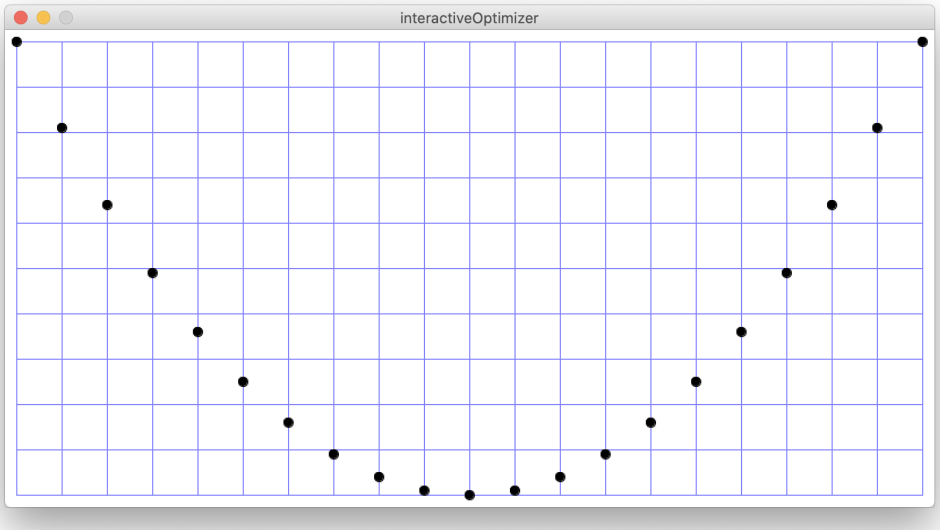


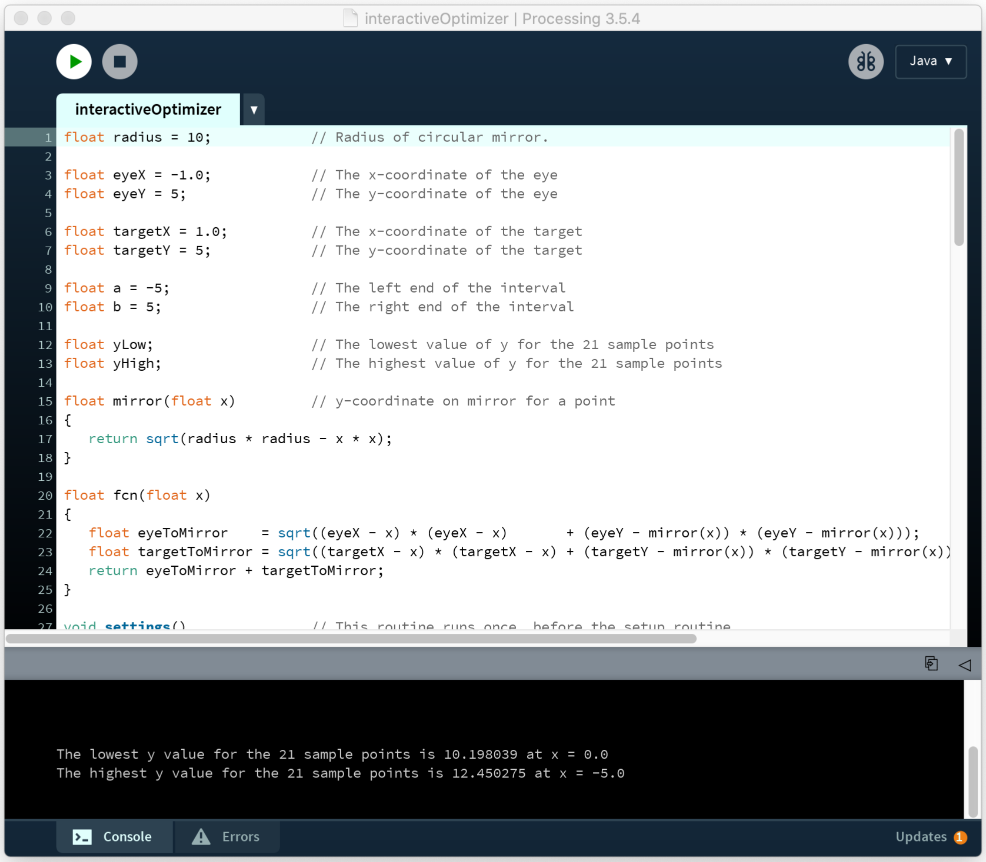
Figure 1: Looking at 21 Sample Points

Interactive Optimizer

This `Processing` program is designed to help find the minimum or maximum of a function,

$$y = f(x),$$

on an interval $[a, b]$ and find the value of x that produces this minimum or maximum value. Figure 1 shows the basic idea underlying this program. In this figure the interval $[a, b] = [-5, 5]$. If you run this program you should see this figure. The program computes the values of a function $f(x)$ for 21 equally spaced sample points in the interval. The minimum appears to be the point in the center of the interval. If you look in the console Figure 2 you will see that the lowest y value for these 21 sample points is 10.198039 at the point $x = 0.0$ and the highest y value for these 21 sample points is 12.450275 at the point $x = -5.0$. The sample points are spaced 0.5 units apart. Based on this figure it looks as if the minimum (which might not be one of the sample points) is in the interval $[-0.5, 0.5]$. If you press the mouse button on the minimum point the program will do the same thing on the interval $[-0.5, 0.5]$ around that point. This will let you narrow down the true minimum. You can repeat this several times narrowing down the location of the



```
float radius = 10;            // Radius of circular mirror.

float eyeX = -1.0;            // The x-coordinate of the eye
float eyeY = 5;               // The y-coordinate of the eye

float targetX = 1.0;          // The x-coordinate of the target
float targetY = 5;            // The y-coordinate of the target

float a = -5;                 // The left end of the interval
float b = 5;                  // The right end of the interval

float yLow;                   // The lowest value of y for the 21 sample points
float yHigh;                  // The highest value of y for the 21 sample points

float mirror(float x)         // y-coordinate on mirror for a point
{
   return sqrt(radius * radius - x * x);
}

float fcn(float x)
{
   float eyeToMirror    = sqrt((eyeX - x) * (eyeX - x)       + (eyeY - mirror(x)) * (eyeY - mirror(x)));
   float targetToMirror = sqrt((targetX - x) * (targetX - x) + (targetY - mirror(x)) * (targetY - mirror(x))
   return eyeToMirror + targetToMirror;
}

void settings()               // This routine runs once, before the setup routine
```

```
The lowest y value for the 21 sample points is 10.198039 at x = 0.0
The highest y value for the 21 sample points is 12.450275 at x = -5.0
```

Figure 2: The Program and Console

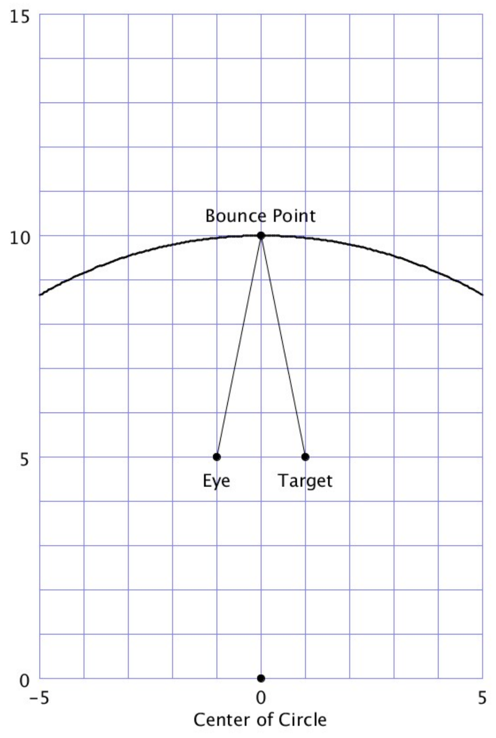


Figure 3: Example Problem

minimum. The calculations involved all involve some roundoff error. So usually you will never get the exact minimum and after narrowing it down several times the function values will look very strange. The same procedure can be used to find a maximum instead of a minimum.

In these notes we illustrate the use of this program by finding the path followed by a light ray traveling from a target to a curved mirror and then to an eye. See Figure 3. The curved mirror is part of the circle of radius $R = 10$ centered at the origin. This circle can be described by the equation:

$$x^2 + y^2 = R^2.$$

If x is the x-coordinate of a point on the mirror then

$$y = \sqrt{R^2 - x^2}$$

is its y-coordinate. For this problem we are looking at light rays that bounce off the top half of the circle.

The eye is located at the point $(-1, 5)$ and the target is located at the point $(1, 5)$. The point at which the light ray bounces off the mirror is determined by Fermat's Principle, which says that in this situation the light ray chooses the shortest possible path. Drawing a rough sketch we can see that the point at which the light ray bounces off the mirror will be in the interval $[-5, 5]$.

Lines 1-10 of the program specify the radius of the mirror, the locations of the eye and the target and the initial interval $[a, b]$. Lines 15-18 define a function that describes the shape of the mirror. Its input is a the x-coordinate of a point on the mirror and its output is the y-coordinate of the point. This function is used in lines 20-25 that specify the function to be minimized or maximized. These lines are the ones you will modify for other optimization problems.

The program is listed below but you should look at it and run it using `Processing`.

Program Listing

```
float radius = 10;            // Radius of circular mirror.

float eyeX = -1.0;            // The x-coordinate of the eye
float eyeY = 5;               // The y-coordinate of the eye

float targetX = 1.0;          // The x-coordinate of the target
float targetY = 5;            // The y-coordinate of the target

float a = -5;                 // The left end of the interval
float b = 5;                  // The right end of the interval

float yLow;                   // The lowest value of y for the 21 sample points
float yHigh;                  // The highest value of y for the 21 sample points

float mirror(float x)         // y-coordinate on mirror for a point
{
   return sqrt(radius * radius - x * x);
}

float fcn(float x)
{
   float eyeToMirror    = sqrt((eyeX - x) * (eyeX - x)       + (eyeY - mirror(x)) * (eyeY - mirror(x)));
   float targetToMirror = sqrt((targetX - x) * (targetX - x) + (targetY - mirror(x)) * (targetY - mirror(x
   return eyeToMirror + targetToMirror;
}

void settings()               // This routine runs once, before the setup routine
{
  size(821, 421);
}

void setup()                  // The setup routine is run once when the program (sketch )starts
{
   plot21();                  // Plot 21 points in the interval
}

void draw()                   // The draw routine is not used but it is necessary
{
}

void plot21()
{
   float x, y, xLowest, xHighest;                      // Used for computations
   background(255);                                    // Clear display area to all white
   stroke(128, 128, 255);                              // Grid lines are green
   for(int i = 0; i <= 20; i = i + 1)                  // Draw vertical grid lines
   {
      line(10 + i * 40, 10, 10 + i * 40, 410);
   }
   for(int i = 0; i <= 10; i = i + 1)                // Draw horizontal grid lines
   {
      line(10, 10 + i * 40, 810, 10 + i * 40);
   }
   yLow = fcn(a);
   xLowest = a;
   yHigh = fcn(a);
   xHighest = a;
   for(int i = 1; i <= 20; i = i + 1)                // Find the minimum and maximum values of y for the 2
   {
     x = a + i * 0.05 * (b - a);                     // x-value of this sample point
     y = fcn(x);                                     // y-value of this sample point
     if(y < yLow)                                    // Check if this sample point is below the previous l
     {
        yLow = y;
        xLowest = x;
     }
     if(y > yHigh)                                   // Check if this sample point is above the previous h
     {
        yHigh = y;
        xHighest = x;
     }
   }
   print("The lowest y value for the 21 sample points is ");
   print(yLow);
   print(" at x = ");
   println(xLowest);
   print("The highest y value for the 21 sample points is ");
   print(yHigh);
   print(" at x = ");
   println(xHighest);
   println("");
   stroke(0);                                        // The dots at the samplepoints are black
   fill(0);
   for(int i = 0; i <= 20; i = i + 1)                // Plot sample points
   {
     x = a + i * 0.05 * (b - a);                     // x-value of this sample point
     y = fcn(x);                                     // y-value of this sample point
     circle(xOf(x), yOf(y), 8);
   }
}

void mousePressed()
{
   float dx = (b - a)/20.0;                          // x-interval width
   int x = round((mouseX - 10.0)/40);
   a = a + (x - 1) * dx;
   b = a + 2 * dx;
   plot21();
```

```
}

int xOf(float x)
{
   return round(10 + 800 * (x - a)/(b - a));               // Return the display area x-coordinate of a po
}

int yOf(float y)
{
   return round(410 - 400 * (y - yLow)/(yHigh - yLow));    //  Return the display area y-coordinate of a p
}
```