```
UT 1 621 4 3 UT 2 438 4 3 NV 1 493 4 3 NV 2 378 4 4 WA 1 1680 4 4 WA 2 1122 4 4 OR 1 1014 4 4 OR 2 756
4 4 CA 1 10643 4 4 CA 2 10114 4 4 AK 1 349 4 4 AK 2 329 4 4 HI 1 273 4 4 HI 2 298
;
run;

proc format;
value regfmt 1='Nordeste'
             2='Sur'
             3='Medio Oeste'
             4='Oeste';
value divfmt 1='Nueva Inglaterra'
             2='Atlantico Medio'
             3='Montana'
             4='Pacifico';
value clifmt 1='Residencial'
             2='Comercial';
run;

proc sort data=energia;
by cliente;
run;

proc means data=energia;
var gastos;
by cliente;
format cliente clifmt.;
run;
```

Esto produce la salida mostrada a continuación donde se tiene un bloque de estadísticos para cada uno de los tipos de cliente, así que en estos bloques en la columna bajo el encabezado de MEAN se tienen los resultados buscados. Pero estos resultados están acompañados de otros que no son de interés originalmente en el problema, por tanto pueden llegar a confundir al analista.

Cliente=Residencial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
22	2104.14	2908.04	194.0000000	10643.00

Cliente=Comercial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
22	1707.05	2588.30	184.0000000	10114.00

Adicionalmente, para contestar el inciso (2) debemos procesar el siguiente código:

```
proc sort data=energia;
by region cliente;
run;

PROC MEANS DATA=ENERGIA;
VAR GASTOS;
BY REGION CLIENTE;
FORMAT REGION REGFMT. CLIENTE CLIFMT.;
RUN;
```

Este último produce los resultados consignados en la siguiente salida SAS, donde ahora se tienen bloques de estadísticos (donde para nuestro caso sólo es de interés el valor bajo la columna MEAN) para todas las combinaciones de valores de las variables REGION y CLIENTE (Nordeste-Residencial, Nordeste-Comercial, Oeste-Residencial y Oeste-Comercial).

Region=Nordeste Cliente=Residencial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
9	2984.00	2999.33	353.0000000	8786.00

Region=Nordeste Cliente=Comercial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
9	2245.22	2519.56	188.0000000	7825.00

Region=Oeste Cliente=Residencial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
13	1495.00	2796.30	194.0000000	10643.00

Region=Oeste Cliente=Comercial

Analysis Variable : Gastos

N	Mean	Std Dev	Minimum	Maximum
13	1334.46	2669.08	184.0000000	10114.00

En resumen, se tiene una gran cantidad de resultados para contestar las dos inquietudes del analista, de manera que si se quieren presentar los resultados a alguien externo, sería necesario imprimir todos estos resultados, y luego explicarle donde están ubicados aquellos de interés.

Otra alternativa sería reescribir los resultados en una tabla en un procesador de texto cualquiera, pero esto implicaría tiempo y puede introducir errores. Finalmente, se puede utilizar un procedimiento diferente, el cual es más apropiado para llevar a cabo la tarea de presentar resultados, y este es el PROC TABULATE.

Enfoque usando PROC TABULATE Para producir exactamente lo que se requería desde el principio, se puede usar el siguiente código:

```
PROC TABULATE DATA=ENERGIA FORMAT=DOLLAR12.;
CLASS REGION CLIENTE;
VAR GASTOS;
TABLE REGION ALL, MEAN*GASTOS=' '*(CLIENTE ALL) / RTS=25;
FORMAT REGION REGFMT. DIVISION DIVFMT. CLIENTE CLIFMT.;
KEYLABEL MEAN='GASTO PROMEDIO';
TITLE1 'GASTOS DE ENERGIA PARA CADA REGION DISCRIMINADOS POR TIPO DE CLIENTE';
TITLE2 '(MILLONES DE DOLARES)';
RUN;
```

Este último genera la salida mostrada a continuación, donde se encuentran los resultados pedidos en una tabla de formato sencillo y listo para imprimir.

GASTOS DE ENERGIA PARA CADA REGION DISCRIMINADOS POR TIPO DE CLIENTE

(MILLONES DE DOLARES)

	GASTO PROMEDIO		
	Cliente		
	Residencial	Comercial	All
Region			
Nordeste	$2,984	$2,245	$2,615
Oeste	$1,495	$1,334	$1,415
All	$2,104	$1,707	$1,906

Ejemplo 2. Calcular varios estadísticos de porcentajes

Este ejemplo muestra cómo usar tres estadísticos de porcentajes acumulados: COLPCTSUM, REPPCTSUM, y ROWPCTSUM. El archivo FUNDRAIS contiene datos de ventas de estudiantes durante un bazar de escuela.

```
options nodate nocenter nonumber linesize=105 pagesize=60;
data fundrais;
length nombre $ 8 salon $ 1;
input @1 equipo $ @8 salon $ @10 nombre $ @19 lapices @23 cuadernos;
ventas=lapices + cuadernos;
cards;
AZUL    A ANA      4   8
ROJO    A MARIA    5  10
VERDE   A JOHN     6   4
ROJO    A BORIS    2   3
AZUL    B FREDY    6   8
VERDE   B LUIS    12   2
AZUL    B ANDREA   .   9
ROJO    B HENRY    8  10
VERDE   A ANDRES   3   5
ROJO    A SAMUEL  12  10
```

```
AZUL   A LINA     7  12
VERDE  A SARA     4   .
AZUL   B MARTIN   9  13
ROJO   B MATEO    7   6
VERDE  B BETY    15  10
ROJO   B LAURA    4   3
;
run;
proc format;
picture pctfmt low-high='009 %';
run;

title 'Ventas del Bazar ;
proc tabulate format=7.; /* La opción FORMAT= especifica hasta siete digitos como el formato por
                            defecto para el valor en cada celda de la tabla*/
class equipo salon;      /*La declaración CLASS identifica como variables clase a EQUIPO y SALON.*/
var ventas;              /*La declaración VAR identifica como variable de análisis a VENTAS.*/
table (equipo all)*ventas=' ',
salon='Salón de Clase'*(sum
colpctsum*f=pctfmt9.
rowpctsum*f=pctfmt9.
reppctsum*f=pctfmt9.)all /* crea una fila para cada valor formateado de EQUIPO. Dentro de cada fila
totaliza las VENTAS para cada EQUIPO. La última fila totaliza las VENTAS para todos los equipos.
La dimensión columna en la declaración TABLE crea una columna para cada valor formateado de SALON.
Anidadas dentro de cada columna están las columnas que totalizan las VENTAS para cada SALON.
La primera columna anidada, etiquetada sum, es la suma de las VENTAS por cada fila para cada SALON DE
CLASES.
La segunda columna anidada, etiquetada ColPctSum, es el porcentaje de la suma de las VENTAS por fila
para cada SALON DE CLASES en relación a la suma de las VENTAS en el SALON DE CLASES.
La tercera columna anidada, etiquetada RowPctSum, es el porcentaje de la suma de las VENTAS por fila
para cada SALON DE CLASES en relación a la suma de las VENTAS por fila para todos los SALONES DE
CLASES.
La cuarta columna anidada, etiquetada RepPctSum, es el porcentaje de la suma de las VENTAS por fila
para cada SALON DE CLASES en relación a la suma de las VENTAS para todos los EQUIPOS y para todos los
SALONES DE CLASES.
La última columna del reporte totaliza las VENTAS por fila para todos los SALONES DE CLASES.*/
/rts=20 row-float; /* RTS= provee 20 caracteres por linea para encabezados de fila. ROW=FLOAT elimina
                      encabezados de fila en blanco.*/
run;
```


A continuación se presenta el reporte final obtenido con el programa anterior.

Ventas del Bazar

	Salón de Clase								
	A				B				All
	Sum	ColPctSum	RowPctSum	RepPctSum	Sum	ColPctSum	RowPctSum	RepPctSum	Sum
equipo									
AZUL	31	34 %	46 %	15 %	36	31 %	53 %	17 %	67
ROJO	42	46 %	52 %	20 %	38	33 %	47 %	18 %	80
VERDE	18	19 %	31 %	8 %	39	34 %	68 %	19 %	57
All	91	100 %	44 %	44 %	113	100 %	55 %	55 %	204

2.5.7 PROC SORT: Ordenando bases de datos.

Este procedimiento permite ordenar un conjunto de datos de acuerdo a una o más variables, bien sean numéricas o alfanuméricas. Para correr ciertas opciones en algunos PROCs es necesario ordenar los datos de acuerdo a alguna variable, por lo tanto se debe ejecutar antes este PROC.

Por ejemplo ordenaremos los datos de los países por zonas:

```
PROC SORT DATA=PAISES;
 BY ZONA;
RUN;

PROC PRINT DATA=PAISES;
 VAR PAIS ZONA;
 TITLE 'Paises por Zonas';
RUN;
```

El resultado resumido es el siguiente:

```
                    Paises por Zonas
            OBS     PAIS                          ZONA

              1     Antigua                      CARIBE
```

SEDE MEDELLÍN
DEPTO. DE BIBLIOTECAS
TECA "EFE" GOMEZ

2	Bahamas	CARIBE
3	Barbados	CARIBE
.	.	.
.	.	.
.		.
14	Trinidad y Tobago	CARIBE
15	Belice	CENTROAM
16	Costarica	CENTROAM
.	.	.
.	.	.
.		.
22	Panama	CENTROAM
23	Canada	NORTEAME
24	USA	NORTEAME
25	Argentina	SURAMER
26	Bolivia	SURAMER
.	.	.
.	.	.
.		.
36	Venezuela	SURAMER

2.5.8 PROC TRANSPOSE.

Convierte las variables en registros y viceversa. Sirve para facilitar la manipulación de datos.

2.5.9 PROC CORR.

Este procedimiento calcula los coeficientes de correlación entre variables incluido el producto de momento de Pearson, y correlaciones de producto-momento ponderadas.

2.5.10 PROC SUMMARY.

Este PROC es muy similar al MEANS, sólo que permite algunas opciones adicionales además de configurar archivos de salida un poco más elaborados.

En nuestro caso se mostrará el ejemplo pero utilizando la clasificación por zonas geopolíticas.

```
PROC SUMMARY DATA=PAISES PRINT;
 CLASS ZONA;
 VAR X1 X11;
 TITLE 'Estadisticos por Zonas de Natalidad (X1) y Poblacion (X11)';
RUN;
```


Los resultados de este procedimento son:

```
Estadisticos por Zonas de Natalidad (X1) y Poblacion (X11)      105
                                          18:03 Sunday, August 7, 1994
ZONA        N Obs  Variable   N        Mean        Std Dev      Minimum
-------------------------------------------------------------------------
CARIBE       14    X1        14    23.2192857     6.0087481    14.1000000
                   X11       14    2039616.57     2954624.65     44224.00
CENTROAM      8    X1         8    32.8750000     6.3793752    24.0000000
                   X11        8   12720733.00    27701137.59    143386.00
NORTEAME      2    X1         2    15.5000000     2.1213203    14.0000000
                   X11        2     138003366     156562644   27296859.00
SURAMER      12    X1        12    26.9250000     5.6820491    17.0000000
                   X11       12   23010802.58    40161912.00    352041.00
-------------------------------------------------------------------------

                ZONA       N Obs  Variable       Maximum
                ----------------------------------------
                CARIBE       14   X1           36.0000000
                                  X11          9723606.00
                CENTROAM      8   X1           40.0000000
                                  X11         81140952.00
                NORTEAME      2   X1           17.0000000
                                  X11           248709873
                SURAMER      12   X1           36.0000000
                                  X11           146154602
                ----------------------------------------
```

2.5.11 PROCEDIMIENTOS PARA ANÁLISIS MULTIVARIADO: PROC CLUSTER y PROC TREE.

Estos dos procedimientos se utilizan coordinadamente para realizar análisis de clusters (grupos) dentro de los procedimientos de análisis multivariado. El PROC CLUSTER produce grupos jerárquicos de observaciones en datos del SAS usando uno de los once métodos de que dispone el programa.

El método a utilizar para realizar el análisis de cluster se especifica con la opción METHOD=. Los métodos utilizados son: AVERAGE, CENTROID, COMPLETE, DENSITY, EML (Máximo Verosimilitud Esférica), FLEXIBLE, MCQUITTY, MEDIAN, SINGLE, TWOSTAGE y WARD.

El PROC TREE está asociado íntimamente con el PROC CLUSTER ya que es el encargado de mostrar gráficamente los resultados del PROC CLUSTER. Dentro de las opciones más útiles se encuentra la

opción HORIZONTAL que permite que el gráfico de clusters se pueda visualizar mejor de manera horizontal que vertical.

En el ejemplo siguiente se utiliza el comando del OPTIONS del SAS en donde se especifican ciertos parámetros de control del programa. En el caso particular de nuestro ejemplo se está especificando el tamaño de la línea (LS=75) y el número de renglones por hoja (PS=66).

También debemos notar como el PROC TREE permite ver los resultados del PROC CLUSTER de dos maneras diferentes para que sea más fácil la interpretación. En particular con la opción HEIGHT=N permite ver la formación de los clusters de acuerdo al número de clusters que se van formando.

```
OPTIONS LS=75 PS 66;
PROC CLUSTER DATA=Paises METHOD=cen PSEUDO;
 var X1-X5 X8;
 id PAIS;
 TITLE 'Análisis de Clusters';
RUN;
OPTION LS=100 PS=66;
PROC TREE;
RUN;
PROC TREE SORT HEIGHT=N;
RUN;
```

Los resultados son:

Análisis de Clusters

Centroid Hierarchical Cluster Analysis

Eigenvalues of the Covariance Matrix

	Eigenvalue	Difference	Proportion	Cumulative
1	7.0249E8	6.8308E8	0.970832	0.97083
2	19403286	17700895	0.026815	0.99765
3	1702391	1702357	0.002353	1.00000
4	33.86634	29.21751	0.000000	1.00000
5	4.648833	2.562334	0.000000	1.00000
6	2.086499	.	0.000000	1.00000

Root-Mean-Square Total-Sample Standard Deviation = 10981.75

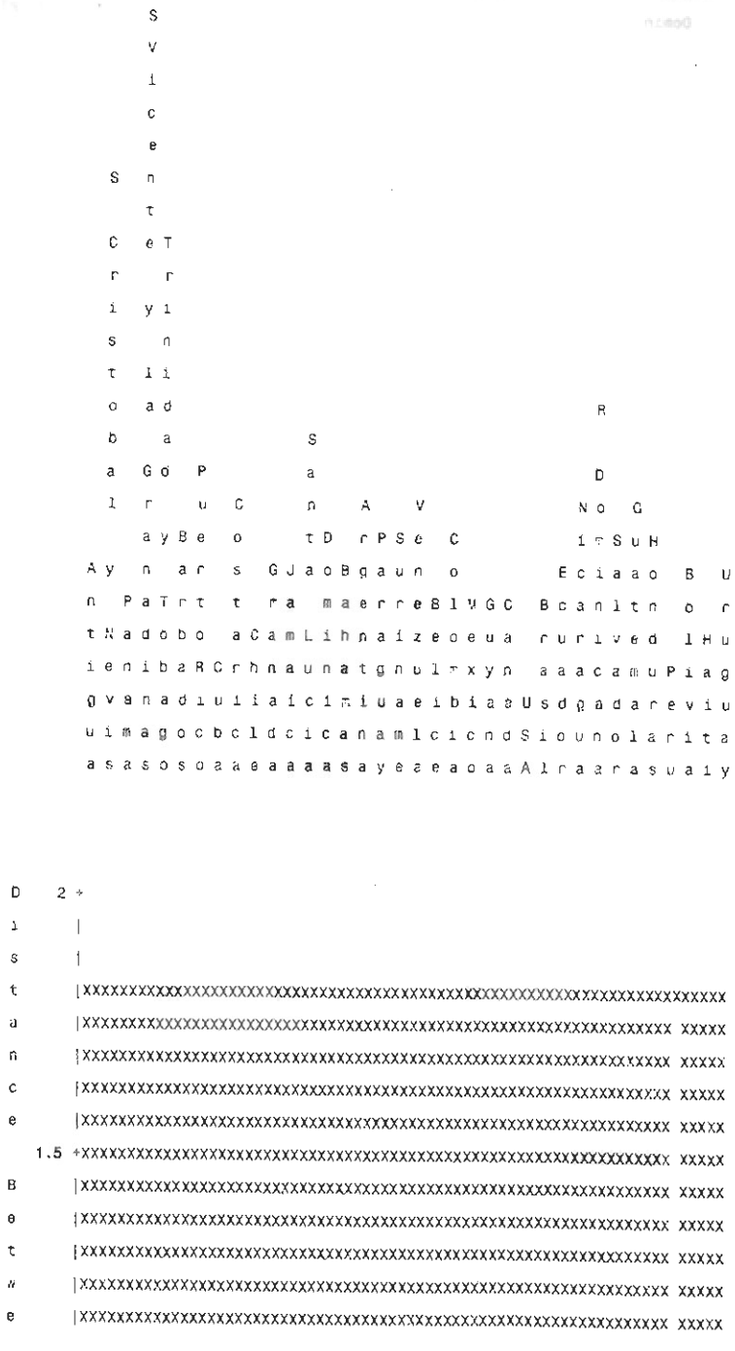
Root-Mean-Square Distance Between Observations = 38041.88

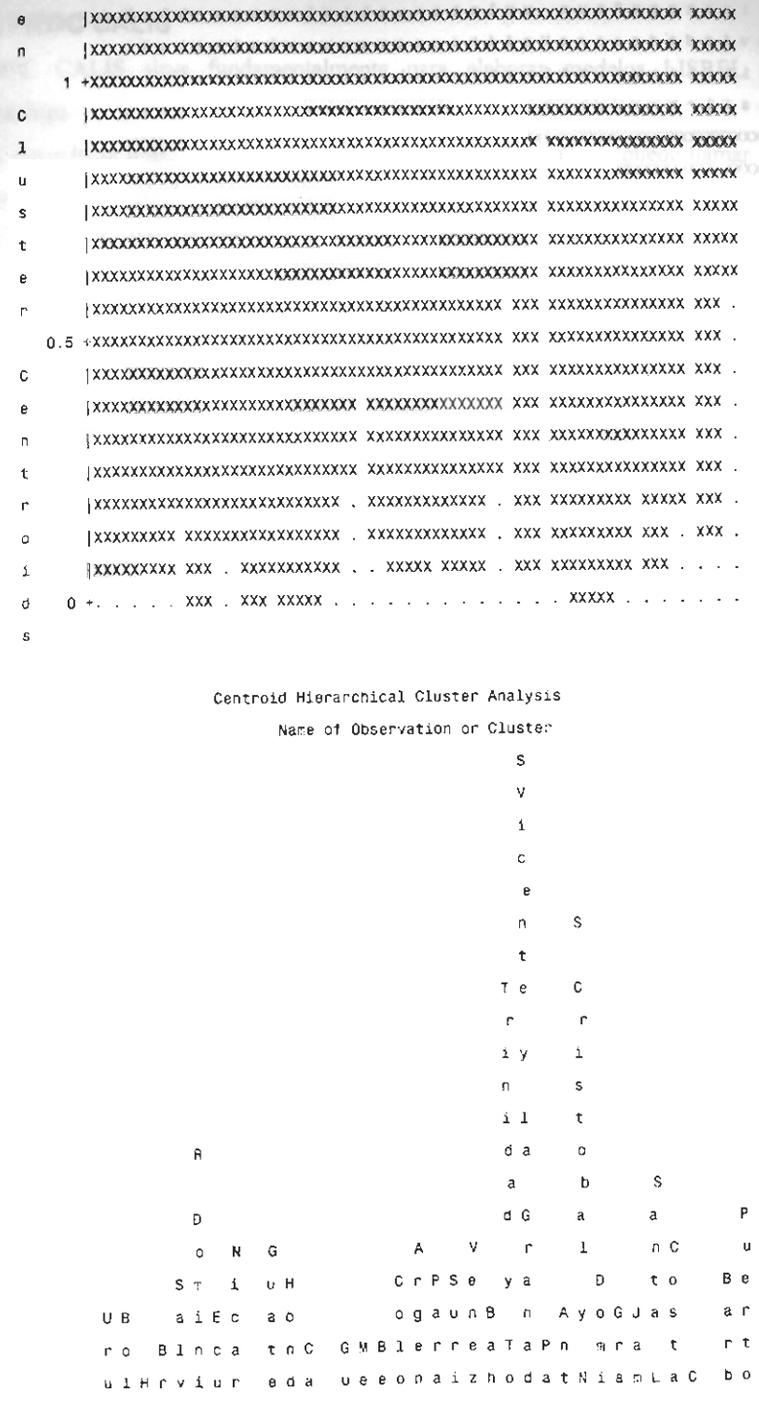
NCL	----------Clusters Joined----------		FREQ	PSF	PST2	Norm Cent Dist	T i e
35	Jamaica	Santa Lucia	2	3220	.	0.01788	
34	Granada	CL35	3	2708	1.5	0.01865	
33	Ecuador	Nicaragua	2	2523	.	0.02274	

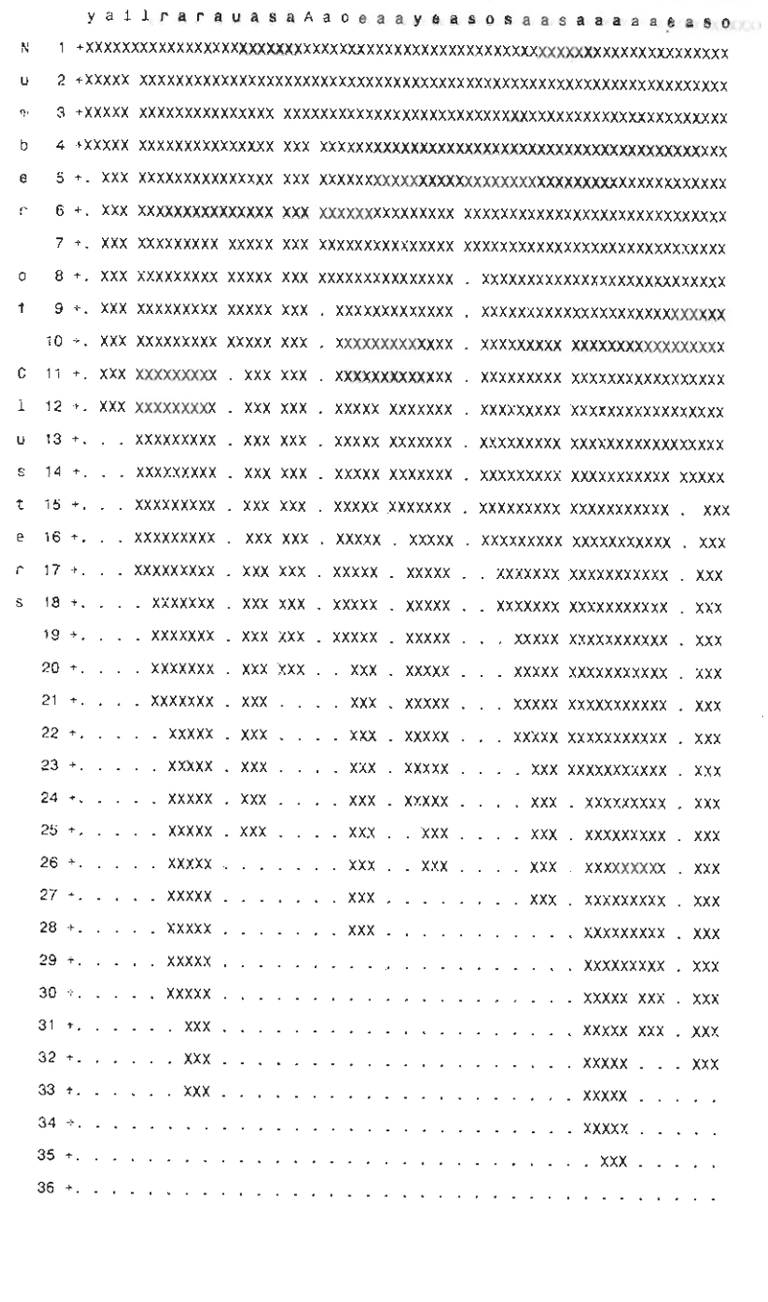
32	Barbados	PuertoRico	2	2342	.	0.02506
31	Costarica	Chile	2	2203	.	0.02682
30	CL33	R Dominicana	3	1974	2.0	0.02766
29	CL31	CL34	5	1354	5.6	0.03411
28	Belice	Colombia	2	1280	.	0.04052
27	Antigua	S Cristobal y Nevis	2	1223	.	0.04249
28	Suriname	Venezuela	2	1189	.	0.04319
25	Guatemala	Honduras	2	1083	.	0.05508
24	Paraguay	CL26	3	927.7	2.6	0.06042
23	CL29	Dominica	6	792.4	6.0	0.06200
22	CL27	Panama	3	745.7	2.9	0.06235
21	CL30	Salvador	4	705.3	7.7	0.06279
20	Canada	USA	2	715.5	.	0.06300
19	CL28	Mexico	3	882.0	4.4	0.07378
18	CL22	S Vicente y la Granadinas	4	652.0	2.4	0.07456
17	Brasil	CL21	5	627.9	3.8	0.07632
16	CL18	Trinidad y Tobago	5	601.5	2.2	0.08424
15	Argentina	CL24	4	550.9	5.2	0.10840
14	CL32	Cuba	3	522.5	28.6	0.11604
13	CL14	CL23	9	388.0	14.1	0.12151
12	Bolivia	Haiti	2	401.5	.	0.13084
11	CL15	CL19	7	338.1	10.2	0.14057
10	CL25	Peru	3	344.4	11.3	0.16013
9	CL16	CL13	14	207.4	26.1	0.20469
8	CL11	Guyana	8	208.9	5.8	0.23571
7	CL9	Bahamas	15	216.0	3.7	0.24112
6	CL17	CL10	8	200.7	27.8	0.25816
5	CL7	CL8	23	76.8	66.6	0.45833
4	CL12	Uruguay	3	89.3	30.7	0.62794
3	CL5	CL20	25	94.3	11.8	0.63345
2	CL3	CL6	33	38.8	74.0	0.95896
1	CL2	CL4	36	.	38.8	1.84109

Centroid Hierarchical Cluster Analysis

Name of Observation or Cluster



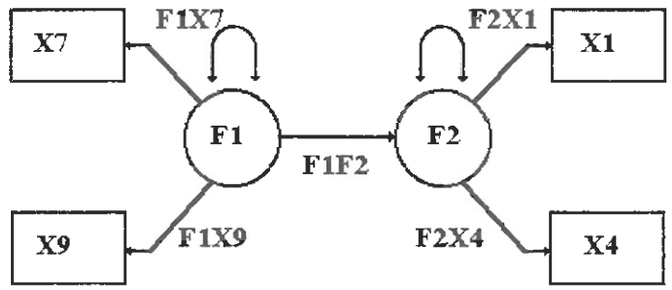




2.5.12PROC CALIS

El PROC CALIS sirve fundamentalmente para elaborar modelos LISREL (Linear Structural Relationship).

Para el ejemplo se tomó el siguiente modelo LISREL: el factor 1 se puede llamar económico-cultural que comprende X7: Alumnos por Habitante y X9: Población económicamente activa, el factor 2 puede indicar la salud de la población que se ve reflejada en la esperanza de vida (X4) y en la natalidad (X1). A continuación se muestra la gráfica correspondiente al modelo planteado:



El programa para ejecutar este modelo es:

```
PROC CALIS DATA=PAISES COV METHOD=GLS OMETHOD=NEWRAP;
 VAR X1 X4 X7 X9;
 RAM
 1      3      5      F1X7,
 1      4      5      F1X9,
 1      1      6      F2X1,
 1      2      6      F2X4,
 1      6      5      F1F2,
 2      1      1      VARX1,
 2      2      2      VARX4,
 2      3      3      VARX7,
 2      4      4      VARX9,
 2      5      5      VARF1,
 2      6      6      VARF2;
RUN;
```

Los resultados son los siguientes:

Análisis de Clusters

Covariance Structure Analysis: Pattern and Initial Values

RAM Model Statement

	Matrix	Rows & Cols		Matrix Type	
TERM 1					
1	_IDE_	4	6	IDENTITY	
2	_A_	6	6	GENERAL	IMINUSINV
3	_P_	6	6	DIAGONAL	

Análisis de Clusters

Covariance Structure Analysis: Pattern and Initial Values

RAM Pattern and Values

Term & Matrix		Row & Column		Parameter	Estimate
1	2	1	6	F2X1	.
		X1	F2		
1	2	2	6	F2X4	.
		X4	F2		
1	2	3	5	F1X7	.
		X7	F1		
1	2	4	5	F1X9	.
		X9	F1		
1	2	6	5	F1F2	.
		F2	F1		
1	3	1	1	VARX1	.
		E1	E1		
1	3	2	2	VARX4	.
		E2	E2		
1	3	3	3	VARX7	.
		E3	E3		
1	3	4	4	VARX9	.
		E4	E4		
1	3	5	5	VARF1	.
		D1	D1		
1	3	6	6	VARF2	.
		D2	D2		

Análisis de Clusters

Covariance Structure Analysis: Generalized Least-Squares Estimation

34 Observations		Model Terms	1
4 Variables		Model Matrices	3
10 Informations		Parameters	11

VARIABLE	Mean	Std Dev
X1	26.73147059	7.017267072
X4	69.08823529	5.077386867
X7	3.49594045	0.653666438
X9	60.27647059	8.084930195

Some initial estimates computed by instrumental variable method.

Some initial estimates computed by two-stage LS method.

Vector of Initial Estimates

F2X1	1	1.00000	Matrix Entry: _A_[1:6]
F2X4	2	-0.64016	Matrix Entry: _A_[2:6]
F1X7	3	1.00000	Matrix Entry: _A_[3:5]
F1X9	4	60.08050	Matrix Entry: _A_[4:5]
F1F2	5	-18.67628	Matrix Entry: _A_[6:5]
VARX1	6	6.11228	Matrix Entry: _P_[1:1]
VARX4	7	8.10522	Matrix Entry: _P_[2:2]
VARX7	8	0.40917	Matrix Entry: _P_[3:3]
VARX9	9	0.01000	Matrix Entry: _P_[4:4]
VARF1	10	0.01811	Matrix Entry: _P_[5:5]
VARF2	11	36.81222	Matrix Entry: _P_[6:6]

Covariance Structure Analysis: Generalized Least-Squares Estimation

Newton-Raphson Minimization

Algorithm for Hessian= 1

Maximum Iterations= 50

Maximum Function Calls= 125

Maximum Absolute Gradient Criterion= 0.001

Number of Estimates= 11 Lower Bounds= 0 Upper Bounds= 0

Minimization Start: Active Constraints= 0 Criterion= 0.661

Maximum Gradient Element= 2.647

Iter	nfun	act	mincrit	maxgrad	difcrit	ridge	rhoratio
1	2	0	0.31731	0.4051	0.3433	0.0313	0.5075
2	3	0	0.30115	0.1238	0.0162	0.00781	0.5426
3	4	0	0.27692	0.0618	0.0242	0.00195	0.8751
4	5	0	0.23518	0.0229	0.0417	0.000488	0.7979
5	6	0	0.21328	0.0229	0.0219	0.000122	0.6392
6	7	0	0.21072	0.0204	0.00256	0.000031	0.6223

7	8	0	0.20921	0.0412	0.00150	7.629E-6	0.8751
8	9	0	0.20623	0.0337	0.00298	1.907E-6	1.0466
9	10	0	0.19473	0.2446	0.0115	4.768E-7	1.0591
10	11	0	0.19280	7.7848	0.00193	1.192E-7	0.0495
11	12	0	0.13377	1.9729	0.0590	2.384E-7	0.5237
12	13	0	0.09879	5.5131	0.0350	2.98E-8	0.5345
13	14	0	0.09086	2.6877	0.00794	7.451E-9	0.5243
14	15	0	0.08593	2.3305	0.00493	1.863E-9	0.5333
15	16	0	0.08416	1.3261	0.00177	4.66E-10	0.5335
16	17	0	0.08341	0.6876	0.000749	1.16E-10	0.5684
17	18	0	0.08316	0.3143	0.000251	2.91E-11	0.6167
18	19	0	0.08308	0.1320	0.000083	2.91E-11	0.6724
19	20	0	0.08305	0.0519	0.000027	2.91E-11	0.7254
20	21	0	0.08304	0.0196	9.052E-6	2.91E-11	0.7607
21	22	0	0.08304	0.007187	3.109E-6	2.91E-11	0.7801
22	23	0	0.08304	0.002589	1.081E-6	2.91E-11	0.7890
23	24	0	0.08303	0.000921	3.778E-7	2.91E-11	0.7927

Minimization Results: Iterations= 23 Function Calls= 24

Derivative Calls= 24

Active Constraints= 0 Criterion= 0.083 Maximum Gradient Element= 0.000921

Ridge= 2.91E-11

NOTE: Convergence criterion satisfied.

Análisis de Clusters

Covariance Structure Analysis: Generalized Least-Squares Estimation

Fit criterion 0.0830

Goodness of Fit Index (GFI) 0.9585

GFI Adjusted for Degrees of Freedom (AGFI). . . . .

Root Mean Square Residual (RMR) 5.6238

Parsimonious GFI (Mulaik, 1989) . . . . . -0.1597

Chi-square = 2.7402 df = -1 Prob>chi**2 - .

Null Model Chi-square: df - 6 480.9420

RMSEA Estimate . . . . . 0.0000 90%C.I.[., .]

Probability of Close Fit . . . . . .

ECVI Estimate . . . . . 0.0000 90%C.I.[., .]

Bentler's Comparative Fit Index 0.9921

Akaike's Information Criterion. 4.7402

Bozdogan's (1987) CAIC. 7.2665

Schwarz's Bayesian Criterion. 6.2665

McDonald's (1989) Centrality. 0.9465

Bentler & Bonett's (1980) Non-normed Index. . . . .

Bentler & Bonett's (1980) NFI 0.9943

James, Mulaix, & Brett (1982) Parsimonious NFI. . -0.1657

Z-Test of Wilson & Hilferty (1931). . . . . .

Bollen (1986) Normed Index Rho1 . . . . . .

Bollen (1988) Non-normed Index Delta2 . . . . . 0.9922

Hoelter's (1983) Critical N . . . . . .

WARNING: The central parameter matrix _P_ has probably 1 negative eigenvalue(s).

NOTE: Hessian matrix is not full rank. Not all parameters are identified. Some parameter estimates are linearly related to other parameter estimates as shown in the following equations:

VARF1 = 1493.5 * F1X7 + 8445.2 * F1X9 - 6074.3 * F1F2 - 699611

VARF2 = 0.0228 * F2X1 - 0.00469 * F2X4 + 0.5226 * F1F2 + 56.8805

NOTE: Moore-Penrose inverse of information matrix computed.

Análisis de Clusters

Covariance Structure Analysis: Generalized Least-Squares Estimation

RAM Pattern and Values

Term & Matrix	Row & Column			Parameter	Estimate	Standard Error	t Value
1	2	1	6	F2X1	1.681230	0.865374	1.943
		X1	F2				
1	2	2	6	F2X4	-0.345084	0.230235	-1.499
		X4	F2				
1	2	3	5	F1X7	9.460103	6.445916	1.468
		X7	F1				
1	2	4	5	F1X9	53.493864	14.171606	3.775
		X9	F1				
1	2	6	5	F1F2	-38.476032	19.983897	-1.925
		F2	F1				
1	3	1	1	VARX1	-71.134875	126.116931	-0.564
		E1	E1				
1	3	2	2	VARX4	16.773675	6.931056	2.420
		E2	E2				
1	3	3	3	VARX7	0.141361	0.176711	0.800
		E3	E3				
1	3	4	4	VARX9	47.650304	13.981430	3.408

		E4	E4				
1	3	5	5	VARF1	0.003167	0.003285	0.964
		D1	D1				
1	3	6	6	VARF2	36.813883	10.455894	3.521
		D2	D2				

Standardized Coefficients

Row & Column		Parameter	Estimate
1	6	F2X1	1.593927
X1	F2		
2	6	F2X4	-0.477060
X4	F2		
3	5	F1X7	0.816840
X7	F1		
4	5	F1X9	0.399754
X9	F1		
6	5	F1F2	-0.336113
F2	F1		

Variances of Endogenous Variables

Variable		Estimate	R-squared
1	E1	46.173402	2.540603
2	E2	21.715924	0.227586
3	E3	0.424797	0.667228
4	E4	56.713291	0.159804
6	D2	41.502492	0.112972

2.5.13 Procedimientos para Análisis de Varianza:

2.5.13.1 PROC GLM.

Este es uno de los más poderosos procedimientos proporcionado en SAS. Permite realizar análisis del modelo lineal general. Permite ejecutar análisis de varianza, de regresión, de covarianza, y análisis multivariable de varianza (MANOVA).

2.5.13.2 PROC ANOVA.

Ejecuta análisis de varianza para diseños balanceados.

2.5.14 Procedimientos para el Análisis de Datos Categóricos.

2.5.14.1 PROC CATMOD.

Ajusta modelos lineales a funciones de datos categóricos.

2.5.14.2 PROC PROBIT.

Procedimientos para Análisis Discriminante

2.5.14.3 PROC STEPDISC.

Este método permite hacer discriminación en etapas: hacia adelante, hacia atrás, o selección stepwise. La idea es seleccionar un subconjunto de variables que permita hacer una buena discriminación.

La descripción de cada uno de estos PROC del SAS es bastante extensa, sin embargo como una primera se mostrará la sintaxis básica del PROC y algunas opciones útiles para realizar tareas específicas. No dude en consultar el manual porque en este puede encontrar las opciones descritas de manera completa.

Capítulo 3: UNA INTRODUCCIÓN AL LENGUAJE MACRO

3.1. INTRODUCCIÓN

La programación Macro es una herramienta muy útil para estandarizar y extender la programación clásica en código SAS. Entre otras virtudes, permite reducir en gran medida la cantidad de código necesario para las tareas más frecuentes.

En resumen, se podría decir que la programación Macro permite asignar a un conjunto de caracteres o de sentencias de código un nombre clave. Esta palabra clave será sustituida por el valor asignado al ser ejecutado y compilado el correspondiente código.

La ventana de programación para crear Macros, es la ventana habitual de síntaxis. La programación Macro tiene dos componentes principales:

- El procesador Macro, encargado de "traducir".
- El lenguaje Macro, la sintaxis que se debe utilizar para comunicarse con el procesador Macro.

Al compilar un texto, existen dos caracteres que activan al procesador Macro:

& : &nombre , que designará a las variables Macro.
% : %nombre , que designará a las Macros creadas por nosotros o ya residentes en el sistema.

3.2. VARIABLES MACRO

Las variables Macro tienen una longitud máxima de 32K caracteres. Su longitud es determinada por la longitud del texto asociado. Las variables Macro contienen tan sólo caracteres, aunque existe la posibilidad de realizar operaciones entre los valores numéricos de variables Macro.

3.2.1 Variables Macro definidas por el sistema.

Algunas de tales variables son:

Sysdate Fecha

Sysday Día

Syslast El último dataset analizado

Syserr Resultado del paso data.

3.2.2 Variables Macro definidas por el usuario.

Tales variables deben tener un nombre que empiece por una letra o "_" aunque no es recomendable que empiecen por SYS, AF, DMS, SQL debido a que son nombres utilizados frecuentemente por el sistema.

Se puede escribir %put _all_; para ver todas las variables Macro creadas por el usuario.

3.2.3 Asignar valores a Variables Macro.

La forma más simple de asignar un valor a una variable Macro es mediante la instrucción %let.

Posibles maneras de crear variables Macro:

%let Asignación simple

%do iterativo El contador

%global Definen variables Macros que serán utilizadas en varias Macros y código SAS

%local Definen variables Macro que serán utilizadas sólo en una Macro

%input Permite asignar valores a variables Macros dentro del código Macro

%macro Dentro de una macro

Symput Dentro de un paso DATA

Into Dentro del Proc SQL

%window Dentro de una ventana Macro

Ejemplos de asignaciones con %let:

Ejemplo:

```
%let city = Barcelona;
Title ||Ciudad: &city=; -> Title ||Ciudad: Barcelona=;
```


La instrucción %let permite asignar a un nombre un conjunto de caracteres utilizando la sintaxis:

```
% let nombre = literal ;
```

```
%let calle=Diagonal;
%let calle=
                    Diagonal;
%let calle=   Diagonal  ;
```

(todas producen el mismo resultado: Diagonal).

```
%let num=123+123
```

(num tiene el valor 123+123).

```
%let num=%eval(123+123);
```

(num tiene el valor 246).

```
%let num=%sysevalf(123.5+122.5);
```

(La función %sysevalf realiza operaciones en punto flotante, num tiene el valor 246).

```
%let casa=;
```

(casa tiene valor nulo).

```
%let nombre=luis;
%let nick=&nombre;
```

(&nombre se convierte en luis y este valor es asignado a la variable nick).

```
%let mac=%pl;
```

(se asigna a mac el valor de la ejecución de la macro pl).

```
%let mac=%nrstr(%pl);
```

(se asigna a mac el literal "%pl").

```
%let plo = %str(proc print;run;);
```

(se asigna proc print;run; evitando así los ";" intermedios).

```
Data a;
```

```
Set b;
If _n_ = _last_ then CALL SYMPUT ('num', left(x));
Run;
```

(se asigna a la variable Macro num el valor de la última observación de la variable "x" del dataset "a").

3.2.4 Referencias a variables Macro.

Al utilizar referencias de variables Macro en expresiones entre comillas es necesario utilizar las comillas dobles: " para obtener la asignación deseada. En caso de comillas simples: ' no se obtiene el valor correcto.

Ejemplos

```
%let name=Juan;
%let name1=Pepe;
%let i=1;
%let Juan1=Maria

&name1 -> Pepe
&name&i -> Juan1
&&name&i -> Pepe
&name.1 -> Juan1
hola&name -> holaJuan
&name.&name -> JuanJuan
&name..&name -> Juan.Juan
&&&name&i -> Maria
```

3.2.5 Visualizar Variables Macro.

%put Escribe en el Log.

Options symbolgen; Escribe en el Log todas las asignaciones que vamos realizando.

Options Mprint Escribe el código generado por las Macros en el Log.

Las referencias a variables Macro ya creadas siempre serán precedidas por el símbolo: "&". Los nombres de variables Macro no pueden contener caracteres extraños cómo: ; , espacio " ' AND OR , etc.

3.3. GENERACIÓN DE CÓDIGO MEDIANTE MACROS

Ejemplo:

```
%macro b;
a
%mend b;
proc print data=%b; -> proc print data=a;
run;
```

Para invocar una macro ya creada, es necesario escribir el prefijo: "%". Aunque invocar una macro parece una instrucción de SAS, no es necesario acabar la instrucción con ";" cómo es habitual.

3.4. MACROS CON CÓDIGO SAS

Ejemplo:

```
%macro pl;
proc plot;
plot y*x;
run;
%mend;
%pl; -> proc plot;
plot y*x;
run;
```

3.5. MACROS CON PARÁMETROS DE ENTRADA

```
%macro pl (data=,yvar=,xvar=);
proc plot data = &data ;
plot &yvar * &xvar ;
run;
%mend;
```

```
%pl (data=a, yvar=y, xvar=x); -> proc plot data=a;
plot y*x;
run;
```

Los parámetros se pueden definir de varias maneras:

```
%macro pl (data=,yvar=,xvar=); %pl (data=a, yvar=y, xvar=x);
%macro pl (data,yvar,xvar); %pl (a,y,x);
%macro pl (data=a,yvar=y,xvar=x); %pl (xvar=x);
```

(en este último caso se definen valores por defecto).

3.6. COMENTARIOS EN LAS MACROS

Dentro de una macro se pueden insertar comentarios utilizando:

```
%* comentario ;
```

3.7. GENERACIÓN DE CÓDIGO MEDIANTE CONDICIONES

Ejemplo:

```
%macro quehago(info=);
%if &info=a %then %do;
proc print;
run;
%else %if &info=b %then %do;
proc print noobs label;
run;
%else %put |NO SE QUE HACER= &INFO |NO ME SIRVE=;
%mend;
```

La instrucción %IF … %THEN … ; %ELSE … ; es la correspondiente instrucción condicional del código Macro. Cuartdo las transformaciones a realizar son varias, éstas deben de escribirse entre un %DO; y un %END;

Para optimizar el código anterior se podría utilizar la función %UPCASE y así evitar posibles confusiones entre mayúsculas y minúsculas. La instrucción %PUT escribe un literal o el valor de una variable Macro en la ventana LOG.

3.8. GENERACIÓN DE CÓDIGO REPETITIVO

Ejemplo:

```
%macro a (num);
%do i=1 %to &num;
%put &num |Me portaré bién en clase ;
%end;
%mend;
%macro a (num,data);
%do i=1 %to &num;
proc print data=&data&num;
run;
%end;
%mend;
%a(2,b) -> proc print data=b1;
run;
proc print data=b2;
run;
```

Capítulo 4: PROC IML (Interactive Matrix Language)

El PROC IML (en versiones anteriores era llamado PROC MATRIX) es uno de los más útiles procedimientos ya que es un lenguaje de programación completo con un conjunto de funciones que permite hacer complejos programas en forma compacta. Es el único PROC que no requiere la especificación de un archivo SAS. El objeto básico sobre el cual opera todo el IML es una matriz de dos dimensiones (fila x columna) numérica o alfanumérica.

La programación es dinámica ya que el uso de la memoria y dimensionamiento de matrices es ejecutado automáticamente. Permite definir funciones y rutinas propias y además permite interactuar con el paso DATA, esto es, leer archivos en formato SAS o sacar matrices con formato SAS para ser usadas por otros PROC's. Las siguientes son unas de las ventajas del Proc IML:

1) Es un lenguaje de programación, posee las estructuras DO/END, START/FINISH, DO, IF-THEN/ELSE, GOTO, LINK, PAUSE, STOP.

2) Opera con matrices

3) Posee una amplia base de operadores

4) Usa operadores que aplican a matrices enteras, por ejemplo, si A y B son matrices, las siguientes operaciones son posibles dentro del IML: A+B, suma de matrices A*B, producto de matrices, A#B , multiplicación elemento a elemento, etc.

5) Puede trabajarse interactivamente

6) El IML es dinámico

7) Procesa datos

8) Produce gráficos

La estructura básica de un programa que sólo usa el IML es como sigue:

```
PROC IML;
...Instrucciones...
QUIT;
```

4.1. DEFINIENDO ESCALARES LITERALES:

```
a=12;  a= . ;
a='Hola a todos';
a='Hola';
```

SEDE MEDELLIN
DEPTO. DE BIBLIOTECAS
TECA "EFE" GOMEZ

Todos estos escalares son matrices de [1 x 1], y tienen la ventaja de que no hay que hacer referencia explícita a la fila y columna.

4.2. MATRICES LITERALES:

```
x={3 2 7 4 5};   /* produce un vector fila llamado x dé [1 x 5] */
y={3,2,7,4,5};   /* produce un vector columna llamado y dé [5 x 1] */
z={3 2, 5 1, 5 9};  /* produce una matriz llamada z dé [2 x 3] */
```

4.3. CARÁCTERES LITERALES:

```
a={abc defg};  /* produce un vector fila [abc , defg]  */
```

4.4. FACTORES DE REPETICIÓN:

```
r={ [2] 'si', [2] 'no'};  /* produce un vector r={'si' 'si', 'no' 'no'} */
```

4.5. OPERADORES

* : multiplicación de matrices o matriz por escalar

+ : adición, concatenación

- : sustracción

: multiplicación elemento a elemento

** : potenciación de matrices

: potenciación de cada elemento de una matriz

/ : división. Esta puede ser elemento a elemento si son dos matrices o matriz y escalar

<> : compara elemento a elemento dos matrices de igual dimensión y genera otra con los máximos

>< : compara elemento a elemento dos matrices de igual dimensión y genera otra con los mínimos

OR : lógico ó & : lógico y (puede ser AND)

< : menor que

<= : menor o igual que

> : mayor que

>= : mayor o igual que

^= : diferente de

= : igual a

|| : concatenación horizontal

Si tenemos las matrices a y b definidas como:

```
a={1 1 1, 7 7 7};
b={0 0 0, 8 8 8};
```

Y realizamos el siguiente comando:

c=a||b; El resultado será c={1 1 1 0 0 0,7 7 7 8 8 8},

// : concatenación vertical

Si tenemos las matrices a y b definidas como:

```
a={ 1 2 };
b={ 0 4 };
```

Y realizamos el siguiente comando:

```
c=a//b;
```

El resultado será c={1 2,0 4}

@ : producto Kronecker

Si tenemos las matrices a y b definidas como:

```
a={1 2, 3 4};
b={0 2};
```

Y realizamos el siguiente comando:

```
c=a@b;
```

El resultado será: c={ 0 2 0 4,0 6 0 8}

4.6. FUNCIONES QUE TIENEN COMO ARGUMENTO MATRICES

det(matriz) : calcula el determinante de una matriz cuadrada

diag(argumento): crea una matriz diagonal, el argumento puede ser una matriz cuadrada o un vector

CALL eigen(valores-propios, vectores-propios, matriz-simétrica): esta rutina regresa los valores y vectores propios de una matriz, por ejemplo,

```
a={1 2, 3 7};
call eigen(m,e,a);
print m e;
```

eigval(matriz-simétrica): retorna un vector columna con los valores propios

eigvec(matriz-simétrica) : retorna una matriz con los vectores propios exp(matriz): genera una matriz donde cada entrada corresponde a la función exponencial de cada elemento de la matriz dada en el argumento

ginv(matriz): calcula la inversa generalizada de Moore-Penrose

half(matriz): calcula la descomposición de Cholesky. Regresa una matriz triangular superior

hdir(matriz1,matriz2): ejecuta un producto directo horizontal

I(dimensión): crea una matriz identidad

inv(matriz): calcula la inversa de una matriz

loc(matriz): encuentra los elementos diferente de cero de una matriz y retorna un vector con su localización

log(matriz): regresa una matriz con los logaritmos de las componentes de la matriz de entrada

max(matriz): encuentra el máximo de una matriz

min(matriz): encuentra el mínimo de una matriz

ncol(matriz): encuentra el número de columnas de una matriz

nrow(matriz): encuentra el número de filas de una matriz

rank(matriz): retorna una matriz conteniendo el rango de cada elemento de la matriz

sqrt(matriz): calcula la raíz cuadrada de cada elemento

ssq(matriz): calcula la suma de cuadrados de todos los elementos de la matriz

sum(matriz): suma todos los elementos de la matriz

CALL svd(u,q,v,a): calcula la descomposición de valor singular. u, q, y v son las matrices de la descomposición retornadas y a es la matriz de entrada.

t(matriz): retorna una matriz que es la transpuesta de la matriz de entrada, es equivalente al comando `, por ejemplo A` es la transpuesta de A, lo mismo que t(A). Tenga cuidado de utilizar correctamente la "comilla" que se utiliza (`) y no confundirla con la comilla normal (').

trace(matriz): suma los elementos de la diagonal principal. Es la función traza.

vecdiag(matriz-cuadrada): retorna un vector columna que contiene los elementos de la diagonal principal de la matriz de entrada

Ejemplos de algunas estructuras de la programación IML.

4.7. IF-THEN/ELSE

Ejemplo: a) `if max(a)<20 then p=0;`

p=0 5


```
else p=1;
```

b) `if max(a)<20 then b=a;`

c) `if any(a^=b) then.....;`

Tenga en cuenta el uso del punto y coma que no es exactamente como en Pascal o C en este tipo de estructura.

4.8. GRUPOS DO:

Los grupos DO\END son bloques de programas que sirven para estructurar el programa adecuadamente. Es equivalente a la estructura Begin\End de Pascal o la agrupación { } en C.

La sintaxis en general es:

```
DO;
   comandos adicionales  un bloque completo
END;


DO;
   comandos;
   DO;
        comandos;    encajonamiento
   END;
END;
```

Aquí se muestra un ejemplo específico:

```
If x<y then
   DO;
       z1=abs(x+y);   un bloque
       z2=abs(x-y);
   END;
else
   DO;
       z1=abs(x-y);   un bloque
       z2=abs(x+y);
   END;
```

Note que la lógica es similar que en la programación estructurada de los lenguajes de 3ra generación. Para utilizar los IF anidados es muy útil estructurar e indentar el programa adecuadamente para observar la lógica del programa. La estructura equivalente para las instrucciones FOR es el DO interactivo. Veamos su sintaxis:

```
do i=10 to 100 by 10;
  comandos...
end;
```

También existe la equivalencia para la estructura WHILE: DO WHILE (expresión). Su sintaxis es:

```
do while (cuenta<5);
   print cuenta;
   cuenta=cuenta+1;
end;
```

La forma UNTIL es la siguiente:

```
DO UNTIL (expresión).
do until(cuenta>5);
    print cuenta;
    cuenta=cuenta+1;
end;
```

Recuerde que esta estructura se diferencia de la de WHILE en que ejecuta al menos una vez, mientras que en WHILE puede no ejecutarse ninguna vez si la condición no se cumple.

También existe la estructura de salto incondicional si se asocia una etiqueta. Recuerde que en un programa bien estructurado no se debe utilizar. Este salto puede convertir el programa en ilegible. La sentencia GOTO sólo puede utilizarse dentro de un bloque DO. Veamos su sintaxis y un ejemplo:

GOTO etiqueta;

```
do;
  if x<0 then goto negativo;
  y=sqrt(x);
  print y;
  stop;
  negativo:
   print "x es negativo";
end;
```


4.9. TRABAJO CON MATRICES

A continuación mostraremos algunos manejos de matrices con comandos subscritos de la matriz. Es decir son operaciones que se realizan internamente en la matriz. Consideremos la matriz:

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 5 & 4 & 3 \\ 7 & 6 & 8 \end{bmatrix}$$

Las operaciones más utilizadas son:

+ Suma

Multiplicación

<> Máximo

>< Minimo

<:> Indice del Máximo

>:< Indice del Mínimo

: Media

Suma de cuadrados

Recuerde que estas operaciones son internas de la matriz, veamos algunos ejemplos:

A[{2,3},+] produce el vector { 12, 21 } (suma las filas 2 y 3 y devuelve la respuesta como un vector columna de 2x1.

A[+,<>] produce {13} una matriz 1x1, el cual es el máximo de la suma de las columnas. El operador + en el lugar de las fila realiza la suma de cada fila, luego determina el máximo de los resultados.

A[<>,+] produce {21} una matriz 1x1 que corresponde la suma de los valores máximos de cada columna. Primero halla el máximo de cada columna luego suma los resultados.

A[,><][+,] produce {9} que corresponde a la suma de los valores mínimos de las filas. Primero halla los valores mínimos de cada fila, luego los suma.

A[,<:>] produce el vector columna {3,1,3} que corresponde al índice de los valores máximos por fila. Es decir, el 3 indica que el tercer número de la primera fila es el máximo, el 1 que el primer número de la segunda fila es el máximo y por último el 3 dice que el tercer número de la tercera columna es el máximo.

A[>:<] produce el vector fila {1 1 1} que muestra el índice de los valores mínimos por columna. Es decir que el primer 1 dice que el primer valor de la primera columna es el mínimo de esta columna y así sucesivamente.

A[:] produce {4}, es una matriz 1x1 que contiene la media de todos los elementos de la matriz.

Generación de secuencias

Veremos el uso de comandos de generación de vectores y matrices mediante algunos ejemplos:

```
I= 7:10;
```

Produce el vector columna {7,8,9,10}.

mientras que:

```
R=10:7;
```

produce el vector columna {10,9,8,7}

4.10. CREANDO MATRICES DE CONSTANTES

La función J(nfilas, ncolumnas, valor) crea una matriz de constantes, por ejemplo,

```
A=J(3,1,5);
```

genera un vector columna {5,5,5} de dimensión 3x1.

4.11. GENERACIÓN DE UN VECTOR CON NÚMEROS ALEATORIOS

Para generar una matriz con n filas y m columnas de números aleatorios uniformemente distribuidos se usa el comando uniform(repeat(0,n,m)), por ejemplo:

```
x=uniform(repeat(0,10,1));
```

Genera un vector columna con 10 números aleatorios de dimensión 10x1.

Para generar observaciones aleatoriamente de otras distribuciones podemos utilizar las siguientes funciones que también se utilizan en el paso DATA del SAS.

ranbin(semilla,n,p): Para la binomial.

rancau(semilla): Para la Couchy.

ranexp(semilla): Para la Exponencial, con media 1.

rangam(semilla,alfa): Para la Gamma. con beta = 1.

normal(semilla): Para la Normal(0,1)

rannor(semilla): Para la Normal(0,1).

ranpoi(semilla,lambda): Para la Poisson.

rantri(semilla,h): Para la Triangular. f(x)=2x/h

ranuni(semilla): Para la Uniforme.(0,1)

Nota: A veces puede ser más rápido generar números aleatorios en un paso DATA e importarlos luego al IML.

4.12. ORDENANDO UN VECTOR COLUMNA

Algunas veces se necesita ordenar algún vector y para esto no existe una función o comando específico. Para ordenar se realiza el siguiente programa. El ejemplo muestra como se ordena el vector x.

```
b=x;
x[rank(x)]=b;
```

Note como en la segunda instrucción se puede asignar a un comando un valor para obtener el resultado deseado.

4.13. ESCRITURA DE UNA MATRIZ A UN ARCHIVO EXTERNO EN ASCII

Es muy común que se requiera generar un archivo con los resultados generados a partir de algún programa realizado dentro del IML.

Veamos el siguiente ejemplo:

```
filename out 'misdat.dat';
file out;
    do i=1 to nrow(x);
        do j=1 to ncol(x);
            put (x[i,j]) 2.0 +2 @;
        end;
        put /;
    end;
closefile out;
```

Note como se utiliza la instrucción filename. Aquí se muestra como el nombre del archivo se asocia a un nombre en particular, en este caso el archivo fisico "misdat.dat" se asocia al nombre out, y es que se va a utilizar para referenciar la salida.

El comando para imprimir el resultado en el archivo es put, el siguiente número indica el formato de escritura, luego se indica los espacios en blanco entre números y luego un formato de salida específico que impide que cambie de línea en la escritura de los resultados. En la instrucción put /, indica un retorno de línea, es decir para que comience una nueva línea.

4.14. ESCRITURA Y EJECUCION DE MODULOS

Los módulos en IML equivalen a lo que en lenguajes de programación como en FORTRAN se conocen como subrutinas y funciones, o en Pascal son Procedimientos y funciones.

La sintaxis general en IML es la siguiente:

```
START <nombre> <(argumentos)> <GLOBAL (argumentos)>;
...instrucciones...
FINISH <nombre>;
```

Para ejecutarlos dentro del programa se invocan con

```
RUN nombre <(argumentos)>;
```

o con

```
CALL nombre <(argumentos)>;
```

Consideremos los siguientes ejemplos:

```
PROC IML;
a=10;
b=20;
c=30;

START ejemplo1;
 p=a+b;
 q=b-a;
 c=40;
FINISH ejemplo1;

RUN ejemplo1;
PRINT a b c p q;
QUIT;
```


En el ejemplo anterior el módulo ejemplo1 no tiene argumentos. Todos los cambios que ocurran en su interior son de carácter global una vez el módulo sea ejecutado. Todas las variables que se definan dentro del módulo siguen siendo vigentes y pueden ser llamadas posteriormente en cualquier parte del programa. Los cambios que se hagan a variables son permanentes también.

El resultado de la impresión será:

A	B	C	P	Q
10	20	40	30	10

Note que después de la ejecución del módulo,

- A sigue siendo 10
- B sigue siendo 20
- C ha cambiado a 40
- P y Q se crearon, se agregaron a la tabla de símbolos, y se les colocaron los valores de 30 y 10 respectivamente.

Ejemplo 2:

```
PROC IML;
a=10;
b=20;
c=30;
d=90;
START ejemplo2(x,y);
 p=x+y;
 q=y-x;
 y=100;
 c=40;
FINISH ejemplo2;

RUN ejemplo2(a,b);
PRINT a b c d;
QUIT;
```

En este ejemplo el módulo ejemplo2 posee dos argumentos x y y. Cuando un módulo tiene argumentos todos los cambios que se hagan sobre estos argumentos son permanentes a las variables referenciadas, una vez ejecutado el módulo los cambios ocasionados por el módulo en las variables de entrada quedarán así cuando se termine la ejecución del módulo.

El resultado de la impresión será:

A	B	C	D
10	100	30	90

Note que:

• A sigue siendo 10

• B ha sido cambiado a 100 ya que el correspondiente argumento Y fue cambiado a 100 en el módulo.

• C sigue siendo 30. Dentro del módulo, la variable local C fue asignado a 25, pero no hay correspondencia entre esta variable local C y la global, es decir son independientes.

• D continúa siendo 90.

Note también que dentro del módulo, las variables A, B, y D no existen. Fuera del módulo, las variables P, Q, X, y Y no existen.

```
PROC IML;
a=10;
b=20;
c=30;
d=90;

START ejemplo3(x,y);
 p=x+y;
 q=y-x;
 y=100;
 c=40;
 RETURN(c);
FINISH ejemplo3;

z= ejemplo3(a,b);
PRINT a b c d z;
QUIT;
```

En este ejemplo el módulo ejemplo3 posee dos argumentos x y y. Además tenemos una nueva línea: RETURN(c). Esta línea nos convierte el módulo en una función y la forma de ejecutarlo cambia. Será entonces necesario asignar la variable pasada en el comando RETURN a una variable dentro del programa para que ésta reciba el valor de la variable en el argumento. Nuevamente cuando un módulo

tiene argumentos todos los cambios que se hagan sobre estos argumentos son globales, es decir son permanentes y una vez ejecutado el módulo los cambios permanecen.

El resultado de la impresión será:

A	B	C	D	Z
10	100	30	90	40

Note que:

• A sigue siendo 10

• B ha sido cambiado a 100 ya que el correspondiente argumento Y fue cambiado a 100 en el módulo. Es decir que hay una correspondencia 1 a 1.

• C sigue siendo 30. Dentro del módulo, la variable local C fue asignado a 25, pero no hay correspondencia entre esta variable local C y la global, es decir son independientes.

• D continúa siendo 90.

• Z se coloca en 40, el cual es el valor de C como variable local en el módulo.

Note también que dentro del módulo, las variables A, B, D y Z no existen. Fuera del módulo, las variables P, Q, X, y Y no existen.

```
PROC IML;
a=10;
b=20;
c=30;
d=90;
START ejemplo4(x,y) GLOBAL(c);
 p=x+y;
 q=y-x;
 y=100;
 c=40;
 d=500;
FINISH ejemplo4;

RUN ejemplo4(a,b);
PRINT a b c d;
QUIT;
```

En este ejemplo el módulo ejemplo4 posee dos argumentos x y y. Además se usa la opción GLOBAL. Como el módulo tiene argumentos todos los cambios que se hagan sobre estos argumentos son

globales, es decir son permanentes y una vez ejecutado el módulo los cambios ocasionados por el módulo en las variables de entrada quedarán así cuando se termine la ejecución del módulo. Como también existe una variable pasada en el comando GLOBAL los cambios que esta variable sufra en el módulo serán permanentes.

El resultado de la impresión será:

A	B	C	D
10	100	40	90

Note que:

• A sigue siendo 10

• B ha sido cambiado a 100.

• C se cambió a 40 porque fue declarada como GLOBAL. La variable C dentro como fuera del módulo es la "misma".

• D continúa siendo 90 y no 500, ya que D existe independientemente en las variables locales y globales.

Note también que cada argumento tiene su propia tabla de variables, y por lo tanto es posible tener una tabla global y varias locales. Una variable puede existir independientemente en una o más de estas tablas. Sin embargo, una variable puede ser compartida entre las tablas globales y locales siempre que se use la cláusula GLOBAL.

Se invita a los usuarios del SAS a que manipulen el programa, realicen ensayos, y que puedan entender y aprovechar las capacidades de este programa. Es muy importante el manual del respectivo PROC que se está utilizando para entender todas las opciones disponibles.

REFERENCIAS

- SAS Institute Inc., SAS Language Reference: Concepts, Cary, NC: SAS Institute Inc., 1999. 554 pages.
- SAS Institute Inc., SAS/IML User's Guide, Version 8, Cary, NC: SAS Institute Inc., 1999. 846 pp.
- SAS Institute Inc., SAS/GRAPH® Software: Reference, Version 8, Cary, NC: SAS Institute Inc., 1999.
- SAS Institute Inc., The Analyst Application, First Edition, Cary, NC: SAS Institute Inc., 1999. 476 pp.
- SAS Institute Inc., SAS/STAT® User's Guide, Version 8, Cary, NC: SAS Institute Inc., 1999.
- DiIorio, F. C. (1991) SAS Applications Programming: A Gentle Introduction. Duxbury Press, Belmont.
- Cody, R. P. y Smith, J. K. (1991) Applied Statistics and the SAS Programming Language. Tercera Edición. Prentice Hall, Englewood Cliffs.