Finding Maxima and Minima of DiffEq Solutions

Chris Rackauckas

March 1, 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)
```

```
ODEProblem with uType Array{Float64,1} and tType Float64. In-place: true
timespan: (0.0, 100.0)
u0: [0.01, 0.01, 0.01, 0.01]
```

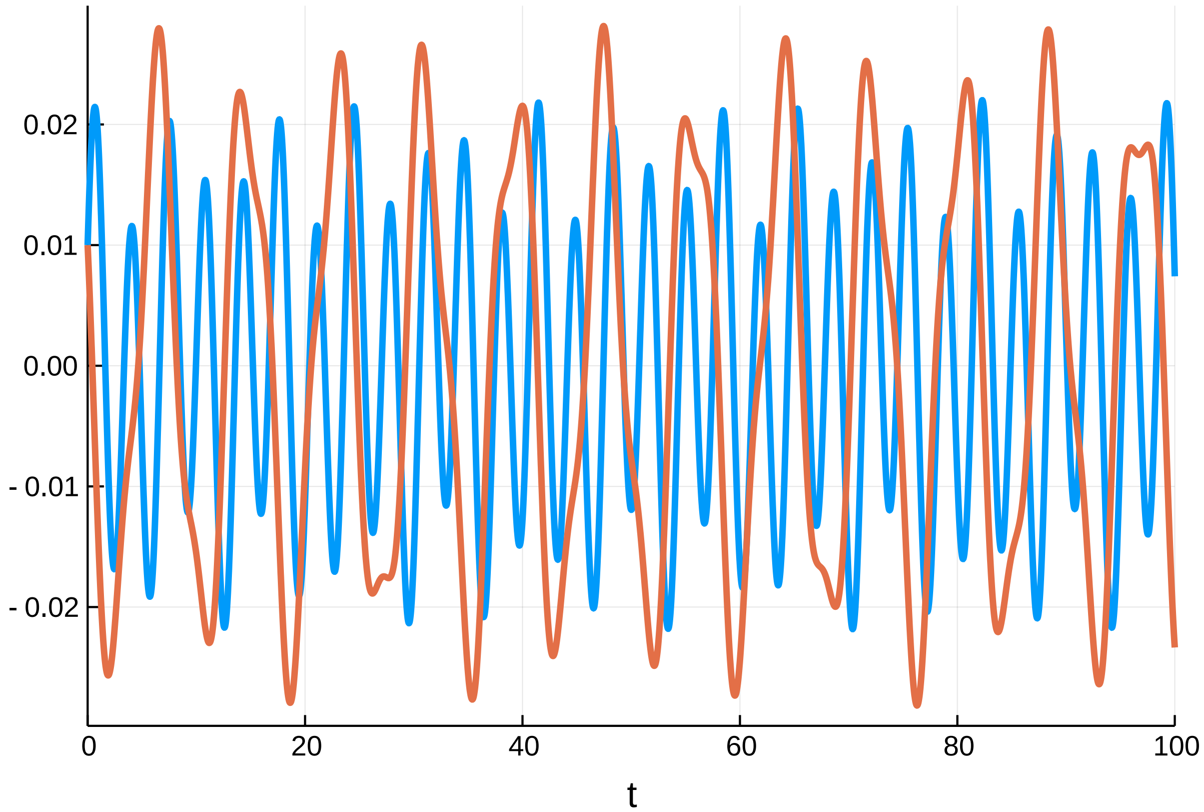
```
sol = solve(poincare, Tsit5())
```

```
retcode: Success
Interpolation: specialized 4th order "free" interpolation
t: 193-element Array{Float64,1}:
   0.0
   0.08332584852065579
```

```
   0.24175280272193683
   0.438953650048967
   0.679732254249109
   0.9647633763375199
   1.317944955684634
   1.7031210236334697
   2.067847793204029
   2.4717825254408226
   ⋮
  95.84571836675161
  96.35777612654947
  96.9291238553289
  97.44678729813481
  97.96247442963697
  98.5118249699588
  99.06081878698636
  99.58283477685136
 100.0
u: 193-element Array{Array{Float64,1},1}:
 [0.01, 0.01, 0.01, 0.01]
 [0.00917069, 0.006669, 0.0124205, 0.00826641]
 [0.00767328, 0.000374625, 0.0164426, 0.00463683]
 [0.00612597, -0.00730546, 0.0199674, -0.000336506]
 [0.0049661, -0.0163086, 0.0214407, -0.00670509]
 [0.00479557, -0.0262381, 0.0188243, -0.0139134]
 [0.00605469, -0.0371246, 0.0100556, -0.0210382]
 [0.00790078, -0.046676, -0.00267353, -0.025183]
 [0.00827652, -0.0527843, -0.0127315, -0.0252581]
 [0.00552358, -0.0552525, -0.0168439, -0.021899]
 ⋮
 [-0.0148868, 0.0423324, 0.0136282, 0.0180291]
 [-0.00819054, 0.0544225, 0.00944831, 0.0177401]
 [0.00412448, 0.0567489, -0.00515392, 0.017597]
 [0.0130796, 0.0480772, -0.0137706, 0.0182866]
 [0.0153161, 0.0316313, -0.00895722, 0.0171185]
 [0.0111156, 0.00992938, 0.0072972, 0.0103535]
 [0.00571392, -0.0117872, 0.020508, -0.00231029]
 [0.00421143, -0.0299109, 0.0187506, -0.0156505]
 [0.00574124, -0.0416539, 0.00741327, -0.023349]
```

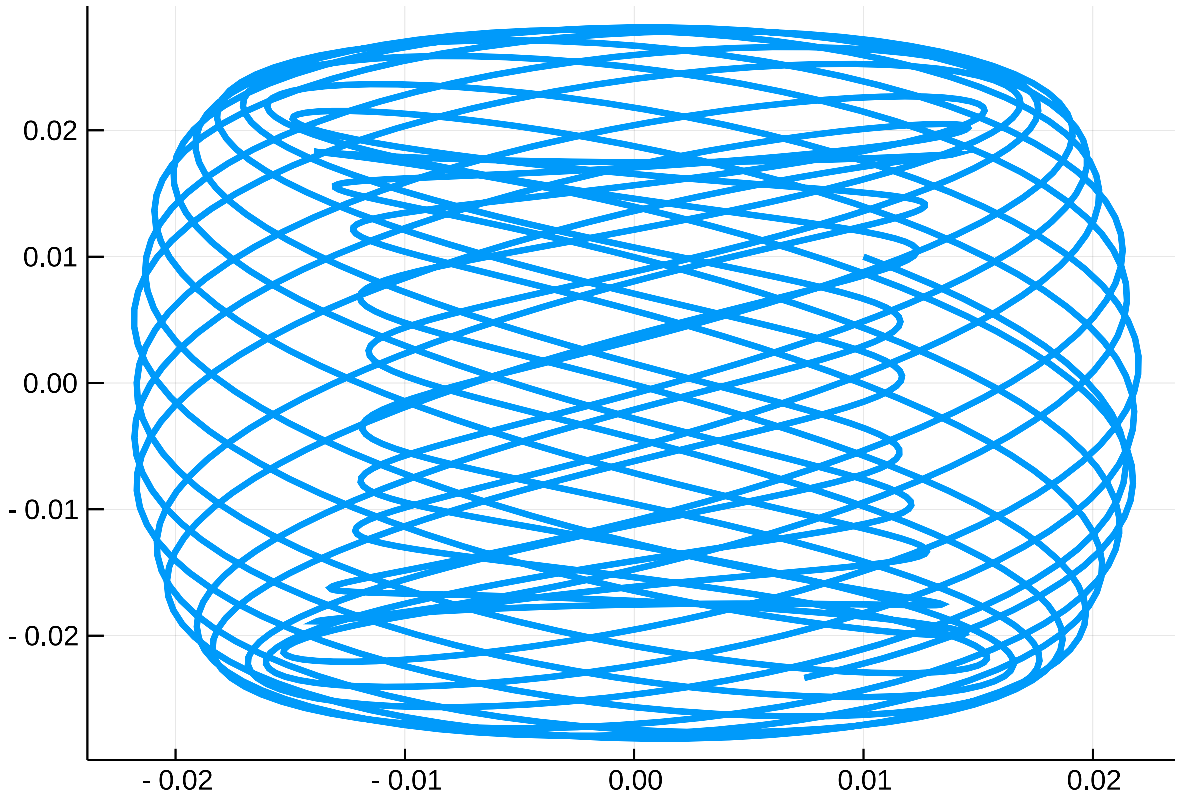
In time, the solution looks like:

```
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 fine 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)
```

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

`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)
```

```
Results of Optimization Algorithm
 * Algorithm: Brent's Method
 * Search Interval: [18.000000, 22.000000]
 * Minimizer: 1.863213e+01
 * Minimum: -2.793164e-02
 * Iterations: 11
 * Convergence: max(|x - x_upper|, |x - x_lower|) <= 2*(1.5e-08*|x|+2.2e-16
): true
 * Objective Function Calls: 12
```

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
```

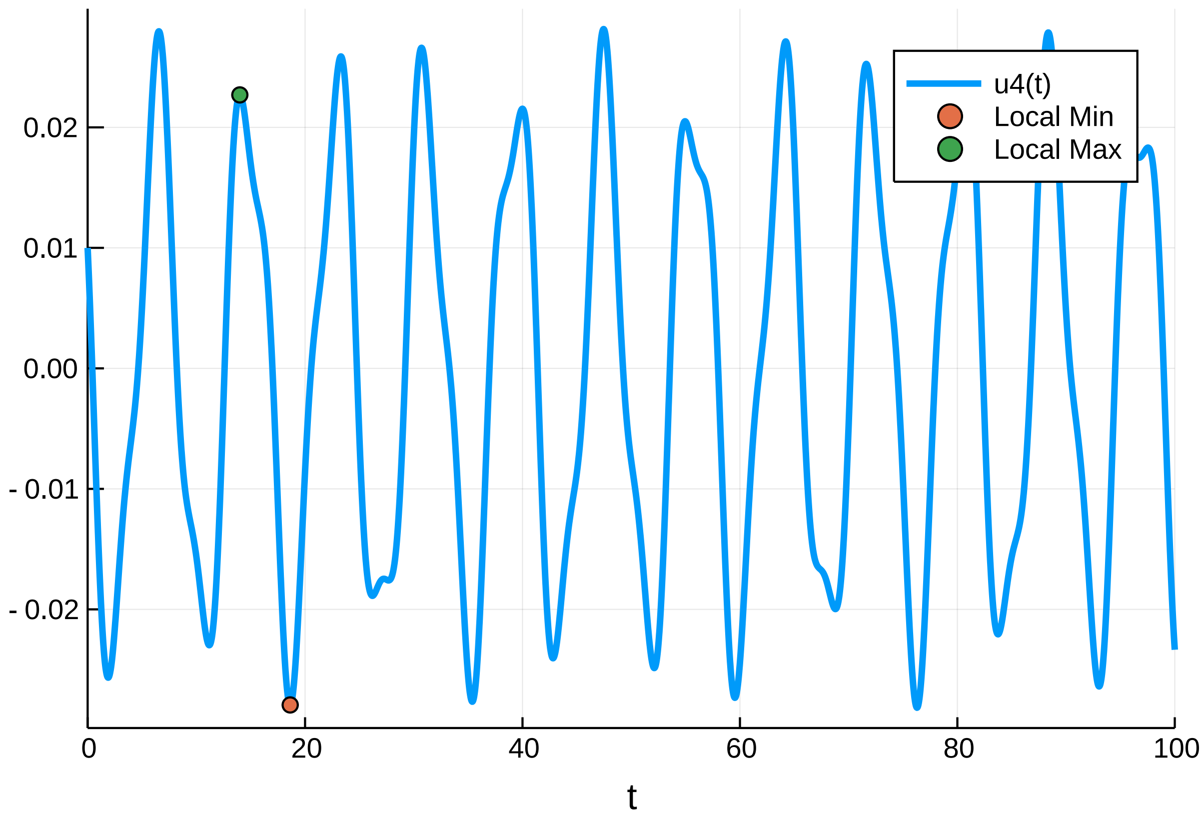
To get the maximum, we just minimize the negative of the function:

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

```
Results of Optimization Algorithm
 * Algorithm: Brent's Method
 * Search Interval: [0.000000, 22.000000]
 * Minimizer: 1.399975e+01
 * Minimum: -2.269411e-02
 * Iterations: 13
 * Convergence: max(|x - x_upper|, |x - x_lower|) <= 2*(1.5e-08*|x|+2.2e-16
): true
 * Objective Function Calls: 14
```

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())
```

```
Results of Optimization Algorithm
 * Algorithm: BFGS
 * Starting Point: [20.0]
```

```
 * Minimizer: [23.29760728871635]
 * Minimum: -2.588588e-02
 * Iterations: 4
 * Convergence: true
   * |x - x'| ≤ 0.0e+00: false
     |x - x'| = 1.11e-04
   * |f(x) - f(x')| ≤ 0.0e+00 |f(x)|: false
     |f(x) - f(x')| = -6.49e-09 |f(x)|
   * |g(x)| ≤ 1.0e-08: true
     |g(x)| = 8.42e-12
   * Stopped by an increasing objective: false
   * Reached Maximum Number of Iterations: false
 * Objective Calls: 16
 * Gradient Calls: 16
```

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 out 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)
```

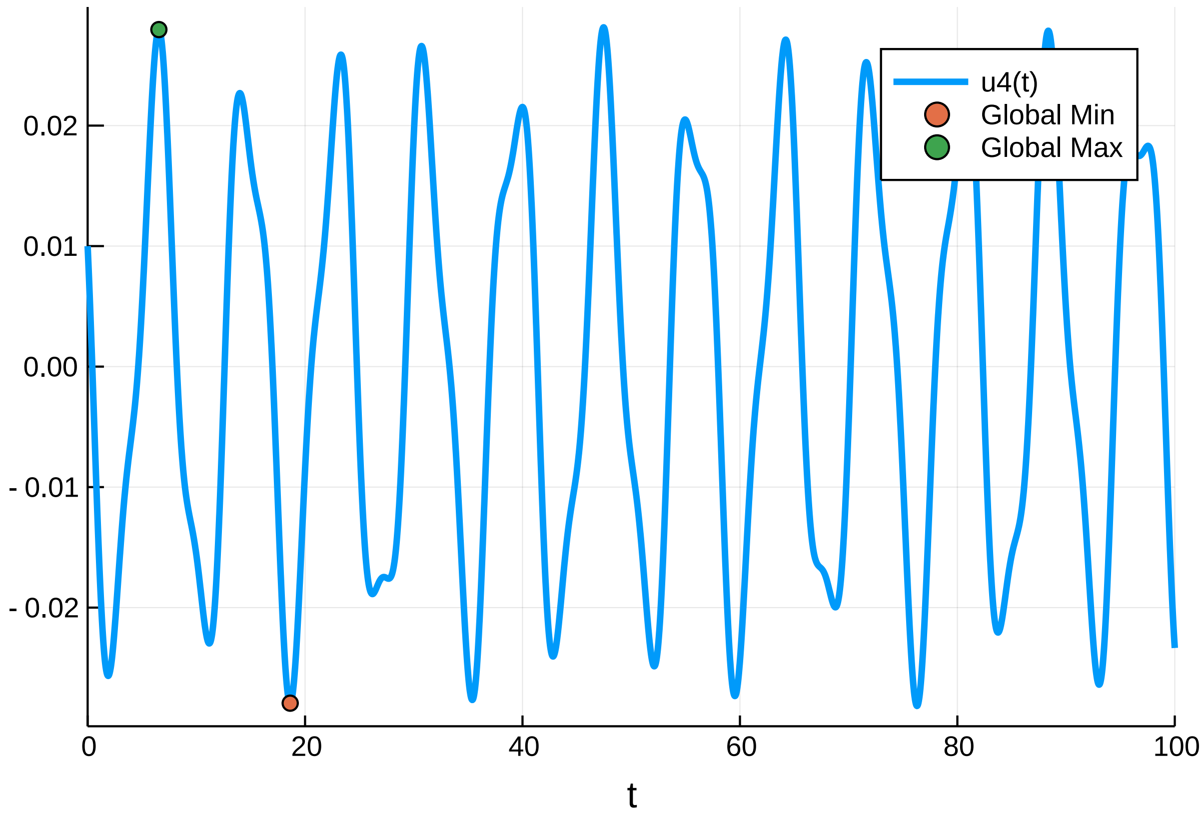
```
-0.027931635264245837 [18.6321] XTOL_REACHED
```

```
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")
```




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

0.1 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("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\external\DiffEqTutorials.jl\src\Project.toml`
```