Assignment E - Brightfield histology color deconvolution

- Author: Catherine Chia and Aoming Sun
- Teacher and TAs: Marten Postma, Aaron Lin, Aoming Sun, Catherine Chia
- Date: 21st June, 2023

Outline of workflow

1. Prerequisites:

- Use ImageJ to crop and export images: Stain 1, Stain 2, Background, **OR**
- Use ImageJ to export the RGB vectors for the same images

2. Preprocessing

3. Color Deconvolution

4. Separate stains

In [37]:
```
#Libraries
from matplotlib import pyplot as plt, patches
import numpy as np


#Enable nice output printing features
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'last_expr_or_assign'
import warnings
warnings.filterwarnings('ignore')

#Add other libraries as you see fit
import tifffile as tif
```

Preprocessing steps

In [38]:
```
#Start coding here

#Import IHC image and split it to RGB
#IHC2
img_ihc = tif.imread("IPQDA_23_ASS_E_DATA\IHC2.tif")

#H_E
#img_ihc = tif.imread("IPQDA_23_ASS_E_DATA\H_E.tif")

#RGB channels
img_ihc_r = img_ihc[:, :, 0]
img_ihc_g = img_ihc[:, :, 1]
img_ihc_b = img_ihc[:, :, 2]

#Import cropped stain1, stain2 and background ROI images, OR import RGB vectors of the ROIs
#IHC2
img_stain1 =  tif.imread("IPQDA_23_ASS_E_DATA\IHC2_stain1.tif")
img_stain2 = tif.imread("IPQDA_23_ASS_E_DATA\IHC2_stain2.tif")
img_background = tif.imread("IPQDA_23_ASS_E_DATA\IHC2_back.tif")
#H_E
#img_stain1 =  tif.imread("IPQDA_23_ASS_E_DATA\H_E_stain1.tif")
#img_stain2 = tif.imread("IPQDA_23_ASS_E_DATA\H_E_stain2.tif")
#img_background = tif.imread("IPQDA_23_ASS_E_DATA\H_E_back.tif")
#End coding here
```

Out[38]:
```
array([[[241, 239, 242],
        [239, 237, 240],
        [243, 241, 244],
        ...,
        [253, 253, 253],
        [255, 255, 255],
        [255, 255, 255]],

       [[240, 238, 241],
        [239, 237, 240],
        [243, 241, 244],
        ...,
        [254, 254, 254],
        [255, 255, 255],
        [255, 255, 255]],

       [[245, 243, 246],
        [244, 242, 245],
        [248, 246, 249],
        ...,
        [254, 252, 253],
        [254, 252, 253],
        [252, 250, 251]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [252, 252, 252],
        [253, 253, 253],
        [251, 251, 251]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [253, 253, 253],
        [254, 254, 254],
        [253, 253, 253]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [252, 252, 252],
        [254, 254, 254],
        [254, 254, 254]]], dtype=uint8)
```

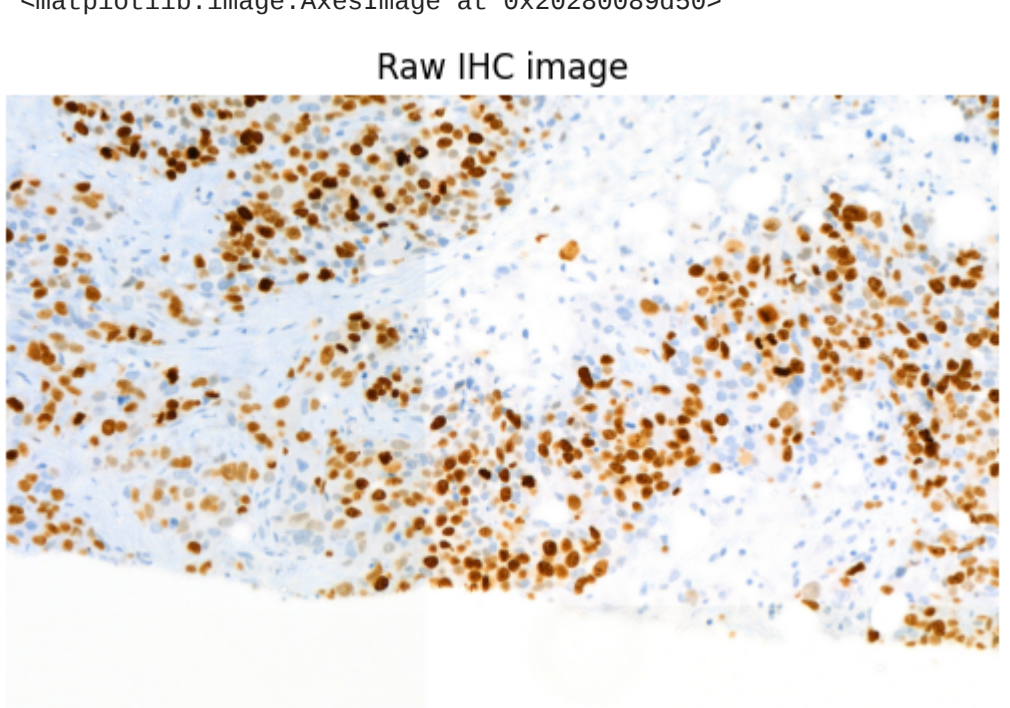
In [39]:
```
img_ihc.shape
```

Out[39]:
```
(2107, 3310, 3)
```

In [40]:
```
#Inspect imported IHC image
plt.title("Raw IHC image")
plt.axis('off')
plt.imshow(img_ihc)
```

Out[40]:
```
<matplotlib.image.AxesImage at 0x20280089d50>
```

Raw IHC image



Calculate RGB mean of the images

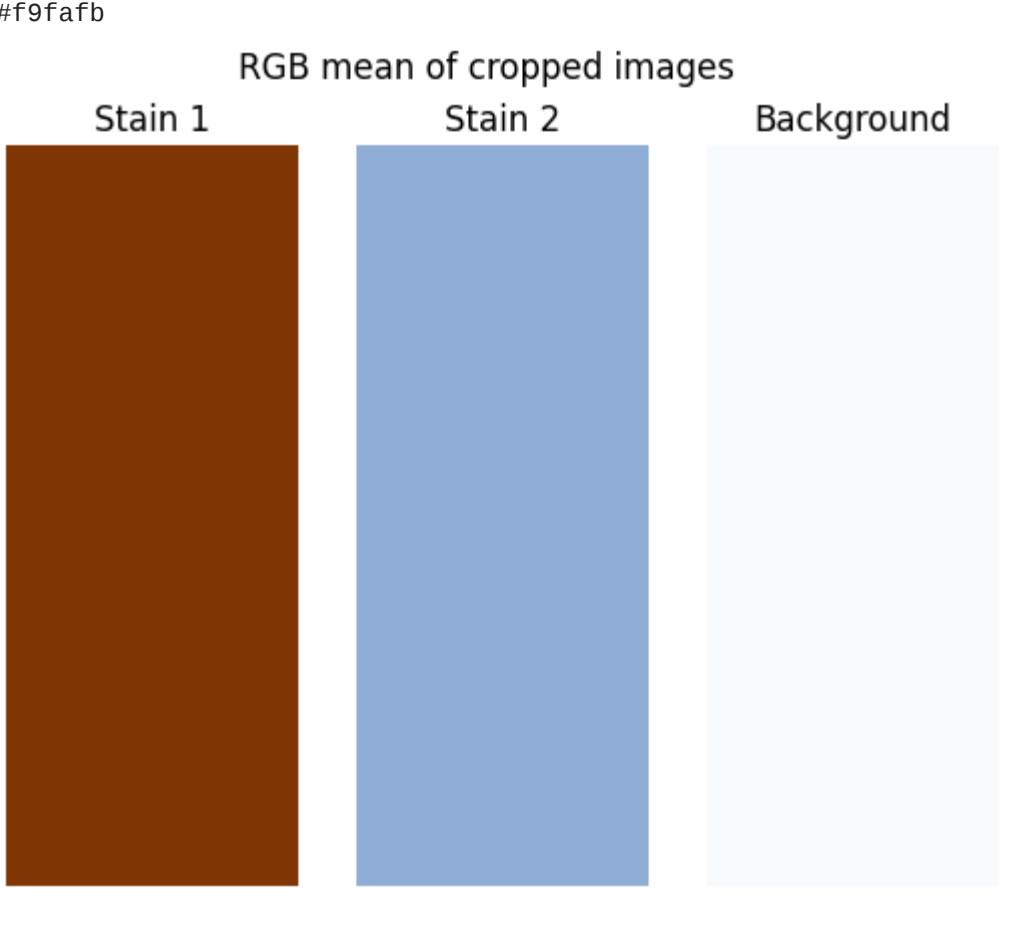
In [41]:
```
#Start coding here

#Calculate mean of image for each RGB channels. If you use RGB vectors, assign them directly to the variables here
mean_img_stain1 = np.mean(img_stain1, axis=(0, 1))
mean_img_stain2 = np.mean(img_stain2, axis=(0, 1))
mean_img_background = np.mean(img_background, axis=(0, 1))

#End coding here

print(mean_img_stain1)
print(mean_img_stain2)
print(mean_img_background)
```

```
[127.08518519  54.66666667   4.56296296]
[144.20061728 173.73765432 214.93518519]
[249.68632783 250.90106902 251.8132033 ]
```

Inspect ROIs of stains and background to ensure correct stain color selection

In [42]:
```
#Convert RGB values to Hex color values for visualization
hex_img_stain1 = '#%02x%02x%02x' % tuple(mean_img_stain1.astype(int))
hex_img_stain2 = '#%02x%02x%02x' % tuple(mean_img_stain2.astype(int))
hex_img_background = '#%02x%02x%02x' % tuple(mean_img_background.astype(int))

print(hex_img_stain1)
print(hex_img_stain2)
print(hex_img_background)

#Visualization of RGB mean of cropped images
fig, axs = plt.subplots(1,3)

fig.suptitle('RGB mean of cropped images')

rectangle_stain1 = patches.Rectangle((0, 0), 1, 1, facecolor=hex_img_stain1)
rectangle_stain2 = patches.Rectangle((0, 0), 1, 1, facecolor=hex_img_stain2)
rectangle_background = patches.Rectangle((0, 0), 1, 1, facecolor=hex_img_background)

axs[0].add_patch(rectangle_stain1)
axs[1].add_patch(rectangle_stain2)
axs[2].add_patch(rectangle_background)
axs[0].set_title('Stain 1')
axs[1].set_title('Stain 2')
axs[2].set_title('Background')

axs[0].axis('off')
axs[1].axis('off')
axs[2].axis('off')
plt.show()
```

```
#7f3604
#90add6
#f9fafb
```



Color Deconvolution

Calculate transmittance, T and convert it to absorbances, OD according to Beer–Lambert law

In [43]:
```
#Calculate transmittances, T for each stain
T_stain1 = mean_img_background/mean_img_stain1
T_stain2 = mean_img_background/mean_img_stain2
OD_stain1 = np.log10(T_stain1)
OD_stain2 = np.log10(T_stain2)

print(OD_stain1)
print(OD_stain2)
```

```
[0.29329984 0.66177992 1.74183155]
[0.23842764 0.15960856 0.06877098]
```

Normalize the absorbances to vector lengths

In [44]:
```
#Start coding here
#Normalize the absorbances
#Normalize by the squared of the sum of the vector values to the power of 2
OD_stain1_norm = OD_stain1 / np.max(OD_stain1)
OD_stain2_norm = OD_stain2 / np.max(OD_stain2)
#End coding here

print(OD_stain1_norm)
print(OD_stain2_norm)
```

```
[0.16838588 0.37993336 1.        ]
[1.         0.66942137 0.28843544]
```

Form a deconvolution matrix

In [45]:
```
#Start coding here

#Combine OD_stain1_norm and OD_stain2_norm to form a normalized OD matrix M
M = np.column_stack((OD_stain1_norm, OD_stain2_norm))

#Calculate the deconvolution matrix according to Linear regression
MT = np.transpose(M)

MT_M = np.dot(MT, M)

inversed_MT_M = np.linalg.inv(MT_M)

D = np.dot(inversed_MT_M, MT)
D=D.T

#End coding here

print("M")
print(M)
print("M transposed")
print(MT)
print("Inversed M transposed multiplied with M")
print(inversed_MT_M)
print("Deconvolution matrix, D")
print(D)
```

```
M
[[0.16838588 1.        ]
 [0.37993336 0.66942137]
 [1.         0.28843544]]
M transposed
[[0.16838588 0.37993336 1.        ]
 [1.         0.66942137 0.28843544]]
Inversed M transposed multiplied with M
[[ 1.18703318 -0.55126738]
 [-0.55126738  0.90904422]]
Deconvolution matrix, D
[[-0.35138776  0.81621858]
 [ 0.08196334  0.39908875]
 [ 1.02802813 -0.28906681]]
```

Calculate the coefficient for each stain

In [46]:
```
#Convert pixel intensity to transmittance to absorbance according to Beer-Lambert Law on the IHC image
#Calculate the transmittance
T_img_ihc = mean_img_background/img_ihc

#Because of the logarithmic function in the next step, we assign all transmittance value less than 1 to 1
T_img_ihc[T_img_ihc<1] = 1
```

In [47]:
```
#Start coding here

#Calculate the absorbance
OD_img_ihc = np.log10(T_img_ihc)

#Coefficient matrix
coeffs = np.dot(OD_img_ihc, D)

#Extracting the individual coefficients from the coefficient matrix
#Which are essentially the orthogonal representation of the stains of the IHC image
coeff_stain1 = coeffs[:,:, 0]
coeff_stain2 = coeffs[:,:, 1]


#End coding here

print(coeff_stain1.shape)
print(coeff_stain2.shape)
```

```
(2107, 3310)
(2107, 3310)
```

Separate stains

Multiply the coefficients with the stain absorbance to get the image absorbance per stain

In [48]:
```
OD_img_ihc
```

Out[48]:
```
array([[[0.03190678, 0.01929127, 0.        ],
        [0.0281789 , 0.01748547, 0.        ],
        [0.04328632, 0.03214659, 0.00487915],
        ...,
        [0.01718352, 0.04731999, 0.13626067],
        [0.01537772, 0.04925449, 0.14580599],
        [0.02081781, 0.05707983, 0.16555005]],

       [[0.03003884, 0.02110461, 0.        ],
        [0.0263269 , 0.01748547, 0.        ],
        [0.03945992, 0.02843465, 0.00140478],
        ...,
        [0.01537772, 0.04925449, 0.14339992],
        [0.00822868, 0.04156766, 0.14100711],
        [0.00645966, 0.04156766, 0.14580599]],

       [[0.0263269 , 0.01748547, 0.        ],
        [0.03190678, 0.02292555, 0.        ],
        [0.04328632, 0.03214659, 0.00487915],
        ...,
        [0.02448276, 0.0590584 , 0.15556583],
        [0.01178849, 0.04731999, 0.14822547],
        [0.00469781, 0.04539407, 0.14822547]],

       ...,

       [[0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.00140478],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.00140478],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]]])
```

In [49]:
```
#Initialize the image absorbance container per stain
OD_img_ihc_stain1 = np.zeros((img_ihc.shape[0], img_ihc.shape[1], img_ihc.shape[2]))
OD_img_ihc_stain2 = np.zeros((img_ihc.shape[0], img_ihc.shape[1], img_ihc.shape[2]))
#Start coding here
print(OD_img_ihc_stain1.shape)
print(coeff_stain1.shape)
#Multiply the coefficients with the stain absorbance per stain. Do it independently for each RGB layer"""
"""for i in range(3):
    OD_img_ihc_stain1[:, :, i] = OD_img_ihc[:, :, i] * coeff_stain1
    OD_img_ihc_stain2[:, :, i] = OD_img_ihc[:, :, i] * coeff_stain2"""
OD_img_ihc_stain1 = np.reshape(coeff_stain1, img_ihc_r.shape)[:, :, np.newaxis] * OD_stain1
OD_img_ihc_stain2 = np.reshape(coeff_stain2, img_ihc_r.shape)[:, :, np.newaxis] * OD_stain2


#End coding here
```

```
(2107, 3310, 3)
(2107, 3310)
```

Out[49]:
```
array([[[ 8.04498575e-03,  5.38548534e-03,  2.32045900e-03],
        [ 7.14767914e-03,  4.78480913e-03,  2.06164397e-03],
        [ 1.11465026e-02,  7.46170698e-03,  3.21504637e-03],
        ...,
        [-1.54454695e-03, -1.03395273e-03, -4.45502077e-04],
        [-2.36977500e-03, -1.58637801e-03, -6.83527092e-04],
        [-1.92726496e-03, -1.29015234e-03, -5.55891516e-04]],

       [[ 7.85401468e-03,  5.25764523e-03,  2.26537617e-03],
        [ 6.78726220e-03,  4.54353833e-03,  1.95768695e-03],
        [ 1.02881028e-02,  6.88707582e-03,  2.96745345e-03],
        ...,
        [-2.20394502e-03, -1.47536788e-03, -6.35695848e-04],
        [-4.16172797e-03, -2.78594962e-03, -1.20038983e-03],
        [-4.83674214e-03, -3.23781852e-03, -1.39508784e-03]],

       [[ 6.78726220e-03,  4.54353833e-03,  1.95768695e-03],
        [ 8.39080171e-03,  5.61698193e-03,  2.42020457e-03],
        [ 1.11465026e-02,  7.46170698e-03,  3.21504637e-03],
        ...,
        [-3.37635932e-04, -2.26020707e-04, -9.73861683e-05],
        [-3.41910073e-03, -2.28881908e-03, -9.86189820e-04],
        [-4.98227000e-03, -3.33523798e-03, -1.43706323e-03]],

       ...,

       [[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        ...,
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]],

       [[-9.68193103e-05, -6.48129149e-05, -2.79261203e-05],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        ...,
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]],

       [[-9.68193103e-05, -6.48129149e-05, -2.79261203e-05],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        ...,
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]]])
```

Convert the image absorbance to image transmittance

In [50]:
```
#Convert absorbance to transmittance
T_img_ihc_stain1 = 10**(-OD_img_ihc_stain1)
T_img_ihc_stain2 = 10**(-OD_img_ihc_stain2)
```

Out[50]:
```
array([[[0.98164626, 0.98767603, 0.99467119],
        [0.98367656, 0.98904304, 0.99526414],
        [0.9746608 , 0.98296554, 0.99262442],
        ...,
        [1.00356278, 1.0023836 , 1.00102633],
        [1.00547152, 1.00365945, 1.00157512],
        [1.00444755, 1.0029751 , 1.00128081]],

       [[0.98207801, 0.98796681, 0.99479736],
        [0.98449324, 0.98959265, 0.9955024 ],
        [0.97658916, 0.984267  , 0.99319048],
        ...,
        [1.00508767, 1.00340294, 1.00146482],
        [1.00962879, 1.00643551, 1.00276782],
        [1.01119926, 1.00748321, 1.00321747]],

       [[0.98449324, 0.98959265, 0.9955024 ],
        [0.98086491, 0.9871497 , 0.99444277],
        [0.9746608 , 0.98296554, 0.99262442],
        ...,
        [1.00077774, 1.00052057, 1.00022427],
        [1.00790384, 1.00528411, 1.00227337],
        [1.01153816, 1.00770923, 1.00331444]],

       ...,

       [[1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       [[1.00022296, 1.00014925, 1.0000643 ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       [[1.00022296, 1.00014925, 1.0000643 ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]]])
```

Clip each layer in the image transmittance to values between 0 and 1, preparing for conversion to values between 0 and 255 later

In [51]:
```
#Clip each layer to 0,1
T_img_ihc_stain1[T_img_ihc_stain1>1] = 1
T_img_ihc_stain2[T_img_ihc_stain2>1] = 1
T_img_ihc_stain1[T_img_ihc_stain1<0] = 0
T_img_ihc_stain2[T_img_ihc_stain2<0] = 0
```

Convert the image transmittance to values between 0 and 255 (integers), so that plotting is possible

In [52]:
```
#Start coding here

T_img_ihc_stain1_norm = (T_img_ihc_stain1 * 255).astype(np.uint8)

T_img_ihc_stain2_norm = (T_img_ihc_stain2 * 255).astype(np.uint8)
#End coding here
```

Out[52]:
```
array([[[250, 251, 253],
        [250, 252, 253],
        [248, 250, 253],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[250, 251, 253],
        [251, 252, 253],
        [249, 250, 253],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[251, 252, 253],
        [250, 251, 253],
        [248, 250, 253],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8)
```

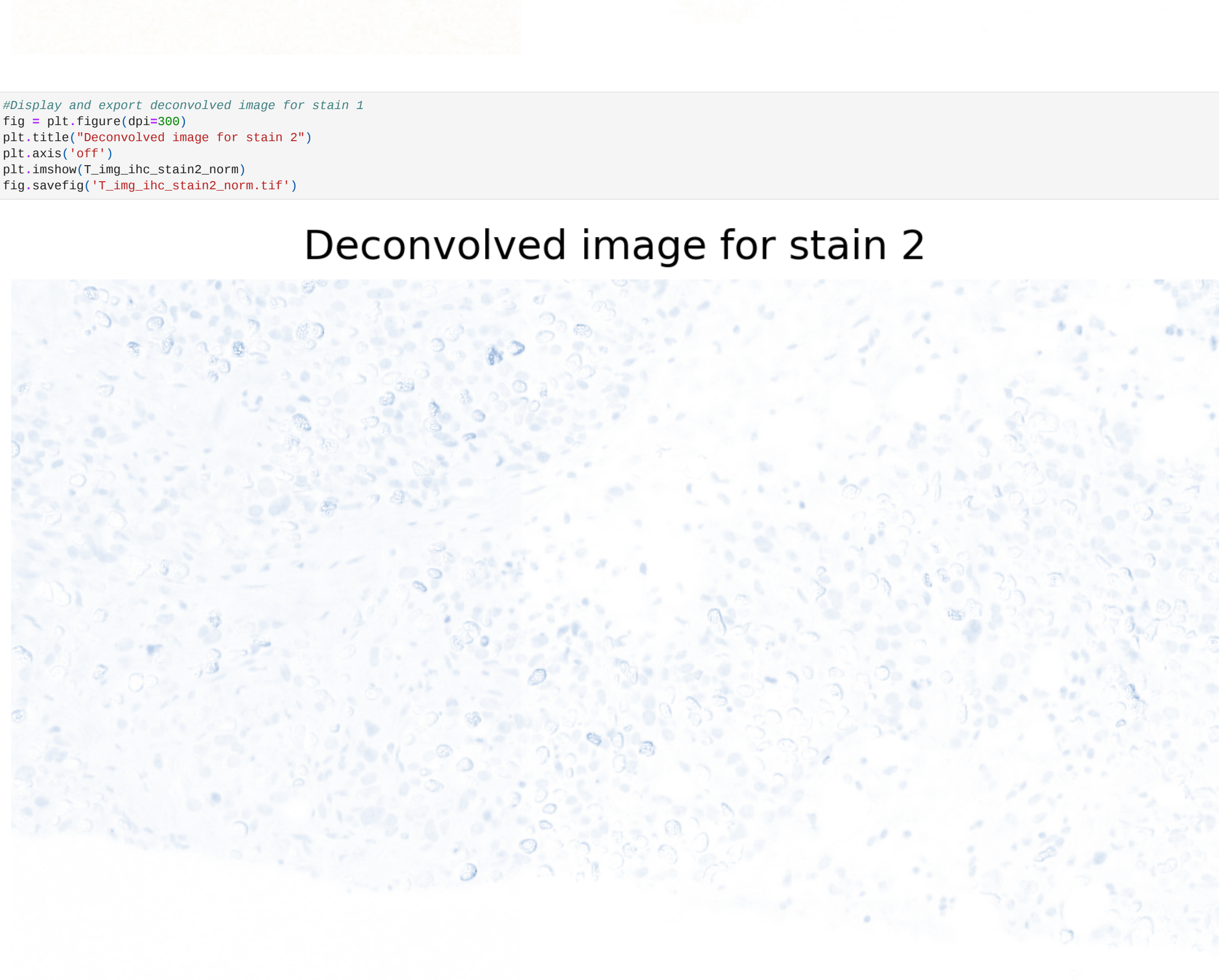
Visualize deconvolved images

In [53]:
```
#Display deconvolved image for stain 1
fig = plt.figure(dpi=300)
plt.title("Deconvolved image for stain 1")
plt.axis('off')
plt.imshow(T_img_ihc_stain1_norm)
fig.savefig('T_img_ihc_stain1_norm.tif')
```

Deconvolved image for stain 1

In [54]:
```
#Display and export deconvolved image for stain 1
fig = plt.figure(dpi=300)
plt.title("Deconvolved image for stain 2")
plt.axis('off')
plt.imshow(T_img_ihc_stain2_norm)
fig.savefig('T_img_ihc_stain2_norm.tif')
```

Deconvolved image for stain 2



In [54]: