CS 3430: S19: SciComp with Py
Assignment 11
Newton-Raphson Algorithm and Line Detection with Hough Transform

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 6, 2019

Learning Objectives

1. Newton-Raphson Algorithm
2. Hough Transform
3. Line Detection
4. Edge Dection

Introduction

In this assignment, we'll implement the Newton-Raphson algorithm and use the gradient-based edge detection algorithm from the previous assignment to implement the version of Hough Transform discussed in lectures 21 and 22.

Problem 1: Newton-Raphson Algorithm (1 point)

Implement the function `nra(poly_fexpr, g, n)` that takes a function expression of a polynomial and two constants, `g` and `n`, where `g` is the initial guess and `n` is the number of iterations, and approximates a root of the polynomial specified by `poly_fexpr` starting with `g` and iterating the Newton-Raphson algorithm `n` times. Ten unit tests are given below.

Test 1

Let's approximate $\sqrt{2}$.

```
>>> import math
>>> fexpr = make_plus(make_pwr('x', 2.0),
                      make_const(-2.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
1.41421356237
>>> math.sqrt(2)
1.4142135623730951
```

Test 2

Let's approximate $\sqrt{3}$.

```
>>> import math
>>> fexpr = make_plus(make_pwr('x', 2.0),
                      make_const(-3.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
1.73205080757
>>> math.sqrt(3)
1.7320508075688772
```

Test 3

Let's approximate $\sqrt{5}$.

```
>>> import math
>>> fexpr = make_plus(make_pwr('x', 2.0),
                      make_const(-5.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
2.2360679775
>>> math.sqrt(5)
2.23606797749979
```

Test 4

Let's approximate $\sqrt{7}$.

```
>>> import math
>>> fexpr = make_plus(make_pwr('x', 2.0),
                      make_const(-7.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
2.64575131106
>>> math.sqrt(7)
2.6457513110645907
```

Test 5

Let's approximate a solution to $e^{-x} = x^2$.

```
>>> import math
>>> fexpr = make_e_expr(make_prod(make_const(-1.0),
                                  make_pwr('x', 1.0)))
>>> fexpr = make_plus(fexpr,
                      make_prod(make_const(-1.0),
                                make_pwr('x', 2.0)))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
0.703467422498
>>> math.e**(-0.703467422498) - (0.703467422498)**2
7.449041383722488e-13
```

Test 6

Let's approximate $11^{\frac{1}{3}}$.

```
>>> import math
>>> fexpr = make_pwr('x', 3.0)
>>> fexpr = make_plus(fexpr,
                      make_const(-11.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
2.22398009057
>>> 11.0**(1.0/3.0)
2.2239800905693152
```

Test 7

Let's approximate $6^{\frac{1}{3}}$.

```
>>> import math
>>> fexpr = make_pwr('x', 3.0)
>>> fexpr = make_plus(fexpr,
                      make_const(-6.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
1.81712059283
>>> 6.0**(1.0/3.0)
1.8171205928321397
```

Test 8

Let's approximate a root of $x^3 + 2x + 2$.

```
>>> import math
>>> fexpr = make_pwr('x', 3.0)
>>> fexpr = make_plus(fexpr,
                      make_prod(make_const(2.0),
                                make_pwr('x', 1.0)))
>>> fexpr = make_plus(fexpr, make_const(2.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
-0.770916997059
>>> x = -0.770916997059
>>> x**3 + 2*x + 2
9.388045896230324e-13
```

Test 9

Let's approximate a root of $x^3 + x - 1$.

```
>>> import math
>>> fexpr = make_pwr('x', 3.0)
>>> fexpr = make_plus(fexpr, make_pwr('x', 1.0))
>>> fexpr = make_plus(fexpr, make_const(-1.0))
>>> print(nra(fexpr, make_const(1.0), make_const(10000)))
0.682327803828
>>> x = 0.682327803828
>>> x**3 + x - 1
-4.6407322429331543e-14
```

Test 10

Let's approximate a root of $e^{(5-x)} = 10 - x$.

```
>>> import math
>>> fexpr = make_e_expr(make_plus(make_const(5.0),
                                  make_prod(make_const(-1.0),
                                            make_pwr('x', 1.0))))
>>> fexpr = make_plus(fexpr, make_pwr('x', 1.0))
>>> fexpr = make_plus(fexpr, make_const(-10.0))
print(nra(fexpr, make_const(1.0), make_const(10000)))
3.06315259278
>>> x = 3.06315259278
>>> math.e**(5 - x) + x - 10
-1.298516849601583e-12
```

Save your implementation of `nra()` in `cs3430_s19_hw11.py`. The above unit tests are coded up in `cs3430_s19_hw11.py` as `nra_ut_01()`, `nra_ut_02()`, etc.

Problem 2: Line Detection with Hough Transform (4 points)

Implement the function `ht_detect_lines(img_fp, magn_thresh=20, spl=20)` that takes a path to an image file, `img_fp`, the keyword `magn_thresh` that specifies a threshold value for gradient edge detection, and the keyword `spl` that specifies a support level for Hough Transform. The function returns four values - the original image array, the image array with all detected lines drawn with blue color, the binary image array with all the edge pixels equal to 255 and non-edge pixels equal to 0, and the hough table.

The unit tests are below. All the images are in `img/`. In each figure, the left image is the original image, the middle image is the image with detected edges, and the right image is the image with detected lines. Your output for tests 1 - 7 should be similar to mine. When I say *similar*, I don't mean to say that your images with detected lines should be identical to mine pixel by pixel. What I mean is that you should be able to detect the same lines, because these images are fairly simple. Tests 8 - 12 don't have to be the same, because these images are much trickier. But, your implementation should detect at least a couple of meaningful lines in each image.

Save your implementation of `ht_detect_lines()` in `cs3430_s19_hw11.py`. Experiment with different values of `magn_thresh` and `spl`. The file `cs3430_s19_hw11.py` contains a test function for the tests below (e.g., `ht_test_01()` for Test 1, `ht_test_02()` for Test 2, etc.). Save your best values of `magn_thresh` and `spl` in these definitions. When we run unit tests, we'll call test functions only with the file paths so that for each test `magn_thresh` and `spl` will default to your best values.

Test 1

Let's detect lines in EdgeImage_01.jpg. Fig. 1 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/EdgeImage_01.jpg',
                                            magn_thresh=35,
                                            spl=110)
```

Test 2

Let's detect lines in EdgeImage_02.jpg. Fig. 2 shows the output images.

Figure 1: Test1: Edges and lines detected in EdgeImage_01.jpg.

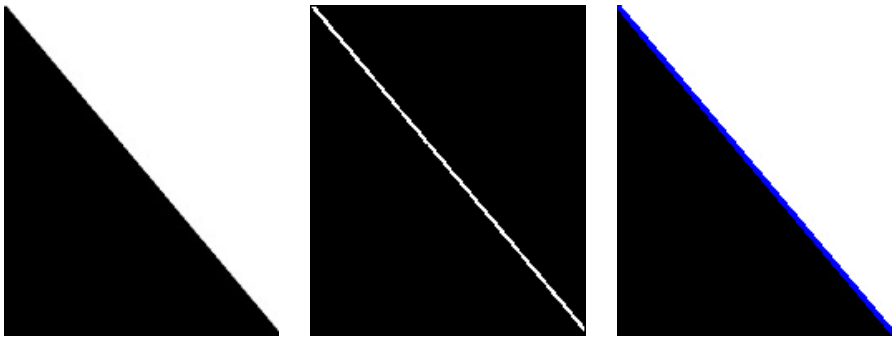

Figure 2: Edges and lines detected in EdgeImage_02.jpg.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/EdgeImage_02.jpg',
                                            magn_thresh=35,
                                            spl=110)
```

Test 3

Let's detect lines in EdgeImage_03.jpg. Fig. 3 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/EdgeImage_03.jpg',
                                            magn_thresh=35,
                                            spl=110)
```

Test 4

Let's detect lines in envelope.jpeg. Fig. 4 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/envelope.jpeg',
                                            magn_thresh=35,
                                            spl=200)
```

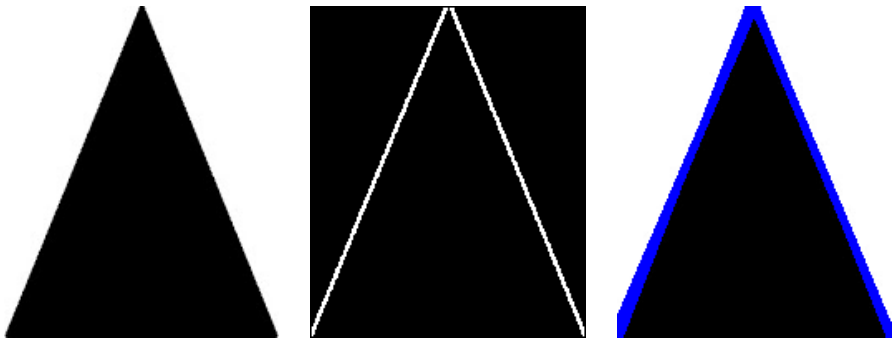

Figure 3: Edges and lines detected in EdgeImage_03.jpg.

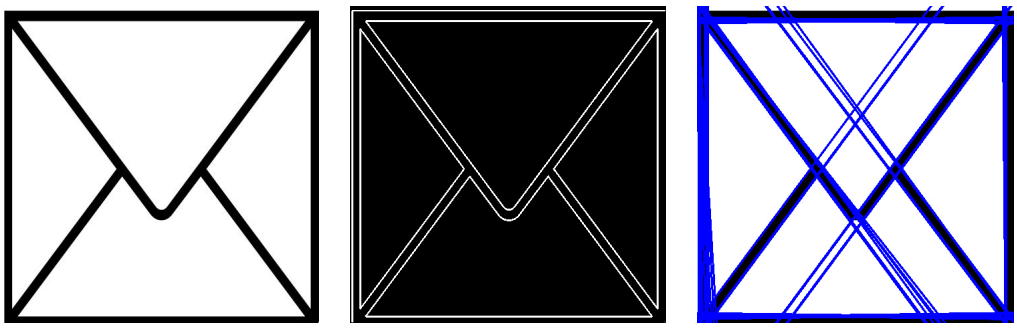

Figure 4: Edges and lines detected in envelope.jpeg.

Test 5

Let's detect lines in horline.png. Fig. 5 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/horline.png',
                                            magn_thresh=35,
                                            spl=100)
```

Test 6

Let's detect lines in verline.png. Fig. 6 shows the output images.

Figure 5: Edges and lines detected in horline.png.

Figure 6: Edges and lines detected in verline.png.

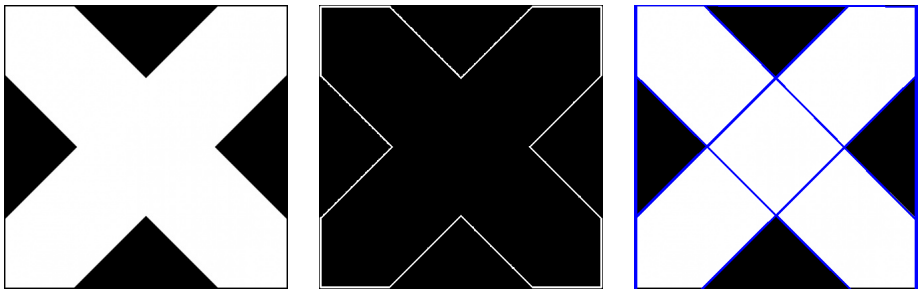

Figure 7: Edges and lines detected in cross.png.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/verline.png',
                                            magn_thresh=35,
                                            spl=100)
```

Test 7

Let's detect lines in cross.png. Fig. 7 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/cross.png',
                                            magn_thresh=15,
                                            spl=200)
```

Test 8

Let's detect lines in tiles.jpeg. Fig. 8 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/tiles.jpeg',
                                            magn_thresh=20,
                                            spl=450)
```

Test 9

Let's detect lines in kitchen.jpeg. Fig. 9 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/kitchen.jpeg',
                                            magn_thresh=15,
                                            spl=400)
```

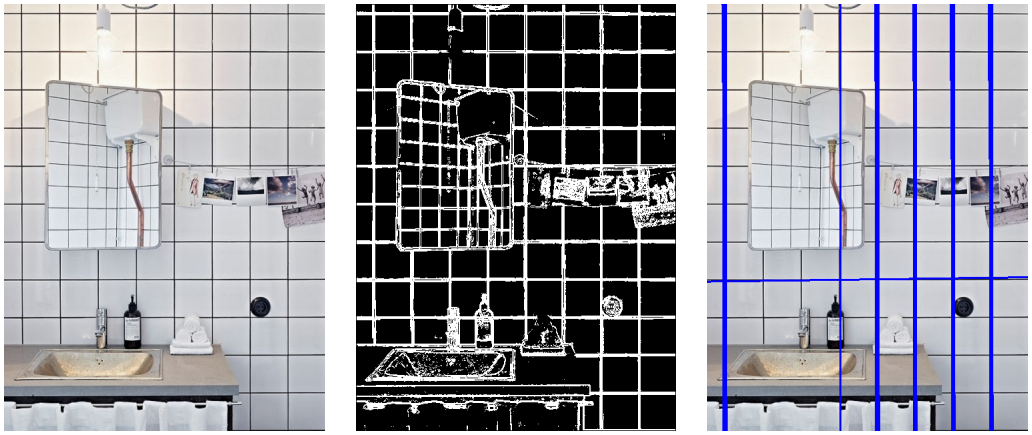

Figure 8: Edges and lines detected in tiles.jpeg.

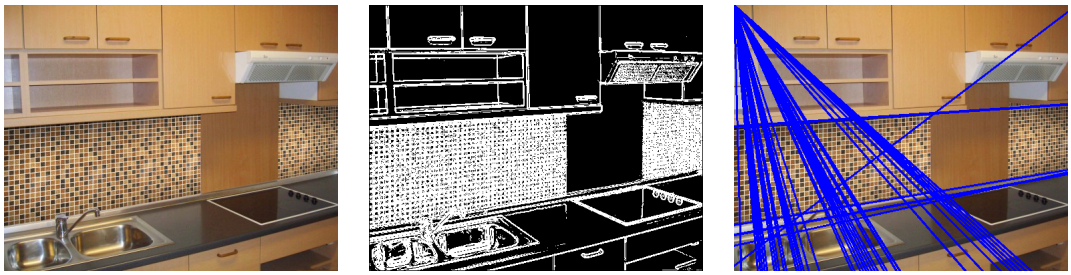

Figure 9: Edges and lines detected in kitchen.jpeg.

Figure 10: Edges and lines detected in road01.png.

Figure 11: Edges and lines detected in road02.png.

Test 10

Let's detect lines in road01.png. Fig. 10 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/road01.png',
                                            magn_thresh=10,
                                            spl=250)
```

Test 11

Let's detect lines in road02.png. Fig. 11 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/road02.png',
                                            magn_thresh=10,
                                            spl=210)
```

Test 12

Let's detect lines in road03.png. Fig. 12 shows the output images.

```
>>> img, lnimg, edimg, ht = ht_detect_lines('img/road03.png',
                                            magn_thresh=5,
                                            spl=205)
```

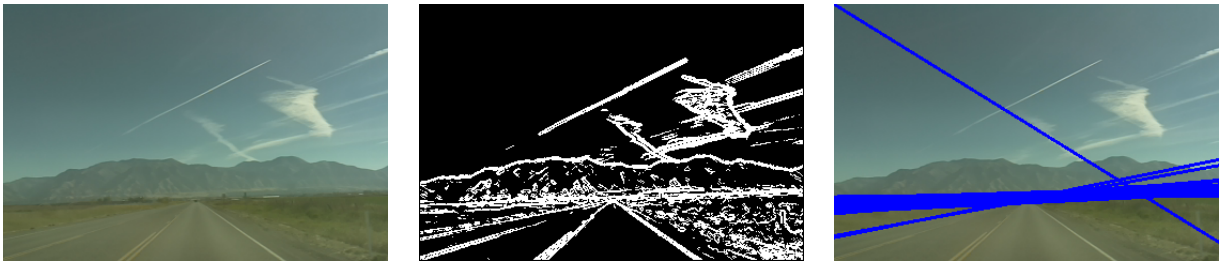

Figure 12: Edges and lines detected in road03.png.

What to Submit

You have to submit `cs3430_s19_hw11.py`. You should also submit all the files needed to run your code. Zip your entire working directory in `hw11.zip` and submit it via Canvas.

I end up with my usual 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!