CS1112-206/211/212–Fall 2009

Lab 8 Solution

Friday, October 23

1 Contour Plotting

(a) `m` is the number of grid points on each row and each column. The higher `m` is, the more refined the grid is.

(b) `nContours` is the number of intervals used to draw contours. This is used in the `linspace` function on the line below. The lower `nContours` is, the higher the difference between the values of the two contours next to each other. This is because the values on the contours that are drawn are exactly the values in the vector (`v`) passed in to `contour`.

(c) `axis equal square` format the axis so that the scale on both axes is the same. That is, it takes the same distance on the screen from $x = 1$ to $x = 2$ and from $y = 1$ to $y = 2$. It also make the overall plot square, meaning x and y at the end of both axes are the same.

(d) `clabel` lets you click on the contour to display the value at the point you click. Remember to press Enter when you are done clicking (instead of closing the figure window).

(e) The original program works for $q = 3$ only. If q is arbitrary, this loop body will not work because it has not considered all the distances from $(x(j), y(i))$ to all the random points. To fix this, we need to have a program structure that can handle an arbitrary number of points, provided that the number of points is known. A `for` loop sounds promising here. What we have to do is to make sure to go over all the random points and calculate the distance from our current point $((x(j), y(i)))$ to each of those random points. In the original program, we store the three distances in three variables. How do we allocate storage that can handle an arbitrary number of points? A vector seems a good option.

Therefore, what we have to do is to create a vector d of length q such that $d(i)$ is the distance from our current point to random point i. Once we are done calculating the distances, we need to take the minimum. The `min` function can take a vector, so `A(i,j)=min(d);` is sufficient here.

(f) We want to find a point P on the grid such that the distance from P to its closest random point (city) is as large as possible. This means that for this point P and some other point Q, the distance from P to its closest city must be at least the distance from Q to its closest city.

What information do we have here? We have the matrix A that stores the distance from each point in the grid to the city closest to that point. Hence, having A, what we have to do is to find the maximum value in A. The problem would be this easy if we just had to display what this maximum value is, but it also asks you to highlight the point on the grid. Hence, in addition to knowing the maximum value, we also have to locate where this point actually is. This can be done by keeping the indices of the matrix when a new maximum is set.

Lastly, we have to display the point. We have the indices of the point, but not the coordinate values of the point itself. Hence, we have to refer to `x` and `y` to get coordinate values.

The course webpage does not have a solution to this part, so you can find it below:

```
[nr,nc]=size(A);

% find maximum value and its location in A
max=-1;
for row=1:nr
    for col=1:nc
        if A(row,col)>max
            max=A(row,col);
            maxX=col;
            maxY=row;
        end
    end
end

% plot the location of maximum value
plot(x(maxX),y(maxY),'or');
```

Note that we can set the initial `max` to be -1 because we know that any distance (in A) must be nonnegative. Be careful about which variable refers to column and which refers to row.

2 Sudoku Checker

We are provided the function `OkVec` that can handle any length-9 (row or column) vector. Hence, we just have to *treat it as a black box* and use it correctly and efficiently. What do we have to do here? We need to consider each row, each column, and each Sudoku submatrix. For the rows and columns, we can simply pass it on to `OkVec` and we are done. For the submatrices, we need to transform it into a correct format that `OkVec` accepts first.

The fragment `A(i,:)` refers to row i of the table. Similarly, the fragment `A(:,j)` refers to column j of the table. What about the submatrix beginning whose top-left cell is (i, j), where $i, j = 1, 4, 7$? In order to use `OkVec`, we need to transform this into a row or a column vector. The most convenient way is to use vectorized code. The fragment

```
[A(i,j:j+2) A(i+1,j:j+2) A(i+2,j:j+2)]
```

does the job. The result is a length-9 *row* vector. Once we have this vector, we can now simply pass it on to `OkVec`.