CS1112 Fall 2014 Project 4 due Monday 10/27 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 Problem 1, you will practice working with matrices and submatrices. In Problem 2, you will use MATLAB as a (black box) tool to solve a system of linear equations. In Problem 3, you will work with the jpeg image format and the type `uint8`. Pay attention to the difference between `uint8` and MATLAB'S default type `double`.

1 Sudoku Checker

Sudoku is a popular Japanese puzzle game that rose to international popularity in 2005. At the higher levels of difficulty, the puzzles are difficult to solve for both human and machine! In this project, you will write a "Sudoku checker" that determines whether a candidate solution is valid—we are not writing a program to solve the puzzle. If you are not familiar with the game, consult the Wiki at `http://en.wikipedia.org/wiki/Sudoku` and try to solve a few puzzles at `http://www.sudoku.com`!

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

Sudoku puzzle

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

Solution

A Sudoku solution is valid if all the following properties are true:

1. The provided solution fits in a 9-by-9 grid.
2. The entire board contains only integers in the range [1, 9].
3. Each row contains the numbers [1..9], with no repeated values.
4. Each column contains the numbers [1..9], with no repeated values.
5. Each 3-by-3 sub-grid (delineated by heavy lines in the puzzle and solution above) contains the numbers [1..9], with no repeated values.

You will write these two functions: `isValidPartition` and `isValidSudoku`.

`isValidPartition` accepts a 9-element array—a length 9 vector or a 3-by-3 matrix—and checks that it contains the numbers [1..9] with no repeated values (properties 3, 4, and 5 above). `isValidPartition` returns true (1) or false (0).

`isValidSudoku` accepts a 9-by-9 matrix that represents a solution to a Sudoku puzzle and returns true (1) if the puzzle solution is valid or false (0) otherwise. This function should make effective use of `isValidPartition`.

You will write the function headers as well as concise and informative function comments for these functions.

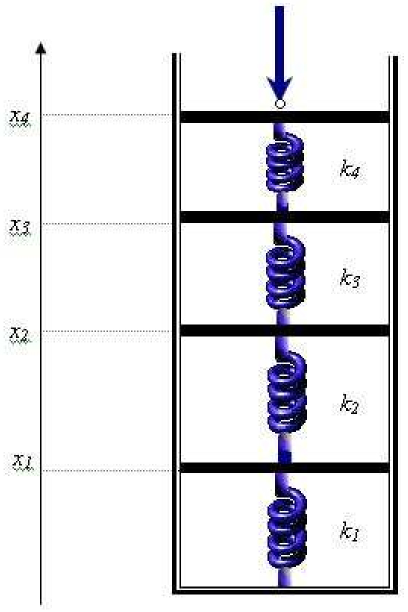
To help you with testing, we provide two example 9-by-9 matrices below. Be sure to do additional testing!

```
validSol = [1 7 2 5 4 9 6 8 3; ...
            6 4 5 8 7 3 2 1 9; ...
            3 8 9 2 6 1 7 4 5; ...
            4 9 6 3 2 7 8 5 1; ...
            8 1 3 4 5 6 9 7 2; ...
            2 5 7 1 9 8 4 3 6; ...
            9 6 4 7 1 5 3 2 8; ...
            7 3 1 6 8 2 5 9 4; ...
            5 2 8 9 3 4 1 6 7]

incorrectSol = [1 7 5 8 3 9 4 2 6; ...
                6 3 8 2 7 4 9 1 5; ...
                4 2 9 6 5 1 3 7 8; ...
                8 1 6 3 9 6 7 4 2; ...
                5 4 7 1 6 2 8 3 9; ...
                2 6 3 4 2 7 6 5 1; ...
                7 5 4 9 2 6 1 8 3; ...
                9 8 1 5 4 3 2 6 7; ...
                3 6 2 7 1 8 5 9 4]
```

Submit your files `isValidPartition.m` and `isValidSudoku.m` on CMS.

2 System of Linear Equations



Systems of linear equations often arise from the modeling of systems of interconnected elements. In your physics class you may have studied the displacement that results from applying a force to a set of blocks connected by springs. Such a model gives a coupled set of equations that must be solved simultaneously; the equations are coupled because the individual parts of the system are influenced by other parts.

[Our objective for this problem is to show how you can use MATLAB *as a solver of systems of linear equations. We do not expect you to know linear algebra nor are we trying to teach linear algebra. We will use the solver as a "black box," so your only real task is to turn a set of* <u>*given*</u> *equations into a matrix and a vector and then use the solver, which is just an operator. So there's no need to be afraid of the physics or math! This discussion and problem is adapted from Introduction to Computing for Engineers by Chapra and Canale.]*

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

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

where the a's are known constant coefficients, the b's are known constants, and the n unknowns, x_1, x_2, ..., x_n, are raised to the first power. This system of equations can be expressed in matrix notation as

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

or

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

"Solving" this linear system of equations is to find the values x_1, x_2, ..., 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 `springMass.m` to find the positions x_1, x_2, x_3, and x_4 of the spring-mass system shown above when a force F of 2000 lbs is applied. The spring constants k_1 through k_4 are 100, 50, 75, and 200 lb/in, respectively. The force-balance equations that define the relationships among the springs are

$$\begin{aligned} k_2(x_2 - x_1) &= k_1 x_1 \\ k_3(x_3 - x_2) &= k_2(x_2 - x_1) \\ k_4(x_4 - x_3) &= k_3(x_3 - x_2) \\ F &= k_4(x_4 - x_3) \end{aligned}$$

You need to

1. Rewrite these equations in the general form of (1) by collecting terms together, where the unknowns are x_1, x_2, x_3, and x_4.
2. Create the 4-by-4 matrix `A` and length 4 column vector `b` as shown in (2).
3. Apply the matrix left division operator as shown in (3) to solve for x_1, x_2, x_3, and x_4.
4. Finally display the values of x_1 through x_4 neatly.

Submit your file `springMass.m` on CMS.

3 Operation zoom

You will write a function `multiZoom` that allows a user to display a jpeg image and select multiple areas to "zoom." Implement the function as specified:

```
function newIm = multiZoom(jName)
% Display an image and "zoom in" user-selected rectangular regions.
% jName is the string that names a JPEG image in the current directory.
% newIm is the uint8 array storing the data of the last "zoom detail."  If no zoom
%   detail has been calculated then newIm is the empty uint8 array, uint8([]).
%
% Detailed specifications:
% After displaying the image named by jName, repeat the following:
%   - Accept 2 user mouseclicks on the original image which define a rectangular area.
%   - Compute the "zoom detail": perform 2-d interpolation on the selected area of the
%     image.
%   - Display the zoom detail in a separate figure window.
% If the user mouse clicks indicate a very small area (4 pixels or smaller in width
%   and/or height), consider that to be a double click which indicates that the user
%   wishes to stop.
```

Note the following:

- We define a "double click" to be two user mouseclicks that are very close together—4 pixels by 4 pixels on the image—or separated by 4 or fewer pixels either horizontally or vertically. The function should stop accepting mouse clicks and end after the user "double clicks."

- Your code should work with any two user mouseclicks for defining a rectangle. That is, the user's second click can be to the top, bottom, left, and/or right of the first click.

- The statement `[a,b] = ginput(1)` returns the x-coordinate, i.e., *column* number of a pixel, in `a` and the *reversed* y-coordinate, i.e., *row* number of a pixel, in `b`. (Reversed y-coordinate means that the y-values at the top of the image are smaller than those at the bottom of the image. The top row of pixels is row 1.) Depending on which function is used to show the image (`imshow` or `image`), and depending on whether the figure window has been resized (by the user or automatically by MATLAB due to a small screen size), the values returned by `ginput` may not be integers! Therefore, be sure to round (`floor` or `ceil`) the returned values from `ginput` before using them as indices of an array.

- If the user clicks outside the image (in the gray area of the figure window), use the nearest pixel in the image as the clicked point. For example, if the user clicks above and to the right of the image, then use the top right pixel as the clicked point.

- Do 2-dimensional interpolation as described in Discussion Exercise 9: add one data point between neighboring pairs of data. For a 3-d array (resulting from a color image), do the 2-d interpolation on the red, green, and blue layers separately. Be careful with *types*! The example in Discussion 9 is for the default numeric type (`double`); you will perform interpolation on values of type `uint8` in this project.

- Since we want to return to the original image after displaying a zoom detail, we need to be able to identify the window that shows the original image. Use the `figure` command: `figure(`k`)` creates a figure window and numbers it k where k is a positive integer, or if a figure window k already exists then that window is made active—shown on top of other windows. The command `figure` on its own starts a new figure window and steps up the figure window counter. Therefore, before displaying the original image, use the command `figure(1)` (or just `figure`). Then the subsequent display of a zoom detail can be coded like this:

  ```
  figure      % start new figure window
  imshow(newIm)
  title('Detail of selected area')
  pause(2)
  figure(1)  % bring figure 1 forward, i.e., make it the active figure
  ```

  The `pause` command is used so that the user can see the zoom detail for a moment before the original image (in figure window 1) is brought forward. Note that you don't lose the window holding the zoom detail—it is simply behind the active figure window. Start function `multiZoom` with the command `close all` which closes all figure windows.

- Use the `title` command appropriately to give instructions to the user on the figure window (e.g. "Make two clicks to select a rectangular area", "goodbye", etc.)

- Most MATLAB installations on campus include the function `imshow`. If `imshow` is not an available function in the version of MATLAB that you are using, you can use built-in function `image` instead. `image` uses the default figure window size instead of sizing the window to the actual image like `imshow` does, but it doesn't change the actual computation that you need to do.

- You may use *sub*functions if you wish.

Submit your file `multiZoom.m` on CMS.