Creating the rainbow from a drop of water

Karl Zylinski
Lab partner: Maximilian Svensson

2018-04-24

Abstract

When light hits a drop of water the rays are refracted and reflected. An observer standing in the correct position might then see a rainbow. What optical phenomena lie behind this? At what angle is it possible to see the rainbow? In this report the mathematical tools needed for answering this are derived from optical first principles. These tools are then put to use in simulations where hundreds of thousands of rays are sent through the drop. The result of these simulations tells us that the primary rainbow is best viewed around 42° and the secondary at around 52°. This is in agreement with other sources and verifies that the derived mathematical tools seem to work properly.

Contents

1 Introduction

Rainbows occur when the light from a source such as the sun interacts with water drops in the air, i.e. rain or mist. But this is just the macroscopic explanation. What really happens when light hits the drops? What optical phenomena describe the creation of rainbows?

There are many kinds of rainbows, two examples are primary and secondary rainbows. These two types usually make up what is known as a double rainbow, where one can see a strong primary rainbow and a fainter secondary rainbow above it. At what angles does an observer see the primary and secondary rainbow?

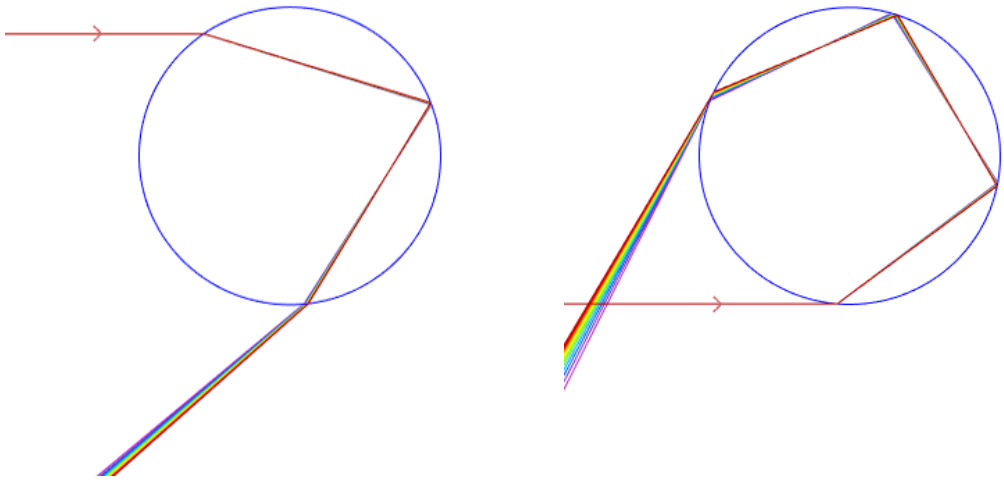

Figure 1: Rays of light traversing a raindrop, forming a primary rainbow (left) and secondary rainbow (right).

The answer to these questions lie in closely examining what happens as light traverses a drop of water. Two images of such drops are pictured in fig. 1. In the left image the light hits the upper half of the drop, refracting into it, bouncing once inside before refracting out again. It is because of these refractions and reflections and the fact that light of different colours possess different refractive indices that the rainbow can appear. Inspecting the image, one can see how the initially bundled light has spread out as the rays leave the drop.

The image to the right is similar, but the light hits the bottom half of the drop, and bounces twice inside. This is what makes the secondary rainbow possible. It should appear at other angles than the primary. Also, it should be fainter due to energy losses from the additional reflections.

This report begins with a short section on optical theory, which contains a couple of optical relations. These are then used to mathematically and optically derive what happens as the light traverses through a drop of water. The derived tools are then put to work in MATLAB, where hundreds of thousands of paths through the drop are traced out. The endpoints of these paths, which describe how the light leaving the drops bunch up, are analysed. By comparing where the light of different colours bunch up, one can find the rainbow and answer the following questions:

- What optical phenomena and mathematical models describe the creation of rainbows?
- At what angles does one see the primary and secondary rainbow?
- What fraction of the incident light survives for different kinds of polarised light?

2 Theory

This section states a couple of physical relations. They are used in the next section to analyse how the light is affected as it traverses the inside of the drop.

2.1 Snell's law

If two mediums or materials have different refractive indices and light travels between them, then the behaviour of the light is governed by Snell's law:

$$n_1 sin(\theta) = n_2 sin(\phi) \tag{1}$$

where n_1, n_2 are the refractive indices of the different materials and θ, ϕ are the angles to the normal of the intersection between the materials.

2.2 Deflection angle

The deflection angle is defined as the angle between the direction of the incident (incoming) and outgoing rays of light. In this project the incident light was assumed to be parallel with the x-axis. The deflection angle is denoted D in fig. 2. This is later used to calculate where the light leaving the drop ends up.

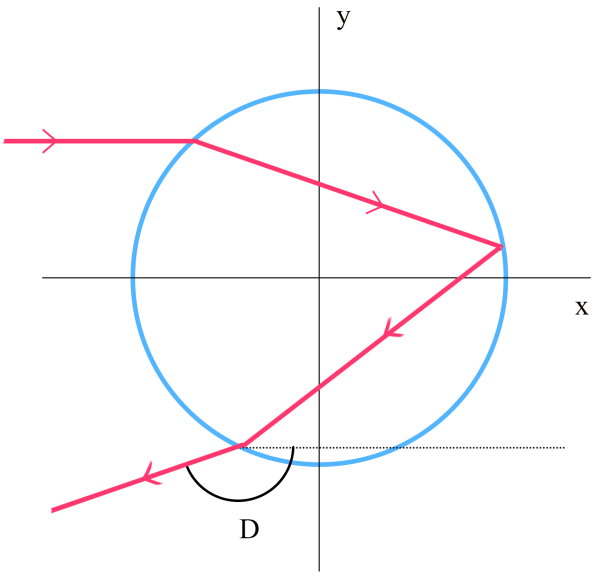

Figure 2: D denotes the deflection angle. The dotted line has the same direction as the incident light, which is assumed to be parallel to the x-axis.

2.3 Fresnel's formulas

The reflection and refraction has on the intensity of light can be seen in Fresnel's formulas. The intensity after a reflection is $I = I_0 R$ where R is the coefficient of reflection and I_0 is the original intensity. After a refraction the intensity is $I = I_0 T$, where T is the coefficient of refraction.

The incident light can come in two flavours of polarisation: p-polarised which oscillates parallel to the plane of incidence, and s-polarised which oscillates perpendicular to the plane of incidence.

The different coefficients are:

$$R_p(I,O) = \Big(\frac{tan(I-O)}{tan(I+O)}\Big)^2 \tag{2}$$

$$T_p(I,O) = \frac{sin(2I)\,sin(2O)}{sin^2(I+O)\,cos^2(I-O)} \tag{3}$$

$$R_s(I,O) = \Big(\frac{sin(I-O)}{sin(I+O)}\Big)^2 \tag{4}$$

$$T_s(I,O) = \frac{sin(2I)\,sin(2O)}{sin^2(I+O)} \tag{5}$$

where I is the incident light's angle to the normal and O is the outbound light's angle to the normal. These coefficients are later used to figure out the decrease in intensity for s- and p-polarised light as the light traverses the water drop.

3 Mathematical procedures: Tracing a ray of light

In this section I describe how to step-by-step follow a ray of light that refracts into a drop of water, bounces around, refracts out and then ends up on some kind of screen, such as your eye. The end goal is taking everything that is derived in this section and using it to run lots of simulations that figure out where light of different colours bunch up on the screen. Where the light bunches up, rainbows might appear. These subsequent simulations are described in section 4 (Programming procedures).

3.1 Refraction at edge of the drop

Consider fig. 3. It shows how a ray of light (red line) enters a drop of water (blue circle). As it hits the interface between the water and air, it is refracted, bending towards the normal (N in the figure). The incident ray of light is assumed to be parallel to the x-axis. θ is the angle between the incident ray and the normal of the drop. θ can be found using trigonometry by noticing that it appears again at the origin, between the x-axis and the normal line. If the radius of the drop is r and the y-position of the incident ray is h, then a triangle with an angle θ, hypotenuse r and opposite height h can be constructed, thus

$$sin(\theta) = \frac{h}{r} \tag{6}$$

$$\theta = arcsin(\frac{h}{r}) \tag{7}$$

Now the angle ϕ, the ray's refracted angle, needs to be found. Let the refractive index outside of the drop be n_1 and the index inside the drop be n_2, then by Snell's law (eq. 1)

$$n_1 sin(\theta) = n_2 sin(\phi)$$

$$sin(\phi) = \frac{n_1 sin(\theta)}{n_2}$$

replacing $sin(\theta)$ with eq. 6 yields

$$sin(\phi) = \frac{n_1 h}{n_2 r} \tag{8}$$

$$\phi = arcsin\Big(\frac{n_1 h}{n_2 r}\Big) \tag{9}$$

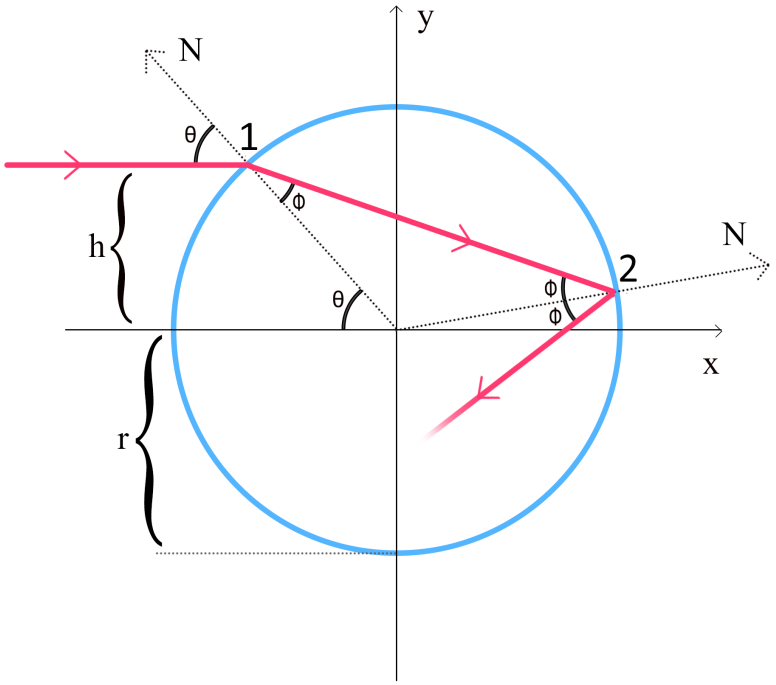

Figure 3: Shows how a ray enters the drop, refracts and then does a single reflection. The path continues after the reflection, but it is not depicted.

3.2 Travel distances inside the drop

Looking again at fig. 3, what is is the distance between the points marked with 1 and 2? This can be found out by noting that the angle of reflection at point 2 is the same angle ϕ as previously used. If we denote the line between 1 and 2 as b, a triangle can be formed using the two normal lines (marked N) and b. The distance b can then be found using the law of cosines:

$$b^2 = a^2 + c^2 - 2ac\,cos(B)$$

here $a = c = r$, $B = \pi - 2\phi$ and b as previously defined:

$$\begin{aligned} b^2 &= 2r^2 - 2r^2\,cos(\pi - 2\phi) \\ b^2 &= 4r^2(1 - sin^2(\phi)) \\ b^2 &= 4r^2(1 - \left(\frac{n_1 h}{n_2 r}\right)^2) \qquad \text{(replace } sin(\phi) \text{ with eq. 8)} \\ b &= 2r\sqrt{1 - \left(\frac{n_1 h}{n_2 r}\right)^2} \end{aligned} \tag{10}$$

This distance b can be used in combination with the calculated refraction and reflection angles (θ and ϕ) to traverse the inside the of water drop for an arbitrary number of bounces.

3.3 Refracting out of the drop and onto a screen

After one or two bounces inside the drop, depending on if you're looking for the primary or secondary rainbow, the light refracts out again. In order study the rainbow phenomenon one

needs to see where the rays leaving the drop goes. One way to do this is to set up a vertical line outside the drop and see where rays land on it. This is depicted in fig. 4, where a vertical line has been set up at x = -L. I assume that we know the position at point $\bar{P} = [P_x, P_y]$ in the picture (a position vector). $\bar{P}$ can be determined using what I derived in the previous two sections; following the ray of light around inside the drop and keeping track of the position.

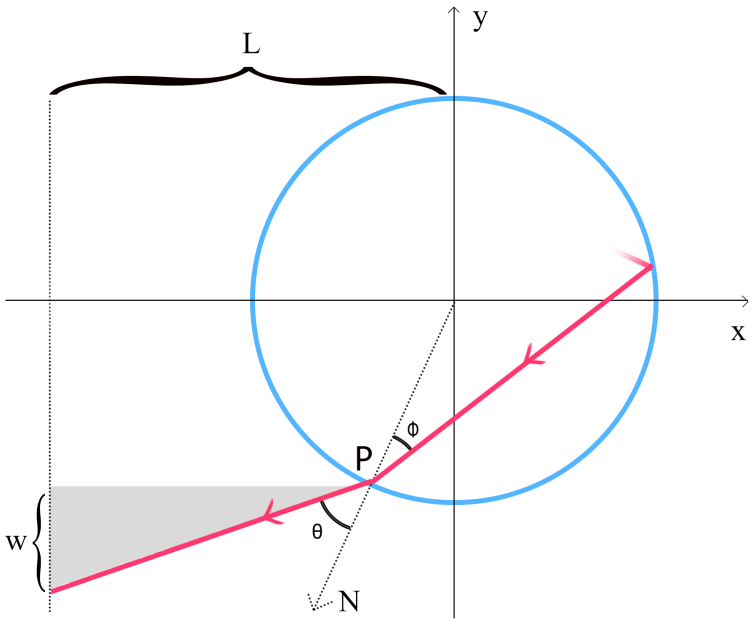

Figure 4: Shows how the ray, after a single reflection, refracts out of the drop again. It then hits the "screen" at $x = -L$.

Since $\bar{P}$ is known, what remains is to find the vertical distance w, which is the additional y-distance the ray travels after leaving the drop. This can again be done using trigonometry. Looking back at fig. 2, the angle which the ray leaving the drop makes with the x-axis is the deflection angle D. One way of figuring out D is by adding θ to the angle of the depicted normal vector N (fig. 4). The normal vector happens to currently be equivalent to $\bar{P}$. D is thus

$$D = \theta + atan2(P_y, P_x) \tag{11}$$

where atan2 is the four-quadrant arctan function[1]. The triangle marked in grey in fig. 4 can thus be constructed using the side w, the angle $\pi - D$ and the side $P_x + L$. This gives:

$$\begin{aligned} tan(\pi - D) &= \frac{w}{P_x + L} \\ w &= (P_x + L)\, tan(\pi - D) \end{aligned} \tag{12}$$

We can then define a position $\bar{R}$ on the line $x = -L$ as

$$\bar{R} = [-L, P_y - w] \tag{13}$$

R_y is later used to study where rays of different colours end up.

[1]This is an arctan function with a range of $-\pi$ to π instead of $-\pi/2$ to $\pi/2$, this is done by taking the y and x-component as separate arguments instead of a quotient. It is available in MATLAB.

4 Programming procedures

This section describes how the formulas of section 3 were used to create MATLAB-simulations and visualisations. We made a function called *calculate_path* which follows a ray of light around in a drop of water. This function can be seen in appendix D, here follows only a conceptual description. In the function, the ray starts with an initial position and direction of travel with which it advances to the edge of the drop. There it refracts, using eq. 9 to find ray's new angle. This gives a new direction. Using this new direction and the length b described by eq. 10, the ray can be advanced to where it again will hit the drop, but this time from the inside. At this position it reflects on the inside of the wall, which again gives a new direction. Using this direction another step can be taken using distance b. For exploring the primary rainbow only one reflection on the inside is used. For the secondary rainbow, an additional reflection is done.

When one or two reflections have been done and the light once again hits the wall from the inside, it is refracted out from the drop. The angle the ray will have after this is the deflection angle D, described by equation 11. One can then find where the light leaving the drop hits a screen using eq. 12 and 13. In these equation there is a distance L, which is the x-distance from the origin where the screen is set up. For our purposes we let $L = -2$, meaning that we let the light hit the line $x = -2$. $\bar{R}$ (the end position of the light on the screen, eq. 13) is thus:

$$\bar{R} = \big[-2,\ P_y - (P_x - 2)\, tan(\pi - \theta + atan2(P_y, P_x)) \big]$$

P_x and P_y are the components of the current position of the ray, in the code this is called *cur_pos*. We use this variable to keep track of where the ray is after each step of the calculation.

One of the first visual results was an animation (see appendix B for code), this was done in order to check if our code was reasonable. By keeping a list of all the points the ray visited as it traversed the drop, one could run a loop which varied the initial y position h in fig. 3, plotting the result for each value of h. The image used in the introduction (fig. 1) is actually a still frame of this animation for one and two bounces.

When the animation worked correctly, the same path tracing function that was used for the animation could be used for more precise simulations. The animated plotting was disabled and h was made to vary between 0 and 1 (1 being the radius of the drop) with 100000 intermediate steps. The path tracing function was run for all these different values of h. All of this was in turn run for 13 different refractive indices, each mapped to a colour. For each of these 13 refractive indices we got a list of positions $\bar{R}$ at which the light hit the screen. Histograms where then made of the R_y-components, one for each of the 13 refractive indices. After colouring each histogram with its respective colour, a rainbow pattern appeared. This can be seen under results in fig. 5. This was repeated for the secondary rainbow case, here the only difference is that two reflections were allowed inside the drop and h varied between 0 and -1. The secondary rainbow result can be seen in fig. 6. The code for making these plots is found in appendix A. This file uses a function called calcuate_rainbow for calculating the paths for all the refractive indices. This function is found in appendix C.

We also explored how the intensity of the light that left the drop changed depending on if the light was p- or s-polarised. In section 2.3 we described the Fresnel equation and the refraction/reflection coefficients T and R. It was assumed that the initial light intensity I_0 was 1 and that each refraction of reflection decreased the value. In other words (for the primary rainbow) we just calculated $1 \cdot T(\theta, \phi) \cdot R(\theta, \phi) \cdot T(\phi, \theta)$, R appearing once because the primary rainbow case does a single reflection. The plots for the cases with p- and s-polarised light can be seen in figures 7 and 8 respectively. This was only done for the primary rainbow.

Since the path tracing function calculated the deflection angle for each value of h, we used this to plot the deflection angle against the angle of incidence (θ in fig. 3). This was done for both the primary and secondary rainbow and the result is seen in fig. 9 and 10. From these graphs one can determine from which angle a primary or secondary rainbow is most likely to be seen, which is done in the discussion section.

5 Result

The following two figures are histograms of where on the line $x = -2$ the rays leaving the water drop hit. They are for the primary and secondary rainbow respectively. Each figure actually contains 13 histograms, corresponding to 13 different colours of light.

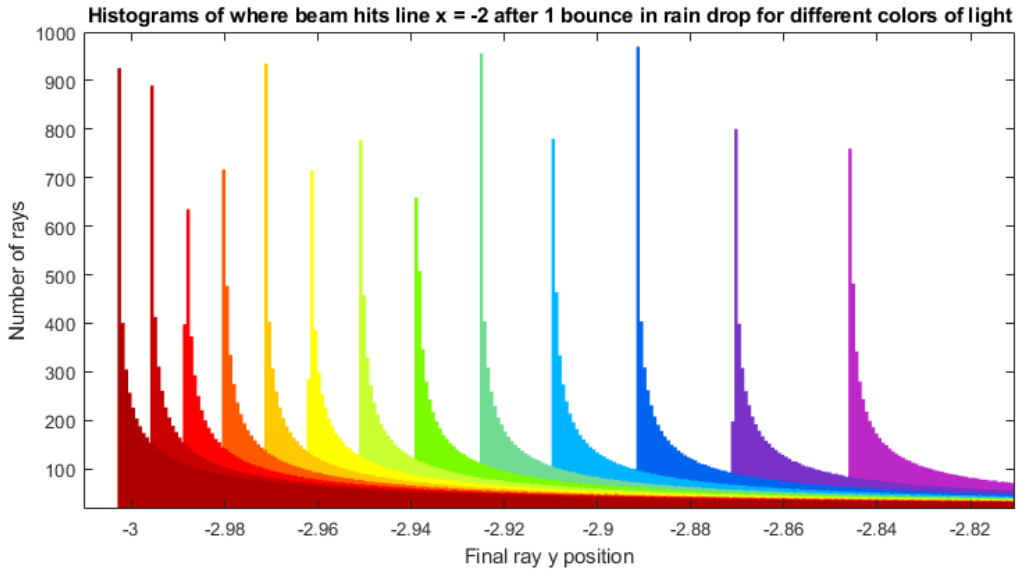

Figure 5: For each different colour of light, which has its own refractive index, this histogram shows the number hits at different positions along the line $x = -2$. This is for the primary rainbow case.

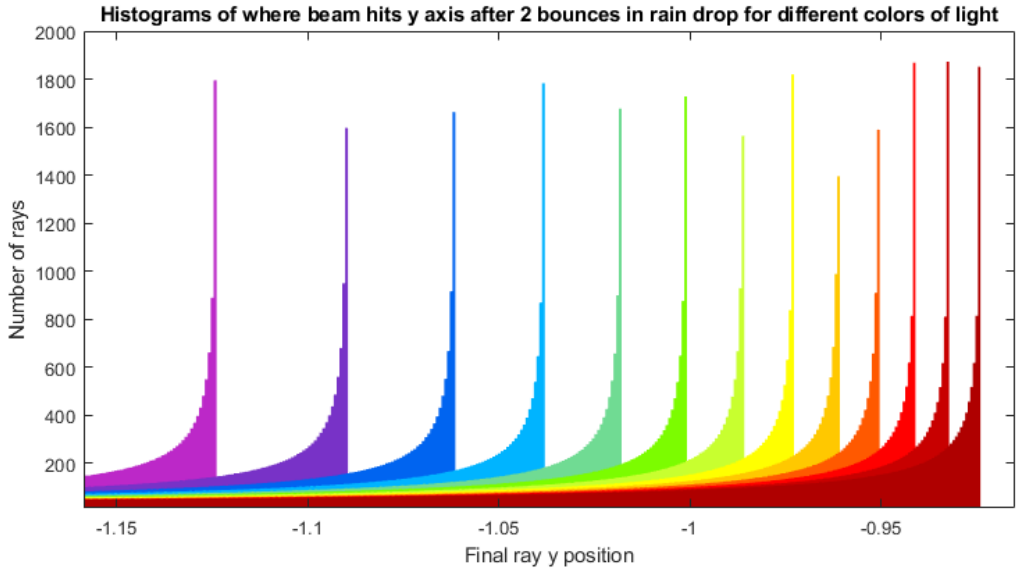

Figure 6: Similar to figure 5, but for the secondary rainbow.

The next two figures show, for p- and s-polarised light respectively, how much intensity of the light that originally entered the drop is left as it leaves.

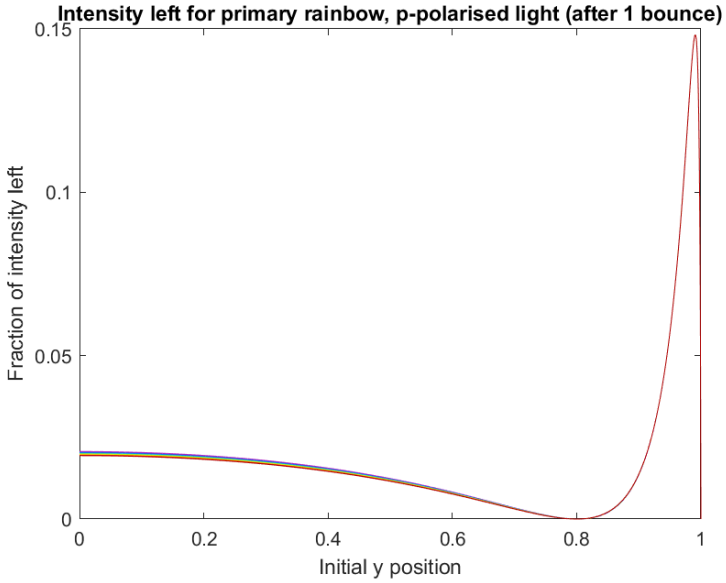

Figure 7: The intensity of the light as it hits $x = -2$, when the incident light is p-polarised. Note the dip at at $y = 0.8$, which happens as the light is reflected (bounces inside the drop) in the Brewster angle (see discussion).

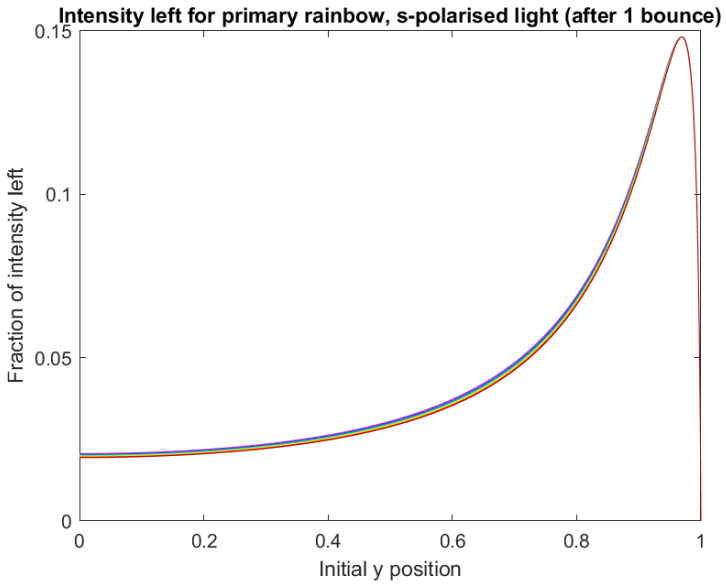

Figure 8: The intensity of the light as it hits $x = -2$, when the incident light is s-polarised.

The following last pair of figures show the deflection angle, as defined in sec. 2.2, plotted against the angle of incidence (the angle the incident ray makes with the normal of the drop). From these one can find the angle at which the rainbow should appear. This is elaborated in the discussion.

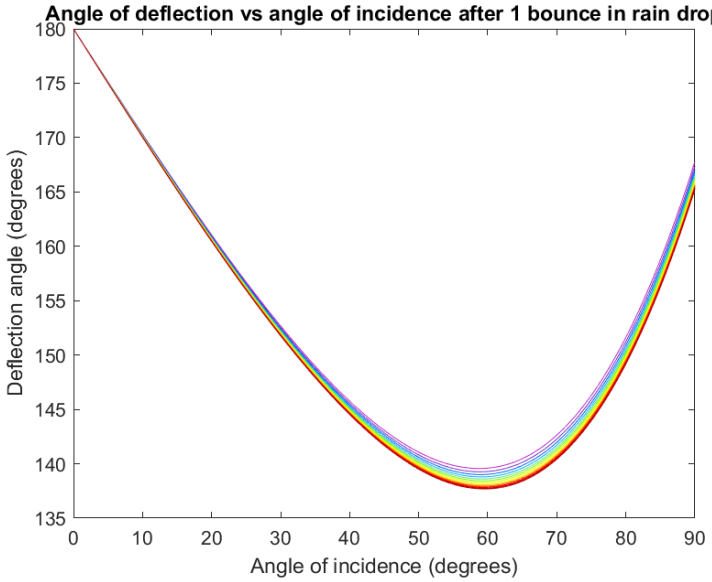

Figure 9: Deflection angle plotted against angle of incidence for single bounce (primary rainbow case).

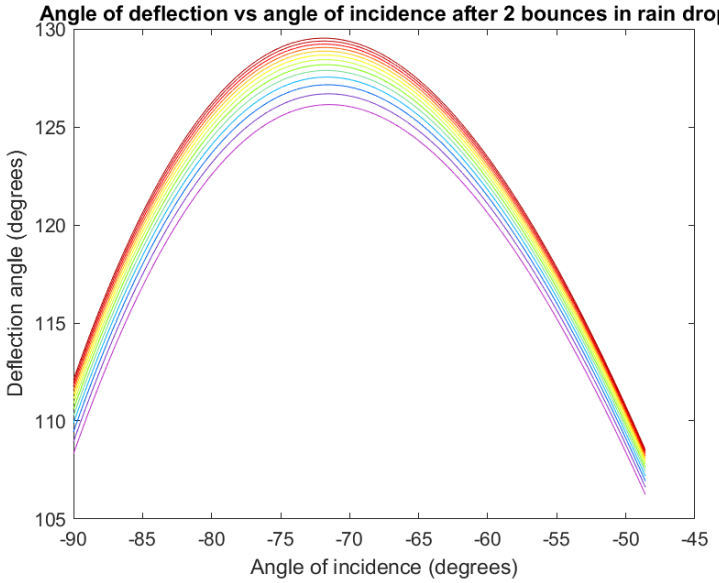

Figure 10: Similar to fig. 9, but for two bounces (secondary rainbow case)

6 Discussion

In the introduction three specific questions were posed. The first one was "What optical phenomena and mathematical models describe the creation of rainbows?", looking at the colourful histograms in fig. 5, one can be sure that we found a way to create rainbows. What is the key component in the underlying mathematics? It is that different colours of light have different refractive indices, making light of different colours refract slightly differently as they enter and exit the drop of water. The reflections on the curved inside of the drop collects the rays and makes it possible for the light to reflect back to our eyes. One could say that the mist in the sky after a rainfall acts as a prism, splitting the light into its constituent colours.

The second question was "At what angles does one see the primary and secondary rainbow?". This can be seen in fig. 9 and 10. These plots show the deflection angle, which controls where on our screen (or eye) the light ends up, plotted against the angle of incidence. We see in the plot for the primary rainbow case that around 59° of incidence, the deflection angle changes very slowly. This means that for this angle, most of the rays end up in roughly the same spot on the screen. All the light going through the drop is refracted into separate parts, but it is only around these 59° of incidence that enough rays bunch up for the rainbow to become visible. However, these 59° are not to confuse with at which angle one sees the rainbow. For this we need to check what angle of deflection the rays have as the rainbow is at it's strongest, which we find by looking at 59° of incidence and finding the corresponding deflection angle. The value is around 138°. Keeping in mind that we want the viewing angle for someone on the ground, and not the deflection angle, we use $180° - 138° = 42°$. This value agrees with most sources on the internet[2]. For the secondary rainbow, the same procedure gives us $180° - 128° = 52°$, which again is in agreement with just about any source I can find.

A third and final question was "What fraction of the incident light survives for different kinds of polarised light?". In fig. 7 and 8 under results it is seen how the surviving fraction of light varies with y-position. The most interesting part is the dip to zero for the p-polarised light. This dip happens because the reflection on the inside of the drop happens in the so-called Brewster angle. This is a special angle in which all p-polarised is transmitted and none is reflected.

The program for doing the path tracing of the light was actually made in two versions. There was an initial attempt. It was simple but naïve. In this version I barley used any trigonometry. Instead the program took tiny steps, marching in the direction of the ray. If the ray ever crossed the boundary of the drop the program decided what to do based on if it came from the inside or outside of the drop and how many bounces it had already done. This very simple approach worked, but it was incredibly slow. The method actually used in this report relies on finding distances and angles using trigonometry. This avoids taking the "tiny steps" and the code only takes a couple of minutes to run, even though we use 100000 rays per refractive index. The "stepping" version took something like 12 hours, and that was with the step size about as large as I dared making it.

7 Conclusion

The mathematical machinery based on simple optical laws and geometry described in section 3 worked well in conjunction with MATLAB-simulations for finding the angle at which rainbows appear. These angles were found to be 42° for the primary rainbow and 52° for the secondary rainbow. These values are in agreement with a majority sources on the Internet.

[2]See for example http://www.sciencecalculators.org/optics/rainbows/

Appendices

A project.m

```
clear all;
close all;

global enable_image_saving;
enable_image_saving = 1;
load_old_data = 1;

% Used for simulating colors of different light.
refractive_indices = [
    1.34451
    1.34235
    1.34055
    1.33903
    1.33772
    1.33659
    1.33560
    1.33472
    1.33393
    1.33322
    1.33257
    1.33197
    1.33141
];

% Each of these colors map the the corresponding refractive index above.
colors = [
    188 40 200;
    120 50 199;
    0 100 240;
    0 180 255;
    112 219 147;
    124 252 0;
    200 255 47;
    255 255 0;
    255 200 0;
    255 89 0;
    255 0 0;
    200 0 0;
    175 0 0;
]/255;

% We can use old, previously calculated data if we just want to experiment
    ↪ with the plotting.
if load_old_data == 1
    load('project_data.mat');
else
    num_beams = 100000;
    [primary_end_y, primary_start_y, primary_deflection_angles,
        ↪ primary_incident_angles, primary_p_intensity_left,
        ↪ primary_s_intensity_left] = calcuate_rainbow(num_beams, 0, 1, 1,
        ↪  refractive_indices, colors);
```

```
    [secondary_end_y, secondary_start_y, secondary_deflection_angles,
        ↪ secondary_incident_angles, secondary_p_itensity_left,
        ↪ secondary_s_intensity_left] = calcuate_rainbow(num_beams, -1,
        ↪ -0.75, 2, refractive_indices, colors);
end

% Histogram from primary rainbow
figure(1);
for ni = 1:length(refractive_indices)
    histogram(primary_end_y(ni, :), 'NumBins', 4000, 'FaceColor', colors(
        ↪ ni, :), 'LineStyle', 'none', 'FaceAlpha', 1);
    hold on;
end
ylabel('Hits at y position');
xlabel('y position');
title('Histograms of where beam hits line x=-2 after 1 bounce in rain
    ↪ drop for different colors of light');

% Plot of deflection angle against angle of incidence for primary rainbow
figure(2);
for ni = 1:length(refractive_indices)
    plot(rad2deg(primary_incident_angles(ni, :)), -rad2deg(
        ↪ primary_deflection_angles(ni, :)), 'color', colors(ni, :));
    hold on;
end
ylabel('Deflection angle (degrees)');
xlabel('Angle of incidence (degrees)');
title('Angle of deflection vs angle of incidence after 1 bounce in rain
    ↪ drop');
save_image('primary_def_angle');

% Plot intensity left for s polarised light for primary rainbow
figure(3);
for ni = 1:length(refractive_indices)
    plot(primary_start_y(ni, :), primary_s_intensity_left(ni, :), 'color',
        ↪  colors(ni, :));
    hold on;
end
ylabel('Fraction of intensity left');
xlabel('Initial y position');
title('Intensity left for primary rainbow, s-polarised light (after 1
    ↪ bounce)');
save_image('primary_s_inten_left');

% Plot intensity left for p polarised light for primary rainbow
figure(4);
for ni = 1:length(refractive_indices)
    plot(primary_start_y(ni, :), primary_p_intensity_left(ni, :), 'color',
        ↪  colors(ni, :));
    hold on;
end
ylabel('Fraction of intensity left');
xlabel('Initial y position');
title('Intensity left for primary rainbow, p-polarised light (after 1
    ↪ bounce)');
save_image('primary_p_inten_left');
```

```
% Histogram for secondary rainbow
figure(5);
for ni = 1:length(refractive_indices)
    histogram(secondary_end_y(ni, :), 'NumBins', 4000, 'FaceColor', colors
        ↪ (ni, :), 'LineStyle', 'none', 'FaceAlpha', 1);
    hold on;
end
ylabel('Hits at y position');
xlabel('y position');
title('Histograms of where beam hits y axis after 2 bounces in rain drop
    ↪ for different colors of light');

% Plot of deflection angle against angle of incidence for secondary
    ↪ rainbow
figure(6);
for ni = 1:length(refractive_indices)
    plot(rad2deg(secondary_incident_angles(ni, :)), -rad2deg(
        ↪ secondary_deflection_angles(ni, :)), 'color', colors(ni, :));
    hold on;
end
ylabel('Deflection angle (degrees)');
xlabel('Angle of incidence (degrees)');
title('Angle of deflection vs angle of incidence after 2 bounces in rain
    ↪ drop');
save_image('secondary_def_angle');

% Plot intensity left for s polarised light for secondary rainbow
figure(7);
for ni = 1:length(refractive_indices)
    plot(secondary_start_y(ni, :), secondary_s_intensity_left(ni, :), '
        ↪ color', colors(ni, :));
    hold on;
end
ylabel('Fraction of intensity left');
xlabel('Initial y position');
title('Intensity left for secondary rainbow, s-polarised light (after 2
    ↪ bounces)');
save_image('secondary_s_inten_left');

% Plot intensity left for p polarised light for secondary rainbow
figure(8);
for ni = 1:length(refractive_indices)
    plot(secondary_start_y(ni, :), secondary_p_intensity_left(ni, :), '
        ↪ color', colors(ni, :));
    hold on;
end
ylabel('Fraction of intensity left');
xlabel('Initial y position');
title('Intensity left for secondary rainbow, p-polarised light (after 2
    ↪ bounces)');
save_image('secondary_p_inten_left');

function save_image(name)
    global enable_image_saving;
    if enable_image_saving == 0
```

```
        return;
    end
    print(name, '-dpng');
end
```

B animation.m

```
clear all;
clf;

refractive_indices = [
    1.34451
    1.34235
    1.34055
    1.33903
    1.33772
    1.33659
    1.33560
    1.33472
    1.33393
    1.33322
    1.33257
    1.33197
    1.33141
];

colors = [
    188 40 200;
    120 50 199;
    0 100 240;
    0 180 255;
    112 219 147;
    124 252 0;
    200 255 47;
    255 255 0;
    255 200 0;
    255 89 0;
    255 0 0;
    200 0 0;
    175 0 0;
]/255;

t = 0;
time_step = 0.1;
while(true)
    t = t + time_step;
    b = abs(cos(t));

    clf;

    for i = 1:length(refractive_indices)
        n = refractive_indices(i);
        c = colors(i, :);
        calculate_path(b, 1, n, c, 1, -2);
```

```
    end

    axis([-2, 2, -4, 2]);
    pause(0.1);
    axis([-2, 2, -4, 2]);
end
```

C calcuate_rainbow.m

```
% Takes a bunch of refractive indices and calls the path tracing function
    ↪ calculate_path for each of them.
% The output variable hits_end_y describes where the rays end up.
function [hits_end_y, hits_start_y, hits_deflection_angles,
    ↪ hits_incident_angles, p_intensity_left, s_intensity_left] =
    ↪ calcuate_rainbow(num_beams, y_start, y_end, num_bounces,
    ↪ refractive_indices, colors)
    hits_end_y = [];
    hits_start_y = [];
    hits_deflection_angles = [];
    hits_incident_angles = [];
    p_intensity_left = [];
    s_intensity_left = [];

    % We use this to parallelize the computation.
    cluster = parcluster('local');
    cluster.NumWorkers = 14;
    saveProfile(cluster);

    % Change parfor to for to remove parallelization.
    parfor ni = 1:length(refractive_indices)
        n = refractive_indices(ni);
        disp(['starting index ', num2str(n)]);
        c = colors(ni, :);

        idx = 1;
        hits = [];
        deflection_angles = [];
        incident_angles = [];
        pi_lefts = [];
        si_lefts = [];
        start_ys = [];
        progress = 0;
        progress_step = num_beams / 20;

        % This loop probably goes from origin to radius = 1 (for primary
            ↪ rainbow) in lots of small steps
        for y = linspace(y_start, y_end, num_beams)
            [hit_y, incident_angle, def_angle, pi_left, si_left] =
                ↪ calculate_path(y, num_bounces, n, c, 0, -2);

            % We use hit_y == 0 as "something went wrong"
            if (hit_y ~= 0)
                hits(idx) = hit_y;
                start_ys(idx) = y;
```

```
                deflection_angles(idx) = def_angle;
                incident_angles(idx) = incident_angle;
                pi_lefts(idx) = pi_left;
                si_lefts(idx) = si_left;
                idx = idx + 1;

                % For showing calculation progress.
                if idx > progress + progress_step
                    disp(['index ', num2str(n),': ', num2str(round((idx
                        ↪ *100)/num_beams)), '%']);
                    progress = progress + progress_step;
                end
            end
        end
        disp(['index ', num2str(n),' done']);
        hits_end_y(ni, :) = hits;
        hits_start_y(ni, :) = start_ys;
        hits_deflection_angles(ni, :) = deflection_angles;
        hits_incident_angles(ni, :) = incident_angles;
        p_intensity_left(ni, :) = pi_lefts;
        s_intensity_left(ni, :) = si_lefts;
    end
end
```

D calculate_path.m

```
function [hit_y, incident_angle, deflection_angle, p_intensity_left,
    ↪ s_intensity_left] = calculate_path(start_height,
    ↪ max_internal_bounces, refractive_index, color, plot_result, x_target
    ↪ )
r = 1;
start_y = r*start_height;

if start_y > r
    hit_y = 0;
    return;
end

start_x = -2;
start = [start_x, start_y];
path_x = [];
path_y = [];
n = refractive_index;

% Represent drop with circle
if plot_result == 1
    viscircles([0, 0], r, 'Color', 'b', 'LineWidth', 0.1);
end

axis([-2, 2, -2, 2]);

i = 1;
function add_to_plot(point)
    if plot_result == 0
```

```
        return;
    end

    path_x(i) = point(1);
    path_y(i) = point(2);
    i = i + 1;
end

add_to_plot(start);

% angle from normal when we hit wall from outside
ext_angle = asin(start_y/r);
incident_angle = ext_angle;
% angle from normal after refracting into drop
int_angle = asin(start_y/(r*n));

% This uses fresnels coefficients
p_intensity_left = 1*((sin(2*ext_angle)*sin(2*int_angle))/(sin(ext_angle+
    ↪ int_angle)^2*cos(ext_angle-int_angle)^2)) ...
                    *(tan(ext_angle-int_angle)/tan(ext_angle+int_angle))
                        ↪ ^(2*max_internal_bounces) ...
                    *((sin(2*int_angle)*sin(2*ext_angle))/(sin(int_angle+
                        ↪ ext_angle)^2*cos(int_angle-ext_angle)^2));

s_intensity_left = 1*((sin(2*int_angle)*sin(2*ext_angle))/sin(ext_angle+
    ↪ int_angle)^2) ...
                    *(sin(ext_angle-int_angle)/sin(ext_angle+int_angle))
                        ↪ ^(2*max_internal_bounces) ...
                    *((sin(2*ext_angle)*sin(2*int_angle))/sin(int_angle+
                        ↪ ext_angle)^2);

x_enter = -sqrt(r^2 - start_y^2); % Advance to edge of drop
internal_bounce_length = 2*r*sqrt(1-(start_y^2/(n*r)^2)); % Distance "b"
    ↪ in report
cur_pos = [x_enter, start_y]; % Keeps track of current position of the ray
add_to_plot(cur_pos);
enter_angle = -(ext_angle - int_angle); % The angle known as theta in the
    ↪ report
dir = [cos(enter_angle), sin(enter_angle)]; % The current angle of the ray

% Do all the bounces inside the drop
for bounce = 1:max_internal_bounces
    cur_pos = cur_pos + dir*internal_bounce_length; % Advance using
        ↪ direction and bounce length
    add_to_plot(cur_pos);
    dir = bounce_inside(cur_pos, dir); % This function is just the
        ↪ reflection formula
end

cur_pos = cur_pos + dir*internal_bounce_length; % Advance one last time
add_to_plot(cur_pos);
deflection_angle = int_angle - ext_angle + atan2(dir(2), dir(1));
wanted_x = cur_pos(1) - x_target;
y_distance_to_screen = -(wanted_x * tan(pi - abs(deflection_angle))); %
    ↪ Find how far it is to the screen
```

```
if deflection_angle > -pi/2 && deflection_angle < 0 % Fix for weird bug
    ↪ where everything inverted for some angles.
    y_distance_to_screen = -y_distance_to_screen;
end

hit_y = cur_pos(2) + y_distance_to_screen; % The end y-position
add_to_plot([x_target, hit_y]);

if plot_result == 1
    hold on;
    plot(path_x, path_y, 'color', color);
end
end
```

E bounce_inside.m

```
function new_dir = bounce_inside(pos, cur_dir)
    towards_origin = -pos./norm(pos);
    backwards = -cur_dir./norm(cur_dir);
    proj = (dot(towards_origin, backwards)/dot(towards_origin,
        ↪ towards_origin))*towards_origin;
    new_dir = backwards - 2*(backwards - proj);
end
```