

Finding Maxima and Minima of DiffEq Solutions

Chris Rackauckas

February 23, 2019

0.0.1 Setup

In this tutorial we will show how to use Optim.jl to find the maxima and minima of solutions. Let's take a look at the double pendulum:

```
#Constants and setup
using OrdinaryDiffEq
initial = [0.01, 0.01, 0.01, 0.01]
tspan = (0.,100.)

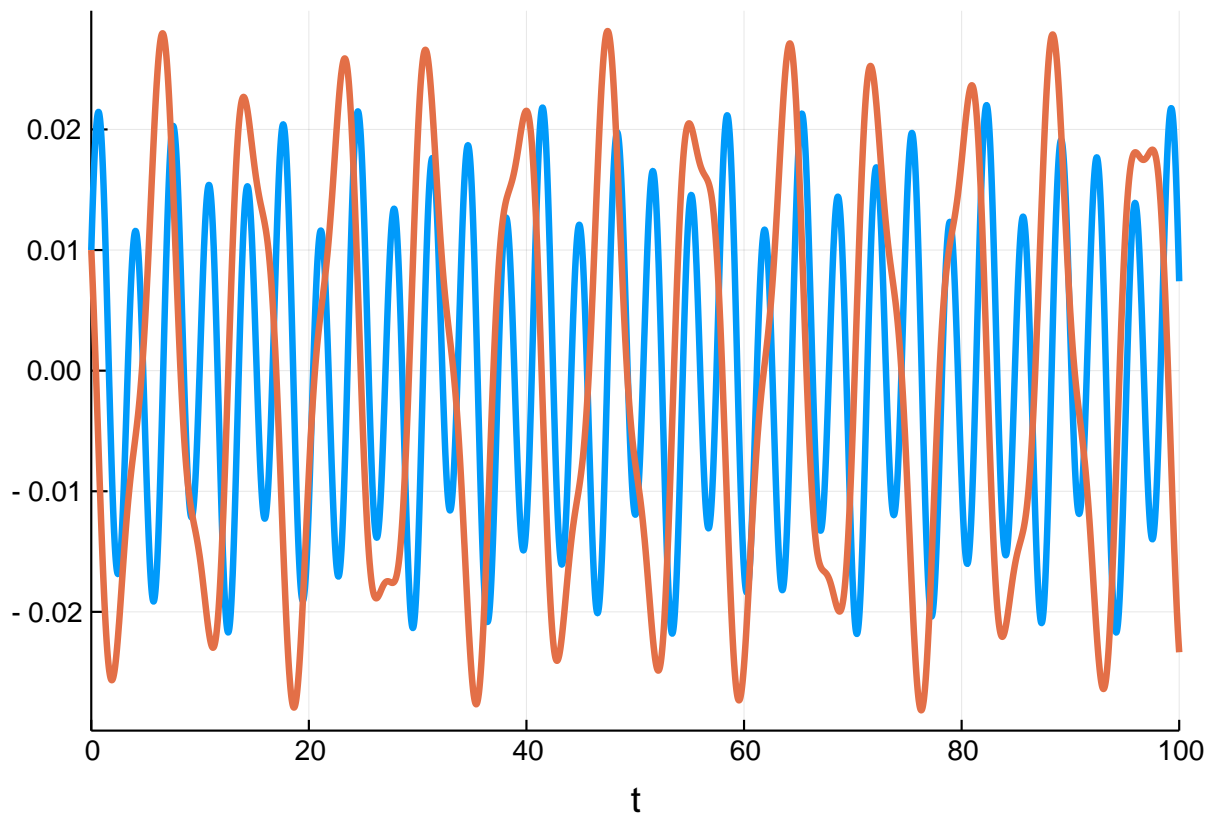
#Define the problem
function double_pendulum_hamiltonian(udot,u,p,t)
    α = u[1]
    lα = u[2]
    β = u[3]
    lβ = u[4]
    udot .=
    [2(lα-(1+cos(β))lβ)/(3-cos(2β)),
     -2sin(α) - sin(α+β),
     2(-(1+cos(β))lα + (3+2cos(β))lβ)/(3-cos(2β)),
     -sin(α+β) - 2sin(β)*(((lα-lβ)lβ)/(3-cos(2β))) + 2sin(2β)*((lα^2 - 2(1+cos(β))lα*lβ
     + (3+2cos(β))lβ^2)/(3-cos(2β))^2)]
end

#Pass to solvers
poincare = ODEProblem(double_pendulum_hamiltonian, initial, tspan)

sol = solve(poincare, Tsit5())
```

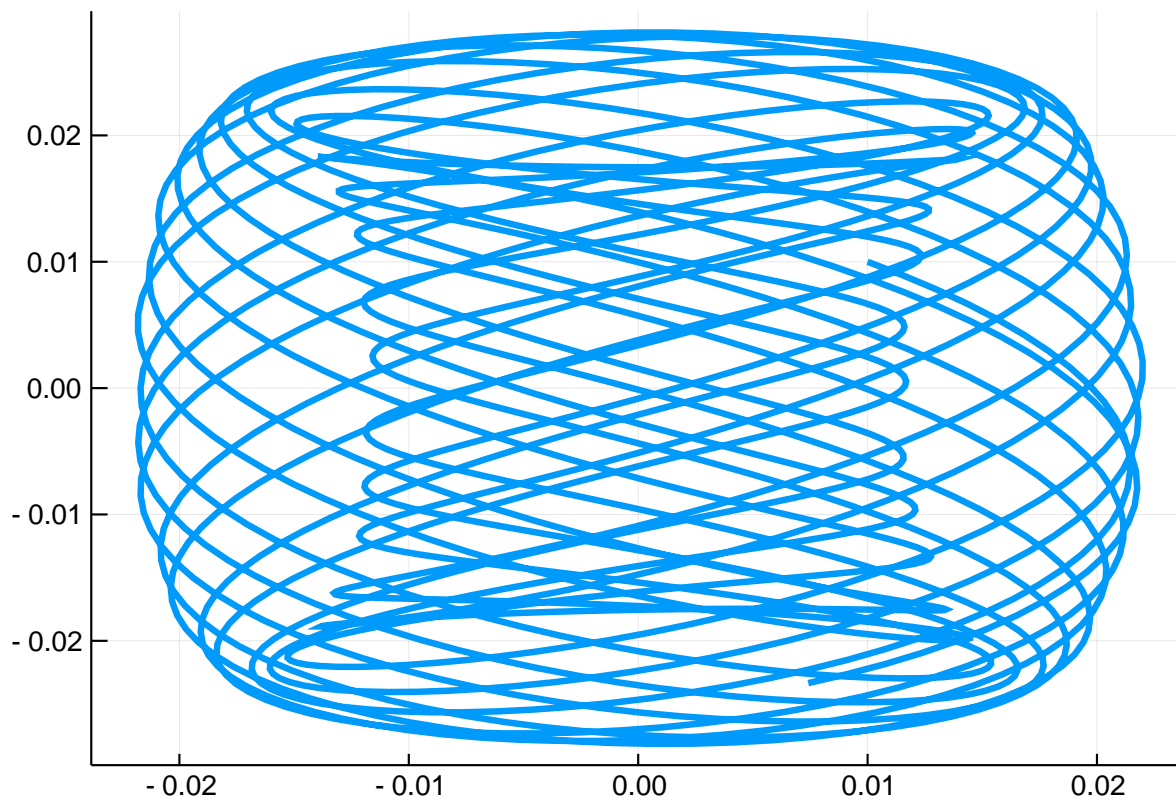
In time, the solution looks like:

```
using Plots; gr()
plot(sol, vars=[(0,3),(0,4)], leg=false, plotdensity=10000)
```



while it has the well-known phase-space plot:

```
plot(sol, vars=(3,4), leg=false)
```



0.0.2 Local Optimization

Let's find out what some of the local maxima and minima are. Optim.jl can be used to minimize functions, and the solution type has a continuous interpolation which can be used. Let's look for the local optima for the 4th variable around $t=20$. Thus our optimization function is:

```
f = (t) -> sol(t,idxs=4)
```

`first(t)` is the same as `t[1]` which transforms the array of size 1 into a number. `idxs=4` is the same as `sol(first(t))[4]` but does the calculation without a temporary array and thus is faster. To find a local minima, we can simply call Optim on this function. Let's find a local minimum:

```
using Optim
opt = optimize(f,18.0,22.0)
```

From this printout we see that the minimum is at $t=18.63$ and the value is $-2.79e-2$. We can get these in code-form via:

```
println(opt.minimizer)
```

```
18.632126799604933
```

```
println(opt.minimum)
```

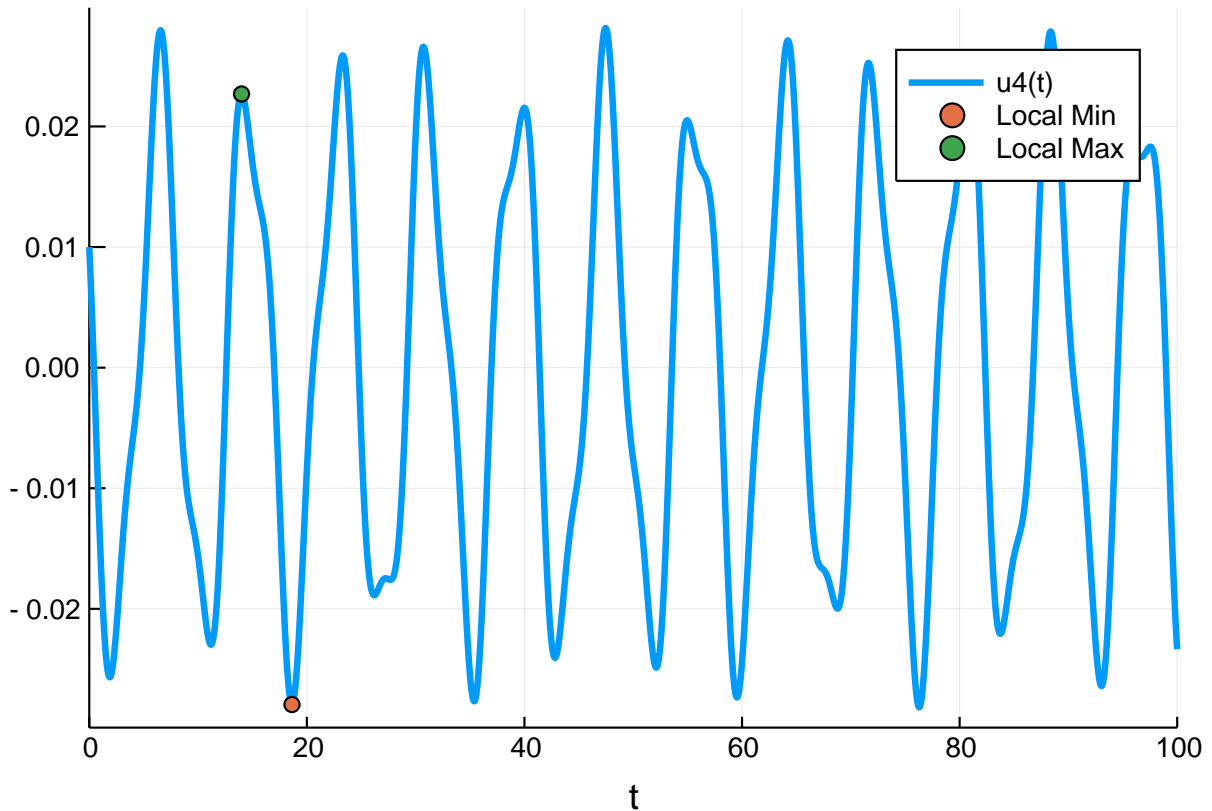
```
-0.027931635264245896
```

To get the maximum, we just minimize the negative of the function:

```
f = (t) -> -sol(first(t),idxs=4)
opt2 = optimize(f,0.0,22.0)
```

Let's add the maxima and minima to the plots:

```
plot(sol, vars=(0,4), plotdensity=10000)
scatter!([opt.minimizer],[opt.minimum],label="Local Min")
scatter!([opt2.minimizer],[-opt2.minimum],label="Local Max")
```



Brent's method will locally minimize over the full interval. If we instead want a local maxima nearest to a point, we can use `BFGS()`. In this case, we need to optimize a vector `[t]`, and thus dereference it to a number using `first(t)`.

```
f = (t) -> -sol(first(t),idxs=4)
opt = optimize(f,[20.0],BFGS())
```

0.0.3 Global Optimization

If we instead want to find global maxima and minima, we need to look somewhere else. For this there are many choices. A pure Julia option is `BlackBoxOptim.jl`, but I will use `NLopt.jl`. Following the `NLopt.jl` tutorial but replacing their function with our own:

```
import NLopt, ForwardDiff

count = 0 # keep track of # function evaluations

function g(t::Vector, grad::Vector)
    if length(grad) > 0
        #use ForwardDiff for the gradients
        grad[1] = ForwardDiff.derivative((t)->sol(first(t),idxs=4),t)
    end
    sol(first(t),idxs=4)
end

opt = NLopt.Opt(:GN_ORIG_DIRECT_L, 1)
NLopt.lower_bounds!(opt, [0.0])
NLopt.upper_bounds!(opt, [40.0])
NLopt.xtol_rel!(opt, 1e-8)
```

```

NLOpt.min_objective!(opt, g)
(minf,minx,ret) = NLOpt.optimize(opt,[20.0])
println(minf," ",minx," ",ret)

```

```

-0.027931635264245837 [18.6321] XTOL_REACHED

```

```

NLOpt.max_objective!(opt, g)
(maxf,maxx,ret) = NLOpt.optimize(opt,[20.0])
println(maxf," ",maxx," ",ret)

```

```

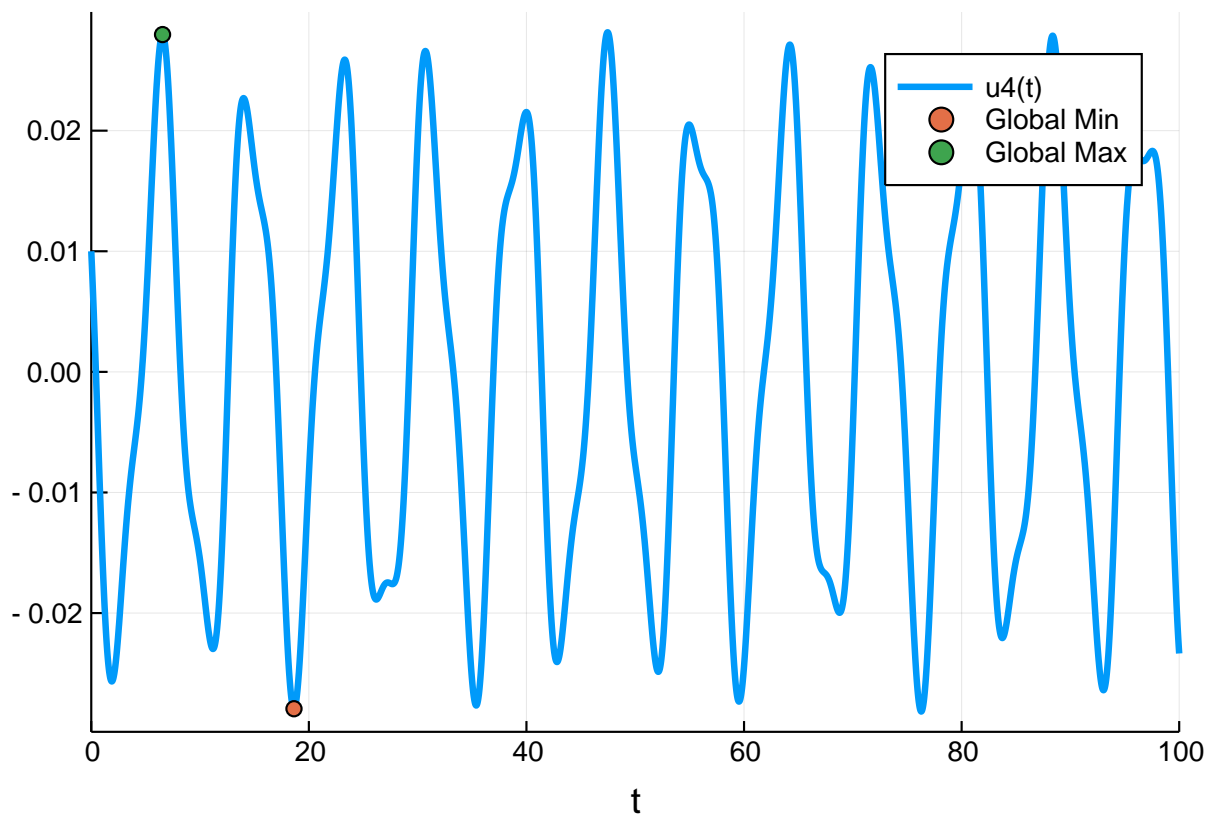
0.027968571933041954 [6.5537] XTOL_REACHED

```

```

plot(sol, vars=(0,4), plotdensity=10000)
scatter!([minx],[minf],label="Global Min")
scatter!([maxx],[maxf],label="Global Max")

```



0.1 Appendix

```

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

```

These benchmarks are part of the DiffEqTutorials.jl repository, found at:

<https://github.com/JuliaDiffEq/DiffEqTutorials.jl>

To locally run this tutorial, do the following commands:

```
using DiffEqTutorials
DiffEqTutorials.weave_file(".", "ode_extras/ode_minmax.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`
[c52e3926] Atom v0.7.14
[6e4b80f9] BenchmarkTools v0.4.2
[336ed68f] CSV v0.4.3
[be33ccc6] CUDANative v1.0.1
[3a865a2d] CuArrays v0.9.1
[a93c6f00] DataFrames v0.17.1
[39dd38d3] Dierckx v0.4.1
[459566f4] DiffEqCallbacks v2.5.2
[f3b72e0c] DiffEqDevTools v2.6.1
[aae7a2af] DiffEqFlux v0.2.0
[c894b116] DiffEqJump v6.1.0+ [C:\Users\accou\.julia\dev\DiffEqJump]
[1130ab10] DiffEqParamEstim v1.5.1
[055956cb] DiffEqPhysics v3.1.0
[225cb15b] DiffEqTutorials v0.0.0 [C:\Users\accou\.julia\external\DiffEq
Tutorials.jl]
[0c46a032] DifferentialEquations v6.3.0
[587475ba] Flux v0.7.3
[f6369f11] ForwardDiff v0.10.3+ [C:\Users\accou\.julia\dev\ForwardDiff]
[7073ff75] IJulia v1.17.0
[c601a237] Interact v0.9.1
[b6b21f68] Ipopt v0.5.4
[4076af6c] JuMP v0.18.5
[e5e0dc1b] Juno v0.5.4
[76087f3c] NLOpt v0.5.1
[429524aa] Optim v0.17.2
[1dea7af3] OrdinaryDiffEq v5.1.4+ [C:\Users\accou\.julia\dev\OrdinaryDif
fEq]
[65888b18] ParameterizedFunctions v4.1.0
[91a5bcdd] Plots v0.23.0
[71ad9d73] PuMaS v0.0.0 [C:\Users\accou\.julia\dev\PuMaS]
[d330b81b] PyPlot v2.7.0
[731186ca] RecursiveArrayTools v0.20.0
```

```
[90137ffa] StaticArrays v0.10.2
[789caeaf] StochasticDiffEq v6.1.1+ [C:\Users\accou\.julia\dev\Stochasti
cDiffEq`]
[c3572dad] Sundials v3.0.0
[1986cc42] Unitful v0.14.0
[44d3d7a6] Weave v0.7.1
```