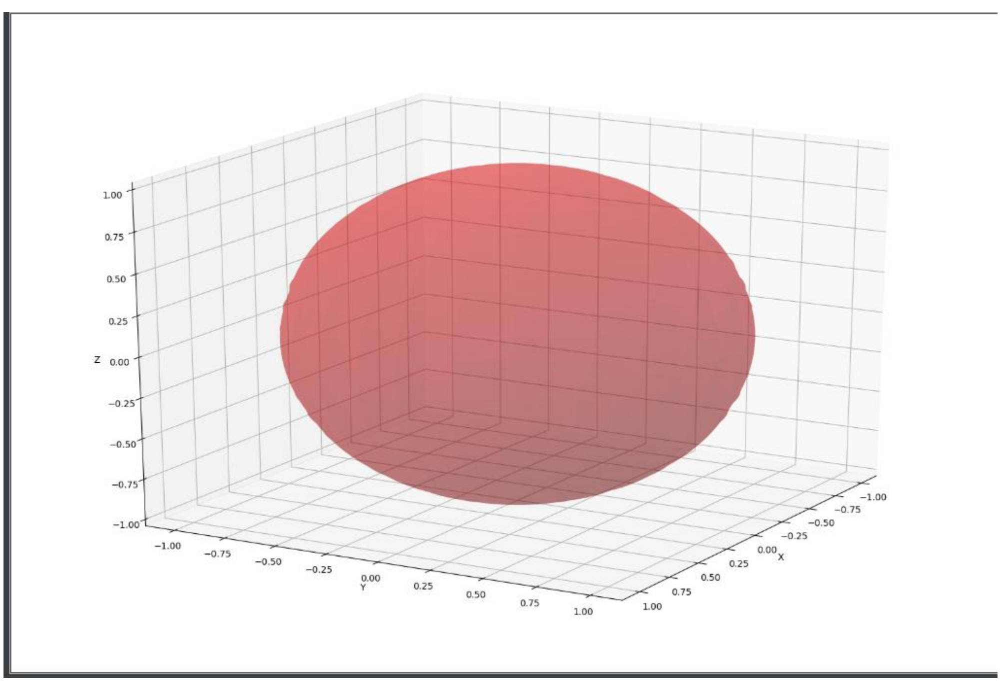
1.Sfera jednostkowa

```
def make_X_Y_Z_sphere():
    s = np.linspace(0, 2 * np.pi, 100)
    t = np.linspace(0, np.pi, 100)

    x = np.outer(np.cos(s), np.sin(t))
    y = np.outer(np.sin(s), np.sin(t))
    z = np.outer(np.ones(np.size(s)), np.cos(t))
    return x, y, z
```

```
def draw_3d(x, y, z, viev_elev=20, view_azim=30):
    fig = plt.figure(figsize=(15, 10))
    ax = fig.add_subplot(111, projection='3d')
    ax.view_init(elev=viev_elev, azim=view_azim)
    ax.plot_surface(x, y, z, rstride=4, cstride=4, color='r', alpha = 0.3)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()
```

```
def draw_sphere():
    x, y, z = make_X_Y_Z_sphere()
    draw_3d(x, y, z)
```



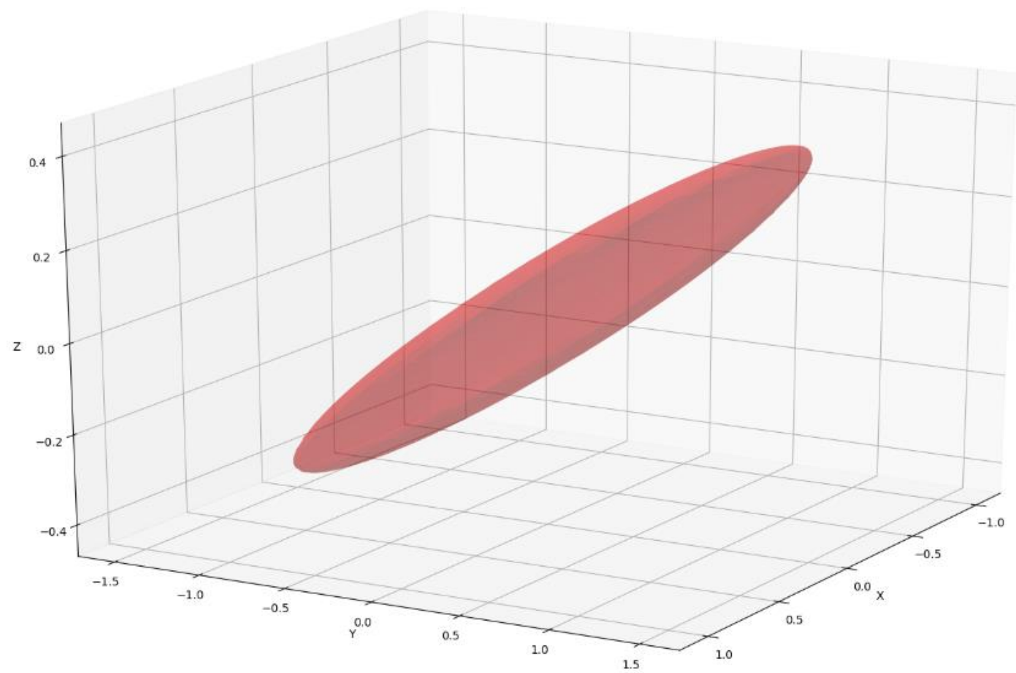
2)EMacierz przekształcające sferę w elipsoidy

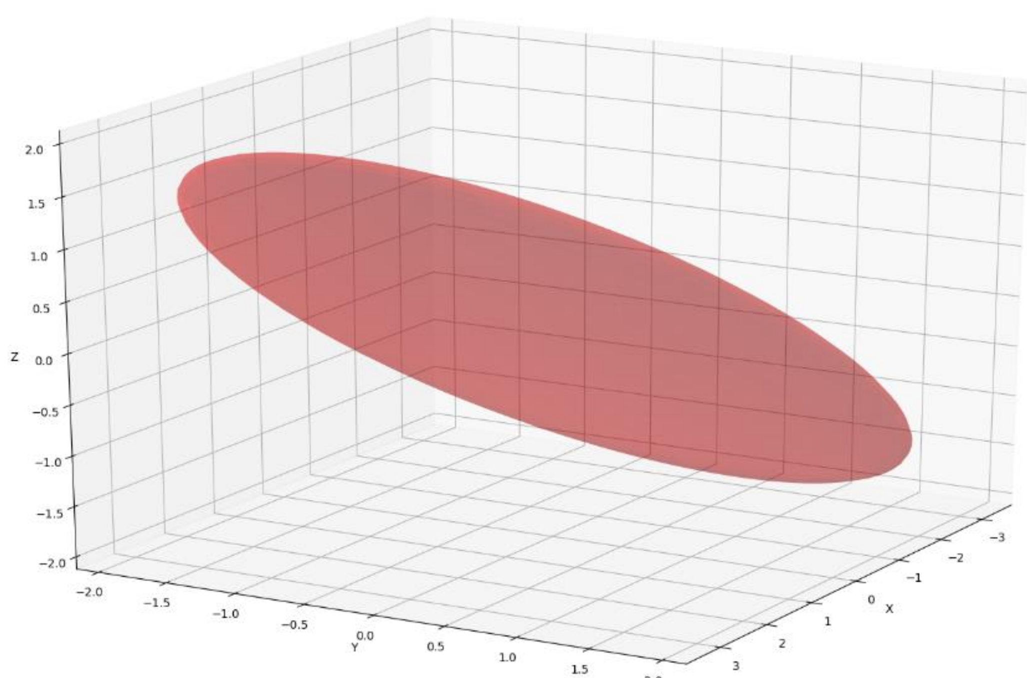
```
def transform_vectors():
    A1 = [[1, 0.02, 0.3], [0.02, 0.45, 1.43], [0.01, 0.03, 0.45]]
    A2 = [[0.5, -3.2, 0.3], [1.4, 1.2, -0.40], [0.7, -1.8, 0.65]]
    A3 = [[-2.3, 0.15, -2], [0.12, -0.45, 1], [1, -0.6, -1.2]]
    return A1, A2, A3
```

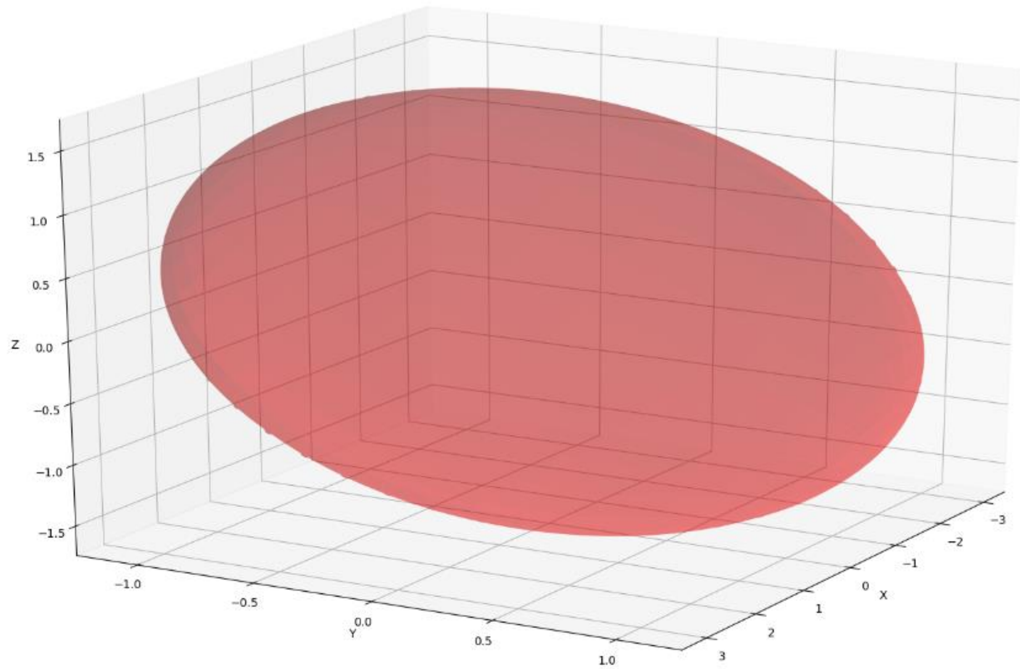
```
def make_X_Y_Z_elipse(A):
    s = np.linspace(0, 2 * np.pi, 100)
    t = np.linspace(0, np.pi, 100)

    x = np.outer(A[0][0] * np.cos(s), np.sin(t))  +\
        np.outer(A[0][1] * np.sin(s), np.sin(t)) +\
        np.outer(A[0][2] * np.ones(np.size(s)), np.cos(t))

    y = np.outer(A[1][0] * np.cos(s), np.sin(t))  + \
        np.outer(A[1][1] * np.sin(s), np.sin(t))  + \
        np.outer(A[1][2] * np.ones(np.size(s)), np.cos(t))

    z = np.outer(A[2][0] * np.cos(s), np.sin(t))  + \
        np.outer(A[2][1] * np.sin(s), np.sin(t)) + \
        np.outer(A[2][2] * np.ones(np.size(s)), np.cos(t))

    return x, y, z
```

```
def draw_elipse(A):
    x, y, z = make_X_Y_Z_elipse(A)
    draw_3d(x, y, z)
```

```
def draw_3_elipse():
    A1, A2, A3 = transform_vectors()
    draw_elipse(A1)
    draw_elipse(A2)
    draw_elipse(A3)
```


Z
0.4
0.2
0.0
−0.2
−0.4
Y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X
−1.0
−0.5
0.0
0.5
1.0


Z
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X
−3
−2
−1
0
1
2
3

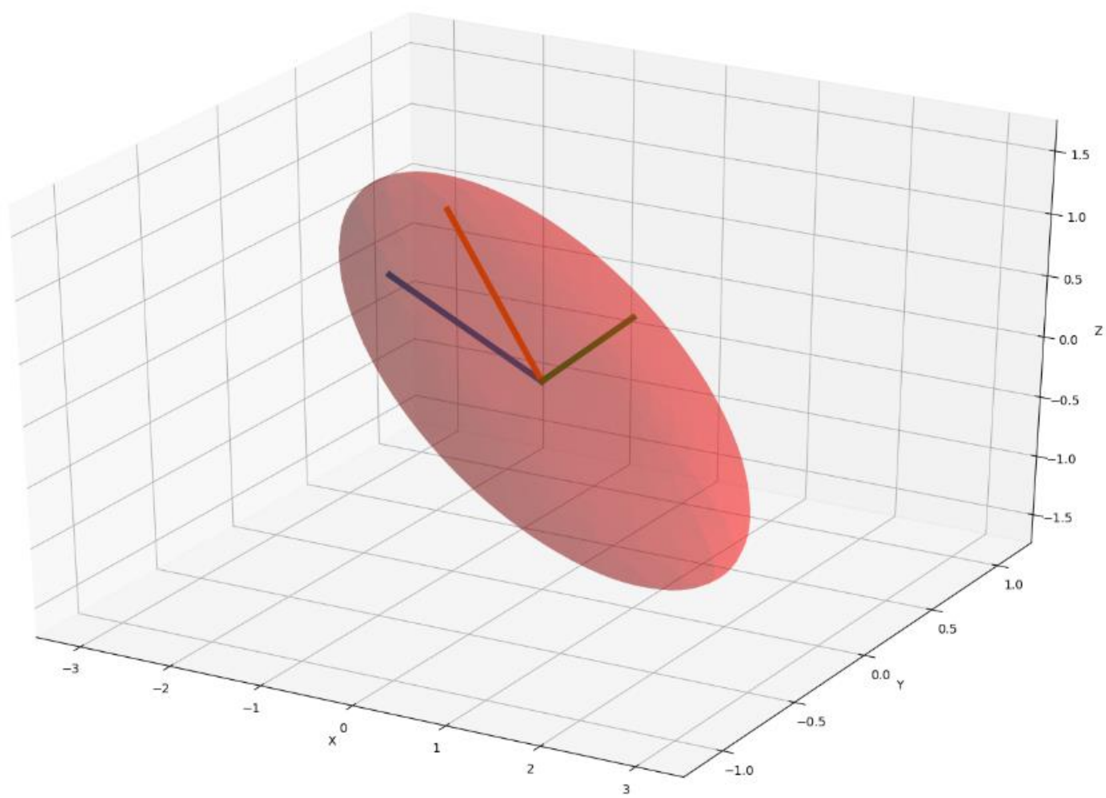


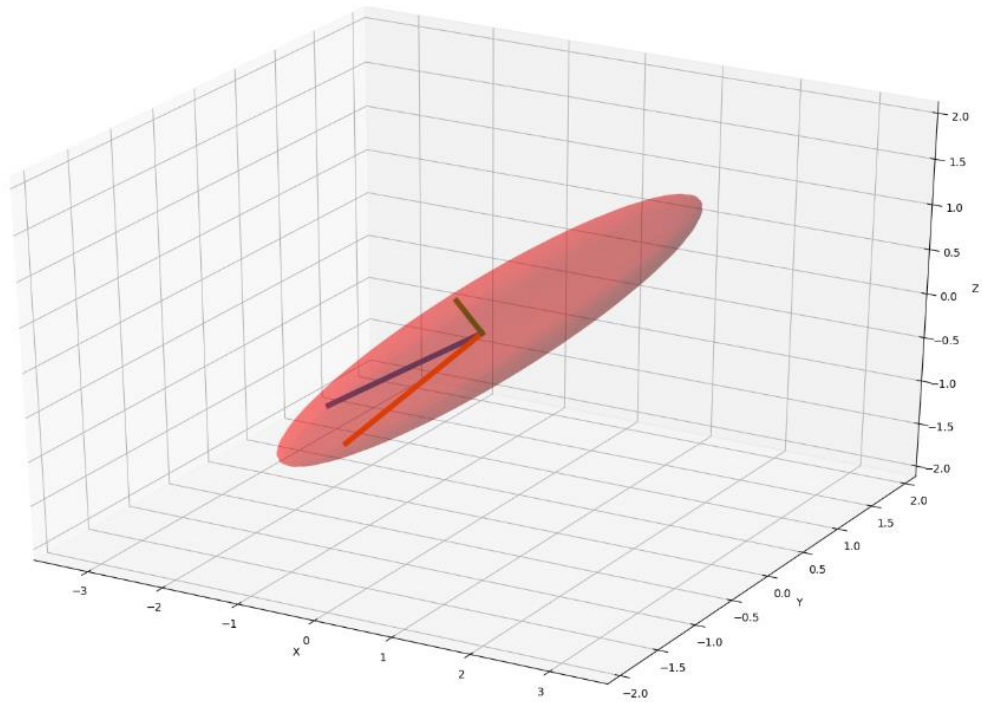
3. Rozkład według wartości osobliwych (SVD) i wizualizacja półosi wyznaczonych za pomocą SVD.

```
def draw_3d_with_singular_values(x, y, z, A):
    fig = plt.figure(figsize=(15, 10))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z, rstride=4, cstride=4, color='r', alpha=0.3)

    #draw singular values
    U, S, VT = np.linalg.svd(A)
    diag_S = np.diag(S)

    for row in diag_S:
        vector = np.dot(U, row)
        ax.plot([0, vector[0]], [0, vector[1]], [0, vector[2]],
linewidth=5)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()
```

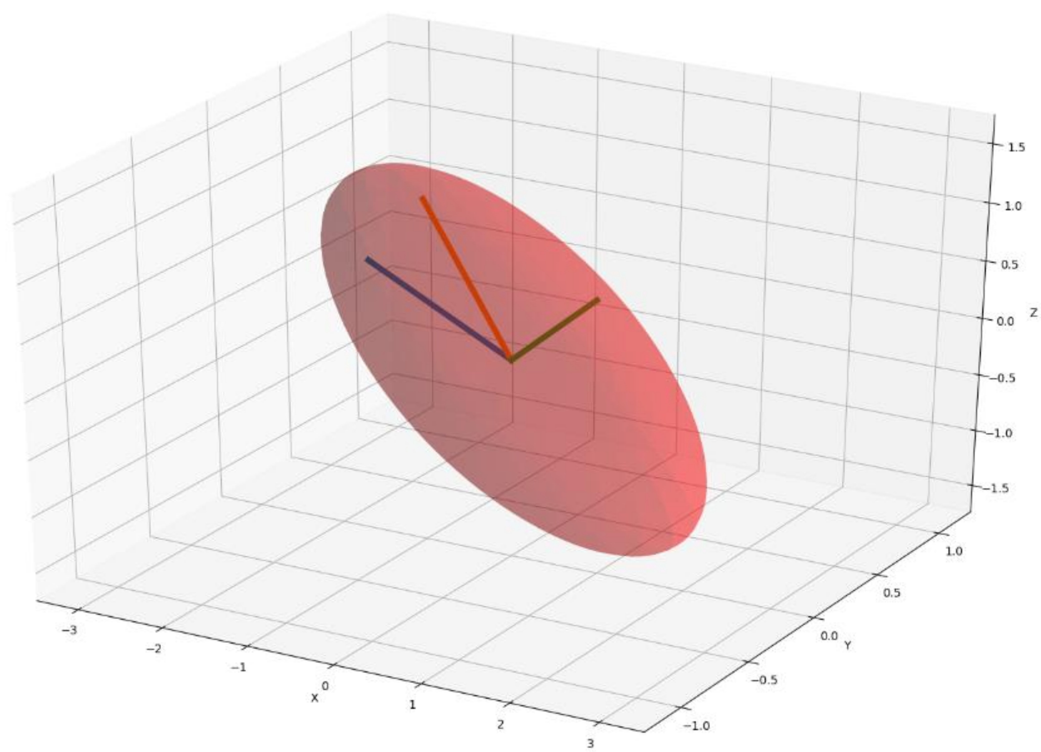
```python
def draw_elipse_with_singular_values(A):
    x, y, z = make_X_Y_Z_elipse(A)
    draw_3d_with_singular_values(x, y, z, A)
```

```python
def draw_3_elipse_with_singular_values():
    A1, A2, A3 = transform_vectors()
    draw_elipse_with_singular_values(A1)
    draw_elipse_with_singular_values(A2)
    draw_elipse_with_singular_values(A3)
```





X
Y
Z



X
Y
Z

4. Macierz o stosunku wartości osobliwych powyżej 100

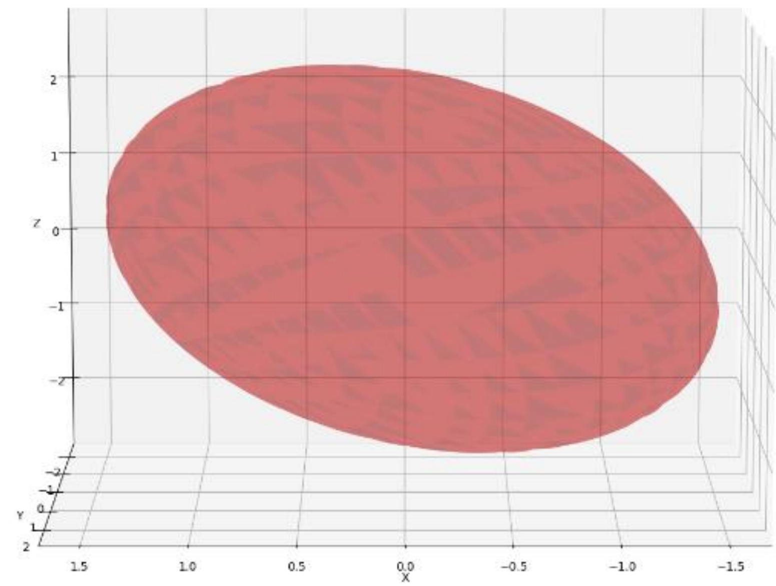
```
def find_100_proportion():
    A = [[1 for _ in range(3)] for _ in range(3)]
    S = [1] * 3
    while S[0] < 100 * S[2]:
        for i in range(3):
            for j in range(3):
                A[i][j] = random.uniform(-3, 3)

        U, S, VT = np.linalg.svd(A)
    diag_S = np.diag(S)
    print(A)
    print(diag_S)
    return A


def draw_100_proportion_elipse():
    A = find_100_proportion()
    x, y, z = make_X_Y_Z_elipse(A)
    draw_3d(x, y, z, 10, 90)
    draw_3d(x, y, z, 20, 40)
    draw_3d(x, y, z, 50, 80)
    draw_3d(x, y, z, 60, 40)
    draw_3d(x, y, z, 100, 100)
```

Macierz A

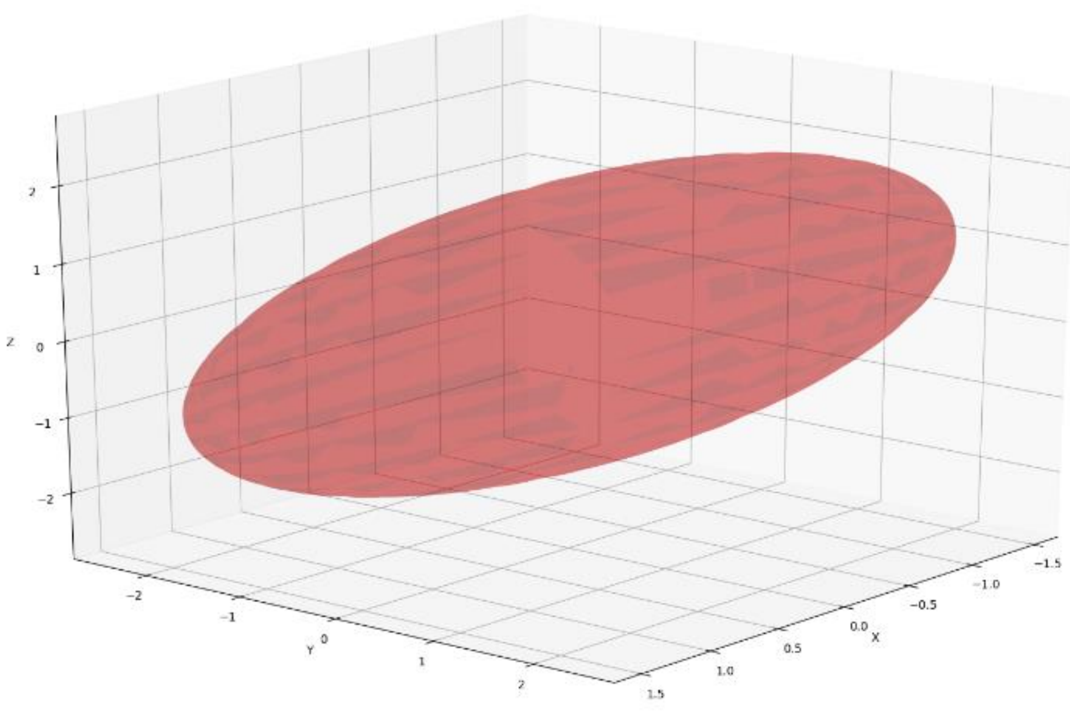
```
[[-0.974844905837573, 0.2727934582052729, -1.0738043784554812],
[-0.524764316083961, -1.1814075194500004, 2.063985942001384],
[-2.174448436273784, -1.2456288211267965, 1.1755957708275915]]
```

Macierz diagonalna

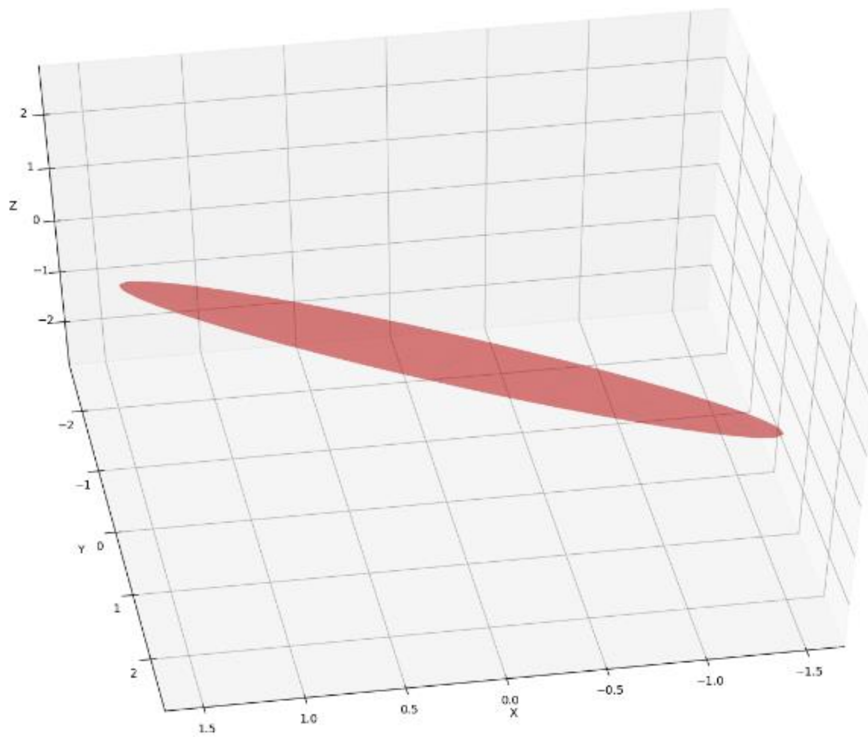
```
[[3.46364829 0.         0.        ]
 [0.         1.94252624 0.        ]
 [0.         0.         0.02255887]]
```



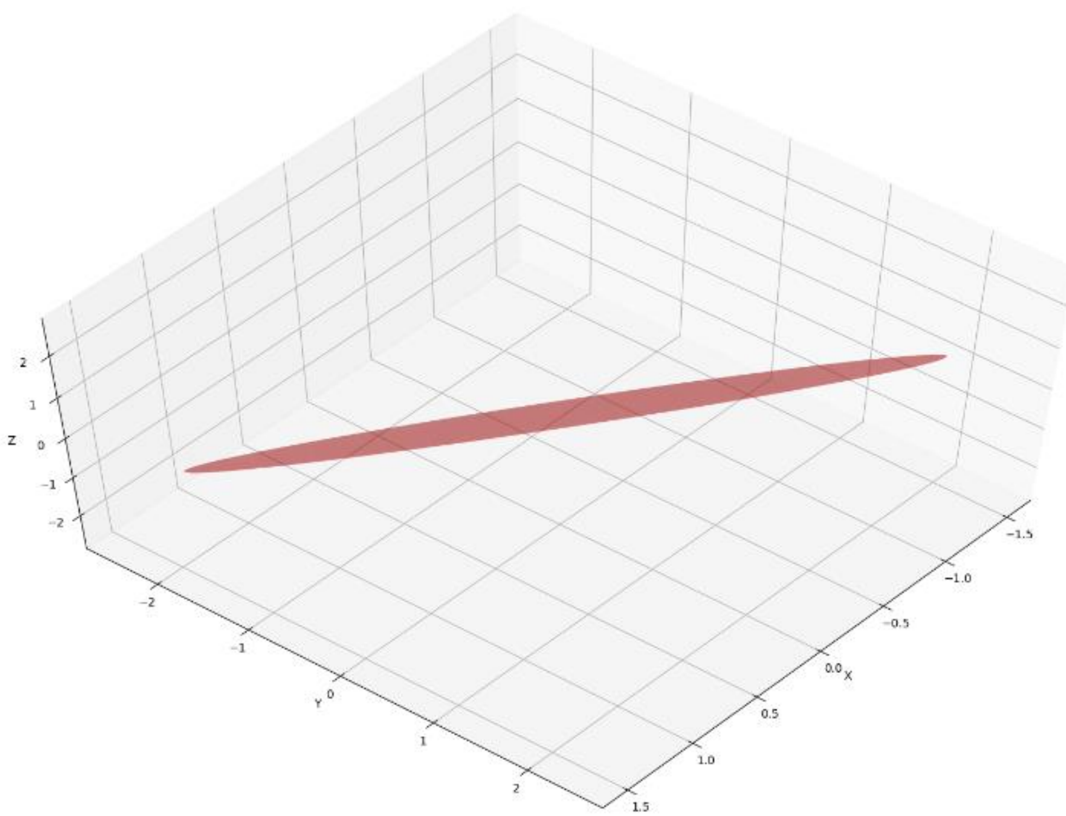
Z
Y
X



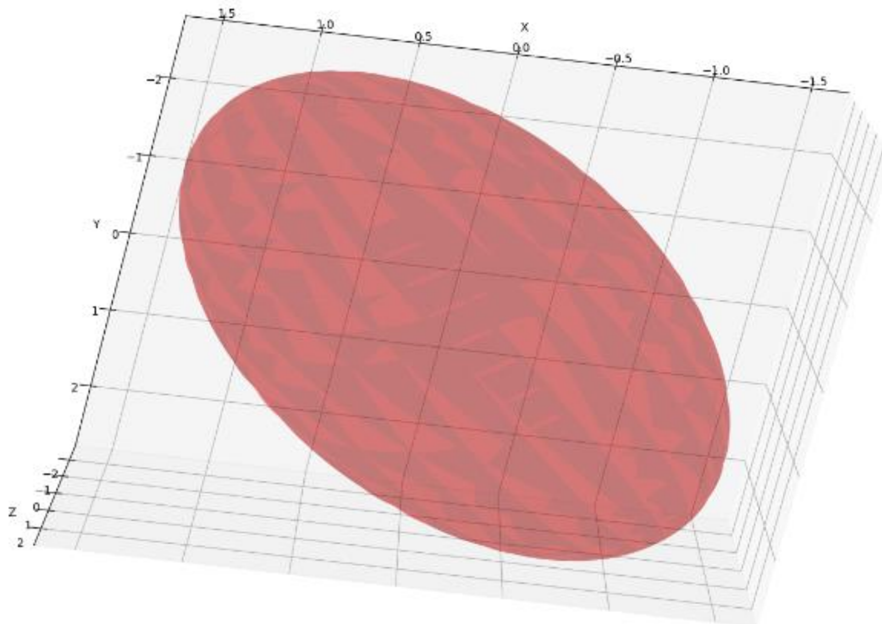
Z
Y
X



Z
2
1
0
−1
−2
Y
−2
−1
0
1
2
X
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

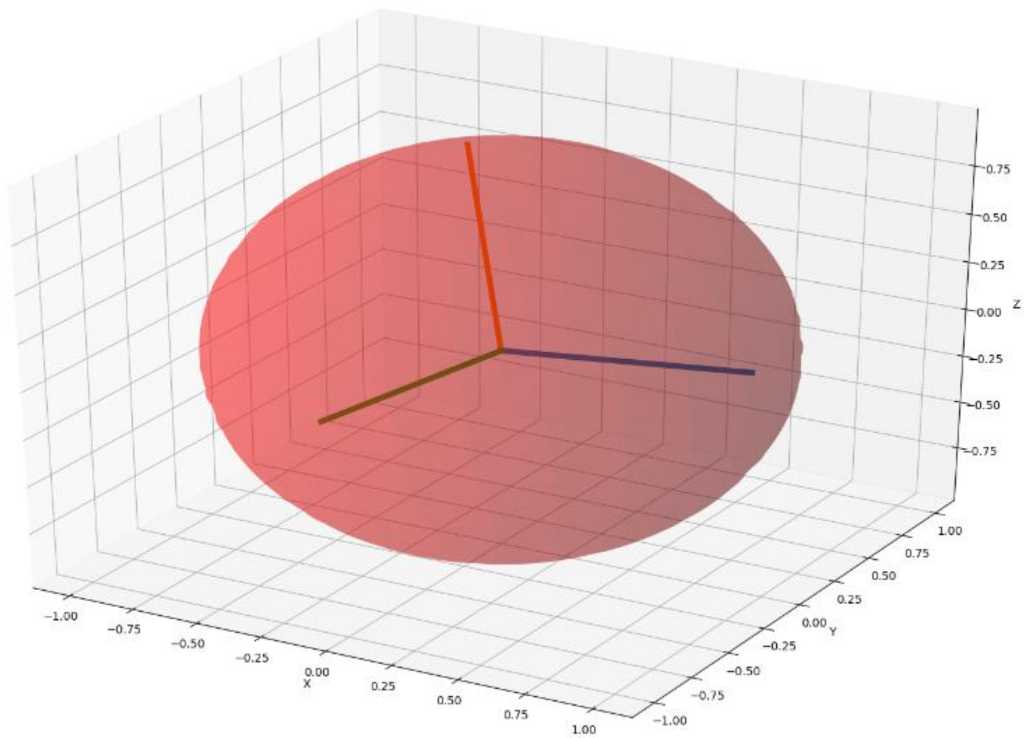


Z
2
1
0
−1
−2
Y
−2
−1
0
1
2
X
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

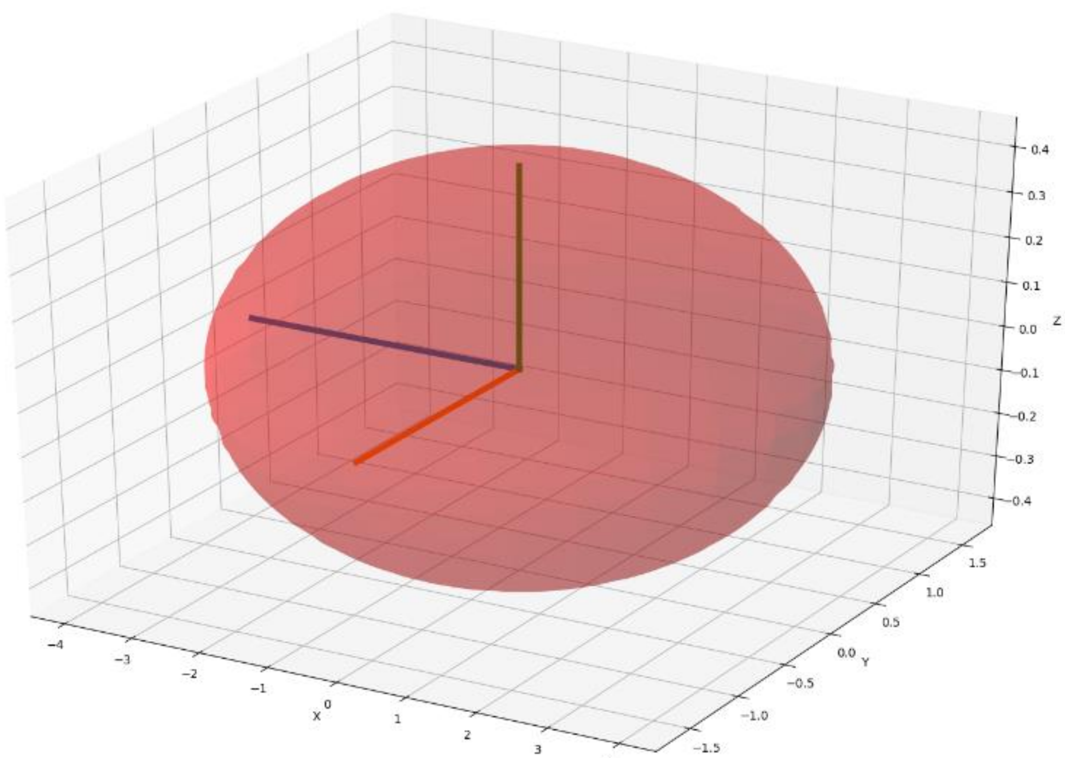


5) Wizualizacje poszczególnych transformacji

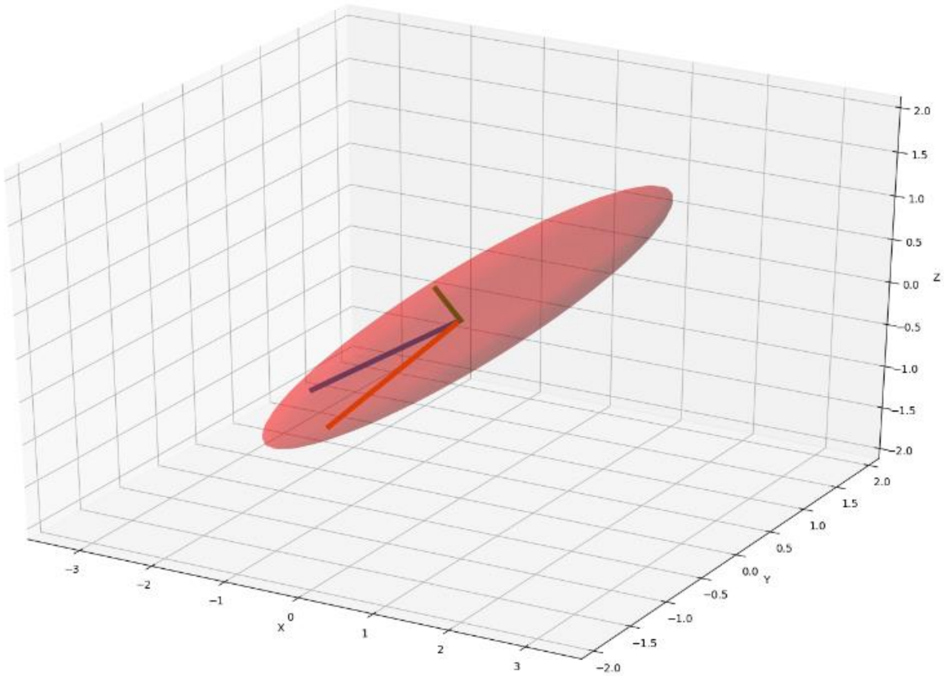
```
def show_transformation():
    A1, A2, A3 =transform_vectors()
    U, S, VT = np.linalg.svd(A2)
    diag_S = np.diag(S)
    draw_elipse_with_singular_values(VT)
    draw_elipse_with_singular_values(diag_S @ VT)
    draw_elipse_with_singular_values(U @ diag_S @ VT)
```



0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Z
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X
Y



0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
Z
−4
−3
−2
−1
0
1
2
3
4
X
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y

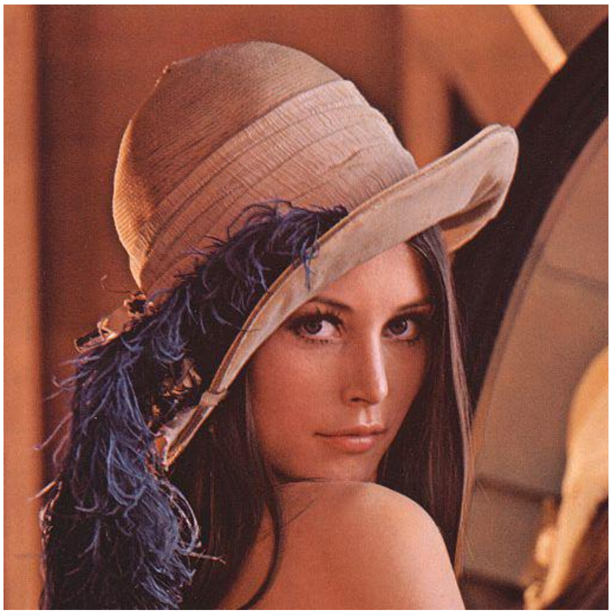


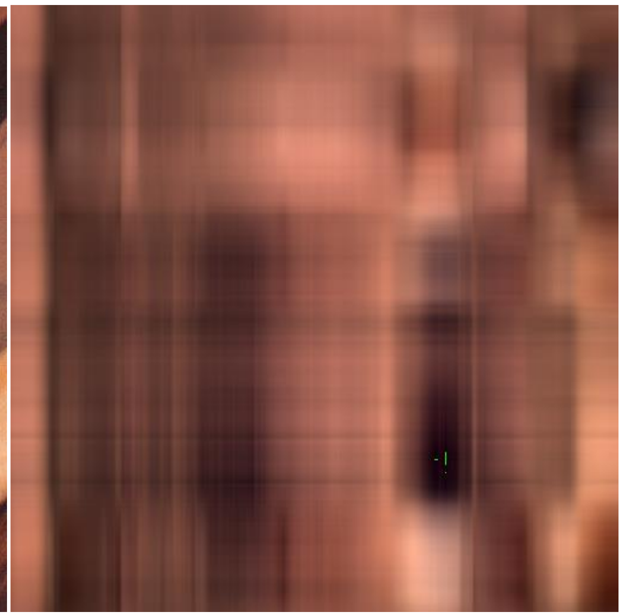
6)Kompresja obrazu

```
def lena_compression(k):
    path, extension = os.path.splitext("lena.png")
    img = Image.open("lena.png")
    width = img.width
    height = img.height
    img_arr = np.array(img)

    arr1 = img_arr.transpose(0, 2, 1).reshape(height * 3, width)
    arr2 = img_arr.transpose(1, 2, 0).reshape(width * 3, height)

    U, S, A1 = linalg.svd(arr1)
    U, S, A2 = linalg.svd(arr2)

    B1 = A1[:k*3, :]
    B2 = A2[:k*3, :]

    C1 = B1.T.dot(B1)
    C2 = B2.T.dot(B2)

    arr2 = np.tensordot(img_arr, C1, (1, 0))
    arr3 = np.tensordot(arr2, C2, (0, 0))
    arr4 = arr3.transpose(2, 1, 0)

    imgk = Image.fromarray(np.uint8(arr4))
    file = path + str(k) + extension
    imgk.save(file)
```

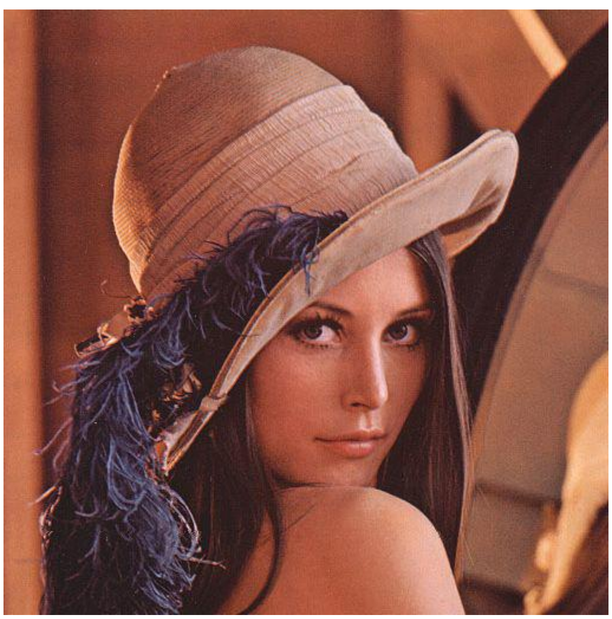
```python
def compress_lena_arr():
    arr = [1, 3, 5, 10, 15, 50, 100, 200]
    for ele in arr:
        lena_compression(ele)
```

k = 1





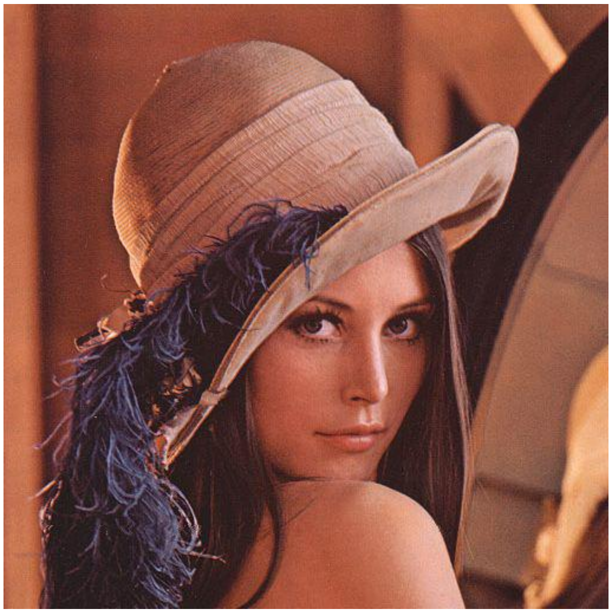
K = 3



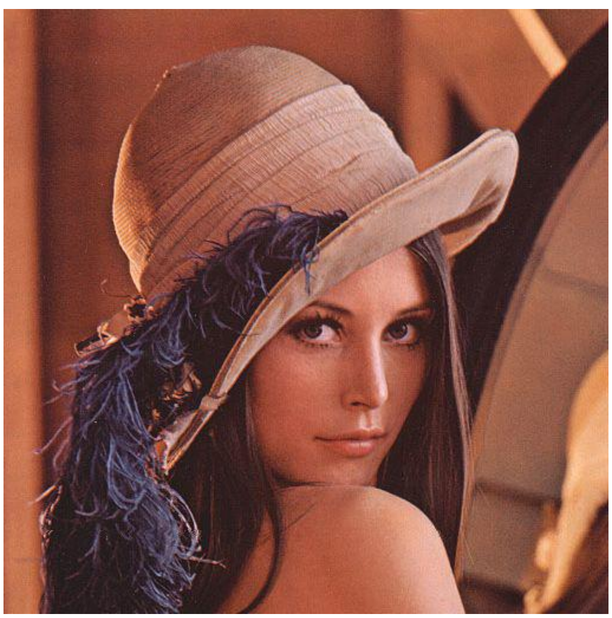
K = 5

K = 10

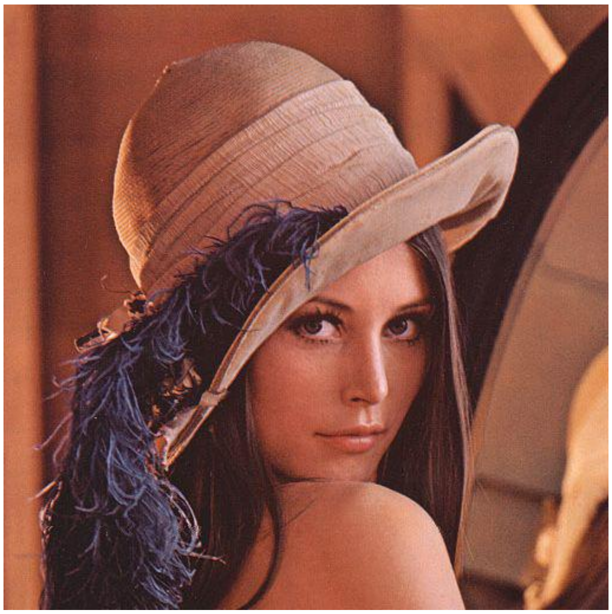
K = 15

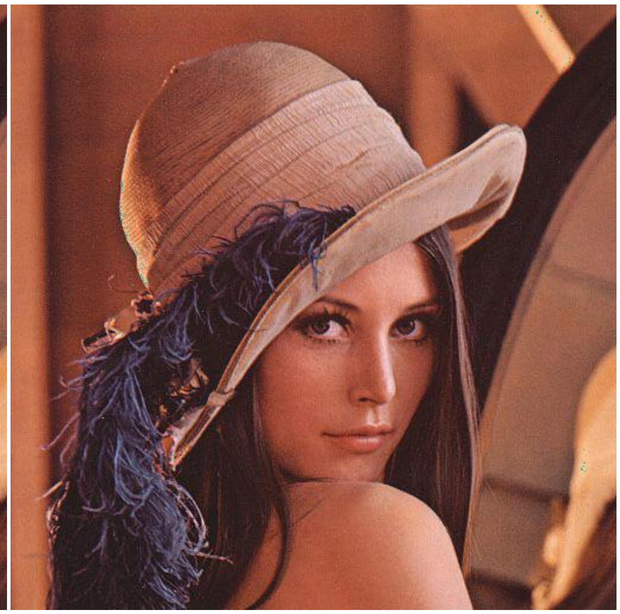


K = 50

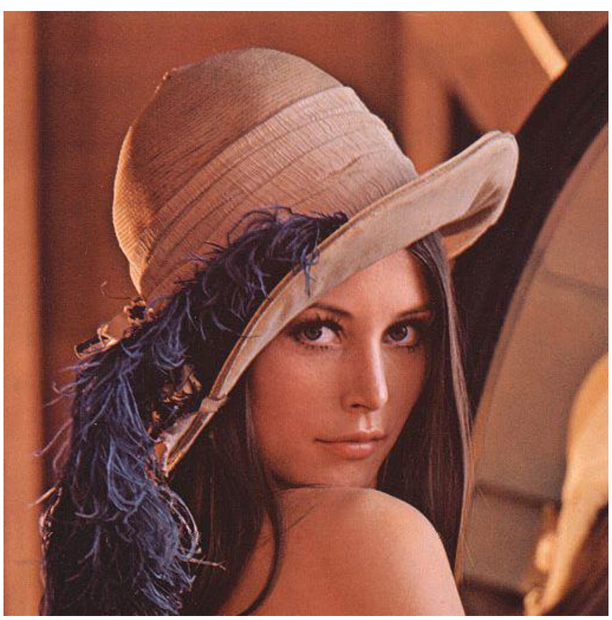


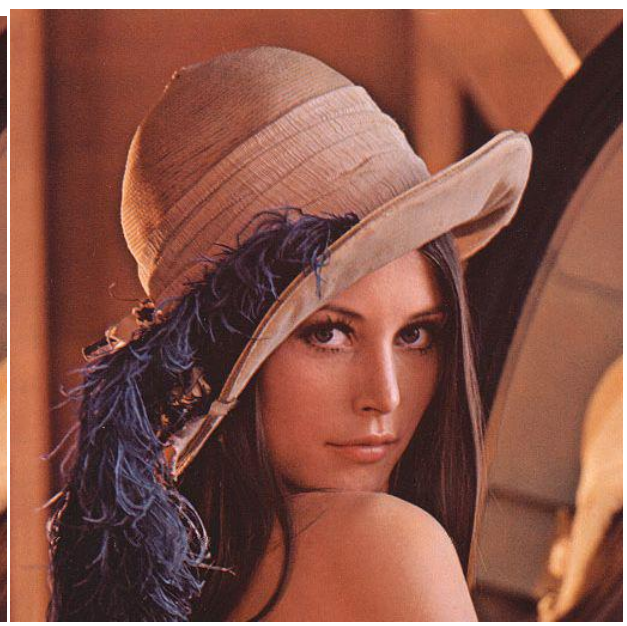
K = 100





K = 200





Wnioski kompresja obrazu przy pomocy SVD jest bardzo opłacalna. Przy k równym około 50, 100 bardzo ciężko jest zauważyć jakiekolwiek różnice pomiędzy obrazem skompresowanym, a obrazem oryginalnym.