CS 3430: S19: SciComp with Py
Exam 3

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 25, 2019

Introduction

1. This exam has 5 problems worth a total of 15 points; it is open books, open notes, which means that it is perfectly fine for you to look at the lecture slides, the reading handouts, or your class notes; it is also OK to consult online documentation at `www.python.org`; it is **not OK** to go to `stackoverflow` or any other technical forum and copy and paste someone else's code – this is plagiarism and, if discovered, will be treated as such; you have to do your own work.

2. You must use your source code from the assignments; you may not use any third-party libraries; the imports of standard Python libraries (e.g., `math`, `numpy`, `matplotlib`, `scipy`) and of the libraries we extensively used in this course (i.e., `cv2` and `PIL/PILLOW`) are fine.

3. You may not talk to anyone during this exam orally, digitally, or in writing.

4. You have 2 hours to complete this exam; the exam is due on Canvas at 5:00pm today (04/25/19), which means that **the submission will close at 4:59:59pm**. I will use the formula $p(t) = 2^t$ to deduct points for late submissions, where t is the number of minutes past 5:00pm. (2 points for 1 minute, 4 - for 2 minutes, etc.) This is a firm deadline. You are always better off submitting something on time for partial credit.

5. The file `cs3430_s19_exam_03.py` is the file where you will write and save all your solutions. It has some starter code for you. You must also submit all the files needed to run your functions in `cs3430_s19_exam_03.py`. The safest thing for you to do is to zip your whole directory into `exam_03.zip` and upload it to Canvas.

6. Most problems on this exam are short conceptual or multi-choice questions. When I ask you to write 2-5 sentences, I really mean it. When I read your answers, I'll be looking for specific keywords. If I see them, you'll get full credit. If I don't see them, it doesn't matter how much you've written. So, don't cover lack of knowledge/understanding with excessive verbiage. If a question is multi-choice, all you need to do is to type one (and only one!) integer specifying your choice. You should type all your answers to conceptual questions as multi-line comments in `cs3430_s19_exam_03.py`.

7. If a problem is conceptual or multi-choice (i.e., requires no coding), it has the phrase **no coding** next to it; if a problem requires coding, it has the phrase **coding** next to it.

8. Do not change the names of the functions in `cs3430_s19_exam_03.py`.

9. Relax, focus, and do your best! I wish you all best of luck and, as always, Happy Hacking! And, for this exam, Happy Thinking!

Problem 1 (3 points)

Problem 1.1: (0.3 point; no coding)

You have a directory of binary images. Your task is to implement an image search engine that converts the images into feature vectors and then finds the top n matches closest to a given input image. Would you use Jaccard similarity to find the closest images? You must answer this question as YES or NO (but not both!) and use 2-5 sentences to explain your answer. In other words, if your answer is YES, justify it briefly. If your answer is NO, justify it and give an alternative similarity metric you would use.

Type and save your solution in a multiline comment under the section Problem 1.1 in `cs3430_s19_exam_03.py`.

Problem 1.2: (0.3 point; no coding)

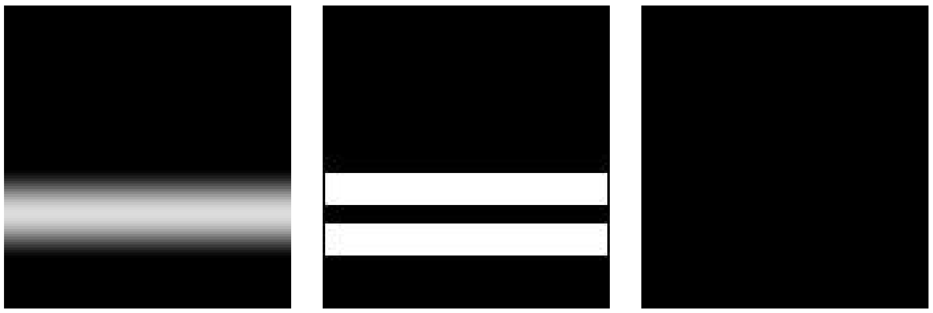

Figure 1: Edge detection: original image (left); gradient edge detection (middle); Canny edge detection (right).

Consider the images in Figure 1. The left image in this figure is an original image with a horizontal line. The middle image shows the edges (white pixels) detected by the gradient edge detection algorithm discussed in lectures 19, 20 and implemented in assignment 10. The right image shows the edges detected by Canny. Explain (no more than 2-5 sentences) why Canny does not detect any edges in the original image while the gradient edge detection algorithm does.

Type and save your solution in a multiline comment under the section Problem 1.2 in `cs3430_s19_exam_03.py`.

Problem 1.3: (0.2 point; no coding)

Consider the gradient edge detection algorithm discussed in lectures 19, 20 and implemented in assignment 10. Briefly (2-5 sentences) describe how one can make this algorithm run faster.

Type and save your solution in a multiline comment under the section Problem 1.3 in `cs3430_s19_exam_03.py`.

Problem 1.4: (0.2 point; no coding)

Consider the Hough Transform line detection algorithm discussed in lectures 21, 22 and implemented in assignment 11. Briefly (2-5 sentences) describe how one can make this algorithm run faster.

Type and save your solution in a multiline comment under the section Problem 1.4 in `cs3430_s19_exam_03.py`.

Problem 1.5: (0.5 point; no coding)

Your task is to write an algorithm to distinguish different textures. Briefly (2-5 sentences) state which CVIP techniques covered in this class you would use to implement it and why.

Type and save your solution in a multiline comment under the section Problem 1.5 in `cs3430_s19_exam_03.py`.

Problem 1.6: (0.5 point; no coding)

Your task to to write an algorithm that finds all instances of an object in a larger image (e.g., all tennis balls in a tennis court photo). You're thinking of using color histograms. Briefly (2-5 sentences) explain when your approach will work and when it will not.

Problem 1.7: (1 point; coding)

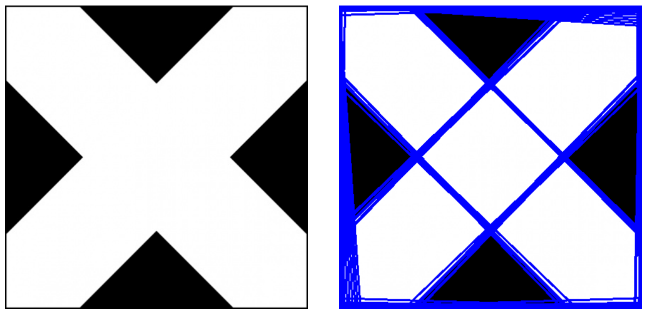

Figure 2: Line detection: original image (left); image with lines drawn.

Write the function `detect_lines(impath1, impath2, magn_thresh=20, spl=20)` that reads an image saved in `impath1`, uses the Hough Transform algorithm to detect lines in it at the gradient magnitude threshold `magn_thresh` and the support level threshold `spl`, and saves in `impath2` the image with all detected lines drawn. You may use your implementations of gradient edge detection and Hough Transform from assignments 10 and 11. You may not use OpenCV for edge detection or Hough Transform. You may use OpenCV to draw lines in the resulting image with whatever color you want so long as they are clearly seen. Figure 2 shows an original image and the same image with the lines drawn blue and saved as `cross_ln.png` with the following call. The exam's zip contains a couple of small images you can use for quick tests.

```
>>> detect_lines('cross.png', 'cross_ln.png', magn_thresh=20, spl=100)
```

Save your implementation of the function `detect_lines()` in `cs3430_s19_exam_03.py`.

Problem 2 (3 points)

Problem 2.1 (0.5 point; no coding)

A fertilizer company makes three types of fertilizers: 20-8-8 for lawns, 4-8-4 for gardens, and 4-4-2 for general purposes. The numbers in each brand (i.e., x-y-z) refer to the weight percentage of nitrate, phosphate, and potash, respectively. The company has 6,000 pounds of nitrate, 10,000 pounds of phosphate, and 4,000 pounds of potash to use. The profit is \$3 per 100 pounds of lawn fertilizer, \$8 per 100 pounds of garden fertilizer, and \$6 per 100 pounds of general-purpose fertilizer. Formulate an LP problem for the numbers of pounds of each fertilizer that should be produced to maximize the company's profit by clearly defining the objective function and the constraints. You do not have to solve this problem, just formulate it.

Type and save your solution in a multiline comment under the section Problem 2.1 in `cs3430_s19_exam_03.py`

Problem 2.2 (0.5 point; no coding)

Suppose that F is the non-empty feasible set for an LP problem and f is the objective function. Let F be bounded. Which one of the following statements is true?

1. f takes arbitrarily large negative values on F;
2. If f is maximized/minimized, it achieves its maximum/minimum value at a corner point;
3. f takes arbitrarily large positive values on F;
4. f achieves its maximum/minimum value on the complement of F.

Type and save exactly one number as your solution in a multiline comment under the section Problem 2.2 in `cs3430_s19_exam_03.py`

Problem 2.3 (0.5 point; no coding)

Suppose that F is the non-empty feasible set for an LP problem and f is the objective function. Let F be unbounded. Suppose that f is to be maximized. Which one of the following statements is true?

1. f achieves its maximum/minimum value on the complement of F.
2. f takes arbitrarily large positive values on F;
3. f takes arbitrarily large negative values on F;

4. f attains its maximum at a corner point or f takes arbitrarily large positive values on F;
5. f attains its maximum at a corner point or f takes arbitrarily large negative values on F;

Type and save exactly one number as your choice in a multiline comment under the section Problem 2.3 in `cs3430_s19_exam_03.py`

Problem 2.4 (0.5 point; no coding)

Suppose that F is the non-empty feasible set for an LP problem and f is the objective function. Let F be unbounded. Suppose that f is to be minimized. Which one of the following statements is true?

1. f takes arbitrarily negative values on F;
2. f attains its minimum at a corner point or f takes arbitrarily large positive values on F;
3. f takes arbitrarily positive values on F;
4. f attains its minimum at a corner point or f takes arbitrarily large negative values on F;
5. f achieves its maximum/minimum value on the complement of F.

Type and save exactly one number as your choice in a multiline comment under the section Problem 2.4 in `cs3430_s19_exam_03.py`

Problem 2.5 (1 point; coding)

Use your implementations of `maximize_obj_fun()` and `line_intersection()` from assignment 12 to solve the following LP problem. Maximize $f(x, y) = x + y$ satisfying the following constraints:

- $x \geq 0$;
- $y \geq 0$;
- $x + 2y \geq 6$;
- $x - y \geq -4$;
- $2x + y \leq 8$.

Save your solution in the function `problem_2_5()` in `cs3430_s19_exam_03.py`. This function returns a 3-tuple (x, y, z), where x and y are the coordinates of the corner point where the objective function attains its maximum value of z.

Problem 3 (3 points)

Use `numpy` functions to solve the following problems.

Problem 3.1 (0.5 point; coding)

Solve the following linear system and save your solution in the function `problem_3_1()` in `cs3430_s19_exam_03.py`. The function `problem_3_1()` returns the 1D numpy array `x` that solves this linear system.

$$\begin{aligned} x + y + 2z &= 9 \\ 2x + 4y - 3z &= 1 \\ 3x + 6y - 5z &= 0 \end{aligned}$$

Problem 3.2 (1 point; coding)

Solve the linear system in problem 3.1 using Cramer's rule and save your solution in the function `problem_3_2()` in `cs3430_s19_exam_03.py`. This function returns the 1D numpy vector `x` that solves this linear system.

Problem 3.3 (1.5 points; coding)

A *proper* factor of a whole number n is a whole number d that divides n with no remainder and is strictly less than n. For example, the proper factors of 8 are 1, 2, and 4. Two integers x and y are called *satyamitra* numbers if the sum of the proper factors of x is equal to y and the sum of the proper factors of y is equal to x. For example, 220 and 284 are satyamitra numbers.

Implement the function `satyamitra_numbers_in_range(lower, upper)` that computes all pairs of satyamitra numbers in the range from `lower` to `upper` where both `lower` and `upper` are positive integers and `lower` $\leq$ `upper`. More specifically, a call to this function returns a list of pairs (i.e., 2-tuples) $[(x_1, x_2), (x_3, x_4), ..., (x_{k-1}, x_k)]$, where, in each pair (x_i, x_j), x_i and x_j are satyamitra numbers such that `lower` $\leq x_i \leq$ `upper` and `lower` $\leq x_j \leq$ `upper`. Here is a sample call.

```
>>> satyamitra_numbers_in_range(0, 2000)
[(220, 284), (1184, 1210)]
```

Write the function `satyamitra_matrix(sn_list)` that takes a list of satyamitra number pairs that contains n numbers, where n is a perfect square (recall that a perfect square is an integer that is the product of two equal integers, e.g., $4 = 2 \cdot 2$, $9 = 3 \cdot 3$, $16 = 4 \cdot 4$, etc.) and creates an $n \times n$ numpy matrix out of these pairs. For example, if `sn_list` $= [(x_1, x_2), (x_3, x_4)]$, then `satyamitra_matrix(sn_list)` creates this matrix

$$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}.$$

If `sn_list` $= [(x_1, x_2), (x_3, x_4), ..., (x_8, x_9)]$, then `satyamitra_matrix(sn_list)` creates this matrix

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}.$$

If `sn_list` $= [(x_1, x_2), (x_3, x_4), (x_5, x_6), ..., (x_{15}, x_{16})]$, the matrix looks like

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix}.$$

Here is a concrete example.

```
>>> satyamitra_matrix(satyamitra_numbers_in_range(0, 2000))
array([[ 220,  284],
       [1184, 1210]])
```

Use these functions to write the boolean function `is_satyamitra_matrix_singular()` that determines if the matrix composed of the satyamitra numbers in the range [1, 20000] (from 1 to twenty thousand) is singular. Save your implementations of `satyamitra_numbers_in_range()`, `satyamitra_matrix()`, and `is_satyamitra_matrix_singular()` in `cs3430_s19_exam_03.py`.

Problem 4 (3 points)

Problem 4.1 (0.5 point; no coding)

When is the profit of a firm maximized?

1. When marginal revenue is strictly smaller than marginal cost;
2. When marginal revenue is strictly greater than marginal cost;
3. When marginal revenue equals marginal cost;
4. When marginal revenue is zero;
5. When marginal cost is zero.

Type and save exactly one number as your solution in a multiline comment under the section Problem 4.1 in `cs3430_s19_exam_03.py`

Problem 4.2 (1.5 points; coding)

Solve $y' = 5y$, $y(0) = 3$ and write the function `problem_4_2()` that plots and displays the graph of your solution where the range of x is limited to [-2, 2] and the range of y is limited to [0, 10].

Problem 4.3 (0.5 point; no coding)

Suppose that the demand for a given product at a given price is elastic. What happens to the change in revenue with respect to the change in price?

1. When the revenue increases, the price decreases and when the revenue decreases, the price increases;
2. When the revenue increases or decreases, the price remains the same;
3. When revenue increases or decreases, the price changes in the same direction;

Type and save your solution in a multiline comment under the section Problem 4.3 in `cs3430_s19_exam_03.py`

Problem 4.4 (0.5 point; no coding)

Suppose that the demand for a given product at a given price is inelastic. What happens to the change in revenue with respect to the change in price?

1. When the revenue increases, the price decreases and when the revenue decreases, the price increases;
2. When the revenue increases or decreases, the price remains the same;
3. When revenue increases or decreases, the price changes in the same direction;

Type and save your solution in a multiline comment under the section Problem 4.4 in `cs3430_s19_exam_03.py`

Problem 5: (3 points)

Problem 5.1 (1 point; coding)

Use the Newton-Raphson algorithm to approximate a solution to $e^{10-x} = 20 - x$. Save your implementation in the function `problem_5_1()`. This function returns a float.

Problem 5.2 (2 points; coding)

Write the function `pell_sqrt(n)` that uses Pell equations to compute $\sqrt{n}$, where $1 \leq n \leq 30$ and n is not a perfect square. Pells do not approximate perfect squares well. An integer n is a perfect square if there exists another integer k such that $k^2 = n$. For example, 1 is a perfect square, because $1^2 = 1$; 4 is a perfect square, because $4 = 2^2$; 9 is a perfect square, because $9 = 3^2$, etc.

You can use the function `test_pell_sqrt(lwr, uppr)` defined in `cs3430_s19_exam_03.py` to test your implementation of `pell_sqrt` in the interval `[lwr, uppr]`, where `lwr` and `uppr` are positive integers such that `lwr` $\leq$ `uppr`.

```
def test_pell_sqrt(lwr, uppr):
  for i in range(lwr, uppr+1):
    try:
      print('pell_sqrt(' + str(i) + ')=' + str(pell_sqrt(i)) + '; ' + \
            'math.sqrt(' + str(i) + ')=' + str(math.sqrt(i)))
    except Exception, e:
      print(str(i) + ' --> ' + str(e))
```

Let's use Pells to approximate square roots of numbers in the range $[1, 10]$.

```
>>> test_pell_sqrt(1, 10)
pell_sqrt(1)=1 is a perfect square; math.sqrt(1)=1.0
pell_sqrt(2)=1.41666666667; math.sqrt(2)=1.41421356237
pell_sqrt(3)=1.73111395647; math.sqrt(3)=1.73205080757
pell_sqrt(4)=4 is a perfect square; math.sqrt(4)=2.0
pell_sqrt(5)=2.23607038123; math.sqrt(5)=2.2360679775
pell_sqrt(6)=2.44948974279; math.sqrt(6)=2.44948974278
pell_sqrt(7)=2.64575127772; math.sqrt(7)=2.64575131106
pell_sqrt(8)=2.82842712474; math.sqrt(8)=2.82842712475
pell_sqrt(9)=9 is a perfect square; math.sqrt(9)=3.0
pell_sqrt(10)=3.16227766017; math.sqrt(10)=3.16227766017
```

Let's use Pells to approximate square roots of numbers in the range $[11, 20]$.

```
>>> test_pell_sqrt(11, 20)
pell_sqrt(11)=3.31662479036; math.sqrt(11)=3.31662479036
pell_sqrt(12)=3.46410161514; math.sqrt(12)=3.46410161514
pell_sqrt(13)=3.60555127546; math.sqrt(13)=3.60555127546
pell_sqrt(14)=3.74165738677; math.sqrt(14)=3.74165738677
pell_sqrt(15)=3.87298334621; math.sqrt(15)=3.87298334621
pell_sqrt(16)=16 is a perfect square; math.sqrt(16)=4.0
pell_sqrt(17)=4.12310562562; math.sqrt(17)=4.12310562562
pell_sqrt(18)=4.24264068712; math.sqrt(18)=4.24264068712
pell_sqrt(19)=4.35889894354; math.sqrt(19)=4.35889894354
pell_sqrt(20)=4.472135955; math.sqrt(20)=4.472135955
```

Let's use Pells to approximate square roots of numbers in the range $[21, 30]$.

```
>>> test_pell_sqrt(21, 30)
pell_sqrt(21)=4.58257569496; math.sqrt(21)=4.58257569496
pell_sqrt(22)=4.69041575982; math.sqrt(22)=4.69041575982
pell_sqrt(23)=4.79583152331; math.sqrt(23)=4.79583152331
pell_sqrt(24)=4.89897948557; math.sqrt(24)=4.89897948557
pell_sqrt(25)=25 is a perfect square; math.sqrt(25)=5.0
pell_sqrt(26)=5.09901951359; math.sqrt(26)=5.09901951359
pell_sqrt(27)=5.19615242271; math.sqrt(27)=5.19615242271
pell_sqrt(28)=5.29150262213; math.sqrt(28)=5.29150262213
pell_sqrt(29)=5.38516480713; math.sqrt(29)=5.38516480713
pell_sqrt(30)=5.47722557505; math.sqrt(30)=5.47722557505
```

Save your implementation of `pell_sqrt(n)` in `cs3430_s19_exam_03.py`.

What to Submit

You must submit `cs3430_s19_exam_03.py` and the files needed to run your functions in `cs3430_s19_exam_03.py`. Zip your whole directory into `exam03.zip` and upload it to Canvas.