Numbers with Uncertainties

Mosè Giordano, Chris Rackauckas

March 1, 2019

The result of a measurement should be given as a number with an attached uncertainties, besides the physical unit, and all operations performed involving the result of the measurement should propagate the uncertainty, taking care of correlation between quantities.

There is a Julia package for dealing with numbers with uncertainties: `Measurements.jl`. Thanks to Julia's features, `DifferentialEquations.jl` easily works together with `Measurements.jl` out-of-the-box.

This notebook will cover some of the examples from the tutorial about classical Physics.

0.1 Caveat about `Measurement` type

Before going on with the tutorial, we must point up a subtlety of `Measurements.jl` that you should be aware of:

```
using Measurements
```

```
5.23 ± 0.14 === 5.23 ± 0.14
```

```
false
```

```
(5.23± 0.14) - (5.23 ± 0.14)
```

0.0 ± 0.2

```
(5.23 ± 0.14) / (5.23 ± 0.14)
```

1.0 ± 0.038

The two numbers above, even though have the same nominal value and the same uncertainties, are actually two different measurements that only by chance share the same figures and their difference and their ratio have a non-zero uncertainty. It is common in physics to get very similar, or even equal, results for a repeated measurement, but the two measurements are not the same thing.

Instead, if you have *one measurement* and want to perform some operations involving it, you have to assign it to a variable:

```
x = 5.23 ± 0.14
x === x
```

```
true
```

```
x - x
```

0.0 ± 0.0

```
x / x
```

1.0 ± 0.0

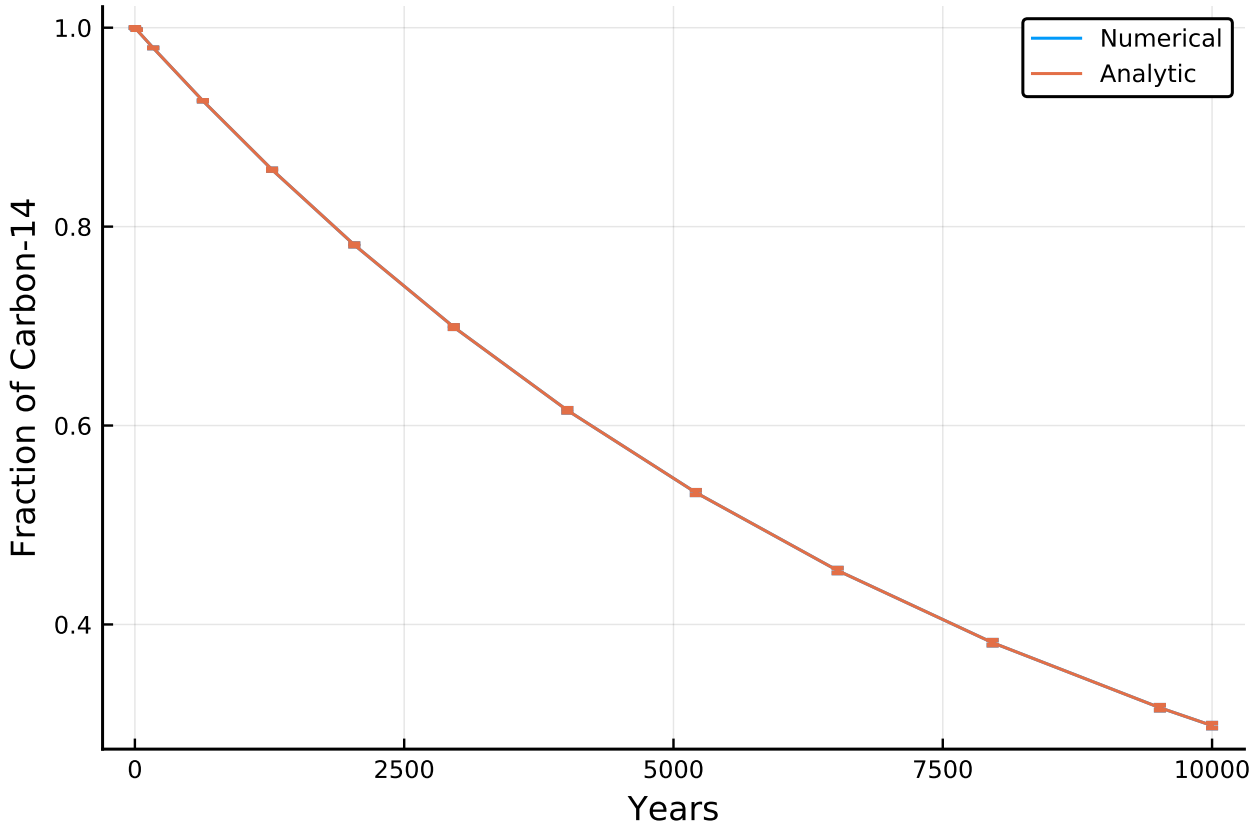
0.2 Radioactive Decay of Carbon-14

The rate of decay of carbon-14 is governed by a first order linear ordinary differential equation

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = -\frac{u(t)}{\tau}$$

where τ is the mean lifetime of carbon-14, which is related to the half-life $t_{1/2} = (5730 \pm 40)$ years by the relation $\tau = t_{1/2}/\ln(2)$.

```
using DifferentialEquations, Measurements, Plots

pyplot()

# Half-life and mean lifetime of radiocarbon, in years
t_12 = 5730 ± 40
τ = t_12 / log(2)

#Setup
u_0 = 1 ± 0
tspan = (0.0, 10000.0)

#Define the problem
radioactivedecay(u,p,t) = - u / τ

#Pass to solver
prob = ODEProblem(radioactivedecay, u_0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-8)

# Analytic solution
u = exp.(- sol.t / τ)

plot(sol.t, sol.u, label = "Numerical", xlabel = "Years", ylabel = "Fraction of
    Carbon-14")
plot!(sol.t, u, label = "Analytic")
```



The two curves are perfectly superimposed, indicating that the numerical solution matches the analytic one. We can check that also the uncertainties are correctly propagated in the numerical solution:

```
println("Quantity of carbon-14 after ",  sol.t[11], " years:")
```

```
Quantity of carbon-14 after 5207.5228514026385 years:
```

```
println("Numerical:  ", sol[11])
```

```
Numerical: 0.5326215661145899 ± 0.0023422116367677525
```

```
println("Analytic:  ", u[11])
```

```
Analytic:  0.5326215654338256 ± 0.002342211664674973
```

Both the value of the numerical solution and its uncertainty match the analytic solution within the requested tolerance. We can also note that close to 5730 years after the beginning of the decay (half-life of the radioisotope), the fraction of carbon-14 that survived is about 0.5.

0.3 Simple pendulum

0.3.1 Small angles approximation

The next problem we are going to study is the simple pendulum in the approximation of small angles. We address this simplified case because there exists an easy analytic solution to compare.

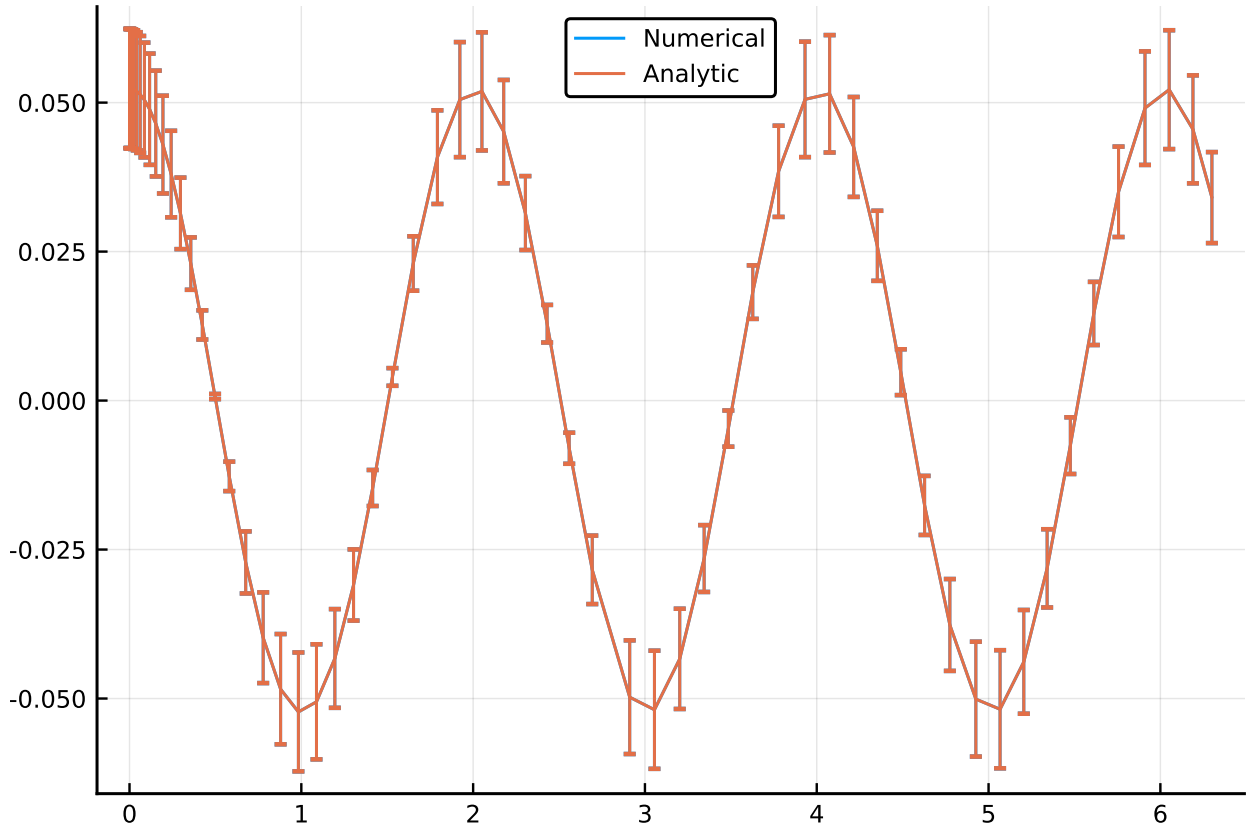
The differential equation we want to solve is

$$\ddot{\theta} + \frac{g}{L}\theta = 0$$

where $g = (9.79 \pm 0.02)$ m/s^2 is the gravitational acceleration measured where the experiment is carried out, and $L = (1.00 \pm 0.01)$ m is the length of the pendulum.

When you set up the problem for `DifferentialEquations.jl` remember to define the measurements as variables, as seen above.

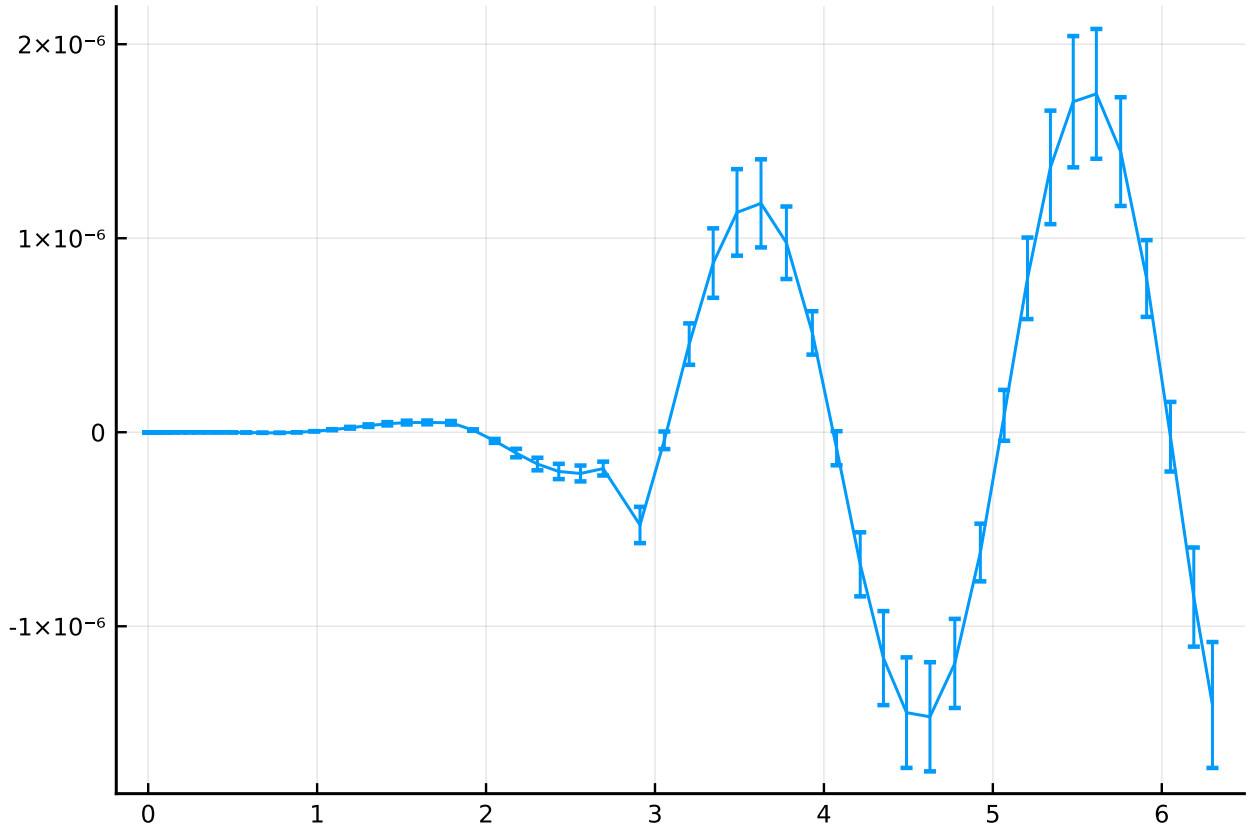
```
using DifferentialEquations, Measurements, Plots

g = 9.79 ± 0.02; # Gravitational constants
L = 1.00 ± 0.01; # Length of the pendulum

#Initial Conditions
u_0 = [0 ± 0, π / 60 ± 0.01] # Initial speed and initial angle
tspan = (0.0, 6.3)

#Define the problem
function simplependulum(du,u,p,t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L)*θ
end

#Pass to solvers
prob = ODEProblem(simplependulum, u_0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-6)

# Analytic solution
u = u_0[2] .* cos.(sqrt(g / L) .* sol.t)

plot(sol.t, getindex.(sol.u, 2), label = "Numerical")
plot!(sol.t, u, label = "Analytic")
```



Also in this case there is a perfect superimposition between the two curves, including their uncertainties.

We can also have a look at the difference between the two solutions:

```
plot(sol.t, getindex.(sol.u, 2) .- u, label = "")
```

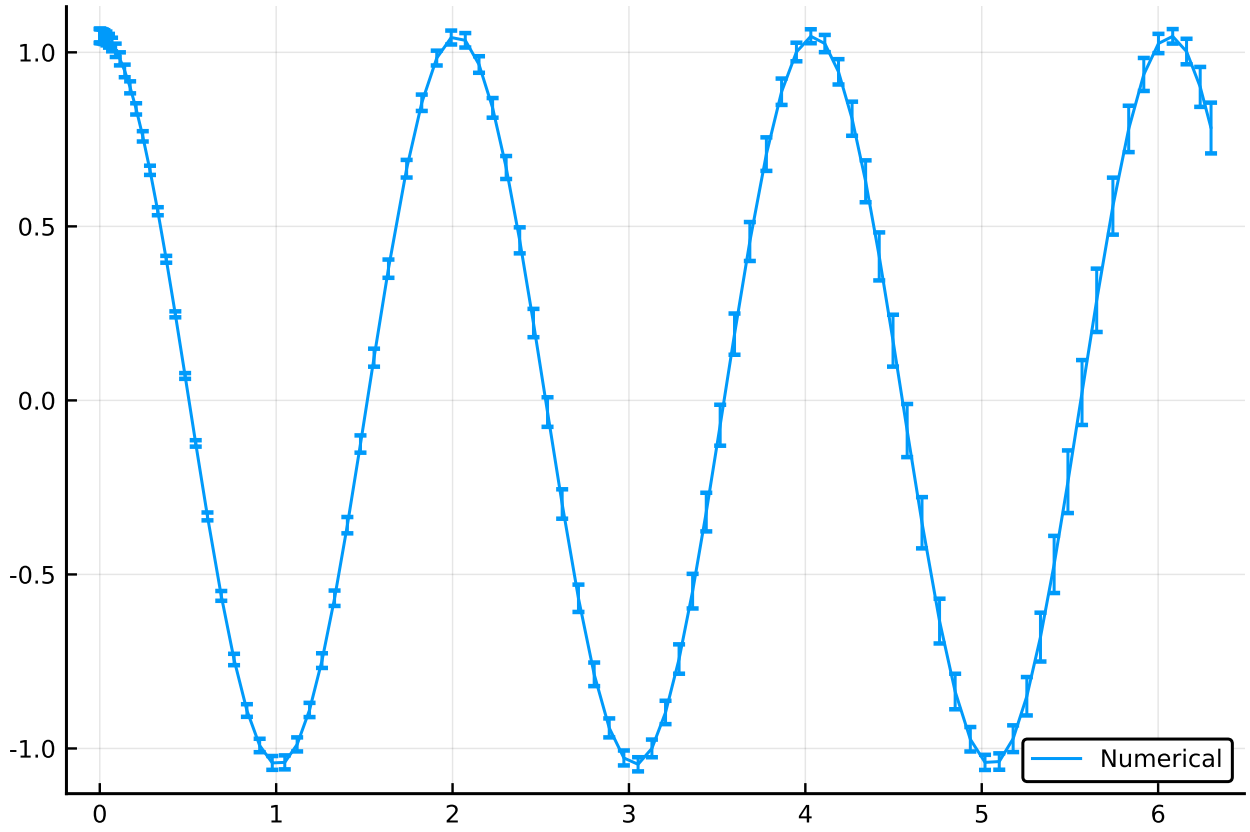


0.4 Arbitrary amplitude

Now that we know how to solve differential equations involving numbers with uncertainties we can solve the simple pendulum problem without any approximation. This time the differential equation to solve is the following:

$$\ddot{\theta} + \frac{g}{L}\sin(\theta) = 0$$

```
g = 9.79 ± 0.02; # Gravitational constants
L = 1.00 ± 0.01; # Length of the pendulum

#Initial Conditions
u_0 = [0 ± 0, π / 3 ± 0.02] # Initial speed and initial angle
tspan = (0.0, 6.3)

#Define the problem
function simplependulum(du,u,p,t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L) * sin(θ)
end

#Pass to solvers
prob = ODEProblem(simplependulum, u_0, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-6)

plot(sol.t, getindex.(sol.u, 2), label = "Numerical")
```



We note that in this case the period of the oscillations is not constant.

```
using DiffEqTutorials
DiffEqTutorials.tutorial_footer(WEAVE_ARGS[:folder],WEAVE_ARGS[:file])
```

0.5 Appendix

These benchmarks are part of the DiffEqTutorials.jl repository, found at: https://github.com/JuliaDiffEq/

To locally run this tutorial, do the following commands:

```
using DiffEqTutorials
DiffEqTutorials.weave_file("type_handling","uncertainties.jmd")
```

Computer Information:

```
Julia Version 1.1.0
Commit 80516ca202 (2019-01-21 21:24 UTC)
Platform Info:
  OS: Windows (x86_64-w64-mingw32)
  CPU: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, skylake)
Environment:
```

```
  JULIA_EDITOR = "C:\Users\accou\AppData\Local\atom\app-1.34.0\atom.exe" -a
  JULIA_NUM_THREADS = 6
```

Package Information:

```
Status `C:\Users\accou\.julia\environments\v1.1\Project.toml`
[7e558dbc-694d-5a72-987c-6f4ebed21442] ArbNumerics 0.3.6
[c52e3926-4ff0-5f6e-af25-54175e0327b1] Atom 0.7.14
[6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf] BenchmarkTools 0.4.2
[336ed68f-0bac-5ca0-87d4-7b16caf5d00b] CSV 0.4.3
[3895d2a7-ec45-59b8-82bb-cfc6a382f9b3] CUDAapi 0.6.0
[be33ccc6-a3ff-5ff2-a52e-74243cff1e17] CUDAnative 1.0.1
[3a865a2d-5b23-5a0f-bc46-62713ec82fae] CuArrays 0.9.1
[a93c6f00-e57d-5684-b7b6-d8193f3e46c0] DataFrames 0.17.1
[55939f99-70c6-5e9b-8bb0-5071ed7d61fd] DecFP 0.4.8
[abce61dc-4473-55a0-ba07-351d65e31d42] Decimals 0.4.0
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.2.0+
[39dd38d3-220a-591b-8e3c-4c3a8c710a94] Dierckx 0.4.1
[2b5f629d-d688-5b77-993f-72d75c75574e] DiffEqBase 5.4.0+
[bb2cbb15-79fc-5d1e-9bf1-8ae49c7c1650] DiffEqBenchmarks 0.0.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.6.1
[aae7a2af-3d4f-5e19-a356-7da93b79d9d0] DiffEqFlux 0.2.0
[c894b116-72e5-5b58-be3c-e6d8d4ac2b12] DiffEqJump 6.1.0+
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.6.0+
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.1.0
[225cb15b-72e6-54e6-9a40-306d353791de] DiffEqTutorials 0.0.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.3.0
[497a8b3b-efae-58df-a0af-a86822472b78] DoubleFloats 0.7.5
[587475ba-b771-5e3f-ad9e-33799f191a9c] Flux 0.7.3
[f6369f11-7733-5829-9624-2563aa707210] ForwardDiff 0.10.3+
[28b8d3ca-fb5f-59d9-8090-bfdbd6d07a71] GR 0.38.1
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.17.0
[c601a237-2ae4-5e1e-952c-7a85b0c7eef1] Interact 0.9.1
[b6b21f68-93f8-5de0-b562-5493be1d77c9] Ipopt 0.5.4
[4076af6c-e467-56ae-b986-b466b2749572] JuMP 0.19.0
[e5e0dc1b-0480-54bc-9374-aad01c23163d] Juno 0.5.4
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.4.0
[eff96d63-e80a-5855-80a2-b1b0885c5ab7] Measurements 2.0.0
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.5.1
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.4.0
[54ca160b-1b9f-5127-a996-1867f4bc2a2c] ODEInterface 0.4.5+
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.0.0
[429524aa-4258-5aef-a3af-852621145aeb] Optim 0.17.2
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.3.0+
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.23.0
```

```
[71ad9d73-34c4-5ce9-b7b1-f7bd31ac38ba] PuMaS 0.0.0
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.7.0
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 0.20.0
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.2
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.1.1+
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.1.0+
[1986cc42-f94f-5a68-af5c-568840ba703d] Unitful 0.14.0
[2a06ce6d-1589-592b-9c33-f37faeaed826] UnitfulPlots 0.0.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.7.2
```