Questions 3 to 5
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
X \in \mathbb{R}^{n \times d} \\
\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.
\]
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
X \in \mathbb{R}^{n \times d} \quad \arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.
\]

\[x_i \in \mathbb{R}^d \quad z_i \in \mathbb{R}^k\]
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$\arg\min_{Z \in \mathbb{R}^{n \times k}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$  \hspace{1cm} (1)

- Recall: principal component analysis (PCA) projects d-dimensional data points to a hyperplane orthogonal to the directions of maximal variance.

$$X \in \mathbb{R}^{n \times d}$$

$$x_i \in \mathbb{R}^d$$

$$z_i \in \mathbb{R}^k$$

$$SVD(X) = U, \Sigma, V^T$$

$$W = V^T_{(1:d,1:k)}$$  \hspace{1cm} \text{Eigenvectors of} \hspace{1cm} \text{(directional vectors)}

$$X^T X$$  \hspace{1cm} \text{Covariance matrix}
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2. \tag{1}
\]

- Recall: principal component analysis (PCA) projects d-dimensional data points to a hyperplane orthogonal to the directions of maximal variance
- PCA preserves **covariance** between the data points
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.
\]

- MDS projects data points to a space where similar data points are clustered together
- MDS preserves distances between points
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
\text{argmin}_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2
\]

- MDS preserves distances between points
Multi-Dimensional Scaling (MDS)

The function `example_MDS` loads the animals dataset and then shows (i) the raw data, (ii) the data projected onto the first two principal components, and (iii) the result of applying gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

\[
\text{argmin}_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2
\]

- We want \(\|x_i - x_j\| \approx \|z_i - z_j\|\) \(\forall i, j\)
- where, for example,
  \[x_i \in \mathbb{R}^{50} \quad z_i \in \mathbb{R}^2\]
```matlab
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);
```
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);

Figure 1 - Displays matrix X

X consists of values 0 and 1
blue is 0; red is 1
Figure 2 - Displays the projection of $X$ onto the first two Principal components

$X(i) = [Z_{i1}, Z_{i2}]$
load animals.mat
[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2);clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);

argmin\limits_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=i+1}^{n} (\|x_i - x_{i'}\| - \|z_i - z_{i'}\|)^2

Figure 3 - Displays the datasets in terms of the two latent features obtained from MDS

X(i) = [Z_{i1}, Z_{i2}]
load animals.mat

[n,d] = size(X);

% Figure 1 shows raw data
figure(1);
imagesc(X);

% Figure 2 shows PCA visualization
figure(2); clf;
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
figure(2);
plot(Z(:,1),Z(:,2),'.');
hold on;
for i = 1:n
    text(Z(i,1),Z(i,2),animals(i,:));
end

% Figure 3 shows MDS visualization
z = visualizeMDS(X,2,animals);

argmin_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2

\begin{align*}
X(i) &= [Z_{i1}, Z_{i2}]
\end{align*}
function [Z] = visualizeMDS(X,k,names)

[n,d] = size(X);

% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';

Z(:) = findMin(@stress,Z(:),500,0,D,names);
end

Computes the distances between each pair of samples $D(i,j) = ||x_i - x_j||^2$

Initializes $Z$ using PCA

$$\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i=i+1}^{n} \left( ||x_i - x_j|| - ||z_i - z_j|| \right)^2$$
function \[ f,g \] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);

for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:) - Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
        dgi = (Z(i,:) - Z(j,:))/Dz;
        dgj = (Z(j,:) - Z(i,:))/Dz;
        g(i,:) = g(i,:) - df*dgi;
        g(j,:) = g(j,:) - df*dgj;
    end
end

g = g(:);
function [f,g] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);
for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
        dgi = (Z(i,:)-Z(j,:))/Dz;
        dgj = (Z(j,:)-Z(i,:))/Dz;
        g(i,:) = g(i,:) - df*dgi;
        g(j,:) = g(j,:) - df*dgj;
    end
end

f = g(:);
function \[ f,g \] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);

for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
        dgi = (Z(i,:)-Z(j,:))/Dz;
        dgj = (Z(j,:)-Z(i,:))/Dz;
        g(i,:) = g(i,:)-df*dgi;
        g(j,:) = g(j,:)-df*dgj;
    end
end

g = g(:);
function \([f, g] = \text{stress}(Z, D, \text{name})\)
\[
\begin{align*}
&\text{n} = \text{length}(D); \\
&k = \text{numel}(Z)/n; \\
&Z = \text{reshape}(Z, [n \, k]); \\
&f = 0; \\
&g = \text{zeros}(n, k); \\
&\text{for } i = 1:n \\
&\quad \text{for } j = i+1:n \\
&\quad\quad \text{\% Objective Function} \\
&\quad\quad Dz = \text{norm}(Z(i,:) - Z(j,:)); \\
&\quad\quad s = D(i,j) - Dz; \\
&\quad\quad f = f + (1/2)*s^2; \\
&\quad\quad \text{\% Gradient} \\
&\quad\quad df = s; \\
&\quad\quad dgi = (Z(i,:) - Z(j,:))/Dz; \\
&\quad\quad dgj = (Z(j,:) - Z(i,:))/Dz; \\
&\quad\quad g(i,:) = g(i,:) - df*dgi; \\
&\quad\quad g(j,:) = g(j,:) - df*dgj; \\
&\text{end} \\
&\text{end} \\
&g = g(:); \\
\end{align*}
\]

MDS

\[
\text{argmin}_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.
\]

MDS function value

\[
f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2
\]
function \([f, g] = \text{stress}(Z, D, \text{names})\)

\[
f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2
\]

MDS function value

MDS gradient w.r.t \(z_i\) and \(z_j\)

\[
s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)
\]

\[
s_{ij} = \|x_i - x_j\| - \|z_i - z_j\| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}
\]

\[
f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \frac{1}{2} s_{ij}^2 \right)
\]
MDS function value

\[
f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2
\]

MDS gradient w.r.t \(z_i\) and \(z_j\)

\[
s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sqrt{\|x_i - x_j\|^2 - \|z_i - z_j\|^2}
\]

\[
f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \frac{1}{2} s_{ij}^2 \right)
\]

Derivative

\[
\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}} \frac{\partial s_{ij}}{z_i}
\]
function [f,g] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;
Z = reshape(Z,[n k]);

f = 0;
g = zeros(n,k);
for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
dgi = (Z(i,:)-Z(j,:))/Dz;
dgj = (Z(j,:)-Z(i,:))/Dz;
g(i,:) = g(i,:) - df*dgi;
g(j,:) = g(j,:) - df*dgj;
    end
end

g = g(:);

MDS function value

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)^2 \]

MDS gradient w.r.t \( z_i \) and \( z_j \)

\[ s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||) \]

\[ s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2 - (z_i - z_j)^2} \]

\[ f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \frac{1}{2} s_{ij}^2 \right) \]

Derivative

\[ \frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}} \frac{\partial s_{ij}}{z_i} \]

\[ \frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial (\frac{1}{2} s_{ij}^2)}{\partial s_{ij}} = ? \]
function \([f,g] = \text{stress}(Z,D,\text{names})\)

\[
\begin{align*}
\text{n} &= \text{length}(D); \\
\text{k} &= \text{numel}(Z)/\text{n}; \\
\text{Z} &= \text{reshape}(Z,[\text{n k}]); \\
\text{f} &= \text{zeros}(\text{n},\text{k}); \\
\text{g} &= \text{zeros}(\text{n},\text{k}); \\
\text{for } \text{i} = 1:\text{n} \\
&\quad \text{for } \text{j} = \text{i}+1:\text{n} \\
&\quad \quad % \text{Objective Function} \\
&\quad \quad \text{Dz} = \text{norm}(\text{Z}(\text{i},:) - \text{Z}(\text{j},:)); \\
&\quad \quad \text{s} = \text{D}((\text{i},\text{j}) - \text{Dz}; \\
&\quad \quad \text{f} = \text{f} + (1/2)*\text{s}^2; \\
&\quad \quad % \text{Gradient} \\
&\quad \quad \text{df} = \text{s}; \\
&\quad \quad \text{dgi} = ((\text{Z}(\text{i},:) - \text{Z}(\text{j},:))/\text{Dz}; \\
&\quad \quad \text{dgj} = ((\text{Z}(\text{j},:) - \text{Z}(\text{i},:))/\text{Dz}; \\
&\quad \quad \text{g}((\text{i},:)) = \text{g}((\text{i},:)) - \text{df}^*\text{dgi}; \\
&\quad \quad \text{g}((\text{j},:)) = \text{g}((\text{j},:)) - \text{df}^*\text{dgj}; \\
&\quad \text{end} \\
&\text{end} \\
\text{g} &= \text{g}(::);
\end{align*}
\]

\[f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)^2\]

MDS gradient w.r.t \(z_i\) and \(z_j\)

\[
s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)
\]

\[
s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}
\]

\[f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\frac{1}{2} s_{ij}^2\right)\]

Derivative

\[
\frac{\partial f_{ij}}{\partial z_i} = \frac{\partial f_{ij}}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial z_i}
\]

\[
\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial (\frac{1}{2} s_{ij}^2)}{\partial s_{ij}} = s_{ij}
\]
function \([f, g] = \text{stress}(Z, D, \text{names})\)

\[
\begin{align*}
\text{n} &= \text{length}(D); \\
\text{k} &= \text{numel}(Z) / \text{n}; \\
Z &= \text{reshape}(Z, [\text{n} \ \text{k}]); \\
f &= 0; \\
g &= \text{zeros}(\text{n}, \text{k}); \\
\text{for } i = 1:\text{n} \\
    \text{for } j = i+1:\text{n} \\
        \% \text{Objective Function} \\
        Dz = \text{norm}(Z(i,:) - Z(j,:)); \\
        s = D(i,j) - Dz; \\
        f = f + (1/2)*s^2; \\
        \% \text{Gradient} \\
        df = s; \\
        dgi = (Z(i,:) - Z(j,:))/Dz; \\
        dgj = (Z(j,:) - Z(i,:))/Dz; \\
        g(i,:) = g(i,:) - df*dgi; \\
        g(j,:) = g(j,:) - df*dgj; \\
    \end{align*}
\]

g = g(:,);

MDS function value

\[
f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)^2
\]

MDS gradient w.r.t \(z_i\) and \(z_j\)

\[
s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)
\]

\[
s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2 - (z_i - z_j)^2}
\]

\[
f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \frac{1}{2} s_{ij}^2 \right)
\]

Derivative

\[
\frac{\partial f_{ij}}{\partial z_i} = \frac{\partial f_{ij}}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial z_i}
\]

\[
\frac{\partial f_{ij}}{\partial s_{ij}} = \frac{\partial \left(\frac{1}{2} s_{ij}^2\right)}{\partial s_{ij}} = s_{ij}
\]

\[
\frac{\partial s_{ij}}{\partial z_i} = \frac{\partial (\sqrt{(x_i - x_j)^2 - (z_i - z_j)^2})}{\partial z_i} = ?
\]
function \([f, g] = \text{stress}(Z, D, \text{name})\)

\[ n = \text{length}(D); \]
\[ k = \text{numel}(Z)/n; \]
\[ Z = \text{reshape}(Z, [n, k]); \]
\[ f = 0; \]
\[ g = \text{zeros}(n, k); \]
\[ \text{for } i = 1:n \]
\[ \quad \text{for } j = i+1:n \]
\[ \quad \quad \% \text{Objective Function} \]
\[ Dz = \text{norm}(Z(i, :) - Z(j, :)); \]
\[ s = D(i, j) - Dz; \]
\[ f = f + (1/2)*s^2; \]
\[ \% \text{Gradient} \]
\[ df = s; \]
\[ dgi = (Z(i, :) - Z(j, :))/Dz; \]
\[ dgj = (Z(j, :) - Z(i, :))/Dz; \]
\[ g(i, :) = g(i, :) - df*dgi; \]
\[ g(j, :) = g(j, :) - df*dgj; \]
\[ \text{end} \]
\[ \text{end} \]
\[ g = g(:, :); \]

MDS function value

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j\neq i}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2 \]

MDS gradient w.r.t \(z_i\) and \(z_j\)

\[ s = \sum_{i=1}^{n} \sum_{j\neq i}^{n} (\|x_i - x_j\| - \|z_i - z_j\|) \]
\[ s_{ij} = \|x_i - x_j\| - \|z_i - z_j\| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2} \]
\[ f(Z) = \sum_{i=1}^{n} \sum_{j\neq i}^{n} \left( \frac{1}{2} s_{ij}^2 \right) \]

Derivative

\[ \frac{\partial f_{ij}}{\partial z_i} = \frac{\partial f_{ij}}{\partial s_{ij}} \cdot \frac{\partial s_{ij}}{\partial z_i} \]
\[ \frac{\partial s_{ij}}{\partial z_i} = \frac{\partial (\sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2})}{\partial z_i} = -\frac{z_i - z_j}{\sqrt{(z_i - z_j)^2}} \]
function \[ f, g \] = stress(Z,D,names)

n = length(D);
k = numel(Z)/n;

Z = reshape(Z, [n k]);

f = 0;
g = zeros(n,k);

for i = 1:n
    for j = i+1:n
        % Objective Function
        Dz = norm(Z(i,:)-Z(j,:));
        s = D(i,j) - Dz;
        f = f + (1/2)*s^2;

        % Gradient
        df = s;
dgi = (Z(i,:)-Z(j,:))/Dz;
dgj = (Z(j,:)-Z(i,:))/Dz;
        g(i,:) = g(i,:) - df*dgi;
g(j,:) = g(j,:) - df*dgj;
    end
end

g = g(:);

MDS function value
\[
f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)^2
\]

MDS gradient w.r.t \( z_i \) and \( z_j \)
\[
s = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (||x_i - x_j|| - ||z_i - z_j||)
\]
\[
s_{ij} = ||x_i - x_j|| - ||z_i - z_j|| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2}
\]
\[
f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\frac{1}{2} s_{ij}^2 \right)
\]

Derivative
\[
\frac{\partial f_{ij}}{z_i} = \frac{\partial f_{ij}}{s_{ij}} \frac{\partial s_{ij}}{z_i}
\]
\[
\frac{\partial f}{z_i} = \frac{\partial f}{\partial s} \frac{\partial s}{z_i} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} s_{ij} \left( -\frac{z_i - z_j}{\sqrt{(z_i - z_j)^2}} \right)\]
Question 3.1: visualizeSammon

Make new function visualizeSammon that implements gradient descent for MDS Sammon mapping objective,

Sammon’s mapping objective function

$$\argmin_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|}.$$ 

MDS objective function

$$\argmin_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.$$
Question 3.1: *visualizeSammon*

Make new function *visualizeSammon* that implements gradient descent for MDS Sammon mapping objective,

Sammon’s mapping objective function

\[
\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|}.
\]

MDS objective function

\[
\arg\min_{Z \in \mathbb{R}^{n \times k}} \frac{1}{2} \sum_{i=1}^{n} \sum_{i=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2.
\]
Question 3.1: *visualizeSammon*

Sammon’s mapping objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|} \]

MDS objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2 \]
Question 3.1: visualizeSammon

Sammon’s mapping objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(||x_i - x_j|| - ||z_i - z_j||)^2}{||x_i - x_j||} \]

Recall: \( D(i, j) = ||x_i - x_j|| \)

MDS objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(||x_i - x_j|| - ||z_i - z_j||)^2}{2} \]

function \([f, g] = stress(Z, D, names)\)

\[
\begin{align*}
n &= \text{length}(D); \\
k &= \text{numel}(Z)/n; \\
Z &= \text{reshape}(Z, [n \ k]); \\
f &= 0; \\
g &= \text{zeros}(n, k); \\
\text{for } j = i+1:n \\
    \quad Dz &= \text{norm}(Z(i, :) - Z(j, :)); \\
    \quad s &= D(i, j) - Dz; \\
    \quad f &= f + \frac{1}{2} s^2; \\
\end{align*}
\]

\[
\begin{align*}
\text{df} &= s; \\
\text{dgi} &= (Z(i, :) - Z(j, :))/Dz; \\
\text{dgj} &= (Z(j, :) - Z(i, :))/Dz; \\
\text{g}(i, :) &= \text{g}(i, :) - \text{df}^*\text{dgi}; \\
\text{g}(j, :) &= \text{g}(j, :) - \text{df}^*\text{dgj}; \\
\end{align*}
\]

\[ g = g(:); \]
Question 3.1: visualizeSammon

Sammon’s mapping objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|} \]

Recall: \( D(i, j) = \|x_i - x_j\| \)

\( s_{ij} = ? \)

MDS objective function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2 \]

\( s_{ij} = \|x_i - x_j\| - \|z_i - z_j\| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2} \)
Question 3.1: visualizeSammon

Sammon’s mapping gradient function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|} \]

\[ s_{ij} = ? \]

MDS gradient function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2 \]

\[ s_{ij} = \|x_i - x_j\| - \|z_i - z_j\| = \sqrt{(x_i - x_j)^2 - (z_i - z_j)^2} \]
Question 3.1: **visualizeSammon**

Sammon’s mapping gradient function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\|x_i - x_j\| - \|z_i - z_j\|)^2}{\|x_i - x_j\|} \]

\[ s_{ij} = ? \]

\[ \frac{\partial f}{z_i} = \frac{\partial f}{\partial s} \frac{\partial s}{z_i} = ? \]

MDS gradient function

\[ f(Z) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\|x_i - x_j\| - \|z_i - z_j\|)^2 \]

\[ s_{ij} = \|x_i - x_j\| - \|z_i - z_j\| = \sqrt{(x_i - x_j)^2} - \sqrt{(z_i - z_j)^2} \]

\[ \frac{\partial f}{z_i} = \frac{\partial f}{\partial s} \frac{\partial s}{z_i} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left( \frac{s_{ij}}{\sqrt{(z_i - z_j)^2}} \right) \]
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function `visualizeISOMAP` that computes the approximate geodesic distance (shortest path through a graph where the edges are only between $k$-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. **Hand in your code and the plot of the result when using the 3-nearest neighbours.**
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function `visualizeISOMAP` that computes the approximate geodesic distance (shortest path through a graph where the edges are only between $k$-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. **Hand in your code and the plot of the result when using the 3-nearest neighbours.**
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function `visualizeISOMAP` that computes the approximate geodesic distance (shortest path through a graph where the edges are only between \(k\)-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. **Hand in your code and the plot of the result when using the 3-nearest neighbours.**
Question 3.2: ISOMAP

This suggests that ISOMAP may give a better visualization. Make a new function `visualizeISOMAP` that computes the approximate geodesic distance (shortest path through a graph where the edges are only between $k$-nearest neighbours, and the edge weights are the distances between these neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Hand in your code and the plot of the result when using the 3-nearest neighbours.
Question 3.2: ISOMAP

- If $x$ is $d$-dimensional,
  - then ISOMAP with $n$-nearest neighbor is MDS
  - Otherwise the only difference is in the distance function
If \( x \) is \( d \)-dimensional,
- then ISOMAP with \( d \)-nearest neighbor is MDS
- Otherwise the only difference is in the distance function

```matlab
function [Z] = visualizeMDS(X,k,names)

[n,d] = size(X);

% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));

% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';

Z(:) = findMin(@stress,Z(:,500,0,D,names);
```
If $x$ is $d$-dimensional,
  
  - then ISOMAP with $d$-nearest neighbor is MDS
  - Otherwise the only difference is in the distance function

**ISOMAP:**

1) Find $k$-nearest neighbors
   The ‘$k$’ points that are closest to each data point (see KNN)
ISOMAP:
2) Create $n \times n$ zero matrix $G$ (the adjacency graph) s.t.

$$G(i, j) = \begin{cases} D(i, j) & \text{if } j \in \text{neighbors}(i) \\ 0 & \text{otherwise} \end{cases}$$

```matlab
function [Z] = visualizeMDS(X,k, names)
[n,d] = size(X);
% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));
% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';
Z(:, :) = findMin(@stress,Z(:,:) ,500,0,D,names);
```
ISOMAP:

2) Create $n \times n$ zero matrix $G$ (the adjacency graph) s.t.

$$G(i, j) = \begin{cases} D(i, j) & \text{if } j \in \text{neighbors}(i) \\ 0 & \text{otherwise} \end{cases}$$

3) Use the Dijkstra function to get the shortest distance between each point $i$ and $j$

$$D(i, j) = \text{dijkstra}(G, i, j)$$

```matlab
function [Z] = visualizeMDS(X,k,names)
[n,d] = size(X);
% Compute all distances
D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(abs(D));
% Initialize low-dimensional representation with PCA
[U,S,V] = svd(X);
W = V(:,1:k)';
Z = X*W';
Z(:) = findMin(@stress,Z(:),500,0,D,names);
```
ISOMAP:
2) Create an n x n zero matrix $G$ (the adjacency graph) s.t.

$$G(i, j) = \begin{cases} 
D(i, j) & \text{if } j \in \text{neighbors}(i) \\
0 & \text{otherwise}
\end{cases}$$

3) Use the Dijkstra function to get the shortest distance between each point $i$ and $j$

$$D(i, j) = \text{dijkstra}(G, i, j)$$

Update the distance matrix The adjacency matrix
Q4: Visualizing a neural net for 1D regression

the data very well. Try to improve the performance of the method by changing the structure of the network \((nHidden)\) is a vector giving the number of hidden units in each layer) and the training procedure (e.g., change the sequence of step sizes, add momentum, or use \(findMin\) from the previous assignment). Hand in your plot after changing the code to have better performance, and list the changes you made.
% Add bias
X = [ones(size(X,1),1) X];
d = d + 1;

% Choose network structure
nHidden = [5];

% Count number of parameters and initialize weights 'w'
nParams = d*nHidden(1);
for h = 2:length(nHidden)
    nParams = nParams+nHidden(h-1)*nHidden(h);
end
nParams = nParams-nHidden(end);
w = randn(nParams,1);

% Train with stochastic gradient
maxIter = 1000000;
stepSize = 1e-4;
funObj = @(w,i)MLPreogressionLoss(w,X(:,i),y(:,i),nHidden);
for t = 1:maxIter
    % Every few iterations, plot the data/model:
    if mod(t,round(maxIter/100)) == 0
        fprintf('Training iteration = %d\n',t-1);
        figure();clf;hold on
        Xhat = [-5:.05:5]';
        Xhat = [ones(size(Xhat,1),1) Xhat];
        yhat = MLPRegressionPredict(w,Xhat,nHidden);
        plot(X(:,2),y,'.');
        h=plot(Xhat(:,2),yhat,'g-');
        set(h,'LineWidth',3);
        legend({'Data','Neural Net'});
        drawnow;
    end

    % The actual stochastic gradient algorithm:
    i = ceil(rand*N);
    [f,g] = funObj(w,i);
    w = w - stepSize*g;
end
load nnetData.mat % Loads data (X,y)
[N,d] = size(X);

% Add bias
X = [ones(N,1) X];
d = d + 1;

% Choose network structure
nHidden = [3];

% Count number of parameters and initialize weights 'w'
nParams = d*nHidden(1);
for h = 2:length(nHidden)
    nParams = nParams+nHidden(h-1)*nHidden(h);
end
nParams = nParams-nHidden(end);
w = randn(nParams,1);

% Train with stochastic gradient
maxIter = 100000;
stepSize = 1e-4;
funObj = @(w,i)MLPRegressionLoss(w,X(i,:),y(i),nHidden);
for t = 1:maxIter
    % Every few iterations, plot the data/model:
    if mod(t-1,round(maxIter/100)) == 0
        fprintf('Training iteration = %d
',t-1);
        figure(1);clf;hold on
        Xhat = [-5:0.05:5];
        yhat = MLPRegressionPredict(w,Xhat,nHidden);
        plot(X(:,2),y,'.');
        h=plot(Xhat(:,2),yhat,'g-');
        set(h,'LineWidth',3);
        legend({'Data','Neural Net'});
        drawnow;
    end

    % The actual stochastic gradient algorithm:
    i = ceil(rand*N);
    [f,g] = funObj(w,i);
w = w - stepSize*g;
end
Improve the performance of your neural network

Number of hidden neurons

The size of the step to take when updating ‘w’

nParams: the total number of variables

i: the index of the next sample ‘x’ to be chosen

t: the epoch number

g: the gradient of that sample with respect to ‘w’
Adjust the number of hidden neurons (the more hidden neurons, the more powerful your model will be, but could cause overfitting)

Adjust the step size
- Large step size can cause oscillations in your function value
- Small step size can cause slow training
Original performance

Target performance