CS1112 Spring 2015 Project 4 Part A due Thursday 3/26 at 11pm

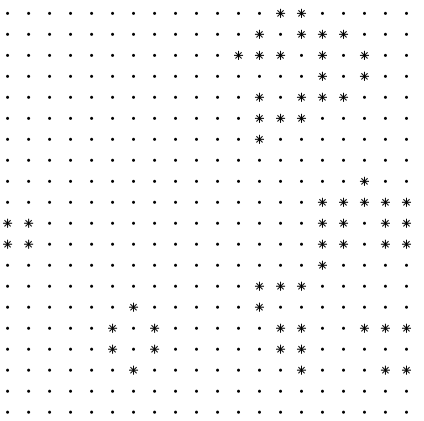
You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of 2-dimensional and 3-dimensional arrays. In Part A, you will write a function to simulate Conway's Game of Life. Additionally, you will use MATLAB as a tool to solve a system of linear equations. In Part B, you will perform image processing.

1 Conway's Game of Life

Conway's "Game" of Life is not actually a game that one plays; rather it is a simulation from some initial state following a set of rules. The state of the game is a grid (matrix) of elements, sometimes called cells. A cell can be either alive or dead. In one step of the game, the cells will either live or die according to some rules. You will write three functions to simulate this game. The rules stated below are taken from the Wikipedia entry on Conway's Game of Life.



1.1 One step

Implement the following function as specified:

```
function outM = oneSweep(inM)
% One step of Conway's Game of Life.
% inM and outM are 0-1 matrices of the same size.  1 is live; 0 is dead.
% inM is the initial state; outM is the state after 1 step of the game.
% Every "cell" in the matrix interacts with its eight neighbors, which
% are the cells that are horizontally, vertically, or diagonally adjacent.
% At each step in time, the following transitions occur:
%   A live cell with fewer than 2 live neighbors dies (under-population).
%   A live cell with 2 or 3 live neighbors lives on.
%   A live cell with more than 3 live neighbors dies (overcrowding).
%   A dead cell with exactly 3 live neighbors becomes live (reproduction).
```

One step of the game involves visiting every cell and determining its new state. Be sure that for every cell you use the initial state when computing the transition.

Helpful syntax and function

```
m = [ 2 4 2 6; ...
      0 1 7 4; ...
      9 3 5 8];
x = m(2:3,1:3);  % x is 2-by-3 matrix containing [0 1 7; 9 3 5]
y = sum(x);      % y is row vector storing column sums of x: [9 4 12]
z = sum(y);      % z is sum of vector y, also the sum of matrix x: 25
```

1.2 Draw the state of the game

Implement the following function as specified:

```
function drawState(m, s)
% m is a 0-1 matrix.  s is the step number in the game.
% Draw an asterisk at (c,-r) if m(r,c) is 1 (live).
% Draw a dot at (c,-r) if m(r,c) is 0 (dead).
% Display the step number in the title area of the figure.
```

In order to draw the state of the Game of Life grid, we need to relate row and column numbers to x and y coordinates. The *column* number maps to the x-coordinate, but notice that row numbers increase going down while the y-coordinate increases going up, which is why we map row `r` to the y-coordinate `-r`.

Begin with the commands `cla` for clearing (refreshing) the figure axis area and `hold on`. End with the command `hold off`. Choose any color combination you like and you can experiment with different marker and font sizes. See Appendices A.6 and A.10 in *Insight*.

1.3 The game, finally

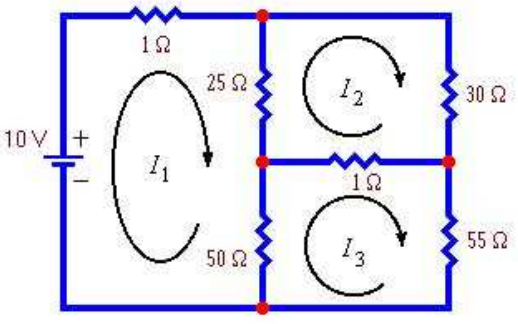
Implement the following function as specified:

```
function m = gameOfLife(m, n)
% Simulate n steps of the Game of Life.
% Pre:  m is a 0-1 matrix representing the initial state.  m is not empty.
% Post: m is a 0-1 matrix representing the state after n steps of the game.
```

In this function you will call your functions `drawState` and `oneSweep` repeatedly in order to simulate the game. Below is a skeleton of the function showing and explaining the relevant graphics commands.

```
close all
figure
axis equal off

% Call drawState to show initial state

% Simulate n steps
for k= 1:n
    pause(.1)  % During program development, change to pause() so that you
               % can manually advance the simulation and check each step

    drawnow  % Force Matlab to complete drawing in the figure window before
             % moving on.  Useful in animation (in loop) when the computation
             % gets done faster than drawing on the screen.

    % Write your code to simulate one step and draw the state

end
```

2 System of Linear Equations



Systems of linear equations often arise from the modeling of systems of interconnected elements. In this problem, you will learn to use MATLAB as a solver of systems of linear equations. Your only real task is to turn a set of given equations into a matrix and a vector and then use the solver, which is just a MATLAB operator, to get the answer. We do not expect you to know linear algebra or Kirchoff's Voltage law. So there's no need to be afraid of the application or the math—we give you the equations! The discussion below is adapted from *Introduction to Computing for Engineers* by Chapra and Canale.

Consider a set of n linear algebraic equations of the general form

$$\begin{array}{llll} a_{11}x_1+ & a_{12}x_2+ & \ldots+ & a_{1n}x_n = \quad b_1 \\ a_{21}x_1+ & a_{22}x_2+ & \ldots+ & a_{2n}x_n = \quad b_2 \\ \quad . & \quad . & & \quad . \quad \quad . \\ \quad . & \quad . & & \quad . \quad \quad . \\ \quad . & \quad . & & \quad . \quad \quad . \\ a_{n1}x_1+ & a_{n2}x_2+ & \ldots+ & a_{nn}x_n = \quad b_n \end{array} \tag{1}$$

where the a's are known constant coefficients, the b's are known constants, and the n unknowns, x_1, x_2, $\ldots$, x_n, are raised to the first power. (I.e., a_{rc} is the coefficient of x_c in the rth equation.) This system of equations can be expressed in matrix notation as

$$\mathbf{Ax} = \mathbf{b}$$

or

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{bmatrix} \tag{2}$$

"Solving" this linear system of equations is to find the values x_1, x_2, $\ldots$, x_n such that $\mathbf{Ax} = \mathbf{b}$. In MATLAB, the solution can be found using the backslash, called the *matrix left divide* operator:

```
x = A\b
```
(3)

where `A` is the n-by-n matrix of coefficients, `b` is the length n *column* vector of constants, and the result `x` is the length n column vector of values such that $\mathbf{Ax} = \mathbf{b}$.

Your job: Write a script `circuit.m` to find the three currents, I_1, I_2, and I_3, flowing in the circuit shown above. From Kirchoff's Voltage Law, we get the following three equations:

$$\begin{aligned} 1I_1 + 25(I_1 - I_2) + 50(I_1 - I_3) &= 10 \\ 25(I_2 - I_1) + 30I_2 + 1(I_2 - I_3) &= 0 \\ 50(I_3 - I_1) + 1(I_3 - I_2) + 55I_3 &= 0 \end{aligned}$$

You need to

1. Rewrite these equations in the general form of (1) by collecting terms together, where the unknowns are I_1, I_2, and I_3.
2. Create the 3-by-3 matrix `A` and length 3 column vector `b` as shown in (2).
3. Apply the matrix left division operator as shown in (3) to solve for I_1, I_2, and I_3.
4. Finally display the values of I_1, I_2, and I_3.

Submit the files `oneSweep.m`, `drawState.m`, `gameOfLife.m` and `circuit.m` on CMS. Part B of Project 4 will appear in a separate document.