1 The Graph of a Function

A univariate function, such as $f(x) = 1 - x^2/2$, can be thought of in many different ways. For example,

- It can be represented through a rule of what it does to x, as above. This is useful for computing numeric values.
- it can be interpreted verbally, as in *square* x, take half then *subtract* from one. This can give clarity to what the function does.
- It can be thought of in terms of its properties: a polynomial, continuous, U-shaped, an approximation for $\cos(x)$ near 0, ...
- it can be visualized graphically. This is useful for seeing the qualitative behaviour of a function.

The graph of a univariate function is just a set of points in the Cartesian plane. These points come from the relation $(x, f(x))$ that defines the function. Operationally, a sketch of the graph will consider a handful of such pairs and then the rest of the points will be imputed.

For example, a typical approach to plot $f(x) = 1 - x^2/2$ would be to choose some values for x and find the corresponding values of y. This might be organized in a "T"-table:

```
 x | y
--------
-2 |  -1
-1 |  1/2
 0 |   1
 1 |  1/2
 2 |  -1
 3 | -7/2
```

These pairs would be plotted in a Cartesian plane and then connected with curved lines. A good sketch is aided by knowing ahead of time that this function describes a parabola which is curving downwards.

It is important to note that this sketch would not include *all* the pairs $(x, f(x))$, as their extent is infinite.

1.1 Graphing a function with Julia

`Julia` has several different options for rendering graphs, all in external packages. We will focus in these notes on the `Plots` package, which provides a common interface to several different plotting backends.

As such, we begin by loading the accompanying `CalculusWithJulia` package and the `Plots` package.

```
using CalculusWithJulia
using Plots
```

`Plots` is a frontend for one of several backends. `Plots` comes with a backend for web-based graphics (call `plotly()` to specify that); a backend for static graphs (call `gr()` for that). If the `PyPlot` package is installed, calling `pyplot()` with set that as a backend. For terminal usage, if the `UnicodePlots` package is installed, calling `unicodeplots()` will enable that usage.

The `plotly` backend is part of the `Plots` package. Other backends require installation. Some others are `pyplot`, for the `PyPlot` package; `gr`, for the `GR` package, `plotlyjs`, for an enhanced Plotly backend through `PlotlyJS`.

We use a mix in these notes: `plotly` for a default, as it has interactive graphics; `gr` for some static graphs; and `pyplot` for plotting surfaces. Just using `pyplot` is an option for all the plots. """)

‘‘‘

With `Plots` loaded, it is straightforward to graph a function.

For example, to graph $f(x) = 1 - x^2/2$ over the interval $[-3, 3]$ we have:

```
f(x) = 1 - x^2/2
plot(f, -3, 3)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

The `plot` command does the hard work behind the scenes. It needs 2 pieces of information:

- **What** to plot. With this invocation, this detail is expressed by passing a function object to `plot`
- **Where** to plot. As with a sketch, it is impossible in this case to render a graph with all possible x values in the domain of f, so we need to pick some viewing window. In the example this is $[-3, 3]$ which is expressed by passing the two endpoints as the second and third arguments.

Plotting a function is then this simple: `plot(f, a, b)`.

> *A basic template:* Many operations we meet will take the form `action(function, args...)`, as the call to `plot` does. The template shifts the focus to the action to be performed. This is a declarative style, where the details to execute the action are only exposed as needed.

To see some other graphs, we can plot the `sin` function over one period with:

```
plot(sin, 0, 2pi)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

We can make a graph of $f(x) = (1 + x^2)^{-1}$ over $[-3, 3]$ with

```
f(x) = 1 / (1 + x^2)
plot(f, -3, 3)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Or a graph of $f(x) = e^{-x^2/2}$ over $[-2, 2]$ with:

```
f(x) = exp(-x^2/2)
plot(f, -2, 2)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

We could skip the first step of defining a function by using an *anonymous function* instead. For example, to plot $f(x) = \cos(x) - x$ over $[0, \pi/2]$ we could do:

```
plot(x -> cos(x) - x, 0, pi/2)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

> The function object in the general pattern `action(function, args...)` is commonly specified in one of three ways: by a name, as with `f`; as an anonymous function; or as the return value of some other action through composition.

Making a graph with `Plots` is easy, but producing a graph that is informative can be a challenge, as the choice of a viewing window can make a big difference in what is seen. For example, trying to make a graph of $f(x) = \tan(x)$, as below, will result in a bit of a mess below - the chosen viewing window crosses several places where the function blows up:

```
f(x) = tan(x)
plot(f, -10, 10)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Though this graph shows the asymptote structure and periodicity, it doesn't give much insight into each period or even into the fact that the function is periodic.

1.2 The details of graph making

The actual details of making a graph of f over $[a, b]$ are pretty simple and follow the steps in making a "T"-table:

- A set of x values are created between a and b.
- A corresponding set of y values are created.
- The pairs (x, y) are plotted as points and connected with straight lines.

The only real difference is that when drawing by hand, we might know to curve the lines connecting points based on an analysis of the function. As `Julia` doesn't consider this, the points are connected with straight lines like a dot-to-dot puzzle.

We can use `range` to create the x values and broadcasting to create the y values.

For example, if we were to plot $f(x) = \sin(x)$ over $[0, 2\pi]$ using 10 points, we might do:

```
f(x) = sin(x)
xs = range(0, 2pi, length=10)
ys = f.(xs)
```

```
10-element Array{Float64,1}:
  0.0
  0.6427876096865393
  0.984807753012208
  0.8660254037844387
  0.3420201433256689
 -0.34202014332566866
 -0.8660254037844385
 -0.9848077530122081
 -0.6427876096865396
 -2.4492935982947064e-16
```

Finally, to plot the set of points and connect with lines. If the collection of points are specified in place of a function, the graph is produced:

```
plot(xs, ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

This plots the points as pairs and then connects them in order using straight lines. Basically, it creates a dot-to-dot graph. The above graph looks primitive, as it doesn't utilize enough points. A solution might be to just use more than 10 points. In `Plots` an algorithm is utilized that adds more points when the function is more "curvy" and uses fewer points when it is "straighter."

NaN values At times it is not desirable to draw lines between each succesive point. For example, if there is a discontinuity in the function or if there were a vertical asymptote, such as what happens at 0 with $1/x$. There is a convention that if the y value for a point is `NaN` that no lines will connect. This is convenient in many cases, though not with $1/x$ as `1/0` is `Inf` and not `NaN`. (Unlike `0/0` which is NaN.)

Here is one way to plot $f(x) = 1/x$ over $[-1, 1]$:

```
f(x) = 1/x
xs = range(-1, 1, length=251)
ys = f.(xs)
ys[xs .== 0.0] .= NaN
plot(xs, ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

By using an odd number of points, we should have that 0.0 is amongst the `xs`. The next to last line replaces the y value that would be infinite with `NaN`.

Alternatively, you might modify the function itself to return `NaN` for values close to 0:

```
g(x) = abs(x) < .05 ? NaN : f(x)
plot(g, -1, 1)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Example: Reflections The graph of a function may be reflected through a line, as those seen with a mirror. For example, a reflection through the y axis takes a point (x, y) to the point $(-x, y)$. We can easily see this graphically, when we have sets of x and y values through a judiciously placed minus sign.

For example, to plot $\sin(x)$ over $(-\pi, \pi)$ we might do:

```
xs = range(-pi, pi, length=100)
ys = sin.(xs)
plot(xs, ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

To reflect this graph through the y axis, we only need to plot `-xs` and not `xs`:

```
plot(-xs, ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Looking carefully we see there is a difference. (How?)

There are four common reflections discussed:

- reflection through the y-axis takes (x, y) to $(-x, y)$.
- reflection through the x-axis takes (x, y) to $(x, -y)$.
- reflection through the origin takes (x, y) to $(-x, -y)$.

- reflection through the line $y = x$ takes (x, y) to (y, x).

For the $\sin(x)$ graph, we see that reflecting through the x axis produces the same graph as reflecting through the y axis:

```
plot(xs, -ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

However, reflecting through the origin leaves this graph unchanged:

```
plot(-xs,  -ys)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

> An *even function* is one where reflection through the y axis leaves the graph unchanged. That is, $f(-x) = f(x)$. An *odd function* is one where a reflection through the origin leaves the graph unchanged, or symbolically $f(-x) = -f(x)$.

If we try reflecting the graph of $\sin(x)$ through the line $y = x$, we have:

```
plot(ys, xs)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

This is the graph of the equation $x = \sin(y)$, but is not the graph of a function as the same x can map to more than one y value. (The new graph does not pass the "vertical line" test.)

However, for the sine function we can get a function from this reflection if we choose a narrower viewing window:

```
xs = range(-pi/2, pi/2, length=100)
ys = [sin(x) for x in xs]
plot(ys, xs)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

The graph is that of the "inverse function" for $\sin(x), x$ in $[-\pi/2, \pi/2]$.

1.3 Layers

Graphing more than one function over the same viewing window is often desirable. This is easily done in `Plots` by specifying a vector of functions as the first argument to `plot` instead of a single function object. The notation is `[f1, f2, ..., fn]`. For example, to see that the polynomial and the cosine function are "close" near 0, we can plot *both* $\cos(x)$ and the function $f(x) = 1 - x^2/2$ over $[-\pi/2, \pi/2]$:

```
f(x) = 1 - x^2/2
plot(cos, -pi/2, pi/2)
plot!(f, -pi/2, pi/2)
```

```
Plot{Plots.PlotlyBackend() n=2}
```

The vector `[cos, f]` is comprised of function objects and not function calls, such as `cos(x)`.

Another useful function to add to a plot is one to highlight the x axis. This makes identifying zeros of the function easier. The anonymous function `x->0` will do this. But, perhaps less cryptically, so will the function `zero`. For example

```
f(x) = x^5 - x + 1
plot(f, -1.5, 1.4)
plot!(zero, -1.5, 1.4)
```

```
Plot{Plots.PlotlyBackend() n=2}
```

1.3.1 Adding layers explicitly

It can be useful at times to create these layers one by one. This might better match the workflow. As well, it affords the possibility of different kinds of layers.

To redo the previous plot adding first the graph of the function `f` and then the function `zero`, we have:

```
plot(f, -1.5, 1.4)
plot!(zero, -1.5, 1.4)
```

```
Plot{Plots.PlotlyBackend() n=2}
```

The `plot!` call adds a layer. We still specify the limits for the plot, though this *could* be computed from the figure. (If using `SimplePlots` or `Makie` with `CalculusWithJulia` this will be the case.)

For another example, suppose we wish to plot the function $f(x) = x \cdot (x-1)$ over the interval $[-1, 2]$ and emphasize with points the fact that 0 and 1 are zeros. We can do this with two layers: the first to graph the function, the second to graph the points.

```
f(x) = x*(x-1)
plot(f, -1, 2)
scatter!([0,1], [0,0])
```

```
Plot{Plots.PlotlyBackend() n=2}
```

The 3 main functions for adding layers are:

- `plot!(f, a, b)` to add the graph of the function `f`
- `scatter!(xs, ys)` to add points $(xs_1, ys_1), (xs_2, ys_2), \ldots$.
- `annotate!((x,y, label))` to add a label at (x, y)

Julia has a convention to use functions named with a `!` suffix to indicate that they mutate some object. In this case, the object is the current graph, though it is implicit. Both `plot!`, `scatter!`, and `annotate!` (others too) do this by adding a layer.

1.3.2 Additional arguments to modify a graphic

The `Plots` package provides many arguments for modifying a graphic, here we mention just a few:

- `plot(..., title="main title", xlab="x axis label", ylab="y axis label")`: add title and label information to a graphic
- `plot(..., color="green")`: this argument can be used to adjust the color of the drawn figure (color can be a string,`"green"`, or a symbol, `:green`)
- `plot(..., linewidth=5)`: this argument can be used to adjust the width of drawn lines
- `plot(..., xlims=(a,b), ylims=(c,d)`: either or both `xlims` and `ylims` can be used to control the viewing window
- `plot(..., legend=false)`: by default, different layers will be indicated with a legend, this will turn off this feature
- `plot(..., linestyle=:dash)`: will change the line style of the plotted lines to dashed lines. Also `:dot`, ...
- `plot(..., aspect_ratio=:equal)`: will keep x and y axis on same scale so that squares look square.

For plotting points with `scatter` or `scatter!` the markers can be adjusted via

- `scatter(..., markersize=5)`: increase marker size
- `scatter(..., marker=:square)`: change the marker (uses a symbol, not a string to specify)

Of course, zero, one or more of these can be used on any given call to `plot`, `plot!`, `scatter` or `scatter!`.

1.4 Parametric graphs

If we have two functions $f(x)$ and $g(x)$ there are a few ways to investigate their joint behaviour. As just mentioned, we can graph both f and g over the same interval by passing the vector of functions `[f,g]` to `plot`. Such a graph allows an easy comparison of the shape of the two functions and can be useful in solving $f(x) = g(x)$. For the latter, the graph of $h(x) = f(x) - g(x)$ is also of value: solutions to $f(x) = g(x)$ appear as crossing points on the graph of `[f,g]`, whereas they appear as zeros (crossings of the x-axis) when `h` is plotted.

A different graph can be made to compare the two side-by-side. This is a parametric plot. Rather than plotting points $(x, f(x))$ and $(x, g(x))$ with two separate graphs, the graph consists of points $(f(x), g(x))$. We illustrate with some examples below:

Example The most "famous" parametric graph is one we are already familiar with, as it follows the parametrization of points on the unit circle by the angle, which we denote by `x`:

```
f(x) = cos(x); g(x) = sin(x)
xs = range(0, 2pi, length=100)
plot(f.(xs), g.(xs))
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Any point (a, b) on this graph is represented by $(\cos(x), \sin(x))$ for some value of x, and in fact multiple values of x, since $x + 2k\pi$ will produce the same (a, b) value as x will.

Making the parametric plot is similar to creating a plot using lower level commands. There is a function to generate a sequence of value to approximate the x values used (`xs`), a set of commands to create the corresponding function values (e.g., `f.(xs)`), and some instruction on how to represent the values, in this case with lines connecting the points.

There is a convenience interface for these plots - just pass the two functions in:

```
plot(f, g, 0, 2pi)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

Example Looking at growth. Comparing x^2 with x^3 can run into issues, as the scale gets big:

```
g(x) = x^2
f(x) = x^3
plot(g, 0, 25)
plot!(f, 0, 25)
```

```
Plot{Plots.PlotlyBackend() n=2}
```

In the above, f is already 25 times larger on the scale of $[0, 25]$ and this only gets worse if the viewing window were to get larger. However, the parametric graph is quite different:

```
xs = range(0, 5, length=100)
plot(g, f, 0, 25)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

In this graph, as $f(x)/g(x) = x$, as x gets large, the ratio stays reasonable.

Example Parametric plots are useful to compare the ratio of values near a point. In the above example, we see how this is helpful for large `x`. This example shows it is convenient for a fixed `x`, in this case `x=0`.

Plot $g(x) = x - \sin(x)$ and $g(x) = x^3$ around $x = 0$:

```
g(x) = x - sin(x)
f(x) = x^3
plot(g, f, -pi/2, pi/2)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

This graph is *nearly* a straight line. At the point $(0, 0) = (g(0), f(0))$, we see that both functions are behaving in a similar manner, though the slope is not 1, so they do not increase at exactly the same rate.

Example: Etch A Sketch Etch A sketch is a drawing toy where two knobs control the motion of a pointer, one knob controlling the x motion, the other the y motion. The trace of the movement of the pointer is recorded until the display is cleared by shaking. Shake to clear is now a motion incorporated by some smart-phone apps.

Playing with the toy makes a few things become clear:

- Twisting just the left knob (the horizontal or x motion) will move the pointer left or right, leaving a horizontal line. Parametrically, this would follow the equations $g(t) = \xi(t)$ for some ξ and $f(t) = 0$.
- Twisting just the right knob (the vertical or y motion) will move the pointer up or down, leaving a vertical line. Parametrically, this would follow the equations $g(t) = 0$ and $f(t) = \psi(t)$ for some ψ.
- Drawing a line with a slope different from 0 or ∞ requires moving both knobs at the same time. A 45° line with slope $m = 1$ can be made by twisting both at the same rate, say through $g(t) = ct$, $f(t) = ct$. It doesn't matter how big c is, just that it is the same for both g and f. Creating a different slope is done by twisting at different rates, say $g(t) = ct$ and $f(t) = dt$. The slope of the resulting line will be d/c.
- Drawing a curve is done by twisting the two knobs with varying rates.

These all apply to parametric plots, as the Etch A Sketch trace is no more than a plot of $(g(t), f(t))$ over some range of values for t, where g describes the movement in time of the left knob and f the movement in time of the right.

Now, thinking about the last problem in the context of this, we saw in the last problem that the parametric graph was nearly a line - so close the eye can't really tell otherwise. That means that the rates of change in t of both $g(t) = t - \sin(t)$ and $f(t) = t^3$ for t around 0 are in a nearly fixed ratio, as otherwise the graph would have more curve in it.

Example: Spirograph Parametric plots can describe a richer set of curves than can plots of functions. Plots of functions must pass the "vertical-line test", as there can be at most one y value for a given x value. This is not so for parametric plots, as the circle example above shows. Plotting sines and cosines this way is the basis for the once popular Spirograph toy. The curves drawn there are parametric plots where the functions come from rolling a smaller disc either around the outside or inside of a larger disc.

Here is an example using a parameterization provided on the Wikipedia page where R is the radius of the larger disc, r the radius of the smaller disc and $\rho < r$ indicating the position of the pencil within the smaller disc.

```
R, r, rho = 1, 1/4, 1/4
g(t) = (R-r) * cos(t) + rho * cos((R-r)/r * t)
f(t) = (R-r) * sin(t) - rho * sin((R-r)/r * t)

plot(g, f, 0, max((R-r)/r, r/(R-r))*2pi)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

In the above, one can fix $R = 1$. Then different values for `r` and `rho` will produce different graphs. These graphs will be periodic if $(R - r)/r$ is a rational. (Nothing about these equations requires $\rho < r$.)

1.5 Questions

⊛ Question
Plot the function $f(x) = x^3 - x$ over $[-2, 2]$. How many zeros are there?

⊛ Question
Plot the function $f(x) = x^3 - x$. When is the function positive?

1. `(-Inf, -0.577)` and `(0.577, Inf)`
2. `(-Inf, -1)` and `(0,1)`
3. `(-1, 0)` and `(1, Inf)`

⊛ Question
Plot the function $f(x) = 3x^4 + 8x^3 - 18x^2$. Where (what x value) is the smallest value? (That is, for which input x is the output $f(x)$ as small as possible.

⊛ Question
Plot the function $f(x) = 3x^4 + 8x^3 - 18x^2$. When is the function increasing?

1. `(-Inf, -4.1)` and `(1.455, Inf)`
2. `(-Inf, -3)` and `(0, 1)`
3. `(-3, 0)` and `(1, Inf)`

⊛ Question
The function $f(x) = (x^3 - 2x)/(2x^2 - 10)$ is a rational function with issues when $2x^2 = 10$, or $x = -\sqrt{5}$ or $\sqrt{5}$.

Plot this function from -5 to 5. How many times does it cross the x axis?

⊛ Question
A trash collection plan charges a flat rate of 35 dollars a month for the first 10 bags of trash and is 4 dollars a bag thereafter. Which function will model this:

1. `f(x) = x <= 4 ? 35.0 : 35.0 + 10.0 * (x-4)`
2. `f(x) = x <= 10 ? 35.0 : 35.0 + 4.0 * (x-10)`
3. `f(x) = x <= 35.0 ? 10.0 : 10.0 + 35.0 * (x-4)`

Make a plot of the model. Graphically estimate how many bags of trash will cost 55 dollars.

⊛ Question
Plot the functions $f(x) = \cos(x)$ and $g(x) = x$. Estimate the x value of where the two graphs intersect.

⊛ Question
Graphing both `f` and the line $y = 0$ can be done through `[f, zero]`. This helps focus on the *zeros* of `f`. When `f(x)=log(x)-2`, plot `f` and the line $y = 0$. Identify the lone zero.

⊛ Question
The fact that only a finite number of points are used in a graph can introduce artifacts. An example can appear when plotting sinusoidal functions. An example is the graph of `f(x) = sin(500*pi*x)` over `[0,1]`.

Make its graph using 250 evenly spaced points, as follows:

```
xs = range(0, 1, length=250)
f(x) = sin(500*pi*x)
plot(xs, f.(xs))
```

1. It should oscillate evenly, but instead doesn't oscillate very much near 0 and 1
2. It oscillates wildly, as the period is $T = 2\pi/(500\pi)$ so there are 250 oscillations.
3. Oddly, it looks exactly like the graph of $f(x) = \sin(2\pi x)$.

The algorithm to plot a function works to avoid aliasing issues. Does the graph generated by `plot(f, 0, 1)` look the same, as the one above?

1. No, but is still looks pretty bad, as fitting 250 periods into a too small number of pixels is a problem.
2. Yes
3. No, the graph shows clearly all 250 periods.

⊛ Question
The *viewport* of a graph is the x-y range of the viewing window. By default, the y-part of the viewport is determined by the range of the function over the specified interval $[a, b]$. For some functions this can produce poor graphs, especially when there is a vertical asymptote. Here are two ways to restrict this with `Plots`:

- Add an argument `ylims = (m, M)` where `m` and `M` are the min and maximum of the range you wish to show.
- trim the y values returned by the function so that if they are bigger than 10, say, they are not shown. For this, a function like the following can be useful:

```
function trimplot(f, a, b, c=20; kwargs...)
   fn = x -> abs(f(x)) < c ? f(x) : NaN
   plot(fn, a, b; kwargs...)
end
```

(This is defined more robustly in the `CalculusWithJulia` package.)

The `trimplot` function would be used just like `plot`:

```
f(x) = 1/x
trimplot(f, -1, 1)
```

Compare the graph of the above with the graph without using `trimplot`. What would you say:

1. There is an error when plotting `trimplot(f)`.
2. They appear the same, using `trimplot` just complicates things.
3. The trimmed graph is not influenced by the vertical asymptote at 0. It is a better graph.

⊛ Question
Make this parametric plot for the specific values of the parameters `k` and `l`. What shape best describes it?

```
R, r, rho = 1, 3/4, 1/4
f(t) = (R-r) * cos(t) + rho * cos((R-r)/r * t)
g(t) = (R-r) * sin(t) - rho * sin((R-r)/r * t)

plot(f, g, 0, max((R-r)/r, r/(R-r))*2pi, aspect_ratio=:equal)
```

1. Four petals, like a flower
2. Four sharp points, like a star
3. A straight line
4. An ellipse

⊛ Question
For these next questions, we use this function:

```
function spirograph(R, r, rho)
  f(t) = (R-r) * cos(t) + rho * cos((R-r)/r * t)
  g(t) = (R-r) * sin(t) - rho * sin((R-r)/r * t)

  plot(f, g, 0, max((R-r)/r, r/(R-r))*2pi, aspect_ratio=:equal)
end
```

```
spirograph (generic function with 1 method)
```

Make this plot for the following specific values of the parameters `R`, `r`, and `rho`. What shape best describes it?

```
R, r, rho = 1, 3/4, 1/4
```

1. Four sharp points, like a star
2. Four petals, like a flower
3. An ellipse
4. A straight line
5. None of the above

Make this plot for the following specific values of the parameters `R`, `r`, and `rho`. What shape best describes it?

```
R, r, rho = 1, 1/2, 1/4
```

1. Four sharp points, like a star

2. Four petals, like a flower
3. An ellipse
4. A straight line
5. None of the above

Make this plot for the specific values of the parameters `R`, `r`, and `rho`. What shape best describes it?

```
R, r, rho = 1, 1/4, 1
```

1. Four sharp points, like a star
2. Four petals, like a flower
3. A circle
4. A straight line
5. None of the above

Make this plot for the specific values of the parameters `R`, `r`, and `rho`. What shape best describes it?

```
R, r, rho = 1, 1/8, 1/4
```

1. Four sharp points, like a star
2. Four petals, like a flower
3. A circle
4. A straight line
5. None of the above