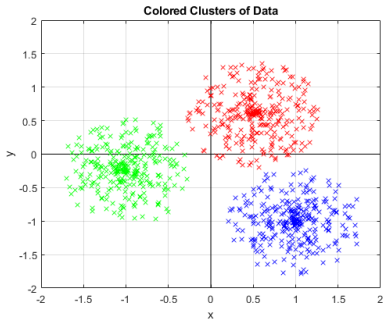
Homework Problem 2: k Means Clustering

The k-Means Clustering algorithm is an unsupervised learning algorithm that classifies data into k distinct clusters. Clustering is the task of grouping data together based on some measure of similarity. For example, we might cluster based on Euclidean distance:



In the above figure we see three clusters of data points centered at different locations. Here we've said the closer together points are the more similar they are.

Typically, clustering is done as a way of looking for possibly unknown patterns in a dataset. In this problem, we implement the k-means algorithm and apply it to a couple of datasets.

In order to solve the clustering problem, you will be implementing the k-means algorithm. The basic idea of the k-means algorithm is to construct a collection of k means and the cluster the rest of the data based on these means. For now, we will use a generalized notion of "distance" to measure similarity. The reason I say "distance" is for different data sets we might be talking about distance that you can measure with a ruler (as in the figure above) or some other notion of distance as measured by some function on the two data points. Anyway, the idea is that if two vectors are closer together, they are more similar. Thus, we cluster the data based on what mean the piece of data is closest to:

$$\text{Cluster of } x_i = \{k | \min_k \|\mu_k - x_i\|\}$$

where $\|\mu_k - x_i\|$ is the "distance" between the kth mean μ and the data point x_i.

We can then use these clusters to update the means, so:

$$\mu_k = \frac{\sum_{x_i \text{ in Cluster } k} x_i}{\text{Number of } x_i \text{ in Cluster } k}$$

Putting it all together, we start by randomly picking data points form the matrix as the means. We then iterate, using the means to first cluster the data, and then using the clusters as feedback on the means. We repeat this process until the means stop changing within tolerance. Or:

$$\|\mu_k^{n+1} - \mu_k^n\| < \text{tol}$$

for all k. Here μ_k^{n+1} is the updated set of μ_k based on clustering using μ_k^n.

Please complete the two algorithms outlined below:

%% The k-Means Algorithm:
`function [means, clusters] = kMeansClustering(data, k, distFunc, tol, maxIter)`

Initialization: Pick k random data points from the data set or use the option k-means++ initialization (see below). Assign these to μ_k^{new}.

Repeat

1. Set $\mu_k^{\text{old}} = \mu_k^{\text{new}}$.
2. Cluster the data based on these old means, i.e. assign each data point a number 1-k corresponding to which mean it is closest to based on the `distFunc`.
3. Take the average of value of each of these clusters and assign them as the new mean values or μ_k^{new}

Stop when either the max iterations is reached or $\max_k \|\mu_k^{\text{new}} - \mu_k^{\text{old}}\|_2 < \text{tol}$

The above function will have following inputs:

- **data** - the data to be clustered. Should be an $n \times d$ matrix where each row should be a new data point in the data set.
- **k** - the number of clusters your algorithm is searching for. At the end, you will have k means that organize the data into k clusters.
- **distFunc** - the distance function you are using to compare the data. Should take two row vector arguments and return a scalar. So something like: `distFunc = @(a, b) norm(a - b)` for the 2-norm distance. **IT MUST TAKE A GENERAL FUNCTION HANDLE**.
- **tol** - the tolerance that we set to say the means have stopped updating. This is similar to the tolerance values from the Bisection and Newton-Raphson methods (see Lab 8). Threshold based on the absolute difference between the means and the updated means, i.e. for all k: $\|\mu_k^{n+1} - \mu_k\|_2 < \text{tol}$. (Hint: use `norm`).
- **maxIter** - the maximum number of iterations the algorithm will run for. This is similar to the max iterations set in the Bisection and Newton-Raphson methods (see Lab 8).

And the following outputs:

- **means** - a $k \times d$ matrix. Each row should be one of the means found by the k means algorithm.
- **clusters** - a length n column vector with each giving the cluster the corresponding row in the data matrix belongs to.

And:

%% Clustering a data set given a set of means
`function [dataLabels] = labelKMeans(means, meansLabels, data, distFunc)`

For Each Data Point

1. Find which of the means stored in `means` it is closest to based on `distFunc` and store the appropriate label.

The above function has the following inputs:

- **means** - the means to use for clustering. Should be a $k \times d$ array of data with each row representing a different mean.
- **meansLabels** - the labels of the means. Should be a $k \times 1$ column array with each row giving the label of the corresponding row in the **means** array.
- **data** - the data to be clustered. Should be an $N \times d$ array of data with each row representing a different data point.
- **distFunc** - the distance function used to compare how close two values are. It should be a general function handle of two $1 \times d$ arrays and calculates the distance between them.

And the following outputs:

- **dataLabels** - the labels of the means. Should be a $N \times 1$ column array with each row giving the label of the corresponding row in the **data** array.

In order to test your algorithm, we've provided a couple of data sets to play around with:

1. `dataset1.mat` - the toy dataset for the problem above.
2. `dataset2.mat` - the second dataset for the problem see below.
3. `dataset3.mat` - the MNIST dataset.

Each of these datasets has four parts:

1. `dataTrain` - $n \times d$ matrix of data points, organized so that each row is a new data point. This is the training data for your algorithm.
2. `labelsTrain` - $n \times 1$ column vector of labels that give the classification of each data point. These are the labels of the training data.
3. `dataTest` - $m \times d$ matrix of data points, organized so that each row is a new data point. This is the test data for your algorithm.
4. `labelsTest` - $m \times 1$ column vector of labels that give the class

Try using the 2-norm of the difference of the vectors as the distance:

$$\|a - b\|_2 = \sqrt{\sum_{i=1}^{N} a_i - b_i}$$

Recall, the 2-norm in MatLab is just: `norm(v)` where `v` is the vector whose 2-norm we want to calculate.

The labels here a meant to be used with the function `mapLabels` (provided). This function uses the mode of the labels in the cluster to relabel the cluster based on the labels in the dataset. Since k-Means uses random initialization, it is likely what we call cluster 1 does not correspond to label 1. This code does the mapping for you using the mode. You can thus check the the accuracy of your k-Means algorithm on both the test and training sets by first mapping the labels using `mapLabels` and comparing the result of `labelKMeans` with the labels provided in the dataset.

The first dataset is very well separated, in the sense that each mean has it's own distinct cluster. We expect to have 100% accuracy in such a situation as we should be able to find a mean based at the center of each of the circular portions. For the second dataset, you need to do the transform described below, but it should give you 100% after that, because it is also well separated.

For MNIST, we don't expect nearly the same level of accuracy. Some people write the numbers 1 and 7 very similarly or 3 and 8. This means that even though we expect there to be 10 distinct clusters, we don't expect each digit to fall exactly into it's cluster. If a 1 looks more like a 7 than a 1, it will likely be labeled by the algorithm as a 1 instead of a 7. Using just the raw image data with out pre-processing, you will likely only get about 50% accuracy.

Note, although these labels are included in the dataset to give you some notion of the of well the algorithm is working, they are not ever used in the k-Means algorithm. That algorithm purely relies on the distance function provided with updates based on feedback from the clustering calculated by the distance function. In some sense the labels are entirely arbitrary with clusters being identified by which means they are closest too. This is most evident in the MNIST results.

In order to receive credit you must turn `kMeansClustering.m` and `labelsKMeans.m`

List of useful functions: `norm`.

Homework Problem 3: Ordinary Differential Equations

Consider a simple pendulum of length $l = 10\,m$, forming an angle $\theta(t)$ with the vertical axis.

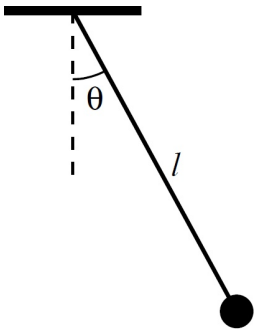


Figure 7: Simple pendulum.

The equation of motion of the pendulum is given by

$$\frac{d^2\theta}{dt^2} + c\frac{d\theta}{dt} + \frac{g}{l}\,sin(\theta) = 0, \tag{7}$$

where $g = 10\,m/s^2$ is the gravitational constant and c is the effect of friction. Consider the initial conditions

$$\theta(0) = \theta_0, \quad \frac{d\theta}{dt}(0) = 0. \tag{8}$$

1. Define a change of variables $x = \theta$ and $y = \dfrac{d\theta}{dt}$ and write this system in first-order form:

$$\frac{d\mathbf{z}}{dt} = \begin{bmatrix} \dfrac{dx}{dt} \\ \\ \dfrac{dy}{dt} \end{bmatrix} = \mathbf{f}(\mathbf{z}, t) = \begin{bmatrix} f_1(x, y, t) \\ \\ f_2(x, y, t) \end{bmatrix}, \tag{9}$$

   where $\mathbf{z} = [x \quad y]^T$. Compute the right-hand side $\mathbf{f}(\mathbf{z}, t)$. What is the initial condition $\mathbf{z}(0)$?

2. Write a function `RK4` that solves for $\theta(t)$ using the built-in `ode45` function. Your function should have the header:

```
function [t,theta,dtheta] = RK4(f, z0, T, reltol, abstol)
% RK4 Solves the IVP dz/dt = f(z,t) using ode45.
%
% [t,theta,dtheta] = RK4(f, z0, T, reltol, abstol) takes as inputs a
% anonymous function f for the right-hand side of Eq.(3), the 2x1 double array
% of initial conditions z0, the finial time of integration T, and the relative
% and absolute errors reltol and abstol, respectively.  It returns the row %
arrays t, theta and its velocity dtheta.
```

3. Use `RK4` to integrate the equations of motion (7) and compute $\theta(t)$, the velocity $\dfrac{d\theta}{dt}(t)$, the acceleration $\dfrac{d^2\theta}{dt^2}(t)$ and the pendulum's normalized energy:

$$e(t) = \frac{1}{2}\dot{\theta}^2 + \frac{g}{l}(1 - cos(\theta)). \tag{10}$$

Use `reltol = 1e-12`, abstol = 1e-12 and `T = 25`.

(a) Using subplots, plot $\theta(t)$, $\frac{d\theta}{dt}(t)$, $\frac{d^2\theta}{dt^2}(t)$ and $\frac{e(t)}{e(0)}$ for $\theta(0) = 30°$ and $c = 0$ (no friction). What is the value you expect for $\frac{e(t)}{e(0)}$? Use this fact to debug your code.

(b) Using subplots, plot $\theta(t)$, $\frac{d\theta}{dt}(t)$, $\frac{d^2\theta}{dt^2}(t)$ and $\frac{e(t)}{e(0)}$ for $\theta(0) = 30°$ and $c = 0.3$. What is the behavior you expect for $\frac{e(t)}{e(0)}$? Use this fact to debug your code.

4. Write a function `period` that solves for the period of the **undamped** oscillating pendulum using the `events` option for `ode45` function. Your solution should work with any value of $\theta(0) \in (0, \frac{\pi}{2})$, but with $\frac{d\theta}{dt}(0) = 0$ (to avoid non-periodic solutions). Your function should have the header:

```
function [t] = period(T, theta0, reltol, abstol)
% PERIOD Solves for the period of the oscillating pendulum with null velocity
% initial condition.
%
% [t] = period(T, z0, reltol, abstol) takes as inputs the finial time of
% integration T, the initial values z0, and the relative and
% absolute errors reltol and abstol, respectively.  It returns the period
% of oscillation of the pendulum t.
```

When testing your function, your final time of integration `T` should be great enough to cover at least one period of oscillation. You should expect the following output:

```
>> t = period(25, 30*pi/180, 1e-12, 1e-12)
t =

6.3926
```

5. The following function animates the motion of a pendulum. Complete the missing gaps (denoted by '???????????') appropriately and run this function with `animate_pendulum( 0, 70, 25 )` and `animate_pendulum( 0.25, 70, 25 )` to visualize the motion of the pendulum.

```
function animate_pendulum( c, theta0, T )

% Record animation and save in .avi file? Yes = 1, No = 0.
record = 0;

% Numerical integration:
z0 = ???????????; % Initial conditions
f = ???????????; % System ODE dz/dt = f(z,t)
reltol = 1e-8; abstol = 1e-8;          % Tolerance
[~,theta,~] = RK4(f, z0, T, reltol, abstol);
```

```
%  Calculate the Cartesian coordinates (x,y) for the simple
   pendulum:
y = ???????????;
x = 10*sin(theta);

%  Figure properties:
H = figure('name', 'Simple Pendulum', 'color', 'w');
anim = subplot(1,1,1);
xlabel('\itx')
ylabel('\ity    ', 'rotation', 0)
set(anim, 'xlim', [-10, 10], 'ylim', [-10, 10]);
grid on; axis equal;

%  Draw the pivot, initial position of pendulum and initial
   position of
%  mass bob at the end:
rod = line('xdata', [0 x(1)], 'ydata', [0 y(1)], 'color', 'r
   ', 'linewidth', 2);
mass = line('xdata', y(1), 'ydata', x(1), 'marker', 'o', '
   color', 'r', 'markerfacecolor', 'r', 'linewidth', 4);
pivot = line('xdata', 0, 'ydata', 0, 'marker', '^', 'color',
    'b', 'markerfacecolor', 'b', 'linewidth', 3);

%  Generate circle with radius l = 10 for reference:
uc = ???????????;
vc = ???????????;
circle = line('xdata', vc, 'ydata', uc, 'color', 'k', '
   linestyle', '--', 'linewidth', 1);

if record == 1
% Record animation:
        aviobj = VideoWriter('pendulum.avi'); open(aviobj);
end

for k = 1:length(theta)
        set(rod, 'xdata', ???????????, 'ydata', ???????????)
           ;
        set(mass, 'xdata', ???????????, 'ydata',
           ???????????);
        drawnow

        if record == 1
                frame = getframe(H);
```

```
                writeVideo(aviobj,frame);
        end
end

if record == 1
        close(aviobj);
end

end
```

For this problem you will turn in three files `RK4.m` and `period.m`.