

Using PROC IML® to do Matrix Balancing

Carol Alderman, University of Kansas
Institute for Public Policy and Business Research

Introduction

The application described here is called matrix balancing. In this case, matrix balancing refers to forcing a given matrix to conform to known row and column totals. Several algorithms have been developed to do this. The one used here is an old method called the RAS algorithm.

PROC IML was used to implement this algorithm. The functions within PROC IML made it fairly easy to translate the algorithm to SAS® code.

The RAS Algorithm

The RAS Algorithm works by alternately scaling the rows and the columns of a matrix. It multiplies each row by a positive constant, forcing the sum of each row to equal the known sum for that row. It then does the same thing to the columns. Adjusting the rows changes the columns and vice versa, so the algorithm iterates between row and column scaling until the matrix is balanced.

The following is the formal definition of the RAS algorithm from Schneider and Zenios (1988):

Let A be an $m \times n$ nonnegative matrix, and u and v be positive vectors of dimensions m and n , respectively.

Step 0: (Initialization) Set $k = 0$ and $A^0 = A$.

Step 1: (Row scaling) For $i = 1, 2, \dots, m$ define

$$\rho_i^k = \frac{u_i}{\sum_j a_{ij}^k}$$

and update A^k by

$$a_{ij}^k \leftarrow \rho_i^k a_{ij}^k,$$

$i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

Step 2. (Column Scaling) For $j = 1, 2, \dots, n$ define

$$\sigma_j^k = \frac{v_j}{\sum_i a_{ij}^k}$$

and define A^{k+1} by

$$a_{ij}^{k+1} = a_{ij}^k \sigma_j^k,$$

$i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

Step 3. Replace $k \leftarrow k + 1$, and return to Step 1.

SAS Program

IML stands for Interactive Matrix Language. PROC IML uses matrices as its fundamental objects. As will be shown below, it has predefined functions for doing everything which is needed to program this algorithm.

The program can be divided into four parts: entering what data to use, checking that the data are valid, running the algorithm, and examining the results.

Before the program is run, the user must modify the first section of the program by filling in the values for a list of macro variables. Figure 1 shows the section of the

program which must be modified by the user. Areas in bold are what the user supplies.

```

1. %LET ORIGMAT = SASDAT67.MATRIX;
   /*Matrix to perform RAS on*/
2. %LET ROWTOT = SASDAT67.ROWTOT;
   /*File containing row totals*/
3. %LET COLTOT = SASDAT67.COLTOT;
   /*File containing column tots*/
4. %LET RASMAT = SASDAT67.NEWMAT;
   /*File to put RASsed matrix*/
5. %LET ROWS = 9;
   /*Number of rows*/
6. %LET COLS = 10;
   /*Number of columns*/
7. %LET TIMES = 10;
   /*Number of iterations*/

```

Figure 1: User Settings

The code in Figure 2 brings the data into PROC IML and checks to make sure it is in the correct form. Statement 19 resets the option LOG to NOLOG so that if any errors are found the print statements will print to the log instead of the listing.

The first check is to ensure that A is nonnegative, which is a requirement of the RAS algorithm. To do this the MIN function is used (statement 21). The MIN function returns the smallest element in the matrix. The program checks if this element is less than zero. If it is, an error message is printed to the log and the ERROR flag is set to zero.

Next, the program checks that A is $M \times N$, i.e. it conforms to the user supplied dimensions. This is accomplished by using the NROW and NCOL functions (statements 25 and 29). NROW returns the number of rows in the matrix, while NCOL returns the number of columns.

The next check ensures that the vectors ROWTOTS and COLTOTS are positive. ROWTOTS corresponds to u in the algorithm and COLTOTS corresponds to

```

8. PROC IML;
9.  RESET NOCENTER;
10. M = %ROWS;
11. N = %COLS;
12. T = %TIMES;

13. USE %ORIGMAT;
14. READ ALL INTO A;

15. USE %ROWTOT;
16. READ ALL INTO ROWTOTS;

17. USE %COLTOT;
18. READ ALL INTO COLTOTS;

   /* Check that matrix and vectors
      conform to specifications. */

19. RESET LOG;
20. ERROR = 0;
   /* A must be nonnegative */
21. IF MIN(A) < 0 THEN DO;
22.   PRINT Error Message;
23.   ERROR = 1;
24. END;
   /* A must be M x N */
25. IF NROW(A) ^= M THEN DO;
26.   PRINT Error Message;
27.   ERROR = 1;
28. END;
29. IF NCOL(A) ^= N THEN DO;
30.   PRINT Error Message;
31.   ERROR = 1;
32. END;
   /* ROWTOTS & COLTOTS must be positive */
33. IF MIN(ROWTOTS) <= 0 THEN DO;
34.   PRINT Error Message;
35.   ERROR = 1;
36. END;
37. IF MIN(COLTOTS) <= 0 THEN DO;
38.   PRINT Error Message;
39.   ERROR = 1;
40. END;
   /* ROWTOTS must be M x 1 */
41. IF NROW(ROWTOTS) ^= M THEN DO;
42.   PRINT Error Message;
43.   ERROR = 1;
44. END;
45. IF NCOL(ROWTOTS) ^= 1 THEN DO;
46.   PRINT Error Message;
47.   ERROR = 1;
48. END;
   /* COLTOTS must be 1 x N */
49. IF NCOL(COLTOTS) ^= N THEN DO;
50.   PRINT Error Message;
51.   ERROR = 1;
52. END;
53. IF NROW(COLTOTS) ^= 1 THEN DO;
54.   PRINT Error Message;
55.   ERROR = 1;
56. END;
   /* Quit if error has been found */
57. IF ERROR = 1 THEN DO;
58.   PRINT Abort Message;
59.   ABORT;
60. END;

61. RESET NOLOG;

```

Figure 2: Check Data

v . A requirement of the algorithm is that u and v be positive. This is accomplished

using the MIN function (statements 33 and 37).

Statements 41-56 check that ROWTOTS and COLTOTS are the correct dimensions, using the NROW and NCOL functions.

The last check is to see if any errors have been found. In the previous checks, if an error was found the variable ERROR was set to equal one. So in this check if ERROR equals one the program is aborted.

Statement 61 resets NOLOG so that further print statements will print to the listing instead of the log.

The first statement in the RAS algorithm is now completed. A is a $m \times n$ nonnegative matrix, and ROWTOTS (u) and COLTOTS (v) are positive vectors of dimensions m and n respectively.

Figure 3 shows the section of the program which performs the RAS algorithm.

```

62. K = 0;
63. DO UNTIL (K = T);

    /* Row scaling */
64. I = 1;
65. R = J(M,1);
66. DO UNTIL (I > M);
67.  ROWSUM = A[I,+];
68.  R[I,1]=ROWTOTS[I,1]/ROWSUM;
69.  I = I + 1;
70. END;
71. A = R # A;

    /* Column scaling */
72. J = 1;
73. S = J(1,N);
74. DO UNTIL (J > N);
75.  COLSUM = A[+,J];
76.  S[1,J]=COLTOTS[1,J]/COLSUM;
77.  J = J + 1;
78. END;
79. A = A # S;

80. K = K + 1;
81. END;

```

Figure 3: RAS Code

The user has supplied the number of times this section will run, now called T. In the RAS algorithm, k is the variable which is incremented for each pass through the algorithm; in statement 62 of the RAS code,

K is set to zero. The first DO UNTIL statement (63) controls the process of iterating through the RAS algorithm. Statement number 80 increments K by one at the end of each pass. The RAS code will stop iterating when K equals T. Since it started at zero, it will iterate T times.

The first step in Row Scaling is finding the value

$$\rho_i^k = \frac{u_i}{\sum_j a_{ij}^k}$$

for $i=1,2,\dots,m$. The DO UNTIL statement (66) controls the successive calculations of

ρ_i^k (R in the program) with i being incremented in statement 69. Statement 65 defines the dimensions of R using the J function.

The denominator in the equation above is the sum of the i th row in A. PROC IML provides a function to automatically sum the row of a matrix - X[S,+], where X is the name of the matrix and S is the row to sum. Statement 67 sets the vector ROWSUM to equal the sum of row I in A.

The numerator in the above equation is the element in the vector of known row totals (ROWTOTS) in row i . PROC IML provides the ability to reference a single element in a matrix by using subscripts - e.g. X[a,b], where X is the name of the matrix, a is the row number, and b is the column number. So the numerator in this equation is ROWTOTS[I,1].

In statement 68 the element in vector R for the particular row is calculated. Statement 71 updates A^k . This is done by doing element-by-element multiplication of the vector R and matrix A using the "#" operator.

Column scaling is done in a statements 72-79. It is very similar to row scaling. In this case sums of the columns in

A are needed. A[+,J] performs this function (statement 75).

The last section of the program, which examines and reports the results, is shown in Figure 4.

```

82. CREATE &RASMAT FROM A;
83. APPEND FROM A;

84. RDIFF = ROWTOTS - A[+,+];
85. ROWDIFF = ABS(RDIFF);

86. CDIFF = COLTOTS - A[+,+];
87. COLDIFF = ABS(CDIFF);

88. ROWMAX = ROWDIFF[<>];
89. COLMAX = COLDIFF[,<>];

Print Statements

```

Figure 4: Examine Results

First, in statements 82 and 83 the RASsed matrix is made into a permanent SAS dataset using the CREATE and APPEND commands. Different calculations are then performed which show the user how close the matrix is to being balanced.

Statement 84 calculates the difference between the known row totals and the RAS produced row totals, by subtracting one from the other. The function A[+,] gives the vector containing the sums of the rows of A. Statement 86 calculates the difference between the known column totals and the RAS produced column totals in the same way.

		SALESMEN									
		1	2	3	4	5	6	7	8	9	10
P	A	230	375	375	100	0	685	215	0	50	0
R	B	330	405	419	175	90	504	515	0	240	105
O	C	268	225	242	0	30	790	301	44	100	0
D	D	595	380	638	275	30	685	605	88	100	160
V	E	340	360	440	200	30	755	475	44	150	0
C	F	132	190	200	0	0	432	130	0	0	0
T	G	309	330	350	125	0	612	474	0	50	50
S	H	365	400	330	150	50	575	600	44	150	110
I	I	210	250	308	125	0	720	256	0	100	50

Figure 5: Estimated Collections

The ABS function is used in statements 85 and 87 to change the values of the difference vectors into their absolute values.

Statement 88 finds the maximum difference between the known row totals and RAS produced row totals. It does this by using the element maximum operator "<>". Statement 89 calculates the maximum difference between column totals and RAS produced column totals in the same way.

The rest of this section prints the results of these calculations to the listing.

Example

As an example of a situation where this algorithm may be useful, let us consider a sales manager with ten salesmen and nine types of products to sell.

Suppose the salesmen get the products from a warehouse at the beginning of the day; then they go out and sell all day. At the end of the day, they turn in the products they have not sold to the warehouse, and they give the money they collected to the manager. Later the sales manager is asked by his supervisor to find out how much money each salesman collected per product. By checking with the warehouse, the manager knows how much of each product was sold. Since he knows how much each product costs he knows the total amount that was collected on each product. He also knows how much money each salesman gave him. He asks the salesmen to tell him how much they collected for each product. They do not remember exactly, but they estimate as well as they can. Figure 5 shows their estimates.

The sums of the rows and columns of this matrix do not equal the amounts the manager knows they should be. See Figure 6.

The manager applies the RAS algorithm to this problem. Figure 7 shows the results.

Row Totals										
Estimated:			Known:			Difference:				
2030					2029					1
2783					2798					-15
2000					1998					2
3556					3566					-10
2794					2794					0
1084					1071					13
2300					2305					-5
2774					2747					27
2019					2015					4
Column Totals										
Actual:										
2779	2915	3302	1150	230	5758	3571	220	940	475	
Known:										
2772	2910	3300	1150	240	5760	3526	220	950	495	
Difference:										
7	5	2	0	-10	-2	45	0	-10	-20	

Figure 6: Known and Estimated Row and Column Totals

Conclusion

Computing the RASsed algorithm required many specialized matrix functions, including: a) finding the minimum element of a matrix, b) counting the number of rows and columns of a matrix, c) summing columns and rows of a matrix, d) referring to individual elements in a matrix, e) doing element-by-element multiplication of matrices, f) subtracting matrices, g) creating a matrix of absolute values, and h) finding the maximum element in a row and column of a matrix. PROC IML provided every function needed.

Trademark Notice

SAS, and SAS/IML are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Bibliography

Schneider, Michael H.; and Stravos A. Zenios. 1988. "A Comparative Study of Algorithms for Matrix Balancing", Operations Group Report Series #88-02, John Hopkins University.

Author

Carol Alderman
Institute for Public Policy and Business
Research
University of Kansas
607 Blake Hall
Lawrence, Kansas 66045
Telephone: (913) 864-3701
Internet:
ALDERMAN@UKANVM.CC.UKANS.
EDU

RASSed Matrix

		SALESMEN									
		1	2	3	4	5	6	7	8	9	10
P	A	229.65	374.87	375.10	100.03	0.00	686.24	212.54	0.00	50.57	0.00
R	B	330.59	406.19	420.49	175.63	94.30	506.58	510.79	0.00	243.54	109.89
O	C	267.31	224.69	241.81	0.00	31.30	790.60	297.25	44.02	101.04	0.00
D	D	595.26	380.61	639.42	275.61	31.39	687.58	599.25	88.30	101.34	167.23
U	E	339.66	360.06	440.34	200.16	31.35	756.75	469.81	44.09	151.79	0.00
C	F	130.33	187.82	197.82	0.00	0.00	427.95	127.08	0.00	0.00	0.00
T	G	309.48	330.90	351.17	125.42	0.00	614.98	470.02	0.00	50.73	52.32
S	H	360.59	395.63	326.60	148.46	51.66	569.95	586.87	43.60	150.11	113.54
I		209.12	249.25	307.26	124.70	0.00	719.38	252.40	0.00	100.87	52.02

Row Differences

Difference between known row totals and
RAS produced row totals:

Maximum difference between known row
totals and RAS produced row totals:

ROWDIFF
3.979E-12
5.059E-12
3.581E-12
5.230E-12
6.821E-13
3.183E-12
6.821E-13
3.809E-12
2.274E-12

ROWMAX
5.23E-12

Column Differences

Difference between known column totals and RAS produced column totals:

COLDIFF

5.116E-13 2.274E-13 2.27E-13 5.684E-14 1.066E-14 0 2.274E-13 5.329E-14 5.684E-14 0

Maximum difference between known column totals and RAS produced column totals:

COLMAX

5.116E-13

Figure 7: RASSed Solution