CS 3430: SciComp with Py
Assignment 12
Linear Programming in 2 Variables and Image Histograms in RGB and HSV Spaces

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 14, 2019

Learning Objectives

1. Linear Programming in 2 Variables
2. Image Histograms
3. Histogram Similarity Functions
4. Image Indexing and Retrieval

Introduction

In this assignment, we'll solve several 2D optimization problems with linear programming and use RGB and HSV histograms to index and retrieve images.

Problem 1 (3 points)

Let's implement a few tools to minimize/maximize linear 2D functions subject to several linear constraints. To begin with, we need to represent linear equations. We'll do it with a class that splits a linear equation into two function expressions: one on the left side of the equation (left-hand side or lhs) and one on the right side of the equation (right-hand side or rhs). This class is implemented in `line_eq.py`.

```
class line_eq(object):
    def __init__(self, lhs=None, rhs=None):
        self.__lhs__ = lhs
        self.__rhs__ = rhs

    def get_lhs(self):
        return self.__lhs__

    def get_rhs(self):
        return self.__rhs__

    def __str__(self):
        return str(self.__lhs__) + ' = ' + str(self.__rhs__)
```

To save ourselves some typing, let's define a maker function and save it in `maker.py`.

```
def make_line_eq(lhs, rhs):
    return line_eq(lhs=lhs, rhs=rhs)
```

Implement the function `line_intersection(fexpr1, fexpr2)` that takes two function expressions of line equations (i.e., `line_eq` objects) and returns a 2D point object (see `point2d.py`) representing the intersection point of 2 lines or `None` if the lines do not intersect. Save your implementation in `linprog.py`. Let's do a few tests.

Test 01

Let's intersect $y = 1$ and $x = 1$.

```
>>> ln1 = make_line_eq(make_var('y'), make_const(1.0))
>>> ln2 = make_line_eq(make_var('x'), make_const(1.0))
>>> p = line_intersection(ln1, ln2)
>>> print(p.get_x())
1.0
>>> print(p.get_y())
1.0
>>> print(p)
(1.0, 1.0)
```

Test 02

Let's intersect $y = 2$ and $y = x - 6$.

```
>>> ln1 = make_line_eq(make_var('y'), make_const(2.0))
>>> ln2 = make_line_eq(make_var('y'), make_plus(make_pwr('x', 1.0),
                                                make_const(-6.0)))
>>> print(line_intersection(ln1, ln2))
(8.0, 2.0)
>>> print(line_intersection(ln2, ln1))
(8.0, 2.0)
```

Test 03

Let's intersect $y = -2$ and $y = x + 10$.

```
>>> ln1 = make_line_eq(make_var('y'), make_const(-2.0))
>>> ln2 = make_line_eq(make_var('y'), make_plus(make_pwr('x', 1.0),
                                                  make_const(10.0)))
>>> print(line_intersection(ln1, ln2))
(-12.0, -2.0)
>>> print(line_intersection(ln2, ln1))
(-12.0, -2.0)
```

Test 04

Let's intersect $y = x$ and $y = -x + 6$.

```
>>> ln1 = make_line_eq(make_var('y'), make_pwr('x', 1.0))
>>> print(ln1)
y = (x^1.0)
>>> ln2 = make_line_eq(make_var('y'), make_plus(make_prod(make_const(-1.0),
                                                            make_pwr('x', 1.0)),
                                                  make_const(6.0)))
>>> print(ln2)
y = ((-1.0*(x^1.0))+6.0)
>>> print(line_intersection(ln1, ln2))
(3.0, 3.0)
>>> print(line_intersection(ln2, ln1))
(3.0, 3.0)
```

Test 05

Let's intersect between $y = -0.2x + 10$ and $y = 0.2x + 5$.

```
>>> ln1 = make_line_eq(make_var('y'), make_plus(make_prod(make_const(-1.0/5.0),
                                                            make_pwr('x', 1.0)),
                                                  make_const(10.0)))
>>> ln2 = make_line_eq(make_var('y'), make_plus(make_prod(make_const(1.0/5.0),
                                                            make_pwr('x', 1.0)),
                                                  make_const(5.0)))
>>> print(ln1)
y = ((-0.2*(x^1.0))+10.0)
>>> print(ln2)
y = ((0.2*(x^1.0))+5.0)
>>> print(line_intersection(ln1, ln2))
```

```
(12.5, 7.5)
>>> print(line_intersection(ln2, ln1))
(12.5, 7.5)
>>> ln1f = tof(ln1.get_rhs())
>>> p = line_intersection(ln1, ln2)
>>> ln1f(p.get_x().get_val())
7.5
>>> ln2f = tof(ln2.get_rhs())
>>> ln2f(p.get_x().get_val())
7.5
>>> ln2f(p.get_x().get_val()) == p.get_y().get_val()
True
>>> ln1f(p.get_x().get_val()) == p.get_y().get_val()
True
```

Test 06

Let's intersect $x = 1$ and $y = 0.5x$.

```
>>> ln1 = make_line_eq(make_var('x'), make_const(1.0))
>>> ln2 = make_line_eq(make_var('y'), make_prod(make_const(0.5),
                                                make_pwr('x', 1.0)))
>>> print(line_intersection(ln1, ln2))
(1.0, 0.5)
```

Implement the functions `maximize_obj_fun(f, corner_points)` and `minimize_obj_fun(f, corner_points)` that take a Python function `f` and an array of corner points, each corner point being a `poind2d` object, and a corner point from `corner_points` (i.e., a `point2d` object) where `f` achieves its maximum/minimum value and a `const` object representing that value. Save your implementations in `linprog.py`. Let's do a couple of quick tests.

```
>>> f1 = lambda x, y: 2*x + y
>>> corner_points = [make_point2d(1, 1),
                     make_point2d(1, 5),
                     make_point2d(5, 1)]
>>> p, maxv = maximize_obj_fun(f1, corner_points)
>>> isinstance(p, point2d)
True
>>> isinstance(maxv, const)
True
>>> print(p)
(5, 1)
>>> print(maxv)
11
>>> f2 = lambda x, y: x - 2*y
>>> p, minv = minimize_obj_fun(f2, corner_points)
>>> print(p)
(1, 5)
>>> print(minv)
-9
```

Use `line_intersection()`, `maximize_obj_fun()`, and `minimize_obj_fun()` to solve the following optimization problems. Save your solutions in the functions `opt_prob_1a()`, `opt_prob_1b()`, and `opt_prob_1c()` in `linprog.py`. In the body of each of these functions, you should write code to identify the corner points and then return the values from `maximize_obj_fun()` or `minimize_obj_fun()`. In a comment before each function, write your answer to each problem in the format $x, y, minv/maxv$.

- **Problem 1a:** Maximize $2x + y$ that satisfies $x \geq 1$, $y \geq 1$, $x \leq 5$, $y \leq 5$, and $x + y \leq 6$.
- **Problem 1b:** Minimize $x/2 + y$ that satisfies $y \geq 2$, $x \geq 0$, $x \geq y$, $x + y \leq 6$.
- **Problem 1c:** Maximize $3x - 2y$ that satisfies $x + y \geq 0$, $x - y \leq 0$, $-2x + 4y \leq 5$.

Problem 2 (2 points)

I posted the following zip archives under Canvas announcements for Assignment 12. The archives are:

- `hist_img_01.zip`;
- `hist_img_02.zip`;

- hist_img_03.zip;
- hist_img_04.zip;
- hist_img_05.zip;
- hist_img_06.zip;
- hist_img_07.zip;
- hist_test.zip;

After you download them, place them into the folder `images` on your computer. There should 318 images of various streets in Logan as well as some lunch tray images from a Logan school. I received the latter from Dr. Heidi Wengreen, a USU nutrition professor, with whom I collaborated on a food texture recognition project a couple of years ago. We'll use the images in the directory `images` for persistent indexing (aka pickling). The `hist_test.zip` contains two image directories: `car_test` and `food_test`. We'll use these for image retrieval. It is possible to compute RGB, HSV, and grayscale histograms. We won't use grayscale histograms in this assignment.

Indexing Images

Implement the function `hist_index_img`. This function takes an image path `imgp`, a string specification of the color space (this can be either `'rgb'` or `'hsv'`), and the bin size, i.e., the number of bins in each color channel in the histogram (we'll use 8 and 16 in this assignment).

The function `hist_index_img` places the key-value pair `(imgp, norm_hist)` in the dictionary `HIST_INDEX`. Depending on the value of `color_space`, the `norm_hist` is either the normalized and flattened RGB histogram of the image in `imgp` or the normalized and flattened HSV histogram of the same image. We'll discuss how to flatten histograms in lecture 25. Save your implementation in `hist_image_index.py`. Let's do a test.

```
def test_01():
  HIST_INDEX = {}
  hist_index_img_dir(IMGDIR, 'rgb', 8, 'rgb_hist8.pck')

>>> test_01()
../images/
indexing ../images/16_07_02_14_23_48_orig.png
../images/16_07_02_14_23_48_orig.png indexed
indexing ../images/16_07_02_15_18_56_orig.png
...
indexing ../images/16_07_02_14_50_47_orig.png
../images/16_07_02_14_50_47_orig.png indexed
indexing finished
```

Calling `test_01()` creates a pickle file `rgb_hist8.pck` in your current working directory, which is the persisted version of the dictionary `HIST_INDEX`. There are three other tests in `hist_image_index.py` to create `rgb_hist16.pck`, `hsv_hist8.pck`, and `hsv_hist16.pck` in a similar fashion.

Retrieving Images

Once you have persisted the dictionary, you can load it into Python as follows:

```
def load_hist_index(pick_path):
  with open(pick_path, 'rb') as histfile:
    return pickle.load(histfile)

>>> hist_index = load_hist_index('rgb_hist8.pck')
>>> len(hist_index)
>>> 318
```

Implement the function `find_sim_rgb_images(imgpath, bin_size, hist_index, hist_sim)` that finds and displays the top 3 images most similar to the image in `imgpath` by computing the similarity scores between this image and all images in a given histogram dictionary `hist_index`. The parameter `bin_size` should be 8 or 16. Use the following strings as values of `hist_sim` (i.e., the histogram similarity metric): `correl` – for correlation, `chisqr` – for chi square, `inter` – for intersection, and `bhatta` – for bhattacharrya. Lots more on these metrics in the upcoming lecture 25. The top three images and the input image are displayed in four separate matplotlib figures along with their similarity scores. The similarity scores are also printed in the Python shell. This function computes a list of 2-tuples of the form (path, sim_score), where sim_score is the similarity score computed with hist_sim between the image in imgpath and the image in path. This list of 2-tuples (i.e., the match list) is then sorted from lowest to highest or from highest to lowest similarity score, depending on the value of hist_sim, because for some similarity metrics the higher the better, for others – vice versa. The list of the top 3 matches is returned.

Let's do a test where we use the 8-bin rgb index to find the top 3 similar images by using the histogram intersection similarity function (i.e., cv2.HISTCMP_INTERSECT).

```
def test_01():
  hist_index = load_hist_index(PICDIR + 'rgb_hist8.pck')
  assert len(hist_index) == 318
  imgpath = IMGDIR + 'food_test/img01.JPG'
  inimg = cv2.imread(imgpath)
  top_matches = find_sim_rgb_images(imgpath,
                  8, hist_index, 'inter')
  for imagepath, sim in top_matches:
    print(imagepath + ' --> ' + str(sim))
  show_images(inimg, top_matches)
  del hist_index
```

Here is the output in the Py shell.

```
images/123461762.JPG --> 2.69072864504
images/123465049.JPG --> 2.63319342056
images/123472255.JPG --> 2.43531483644
```

The figures of the input and top 3 matches are in Figures ?? and ??. To show these figures, you should implement the function show_images(imgpath, match_list) that takes the path to the image you're matching and the match list returned by find_sim_rgb_images() or find_sim_hsv_images() (see below).

Figure 1: Input image (Figure 1) and first match image (Figure 2)

Implement the function find_sim_hsv_images(imgpath, bin_size, hist_index, hist_sim) that behaves in the same way as find_sim_rgb_images(), except hist_index is an HSV dictionary. More tests are in hist_index_retrieval.py

What To Submit

You have to submit linprog.py, hist_image_index.py, and hist_image_retrieval.py. You should also submit all the files needed to run your code (i.e., line_eq.py, maker.py, etc.). Zip your entire working directory in hw12.zip and

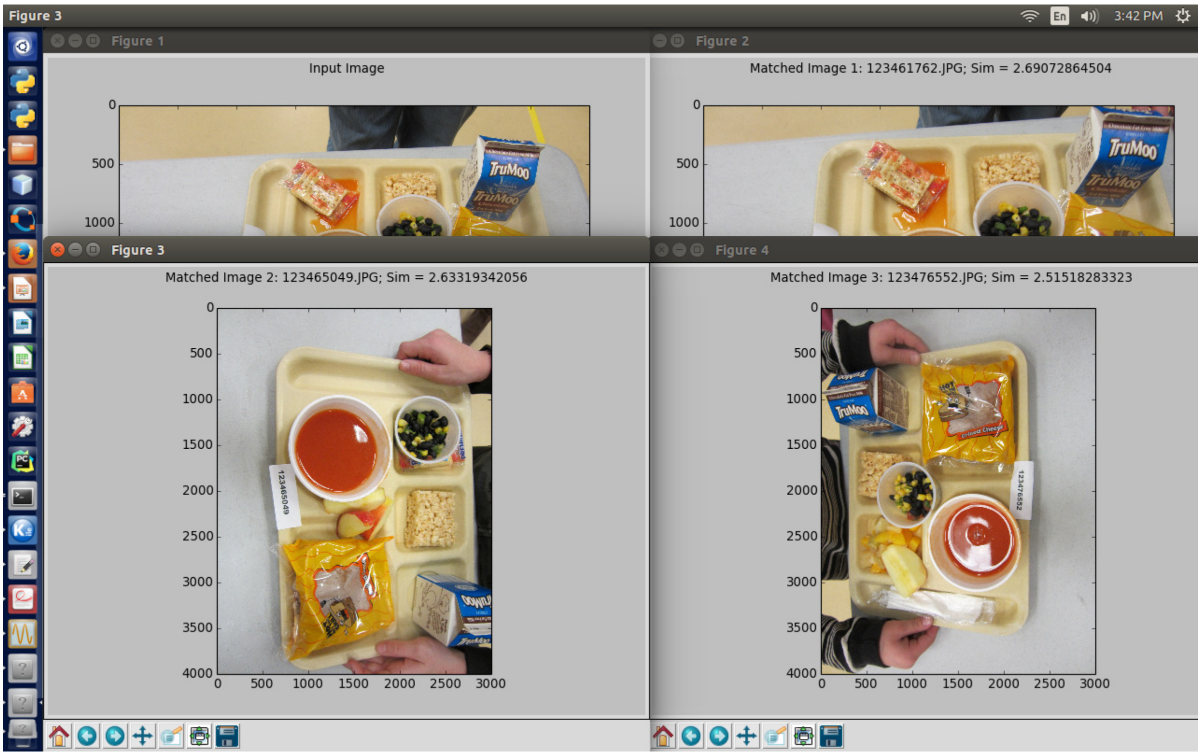

Figure 2: Second match image (Figure 3) and third match image (Figure 4)

submit it via Canvas.

My perpetual refrain! Do not change the names of the files that were given to you or the names of the functions you are asked to implement. The unit tests that I write for the grader every week depend on these names remaining the same.

Happy Hacking!