CS1112-202/206/207–Spring 2010

Section 11 Solution

Friday, April 16

1 Where to Put Your Long-Burning Light Bulb?

First, let us clarify what `digitToShow()` is supposed to do. It is best to think that the function has its own internal state, keeping track of how many times it has been called, so that it returns the next digit to be displayed correctly. For instance, calling it third time results in the third digit to be displayed. Once again, you do not have to know the implementation of the function—the idea of black boxes applies here. Also, you cannot assume that each digit is equally likely to occur. For example, the function might always return 1, i.e., the display board always displays the number 1. Otherwise, if each digit is equally likely to come up, we can simply count which lightbulb is turned on most and just put the longlasting bulb there.

The general strategy for both parts are the same, and we discuss it now.

We need to keep calling `digitToShow()` for a certain number of times to ensure that we have a good approximation of the likelihood distribution for each number. That is, we must obtain a sequence of digits to be displayed from the beginning. How many digits should we go for? Ideally, it should be large enough that some bulbs burn out. Since we don't know this number, simply aim for a high number, say 100000.

We also need the information about which light bulbs are on for each possible digit. For this, we simply call `TheDigits()` to obtain the cell array containing the information.

For each time we have the next digit, we have to keep track of the statistics that will will use later to determine the position we should put the longlasting light bulb in. Because the display board is two dimensional, the best representation of statistical variables is a matrix: Each component of the matrix keeps track of the frequency of our interest of its corresponding light bulb. That is, we first initialize a zero 7-by-5 matrix. Then, for each digit in the sequence, we update the components of the matrix that should be updated, depending on the condition of our interest.

Once we are done with all the digits in the sequence, we have to find out which position we should put the longlasting bulb in. Assuming that we want to place the light bulb in the position such that its corresponding frequency (in the statistic matrix) is highest, we simply have to determine the *location* of the maximum value in the frequency matrix. Once we have the location, we can just display to the screen to let the user know.

So far, we have the skeleton that looks like the following:

```
M = 100000; % repeat M times of random change
nRow = 7;   % size of the digit board
nCol = 5;

digits = TheDigits();

freq=zeros(7,5);
```

```
for k = 1:M
    % the next digit
    nextn = digitToShow();

    % update freq
end

% find location of maximum value
```

Updating the frequency in the `for` loop will depend on condition of interest, which we discuss below. Finding the location of a maximum value of `freq` can be done as follows. First, we initialize the maximum value to be -1 (and we will update it as we go along), corresponding to row 0 and column 0 (which does not exist, but we will update later anyway). Then, we need to go through each row and column of `freq` and see if the value in that component is greater than the maximum. If it is, we update the maximum, along with the current row and column numbers. When we are done with all the components, we will have the correct row and column numbers of the maximum value of `freq`. We can then display these two numbers.

Turning the above idea into the code, we can now append the following to the code above:

```
maxVal=-1;
maxRow=0;
maxCol=0;
for r=1:nRow
    for c=1:nCol
        if freq(r,c)>maxVal
            maxVal=freq(r,c);
            maxRow=r;
            maxCol=c;
        end
    end
end
fprintf('Row %d, col %d should use longlasting bulb.\n',maxRow,maxCol)
```

We now discuss the specific strategy for each case:

1. Put the bulb where it is turned on most:

   In this case, we need to update the frequency matrix for the light bulb at position (r, c) if that light bulb is on for the current digit *nextn*, that is, if `digits{nextn}(r,c)` is 1.

   Caveat: If the digit is 0, `digits{0}` is undefined. The light information for digit 0 is stored in `digits{10}`. Hence, we need to readjust the index if the digit is zero.

   In essence, if the light information is 1 for a component, then we add 1 to `freq`. That is, we are effectively adding the light information (which is a 0-1 matrix) to `freq`. Therefore, the frequency update part will look like the following:

```
% readjust index if necessary
if nextn==0
    nextn=10;
end

freq=freq+digits{nextn};
```

2. Put the bulb where it is switched on/off most:

   In this case, we need to update the frequency matrix for the light bulb at position (r, c) if that light bulb changes its state (i.e., from on to off, or from off to on) from the last digit. This suggests immediately that we need to keep track of the last digit in the sequence. What is the last digit before the first digit? Here we need to make a design decision whether we want the display board to display the first digit right away, or we start from every light bulb off and then display the first digit. We will go with the latter approach.

   For this approach, everything is off in the beginning, so the state of the display board is such that every component is zero. We need to initialize this before we go into the loop. That is, we insert the following line right after we initialize `freq`:

```
lastn=zeros(7,5);
```

   Once we have the next digit, we compare it with the last digit. For the light bulb at position (r, c), we add 1 to `freq(r,c)` if the state of the bulb at position (r, c)—i.e., `digits{nextn}(r,c)`—is different from `digits{lastn}(r,c)`. Note that if the two values are different, their difference must be either 1 or -1. That is, the absolute value of their difference is 1. If the two values are the same, the difference is 0. Hence, we add 1 to `freq(r,c)` if `abs(digits{nextn}(r,c)-digits{lastn}(r,c))` is 1. Note that we still have the same caveat as before.

   In effect, we are adding the absolute value of the difference between `digits{nextn}` and `digits{lastn}` (which is a 0-1 matrix) to `freq`. Therefore, the frequency update part will look like the following:

```
% readjust index if necessary
if nextn==0
    nextn=10;
end

freq=freq+abs(digits{nextn}-digits{lastn});
```

   Finally, before we go on to the next digit, we have to replace the last digit with the current digit, resulting in the following line before the end of the `for` loop:

```
lastn=nextn;
```

Note that we can combine these two cases into the same script by renaming variables to avoid conflicts. Since we can use the same sequence of numbers, we only need to rename the frequency matrix for each case to be unique.

2 Challenge Question

The strategy of solving this problem remains the same, except for at the end. Instead of finding the location of the maximum value in the frequency matrix, we now have to find the locations of five largest values in the matrix.

Hence, instead of keeping one value, we now have to multiply everything by five. That is, we will have a vector of length 5 for each of the largest value, the row number, and the column number. Specifically,

- `maxVal` is a vector of length 5 such that `maxVal(i)` is the i^{th} largest value. For instance, the maximum (first largest) value resides in `maxVal(1)`.
- `maxRow` and `maxCol` are vectors, each of length 5, such that

$$\texttt{freq}(\texttt{maxRow(i)}, \texttt{maxCol(i)}) == \texttt{maxVal(i)}.$$

Now, when we encounter a value v, we begin by comparing it with `maxVal(5)`. If it is less than `maxVal(5)`, then we know that it cannot be among the five maximum values, and we are done. Otherwise, `maxVal(5)` loses its place because—at least—v will be the 5th largest value. That is, we can update `maxVal(5)` and its corresponding row and column numbers to be those of v.

v might not be the 5th largest value, however. We are not done yet. It might be the case that v is greater than `maxVal(4)`, in which case we should swap the information of the 4th and 5th largest values. We need to keep doing this—going down to the first component of the vectors—until we find the appropriate place for v.

We can turn the above idea into the code as follows:

```
M=5;  % make this a parameter so we can easily adjust it if we want
maxVal=-ones(1,M); % vector of -1's
maxRow=zeros(1,M);
maxCol=zeros(1,M);
for r=1:nRow
    for c=1:nCol
        if freq(r,c)>maxVal(M) % larger than last largest value
            maxVal(M)=freq(r,c);
            maxRow(M)=r;
            maxCol(M)=c;

            % move the new value to the left to make maxVal sorted
            k=M-1;
            while k>=1 && maxVal(k)<maxVal(k+1)
                tmp=maxVal(k); maxVal(k)=maxVal(k+1); maxVal(k+1)=tmp;
                tmp=maxRow(k); maxRow(k)=maxRow(k+1); maxRow(k+1)=tmp;
                tmp=maxCol(k); maxCol(k)=maxCol(k+1); maxCol(k+1)=tmp;
                k=k-1;
            end
        end
    end
end
```

The following is what you will learn later in the course, but we will make a connection between it and our solution now.

Observe that the above algorithm essentially performs the `Insert` operation as used in insertion sort. We need to keep inserting (i.e., move left) a value as long as the value left to it is less than itself. Here we are inserting to achieve the descending order, while the regular insertion sort aims for ascending order.