ, el resultado de la integral numérica se aproxima más a la solución exacta.

El propósito de la tarea será familiarizarte con la integración numérica y comparar cómo el valor estimado varía en función de la longitud de los subintervalos. La figura C.2 muestra los resultados de la integral numérica para distintas longitudes de los subintervalos, y los compara con la solución exacta.

Fórmulas abiertas de Newton-Cotes

Consideremos un nuevo problema: la dinámica de un cuerpo en caída libre. La aceleración viene dada por

$$a = \frac{dv}{dt} = \alpha - \beta \cdot v^2 \,, \tag{C.7}$$

donde v es la velocidad, y α y β son constantes. En cursos más adelantados aprenderás a resolver la ecuación diferencial C.7. La solución es:

$$v(t) = \sqrt{\frac{\alpha}{\beta}}\,\frac{e^{2\gamma\,(t-t_0)} - A}{e^{2\gamma\,(t-t_0)} + A} \,, \tag{C.8}$$

$$y(t) = y(0) + \beta^{-1}\left(\log(e^{\gamma\,(t-t_0)} + Ae^{-\gamma\,(t-t_0)}) - \log(1+A)\right) \tag{C.9}$$

$$\text{con}\;\; \gamma = \sqrt{\alpha\beta},\;\; \text{y}\; A = \frac{1 - \sqrt{\beta/\alpha}\cdot v_0}{1 + \sqrt{\beta/\alpha}\cdot v_0} \,, \tag{C.10}$$

Como alternativa, resolveremos la ecuación C.7 por integración numérica. De nuevo, conocer de antemano la solución analítica nos servirá para validar nuestra solución. La velocidad en un instante t se obtiene integrando la aceleración ... ¡pero para conocer la aceleración necesitamos conocer la velocidad! El problema es resolver la integral:

$$v(t) = v(0) + \int_0^t \left(\alpha - \beta \cdot v^2\right) dt' \,, \tag{C.11}$$

donde nos encontramos con la (aparente) paradoja de que para calcular v tenemos que conocer v. A diferncia del ejemplo anterior, en que conocíamos de antemano la función que necesitábamos integrar, aquí sólo conocemos el valor inicial de dicha función: deberemos construirla paso a paso.

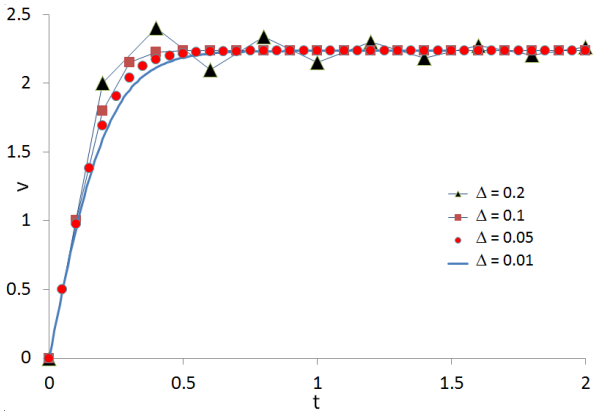


Figura C.3: Integral C.11 para los parámetros $\alpha = 10, \beta = 2$, $v(0) = 0$, realizada aproximado la función a integrar por rectángulos, con intervalos de longitud constante $[t_n, t_{n+1}] = \Delta$. Se comparan las integrales tomando $\Delta = 0{,}2$ (▲), $\Delta = 0{,}1$ (■), $\Delta = 0{,}05$ (•), y $\Delta = 0{,}01$ (línea continua); en la figura, el resultado de esta última integral se superpone con la solución exacta dada por la ecuación C.8.

La estrategia sigue siendo descomponer el intervalo de integración en subintervalos $[t_n, t_{n+1}]$. Por simplicidad, todos ellos tendrán la misma longitud: $t_{n+1} - t_n = \Delta$. En cada subintervalo, sustituiremos la función por un polinomio. A diferencia del apartado anterior, no conocemos el valor de la función a integrar en todos los puntos: sólo conocemos la velocidad inicial, $v(t = 0)$. La idea es la siguiente: en cada subintervalo de integración aproximaremos v mediante una función constante (el único valor de v que conocemos: el inicial). Así, calcularemos $v(t_1) = a(t = 0) \cdot \Delta$; a partir de ese valor ya podemos calcular $v(t_2) = a(t = 1) \cdot \Delta$, y así sucesivamente, de modo que $v(t_{n+1}) = a(t_n) \cdot \Delta$. Observa que este procedimiento equivale a aproximar la integral por el sumatorio de una serie de rectángulos, tal como se muestra en la Figura C.1 (derecha). En otras palabras, aproximamos la integral por el sumatorio

$$v(t) = v(0) + \int_0^t a(v)dt' \to v(0) + \sum_{n=0}^{n_F} a_n \cdot \Delta \,, \tag{C.12}$$

donde $a_n = a(v(t_n)) = \alpha - \beta \cdot v^2(t_n)$. La figura C.3 muestra los resultados de la integral numérica para distintas longitudes de los subintervalos, y los compara con la solución exacta.

Apéndice D

Cálculo de raíces de funciones: el teorema de Bolzano

El *Teorema de Bolzano* permite asegurar que una función se anula en un intervalo cuando se cumplen una serie de condiciones. Este teorema se puede utilizar para resolver distintos problemas, como por ejemplo, demostrar que una función toma un valor determinado o demostrar que dos funciones se cortan. El enunciado del teorema es el siguiente.

Teorema D.0.1. *Sea f una función real continua en un intervalo cerrado $[a, b]$ con $f(a)$ y $f(b)$ de signos contrarios. Entonces existe al menos un punto c del intervalo abierto (a, b) con $f(c) = 0$.*

El Teorema de Bolzano se entiende de forma muy intuituva si se representa geométricamente. Lo que nos dice es que si dos puntos $(a, f(a))$ y $(b, f(b))$ de la gráfica de una función continua están situados en diferentes lados del eje x, entonces la gráfica corta al eje x en al menos un punto entre a y b (ver figura D.1). Los valores de x para los que una función se hace cero se llaman *raíces* de la función. El teorema no especifica el número de raíces de la función en el intervalo $[a, b]$, solo afirma que como mínimo existe una.

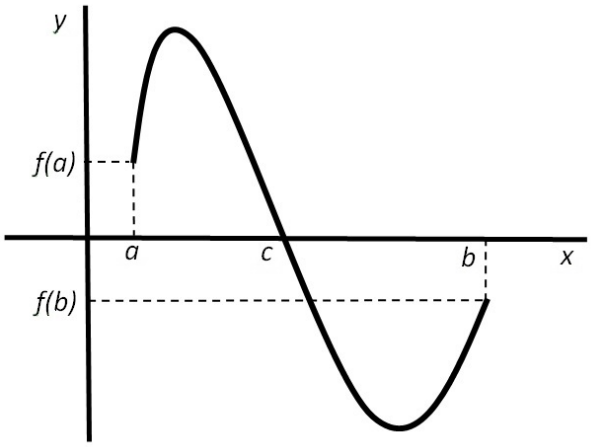


Figura D.1: Función continua que corta al eje x.

Dada una función f en un intervalo $[a, b]$ que cumpla las condiciones del Teorema de Bolzano, existen distintos métodos para calcular una aproximación a una raíz de la función.

Uno de estos métodos es el *Método de Bisección* que consiste en dividir el intervalo $[a, b]$ por la mitad, eligiendo el subintervalo en donde f cambia de signo (para asegurar al menos una raíz) y repetir el proceso varias veces hasta encontrar una aproximación suficientemente buena.

Suponiedo que se quiere calcular una raíz de la función f con una precisión δ, los pasos a seguir son los siguientes.

1. Se calcula el punto medio del intervalo $c \leftarrow (a + b)/2$.
2. Se analizan las posibles situaciones que pueden darse:
   - Si $f(c) = 0$ se ha encontrado un cero de la función y por lo tanto el proceso acaba.
   - Si $f(a)$ y $f(c)$ tienen signos contrarios quiere decir que existe una raíz de la función en la mitad izquierda del intervalo, por lo que se puede tomar como límite derecho del intervalo el valor de c, es decir, se redefine $b \leftarrow c$.
   - En otro caso $f(c)$ y $f(b)$ tienen signos contrarios por lo que existe una raíz de la función en la mitad derecha del intervalo y se puede tomar como límite izquierdo del intervalo el valor de c, es decir, se redefine $a \leftarrow c$.
3. Se repiten los pasos anteriores hasta que el proceso acabe (se encuentre una raíz de la función) o el tamaño del intervalo esté acotado por la precisión deseada ($b - a < \delta$), en cuyo caso la raíz de la función se aproxima por $(a + b)/2$.

El cáculo de raíces puede utilizarse para calcular la función inversa de una función $y = f(x)$ estrictamente creciente o decreciente en un intervalo cerrado $[a, b]$. En concreto, para cada valor y_0 el problema se resuelve calculando la raiz de la función $f(x) - y_0$. En este caso, como la función $f(x)$ es estrictamente creciente o decreciente, se puede asegurar que sólo existe una raíz.

Apéndice E

Funciones con strings

E.1. Trabajo con cadenas de caracteres

En C++ se dispone de dos tipos de representación de cadenas: El estilo de cadena de caracteres de C y la clase cadena introducida con el estandar C++. En este apéndice, nos centraremos en el primer tipo de representación.

El estilo de cadena de caracteres de C apareció en el lenguaje C y sigue siendo soportado en el lenguaje C++. En este estilo, una cadena de caracteres es un array unidimensional de caracteres terminado en un carácter nulo ('\0'). Es posible inicializar una cadena de caracteres con un string:

```
char texto1[100] = "Hola";
```

Con esta declaración, se reservan 100 caracteres para la variable cadena `texto1`. Al inicializar una variable cadena es preciso asegurarse de que la variable tiene espacio suficiente para almacenar todos los componentes de la cadena (incluido el carácter final "\0"). Así, para guardar exactamente la cadena `"Hola"` debemos declarar

```
char texto2[5] = "Hola";
```

donde reservamos cuatro caracteres para la cadena y un carácter más para el carácter de terminación de cadena '\0'. Alternativamente, con la declaración

```
char texto2[] = "Hola";
```

también se reservan 5 caracteres para la variable `texto2`.

Para operar con cadenas de caracteres, puedes utilizar las funciones que están definidas en el fichero de cabecera `string.h`, en particular las siguientes:

```
size_t strlen ( const char * cadena );
```

Descripción: Devuelve la longitud de la cadena a la que apunta *cadena* (el número de caracteres que hay desde su inicio hasta la primera aparición del carácter `'\0'`).

Entradas:

cadena (pvalor): puntero a constante. Apunta a una cadena de caracteres. A través de él no se puede modificar el contenido de la cadena.

Salidas:

Devuelve: Un entero, la longitud de `cadena`

Ejemplo:

```
void ejemploStrlen()
```

```
{
    char texto[100] = "Hola";
    cout << strlen (texto) << endl;
}
```

Al ejecutar `ejemploStrlen` se muestra por pantalla:

```
4
```

`char * strcpy ( char * destino, const char * origen );`

Descripción: Copia en la cadena apuntada por `destino` los caracteres de la cadena apuntada por `origen`, incluido el carácter de fin de cadena (`'\0'`).
Entradas:

origen (pvalor): puntero a constante. Apunta a una cadena de caracteres. A través de él no se puede modificar el contenido de la cadena.

destino (pvalor): puntero a la cadena de caracteres donde se copiará el contenido de `origen`

Salidas:

Devuelve: El puntero `destino`

Ejemplo:

```
void ejemploStrcpy()
{
    char texto[10] = "Hola";
    cout << texto << endl;
    strcpy (texto, "Adios");
    cout << texto << endl;
}
```

Al ejecutar `ejemploStrcpy` se muestra por pantalla:

```
Hola
Adios
```

`char * strcat ( char * destino, const char * origen );`

Descripción: Concatena cadenas. Añade al final de la cadena apuntada por `destino` los caracteres de la cadena apuntada por `origen`, incluido el carácter de fin de cadena (`'\0'`).
Entradas:

origen (pvalor): puntero a constante. Apunta a una cadena de caracteres. A través de él no se puede modificar el contenido de la cadena.

destino (pvalor): puntero a la cadena de caracteres donde se copiará el contenido de `origen`

Salidas:

Devuelve: El puntero `destino`

Excepciones:

a. Para evitar desbordamientos, el tamaño de la cadena destino debe ser lo suficientemente larga como para almacenar el contenido de la cadena resultante de concatenar ambas (incluyendo el carácter de fin de cadena).

Ejemplo

```
void ejemploStrcat()
{
      char texto1[20] = "Good bye", texto2[]="blue sky";
      cout << texto1 << endl;
      strcat (texto1, ", ");
      strcat (texto1, texto2);
      cout << texto1 << endl;
}
```

Al ejecutar `ejemploStrcat` se muestra por pantalla:

```
Good bye
Good bye, blue sky
```

```
char * strcmp( const char * cad1, const char * cad2 );
```

Descripción: Compara cadenas

La función comienza comparando el primer carácter de cada cadena. Si son iguales, continúa con el siguiente par hasta que los caracteres sean diferentes o hasta que se alcance el terminador de cadena.

Entradas:

cad1 (pvalor): Cadena que se compara

cad2 (pvalor): Cadena que se compara

Salidas:

Devuelve: Un valor entero que indica la relación entre ambas cadenas

<0 el primer carácter que no empareja tiene un valor menor en cad1 que en cad2

0 el contenido de ambas cadenas es el mismo

>0 el primer carácter que no empareja tiene un valor menor en cad2 que en cad1

Ejemplo

```
void ejemploStrcmp()
{
      char texto1[20] = "Good bye", texto2[]="blue sky";
      if (strcmp(texto1,texto2)==0) cout<<"iguales"<<endl;
     else cout<<"distintos"<<endl;
}
```

Al ejecutar `ejemploStrcmp` se muestra por pantalla:

```
distintos
```

Arrays y punteros

Una cadena de caracteres es un array cuyas componentes son caracteres. En el lenguaje C/C++, el identificador de un array es un puntero constante que apunta a la primera componente del array y, por tanto, su tipo es `tipo const*`, donde `tipo` es el tipo de las componentes del array. Así, la variable array puede ser asignada a cualquier otro puntero de tipo `tipo*` o `tipo const*`, consiguiendo así que el nuevo puntero apunte también a las componentes del array (no se obtiene una copia del array). Observa que la operación inversa no está permitida: no se puede asignar en una variable array el valor de otro puntero, aunque sea del mismo tipo, ya que si se permitiera la variable array dejaría de apuntar a las componentes del array.

El valor de un puntero es una dirección de memoria. Al sumar o restar un entero a un puntero se obtiene otra dirección de memoria (otro puntero), que puede ser desreferenciado con el operador `*`, que es igual a la dirección del puntero inicial más (o menos) el entero multiplicado por el tamaño en bytes (ver operador `sizeof`) del tipo de dato apuntado por el puntero. Tiene sentido utilizar esta aritmética con punteros sólo en el contexto de un array. Algunos ejemplos son:

`int x[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`
: array de 10 enteros: el valor de cada componente coincide con el de su índice.

`int *p = &x[3];`
: El puntero `p` apunta a la componente `x[3]`

`cout << p[2];`
: `p` se considera como array, la primera componente, que ocupa 4 bytes, está en la dirección de memoria directamente apuntada por `p` (es `x[3]`); las siguientes componentes son los grupos de 4 bytes situados a continuación en la memoria (`x[4]`, `x[5]`,...). Por tanto, se muestra en pantalla el valor 5.

`cout << *(p+2);`
: muestra en pantalla el valor apuntado por el puntero `p+2`. Como `p` es un puntero a datos `int`, la dirección `p+2` es `&x[3]` más `2*sizeof(int)`, donde `sizeof(int)` es el tamaño en bytes (4) de los datos de tipo `int`. Por tanto, también se muestra en pantalla el entero 5.

A continuación se muestran dos posibles implementaciones para la función `strcat` anterior: la primera utilizando únicamente notación de punteros, la segunda usando notación de arrays.

```
char* miStrcat(char* destino, const char* origen) {
    char* resultado = destino;
    while(*destino) destino++;
    do {
        *destino = *origen;
        destino++;
    } while(*origen++);
    return resultado;
}
```

```
char* miStrcat2(char destino[], const char origen[]) {
    int i = 0;
    while(destino[i]!='\0') i++; //o, simplemente,
                                 // while(destino[i]) i++;
    int j = 0;
    do {
        destino[i] = origen[j];
        i++;
    } while(origen[j++]!='\ 0'));
    return destino;
}
```

Apéndice F

Caos Determinista

F.1. Caos determinista

En este apéndice recapitularemos lo que hemos aprendido sobre la dinámica del péndulo en las tareas 3.7 y 5.9. El péndulo simple es un sistema no lineal (la aceleración depende de seno θ, que es una función no lineal de θ). En consecuencia, su ecuación diferencial es más difícil de resolver que la del oscilador armónico, lo que nos ha obligado a recurrir a métodos numéricos. Pero, a cambio, sus soluciones exhiben un comportamiento mucho más rico y complejo. Un aspecto fascinante de la dinámica del péndulo es que para determinados valores de sus parámetros aparece un comportamiento caótico.

Para caracterizar los distintos tipos de solución empezaremos presentando las nociones de *punto fijo*, *atractor* y *cuenca (o base) de atracción*.

F.2. Puntos fijos, atractores y cuencas de atracción

El lector interesado puede repoducir las figuras de esta sección procesando el fichero binario `datosEntrada_PenduloCompletoExtendido.dat` con el programa desarrollado para la tarea 5.9. Los seis primeros ejemplares de `DatosSimulacion` son los ya analizados en la tarea 5.9. Los ejemplares `07` a `15` corresponden a un péndulo sin pérdida ($\Gamma = \beta = 0$). Los ejemplares `16` a `24` corresponden a los parámetros $\alpha = 1$, $\beta = 0{,}001$, $\Gamma = 0{,}5$, $w_D = 0{,}25$. Tarda 35 minutos en ejecutarse en un ordenador con procesador Intel(R) Core(TM) i5-3460, con 3.20 GHz de frecuencia y 15 GB de RAM.

Un *punto fijo* es un punto del espacio de fases que representa un estado que permanece constante a lo largo del tiempo. ¿Cuáles son los puntos fijos del péndulo? ¿Qué ángulo y velocidad iniciales $\theta_0, \dot{\theta}_0$ debe tener un péndulo para permanecer estáticamente en ese estado? La respuesta es inmediata: en ausencia de término forzante ($\Gamma = 0$) hay sólo dos puntos fijos: $(0, 0)$ y $(-\pi, 0)$ - recuerda que suponemos una cuerda es rígida y que por convenio restringimos los valores de θ al intervalo $[-\pi, \pi)$-.

Si además $\beta \neq 0$ el primero es estable (al perturbar el estado inicial, al cabo de un tiempo el péndulo regresa a él); $(-\pi, 0)$ es claramente inestable (una mínima perturbación hace que el péndulo se aleje del estado inicial).

Un *atractor* es una región del espacio de fases donde se mueve el péndulo al llegar al estado estacionario. La *cuenca de atracción* de un atractor es la región de puntos del espacio de fases que evoluciona hacia ese atractor. La intuición de *cuenca* es la misma que

cuando hablamos de la cuenca de un río: la región cuyas aguas acaban fluyendo al mismo. Al igual que en ese caso, un atractor está contenido en su *cuenca de atracción*.

Ya hemos visto el tipo más sencillo de atractor: el punto fijo. Para $\beta \neq 0, \Gamma = 0$ la cuenca de atracción de (0, 0) es prácticamente todo el espacio de fases, salvo el punto $(-\pi, 0)$ y un segmento: la órbita de un péndulo que, dotado de cierta velocidad inicial, tiende asintóticamente a $(-\pi, 0)$ con velocidad final exactamente igual a 0. Fijado β, para cada valor de θ existe un único valor de $\dot{\theta}$ que hace que el péndulo termine en la vertical: si $\dot{\theta}$ fuera un poquito menor, el péndulo no llegaría a la vertical; si fuera un poquito mayor, el péndulo se volvería. La cuenca de atracción de $(-\pi, 0)$ es el propio punto junto con la órbita recién descrita.

La Figura F.1 ilustra los conceptos previos. La subfigura F.1a muestra una serie de estados iniciales $(\theta_0, \dot{\theta}_0)$. La subfigura F.1c muestra sus estados respectivos para los parámetros ($\alpha = 1$, $\beta = 0{,}001$, $\Gamma = 0$) para un tiempo $t > 5000T$; como era de esperar, todos los péndulos han sido atraidos por el punto fijo $(0, 0)$ salvo el que inicialmente se encuentra en $(-\pi, 0)$.

En el régimen $\beta = \Gamma = 0$ las cosas cambian. Al no haber pérdida ni aporte de energía, la energía se mantiene constante y la órbita del péndulo es periódica. La subfigura F.1b muestra las órbitas obtenidas a partir de los nueve estados iniciales mostrados en F.1a con $\alpha = 1$ (ejemplares `07` a `15` de `datosEntrada_PenduloCompletoExtendido.dat`). Cada una de ellas es un atractor. Hay, pues, infinitos atractores, cada uno de los cuales corresponde a un valor distinto de la energía. La cuenca de cada atractor es él mismo.

Hasta ahora hemos visto dos tipos de atractores: punto fijo y ciclo. En presencia del término forzante ($\Gamma \neq 0$) la situación se vuelve más compleja.

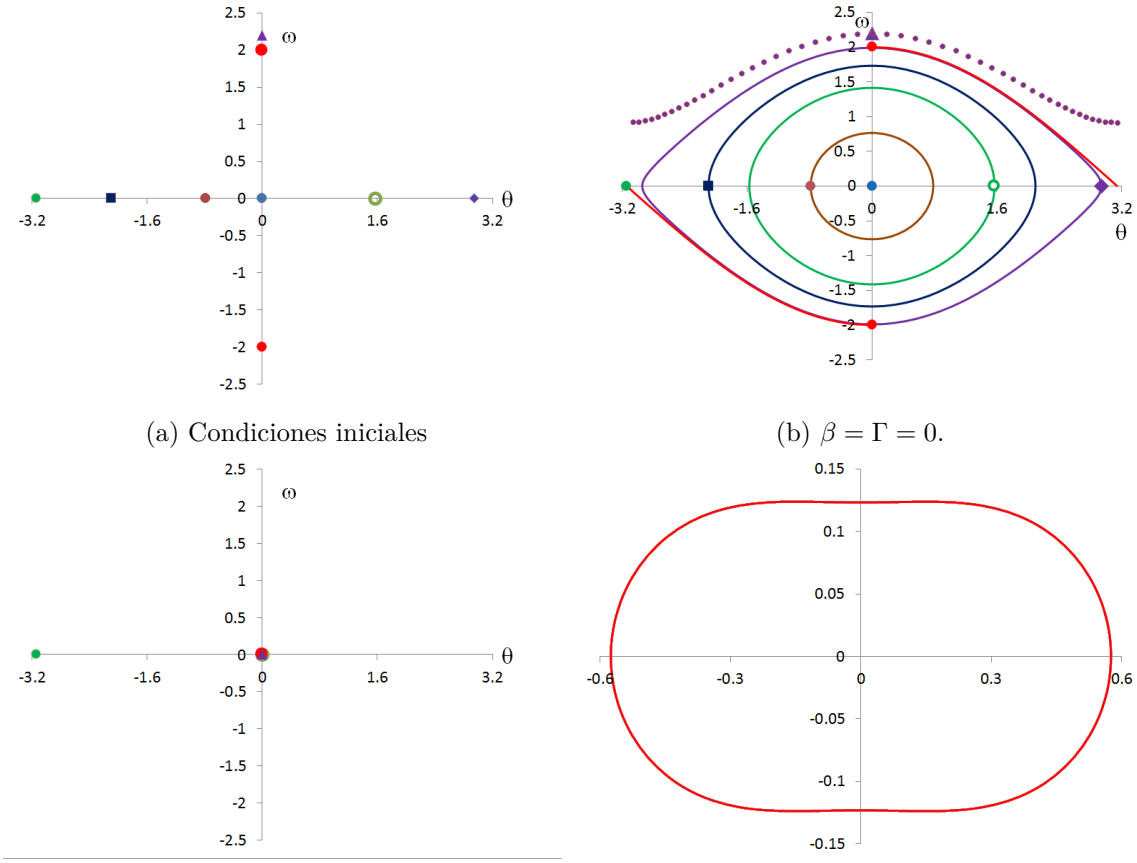
La subfigura F.1d muestra las órbitas estacionarias obtenidas a partir de los nueve estados iniciales mostrados en F.1a para los parámetros ($\alpha = 1$, $\beta = 0{,}001$, $\Gamma = 0{,}5$, $w_D = 0{,}25$) (ejemplares `16` a `24` de `datosEntrada_PenduloCompletoExtendido.dat`). Al igual que ocurre con el oscilador armónico, al cabo de un tiempo suficientemente largo (en este caso, $5000T$), el péndulo acaba oscilando a la frecuencia del término forzante. Más aún, las nueve órbitas acaban fundiéndose en una sola. Independientemente de su estado inicial, la dinámica del péndulo ha sido atraída por ella. Esta órbita es un atractor (un ciclo límite), y los nueve puntos iniciales pertenecen a su cuenca de atracción.

Lo interesante es que al cambiar los parámetros podemos encontrar comportamientos distintos. Para ($\alpha = 1$, $\beta = 0{,}001$, $w_D = 2/3$) en la tarea 5.9 observamos que, en función del valor de Γ, el periodo del péndulo puede duplicar o cuadruplicar el del término forzante. Incluso encontramos un atractor extraño para $\Gamma = 1{,}19$, indicio de comportamiento caótico (ejemplares `02` a `06` de `datosEntrada_PenduloCompletoExtendido.dat`). Las órbitas y sus respectivas secciones de Poincaré se muestran en las Figuras 5.8 a 5.12.

F.3. Caos determinista

La dinámica el péndulo es **determinista**: su estado para cualquier instante de tiempo $(\theta(t), \dot{\theta}(t))$ está determinado por su valor en $t = 0$. Dicho de otro modo: dado un estado inicial $(\theta_0, \dot{\theta}_0)$, la ecuación de Newton tiene una única solución para todo valor de t.

En la sección anterior hemos visto cómo, para $\Gamma = 0$, el comportamiento del péndulo a largo plazo no depende del valor exacto del ángulo y velocidad iniciales. Su estado acaba atrapado en un *atractor convencional* (un punto fijo, o un ciclo límite). Esto ocurre también para algunas combinaciones de valores (Γ, w_D).



(a) Condiciones iniciales

(b) $\beta = \Gamma = 0$.

(c) $\beta = 0{,}01, \Gamma = 0$.

(d) $\beta = 0{,}001, \Gamma = 0{,}5, \omega_D = 0{,}25$.

Figura F.1: Órbitas en el espacio de fases para el régimen estacionario. En todas las simulaciones $\alpha = 1$. Todas las órbitas son atraídas por un punto fijo (en F.1c) y por un ciclo límite (en F.1d), o por atractores de ambos tipos (en F.1b).

Sin embargo, esto no siempre es así: para determinados valores de los parámetros $(\alpha, \beta, \Gamma, \omega_D)$, **el comportamiento del sistema es aperiódico, y muy sensible a modificaciones infinitesimales en las condiciones iniciales, con lo que es imposible predecirlo a largo plazo**. En este caso hablamos de *caos determinista* [32]. El primero en vislumbrar este tipo de soluciones fue el extraordinario matemático Henri Poincaré, a finales del siglo XIX, al estudiar el problema de los N-cuerpos (la dinámica de un sistema de más de dos cuerpos que interactúan gravitatoriamente, como puede ser el Sistema Solar). Cualitativamente, identificamos tres aspectos clave:

i) La dinámica es aperiódica: las trayectorias en el espacio de fases no convergen a puntos fijos ni a órbitas periódicas. Son irregulares, y tienen una estructura compleja (hablamos de *atractores extraños*).

ii) El comportamiento es determinista: su irregularidad se debe a la no-linealidad de la dinámica (la ecuación de Newton es no lineal), y no a la existencia de ruido o a cualquier otro componente aleatorio en las fuerzas. En otras palabras: en el caso del péndulo, el responsable de que haya caos es exclusivamente el seno.

iii) El comportamiento del sistema es muy sensible a las condiciones iniciales. Partiendo

de dos sistemas cuyas condiciones iniciales sean *prácticamente iguales* (salvo una diferencia infinitesimal), sus dinámicas en el espacio de fases divergen muy rápidamente (en seguida se separan). Pasado un tiempo, y dado que no es posible conocer el estado inicial con una precisión infinita, no podemos aventurar en qué punto del espacio de fases encontraremos el sistema.

El lector interesado puede encontrar una buena introducción al caos determinista en el artículo *Caos*, de J.P. Crutchfield, J.D. Farmer J. D., N.H. Packard y R.S. Shaw, aparecido en *Complejidad y Caos* colección *Temas de investigación y Ciencia* 95 (2019).

F.4. Dinámica caótica en el péndulo

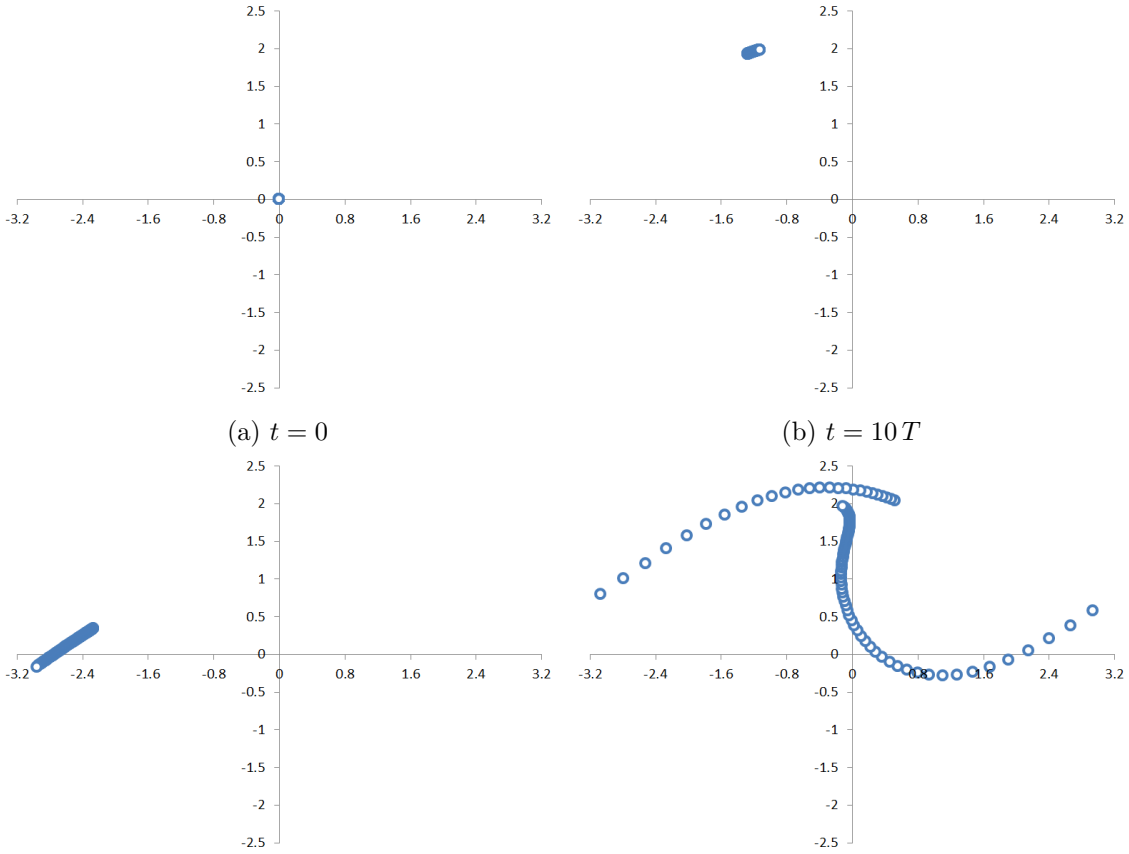
En esta sección vamos a ilustrar el comportamiento del péndulo en el caso $\Gamma = 1{,}19s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $w_D = 2/3s^{-1}$.

Comprobaremos, en primer lugar, que la dinámica del péndulo es extraordinariamente sensible al estado inicial. La Figura F.2[1] muestra la dinámica en el espacio de fases de 101 péndulos distintos para los parámetros $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-1}$, $\delta t = 10^{-3}T$.

Cada péndulo parte de un estado inicial $(\theta_0, \dot{\theta}_0)$ distinto, con $|\theta_0| \leq 0{,}000001^o, \dot{\theta}_0 = 0^o\mathrm{s}^{-1}$. En $t = 0$ los ángulos y velocidades son tan próximos que se aprecia un único punto. A medida que el tiempo avanza, el conjunto de puntos experimenta una sucesión de estiramientos y repliegues. En $t = 12T$ los distintos puntos están esparcidos por todo el atractor: la distancia entre dos péndulos, aunque inicialmente sea infinitesimal, aumenta rápidamente, y el estado del péndulo en un instante dado es completamente impredecible.

Para soluciones periódicas el atractor en la sección de Poincaré es muy simple: consta de un número finito de puntos. En el caso de comportamiento caótico, tendremos un conjunto infinito de puntos. La dinámica caótica se caracteriza por que el atractor muestra una estructura compleja. A continuación observaremos con detalle dicha estructura del *atractor extraño* con los resultados del ejemplar `04` de `datosTarea04Extendido.dat`. A medida que ampliamos sus regiones más y más encontramos estructuras más y más finas que siguen el mismo patrón, una curva que se repliega sobre sí misma una y otra vez. El atractor es un *fractal* (Figura F.3).

[1] Esta es la única Figura que no puedes reproducir con los programas que has desarrollado en las tareas de este curso, si bien a un estudiante interesado no le costaría demasiado esfuerzo escribir el programa necesario para crearla.



(a) $t = 0$ (b) $t = 10\,T$

(c) $t = 11{,}4\,T$ (d) $t = 12\,T$

Figura F.2: Dinámica en el espacio de fases para 101 péndulos para los parámetros $\alpha = 1s^{-2}, \beta = 0{,}5s^{-1}, \Gamma = 1{,}19s^{-2}, \omega_D = 2/3s^{-1}$. Para todos ellos, las condiciones iniciales son : $|\theta_0| \leq 10^{-6}$ grados sexagesimales ($|\theta_0| \leq 1{,}75\,10^{-8}$ radianes), $\dot{\theta}_0 = 0^o\,\mathrm{s}^{-1}$.

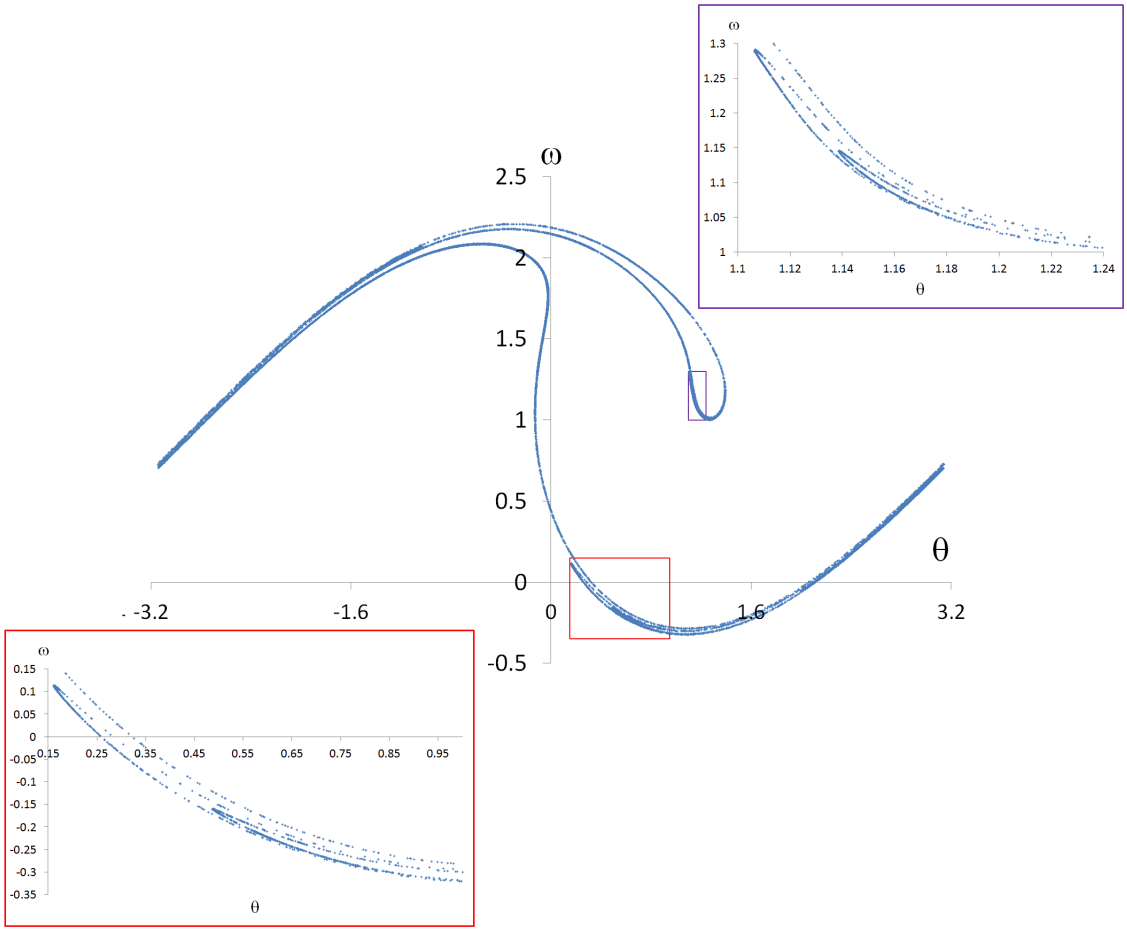


Figura F.3: Atractor para el caso $\Gamma = 1{,}19s^{-2}$, $\alpha = 1s^{-2}$, $\beta = 0{,}5s^{-1}$, $\omega = 2/3s^{-2}$. La figura muestra la ampliación de dos regiones, donde se aprecian estructuras más finas que siguen el mismo patrón que la figura principal: una curva que se repliega sobre sí misma.

Bibliografía

[1] José Manuel Abad Liñán. *Golpe en la carrera de los superordenadores*. http://tecnologia.elpais.com/tecnologia/2015/08/06/actualidad/1438861261_958750.html. Consultado el 01-09-2015. 2015.

[2] Tom M Apostol. *Introduction to analytic number theory*. Vol. 1. Springer Science & Business Media, 1976.

[3] Javier Arbonés y Pablo Milrud. *La armonía es numérica: música y matemáticas*. Rba Coleccionables, 2014.

[4] Gregory L Baker y James A Blackburn. *The pendulum: a case study in physics*. Oxford University Press, 2005.

[5] *Black Hole Sound Waves*. http://science1.nasa.gov/science-news/science-at-nasa/2003/09sep_blackholesounds. Consultado el 16-03-2016.

[6] André Blais y Louis Massicotte. «Electoral systems». En: *Comparing Democracies: Elections and Voting in Global Perspective, Sage* (1996), págs. 40-69.

[7] Steven C Chapra. «Applied numerical methods». En: *With MATLAB for Engineers and Scientists* (2012).

[8] *Código Cuenta Cliente*. http://es.wikipedia.org/wiki/Código_cuenta_cliente. Consultado el 14-03-2016.

[9] C Cohen-Tannoudji, B Diu y F Laloë. *Quantum Mechanics*. Wiley, 1977.

[10] Robert Martín Resnick Eisberg, Robert R Eisberg, Robert Resnick y col. *Física cuántica: átomos, moléculas, sólidos, núcleos y partículas*. 530.145 EIS. 1979.

[11] R Fernández-Prini y RB Dooley. «Release on the refractive index of ordinary water substance as a function of wavelength, temperature and pressure». En: *International Association for the Properties of Water and Steam* (1997), págs. 1-7.

[12] Jens Franke. *We have factored RSA640 by GNFS*. http://www.crypto-world.com/announcements/rsa640.txt. Consultado el 01-09-2015 en la url. 2005.

[13] *Geant4*. http://geant4.cern.ch/. Consultado el 16-03-2016.

[14] Robin Michael Green. *Spherical astronomy*. Cambridge University Press, 1985.

[15] *In the Dark: the Big-Bang Acoustics*. https://telescoper.wordpress.com/2012/03/12/big-bang-acoustics/. Consultado el 16-03-2016.

[16] *International Bank Account Number*. http://en.wikipedia.org/wiki/ International_Bank_Account_Numb Consultado el 14-03-2016.

[17] Tom WB Kibble. *Mecánica clásica*. Urmo, 1972.

[18] Thorsten Kleinjung. *We have factored RSA200 by GNFS*. http://www.crypto-world.com/announcements/rsa200.txt. Consultado el 01-09-2015. 2005.

[19] Thorsten Kleinjung y col. *Factorization of a 768-bit RSA modulus*. Cryptology ePrint Archive, Report 2010/006. http://eprint.iacr.org/. 2010.

[20] Arjen K Lenstra y col. *The number field sieve*. Springer, 1993.

[21] Walter Lewin. *For the Love of Physics: From the End of the Rainbow to the Edge of Time-a Journey Through the Wonders of Physics*. Simon y Schuster, 2012.

[22] Philip K Maini, Ruth E Baker y Cheng-Ming Chuong. «The Turing model comes of molecular age». En: *Science (New York, NY)* 314.5804 (2006), pág. 1397.

[23] *Mars Climate Orbiter*. http://es.wikipedia.org/wiki/Mars_Climate_Orbiter. Consultado el 14-03-2016.

[24] Richard A Muller. *Physics and technology for future presidents: an introduction to the essential physics every world leader needs to know*. Princeton University Press, 2010.

[25] Harry Ferdinand Olson. *Music, physics and engineering*. Vol. 1769. Courier Corporation, 1967.

[26] *Péndulo Simple*. https://es.wikipedia.org/wiki/Péndulo_simple. Consultado el 16-03-2016.

[27] William H Press y col. *Numerical recipes in C*. Vol. 2. Cambridge university press Cambridge, 1992.

[28] *RESTsoft*. http://gifna.unizar.es/rest/index.php/Main_Page. Consultado el 16-03-2016.

[29] *ROOT*. https://root.cern.ch/. Consultado el 16-03-2016.

[30] Ronald L Rivest, Adi Shamir y Len Adleman. «A method for obtaining digital signatures and public-key cryptosystems». En: *Communications of the ACM* 21.2 (1978), págs. 120-126.

[31] Herbert Solomon. *Geometric probability*. SIAM, 1978.

[32] Steven H Strogatz. «Nonlinear Dynamics and Chaos with Student Solutions Manual: With Applications to Physics, Biology». En: *Chemistry, and Engineering. CRC Press* (2018).

[33] Paul Allen Tipler y Gene Mosca. *Física para la ciencia y la tecnología*. Vol. 2. Reverté, 2005.

[34] Albrecht Unsöld y Bodo Baschek. *The new cosmos: an introduction to astronomy and astrophysics*. Springer Science & Business Media, 2013.

[35] *Walter Levin's last lecture*. https://www.youtube.com/watch?v=4a0FbQdH3dY. Consultado el 15-03-2016.