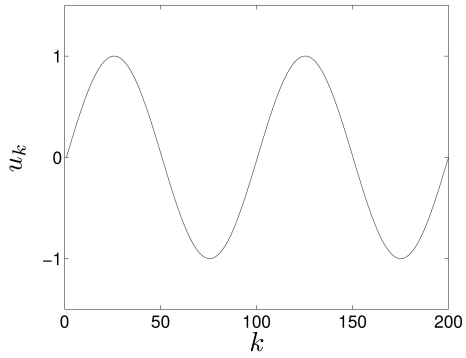
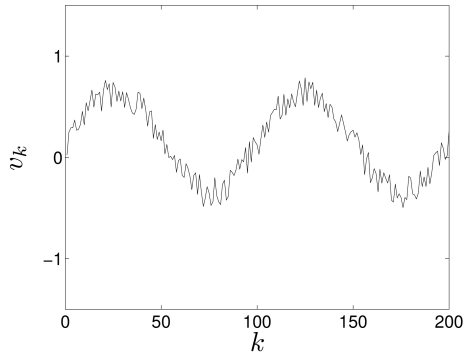
1. *Decoding using inner products.* An input signal is sent over a noisy communication channel. The channel attenuates the input signal by an unknown factor α and adds noise to it. We represent the input signal as an n-vector u, the output signal as an n-vector v, and the noise signal as an n-vector w. Therefore $v = \alpha u + w$, where the elements u_k, v_k, w_k of the three vectors give the values of the signals at time k. The two plots below show an example with $\alpha = 0.5$.





Suppose we know that the input signal u was chosen from a set of four possible signals $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, $x^{(4)}$. We know these signals $x^{(i)}$, but we don't know which one was used as input signal u. A simple method for estimating the input signal, based on the received signal v, is to calculate the angles between v and $x^{(k)}$ and pick the signal $x^{(k)}$ that makes the smallest angle with v.

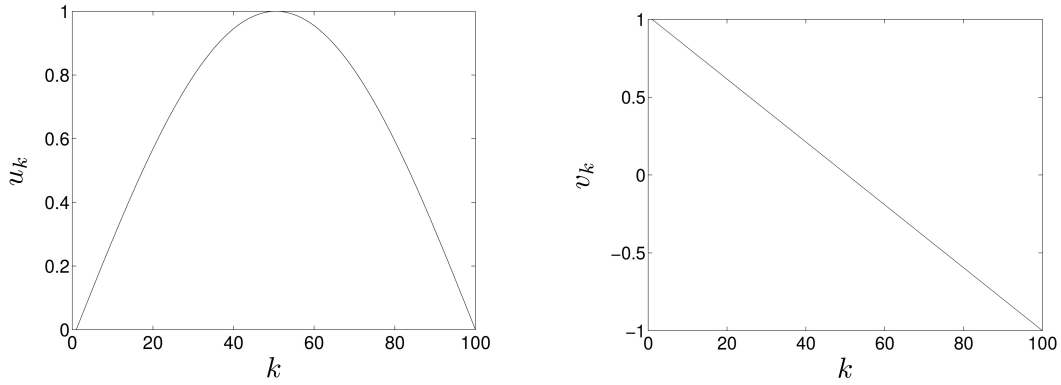
Download the file `decoding.m` from the class Teams channel, save it in your working directory, and execute it in MATLAB using the command `[x1, x2, x3, x4, v] = decoding`. The first four output arguments are the possible input signals $x^{(k)}$, $k = 1, 2, 3, 4$. The fifth output argument is the received (output) signal v. The length of the signals is $n = 200$.

(a) Plot the vectors v, $x(1)$, $x(2)$, $x(3)$, $x(4)$. Visually, it should be obvious which input signal was used to generate v.

(b) Calculate the angle θ_k of v with each of the four signals $x^{(k)}$. Which signal $x^{(k)}$ makes the smallest angle with v? Does this confirm your conclusion in part (a)?

2. *Multiaccess communication.* A communication channel is shared by several users (transmitters), who use it to send binary sequences (sequences with values $+1$ and -1) to a receiver. The following technique allows the receiver to separate the sequences transmitted by each transmitter. We explain the idea for the case with two transmitters.

   We assign to each transmitter a different signal or *code*. The codes are represented as n-vectors u and v: u is the code for user 1, v is the code for user 2. The codes are chosen to be orthogonal ($u^T v = 0$). The figure shows a simple example of two orthogonal codes of length $n = 100$.



   Suppose user 1 transmits a binary sequence $b_1, b_2, \cdots, b_m$ (with values $b_i = 1$ or $b_i = -1$), and user 2 transmits a sequence $c_1, c_2, \cdots, c_m$ (with values $c_i = 1$ or $c_i = -1$). From these sequences and the user codes, we construct two signals x and y, both of length mn, as follows:

$$x = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix}, \quad y = \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix}.$$

   (Note that here we use block vector notation: x and y consist of m blocks, each of size n. The first block of x is $b_1 u$, the code vector u multiplied with the scalar b_1, etc.) User 1 sends the signal x over the channel, and user 2 sends the signal y. The receiver receives the sum of the two signals. We write the received signal as z:

$$z = x + y = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix} + \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix}.$$

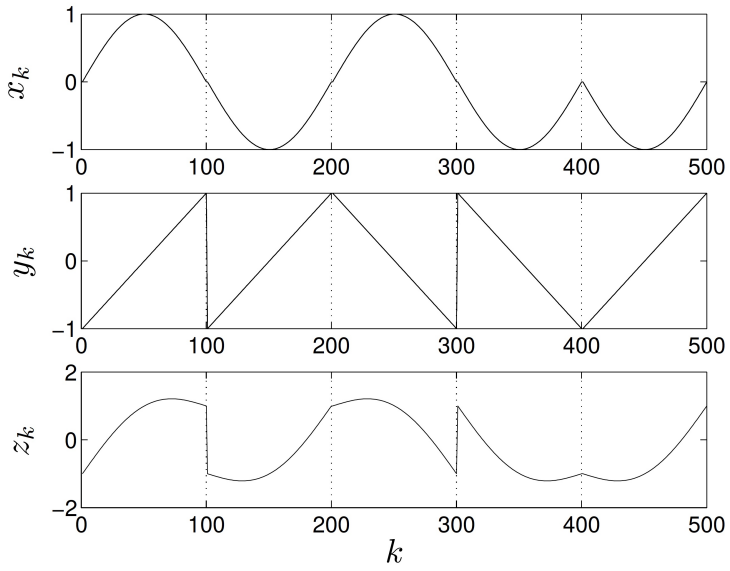
   The figure shows an example where we use the two code vectors u and v shown before. In this example $m = 5$, and the two transmitted sequences are $b = (1, -1, 1, -1, -1)$ and $c = (-1, -1, 1, 1, -1)$.

   How can the receiver recover the two sequences b_k and c_k from the received signal z? Let us denote the first block (consisting of the first n values) of the received signal z as $z^{(1)}$: $z^{(1)} = b_1 u + c_1 v$. If we make the inner product of $z^{(1)}$ with u, and use the fact that $u^T v = 0$, we get

$$u^T z^{(1)} = u^T (b_1 u + c_1 v) = b_1 u^T u + c_1 u^T v = b_1 \|u\|^2.$$

   Similarly, the inner product with v gives

$$v^T z^{(1)} = v^T (b_1 u + c_1 v) = b_1 v^T u + c_1 v^T v = c_1 \|v\|^2.$$



We see that b_1 and c_1 can be computed from the received signal as

$$b_1 = \frac{u^T z^{(1)}}{\|u\|^2}, \quad c_1 = \frac{v^T z^{(1)}}{\|v\|^2}.$$

Repeating this for the other blocks of z allows us to recover the rest of the sequences b and c:

$$b_k = \frac{u^T z^{(k)}}{\|u\|^2}, \quad c_k = \frac{v^T z^{(k)}}{\|v\|^2}.$$

if $z^{(k)}$ is the kth block of z.

Download the file `multiaccess.m` from the class Teams channel, and run it in MATLAB as `[u,v,z] = multiaccess`.

This generates two code vectors u and v of length $n = 100$, and a received signal z of length $mn = 500$. (The code vectors u and v are different from those used in the figures above.) In addition we added a small noise vector to the received signal z, i.e., we have

$$z = x + y = \begin{bmatrix} b_1 u \\ b_2 u \\ \vdots \\ b_m u \end{bmatrix} + \begin{bmatrix} c_1 v \\ c_2 v \\ \vdots \\ c_m v \end{bmatrix} + w,$$

where w is unknown but small compared to u and v.

(a) Calculate the angle between the code vectors u and v. Verify that they are nearly (but not quite) orthogonal. As a result, and because of the presence of noise, the formulas for b_k and c_k are not correct anymore. How does this affect the decoding scheme? Is it still possible to compute the binary sequences b and c from z?

(b) Compute $(b_1, b_2, b_3, b_4, b_5)$ and $(c_1, c_2, c_3, c_4, c_5)$.

3. (a) *Regression line.* Let a, b be two real n-vectors. To simplify notation we write the vector averages as

$$m_a = \text{avg}(a) = \frac{1^T a}{n}, \quad m_b = \text{avg}(b) = \frac{1^T b}{n},$$

and their standard deviations as

$$s_a = \text{std}(a) = \frac{1}{\sqrt{n}}\|a - m_a 1\|, \quad s_b = \text{std}(b) = \frac{1}{\sqrt{n}}\|b - m_b 1\|.$$

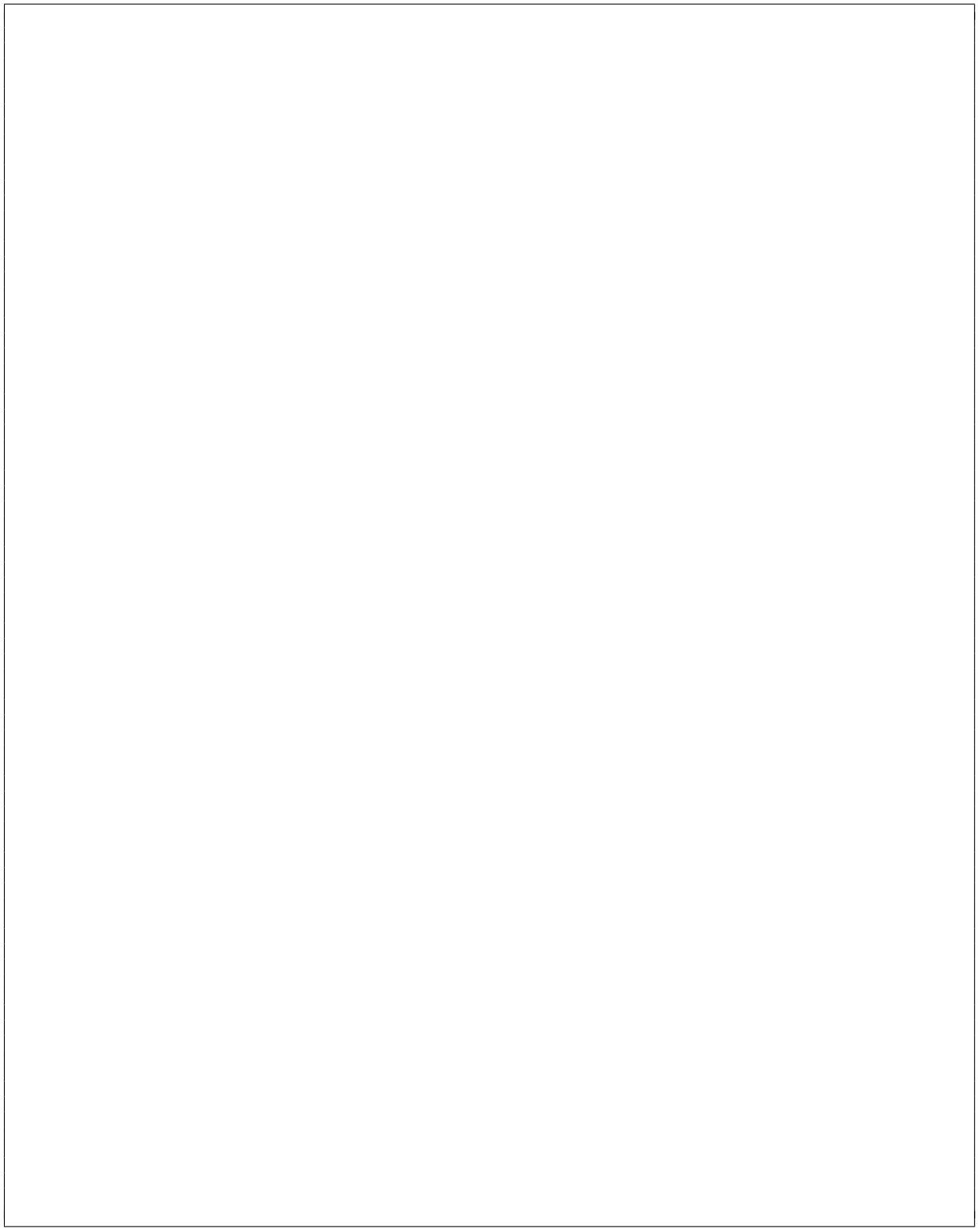
We assume the vectors are not constant ($s_a \neq 0$ and $s_b \neq 0$) and write the correlation coefficient as

$$\rho = \frac{1}{n}\frac{(a - m_a 1)^T (b - m_b 1)}{s_a s_b}.$$

We considered the problem of fitting a straight line to the points (a_k, b_k), by minimizing

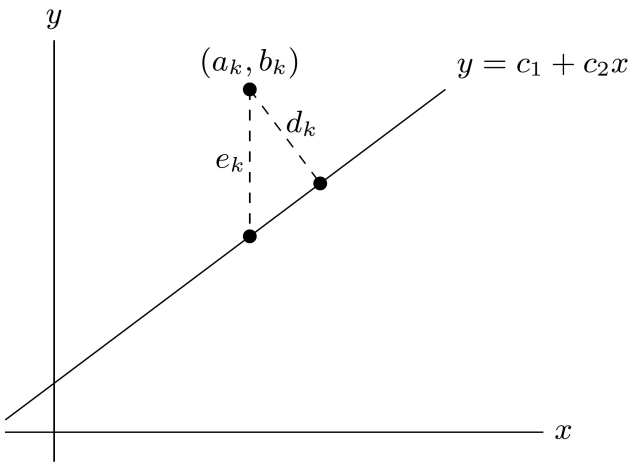
$$J = \frac{1}{n}\sum_{k=1}^{n}(c_1 + c_2 a_k - b_k)^2 = \frac{1}{n}\|c_1 1 + c_2 a - b\|^2.$$

Show that the optimal coefficients are $c_2 = \rho s_b / s_a$ and $c_1 = m_b - m_a c_2$. Show that for those values of c_1 and c_2, we have $J = (1 - \rho^2)s_b^2$.



(b) *Orthogonal distance regression.* We use the same notation as in (a): a, b are non-constant n-vectors, with means m_a, m_b, standard deviations s_a, s_b, and correlation coefficient ρ. For each point (a_k, b_k), the vertical deviation from the straight line defined by $y = c_1 + c_2 x$ is given by

$$e_k = |c_1 + c_2 a_k - b_k|.$$



The least squares regression method in (a) minimizes the sum $\sum_k e_k^2$ of the squared vertical deviations.

The orthogonal (shortest) distance of (a_k, b_k) to the line is

$$dk = \frac{|c_1 + c_2 a_k - b_k|}{\sqrt{1 + c_2^2}}.$$

As an alternative to the least squares method, we can find the straight line that minimizes the sum of the squared orthogonal distances $\sum_k d_k^2$. Define

$$J = \frac{1}{n} \sum_{k=1}^{n} d_k^2 = \frac{\|c_1 1 + c_2 a - b\|^2}{n(1 + c_2^2)}.$$

i. Show that the optimal value of c_1 is $c_1 = m_b - m_a c_2$, as for the least squares fit.

ii. If we substitute $c_1 = m_b - m_a c_2$ in the expression for J, we obtain

$$J = \frac{\|c_2(a - m_a 1) - (b - m_b 1)\|^2}{n(1 + c_2^2)}.$$

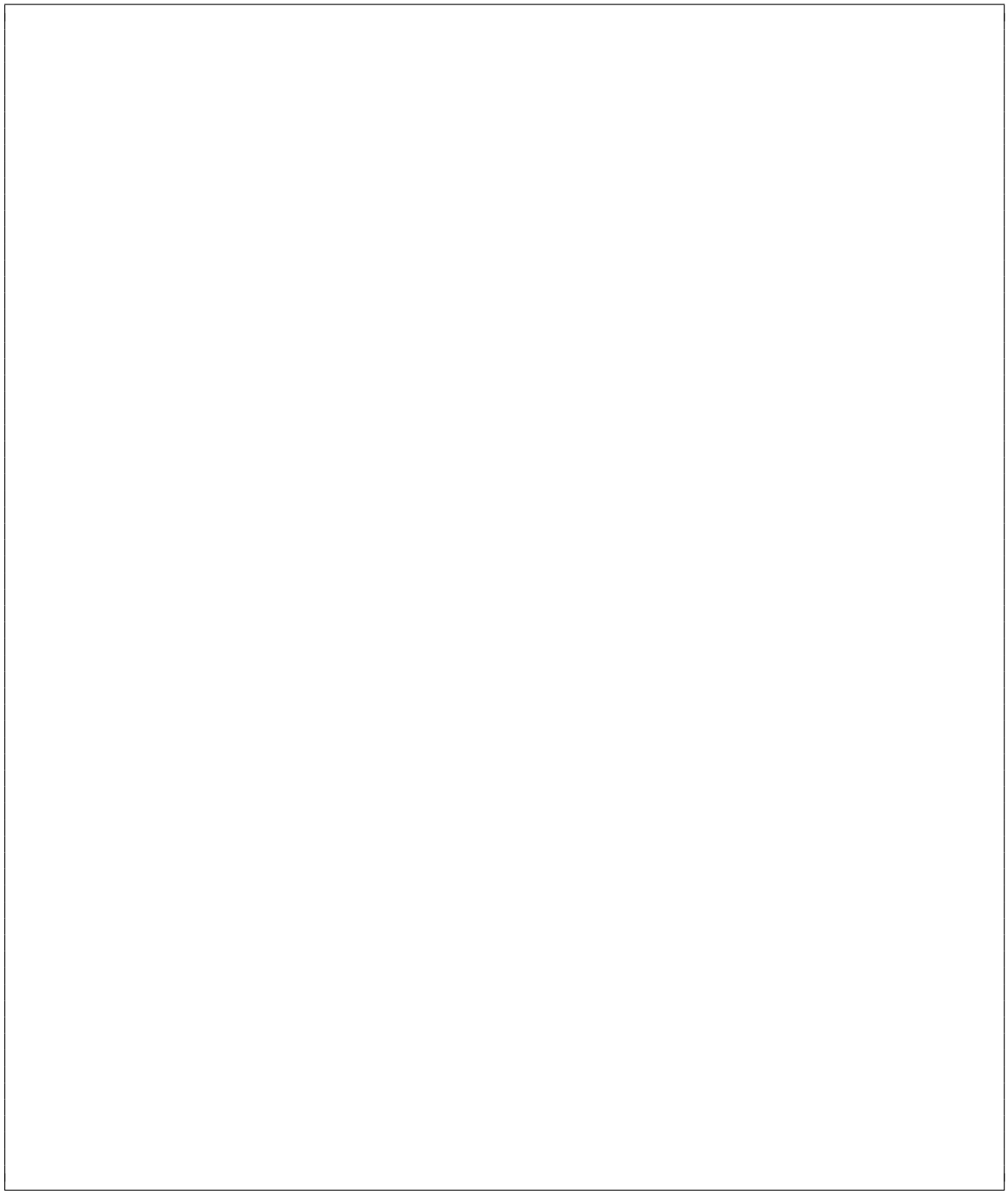
Simplify this expression and show that it is equal to

$$J = \frac{s_a^2 c_2^2 + s_b^2 - 2\rho s_a s_b c_2}{1 + c_2^2}.$$

Set the derivative of J with respect to c_2 to zero, to derive a quadratic equation for c_2:

$$\rho c_2^2 + \left(\frac{s_a}{s_b} - \frac{s_b}{s_a} \right) c_2 - \rho = 0.$$

If $\rho = 0$ and $s_a = s_b$, any value of c_2 is optimal. If $\rho = 0$ and $s_a \neq s_b$ the quadratic equation has a unique solution $c_2 = 0$. If $\rho \neq 0$, the quadratic equation has a positive and a negative root. Show that the solution that minimizes J is the root c_2 with the same sign as ρ.



iii. Download the file `orthregdata.m` and execute it in MATLAB to create two arrays a, b of length 100. Fit a straight line to the data points (a_k, b_k) using orthogonal distance regression and compare with the least squares solution. Make a MATLAB plot of the two lines and the data points.

4. *The k-means algorithm.* In this exercise we apply the k-means algorithm to the example in Section 4.4.1 of the BV textbook. Download the file `mnist_train.mat` from the class Teams channel and load it in MATLAB using the command `load mnist_train`. This creates two variables: a 784×60000 matrix `digits` and a 1×60000 matrix `labels`. We will not need `labels`. Each column of `digits` is a 28×28 grayscale image, stored as a vector of length $28^2 = 784$ with elements between 0 and 1 (0 denotes a black pixel and 1 a white pixel). Figure 4.6 in the book shows the first 25 images. To display the image in the ith column of `digits` you can use the commands

```
X = reshape(digits(:,i), 28, 28);
imshow(X);
```

The first line converts column i of `digits` to a 28×28 matrix. The second command displays the matrix as an image. To speed up the computations we will use only the first 10000 `digits`:

```
digits = digits(:, 1:10000);
```

You are asked to apply the k-means algorithm to this set of $N = 10000$ vectors, with $k = 20$ groups, and starting from a random initial group assignment (as opposed to starting from 20 randomly generated group representatives, as in Algorithm 4.1 (page 72–73).
In the following list of MATLAB hints and comments we assume that the 20 group representatives are stored as columns of a 784×20 matrix Z, and that the group assignment is represented by a 1×10000 matrix (i.e., row vector) `group`. The ith element `group(i)` is an integer between 1 and 20, with the index of the group that column i of `digits` is assigned to.

- You can create an initial group assignment using the `randi` function:

  ```
  group = randi(20, 1, 10000);
  ```

  This generates a pseudorandom 1×10000 matrix of integers between 1 and 20.
- Since we start from a random partition, the order of the two steps in algorithm 4.1 is switched. The first step in each cycle is to compute the group representatives for the current assignment. We can find the columns of `digits` that are assigned to group i using the function `find`. The command

  ```
  I = find(group == i);
  ```

  defines an index vector I such that `digits(:,I)` is the submatrix of `digits` with the columns assigned to class i. To find the average of the columns in this matrix, you can use for-loops, or the MATLAB functions `sum` or `mean`. (Be sure to look up what `sum` or `mean` do when applied to a matrix; see `help sum` and `help mean`.)
- We evaluate the quality of the clustering using the clustering objective

$$J = \frac{1}{N} \sum_{i=1}^{N} \min_{j=1,\cdots,k} \|x_i - z_j\|^2.$$

  The algorithm is terminated when J is nearly equal in two successive iterations (e.g., we terminate $|J - J_{\text{prev}}| \leq 10^{-5} J$, where J_{prev} is the value of J after the previous iteration).
- After running the k-means algorithm you can display the representative vectors of the 20 groups as follows:

```
for  k=1:20
subplot(4,5,k)
imshow(reshape(Z(:,k), 28, 28));
end
```

This produces a figure similar to figures 4.8 and 4.9 in the textbook. Your results will be different because figures 4.8 and 4.9 were computed using the full set of 60000 `digits` and, moreover, the result of the k-means algorithm depends on the starting point.

- Include the code and a figure of a typical set of group representatives with your solution.

5. *The k-means algorithm.* We apply the k-means algorithm to the example in Section 4.4.2 of the textbook. In this example we cluster the word histogram vectors of 500 Wikipedia articles, using a dictionary of 4423 words.

   First download the binary file `wikipedia_m.mat` from the dataset directory on the class Teams channel, and import it in MATLAB using the command `load wikipedia_m`. Three variables will be defined: `tdmatrix` (which stands for term-document matrix), `articles`, `dictionary`. The variable `tdmatrix` is a 4423×500 matrix with the 500 word histogram vectors as its columns. The variable `articles` is an array of length 500 with the article titles: `articles(j)` is the title of the article represented by column j in `tdmatrix`. The variable `dictionary` is an array of length 4423 with the words in the dictionary: `dictionary(i)` is the word referred to by row i of `tdmatrix`.

   You are asked to apply the k-means algorithm to this set of $N = 500$ vectors, with $k = 8$ groups and starting from a random initial group assignment (as opposed to starting from randomly generated group representatives, as in Algorithm 4.1 in the textbook). To evaluate the quality of the clustering we use the clustering objective

$$J = \frac{1}{N} \sum_{i=1}^{N} \min_{j=1,\cdots,k} \|x_i - z_j\|^2.$$

   The algorithm is terminated when J is nearly equal in two successive iterations (e.g., we terminate $|J - J_{\text{prev}}| \leq 10^{-8} J$, where J_{prev} is the value of J after the previous iteration).

   Test your code with different random initial assignments, and summarize the results for the best clustering you find (with the lowest clustering objective). To summarize the result, give the five top terms for each cluster (the words associated with the highest five components of the cluster representative) and the titles of the five articles in the cluster closest to the representative.

   MATLAB hints

   - You can create an initial group assignment using the `randi` function: the command

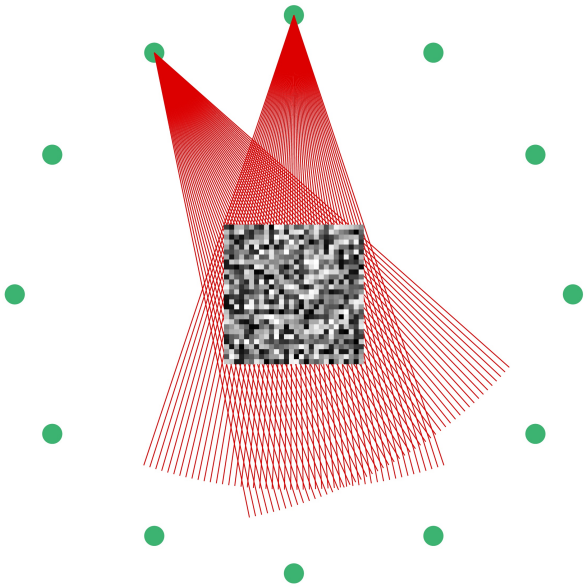
     ```
     group = randi(8, 1, 500);
     ```

     creates a pseudorandom 1×500 array `group` of integers between 1 and 8.
   - Since we start from a random partition, the order of the two steps in Algorithm 4.1 is switched. The first step in each cycle is to compute the group representatives for the current assignment. Suppose we store the assignment in an array `group` of length 500, as initialized above. We can find the columns of `tdmatrix` (i.e., the articles) that are assigned to group i using the function `find`. The command

     ```
     I = find(group == i);
     ```

     defines an index vector I with the indexes of the articles in group i, so that `tdmatrix(:,I)` is the submatrix of `tdmatrix` with the columns assigned to group i. To find the average of the columns in this matrix, you can use for-loops, or the functions `sum` or `mean`. (Be sure to look up what `sum` or `mean` do when applied to a matrix; see `help sum` and `help mean`.)
   - The function `sort` with two output arguments is useful to find the top terms for each group representative, and the articles in each cluster closest to the representative.

6. *Tomography.* Download the file `tomography.mat` from the class Teams channel and load it in MATLAB using the command `load tomography`. This creates a matrix A of size 576×784 and a vector b of length 576. The matrix A and the vector b describe a toy tomography example and were constructed using the MATLAB AIR Tools package that can be found at `www2.compute.dtu.dk/~pcha/AIRtools`. (You do not need this package for the exercise.) The geometry is shown in the figure below.



The image at the center is a black-and-white image of size 28×28. The figure shows a random image but in the actual problem we used one of the images of handwritten digits of exercise 4. The green dots are twelve source locations. For each source location we generate 48 rays emanating from the source. (The figure shows the rays for two sources only.) For each ray, we calculate the line integral of the pixel intensities along the ray. This gives $12 \times 48 = 576$ linear equations

$$\sum_{j=1}^{784} A_{ij} x_j = b_i, \quad i = 1, \cdots, 576.$$

Here x_j denotes the intensity in pixel j of the image (images are stored as vectors of length 784, as in exercise 4), A_{ij} is the length of the intersection of ray i with pixel j, and b_i is the value of the line integral along ray i. These equations can be written in matrix form as $Ax = b$ where A and b are the data in `tomography.mat`, or as

$$a_i^T x = b_i, \quad i = 1, \cdots, 576$$

where a_i^T is row i of A.

The purpose is to reconstruct the image x from the line integral measurements. We will use Kaczmarz's iterative algorithm for this purpose. Note that this is a small problem and easy to solve using the standard non-iterative methods that are discussed later in the course.

Kaczmarz's algorithm starts at an arbitrary point x (for example, a zero vector) and then cycles through the equations. At each iteration we take a new equation, and update x by replacing it with its projection on the hyperplane defined by the equation. If K is the number of cycles and $m = 576$ is the number of equations, the algorithm can be summarized as follows.

Initialize x.
For $k = 1, \cdots, K$:
    For $i = 1, \cdots, m$:
        Project x on the ith hyperplane:
            $x := x - \frac{a_i^T x - bi}{\|a_i\|^2} a_i$.
    end
end

Run the algorithm for $K = 10$ cycles, starting at $x = 0$ (a black image). Compute the error $\|Ax - b\|/\|b\|$ after each cycle. Display the reconstructed image x (using the command `imshow(reshape(x, 28, 28))`).