PhUSE 2009

PaperCC03

Sudoku's resolution Using SAS® Arrays

Michel Gauthier, Novartis, Basel, Switzerland

ABSTRACT

The Phuse Contest of 2008 was about creating a SAS code that would solve a Sudoku's grid. I submitted my solution using a three dimensional array, and won the prize. I know that the use of arrays is not the most popular piece of code in the SAS world, let alone two dimensional arrays, and it goes without saying that three dimensional arrays do not find a practical use in daily practice. But it was the ideal solution to solve the Sudoku's grid. During my presentation I will try to demystify for you the uses of array, and explain my solution.

.

INTRODUCTION

This paper will cover some high level insight / reminders on SAS Arrays. The rules of Sudoku will be explained, and a step-by-step approach will be outlined to solve part of the game. This approach will be transposed into SAS code using two dimensional arrays. Once the concept is established the three dimensional final solution will be revealed.

FEW PRINCIPLES ON ARRAYS

One can declare within an array a succession of variables, not necessarily of the same name (Prefix + suffix, as day1 day2 …Dayn) as long as the list of variables are of same nature and definition (Char, Numeric …)

These variables do not need to exist already; they could be created by the declaration, as new variables or temporary variables.

ARRAY Wkday(5) Monday Tuesday Wednesday Thursdays Friday;
As long as the 5 variables are of the same nature and definition, they will be used if they exist and created otherwise.

ARRAY Wkday(5) ; will create 5 numerical variables wkday1, wkday2 … wkday5.

ARRAY Wkday(5) $; Will create 5 character variables wkday1, wkday2 … wkday5.

ARRAY Wkday(5) _temporary_ ; will created 5 numerical variables, which are dropped at the end of the datastep.

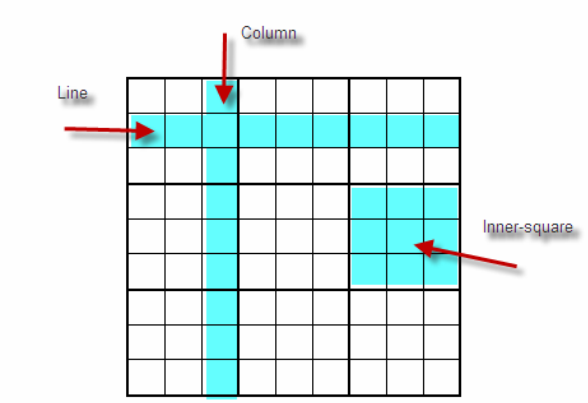
The variables (not the temporary) can be accessed at any time directly either by their names or, via the reference position in the array.

The variable Wednesday, and the reference wkday(3) in the first example are the same variable.

The variable Wkday3, and the reference wkday(3) in the second and third example are the same variable.

FEW PRINCIPLES ON SUDOKU GAME

The goal is to complete a 9x9 grid with the numbers 1 to 9 where the same number cannot appear twice in the same line, the same column or in one of the nine 3x3 inner-squares composing the general grid.



On a general note, due to those three forementioned rules it is usually easier to determine where it is not permitted to lay down a number than to find a valid position to introduce a new number in the grid. But as the numbers do need to be put down somewhere I figured that by identifying a situation where there is only one spot available for a given number on a Line, Column, or Inner-Square, then this is where the number as to be.

Let's use the submission query of the 2008 Phuse contest as the starting grid (Figure 1)

Submission Grid

						4	5	7
	4	7		9	1			
3	8				9			
6			2	1	4			
			7				2	9
			9	3		1	8	
5	9	1						

Figure 1

If we would like to solve the grid by attempting to insert a number 3, we will first highlight all positions where any number already exists, as impossible positions to insert a number 3 (Figure 2)

All current positions are considered valid therefore it is safe to assume that those positions are positions where we cannot insert a number 3.

Next, for every line where a number 3 is already present, all elements of that line will also be highlighted, as impossible positions to insert a 3 (Figure 3)

Similarly, for every column where a number 3 is already present, all elements of that column will also be highlighted, as impossible positions to insert a 3. (Figure 4)

						4	5	7
	4	7		9	1			
3	8				9			
6			2	1	4			
			7				2	9
			9	3		1	8	
5	9	1						

Figure 2

Finally, for every Inner-square where a number 3 is already present, all elements of that inner-square will also be highlighted, as impossible positions to insert a number 3. (Figure 5)

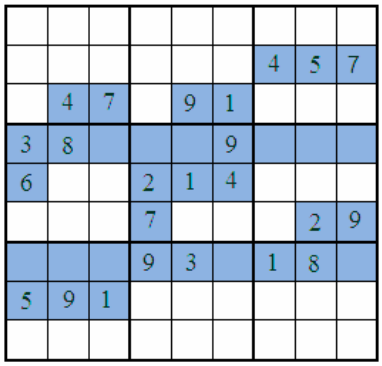


Figure 3

By eliminating (highlighting) all positions where it is impossible to insert a number 3 (Figure 5) conversely we have shown all remaining places that may contain a number 3, but as rules says, there can only be one number 3 per line, per column and per inner square.

Looking at each line, column and Inner-Square we are now looking for a situation where a SINGLE open position is available for that line, column or Inner Square.

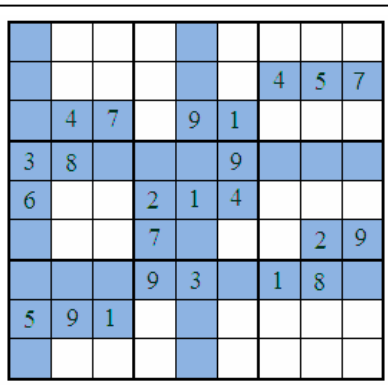


Figure 4

From the figure 5 we can deduce that the element on line 6 column 6 must be a number 3, because there is only one open position for the number 3 in the Inner-Square [line 4-6, column 4-6]. On this occasion there were no situations where a line or column was carrying a single open position. That is basis of the logic that I applied to solve the game.

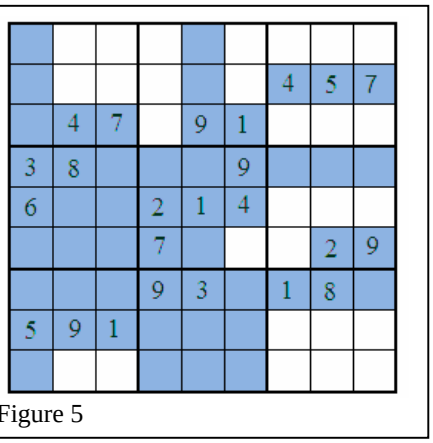


Figure 5

LET'S DO THE SAME EXERCISE IN SAS WITH ARRAYS

We will work with two arrays, one called SUDOKU carrying the initial grid (Figure 1), one called level3 carrying the Boolean check (equivalent to the highlight in the previous pictures). *We will used the level3 for now, just to mimic the previous examples, once this knowledge has been translated into SAS code, we will look into a more global solution, and the use of a 3D array to solve all nine numbers*

```
ARRAY Sudoku (9,9) S1-S81 ; /*values already capture in the dataset*/
ARRAY level3(9,9) _temporary_ ;   /* all of its values are initiated as missing numerical.*/

  do col = 1 to 9 ;
   do lin = 1 to 9 ;
     if sudoku(lin,col) > 0 then level3 (lin,col)= 1 ;
   end;
  end;
```

Via two embedded loops, we scan through the 81 positions of the 9x9 SUDOKU grid.
This piece of code is analogous to figure 2. Instead of highlighting the positions where a number is present, we set a one(1) in the equivalent positions in the Level3 grid. Because the coordinate [Line, Column] used by the loop in the SUDOKU grid checking if the position carries a number [*sudoku(lin,col) > 0*]are immediately used to set the LEVEL3 grid at the same coordinate to set it to one(1)[*level3 (lin,col)= 1* .]

Within the same code it is fairly easy to add the code that would mimic the LINE and COLUMN highlighting, as in figure 3 and 4.

```
do col = 1 to 9 ;
 do lin = 1 to 9 ;
  if sudoku(lin,col) > 0 then level3 (lin,col)= 1 ;
  if sudoku(lin,col) = 3 then do ;
    do _cc = 1 to 9 ;
      do _ll = 1 to 9 ;
        level3 (_ll,col) = 1 ;
        level3 (lin,_cc) = 1 ;
      end;
    end ;
   end;
  end;
 end;
end;
```

The line of code *if sudoku(lin,col) = 3 then do ;* determines at what coordinate a 3 is present on the SUDOKU grid. From this coordinate [Line, Column] we must fill all other position on the same LINE and on the same COLUMN in the LEVEL3 grid.

To fill the column : we lock the **line** but loop from 1 to 9 at the column level, by introducing a new temporary variable to loop the Column[_cc] :

```
do _cc = 1 to 9 ;
   level3 (lin,_cc) = 1 ;
```

To fill the line : we lock the **column** but loop from 1 to 9 at the line level, by introducing a new temporary variable to loop the Line[_ll] :

```
do _ll = 1 to 9 ;
  level3 (_ll,col) = 1 ;
```

Doing the same in the SUDOKU grid for the 9 inner squares, requires a double set of coordinates to give the start line and end line of the square [1-3,4-6,7-9] and start and end column as well [1-3,4-6,7-9]. A macro accepting these parameters will do.

```
%macro CheckSqre(col_start,col_end,lin_start,lin_end);
   do col = & col_start. to & col_end. ;
    do lin = & lin_start. to & lin_end. ;
     if sudoku(lin,col) = 3 then do ;
       do _cc = & col_start. to & col_end. ;
         do _ll = & lin_start. to & lin_end. ;
           Level3(_ll,_cc) = 1 ;
         end;
       end ;
     end;
    end;
   end;
%mend;

%CheckSqre(1,3,1,3);
%CheckSqre(1,3,4,6);
%CheckSqre(1,3,7,9);
%CheckSqre(4,6,1,3);
%CheckSqre(4,6,4,6);
%CheckSqre(4,6,7,9);
%CheckSqre(7,9,1,3);
%CheckSqre(7,9,4,6);
%CheckSqre(7,9,7,9);
```

The part in bold with the imbedded loop on the column and line, is there to search within the boundary set by the macro call, while looping within these nine positions of the Inner-Square we are looking for a 3 present in the SUDOKU grid [*if sudoku(lin,col) = 3 then do*], when one is found a second imbedded loop on the column and line is created with temporary variables [_ll, _cc] still within the boundaries set by the calling coordinate to write on the LEVEL3 grid, hence covering all 9 positions in the LEVEL3 with 1's.

The macro call *%CheckSqre(4,6,1,3) and*%CheckSqre(7,9,4,6); reflects the transition from figure4 to figure5.

GOING THREE DIMENSION

The first explanations were based on a single number (3), it is obvious that this search and find technique needs to be applied on all nine numbers of the game, I could have defined nine two dimensional grids 9 (9x9) , but decided to use one 3D instead … 9x9x9. Having the nine LEVEL grid laid on top of each other, this will allow searching across level at a later stage.

The games numbers are contained in the SUDOKU array (9x9) but now instead of using level3 array to carry the Boolean values [.,0,1], I will define the CHECK array (9x9x9)

```
data resolution  ;
set base ;

array sudoku(9,9) I1-i81 ;
array check(9,9,9) _temporary_ ;
```

To *highlight* the position where a particular number cannot be present I have this code: Instead of looking only for a 3, An extra loop is added ***level*** that does the same job, but for all nine numbers of the game

```
do level = 1 to 9 ;
  do col = 1 to 9 ;
    do lin = 1 to 9 ;
      if sudoku(lin,col) > 0 then check(lin,col,level)= 1 ;
      if sudoku(lin,col) = level then do ;
        do _cc = 1 to 9 ;
          do _ll = 1 to 9 ;
            check(_ll,col,level ) = 1 ;
            check(lin,_cc,level ) = 1 ;
          end;
        end ;
      end;
    end;
  end;
end;
```

…

As seen earlier, a similar macro is defined for the Inner-square checks and all nine combinations are called

```
%CheckSqre(1,3,1,3);
%CheckSqre(1,3,4,6);
…
```

Looping on ***level*** will set the CHECK grid, one level at the time, filled with missing and 1, corresponding to possible / impossible positions for the number that represents that level (Level one is for number 1, level two is for number 2, ect.).

Once the positions where a number is (permissible/non-permissible), are well defined, we need to identify which of the permissible positions are unique on a single line, column or Inner-Square for a specific level..

```
do level = 1 to 9 ;
  do col = 1 to 9 ;
    dotfound = 0 ;
    do lin = 1 to 9 ;
      dotfound = dotfound + (check(lin,col,level ) = .);
    end;
    if dotfound = 1 then do ;
      do lin = 1 to 9 ;
        if check(lin,col,level ) = . then sudoku(lin,col) = level ;
      end;
    end ;
  end;
end;
```

Looping through the Level, Column and Line, resetting a counter *dotfound* to zero at each new column, we count the number of missing values in the CHECK grid under that Level for each column. If the number of missing found in a single column is one, that means that for this level, in that column, only one open position is found for that level and column. Therefore at the coordinate of that Column, where the Line is set to missing in the CHECK grid, the SUDOKU grid is updated with the Level Value.

A similar exercise can (and must) be done for the LINE and the INNER-Square.

In Short, there are three different checks done on the SUDOKU grid to set the permissible / non-permissible flag in the 3D CHECK grid, Line, column, Inner-Square. There are also three searches that can be performed on the CHECK grid to set the numbers back on the SUDOKU grid, Line, Column, Inner-Square.

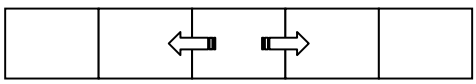
Obviously, as soon as you change the SUDOKU grid, a new set of checks needs to be done back on the CHECK grid. You need a program that will loop until the 81 numbers are placed into the Sudoku grid

For *Easy level* games, this will suffice, for harder one, most likely not. This is where the third dimension delivers its benefits.

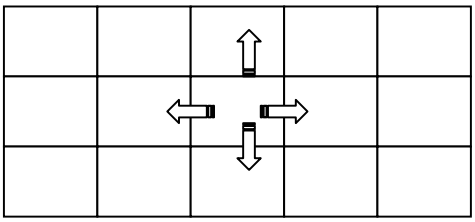
It is possible that on a single level alone you cannot come to a conclusion because on the permissible positions left on that level, more than one position is possible for each Line, Column, and Inner-square.

Arrays, in all cases (one, two or three dimensional) have all of their variables pertaining to this array stored on a single record.

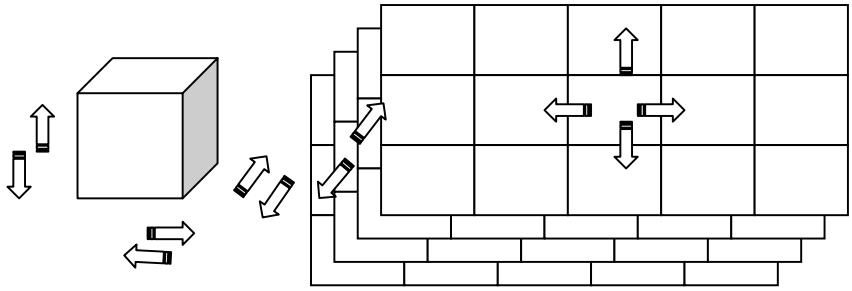
A single dimensional array is a linear succession of variable where neighboring variables are the next one, or the previous one.



In two dimension arrays the logical representation is more of a rectangle, where neighboring variables could be next and previous one (as in one dimension array) but also upper and lower variables are also neighbors.



Although the CHECK array is physically a succession of 729 variables (9x9x9), its logical disposition is one of a cube, where neighboring variables could be next and previous one upper and lower one, and on top or under.



Remember that all nine Check grids are layered on top of each other. If instead of looking for a Single permissible position on a Line, or a Column of any given Level, we were searching for a single permissible Level, at a given coordinate [Line, Column]. Instead of scanning on the X of Y level of the cube as previous exercise, we will drill down the Z axis

Basically the exercise consists into looking into the different coordinates of the SUDOKU grid that are not filled yet and track how many permissible open positions are left at that coordinate in the different levels of the CHECK grid, if the permissible condition is available for only one level, then the level where that open position is found will become the number used to fill the SUDOKU grid at those coordinates.

```
do lin = 1 to 9 ;
   do col = 1 to 9 ;
    alone = . ;
    do level = 1 to 9 ;
      if check(lin,col,level) = . then do ;
        if (alone=.) then alone = level ;
        else alone = 0 ;
      end;
    end;
    if alone then sudoku(lin,col) = alone ;
  end;
 end;
```

Looping through the coordinates [Line, Column] , at each new coordinate set a counter variable *alone* to missing, loop through the level, look for a missing value, if a missing value is found on those coordinate set *alone* to the value of the level, if more than one missing is found forfeit the search by setting *alone* to zero (more than one level with permissible flag on those coordinate). If that single missing value is found then promote the value of that level on those coordinate [Line, Column] on the SUDOKU grid.

Reset your CHECK grid.
Keep searching until you found all 81 numbers.

Good luck.

.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michel Gauthier
Novartis Pharma AG
Postfach
CH-4002 Basel
SWITZERLAND
+41 61 32 40407
Email: Michel.gauthier@novartis.com