```julia
begin
    import Pkg
    Pkg.activate(mktempdir())
    Pkg.add(["Plots", "PlutoUI"])

    using Plots
    using PlutoUI
end
```

Our linear ODE:

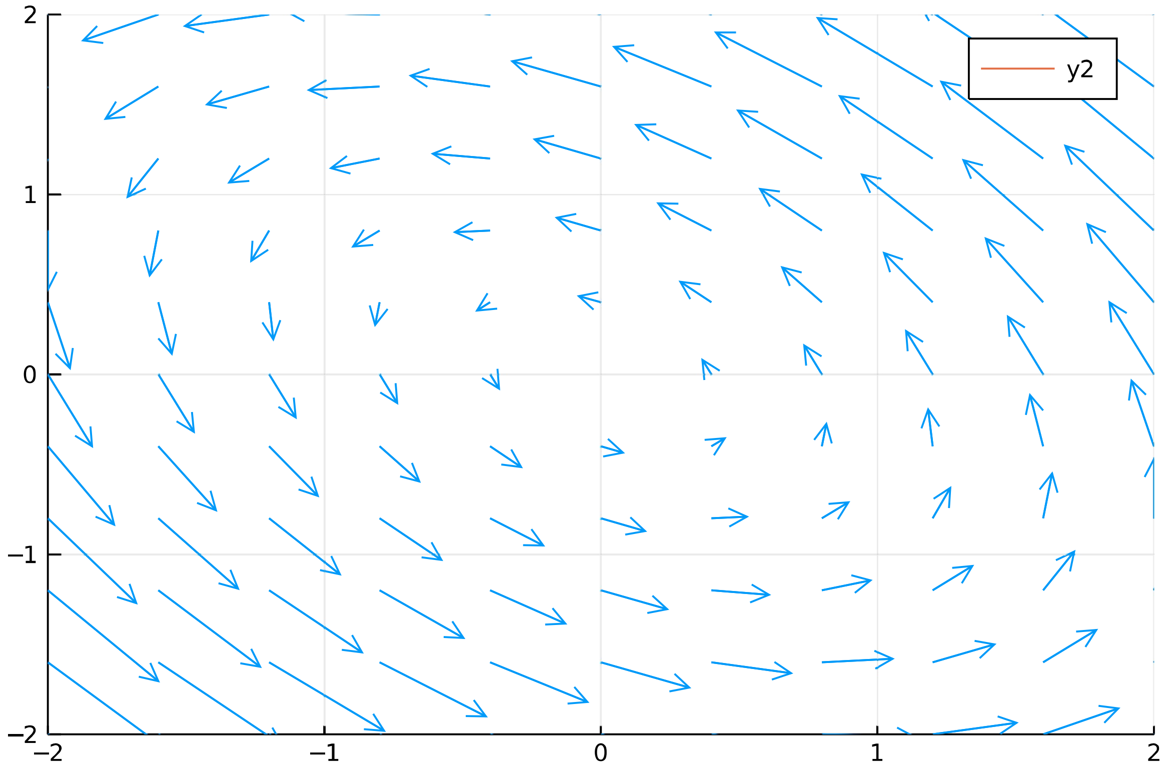
$$\dot{x}(t) = Ax(t)$$

$$x(0) = x_0$$

```
A = 2×2 Array{Float64,2}:
 -0.4  -1.0
  1.0   0.44
```

```julia
A = [-.4  -1
     1    .44]
```

x_0




```julia
let
    plt = plot()
```

```
    vectorfield!(plt)
    plot!(plt, first.(sol), last.(sol), xlim=(-2,2), ylim=(-2,2))
end
```

```
Enter cell code...
```

```
T = 250-element LinRange{Float64}:
     0.0,0.12048192771084336,0.24096385542168672,…,29.879518072289155,30.0
```

```
T = LinRange(0.0, 30.0, 250)
```

```
ΔT = 0.12048192771084337
```

```
ΔT = step(T)
```

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

```
f(t,x) = A*x
```

Explicit Euler:

```
sol = ▸Any[Float64[0.0, 0.0], Float64[0.0, 0.0], Float64[0.0, 0.0], Float64[0.0,
```

```
sol = accumulate(T; init=x0) do x_prev, t
    x_prev + ΔT * f(t,x_prev)
end
```

```
x0 = ▸Float64[0.0, 0.0]
```

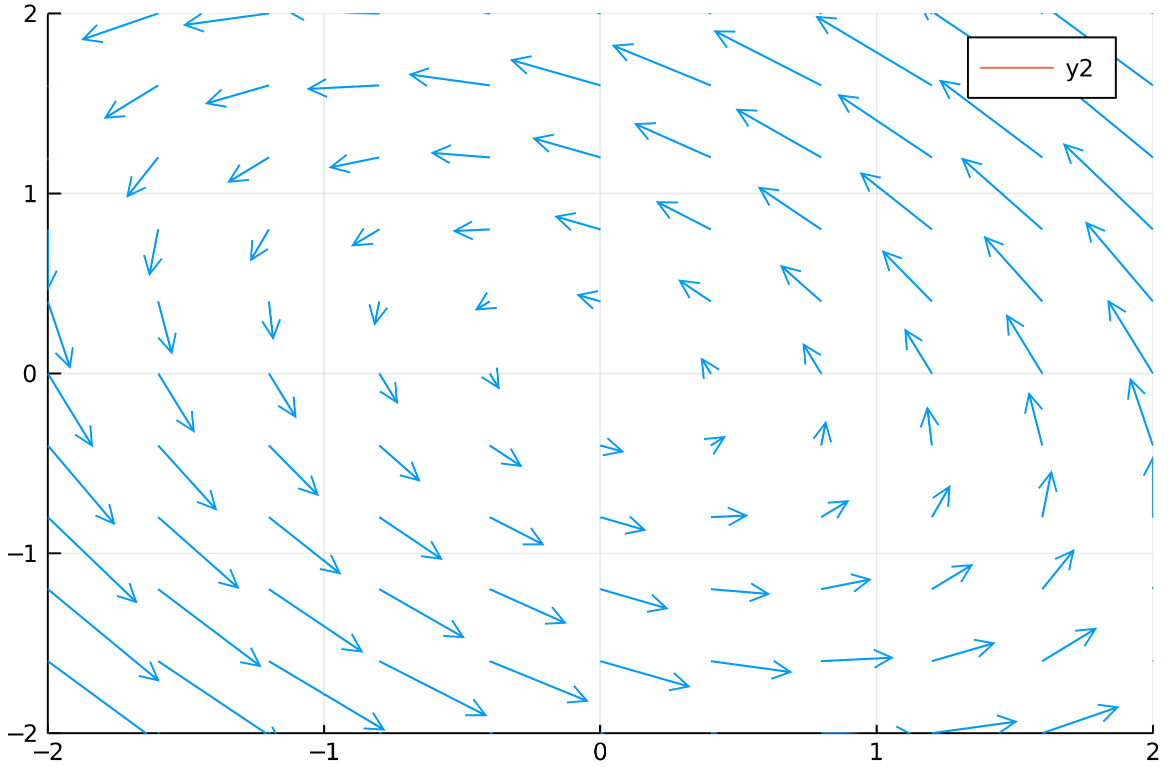
```
x0 = [x0_x, x0_y]
```

We can make the interaction fast by putting everything into a function. The bottleneck for fast interactivity is Julia code compiling, and all cells that depend on a cell will re-run, and hence recompile. In the interaction above, this is the code defining *sol*, and the code creating the plot. Not a lot of code, but enough for the precompilation to take more than 30fps.

Below, the function *definition* does not recompile, only the function *call* does. The call is very simple, so that compilation is fast.

```
@bind x0_x Slider(-2.0:.01:2.0)
```

```
@bind x0_y Slider(-2.0:.01:2.0)
```




```
fast(x0_xf, x0_yf)
```

fast (generic function with 1 method)

```
function fast(x0_x, x0_y)
    x0 = [x0_x, x0_y]
    sol = accumulate(T; init=x0) do x_prev, t
        x_prev + ΔT * f(t,x_prev)
    end
    plt = plot()

    vectorfield!(plt)
    plot!(plt, first.(sol), last.(sol), xlim=(-2,2), ylim=(-2,2))
end
```

vectorfield! (generic function with 1 method)

```
function vectorfield!(plt)
    xs = [[x, y] for x in -2.0:.4:2.0 for y in -2.0:.4:2.0]
    f(x) = .2 * A*x
    quiver!(plt, first.(xs), last.(xs), quiver=(first.(f.(xs)), last.(f.(xs))))
end
```