1 Vector-valued functions, $f : R \to R^n$

We discuss functions of a single variable that return a vector in R^n. There are many parallels to univariate functions (when $n = 1$) and differences.

Before beginning, we load our `CalculusWithJulia` package, which will provide a few functions used in the following.

```
using CalculusWithJulia
using Plots
```

1.1 Definition

A function $\vec{f} : R \to R^n$, $n > 1$ is called a vector valued function. Some examples:

$$\vec{f}(t) = \langle \sin(t), 2\cos(t) \rangle, \quad \vec{g}(t) = \langle \sin(t), \cos(t), t \rangle, \quad \vec{h}(t) = \langle 2, 3 \rangle + t \cdot \langle 1, 2 \rangle.$$

The components themselves are also functions of t, in this case univariate functions. Depending on the context, it can be useful to view vector-valued functions as a function that returns a vector, or a vector of the component functions.

The above example functions have n equal 2, 3, and 2 respectively. We will see that many concepts of calculus for univariate functions ($n = 1$) have direct counterparts.

1.2 Representation in Julia

In `Julia`, the representation of a vector-valued function is straightforward: we define a function of a single variable that returns a vector. For example, the three functions above would be represented by:

```
f(t) = [sin(t), 2*cos(t)]
g(t) = [sin(t), cos(t), t]
h(t) = [2, 3] + t * [1, 2]
```

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

For a given `t`, these evaluate to a vector. For example:

```
h(2)
```

```
2-element Array{Int64,1}:
 4
 7
```

We can create a vector of functions, e.g., `F = [cos, sin, identify]`, but calling this object, as in `F(t)`, would require some work, such as `t = 1; [f(t) for f in F]`.

1.3 Space curves

A vector-valued function is typically visualized as a curve. That is, for some range, $a \leq t \leq b$ the set of points $\{\vec{f}(t) : a \leq t \leq b\}$ are plotted. If, say in $n = 2$, we have $x(t)$ and $y(t)$ as the component functions, then the graph would also be the parametric plot of x and y. The term *planar* curve is common for the $n = 2$ case and *space* curve for the $n \geq 3$ case.

This plot represents the vectors with their tails at the origin.

There is a convention for plotting the component functions to yield a parametric plot within the `Plots` package (e.g., `plot(x, y, a, b)`). This can be used to make polar plots, where `x` is `t -> r(t)*cos(t)` and `y` is `t -> r(t)*sin(t)`.

However, we will use a different approach, as the component functions are not naturally produced from the vector-valued function.

In `Plots`, the command `plot(xs, ys)`, where, say, `xs=[x1, x2, ..., xn]` and `ys=[y1, y2, ..., yn]`, will make a connect-the-dot plot between corresponding pairs of points. This can be used as an alternative to plotting a function through `plot(f, a, b)`: first make a set of x values, say `xs=range(a, b, length=100)`; then the corresponding y values, say `ys = f.(xs)`; and then plotting through `plot(xs, ys)`. (If using `Julia 1.0` the second argument to `range` should be named, as in `stop=b`.)

Similarly, were a third vector, `zs`, for z components used, `plot(xs, ys, zs)` will make a 3-dimensional connect the dot plot

However, our representation of vector-valued functions naturally generates a vector of points: `[[x1,y1], [x2, y2], ..., [xn, yn]]`, as this comes from broadcasting `f` over some time values. That is, for a collection of time values, `ts`, typically generated, as above, by `range`, the command `f.(ts)` will produce the vector of points.

To get the `xs` and `ys` from this, is conceptually easy: just iterate over all the points and extract the corresponding component. For example, to get `xs` we would have a command like `[p[1] for p in f.(ts)]`. Similarly, the `ys` would use `p[2]` in place of `p[1]`. The following function, from the `CalculusWithJulia` package and employed previously, does this for us, returning the vectors in a tuple:

```
unzip(vs) = Tuple(eltype(first(vs))[xyz[j] for xyz in vs] for j in eachindex(first(vs)))
```

The name comes from how the `zip` function in base `Julia` takes two vectors and returns a vector of the values paired off. This is the reverse.

It was noted in vectors that the `unzip` function turns a vector of points (or vectors) into a tuple of vectors collecting the x values, the y values, and, if present, the z values:

```
[[x1, y1, z1],          (\ensuremath{\lceil}x1\ensuremath{\rceil},  \ensuremath{\lceil}y
 [x2, y2, z2],           |x2|, |y2|, |z2|,
 [x3, y3, z3],   -->     |x3|, |y3|, |z3|,
    \ensuremath{\vdots}                          \ensuremath{\vdots}
 [xn, yn, zn]]           \ensuremath{\lfloor}xn\ensuremath{\rfloor},  \ensuremath{\lfloo
```

To turn a tuple of vectors into separate arguments for a function, splatting (the `...`) is used.

Finally, with these definitions, we can visualize the three functions we have defined.

Here we show the plot of `f` over the values between 0 and 2π and also add a vector anchored at the origin defined by `f(1)`.

```
ts = range(0, 2pi, length=200)
plot(unzip(f.(ts))...)
arrow!([0, 0], f(1))
```

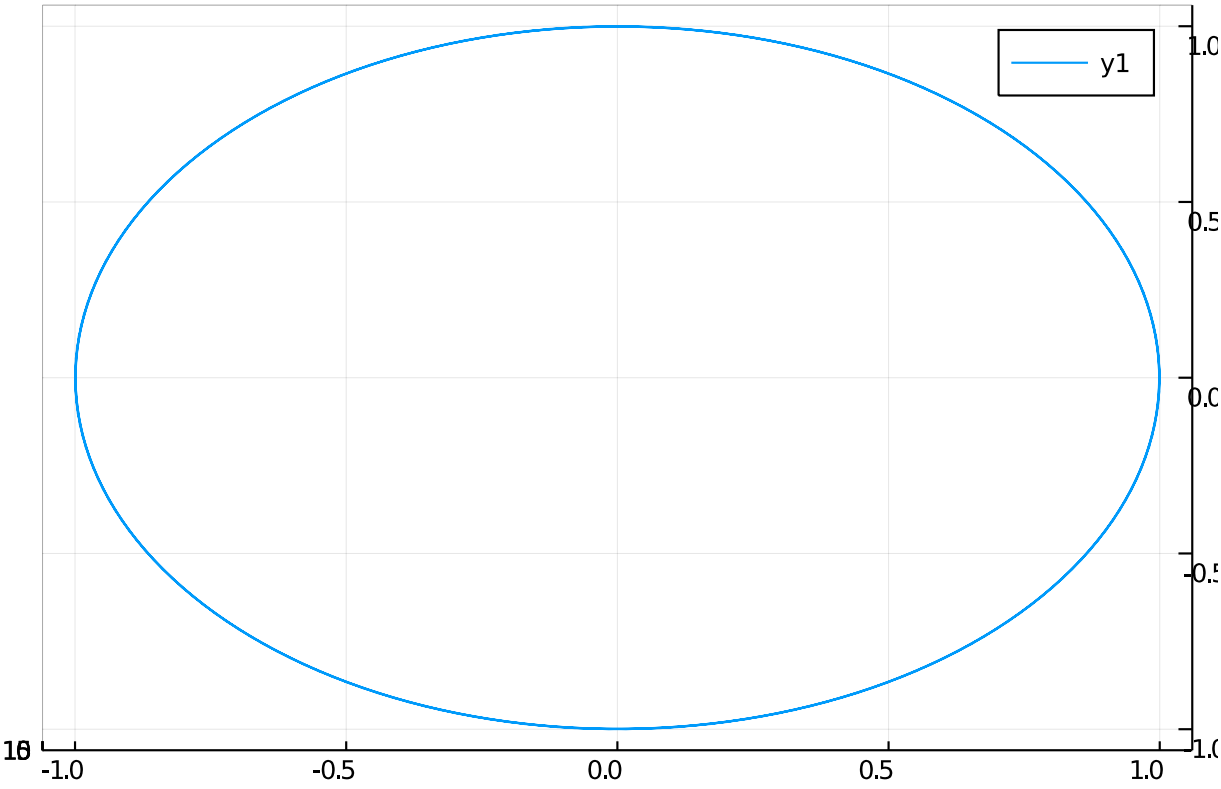
The trace of the plot is an ellipse. If we describe the components as $\vec{f}(t) = \langle x(t), y(t) \rangle$, then we have $x(t)^2 + y(t)^2/4 = 1$. That is, for any value of t, the resulting point satisfies the equation $x^2 + y^2/4 = 1$ for an ellipse.

The plot of g needs 3-dimensions to render. For most plotting backends, the following should work with no differences, save the additional vector is anchored in 3 dimensions now:

```
ts = range(0, 6pi, length=200)
plot(unzip(g.(ts))...)
arrow!([0, 0, 0], g(2pi))
```

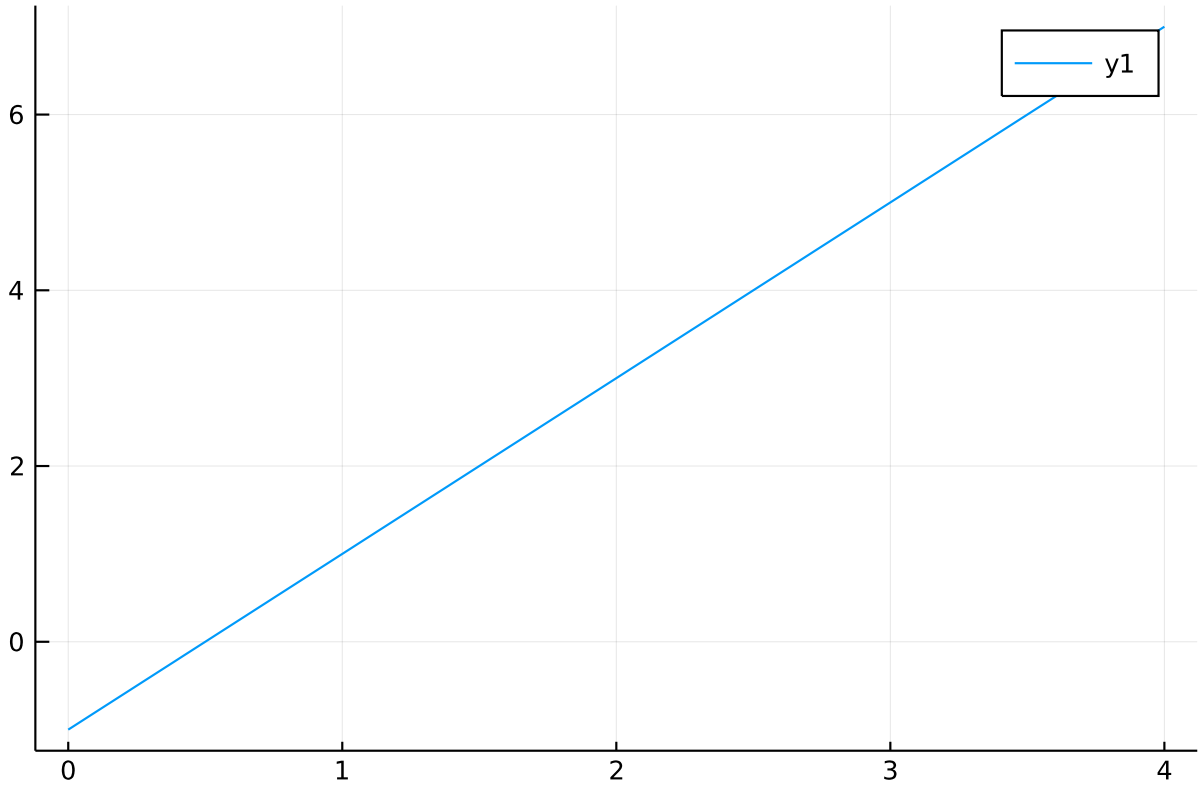
Here the graph is a helix; three turns are plotted. If we write $g(t) = \langle x(t), y(t), z(t) \rangle$, as the x and y values trace out a circle, the z value increases. When the graph is viewed from above, as below, we see only x and y components, and the view is circular.

```
plot(unzip(g.(ts))..., camera=(0, 90))
```



The graph of h shows that this function parameterizes a line in space. The line segment for $-2 \leq t \leq 2$ is shown below:

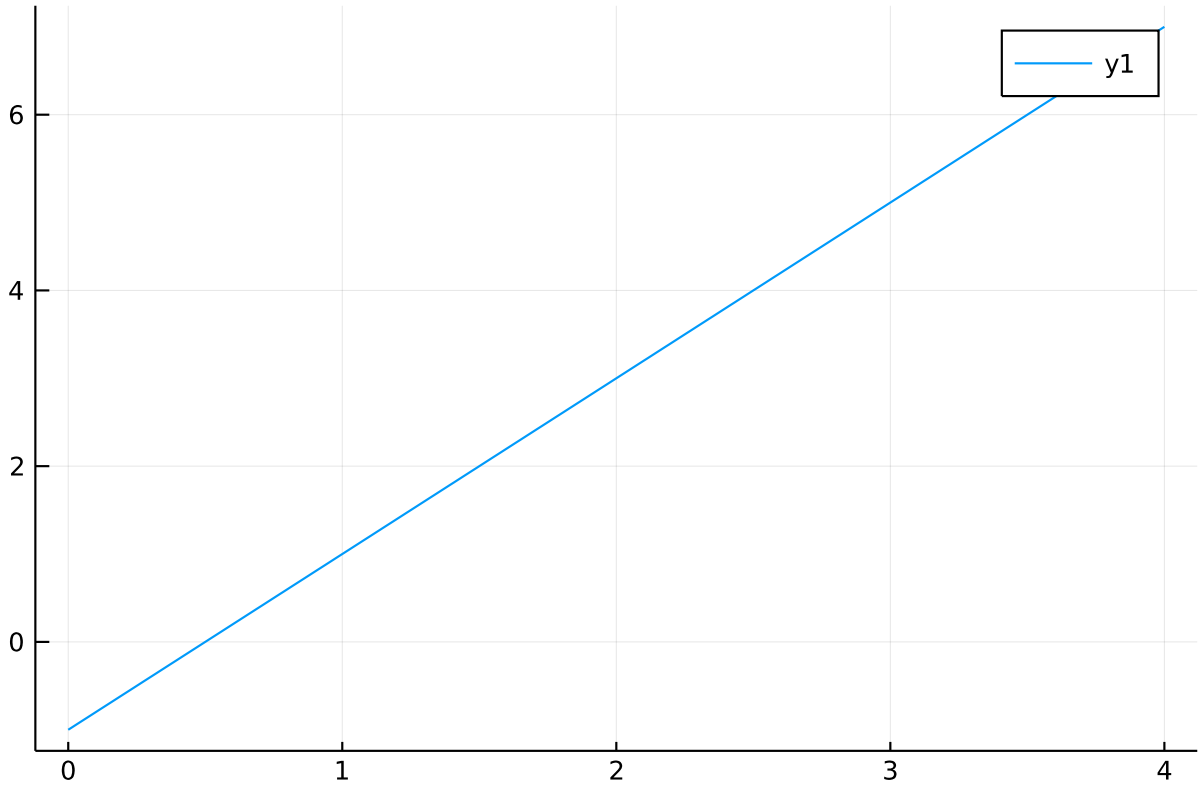
```
ts = range(-2, 2, length=200)
plot(unzip(h.(ts))...)
```



1.3.1 The `plot_parametric_curve` function

While the `unzip` function is easy to understand as a function that reshapes data from one format into one that `plot` can use, it's usage is a bit cumbersome. The `CalculusWithJulia` package provides a function `plot_parametric_curve` which hides the use of `unzip` and the splatting within a function definition. It expects a vector-valued function and a range of t values specified by two values. For example, the last plot can be produced with

```
plot_parametric_curve(h, -2, 2)
```



Defining plotting functions in `Julia` for `Plots` is facilitated by the `RecipesBase` package. There are two common choices: creating a new function for plotting, as is done with `plot_parametric_curve` and `plot_polar`; or creating a new type so that `plot` can dispatch to an appropriate plotting method. The latter would also be a reasonable choice, but wasn't taken here.

Example Familiarity with equations for lines, circles, and ellipses is important, as these fundamental geometric shapes are often building blocks in the description of other more complicated things.

The point-slope equation of a line, $y = y_0 + m \cdot (x - x_0)$ finds an analog. The slope, m, is replaced with a vector $\vec{v}$ and the point, (x_0, y_0) is replaced with a vector $\vec{p}$ identified with a point in the plane. A parameterization would then be $f(t) = \vec{p} + (t - t_0)\vec{v}$. From this, we have $f(t_0) = \vec{p}$.

The unit circle is instrumental in introducing the trigonometric functions though the identification of and angle t with a point on the unit circle (x, y) through $y = \sin(t)$ and $x = \cos(t)$. With this identification certain properties of the trigonometric functions are immediately seen, such as the period of sin and cos being 2π, or the angles for which sin and cos are positive or even increasing. Further, this gives a natural parameterization for a vector-valued function whose plot yields the unit circle, namely $\vec{f}(t) = \langle \cos(t), \sin(t) \rangle$. This parameterization starts (at $t = 0$) at the point $(1, 0)$. More generally, we might have additional parameters $\vec{f}(t) = \vec{p} + R \cdot \langle \cos(\omega(t - t_0)), \sin(\omega(t - t_0)) \rangle$ to change the origin, $\vec{p}$; the radius, R; the starting angle, t_0; and the rotational frequency, ω.

An ellipse has a slightly more general equation than a circle and in simplest forms may satisfy the equation $x^2/a^2 + y^2/b^2 = 1$, where *when* $a = b$ a circle is being described. A vector-valued function of the form $\vec{f}(t) = \langle a \cdot \cos(t), b \cdot \sin(t) \rangle$ will trace out an ellipse.

The above description of an ellipse is useful, but it can also be useful to re-express the ellipse so that one of the foci is at the origin. With this, the ellipse can be given in *polar* coordinates through a description of the radius:

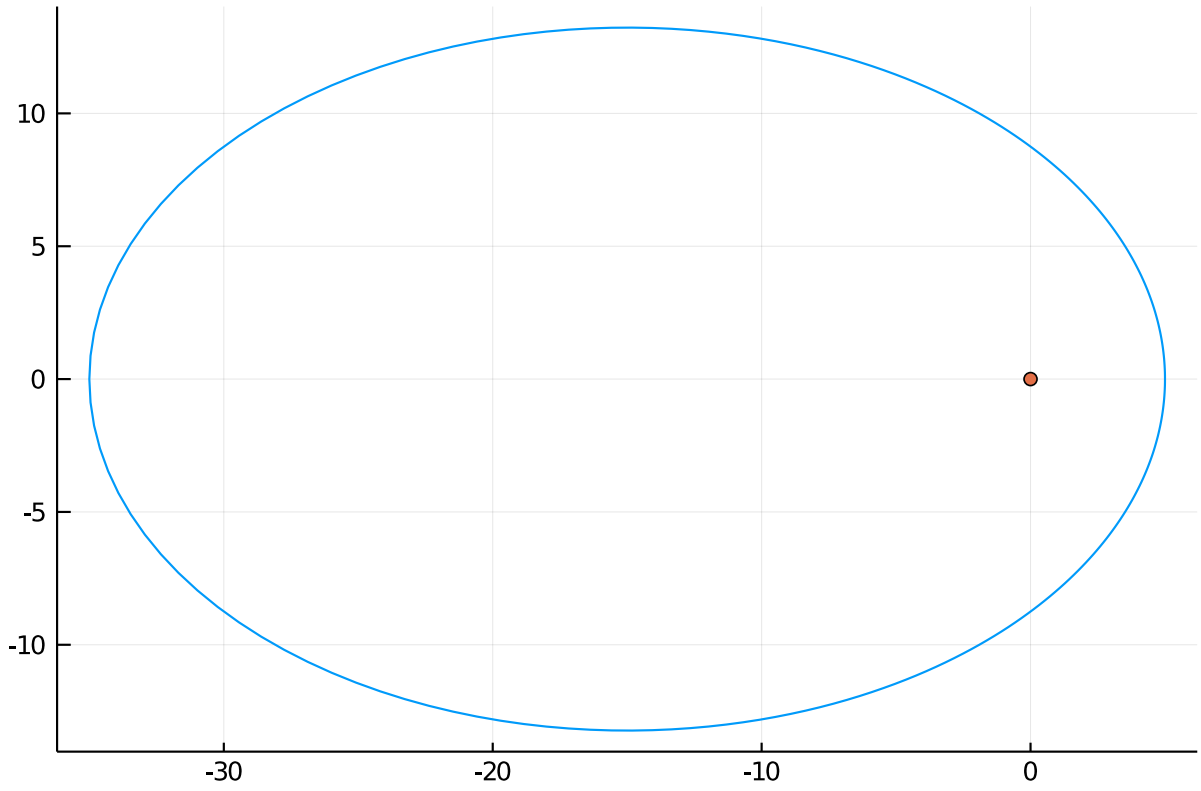
$$r(\theta) = \frac{a(1-e^2)}{1 + e\cos(\theta)}.$$

Here, a is the semi-major axis ($a > b$); e is the *eccentricity* given by $b = a\sqrt{1-e^2}$; and θ a polar angle.

Using the conversion to Cartesian equations, we have $\vec{x}(\theta) = \langle r(\theta)\cos(\theta), r(\theta)\sin(\theta)\rangle$.

For example:

```
a, ecc = 20, 3/4
f(t) = a*(1-ecc^2)/(1 + ecc*cos(t)) * [cos(t), sin(t)]
plot_parametric_curve(f, 0, 2pi, legend=false)
scatter!([0],[0], markersize=4)
```



Example The Spirograph is "... a geometric drawing toy that produces mathematical roulette curves of the variety technically known as hypotrochoids and epitrochoids. It was developed by British engineer Denys Fisher and first sold in 1965." These can be used to make interesting geometrical curves.

Following Wikipedia: Consider a fixed outer circle C_o of radius R centered at the origin. A smaller inner circle C_i of radius $r < R$ rolling inside C_o and is continuously tangent to it. C_i will be assumed never to slip on C_o (in a real Spirograph, teeth on both circles prevent such

slippage). Now assume that a point A lying somewhere inside $C_i$is located a distance $\rho < r$ from C_i's center.

The center of the inner circle will move in a circular manner with radius $R-r$. The fixed point on the inner circle will rotate about this center. The accumulated angle may be described by the angle the point of contact of the inner circle with the outer circle. Call this angle t.

Suppose the outer circle is centered at the origin and the inner circle starts ($t = 0$) with center $(R - r, 0)$ and rotates around counterclockwise. Then if the point of contact makes angle t, the arc length along the outer circle is Rt. The inner circle will have moved a distance rt' in the opposite direction, so $Rt = -rt'$ and solving the angle will be $t' = -(R/r)t$.

If the initial position of the fixed point is at $(\rho, 0)$ relative to the origin, then the following function will describe the motion:

$$\vec{s}(t) = (R - r) \cdot \langle \cos(t), \sin(t) \rangle + \rho \cdot \langle \cos(-\frac{R}{r}t), \sin(-\frac{R}{r}t) \rangle.$$

To visualize this we first define a helper function to draw a circle at point P with radius R:

```
circle!(P, R; kwargs...) = plot_parametric_curve!(t -> P + R*[cos(t), sin(t)], 0,
2pi;kwargs...)
```

```
circle! (generic function with 1 method)
```

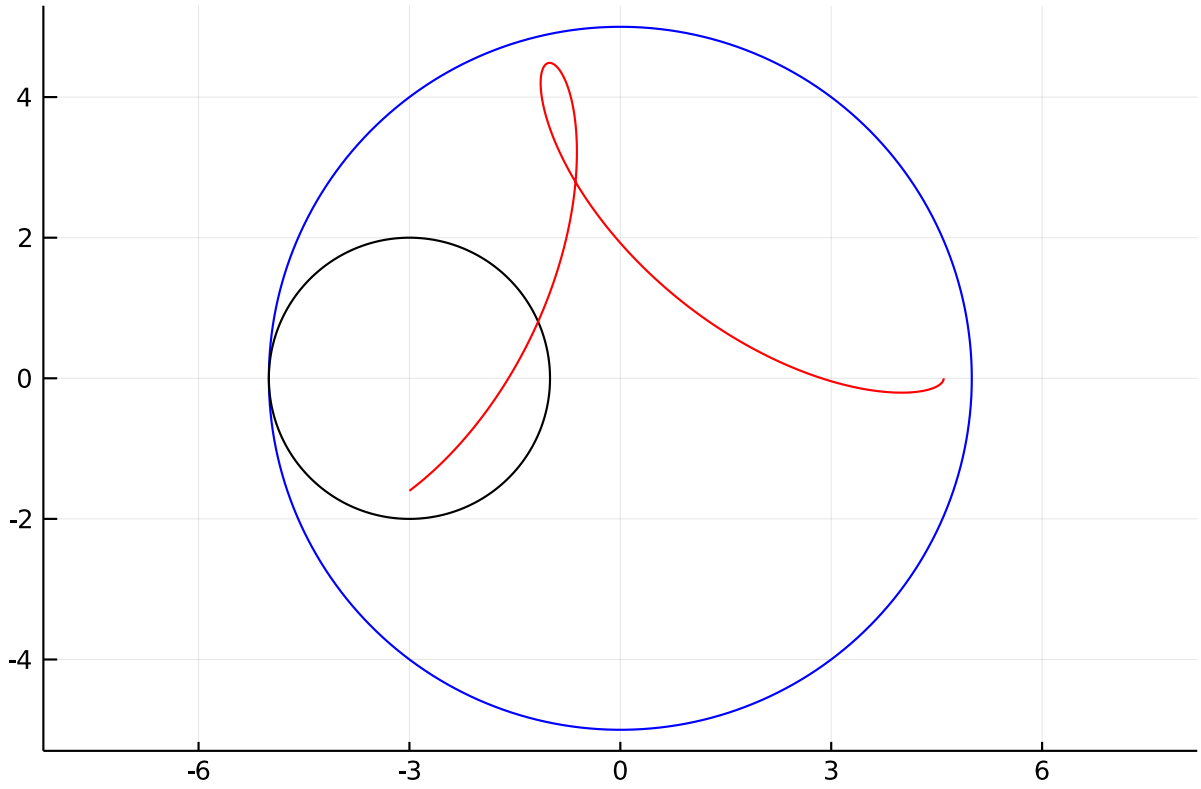
Then we have this function to visualize the spirograph for different t values:

```
function spiro(t; r=2, R=5, rho=0.8*r)

    cent(t) = (R-r) * [cos(t), sin(t)]

    p = plot(legend=false, aspect_ratio=:equal)
    circle!([0,0], R, color=:blue)
    circle!(cent(t), r, color=:black)

    tp(t) = -R/r * t

    s(t) = cent(t) + rho * [cos(tp(t)), sin(tp(t))]
    plot_parametric_curve!(s, 0, t, n=1000, color=:red)

    p
end
```

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

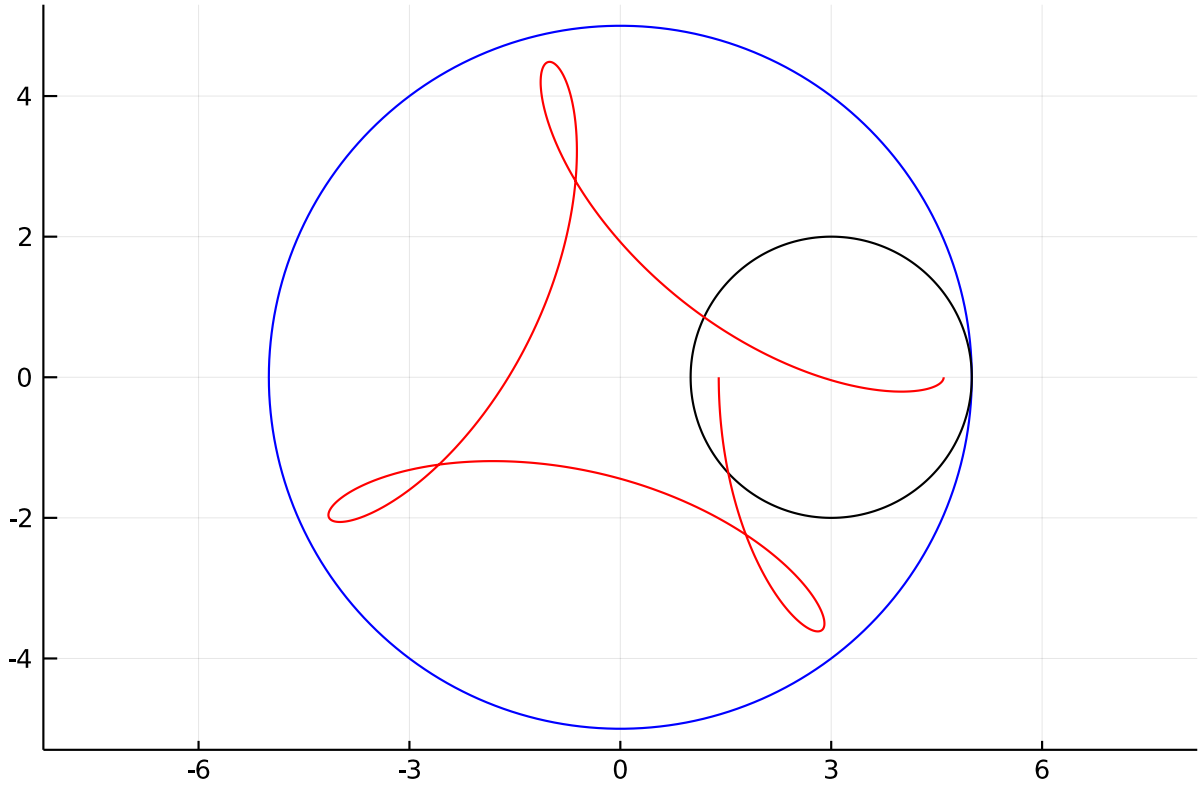
And we can see the trace for $t = \pi$:

```
spiro(pi)
```



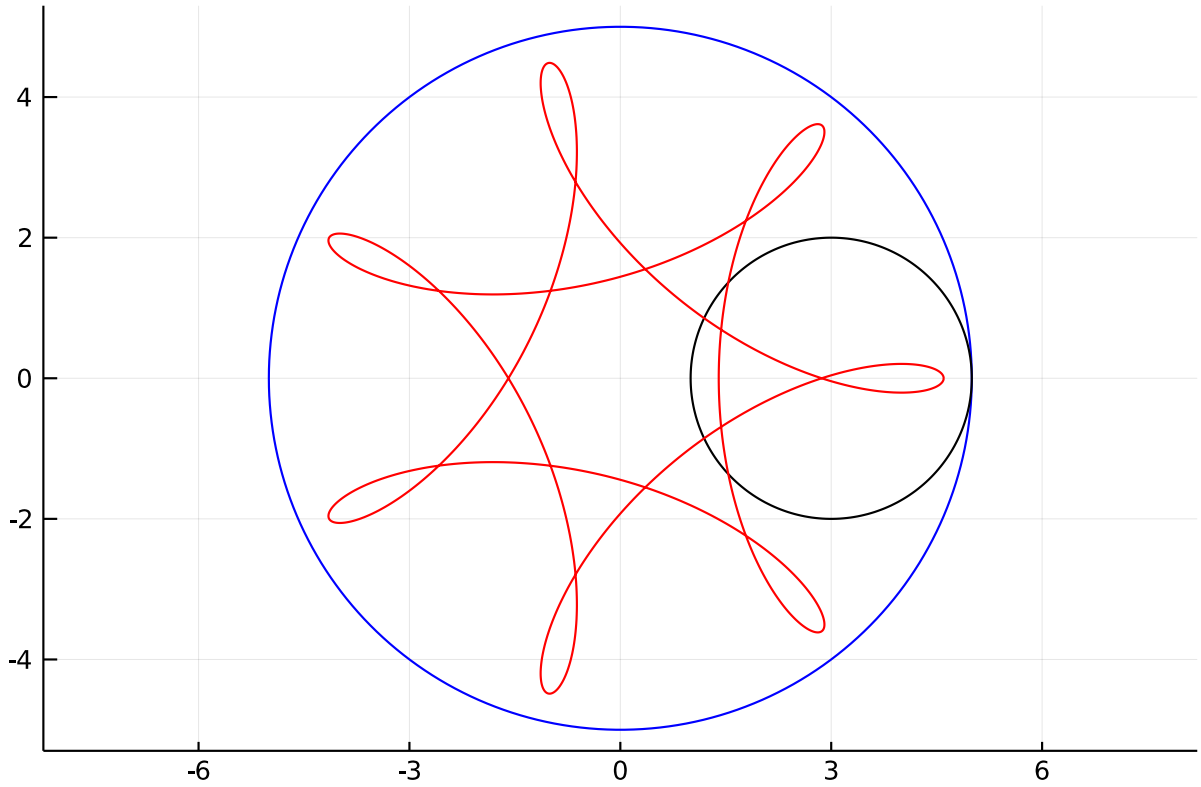
The point of contact is at $(-R, 0)$, as expected. Carrying this forward to a full circle's worth is done through:

```
spiro(2pi)
```



The curve does not match up at the start. For that, a second time around the outer circle is needed:

```
spiro(4pi)
```



Whether the curve will have a period or not is decided by the ratio of R/r being rational or irrational.

Example Ivars Peterson described the carnival ride "tilt-a-whirl" as a chaotic system, whose equations of motion are presented in American Journal of Physics by Kautz and Huggard. The tilt-a-whirl has a platform that moves in a circle that also moves up and down. To describe the motion of a point on the platform assuming it has radius R and period T and rises twice in that period could be done with the function:

$$\vec{u}(t) = \langle R\sin(2\pi t/T), R\cos(2\pi t/T), h + h\cdot\sin(2\pi t/T)\rangle.$$

A passenger sits on a circular platform with radius r attached at some point on the larger platform. The dynamics of the person on the tilt-a-whirl depend on physics, but for simplicity, let's assume the platform moves at a constant rate with period S and has no relative z component. The motion of the platform in relation to the point it is attached would be modeled by:

$$\vec{v}(t) = \langle r\sin(2\pi t/S), r\sin(2\pi t/S), 0\rangle.$$

And the motion relative to the origin would be the vector sum, or superposition: