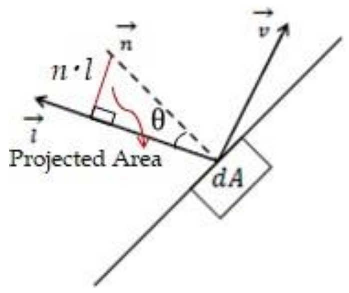
16720 – hw5
mrohitth

Problem 1(a)

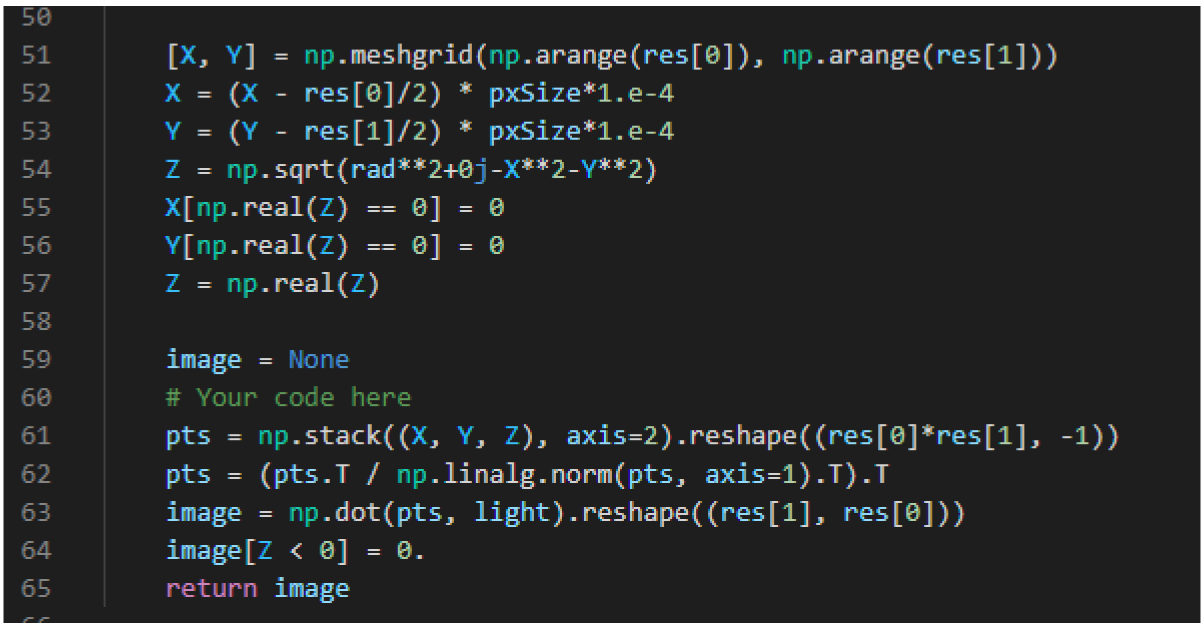
As the object is Lambertian it follows Lambert's cosine law that says that the radiant intensity or luminous intensity observed from an ideal diffusely reflecting surface or ideal diffuse radiator is directly proportional to the cosine of the angle θ between the direction of the incident light and the surface normal.



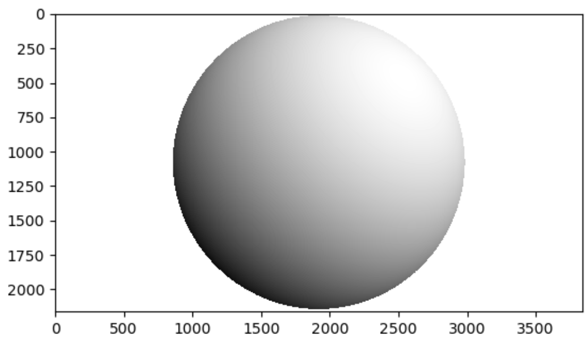
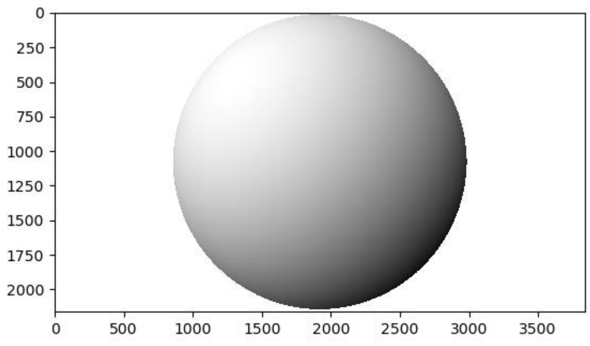
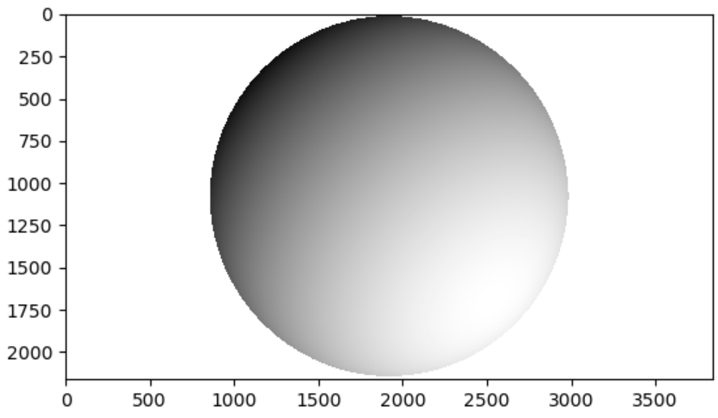
$$dA_{projected} = cos\theta = (n) \cdot (l)$$

When an area element is radiating as a result of being illuminated by an external source, the irradiance (energy or photons/time/area) landing on that area element will be proportional to the cosine of the angle between the illuminating source and the normal. A Lambertian scatterer will then scatter this light according to the same cosine law as a Lambertian emitter. This means that although the radiance of the surface depends on the angle from the normal to the illuminating source, it will not depend on the angle from the normal to the observer.

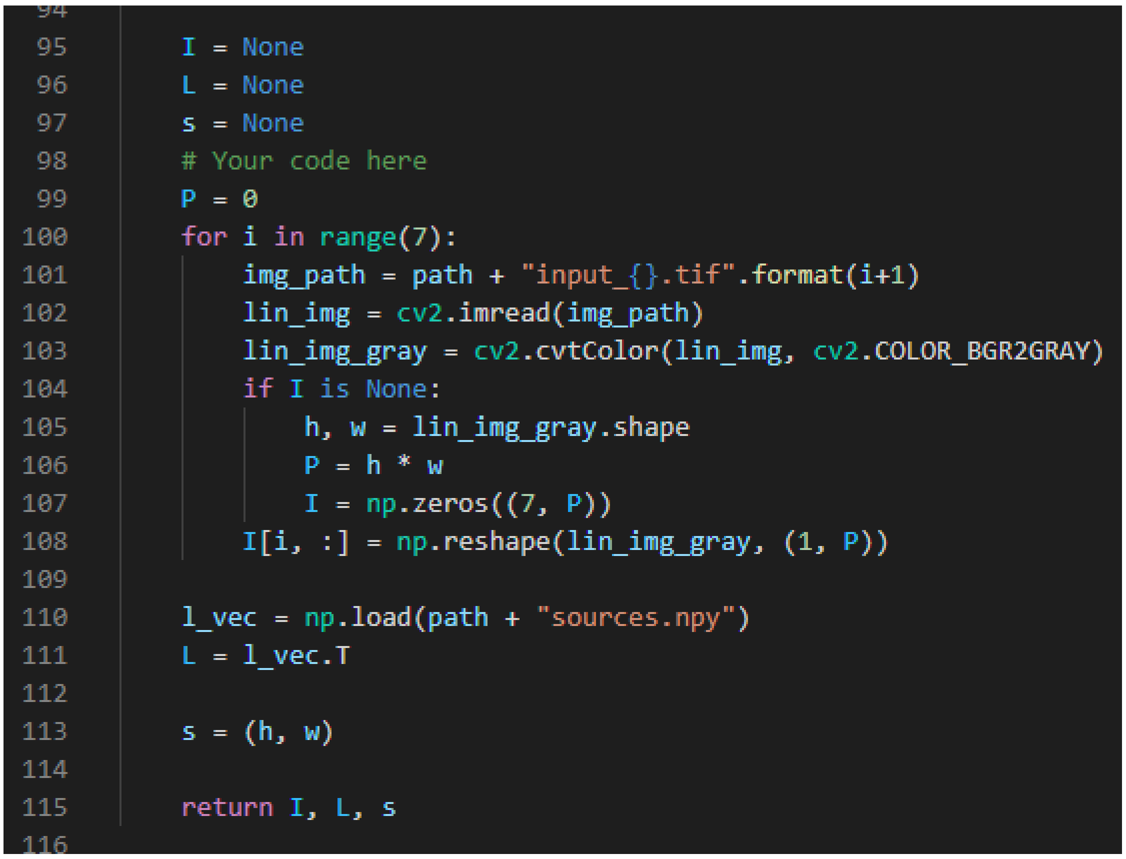
Problem 1(b)



```
    [X, Y] = np.meshgrid(np.arange(res[0]), np.arange(res[1]))
    X = (X - res[0]/2) * pxSize*1.e-4
    Y = (Y - res[1]/2) * pxSize*1.e-4
    Z = np.sqrt(rad**2+0j-X**2-Y**2)
    X[np.real(Z) == 0] = 0
    Y[np.real(Z) == 0] = 0
    Z = np.real(Z)

    image = None
    # Your code here
    pts = np.stack((X, Y, Z), axis=2).reshape((res[0]*res[1], -1))
    pts = (pts.T / np.linalg.norm(pts, axis=1).T).T
    image = np.dot(pts, light).reshape((res[1], res[0]))
    image[Z < 0] = 0.
    return image
```

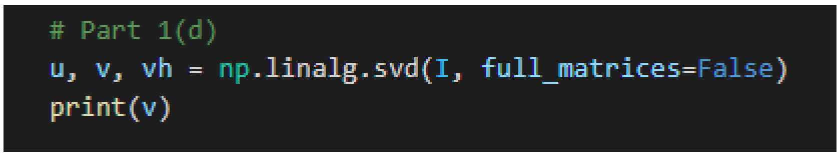
Problem 1(c)



```
    I = None
    L = None
    s = None
    # Your code here
    P = 0
    for i in range(7):
        img_path = path + "input_{}.tif".format(i+1)
        lin_img = cv2.imread(img_path)
        lin_img_gray = cv2.cvtColor(lin_img, cv2.COLOR_BGR2GRAY)
        if I is None:
            h, w = lin_img_gray.shape
            P = h * w
            I = np.zeros((7, P))
        I[i, :] = np.reshape(lin_img_gray, (1, P))

    l_vec = np.load(path + "sources.npy")
    L = l_vec.T

    s = (h, w)

    return I, L, s
```

Problem 1(d)

The rank of matrix I is expected to be 3. After performing singular value decomposition, the rank of the matrix I comes out to be 7. This is because the image capture is not ideal, the image is capturing all inter-reflected lights and more noise from the surroundings due to which we have more independent measurements (7 per pixel) than variables (3 per pixel) leading to a rank of 7 rather than 3. Therefore, we may need more capture images lighted from different directions to help in the reconstruction.



```
# Part 1(d)
u, v, vh = np.linalg.svd(I, full_matrices=False)
print(v)
```

singular values: [66066.78102606 7845.7669138   5478.12153502 1666.10512245 1265.81052874 1000.85704087   815.4020153]

Problem 1(e)

The equation to solve is as follows:

$$I = L^T B$$

$$L^{T^{-1}} I = B$$

Which can be written as $Ax = y$
Where,

$$A = L^{T^{-1}}$$

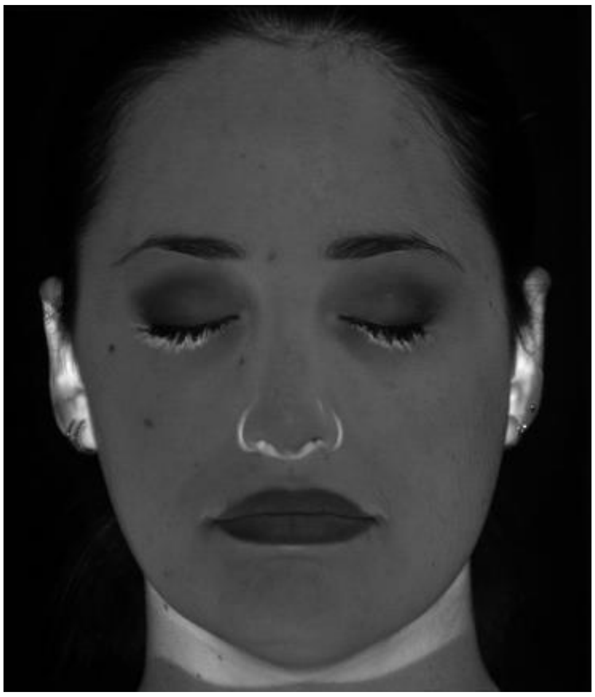
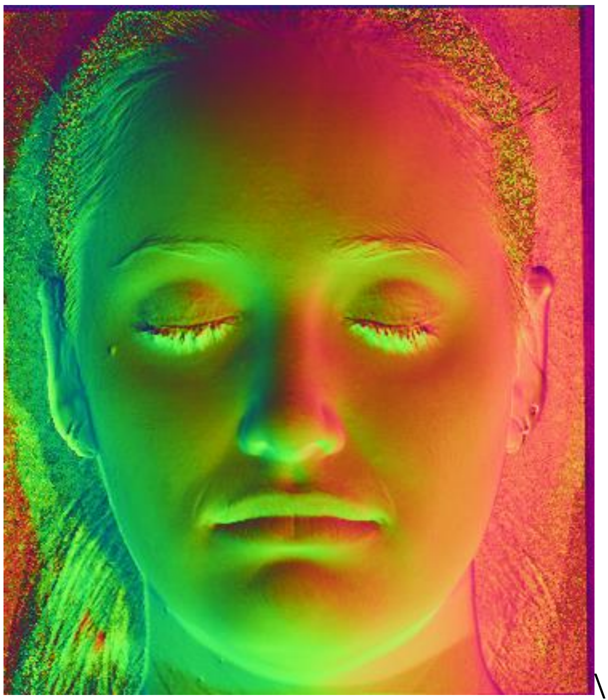
$$x = I$$

$$y = B$$

As mentioned in the question that the pseudonormals are to be estimated in least square sense, we use the numpy function numpy.linalg.lstsq for calculating B. In numpy.linalg.lstsq we need not perform any additional construction of matrix A , but just use in the form as,

$B = numpy.linalg.lstsq(L.T, I, rcond = None)$[0]

Problem 1(f)

\

In the above left image (Albedo) the regions which is dark in the actual image (near nose and ears) have become brighter, this is due to the fact that the image capture process violates the n-dot-l model as it is capturing inter-reflections from the corners as well.

In the above right image (Normals) the normals match the curvature of the face.

Problem 1(g)

For a depth problem, we have:

$$V_1 = (1, 0, Z_{x+1,y} - z_{x,y})$$

$$0 = N \cdot V_1$$

$$= n_1 + n_3(z_{x+1,y} - z_{x,y})$$

Similarly, we have:

$$V_2 = (1, 0, Z_{x+1,y} - z_{x,y})$$

$$0 = N \cdot V_2$$

$$= n_2 + n_3(z_{x+1,y} - z_{x,y})$$

Therefore:

$$\frac{\partial f\ (x,y)}{\partial x} = -\frac{n_1}{n_3}$$

$$\frac{\partial f\ (x,y)}{\partial x} = -\ \frac{n_1}{n_3}$$

Problem 1(h)

$$g_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

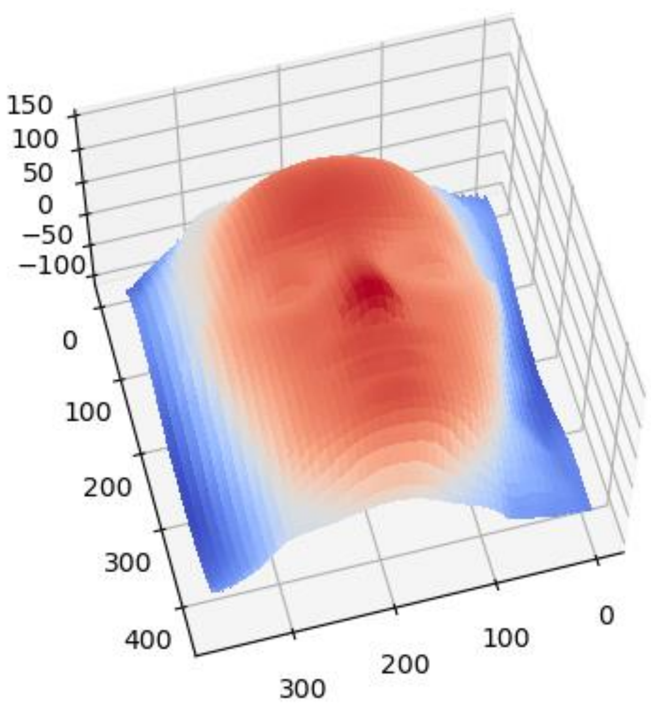
$$g_y = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

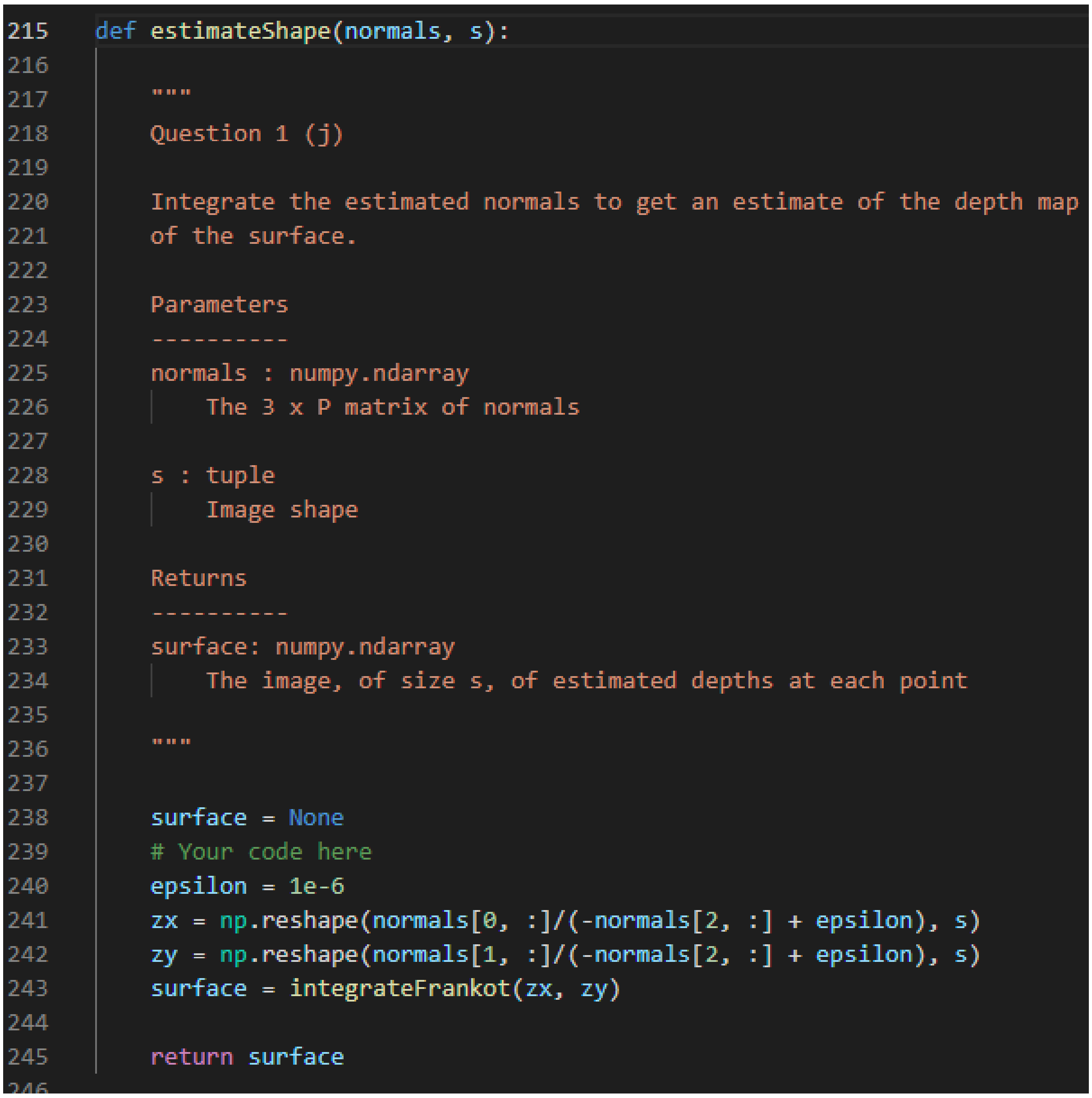
Reconstructed g matrices from both ways are the same:

$$g_y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

When gradients toward a specific direction is modified, the g_x and g_y would not be non-integrable, therefore, when we calculate gradients from both x and y directions in (g), there is the possibility that in some slope and edges the gradients will be non-integrable.

Problem 1(i)





```
def estimateShape(normals, s):

    """
    Question 1 (j)

    Integrate the estimated normals to get an estimate of the depth map
    of the surface.

    Parameters
    ----------
    normals : numpy.ndarray
        The 3 x P matrix of normals

    s : tuple
        Image shape

    Returns
    ----------
    surface: numpy.ndarray
        The image, of size s, of estimated depths at each point

    """

    surface = None
    # Your code here
    epsilon = 1e-6
    zx = np.reshape(normals[0, :]/(-normals[2, :] + epsilon), s)
    zy = np.reshape(normals[1, :]/(-normals[2, :] + epsilon), s)
    surface = integrateFrankot(zx, zy)

    return surface
```

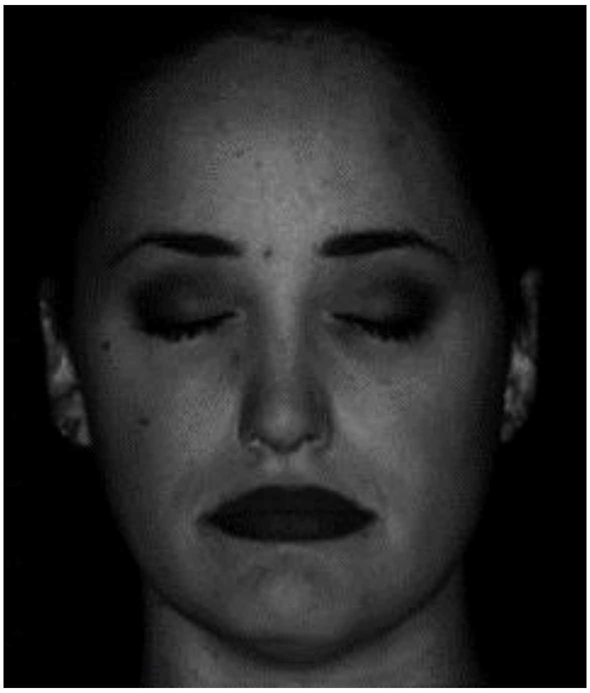
Problem 2(a)

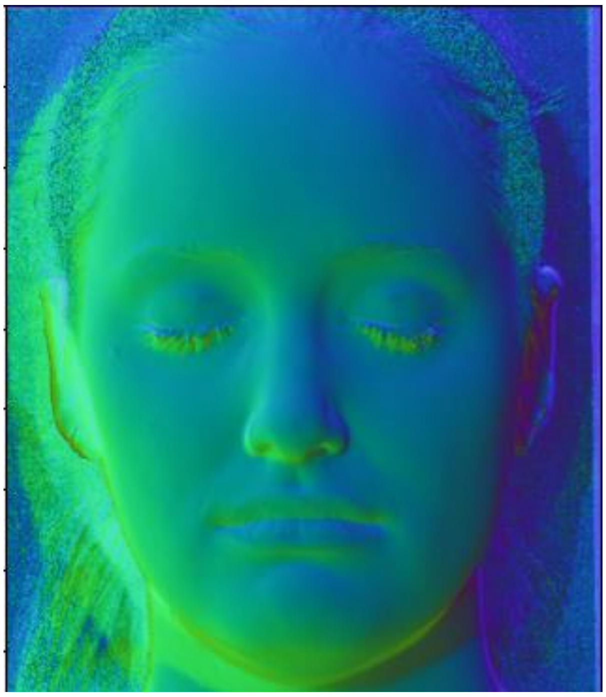
For singular value decomposition we have $\mathbf{M} = \mathbf{U\Sigma V}^T$, therefore, we could do same thing to matrix **I**:

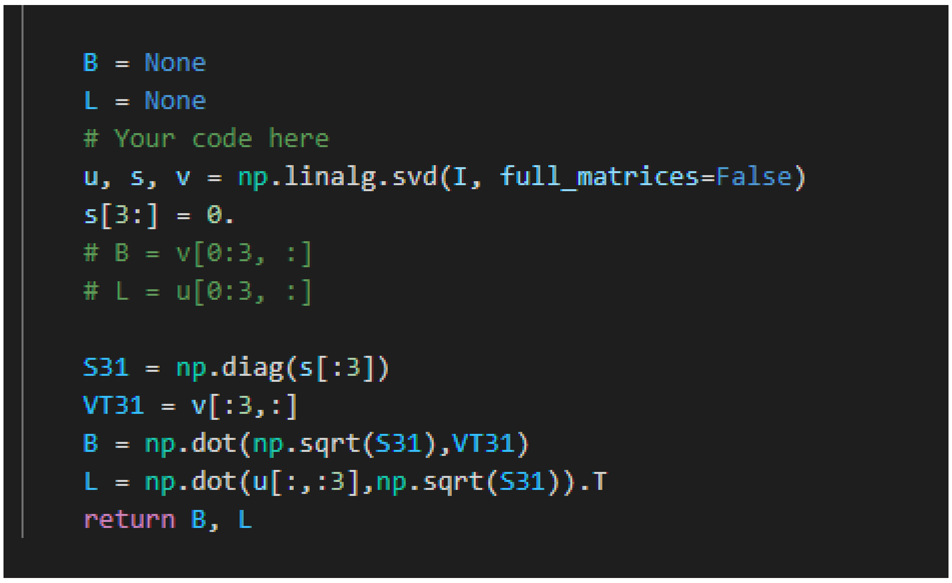
$$\mathbf{I} = \mathbf{U\Sigma V}^T$$

In this case, **I** has a dimension of $7 \times P$, so **U** is 7×7 and **V** is $P \times P$. Set all singular valuesexcept top k from Σ to 0 and choose top 3 vector from **U** as **L** and top 3 vector from **V** as **B**.

Problem 2(b)







```
B = None
L = None
# Your code here
u, s, v = np.linalg.svd(I, full_matrices=False)
s[3:] = 0.
# B = v[0:3, :]
# L = u[0:3, :]

S31 = np.diag(s[:3])
VT31 = v[:3,:]
B = np.dot(np.sqrt(S31),VT31)
L = np.dot(u[:,:3],np.sqrt(S31)).T
return B, L
```

Problem 2(c)

L_0 ground truth lighting direction is:

```
[[-0.1418     0.1215     -0.069     0.067      -0.1627     0.         0.1478]
 [-0.1804     -0.2026    -0.0345    -0.0402     0.122      0.1194     0.1209]
 [-0.9267     -0.9717    -0.838     -0.9772    -0.979     -0.9648    -0.9713]]
```

$\hat{L}$ obtained by factorization is:

```
[[-2.85201979     -3.70224968    -2.29991163     -3.58159342    -3.42762044     -3.23473591    -3.19937559]
 [ 0.89993439     -2.2191511      0.47629996     -0.59379374     2.22099759      0.45043184    -0.74679075]
 [ 1.79851522       0.96528268     0.410125       -0.0210451     -0.28691153     -0.87230035    -1.80219843]]
```

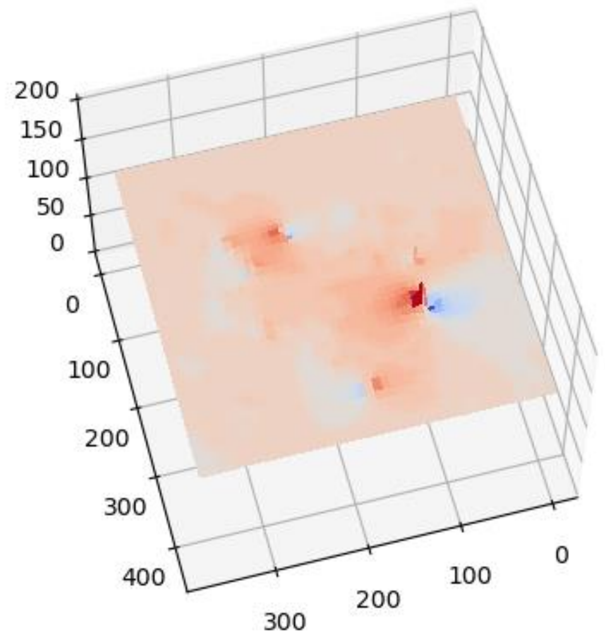
Therefore, L_0 and $\hat{L}$ are not similar.

I can be factorized as below also:

$$L^T = U_3, B = \Sigma_3 V_3^T$$

The above equation will keep the images rendered.

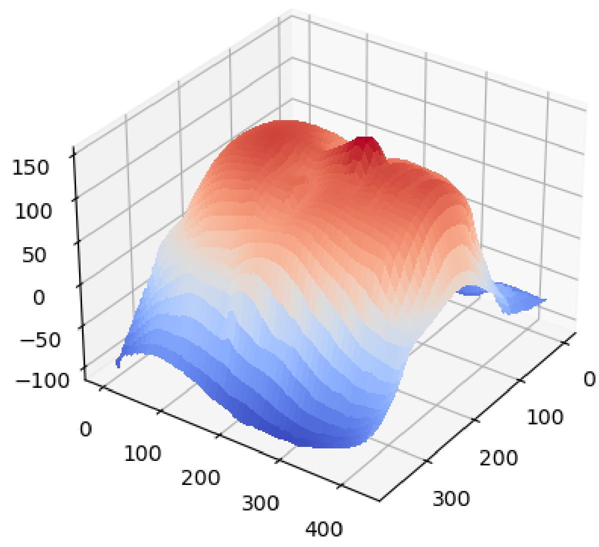
Problem 2(d)

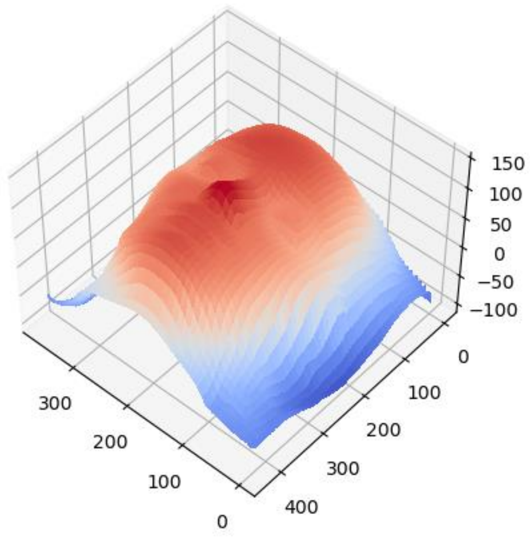


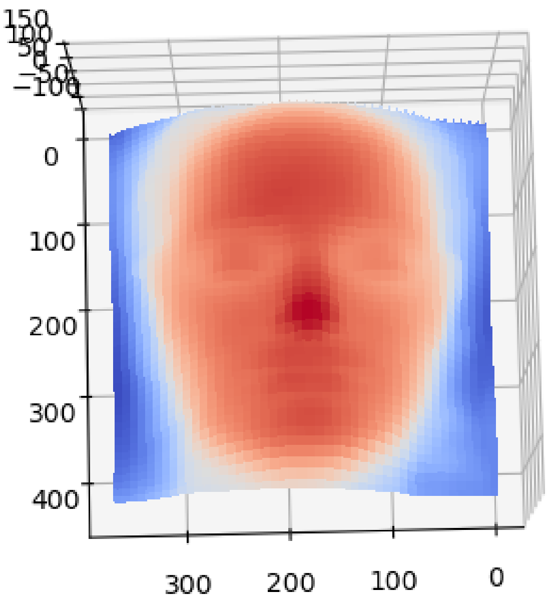
The reconstruction does not look looks like a face from the calibrated photometric stereo

16720 – hw5
mrohitth

Problem 2(e)

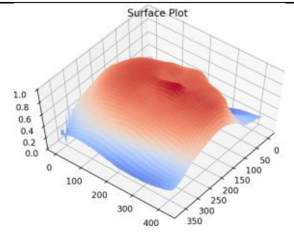
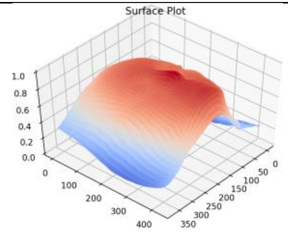
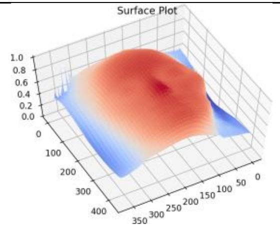




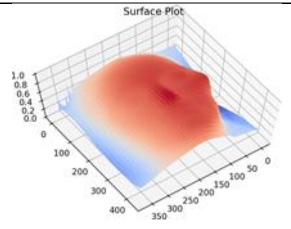
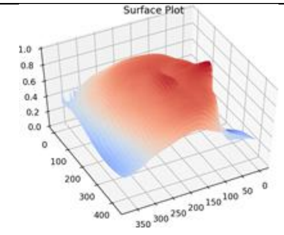
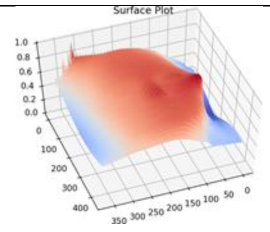


From the above plots we see that it looks like a face similar to calibrated photometric stereo from Q1.i

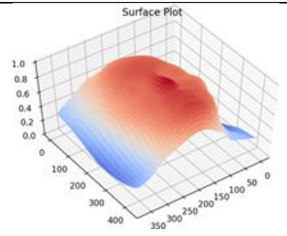
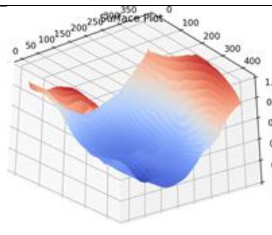
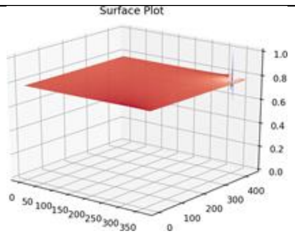
Problem 2(f)

 μ = -0.5	 μ = 0.5	 μ = 1

Changing μ in a small range does not affect the reconstruction much.

 ν = 0.5	 ν = 1	 ν = 3

Noticed that increasing ν would stretch one side of the face, which indicates the increase of gradients.

 λ = -1	 λ = 1.5	 λ = 3

Noticed that with large λ the reconstruction became a flat area, which may be gradients vanished.

Problem 2(g)

To make the estimated surface flattest, I would choose large λ and set μ and ν to zero.

Problem 2(h)

Acquiring more pictures from more lighting directions would help in finding a better reconstruction but it could be computational consuming.