CSE252A_FA25_assignment_0

October 5, 2025

1 CSE 252A Computer Vision I, Fall 2025 - Assignment 0

Instructor: Ben Ochoa

Assigment due: Wed, Oct 8, 11:59 PM

Name:

PID:

1.1 Prior knowledge + certification of commencement of academic activity

In every course at UC San Diego, per the US Department of Education, we are now required to certify whether students have commenced academic activity for a class to be counted towards eligibility for Title IV federal financial aid. This certification must be completed during the first two weeks of instruction.

For CSE 252A, this requirement will be fulfilled via an ungraded prior knowledge quiz, which will assist the instructional team by providing information about your background coming into the course. Although this will be part of the first assignment, you are welcome to complete the quiz early. In Canvas, go to the CSE 252A course and navigate to Quizzes. Then, click on the "First Day Survey: Prior Knowledge #FinAid"

1.2 Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please follow the Academic Integrity Policy and Collaboration Policy on Canvas.
- All the solutions must be written in this Jupyter notebook.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.
- You must **submit 3 files: the Notebook, the PDF and the python file** (i.e. the `.ipynb`, the `.pdf` and the `.py` files) on Gradescope. **You must mark each problem on Gradescope in the pdf.**
  - To convert the notebook to PDF, you can choose one way below:

    - You may first export the notebook as HTML, and then print the web page as PDF
        - e.g., in Chrome: File → Save and Export Notebook as → "HTML"; or in VScode: Open the Command Palette by pressing Ctrl+Shift+P (Windows/Linux) or Cmd+Shift+P (macOS), search for Jupyter: Export to HTML
        - Open the saved web page and right click → Print... → Choose "Destination: Save as PDF" and click "Save")
    - If you have XeTex installed on your machine, you may directly export the notebook as PDF: e.g., in Chrome, File → Save and Export Notebook as → "PDF"
    - You may use nbconvert to convert the ipynb file to pdf using the following command `jupyter nbconvert --allow-chromium-download --to webpdf filename.ipynb`
  - To convert the notebook to python file, you can choose one way below:
    - You may directly export the notebook as py: e.g., in Chrome, File → Save and Export Notebook as → "Executable script"; or in VScode: Open the Command Palette and search for Jupyter: Export to Python Script
    - You may use nbconvert to convert the ipynb file to python file using the following command `jupyter nbconvert --to script filename.ipynb --output output_filename.py`
- Please make sure the content in each cell (e.g. code, output images, printed results, etc.) are clearly visible and are not cut-out or partially cropped in your final PDF file.
- While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

1.3 Introduction

Welcome to **CSE 252A Computer Vision I**!

This course provides a comprehensive introduction to computer vision providing broad coverage including low level vision (image formation, photometry, color, image feature detection), inferring 3D properties from images (shape-from-shading, stereo vision, motion interpretation) and object recognition.

We will use a variety of tools (e.g. some packages and operations) in this class that may require some initial configuration. To ensure smooth progress, we will setup the majority of the tools to be used in this course in this **Assignment 0**. You will also practice some basic image manipulation techniques.

1.4 Piazza, Gradescope and Python

Piazza

All students are automatically added to the class in Piazza once enrolled in this class. You can get access to it from Canvas. You'll be able to ask the professor, the TAs and your classmates questions on Piazza. Class announcements will be made using Piazza, so make sure you check your email or Piazza frequently.

Gradescope

All students are automatically added to the class in Gradescope once enrolled in this class. You can also get access to it from Canvas. All the assignments are required to be submitted to Gradescope for grading. Make sure that you mark each page for different problems.

Python

We will use the Python programming language for all assignments in this course, with a few popular libraries (`NumPy`, `Matplotlib`). Assignments will be given in the format of web-based Jupyter notebook that you are currently viewing. We expect that many of you have some experience with `Python` and `NumPy`. And if you have previous knowledge in `MATLAB`, check out the NumPy for MATLAB users page. The section below will serve as a quick introduction to `NumPy` and some other libraries.

1.5 Getting Started with NumPy

`NumPy` is the fundamental package for scientific computing with Python. It provides a powerful N-dimensional array object and functions for working with these arrays. Some basic use of this packages is shown below. This is **NOT** a problem, but you are highly recommended to run the following code with some of the input changed in order to understand the meaning of the operations.

1.5.1 Arrays

[1]:
```
import numpy as np              # Import the NumPy package

v = np.array([1, 2, 3])         # A 1D array
print(v)
print(v.shape)                  # Print the size / shape of v
print("1D array:", v, "Shape:", v.shape)

v = np.array([[1], [2], [3]])   # A 2D array
print("2D array:", v, "Shape:", v.shape) # Print the size of v and check the␣
 ↪difference.

# You can also attempt to compute and print the following values and their size.

v = v.T                         # Transpose of a 2D array
m = np.zeros([3, 4])            # A 2x3 array (i.e. matrix) of zeros
v = np.ones([1, 3])             # A 1x3 array (i.e. a row vector) of ones
v = np.ones([3, 1])             # A 3x1 array (i.e. a column vector) of ones
m = np.eye(4)                   # Identity matrix
```

```
m = np.random.rand(2, 3)        # A 2x3 random matrix with values in [0, 1]␣
↪(sampled from uniform distribution)
```

```
[1 2 3]
(3,)
1D array: [1 2 3] Shape: (3,)
2D array: [[1]
 [2]
 [3]] Shape: (3, 1)
```

1.5.2 Array Indexing

[2]:
```
import numpy as np

print("Matrix")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m)

print("\nAccess a single element")
print(m[0, 1])                          # Access an element
m[1, 1] = 100                           # Modify an element
print("\nModify a single element")
print(m)

print("\nAccess a subarray")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m[1, :])                          # Access a row (to 1D array)
print(m[1:2, :])                        # Access a row (to 2D array)
print(m[1:3, :])                        # Access a sub-matrix
print(m[1:, :])                         # Access a sub-matrix

print("\nModify a subarray")
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
v1 = np.array([1,1,1])
m[0] = v1
print(m)
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
v1 = np.array([1,1,1])
m[:,0] = v1
print(m)
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
m1 = np.array([[1,1],[1,1]])
m[:2,:2] = m1
print(m)

print("\nTranspose a subarray")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
```

```
print(m[1, :].T)                                    # Notice the difference of the dimension of resulting array
print(m[1:2, :].T)
print(m[1:, :].T)
print(np.transpose(m[1:, :], axes=(1,0)))           # np.transpose() can be used to transpose according given axes list.

print("\nReverse the order of a subarray")
print(m[1, ::-1])                                   # Access a row with reversed order (to 1D array)

# Boolean array indexing
# Given a array m, create a new array with values equal to m
# if they are greater than 2, and equal to 0 if they less than or equal to 2
m = np.array([[1, 2, 3], [4, 5, 6]])
m[m > 2] = 0
print("\nBoolean array indexing: Modify with a scaler")
print(m)

# Given a array m, create a new array with values equal to those in m
# if they are greater than 0, and equal to those in n if they less than or equal 0
m = np.array([[1, 2, -3], [4, -5, 6]])
n = np.array([[1, 10, 100], [1, 10, 100]])
n[m > 0] = m[m > 0]
print("\nBoolean array indexing: Modify with another array")
print(n)
```

```
Matrix
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Access a single element
2

Modify a single element
[[  1   2   3]
 [  4 100   6]
 [  7   8   9]]

Access a subarray
[4 5 6]
[[4 5 6]]
[[4 5 6]
 [7 8 9]]
[[4 5 6]
```

```
 [7 8 9]]

Modify a subarray
[[1 1 1]
 [4 5 6]
 [7 8 9]]
[[1 2 3]
 [1 5 6]
 [1 8 9]]
[[1 1 3]
 [1 1 6]
 [7 8 9]]

Transpose a subarray
[4 5 6]
[[4]
 [5]
 [6]]
[[4 7]
 [5 8]
 [6 9]]
[[4 7]
 [5 8]
 [6 9]]

Reverse the order of a subarray
[6 5 4]

Boolean array indexing: Modify with a scaler
[[1 2 0]
 [0 0 0]]

Boolean array indexing: Modify with another array
[[  1   2 100]
 [  4  10   6]]
```

1.5.3 Array Dimension Operation

[3]:
```
import numpy as np

print("Matrix")
m = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
print(m, m.shape)

print("\nReshape")
re_m = m.reshape(1,2,2)  # Add one more dimension at first.
print(re_m, re_m.shape)
```

```
re_m = m.reshape(2,1,2)  # Add one more dimension in middle.
print(re_m, re_m.shape)
re_m = m.reshape(2,2,1)  # Add one more dimension at last.
print(re_m, re_m.shape)

print("\nStack")
m1 = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
m2 = np.array([[1, 1], [1, 1]]) # Create a 2x2 array.
print(np.stack((m1,m2)))

print("\nConcatenate")
m1 = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
m2 = np.array([[1, 1], [1, 1]]) # Create a 2x2 array.
print(np.concatenate((m1,m2)))
print(np.concatenate((m1,m2), axis=0))
print(np.concatenate((m1,m2), axis=1))
```

```
Matrix
[[1 2]
 [3 4]] (2, 2)

Reshape
[[[1 2]
  [3 4]]] (1, 2, 2)
[[[1 2]]

 [[3 4]]] (2, 1, 2)
[[[1]
  [2]]

 [[3]
  [4]]] (2, 2, 1)

Stack
[[[1 2]
  [3 4]]

 [[1 1]
  [1 1]]]

Concatenate
[[1 2]
 [3 4]
 [1 1]
 [1 1]]
[[1 2]
 [3 4]
 [1 1]
```

```
 [1 1]]
[[1 2 1 1]
 [3 4 1 1]]
```

1.5.4 Math Operations on Array

Element-wise Operations

[4]:
```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
print(a * 3)                                          # Scalar multiplication
print(a / 2)                                          # Scalar division
print(np.round(a / 2))
print(np.power(a, 2))
print(np.log(a))
print(np.exp(a))

b = np.array([[1, 1, 1], [2, 2, 2]], dtype=np.float64)
print(a + b)                                          # Elementwise sum
print(a - b)                                          # Elementwise difference
print(a * b)                                          # Elementwise product
print(a / b)                                          # Elementwise division
print(a == b)                                         # Elementwise comparison
```

```
[[ 3.  6.  9.]
 [12. 15. 18.]]
[[0.5 1.  1.5]
 [2.  2.5 3. ]]
[[0. 1. 2.]
 [2. 2. 3.]]
[[ 1.  4.  9.]
 [16. 25. 36.]]
[[0.         0.69314718 1.09861229]
 [1.38629436 1.60943791 1.79175947]]
[[  2.71828183   7.3890561   20.08553692]
 [ 54.59815003 148.4131591  403.42879349]]
[[2. 3. 4.]
 [6. 7. 8.]]
[[0. 1. 2.]
 [2. 3. 4.]]
[[ 1.  2.  3.]
 [ 8. 10. 12.]]
[[1.  2.  3. ]
 [2.  2.5 3. ]]
[[ True False False]
 [False False False]]
```

Broadcasting

[5]:
```
# Note: See https://numpy.org/doc/stable/user/basics.broadcasting.html
#       for more details.
import numpy as np
a = np.array([[1, 1, 1], [2, 2, 2]], dtype=np.float64)
b = np.array([1, 2, 3])
print(a*b)
```

```
[[1. 2. 3.]
 [2. 4. 6.]]
```

Sum and Mean

[6]:
```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
print("Sum of array")
print(np.sum(a))                # Sum of all array elements
print(np.sum(a, axis=0))        # Sum of each column
print(np.sum(a, axis=1))        # Sum of each row

print("\nMean of array")
print(np.mean(a))               # Mean of all array elements
print(np.mean(a, axis=0))       # Mean of each column
print(np.mean(a, axis=1))       # Mean of each row
```

```
Sum of array
21
[5 7 9]
[ 6 15]

Mean of array
3.5
[2.5 3.5 4.5]
[2. 5.]
```

Vector and Matrix Operations

[7]:
```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[1, 1], [1, 1]])
print("Matrix-matrix product")
print(a.dot(b))                 # Matrix-matrix product
print(a.T.dot(b.T))
print(np.matmul(a,b))
print(np.matmul(a.T, b.T))
print(a @ b)
print(a.T @ b.T)
```

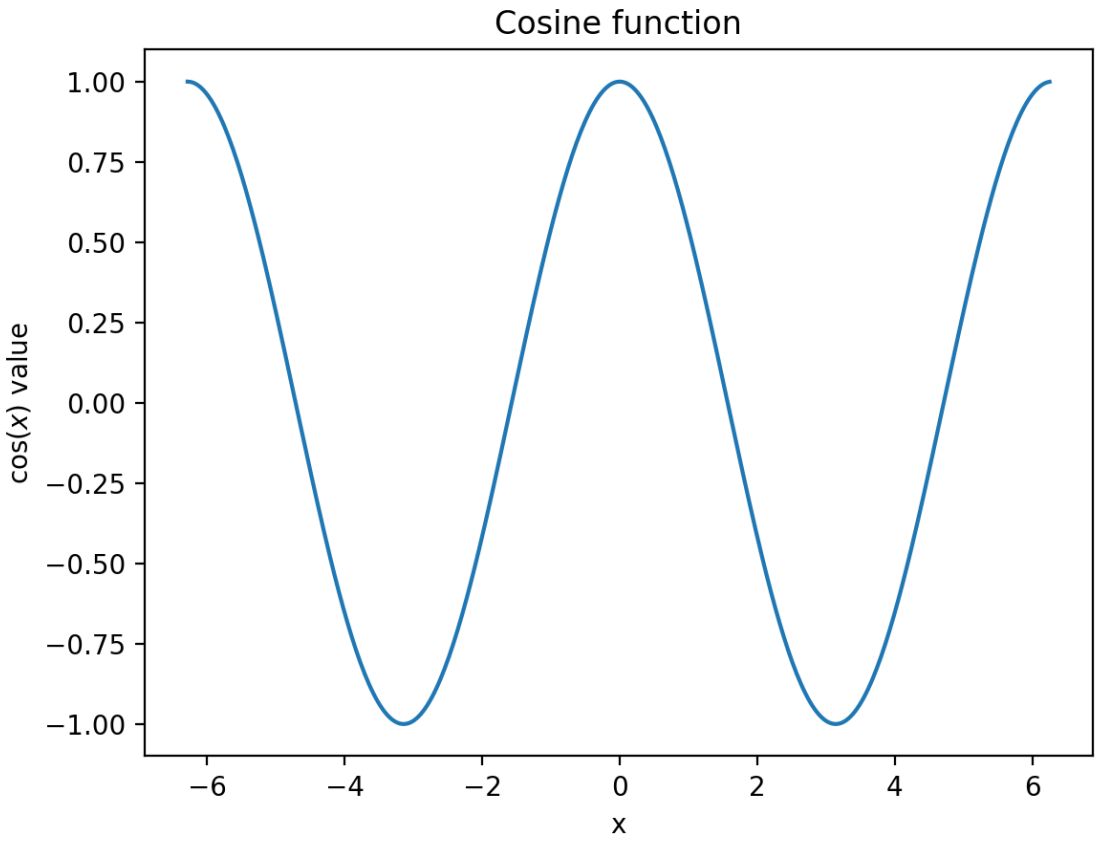
```
x = np.array([3, 4])
print("\nMatrix-vector product")
print(a.dot(x))                  # Matrix-vector product
print(np.matmul(a,x))
print(a @ x)

x = np.array([1, 2])
y = np.array([3, 4])
print("\nVector-vector product")
print(x.dot(y))                  # Vector-vector product
```

```
Matrix-matrix product
[[3 3]
 [7 7]]
[[4 4]
 [6 6]]
[[3 3]
 [7 7]]
[[4 4]
 [6 6]]
[[3 3]
 [7 7]]
[[4 4]
 [6 6]]

Matrix-vector product
[11 25]
[11 25]
[11 25]

Vector-vector product
11
```

1.5.5 Matplotlib

`Matplotlib` is a plotting library. We will use it to show the result in this assignment.

[8]:
```
%config InlineBackend.figure_format = 'retina' # For high-resolution.
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-2., 2., 0.01) * np.pi
plt.plot(x, np.cos(x))
plt.xlabel('x')
plt.ylabel('$\cos(x)$ value') # '$...$' for a LaTeX formula.
plt.title('Cosine function')

plt.show()
```



This brief overview introduces many basic functions from `NumPy` and `Matplotlib`, but is far from complete. Check out more operations and their use in documentations for NumPy and Matplotlib.

1.6 Problem 1: Image Operations and Vectorization (15 points)

Vector operations using `NumPy` can offer a significant speedup over doing an operation iteratively on an image. The problem below will demonstrate the time it takes for both approaches to change the color of quadrants of an image.

The problem reads an image `SunGod.jpg` that you will find in the assignment folder. Two functions are then provided as different approaches for doing an operation on the image.

The function `iterative()` demonstrates the image divided into 4 parts:

(Top Left) The original image. (Top Right) Blue channel image. (Bottom Left) (B,G,R) colored image. (Bottom Right) Grayscale image.

For your implementation:

(1) For the blue channel image, write your implementation to extract a single channel from a colored image. This means that from the $H \times W \times 3$ shaped image, you'll get three matrices of the shape $H \times W$ (Note that it's 2-dimensional).

(2) For the (B,G,R) colored image, write your implementation to merge those single channel images back into a 3-dimensional colored image in the reversed channels order (B,G,R).

(3) For the grayscale image, write your implementation to conduct operations with the extracted channels. You must use the following equation for computing the grayscale value from (R,G,B) channels.

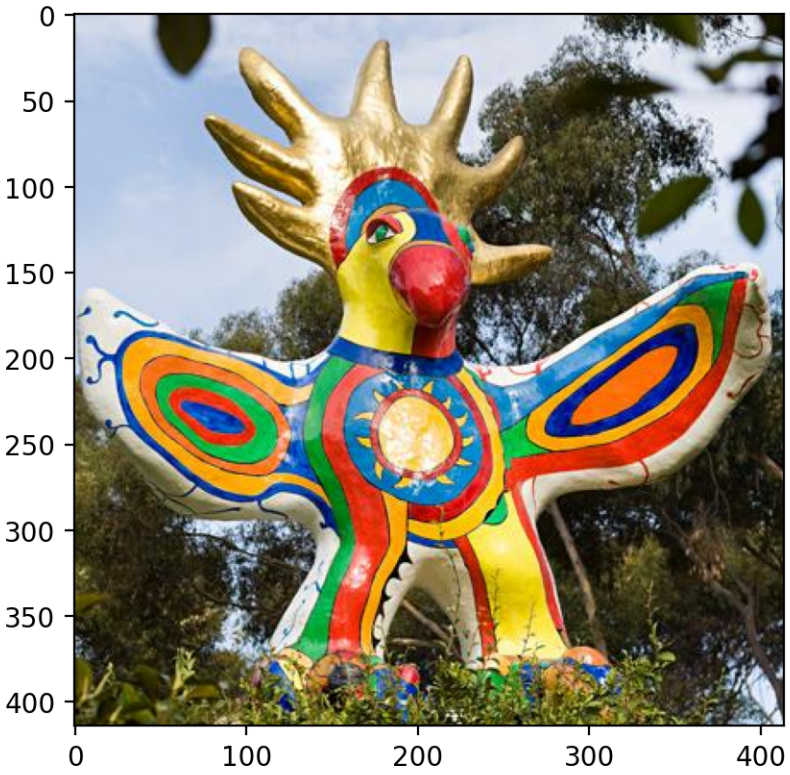
$$gray = 0.21263903 * R + 0.71516871 * G + 0.072192319 * B$$

Note: In a grayscale image, a pixel has the same grayscale value for its red (R), green (G), and blue (B) channels.

Your task is to follow through the code and fill the blanks in `vectorized()` function and compare the speed difference between `iterative()` and `vectorized()`. Make sure your final generated image in the `vectorized()` is the same as the one generated from `iterative()`.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt

img = plt.imread('SunGod.jpg') # Read an image
print("Image shape:", img.shape)           # Print image size and color depth.␣
 ↪The shape should be (H,W,C).

plt.imshow(img)                            # Show the original image
plt.show()
```

```
Image shape: (414, 414, 3)
```



[14]:
```
import copy

# Iterative Approach. No change needed here.
def iterative(img):
    """ Iterative operation. """
    image = copy.deepcopy(img)              # Create a copy of the image matrix
    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            #Top Right
            if y < image.shape[0]/2 and x >= image.shape[1]/2:
                image[y,x] = image[y,x] * np.array([0,0,1])    # Keep the blue␣
 ↪channel
            #Bottom Left
            elif y >= image.shape[0]/2 and x < image.shape[1]/2:
                image[y,x] = [image[y,x][2], image[y,x][1], image[y,x][0]] ␣
 ↪#(B,G,R) image
            #Bottom Right
            elif y >= image.shape[0]/2 and x >= image.shape[1]/2:
                r,g,b = image[y,x]
                image[y,x] = 0.21263903 * r + 0.71516871 * g + 0.072192319 * b
    return image
```

1.6.1 1.1 Extract Channels (5 points)

[18]:
```
import copy

def get_channel(img, channel):
    """ Function to extract 2D image corresponding to a channel index from a␣
 ↪color image.
    This function should return a H*W array which is the corresponding channel␣
 ↪of the input image. """
    img = copy.deepcopy(img)     # Create a copy so as to not change the␣
 ↪original image
    #### Your code start here. ####

    return img[:,:,channel]

# print(img.shape)
# print(get_channel(img, 0).shape)
```

```
(414, 414, 3)
(414, 414)
```

1.6.2 1.2 Merge Channels (5 points)

[31]:
```
def merge_channels(img0, img1, img2):
    """ Function to merge three single channel images to form a color image.
    This function should return a H*W*3 array which merges all three single␣
↪channel images
    (i.e. img0, img1, img2) in the input."""
    #### Your code start here. ####
    # Hint: There are multiple ways to implement it.
    #       1. For example, create a H*W*C array with all values as zero and
    #          fill each channel with given single channel image.
    #          You may refer to the "Modify a subarray" section in the brief␣
↪NumPy tutorial above.
    #       2. You may find np.stack() / np.concatenate() / np.reshape() useful␣
↪in this problem.

    return np.stack([img0, img1, img2], axis=2)
```

1.6.3 1.3 Vectorized Implement (5 points)

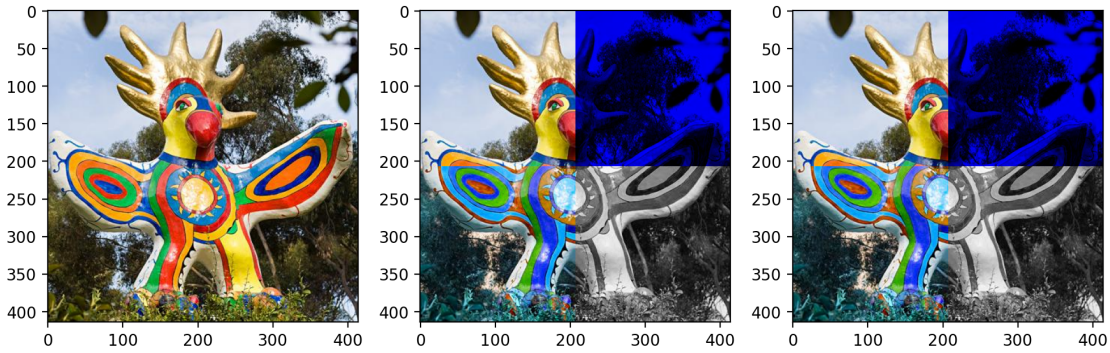
[42]:
```
def vectorized(img):
    """ Vectorized operation. """
    image = copy.deepcopy(img)
    a = int(image.shape[0]/2)
    b = int(image.shape[1]/2)
    # Please also keep the red / green / blue channel respectively in the␣
↪corresponding part of image
    # with the vectorized operations. You need to make sure your final␣
↪generated image in this
    # vectorized() function is the same as the one generated from iterative().

    #### Your code start here. ####

    # gray = 0.21263903 * R + 0.71516871 * G + 0.072192319 * B

    #Top Right: keep the blue channel
    image[:a, b:] = np.stack([np.zeros_like(get_channel(image[:a, b:], 2)), np.
↪zeros_like(get_channel(image[:a, b:], 2)), get_channel(image[:a, b:], 2)],␣
↪axis=2)

    #Bottom Left: (B,G,R) image
    image[a:,:b] = merge_channels(get_channel(image[a:,:b], 2),␣
↪get_channel(image[a:,:b], 1), get_channel(image[a:,:b], 0))

    #Bottom Right: Grayscale image
    gray = np.dot(image[a:, b:, :3], [0.21263903, 0.71516871, 0.072192319])
    image[a:, b:, :3] = np.stack([gray, gray, gray], axis=-1)
```

```
    return image
```

Now, run the following cell to compare the difference between iterative and vectorized operation.

[45]:
```
import time

def compare():
    img = plt.imread('SunGod.jpg')
    cur_time = time.time()
    image_iterative = iterative(img)
    print("Iterative operation (sec):", time.time() - cur_time)

    cur_time = time.time()
    image_vectorized = vectorized(img)
    print("Vectorized operation (sec):", time.time() - cur_time)

    # Note: The shown figures of image_iterative and image_vectorized should be␣
 ↪identical!
    assert np.array_equal(image_iterative, image_vectorized), "The two images␣
 ↪are not identical!"

    return image_iterative, image_vectorized

# Test your implemented get_channel()
assert len(get_channel(img, 0).shape) == 2  # Index 0

# Run the function
image_iterative, image_vectorized = compare()

# Plotting the results in sepearate subplots.
plt.figure(figsize=(12,4))   # Adjust the figure size.
plt.subplot(1, 3, 1)         # Create 1x3 subplots, indexing from 1
plt.imshow(img)              # Original image.

plt.subplot(1, 3, 2)
plt.imshow(image_iterative)  # Iterative operations on the image.

plt.subplot(1, 3, 3)
plt.imshow(image_vectorized) # Vectorized operations on the image.

plt.show()                   # Show the figure.
```

```
Iterative operation (sec): 0.37195897102355957
Vectorized operation (sec): 0.0018358230590820312
```

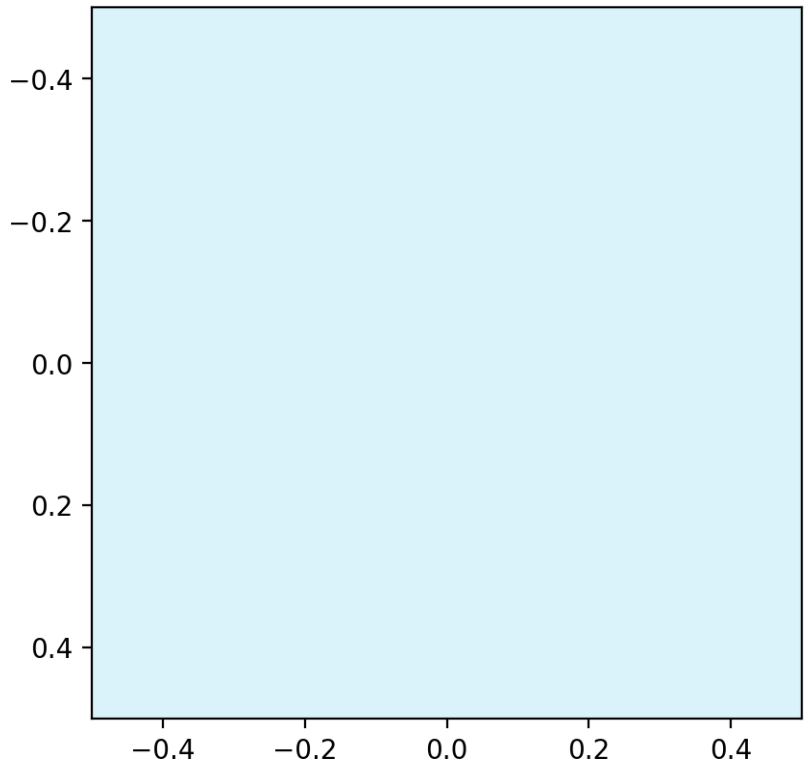


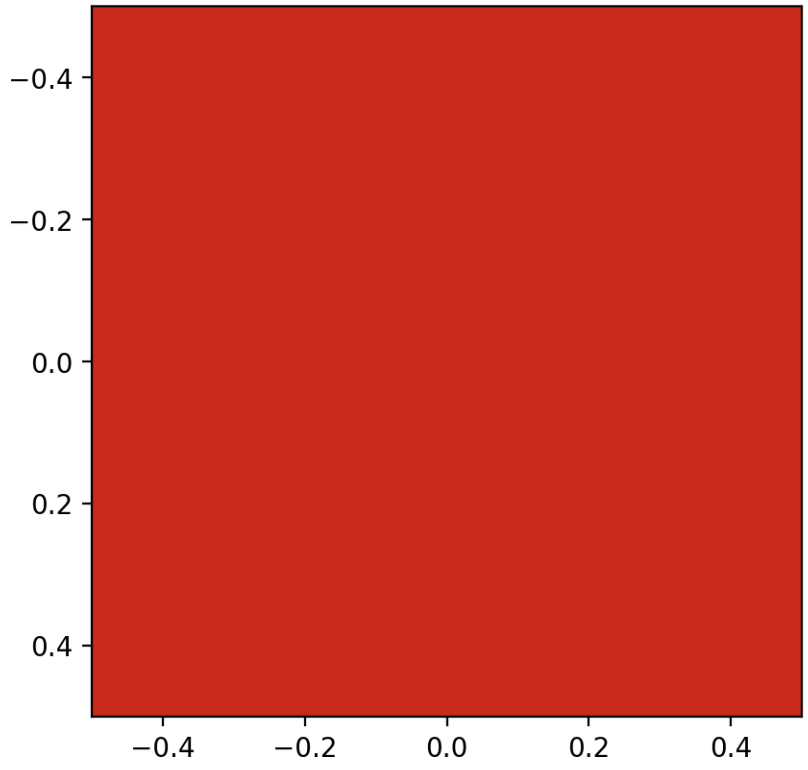
1.7 Problem 2: Coordinates vs Row-col (5 points)

We have already loaded the `SunGod.jpg` image. Its shape in numpy is $H * W * C$. Now get the color values `[r, g, b]` for - 1. the pixel at coordinate $(x, y) = (100, 200)$. The origin is at the top left corner of the image. The positive x axis points to the right (width direction) while the positive y axis points down (height direction). 2. the pixel on the 100^{th} *row* and 200^{th} *column*.

Now call the `show_color` method for these pixels. Are the values/colors the same? Explain the reason for your observation.

[46]:
```
def show_color(pixel):
    """ Function to show the color of a pixel.
    This function takes a 1D array of 3 elements (i.e. a pixel) as input."""
    plt.imshow(pixel.reshape(1,1,3))
    plt.show()

def get_pixel_coord(img, x, y):
    #### Your code start here. ####
    return img[y, x]

def get_pixel_rowcol(img, row, col):
    #### Your code start here. ####
    return img[row, col]

show_color(get_pixel_coord(img, 100, 200))
show_color(get_pixel_rowcol(img, 100, 200))
```


−0.4
−0.2
0.0
0.2
0.4
−0.4
−0.2
0.0
0.2
0.4


−0.4
−0.2
0.0
0.2
0.4
−0.4
−0.2
0.0
0.2
0.4

Write your obersvation here This returns different coordinates because Numpy uses row,col = y,x, but cartesian coordinates are x,y which is why there is a difference.

1.8 Problem 3: More Image Manipulation (30 points)

In this problem you will use the image `usa_map.png`. Being a color image, this image has three channels, corresponding to the primary colors of red, green and blue.

(1) Read the image. (**1 point**)

(2) Write a function to flip the original image from top to bottom. For this function, you **MUST ONLY use Array Indexing** to implement it. To receive credit, you **MUST NOT** use even a single for-loop and you **MUST NOT** use library functions (e.g. `np.flip()`) that directly flips the matrix. (**6 points**)

(3) Next, write another function to rotate the original image 90 degrees **clockwise**. You **MUST ONLY use Array Indexing** to implement this function. To receive credit, you **MUST NOT** use even a single for-loop and you **MUST NOT** use library functions (e.g. `np.rot90()`) that directly rotates the matrix. Display three images, by applying the rotation function once (i.e. 90-degree rotation), twice (i.e. 180-degree rotation) and thrice (i.e. 270-degree rotation). [**HINT:** Can you reuse the previous flip method you wrote?] (**6 points**)

(4) Read the `california_map.png` image and the corresponding `california_map_mask.png` binary mask image. (**1 point**)

(5) Given the `start_x` and `start_y` on the California image indicating the starting position (top-left corner) of the California image, you need to write a function to place just the cutout of the California map at the right position on the USA map. (**Hint: Mask** pixel values of 1 indicate **image** pixels to show.) To receive credit, you **MUST NOT** use even a single for-loop. (**6 points**)

(6) Finally, consider **4 color images** you obtained: 1 original USA map image, 1 from flipping (top to bottom), 1 from rotation (180-degree), and 1 after placing the California cutout on the USA map. (**10 points**)

Now,

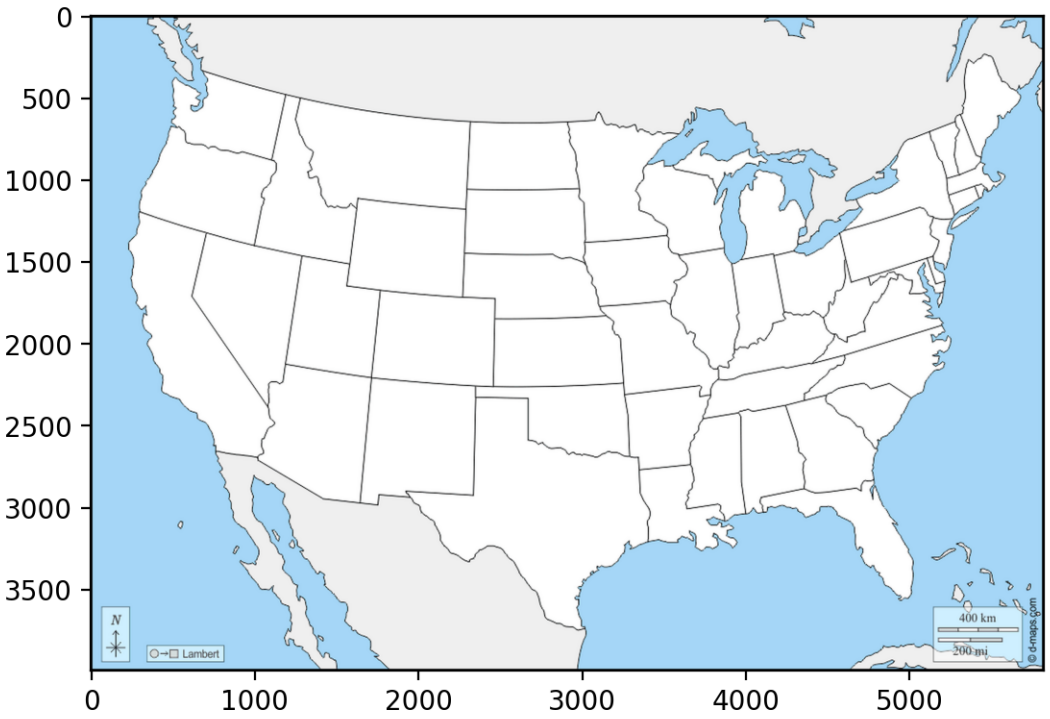
- change the order of the channels of the original world map image from RGB to GBR (you can reuse problem 1 functions),
- from the flipped image remove the green channel,
- from the rotated image, remove the red channel and
- from the final USA coutout image remove the blue channel.

Using these 4 images, create one single image by tiling them together. To receive credit, you **MUST NOT** use any loops. The image will have 2×2 tiles making the shape of the final image $2H \times 2W \times 3$. The order in which the images are tiled does not matter. Show the tiled image.

Note: For all the above tasks, to receive credits, **DO NOT** use any loops.

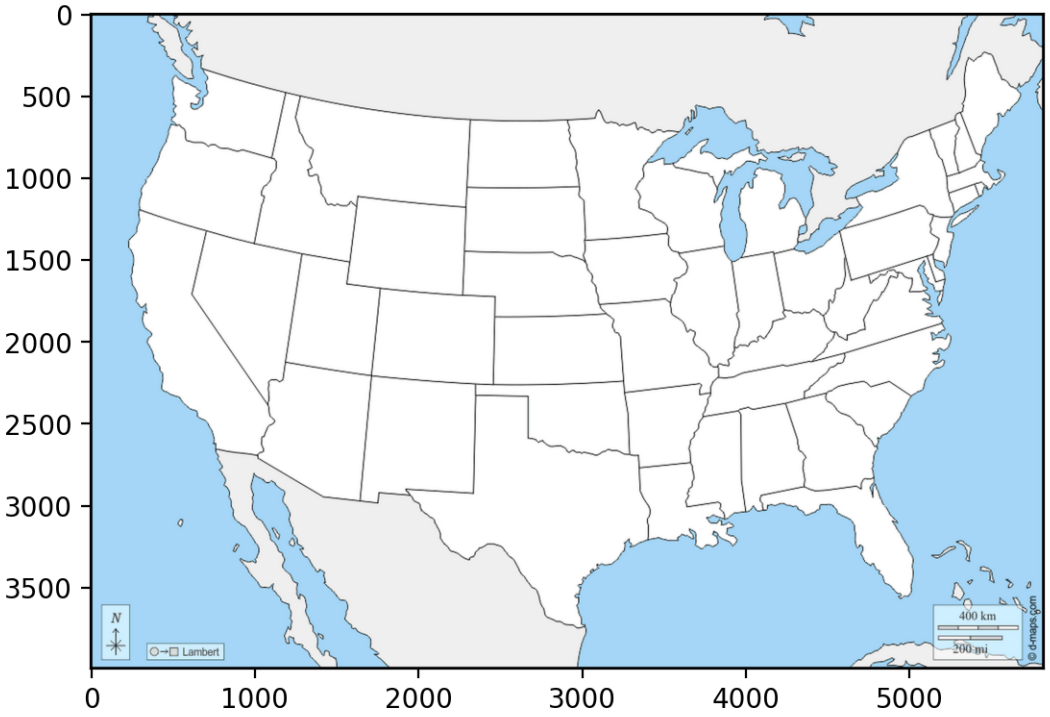
[48]:
```
import numpy as np
import matplotlib.pyplot as plt
import copy
```

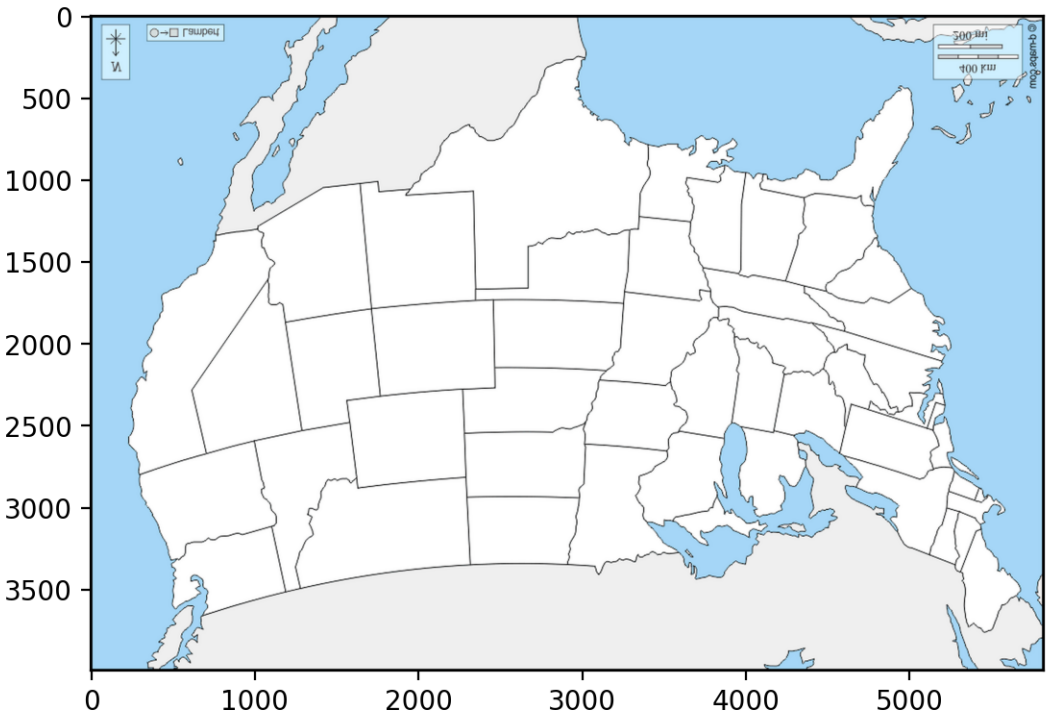
[51]:
```
# (1) Read the image.
#### Write your code here. ####
img = plt.imread('usa_map.png') # Read an image

plt.imshow(img) # Show the image after reading.
plt.show()
```



[50]:
```
# (2) Flip the image from top to bottom.
def flip_img(img):
    """
    Function to mirror image from top to bottom.
    This function should return a H*W*3 array which is the flipped version of␣
 ↪original image.
    """
    #### Write your code here. ####
    img = img.copy()
    return img[::-1]

plt.imshow(img)
```

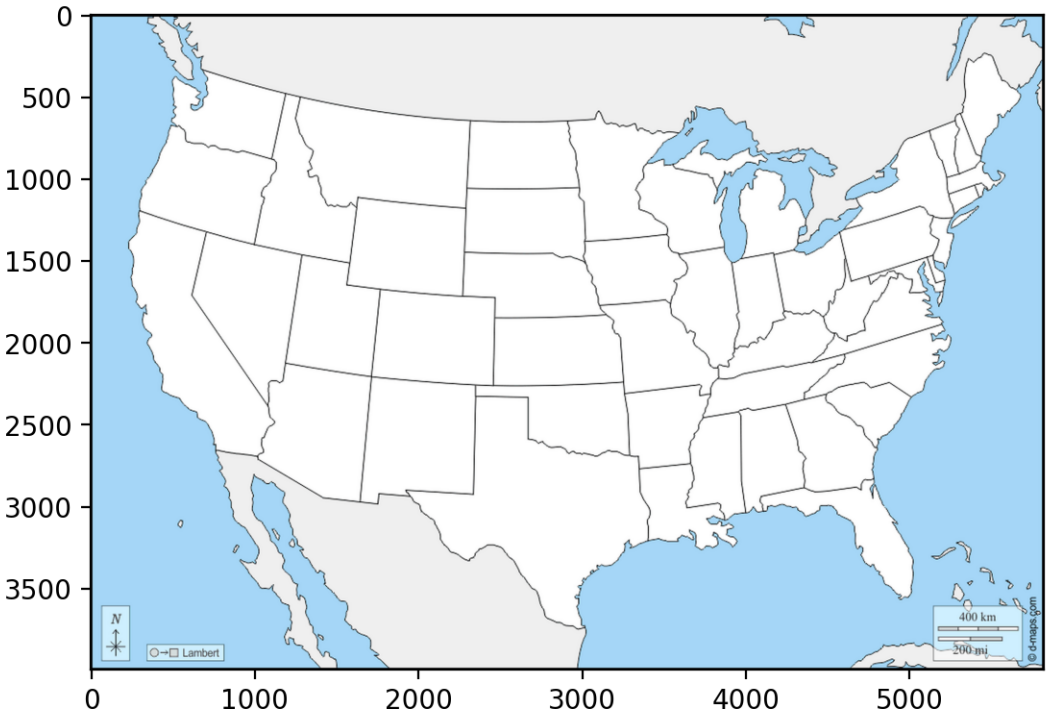
```
plt.show()
flipped_img = flip_img(img)
plt.imshow(flipped_img)
plt.show()
```

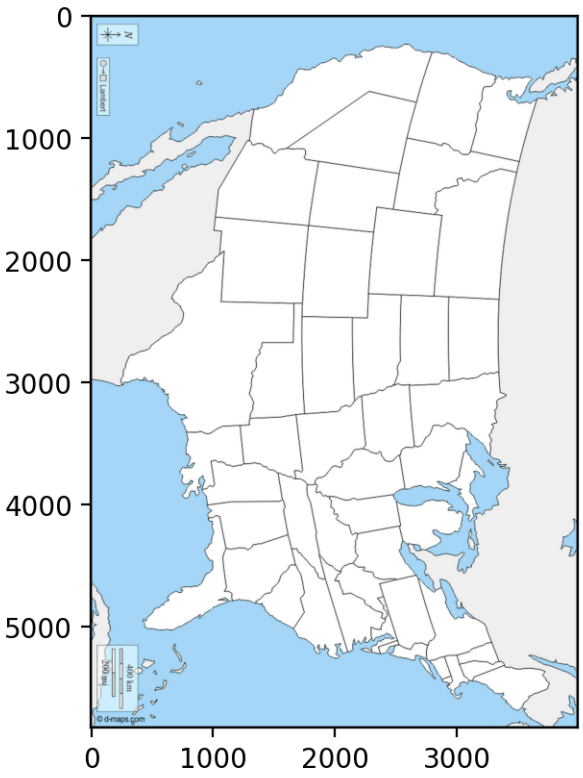


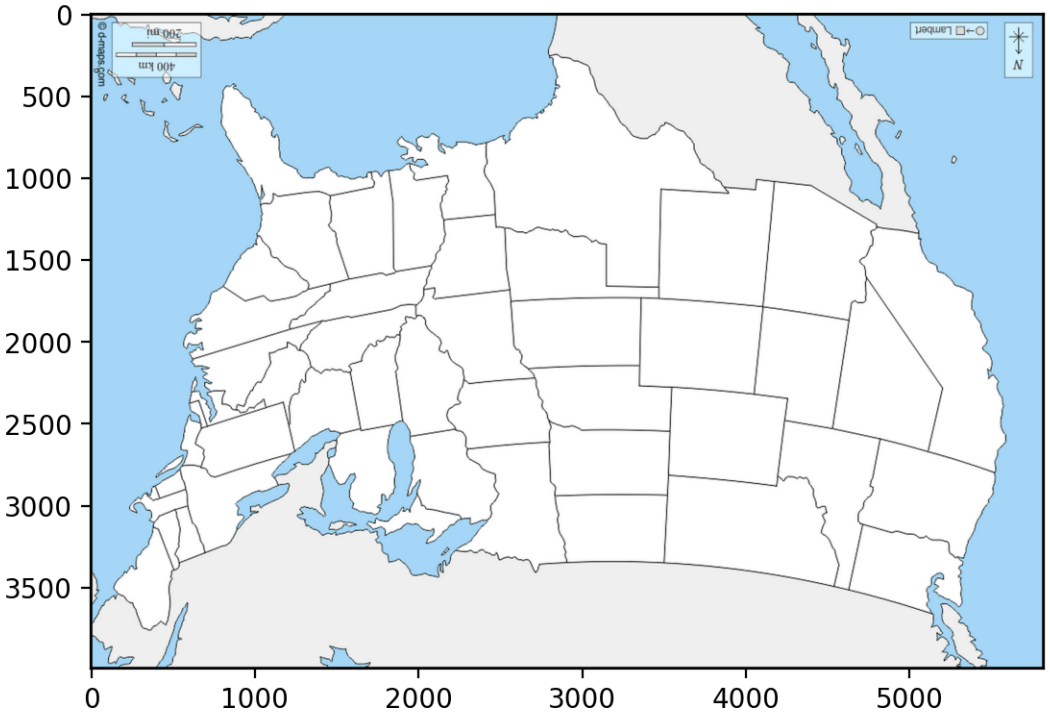


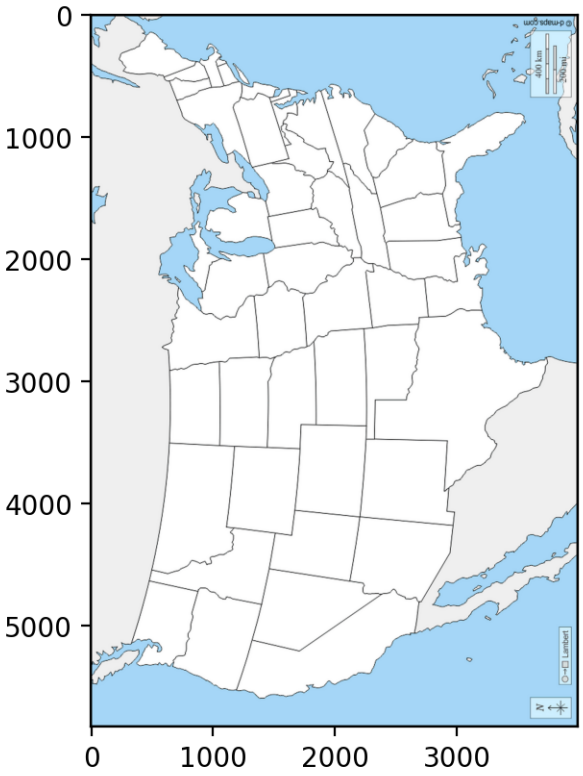
[54]:
```
# (3) Rotate image.
def flip_img_left_to_right(img):
    """
    Function to mirror image from left to right.
    This function should return a H*W*3 array which is the flipped version of␣
 ↪original image.
    """
    #### Write your code here. ####
    img = img.copy()
    return img[:,::-1]

def rotate_90(img):
    """
    Function to rotate image 90 degrees clockwise.
    This function should return a W*H*3 array which is the rotated version of␣
 ↪original image. """
    #### Write your code here. ####
    img = img.copy()
    # rotate 90 degrees clockwise means flip up and down and then transpose
    # have to mind channels

    return  img[::-1, :, :].transpose(1, 0, 2)
```

```
plt.imshow(img)
plt.show()
rot90_img = rotate_90(img)
plt.imshow(rot90_img)
plt.show()
rot180_img = rotate_90(rotate_90(img))
plt.imshow(rot180_img)
plt.show()
rot270_img = rotate_90(rotate_90(rotate_90(img)))
plt.imshow(rot270_img)
plt.show()
```



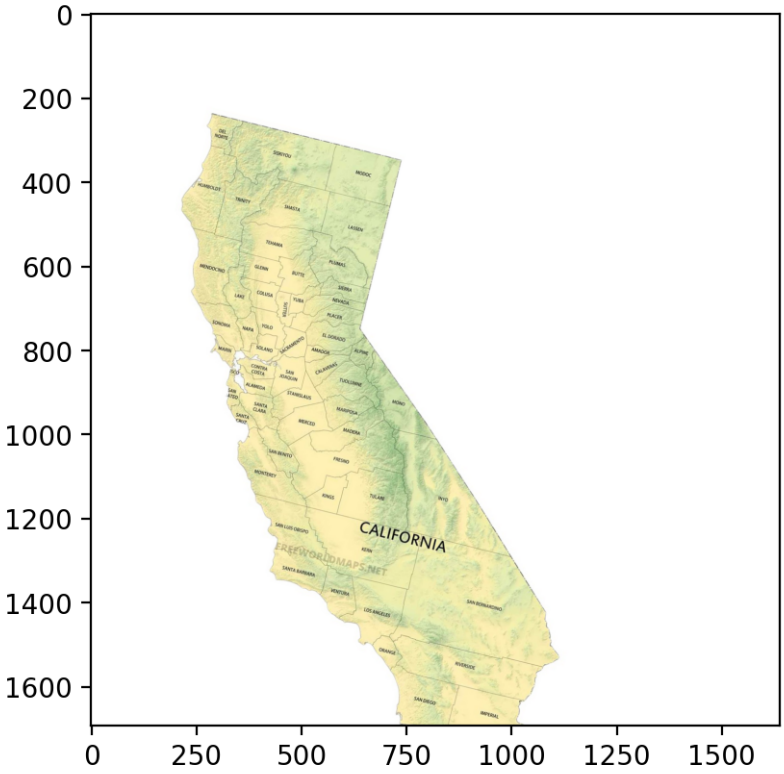

0
1000
2000
3000
4000
5000
0
1000
2000
3000

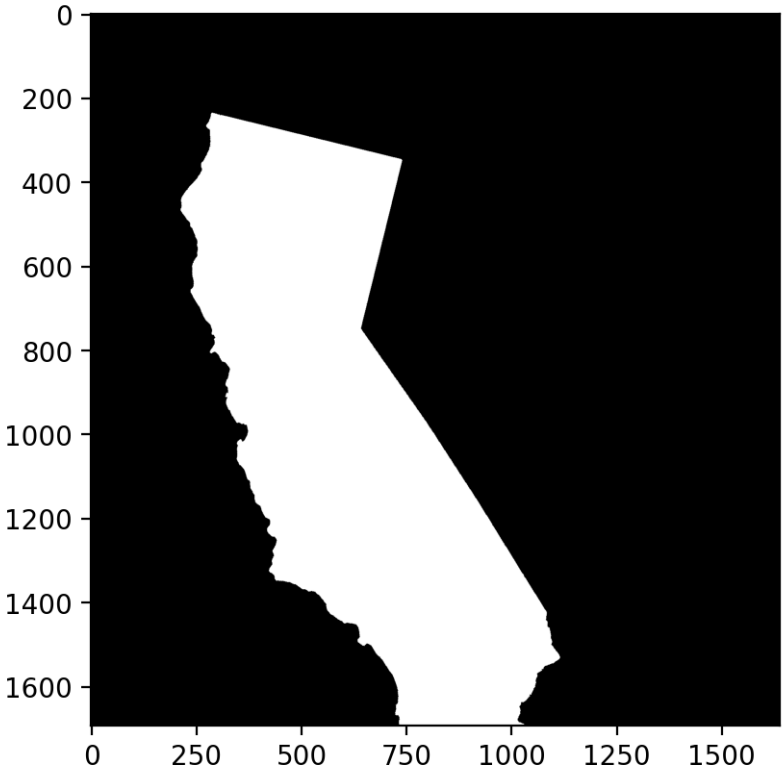

0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000
Lambert
N



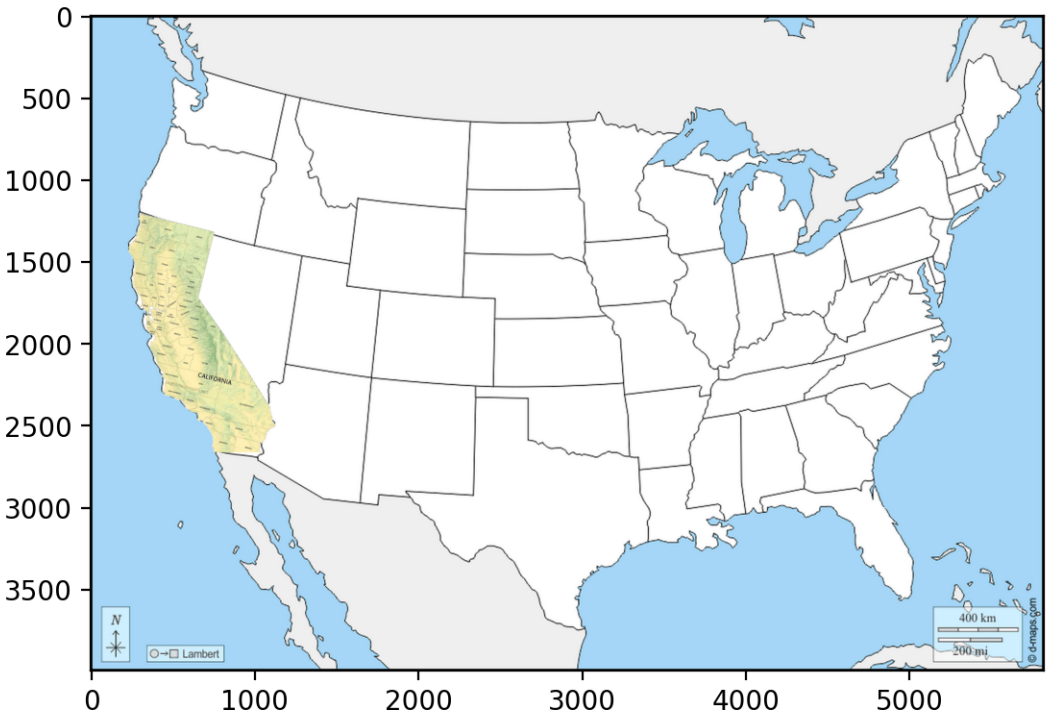
[58]:
```
# (4)Read the usa image and the binary mask image

#### Write your code here. ####

california_img = plt.imread('california_map.png')
bi_mask_img = plt.imread('california_map_mask.png')

print("California Image Size: ")
print(california_img.shape)
print("California Binary Mask Image Size: ")
print(bi_mask_img.shape)

plt.imshow(california_img)
plt.show()
plt.imshow(bi_mask_img)
plt.show()
```

```
California Image Size:
(1692, 1639, 3)
California Binary Mask Image Size:
(1692, 1639, 3)
```

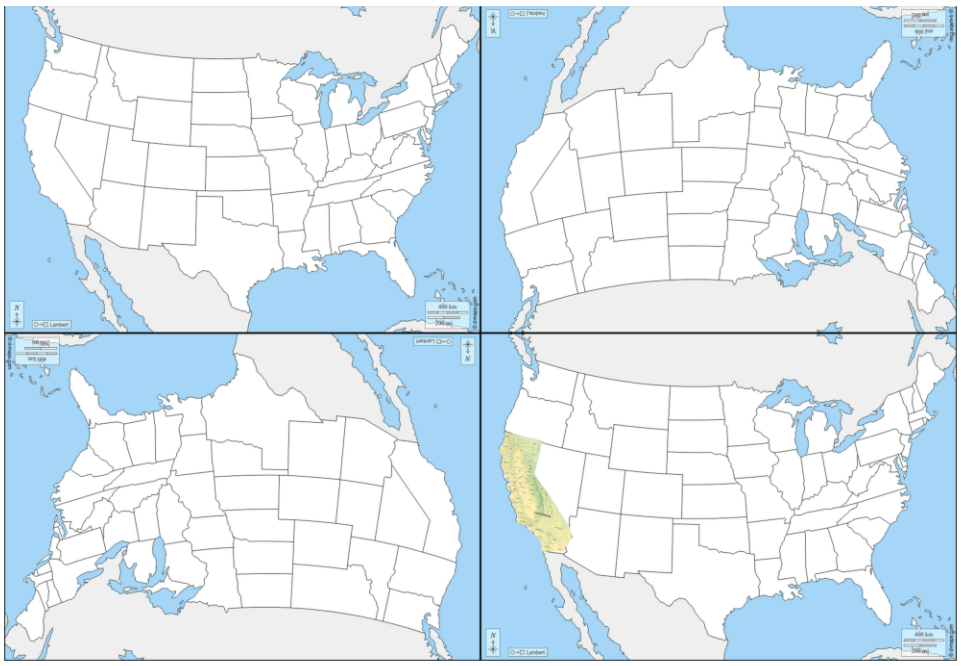

0
200
400
600
800
1000
1200
1400
1600
0
250
500
750
1000
1250
1500
CALIFORNIA


0
200
400
600
800
1000
1200
1400
1600
0
250
500
750
1000
1250
1500

[ ]:
```
# (5) Just place the cutout of the USA at the right position on the world map␣
 ↪(very closely aligns with USA map California state boundary, but not␣
 ↪perfectly)
start_x = 17
start_y = 970

final_img = copy.deepcopy(img)

#### Write your code here. ####

# Given the `start_x` and `start_y` on the California image indicating the␣
 ↪starting position (top-left corner) of the California image,
# you need to write a function to place just the cutout of the California map␣
 ↪at the right position on the USA map.
# (**Hint:** **Mask** pixel values of 1 indicate **image** pixels to show.)
# To receive credit, you **MUST NOT** use even a single for-loop. (**6 points**)


# isoluate the california area on the big map
h, w, c = california_img.shape
roi = final_img[start_y:start_y+h, start_x:start_x+w, :] # acts as a refernence

# if the mask is 1, then place the california image into the area
final_img[start_y:start_y+h, start_x:start_x+w, :] = np.where(
    bi_mask_img == 1,
    california_img,
    roi
)

plt.imshow(final_img)
plt.show()
```



[67]:
```
# (6) Write your code here to tile the four images and make a single image.
# You can use the img, flipped_img, rot180_img, final_img to represent the four␣
 ↪images.
# After tiling, please display the tiled image.
#The shape of the tiled image should be 2 ×2 ×3

#### Write your code here. ####

# Top row is original and flipped and Bottom row is 180 rotated and final
top_row = np.hstack((img, flipped_img))
bottom_row = np.hstack((rot180_img, final_img))

# Combine for final title image
tiled_img = np.vstack((top_row, bottom_row))

# Display
plt.imshow(tiled_img)
plt.axis('off')
plt.show()


print(img.shape)
print(tiled_img.shape)
```



```
(3993, 5824, 3)
(7986, 11648, 3)
```