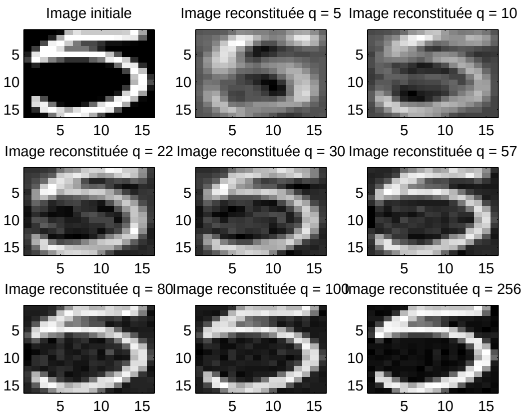
Data Mining
TP2 - Compression par ACP

Rémi MUSSARD - Thomas ROBERT

1 Tests de l'ACP

On utilise nos fonctions d'ACP sur des données contenant des images représentant des chiffres manuscrits. Après l'ACP, on reconstruit les données initiales.

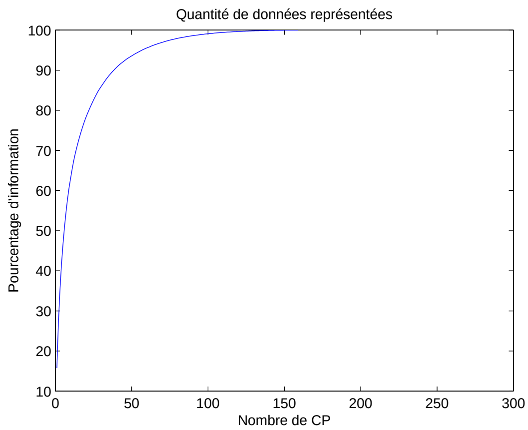
```
load data_iris_usps_asi/uspsasi.mat

caract = 5;
lines = find(y == caract);

% récupération du caractère <caract>
X = x(lines, :);

% préparation de l'ACP
[D, U, moy]=mypca(X);

i = 2;

figure();
subplot(3,3,1);
imagesc(reshape(X(1,:),16,16)')
colormap gray;
title('Image initiale');

for q = [5 10 22 30 57 80 100 256]

    % récupération des composantes principales
    P = U(:, 1:q);

    % projection sur les composantes principales
    Ct = projpca(X, moy, P);

    % reconstruction des données initiales
    Xhat = reconstuctpca(Ct, P);

    % representation de la reconstruction
    subplot(3,3,i);
    imagesc(reshape(Xhat(1,:),16,16)')
    title(['Image reconstituée q = ' int2str(q)]);
    colormap gray;

    i = i + 1;
end
```



On remarque que comme prévu, plus on ajoute de composantes sur lesquelles on projette, plus l'image reconstituée est proche de l'image initiale. La différence entre l'image initiale et l'image avec toutes les composantes est sans doute due aux erreurs numériques lors des projections.

```
% nombre de CP approprié
figure();
qteRepresentee = cumsum(D)/sum(D)*100;
plot(qteRepresentee);
title('Quantité de données représentées');
xlabel('Nombre de CP');
ylabel('Pourcentage d''information');
fprintf('Nombre de CP pour avoir 95%% : %i\n', find(qteRepresentee > 95, 1));
```

```
Nombre de CP pour avoir 95% : 57
```



Avec 57, on aura 95% de l'information reconstruite. Etant donné l'allure de la courbe, cette valeur nous semble un bon compromis.

Visages propres

On charge les données et on calcule l'ACP sur le premier jeu.

```
% chargements des données

load YaleFace/Subset1YaleFaces.mat

X1 = X;
Y1 = Y;

```

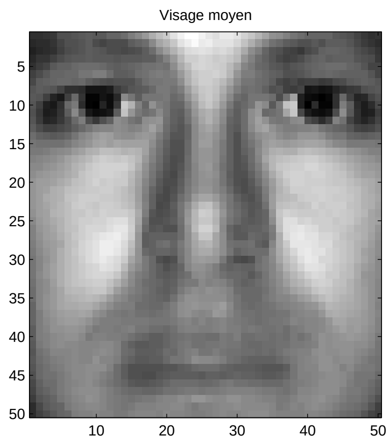
```
load YaleFace/Subset2YaleFaces.mat

X2 = X;
Y2 = Y;

load YaleFace/Subset3YaleFaces.mat

X3 = X;
Y3 = Y;

% calcul de l'ACP sur le premier ensemble
[D, U, moy]=mypca(X1);
```

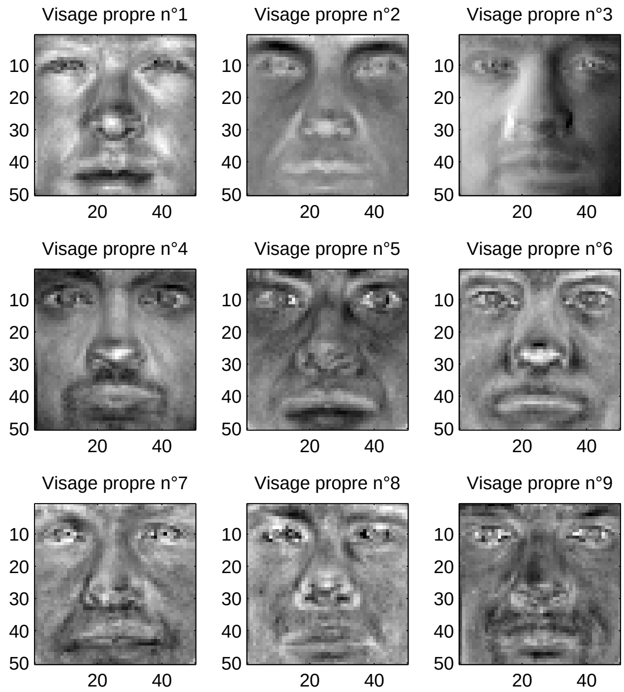
1.1 Visualisation des résultats de l'ACP

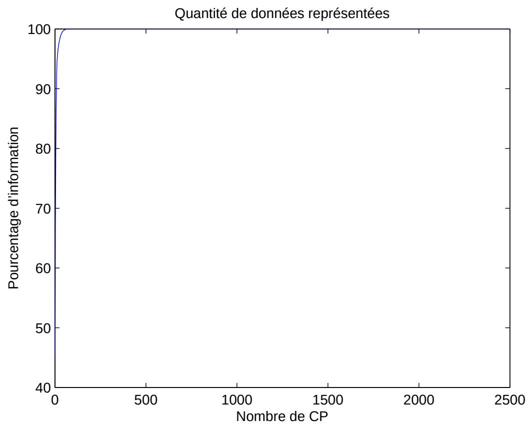
On affiche le visage moyen, les 9 premiers visages propres, et la quantité de données représentées.

```
% visage moyen
figure();
imagesc(reshape(moy,50,50));
colormap gray;
title('Visage moyen');

% visages propres
figure();
for i=1:9
    subplot(3,3,i);
    imagesc(reshape(U(:,i),50,50));
    colormap gray;
    title(['Visage propre n°' int2str(i)]);
end

% quantité d'info
figure();
qteRepresentee = cumsum(D)/sum(D)*100;
plot(qteRepresentee);
title('Quantité de données représentées');
xlabel('Nombre de CP');
ylabel('Pourcentage d''information');
fprintf('Nombre de CP pour avoir 95%% : %i\n', find(qteRepresentee > 95, 1));
```

```
Nombre de CP pour avoir 95% : 13
```





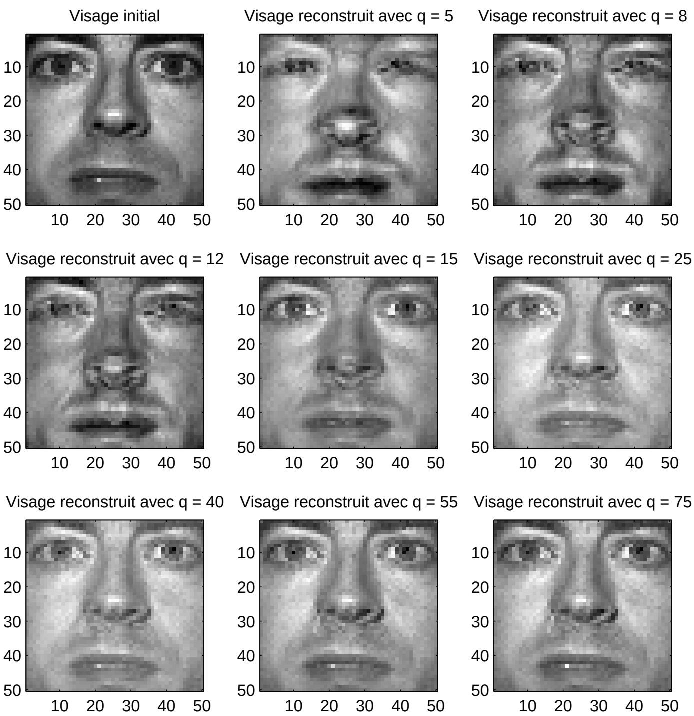


1.2 Reconstruction des données

Affichage des résultats de la compression par ACP.

```
% visage a afficher
visage = 40;

i = 2;
figure();
subplot(3,3,1);
imagesc(reshape(X1(visage,:),50,50));
title('Visage initial');
colormap gray;

for q=[5 8 12 15 25 40 55 75]
    % récupération des composantes principales
    P = U(:, 1:q);

    % projection sur les composantes principales
    Ct = projpca(X1, moy, P);

```

```
    % reconstruction des données initiales
    Xhat = reconstuctpca(Ct, P);

    % affichage
    subplot(3,3,i);
    imagesc(reshape(Xhat(visage,:),50,50));
    title(['Visage reconstruit avec q = ' int2str(q)]);
    colormap gray;

    i = i + 1;
end
```

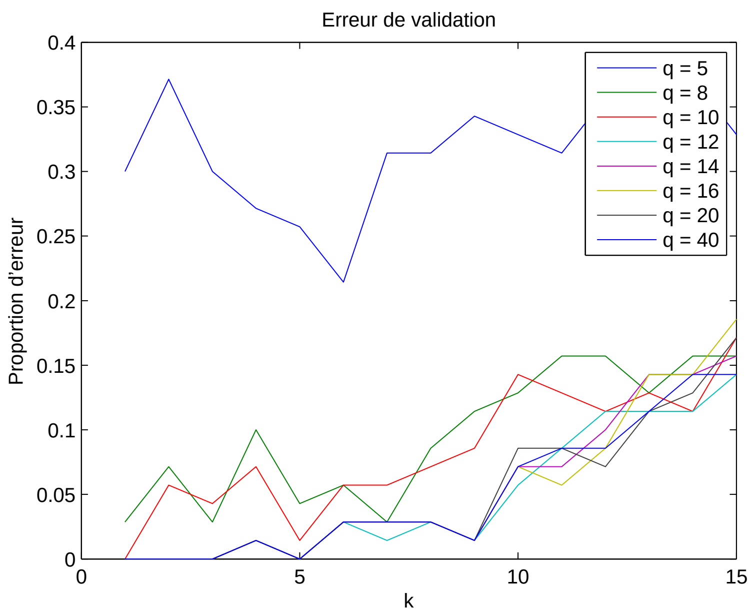


1.3 Détermination des meilleurs k et q

On détermine les meilleurs k et q en utilisant le premier jeu de données en apprentissage et le deuxième en validation.

```
% pour chaque bloc
qVals = [5 8 10 12 14 16 20 40];
qValsLabel = {'q = 5', 'q = 8', 'q = 10', 'q = 12', 'q = 14', 'q = 16', 'q = 20', 'q = 40'};
kmax = 15;

errVal = zeros(kmax, length(qVals));
for j = 1:length(qVals)

    % choix q
    q = qVals(j);

    % projection
    P = U(:, 1:q);
    Ct1 = projpca(X1, moy, P);
    Ct2 = projpca(X2, moy, P);

    % pour chaque k
```

```
    for k = 1:kmax

        % prédiction
        [Y2pred, Dist] = knn(Ct2, Ct1, Y1, k);

        % calcul proportion d'erreur pour le k choisi
        errVal(k, j) = mean(Y2pred ~= Y2);
    end
end

% affichage erreur
figure();
plot(errVal);
title('Erreur de validation');
xlabel('k');
ylabel('Proportion d''erreur');
legend(qValsLabel);
```



On voit qu'en choisissant k = 5, a partir d'une valeur suffisamment grande de q, l'erreur devient nulle. Ceci étant vrai à partir de q = 12, il n'est pas nécessaire de conserver plus de composantes qui alourdirait la base de données.

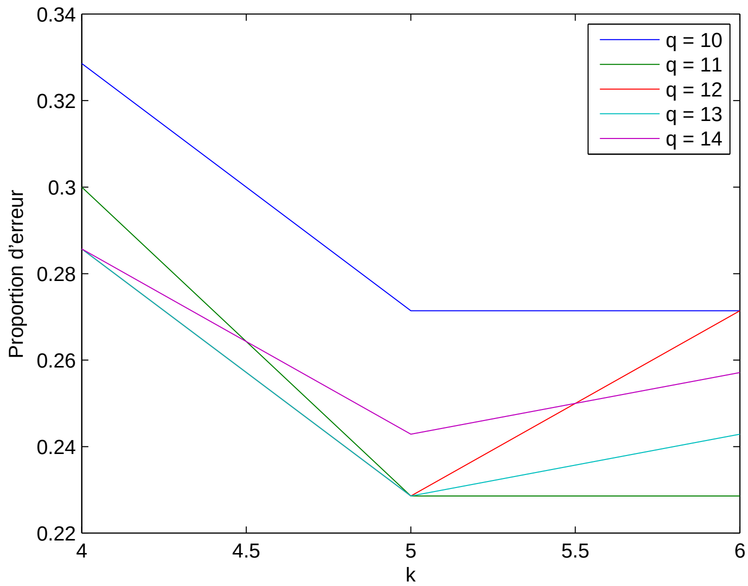
Les valeurs optimales sont donc k = 5 et q = 12.

1.4 Test de performance

On teste la performance de notre *k-nn* en utilisant le troisième jeu de données en test.

```
qTests = [10 11 12 13 14];
qTestsLabel = {'q = 10', 'q = 11', 'q = 12', 'q = 13', 'q = 14'};
kTests = 4:6;

errTest = zeros(length(kTests), length(qTests));
for j = 1:length(qTests)

    % choix q
    q = qTests(j);

    % projection
    P = U(:, 1:q);
    Ct1 = projpca(X1, moy, P);
    Ct3 = projpca(X3, moy, P);
```

```

    % pour chaque k
    for i = 1:length(kTests)

        % choix k
        k = kTests(i);

        % prédiction
        [Y3pred, Dist] = knn(Ct3, Ct1, Y1, k);

        % calcul proportion d'erreur pour le k choisi
        errTest(i, j) = mean(Y3pred ~= Y3);
    end
end

% affichage erreur
figure();
plot(kTests, errTest);
xlabel('k');
ylabel('Proportion d''erreur');
legend(qTestsLabel);
```



On voit que les valeurs k = 5 et q = 12 sont effectivement les valeurs optimales pour les résultats sur l'ensemble de test. La proportion d'erreur reste de 22% mais c'est le meilleur résultat qu'on ai pu obtenir sur l'ensemble de test. Afin d'augmenter ce résultat, il faudrait agrandir la base d'apprentissage.

2 Conclusion

Ce TP aura permis de voir que l'ACP permet de compression les données de façon significatives. Par exemple, il est possible de conserver quasiment l'intégralité des données d'une photo sur 13 composantes au lieu de 2500.

Cette compression permet par ailleurs de pouvoir utiliser des méthodes comme le *k-nn* dans une base de taille réduite par rapport aux données initiales et donc d'augmenter grandement la rapidité des calculs.