CogSci C131: Computational Models of Cognition — **Problem Set #6**
UC Berkeley, Spring 2014 — **Due: May 1st, 2014 at 2:00pm**

The problem sets in this class will be done using Matlab. You may need to use the computers in the EECS labs to get access to Matlab. If you have problems accessing these computers, contact the GSIs at `gsi-cogsci131@lists.berkeley.edu`.

Collaboration Policy

Class policy is that collaboration is permitted when solving problems, but each student must: (1) write his or her own code; (2) write the document containing his or her answers independently; and (3) list the people he or she worked with on his or her problem set.

Submission Instructions

Problems will require you to either turn in Matlab code, written answers, or both. All of the files you need are provided for you – you just need to fill them in with your answers. **Make sure to fill in your name and collaborators for each part of the homework.**

All parts that include Matlab code come with a validation script called `validatePN.m` which you can run by calling `validatePN()` from Matlab (replace `N` with the problem number, e.g. for Problem 1, run `validateP1()`). Make sure you run these functions for each part of the homework before submitting your assignment. **If your code does not pass the validation script, you will NOT receive credit.** Note, however, that this is not a grading script; it is entirely possible for incorrect answers to pass the validation script.

You will be submitting your homework on bSpace. For each problem set, you will submit a single zip file containing your filled-in files from each problem. The directory structure within your submission should be the same as when the problem set was released. The zip file should be named:

```
YourLastName_YourFirstName_ProblemSetNumber.zip
```

For example, if your name is Jane Doe, your submitted file should be named `Doe_Jane_N.zip`, where `N` is the problem set number. You can update your submission until the due date. If you make any changes to your submission after the due date, we will regard your problem set as having been turned in at the last time it was changed.

If you use Matlab on the computers in Cory 105 or log in with remote desktop, please note that files saved to the Desktop are removed when you log out. Instead, save any files to your user directory, `U:/`. The `U:/` drive can also be accessed through a shortcut on the Desktop.

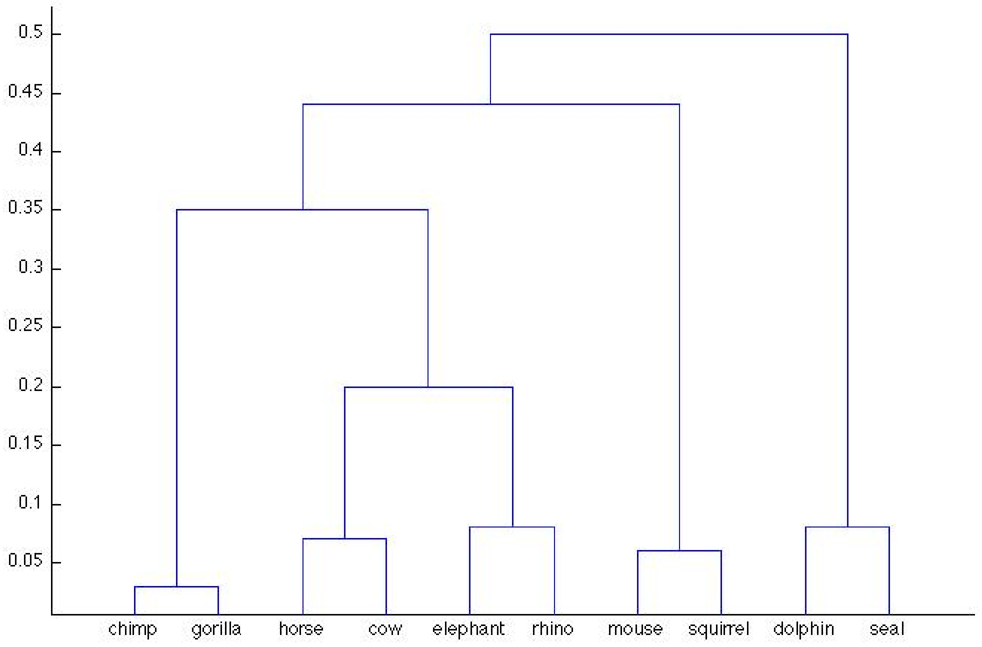
Late Problem Set Policy

This problem set is due on **May 1st, 2014 at 2:00pm**. You get **THREE LATE DAYS** total for the entire semester, to account for foreseeable minor crises. If you turn in a problem set late after using these late days, 20% of the total points on the problem set will be deducted for each 24 hours (or portion thereof) that it is late.

1 Bayesian generalization (5 points)

This question explores using Bayesian inference to explain people's generalizations, based on the Bayesian generalization model described in Tenenbaum and Griffiths (2001). Provided with an example of an object X possessing a particular property, this model indicates how one should compute the probability that another object Y will also possess that property. The model uses Bayesian inference to compare different hypotheses about the set of objects that possess the property.

In this example, the "objects" you will be working with are ten animals. As in Problem Set 5, we can create a hierarchical clustering of these animals derived from people's similarity judgments. A dendrogram for these 10 animals is shown here.



The file `animalDataset.mat` contains a set of hypotheses based on the hierarchical clustering shown above. Each hypothesis corresponds to one of the clusters identified in the hierarchy (e.g., only horses are in the first hypothesis, the chimps and gorillas share the 11th hypothesis, and all animals are in the last hypothesis). In the context of Bayesian generalization, each hypothesis corresponds to a belief about the set of animals that have the property we're interested in.

The prior probability for each hypothesis is determined by the height at which that cluster appears (clusters appearing higher in the hierarchy have lower prior probability). The hypotheses are represented by the matrix `hyps`, in which each row corresponds to a hypothesis, and each column corresponds to an animal. The elements of the matrix take the value 1 if an animal is a member of the set (cluster) identified by that hypothesis, and 0 if not.

The column vector `prior` contains the prior distribution over hypotheses. In order to apply Bayes' rule, you also need to know the likelihood. We will be using the *strong sampling* assumption. For data d consisting of an animal X being sampled from one of the hypotheses, the likelihood for hypothesis h is

$$P(d|h) = \begin{cases} \frac{1}{|h|} & \text{if animal } X \text{ is in hypothesis } h \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $|h|$ is the number of animals in the set identified by hypothesis h.

Part A

Now, suppose you discovered that horses have a particular biological property: their blood contains protein K. Complete the function `probClustersHaveK.m` so that it computes a posterior distribution over hypotheses as to which sets of animals have blood containing protein K, using the Bayesian generalization model.

The function `probClustersHaveK` should take as its argument the index of a kind of animal observed to have protein K, and should return a column vector containing the posterior probability of each hypothesis. Note that the animal dataset is loaded for you at the beginning of the function.

Part B

Complete the function in `plotProbClustersHaveK.m` to produce a bar plot of the posterior distribution for the kind of animal having protein K, showing the probability of the different hypotheses, using the `bar` command. Be sure to include a proper legend, title, and axis labels! To find out how to set these programmatically, try `help legend`, `help title`, `help xlabel`, and/or `help ylabel`.

After completing the function, run it with the argument "elephant", i.e.:

```
>> plotProbClustersHaveK('elephant')
```

Save the resulting plot as as a JPEG named `postHypothesisPlot.jpg` using the File → Save As menu in the figure.

Part C

The probability that another animal Y has blood that contains protein K can be computed by averaging over the set of hypotheses, using the posterior distribution. That is, the probability that animal Y is in the set C of animals that have blood containing protein K is

$$P(Y \in C|d) = \sum_h P(Y \in C|h)P(h|d) \tag{2}$$

where $P(Y \in C|h) = 1$ if the animal Y is in hypothesis h, and 0 otherwise. Complete the provided function `probAnimalsHaveK.m` to compute the probability that each of the nine other animals has blood that contains protein K.

The function `probAnimalsHaveK` should take as its argument the index of a kind of animal observed to have protein K, and should return a column vector containing the posterior probability of each animal having protein K. You should call your previous function, `probClustersHaveK`, as part of your computation. Note also that the animal dataset is loaded for you at the beginning of the function.

Part D

Complete the function in `plotProbClustersHaveK.m` to produce a bar plot of the posterior distribution for the kind of animal having protein K, showing the probability of the different hypotheses, using the `bar` command. Be sure to include a proper legend, title, and axis labels! Additionally, use **`set(gca, 'xticklabel', names);`** to display the animals names instead of just their indices. To find out how to set these programmatically, try **`help legend`**, **`help title`**, **`help xlabel`**, and/or **`help ylabel`**.

After completing the function, run it with the argument "elephant", i.e.:

```
>> plotProbAnimalsHaveK('elephant')
```

Save the resulting plot as as a JPEG named `postAnimalPlot.jpg` using the File → Save As menu in the figure.

Part E

Find a willing experimental participant who is not in this class. Tell them (in the context of the experiment) that scientists recently discovered that the blood of elephants contains protein K. Then, for the other nine animals, ask them to rate how likely it is that each animal has blood containing protein K, using a scale from 1 to 7. On this scale, 7 indicates that the animal definitely DOES have blood containing protein K, 1 indicates that it definitely DOES NOT have blood containing protein K, and 4 indicates an even bet.

Record their judgments in Matlab as a column vector called `ratings` in the file `participantRatings.m`. Make a bar plot of your participant's ratings, using the same format as the plot in part D, and assign the output of the `bar` command to `ratingsPlot`. As before, be sure to include a proper legend, title, and axis labels! Additionally, use **`set(gca, 'xticklabel', names);`** to display the animals names instead of just their indices. To find out how to set these programmatically, try **`help legend`**, **`help title`**, **`help xlabel`**, and/or **`help ylabel`**.

Save the resulting plot as as a JPEG named `ratingsPlot.jpg` using the File → Save As menu in the figure.

Part F

Compare your experimental results with the predictions of the Bayesian generalization model using a correlation. You can compute the correlation between the model predictions and your participant's judgments using the `corr` command, i.e.:

```
>> model = probAnimalsHaveK('elephant');
>> human = participantRatings();
>> corr(model, human)
```

What is the correlation? Based on looking at your plots from parts D and E, are there any systematic differences between your participant's judgments and the model predictions? Record your answers in the provided file `writtenAnswersPartF.txt`.

2 Bayesian regression (5 points)

The priors used in Bayesian inference provide a method for controlling the flexibility of learning algorithms. In Problem Set 5, you looked at a crude way of controlling flexibility for polynomial regression: constraining the set of functions from which the algorithm was allowed to choose. In Bayesian inference, controls on flexibility are implemented through the choice of a prior over functions.

Part A

The functions we're going to be working with are the 7-th order polynomials, $g(x) = \theta_7 x^7 + \ldots + \theta_1 x + \theta_0$. These functions are entirely characterized by the coefficients $\theta = (\theta_7, \ldots, \theta_0)$. We can define a prior over these functions by defining a distribution over θ. We will be assuming that each θ_j follows a Gaussian (i.e. normal) distribution, so

$$p(\theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\{-\theta_j^2/2\sigma_j^2\} \tag{3}$$

where $\sigma_j = \alpha\beta^j$ (and $\beta \leq 1$), and that the θ_j are independent, so $p(\theta) = \prod_{j=0}^{7} p(\theta_j)$.

The function `funcsamp` (which is provided in the problem set materials) samples a vector θ from this prior, when supplied with values of α, β, and k (the degree of the polynomial).

Using $\alpha = 5$ and $k = 7$, sample the coefficients θ for each of β = 1, 0.5, 0.3, 0.2, 0.1, 0.05, 0.01 and 0.001. Then, for each of these, plot the function, which you can compute using `polyval` with the parameters θ and with `plotx=-1:0.01:1` for the range of x. Write your code to do this in the given file `findAndDisplayFunctions.m`.

Save the resulting plot as as a JPEG named `functions.jpg` using the File → Save As menu in the figure.

Explain the effect of varying β (i.e. what functions are favored by the prior with different values of β?). Write your answer in the provided file `writtenAnswersPartA.txt`.

Part B

Our data, d, will be vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. We will use a Gaussian likelihood, assuming that the probability of a value of y decreases exponentially as the squared distance between y and $g_\theta(x)$ increases,

$$p(y_i|x_i, \theta) = \frac{1}{\sqrt{2\pi}\sigma_Y} \exp\{-(y_i - g_\theta(x_i))^2/2\sigma_Y^2\} \tag{4}$$

where σ_Y encodes how much we expect y to vary from $g(x)$. Assuming that the y_i are independent conditioned on x_i and θ, we have

$$p(d|\theta) = \left(\frac{1}{\sqrt{2\pi}\sigma_Y}\right)^n \exp\{-\sum_{i=1}^{n}(y_i - g_\theta(x_i))^2/2\sigma_Y^2\} \tag{5}$$

as our likelihood.

We will be choosing θ by maximum *a posteriori* (MAP) estimation, taking the value that maximizes $p(\theta|d)$. Since $p(\theta|d)$ is proportional to $p(d|\theta)p(\theta)$ (by Bayes' rule), we want the function that maximizes $p(d|\theta)p(\theta)$. Maximizing any monotonic transformation[1] of this quantity is equivalent, so we can also choose to maximize $\log p(d|\theta)p(\theta) = \log p(d|\theta) \log p(\theta)$. From our choice of likelihood, this becomes

$$\frac{1}{2\sigma_Y^2}\sum_{i=1}^{n}(y_i - g_\theta(x_i))^2 + \log p(\theta) + c \tag{6}$$

where c is a constant determined by σ_Y. The first term is just the MSE on the training data multiplied by $n/2\sigma_Y$. A function with a high MSE on the training data can still have high posterior probability, if its prior probability is sufficiently high. The MAP estimate will thus be a function that compromises between fitting the training data well and having high prior probability.

We have provided a function `bayespolyfit` that finds the MAP estimate of θ, when supplied with $\mathbf{x}$, $\mathbf{y}$, k, σ_Y, α, and β.

Using `traindata` and `testdata` (contained in `xyData.mat`, which is the same as `xyData.mat` from Problem Set 5), write a script `plotBayesPolyfit.m` that plots the function returned by `bayespolyfit` for the training data, and computes the MSE on the training data and test data for the values of α, β, and k from Part A, assuming that $\sigma_Y = 0.05$. You may use your MSE code from Problem Set 5, build new code, or use Matlab's MSE function. Store the MSE for the training data in the variable `MSE_train` and the MSE for the test data in `MSE_test`.

Note that when you run `bayespolyfit`, `you may see a warning` like "Matrix is close to singular or badly scaled. Results may be inaccurate." This is fine, and can be ignored.

To plot the functions returned by `bayespolyfit`, you should make a figure that has eight subplots, one for each value of β, where each subplot contains the function and the training data points. You can create the i^{th} subplot and then plot to it like this:

```
subplot(2, 4, i);
plot(X, Y);
```

Label your subplots so that we know which is which. **Save the resulting plots** as as a JPEG named `fits.jpg` using the File → Save As menu in the figure.

Your script should also plot the MSE on the training and test data as a function of β (as a separate figure); again, label your figure appropriately. **Save the resulting plots** as as a JPEG named `mse.jpg` using the File → Save As menu in the figure.

How does the prior affect the functions found using this procedure? Write your answer in the provided file `writtenAnswersPartB.txt`.

[1]A monotonic transformation $f(x)$ is one that preserves the order of its inputs. So if $x > y$, then $f(x) > f(y)$.

Part C

The MSE on the test set is an approximation to the generalization error produced by a learning algorithm. In class, we discussed how the generalization error of a learning algorithm is affected by both its bias and its variance. Both the bias and the variance depend upon the true function that is being estimated. An algorithm with high bias will systematically produce predictions that differ from the true function. An algorithm with high variance will produce predictions that can deviate wildly depending on the specifics of the dataset.

Suggest how the results from the MSE on the test set could be explained in terms of the bias and variance of the learning algorithms that result from using different priors over functions. Write your answer in the provided file `writtenAnswersPartC.txt`.

Part D

The 7th-order polynomials are a very flexible class of functions. Why were we sometimes able to get away with using a learning algorithm that can choose from such a large set of functions without suffering from overfitting? Write your answer in the provided file `writtenAnswersPartD.txt`.

3 *k*-Means Clustering (6 points)

In this problem we will use k-means clustering on a dataset consisting of observations of dogs, cats, and mops. An example observation from each type of object is presented below:

We assume each observation can be represented as a pair of (x, y) coordinates, i.e., each object is represented in two-dimensional space. Suppose we have observed ten instances from each type of object, but have lost the information as to which instance belongs to which type!

To try and recover this information we will use an unsupervised learning algorithm called *k-means clustering*. As you may recall from lecture, the k refers to how many types of clusters exist and the goal of the algorithm is to assign labels to the data points based on finding the centers (or means) of the clusters. For this particular problem, we assume $k = 3$. After randomly initializing cluster centers, the algorithm can be broken down into two alternating steps: (1) Update the label assignments of the data points based on the nearest cluster centers, and (2) Update the positions of the cluster centers to reflect the updated assignments of data points.

Before you begin, take a look at the script `kMeansClustering.m`. This is the master script you will run once you have completed the functions in parts A and B.

Part A

You will begin by completing the function template in `updateAssignments.m` to take three inputs:

- `numClusters`: an integer representing the number of clusters the data is partitioned into. For the current problem, `numClusters = 3`
- `X`: a 30×2 matrix of observation data
- `centers`: a 3×2 matrix of cluster centers where each row corresponds to the cluster number (e.g., `centers(2, 1:2)` contains x, y coordinates for the center of cluster 2).

and return a single output:

- `clusterAssignments`: a column vector containing the cluster label assignments for each data point in X. The labels will be either 1, 2, or 3, corresponding to which cluster center (labeled using the row number in `centers`) is closet to a particular data point.

To complete the function, you will need to compute the distance between each cluster center and each row in the matrix X. Use *Euclidean distance* to compute the distance between two points (x_1, y_1) and (x_2, y_2). Recall that Euclidean distance in $\mathbb{R}^2$ is calculated as:

$$distance((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{7}$$

Use this formula within your function to calculate the cluster center with the shortest Euclidean distance to a given data point. The `clusterAssignments` vector should contains cluster label assignments for *each point* in the matrix `X`.

Part B

Next you will complete the function template in `updateParameters.m` to take three inputs:

- `numClusters`: an integer representing the number of clusters the data is partitioned into. For the current problem, `numClusters = 3`
- `X`: a 30×2 matrix of observation data
- `clusterAssignments`: the column vector of cluster labels assigned to each data point from the completed function `updateAssignments`

and return a single output:

- `updatedCenters`: the mean of each of the clusters you defined using `clusterAssignments`. This will be a 3×2 matrix containing the new positions for each of the cluster centers.

The completed `updateParameters` function will reposition the cluster centers to be the mean of the data points assigned to a given cluster label.

Part C

At this point you are ready to implement the k-means clustering algorithm! Fill in steps (1) and (2) in `kMeansClustering.m` to call the functions you wrote from Parts A and B. You can test your code by running the script. If the functions you completed above are working properly, you should see three figures.

1. The first figure contains a subplot of the output from steps (1) and (2) for four iterations of the algorithm. This plot should give you a sense of how the algorithm progresses over time. The data points are each assigned to one of three colors corresponding to their current cluster label. The cluster centers are plotted as stars ($*$).

2. The second figure is a plot of the final iteration, with a black circle representing a new data point that will be explained in part (d).

3. The third figure is a plot of the *true* cluster label assignments corresponding to dogs, cats, and mops. Don't worry if the coloring in this figure is different from the coloring in the second figure. What matters is whether the data points in the second figure are assigned the same labels as those in figure 3.

Save a copy of your results as jpegs labeled `iterations`, `final`, and `trueLabels` respectively. Do you notice any differences between Figure 2 and Figure 3? In `writtenAnswersPartC.txt` write a few sentences commenting on any differences you found and why these differences might exist.

Part D

Now that we have assigned cluster labels to each datapoint, let's investigate how we should classify a *new* object:

You will be completing the function template in `assignNewObject.m` to determine the appropriate cluster label for this new object. This function is passed the following arguments:

- `numClusters`: the number of clusters.
- `newObject`: the (x, y) coordinates representing the new observation.
- `updatedCenters`: the 3×2 matrix of current positions for the cluster centers that you calculated with `updateParameters` from Part B.

Your function should return a single integer, `newObjectAssignment`, corresponding to the cluster number (i.e., row in `updatedCenters`) that best fits this observation.

Note 1: To complete the function, you will need to compute the distance between each cluster center and the new observation. Use the *Euclidean distance* formula given above in Equation (7).

Note 2: When you have completed the function you will need to edit `kMeansClustering.m`. Uncomment the line that contains:

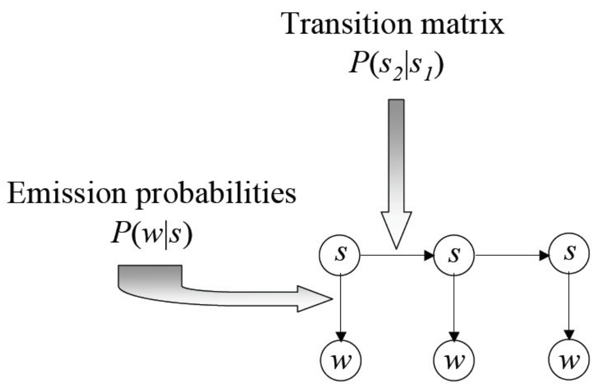
```
newObjectAssignment = assignNewObject(numClusters,newObject,centers);
```

Once you have done this, run `kMeansClustering` and observe how the new object is labeled (it is the point plotted as a big circle). Save your new Figure 2 as a jpeg labeled `newFig2`.

In the file `writtenAnswersPartD.txt` comment on whether the algorithm's labeling of the new object is what you would expect. Briefly explain why or why not.

4 Hidden Markov models of language (4 points)

In this problem, we will use a hidden Markov model (HMM) as a model of human language. The operation of our HMM is summarized below:



Recall that in an HMM a sequence of T observations $(w_1, w_2, \ldots, w_T)$ is assumed to be generated by an underlying sequence of T hidden (i.e., unobserved) *state variables*, $(s_1, s_2, \ldots, s_T)$. The *Markov* in the model name comes from the assumption that the next state variable, s_{t+1}, depends only on the current state, s_t. In this way, the state variables $(s_1, s_2, \ldots, s_T)$ form a **Markov chain**. It is assumed that each observation variable w_t is dependent on the corresponding state variable, s_t (this is reflected in the diagram above – at step t an observation node/variable w_t is connected to a single state node/variable s_t).

Using the HMM as a model of human language, the variables $(w_1, w_2, \ldots, w_T)$ correspond to the sequence of words in an observed sentence of length T, and the hidden state variables $(s_1, s_2, \ldots, s_T)$ correspond to the (unobserved) parts of speech of the words in the sentence, such as *noun*, *verb*, and *adjective*.

Recall that an HMM has three sets of parameters: (1) the initial state probability $P(s_1)$, which gives the probability of the first hidden state variable taking on each possible value, (2) the transition probabilities $P(s_t|s_{t-1})$, which give the probability of the next hidden state variable conditioned on the previous one, and (3) the emission probabilities $P(w_t|s_t)$, which give the probability of the observation at time t conditioned on the hidden state at time t.

There are two types of problems that are typically solved using an HMM: (1) *State estimation*, where the model parameters from the preceding paragraph are all known, and we are trying to recover the sequence of hidden states $(s_1, s_2, \ldots, s_T)$ from the sequence of observations $(w_1, w_2, \ldots, w_T)$. In our language modeling context, this would be like assuming we already have a good model of English grammar that specifies the HMM model parameters, and we want to take an observed English sentence and perform part-of-speech tagging to determine the part of speech of each word in the sentence. (2) *Parameter estimation*, where the model parameters from the preceding paragraph are unknown, and we are trying to recover them from only a set of observation sequences of the form $(w_1, w_2, \ldots, w_T)$. Because the hidden state variables are also unknown when performing parameter estimation, they must also be estimated simultaneously. In this situation, the expectation-maximization (EM) algorithm is used to perform simultaneous state estimation and parameter estimation.

Part A

We will use a simplistic English grammar consisting of only three parts of speech (*noun*, *verb*, and *adjective*) and ten words (*john*, *sally*, *reddit*, *love*, *parks*, *dogs*, *exhausted*, *marbled*, *big*, and *inappropriate*). Some of the words can be used as multiple parts of speech, as demonstrated in the following table:

Word	Noun	Verb	Adjective
john	✓		
sally	✓		
reddit	✓		
love	✓	✓	
parks	✓	✓	
dogs	✓	✓	
exhausted		✓	✓
marbled		✓	✓
big			✓
inappropriate			✓

Furthermore, we will assume that all sentences consist of exactly five words, so that $T = 5$ in all cases.

In Matlab, load the file `hmmLanguageModel.mat`. This data structure contains

- `partsOfSpeech`: A cell array containing the three possible values of the hidden state variables (corresponding to the three parts of speech). Each part of speech (*noun*, *verb*, and *adjective*) corresponds to a numerical index according to its position in this cell array
- `words`: A cell array containing each of the ten words in our vocabulary. Each of word is given a numerical index according to its position in this cell array
- `initProbs`: A vector containing the initial state probabilities $P(s_1)$, where the entry in row i and column j gives the probability $P(s_t = i|s_{t-1} = j)$ of transitioning from state j to state i.
- `emissionProbs`: A matrix containing the emission probabilities, where the entry in row i and column j gives the probability $P(w_t = i|s_t = j)$ of observing word i when in state j.

You will complete the function template in `generateSentence.m` to generate a random sentence from the grammar. In your solution, feel free to call the provided function `randomSample`, which returns a random sample from a vector of probability values. For example, calling `randomSample([0.5;`

`0.25; 0.25])` would return the value `1` with probability 50%, the value `2` with probability 25%, and the value `3` with probability 25%.

Once you complete `generateSentence`, you should be able to call it to generate a sentence composed of five words. For example, you could do:

```
>> sentence = generateSentence(initProbs, transitionProbs, emissionProbs)
   sentence =
      10     6     4     4     8
>> disp(words(sentence))
  'inappropriate'    'dogs'    'love'    'love'    'marbled'
```

Repeat these commands several times to see multiple random samples from the HMM language model. Paste three of them into the answer file `writtenAnswersPartA.txt`. Judging from the randomly-sampled sentences, does the HMM seem to be a good model of our limited subset of the English language? Why or why not? Include your answers in `writtenAnswersPartA.txt`

Part B

We will now tag parts of speech in our sentence by estimating the hidden state variables in our hidden Markov model. As described above, state estimation refers to estimating the sequence of hidden states of an HMM corresponding to a sequence of observations. This can only be performed when the model parameters (i.e., the initial hidden state probabilities, the transition probabilities, and the emission probabilities) are already known; otherwise, simultaneous *parameter estimation* must also be done using the **EM algorithm**.

For now, we will assume that all the parameters for our HMM are known, which makes the problem easier. Matlab performs state estimation using the **Viterbi algorithm**. The Viterbi algorithm is a popular algorithm for HMMs that returns the sequence of hidden states which maximizes the total joint probability of all the hidden states and observations in the graphical model for the HMM.

We have created a wrapper function `viterbi` which takes as inputs a sentence and the HMM model parameters and calls the Viterbi algorithm (implemented as `hmmviterbi_nostats`) to return the most likely sequence of hidden state variables for the given sentence. Write a script called `partOfSpeechTagging.m` that uses the provided `viterbi` function to perform part-of-speech tagging on the following sentences:

```
1. exhausted dogs love marbled parks
2. inappropriate sally love inappropriate reddit
3. big big john parks sally
4. sally dogs big exhausted john
5. big john exhausted exhausted dogs
```

When run, your script should return two matrices, the first containing words and the second containing part-of-speech tags. In both cases, each row will correspond to a single sentence. Make sure your script displays the words of the sentences and the names of the parts of speech, *not* their numerical indices. Sample output for `sentences = {‘sally’, ‘big’, ‘dogs’, ‘marbled’, ‘exhausted’}` would be:

```
>> [sentences, speechTags] = partOfSpeechTagging()
sentences =
  'sally'    'big'    'dogs'    'marbled'    'exhausted'

speechTags =
  'noun'    'adjective'    'noun'    'verb'    'adjective'
```

Does the HMM model do a good job of recovering the parts of speech of the words in our limited subset of the English language? Which words in the above sentences are used as more than one part of speech? Was the HMM able to determine the correct part of speech for each occurrence of these words? Give a short explanation of why the HMM model was or was not able to accomplish this disambiguation task. Write your answer in the space provided in `writtenAnswersPartB.txt`.