Eight Queens Puzzle Solution Using MATLAB
EE2013 Project

Matric No: U066584J

January 20, 2010

1 Introduction

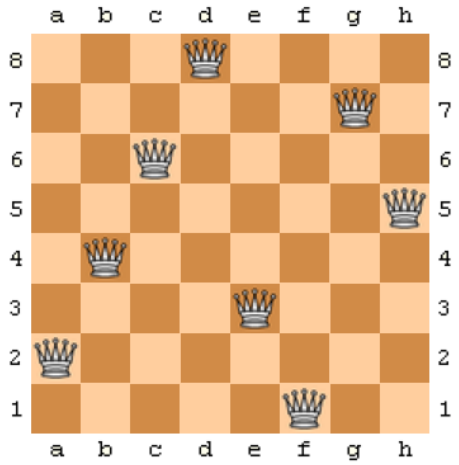

Figure 1: One of the Solution for Eight Queens Puzzle

The eight queens puzzle is the problem of putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The queens must be placed in such a way that no two queens would be able to attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

Nowadays, the eight queens puzzle can be solved easily and almost instantly with the help of a computer. Various programs have been created to solve this puzzle, some with even more queens at a larger board. In this project, I will solve the eight queens puzzle using MATLAB programming language, and further improve the program to tackle puzzles with more or lesser queens.

2 Algorithm

To solve this puzzle in a systematic way, I have divided this problem into a few stages: Initialisation, Update, Checking, and Printing Stages.

Initialisation Stage
Matrices to be used will be initialised at this stage. Size of the matrices depend on the user input, corresponding to number of queens in the queens puzzle.

Update and Checking Stages
The chess board is updated with the placement of queen at the available empty space at each iteration. Then checking will be done to validate the previous placement. If the placement is valid, the process will be continued with the Update and Checking Stages until all eight queens are filled. If the placement is void, it will roll back to the previous move and search for the next available empty space to place the queen.

Printing Stage
When all eight queens are placed correctly on the board, the final matrix will be printed out.

The idea was translated into the following flow chart to design the algorithm to solve this problem.

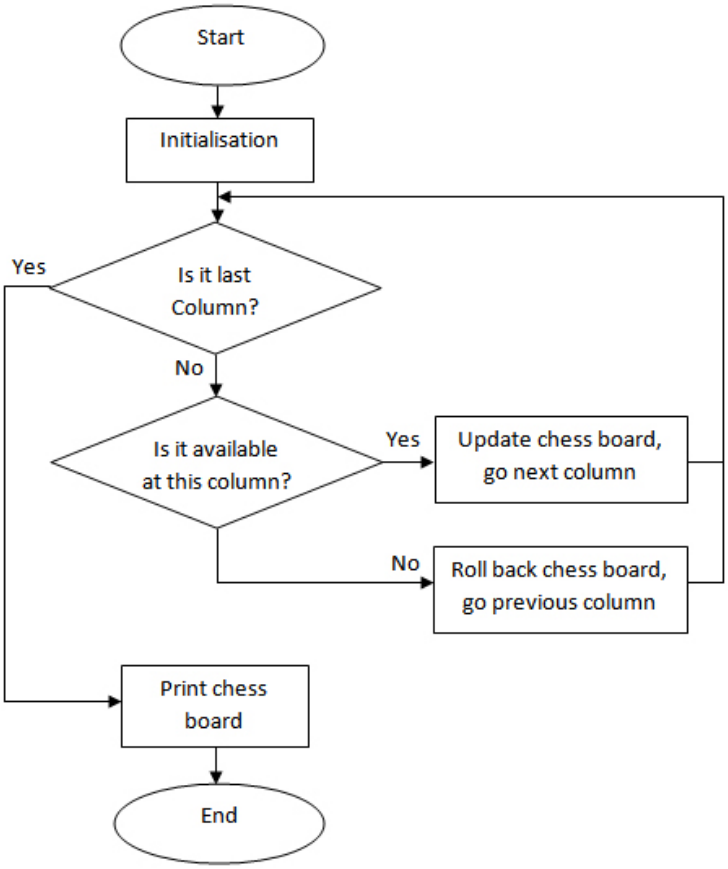

Figure 2: Flow Chart of the Program

3 Programming

In this section, I will comment about the actual coding of the program. First, I have created two subfunctions to be used in the main program function.

3.1 Subfunction `queensetupboard.m`

Calling this subfunction creates a `size` by `size` matrix to represent the actual chess board. A matrix **C** of the same size will also be created to perform updates and checking of the available space. Lastly, a three dimensional matrix is created to store the last value of the matrix **C** at each column loop. It is used for rolling back of previous matrix **C** if they are needed.

3.2 Subfunction `queenupdate.m`

This subfunction input the operation matrix together with the coordinate of the new queen. It will return an updated operation matrix with all the non available spaces marked as 1 while the rest 0. It also return a non-updated operation matrix with the coordinate of queen placed marked as 2 for roll back purposes. It will replace the updated matrix in the main function if the current queen placement is found to be invalid later.

Technically, the non available spaces to be marked would be the row, column, and the diagonal of the given coordinate. Marking of row and column of matrix can be done easily using the function `matrix_A(row,:)=1` and `matrix_A(:,column)=1`. Diagonal of $(row, column)$ would be at all the coordinates of $(row \pm 1, column \pm 1)$ as shown in the codes.

3.3 Main Function `queenpuzzle.m`

An input number from users will be used to determine the number of queens and also the size of the chess board. Initially, the program will generate a few matrices as explained in the first subfunction above.

In the computational part, a queen will be placed at the available space of the current column (column starts counting from 1) of the matrix. If there is no available space left, the matrix will be rolled back to the previous matrix and it will search for new available space at the previous column. Or else if the queen successfully placed, the matrix will be updated to marked out all the non available spaces (row, column and diagonal of queen). The loop will be ended when the column number equal to the size of the matrix. By then, a solution should be obtained. For the case that there is no solution for the puzzle, an empty matrix will be returned.

Finally, the matrix representing the chess board will be updated according to the solution, with 1 represent queen, and 0 represent empty space on the board.

4 Source Codes

4.1 queensetupboard.m

```
1     % This function initialise matrices to be used in an
2     % <size> Queen Puzzle solution.
3     function [matrix_A, matrix_B, matrix_C] = queensetupboard(size)
4     % to create an empty board for working
5     matrix_A = zeros(size,size);
6     % to store previous matrix for retrieving purposes
7     matrix_B = zeros(size,size,size);
8     % the real solution matrix
9     matrix_C = zeros(size,size);
10    end
```

4.2 queenupdate.m

```
1     % This function update the chess board after a queen is
2     % placed. (row and column is the index of the new queen)
3     function [matrix_A, matrix_prev] = queenupdate(matrix_B, row, column)
4     matrix_prev = matrix_B;
5     matrix_prev(row,column) = 2;
6
7     n = length(matrix_B);
8     % Update Board
9     matrix_A = matrix_B;
10    % Update Row
11    matrix_A(row,:) = 1;
12    % Update Column
13    matrix_A(:,column) = 1;
14    % Update Diagonal (only needed for those at the right)
15    % First loop to update upper diagonal
16    column_temp = column;
17    row_temp = row;
18    while (column_temp < n && row_temp > 1)
19        column_temp = column_temp + 1;
20        row_temp = row_temp - 1;
21        matrix_A(row_temp,column_temp) = 1;
22    end
23    % Second loop to update lower diagonal
24    column_temp = column;
25    row_temp = row;
26    while (column_temp < n && row_temp < n)
27        column_temp = column_temp + 1;
28        row_temp = row_temp + 1;
29        matrix_A(row_temp,column_temp) = 1;
30    end
31    end
```

4.3 queenpuzzle.m

```
1   % This is the main program to solve an <size> Queens Puzzle
2   function matrix_solve = queenpuzzle(size)
3
4   % Create Board
5   [matrix_now, matrix_stage, matrix_solve] = queensetupboard(size);
6
7   % Loop for column starting from 1 and end at <size>
8   column = 1;
9   while (column <= size)
10      % Check availability for current column
11      available = find(matrix_now(:,column) == 0, 1);
12
13      % If is available, 1) update solution board
14      % 2) update final result 3) increment column count
15      if(available > 0)
16          [matrix_now, matrix_stage(:,:,column)] ...
17              = queenupdate(matrix_now, available, column);
18          matrix_solve(available,column) = 1;
19          column = column + 1;
20
21      % If is not available, 1) decrease count (go back to previous column)
22      % 2) clear final result 3) retrieve the old solution board
23      else
24          if column == 1
25              break
26          end
27          column = column - 1;
28          matrix_solve(:,column) = 0;
29          matrix_now = matrix_stage(:,:,column);
30      end
31  end
32  end
```

5 Results

The result of the puzzle can be obtained by entering `queenpuzzle(size)` in MATLAB Command Window where `size` is the number of queen desired. Results for eight, nine, and ten queens puzzle are shown below.

```
>> queenpuzzle(8)

ans =

     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     1
     0     1     0     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     0     1     0     0
     0     0     1     0     0     0     0     0

>> queenpuzzle(9)

ans =

     1     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0
     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     1
     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0
     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     1     0     0

>> queenpuzzle(10)

ans =

     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     1     0     0     0     0
     0     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     1     0     0     0     0     0
```