2020

Inteligencja obliczeniowa i jej zastosowania

METODY REDUKCJI WYMIAROWOŚCI

DAWID MIKOWSKI 251674 PIOTR CHOROŚCIN 228937

Metoda NMF

Metoda nieujemnej faktoryzacji macierzy (NMF), podobnie jak metoda analizy składowych głównych (PCA) jest metodą redukcji wymiarowości, w której macierz obserwacji można przybliżyć iloczynem pewnych dwóch macierzy.

W przypadku PCA tymi macierzami były: $\mathbb{U}$ - macierz cech oraz $\mathbb{Z}$ - macierz komponentów (składowych) głównych.

Idea NMF jest podobna, lecz istnienie tutaj dodatkowe założenie – wszystkie elementy macierzy, których iloczyn przybliża macierz obserwacji – faktory - muszą być nieujemne (stąd nazwa metody). Założenie nieujemności macierzy sprawia, że metoda ta jest bardziej zbliżona to sposobu działania ludzkiego mózgu.

Zatem należy estymować macierz $\mathbb{Y} \cong \mathbb{A}\mathbb{X}$ takimi faktorami $\widehat{\mathbb{A}}$ $\widehat{\mathbb{B}}$, że spełnione są warunki:

- $\forall a \epsilon A: a \geq 0$, oraz
- $\forall b \epsilon B: b \geq 0$,

a optymalizowana jest funkcja niepodobieństwa D macierzy $\mathbb{Y}$ *do iloczynu* $\mathbb{A}\mathbb{X}$, zatem:

$$\{\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{X}}\} = argmin_{\boldsymbol{A},\boldsymbol{X}} D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})$$

Jednak problem ten należy do klasy problemów NP.-trudnych oraz wiąże się z niejednoznacznością rozwiązań. W związku z tym stosuje się algorytm optymalizacji naprzemiennej. Zakłada on iteracyjne wyznaczanie macierzy A oraz X aż do osiągnięcia określonej zbieżności. Początkowo macierze inicjowane są losowo.

Zadanie 1

Generowanie syntetycznych obserwacji

Pierwszym krokiem jest syntetyczne wygenerowanie próbek (próbki te imitują dane z prawdziwych obserwacji). Wygenerowano więc losowo nieujemne macierze **Aw** i **Bw**, a macierz obserwacji zdefiniowano jako iloczyn tych macierzy.

```
% generowanie próbek
Aw = max(0, randn(1000, 10));
Xw = max(0, randn(10, 100));
Y = Aw * Xw;
```

Implementacja algorytmu optymalizacji naprzemiennej

Zdefiniowano funkcję realizującą algorytm optymalizacji naprzemiennej:

```
% wyznaczenie estymowanych faktorów A i B
% optymalizacja naprzemienna
function [X, A, RES, MSE, SIR, elapsed_time] = optymalizacja_naprzemienna(Y, next_X_fun, next_A_fun, N)
    t_start = tic;
    % inicjalizacja
    X = max(0, randn(10, 100));
    A = max(0, randn(1000, 10));
    RES = zeros(1, N); MSE = zeros(1, N); SIR = zeros(1, N);
    for n=1:N
        % kroki algorytmu
        A = next_A_fun(A, X, Y);
        X = next_X_fun(A, X, Y);
        % obliczenie błędów dla n
        [RES(n), MSE(n), SIR(n)] = calcErrors(A*X, Y);
    end
    elapsed_time = toc(t_start);
end
```

Funkcja jako argumenty przyjmuje:

- **Y** – macierz obserwacji,
- **next_X_fun** – funkcję wyznaczającą następną wartość macierzy X,
- **next_A_fun** – funkcję wyznaczającą następną wartość macierzy A,
- **N** – liczbę iteracji algorytmu.

Funkcja zwraca:

- **X** – estymowaną macierz cech,
- **A** – estymowaną macierz wektorów kodujących,
- **RES** – tablicę błędów residualnych dla kolejnych iteracji,
- **MSE** – tablicę błędów średniokwadratowych dla kolejnych iteracji,
- **SIR** – tablicę wartości stosunku sygnału do zakłóceń dla kolejnych iteracji.
- **elapsed_time** – czas wykonania mierzony za pomocą funkcji tic / toc.

Pierwszym krokiem algorytmu jest losowa inicjalizacja estymowanych macierzy, jednak w taki sposób aby były one nieujemne.

```
% inicjalizacja
X = max(0, randn(10, 100));
A = max(0, randn(1000, 10));
```

Następnie, iteracyjnie wyznaczane są kolejne wartości macierzy A i X i mierzone są błędy dla danej iteracji:

```
for n=1:N
    % kroki algorytmu
    A = next_A_fun(A, X, Y);
    X = next_X_fun(A, X, Y);
    % obliczenie błędów dla n
    [RES(n), MSE(n), SIR(n)] = calcErrors(A*X, Y);
end
```

Implementacja algorytmów

Do implementacji algorytmu optymalizacji naprzemiennej wykorzystano funkcyjne cechy języka MatLab, które pozwalają przekazać funkcję jako argument do innej funkcji. Dzięki temu zaimplementowano 3 pary funkcji wyznaczające kolejne wartości macierzy X oraz Y dla różnych algorytmów, a następnie przekazano je jako argumenty do funkcji optymalizacji naprzemiennej.

```
= optymalizacja_naprzemienna(Y, @mue_next_x, @mue_next_a, N);
= optymalizacja_naprzemienna(Y, @als_next_x, @als_next_a, N);
time] = optymalizacja_naprzemienna(Y, @hals_next_x, @hals_next_a, N);
```

Algorytm MUE

Algorytm multiplikatywny minimalizuje niepodobieństwo zbiorów rozumiane jako kwadrat z normy Fobiusa różnicy między macierzą obserwacji a iloczynem macierzy estymowanych:

$$D(\boldsymbol{Y}||\boldsymbol{AX}) = \frac{1}{2}\,||\,\boldsymbol{Y} - \boldsymbol{AX}\,||_F^2$$

Algorytm ten zakłada następujące reguły iteracyjne:

$$a_{ij} \leftarrow a_{ij}\frac{[\boldsymbol{YX^T}]_{ij}}{[\boldsymbol{AXX^T}]_{ij}}$$

$$x_{jt} \leftarrow x_{jt}\frac{[\boldsymbol{A^TY}]_{jt}}{[\boldsymbol{A^TAX}]_{jt}}$$

Konieczne jest także skalowanie (normalizacja) jednej z macierzy (A). Reguły iteracyjne zdefiniowano jako odpowiednie funkcje:

```
% Alg. MUE
function [A_next] = mue_next_a(A, X, Y)
    A_next = A.*(Y*X')./(A*(X*X') + eps);
    A_next = A_next * diag( 1 ./ sum(A_next, 1));
end

function [X_next] = mue_next_x(A, X, Y)
    X_next = X.*(A'*Y)./(A'*A*X);
end
% /Alg. MUE
```

Rozwiązanie jest klasy BLAS-3, co oznacza że wykorzystuje mnożenie macierz razy macierz, zamiast pętli for.

Algorytm ALS

Algorytm ALS wykorzystuje pojęcie pseudoodwrotności $\boldsymbol{A}^+$, które pozwala rozwiązać równanie macierzowe tak, jak gdyby macierz $\boldsymbol{A}$ była macierzą kwadratową.

$$\boldsymbol{Y} = \boldsymbol{A}^+\boldsymbol{X}$$

Gdzie $\boldsymbol{A}^+ = (\boldsymbol{A}^T\boldsymbol{A} + \alpha_A\mathbb{I})^{-1}\boldsymbol{A}^T$. W rozwiązaniu przyjęto $\alpha_A = 0$.

Konieczne jest zapewnienie, że wszystkie elementy macierzy są nie ujemne – więc te, które byłyby ujemne zostają wyzerowane.

Wówczas reguła iteracyjna dla macierzy X przyjmuje postać:

$$\boldsymbol{X = A^{+}Y}$$

```
function [X_next] = als_next_x(A, X, Y)
    % X = A^+ * Y
    X_next = inv(A' * A) * A' * Y;
    X_next = max(0, X_next);
end
% /Alg. ALS
```

W analogiczny sposób wykorzystano pseudoodwrotność macierzy $\boldsymbol{X}$.

$$\boldsymbol{A = YX^{+}}$$

```
function [A_next] = als_next_a(A, X, Y)
    % A = Y * X^+
    A_next = Y * (X' * inv(X*X'));
    A_next = max(0, A_next);
end
```

Dlaczego do funkcji als_next_a przekazywana jest bieżąca wartość macierzy A, mimo że algorytm jej nie wykorzystuje? (Podobnie als_next_x z macierzą X).

Jest to spowodowane założeniem o przekazywaniu funkcji opisujących korki iteracyjne w poszczególnych algorytmach jako argumentów funkcji optymalizacja_naprzemienna.

W związku z tym, musza posiadać one taką samą sygnaturę.

Algorytm HALS

W algorytmie HALS obowiązuje następujące założenie dot. funkcji niepodobieństwa:

$$D(\boldsymbol{Y}||\boldsymbol{AX}) = \frac{1}{2}\,||\,\boldsymbol{Y} - \boldsymbol{AX}\,||_F^2 = \frac{1}{2}\,||\,\boldsymbol{Y} - \sum_{\boldsymbol{r\neq j}} \boldsymbol{a_r x_r} - \boldsymbol{a_j x_j}\,||_F^2$$

Estymacja kolejnej wartości macierzy **A** odbywa się zgodnie z regułą operującą na kolumnach:

$$a_j \leftarrow \left[a_j + \frac{[\boldsymbol{YX^T}]_{*j} - \boldsymbol{A}[\boldsymbol{XX^T}]_{*j}}{[\boldsymbol{XX^T}]_{jj}}\right]_+$$

Jednak tutaj konieczna jest już pętla for, więc mnożenie nie odbywa się na poziomie macierzy, a wektorów (BLAS-2). Wykorzystuje się również dodatkową, zewnętrzną pętlę o niewielkiej liczbie iteracji. W rozwiązaniu przyjęto liczbę 5. Konieczna jest także projekcja, celem usunięcia ujemnych elementów macierzy.

```
% Alg. HALS
function [A_next] = hals_next_a(A, X, Y)
    C = Y * X';
    B = X * X';
    J = size(A, 2); % J kolumn
    A_next = A;
    for k=1:5
        for j=1:J
            A_with_negatives = A_next(:, j) +  (C(:, j) - A_next * B(:, j)) / B(j, j);
            A_next(:, j) = max(0, A_with_negatives);
        end
    end
end
```

Podobnie dla macierzy X, obowiązuje reguła iteracyjna (tutaj operuje się na wierszach):

$$x_j \leftarrow \left[x_j + \frac{[\boldsymbol{A}^T\boldsymbol{Y}]_{j*} - [\boldsymbol{A}^T\boldsymbol{A}]_{*j}\boldsymbol{X}}{[\boldsymbol{A}^T\boldsymbol{A}]_{jj}} \right]_+$$

```
function [X_next] = hals_next_x(A, X, Y)
    C = A' * Y;
    B = A' * A;
    J = size(X, 1); % J wierszy
    X_next = X;
    for k=1:5
        for j=1:J
            X_with_negatives = X_next(j, :) + (C(j, :) - B(j, :) * X_next) / B(j, j);
            X_next(j, :) = max(0, X_with_negatives);
        end
    end
end
% / Alg. HALS
```

Obliczanie błędów i przygotowanie badań

Do wyznaczania błędów (residualnego, średniokwadratowego) oraz stosunku SIR zdefiniowano funkcję calcErrors, która jako parametry przyjmuje macierz estymowaną oraz oryginalną.

```
function [res, mse, sir] = calcErrors(Y_est, Y)
    res = norm(Y - Y_est)/norm(Y);
    mse = immse(Y_est, Y);
    sir = mean(CalcSIR(normalize(Y), normalize(Y_est)));
end
```

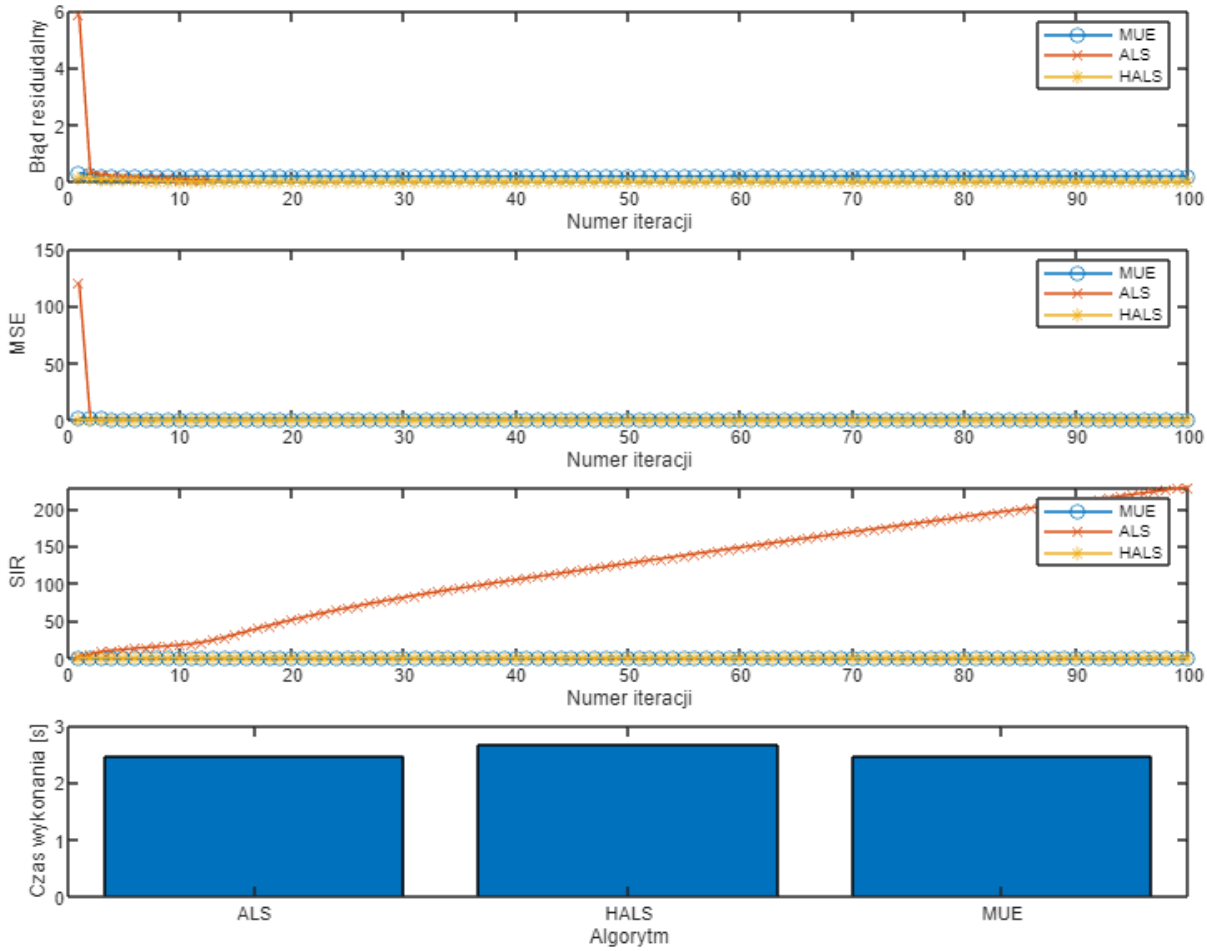
Do obliczania SIR skorzystano z funkcji CalcSIR pobranej ze strony:

https://github.com/andrewssobral/TDALAB/blob/master/CalcSIR.m

Przeprowadzono badania dla 100 iteracji, a wyniki zilustrowano wykresami.

Wyniki badań

Wyniki badań przedstawiono w formie wykresów dla różnych badanych parametrów.



Obserwacje:

- wyniki ALS wydają się nieco dziwne (bardzo wysoka wartość SIR).

Metoda NTF

Tensorowe metody redukcji wymiarowości pozwalają na rozkład tensora obserwacji na określoną liczbę macierzy – faktorów $\mathbf{U}_i$. W przypadku NMF faktoryzacji podlegała macierz obserwacji – a więc tensor 2-rzędu (o 3 modach). W przypadku NTF może być to również tensor $\boldsymbol{\mathcal{Y}}$ wyższego rzędu - np. trzeciego, tak jak w zadaniu 2 i 3.

W przypadku NMF stosowano litery - $\boldsymbol{A}$, $\boldsymbol{X}$ – na określenie macierzy czynnikowych. Jednak z uwagi na to, że w przypadku metod tensorowych rząd tensora może być wyższy niż liczba dostępnych liter w alafabecie, macierze czynnikowe oznacza się literą $\boldsymbol{U}^i$ gdzie i należy do przedziału - {1, N} (N – liczba modów tensora obserwacji). Zatem:

$$\boldsymbol{\mathcal{Y}} = \sum_{j=1}^{J} u_j^{(1)} \circ u_j^{(2)} \circ \ldots \circ u_j^{(N)} = \boldsymbol{I} \times_1 \boldsymbol{U^{(1)}} \times_2 \boldsymbol{U^{(2)}} \times_3 \ldots \times_N \boldsymbol{U^{(N)}} \quad (model\ CP)$$

Lub

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{G} \times_1 \boldsymbol{U^{(1)}} \times_2 \boldsymbol{U^{(2)}} \times_3 \ldots \times_N \boldsymbol{U^{(N)}}\ (model\ dekompozycji\ Tuckera)$$

Sposób wyznaczenia macierzy czynnikowych $\boldsymbol{U^{(1)}} \times_2 \boldsymbol{U^{(2)}} \times_3 \ldots \times_N \boldsymbol{U^{(1)}}$ zależy od wybranego algorytmu.

Zadanie 2

Generowanie syntetycznych obserwacji

Podobnie jak w poprzednim zadaniu, pierwszym krokiem jest syntetyczne wygenerowanie próbek – wygenerowano 3 losowe nieujemne macierze **U{i}**, które przechowywano w tablicy komórek (cell array).

```
% Generowanie faktorów
I = [10 20 30]; % liczby elementów w poszczególnych modach
J = 5; % rząd faktoryzacji
U{1} = max(0, rand(I(1), J));
U{2} = max(0, rand(I(2), J));
U{3} = max(0, rand(I(3), J));
```

Na podstawie macierzy U{i} wygenerowano tensor syntetycznych obserwacji, zgodny z modelem CP (tzn. wagi wynoszą 1).

```
% Generowanie syntetycznych obserwacji
% ones(J, 1) - wektor kolumnowy wag (wszystkie takie same i równe 1)
Y=ktensor(ones(J, 1), U);
```

Do wygenerowania syntetycznych obserwacji wykorzystano funkcję ktensor pakietu Tensor Toolbox. Według oficjalnej dokumentacji:

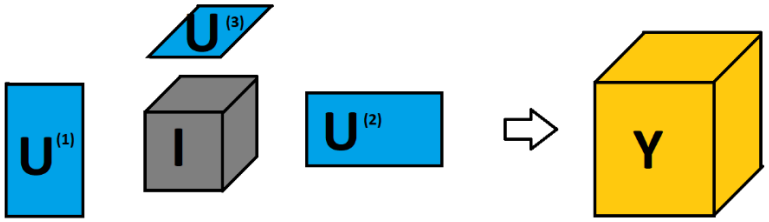
> *Kruskal format is a decomposition of a tensor X as the sum of the outer products as the columns of matrices*[1]

Dalej jest również informacja, że klasa ktensor przechowuje faktory do utworzenia tensora i wykonuje operacje charakterystyczne tensora bez jawnego formatowania tensora Y.

Zatem najpierw wygenerowano syntetyczne obserwacje:

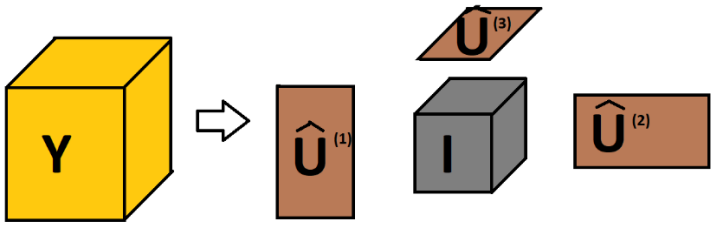
[1] https://www.tensortoolbox.org/ktensor_doc.html

$$\boldsymbol{I} \times_1 \boldsymbol{U^{(1)}} \times_2 \boldsymbol{U^{(2)}} \times_3 \boldsymbol{U^{(3)}} = \boldsymbol{\mathcal{y}} \ (\boldsymbol{ktensor})$$



po to, żeby następnie, przy użyciu algorytmu CP ALS rozłożyć syntetycznie wygenerowany tensor na macierze czynnikowe (dokonać dekompozycji CP ALS):

$$\boldsymbol{\mathcal{y}} = \boldsymbol{I} \times_1 \boldsymbol{\hat{U}^{(1)}} \times_2 \boldsymbol{\hat{U}^{(2)}} \times_3 \boldsymbol{\hat{U}^{(3)}}$$



Implementacja algorytmu dekompozycji CP ALS

Deklaracja funkcji

Algorytm dekompozycji CP ALS zaimplementowano jako funkcję CP jako argumenty:

- Y – N-wymiarowy tensor obserwacji (syntetycznie wygenerowanych, a więc **ktensor**),
- P – typ normy stosowanej do normalizacji wektorów,
- J – rząd faktoryzacji,
- Iterations – liczba przebiegów pętli,
- unfold – funkcję służącą do matrycyzacji tensora względem n-tego modu,

Funkcja zwraca:

- U – tablicę komórek (cel array) zawierającą poszczególne faktory estymowane U{i}
- errors – 2 – wymiarową tablicę błędów / wskaźników, której kolumny odpowiadają:
  - errors(: ; 1) – wartościom błędu residualnego w kolejnych iteracjach,
  - errors(: ; 2) – wartościom błędu średniokwadradowego w kolejnych iteracjach
  - errors(: ; 3) – wartościom współczynnika SIR w kolejnych iteracjach

Pierwszym krokiem właściwego algorytmu jest losowa inicjalizacja faktorów z warunkiem nieujemności elementów. Tensor jest konwertowany na tablicę wielowymiarową, co jest później wykorzystywane w operacji matrycyzacji.

Wyznaczanie faktorów powtórzone jest *iterations* razy w pętli zewnętrznej.

W pętli wewnętrznej wyznacza się wartkości kolejnych faktorów od 1 do N – na poniższym wycinku kodu**, pętla ta została zwinięta, gdyż zostanie szerzej omówiona w kolejnym punkcie.**

Na końcu każdej iteracji, kiedy wyznaczone jest już N faktorów, uśredniane są błędy (i wskaźniki) estymacji poszczególnych faktorów **U{i}** (dla danej iteracji).

```
function [U, errors] = CP(Y, p, J, iterations, unfold)
%CP wykonuje dekompozycje tensora Y na faktory z rzędem faktoryzacji J
% p określa normę zastosowaną do normalizacji faktorów
% itenrations określa liczbę iteracji
% unfold to funkcja o sygnaturze (tensor, mod) => macierz
% przekazanie funkcji unfold jako praramter pozwala w przyszłości rozwinąć
% ten program dla liczby modów > 3
% (obecnie zaimplementowaliśmy tylko unfold dla N=3 modów)

   % U_original to cell array oryginalnych faktorów, która służy do liczenia
   % błędów
   U_original = Y.u;
   % Pomiar błędów w funkcji iteracji
   errors = zeros(iterations, 3);

   % N - liczba modów
   I = size(Y);

   % losowa inicjalizacja faktorów
   U = cell(1, 3);
   for i=1:size(I, 2)
       U{i} = max(0, rand(I(i), J));
   end

   N = size(Y.u, 1); % liczba modów
   Y_arr = double(Y); % konwersja tensora na tablicę wielowymiarową

   % wyznaczanie faktorów powtórzone iterations razy
   for i = 1:iterations
       % błędy dla poszczgólnych faktorów w iteracji
       n_errors = zeros(N, 3); % mierzymy 3 typy błędów

       % przemiatanie po wszystkich (N) modach tensora obserwacji Y
       for n = 1:N···
       end

       % uśrednione błędy ze wszystkich modów dla i-tej iteracji
       errors(i, :) = mean(n_errors);
   end
end
```

Pętla wewnętrzna

Wewnętrzna pętla (w danej iteracji) przemiata po poszczególnych modach tensora wykonując następujące operacje:

- **matryzycacja tensora** względem n-tego modu, z wykorzystaniem funkcji unfold przekazanej jako argument,

```
% przemiatanie po wszystkich (N) modach tensora obserwacji Y
for n = 1:N
    % Matrycyzacja względem n-tego modu
    Yn = unfold(Y_arr, n);
```

  Funkcja unfold jest przekazywana jako argument, aby implementacja działała dla innych n (również większych od 3). Jednak w ramach laboratorium zaimplementowano jedynie funkcję działającą dla liczby modów równej 3, stąd jest ona potem przekazywana jako argument przy wywołaniu funkcji CP.

```
function Y_unfolded = unfold3(Y, mode)
    % Przeprowadza matrycyzację tensora Y względem modu mode
    % Y - tensor 3-modalny do poddania matrycyzacji
    % mode - mod, względem którego ma być przeprowadzona matrycyzacja
    switch mode
        case 1
            permutation = [1 2 3];
        case 2
            permutation = [2 1 3];
        case 3
            permutation = [3 1 2];
        otherwise
            err("Mode number exeeded maximum allowed value (3)");
    end
    Y_permuted = permute(Y, permutation);
    dim1 = size(Y, permutation(1));
    dim2 = size(Y, permutation(2)) * size(Y, permutation(3));
    Y_unfolded = reshape(Y_permuted, [dim1 dim2]);
end
```

Implementacja funkcji unfold dla tensora o 3 modach.

```
% dekompozycja tensora obserwacji i pomiar błędów
[U_est, err] = CP(Y, 2, J, 30, @unfold3);
```

Przekazanie funkcji unfold3 przy wywołaniu funkcji CP.

- **Estymacja n-tego faktora** za pomocą metody najmniejszych kwadratów. Model CP zakłada, że n-tą macierz czynnikową wyznacza się z poniższego równania:

$$\mathcal{Y}_{(n)} = \boldsymbol{U}^{(n)}(\boldsymbol{U}^{\odot -n})^{T}$$

Gdzie:

  - $\mathcal{Y}_{(n)}$ – matryzyzacja tensora względem n-tego modu,
  - $\boldsymbol{U}^{(n)}$ – n-ta macierz czynnikowa,
  - $\boldsymbol{U}^{\odot -n}$ – iloczyn Kathri-Rao poszczególnych macierzy czynnikowych, ustawionych w kolejności od N do 1 z pominięciem macierzy n-tej.

Jednak skorzystano z pewnego przybliżenia, które zakłada wyznaczenie $\boldsymbol{U}^{(n)}$ z następującego równania:

$$\boldsymbol{U}^{(n)} = (\mathcal{Y}_{(n)}\boldsymbol{U}^{\odot -n})\boldsymbol{B}^{-1}$$

Macierz B jest przybliżeniem iloczynu $(\boldsymbol{U}^{\odot -n})^{T}(\boldsymbol{U}^{\odot -n})$, za pomocą iloczynów Hadamarda.

$$\boldsymbol{B} = (\boldsymbol{U}^{(1)})^{T}(\boldsymbol{U}^{(1)}) \star \ldots \star (\boldsymbol{U}^{(N)})^{T}(\boldsymbol{U}^{(N)})$$

Wyznaczanie $\boldsymbol{B}$ zaimplementowano w pętli:

```
% Iloczyn Hadamana faktorów poza n-tym
indexes_without_n = copy_without_element(1:N, n);
B = ones(J, J);
for j = indexes_without_n
    B = B .* (U{j}' * U{j});
end
```

Wyznaczanie $\boldsymbol{B}$

Wyznaczanie $\boldsymbol{U}$ zaimplementowano w pętli:

```
% Iloczyn Khatri-Rao faktorów poza n-tym
kr_indexes = flip(indexes_without_n);
for j = 1:size(kr_indexes, 2)-1
    current_index = kr_indexes(j);
    next_index = kr_indexes(j+1);
    U_kr = kr(U{current_index}, U{next_index});
end
```

Wyznaczanie $\boldsymbol{U^{\odot -n}}$

Do obliczenia iloczynu Kathri-Rao między macierzami U{i} skorzystano z funkcji *kr* pobranej ze strony:
https://www.mathworks.com/matlabcentral/fileexchange/28872-khatri-rao-product?focused=5171022&tab=function

Następnie wyznaczana jest macierz czynnikowa $\boldsymbol{U^{(n)}}$**:**

```
% Estymacja U{n}
U{n} = (Yn * U_kr) / B;
```

Każda macierz za wyjątkiem N-tej jest poddawana normalizacji zgodnie z zadaną normą p.

```
% Skalowanie n-tego faktora
if n ~= N
    coefficient = inv(diag(vecnorm(U{n}, p)));
    U{n} = U{n} * coefficient;
end
```

- Ostatnim krokiem pętli wewnętrznej jest **obliczenie błędów** przybliżenia n-tej (bieżącej) macierzy czynnikowej w bieżącej (i-tej) iteracji:

```
    % Liczenie błędów dla n-tego modu w i-tej iteracji
    [res, mse, sir] = calcErrors(U{n}, U_original{n});
    n_errors(n, :) = [res, mse, sir];
end
```

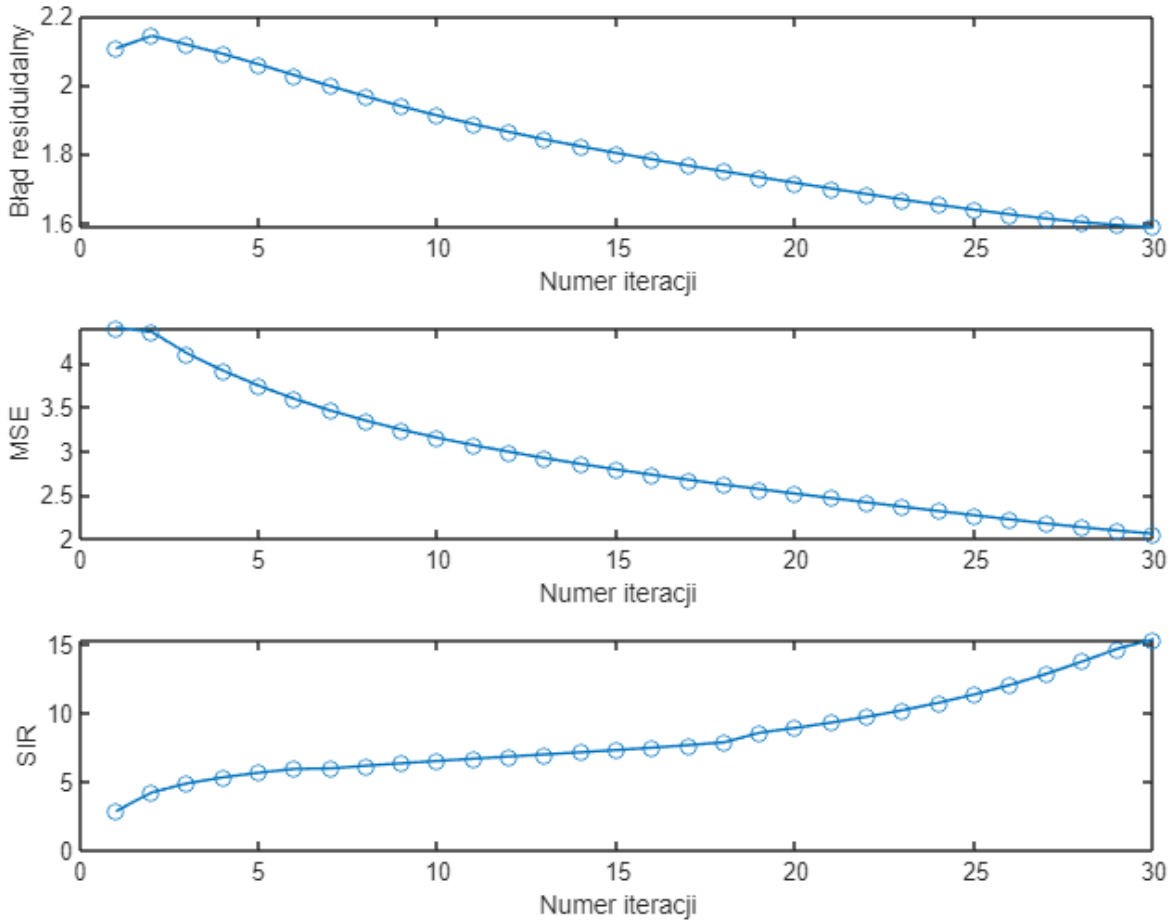
Można zauważyć, że błędy są liczone pomiędzy estymacją macierzy czynnikowej $\widehat{\boldsymbol{U}}^{(\boldsymbol{n})}$ a „oryginalną" macierzą czynnikową $\boldsymbol{U^{(n)}}$**.** Takie podejście jest możliwe jedynie w przypadku syntetycznie wygenerowanych obserwacji (**ktensor**). W przypadku rzeczywistych obserwacji (niesyntetycznych) taka sytuacja nie byłaby możliwa bowiem obserwacje te nie byłyby wynikiem sztucznego wygenerowania z założonych macierzy czynnikowych $\boldsymbol{U^{(n)}}$**.**

Wyniki badań

Zmianę poszczególnych wskaźników:

- Średniej wartości błędu residualnego dla wszystkich macierzy czynnikowych,
- Średniej wartości błędu średniokwadratowego dla wszystkich macierzy czynnikowych,
- Średniej współczynnika dla wszystkich macierzy czynnikowych,

Dla wszystkich macierzy czynnikowej zbadano w funkcji numeru iteracji. Wyniki przedstawiono na wykresach.


Błąd residualny
2.2
2
1.8
1.6
0
5
10
15
20
25
30
Numer iteracji

MSE
4
3.5
3
2.5
2
0
5
10
15
20
25
30
Numer iteracji

SIR
15
10
5
0
0
5
10
15
20
25
30
Numer iteracji

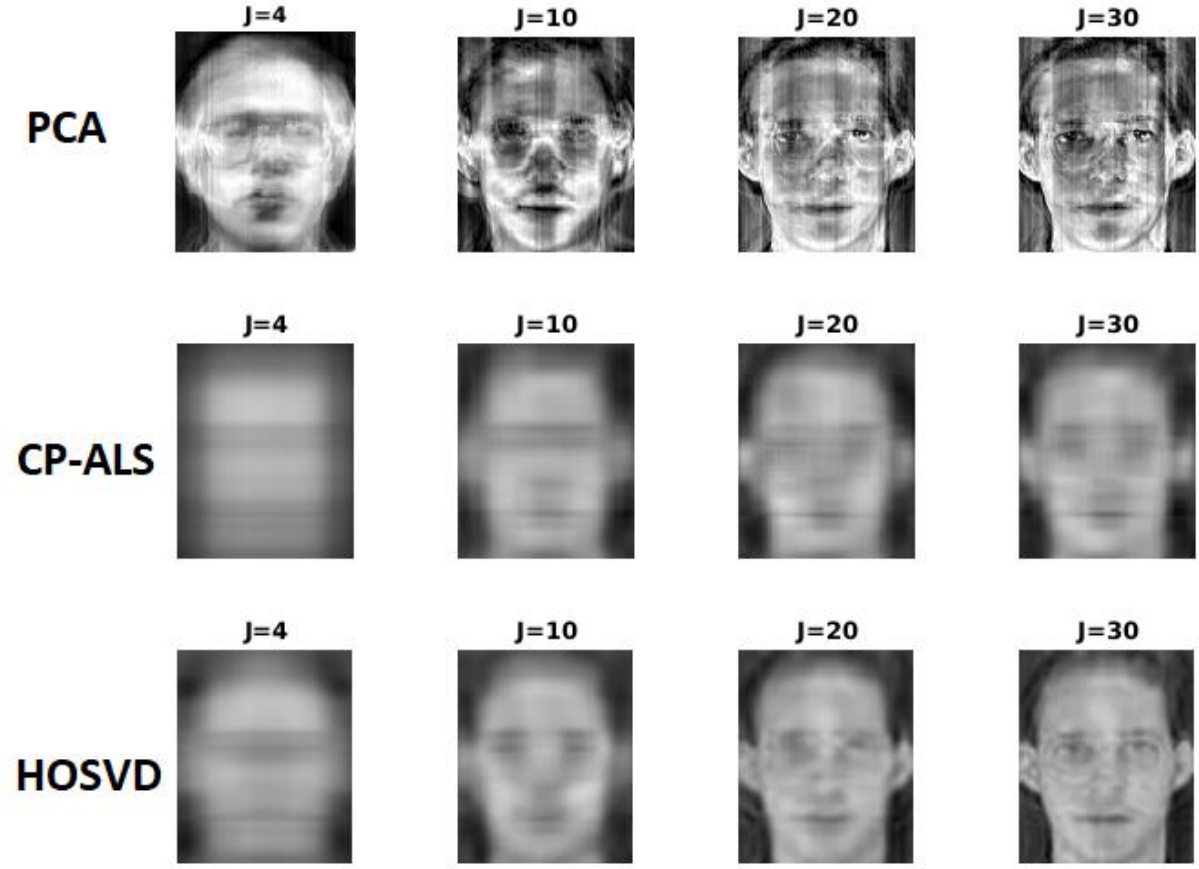
Zadanie 3

Zdanie 3. jest o tyle trudniejsze od zadania 2., że nie tensor obserwacji nie jest tutaj syntetycznie generowany, a tworzony jest na podstawie zbioru danych zdjęć ORL.

Ocena wizualna obrazów uzyskanych ze zredukowanych wymiarów dla różnych metod redukcji wymiarowości

Pierwszym krokiem było wczytanie zdjęć do tablicy 3-wymiarowej. Zdefiniowano funkcję podobną do tej z listy 1, z tym że tutaj wczytanie odbywało się do tablicy 3-wymiarowej, a nie 2-wymiarowej.

Następnie dokonano wizualnej oceny jakości odtworzenia zdjęcia z użyciem różnych metod redukcji wymiarowości oraz dla różnej liczby J (liczby wektorów głównych w PCA lub stopnia faktoryzacji w NTF). Rezultaty można zaobserwować na poniższym rysunku.



Badanie polegało więc na redukcji wymiarowości zbioru (tensora) obserwacji (zdjęć) a następnie na odtworzeniu obrazów z faktorów dla różnych metod oraz ocenie wizualnej pierwszego z tych obrazów. Badanie przeprowadzono w pętli dla różnych wartości parametru J (4, 10, 20, 30).

```
for i=1:number_of_J_values
    J = J_values(i);

    % Dekompozycja tensora treningowego wg różnych algorytmów
    U_cp_als = CP_ALS(Y, 2, J, cp_als_iterations, @unfold3); % faktory estymowane CP-APLS
    [U_hosvd, G] = HOSVD(Y, [J J J], @unfold3); % faktory estymowane HOVD
    [U_pca, Z] = PCA(Y_pca, J); % wektory cech estymowane PCA
```

Redukcja wymiarowości metodą PCA

W PCA obraz był odtwarzany z użyciem funkcji PCA zdefiniowany w sposób analogiczny do listy 1, z tym że kod przeniesiono do osobnej funkcji co pozwoliło na jego wielokrotne użycie bez powtarzania.

```
function [U, Z] = PCA(Y, J)
    %PCA(Y, J) dokonuje dekompopozycji macierzy obserwacji
    % param Y - macierz obserwacji
    % param J - liczba składowych głównych
    % returns U - macierz cech
    % returns Z - macierz składowych głównych
    covariance_matrix = double(Y) * double(Y');
    % Wyznaczenie J-wektorów własnych i J-wartości własnych
    [eigen_vectors, eigen_values] = eigs(covariance_matrix, J);
    U = eigen_vectors;     % macierz wektorów cech
    % Wyznaczenie i normalizacja macierzy składowych głównych
   % Z = double(Y') * U;
    Z = U' * double(Y);
end
```

Funkcja pobiera macierz obserwacji $\boldsymbol{Y}$ oraz liczbę składowych głównych (twarzy własnych) $\boldsymbol{J}$ zwraca parę (tablicę) $[\boldsymbol{U}, \boldsymbol{Z}]$, gdzie $\boldsymbol{U}$ oznacza wektory cech (macierz), a $\boldsymbol{Z}$ – macierz składowych głównych. (twarze własne).

Ponieważ PCA operuje na 2-wymiarowej tablicy obserwacji, dla tej metody najpierw przeprowadzono matrycyzację 3-wymiarowej tablicy obserwacji (wczytanej z plików ze zdjęciami)

```
% dla PCA trzeba zmienić macierz 3D na 2D
Y_pca = unfold3(pictures, 3);
```

Obserwacje można aproksymować jako iloczyn odpowiadających wag i komponentów głównch:

$$\widetilde{\boldsymbol{Y}} = \boldsymbol{UZ}$$

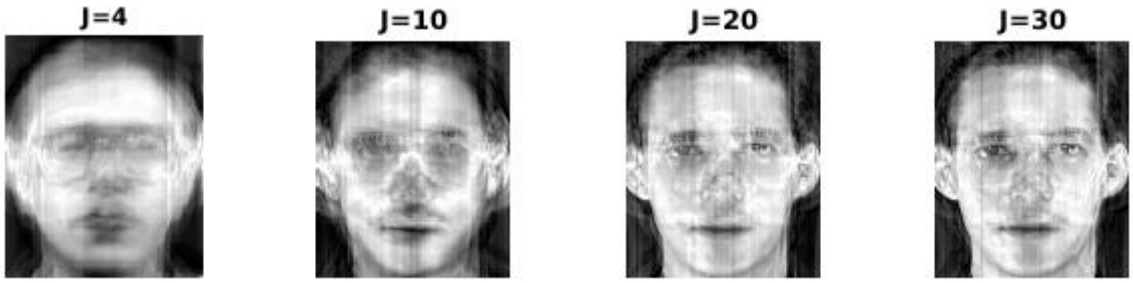
Zastosowanie tego wzoru można zaobserwować bezpośrednio poniżej:

```
% odtworzenie obrazów PCA
Y_est_pca = U_pca * Z;
Y_est_pca_3D = fold3_3(Y_est_pca, I);
pca_recreated_images = 255 * normalize(Y_est_pca_3D, 'range');
```

Poza wyznaczeniem macierzy estymowanej $\widetilde{\boldsymbol{Y}} = \boldsymbol{UZ}$ widoczne są też 2 kolejne kroki:

- zmiana liczby modów z 2 do 3 (operacja odwrotna do matryzyzacji – więcej w punkcie ***Uwaga 1***)
- normalizacja i skalowanie wyników – żeby uzyskać zakres 0 – 255.

Wyświetlenie pierwszego odtworzonego obrazu (pierwszego obrazu pierwszej osoby) dla różnej liczby składowych głównych J dało następujące efekty:



Uwaga 1:

Po obliczeniu estymowanej macierzy 2-wymiarowej konieczne było znowu przekształcenie jej do macierzy 3-wymiarowej w celu wyświetlenia. Prosta funkcja reshape na macierzy estymowanej nie sprawdzała się tutaj, ponieważ brała piksele w złej kolejności (wyświetlana była czarno-biała kratka). Zaimplementowano więc funkcję fold3_3, która realizuje operację odwrotną do matrycyzacji 3-wymiarowej macierzy względem 3. modu.

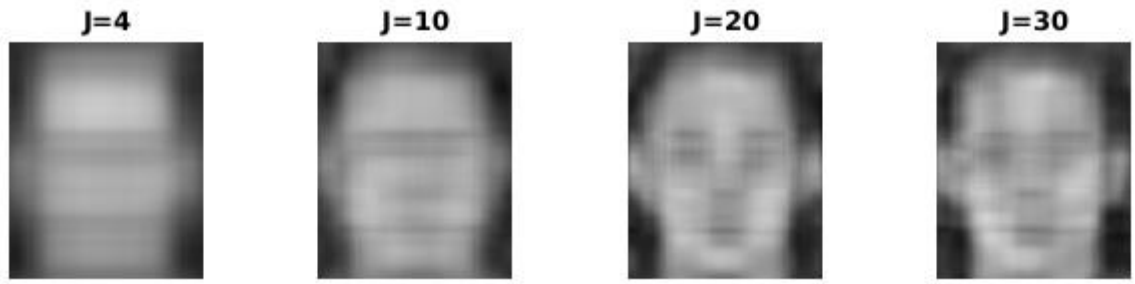
```
function Y_3D = fold3_3(Y_2D, I)
    %FOLD3 Cofa operacje matrycyzacji tensora 3-modalnego względem 3-modu
    % param Y_2D - macierz 2-wymiarowa będąca wynikiem matrycyzacji pewnego
    % tensora 3-modalnego względem 3-modu
    % param I - 3-elementowy wektor wymiarów tensora wynikowego
    % Y_3D - odtworzony tensor (tablica 3D) sprzed matrycyzacji
    Y_3D = zeros(I);
    for i=1:I(3)
        Y_3D(:, :, i) = reshape(Y_2D(i, :), [I(1),  I(2)]);
    end
end
```

Redukcja wymiarowości metodą NTF – algorytm dekompozycji CP-ALS

Do dekompozycji tensora obserwacji metodą CP-ALS wykorzystano lekko zmodyfikowaną funkcję z zadania 2. Usunięto bowiem kroki odpowiadające za pomiar błędów (residualnego, MSE).

Teoria związana z algorytmem CP-ALS została opisana w poprzednim zadaniu – tutaj zostanie więc odtworzenie obserwacji uprzednio zdekomponowanych algorytmem CP-ALS.

```
% odtworzenie obrazów CP ALS
Y_est_cp_als = ktensor(ones(J, 1), U_cp_als);
cp_als_recreated_images = double(Y_est_cp_als);
```



Redukcja wymiarowości metodą NTF – algorytm dekompozycji HOSVD

Algorytm High Order SVD został zaimplementowany jako funkcja HOSVD. Funkcja przyjmuje na wejściu parametry:

- N-wymiarowy tensor obserwacji Y,
- J – wektor rzędów faktoryzacji,
- *unfold* – funkcję służącą do matrycyzacji tensora.

Funkcja zwraca:

- U – tablicę komórek (cel array) złożoną z poszczególnych faktorów,
- G – tensor rdzeniowy.

Przekazywanie funkcji *unfold* jako argumentu jest podyktowane tymi samymi względami co przy funkcji CP_ALS – zaimplementowano bowiem tylko funkcję unfold3, która przeprowadza matrycyzację tensora 3-wymiarowego. Gdyby jednak w przyszłości zaimplementować funkcję *unfold* dla tensorów wyższego rzędu – wystarczyłoby ją przekazać jako parametr.

Poza tym aspektem funkcja jest w zasadzie implementacją pseudokodu zaprezentowanego na wykładzie. Pętla for przemiata po wszystkich modach tensora obserwacji. Wewnątrz pętli następuje matrycyzacja tensora Y względem n-tego modu oraz wyznaczenie n-tej macierzy czynnikowej Un.

Tensor rdzeniowy można wyznaczyć ze wzoru:

$$\boldsymbol{G} = \boldsymbol{\mathcal{y}} \times_{\mathbf{1}} (\boldsymbol{U}^{(\mathbf{1})})^{\mathrm{T}} \times_{\mathbf{2}} (\boldsymbol{U}^{(2)})^{\mathrm{T}} \times_{\mathbf{3}} \ldots \times_{\boldsymbol{N}} (\boldsymbol{U}^{(\mathrm{N})})^{\mathrm{T}}$$

Operację tę implementuje biblioteczna funkcja ***ttm*** z parametrem „t” (transponse).

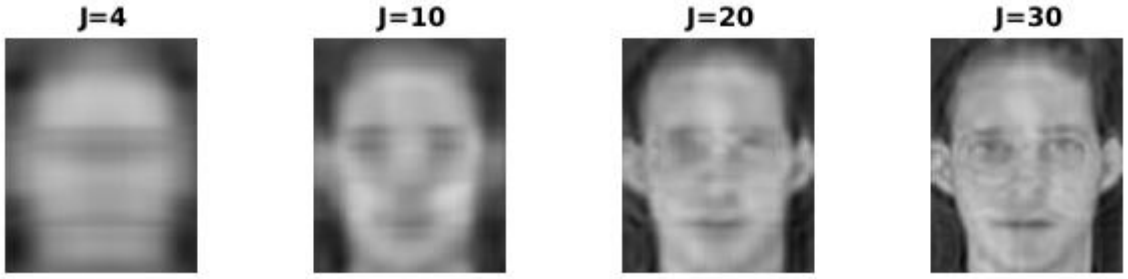
```
function [U, G] = HOSVD(Y, J, unfold)
%HOSVD dokonuje dekomozycji N-wymiarowego tensora obserwacji Y
% J to wektor określający poszczególne stopnie faktoryzacji - J1, ..., JN
% unfold to funkcja matrycyzująca tensor
% U - cell array estymowanych faktorów
% G - tensor rdzeniowy

    N = size(size(Y), 2); % liczba modów

    U = cell(1, N);
    Y_arr = double(Y); % konwersja tensora na tablicę wielowymiarową

    % przemiatanie po wszystkich (N) modach tensora obserwacji Y
    for n = 1:N
        % Matrycyzacja względem n-tego modu
        Yn = unfold(Y_arr, n);
        correlation_matrix = Yn * (Yn)';

        % Wyznacznienie J(n) wektorów wektorów własnych i wartości własnych
        % macierzy korelacji
        [eigen_vectors, eigen_values_matrix] = eigs(correlation_matrix, J(n));
        U{n} = eigen_vectors;
    end
    G = ttm(Y, U, 't'); % t oznacza transpozycję macierzy U
end
```

Tensor obserwacji można estymować (odtworzyć) z faktorów i tensora rdzeniowego za pomocą iloczynu:

Efekt taki można osiągnąć stosując funkcję biblioteczną ttensor.

```
% odtworzenie obrazów CP HSV
Y_est_hosvd = ttensor(G, U_hosvd);
hosvd_recreated_images = double(Y_est_hosvd);
```

Przybliżone obrazy dla różnych rzędów faktoryzacji przedstawiono na rysunku. Podany jest tylko jeden rząd, ponieważ założono, że wszystkie 3 rzędy faktoryzacji dla pojedynczego wywołania będą takie same (J = [4, 4, 4], J = [10, 10, 10] itd.).



Ocena wizualna

Wizualnie, wyniki najwierniejsze oryginałowi wydawała się dawać metoda PCA.

Badanie poprawności klasteryzacji metodą k-średnich aproksymowanego zbioru treningowego 50 obserwacji (5 grup)

Badania przeprowadzono na podzbiorze 50 zdjęć (5 osób każda po 10 zdjęć), aby można było wizualnie skontrolować jakość odtworzonych obrazów i poprawność klasteryzacji. Dokonywano redukcji wymiarowości tensora (3. rzędu w metodach tensorowych i 2. rzędu w metodzie PCA) zdjęć różnymi metodami, dla różnych wartości parametru J ([4, 10, 20, 30]), tzn. liczby wektorów głównych w PCA lub stopnia faktoryzacji w metodach tensorowych. Do drukowania macierzy konfuzji wykorzystano funkcję *plotConfMat* pobraną ze strony https://www.mathworks.com/matlabcentral/fileexchange/64185-plot-confusion-matrix.

Badania przeprowadzono zatem dla następujących parametrów (var oznacza zmienną w badaniu, a const stałą w badaniu).

$var\ \boldsymbol{metoda} = PCA, CP\text{-}ALS, HOSVD$

$var\ \boldsymbol{J} = 4{,}10{,}20{,}30$

const $\boldsymbol{liczba\ osób} = 50$

Do przeprowadzenia badań zdefiniowano skrypt w postaci funkcji zadanie3_clustering_test.

```
function zadanie3_clustering_test(persons_count, method)
    %ZADANIE3_CLUSTERINT_TEST porównuje jakość klasyfikacji i drukuje
    %odtworzone obrazy
    % param persons_count - liczba osób do wczytania z pliku
    % param method - jedna z metod redukcji wym.: 'PCA', 'CP-ALS', 'HOSVD'
```

Pomijając fragmenty związane z inicjalizacją zmiennych (tablic) i wczytywaniem zdjęć, główne fragment tej funkcji to pętla for przemiatająca po wartościach J, wewnątrz których wykonywane są kroki: dekompozycja tensora, odworzenie zdekompowanego tensora, klasteryzacja, odtworzenie obrazu i wyświetlenie wyników.

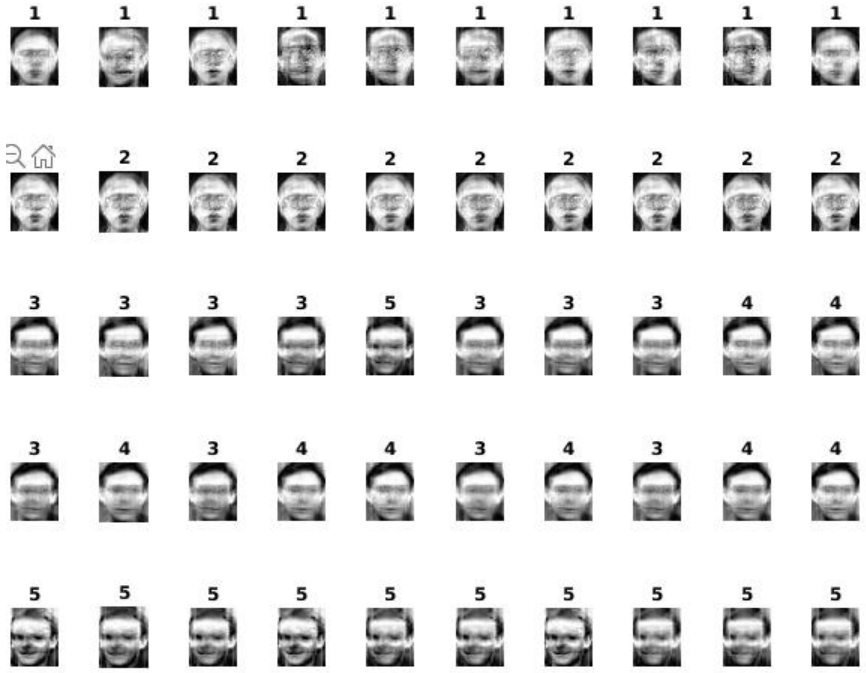
```
for i=1:number_of_J_values
    J = J_values(i);

    % W zależności od metody kroki wyglądają inaczej
    % 1. Dekompozycja tensora treningowego
    % 2. Odtworzenie (aproksymacja) tensora obserwacji
    % 3. Klastrowanie
    % 4. Odtworzenie obrazu
    switch method
        case 'PCA'
            [U_pca, Z] = PCA(Y_pca, J); % składowe główne dla PCA
            Y_est = U_pca * Z; % dla PCA
            clustering_labels = kmeans(Y_est, persons_count); % dla PCA
            Y_est_pca_3D = fold3_3(Y_est, I); % dla PCA
            recreated_images = 255 * normalize(Y_est_pca_3D, 'range'); % dla PCA
        case 'CP-ALS'
            U_cp_als = CP_ALS(pictures, 2, J, cp_als_iterations, @unfold3); % faktory
            Y_est = ktensor(ones(J, 1), U_cp_als); % dla CP ALS
            clustering_labels = kmeans(U_cp_als{3}, persons_count); % dla CP ALS
            recreated_images = double(Y_est); % dla HOSVD i dla CP ALS
        case 'HOSVD'
            [U_hosvd, G] = HOSVD(Y, [J J J], @unfold3); % faktory estymowane HOSVD
            Y_est = ttensor(G, U_hosvd); % dla HOSVD
            clustering_labels = kmeans(U_hosvd{3}, persons_count); % dla HOSVD
            recreated_images = double(Y_est); % dla HOSVD i dla CP ALS
    end

    % Ocena jakości klasteryzacji
    [Acc,rand_index,match] = AccMeasure(labels, clustering_labels');
    acc_measure_coeffitients(:, i) = [Acc,rand_index];
    match_transposed = match';
    acc_measure_mappings(:, :, i) = match_transposed;

    % wyświetlenie obrazów
```

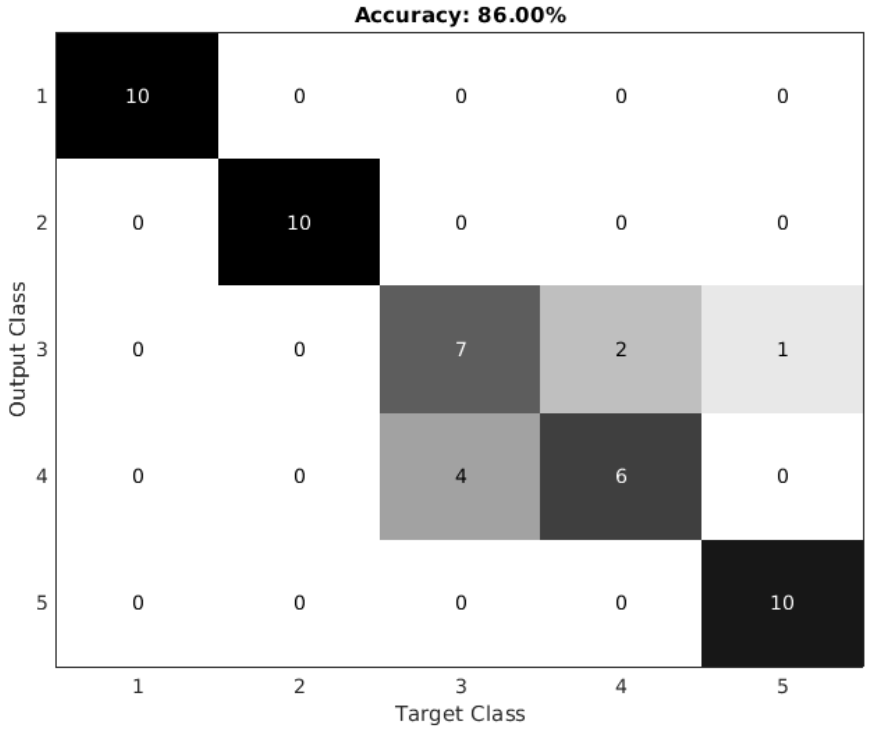
Badania dla PCA

W metodzie analizy składowych głównych do klasyfikacji wykorzystano cały aproksymowany zbiór obserwacji $\widetilde{\boldsymbol{Y}} = \boldsymbol{UZ}$.

Wyniki dla poszczególnych wartości J (liczby składowych głównych znajdują się poniżej):

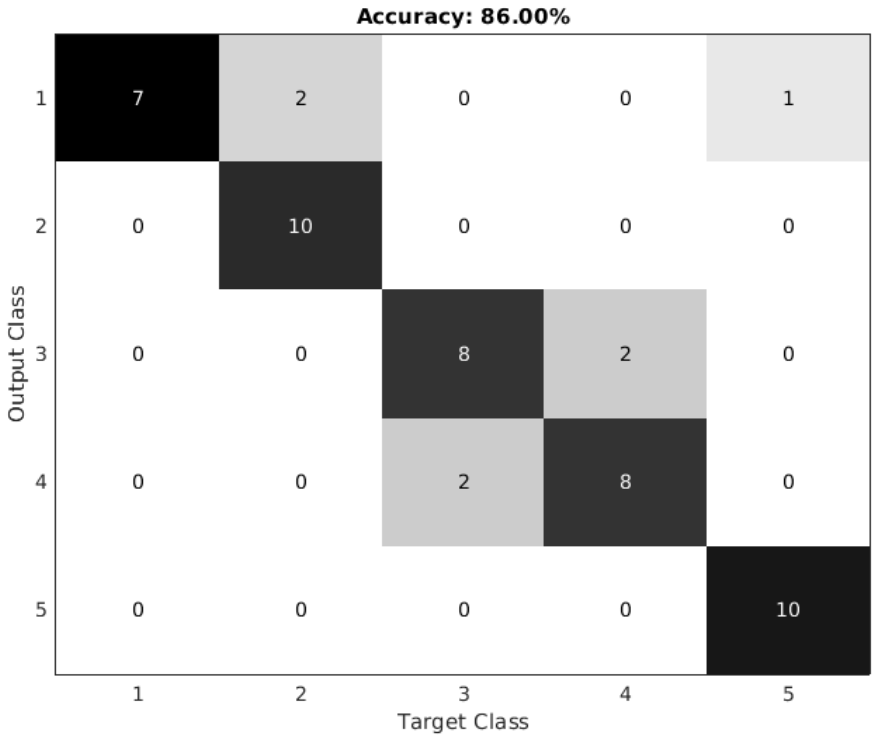
- PCA dla J=4





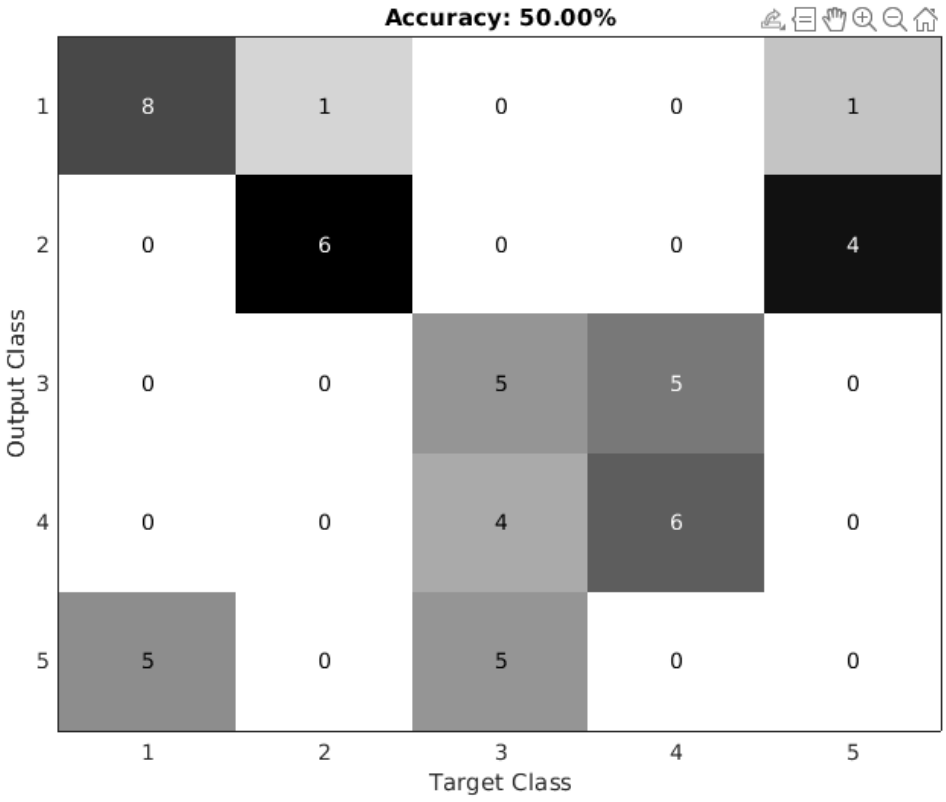
- PCA dla J=10


2 5 2 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 4 4 3 3 3 3 3
4 4 4 4 3 4 4 4 4 3
5 5 5 5 5 5 5 5 5 5


Accuracy: 86.00%
1 7 2 0 0 1
2 0 10 0 0 0
3 0 0 8 2 0
4 0 0 2 8 0
5 0 0 0 0 10
Output Class
1 2 3 4 5
Target Class

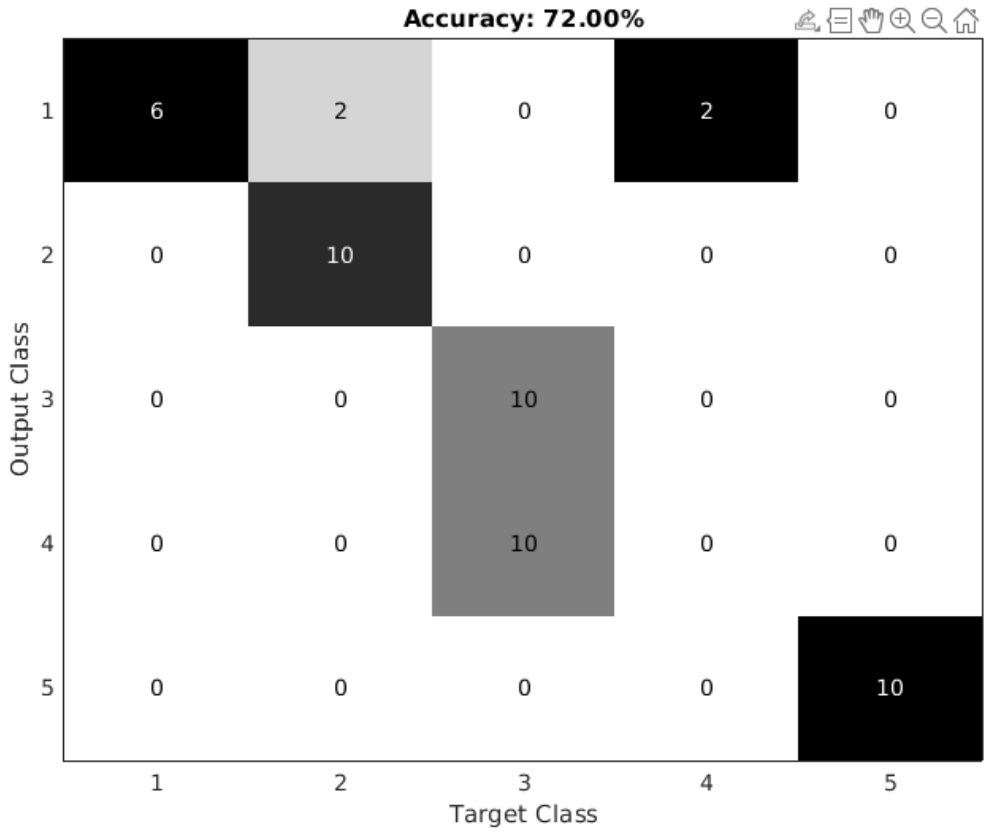
- PCA dla J=20


5 1 2 1 1 1 1 1 1 1
2 5 2 5 2 5 2 2 5 2
4 4 4 3 3 3 3 3 4 4
3 4 3 4 4 3 4 3 4 4
3 3 3 3 1 1 3 1 1 1


Accuracy: 50.00%
1 8 1 0 0 1
2 0 6 0 0 4
3 0 0 5 5 0
4 0 0 4 6 0
5 5 0 5 0 0
Output Class
1 2 3 4 5
Target Class

- PCA dla J=30


2 4 2 1 4 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3
5 5 5 5 5 5 5 5 5 5

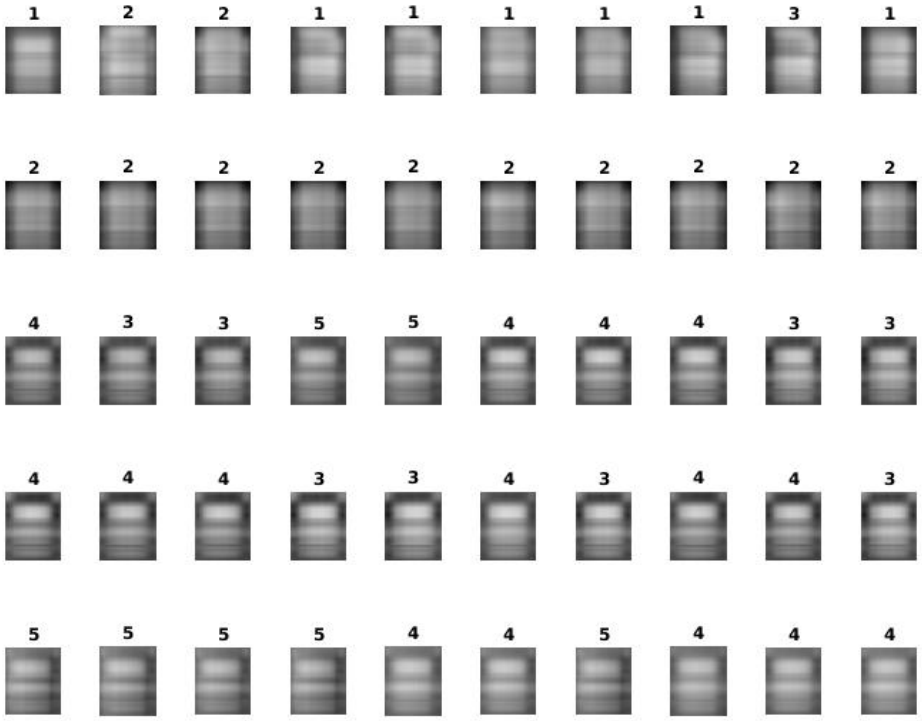

Accuracy: 72.00%
6 2 0 2 0
0 10 0 0 0
0 0 10 0 0
0 0 10 0 0
0 0 0 0 10
Output Class
1 2 3 4 5
Target Class

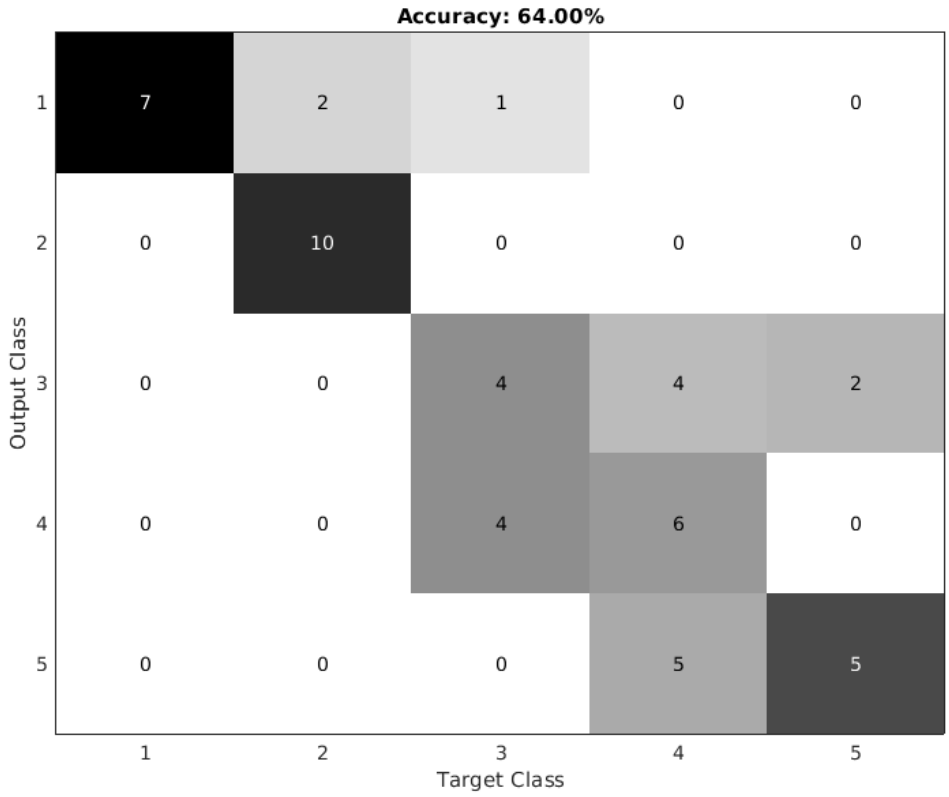
Badania dla CP-ALS

Przy dekompozycji CP-ALS do klasyfikacji wykorzystano faktor odpowiadający trzeciemu modowi - $\boldsymbol{U^{(3)}}$.

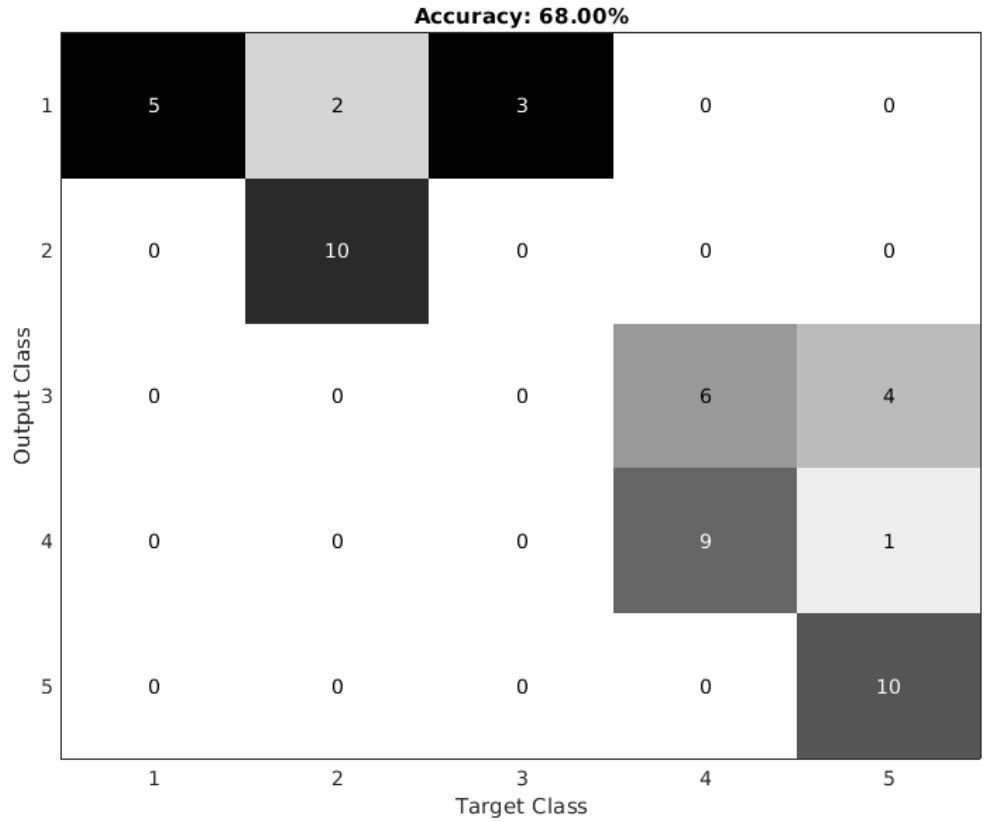
Wyniki dla poszczególnych wartości J (liczby składowych głównych znajdują się poniżej):

- CP-ALS dla J=4

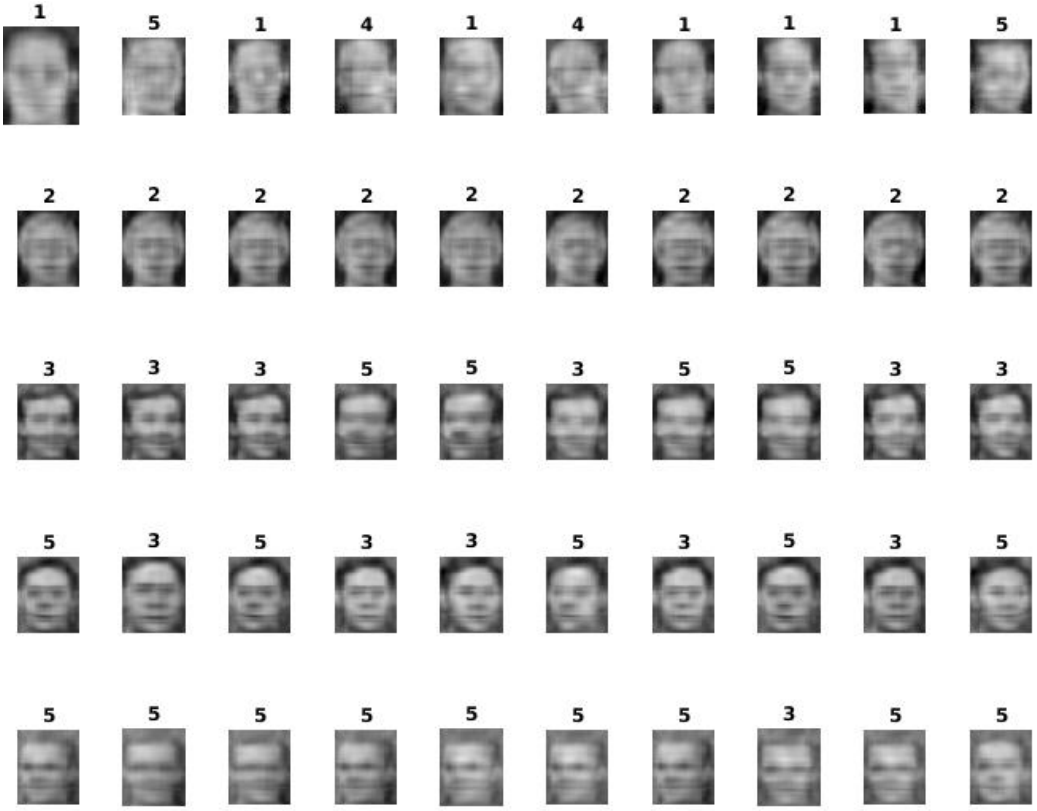


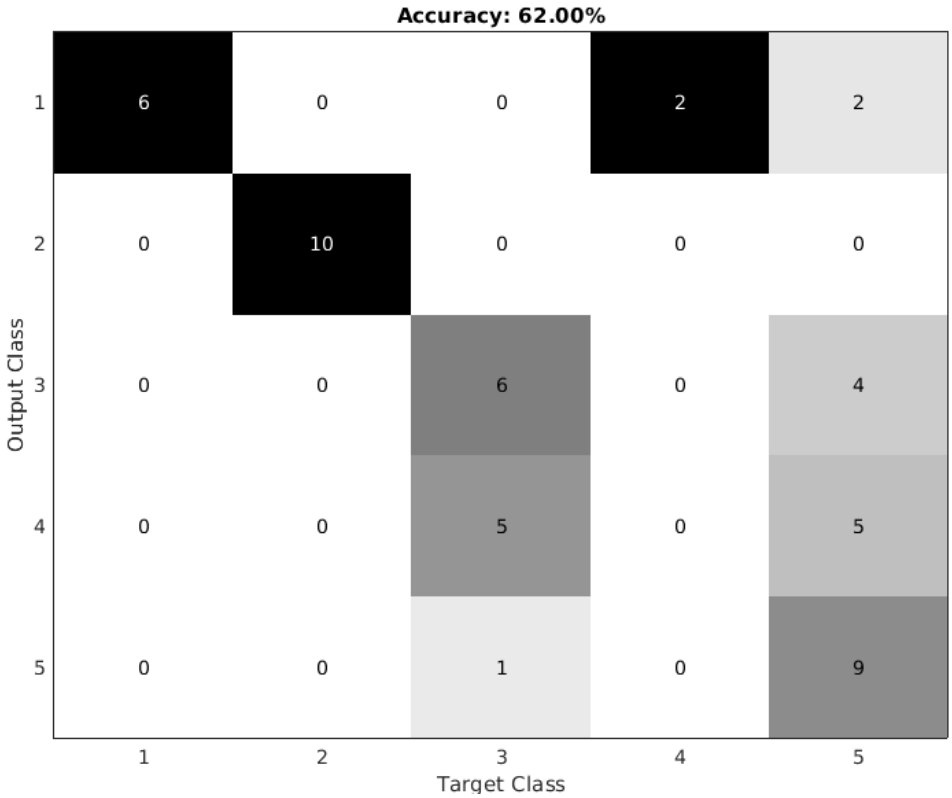


- CP-ALS dla J=10

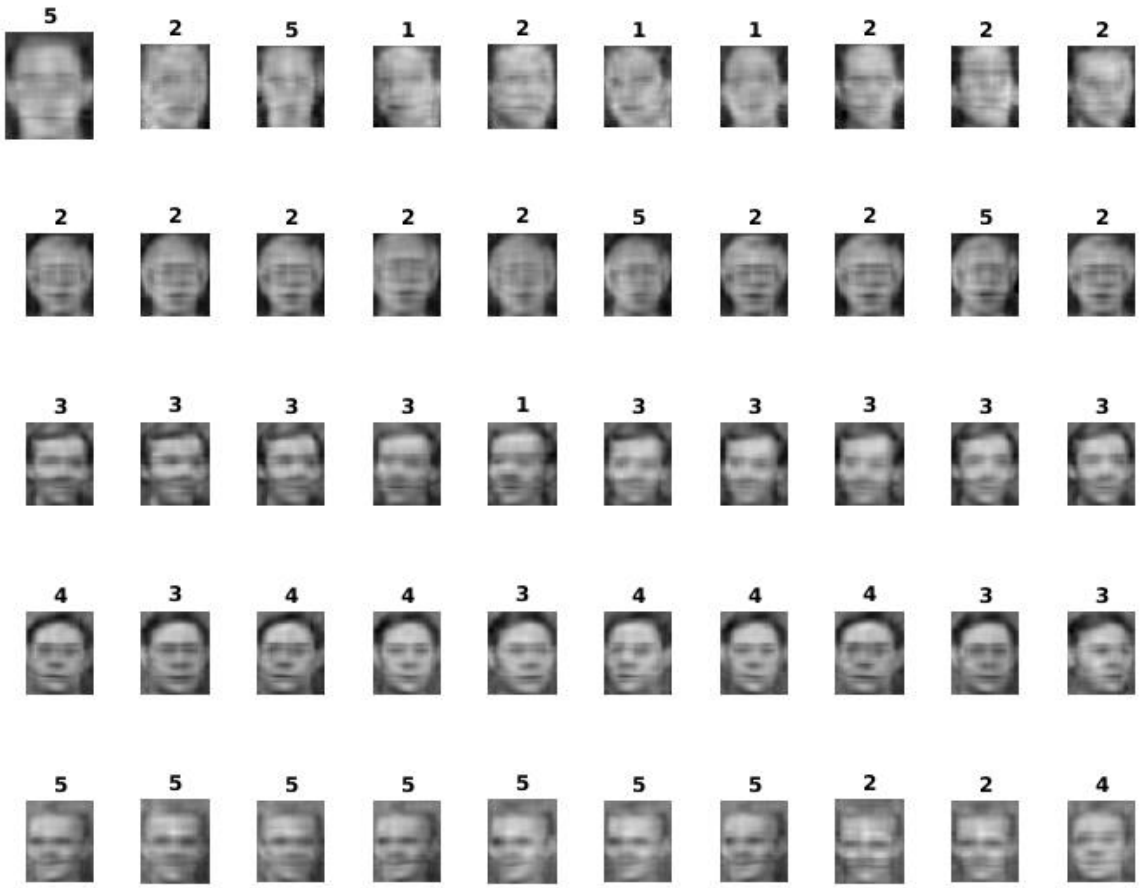


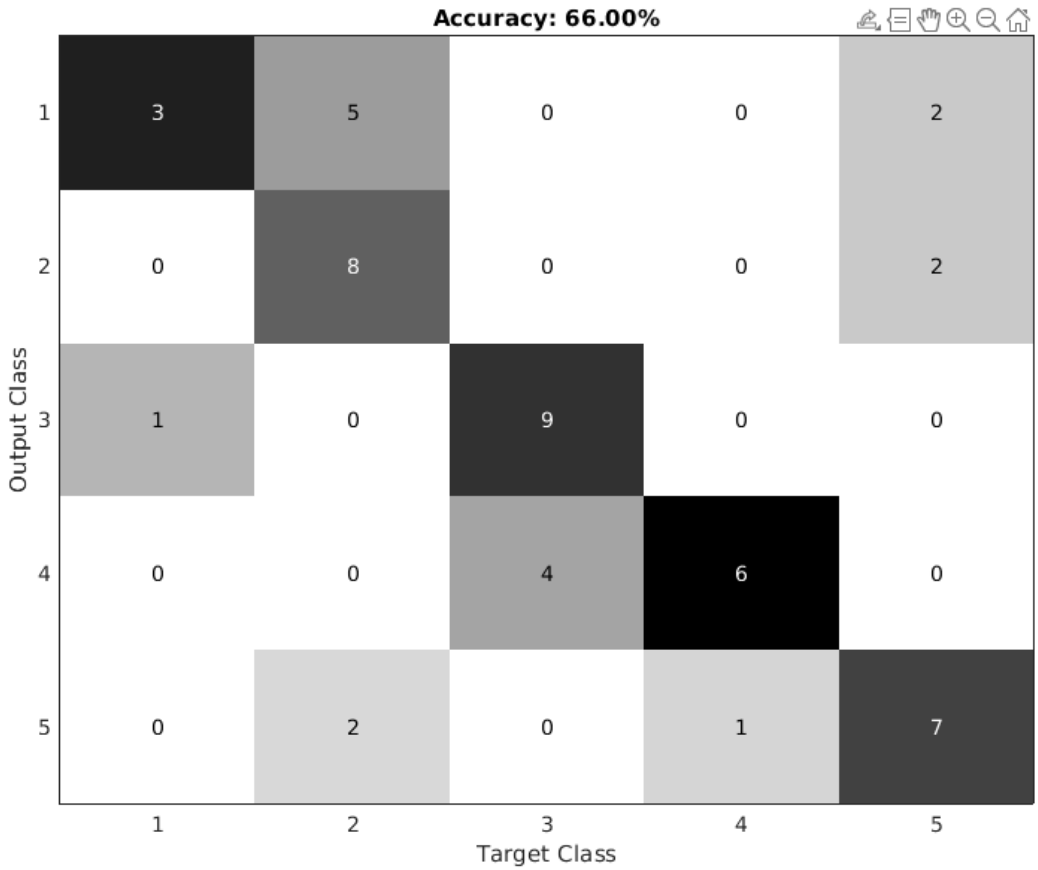
- CP-ALS dla J=20





- CP-ALS dla J=30



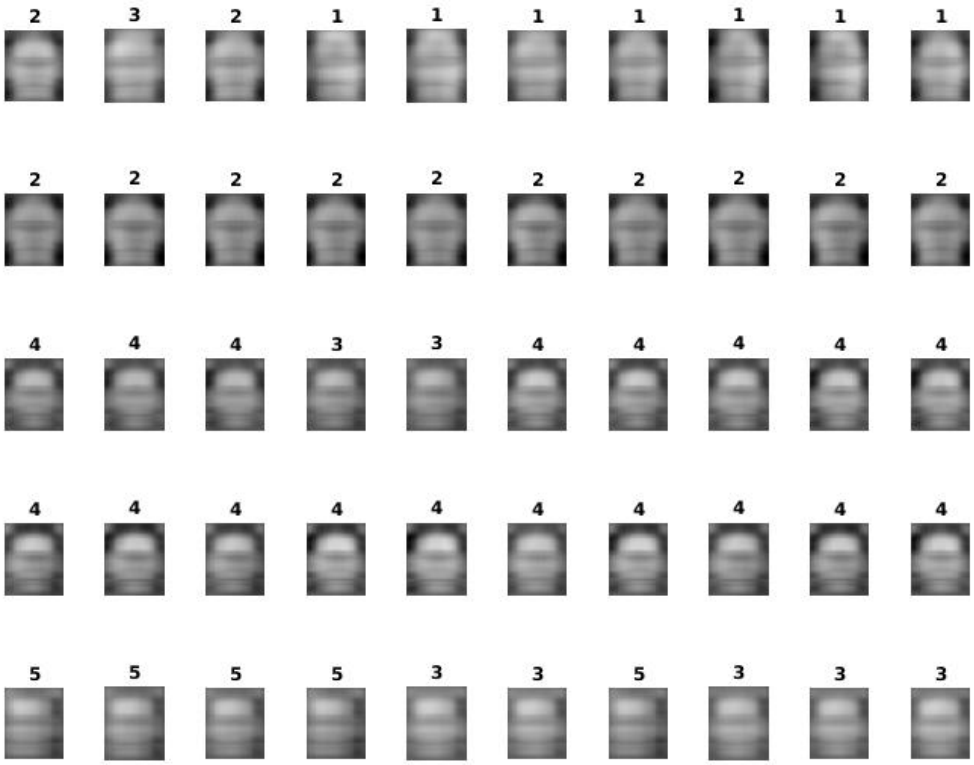


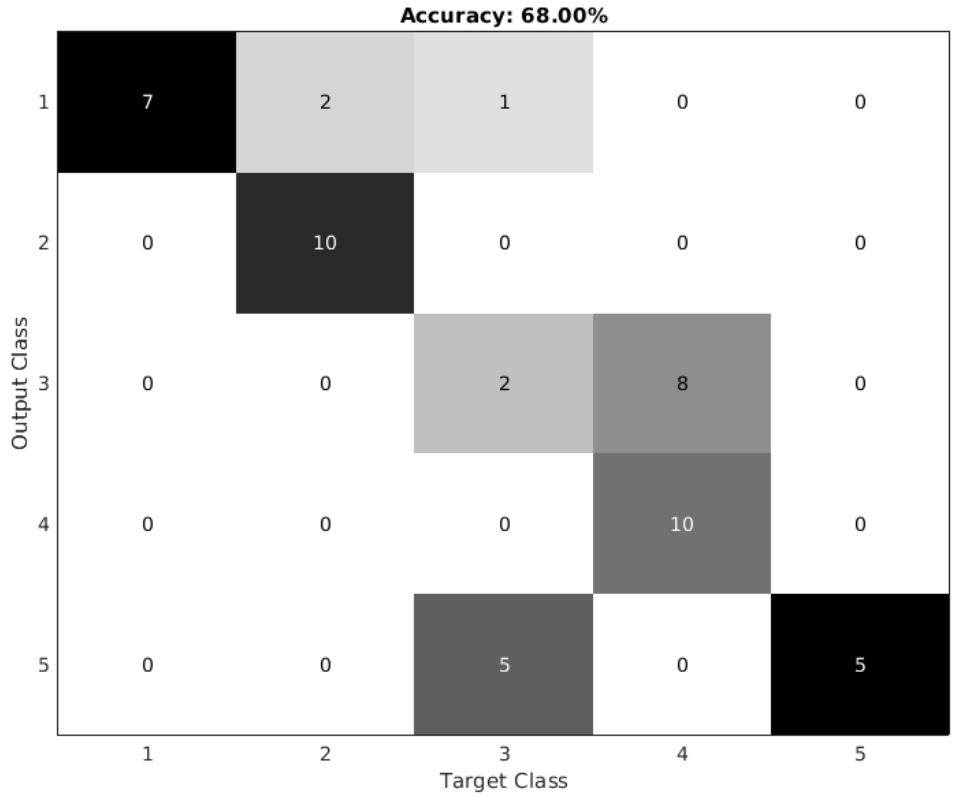
Badania dla HOSVD

Przy **dekompozycji HOSVD** do klasyfikacji wykorzystano faktor odpowiadający trzeciemu modowi - $\boldsymbol{U}^{(3)}$.

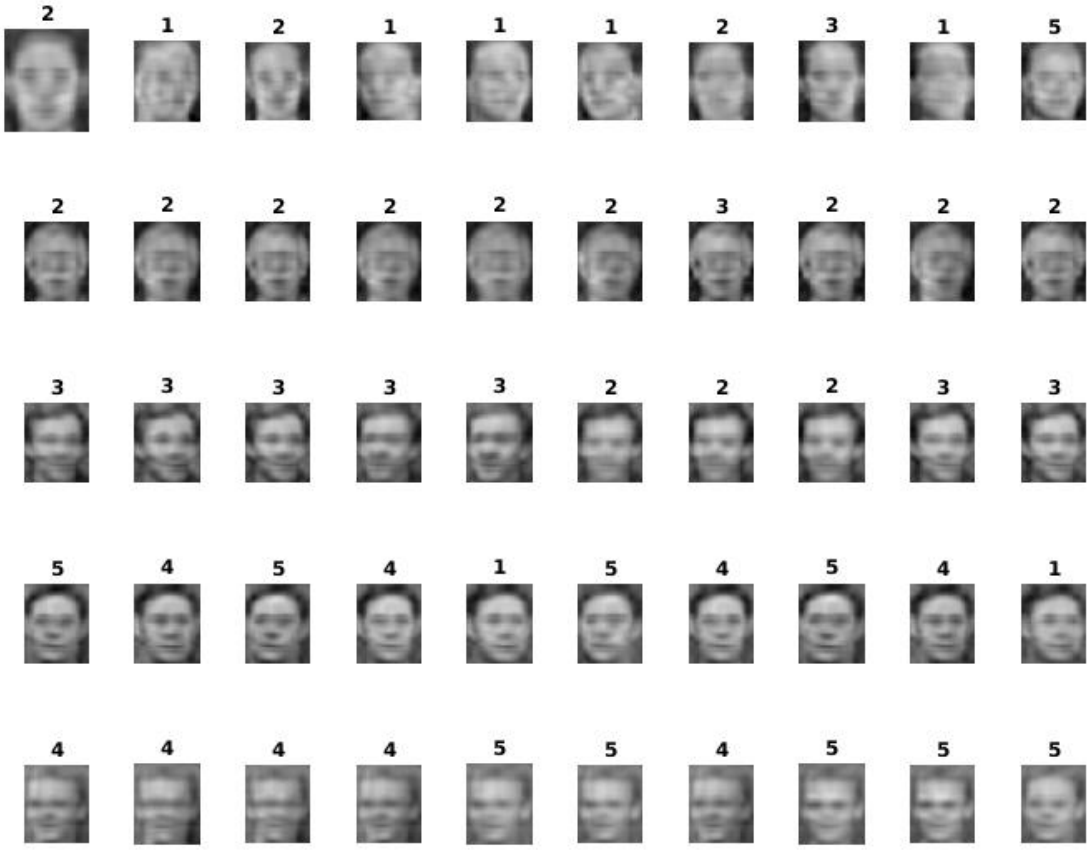
Wyniki dla poszczególnych wartości J (liczby składowych głównych znajdują się poniżej):

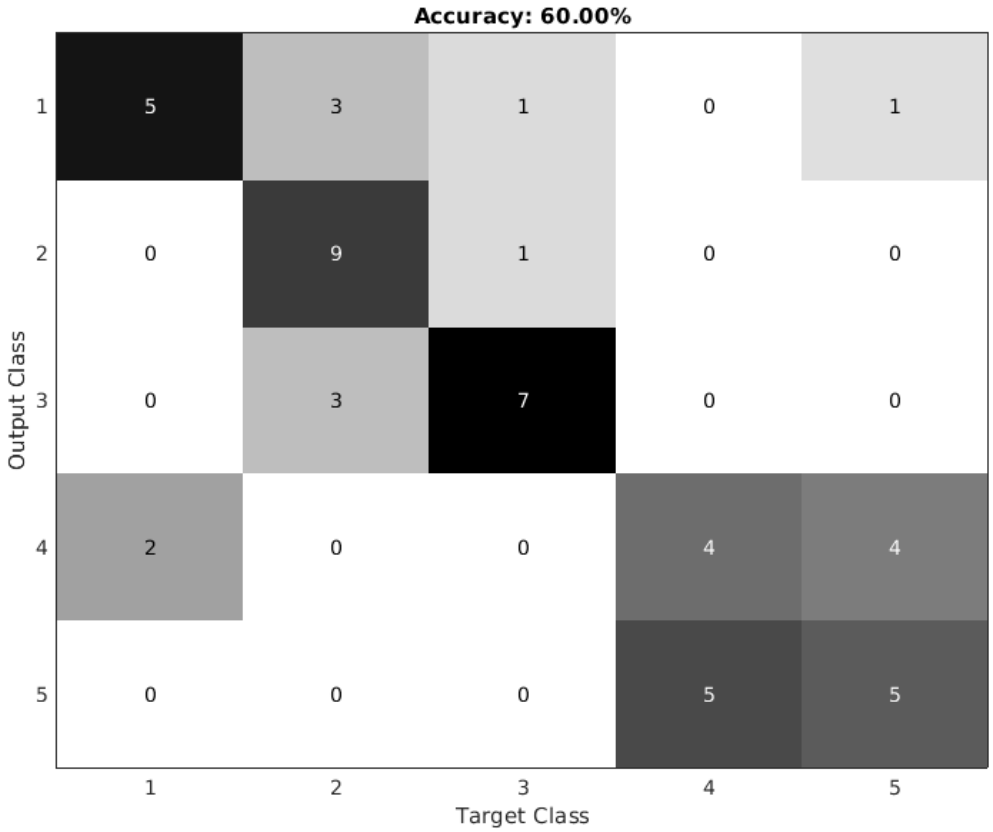
- HOSVD dla J=4



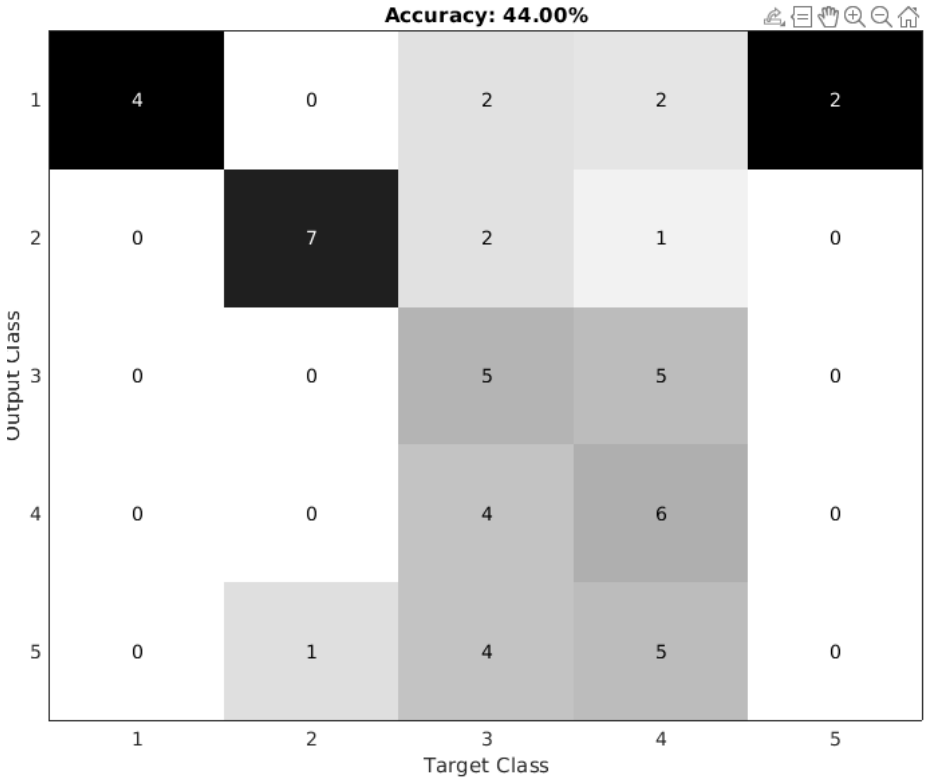


- HOSVD dla J=10

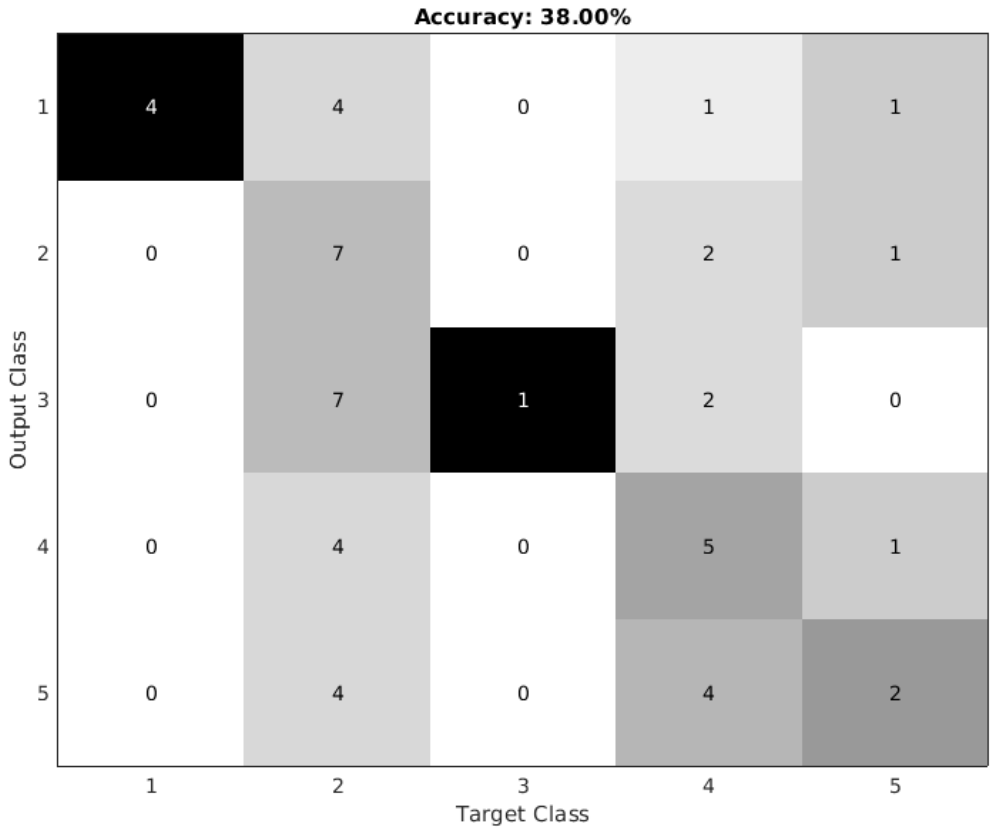




- HOSVD dla J=20



- HOSVD dla J=30



Zestawienie wyników dla poszczególnych metod

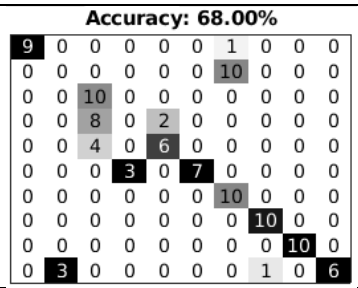
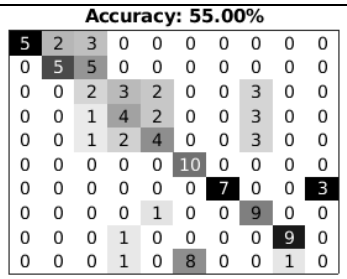
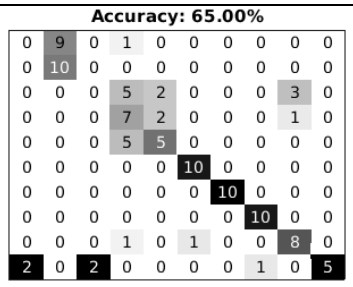
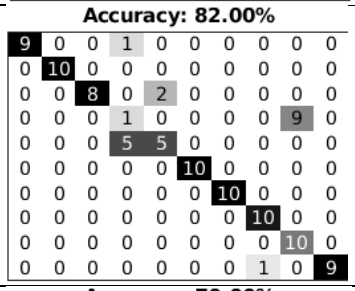
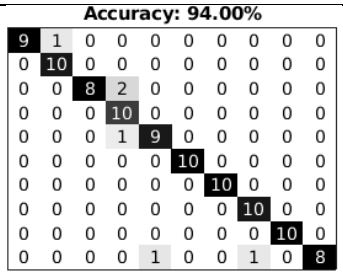
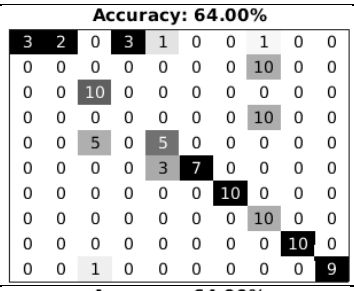
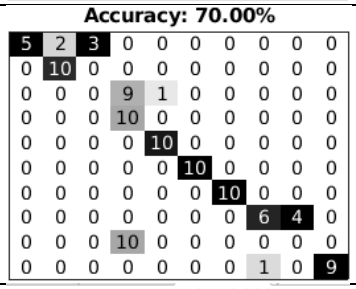
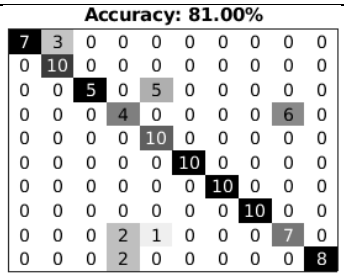
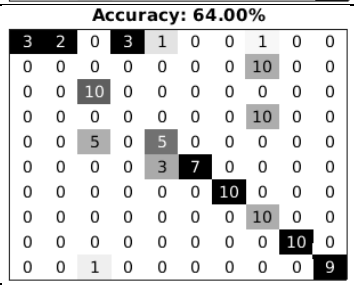
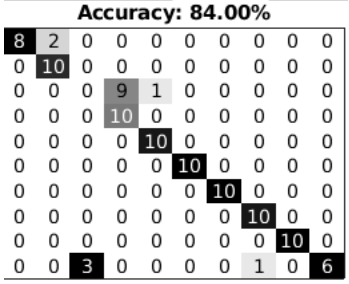
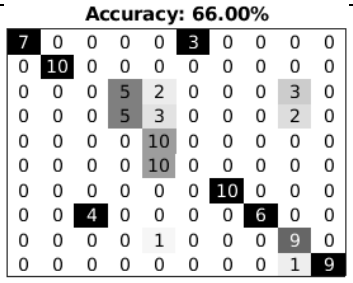
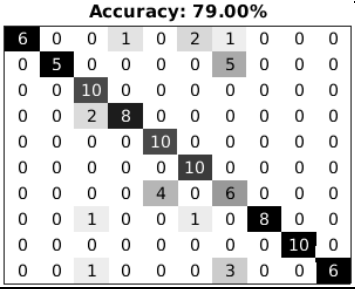
Wyniki zestawiono w poniższej tabeli. W przypadku PCA wartość parametru J oznacza liczbę twarzy własnych. Natomiast w metodach dekompozycji tensorów jest to stopień faktoryzacji.

Metoda \ Dokładność dla	J=4	J=10	J=20	J=30
PCA	86%	86%	50%	72%
NTF – CP-ALS	64%	68%	62%	66%
NTF - HOSVD	68%	60%	44%	38%

Co ciekawe, zwiększenie stopnia faktoryzacji pogarszało dokładność klasteryzacji zdjęć. Stąd wniosek, że obraz pozbawienie zbędnych szczegółów (bo im niższy stopnień faktoryzacji tym mniej „szczegółowy" obraz) poprawia działanie klasteryzacji k-średnich.

Badanie poprawności klasteryzacji aproksymowanego zbioru treningowego 100 obserwacji (10 grup)

Wyniki badania z identycznymi parametrami, ale **dla 10 grup (10 osób po 10 zdjęć)** są w tabeli:

J\Metoda	PCA	CP-ALS	HOSVD
J=4			
J=10			
J=20			
J=30			

Projekcja obrazów z tensora testowego i klasyfikacja kNN

Zdefiniowano funkcję ***zadanie3_knn***, która jako argument przyjmuje liczbę osób (grup) do wczytania z pliku, oraz:

- dzieli dane na tensor treningowy i tensor testujący (5x2cv)
- dla wartości J=4, 10, 20, 30:
    - przeprowadza dekompozycję obserwacji tensora,
    - przeprowadza projekcję obrazów z tensora testowego na przestrzeń cech generowaną faktorami otrzymanymi z tensora treningowego
    - trenuje model na faktorze z tensora treningowego
    - mierzy dokładność klasyfikacji dla danych testowych.

Tutaj ograniczono się do metody HOSVD.

Tabela: Dokładność kNN dla różnych stopni faktoryzacji

Liczba grup (liczba zdjęć)	**J = 4**	**J = 10**	**J = 20**	**J = 30**
5 (50)	100%	100%	70%	70%

```
function zadanie3_knn(persons_count)
    %ZADANIE3_KNN dokonuje dekompozycji na zbiorze treningowym i testuje
    %jakość klasfikacji kNN (dokładność)
    % param persons_count - liczba osób do wczytania z pliku

    if(persons_count > 40)
        err("Maxiumum persons count is 40");
    end

    % Stałe
    K=5; % Parametr K-krotnej walidacji przyżowej walidacji
    J_values = [4, 10, 20, 30];
    images_in_row = 10;

    % Zdjęcia wczytujemy jako tablicę 3D wraz z ich poprawnymi grupami
    [pictures, labels] = load_att_pictures('att_faces', persons_count * images_in_row);

    % Wektor kolejnych rozmiarów tensora obrazów
    I = size(pictures);
    pictures_indexes = 1:I(3);

    % 5-fold-cv
    cv_partitions = cvpartition(pictures_indexes, 'KFold', K);

    %%for partition_index = 1:cv_partitions.NumTestSets
    partition_index = 1; % TODO
    training_mask = cv_partitions.training(partition_index);
    test_mask = cv_partitions.test(partition_index);
    trainging_indexes = pictures_indexes(training_mask);
    test_indexes = pictures_indexes(test_mask);

    % zbiór trenujący i jego prawidłowe grupowanie
    Y_r = tensor(pictures(:, :, trainging_indexes)); % tensor obserwacji treningowych
    Y_t = tensor(pictures(:, :, test_indexes));
    labels_r = labels(trainging_indexes);
    labels_t = labels(test_indexes);


    number_of_J_values = size(J_values, 2);

    % pomiar poprawności klastrowania dla każdego J
    acc_measure_coeffitients = zeros(number_of_J_values, 1);

   for i=1:number_of_J_values
        J = J_values(i);
           [U_r, G_r] = HOSVD(Y_r, [J J J], @unfold3); % faktory estymowane HOSVD
           G_r3 = unfold3(double(G_r), 3);
           Ur2_t_Ur1 = kron(U_r{2}, U_r{1});
           G_3 = G_r3 * Ur2_t_Ur1';
           Y_t3 = unfold3(double(Y_t), 3);
           U_t3 = Y_t3 * pinv(G_3);
           model = fitcknn(U_r{3}, labels_r);
           predictions = predict(model, U_t3);
           accuracy = 100 * sum(predictions' == labels_t) / numel(labels_t);
           acc_measure_coeffitients(i) = accuracy;
   end
end
```

Podsumowanie i wnioski

Tensorowe metody redukcji wymiarowości pozwalają znacznie zmniejszyć wolumen przechowywanych zadanych. Szczególnie obrazuje to zadanie 3 – zamiast przechowywać kilkadziesiąt zdjęć (tablic 2-wymiarowych) wystarczy przechowanie 3 tablic 2-wymiarowych. Oszczędność pamięci trwałej jest bardzo duża.

Jakość odtworzonych obserwacji zależy od stopnia faktoryzacji – im większy tym więcej szczegółów można odtworzyć. Jednak np. w klasyfikacji nadmierna szczegółowość powodowała obniżenie dokładności.