CS1132 Fall 2008 Assignment 2

Adhere to the Code of Academic Integrity. You may discuss background issues and general solution strategies with others and seek help from course staff, but the homework you submit must be the work of just you. When submitting you assignment, be careful to follow the instructions summarized in Section 4 of this document.

1 The Coin Game

In this game there are two players. Each player has a coin and each player chooses to show either the heads or tails side of the coin. If both coins show heads, player B pays player A $3. If both show tails, B pays A $1. If they don't match, player A pays player B $2.
If both players choose heads or tails with equal probability then the game is even: out of four random rounds one can expect one case of Heads-Heads, one case of Tails-Tails, one case of Heads-Tails and one case of Tails-Heads. This translates in $0 for each player. However, either player can skew the results by playing heads or tails more often.
Through this exercise you will get an intuitive feel for the possible outcomes discussed above by way of simulating a large number of games and varying the probabilities.

1.1 Simulating one Game

Your first task is to implement a function `playCoinGame(pA, pB)` that will simulate the game:

```
function gain = playCoinGame(pA, pB)
% pA: the probability that player A plays heads
% pB: the probability that player B plays heads
% gain:  the profit player A makes over player B as a result of this game.  Positive
% number (1 or 3) means player A gets the $, a negative number (-2) means player B
% gets the $.
```

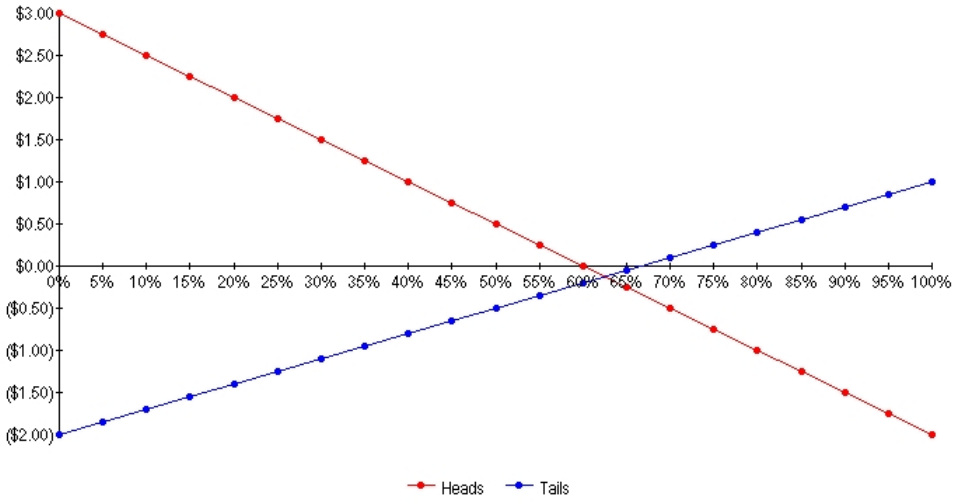
Your code must simulate the game: each player *randomly* and *independently* decides which face to show based on their probabilities (pA for player A and pB for player B).

1.2 Fair Game?

In this part, you will simulate 10^4 games in which it is equally likely for either player to show heads or tails. Your code must keep track of each player's performance (in terms of $ amounts lost, or gained, as the case might be). Present the final results in a visually pleasing way. Make sure to include the names of the players (A and B), the number of simulated games, and the final $ amount for each player. Save your script as `fairGame.m`.

1.3 Smart Playing

While the game is fair if both players act randomly (i.e. display either face with equal probability), chances are that some players might decide to improve their luck. In the following chart, the horizontal axis shows the percent of time that player B plays tails (e.g. 75% means that player B plays tails with a probability of .75). The red line shows player A's winnings per round when she plays heads; the blue line shows her winnings per round when she play tails.
Write a script that produces a similar plot. Apart from the two plot lines add two more: one that corresponds to player A playing heads and tails with equal probability, and another for the case of player A choosing tails 3 times more often than heads. Include a legend, a title, and axis labels in the figure. Save your script as `smartPlaying.m`.



2 Ranking Fun!

Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which describe documents, or searching within databases. The term "information retrieval" was coined by Calvin Mooers in 1948-50.
IR is a broad interdisciplinary field that draws on many other disciplines. It stands at the junction of many established fields, and draws upon cognitive psychology, information architecture, information design, human information behavior, linguistics, information science, computer science, librarianship, and statistics.
Automated information retrieval systems were originally used to manage the information explosion in scientific literature in the last few decades. Many universities and public libraries use IR systems to provide access to books, journals, and other documents. IR systems are often related to queries. Queries are formal statements of information needs that are put to an IR system by the user. User queries are matched to documents stored in a database.
IR had a tremendous impact in the past decades, especially in the Internet age. Web search engines such as Google and Yahoo are amongst the most visible applications of information retrieval research.
To better understand IR, consider a collection of documents, for example all the "help" information for built-in functions in Matlab. You want to be able to retrieve relevant documents for a given query, much like Google returns web-pages. Suppose someone says, "I want to know about the *ceil* and *floor* functions in Matlab." Using IR techniques, the computer is able to rank a collection of documents according to the frequency of these keywords in the documents, and thus identify the most significant documents.

The Actual Exercise

We will use an IR technique called Latent Semantic Indexing (LSI) to find relevant Matlab `help` documents given a user query in a natural language, e.g., "how do I find the roots of a polynomial function?"
We provide the following files (download them from the course website), which include the data, the main (driver) function, and a function that deals with some necessary Linear Algebra:

- the data set (the corpus, in this case files obtained from Matlab `help` documents) that you will use to test your code:
  - `docTitles.dat`: a document with the titles of the help pages (there are 48 documents in the corpus, hence this file has 48 rows). The corpus is the collection of documents that we want to search.
  - `termList.dat`: a file containing 1003 relevant terms that appear in this corpus. Each term appears on a separate line.

  - `tdm`: the term-document matrix, which is 1003 by 48. In this matrix `tdm(i,j)` represents the number of times term i appears in document j. The matrix is saved in the file `hw2p2.mat`. To load the matrix in the Matlab workspace use: `load hw2p2.mat`.

- `function scores = scoredoc(q, U, S, V, k)`: This function returns a score for each document (a vector of size 48 for this corpus) for the query using the LSI operations described later. The score is a measure of a document's relevance to the query. Don't worry about the content of this function or the meaning of the parameters—the driver function we provide will call this function.

- `function doc_rank(query, docTitles, tdm, termList)`: This function has no return values. It is the driver that wraps the entire LSI example by calling the necessary functions, including those that you will implement.

Your job is to implement the following functions:

- `function A = readFileToCell(filename)`. The function opens the file specified as the input argument and creates and returns a *cell array*. Each entry in the cell array corresponds to one line in the file.

- `function A = readFileToCharMatrix(filename)`. The function opens the file specified as the input argument, and creates and returns a *character matrix*. Each line in the file will be stored in one row. Pay attention to the dimensions of the matrix. The width of the matrix must be equal to the longest line (of text) in the file. Shorter lines should be padded with blank spaces so that all rows have the same length.

- `function [token, tail] = my_strtok(str, delimiter)`. The function has two parameters:

  - `str`: a string
  - `delimiter`: a character

  It has two output values, both strings, `token` and `tail`. The function returns in `token` that part of the input string `str` that precedes the first delimiter. The action of separating a string into separate pieces or tokens is called parsing. Parsing of the string `str` begins at the first nondelimiting (i.e., nonwhitespace) character and continues to the right until either a delimiter is located or the end of the string is reached. The function returns in `tail` a substring of the input string that begins immediately after the `token` substring and ends with the last character in `str`. If no delimiters are found in the body of the input string, then the entire string (excluding any leading delimiting characters) is returned in `token`, and `tail` is an empty string (''). To check the correctness of your function, you can compare its behavior to that of the Matlab built-in function `strtok`. Here is an example run of your function:

```
>> [token, tail]=my_strtok('estimate condition number', ' ')
token =
       estimate
tail =
       condition number
```

- `function newstr = my_lower(str)` returns the string formed by converting any uppercase characters in the string `str` to the corresponding lowercase characters and leaving all other characters unchanged (hence, `newstr` is also a string). For example:

```
>> newstr = my_lower('J Lo iS 39.')
newstr =
j lo is 39.
```

- `function row = my_strmatch(str, STRS)` looks through the rows of the character array `STRS` and returns the row number of the string in `STRS` that matches `str` exactly. If no such row exists, the function returns 0. For the purpose of this exercise, *do a character by character comparison of the strings, do not use "vectorized code."* Therefore, pay attention to the length of the string `str` and the second dimension of the matrix `STRS`. You may need to pad the shorter string with the space character (add ' ' as many times as needed at the end of `str`). Below is an example run of your function:

```
>> M = ['max    '; 'minimax'; 'maximum'];
>> i = my_strmatch('maximum', M)
i =
   3
```

`M` is a char matrix with 3 rows, one for each string, with enough white space padded to make all rows the same length. To check the correctness of your function, you can compare its behavior to that of the Matlab built-in function `strmatch(str, STRS, 'exact')`. Here is an example:

```
>> i = strmatch('max', M, 'exact')
i =
    1
```

- `function q = convert_query(query, termList)`: the function matches the terms in the query against the terms in `termList`. `query` is a string that is a natural language query. The function converts `query` to a vector that has the length equal to the number of rows in `termList` (1003 if you use the `termList` provided in `termList.dat`) where the components will be either 0 or 1, reflecting presence or absence of the ith term of `termList` in `query`: if the word in row i of `termList` is present in the query, then $q(i)$ is 1, otherwise $q(i)$ is 0. You must use the string-operation functions you have implemented: `my_lower`, `my_strtok` and `my_strmatch`.

  To better understand what this function is supposed to do, consider the following example:

```
>> M = ['ceil    '; 'floor   '; 'random  '; 'function'];
>> q = convert_query('tell me more about the ceil function', M)
q =
   1
   0
   0
   1
```

- `function topthree(doctitles, scores, resultsfile)`: this function has no return values; it creates the file `resultsfile` and writes the top three LSI matches and their scores based on the scores computed by the function `scoredoc`. You can choose any neat format.

How do you run the whole thing?

To use LSI to perform a search over the help files in Matlab, you would use the functions and data files given to you and those written by you in the following fashion:

```
>> load hw2p2.mat;
>> docTitles = readFileToCell('docTitles.dat')
>> termList = readFileToCharMatrix('termList.dat')
>> doc_rank('How do I estimate the condition number', docTitles, tdm, termList)
```

A file will be created, containing the top three documents and the corresponding scores:

```
funm 0.373740
normest1 0.265785
```

```
condest 0.260844
```

So the three most relevant files for the current query (*How do I estimate the condition number*) are `funm`, `normest1` and `condest`.

Latent Semantic Indexing

If you are interested in learning more on document ranking, we provide additional information in the following paragraphs. You can learn even more about the numerical methods we have used in this project in the these courses: CS/ENGRD 3220 (Intro to Scientic Computation) and CS 4210 (also MATH 4250, Numerical Analysis).
Latent semantic analysis (LSA) is a technique in natural language processing, patented in 1988 by Scott Deerwester, Susan Dumais, George Furnas, Richard Harshman, Thomas Landauer, Karen Lochbaum and Lynn Streeter. In the context of its application to information retrieval, it is sometimes called latent semantic indexing (LSI).
LSI uses a term-document matrix which describes the occurrences of terms in documents; it is a matrix whose rows correspond to documents and whose columns correspond to terms (words that appear in the documents). The entries in the matrix are proportional to the number of times that the terms appear in each document.
Consider the example above with the "help" information for built-in functions in Matlab. How does the computer solve such a problem?
The first idea is the vector space model (VSM) of documents. We assume that there is a very long but finite list of possible terms; terms could just be all the words used in the help files. Suppose that we number all the relevant terms (words that we care about, e.g. keywords in Matlab) from 1 through m, where m is something like one thousand or so. Then, each document d taking values from 1 to n (where n is the number of documents or help files) is associated with a vector $\mathbf{v}_d \in \mathbb{R}^m$, where $v_d(i)$ equals the number of times term i appears in document d. There are variations on this, but this is the simplest possibility.
The collection of all the documents is sometimes called a *corpus*. All these v_d vectors are concatenated together in a matrix:

$$A = [\mathbf{v}_1, \cdots, \mathbf{v}_m]$$

$A \in \mathbb{R}^{m \times n}$ describes the corpus. VSM originated with the late Gerald Salton in 1960s at Cornell University, and at that time it went against the tide of Artificial Intelligence research. However, VSMs are very mainstream today! Back in the 60s, AI researchers thought logic was the key, but Salton successfully introduced a statistical model of intelligence.
Once we have this matrix A, we can use it to find relevant documents in the corpus for our query. How do we use this query ("I want to know about the *ceil* and *floor* functions in Matlab.")? We need to write the query as a vector $\mathbf{q} \in \mathbb{R}^m$. In other words, we consider it as a very small document.
How do we use this vector $\mathbf{q}$ to identify relevant documents?
First we will decompose the matrix A into a product of three matrices using Singular Value Decomposition(SVD):

$$A = U\Sigma V^T$$

We will provide this decomposition for you, in fact there exists a build-in function in Matlab called *svd*. Once we have the decomposition, we will apply numerical analysis methods to the query vector $\mathbf{q}$ and the three matrices that resulted from the decomposition, using a certain number of singular values. This part is also provided to you in **scoredoc.m**. If you want to know more about these methods you should consider taking CS 322 or 421 (Numerical Analysis).

3 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to

resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.
Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given. In particular, the functions required in this project are:
  1. Function `playCoinGame(pA, pB)`, which takes as input two values between 0 and 1 and returns a number corresponding to player A's gain.
  2. Script `fairGame.m`, which displays a table (plain text).
  3. Script `smartPlaying.m`, which displays a figure with a title, legend, etc.
  4. Function `readFileToCell(filename)`, which copies the contents of a file into a cell array.
  5. Function `readFileToCharMatrix(filename)`, which copies the contents of a file into a character matrix.
  6. Function `my_strtok(str, delimiter)`, which takes a string and a character and returns two strings.
  7. Function `my_lower(str)`, which takes a string and returns a string.
  8. Function `my_strmatch(str, STRS)`, which takes a string and a character matrix and returns a number.
  9. Function `convert_query(query, termList)`, which takes an array and a character array and returns a vector of 0s and 1s.
  10. Function `topthree(docTitles, scores, resultsfile)`, which takes a cell array, a vector and a string. No return value.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e. end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `playCoinGame.m`, `fairGame.m`, `smartPlaying.m`, `readFileToCell.m`, `readFileToCharMatrix.m`, `my_strtok.m`, `my_lower.m`, `my_strmatch.m`, `convert_query.m` and `topthree.m` to CMS in the submission area corresponding to Assignment 1 in CMS.
2. Please don't make another submission until you have received and read the grader's comments.
3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you are asked to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Well done!