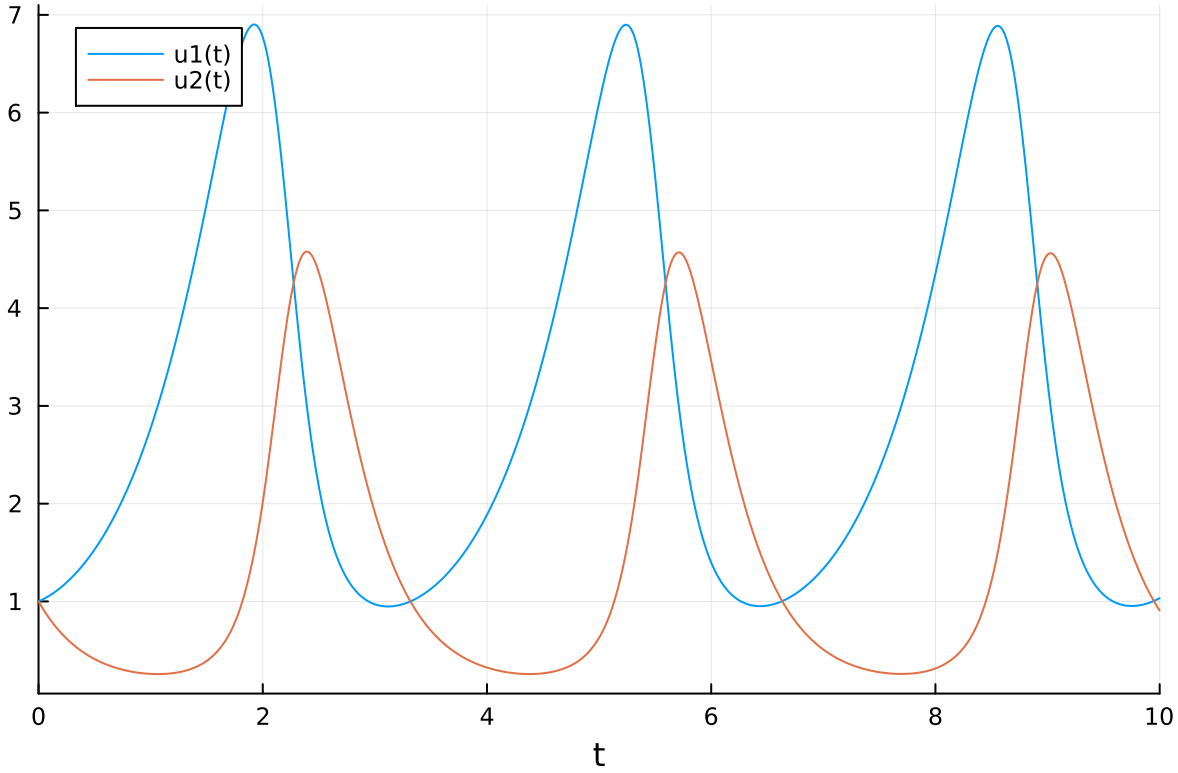
```
begin
    import Pkg
    Pkg.activate("/home/noams/repositories/neurohelp/neurohelp-julia-
    codebase/DelayedDifferentialAnalysis")
end
```

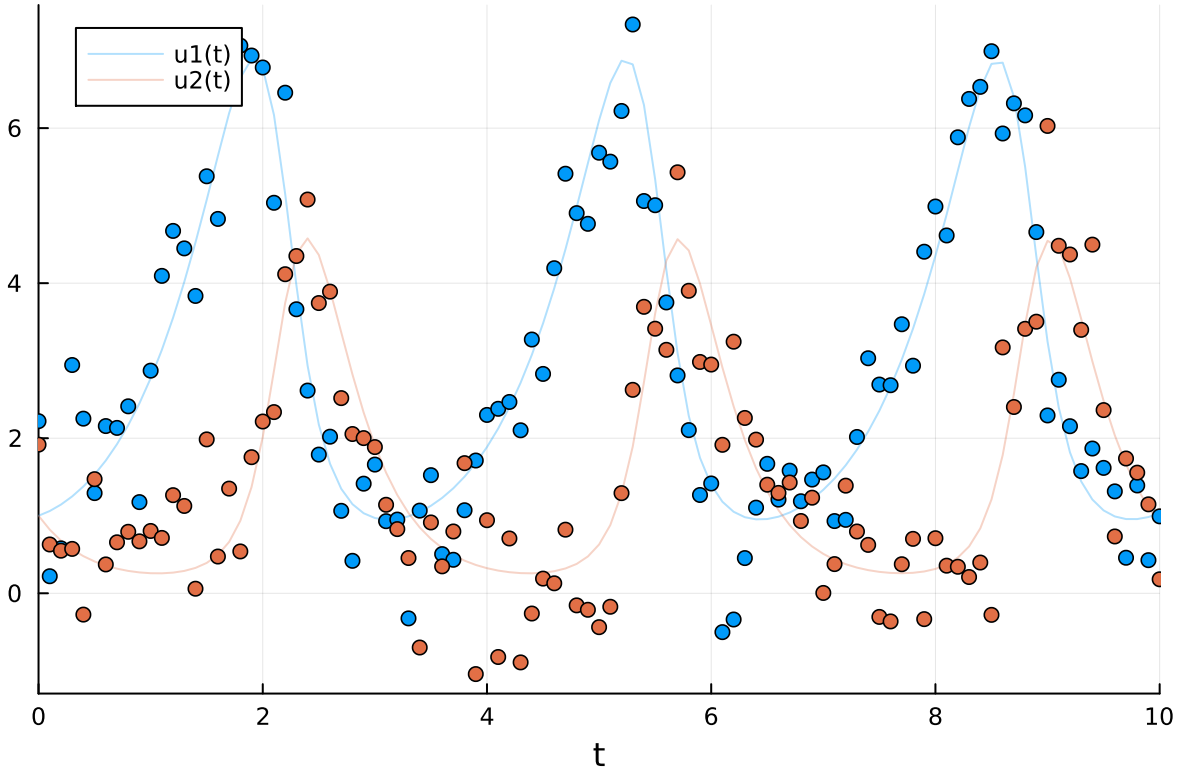
```
  Activating project at `~/repositories/neurohelp/neurohelp-julia-codebas
e/DelayedDifferentialAnalysis`
```

```
TaskLocalRNG()
```

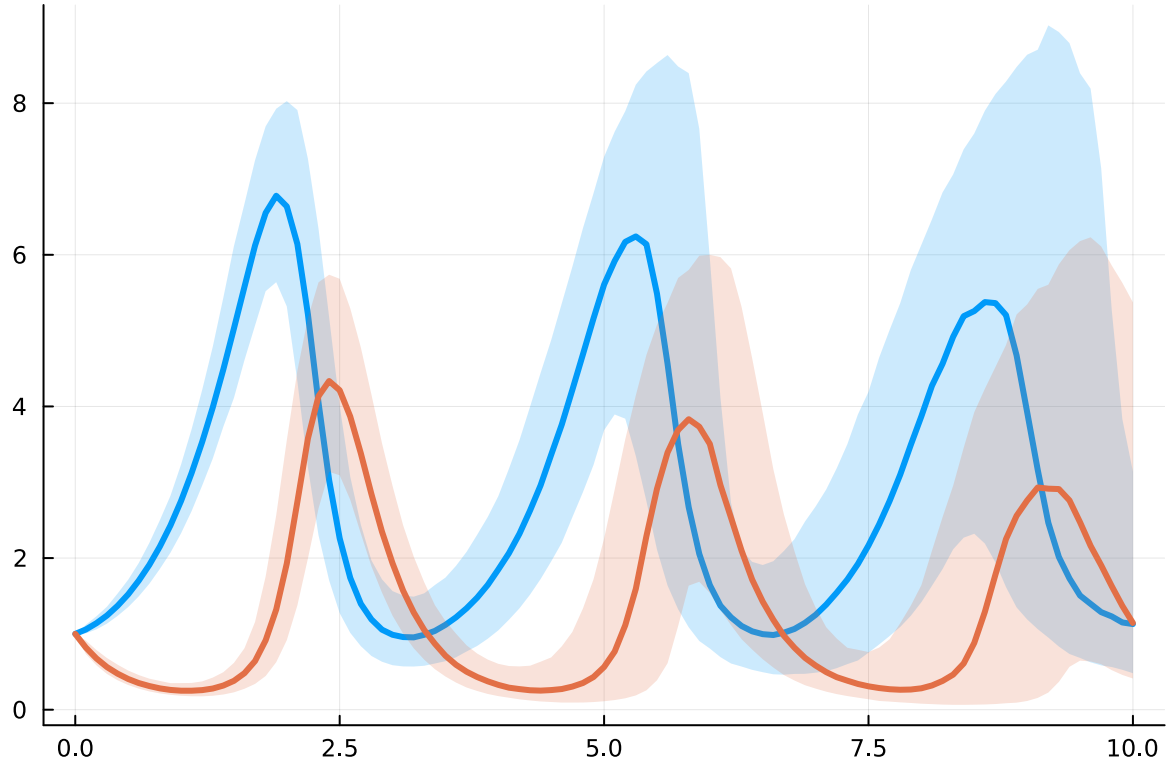
```
begin
    using Turing
    using DifferentialEquations
    using StatsPlots
    using LinearAlgebra
    using Random
    using PlutoUI
    using Zarr
    using Printf
    # using CUDA
    # using CuArrays
    # using TopoPlots
    # using Neuroimaging
    Random.seed!(16);
end
```




```julia
begin
    # Define Lotka-Volterra model.
    function lotka_volterra(du, u, p, t)
        # Model parameters.
        α, β, γ, δ = p
        # Current state.
        x, y = u

        # Evaluate differential equations.
        du[1] = (α - β * y) * x # prey
        du[2] = (δ * x - γ) * y # predator

        return nothing
    end

    # Define initial-value problem.
    u0 = [1.0, 1.0]
    p = [1.5, 1.0, 3.0, 1.0]
    tspan = (0.0, 10.0)
    prob = ODEProblem(lotka_volterra, u0, tspan, p)

    # Plot simulation.
    plot(solve(prob, Tsit5()))
end
```




```
begin
    sol = solve(prob, Tsit5(); saveat=0.1)
    odedata = Array(sol) + 0.6 * randn(size(Array(sol)))
    # Plot simulation and noisy observations.
    plot(sol; alpha=0.3)
    scatter!(sol.t, odedata'; color=[1 2], label="")
end
```




```
begin
    function multiplicative_noise!(du, u, p, t)
        x, y = u
        du[1] = p[5] * x
        du[2] = p[6] * y
    end
    p_sde = [1.5, 1.0, 3.0, 1.0, σ₁, σ₂]

    function lotka_volterra!(du, u, p, t)
        x, y = u
        α, β, γ, δ = p
        du[1] = dx = α * x - β * x * y
        du[2] = dy = δ * x * y - γ * y
    end

    prob_sde = SDEProblem(lotka_volterra!, multiplicative_noise!, u0, tspan, p_sde)

    ensembleprob = EnsembleProblem(prob_sde)
    data = solve(ensembleprob, SOSRI(); saveat=0.1, trajectories=1000)
    plot(EnsembleSummary(data))
end
```

0.1

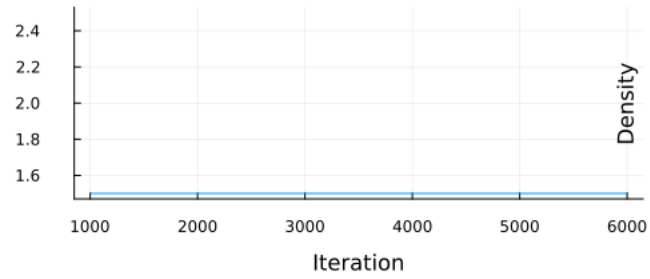
```
@bind σ₁ Slider(0:0.1:1, default=0.1, show_value=true)
```

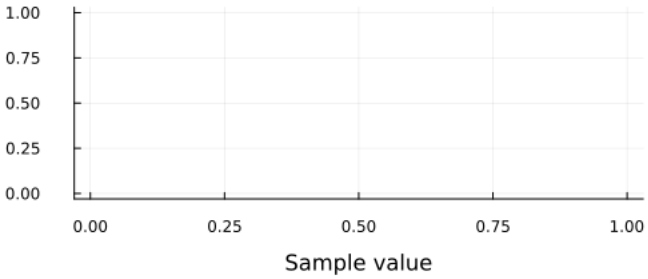
0.2

```
@bind σ₂ Slider(0:0.1:1, default=0.1, show_value=true)
```

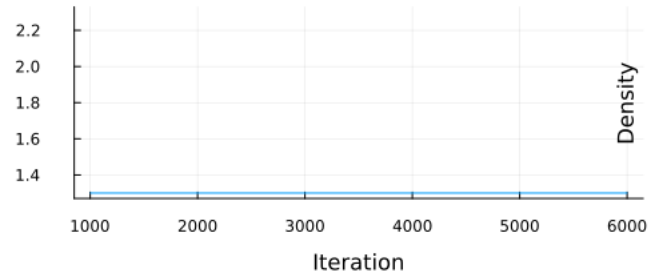
```julia
@model function fitlv_sde(data, prob)
    # Prior distributions.
    σ ~ InverseGamma(2, 3)
    α ~ truncated(Normal(1.3, 0.5), 0.5, 2.5)
    β ~ truncated(Normal(1.2, 0.25), 0.5, 2)
    γ ~ truncated(Normal(3.2, 0.25), 2.2, 4.0)
    δ ~ truncated(Normal(1.2, 0.25), 0.5, 2.0)
    ϕ1 ~ truncated(Normal(0.12, 0.3), 0.05, 0.25)
    ϕ2 ~ truncated(Normal(0.12, 0.3), 0.05, 0.25)

    # Simulate stochastic Lotka-Volterra model.
    p = [α, β, γ, δ, ϕ1, ϕ2]
    predicted = solve(prob, SOSRI(); p=p, saveat=0.1)

    # Early exit if simulation could not be computed successfully.
    if predicted.retcode !== :Success
        Turing.@addlogprob! -Inf
        return nothing
    end

    # Observations.
    for i in 1:length(predicted)
        data[:, i] ~ MvNormal(predicted[i], σ^2 * I)
    end

    return nothing
end;
```

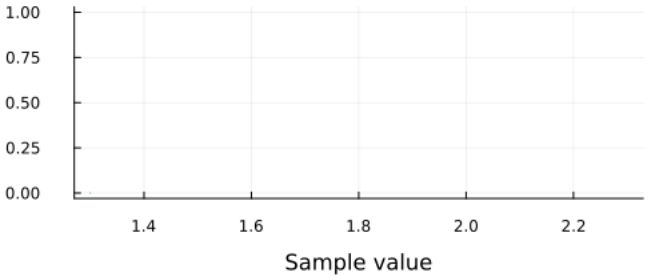
σ


2.4
2.2
2.0
1.8
1.6
Density
1000
2000
3000
4000
5000
6000
Iteration

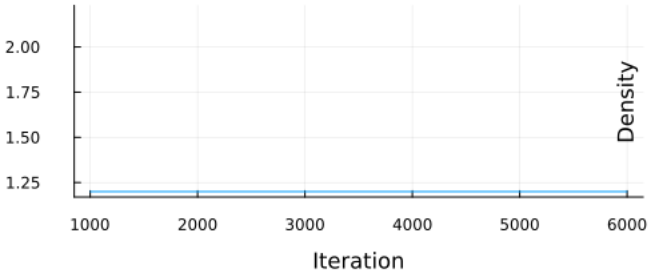

1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Sample value

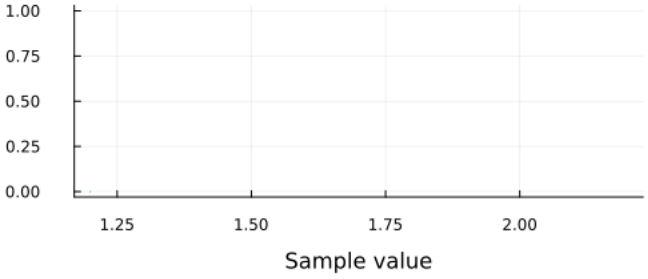
α


2.2
2.0
1.8
1.6
1.4
Density
1000
2000
3000
4000
5000
6000
Iteration

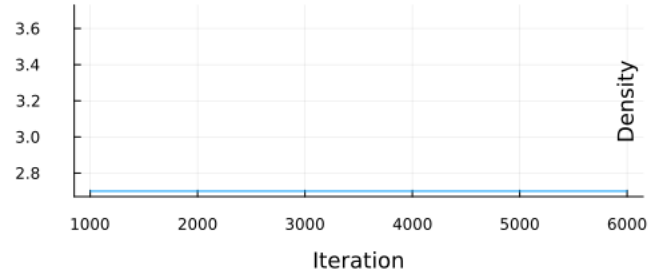

1.00
0.75
0.50
0.25
0.00
1.4
1.6
1.8
2.0
2.2
Sample value

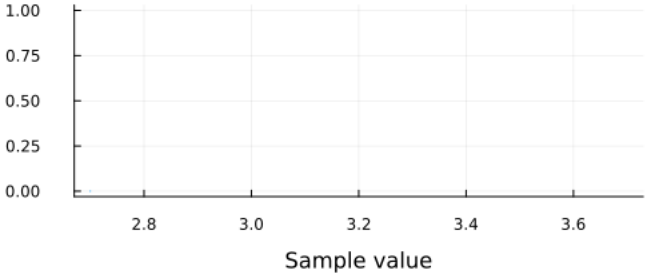
β


2.00
1.75
1.50
1.25
Density
1000
2000
3000
4000
5000
6000
Iteration

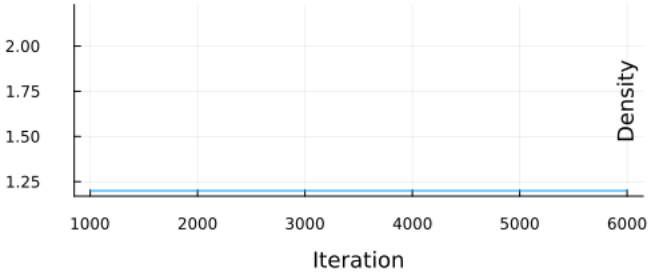

1.00
0.75
0.50
0.25
0.00
1.25
1.50
1.75
2.00
Sample value

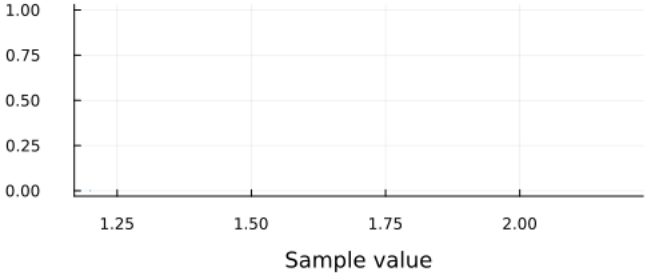
γ


3.6
3.4
3.2
3.0
2.8
Density
1000
2000
3000
4000
5000
6000
Iteration

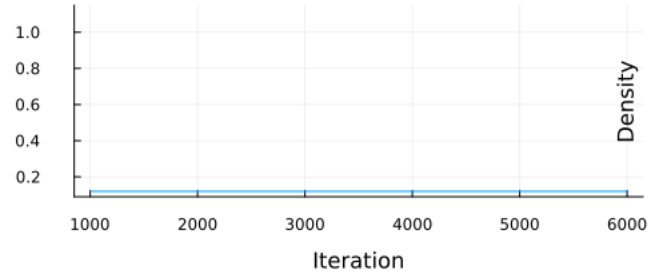

1.00
0.75
0.50
0.25
0.00
2.8
3.0
3.2
3.4
3.6
Sample value

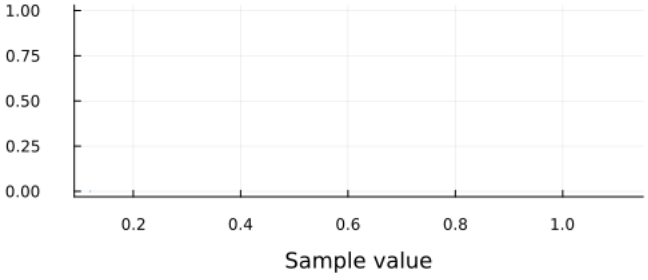
δ


2.00
1.75
1.50
1.25
Density
1000
2000
3000
4000
5000
6000
Iteration


1.00
0.75
0.50
0.25
0.00
1.25
1.50
1.75
2.00
Sample value

ϕ1


1.0
0.8
0.6
0.4
0.2
Density
1000
2000
3000
4000
5000
6000
Iteration


1.00
0.75
0.50
0.25
0.00
0.2
0.4
0.6
0.8
1.0
Sample value

ϕ2


1.0


1.00

```
begin
    model_sde = fitlv_sde(odedata, prob_sde)

    setadbackend(:forwarddiff)
    chain_sde = sample(
        model_sde,
        NUTS(0.25),
        5000;
        init_params=[1.5, 1.3, 1.2, 2.7, 1.2, 0.12, 0.12],
        progress=false,
    )
    plot(chain_sde)
end
```

```
Found initial step size
ϵ: 0.07500000000000001

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.

Incorrect ϵ = NaN; ϵ_previous = 0.07500000000000001 is used instead.
```

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