CSC 240: Computer Graphics

Midterm: Fall 2015

Due: $\boxed{\text{Friday, October 30 at 1pm}}$

- This is a take-home exam with unlimited time from when it is out to when it is due.
- It is open-notes, so you may use any course materials. If you use any online resources that haven't been part of this class, please cite them.
- No communication about the exam with anyone in the class (or outside the class). However, you can email me if you need clarification.
- If there is a clarification I think should be made to the entire class, I'll post it on Piazza.
- I will still have office hours on Monday 4-6pm and Wednesday 5:30-7pm, but I might not say much!
- Turn in your exam to me in person or put it under the door of my office if I am not there (not in the bin outside).
- If you are unable to make it to my office to turn in the exam before the deadline, email me to make other arrangements such as scanning your exam.
- If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.

Name	Solution Set

(sketches)

Part 1	/20	✓
Part 2	/10	✓
Part 3	/20	✓
Part 4	/30	✓
Part 5	/20	✓
Total	/100	

Part 1: Lines

In HW1 we saw how to implement an algorithm for drawing a line between two points: $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, which relied on the slope of the line. For this question, write pseudocode for a line algorithm that would achieve the same purpose, but this time using a parametric Bézier approach. Complete the draw method below (ignore our usual img and color arguments). (20pts)

Your algorithm should:

(a) Make the line look "connected" (no gaps). Diagonal pixels are considered connected.

(b) No pixel should be colored more than once.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self):
        setPixel(self.x, self.y)

class Line:
    # p1 and p2 are of type Point
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    # complete the draw method below, using a parametric Bezier approach
    def draw(self):
```

```
        # number of pixels to draw
        num-p = max { abs(p1.x-p2.x), abs(p1.y - p2.y) }

        # loop over num-p
        for i in range(num-p):
            t = i/num-p     # parametric t ∈ [0,1]
            x = (1-t)·p1.x + t·p2.x
            y = (1-t)·p1.y + t·p1.y
            next-point = Point(round(x), round(y))
            next-point.draw()
```

this is the key idea for this algorithm: it is like checking whether $|m| < 1$ or $|m| \geq 1$

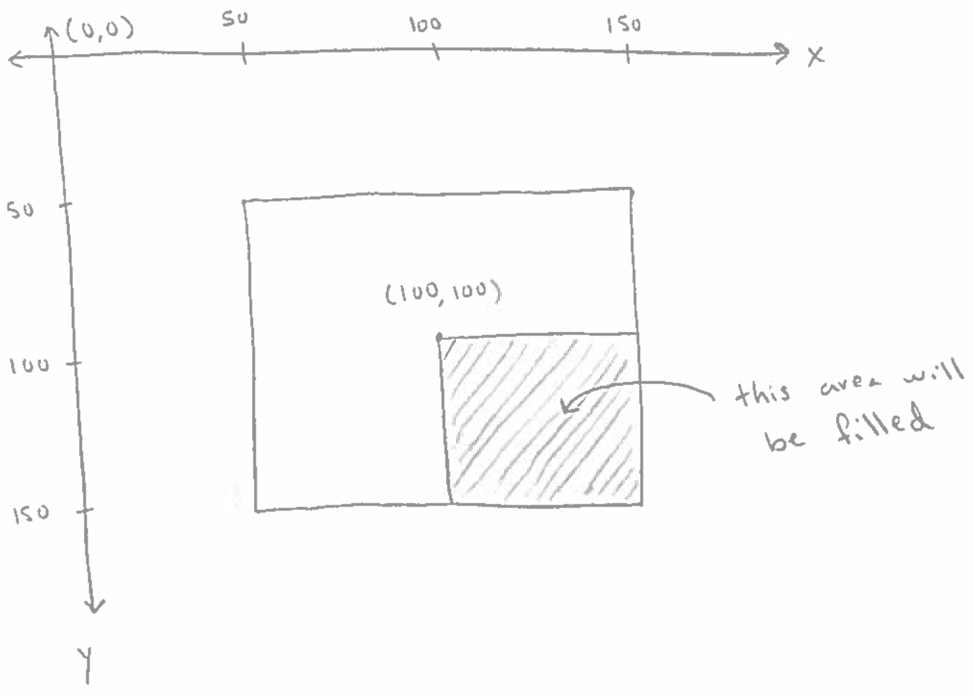
More space for Part 1:

Part 2: Polygons and Fill

After noticing that flood fill (Lab 3) gives a "maximum recursion depth exceeded" if the shape to be filled is too large, you've been tasked with modifying flood fill to make it more usable. (10pts) A fellow student suggests removing some of the recursive calls, as shown in `flood_fill2` below:

```
def flood_fill2(img, x, y, old_color, new_color):

    curr_color = img.getPixel(x, y)

    if curr_color == new_color:
        return # nothing to do, finished
    if curr_color != old_color:
        return # hit an edge, finished

    img.setPixel(x, y, new_color) # curr_color must have been old_color, "fill" it

    # recurse!
    flood_fill2(img, x+1, y, old_color, new_color) # east
    flood_fill2(img, x, y+1, old_color, new_color) # south

    return
```

(a) Given a square centered at the point (100,100) with side length 100 in the raster framework, what will happen if `flood_fill2` is called with $x = 100$, and $y = 100$? Sketch what area of the square will be filled.



(b) Without modifying `flood_fill2`, how could you call it in a different way to make sure the entire square is filled? Write some short pseudocode to demonstrate this.

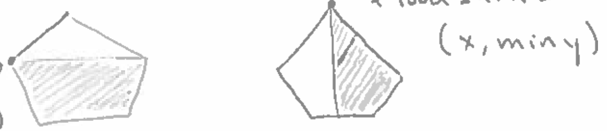
Start flood-fill 2 at the $\underbrace{\text{top-left}}_{(\text{min } x \text{ \& min } y)}$ hand corner instead:

```
flood_fill2(img, 51, 51, old-color, new-color)
```

depending on the implementation details, these may need to be 51 to 50 or 51 to start off the edge, but is the main idea.

(c) Would this same modification work for any regular polygon? Argue why or why not. If not, what modification could you make to fix it?

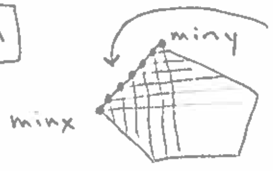
$\boxed{\text{No}}$ There might not always be a "top-left" corner of the shape: example



- If we called flood-fill 2 on the pixel with the smallest x-coordinate and the smallest y-coordinate, that (still) won't always work.

- Variety of modifications work:

$\boxed{\text{option 1}}$



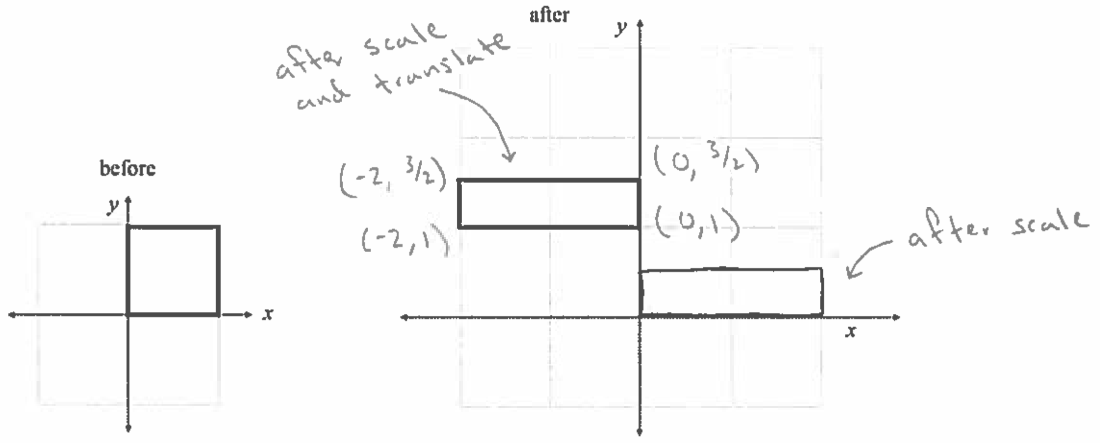
call flood_fill2 on all these points between furthest north pixel & furthest west pixel

$\boxed{\text{option 2}}$ I also accepted adding one more recursive call to flood-fill 2 (either west or north), since that is in the spirit of reducing recursive calls.

Part 3: Transformations

(a) For each of the transformations below to the unit square, provide the transformation matrix/matrices (there could be more than one way to get the same effect). If you use more than one matrix, explain the order. For at least one vertex on the square, use matrix multiplication to show that your transformations do the right thing. (10pts)

i.



Scale matrix: $\begin{bmatrix} 2 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ translate matrix: $\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

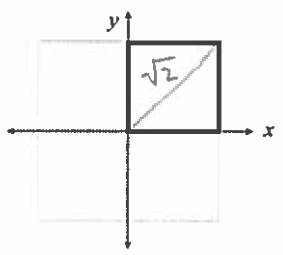
matrix multiplication:

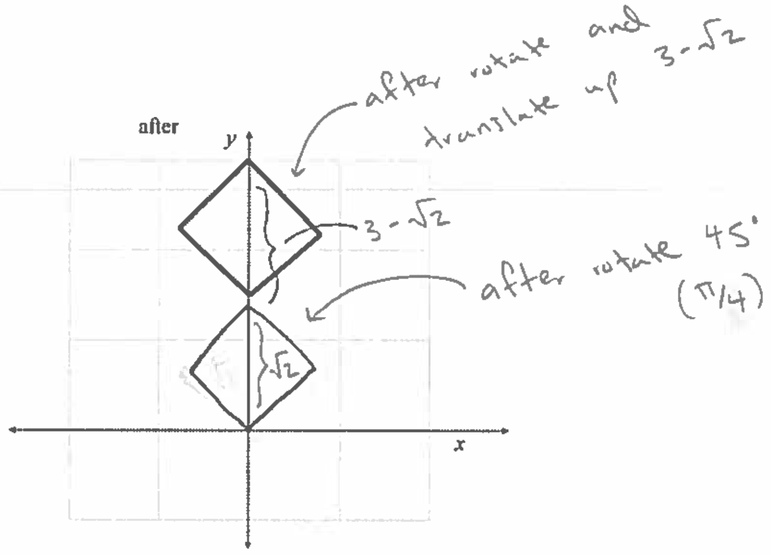
$$\underbrace{\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{translate second}} \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{scale first}} \underbrace{\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{"square" matrix}}$$

$$= \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 1/2 & 1/2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{after scale}} = \underbrace{\begin{bmatrix} -2 & -2 & 0 & 0 \\ 1 & 3/2 & 3/2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{after scale + translate}}$$

transformed square ✓

ii.




after
y
x
after rotate and translate up $3-\sqrt{2}$
$3-\sqrt{2}$
after rotate 45° ($\pi/4$)
$\sqrt{2}$

rotate matrix : $\begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} & 0 \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ translate matrix : $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3-\sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}$

matrix multiplication:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3-\sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}}_{\text{translate second}} \underbrace{\begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotate first}} \underbrace{\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{"Square" matrix}}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3-\sqrt{2} \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 0 & -\sqrt{2}/2 & 0 & \sqrt{2}/2 \\ 0 & \sqrt{2}/2 & \sqrt{2} & \sqrt{2}/2 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{after rotate}} = \underbrace{\begin{bmatrix} 0 & -\sqrt{2}/2 & 0 & \sqrt{2}/2 \\ 3-\sqrt{2} & 3-\sqrt{2}/2 & 3 & 3-\sqrt{2}/2 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{after rotate + translate.}}$$

transformed square ✓

(b) Do shear (x) transformations *always* commute with translations? Put another way: is it true that for *any* shear transformation matrix H_x and *any* translation matrix T, that $H_xT = TH_x$? For either yes or no, provide *both* a picture and a matrix multiplication argument. If yes, the picture should explain the general intuition and the matrix multiplication should be encompass all possible H_x and T. If no, provide a counterexample in the form of matrix multiplication results. Then apply those same transformations in both orders to an example image and sketch the results. (10pts)

$\boxed{\text{No}}$ Let $H_x = \begin{bmatrix} 1 & l_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ + $T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

$$H_x \cdot T = \begin{bmatrix} 1 & l_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & l_x & t_x + l_x t_y \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

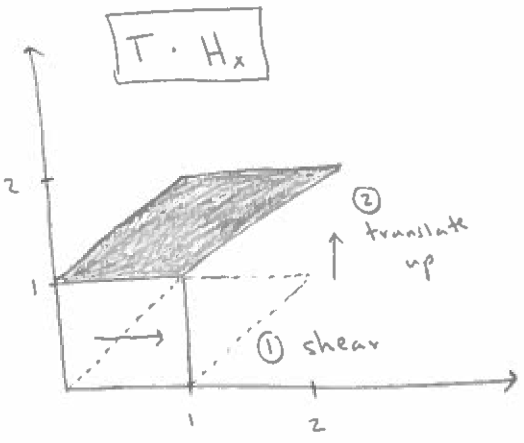
$\neq$ (unless $t_y = 0$)

$$T \cdot H_x = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & l_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & l_x & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

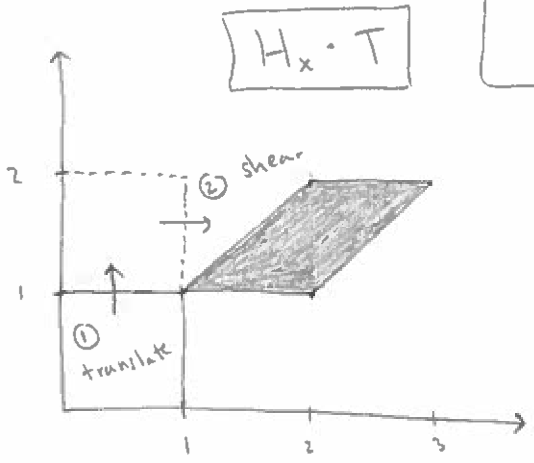
<u>example:</u> $l_x = 1$, $t_x = 0$, $t_y = 1$

$H_x = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$



start with unit square



[do not get the same shape!]

Part 4: OpenGL

For this part you are asked to write two animations in OpenGL, starting from the code `midterm.py` on the course website. Questions (a) and (b) are highlighted in the code - for (b), comment out your code for (a), but leave it there so I can alternate commenting each section, i.e. make it clear in your code what is (a) and what is (b)!

When you run `python midterm.py`, you should get our familiar white square. Write your transformation code in the display method and submit your code (one file) on Moodle. Make sure your animations are slow enough that the effects are easily visible.

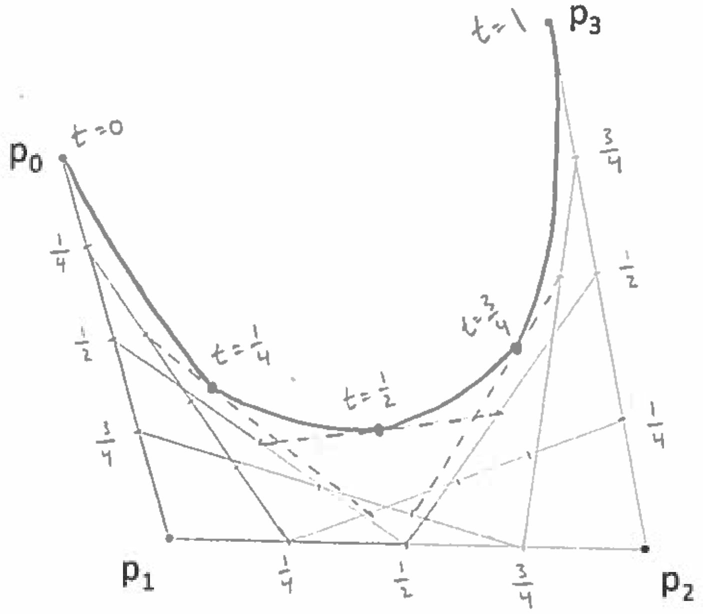
(a) *Spinning and disappearing square*: write an animation to make the square rotate around the origin, while simultaneously getting smaller and smaller, until it eventually becomes a speck (10pts). After it essentially disappears, make it get bigger and bigger until it is back to its original size, then get smaller again. In short, make it alternate between original size → disappear → back to original size, indefinitely, rotating all the while (5pts).

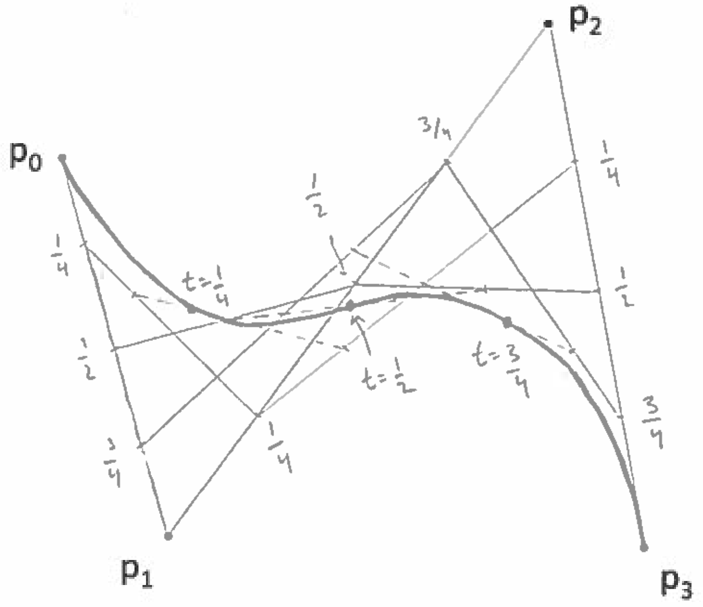
(b) *Rotating in two ways*: write a separate animation to make the square rotate in a circle centered at the origin, while simultaneously rotating about itself. The relative rate of rotation doesn't matter, but the effect should be clear. *Imagine running around a track while also spinning in circles.* (15pts)

see midterm-sol.py

Part 5: Bézier Curves and Cubic Splines

(a) Draw the Bézier curve for each set of control points below (the control points are the same, but the order is different). Label/utilize the points on the curve when $t = 0$, $1/4$, $1/2$, $3/4$, and 1. (10pts)





(b) You are given the following two cubic splines (for each of x and y) in 2D:

$$X_0(t) = 4 - \frac{11}{2}t + \frac{5}{2}t^3 \qquad Y_0(t) = 3 - t$$
$$X_1(t) = 1 + 2t + \frac{15}{2}t^2 - \frac{3}{2}t^3 \qquad Y_1(t) = 2 - t$$

For the parts below, write out your steps and/or explain your reasoning. (10pts)

i. What are the three ordered control points p_0, p_1, and p_2? (both their x and y coordinates)

x - coordinates

$p_0.x = X_0(0) = 4$

$p_1.x = X_0(1) = 4 - \frac{11}{2} + \frac{5}{2} = 4 - 3 = 1$

$= X_1(0) = 1$ ✓

$p_2.x = X_1(1) = 1 + 2 + \frac{15}{2} - \frac{3}{2} = 3 + 6 = 9$

y - coordinates

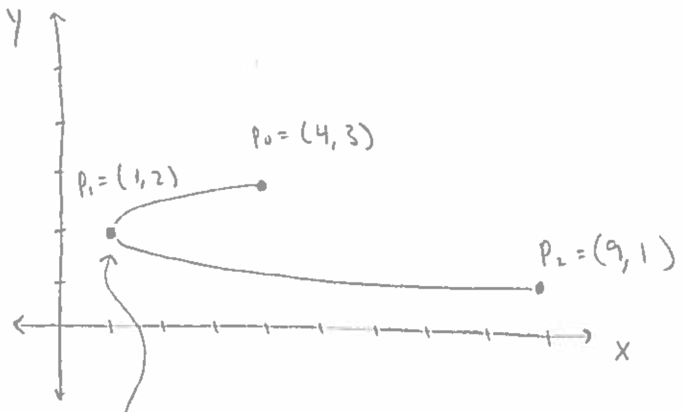
$p_0.y = Y_0(0) = 3$

$p_1.y = Y_0(1) = Y_1(0) = 2$

$p_2.y = Y_1(1) = 1$

$$\boxed{\begin{aligned} p_0 &= (4,3) \\ p_1 &= (1,2) \\ p_2 &= (9,1) \end{aligned}}$$

ii. Roughly plot the control points and sketch the cubic splines above.



iii. Show that the first derivative matches where the two splines meet (for both x and y).

We want: $\boxed{X_0'(1) = X_1'(0)}$

$X_0'(1) = -\frac{11}{2} + \frac{15}{2}t^2 \Big|_{t=1}$

$= -\frac{11}{2} + \frac{15}{2}$

$\boxed{X_0'(1) = 2}$

$X_1'(0) = 2 + 15t - \frac{9}{2}t^2 \Big|_{t=0}$

$\boxed{X_1'(0) = 2}$

✓ yes they match.

More space for Part 5.

Also want: $Y_0'(1) = Y_1'(0)$

$\boxed{Y_0'(1) = -1}$, $\boxed{Y_1'(0) = -1}$ ✓ they match too.