DifferentialEquations.jl Workshop Exercise Solutions

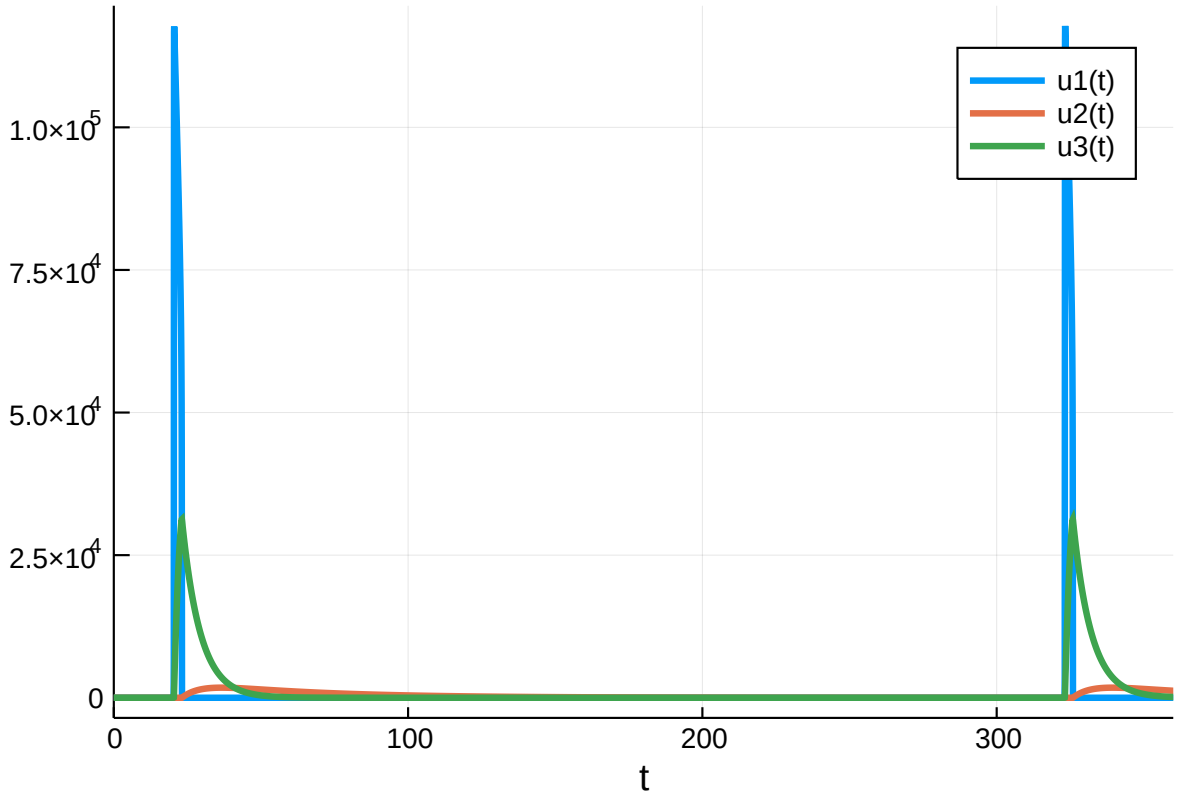
Chris Rackauckas

July 15, 2019

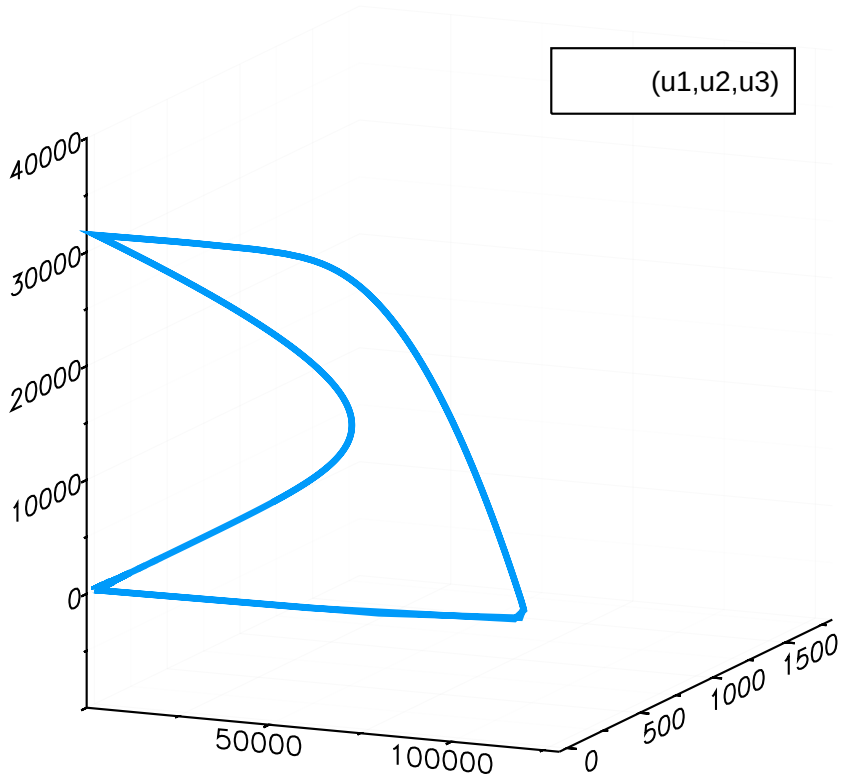
1 Problem 1: Investigating Sources of Randomness and Uncertainty in a Biological System

1.1 Part 1: Simulating the Oregonator ODE model

```
using DifferentialEquations, Plots
function orego(du,u,p,t)
  s,q,w = p
  y1,y2,y3 = u
  du[1] = s*(y2+y1*(1-q*y1-y2))
  du[2] = (y3-(1+y1)*y2)/s
  du[3] = w*(y1-y3)
end
p = [77.27,8.375e-6,0.161]
prob = ODEProblem(orego,[1.0,2.0,3.0],(0.0,360.0),p)
sol = solve(prob)
plot(sol)
```




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



1.2 Part 2: Investigating Stiffness

```
using BenchmarkTools
```

```
prob = ODEProblem(orego,[1.0,2.0,3.0],(0.0,50.0),p)
@btime sol = solve(prob,Tsit5())

3.134 s (8723181 allocations: 920.68 MiB)
retcode: Success
Interpolation: specialized 4th order "free" interpolation
t: 872306-element Array{Float64,1}:
  0.0
  0.016189218375969157
  0.023553748136851134
  0.038180267679755686
  0.050503297498957454
  0.06810643682329323
  0.08676314534460629
  0.11145303081419239
  0.1410587592990862
  0.18104737342146363
  ⋮
 49.99977332573522
 49.9998045817995
 49.999835837885485
 49.99986709399317
 49.99989835012255
 49.99992960627363
 49.999960862446414
 49.9999921186409
 50.0
u: 872306-element Array{Array{Float64,1},1}:
 [1.0, 2.0, 3.0]
 [1.71286, 1.99961, 2.99591]
 [1.83763, 1.99937, 2.99447]
 [1.94804, 1.99883, 2.99191]
 [1.98078, 1.99836, 2.98988]
 [1.99652, 1.99768, 2.98705]
 [2.00125, 1.99696, 2.98409]
 [2.00327, 1.996, 2.98019]
 [2.00461, 1.99484, 2.97555]
 [2.0062, 1.99328, 2.96932]
 ⋮
 [1.00114, 1453.02, 414.832]
 [1.00114, 1453.02, 414.83]
 [1.00114, 1453.02, 414.828]
 [1.00114, 1453.01, 414.826]
 [1.00114, 1453.01, 414.824]
 [1.00114, 1453.01, 414.822]
 [1.00114, 1453.01, 414.82]
 [1.00114, 1453.01, 414.818]
 [1.00088, 1453.01, 414.817]

@btime sol = solve(prob,Rodas5())

544.899 μs (2920 allocations: 175.86 KiB)
retcode: Success
Interpolation: 3rd order Hermite
t: 110-element Array{Float64,1}:
  0.0
  0.019615259849088615
  0.029598267922660175
```

```
  0.047052910887750835
  0.06489945147114441
  0.08933211282883743
  0.12069352237075688
  0.16655179061086892
  0.24088874148540496
  0.39558172278217235
  ⋮
 26.75710407571649
 27.982394888737232
 29.7694090380865
 32.21886344926688
 35.09441917419525
 38.498626966839055
 42.33882931016379
 46.609195570565284
 50.0
u: 110-element Array{Array{Float64,1},1}:
 [1.0, 2.0, 3.0]
 [1.78041, 1.9995, 2.99522]
 [1.89877, 1.99915, 2.99338]
 [1.97458, 1.9985, 2.99044]
 [1.995, 1.99781, 2.98756]
 [2.0016, 1.99686, 2.98368]
 [2.00375, 1.99564, 2.97874]
 [2.00564, 1.99384, 2.97157]
 [2.00859, 1.99093, 2.9601]
 [2.01481, 1.98485, 2.93677]
 ⋮
 [1.00095, 1052.21, 17454.4]
 [1.00079, 1266.47, 14329.7]
 [1.00067, 1490.32, 10747.2]
 [1.0006, 1670.97, 7245.14]
 [1.00057, 1758.48, 4560.57]
 [1.00057, 1757.6, 2636.71]
 [1.00059, 1683.83, 1421.32]
 [1.00064, 1561.03, 715.164]
 [1.00069, 1452.9, 414.722]
```

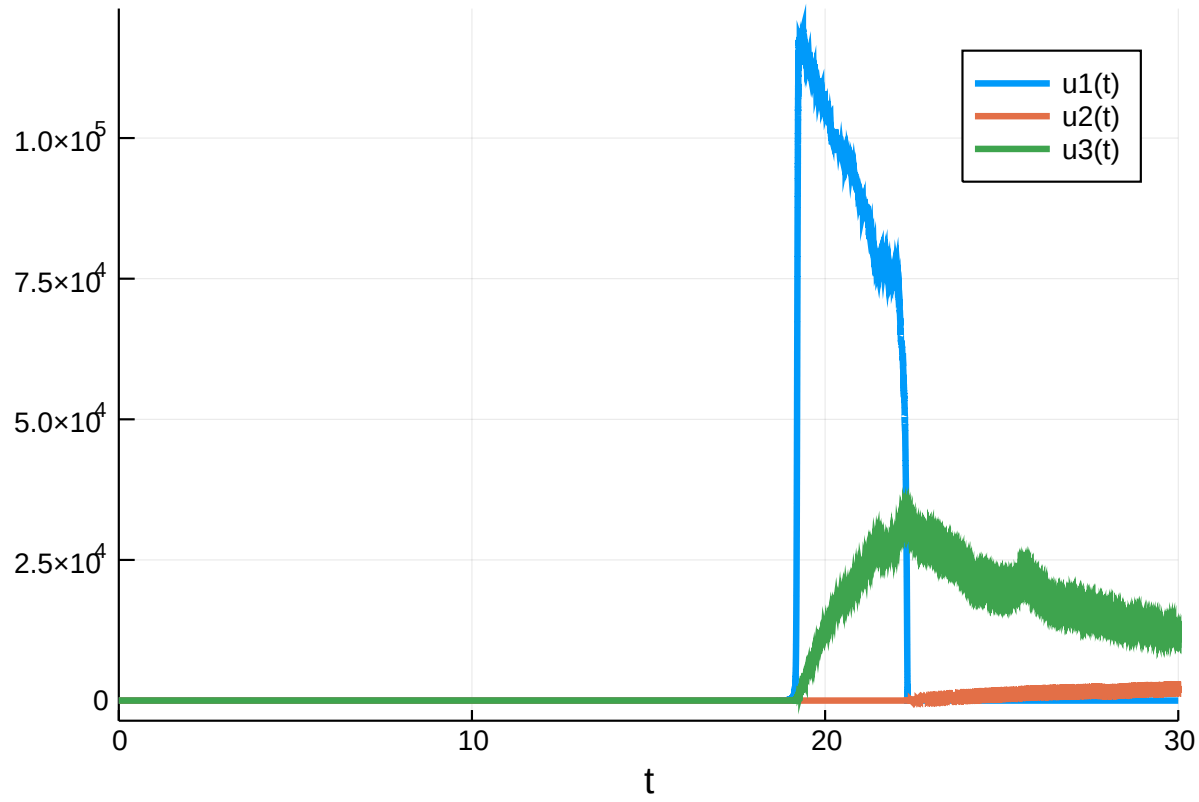
1.3 (Optional) Part 3: Specifying Analytical Jacobians (I)

1.4 (Optional) Part 4: Automatic Symbolicification and Analytical Jacobian Calculations

1.5 Part 5: Adding stochasticity with stochastic differential equations

```
function orego(du,u,p,t)
  s,q,w = p
  y1,y2,y3 = u
  du[1] = s*(y2+y1*(1-q*y1-y2))
  du[2] = (y3-(1+y1)*y2)/s
  du[3] = w*(y1-y3)
end
function g(du,u,p,t)
```

```
  du[1] = 0.1u[1]
  du[2] = 0.1u[2]
  du[3] = 0.1u[3]
end
p = [77.27,8.375e-6,0.161]
prob = SDEProblem(orego,g,[1.0,2.0,3.0],(0.0,30.0),p)
sol = solve(prob,SOSRI())
plot(sol)
```




```
sol = solve(prob,ImplicitRKMil()); plot(sol)
```

```
sol = solve(prob,ImplicitRKMil()); plot(sol)
```

1.6 Part 6: Gillespie jump models of discrete stochasticity

1.7 Part 7: Probabilistic Programming / Bayesian Parameter Estimation with DiffEqBayes.jl + Turing.jl (I)

The data was generated with:

```
function orego(du,u,p,t)
  s,q,w = p
  y1,y2,y3 = u
  du[1] = s*(y2+y1*(1-q*y1-y2))
  du[2] = (y3-(1+y1)*y2)/s
  du[3] = w*(y1-y3)
end
p = [60.0,1e-5,0.2]
prob = ODEProblem(orego,[1.0,2.0,3.0],(0.0,30.0),p)
sol = solve(prob,Rodas5(),abstol=1/10^14,reltol=1/10^14)
```

```
retcode: Success
Interpolation: 3rd order Hermite
t: 48799-element Array{Float64,1}:
  0.0
  0.00013773444266123363
  0.000201070235058078
  0.0003020539265117362
  0.0004030376179653944
  0.000505840575661047
  0.0006092634319273482
  0.0007136360693805303
  0.0008186320050683297
  0.000924255465457595
  ⋮
 29.79488308974022
 29.82256859717493
 29.850254104609636
 29.877939612044344
 29.90562511947905
 29.93331062691376
 29.960996134348466
 29.988681641783174
 30.0
u: 48799-element Array{Array{Float64,1},1}:
 [1.0, 2.0, 3.0]
 [1.00823, 2.0, 2.99995]
 [1.01199, 2.0, 2.99992]
 [1.01796, 1.99999, 2.99988]
 [1.02389, 1.99999, 2.99984]
 [1.02989, 1.99999, 2.9998]
 [1.0359, 1.99999, 2.99976]
 [1.04191, 1.99999, 2.99972]
 [1.04793, 1.99999, 2.99968]
 [1.05395, 1.99998, 2.99964]
 ⋮
 [1.00065, 1541.44, 2708.42]
 [1.00065, 1541.27, 2693.47]
 [1.00065, 1541.08, 2678.6]
 [1.00065, 1540.89, 2663.82]
 [1.00065, 1540.7, 2649.12]
 [1.00065, 1540.49, 2634.49]
 [1.00065, 1540.28, 2619.95]
 [1.00065, 1540.07, 2605.49]
 [1.00065, 1539.98, 2599.6]
```

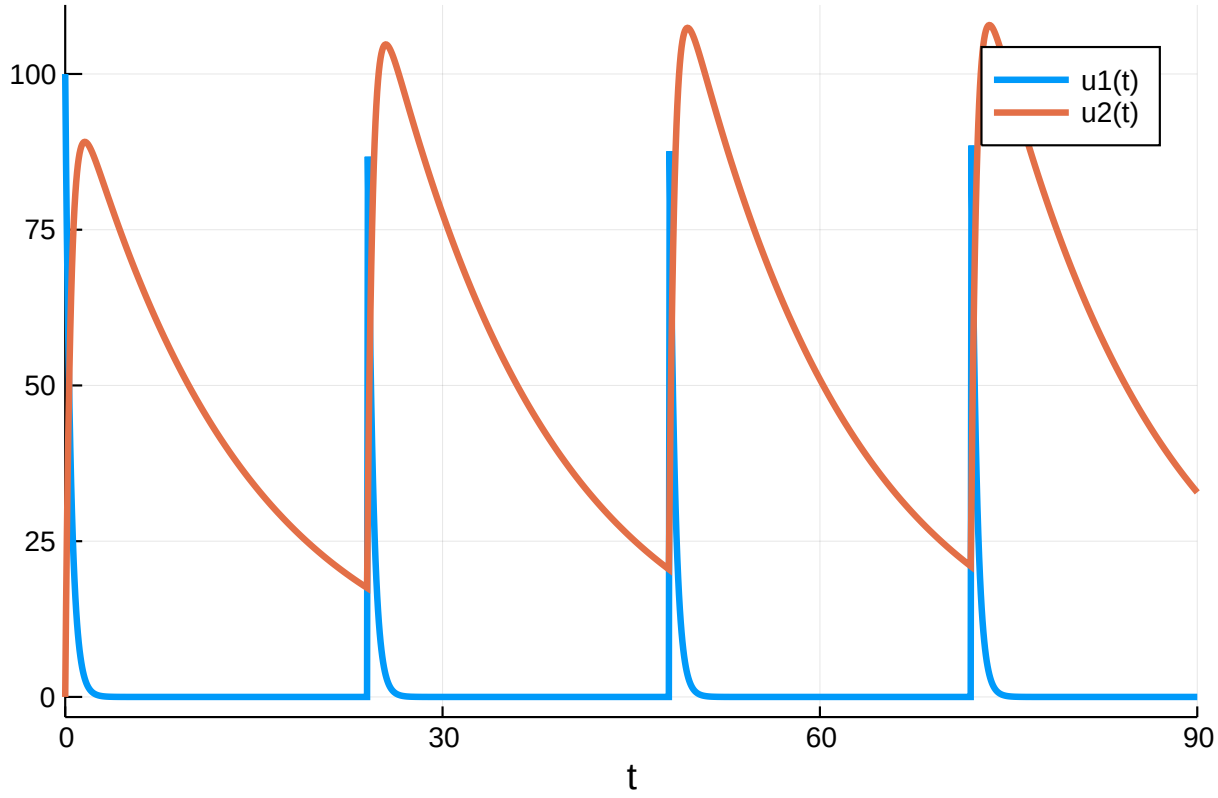
1.8 (Optional) Part 8: Using DiffEqBiological's Reaction Network DSL

2 Problem 2: Fitting Hybrid Delay Pharmacokinetic Models with Automated Responses (B)

2.1 Part 1: Defining an ODE with Predetermined Doses

```
function onecompartment(du,u,p,t)
  Ka,Ke = p
  du[1] = -Ka*u[1]
  du[2] =  Ka*u[1] - Ke*u[2]
end
p = (Ka=2.268,Ke=0.07398)
prob = ODEProblem(onecompartment,[100.0,0.0],(0.0,90.0),p)

tstops = [24,48,72]
condition(u,t,integrator) = t ∈ tstops
affect!(integrator) = (integrator.u[1] += 100)
cb = DiscreteCallback(condition,affect!)
sol = solve(prob,Tsit5(),callback=cb,tstops=tstops)
plot(sol)
```

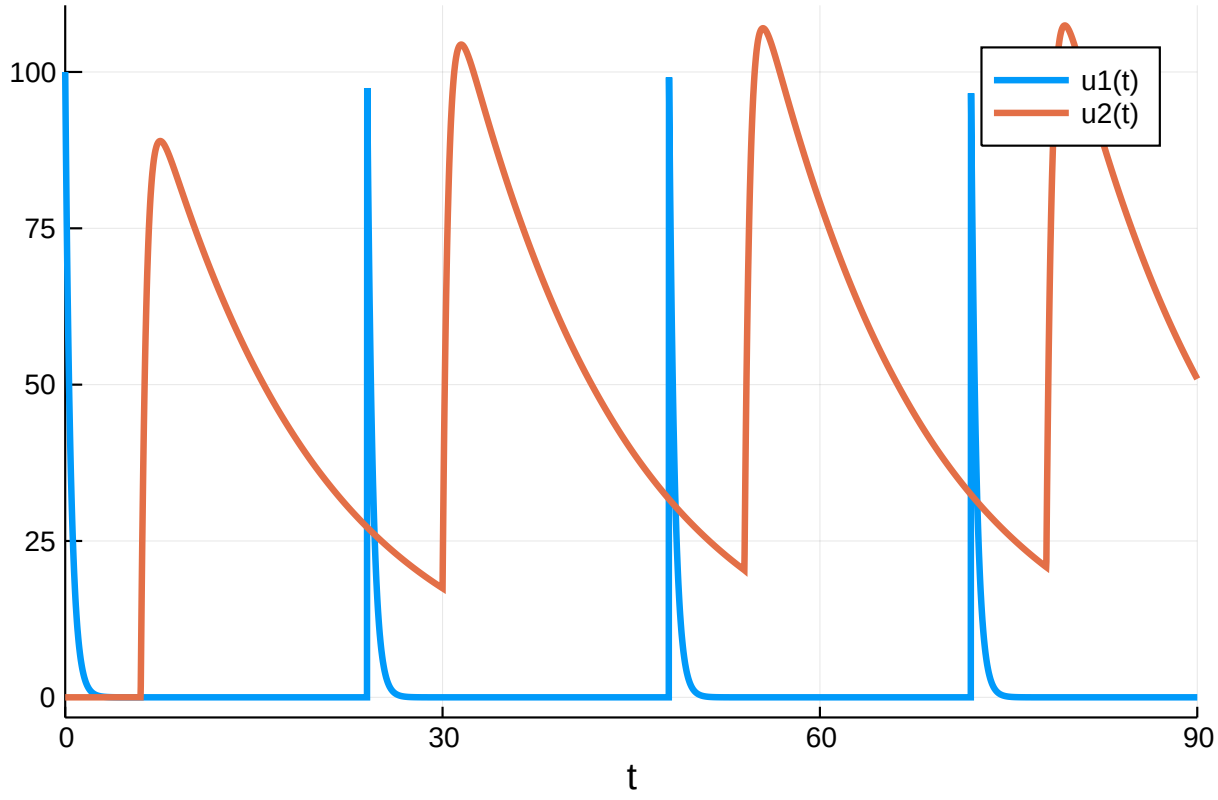


2.2 Part 2: Adding Delays

```
function onecompartment_delay(du,u,h,p,t)
  Ka,Ke,τ = p
  delayed_depot = h(p,t-τ)[1]
  du[1] = -Ka*u[1]
  du[2] =  Ka*delayed_depot - Ke*u[2]
end
p = (Ka=2.268,Ke=0.07398,τ=6.0)
h(p,t) = [0.0,0.0]
prob = DDEProblem(onecompartment_delay,[100.0,0.0],h,(0.0,90.0),p)

tstops = [24,48,72]
condition(u,t,integrator) = t ∈ tstops
affect!(integrator) = (integrator.u[1] += 100)
cb = DiscreteCallback(condition,affect!)
sol = solve(prob,MethodOfSteps(Rosenbrock23()),callback=cb,tstops=tstops)
```

```
plot(sol)
```



2.3 Part 3: Automatic Differentiation (AD) for Optimization (I)

2.4 Part 4: Fitting Known Quantities with DiffEqParamEstim.jl + Optim.jl

The data was generated with

```
p = (Ka = 0.5, Ke = 0.1, τ = 4.0)
```

```
(Ka = 0.5, Ke = 0.1, τ = 4.0)
```