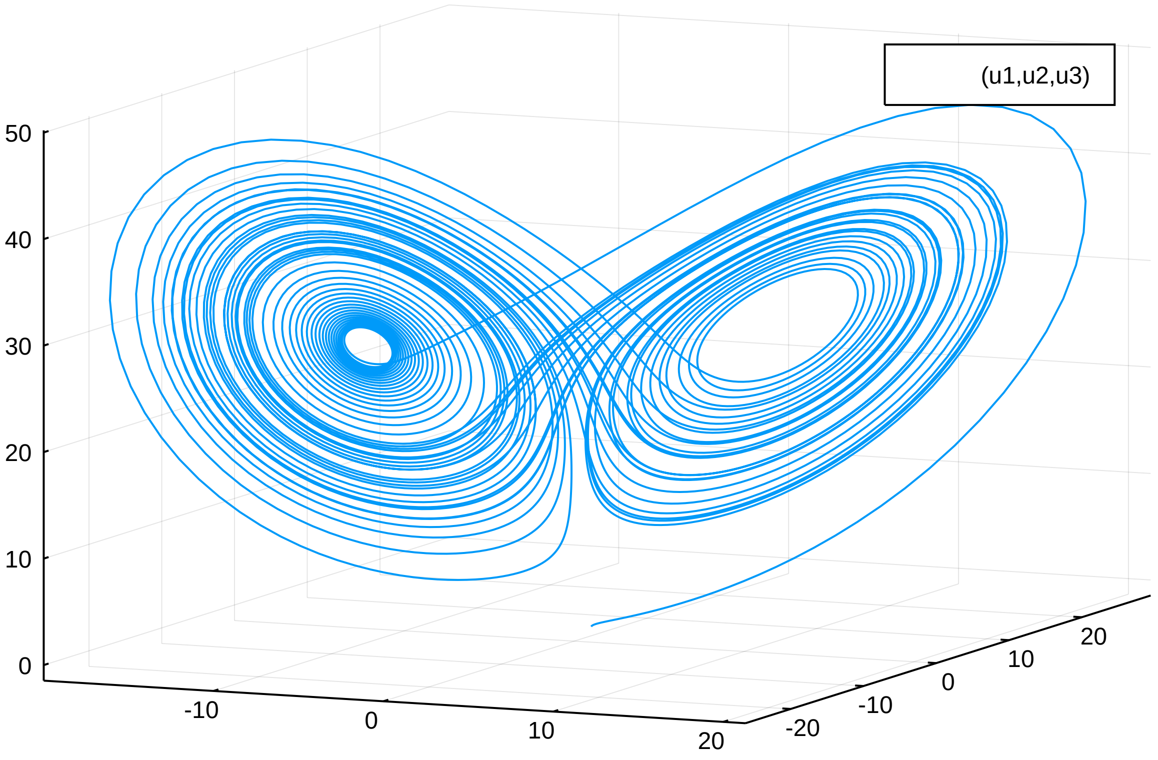
```
begin
    using DifferentialEquations
    using Plots
end
```

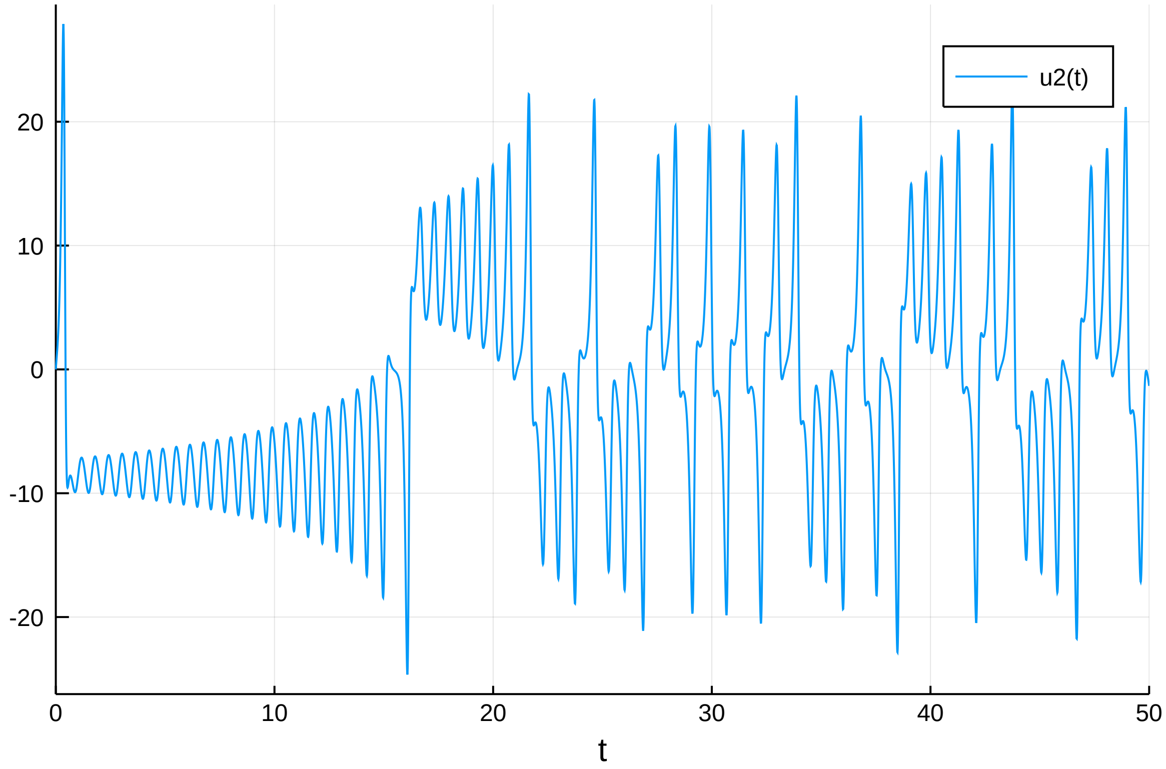
```
begin
    using Pkg
    Pkg.add("WGLMakie")
    Pkg.add("AbstractPlotting")
end
```

Chaos Theory

```
begin
    function parameterized_lorenz!(du,u,p,t)
     du[1] = p[1]*(u[2]-u[1])
     du[2] = u[1]*(p[2]-u[3]) - u[2]
     du[3] = u[1]*u[2] - p[3]*u[3]
    end

    u0 = [1.0,0.0,0.0]
    tspan = (0.0,50.0)
    p = [9.8,28.2,8/3]
    prob = ODEProblem(parameterized_lorenz!,u0,tspan,p)
    sol = solve(prob)
    nothing
end
```



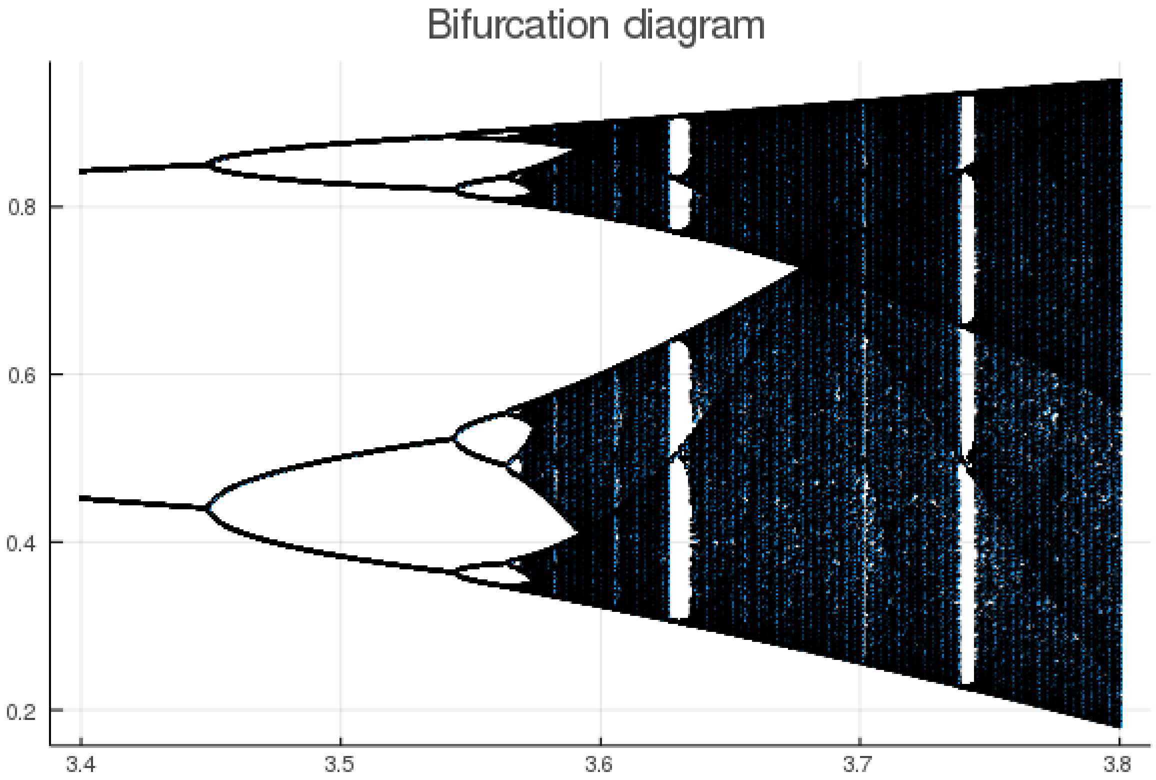
```
plot(sol,vars=(1,2,3))
```



```
plot(sol,vars=(0,2))
```

```
logistic_map (generic function with 3 methods)
```

```
plot_bfdiagram (generic function with 4 methods)
```



```
plot_bfdiagram(3.4, 0.001, 3.8, n = 1000)
```

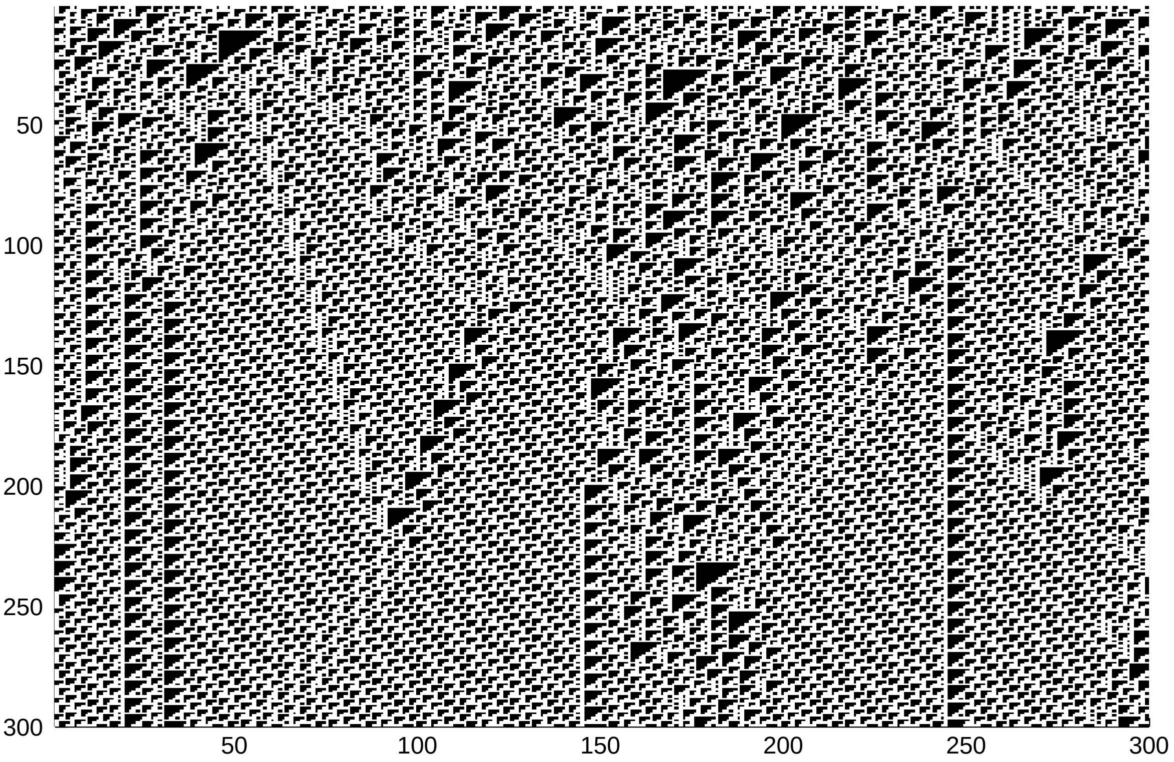
```
ca_run (generic function with 1 method)
```

```
begin
    rule2poss(rule) = [rule & (1 << (i - 1)) != 0 for i in 1:8]
```

```julia

    function transform(bset, ruleposs)
        newbset = map(x->ruleposs[x],
            [bset[i - 1] * 4 + bset[i] * 2 + bset[i + 1] + 1
             for i in 2:length(bset)-1])
        vcat(newbset[end], newbset, newbset[1])
    end

    function ca_run(startset, steps, rul)
        res = Array{Bool}(undef, length(startset), steps)
        bset = vcat(startset[end], startset, startset[1])
        rp = rule2poss(rul)
        for i in 1:steps
            res[:,i] .= bset[2:end-1]
            bset = transform(bset, rp)
        end
        res
    end
end
```

```julia
begin
    startset = rand(Bool, 300)  # fill(false, 500)
    # startset[250] = true
    res = ca_run(startset, 300, 110)
    nothing
end
```

```julia
using Colors
```



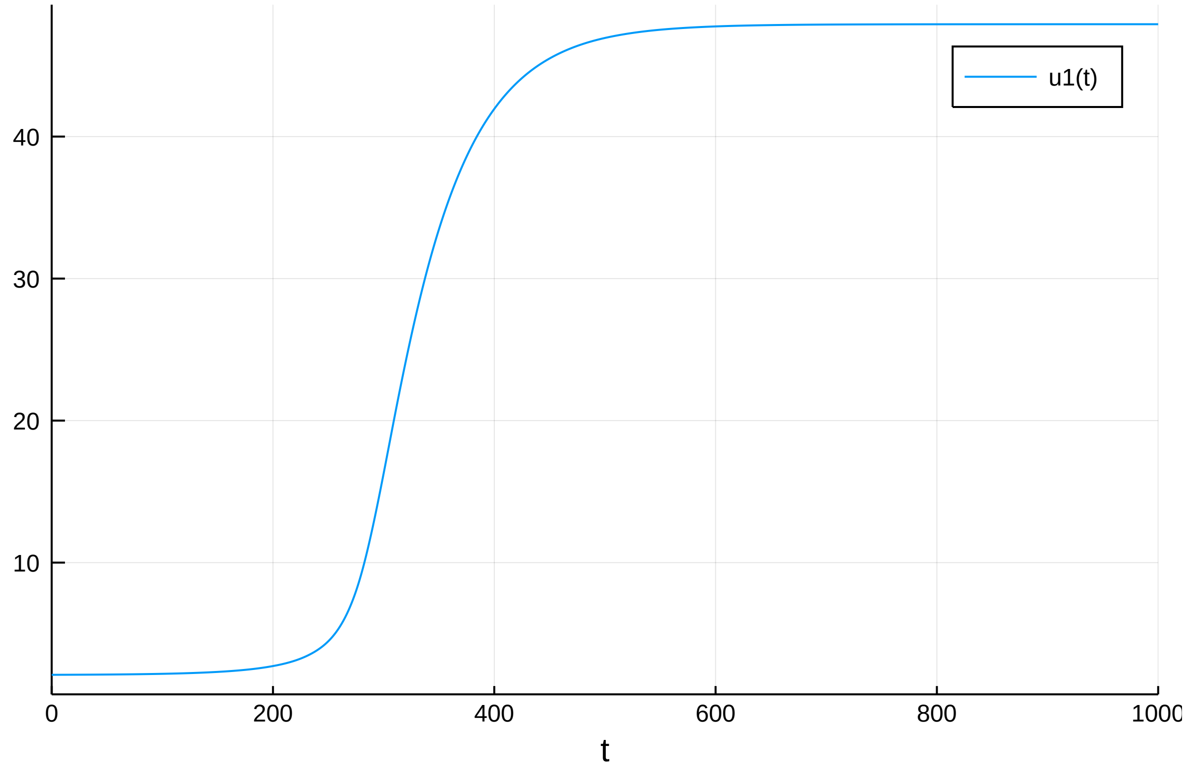
```julia
plot(Colors.Gray.(res |> transpose))
```

Control Theory

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

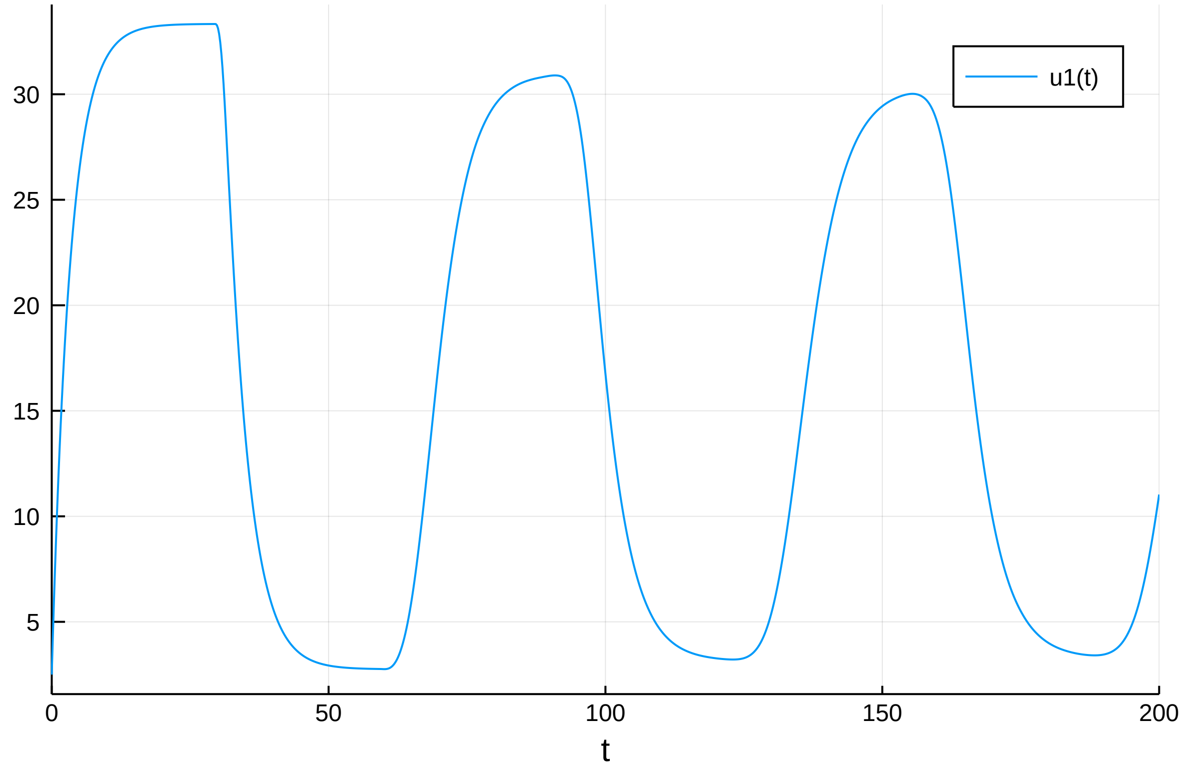
```julia
begin
    function self_feedback!(du,u,p,t)
```

```
     du[1] = hill_act(u[1], p[1], p[2], p[3]) - p[4]*u[1]
    end

    u0_1 = [2.1] # 2.0
    tspan_1 = (0.0, 1000.0)
    p_1 = [2., 1., 10., 0.02]
    prob_1 = ODEProblem(self_feedback!,u0_1,tspan_1,p_1)
    sol_1 = solve(prob_1)
    nothing
end
```



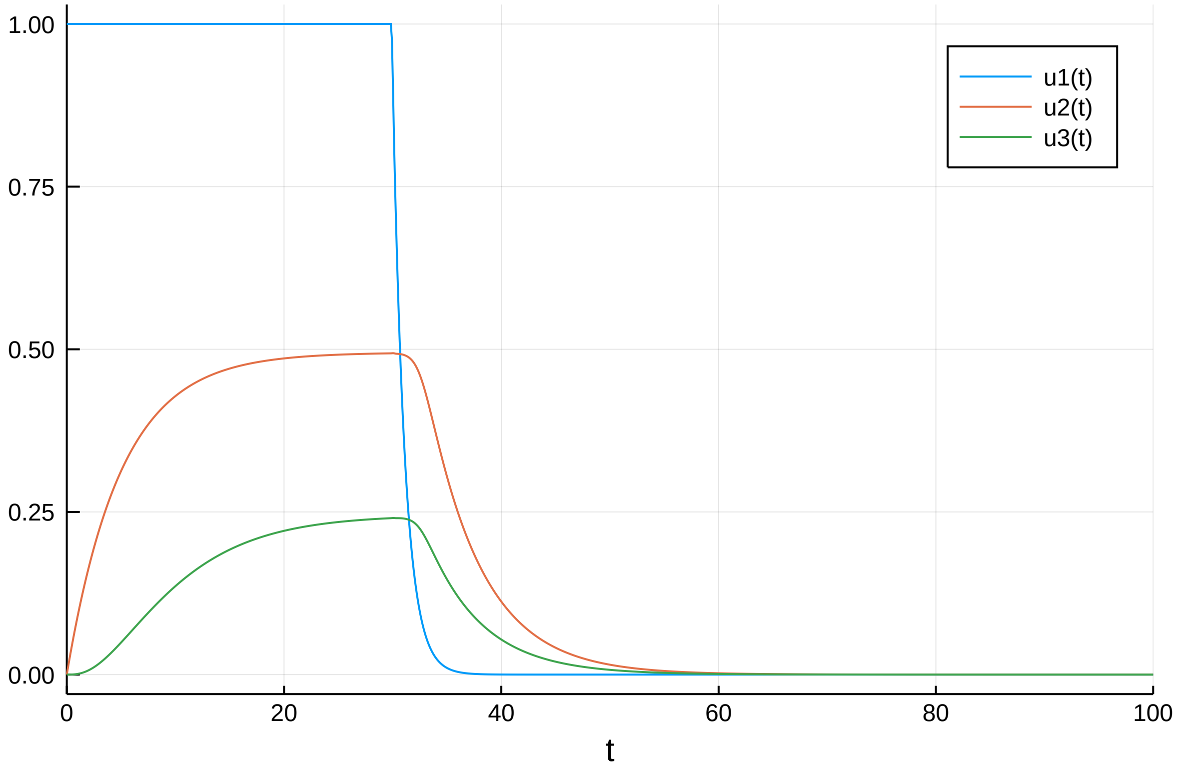
```
plot(sol_1)
```

```
begin
    function negative_feedback!(du,u,h,p,t)
        u_past = h(p, t-p[5])[1]
        du[1] = hill_inh(u_past, p[1], p[2], p[3]) - p[4]*u[1]
    end

    u0_2 = [2.5]
    tspan_2 = (0.0, 200.0)
    p_2 = [2., 10., 10., 0.3, 30]
    h_2(p, t) = zeros(1)
    prob_2 = DDEProblem(negative_feedback!,u0_2,h_2,tspan_2,p_2)
    sol_2 = solve(prob_2)
    nothing
end
```

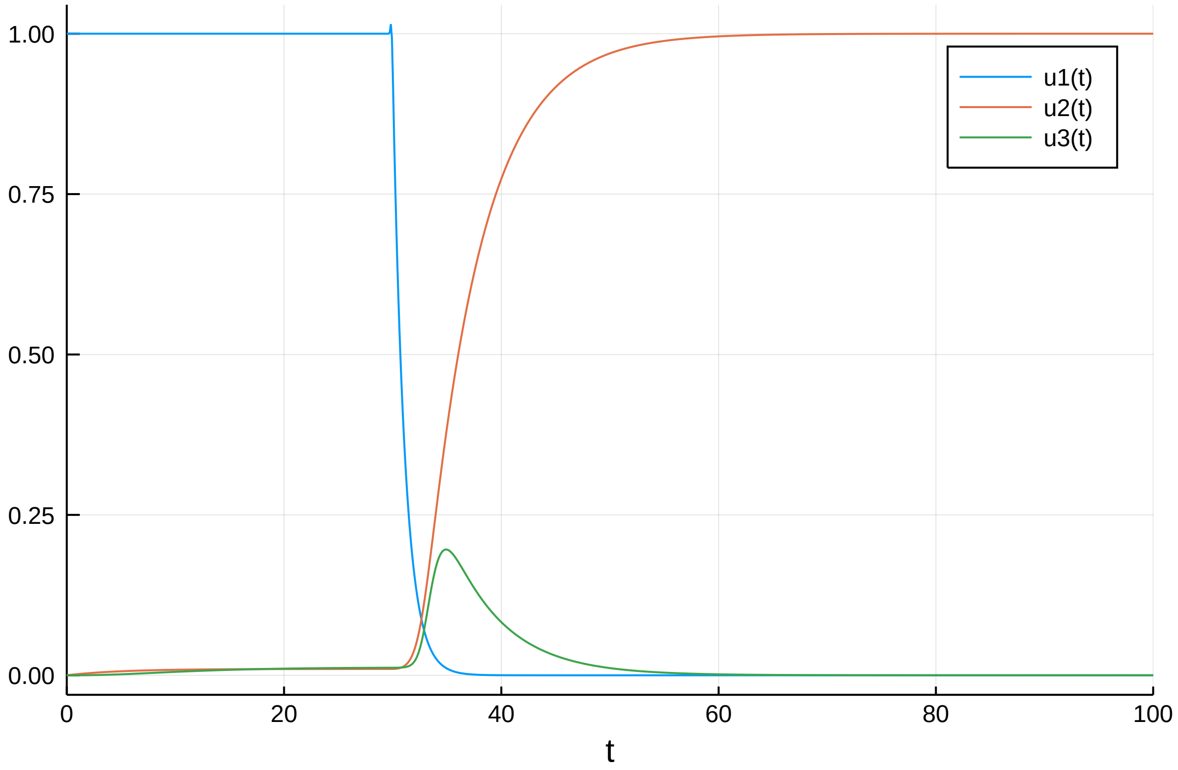


```
plot(sol_2)
```

```
begin
    act_inh(t) = t ? hill_act : hill_inh

    function feed_forward_loop!(du,u,p,t)
        f1 = act_inh(p[1][1])
        f2 = act_inh(p[1][2])
        f3 = act_inh(p[1][3])

        if t < p[2] && u[1] > 0
            du[1] = 0
        else
            du[1] = -0.9*u[1]
        end

        y = p[1][4] ? u[2] : 0.0

        du[2] = p[3] + f1(u[1], p[4], p[5], p[6]) - p[7]*y
        du[3] = p[8] + f2(u[1], p[9], p[10], p[11]) * f3(y, p[12], p[13], p[14]) -
p[15]*u[3]
    end


    nothing
end
```



```
begin
    u0_3 = [1.0, 0, 0]
    tspan_3 = (0.0, 100.0)
    p_3 = [(true, true, true, true), 30,
            0., 2., 0.1, 0.1, 0.2,
            0., 2., 0.1, 0.1,  2., 1., 0.5, 0.2]
    prob_3 = ODEProblem(feed_forward_loop!, u0_3, tspan_3, p_3)
    sol_3 = solve(prob_3)

    plot(sol_3)
end
```



```
begin
    u0_4 = [1.0, 0, 0]
```

```julia
    tspan_4 = (0.0, 100.0)
    p_4 = [(false, true, true, true), 30,
            0., 2., 0.2, 0.1, 0.2,
            0., 2., 5., 0.1,  2., 5., 1.0, 0.2]
    prob_4 = ODEProblem(feed_forward_loop!, u0_4, tspan_4, p_4)
    sol_4 = solve(prob_4)

    plot(sol_4)
end
```