Revista internacional de métodos numéricos para cálculo y diseño en ingeniería, Vol. 2, 3, 321-336 (1986)

OPTIMIZACION DE ALGORITMOS DE CALCULO NUMERICO

LUIS GAVETE
FRANCISCO MICHAVILA
y
LUIS SANCHEZ-TEMBLEQUE

E.T.S. Ingenieros de Minas
Universidad Politécnica de Madrid

RESUMEN

Un algoritmo numérico ha de conseguir la mayor economía posible, en el proceso de solución de un problema, desde el punto de vista de:

a) La memoria requerida en la ejecución de dicho programa, lo cual incide en la capacidad del ordenador a utilizar.

b) El tiempo de ejecución del programa en el ordenador, que incide directamente en el costo de la solución de un problema.

Un algoritmo por sí solo no determina un proceso de resolución en el ordenador. Es importante optimizar la programación de algoritmos; si esto no se realiza, no es posible establecer entre ellos comparaciones.

En este artículo se indican algunos puntos a seguir en la optimización de algoritmos numéricos.

SUMMARY

It is important, to design the numerical algorithms in order to obtain the best economy in their solution process, taking account:

a) The available capacity, depending of the computer to be used.

b) The computer consuming time, which is the most important parameter influencing the computing costs.

An algorithm by itself does not determine a solution process in the computer. It is very important to choose the best algorithm's programming to compare different algorithms.

In this paper we give some ideas to be followed to obtain the very best programming of a numerical algorithm.

INTRODUCCION

En la Técnica o la Física Matemática, numerosos problemas vienen gobernados por ecuaciones en derivadas parciales, ecuaciones diferenciales ordinarias, ecuaciones algebraicas, etc., cuya solución es imposible determinar explícitamente o bien ésta es tan complicada que resulta inútil y en consecuencia, una vez asegurada la existencia y unicidad de su solución buscada, es necesario desarrollar un algoritmo que obtenga una aproximación de tal solución.

Recibido: Enero 1986

ISSN 0213-1315

La elaboración de algoritmos exige, en primer lugar un amplio conocimiento de las propiedades de las expresiones que pretendemos calcular. Así la acumulación de errores de redondeo en cada operación efectuada, puede, en ciertos casos, falsear groseramente los resultados y nos debemos limitar a "algoritmos estables", es decir poco sensibles a los errores de redondeo de modo que las soluciones aproximadas permanezcan acotadas en un cierto sentido.

El primer problema que se plantea para un método numérico es establecer su convergencia y si el método no siempre es convergente deben establecer las condiciones bajo las cuales lo es. Si el método converge lentamente, para obtener la exactitud necesaria puede requerirse un elevado número de cálculos. Así la convergencia insuficientemente rápida es uno de los criterios de penalización de un algoritmo dado. No obstante y al considerar la conveniencia de realizar un tipo determinado de cálculos, a veces entre dos algoritmos, si los cálculos a que conduce son más simples, se puede preferir aquél cuya velocidad de convergencia sea relativamente menor.

Elegido un método numérico, hay que trasladarlo al lenguaje algorítmico, para lo cual se emplean las descomposiciones lógicas[1] y los organigramas[2 y 3]. Un algoritmo no determina, por sí solo, un proceso único de solución en el ordenador. Vamos en lo que sigue a analizar un modo de optimizar la programación de un algoritmo dado. Se completará así, en el proceso de resolución de un problema que va desde su análisis numérico hasta la obtención de resultados, la optimización económica de los cálculos buscada.

Para minimizar la memoria requerida por un programa y el tiempo de su ejecución, utilizaremos una serie de parámetros, quedarán de una forma analítica, el grado de optimización del programa escrito en un lenguaje determinado.

La descomposición lógica de un problema[1] permite fácilmente definir los puntos a partir de los cuales ya no se vuelven a utilizar ciertas variables para un determinado fin. Gracias a lo cual reduciremos el número de posiciones de memoria a utilizar.

En el caso de almacenamiento de una matriz, conviene que éste sea lo más ajustado posible. Así en el caso de matrices diagonales, en banda o simétricas se debe acudir a un almacenamiento en vector columna o bien en matriz compacta, con el fin de no guardar posiciones de memoria para términos nulos o iguales a sus simétricos. Esta es una buena práctica en el empleo del método de elementos finitos[4].

Asimismo el resultado de una operación entre matrices puede almacenar donde se hallaba una de las iniciales. Por ejemplo, en un producto de dos matrices cuadradas de dimensión N;

$$[C] = [A] * [B]$$

se puede reducir memoria utilizando el siguiente algoritmo:

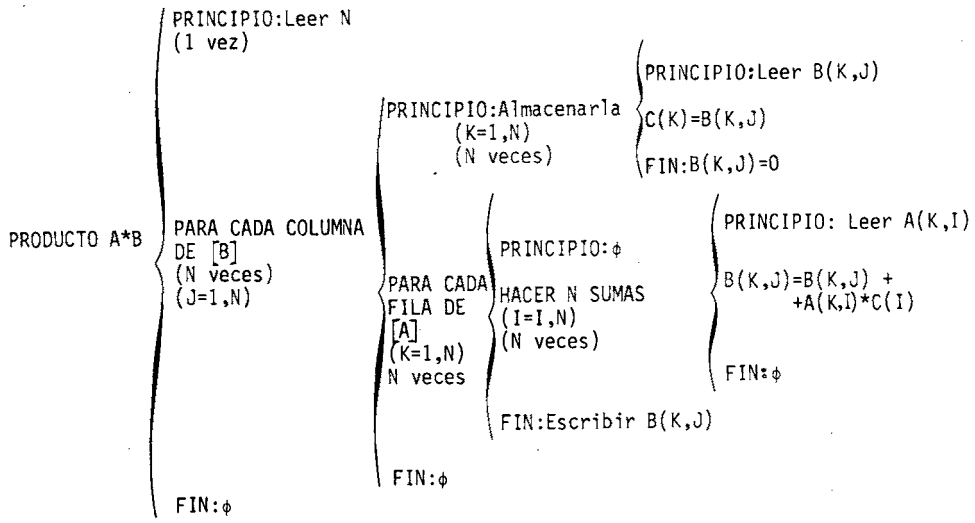
PARA CADA COLUMNA DE $[B]$

* Almacenar en un vector auxiliar $\{C(K)\}$

PARA CADA FILA DE $[A]$

* Almacenar sumatorio producto por vector $\{C(K)\}$
* Almacenar resultados en lugar correspondiente $[B]$

Es evidente el ahorro de memoria que será de $Nx(N\text{-}1)$ posiciones. La descomposición lógica del problema será:



El encadenamiento de programas y la utilización de subrutinas es importante en un intento de ahorro de la memoria ocupada por el propio programa, así como el empleo de archivos. La descomposición lógica proporciona una visión del conjunto memoria más archivos.

En general el ahorro de memoria, lleva consigo una mayor complicación en la lógica de la programación y por ello cuanto más optimizado esté un programa en cuanto a sus almacenamientos, más difícil será de entender y de hacer modificaciones sobre él.[1, 4, 5]

TIEMPO DE EJECUCION

A partir de la descomposición lógica de un problema complejo, se puede contabilizar fácilmente el número de operaciones básicas a realizar en cada estructura, teniendo en cuenta el tipo de operación y el tipo de variable a operar. Se puede asimismo, dar un peso a cada tipo de operación básica proporcional al trabajo que le cueste al ordenador realizarla. No es igual hacer, por ejemplo, sumas que multiplicaciones, como tampoco es igual hacer sumas cuyos sumandos sean variables simples, o cuyos sumandos sean de tipo matricial.

De esta forma, conociéndolo mediante el análisis, el número de operaciones de un tipo determinado a realizar, y el peso que tiene esa operación su producto nos dará el tiempo que emplea el ordenador en realizar esas operaciones. Para reducir el número de operaciones se deben obviar aquellas cuyo resultado vaya a ser cero y que de antemano conozcamos. Así por ejemplo, en el empleo del algoritmo de Gauss, con pivote parcial[7], no se opera con la parte de columna que se quiere anular. Por otra parte el método de Gauss con pivote total al emplear también cambios de columnas penaliza enormemente el número de operaciones.

Con la ayuda de la descomposición lógica de un algoritmo de cálculo numérico, se obtiene fácilmente el número de comparaciones que se realizan en cada estructura. Este

parámetro tiene gran importancia a la hora de optimizar los métodos de ordenación.[6] Hay que hacer notar que para cada tipo de comparación existe un peso determinado relativo al tiempo de ejecución de esa comparación.

En cuanto a los pasos de bucle y su tipo los podemos obtener asimismo, a partir de la descomposición lógica. No es igual realizar un paso en un bucle externo que en otro interno. Por otro lado, es interesante poder reducir el número de pasadas de un bucle. Asimismo es importante conocer el número de funciones y su tipo. Nos referimos al empleo de funciones predefinidas que calculen el valor absoluto, la parte entera, etc.

METODOLOGIA DE LA OPTIMIZACION

Planteado el problema de cálculo numérico y elegido el algoritmo que se ha de emplear para su resolución, se programará éste.

La forma de hacerlo es utilizar su descomposición lógica que se puede variar para tratar de optimizar la programación. Para ello se utilizarán ciertos parámetros cuyos valores nos darán una idea previa de donde está la mayor dificultad en la ejecución.

Los parámetros son los que se describen en la tabla siguiente, y en ella se muestra también, el tiempo de su ejecución en un ordenador SPERRY P.C., usando lenguaje BASIC y utilizando 100.000 repeticiones en cada caso.

La columna de PESOS muestra la relación entre los tiempos de ejecución de cada parámetro y el paso de bucle y es justamente la tabla de pesos que tendremos que asignar en nuestros recuentos. (Ver Tabla 1).

Se ha comprobado la existencia de anomalías en los tiempos de ejecución de algunas combinaciones de operaciones simples. Se explican a continuación algunos casos:

a) Bucles concatenados.– Se ha pasado el caso de dos bucles concatenados, calculando el tiempo de ejecución del paso de bucle interno.

```
RUN
CASO DE PRUEBA ? DOS BUCLES (I=2 TO N) (J=I - 1 TO 1)
NUMERO DE ELEMENTOS = ? 200
EL TIEMPO DE CPU EN SG.ES = 21,08985
```

$$t_{bucle} = \frac{21{,}08985 - 199 \times 9{,}9031 \times 10^{-4} - 199 \times 8{,}8879 \times 10^{-4}}{19900} = 10{,}4 \times 10^{-4} \text{ sg.}$$

$$\text{peso} = \frac{10{,}4}{9{,}9031} = 1{,}0502$$

TABLA 1

TIPO DE OPERACION	CASO DE PRUEBA	TIEMPO OBTENIDO(SG.)		TIEMPO UNITARIO(x10-4SG.)		PESO
BUCLE	-	99.03125	a	9.9031	a	1.000
ASIGNACION SIMPLE	A=I	201.8984	b	10.2867	b-a	1.039
" MATR.SIMPLE	A(I)=A(J)	355.5781	c	25.6547	c-a	2.591
" " DOBLE	A(I,J)=B(K,L)	533.4883	d	43.4457	d-a	4.387
" MIXTA SIMPLE	A(I)=H	284.5117	e	18.548	e-a	1.873
" " DOBLE	A(I,J)=H	372.5586	f	27.353	f-a	2.762
SUMA SIMPLE	A=I+I	285.0078	g	8.3109	g-b	0.839
" MATR.SIMPLE	A(I)=B(J)+C(K)	519.8672	h	23.5350	h-e	2.376
" " DOBLE	A(I,J)=A(K,L)+A(M,J)	773.129	i	40.0570	i-f	4.045
" MIXTA SIMPLE	A(I)=A(J)+H	438.8009	j	15.4289	j-e	1.558
" " DOBLE	A(I,J)=A(K,L)+H	611.1524	k	23.8594	k-f	2.409
RESTA SIMPLE	A=I-I	290.7774	l	8.8879	l-b	0.898
" MATR.SIMPLE	A(I)=B(J)-C(K)	525.0274	m	24.05157	m-e	2.429
" " DOBLE	A(I,J)=A(K,L)-A(M,J)	776.7618	n	40.4203	n-f	4.082
" MIXTA SIMPLE	A(I)=A(J)-H	444.2891	ñ	15.9777	ñ-e	1.614
" " DOBLE	A(I,J)=A(K,L)-H	614.2305	o	24.1612	o-f	2.441
PRODUCTO SIMPLE	A=I*I	303.2422	p	10.1344	p-b	1.024
" MATR.SIMPLE	A(I)=A(J)*A(K)	528.9883	q	24.4476	q-e	2.469
" " DOBLE	A(I,J)=A(K,L)*A(M,N)	789.7735	r	41.7214	r-f	4.213
" MIXTO SIMPLE	A(I)=A(J)*H	456.4063	s	17.1891	s-e	1.736
" " DOBLE	A(I,J)=A(K,L)*H	631.6797	t	25.9121	t-f	2.617
DIVISION SIMPLE	A=I/I	297.6953	u	9.5797	u-b	0.968
" MATR.SIMPLE	A(I)=A(J)/A(K)	524.6094	v	24.0098	v-e	2.425
" " DOBLE	A(I,J)=A(J,K)/A(K,L)	781.0938	x	40.8535	x-f	4.125
" MIXTA SIMPLE	A(I)=A(J)/H	454.2969	y	16.9785	y-e	1.715
" " DOBLE	A(I,J)=A(J,K)/H	621.7969	z	24.9238	z-f	2.517

TABLA 1 (Cont.)

TIPO DE OPERACION	CASO DE PRUEBA	TIEMPO OBTENIDO(SG.)		TIEMPO UNITARIO($x10^{-4}$SG.)		P E S O
GO TO	-	130.6094	A	3.15781	A-a	0.319
DEFINICION FUNCION	DEF FN F(X)=X	201.018	B	10.1987	B-a	1.0298
CALCULO FUNC.DEFINIDA.	H = FN F(X)	450.2734	C	24.8375	C-b	2.508
VALOR ABS.SIMPLE	H=ABS(I)	237.3044	D	3.5406	D-b	0.358
" "MATR.DOBLE	H=ABS(A(I,J))	402.5764	E	20.0678	E-b	2.027
PARTE ENTERA SIMP.	H=INT(I)	263.6714	F	6.1773	F-b	0.624
COMPARAC.IGUAL SIMPLE	IF X=Y	285.332	G	18.63	G-a	1.881
COMPARAC.IGUAL MAT. SIMPLE	IF A(I)=B(J)	426.504	H	32.747	H-a	3.307
COMPARAC.IGUAL MAT. DOBLE	IF A(I,J)=B(I,J)	589.297	I	49.027	I-a	4.951
COMPARAC.IGUAL MIXTA SIMPLE	IF A(I)=H	350.449	J	25.142	J-a	2.539
COMPARAC.IGUAL MIXTA DOBLE	IF A(I,J)=H	434.1602	K	33.513	K-a	3.384
COMP.<ó> SIMPLE	IF X>Y	273.8086	L	17.478	L-a	1.765
" " MAT.SIMP.	IF A(I)>B(J)	428.1446	M	32.911	M-a	3.324
" " " DOBLE	IF A(I,J)>B(I,J)	592.9102	N	49.388	N-a	4.987
" " MIX.SIMP.	IF A(I)>H	351.504	Ñ	25.247	Ñ-a	2.550
" " " DOBLE	IF A(I,J)>H	432.801	O	33.377	O-a	3.371
COMP.≤ó≥ SIMPLE	IF X≥Y	280.4102	P	18.138	P-a	1.832
" " MAT.SIMP.	IF A(I)≥B(J)	435.5665	Q	33.654	Q-a	3.398
" " " DOBLE	IF A(I,J)≥B(I,J)	592.598	R	49.357	R-a	4.984
" " MIX.SIMP.	IF A(I)≥H	360.5469	S	26.152	S-a	2.641
" " " DOBLE	IF A(I,J)≥H	441.0938	T	34.206	T-a	3.454
COMP.<> SIMPLE	IF X<>Y	288.652	U	18.962	U-a	1.915
" " MAT.SIMP.	IF A(I)<>B(J)	427.578	V	32.855	V-a	3.318
" " " DOBLE	IF A(I,J)<>B(I,J)	592.949	X	49.392	X-a	4.988
" " MIX.SIMP.	IF A(I)<>H	363.651	Y	26.462	Y-a	2.672
" " " DOBLE	IF A(I,J)<>H	447.651	Z	34.862	Z-a	3.521

Al tiempo obtenido se le restan los *(N-1)* pasos del bucle externo más las *(N - 1)* restas simples que se producen en el bucle interno. El resultado se divide por el número total de pasos del bucle interno $\frac{N}{2}$ *(N - 1)*.

b) **Salto de un bucle interno a otro externo.**– Se han pasado cinco casos de prueba con salto condicional desde un bucle interno al bucle externo, variando el tipo de comparación que produce el salto.

Se ha tomado como valor del peso, la media aritmética de los valores obtenidos.

$X>=I$	4.63562
$A(I)\geqslant J$	4.753663
$A(I,J)=A(K,L)$	5.0907
$H>A(I)$	4.6588
$H>A(I,J)$	4.9556

APLICACION A UN METODO DE INTEGRACION NUMERICA

Vamos a aplicar los resultados anteriores a una fórmula de integración numérica compuesta de Simpson, que es de tipo interpolatorio de la familia Newton-Cotes. En un intervalo $[a,b]$, siendo $x_1 = a$, $x_{N+1} = b$:

$$\int_a^b f(x)dx = \sum_{\substack{j=1 \\ j,\ \text{impar}}}^{N-1} \int_{x_j}^{x_{j+2}} f(x)dx = \sum_{\substack{j=1 \\ j,\ \text{impar}}}^{N-1} \frac{h}{3}$$

$$[f(x_j) + 4f(x_{j+1}) + f(x_{j+2})] \quad ; \quad h = x_{j+1} - x_j$$

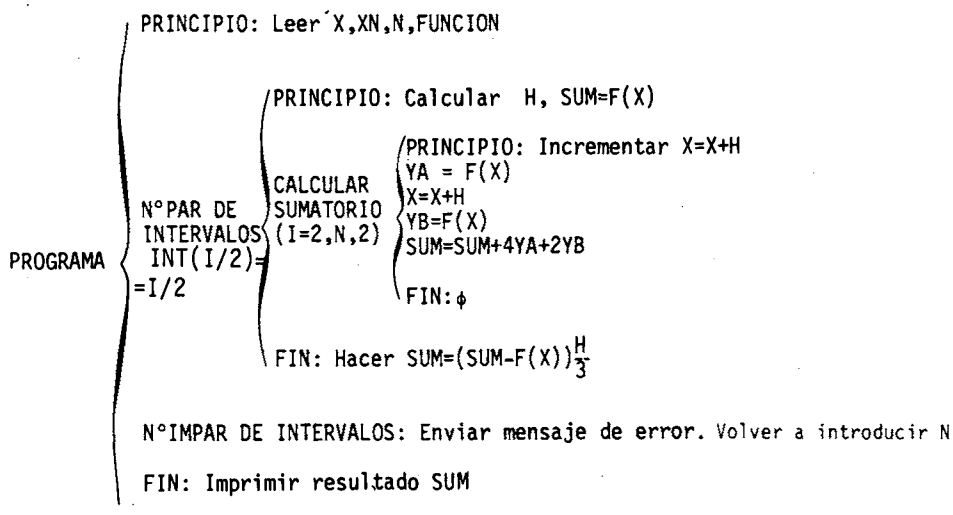
En[7] se establecen sus propiedades y algoritmos de programación, con la descomposición lógica siguiente:

```
PROGRAMA
  PRINCIPIO: Introducir X ,XN,N , FUNCION
  * N° PAR DE INTERVALOS (INT(I/2)=I/2)
      PRINCIPIO:Calcular H, SUM=F(X)
      CALCULAR SUMATORIO (N-1)vec. I=2,N
          PRINCIPIO: Incrementar X=X+ H
          Y=FUN(X)
          * TERMINO PAR: SUM=SUM+4Y
          * TERMINO IMPAR:SUM=SUM+2Y
          FIN:φ
      X=XN
      FIN : Hacer SUM = (SUM+F(XN))H/3
  * N° IMPAR DE INTERVALOS: Enviar mensaje de error. Volver a introducir N
  FIN: Imprimir resultado:SUM
```

Podemos obtener otra versión que calcule el término

$$SUM = SUM + 4y_i + 2y_{i+1}$$

directamente, y que por tanto no distinga términos pares e impares. Para ello utilizaremos la descomposición lógica:



A la vista de las descomposiciones lógicas de las dos versiones, se analizan a continuación las partes de las mismas que presentan diferencias:

TABLA 2

METODO DE SIMPSON DE INTEGRACION NUMERICA			VERSION-1	VERSION-2
N° DE POSICIONES DE MEMORIA DE LAS VARIABLES.			8	9
OPERACIONES ELEMENTALES	N° de operaciones algebraicas	SUMAS SIMPLES	2N-1	2N
		RESTAS SIMPLES	—	1
		PRODUCTOS SIMPLES	N	N+1
		DIVISIONES SIMPLES	2N-1	1
	N° Com.	COMP. IGUALDAD SIMPLE	N-1	—
	N° Asig.	ASIGNACION SIMPLE	3N-1	$\frac{5N}{2}+1$
	N° PASOS DE BUCLE		N-1	N/2
	SALTO INCONDICIONAL (GO TO)		N/2	—
	CALCULO FUNCION DEFINIDA		N	N+1
	PARTE ENTERA SIMPLE		N-1	—

TABLA 3

VERSION 1 N=1000	TERMINO GENERAL	Nº TOTAL OPERAC.	P E S O	T O T A L	TIEMPO REAL(Sg.)	
POSIC.DE MEMORIA	8	——	—	—		
SUMAS SIMPLES	2N-1	1999	0.839	1677.161		
RESTAS SIMPLES	——	——	0.898	——		
PRODUC.SIMPLES	N	1000	1.024	1024		
DIVIS.SIMPLES	2N-1	1999	0.968	1935.032		
COMP.IGUALD.SIMP.	N-1	999	1.881	1879.119		
ASIGNAC.SIMPLE	3N-1	2999	1.039	3115.961		
PASOS DE BUCLE	N-1	999	1	999		
GO TO	N/2	500	0.319	159.5		
CALCULO FUNC.DEF.	N	1000	2.508	2508		
PARTE ENTERA SIMP.	N-1	999	0.624	623.376		
			TOTAL	13921.149		
		TIEMPO EN SG. (x9.9031x10-4)		13.786	13.887	Error = = 0,73%

TABLA 4

VERSION 2 N=1000	TERMINO GENERAL	N°TOTAL OPERAC.	P E S O	T O T A L	TIEMPO REAL(Sg.)	
POSIC.DE MEMORIA	9	——	—	—		
SUMAS SIMPLES	2N	2000	0.839	1678		
RESTAS SIMPLES	1	1	0.898	0.898		
PRODUCTOS SIMPLES	N+1	1001	1.024	1025.024		
DIVISIONES SIMPLES	1	1	0.968	0.968		
ASIGNAC, SIMPLE	$\frac{5N}{2}$ +1	2501	1.039	2598.539		
PASOS DE BUCLE	N/2	500	1	500		
CALCULO FUNC.DEF.	N+1	1001	2.508	2510.508		
			TOTAL	8313.937		
		TIEMPO EN SG (x9.9031x 10-4)		8.2334	8.2891	Error= = 0.68%

Como vemos esta versión mejora sensiblemente el tiempo de ordenador a costa de ocupar una posición más de memoria.

APLICACION A LA RESOLUCION DE UN SISTEMA DE ECUACIONES POR EL METODO DE GAUSS-JORDAN.

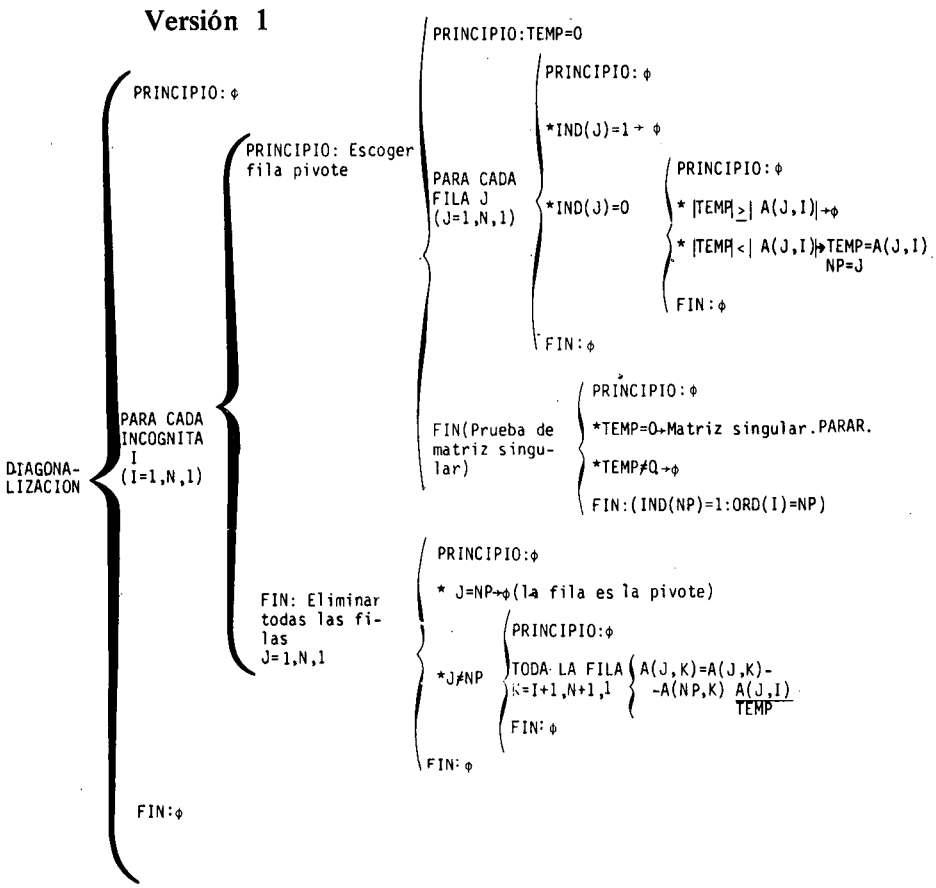
El método de Gauss-Jordan básicamente se diferencia poco del clásico de Gauss. Consiste en transformar el sistema de ecuaciones de modo que al final se llegue a una matriz de coeficientes diagonal, en vez de triangular superior. Ello se traduce en el número de veces a realizar cada paso de eliminación para cada variable x_i; mientras en el método de Gauss se elimina x_i de todas las filas *"j"* tales que $j>i$, en el de Gauss-Jordan se elimina x_i de todas las filas *"j"* excepto la correspondiente al valor $j=i$.

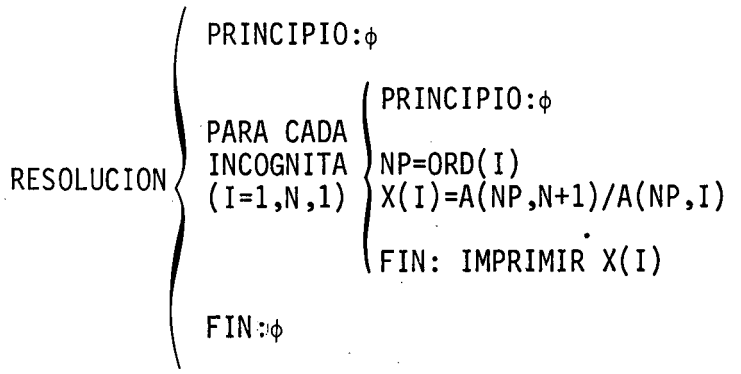
En[7] se establece la siguiente versión de aplicación de dicho algoritmo:

Versión 1.–

Es un método de elección de fila pivote sin intercambiar filas. (pivote parcial)

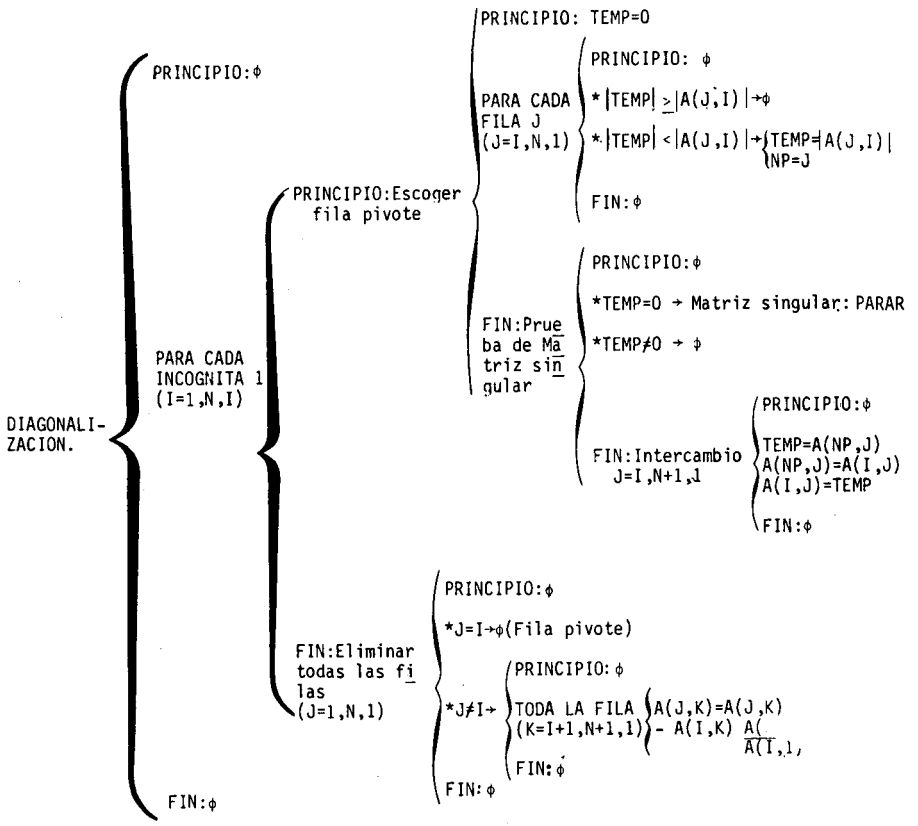
Para ello hacen falta dos vectores. En uno de ellos, *IND*(I), se almacena un "1" en el lugar correspondiente a la fila escogida como pivote. Así se sabrá, en posteriores procesos que esa fila ya ha sido utilizada como pivote. En el otro vector, *ORD*(I) se almacena el orden en que cada fila ha sido utilizada. Ello será necesario posteriormente en el proceso de resolución.





En[6] se establece este método a partir del método de Gauss con pivote parcial desarrollado en[7], constituyendo otra versión.

Versión 2



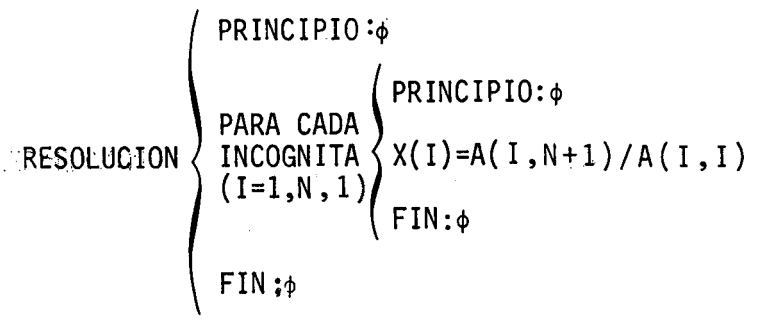


TABLA 5

METODO DE GAUSS-JORDAN DE RESOLUCION DE SISTEMAS DE ECUACIONES.				VERSION-1	VERSION-2
Nº DE POSICIONES DE MEMORIA DE LAS VARIABLES.				N^2+4N+6	N^2+2N+6
OPERACIONES ELEMENTALES	Nº OPERACIONES ALGEBRAICAS	SUMAS SIMPLES		$(2N-1)N$	$2N^2$
		DIVISIONES SIMPLES		$(N^2-1)N/2$	———
		RESTA MIXTA DOBLE		$(N^2-1)N/2$	$(N^2-1)N/2$
		PRODUCTO MATRICIAL DOBLE		$(N^2-1)N/2$	$(N^2-1)N/2$
		DIVISION MATRICIAL DOBLE		N	N
		DIVISION MIXTA DOBLE		———	$(N^2-1)N/2$
	Nº COMPARACIONES	COMPARACION IGUALD. SIMPLE		N^2+N	$N(N+1)$
		COMPARAC.IGUALD.MIXTA SIMPLE		N^2	——
		COMPARACION > SIMPLE		$(N+1)N/2$	$(N+1)N/2$
	Nº ASIGNACIONES	ASIGNACION SIMPLE		$(N^2+3N)/2$	$(N^2+3N)/2$
		ASIGNACION MATRICIAL DOBLE		——	$(N+3)N/2$
		ASIGNACION MIXTA SIMPLE		4N	N
		ASIGNACION MIXTA DOBLE		$(N+1)N^2/2$	$(N^2+3N+6)N/2$
	PASOS DE BUCLE		BUCLES EXTERNOS	2N	2N
			BUCLES INTERNOS	$\frac{N}{2}(N^2+4N-1)$	$\frac{N}{2}(N^2+4N+3)$
	VALOR ABSOLUTO SIMPLE			$(N+1)N/2$	$(N+1)N/2$
	VALOR ABSOLUTO DOBLE			$(N+1)N/2$	$(N+1)N/2$

TABLA 6

VERSION 1 N=10		TERMINO GENERAL	N° TOTAL OPERAC.	P E S O	T O T A L	TIEMPO REAL (Sg.)
POSIC.DE MEMORIA		N^2+4N+6	146	—	—	
SUMAS SIMPLES		(2N-1)N	190	0.839	159.41	
RESTAS SIMPLES		—	—	—	—	
RESTA MIXT.DOBLE		$(N^2-1)N/2$	495	2.441	1208.295	
PROD.MATRIC. DOBLE		$(N^2-1)N/2$	495	4.213	2085.435	
DIV.MATRIC.DOBLE		N	10	4.125	41.25	
DIV.SIMPLE		$(N^2-1)N/2$	495	0.968	479.16	
COMP.IGUALD.SIMP.		N^2+N	110	1.881	206.91	
COMP.IGUALD.MIX. SIMPLE		N^2	100	2.539	253.9	
COMPAR. > SIMPLE		(N+1)N/2	55	1.765	97.075	
ASIGNAC.SIMPLE		$(N^2+3N)/2$	65	1.039	67.535	
ASIG. MIXTA SIMPLE		4N	40	1.873	74.92	
ASIG. MIXTA DOBLE		$(N+1)N^2/2$	550	2.762	1519.1	
PASO BUCLE	BUCLES EXTER.	2N	20	1	20	
	BUCLES INTER.	$\frac{N}{2}(N^2+4N-1)$	695	1.0502	729.889	
VALOR ABSOLUTO SIMPLE		(N+1)N/2	55	0.358	19.69	
VALOR ABSOLUTO DOBLE		(N+1)N/2	55	2.027	111.485	
				TOTAL	7074.054	
TIEMPO EN SG.($x9.9031x10^{-4}$)					7.0055	7.421875

Error= =6%

TABLA 7

VERSION-2 N=10		TERMINO GENERAL	N° TOTAL OPERAC.	P E S O	T O T A L	TIEMPO REAL (SG.)
POS.MEMORIA		N^2+2N+6	126	—	—	
SUMAS SIMPLES		$2N^2$	200	0.839	167.8	
DIV. MIXTA DOBLE		$(N^2-1)N/2$	495	2.517	1245.915	
RESTA MIXTA DOBLE		$(N^2-1)N/2$	495	2.441	1208.295	
PROD. MATRIC. DOBLE		$(N^2-1)N/2$	495	4.213	2085.435	
DIV. MATRIC. DOBLE		N	10	4.125	41.25	
COMP. IGUALD. SIMPLE		(N+1)N	110	1.881	206.91	
COMP. > SIMPLE		(N+1)N/2	55	1.765	97.075	
ASIGNACION SIMPLE		$(N^2+3N)/2$	65	1.039	67.535	
ASIGNACION MATR. DOBLE		(N+3)N/2	65	4.387	285.155	
ASIG. MIXTA SIMPLE		N	10	1.873	18.73	
ASIG. MIXTA DOBLE		$(N^2+3N+6)N/2$	680	2.762	1878.16	
PASOS BUCLE	BUCLES EXT.	2N	20	1	20	
	BUCLES INTER.	$\frac{N}{2}(N^2+4N+3)$	715	1.0502	750.893	
VALOR ABSOLUTO SIM.		(N+1)N/2	55	0.358	19.69	
VALOR ABSOLUTO DOB.		(N+1)N/2	55	2.027	111.485	
				TOTAL	8204.328	
TIEMPO EN SG. $(x9.9031x10^{-4})$					8.1248	8.1836

Error = =0,7%

Como vemos, las dos versiones a pesar de ser muy diferentes en su concepción pueden considerarse equivalentes en cuanto a su tiempo de cálculo, aunque parece algo mejor no realizar los intercambios de filas (Versión 1), a costa de ocupar algo más de memoria.

Sin embargo, si nos fijamos atentamente en las descomposiciones lógicas de las versiones 1 y 2, vemos que ambas se pueden optimizar dado que podemos calcular fuera del bucle de índice K, los términos $A(J,I)/TEMP$ y $A(J,I)/A(I,I)$. Con este pequeño cambio el tiempo real (para $N=10$) pasa a ser de 6.586 para la versión 1, y 6.918 para la versión 2 respectivamente.

Esta última todavía es mejorable si mantenemos el pivote en *TEMP* (igual que hacemos en la versión 1), para evitar la división mixta doble que tiene un peso mucho mayor que la división simple. Para ello, tendremos que crear una nueva variable auxiliar *AUX* para realizar el intercambio de filas en vez de la variable *TEMP* que antes utilizábamos; con ello se reduce el tiempo de ordenador a 6.808. Por todo ello, parece mejor la versión 2 con algo más de tiempo de cálculo a costa de ocupar $2N$-1 posiciones de memoria menos.

CONCLUSIONES

La resolución del problema parte de un resultado teórico que es aplicado mediante un algoritmo determinado. En la programación de este algoritmo es donde radica principalmente la economía de la resolución del problema.

Es necesario, una vez elegido el algoritmo, analizar y optimizar su programación, cuestión a la que hasta ahora se ha dado una importancia menor[8].

Para optimizar la programación de un algoritmo, se emplean una serie de parámetros, que indican la memoria ocupada y el tiempo necesario para la resolución de un problema. Mediante el uso de las descomposiciones lógicas podemos obtener de una forma rápida el valor de estos parámetros, los cuales determinarán qué versión es óptima en la aplicación del algoritmo.

Es preciso hacer notar que los parámetros están interrelacionados entre sí, y su importancia es diferente de unos problemas a otros. Así, puede que sea mejor tener mayor número de operaciones sencillas si por el contrario se reduce el número de comparaciones. Además no es lo mismo operar o comparar variables simples que variables de una matriz.

Es interesante restringir al máximo el uso de matrices y caso en que sea necesario intentar operar sólo en matrices de una dimensión. También es interesante tratar de evitar las comparaciones. Calculando los pesos con exactitud y teniendo en cuenta los conceptos desarrollados hemos visto cómo se puede calcular el tiempo de cálculo con un error menor del 10% lo cual representa un grado de aproximación aceptable. En cualquier caso es necesario ajustar los pesos de los parámetros para el ordenador y lenguaje que se quiera aplicar, todo ello en función de la aproximación deseada. Otro punto importante es ver la relación entre el tiempo de ordenador y los datos de entrada para saber si la variación es lineal, cuadrática, etc...

REFERENCIAS

1. J. Warnier, B. Flanagan. "Programación lógica" Tomo I *Ed. Técnicos Asociados,* (1973).
2. N. Epelboim, B.J. Goodno. "Memory management in structural analysis on microcomputers". *Advances in Engineering Software,* Vol. **7**, n.º 4, (1985).
3. A. Perronet. "Cours D.E.A. 81-82". *Lab. Analyse Numérique,* Univer. París VI, (1982).
4. O.C. Zienkiewicz. "El método de los elementos finitos". *Ed. Reverté,* (1980).
5. H.N. Wirth. "Algoritmos + estructuras de datos = programas". *Ed. del Castillo,* (1980).
6. L. Gavete, L.J. Sánchez-Tembleque. "Análisis de algoritmos numéricos utilizando descomposiciones lógicas". *I Congreso de lenguajes naturales y lenguajes formales.* Barcelona, octubre, (1985).
7. F. Michavila, L. Gavete. "Programación y Cálculo Numérico". *Ed. Reverté,* (1985).
8. H. Schad. "Computing costs for FEM analysis of foundation engineering problems and possible ways of increasing efficiency". *Int. J. Num. and Anal. Meth. Geomec.* Vol. **9**, 261-275, (1985).