ACM 11, SPRING 2016, MATLAB PROJECT

Due: Monday, May 17

Submit by 23:59:59 PDT, as a zip file `LastnameFirstnameMATLABproject.zip` to acm11spring2016@gmail.com, with a subject line "Lastname Firstname Matlab Project."

Background

The MATLAB project is a very small research problem that takes a few hours. Emphasis is on setting up and formulating an interesting problem, not on writing hundreds of lines of code.

There are two options for the project: either choose your own topic and problem (I encourage you to choose this option), or pick one of the problems suggested below. If you choose your own problem, you must email the TA or instructor and get permission. In the email, describe the problem and how you plan on solving it. If you choose your own problem, it can be relatively easy.

The project must be your own work. If you choses one of the problem described below, you are allowed to discuss the details of the problem with other students who work on the same problem. But you are not allowed to exchange code, your code should be written by you.

Your code must be very well documented, and should explain what you are doing. Don't force the grader to redo your project just to figure out what you were doing!

General Requirements

- Vectorize where possible.
- Use sparse matrices where appropriate.
- Comment and document code properly. Provide help text and use cells to organize.
- Produce well-structured and readable code.
- Produce useful and properly labeled plots and visualizations.

Project 1: Google PageRank

Read section 2.11 in Chapter 2 of Moler's online textbook, `http://www.mathworks.com/moler/lu.pdf`, and do problems 2.23, 2.25 and 2.26. The dataset `harvard500` and function `pagerank` are available on the piazza page.

Project 2: Rare Event Estimation

Read the following paper `http://arxiv.org/abs/1508.05047` and implement the Monte Carlo method for the illustrative example in Section VI. Bonus problem: implement the Subset Simulation method, which is a more efficient (and more advanced) method for estimating small probabilities of rare events.

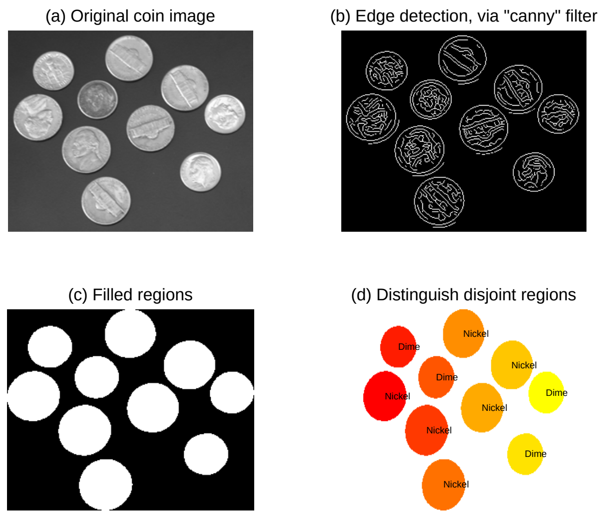

Figure 1. A sequential look at the coin identification problem.

Project 3: Coin Identification

Suppose we wish to determine the value of coins in a picture. For this project, for simplicity, we will only worry about disjoint coins, and all coins will be away from the edge. The instructions here guide us through the process step-by-step. If you find a simpler or better method, that is fine also. These instructions assume that the Image Processing Toolbox is installed.

The first step is to load a sample image of coins. MATLAB has an image called `coins.png`, and it should be in the path, so you can load it with just `imread('coins.png')`. We get a grayscale image; the function `imshow` will plot it for us. This gives plot (a) in Figure 1.

Second, we detect the edges of the coins. The toolbox has the `edge` command, which will do this. You can choose from several methods; the 'canny' method should give good results. You may also choose your own threshold values if you are getting poor results. Algorithmically, this step is not easy, but MATLAB has done the hard-work for us. By plotting, we get plot (b) in Figure 1. To see some examples of the `edge` command and the various commands, run `edgedemo`.

Now, we wish to "fill-in" the regions. Use the `regionprops` command with the `'FilledImage'` flag. The results are saved in a field called `FilledImage`, which is a matrix of 0s and 1s, with a 1 corresponding to being inside a region. This gives plot (c) in Figure 1. There are actually some very tiny regions outside the coins, barely visible on the plot, which we need to worry about later.

To go further, we need to be able to distinguish one region from another. This would not be a pleasant task to do ourselves, but fortunately MATLAB has a built-in function called `bwlabel` that will give each contiguous region its own label. For plot (d), the data was further processed using the `label2rgb` command which makes each region display in a new color, but this is only for visual effect and is not necessary.

To infer information about the coins, we would like to know the area of each region. Using the result of the `bwlabel` command as input to `regionprops`, use the `'area'` flag for `regionprops` to return the area for each region. The output of this command is an array of structures, and a bit unweildy. We can convert it to more usable form using a special form of the square brackets:

```
stats = regionprops( ... );
areas = [stats.Area];
```

With this information, we are done with the image processing. We might wish to exclude any areas that are too small (e.g. due to an artifact in the edge detection process). For the large areas that correspond to coins, we wish to determine which type of coin it is from. The dimensions of American coins are available on the internet, so it would be easy to classify the coins if we had a scale. Since there is no scale for this image, we can normalize by the largest area, and "cheat" by using the fact that the largest coin in the picture is a nickel. Then it is a simple matter to find which area corresponds to which coin and count the amount of money in the picture. Plot (d) was labelled using `text`, with centers of the coins determined via the 'Centroid' flag for `regionprops` command. Visually check the picture, and verify that your program was correct.

Your project should consist of a script file called `coins.m` that produces a figure similar to Figure 1. You may also have functions, for example, for checking the area of coins.

Project 4: Blackjack

In this assignment, you will implement a simple version of Blackjack, where a human user can play against a computer "dealer." Please see `http://en.wikipedia.org/wiki/Blackjack` for background and rules on Blackjack. For the project, use standard Blackjack rules with the following modifications and clarifications:

(1) The player begins a game with a specified amount of chips; the dealer is assumed to have an infinite supply of chips. At each round, the player specifies an arbitrary number of chips to bet. All bets are made in whole numbers of chips (e.g. no 0.3 chip bets).
(2) At every round, the objective of the player is to make the sum of the cards in his/her hand as close to 21 as possible without exceeding 21 (exceeding 21 is a "bust").
(3) At the end of every round, the winner is the person with the highest sum of cards not exceeding 21. The winner gets back their bet as well as an equal amount from the opposing player; the loser forfeits his/her bet. In the event of a tie between a user and dealer, each user gets his/her money back.
(4) Cards 2-10 are worth face falue; face cards (Jack, Queen, King) are worth 10 points. The Ace is worth 11 points unless this would cause a bust, in which case the Ace is worth 1 point.
(5) At every round, the player and dealer are both dealt two cards each (we can assume an infinite deck of cards, so this is "with replacement"). The player is allowed to see one of the dealer's cards. The player can choose to either hit (receive one more card) or stand. The player may hit repeatedly until he/she has 5 cards, at which point he/she must stand. After standing,

it is the dealer's turn. The dealer "hits" automatically until the dealer's hand is 17 or higher (or until a bust).

(6) The order of play is important! If a player busts, he/she loses, since the player played first. If the player does not bust but the dealer does, then the player wins.

(7) The program should keep track of the statistics of play, as well as the chip count. The game is over when the user has no chips left.

(8) The program should prompt the user for input, such as the amount of chips to bet, and whether to hit or stand. The program must be robust to input! If the user specifies an invalid action, or an invalid amount of chips, the program should send a warning and ask for the input again.

(9) The main function, `BlackJack.m`, should first ask the user how many chips to play with. Inside this function, you may want subfunctions, such as `Card2String` (converts a "numeric" card to a nicely formatted string that the user can understand, such as "Queen"), and `ScoreHand`, which scores the hands and determines a winner.

(10) To prompt the user for input, you may uses either `input` (which displays a string and then waits for a keyboard response), or `inputdlg` which will ask for information from a popup box.