Chapter 2
Scan Conversion

Credit hours: 7 hrs

Contents

- 2.1 Scan Conversion
- 2.2 Line Drawing Algorithms
    - 2.2.1 Digital Differential Analyzer
    - 2.3.2 Bresenham's Algorithm
- 2.3 Circle Generation Algorithm
- 2.4 Ellipse Generation Algorithm
- 2.5 Filled Area Primitives
    - 2.5.1 Scan Line Polygon Fill Algorithm
    - 2.5.2 Boundary Fill Algorithm, Flood Fill Algorithm

2.1 Scan Conversion

geometric representation of an object → pixel-based representation

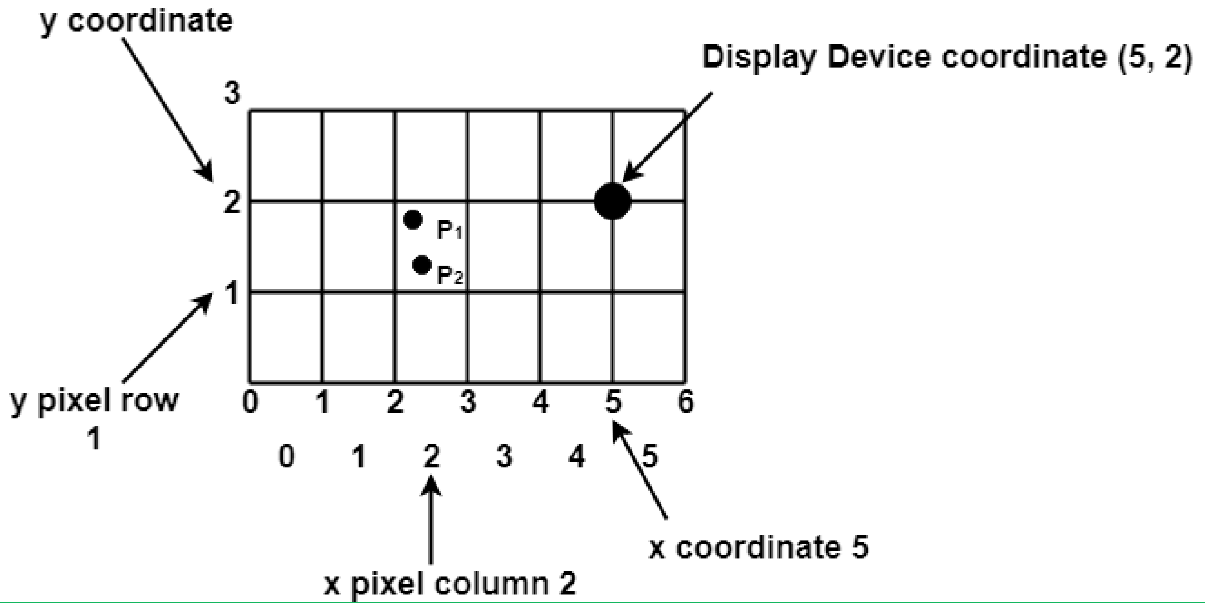
Definition

- Scan conversion is the **process of converting a geometric representation of an object**, such as a line or curve, **into a pixel-based representation** that can be displayed on a computer screen or printed on a page.
- Scan conversion is the process of transforming vector graphics into raster graphics, which are made up of pixels or dots that form a digital image.
- It is a **process of representing graphics objects**, a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either **ON** or **OFF** state.
- The **circuitry of the video display device** of the computer is capable of converting binary values (0, 1) into a pixel ON and pixel OFF information. 0 is represented by pixel OFF. 1 is represented using pixel ON. Using this ability, graphics computer represent picture having discrete dots.

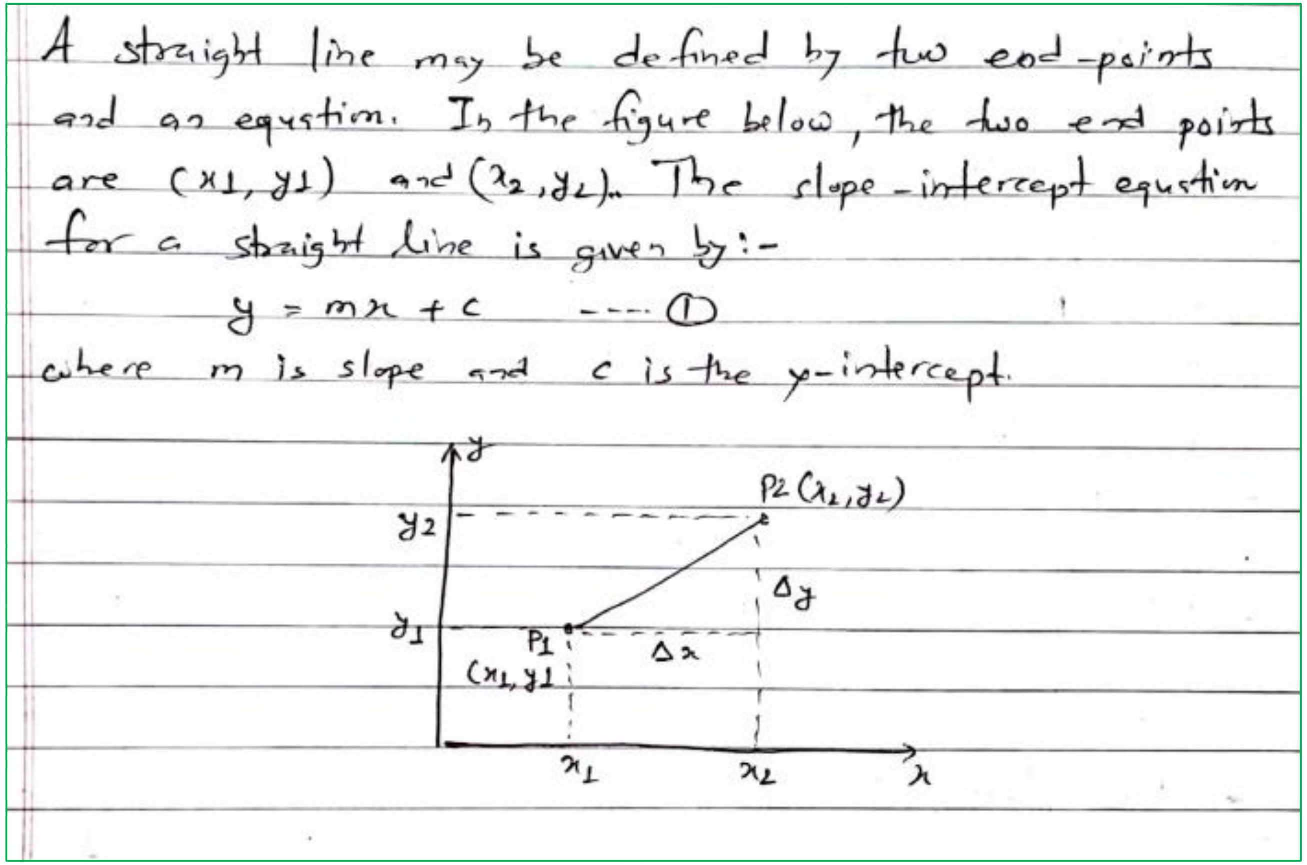
2.1.1 Point Scan Conversion

- A mathematical point (x,y) where x and y are real numbers within an image area, needs to be scan converted to a pixel at location (x’, y’).
- So, point plotting is accomplished by converting a single coordinate position into appropriate operation for the output device in use.

- **Example:** Display coordinates points $P_1\ (2\frac{1}{4}, 1\frac{3}{4})\&\ P_2(2\frac{2}{3}, 1\frac{1}{4})$ as shown in fig would both be represented by pixel (2, 1).
- In general, a point p (x, y) is represented by the integer part of x & the integer part of y that is pixels [(INT (x), INT (y)].



2.1.2. Line Scan Conversion

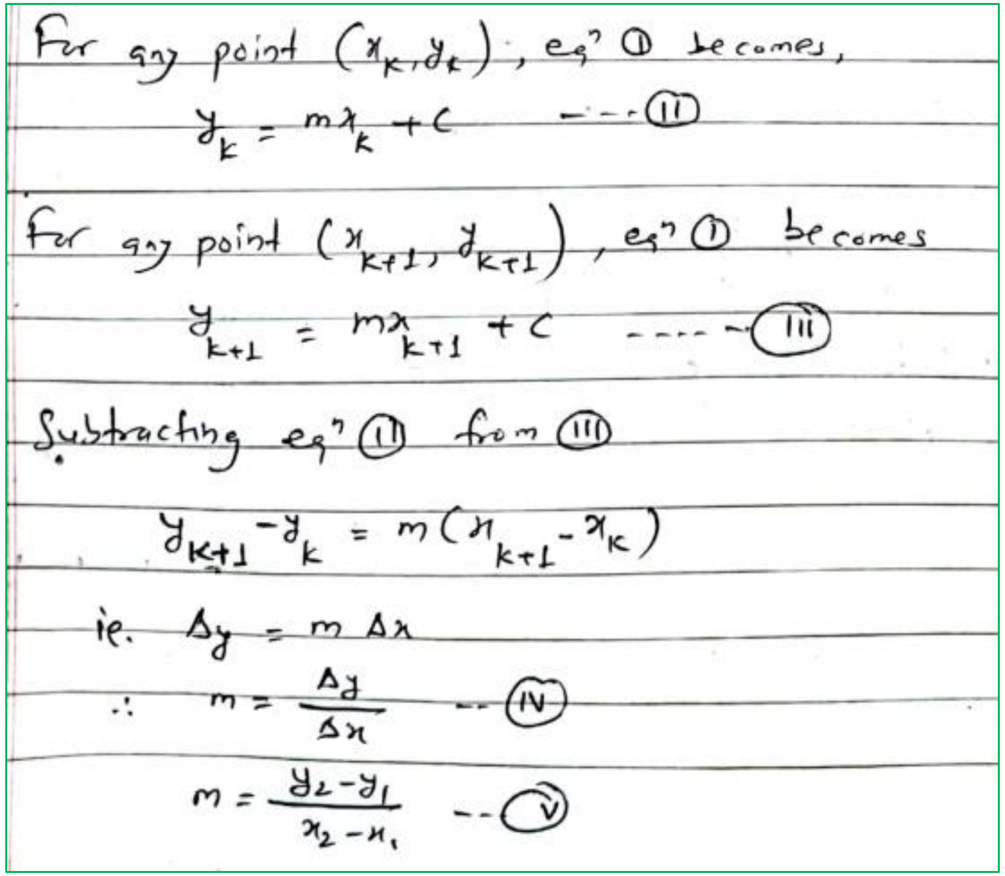


A straight line may be defined by two end-points and an equstion. In the figure below, the two end points are (x_1, y_1) and (x_2, y_2). The slope-intercept equstion for a straight line is given by:-

$$y = mx + c \quad ---- \text{①}$$

where m is slope and c is the y-intercept.

y
y_2
y_1
$P_2 (x_2, y_2)$
P_1
(x_1, y_1)
Δy
Δx
x_1
x_2
x



For any point (x_k, y_k), eq^n ① becomes,

$$y_k = mx_k + c \quad --- \text{(II)}$$

For any point (x_{k+1}, y_{k+1}), eq^n ① becomes

$$y_{k+1} = mx_{k+1} + c \quad ---- \text{(III)}$$

Subtracting eq^n (II) from (III)

$$y_{k+1} - y_k = m(x_{k+1} - x_k)$$

ie. $\Delta y = m \, \Delta x$

$$\therefore \quad m = \frac{\Delta y}{\Delta x} \quad -- \text{(IV)}$$

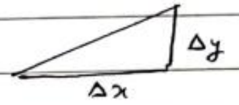
$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad -- \text{(V)}$$

→ The above eqn (IV) forms the basis for determining the intermediate pixel's position (ie. deflection voltage in analog devize like CRT).

→ When the value of m is calculated using eqn (V), test for three cases can be performed as:-

Case I: For $|m| < 1$

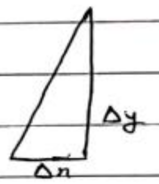
Δx can be increased by 1, and corresponding Δy is calculated from eqn (IV)



$$\frac{\Delta y}{\Delta x} < 1$$

Case II: For $|m| > 1$

Set Δy to increase by 1 and calculate the corresponding Δx from eqn (IV)

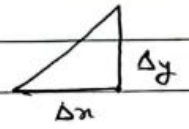


$$\frac{\Delta y}{\Delta x} > 1$$

Case III: For $|m| = 1$

Set $\Delta x = \Delta y$.

Increase both of them equally.

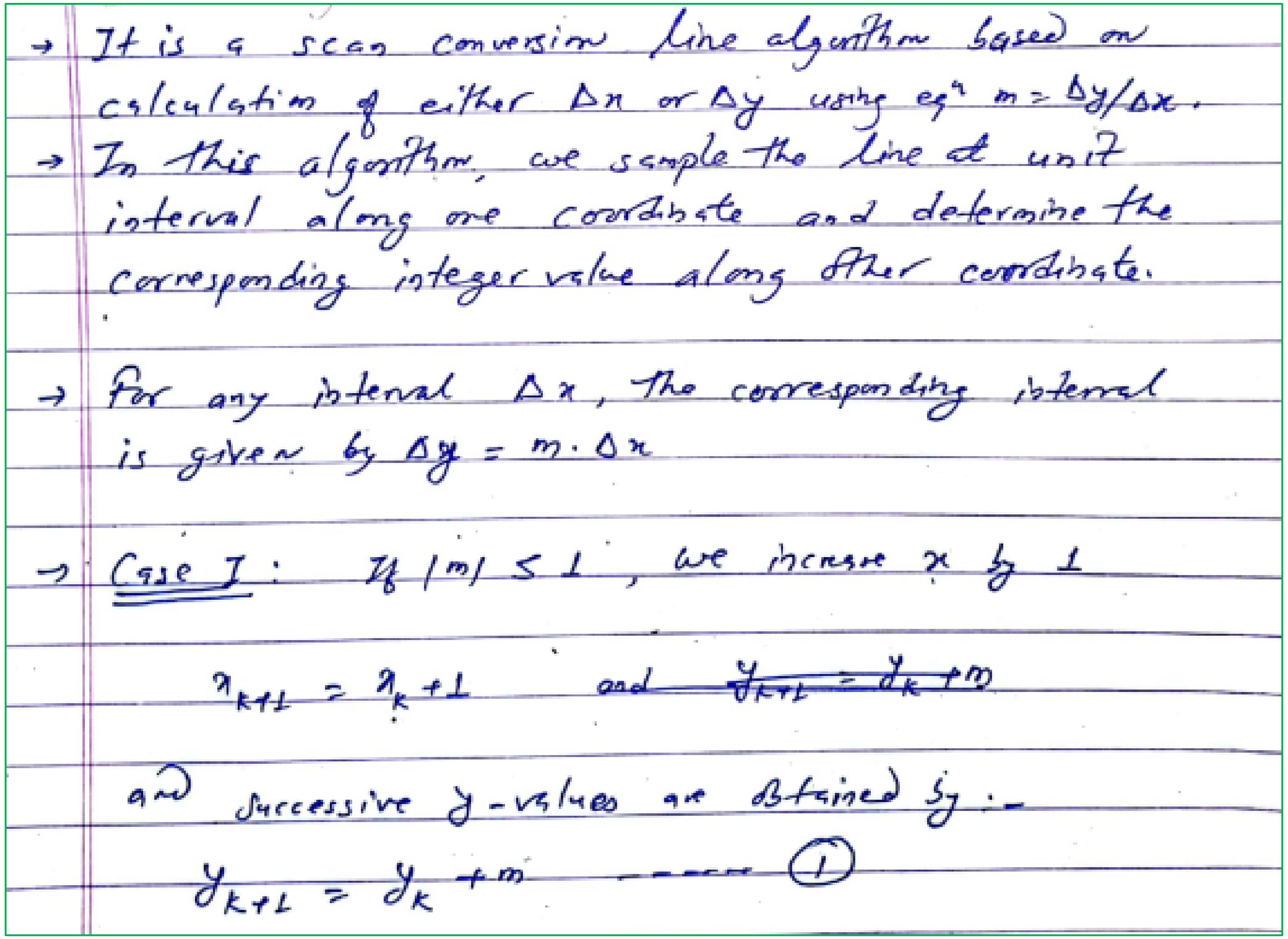


$$\frac{\Delta y}{\Delta x} = 1$$

2.2 Line Drawing Algorithms

- DDA (Digital Differential Analyzer)
- Bresenham’s Algorithm

2.2.1 DDA



→ It is a scan conversion line algorithm based on calculation of either Δx or Δy using eqn $m = \Delta y / \Delta x$.

→ In this algorithm, we sample the line at unit interval along one coordinate and determine the corresponding integer value along other coordinate.

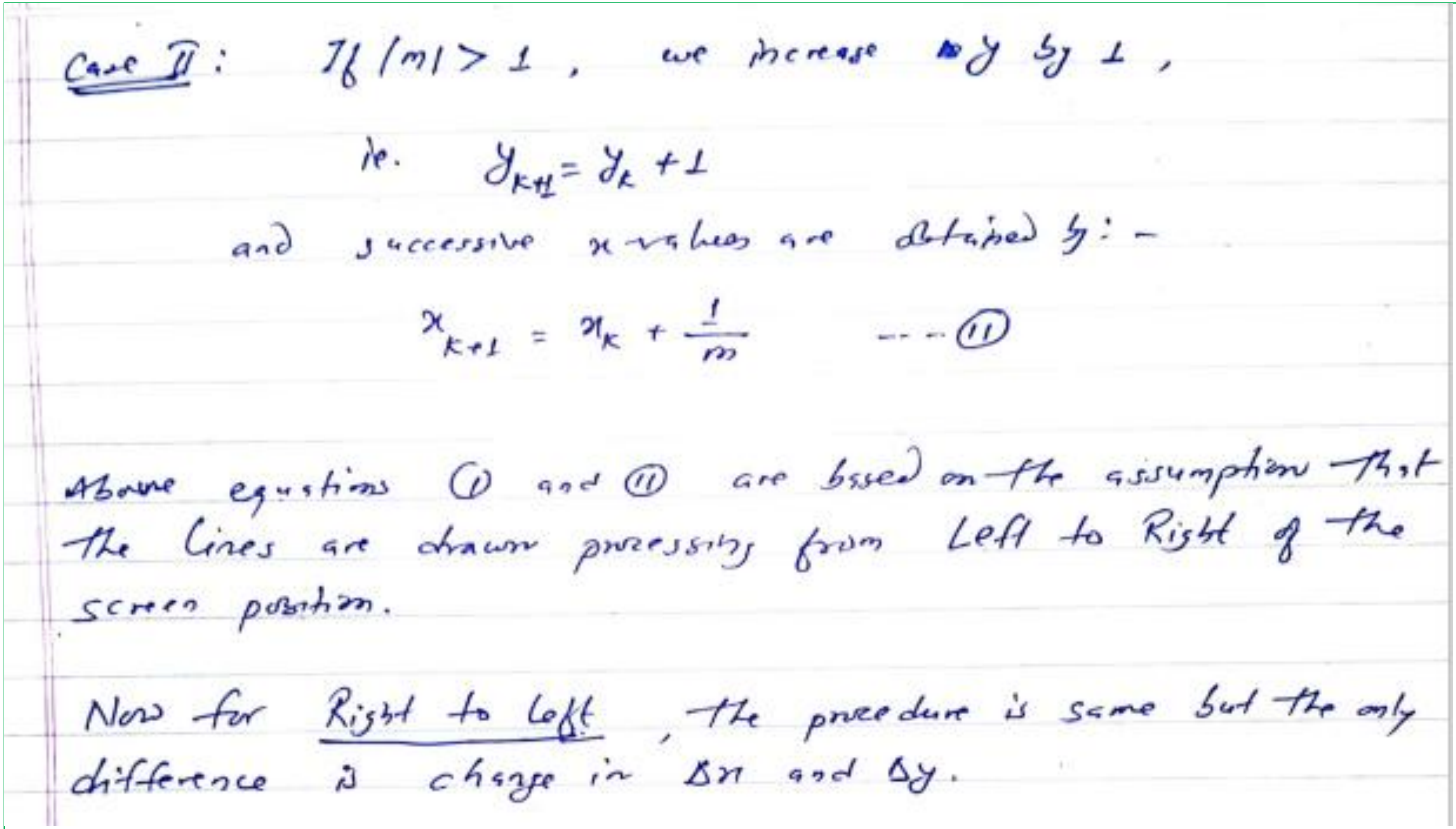
→ For any interval Δx, the corresponding interval is given by $\Delta y = m \cdot \Delta x$

→ Case I : If $|m| \leq 1$, we increase x by 1

$x_{k+1} = x_k + 1$ and ~~$y_{k+1} = y_k + m$~~

and successive y-values are obtained by :-

$y_{k+1} = y_k + m$ ------ $\textcircled{1}$



Case II : If $|m| > 1$, we increase y by 1 ,

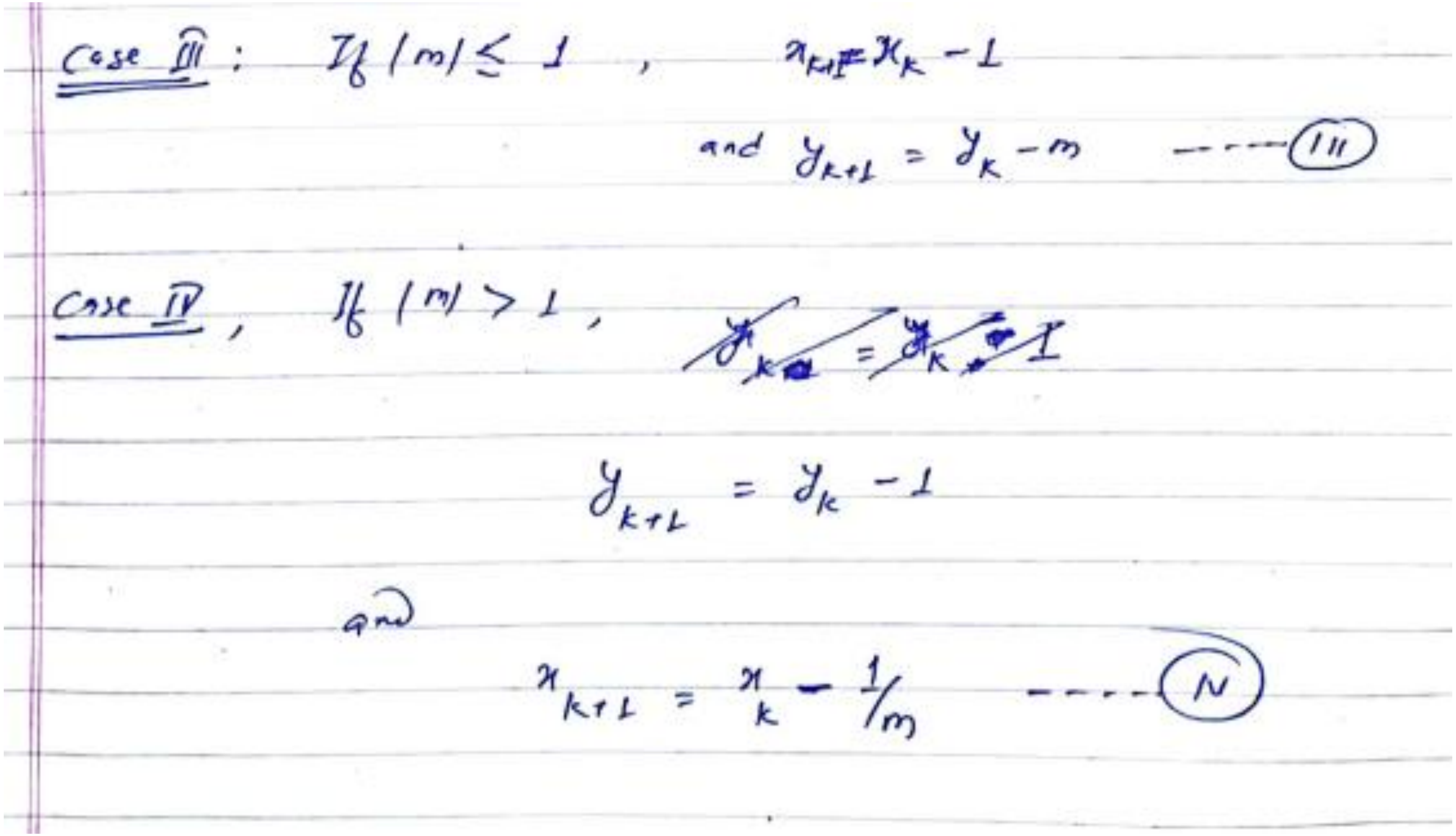
i.e. $y_{k+1} = y_k + 1$

and successive x-values are obtained by :-

$$x_{k+1} = x_k + \frac{1}{m} \quad \text{----} \ (ii)$$

Above equations (i) and (ii) are based on the assumption that the lines are drawn processing from Left to Right of the screen position.

Now for Right to Left , the procedure is same but the only difference is change in Δx and Δy.



Case III : If $|m| \leq 1$, $x_{k+1} = x_k - 1$

and $y_{k+1} = y_k - m$ ---- (III)

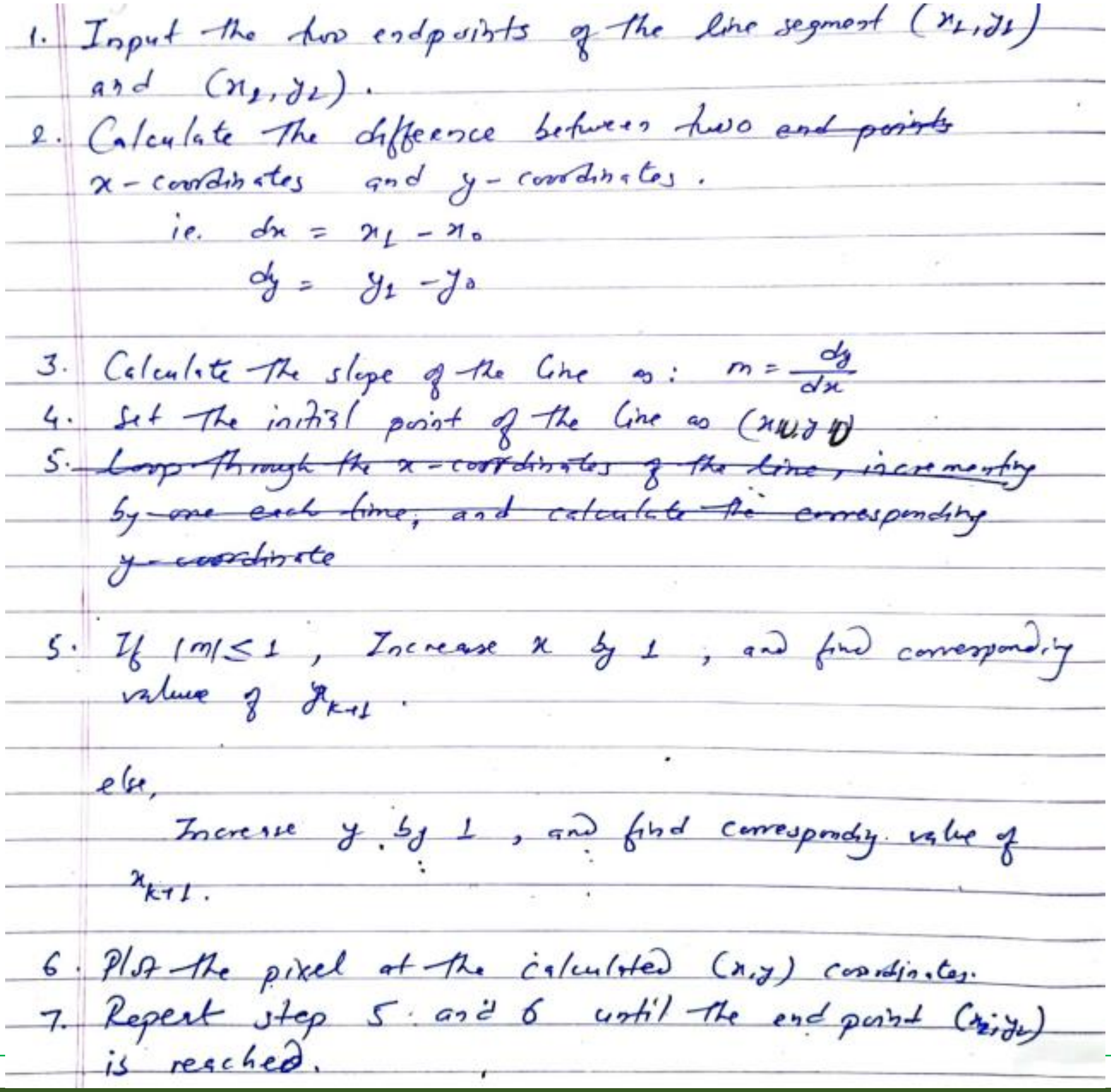
Case IV , If $|m| > 1$, ~~$y_{k+1} = y_k + 1$~~

$$y_{k+1} = y_k - 1$$

and

$$x_{k+1} = x_k - 1/m \quad ---- (IV)$$

- **Algorithm:**



1. Input the two endpoints of the line segment (x_1, y_1) and (x_2, y_2).
2. Calculate the difference between two ~~end points~~ x-coordinates and y-coordinates.

   i.e. $dx = x_1 - x_0$

   $dy = y_1 - y_0$

3. Calculate the slope of the line as: $m = \frac{dy}{dx}$
4. Set the initial point of the line as (x_0, y_0)
5. ~~Loop through the x-coordinates of the line, incrementing by one each time, and calculate the corresponding y-coordinate~~

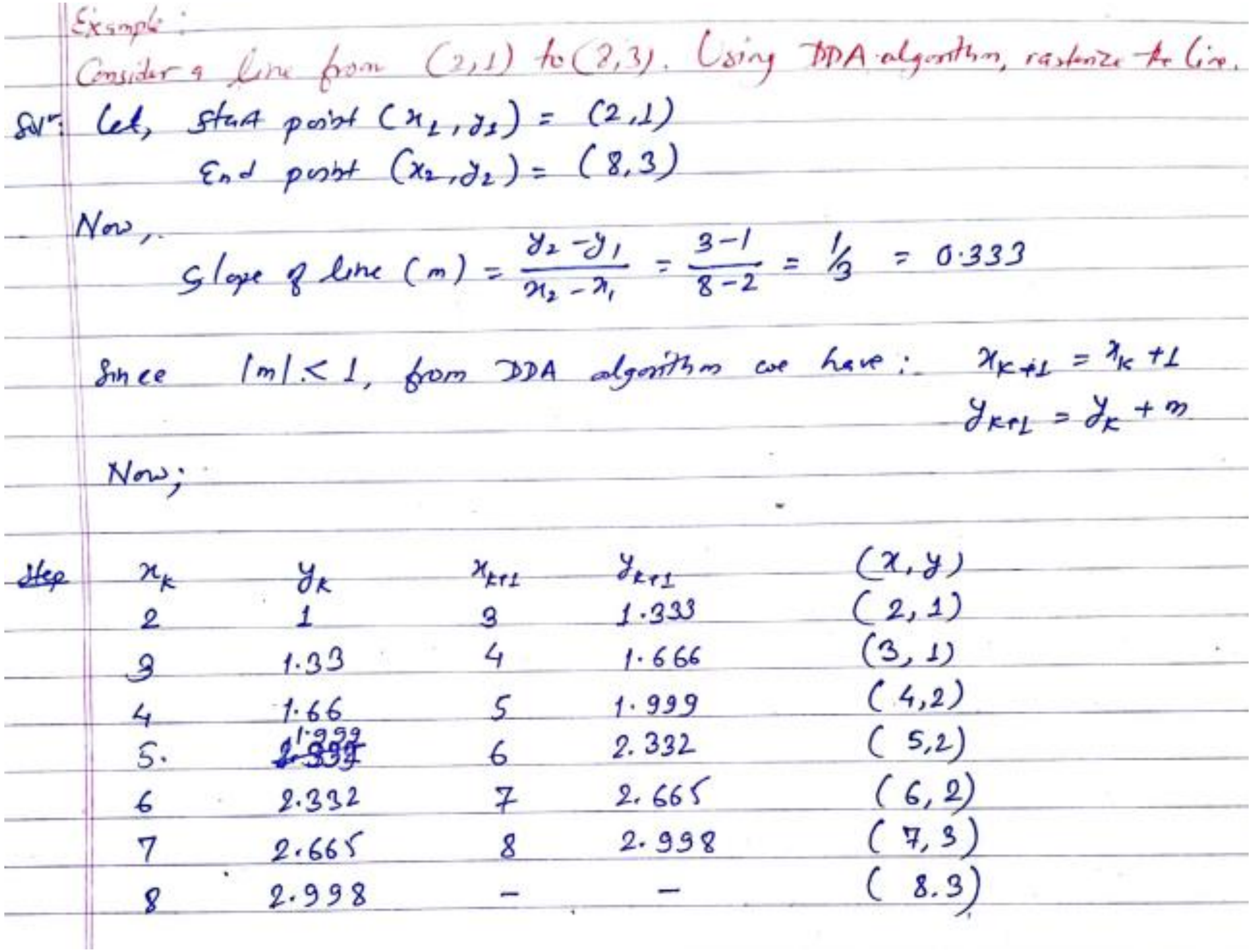
5. If $|m| \leq 1$, Increase x by 1, and find corresponding value of y_{k+1}.

   else,

   Increase y by 1, and find corresponding value of x_{k+1}.

6. Plot the pixel at the calculated (x, y) coordinates.
7. Repeat step 5 and 6 until the end point (x_2, y_2) is reached.

- **Numerical:**



Example :

Consider a line from (2,1) to (8,3). Using DDA algorithm, rasterize the line.

Sol^n: Let, start point $(x_1, y_1) = (2,1)$

End point $(x_2, y_2) = (8,3)$

Now,

$$\text{Slope of line } (m) = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-1}{8-2} = \frac{1}{3} = 0.333$$

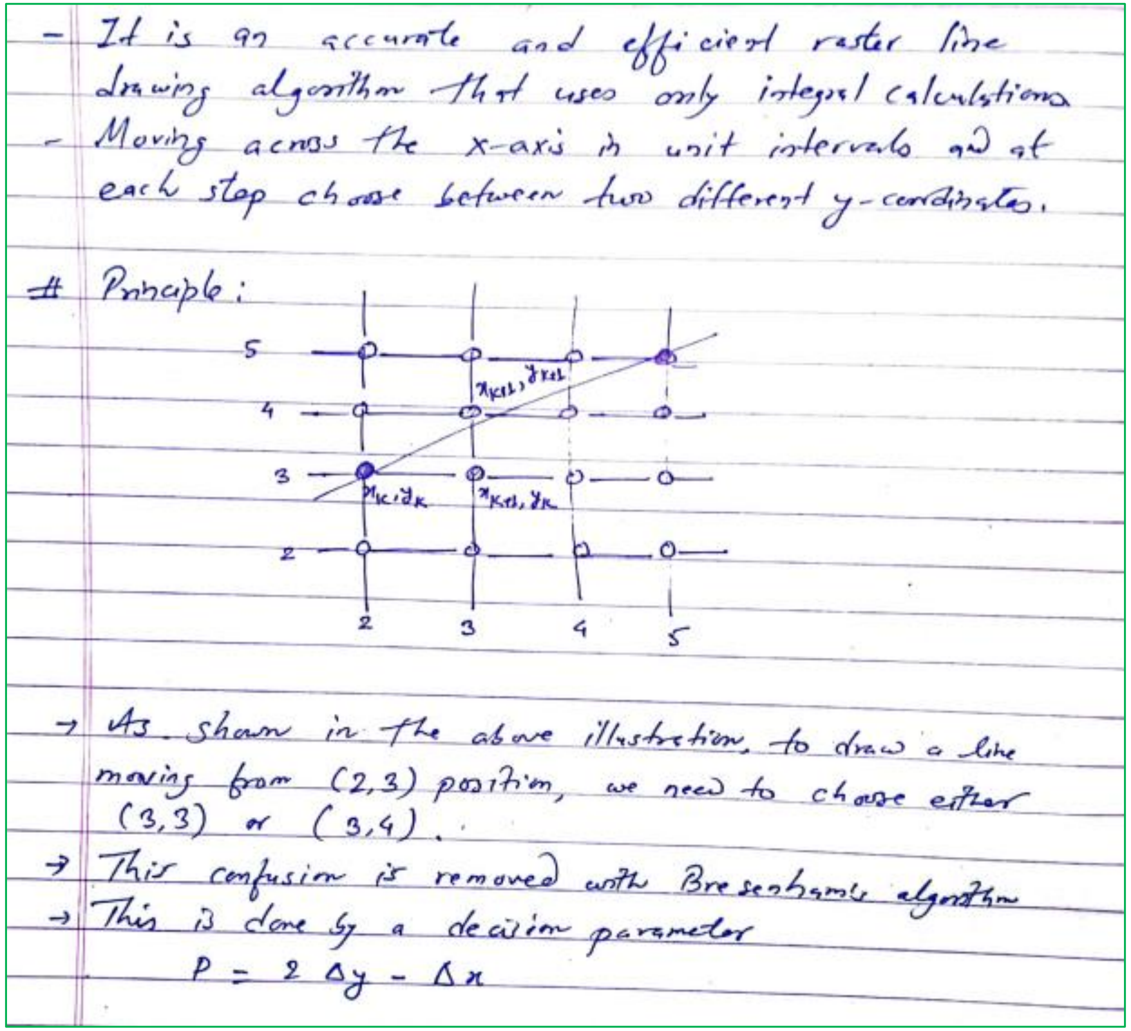
Since $|m| < 1$, from DDA algorithm we have: $x_{k+1} = x_k + 1$

$y_{k+1} = y_k + m$

Now;

Step	x_k	y_k	x_{k+1}	y_{k+1}	(x, y)
	2	1	3	1.333	(2, 1)
	3	1.33	4	1.666	(3, 1)
	4	1.66	5	1.999	(4,2)
	5.	~~2.333~~ 1.999	6	2.332	(5,2)
	6	2.332	7	2.665	(6, 2)
	7	2.665	8	2.998	(7, 3)
	8	2.998	–	–	(8.3)

2.2.2 Bresenham’s Algorithm



- It is an accurate and efficient raster line drawing algorithm that uses only integral calculations
- Moving across the x-axis in unit intervals and at each step choose between two different y-coordinates.

Principle:

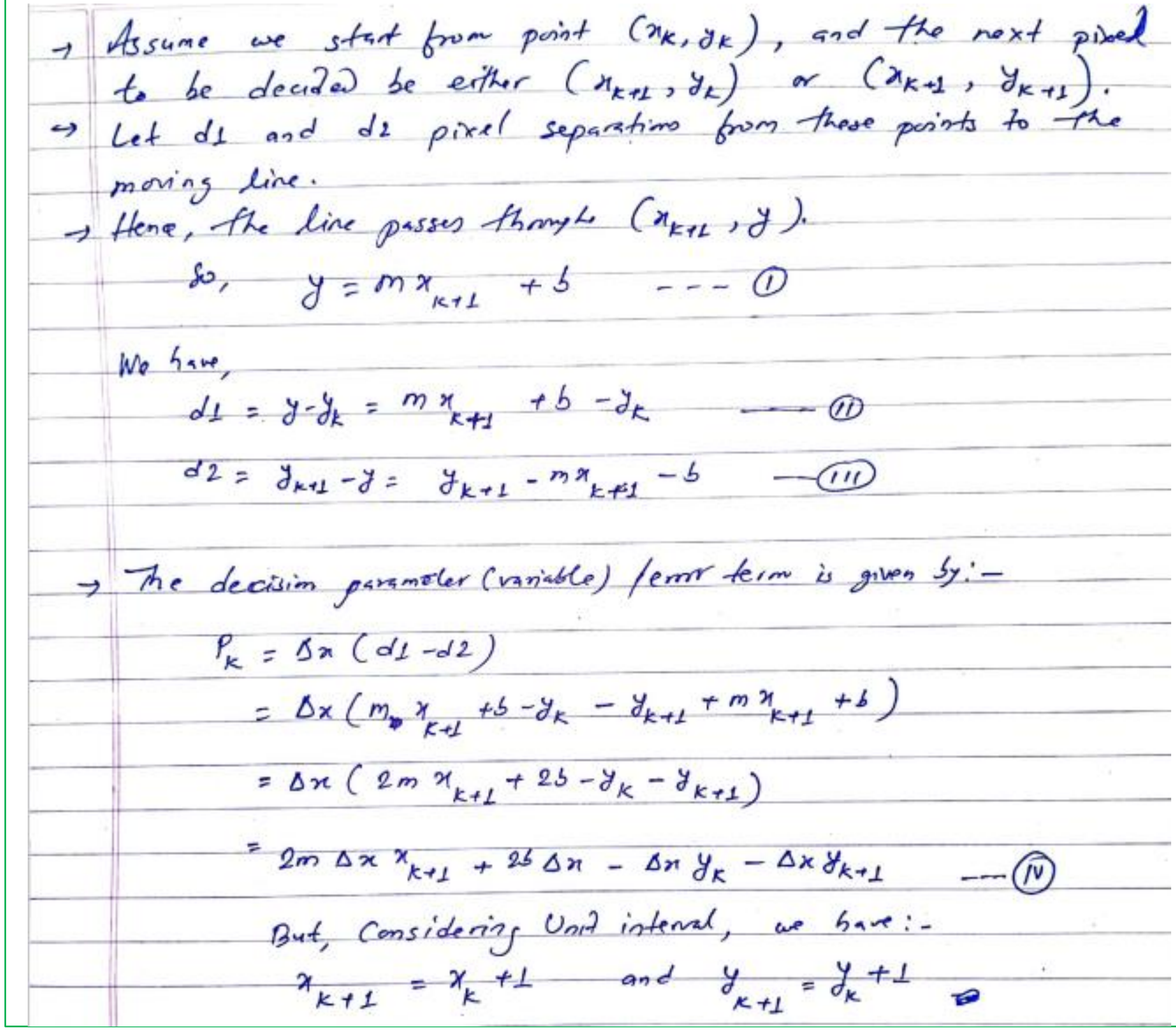
5
4
3
2
2
3
4
5
x_{k+1}, y_{k+1}
x_k, y_k
x_{k+1}, y_k

→ As shown in the above illustration, to draw a line moving from (2,3) position, we need to choose either (3,3) or (3,4).

→ This confusion is removed with Bresenham's algorithm

→ This is done by a decision parameter

$$P = 2\Delta y - \Delta x$$



→ Assume we start from point (x_k, y_k), and the next pixel to be decided be either (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}).

→ Let d1 and d2 pixel separations from these points to the moving line.

→ Here, the line passes through (x_{k+1}, y).

So, $y = mx_{k+1} + b$ --- ①

We have,

$d1 = y - y_k = mx_{k+1} + b - y_k$ —— (ii)

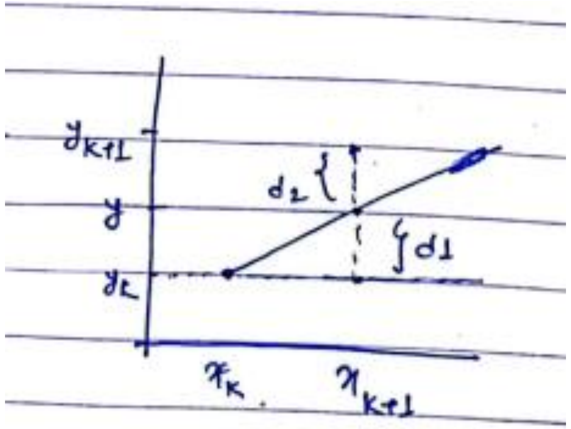
$d2 = y_{k+1} - y = y_{k+1} - mx_{k+1} - b$ —— (iii)

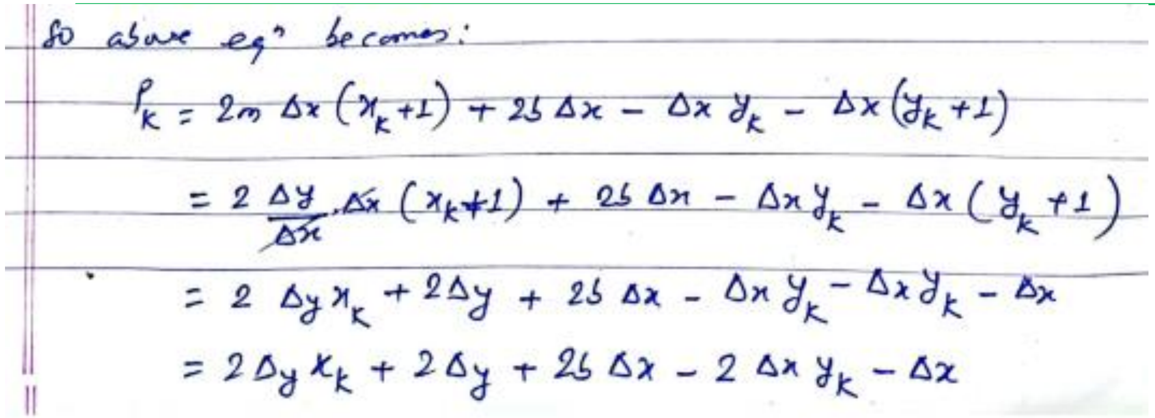
→ The decision parameter (variable) / error term is given by:-

$$P_k = \Delta x (d1 - d2)$$
$$= \Delta x (mx_{k+1} + b - y_k - y_{k+1} + mx_{k+1} + b)$$
$$= \Delta x (2mx_{k+1} + 2b - y_k - y_{k+1})$$
$$= 2m\Delta x\, x_{k+1} + 2b\Delta x - \Delta x\, y_k - \Delta x\, y_{k+1} \quad \text{--- (IV)}$$

But, considering unit interval, we have:-

$x_{k+1} = x_k + 1$ and $y_{k+1} = y_k + 1$





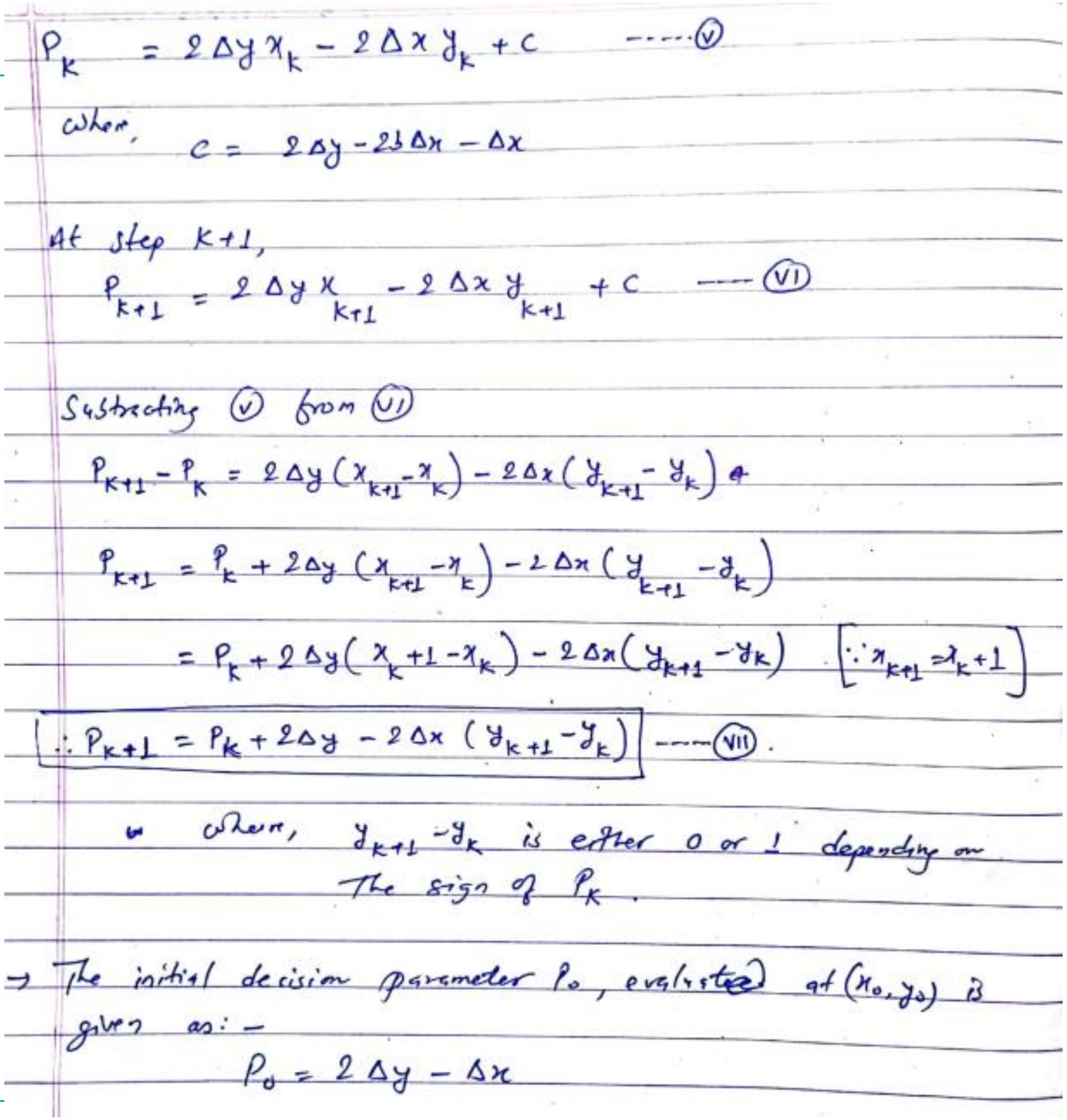
So above eqn becomes:

$$P_k = 2m\,\Delta x(x_k+1) + 2b\,\Delta x - \Delta x\, y_k - \Delta x(y_k+1)$$

$$= 2\frac{\Delta y}{\cancel{\Delta x}}.\cancel{\Delta x}(x_k+1) + 2b\,\Delta x - \Delta x\, y_k - \Delta x(y_k+1)$$

$$= 2\,\Delta y\, x_k + 2\Delta y + 2b\,\Delta x - \Delta x\, y_k - \Delta x\, y_k - \Delta x$$

$$= 2\Delta y\, x_k + 2\Delta y + 2b\,\Delta x - 2\,\Delta x\, y_k - \Delta x$$



$P_k = 2\Delta y\, x_k - 2\Delta x\, y_k + c$ ----(v)

where, $c = 2\Delta y - 2b\Delta x - \Delta x$

At step k+1,

$P_{k+1} = 2\Delta y\, x_{k+1} - 2\Delta x\, y_{k+1} + c$ ---- (VI)

Substracting (v) from (VI)

$$P_{k+1} - P_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

$$P_{k+1} = P_k + 2\Delta y\,(x_{k+1} - x_k) - 2\Delta x\,(y_{k+1} - y_k)$$

$$= P_k + 2\Delta y(x_k + 1 - x_k) - 2\Delta x(y_{k+1} - y_k) \quad [\because x_{k+1} = x_k + 1]$$

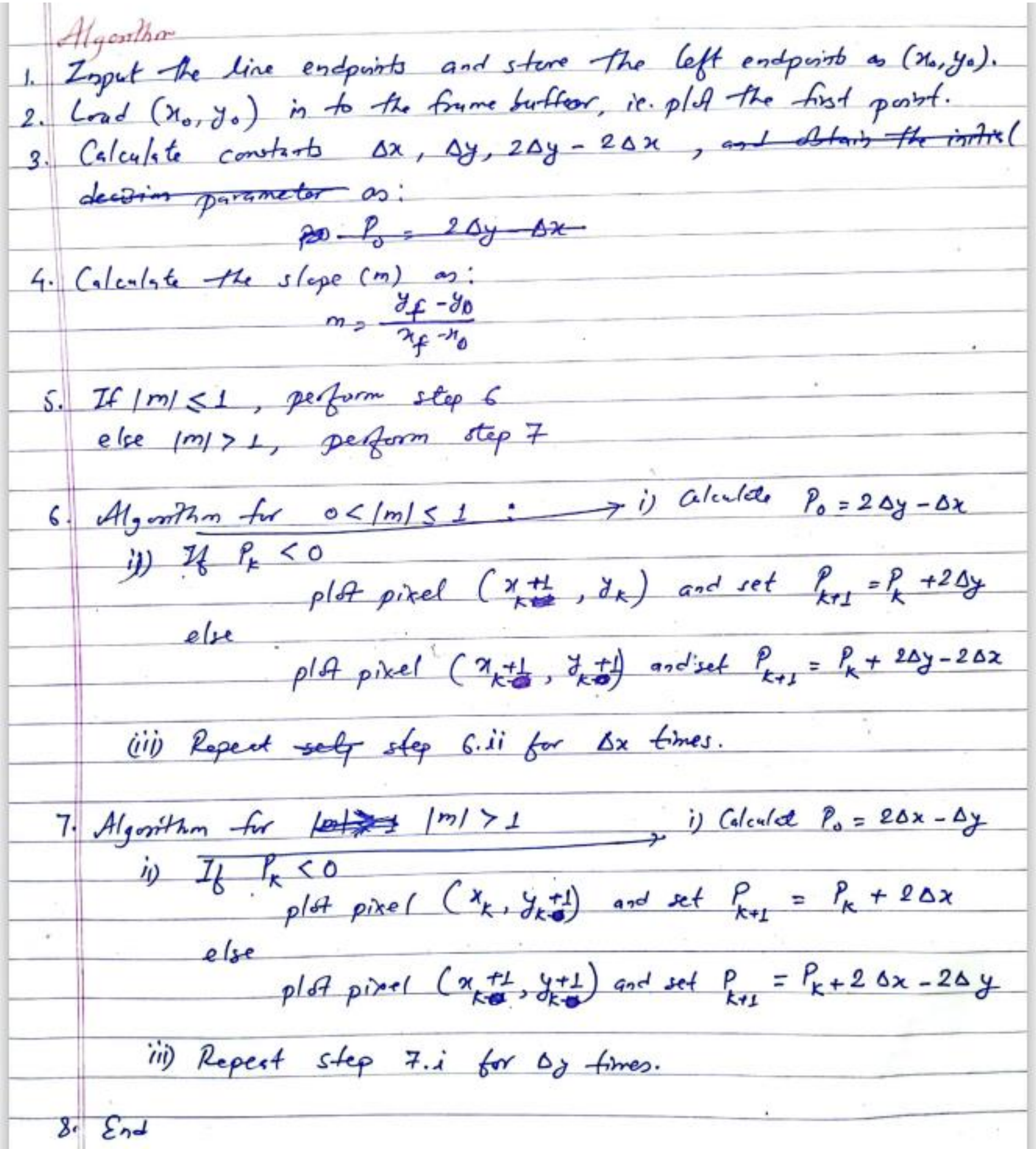
$$\boxed{\therefore P_{k+1} = P_k + 2\Delta y - 2\Delta x\,(y_{k+1} - y_k)}$$ ----(VII).

- where, $y_{k+1} - y_k$ is either 0 or 1 depending on the sign of P_k.

→ The initial decision parameter P_0, evaluated at (x_0, y_0) is given as:-

$$P_0 = 2\Delta y - \Delta x$$

- Algorithm:



Algorithm

1. Input the line endpoints and store the left endpoint as (x_0, y_0).
2. Load (x_0, y_0) in to the frame buffer, i.e. plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y - 2\Delta x$, ~~and obtain the initial decision parameter~~ as:

   ~~$P_0 = 2\Delta y - \Delta x$~~

4. Calculate the slope (m) as:

$$m = \frac{y_f - y_0}{x_f - x_0}$$

5. If $|m| \le 1$, perform step 6
   else $|m| > 1$, perform step 7

6. Algorithm for $0 < |m| \le 1$: → i) Calculate $P_0 = 2\Delta y - \Delta x$

   ii) If $P_k < 0$

   plot pixel (x_k+1, y_k) and set $P_{k+1} = P_k + 2\Delta y$

   else

   plot pixel (x_k+1, y_k+1) and set $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

   (iii) Repeat ~~self~~ step 6.ii for Δx times.

7. Algorithm for ~~[illegible]~~ $|m| > 1$ → i) Calculate $P_0 = 2\Delta x - \Delta y$

   ii) If $P_k < 0$

   plot pixel (x_k, y_k+1) and set $P_{k+1} = P_k + 2\Delta x$

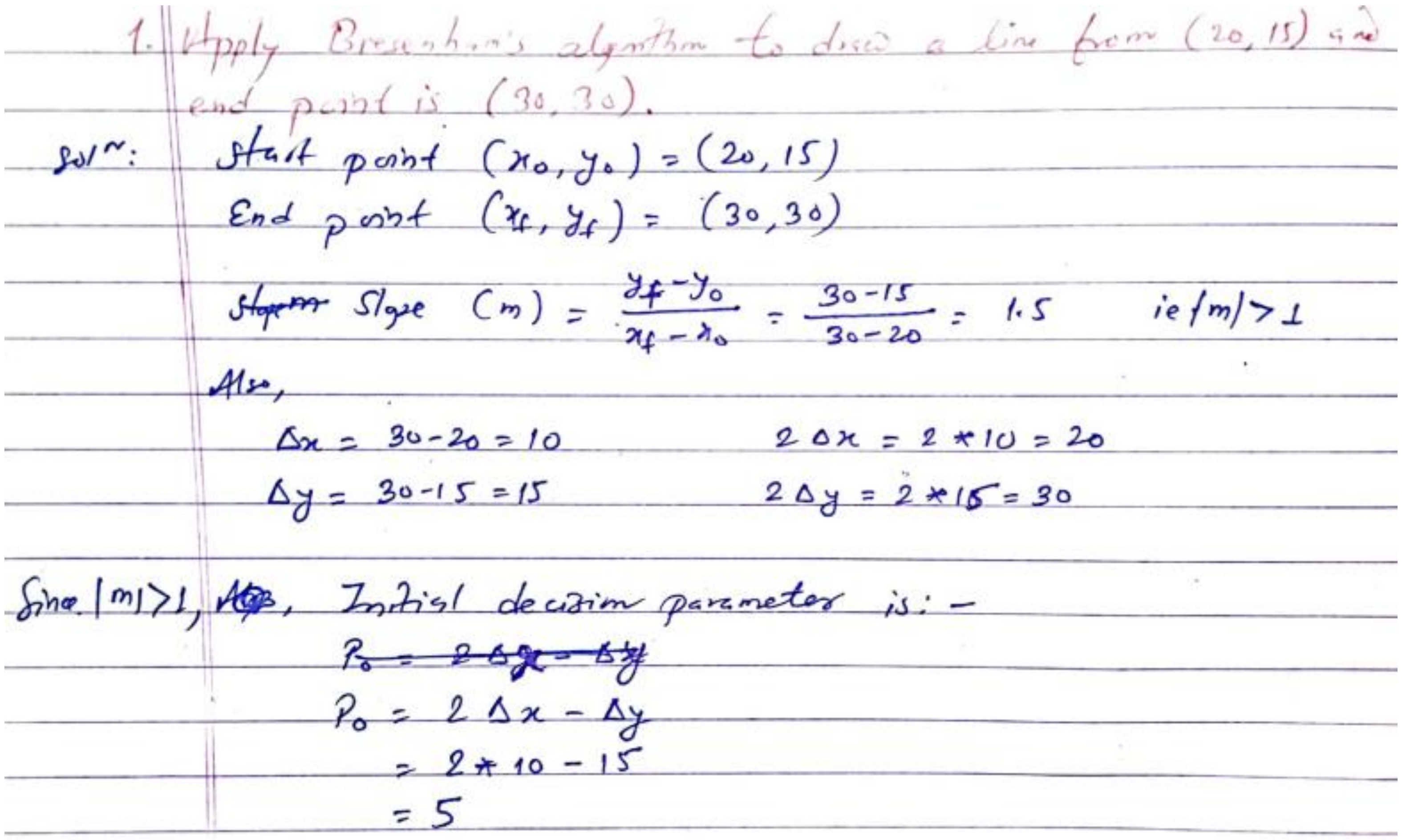
   else

   plot pixel (x_k+1, y_k+1) and set $P_{k+1} = P_k + 2\Delta x - 2\Delta y$

   iii) Repeat step 7.i for Δy times.

8. End

- Numerical:



1. Apply Bresenham's algorithm to draw a line from (20, 15) and end point is (30, 30).

Soln: Start point $(x_0, y_0) = (20, 15)$

End point $(x_f, y_f) = (30, 30)$

~~Stopm~~ Slope $(m) = \frac{y_f - y_0}{x_f - x_0} = \frac{30-15}{30-20} = 1.5$ ie $|m| > 1$

Also,

$\Delta x = 30 - 20 = 10$ $2\Delta x = 2 * 10 = 20$

$\Delta y = 30 - 15 = 15$ $2\Delta y = 2 * 15 = 30$

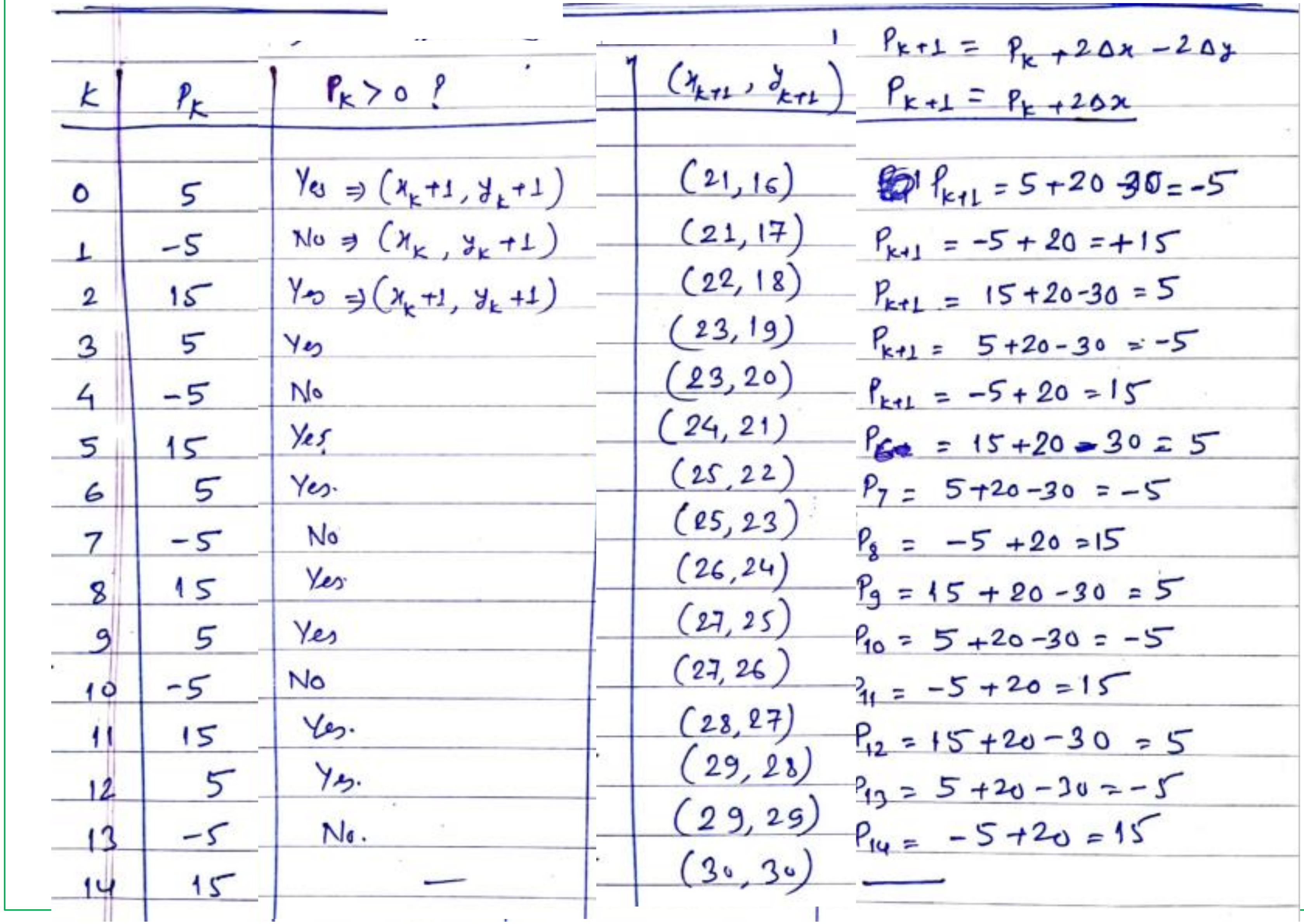
Since $|m| > 1$, ~~Also~~, Initial decision parameter is: -

~~$P_0 = 2\Delta x - \Delta y$~~

$P_0 = 2\Delta x - \Delta y$

$= 2 * 10 - 15$

$= 5$

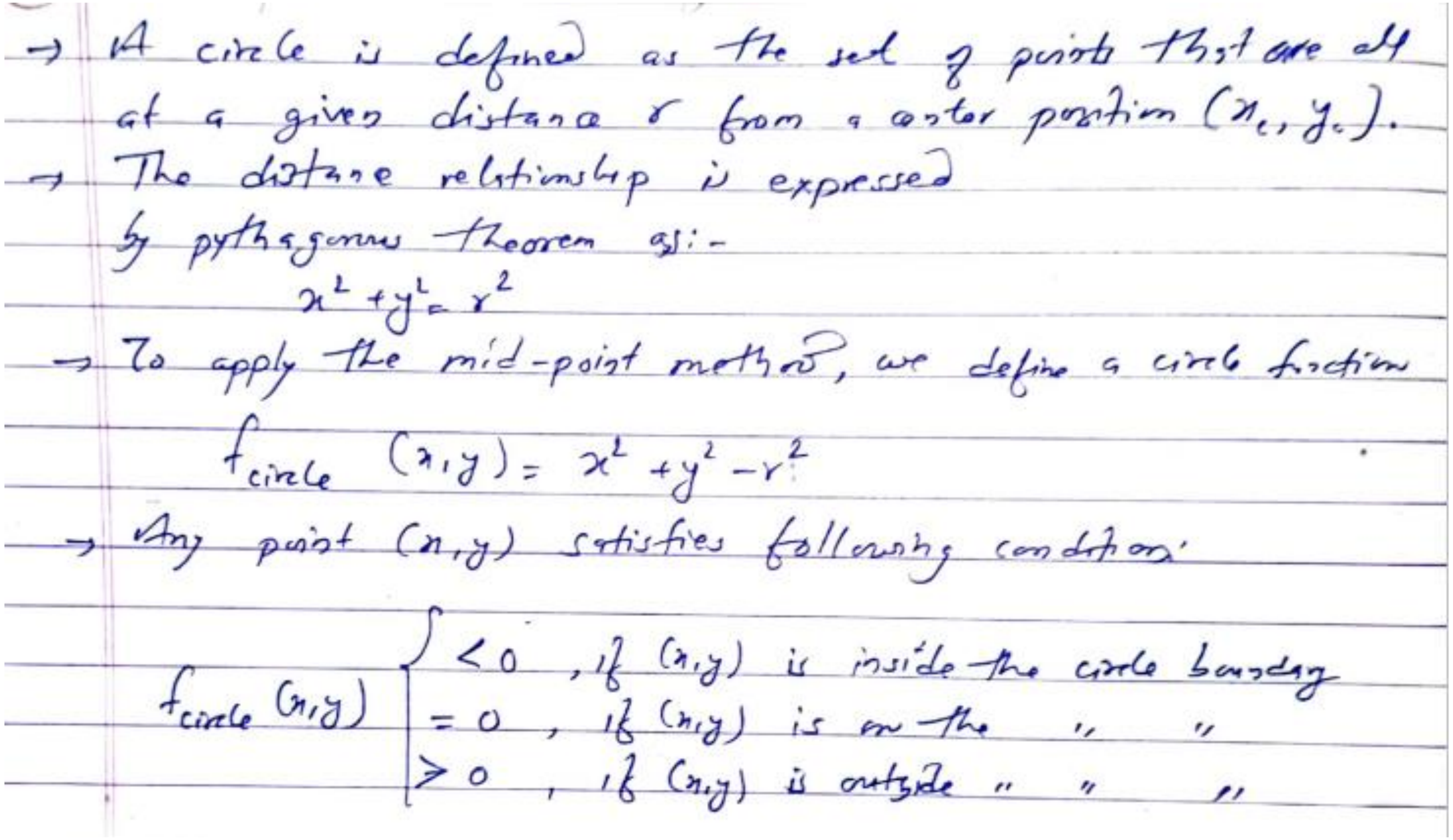


$P_{k+1} = P_k + 2\Delta x - 2\Delta y$

$P_{k+1} = P_k + 2\Delta x$

k	P_k	$P_k > 0$?	(x_{k+1}, y_{k+1})	
0	5	Yes ⇒ (x_k+1, y_k+1)	(21, 16)	$P_{k+1} = 5+20-30 = -5$
1	-5	No ⇒ (x_k, y_k+1)	(21, 17)	$P_{k+1} = -5+20 = +15$
2	15	Yes ⇒ (x_k+1, y_k+1)	(22, 18)	$P_{k+1} = 15+20-30 = 5$
3	5	Yes	(23, 19)	$P_{k+1} = 5+20-30 = -5$
4	-5	No	(23, 20)	$P_{k+1} = -5+20 = 15$
5	15	Yes	(24, 21)	$P_6 = 15+20-30 = 5$
6	5	Yes.	(25, 22)	$P_7 = 5+20-30 = -5$
7	-5	No	(25, 23)	$P_8 = -5+20 = 15$
8	15	Yes	(26, 24)	$P_9 = 15+20-30 = 5$
9	5	Yes	(27, 25)	$P_{10} = 5+20-30 = -5$
10	-5	No	(27, 26)	$P_{11} = -5+20 = 15$
11	15	Yes.	(28, 27)	$P_{12} = 15+20-30 = 5$
12	5	Yes.	(29, 28)	$P_{13} = 5+20-30 = -5$
13	-5	No.	(29, 29)	$P_{14} = -5+20 = 15$
14	15		(30, 30)	

2.3 Circle Generation Algorithm



→ A circle is defined as the set of points that are all at a given distance r from a center position (x_c, y_c).

→ The distance relationship is expressed by pythagorus theorem as:-

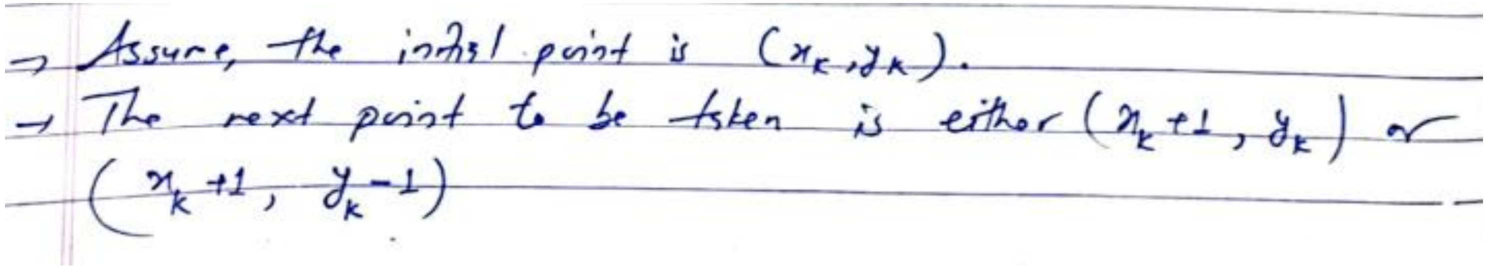
$$x^2 + y^2 = r^2$$

→ To apply the mid-point method, we define a circle function

$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

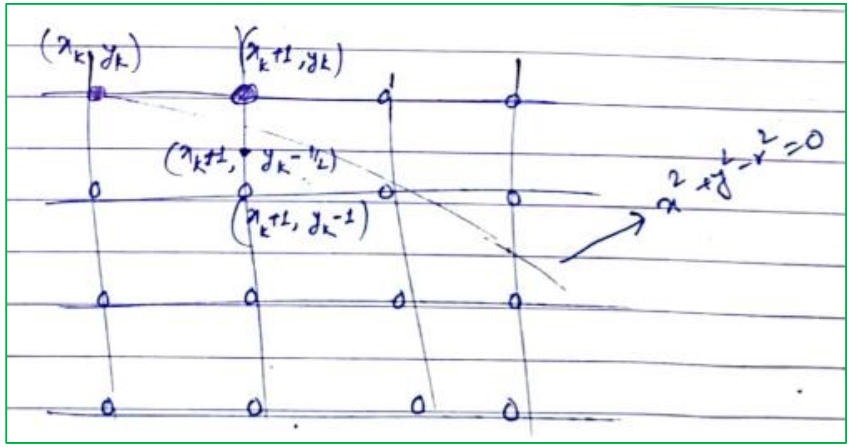
→ Any point (x,y) satisfies following conditions:

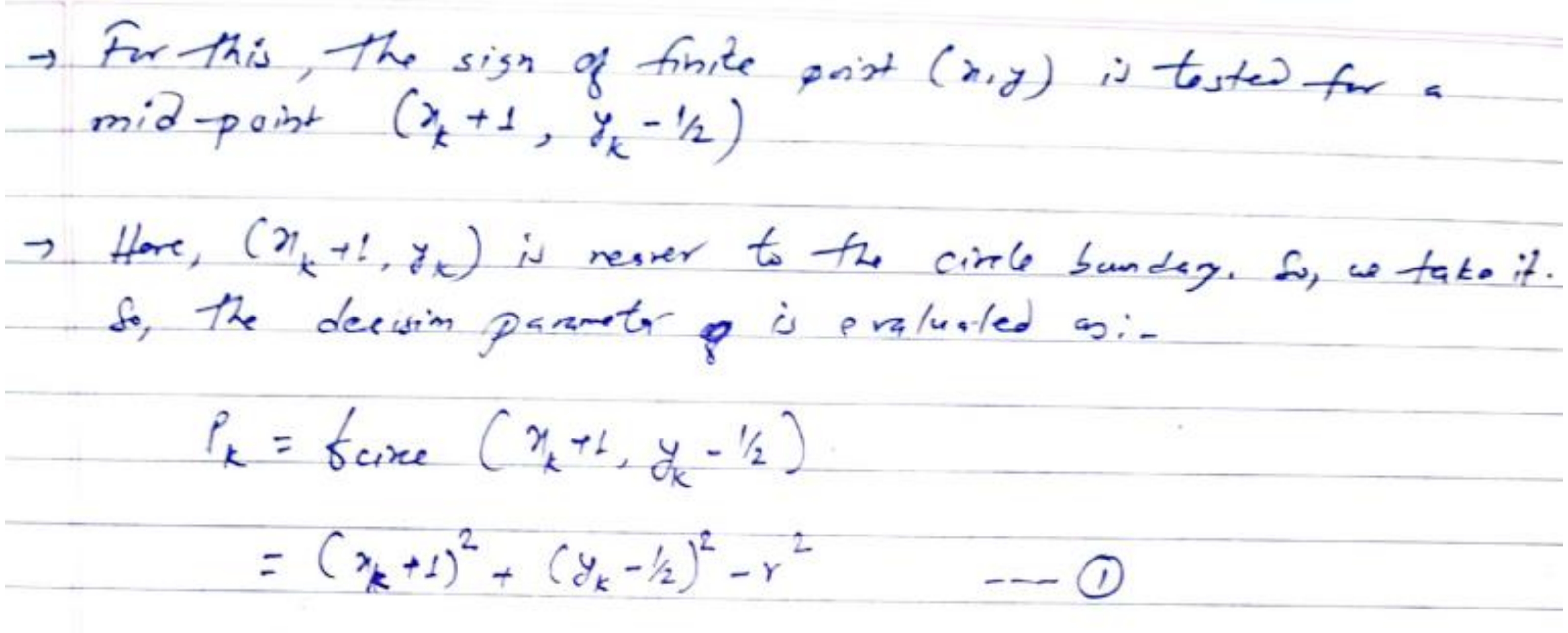
$$f_{circle}(x,y) \begin{cases} <0 & \text{, if } (x,y) \text{ is inside the circle boundary} \\ =0 & \text{, if } (x,y) \text{ is on the } \quad '' \quad '' \\ >0 & \text{, if } (x,y) \text{ is outside } \quad '' \quad '' \quad '' \end{cases}$$



→ Assume, the initial point is (x_k, y_k).

→ The next point to be taken is either (x_k+1, y_k) or (x_k+1, y_k-1)



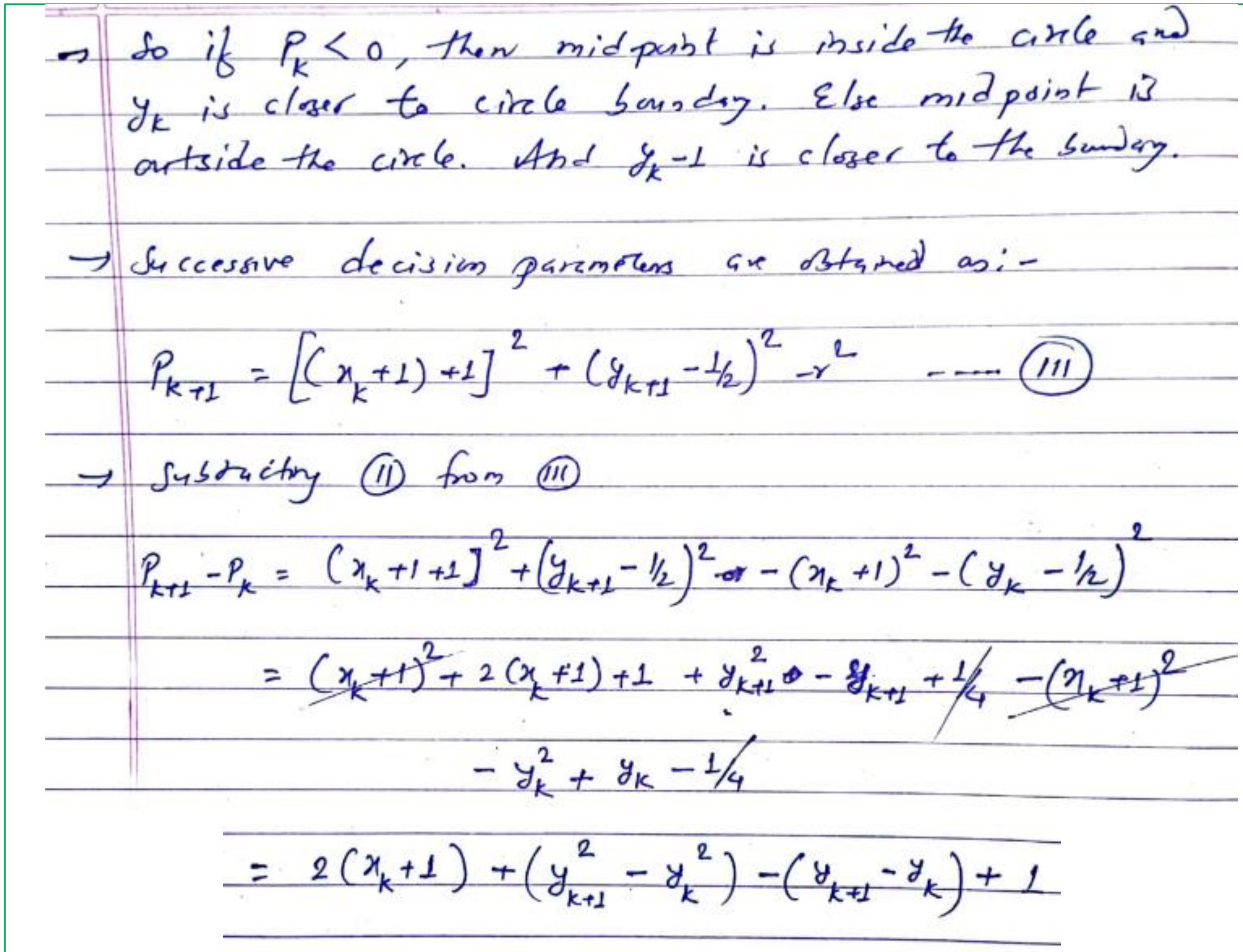


→ For this, the sign of finite point (x, y) is tested for a mid-point $(x_k+1, y_k-1/2)$

→ Here, (x_k+1, y_k) is nearer to the circle boundary. So, we take it. So, the decision parameter p is evaluated as:-

$$P_k = f_{circ}(x_k+1, y_k-1/2)$$

$$= (x_k+1)^2 + (y_k-1/2)^2 - r^2 \quad \text{---} \ (1)$$



→ So if $P_k < 0$, then midpoint is inside the circle and y_k is closer to circle boundary. Else midpoint is outside the circle. And $y_k - 1$ is closer to the boundary.

→ Successive decision parameters are obtained as:-

$$P_{k+1} = [(x_k+1)+1]^2 + (y_{k+1} - 1/2)^2 - r^2 \quad ---- \text{(III)}$$

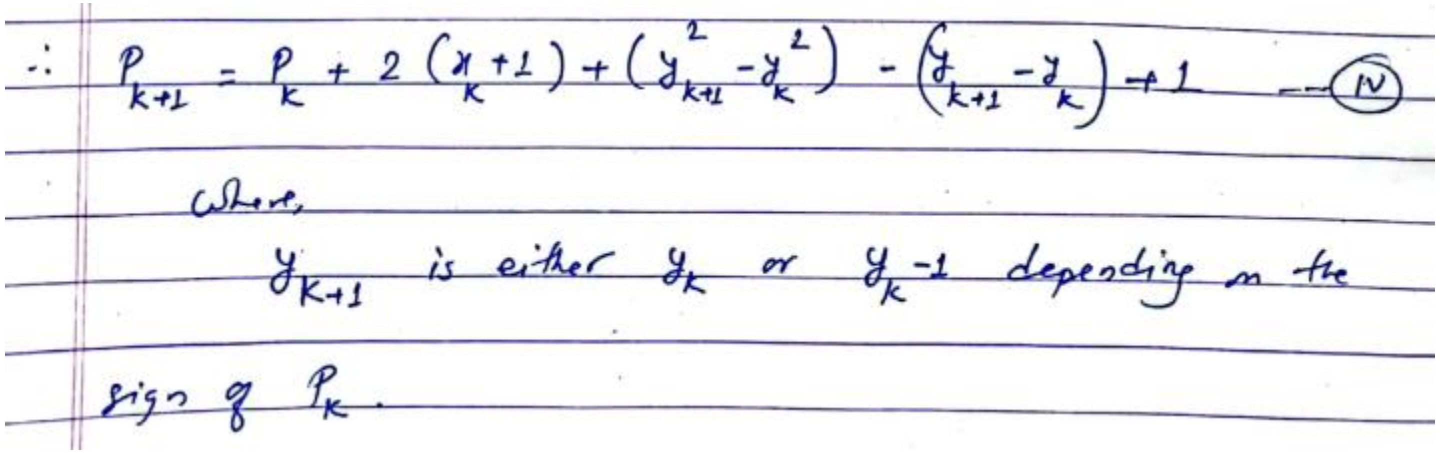
→ Substracting (II) from (III)

$$P_{k+1} - P_k = (x_k+1+1]^2 + (y_{k+1} - 1/2)^2 - (x_k+1)^2 - (y_k - 1/2)^2$$

$$= \cancel{(x_k+1)^2} + 2(x_k+1) + 1 + y_{k+1}^2 - y_{k+1} + \cancel{1/4} - \cancel{(x_k+1)^2}$$

$$- y_k^2 + y_k - \cancel{1/4}$$

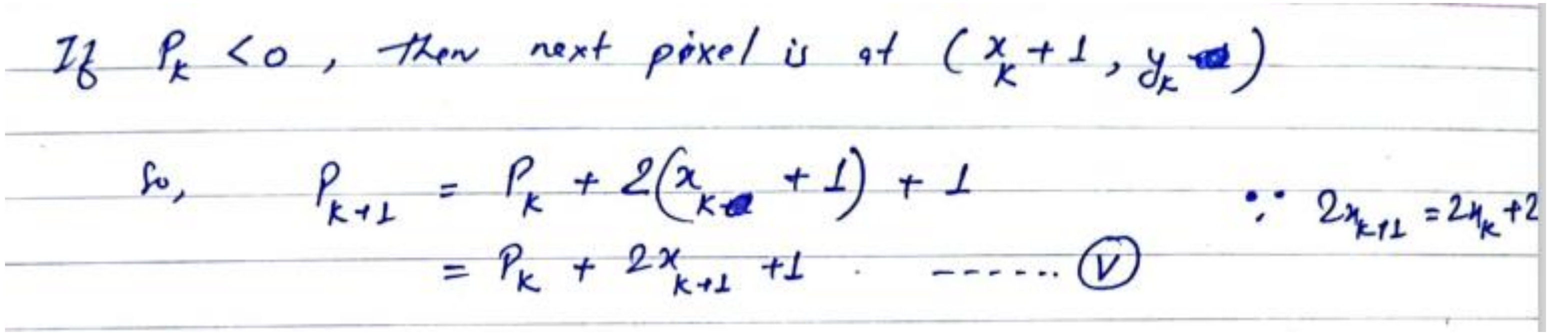
$$= 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$



$$\therefore P_{k+1} = P_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \quad \text{----} \ \text{(IV)}$$

Where,

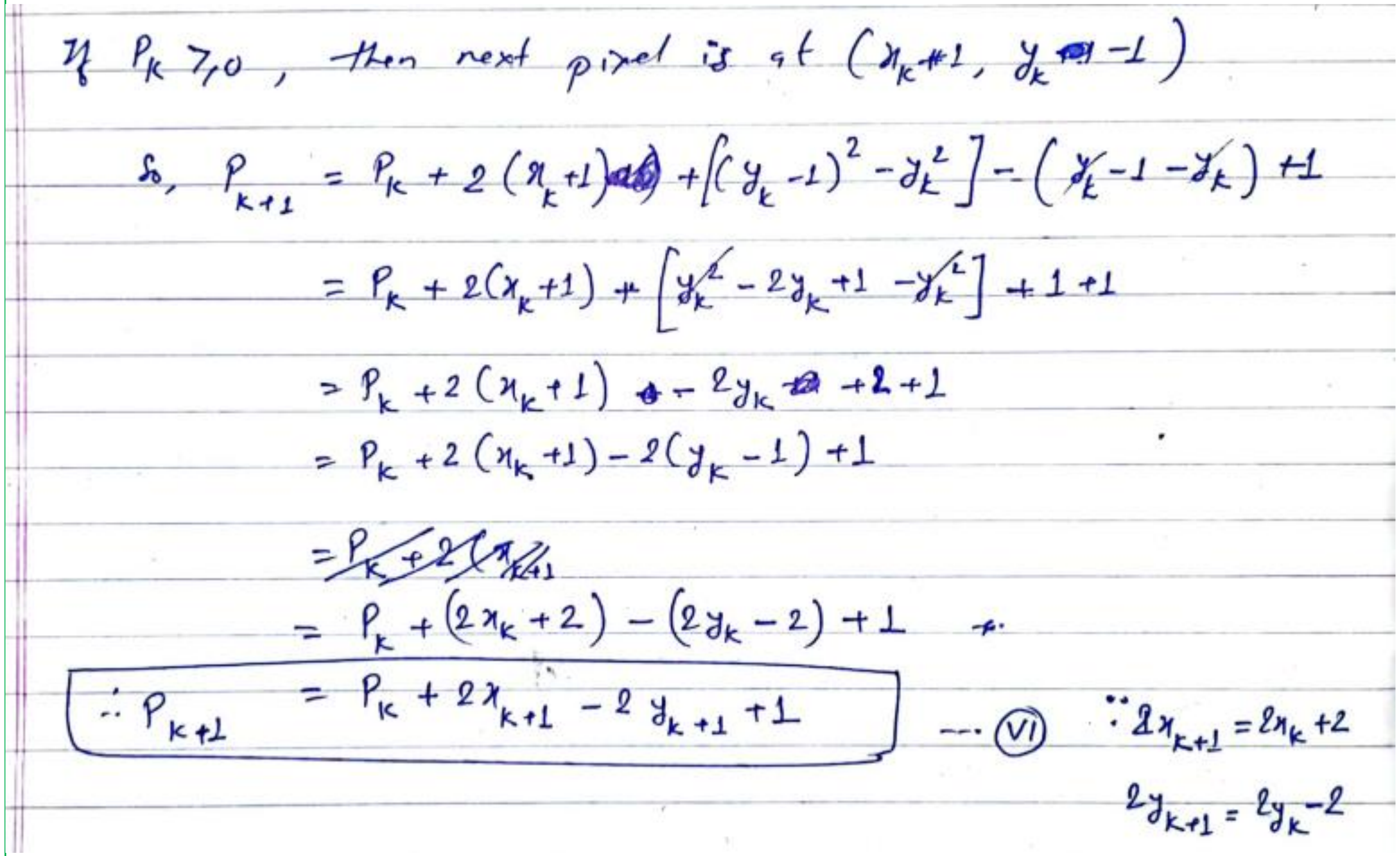
y_{k+1} is either y_k or $y_k - 1$ depending on the sign of P_k.



If $P_k < 0$, then next pixel is at $(x_k + 1, y_k)$

So, $P_{k+1} = P_k + 2(x_k + 1) + 1$ $\quad \because 2x_{k+1} = 2x_k + 2$

$= P_k + 2x_{k+1} + 1$ ------ (V)



If $P_k \geq 0$, then next pixel is at (x_k+1, y_k-1)

So, $P_{k+1} = P_k + 2(x_k+1) + [(y_k-1)^2 - y_k^2] - (\cancel{y_k} - 1 - \cancel{y_k}) + 1$

$= P_k + 2(x_k+1) + [\cancel{y_k^2} - 2y_k + 1 - \cancel{y_k^2}] + 1 + 1$

$= P_k + 2(x_k+1) - 2y_k + 2 + 1$

$= P_k + 2(x_k+1) - 2(y_k-1) + 1$

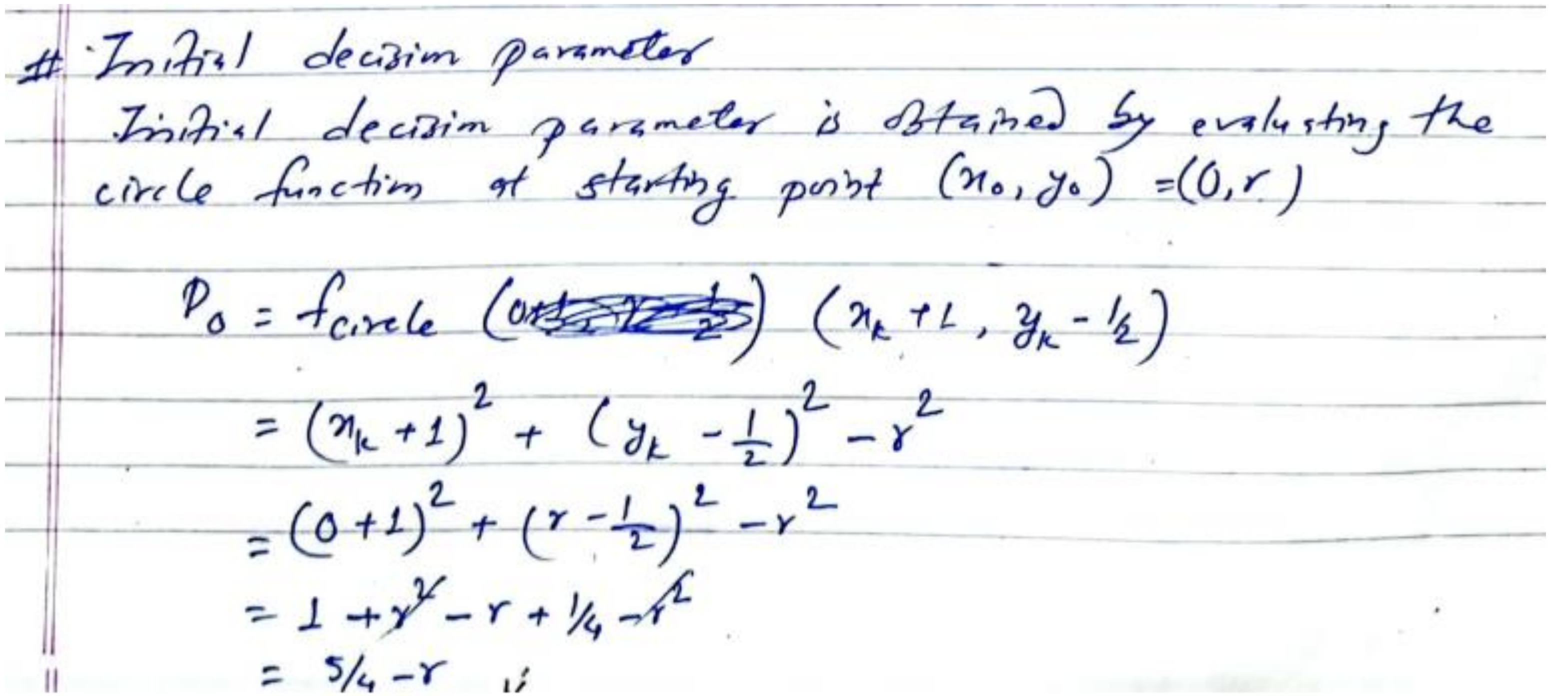
$\cancel{= P_k + 2(x_{k+1}}$

$= P_k + (2x_k+2) - (2y_k-2) + 1$

$\boxed{\therefore P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1}$ --- (VI)

$\because 2x_{k+1} = 2x_k + 2$

$2y_{k+1} = 2y_k - 2$



Initial decision parameter

Initial decision parameter is obtained by evaluating the circle function at starting point $(x_0, y_0) = (0, r)$

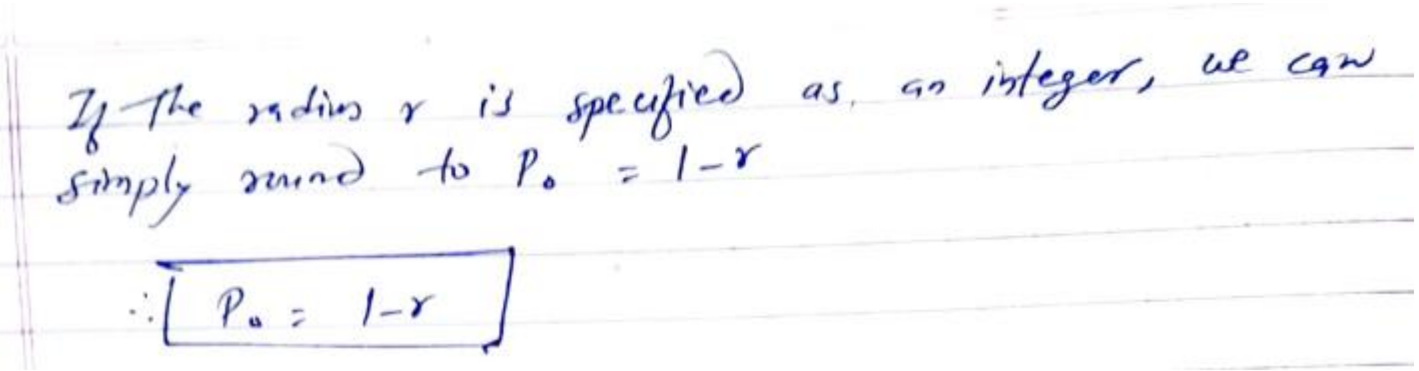
$$P_0 = f_{circle}\,(0\ \text{[illegible]})\ (x_k + 1,\ y_k - \tfrac{1}{2})$$

$$= (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$$

$$= (0+1)^2 + (r - \frac{1}{2})^2 - r^2$$

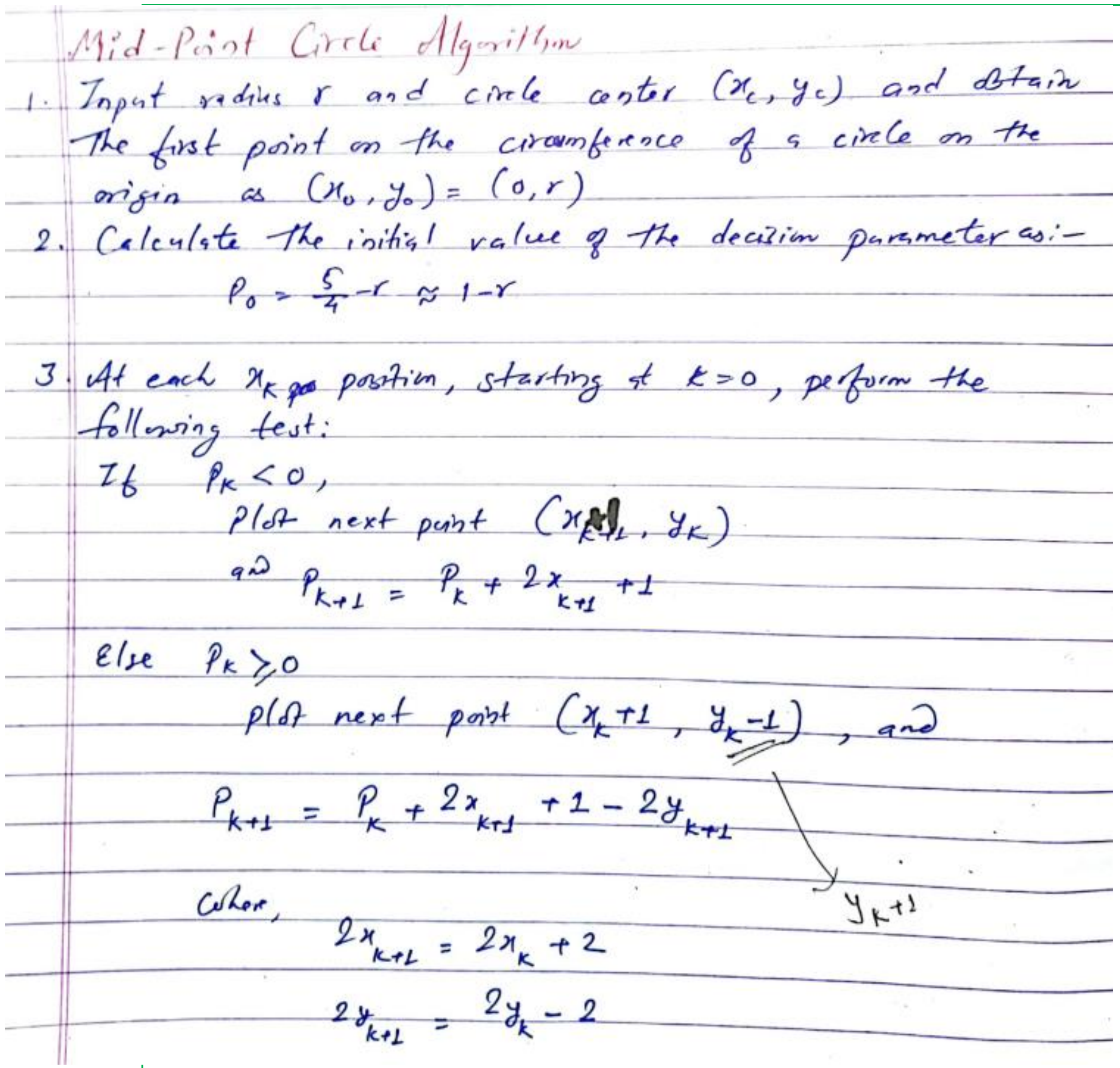
$$= 1 + \cancel{r^2} - r + 1/4 - \cancel{r^2}$$

$$= 5/4 - r$$



If the radius r is specified as, an integer, we can simply round to $P_0 = 1 - r$

$$\therefore \boxed{P_0 = 1 - r}$$



Mid-Point Circle Algorithm

1. Input radius r and circle center (x_c, y_c) and obtain The first point on the circumference of a circle on the origin as $(x_0, y_0) = (0, r)$

2. Calculate the initial value of the decision parameter as:-

$$P_0 = \frac{5}{4} - r \approx 1 - r$$

3. At each x_k position, starting at $k=0$, perform the following test:

If $P_K < 0$,
plot next point (x_{k+1}, y_K)
and $P_{K+1} = P_K + 2x_{K+1} + 1$

Else $P_K \geq 0$
plot next point $(x_k + 1, y_k - 1)$, and

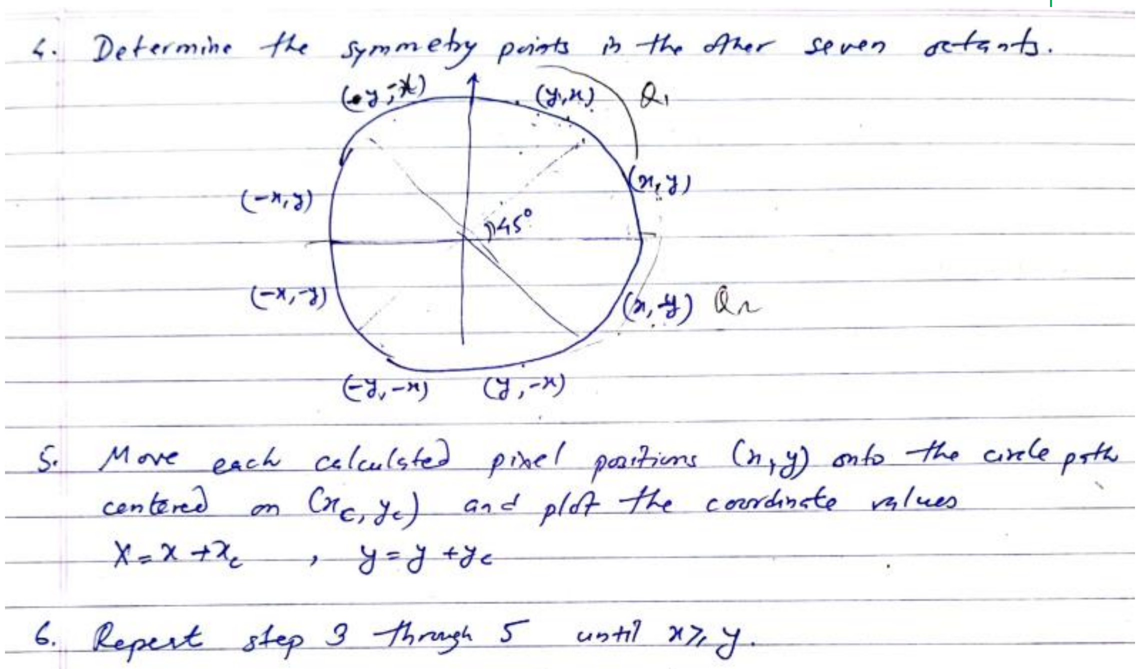
$$P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1}$$

$y_k + 1$

Where,

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k - 2$$



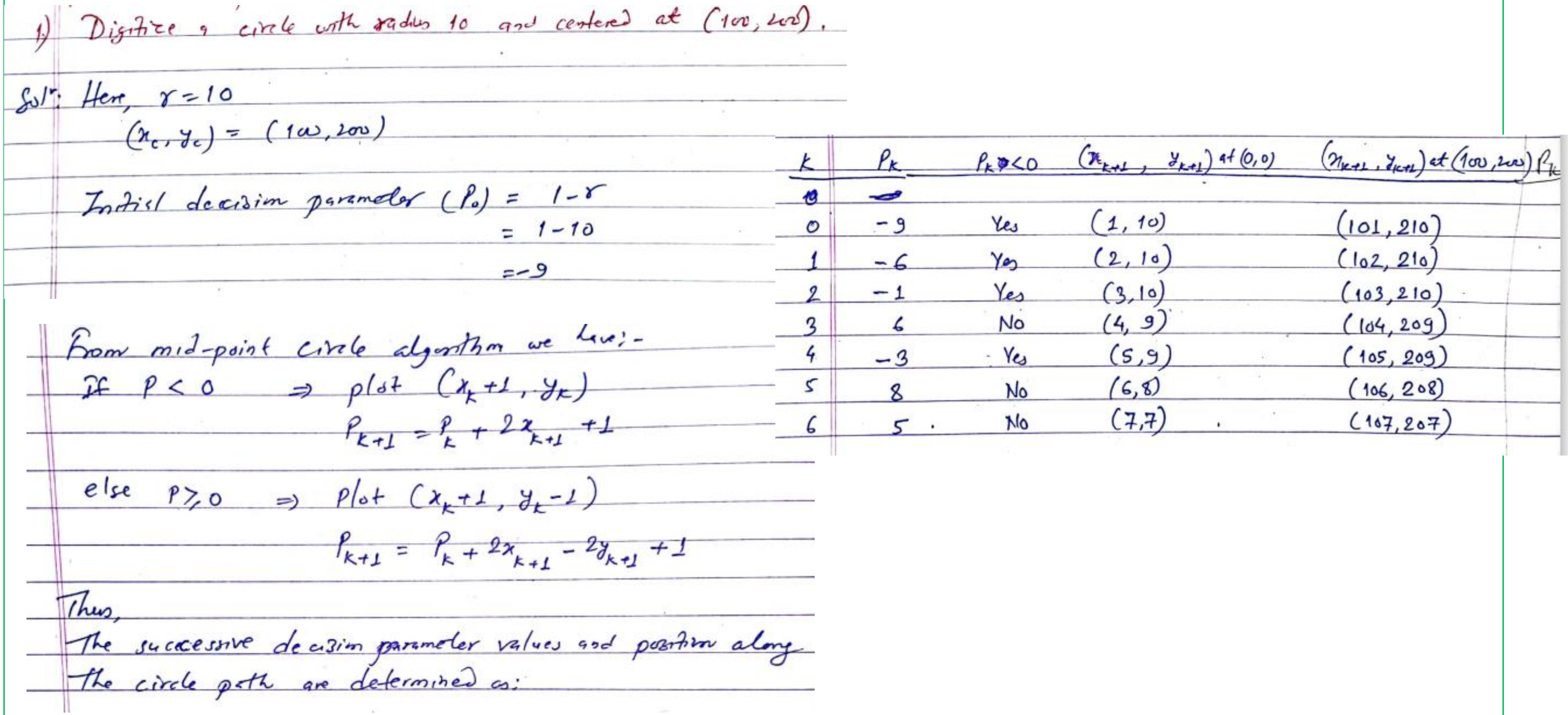
4. Determine the symmetry points in the other seven octants.

(-y, x)
(y, x)
Q_1
(x, y)
(-x, y)
45°
(-x, -y)
(x, -y)
Q_2
(-y, -x)
(y, -x)

5. Move each calculated pixel positions (x, y) onto the circle path centered on (x_c, y_c) and plot the coordinate values
$X = x + x_c$, $y = y + y_c$

6. Repeat step 3 through 5 until $x \geq y$.

- **Numerical:**



1) Digitize a circle with radius 10 and centered at (100, 200).

Solⁿ: Here, $r = 10$

$(x_c, y_c) = (100, 200)$

Initial decision parameter $(P_0) = 1 - r$

$= 1 - 10$

$= -9$

From mid-point circle algorithm we have:-

If $P < 0 \Rightarrow$ plot $(x_k + 1, y_k)$

$P_{k+1} = P_k + 2x_{k+1} + 1$

else $P \geq 0 \Rightarrow$ plot $(x_k + 1, y_k - 1)$

$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

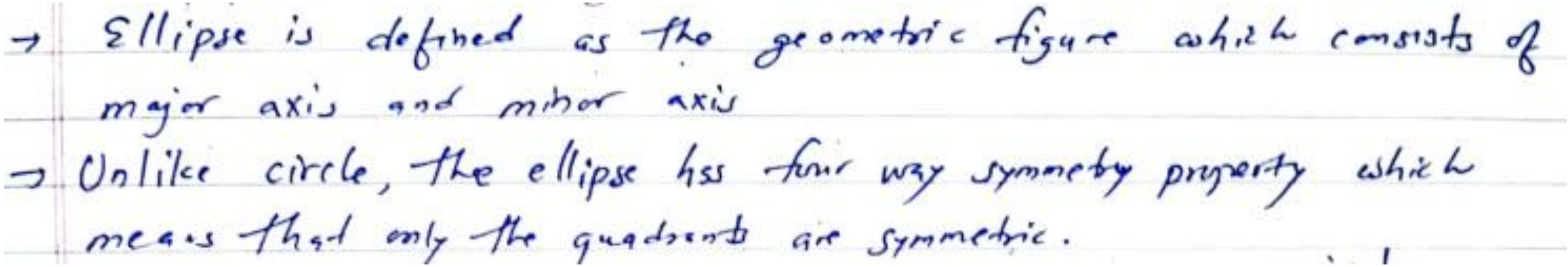
Thus,

The successive decision parameter values and position along the circle path are determined as:

k	P_k	$P_k < 0$	(x_{k+1}, y_{k+1}) at (0,0)	(x_{k+1}, y_{k+1}) at (100, 200)	P_k
0	-9	Yes	(1, 10)	(101, 210)	
1	-6	Yes	(2, 10)	(102, 210)	
2	-1	Yes	(3, 10)	(103, 210)	
3	6	No	(4, 9)	(104, 209)	
4	-3	Yes	(5, 9)	(105, 209)	
5	8	No	(6, 8)	(106, 208)	
6	5	No	(7, 7)	(107, 207)	

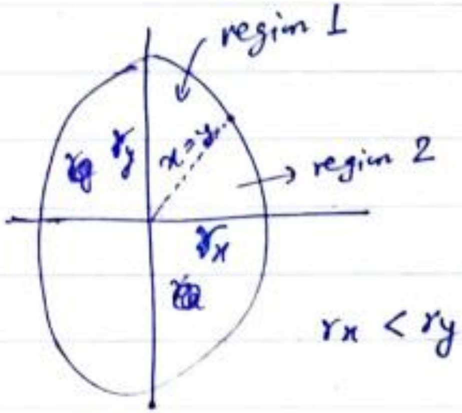
2.4 Ellipse Generation Algorithm

- **Mid-point Ellipse algorithm:**



→ Ellipse is defined as the geometric figure which consists of major axis and minor axis

→ Unlike circle, the ellipse has four way symmetry property which means that only the quadrants are symmetric.



→ The mid-point ellipse drawing algorithm is used to calculate all the perimeter points of an ellipse.

→ In this algorithm, the mid-point between the two pixels is calculated which helps in calculating the decision parameter (p).

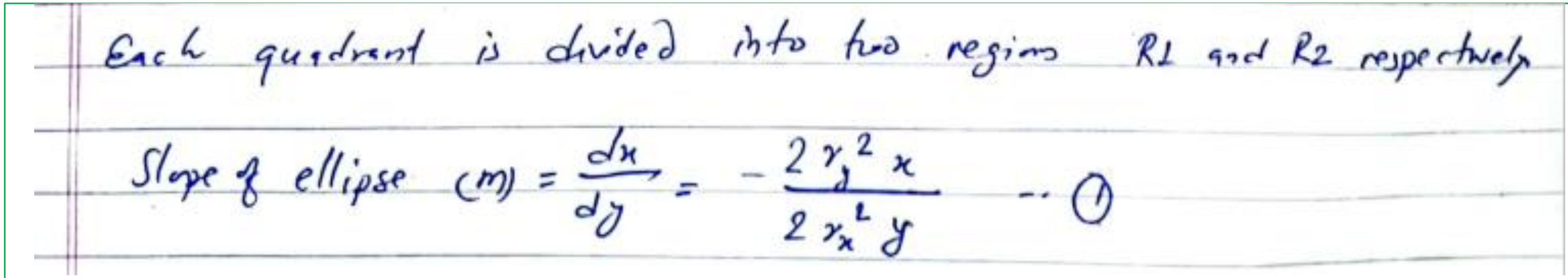
→ The value of p determines whether the mid-point lies inside, outside or on the elipse boundary.

→ Then position of this mid-point helps in drawing the ellipse.

→ Let, r_x = semi-major axis

r_y = semi-minor axis

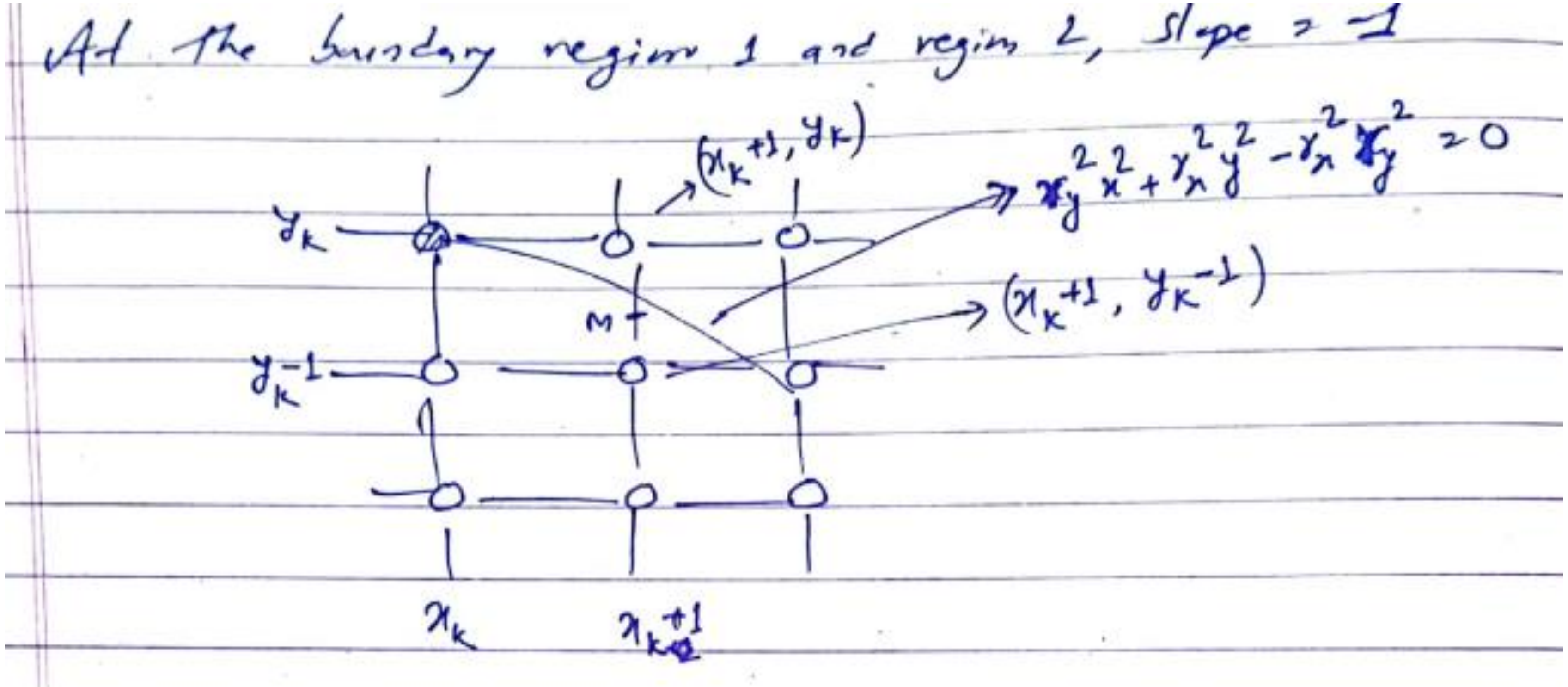
equation of ellipse is: $r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 = 0$ ----- ①



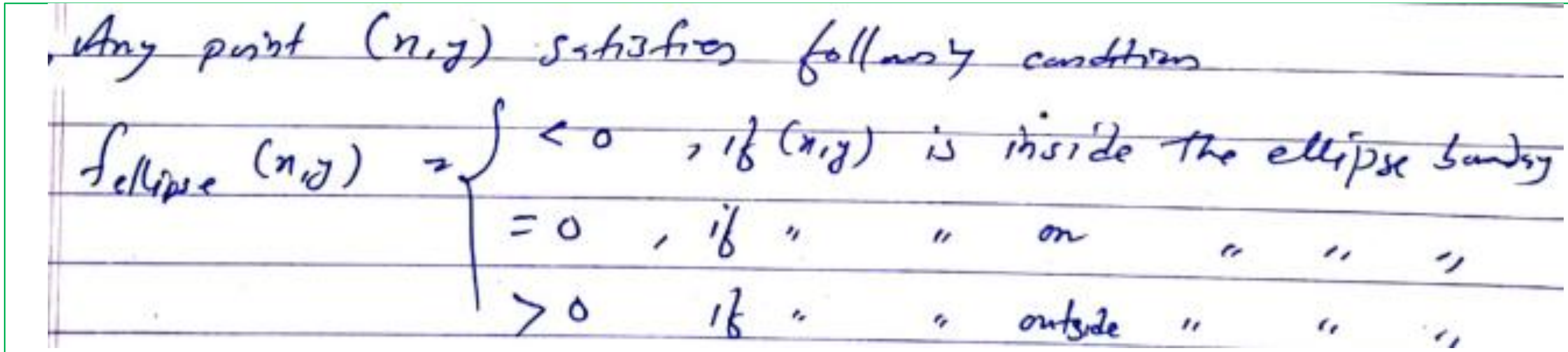
Each quadrant is divided into two regions R1 and R2 respectively

Slope of ellipse (m) = $\frac{dx}{dy} = -\frac{2r_y^2 x}{2r_x^2 y}$ -- ①

Region 1:

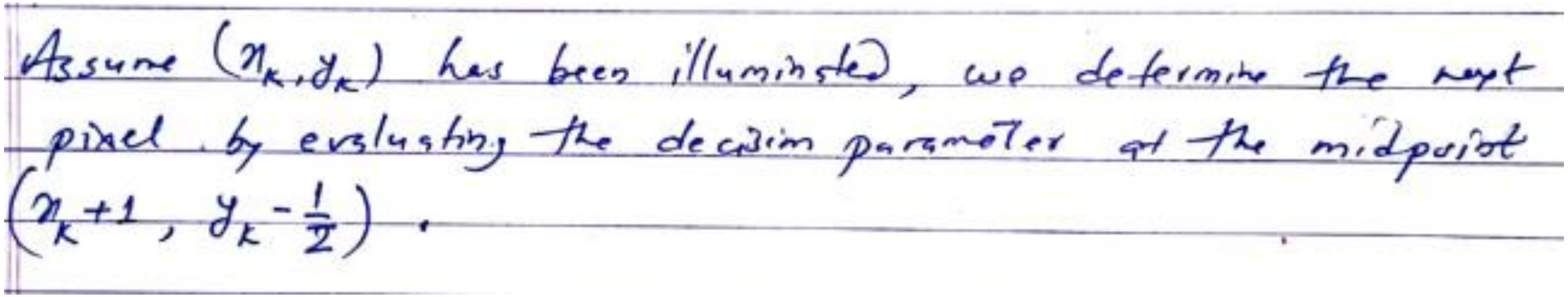


At the boundary region 1 and region 2, slope = -1

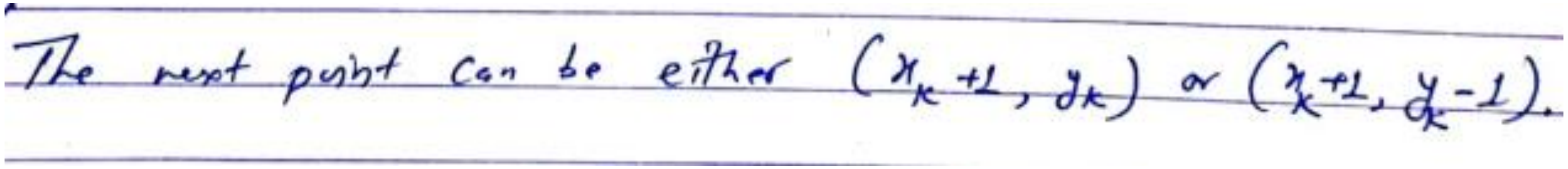


Any point (x,y) satisfies following conditions

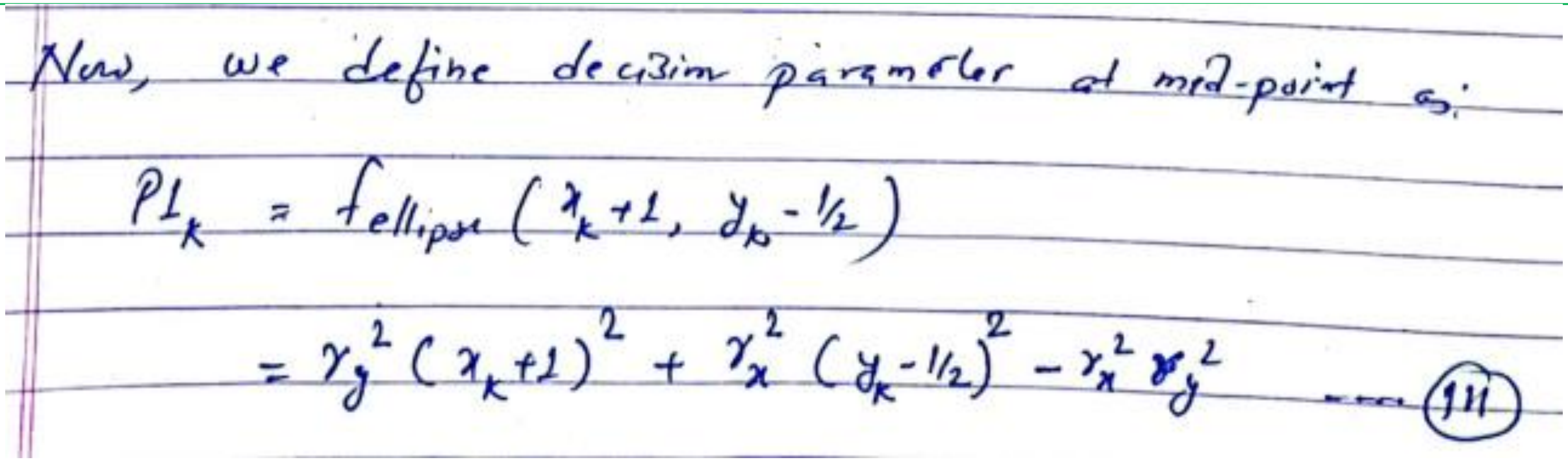
$$f_{ellipse}(x,y) = \begin{cases} <0 & \text{, if } (x,y) \text{ is inside the ellipse boundary} \\ =0 & \text{, if '' '' on '' '' ''} \\ >0 & \text{ if '' '' outside '' '' ''} \end{cases}$$



Assume (x_k, y_k) has been illuminated, we determine the next pixel by evaluating the decision parameter at the midpoint $(x_k+1, y_k-\frac{1}{2})$.



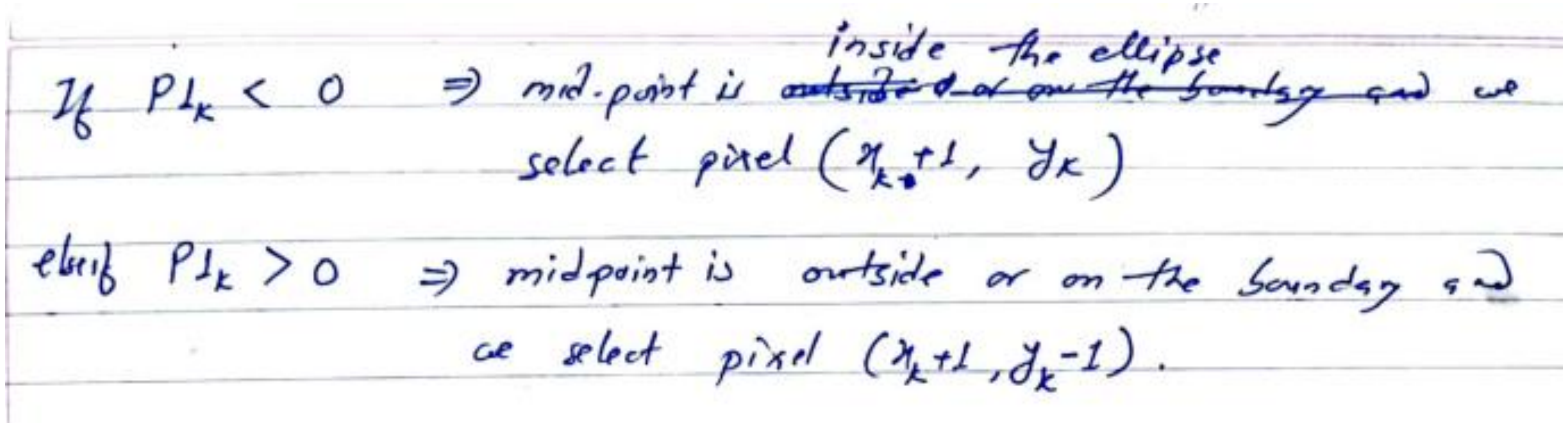
The next point can be either (x_k+1, y_k) or (x_k+1, y_k-1).



Now, we define decision parameter at mid-point as:

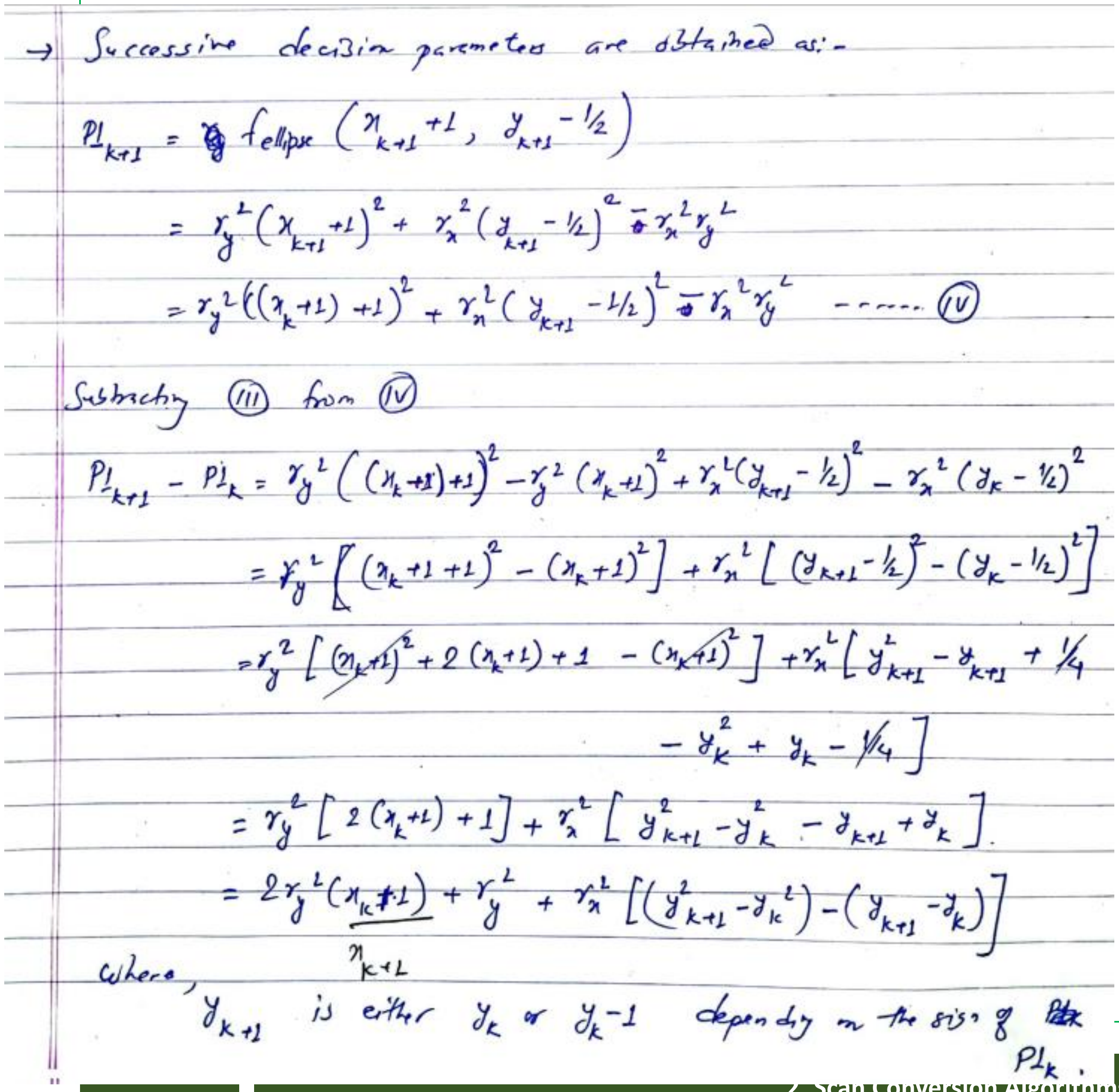
$$P1_k = f_{ellipse}(x_k+1,\ y_k-1/2)$$

$$= r_y^2(x_k+1)^2 + r_x^2(y_k-1/2)^2 - r_x^2 r_y^2 \quad \ldots\ldots \boxed{11}$$



If $P1_k < 0$ ⇒ mid-point is inside the ellipse ~~outside or on the boundary and~~ we select pixel $(x_k+1,\ y_k)$

elseif $P1_k > 0$ ⇒ midpoint is outside or on the boundary and we select pixel $(x_k+1,\ y_k-1)$.



→ Successive decision parameters are obtained as:-

$$P1_{k+1} = f_{ellipse}\left(x_{k+1}+1,\ y_{k+1}-\tfrac{1}{2}\right)$$

$$= r_y^2(x_{k+1}+1)^2 + r_x^2(y_{k+1}-\tfrac{1}{2})^2 - r_x^2 r_y^2$$

$$= r_y^2((x_k+1)+1)^2 + r_x^2(y_{k+1}-1/2)^2 - r_x^2 r_y^2 \quad \text{------} \ (IV)$$

Subtracting (III) from (IV)

$$P1_{k+1} - P1_k = r_y^2\left((x_k+1)+1\right)^2 - r_y^2(x_k+1)^2 + r_x^2(y_{k+1}-\tfrac{1}{2})^2 - r_x^2(y_k-\tfrac{1}{2})^2$$

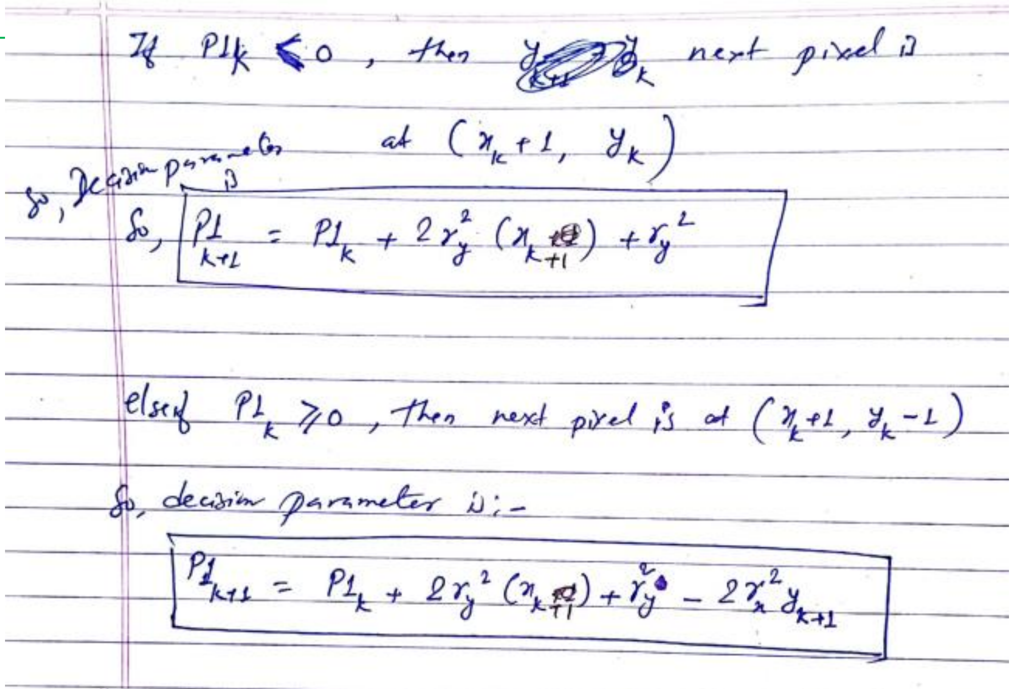
$$= r_y^2\left[(x_k+1+1)^2 - (x_k+1)^2\right] + r_x^2\left[(y_{k+1}-\tfrac{1}{2})^2 - (y_k-\tfrac{1}{2})^2\right]$$

$$= r_y^2\left[\cancel{(x_k+1)^2} + 2(x_k+1) + 1 - \cancel{(x_k+1)^2}\right] + r_x^2\left[y_{k+1}^2 - y_{k+1} + \tfrac{1}{4} - y_k^2 + y_k - \tfrac{1}{4}\right]$$

$$= r_y^2\left[2(x_k+1)+1\right] + r_x^2\left[y_{k+1}^2 - y_k^2 - y_{k+1} + y_k\right]$$

$$= 2r_y^2\underbrace{(x_k+1)}_{x_{k+1}} + r_y^2 + r_x^2\left[(y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)\right]$$

Where, y_{k+1} is either y_k or y_k-1 depending on the sign of $P1_k$.



If $P1_k < 0$, then y_k next pixel is at $(x_k+1,\ y_k)$

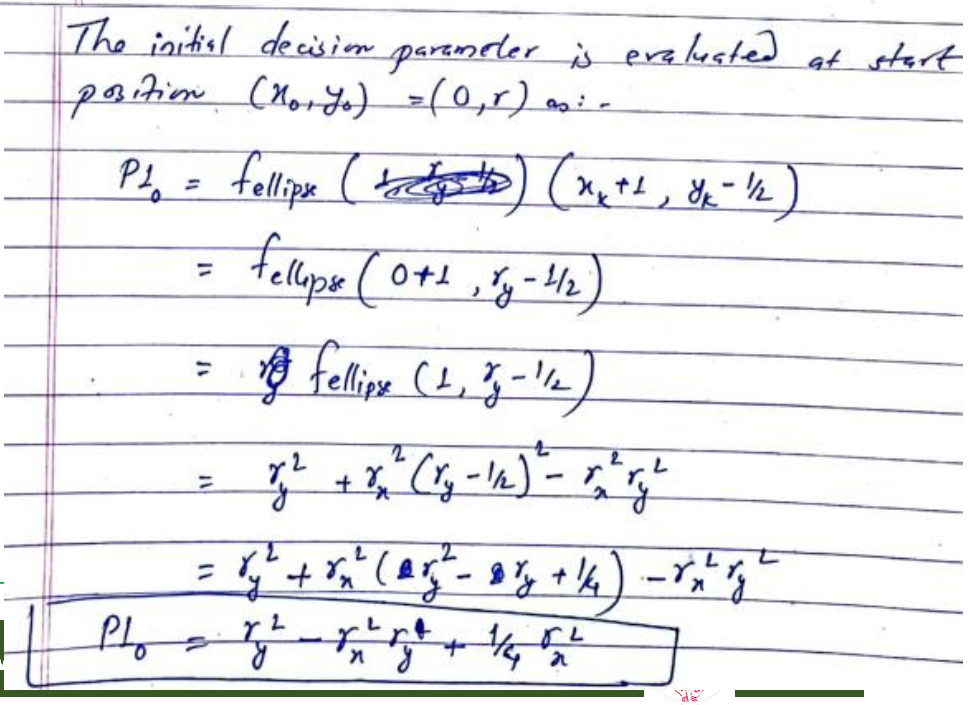
So, Decision parameter is

$$\text{So, } \boxed{P1_{k+1} = P1_k + 2r_y^2(x_{k+1}) + r_y^2}$$

else if $P1_k \geq 0$, Then next pixel is at $(x_k+1,\ y_k-1)$

So, decision parameter is:-

$$\boxed{P1_{k+1} = P1_k + 2r_y^2(x_{k+1}) + r_y^2 - 2r_x^2 y_{k+1}}$$



The initial decision parameter is evaluated at start position $(x_0, y_0) = (0, r)$ as:-

$$P1_0 = f_{ellipse}\left(x_k+1,\ y_k-\tfrac{1}{2}\right)$$

$$= f_{ellipse}\left(0+1,\ r_y-1/2\right)$$

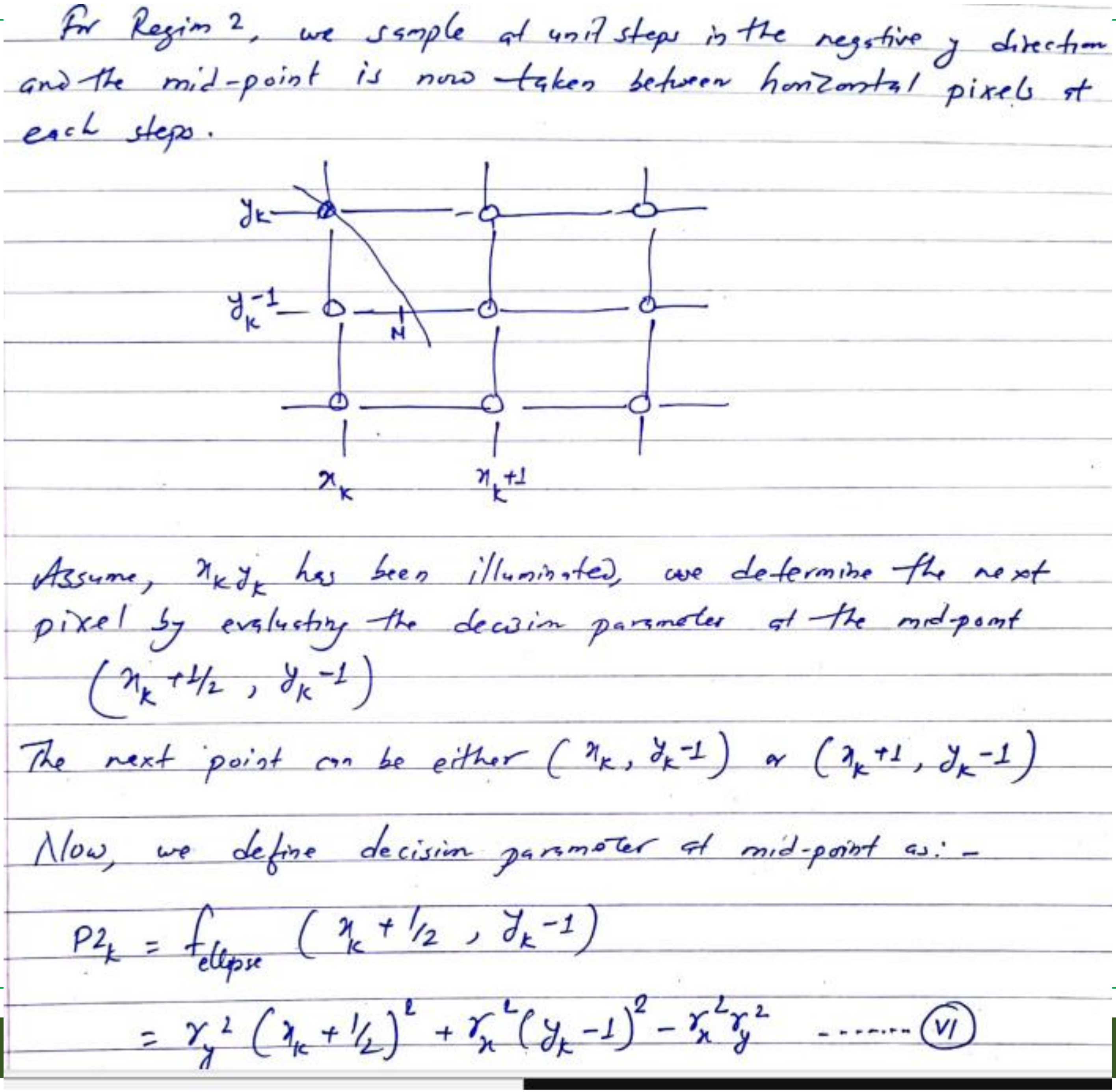
$$= f_{ellipse}\left(1,\ r_y-\tfrac{1}{2}\right)$$

$$= r_y^2 + r_x^2(r_y-\tfrac{1}{2})^2 - r_x^2 r_y^2$$

$$= r_y^2 + r_x^2(r_y^2 - r_y + \tfrac{1}{4}) - r_x^2 r_y^2$$

$$\boxed{P1_0 = r_y^2 - r_x^2 r_y + \tfrac{1}{4} r_x^2}$$

- **Region 2:**



For Regim 2, we sample at unit steps in the negative y direction and the mid-point is now taken between horizontal pixels at each steps.

y_k
y_k-1
M
x_k
x_k+1

Assume, $x_k y_k$ has been illuminated, we determine the next pixel by evaluating the decision parameter at the mid-point

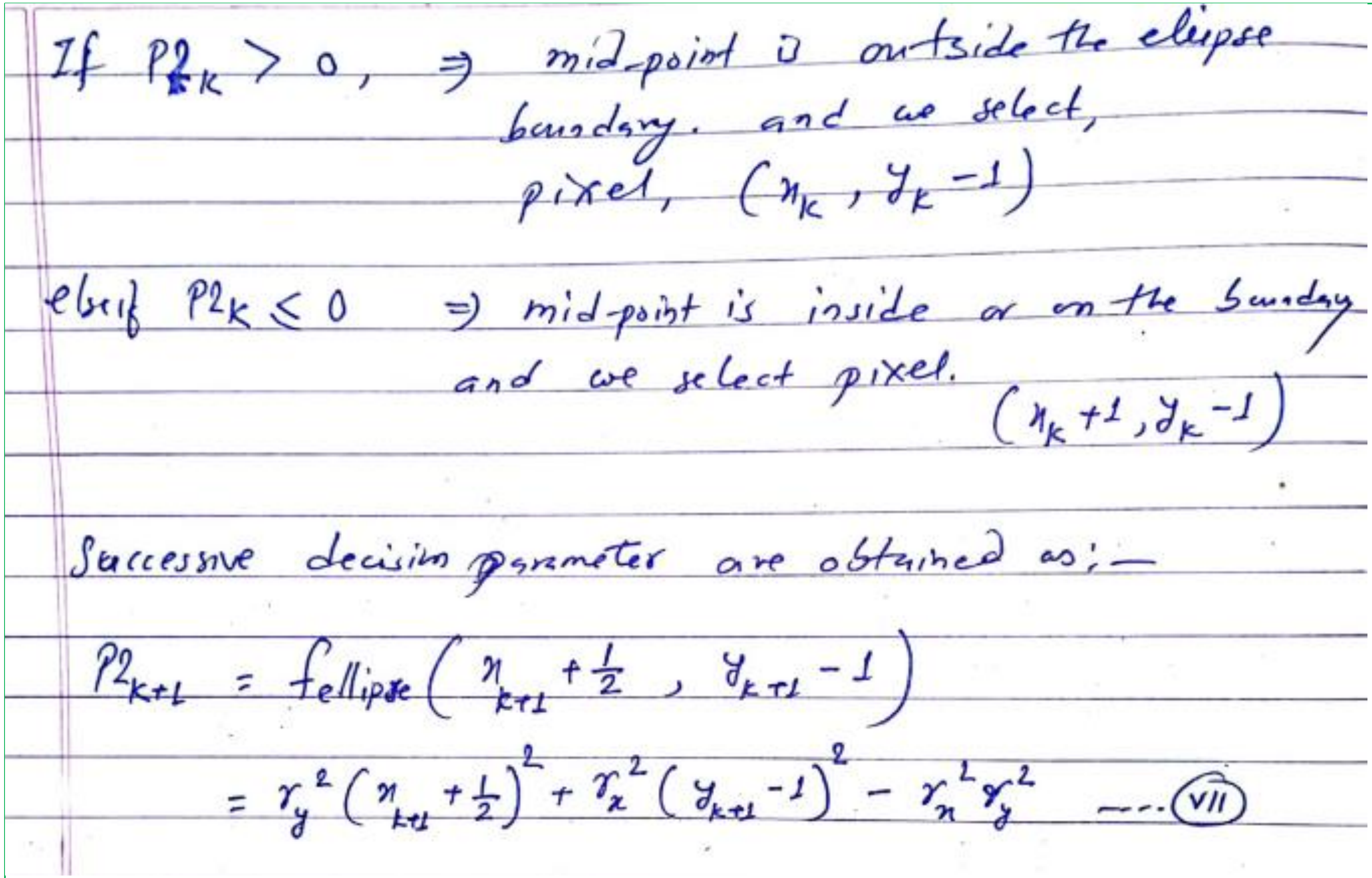
$(x_k+1/2, y_k-1)$

The next point can be either (x_k, y_k-1) or (x_k+1, y_k-1)

Now, we define decision parameter at mid-point as:-

$$P2_k = f_{ellipse}\left(x_k+1/2,\ y_k-1\right)$$

$$= r_y^2\left(x_k+1/2\right)^2 + r_x^2\left(y_k-1\right)^2 - r_x^2 r_y^2 \quad \ldots\ldots \text{(VI)}$$



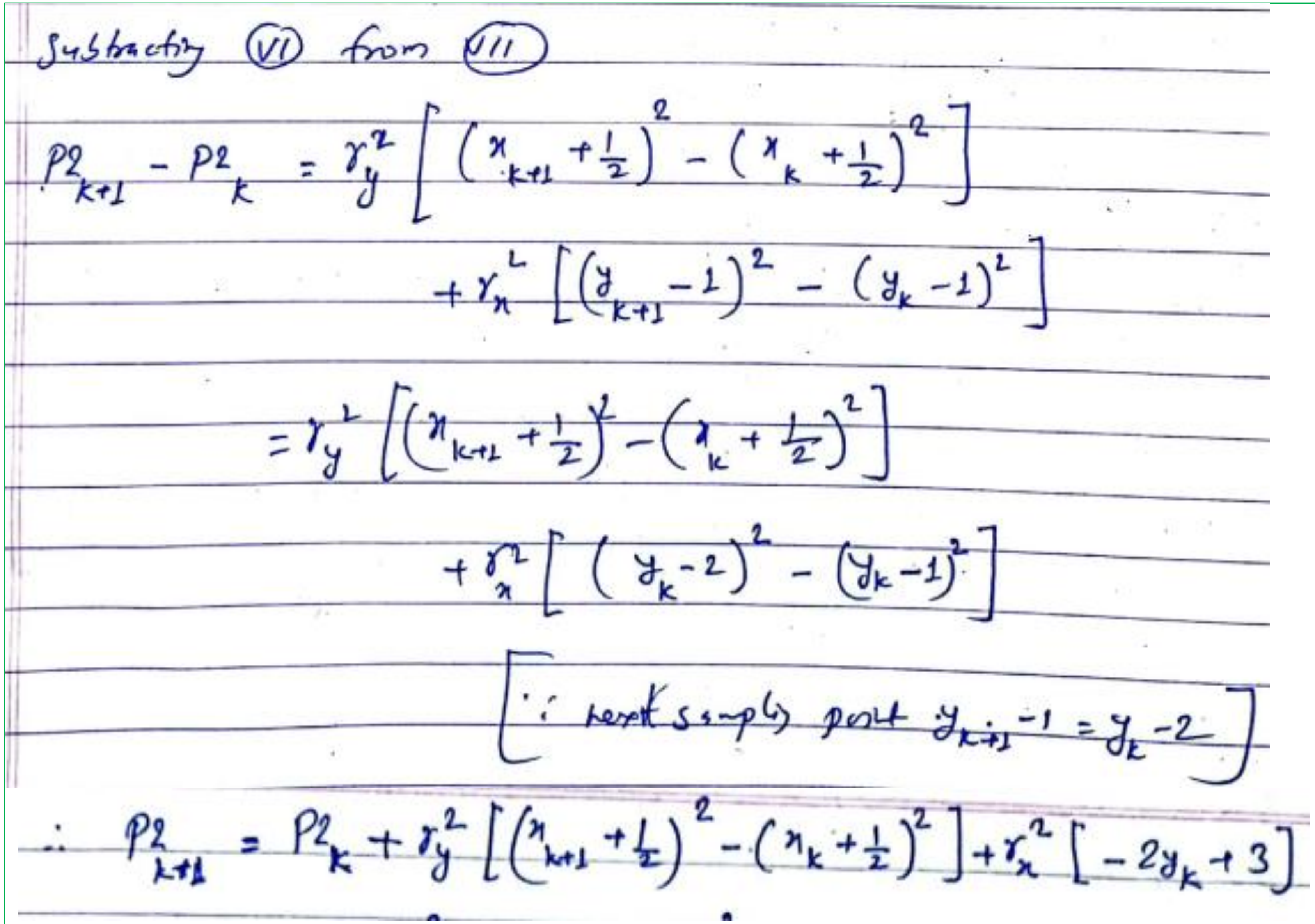
If $P2_K > 0$, ⇒ mid-point is outside the ellipse boundary. and we select, pixel, $(x_K, y_K - 1)$

elseif $P2_K \le 0$ ⇒ mid-point is inside or on the boundary and we select pixel. $(x_K + 1, y_K - 1)$

Successive decision parameter are obtained as:—

$$P2_{K+1} = f_{ellipse}\left(x_{K+1} + \frac{1}{2}, y_{K+1} - 1\right)$$

$$= r_y^2\left(x_{K+1} + \frac{1}{2}\right)^2 + r_x^2\left(y_{K+1} - 1\right)^2 - r_x^2 r_y^2 \quad \text{---} \text{(vii)}$$



Subtracting (VI) from (VII)

$$P2_{k+1} - P2_k = r_y^2\left[\left(x_{k+1}+\frac{1}{2}\right)^2 - \left(x_k+\frac{1}{2}\right)^2\right]$$

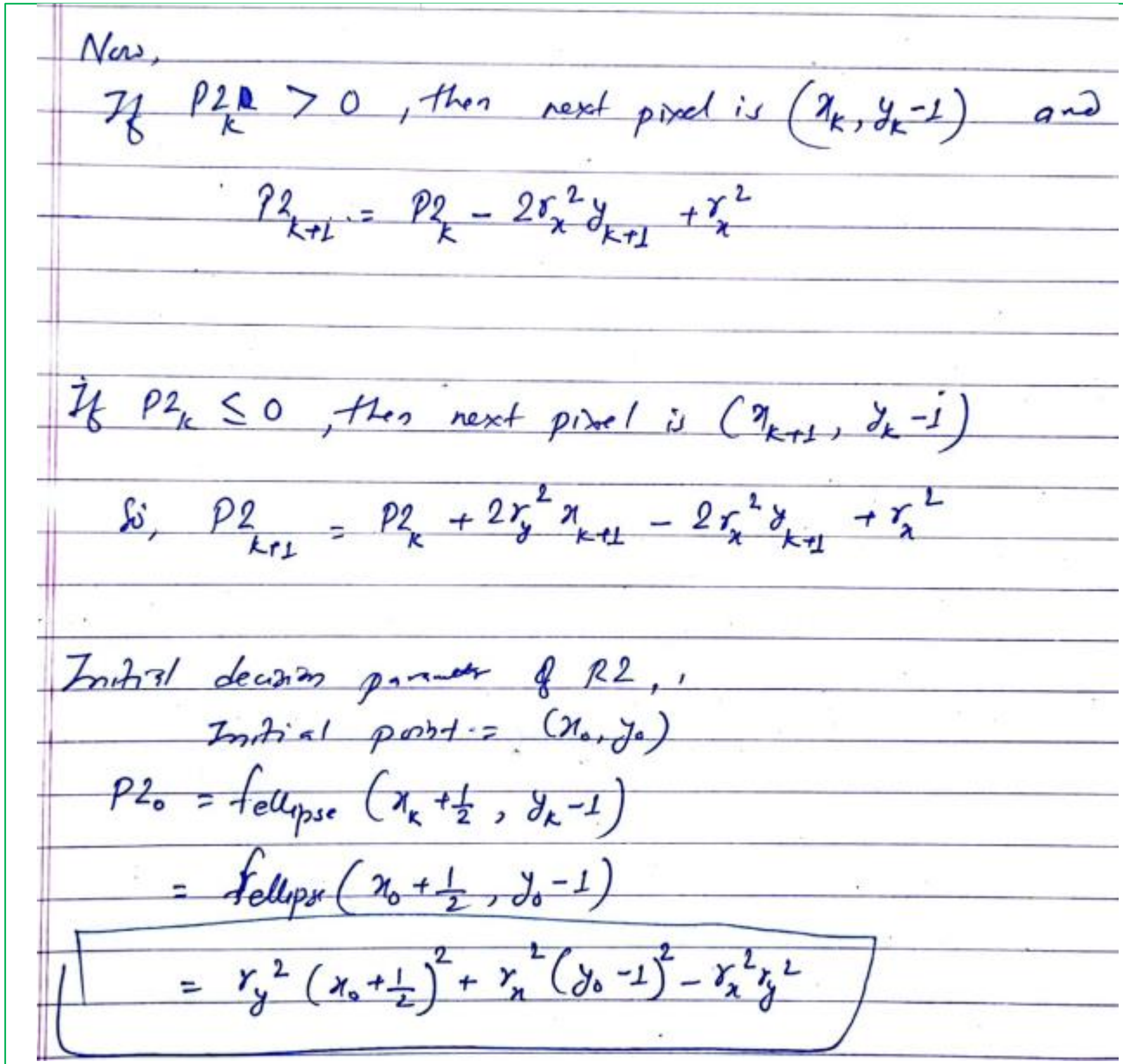
$$+ r_x^2\left[\left(y_{k+1}-1\right)^2 - \left(y_k-1\right)^2\right]$$

$$= r_y^2\left[\left(x_{k+1}+\frac{1}{2}\right)^2 - \left(x_k+\frac{1}{2}\right)^2\right]$$

$$+ r_x^2\left[\left(y_k-2\right)^2 - \left(y_k-1\right)^2\right]$$

[∵ next sampling point $y_{k+1}-1 = y_k-2$]

$$\therefore P2_{k+1} = P2_k + r_y^2\left[\left(x_{k+1}+\frac{1}{2}\right)^2 - \left(x_k+\frac{1}{2}\right)^2\right] + r_x^2\left[-2y_k+3\right]$$



Now,

If $P2_k > 0$, then next pixel is (x_k, y_k-1) and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

If $P2_k \le 0$, then next pixel is (x_{k+1}, y_k-1)

So, $P2_{k+1} = P2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

Initial decision parameter of R2,

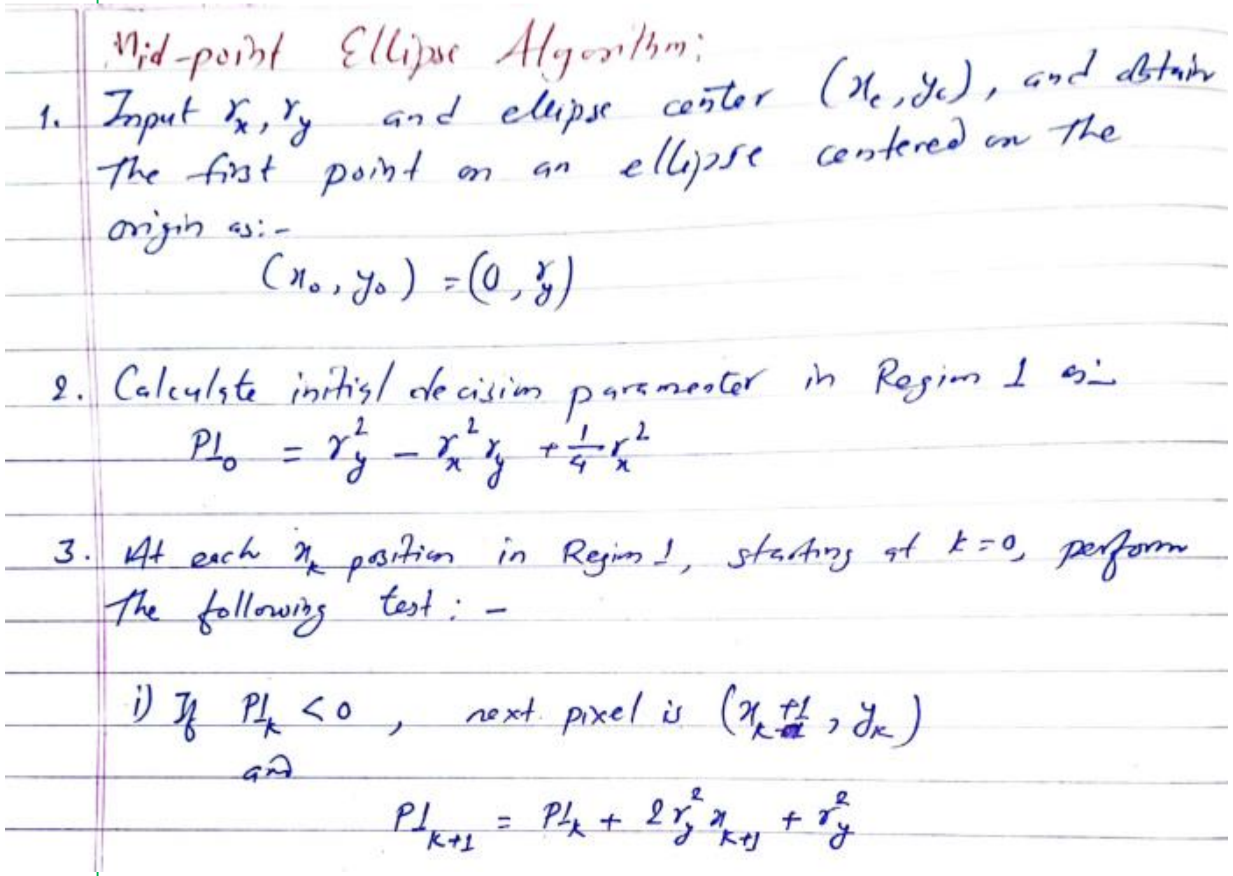
Initial point := (x_0, y_0)

$$P2_0 = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$= f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$\boxed{= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0-1)^2 - r_x^2 r_y^2}$$

• Algorithm:



Mid-point Ellipse Algorithm:

1. Input r_x, r_y and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as:-

$$(x_0, y_0) = (0, r_y)$$

2. Calculate initial decision parameter in Region 1 as:-

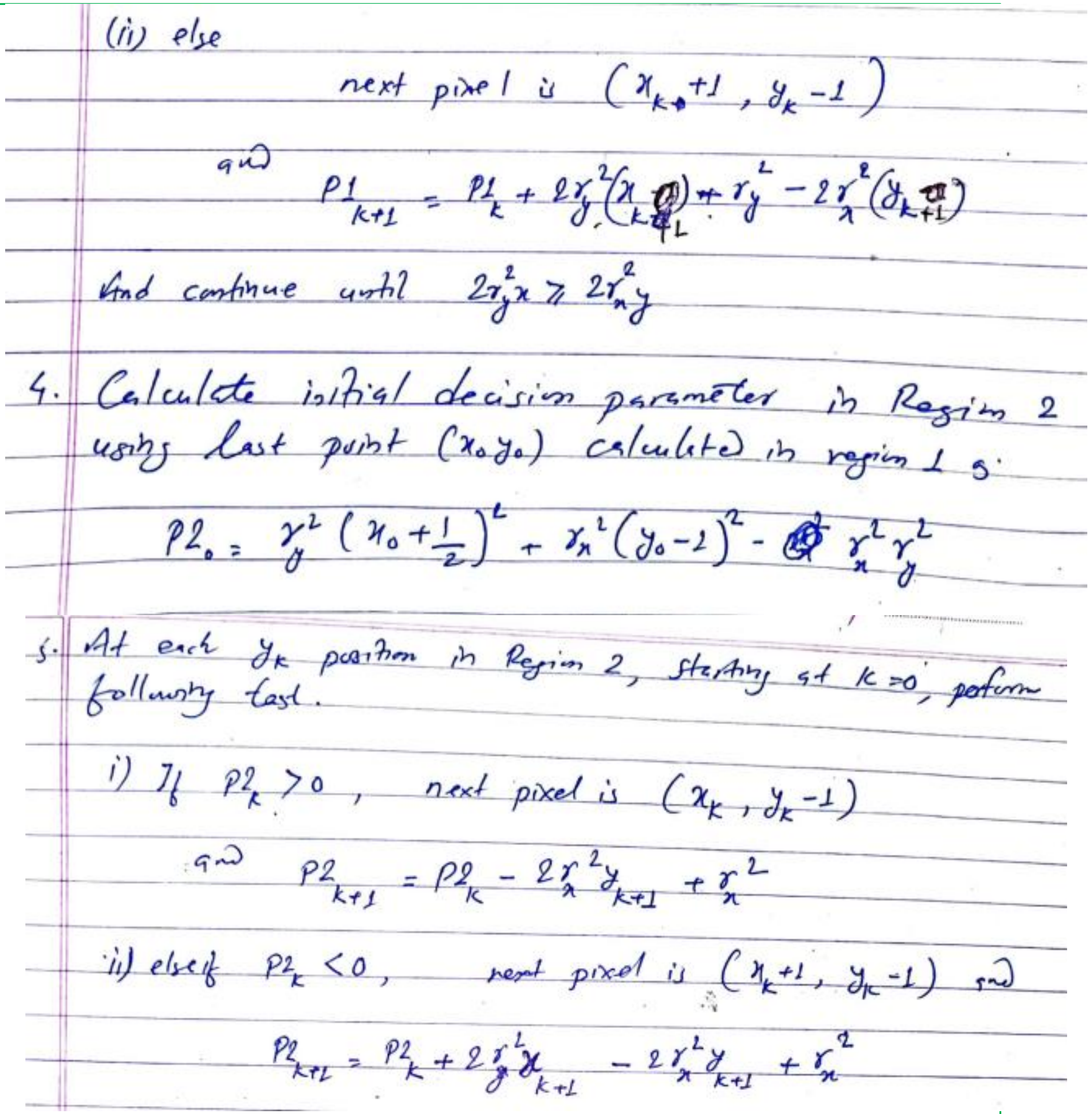
$$P1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in Region 1, starting at $k=0$, perform the following test: -

i) If $P1_k < 0$, next pixel is (x_{k+1}, y_k)

and

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2$$



(ii) else

next pixel is (x_k+1, y_k-1)

and

$$P1_{k+1} = P1_k + 2r_y^2(x_{k+1}) + r_y^2 - 2r_x^2(y_{k+1})$$

And continue until $2r_y^2 x \geq 2r_x^2 y$

4. Calculate initial decision parameter in Region 2 using last point $(x_0 y_0)$ calculated in region 1 as:

$$P2_0 = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in Region 2, starting at $k=0$, perform following test.

i) If $P2_k > 0$, next pixel is $(x_k, y_k - 1)$

and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

ii) elseif $P2_k < 0$, next pixel is (x_k+1, y_k-1) and

$$P2_{k+1} = P2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

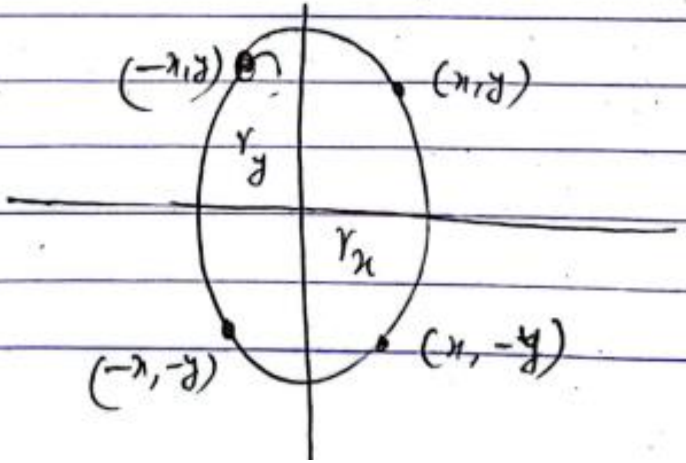
6. Determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x,y) onto the ~~elep~~ elliptical path centered on (x_c, y_c) and plot the coordinate values.

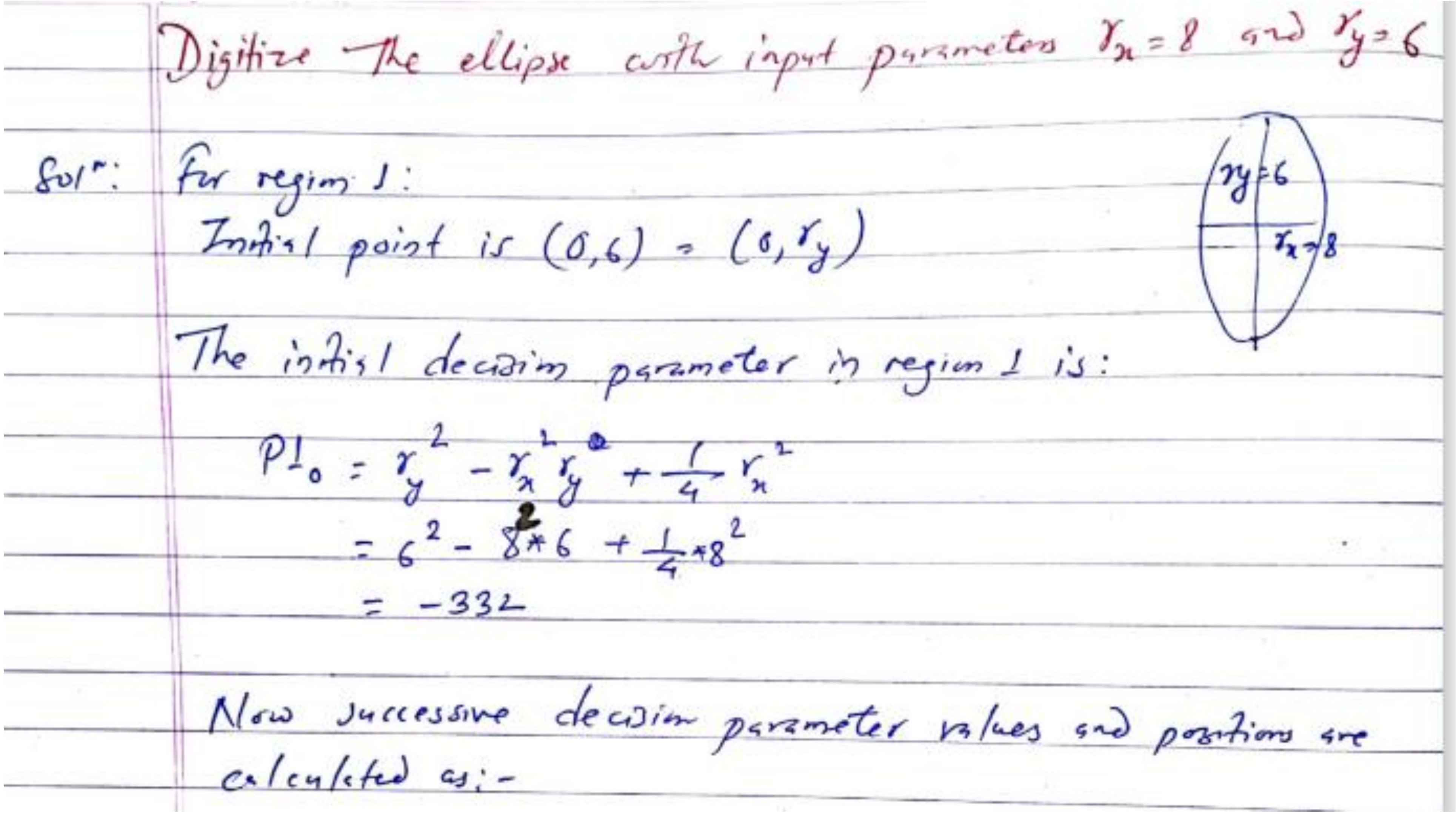
$$x = x + x_c$$

$$y = y + y_c$$

8. Repeat the steps for region 1 until $2r_y^2 x \geq 2r_x^2 y$



- **Numerical:**



Digitize the ellipse with input parameters $r_x = 8$ and $r_y = 6$

Solⁿ: For region 1:

Initial point is $(0,6) = (0, r_y)$

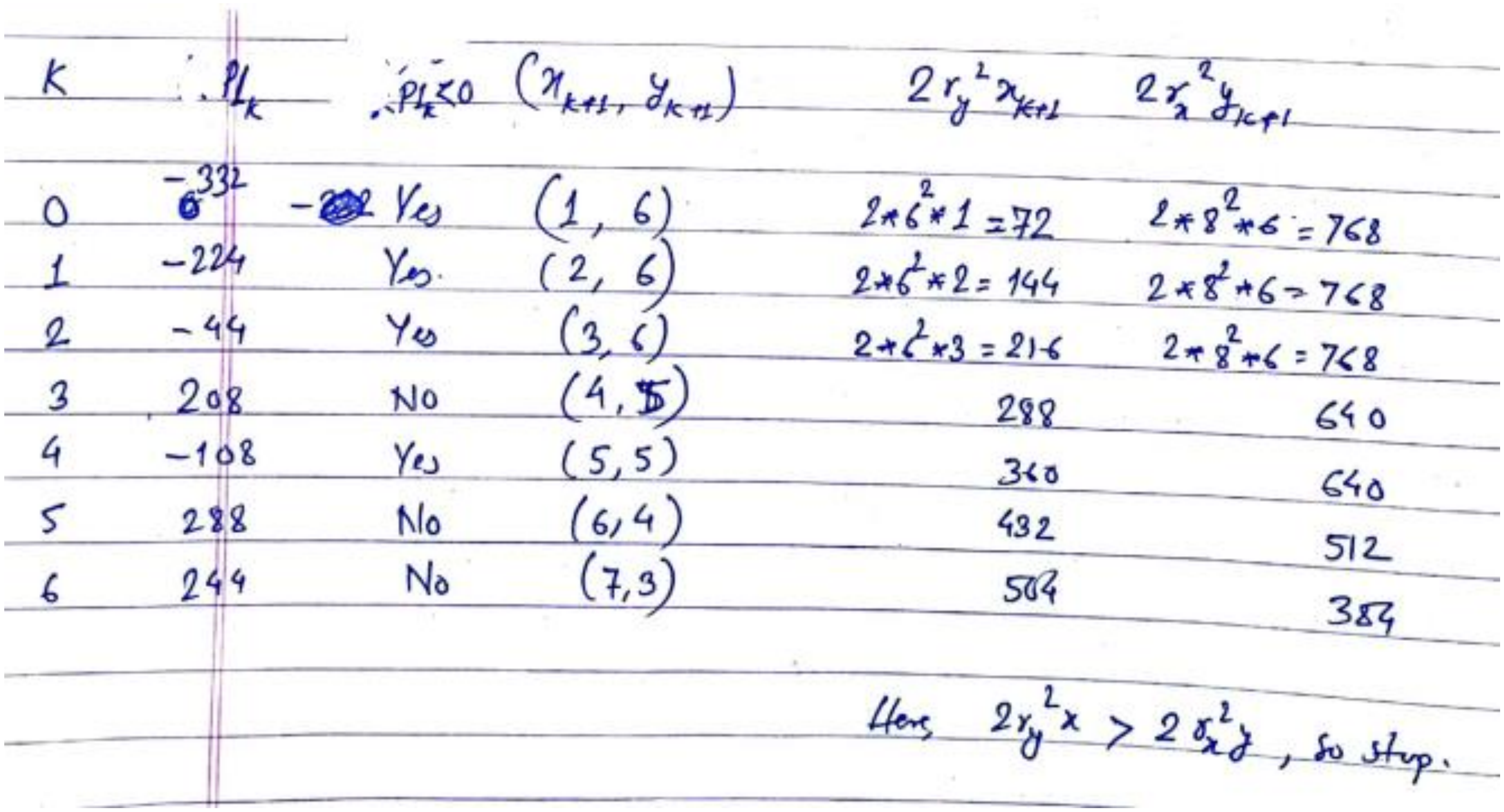
The initial decision parameter in region 1 is:

$$P1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

$$= 6^2 - 8^2 * 6 + \frac{1}{4} * 8^2$$

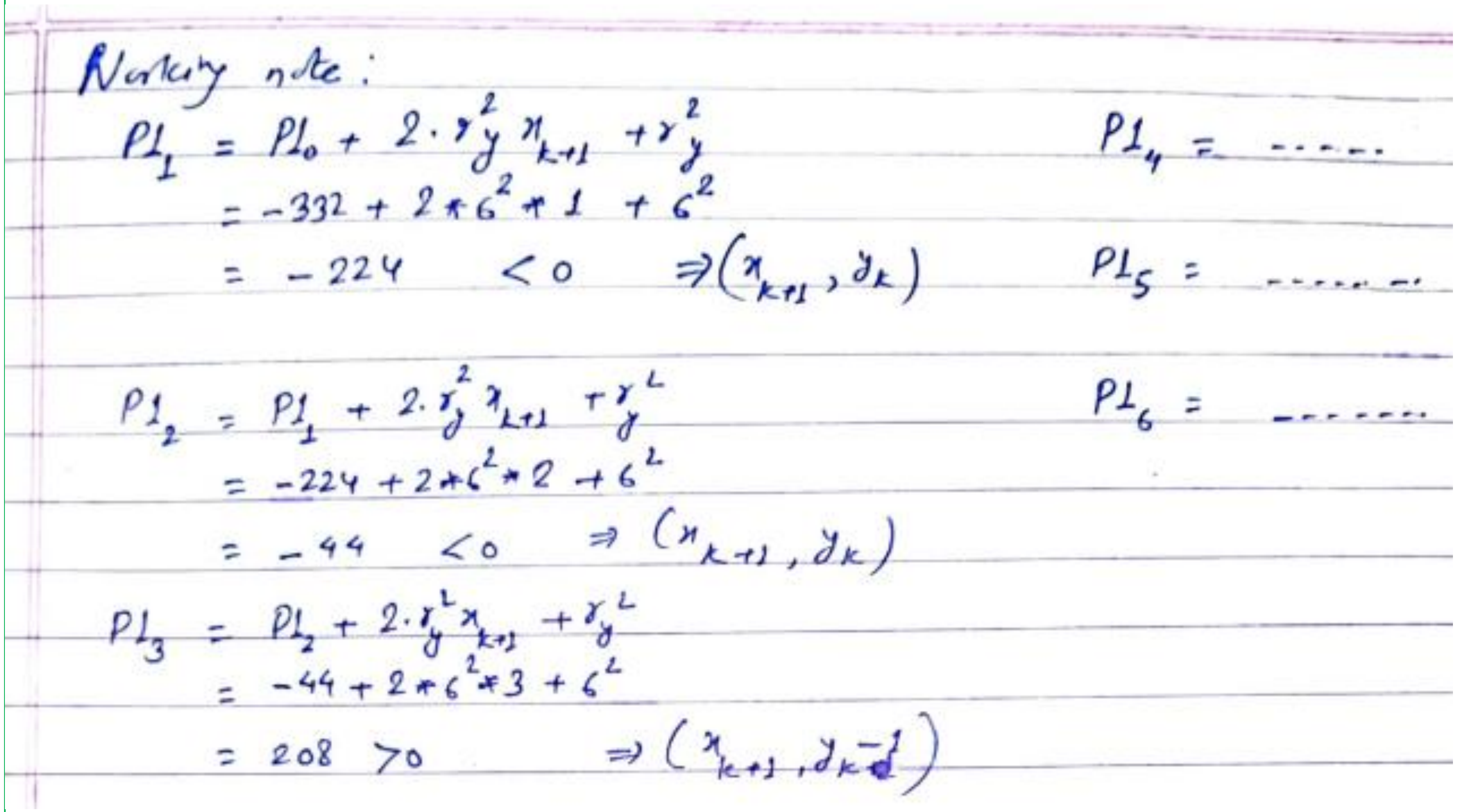
$$= -332$$

Now successive decision parameter values and positions are calculated as:-



K	$P1_k$	$P1_k<0$	(x_{k+1}, y_{k+1})	$2r_y^2x_{k+1}$	$2r_x^2y_{k+1}$
0	-332	Yes	(1, 6)	$2*6^2*1 = 72$	$2*8^2*6 = 768$
1	-224	Yes.	(2, 6)	$2*6^2*2 = 144$	$2*8^2*6 = 768$
2	-44	Yes	(3, 6)	$2*6^2*3 = 216$	$2*8^2*6 = 768$
3	208	No	(4, 5)	288	640
4	-108	Yes	(5, 5)	360	640
5	288	No	(6, 4)	432	512
6	244	No	(7, 3)	504	384

Here $2r_y^2x > 2r_x^2y$, so stop.



Working note:

$$P1_1 = P1_0 + 2 \cdot r_y^2 x_{k+1} + r_y^2$$

$$= -332 + 2 * 6^2 * 1 + 6^2$$

$$= -224 < 0 \Rightarrow (x_{k+1}, y_k)$$

$$P1_2 = P1_1 + 2 \cdot r_y^2 x_{k+1} + r_y^2$$

$$= -224 + 2 * 6^2 * 2 + 6^2$$

$$= -44 < 0 \Rightarrow (x_{k+1}, y_k)$$

$$P1_3 = P1_2 + 2 \cdot r_y^2 x_{k+1} + r_y^2$$

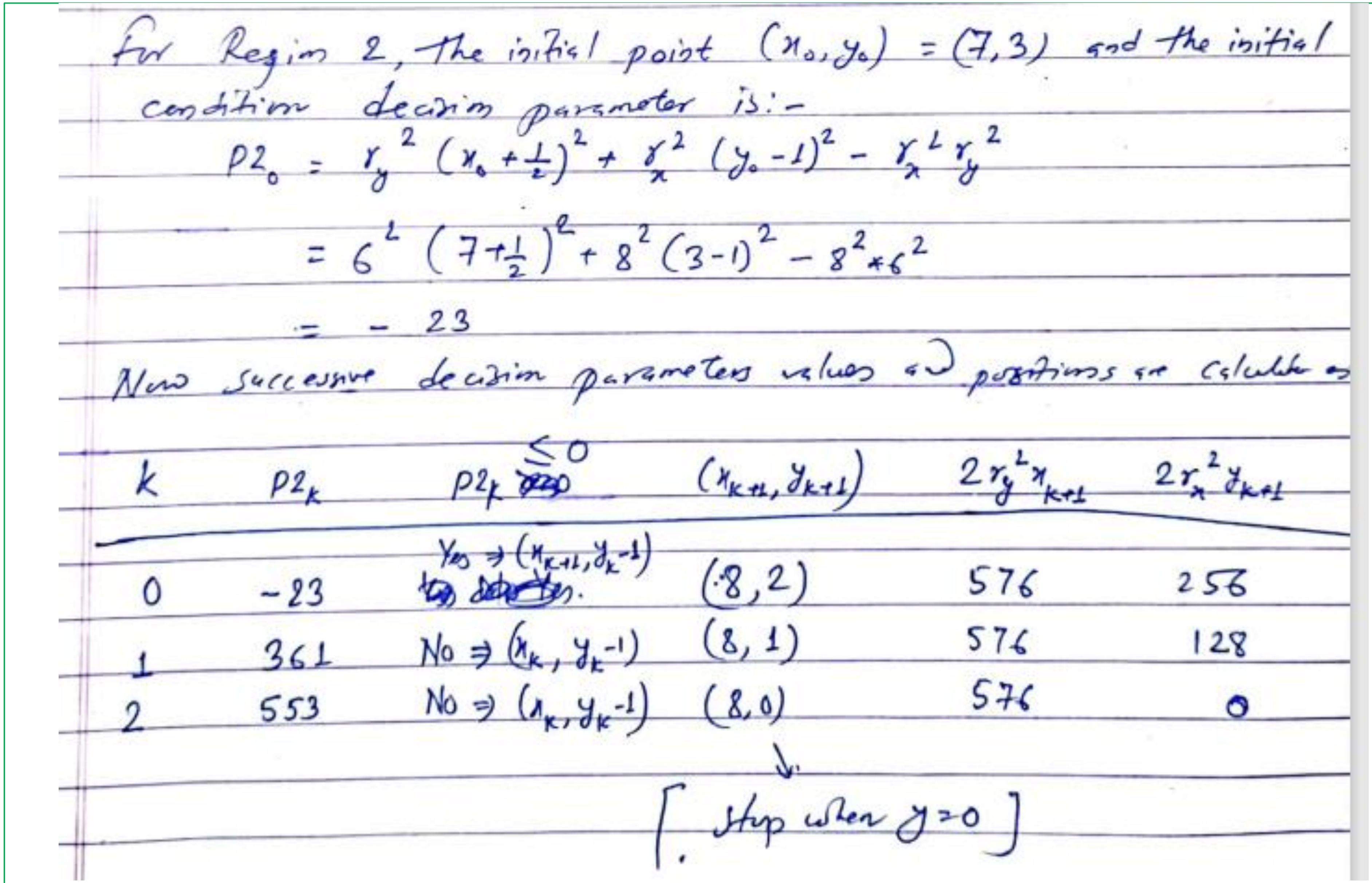
$$= -44 + 2 * 6^2 * 3 + 6^2$$

$$= 208 > 0 \Rightarrow (x_{k+1}, y_k - 1)$$

$P1_4 =$ -----

$P1_5 =$ -------

$P1_6 =$ -------



For Regim 2, the initial point $(x_0, y_0) = (7,3)$ and the initial condition decision parameter is:-

$$P2_0 = r_y^2 (x_0 + \tfrac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

$$= 6^2 (7 + \tfrac{1}{2})^2 + 8^2 (3-1)^2 - 8^2 * 6^2$$

$$= -23$$

Now successive decision parameters values and positions are calculate as

k	$P2_k$	$P2_k \leq 0$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	-23	Yes ⇒ $(x_{k+1}, y_k - 1)$	(8,2)	576	256
1	361	No ⇒ $(x_k, y_k - 1)$	(8, 1)	576	128
2	553	No ⇒ $(x_k, y_k - 1)$	(8,0)	576	0

↓

[Stop when $y = 0$]

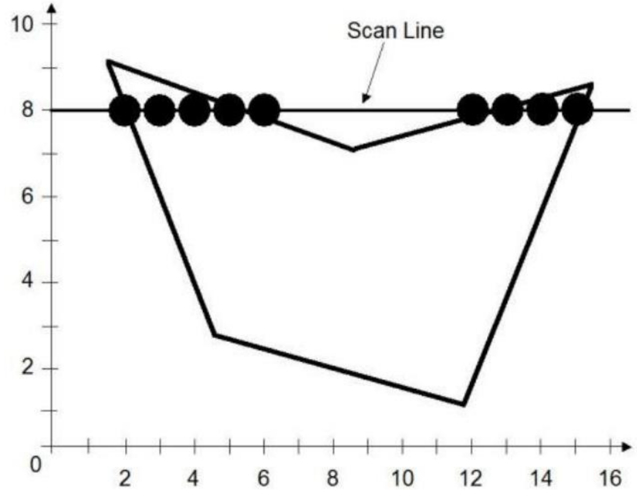
2.5 Area Filling Algorithms

- Area filling is the process of coloring in a fixed area or region.
- Basic procedure is:
  - Find all pixels in the polygon
  - Fill pixels with specified colors.
- Two approaches used:
  - Scan Line
  - Seed Fill
    - Boundary Fill
    - Flood Fill

a) Scan Line Filling

- Scan fill algorithm is an area-filling algorithm that fill colors by scanning horizontal lines.
- These horizontal lines intersect the boundaries of the polygon and fill colors between the intersection points.
- Its main purpose is to fill colors in the interior pixels of the polygon.
- The basic scan-line algorithm is as follows:
  - Find the intersections of the scan line with all edges of the polygon
  - Sort the intersections by increasing x coordinate
  - Fill in all pixels between pairs of intersections that lie interior to the polygon

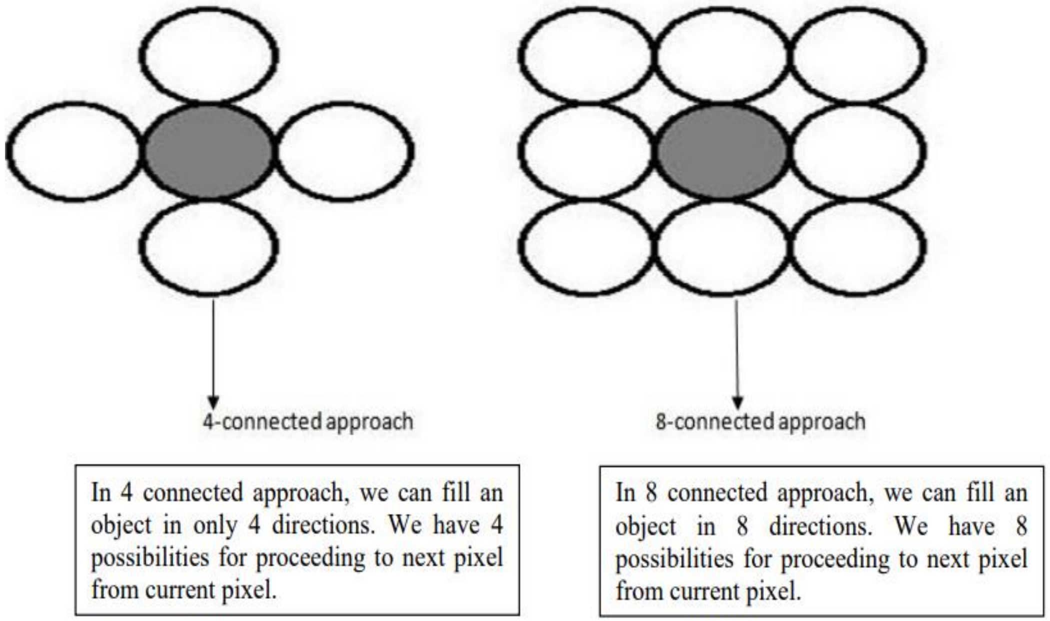
- The scan-line polygon-filling algorithm involves below steps:
  - Locate the intersection points of the scan line with the polygon edges and make even pairs of intersection points.
  - the horizontal scanning of the polygon from its topmost to its bottommost vertex,
  - identifying which edges intersect the scan-line,
  - and finally drawing the interior horizontal lines with the specified fill color process.



Reference video:
https://www.youtube.com/watch?v=XtE_ZKL7jEs
https://www.youtube.com/watch?v=x2TY95ZS-OQ

b) Seed Fill Algorithm

- First of all, a starting pixel called as the seed is considered.
- The filling is done using four connected or eight connected approaches.
- Two techniques:
    - Boundary fill
    - Flood fill

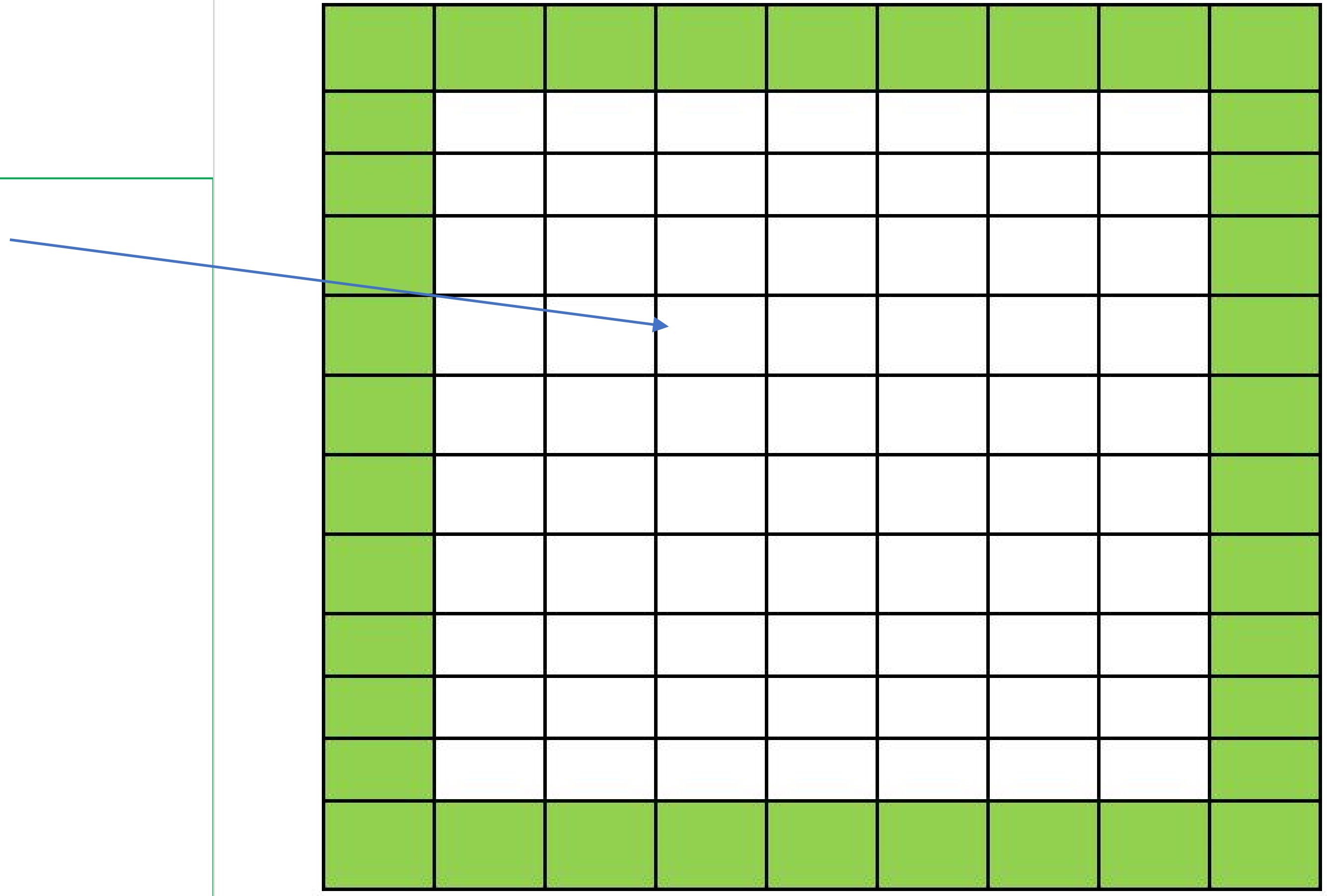


In 4 connected approach, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.

In 8 connected approach, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel.

a) Boundary Fill Algorithm (BFA)

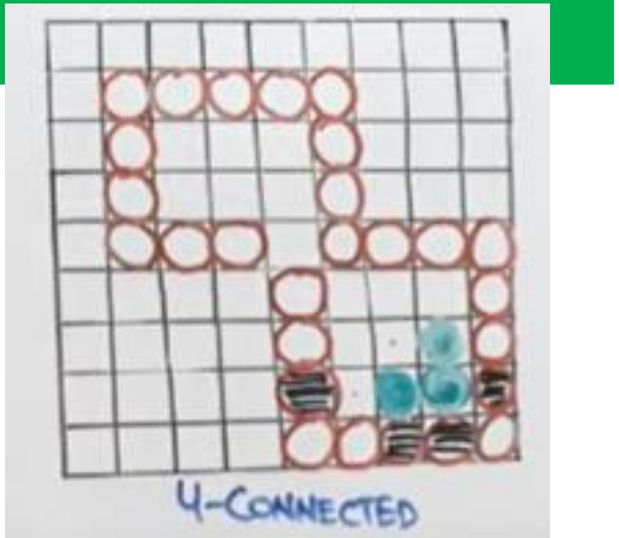
- How to fill this inner area?

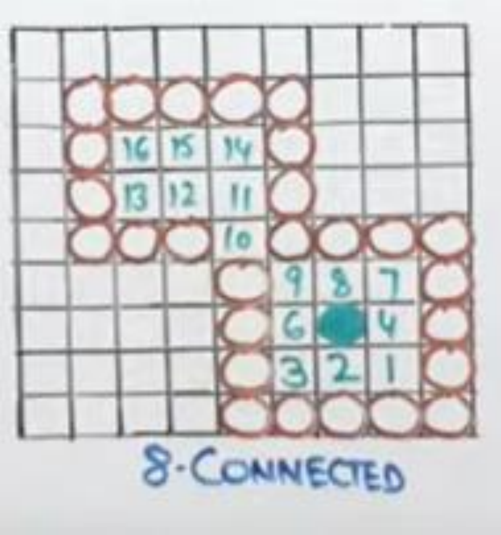


a) Boundary Fill Algorithm (BFA)

- Start at a point inside a region and paint the interior outward toward the boundary.
- If the boundary is specified in a single color, the fill algorithm processed outward pixel by pixel until the boundary color is encountered.
- A boundary-fill procedure accepts as input :
  - the coordinate of the interior point (x, y),
  - a fill color, and
  - a boundary color.
- **Limitation:** If the polygon has boundary with different colors, then the algorithm fails.

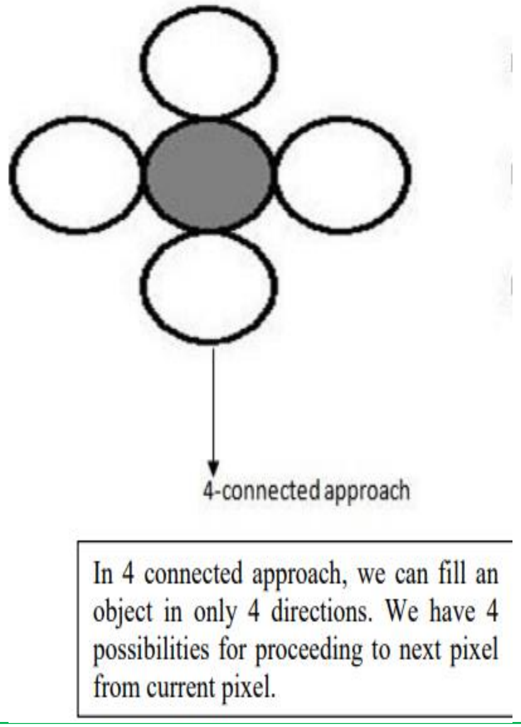
Reference:
https://www.youtube.com/watch?v=tB3PYPfTIYI





4-connected BFA

- In this approach, left, right, above, below pixels are tested.
- **Advantages:** The boundary fill algorithm is used to create attractive paintings.
- **Disadvantages:** In the 4-connected approach, it does not color corners.



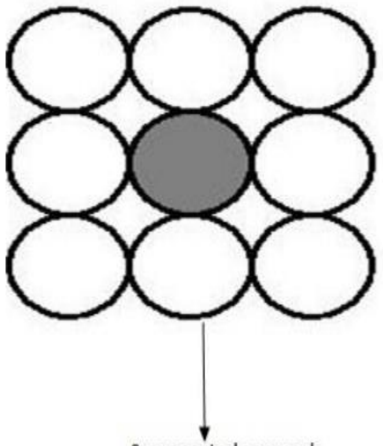
In 4 connected approach, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.

Function for 4 connected approach:

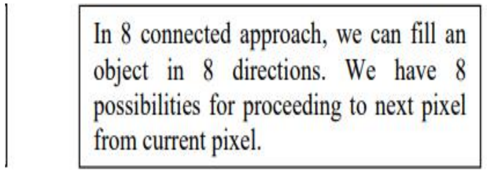
```
void boundary_fill(int x, int y, int fcolor, int bcolor)
{
   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
    {             delay(10);
                 putpixel(x, y, fcolor);
                 boundary_fill(x + 1, y, fcolor, bcolor);
                 boundary_fill(x - 1, y, fcolor, bcolor);
                 boundary_fill(x, y + 1, fcolor, bcolor);
                 boundary_fill(x, y - 1, fcolor, bcolor);
   }
}
```

8-connected BFA

- In this approach, we can fill an object in 8 directions as shown in the figure aside.



8-connected approach



In 8 connected approach, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel.

Function for 8 connected approach:

```
void boundary_fill(int x, int y, int fcolor, int bcolor)
{

   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
   {                    delay(10);
         putpixel(x, y, fcolor);
         boundary_fill(x + 1, y, fcolor, bcolor);
         boundary_fill(x , y+1, fcolor, bcolor);
         boundary_fill(x+1, y + 1, fcolor, bcolor);
         boundary_fill(x-1, y - 1, fcolor, bcolor);
         boundary_fill(x-1, y, fcolor, bcolor);
         boundary_fill(x , y-1, fcolor, bcolor);
         boundary_fill(x-1, y + 1, fcolor, bcolor);
         boundary_fill(x+1, y - 1, fcolor, bcolor);

    }
}
```

b) Flood Fill Algorithm

How to fill color in this section?

Reference: https://www.youtube.com/watch?v=1dQhtqqaWdY

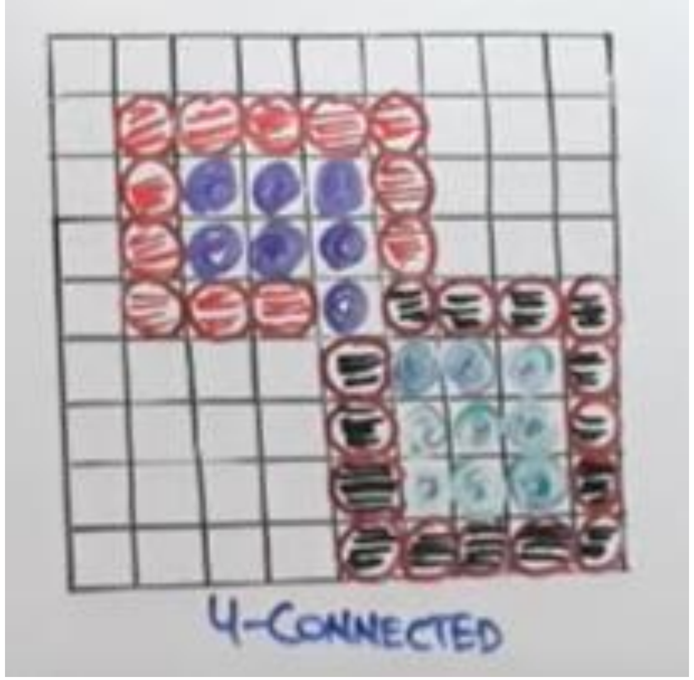
b) Flood Fill Algorithm

- Sometimes we want to fil-in (recolor) an area that is not defined within a single color boundary.
- Instead of searching for a boundary color value, we paint such areas by replacing a specified interior color.
- This approach is called a flood-fill algorithm. The most approached implementation of the algorithm is a stack-based recursive function.
- Basic process is:
  1. Start from a specified interior pixel (x, y) and replace its old color with new fill color.
  2. Check for the neighbor pixel for old color, and replace it with new fill color.
  3. Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.

Reference: https://www.youtube.com/watch?v=1dQhtqqaWdY

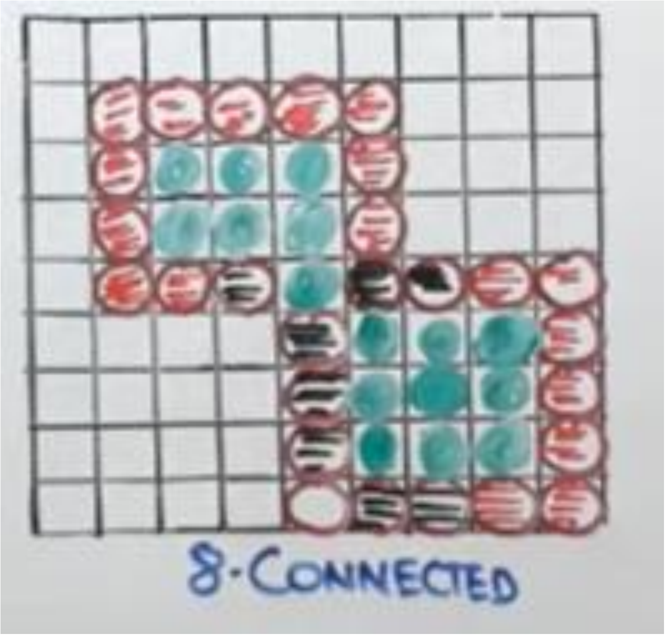
4-connected Flood Fill approach:

```
function floodfill (x, y, fill_ color, old_color)
     If (getpixel (x, y)=old_color)
     {
          setpixel (x, y, fill_color);
           floodfill  (x+1, y, fill_color, old_color);
           floodfill  (x-1, y, fill_color, old_color);
           floodfill  (x, y+1, fill_color, old_color);
           floodfill  (x, y-1, fill_color, old_color);
    }
}
```



8-connected Flood Fill approach:

```
function floodfill (x, y, fill_ color, old_color)
        If (getpixel (x, y)=old_color)
        {
                putpixel(x, y, fill_col);
                floodfill (x+1, y, fill_col, old_col);
                floodfill (x-1, y, fill_col, old_col);
                floodfill (x, y+1, fill_col, old_col);
                floodfill (x, y-1, fill_col, old_col);
                floodfill (x + 1, y - 1, fill_col, old_col);
                floodfill (x + 1, y + 1, fill_col, old_col);
                floodfill (x - 1, y - 1, fill_col, old_col);
                floodfill (x - 1, y + 1, fill_col, old_col);
    }
}
```



Exam Questions

1. Derive the Bresenham's line algorithm for |m|>1. [2011 fall/2012 spring/ 2014 fall]
2. Write Bresenham's line drawing algorithm for slope |m|<1. how does it differ from the algorithm for slope |m|>1 ? [2011 spring]
3. Describe the symmetric property of a circle. Also derive the mid-point circle algorithm. [2012 fall]
4. Digitize a standard form circle using midpoint algorithm having radius of 10 units. [2012 spring]
5. Derive an equation for calculating points of a circle using midpoint algorithm. [2013 fall]
6. Write a code for drawing a full circle points. [or qstn. 2013 fall]
7. While scan converting an ellipse, how do we know that we have reached the second region of the first quadrant of the ellipse? Explain with expressions. [2013 spring]
8. Digitize a line with end points A(2,10) and B(5,18) using Bresenham's line drawing algorithm. [2013 spring]
9. Rasterize the circle of 10 unit radius. [2014 fall]
10. What is DDA? Derive the Bresenham's line drawing algorithm for the slope greater than one. [2015 fall]

11. Find the raster position along the region 1 of the ellipse path in first quadrant. The semi major and semi minor axes are 8 and 7 respectively. [2015 fall]
12. Digitize a circle centered at (100,200) and having radius 8. [2015 spring]
13. Explain the logic used for drawing lines with positive and negative slopes using Bresenham’s line drawing algorithm. [2015 spring]
14. Derive Bresenham’s line drawing algorithm for slope less than one. How can this line (with end points A(x1,y1), B(x2,y2) and slope less than 1) be drawn if the starting point is taken as B(x2, y2)? [2016 spring]
15. Define boundary fill technique. Differentiate between Bresenham’s line and DDA line drawing algorithm. [2016 spring]
16. Derive Bresenham's line drawing algorithm for |m|<1. [2014 spring]
17. Digitize one octant of a circle by using midpoint circle generation algorithm center at (10, 20) and radius is 10. [2016 spring]
18. Using the bresenham’s line drawing algorithm predict the pixels on the line from (2,2) to (12,10). [2017 fall]
19. Digitize one octant of a circle by using midpoint circle generation algorithm center at origin and radius is 12. [2017 spring]
20. Derive an equation for drawing a line using Bresenham’s algorithm for slope less than one. [2017 spring]

21. How decision parameter is calculated in midpoint circle method. Show all necessary derivation. [2018 fall]
22. Explain the boundary fill and flood fill algorithm in detail. [2018 fall]
23. Explain scan line method. [2018 spring]
24. Rasterize the points of given line end points A(-2,-4) and B(-6,-9) using Bresenham's line drawing algorithm. [2018 spring]
25. Explain the working of DDA line drawing algorithm with suitable examples. Write its advantages and disadvantages. [2019 fall]
26. Explain symmetrical property of circle. Write midpoint circle algorithm and apply that algorithm to find the pixel values of the circle whose radius r=10 and center of the circle =(0,0). [2019 fall]
27. Define decision parameter in Bresenham's line drawing. Digitize a circle $(x-2)^2+(y-3)^2=25$ using a midpoint circle drawing algorithm. [2019 fall]
28. Explain the Bresenham's line drawing algorithm with suitable example. [2019 spring]
29. Derive mid-point circle algorithm. [2019 spring]
30. Derive an equation for calculating points of an ellipse. [2020 fall]

31. Rasterize the points of given line end points A(-2,-4) and B(-6,-9) using Bresenham's line drawing algorithm. [2020 fall]

32. Throughout the first quadrant dividing it into two parts, the regions can be formed by considering the slope of the curve. Assume the slope of curve is less than one than we are in region one and when the slope becomes greater than -1 then in region 2. Considering major and minor axis throughout the first quadrant. What would be appropriate method used to draw the elongated part of circle.? [2020 spring]

33. Digitize the first octant of a circle having radius r=8 and centered at (3,4). [2021 fall]

34. Write Bresenham's line drawing algorithm along with necessary derivation for positive slope less than 1. trace the algorithm [2021 spring]

35. Consider a line from (2,1) to (8,3) suing DDA algorithm to rasterize a line. [2022 fall]

36. Explain the boundary fill algorithm in detail. How this approach differs from flood fill? [2022 fall]

End of chapter