CAPITULO I. INTRODUCCION AL ANALISIS NUMERICO

1. ALGORITMOS Y DIAGRAMAS DE FLUJO

El hecho de que el Análisis Numérico sea tanto una ciencia como un arte es la opinión de los especialistas en este campo pero, frecuentemente es mal entendido por los no especialistas. ¿Se dice que es un arte, y a la vez una ciencia, únicamente como eufemismo, para ocultar el hecho de que el Análisis Numérico no es una disciplina suficientemente precisa para merecer el que se le considere como una ciencia? ¿Es cierto que el nombre de *análisis numérico* se emplea erróneamente, porque el significado clásico del análisis en matemáticas no es aplicable al trabajo numérico? De hecho, la respuesta a ambas preguntas es "no". Más bien la yuxtaposición de ciencia y arte se debe a un principio de incertidumbre que frecuentemente se presenta en la solución de problemas, es decir, el hecho de que para determinar la mejor forma de resolver un problema, puede ser necesaria la solución del problema en sí. En otros casos, la mejor forma de resolver un problema puede depender de un conocimiento de las propiedades de las funciones involucradas, las que no se pueden obtener ni teórica ni prácticamente.

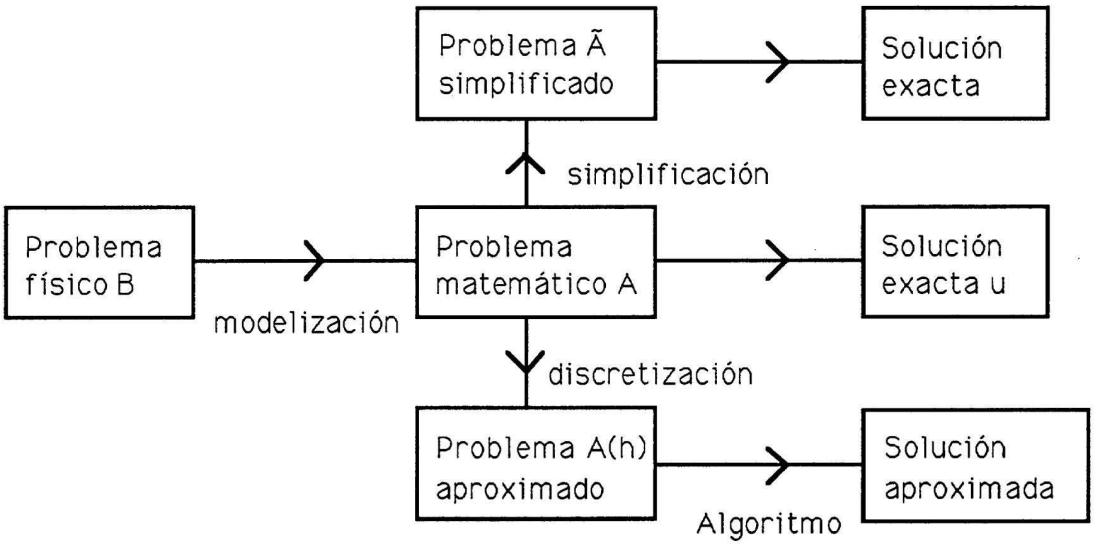
Como una ciencia, el Análisis Numérico está interesado en los procesos por los cuales pueden resolverse los problemas matemáticos, por las operaciones de la aritmética. Algunas veces esto involucrará el desarrollo de algoritmos para resolver un problema que está ya en una forma en la cual pueda encontrarse la solución por medio aritméticos. Frecuentemente involucrará la necesidad de sustituir cantidades que no pueden ser calculadas aritméticamente, por aproximaciones que permiten que sea determinada una solución aproximada. En este caso estaríamos interesados, naturalmente, en los errores cometidos en nuestra aproximación. Pero, en cualquier caso, las herramientas que usaríamos en el desarrollo de los procesos de análisis numérico, serán las herramientas del análisis matemático exacto, tan conocidas clásicamente.

Como un arte, el Análisis Numérico está interesado en la elección del procedimiento, y conveniente aplicación del mismo, "más" adecuado a la solución de un problema particular. Esto implica la necesidad de desarrollar la experiencia y con ello esperar que se desarrolle la intuición del especialista.

Así pues, el Análisis Numérico trata de diseñar métodos para aproximar, de una manera eficiente, las soluciones de problemas expresados matemáticamente. La eficiencia del método depende tanto de la precisión que se requiera como de la facilidad con la que pueda implementarse. En una situación práctica, el problema matemático se deriva de un fenómeno físico sobre el cual se han hecho algunas suposiciones para simplificarlo y para poderlo representar matemáticamente. Generalmente cuando se relajan las suposiciones físicas llegamos a un modelo matemático más apropiado pero, al mismo tiempo, más difícil o imposible de resolver explícitamente.

Ya que normalmente el problema matemático no resuelve el problema físico exactamente, resulta con frecuencia más apropiado encontrar una solución aproximada del modelo matemático más complicado que encontrar una solución exacta del modelo sim-

plificado. Para obtener tal aproximación se idea un método llamado **algoritmo**. El algoritmo consiste en una secuencia de operaciones algebráicas y lógicas que permiten la aproximación al problema matemático y se espera que también al problema físico, con una tolerancia o precisión predeterminada. Los algoritmos determinan los métodos constructivos de resolución de problema matemáticos. Un **método constructivo** es todo proceso que permite obtener la solución a un problema con la precisión que se desee, en un número finito de pasos que se pueden efectuar racionalmente. Obviamente el número de pasos requeridos dependerá de la precisión que se desee en la solución.



Como se ha dicho, los métodos constructivos en matemáticas son métodos que muestran cómo construir soluciones de un problema matemático. Por ejemplo, una demostración constructiva de la existencia de una solución de un problema, no sólo hace ver que la solución existe, si no que describe también cómo se puede determinar esa solución. Una demostración que muestra la existencia de una solución por *reducción al absurdo* no es constructiva.

Ejemplo 1. Demostrar que la ecuación cuadrática

$$x^2 + 2\ b\ x + c = 0 \tag{$I.1$}$$

con coeficientes reales b y c tales que $b^2 > c$ tiene por lo menos dos raíces reales.

Demostración constructiva. Para todo x, b y c

$$x^2 + 2\ b\ x + c = x^2 + 2\ b\ x + b^2 + c - b^2 =$$
$$= (x+b)^2 + c - b^2\ .$$

Entonces, x es una raíz de $(I.1)$ si y sólo si

$$(x+b)^2 + c - b^2 = 0\ ,$$

es decir,

$$(x+b)^2 = b^2 - c\ .$$

Y tomando la raíz cuadrada, como $b^2 - c > 0$, x será una raíz si y sólo si

$$x + b = \pm\sqrt{b^2 - c}\ ,$$

es decir,

$$x = -b \pm \sqrt{b^2 - c}\ . \qquad (I.2)$$

Luego hay dos raíces reales de $(I.1)$ y la fórmula $(I.2)$ muestra como calcularlas.

Demostración no constructiva. Sea

$$q(x) = x^2 + 2\ b\ x + c\ .$$

Supongamos que no existen raíces de $(I.1)$, con lo que no hay ceros de q. Dado que $q(x)$ es una función continua, $q(x)$ será siempre positiva o siempre negativa para cada valor de $x \in \mathcal{R}$. Ahora

$$q(-b) = b^2 - 2b^2 + c = c - b^2 < 0$$

por la condición dada sobre b y c. Entonces q tiene que ser siempre negativa. Sin embargo, para $|x|$ grandes

$$x^2 > |2\ b\ x + c|\ ,$$

con lo que resulta ser $q(x) > 0$. Así que tenemos una contradicción.

Supongamos ahora que q tiene sólo un cero. Dada la continuidad de q se sigue que $q(x) > 0$ para valores x grandes y $q(x) < 0$ para valores $-x$ grandes, o viceversa. Sin embargo, hemos visto que $q(x) > 0$ para valores $|x|$ grandes, cualquiera que sea el signo de x. Tenemos, pues, otra contradicción.

Así, esta demostración dice que existen dos raíces, pero no muestra cómo calcularlas.

Los algoritmos tienen que satisfacer los siguientes requisitos:

a) *generalidad*: un algoritmo se tiene que poder aplicar a cualquier conjunto de datos que pertenezcan a un dominio establecido;
b) *finitud*: un algoritmo tiene que estar constituído por una sucesión de instrucciones que pueden ser ejecutadas por el ordenador un número finito de veces;
c) *no ambigüedad*: un algoritmo no tiene que estar constituído por instrucciones que se contradigan o que lleguen a una paradoja.

Los valores sobre los cuales operan las instrucciones de un lenguaje de programación para producir nuevos valores pueden ser:

a) *numéricos*; ejemplo: 3, 25.4, 0.0004.
b) *lógicos*; ejemplo: Verdadero, Falso (True, False).
c) *alfanuméricos*; ejemplo: "7/1/1991", "1. Primer tema".

Nótese que los valores lógicos dados en el apartado b) son los dos únicos existentes.

Con reglas adecuadas los operadores actuán sobre las variables y las constantes para obtener valores nuevos. Una serie de símbolos usados para indicar los operadores se da

en la tabla 1. Y en la tabla 2 se dan los resultados de los operadores *and*, *or* y *xor* de las variables lógicas.

Tabla 1

Simbolo	Tipo de valor del resultado	Operación
$+$	numérico	suma
$-$	numérico	resta
$*$	numérico	multiplicación
$**$	numérico	exponenciación
$/$	numérico	división
$[\,]$	numérico	parte entera
$\sum$	numérico	suma finita
$=$	lógico	igualdad
$\neq$	lógico	no igualdad
$<$	lógico	menor que
$>$	lógico	mayor que
$\leq$	lógico	menor o igual que
$\geq$	lógico	mayor o igual que
not	lógico	cambio de T (F) en F (T)
and	lógico	(a la vez)
or	lógico	(o bien)
xor	lógico	(o bien exclusivo)

Tabla 2

A	B	not A	A and B	A or B	A xor B
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

La mayoría de las veces, los métodos de tipo constructivo directos dan lugar a algoritmos finitos, mientras que los métodos iterativos producen algoritmos infinitos (convergentes).

Ejemplo de algoritmo finito.

Un ejemplo clásico de algoritmo finito lo constituye el algoritmo de Euclides para el cálculo del *máximo común divisor* (m.c.d.) de dos números. Sean a, b dos números enteros, $a > b$. Entonces:

$$\begin{aligned}
a &= b * q_1 + r_2, && 0 < r_2 < b \\
b &= r_2 * q_2 + r_3, && 0 < r_3 < r_2 \\
r_2 &= r_3 * q_3 + r_4, && 0 < r_4 < r_3 \\
&\ldots. \\
r_{m-2} &= r_{m-1} * q_{m-1} + r_m, && 0 < r_m < r_{m-1} \\
r_{m-1} &= r_m * q_m \ .
\end{aligned}$$

Entonces $r_m = \text{m.c.d.}(a, b)$.
El algoritmo sería:

- hacer $r_0 = \text{máx}\ (a, b)$ y $r_1 = \text{mín}\ (a, b)$
- hallar $r_n =$ resto de dividir r_{n-2} entre r_{n-1}, $n = 2, 3, ...$
- cuando $r_n = 0$, parar: el m.c.d. es r_{n-1}.

Ejemplo a). $a = 37$, $b = 1218$

$1218 = 37 * 32 + 34$
$37 = 34 * 1 + 3$
$34 = 3 * 11 + 1$
$3 = 1 * 3 + 0$
$\text{m.c.d.}(37, 1218) = 1, \Rightarrow a$, b primos.

Ejemplo b). $a = 44$, $b = 420$

$420 = 44 * 9 + 24$
$44 = 24 * 1 + 20$
$24 = 20 * 1 + 4$
$20 = 4 * 5 + 0$
$\text{m.c.d.}(44, 420) = 4$.

Ejemplo de algoritmo iterativo.

Un método para el cálculo de $\sqrt{N}$ es el siguiente:
- hacer $x_0 = \frac{1+N}{2}$
- $x_{n+1} = \frac{1}{2}\left[x_n + \frac{N}{x_n}\right], \qquad n = 0, 1, ...$,

entonces $\lim_{n\to\infty} x_n = \sqrt{N}$.

Es evidente que este método es infinito (la sucesión x_{n+1} no se hace constante porque $\sqrt{N}$ no es racional en la mayor parte de los casos), por lo que debemos indicar un criterio de parada para que el algoritmo sea efectivo. Usualmente el criterio es:

$$|x_{n+1} - x_n| < \varepsilon$$

donde ε es la tolerancia permitida.

Si el algoritmo es visto como una serie temporal de operaciones, una pregunta fundamental es ¿cómo viene controlado el flujo de las operaciones? Cuando el programa en ejecución ha llegado a una instrucción particular ¿cómo determina el ordenador cuál es la próxima instrucción que tiene que ejecutar?

Se ha demostrado que sólo tres principios de control son suficientes para describir cualquier algoritmo.
El primer principio es la **noción de secuencia**; excepto que el ordenador sea instruido distintamente, él ejecuta las instrucciones de un programa secuencialmente.
El segundo principio es la **ejecución condicional** que se indica generalmente en el programa con una instrucción del tipo "*If ... then*" (si ... entonces). En la instrucción *if B then S*, B es una expresión boleana, que puede producir sólo los valores verdadero o falso,

y S es una instrucción cualquiera o grupo de instrucciones. Se evalúa B y se ejecuta S sólo si el resultado es verdadero.
El tercer principio es la **repetición** que puede ser indicado con una instrucción "*While ... do*" (mientras ... ejecuta). *While B do S* examina el valor de B y, si es verdadero, ejecuta S: los dos pasos se repiten hasta que una evaluación de B produce el valor falso. En la mayoría de los casos una evaluación de S determina el cambio del valor de B, de manera que el ciclo no continúe para siempre. Otra manera para indicar la repetición es el bucle *Do ... var = vari, varf, vars S continue*. El *Do ... continue* repite las instrucciones del bloque *S* para los valores de la variable *var* desde *vari* hasta *varf* con paso *vars*.

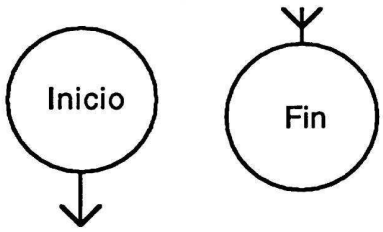
En cada lenguaje de programación, los valores sobre los cuales operan las instrucciones son las constantes y las variables (numéricas, lógicas o alfanuméricas). Además, las instrucciones fundamentales, que se pueden individualizar con un nombre o con un número (llamado *dirección*), son de los tipos siguientes:

a) **instrucciones de asignación**, que permiten asignar el valor de una expresión a una variable;
b) **instrucciones de salto incondicional**, que permiten interrumpir el orden normal de ejecución de las instrucciones de un algoritmo;
c) **instrucciones de condición**, que comparando dos valores, condicionan la ejecución de unas instrucciones en lugar de otras;
d) **instrucciones de transmisión**, que permiten transferir valores entre el mundo externo y el ordenador;
e) **instrucciones de principio de ejecución y de fin de ejecución**, que comandan el inicio o fin de la ejecución de instrucciones del algoritmo.

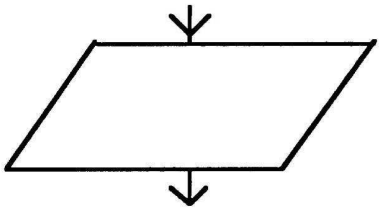
Como se ha dicho antes, excepto las instrucciones de salto incondicional, todas las otras se ejecutan en el orden en el cual están escritas, y la ejecución de una instrucción no empieza hasta que no haya acabado la ejecución de la instrucción previa.

La estructura de un algoritmo se puede representar gráficamente con un diagrama dinámico de líneas que conectan sucesiones de instrucciones del algoritmo. Cada una de esa sucesión de instrucciones es incluida en una figura y las líneas indican la interconexión entre las sucesiones. Conviene dar forma distinta a las figuras dependiendo del tipo de instrucciones que contenga (ver tabla 3). El diagrama dinámico así realizado se llama **diagrama de flujo** (flow chart).

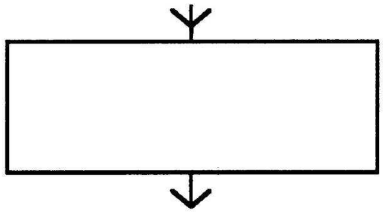
Tabla 3



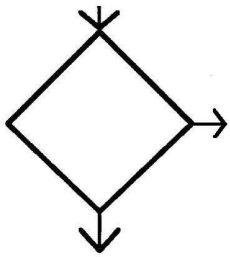
Instrucción de principio
o de fin de ejecución



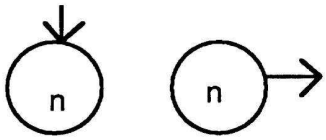
Instrucción de transmisión



Instrucción de asignación

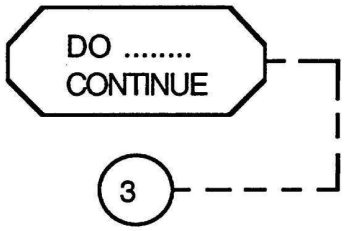


Instrucción de condición

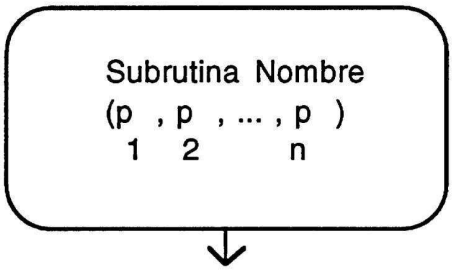


Instrucción de salto incondicional
a la instrucción número n

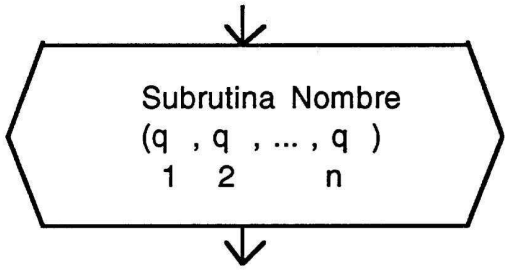
Dirección n de una instrucción



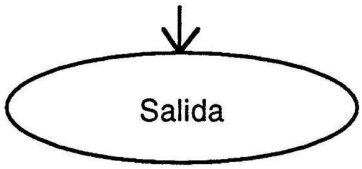
Instrucción para bucle



Instrucción de principio de una
"subroutine" llamada Nombre,
y con parámetros formales p



Instrucción de llamada de un
subrutina llamada Nombre,
y con parámetros actuales q



Instrucción de fin de una
subrutina

Ejemplo 2. Buscar las raíces reales de la ecuación

$$a\ x^2 + b\ x + c = 0\ .$$

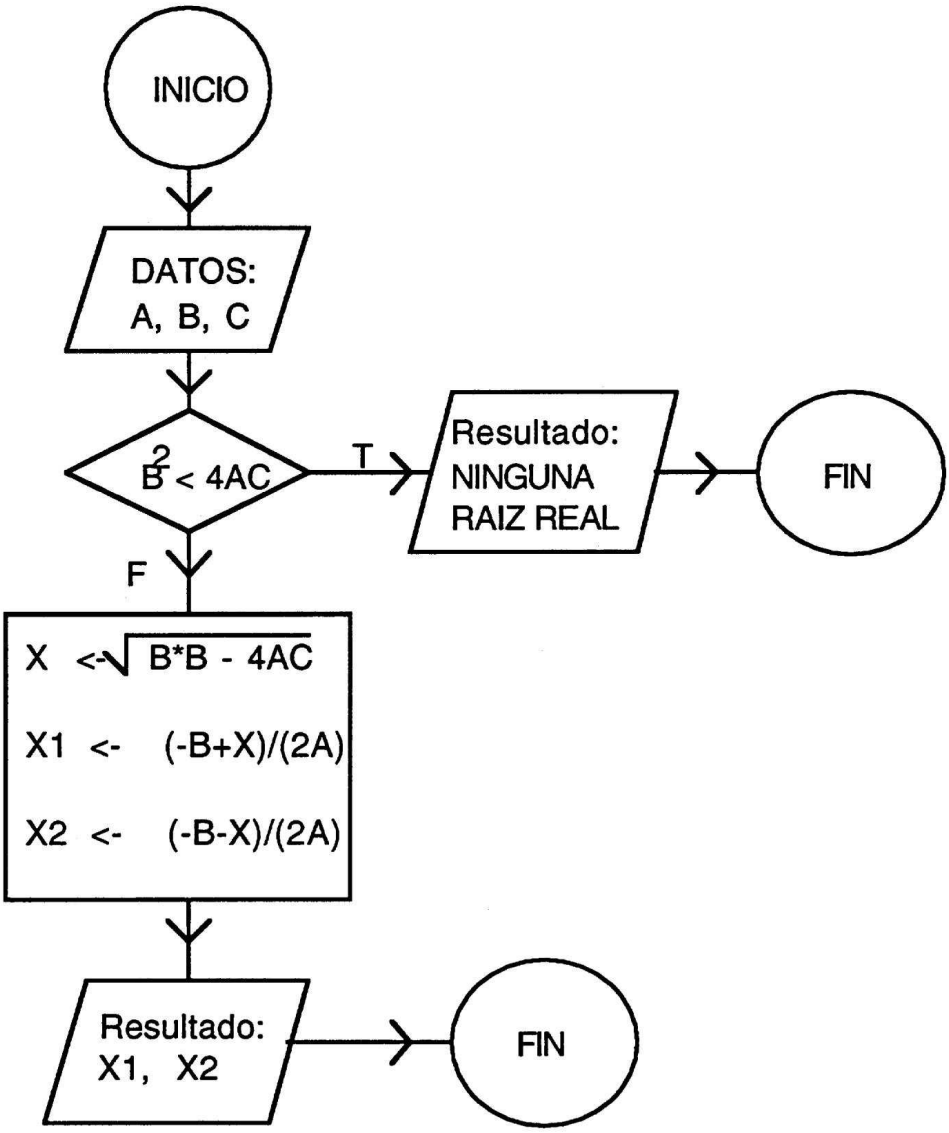
Para $a \neq 0$ las raíces son:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad\qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}\ .$$

Para que x_1 y x_2 sean reales, la expresión $b^2 - 4ac$ no puede ser negativa.

El diagrama de flujo es el siguiente:

Figura 1

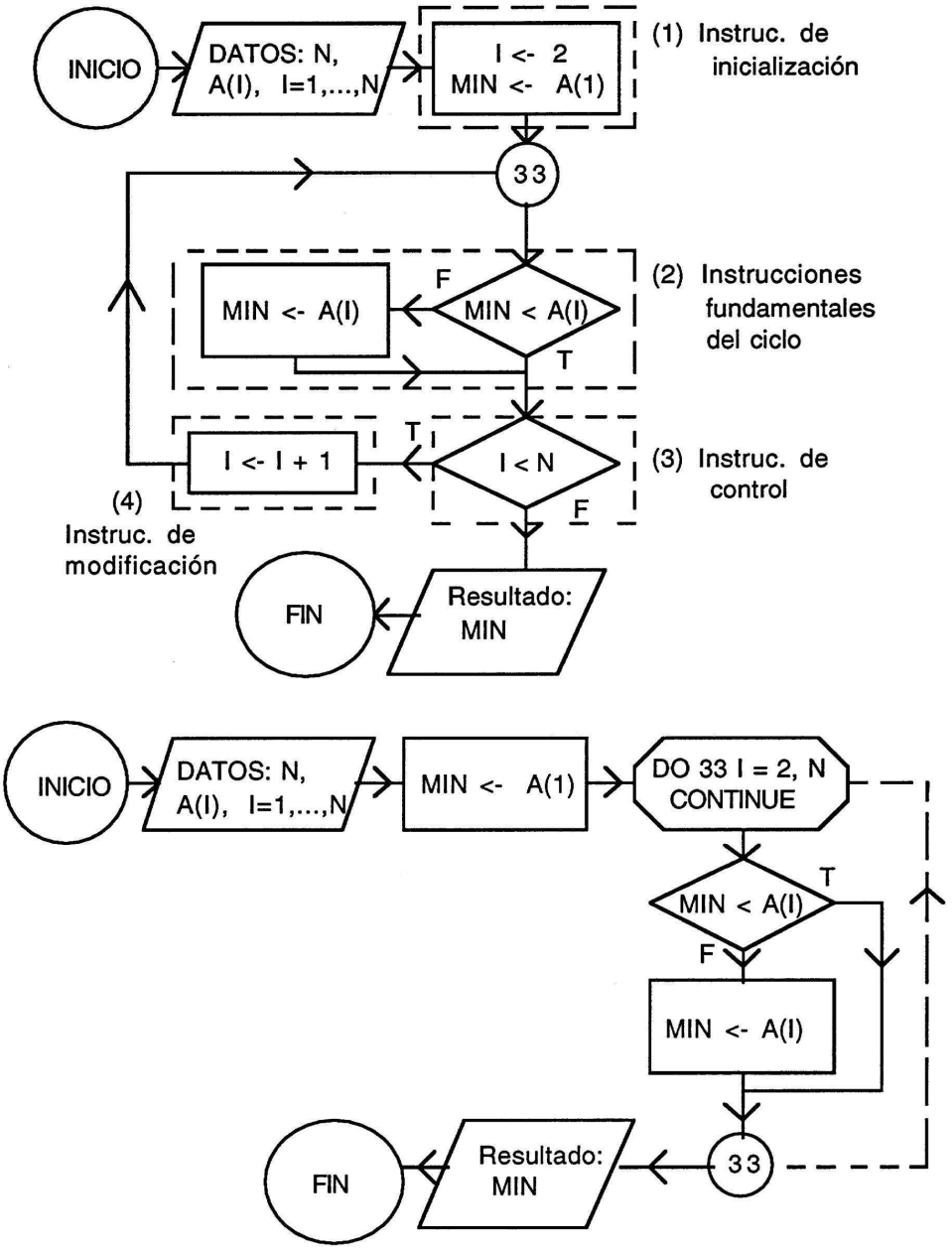


Una parte fundamental de los algoritmos es el **ciclo** (loop). Un ciclo es una sucesión particular de instrucciones que permiten que el ordenador las repita hasta que no sean verificadas unas condiciones que alteren el orden de ejecución. Un ciclo está constituido por cuatro tipos de instrucciones:

(1) instrucciones de inicialización;
(2) instrucciones fundamentales del ciclo;
(3) instrucciones de control;
(4) instrucciones de modificación.

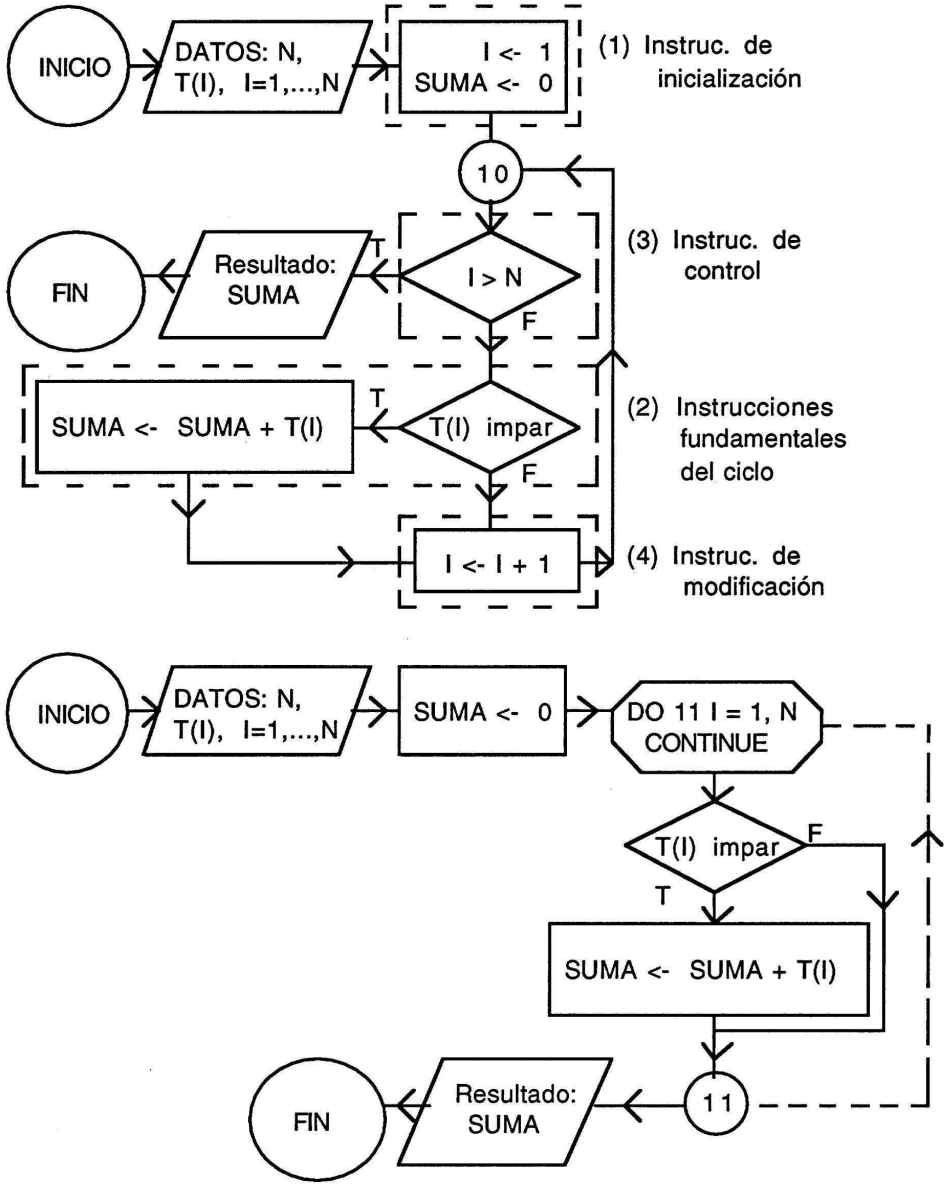
Ejemplo 3. Buscar el minimo de N números reales $a_1, a_2, ..., a_N$.

Figura 2



Ejemplo 4. Buscar la suma de los elementos impares de un vector de N elementos.

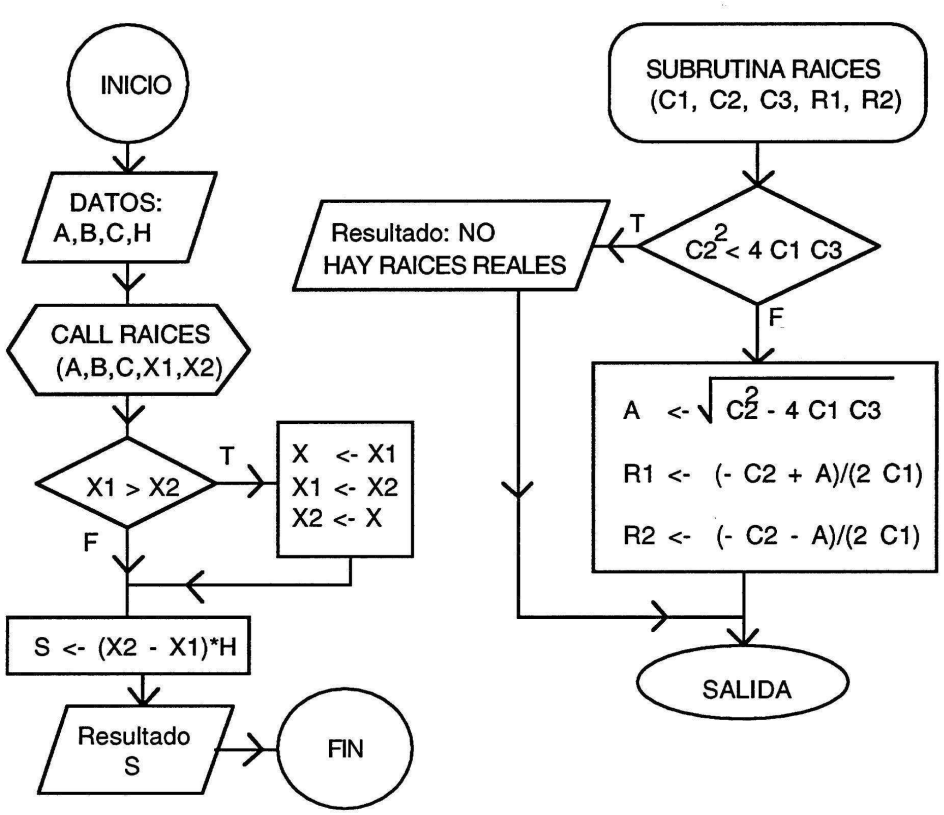
Figura 3



A menudo es conveniente que un problema caracterizado por un algoritmo A sea dividido en un número finito de problemas más sencillos, llamados subrutinas (**subroutines**). Uno de los motivos principales por los cuales es conveniente efectuar esa división en problemas más sencillos es que si se necesita resolver el mismo problema en más de un lugar del algoritmo principal, con diferentes datos, no es muy eficiente repetir las mismas instrucciones que tienen sólo nombres distintos por las variables sobre las cuales operan. Sin embargo, es más conveniente escribir un algoritmo separado que resuelva el problema parcial con **datos formales**, y organizar el problema principal originario de manera que las partes distintas se conectan a la subroutine, transmitiendo los **datos actuales**.

Ejemplo 5. Dada la ecuación $a\ x^2 + b\ x + c = 0$ con raíces x_1, x_2 reales, y $x_2 \geq x_1$, calcular el área del rectángulo de lados h y $x_2 - x_1$.

Figura 4



2. ORIGEN Y EVOLUCION DEL ANALISIS NUMERICO

Debido a la estrecha relación existente entre las diferentes ramas de la Ciencia (y en particular de las Matemáticas), no es fácil determinar dónde acaba una y empieza otra. Por ello la extensión exacta del Análisis Numérico no es conocida. De hecho, el concepto de **Análisis Numérico** no fue creado hasta 1947 en que se fundó el Instituto de Análisis Numérico en la Universidad de California. Sin embargo, el nombre parece estar asociado a aquellos temas que requieran un *procesamientos de datos*. Como la extensión de estos temas es considerable (puede ir, por ejemplo, desde la interpretación de datos médicos hasta la reserva automática de plazas de avión o gestión de una biblioteca), nos limitaremos a ciertos aspectos matemáticos de la idea.

Al principio, la mayor parte del trabajo que se efectuaba en el campo de las **Matemáticas**, inspirado por cuestiones y problemas concretos, se basaba en métodos constructivos para determinar la solución (predicciones sobre eclipses, aparición de un cometa, etc...).

El punto culminante de la utilización de los algoritmos está en Euler (1707–1783), que en los 70 volúmenes que comprenden sus trabajos incluye gran número de algoritmos y fórmulas. Los algoritmos infinitos que presenta, aparecen, normalmente, como desarrollos en serie.

Posteriormente, la perfección de los conocimientos matemáticos y la generalización de los problemas hacen que se sustituyan los razonamientos constructivos por otros de tipo lógico. Así, interesa más determinar si existe la solución a un determinado problema, que calcularlo de forma efectiva. Este proceso sigue hasta aproximadamente el año 1950. La razón del proceso de abstracción era que los algoritmos para el cálculo de las soluciones de los problemas eran, aunque finitos, irrealizables por la gran cantidad de cálculos que exigían. A partir de la segunda mitad del siglo XX, la aparición de las computadoras liberan al algoritmo de la pesadez del cálculo, lo que supone un nuevo auge para los métodos constructivos. Podríamos decir que si desde la antigüedad hasta 1945 la velocidad de cálculo se había multiplicado por 10 mediante rudimentarios artefactos (como el ábaco), desde entonces hasta ahora se ha multiplicado por un millón o más. Esto supone que 1 hora de trabajo de ordenador equivale a 200 años de trabajo de una persona, lo que permite realizar tareas inalcanzables en otros tiempos. Esto no significa que todos los algoritmos puedan ser tratados por un ordenador, pues algunos exigen más de 100 años de trabajo del ordenador actual más potente para poder ser llevados a cabo.

Como la eficiencia de un método depende de su facilidad de implementación, la elección del método apropiado para aproximar la solución de un problema está influenciada significativamente por los cambios tecnológicos en calculadoras y computadoras. El factor limitante en la actualidad es generalmente la capacidad de almacenamiento de la computadora, a pesar de que el costo asociado con los tiempos de cómputo es, desde luego, también un factor importante.

3. OBJETIVOS

El *Análisis Numérico* es *Matemática Aplicada* en el sentido de que toca problemas concretos, reales, de aplicación práctica, pero aprovechando los potentes métodos de la Matemática Pura. Por tanto no son materias opuestas, sino complementarias, lo que hace que la importancia de ambas sea cada vez mayor.

Algunos de los problemas que toca el Análisis Numérico son los siguientes:

a) **Problemas de interpolación**, en los que se sustituye una función poco manejable por otra más sencilla que cumple ciertas condiciones de coincidencia con la primera;
b) Problemas derivados de los anteriores, como pueden ser la **integración aproximada** (cuadratura, cubatura), o **derivación aproximada** de funciones poco manejables;
c) **Problemas de aproximación**, análogos a los anteriores, pero en los que se sustituye una función por otra que sea "próxima", en cierto sentido, a la primera;
d) **Resolución aproximada de ecuaciones diferenciales** tanto ordinarias como en derivadas parciales;
e) Los problemas presentados anteriormente producen, en muchos casos, **sistemas de ecuaciones lineales** con gran número de ecuaciones e incógnitas que por su coste de cálculo son irresolubles por métodos clásicos como la regla de Cramer;
f) **Problemas de tipo matricial**, (hallar valores propios, invertir matrices, etc...) relacionados con los anteriores;
g) **Problemas de optimización**, en los que se maximiza o se minimiza un funcional;
h) **Resolución aproximada de ecuaciones algebráicas y sistemas de ecuaciones no lineales**.

EJERCICIOS.

1. Construir un algoritmo (y dibujar su diagrama de flujo) que tenga como entrada un entero $n \geq 1$, $n+1$ puntos $x_0, x_1, ..., x_n$ y un punto x y que produzca como salida el producto $P = (x - x_0) * (x - x_1) * ... * (x - x_n)$.

2. A) Construir un algoritmo (y dibujar su diagrama de flujo) para buscar la suma de los elementos pares de un vector de n elementos.
B) Ejecutar el algoritmo en el caso del vector: $(2, 3, 4, 7, 8, 9, 12, 27, 38)$.

3. A) Construir un algoritmo (y dibujar su diagrama de flujo) que tomando la ecuación $a\ x^2 + b\ x + c = 0$ con raices reales x_1 y x_2, $x_1 \geq x_2$,
- calcule el perímetro y el área del cuadrado de lado $x_1 - x_2$;
- calcule la circunferencia y el área del círculo de radio $x_1 - x_2$;
B) Ejecutar los algoritmos en el caso que los coeficientes a, b, c de la ecuación sean $a = 3$, $b = -27$ y $c = 42$.

4. Construir un algoritmo (y dibujar su diagrama de flujo) que calcule la media aritmética de n valores dados $a_1, a_2, ..., a_n$.

5. A) Construir un algoritmo (y dibujar su diagrama de flujo) para buscar la suma y el producto de los elementos pares y de los elementos impares de un vector de n elementos.
B) Ejecutar el algoritmo en el caso que el vector esté constituido por los enteros entre 1 y 9 (incluídos).

6. A) Construir un algoritmo (y dibujar su diagrama de flujo) para buscar el cuadrado de los elementos pares y el cubo de los elementos impares de un vector de n elementos.
B) Ejecutar el algoritmo en el caso que el vector esté constituido por los enteros entre 1 y 9 (incluídos).

7. Representar los diagramas de flujo de los algoritmos presentados en este capítulo.