Lorenz Parameter Estimation Benchmarks

finmod, Chris Rackauckas, Vaibhav Dixit

July 5, 2020

1 Estimate the parameters of the Lorenz system from the dataset

Note: If data is generated with a fixed time step method and then is tested against with the same time step, there is a biased introduced since it's no longer about hitting the true solution, rather it's just about retreiving the same values that the ODE was first generated by! Thus this version uses adaptive timestepping for all portions so that way tests are against the true solution.

```
using ParameterizedFunctions, OrdinaryDiffEq, DiffEqParamEstim
using BlackBoxOptim, NLopt, Plots, QuadDIRECT
```

```
Error: ArgumentError: Package QuadDIRECT not found in current path:
- Run `import Pkg; Pkg.add("QuadDIRECT")` to install the QuadDIRECT package
.
```

```
gr(fmt=:png)
```

```
Plots.GRBackend()
```

```
Xiang2015Bounds = Tuple{Float64, Float64}[(9, 11), (20, 30), (2, 3)] # for local
optimizations
xlow_bounds = [9.0,20.0,2.0]
xhigh_bounds = [11.0,30.0,3.0]
LooserBounds = Tuple{Float64, Float64}[(0, 22), (0, 60), (0, 6)] # for global
optimization
GloIniPar = [0.0, 0.5, 0.1] # for global optimizations
LocIniPar = [9.0, 20.0, 2.0] # for local optimization
```

```
3-element Array{Float64,1}:
  9.0
 20.0
  2.0
```

```
g1 = @ode_def LorenzExample begin
  dx = σ*(y-x)
  dy = x*(ρ-z) - y
  dz = x*y - β*z
end σ ρ β
p = [10.0,28.0,2.66] # Parameters used to construct the dataset
r0 = [1.0; 0.0; 0.0]                #[-11.8,-5.1,37.5] PODES Initial values of the
system in space # [0.1, 0.0, 0.0]
```

```
tspan = (0.0, 30.0)                 # PODES sample of 3000 observations over the (0,30)
timespan
prob = ODEProblem(g1, r0, tspan,p)
tspan2 = (0.0, 3.0)                 # Xiang test sample of 300 observations with a
timestep of 0.01
prob_short = ODEProblem(g1, r0, tspan2,p)
```

```
ODEProblem with uType Array{Float64,1} and tType Float64. In-place: true
timespan: (0.0, 3.0)
u0: [1.0, 0.0, 0.0]
```

```
dt = 30.0/3000
tf = 30.0
tinterval = 0:dt:tf
t  = collect(tinterval)
```

```
3001-element Array{Float64,1}:
  0.0
  0.01
  0.02
  0.03
  0.04
  0.05
  0.06
  0.07
  0.08
  0.09
  ⋮
 29.92
 29.93
 29.94
 29.95
 29.96
 29.97
 29.98
 29.99
 30.0
```

```
h = 0.01
M = 300
tstart = 0.0
tstop = tstart + M * h
tinterval_short = 0:h:tstop
t_short = collect(tinterval_short)
```

```
301-element Array{Float64,1}:
 0.0
 0.01
 0.02
 0.03
 0.04
 0.05
 0.06
 0.07
 0.08
 0.09
 ⋮
 2.92
```

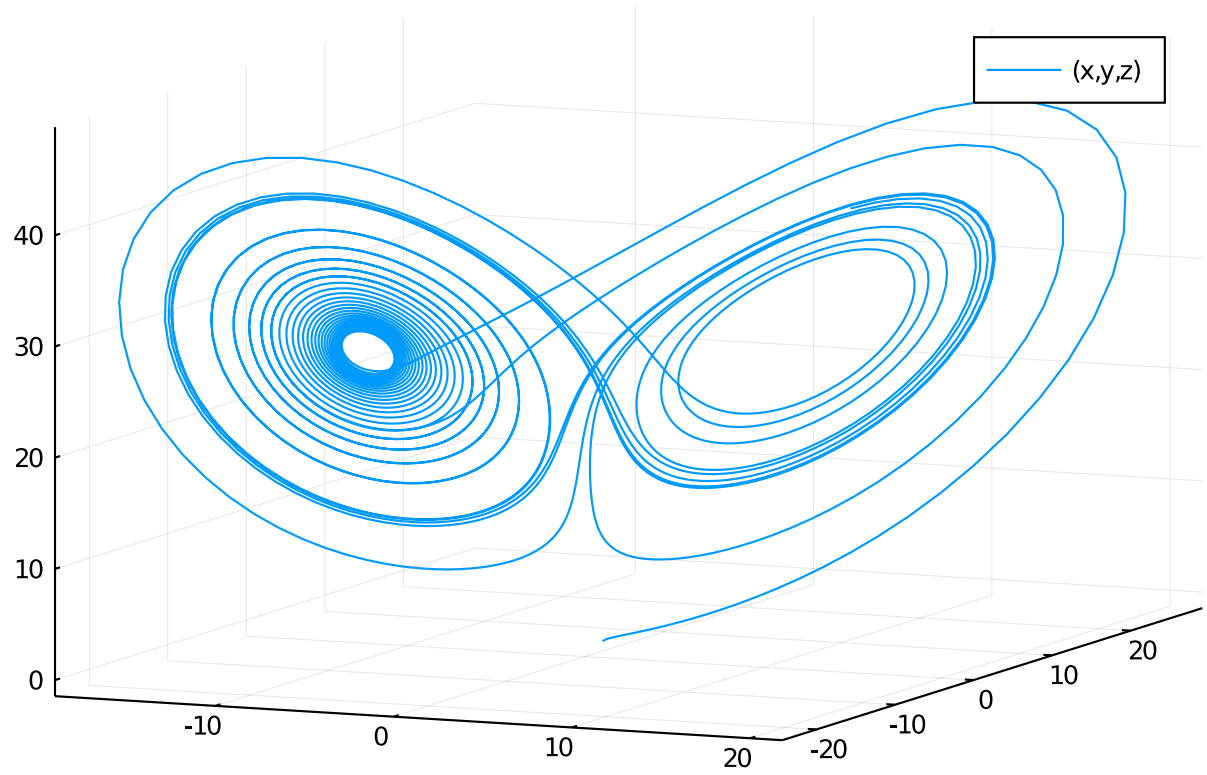
```
 2.93
 2.94
 2.95
 2.96
 2.97
 2.98
 2.99
 3.0
```

```
# Generate Data
data_sol_short = solve(prob_short,Vern9(),saveat=t_short,reltol=1e-9,abstol=1e-9)
data_short = convert(Array, data_sol_short) # This operation produces column major
dataset obs as columns, equations as rows
data_sol = solve(prob,Vern9(),saveat=t,reltol=1e-9,abstol=1e-9)
data = convert(Array, data_sol)
```

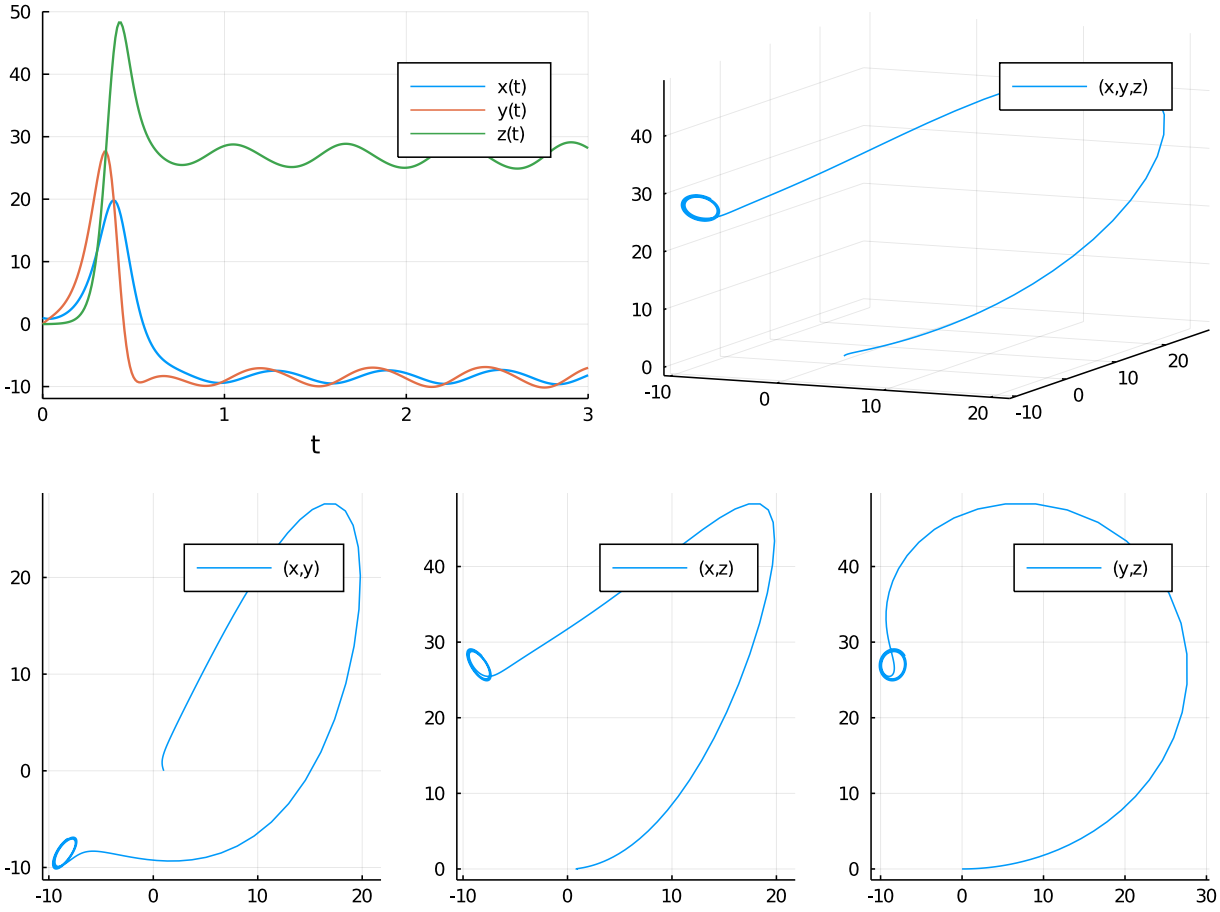
```
3×3001 Array{Float64,2}:
 1.0  0.917924    0.867919    0.84536     ...  13.8987   13.2896  12.5913
 0.0  0.26634     0.51174     0.744654        8.31875   6.7199   5.22868
 0.0  0.00126393  0.00465567  0.00983655     39.19     39.1699  38.904
```

Plot the data

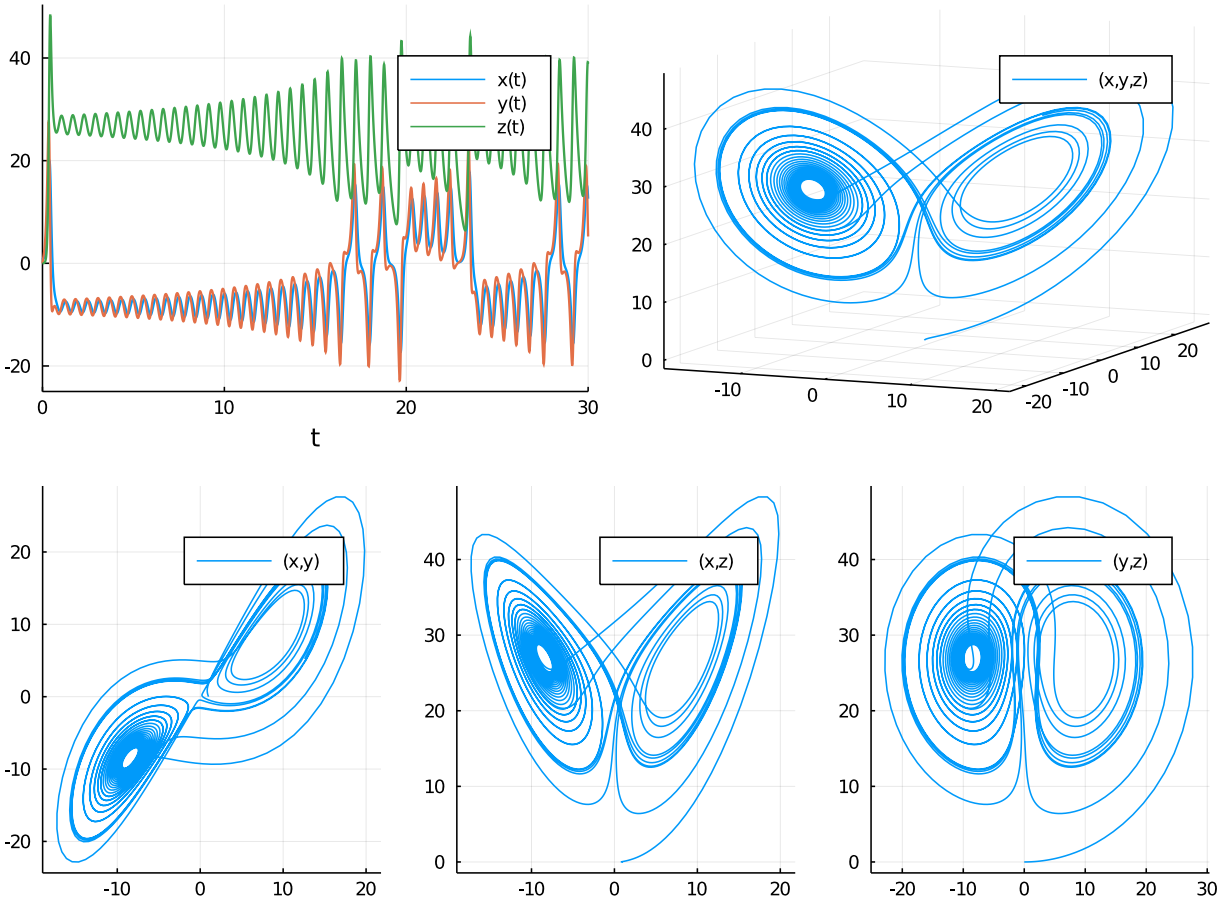
```
plot(data_sol_short,vars=(1,2,3)) # the short solution
plot(data_sol,vars=(1,2,3)) # the longer solution
interpolation_sol = solve(prob,Vern7(),saveat=t,reltol=1e-12,abstol=1e-12)
plot(interpolation_sol,vars=(1,2,3))
```




```
xyzt = plot(data_sol_short, plotdensity=10000,lw=1.5)
xy = plot(data_sol_short, plotdensity=10000, vars=(1,2))
xz = plot(data_sol_short, plotdensity=10000, vars=(1,3))
yz = plot(data_sol_short, plotdensity=10000, vars=(2,3))
xyz = plot(data_sol_short, plotdensity=10000, vars=(1,2,3))
plot(plot(xyzt,xyz),plot(xy, xz, yz, layout=(1,3),w=1), layout=(2,1), size=(800,600))
```




```
xyzt = plot(data_sol, plotdensity=10000,lw=1.5)
xy = plot(data_sol, plotdensity=10000, vars=(1,2))
xz = plot(data_sol, plotdensity=10000, vars=(1,3))
yz = plot(data_sol, plotdensity=10000, vars=(2,3))
xyz = plot(data_sol, plotdensity=10000, vars=(1,2,3))
plot(plot(xyzt,xyz),plot(xy, xz, yz, layout=(1,3),w=1), layout=(2,1), size=(800,600))
```



1.1 Find a local solution for the three parameters from a short data set

```
obj_short =
build_loss_objective(prob_short,Tsit5(),L2Loss(t_short,data_short),tstops=t_short)
res1 = bboptimize(obj_short;SearchRange = LooserBounds, MaxSteps = 7e3)
```

```
Starting optimization with optimizer BlackBoxOptim.DiffEvoOpt{BlackBoxOptim
.FitPopulation{Float64},BlackBoxOptim.RadiusLimitedSelector,BlackBoxOptim.A
daptiveDiffEvoRandBin{3},BlackBoxOptim.RandomBound{BlackBoxOptim.Continuous
RectSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 3240 evals, 3121 steps, improv/step: 0.281 (last = 0.2813), fitn
ess=0.001670629
1.00 secs, 6515 evals, 6397 steps, improv/step: 0.277 (last = 0.2726), fitn
ess=0.000000007

Optimization stopped after 7001 steps and 1.11 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 6318.34
Function evals per second = 6424.84
Improvements/step = 0.27729
Total function evaluations = 7119


Best candidate found: [10.0, 28.0, 2.66]
```

```
Fitness: 0.000000001

# Tolernace is still too high to get close enough

obj_short =
build_loss_objective(prob_short,Tsit5(),L2Loss(t_short,data_short),tstops=t_short,reltol=1e-9)
res1 = bboptimize(obj_short;SearchRange = LooserBounds, MaxSteps = 7e3)

Starting optimization with optimizer BlackBoxOptim.DiffEvoOpt{BlackBoxOptim
.FitPopulation{Float64},BlackBoxOptim.RadiusLimitedSelector,BlackBoxOptim.A
daptiveDiffEvoRandBin{3},BlackBoxOptim.RandomBound{BlackBoxOptim.Continuous
RectSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 2165 evals, 2050 steps, improv/step: 0.281 (last = 0.2815), fitn
ess=0.464525909
1.00 secs, 4330 evals, 4215 steps, improv/step: 0.284 (last = 0.2859), fitn
ess=0.000025481
1.50 secs, 6585 evals, 6470 steps, improv/step: 0.281 (last = 0.2772), fitn
ess=0.000000000

Optimization stopped after 7001 steps and 1.63 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 4284.83
Function evals per second = 4354.60
Improvements/step = 0.28286
Total function evaluations = 7115


Best candidate found: [10.0, 28.0, 2.66]

Fitness: 0.000000000

# With the tolerance lower, it achieves the correct solution in 3.5 seconds.

obj_short =
build_loss_objective(prob_short,Vern9(),L2Loss(t_short,data_short),tstops=t_short,reltol=1e-9,abstol=1
res1 = bboptimize(obj_short;SearchRange = LooserBounds, MaxSteps = 7e3)

Starting optimization with optimizer BlackBoxOptim.DiffEvoOpt{BlackBoxOptim
.FitPopulation{Float64},BlackBoxOptim.RadiusLimitedSelector,BlackBoxOptim.A
daptiveDiffEvoRandBin{3},BlackBoxOptim.RandomBound{BlackBoxOptim.Continuous
RectSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 2308 evals, 2191 steps, improv/step: 0.301 (last = 0.3012), fitn
ess=0.160143611
1.00 secs, 4631 evals, 4515 steps, improv/step: 0.285 (last = 0.2694), fitn
ess=0.000011792
1.50 secs, 6982 evals, 6868 steps, improv/step: 0.288 (last = 0.2928), fitn
ess=0.000000000

Optimization stopped after 7001 steps and 1.53 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 4581.98
Function evals per second = 4656.59
Improvements/step = 0.28729
Total function evaluations = 7115


Best candidate found: [10.0, 28.0, 2.66]

Fitness: 0.000000000
```

```
# With the more accurate solver Vern9 in the solution of the ODE, the convergence is
less efficient!

# Fastest BlackBoxOptim: 3.5 seconds
```

2 Using NLopt

First, the global optimization algorithms

```
obj_short =
build_loss_objective(prob_short,Vern9(),L2Loss(t_short,data_short),tstops=t_short,reltol=1e-9,abstol=1

(::DiffEqParamEstim.DiffEqObjective{DiffEqParamEstim.var"#43#48"{Nothing,Bo
ol,Int64,typeof(DiffEqParamEstim.STANDARD_PROB_GENERATOR),Base.Iterators.Pa
irs{Symbol,Any,Tuple{Symbol,Symbol,Symbol},NamedTuple{(:tstops, :reltol, :a
bstol),Tuple{Array{Float64,1},Float64,Float64}}},DiffEqBase.ODEProblem{Arra
y{Float64,1},Tuple{Float64,Float64},true,Array{Float64,1},Main.##WeaveSandB
ox#328.LorenzExample{Main.##WeaveSandBox#328.var"###ParameterizedDiffEqFunc
tion#348",Main.##WeaveSandBox#328.var"###ParameterizedTGradFunction#349",Ma
in.##WeaveSandBox#328.var"###ParameterizedJacobianFunction#350",Nothing,Not
hing,ModelingToolkit.ODESystem},Base.Iterators.Pairs{Union{},Union{},Tuple{
},NamedTuple{(),Tuple{}}},DiffEqBase.StandardODEProblem},OrdinaryDiffEq.Ver
n9,DiffEqParamEstim.L2Loss{Array{Float64,1},Array{Float64,2},Nothing,Nothin
g,Nothing},Nothing},DiffEqParamEstim.var"#47#53"{DiffEqParamEstim.var"#43#4
8"{Nothing,Bool,Int64,typeof(DiffEqParamEstim.STANDARD_PROB_GENERATOR),Base
.Iterators.Pairs{Symbol,Any,Tuple{Symbol,Symbol,Symbol},NamedTuple{(:tstops
, :reltol, :abstol),Tuple{Array{Float64,1},Float64,Float64}}},DiffEqBase.OD
EProblem{Array{Float64,1},Tuple{Float64,Float64},true,Array{Float64,1},Main
.##WeaveSandBox#328.LorenzExample{Main.##WeaveSandBox#328.var"###Parameteri
zedDiffEqFunction#348",Main.##WeaveSandBox#328.var"###ParameterizedTGradFun
ction#349",Main.##WeaveSandBox#328.var"###ParameterizedJacobianFunction#350
",Nothing,Nothing,ModelingToolkit.ODESystem},Base.Iterators.Pairs{Union{},U
nion{},Tuple{},NamedTuple{(),Tuple{}}},DiffEqBase.StandardODEProblem},Ordin
aryDiffEq.Vern9,DiffEqParamEstim.L2Loss{Array{Float64,1},Array{Float64,2},N
othing,Nothing,Nothing},Nothing}}}) (generic function with 2 methods)
```

```
opt = Opt(:GN_ORIG_DIRECT_L, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Accurate 3.2 seconds

0.831494 seconds (1.50 M allocations: 295.220 MiB, 4.09% gc time)
(7.403132789058194e-18, [10.000000000174282, 28.000000000007077, 2.66000000
00125332], :XTOL_REACHED)
```

```
opt = Opt(:GN_CRS2_LM, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Accurate 3.0 seconds

0.668899 seconds (1.23 M allocations: 241.366 MiB, 2.53% gc time)
(2.946986594657415e-18, [10.00000000005189, 28.000000000017266, 2.660000000
0057276], :XTOL_REACHED)
```

```
opt = Opt(:GN_ISRES, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Accurate to single precision 8.2
seconds
```

```
2.130687 seconds (3.86 M allocations: 758.057 MiB, 3.83% gc time)
(0.0007005593016719326, [9.999185194323559, 28.000559309786777, 2.659815227
28645], :MAXEVAL_REACHED)
```

```
opt = Opt(:GN_ESCH, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Approximatively accurate, good
starting values for local optimization
```

```
2.105291 seconds (3.86 M allocations: 758.057 MiB, 3.04% gc time)
(25.709313175760744, [9.993450814197063, 28.17469327985343, 2.6532388622056
35], :MAXEVAL_REACHED)
```

Next, the local optimization algorithms that could be used after the global algorithms as a check on the solution and its precision. All the local optimizers are started from LocIniPar and with the narrow bounds of the Xiang2015Paper.

```
opt = Opt(:LN_BOBYQA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # 0.1 seconds
```

```
0.023483 seconds (43.63 k allocations: 8.566 MiB)
(2.7676783472649543e-18, [10.000000000052372, 28.000000000022467, 2.6600000
00008028], :SUCCESS)
```

```
opt = Opt(:LN_NELDERMEAD, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.29 sec
```

```
0.080982 seconds (121.60 k allocations: 23.879 MiB, 19.86% gc time)
(2.9041679465906267e-18, [10.00000000005633, 28.00000000003226, 2.660000000
009111], :XTOL_REACHED)
```

```
opt = Opt(:LD_SLSQP, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.21 sec
```

```
0.065828 seconds (169.48 k allocations: 19.029 MiB)
(1.1112284907903994e-15, [9.999999999755289, 28.000000001197094, 2.66000000
00042696], :XTOL_REACHED)

opt = Opt(:LN_COBYLA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 1.84 sec

0.414395 seconds (745.37 k allocations: 146.381 MiB, 4.27% gc time)
(2.9382755008911024e-18, [10.000000000027239, 28.000000000027903, 2.6600000
000065447], :XTOL_REACHED)

opt = Opt(:LN_NEWUOA_BOUND, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.18 sec ROUNDOFF
LIMITED

0.103732 seconds (98.44 k allocations: 19.331 MiB)
(2.0977630616285726e-8, [10.000007614669004, 28.000000291860562, 2.66000050
93015358], :SUCCESS)

opt = Opt(:LN_PRAXIS, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.18 sec

0.272412 seconds (910.20 k allocations: 47.167 MiB)
(22114.883301973412, [10.914376136966279, 22.25851546723328, 2.127172118829
7965], :SUCCESS)

opt = Opt(:LN_SBPLX, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.65 sec

0.153742 seconds (288.74 k allocations: 56.703 MiB)
(2.784069880414099e-18, [10.000000000058924, 28.000000000020663, 2.66000000
0008665], :XTOL_REACHED)

opt = Opt(:LD_MMA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.7 sec
```

```
0.188515 seconds (318.73 k allocations: 62.578 MiB, 11.49% gc time)
(2.513727923789831e-16, [9.999999999464693, 28.000000000538954, 2.659999999
922903], :XTOL_REACHED)

opt = Opt(:LD_LBFGS, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.12 sec

0.024864 seconds (48.63 k allocations: 9.546 MiB)
(1.1160505492326872e-15, [9.999999999753967, 28.00000000119966, 2.660000000
0043185], :SUCCESS)

opt = Opt(:LD_TNEWTON_PRECOND_RESTART, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Accurate 0.15 sec

0.032645 seconds (62.13 k allocations: 12.198 MiB)
(1.1162287428886917e-15, [9.999999999752498, 28.000000001199844, 2.66000000
00043726], :SUCCESS)
```

2.1 Now let's solve the longer version for a global solution

Notice from the plotting above that this ODE problem is chaotic and tends to diverge over time. In the longer version of parameter estimation, the dataset is increased to 3000 observations per variable with the same integration time step of 0.01. Vern9 solver with reltol=1e-9 and abstol=1e-9 has been established to be accurate on the time interval [0,50]

```
# BB with Vern9 converges very slowly. The final values are within the NarrowBounds.
obj = build_loss_objective(prob,Vern9(),L2Loss(t,data),tstops=t,reltol=1e-9,abstol=1e-9)

res1 = bboptimize(obj;SearchRange = LooserBounds, MaxSteps = 4e3) # Default
adaptive_de_rand_1_bin_radiuslimited 33 sec [10.2183, 24.6711, 2.28969]

Starting optimization with optimizer BlackBoxOptim.DiffEvoOpt{BlackBoxOptim
.FitPopulation{Float64},BlackBoxOptim.RadiusLimitedSelector,BlackBoxOptim.A
daptiveDiffEvoRandBin{3},BlackBoxOptim.RandomBound{BlackBoxOptim.Continuous
RectSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 244 evals, 159 steps, improv/step: 0.434 (last = 0.4340), fitnes
s=564351.029993426
1.00 secs, 487 evals, 377 steps, improv/step: 0.363 (last = 0.3119), fitnes
s=532404.315590659
1.51 secs, 737 evals, 623 steps, improv/step: 0.319 (last = 0.2520), fitnes
s=527763.362022145
2.01 secs, 992 evals, 874 steps, improv/step: 0.291 (last = 0.2191), fitnes
s=527763.362022145
2.51 secs, 1239 evals, 1121 steps, improv/step: 0.269 (last = 0.1903), fitn
ess=522998.306595314
3.01 secs, 1486 evals, 1368 steps, improv/step: 0.248 (last = 0.1538), fitn
```

```
ess=522998.306595314
3.51 secs, 1733 evals, 1615 steps, improv/step: 0.230 (last = 0.1336), fitn
ess=507417.954977803
4.02 secs, 1980 evals, 1862 steps, improv/step: 0.218 (last = 0.1336), fitn
ess=507417.954977803
4.52 secs, 2227 evals, 2109 steps, improv/step: 0.205 (last = 0.1134), fitn
ess=505255.622853230
5.02 secs, 2474 evals, 2356 steps, improv/step: 0.197 (last = 0.1296), fitn
ess=487354.771948613
5.52 secs, 2721 evals, 2603 steps, improv/step: 0.188 (last = 0.0972), fitn
ess=487354.771948613
6.02 secs, 2967 evals, 2850 steps, improv/step: 0.179 (last = 0.0850), fitn
ess=487354.771948613
6.52 secs, 3212 evals, 3095 steps, improv/step: 0.168 (last = 0.0408), fitn
ess=487354.771948613
7.02 secs, 3465 evals, 3348 steps, improv/step: 0.160 (last = 0.0672), fitn
ess=487354.771948613
7.52 secs, 3712 evals, 3595 steps, improv/step: 0.153 (last = 0.0486), fitn
ess=487354.771948613
8.02 secs, 3959 evals, 3842 steps, improv/step: 0.145 (last = 0.0405), fitn
ess=472777.941165304

Optimization stopped after 4001 steps and 8.35 seconds
Termination reason: Max number of steps (4000) reached
Steps per second = 479.22
Function evals per second = 493.24
Improvements/step = 0.14150
Total function evaluations = 4118


Best candidate found: [11.6463, 24.5431, 2.30114]

Fitness: 472777.941165304
```

```
#res1 = bboptimize(obj;SearchRange = LooserBounds, Method = :adaptive_de_rand_1_bin,
MaxSteps = 4e3) # Method 32 sec [13.2222, 25.8589, 2.56176]
#res1 = bboptimize(obj;SearchRange = LooserBounds, Method = :dxnes, MaxSteps = 2e3) #
Method dxnes 119 sec  [16.8648, 24.393, 2.29119]
#res1 = bboptimize(obj;SearchRange = LooserBounds, Method = :xnes, MaxSteps = 2e3) #
Method xnes 304 sec  [19.1647, 24.9479, 2.39467]
#res1 = bboptimize(obj;SearchRange = LooserBounds, Method =
:de_rand_1_bin_radiuslimited, MaxSteps = 2e3) # Method 44 sec  [13.805, 24.6054, 2.37274]
#res1 = bboptimize(obj;SearchRange = LooserBounds, Method = :generating_set_search,
MaxSteps = 2e3) # Method 195 sec [19.1847, 24.9492, 2.39412]

# using Evolutionary
# N = 3
# @time result, fitness, cnt = cmaes(obj, N; μ = 3, λ = 12, iterations = 1000) # cmaes(
rastrigin, N; μ = 15, λ = P, tol = 1e-8)

opt = Opt(:GN_ORIG_DIRECT_L, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Fail to converge
```

```
4.322542 seconds (6.61 M allocations: 1.290 GiB, 2.64% gc time)
(470298.7356885679, [7.04665993025209, 23.666102233396032, 1.80660129722654
```

```
62], :XTOL_REACHED)

opt = Opt(:GN_CRS2_LM, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 20000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Hit and miss. converge
approximately accurate values for local opt.91 seconds

40.400807 seconds (61.84 M allocations: 12.071 GiB, 2.53% gc time)
(328858.4530317603, [10.250889980862137, 25.577593556523702, 2.400175444372
311], :MAXEVAL_REACHED)

opt = Opt(:GN_ISRES, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 50000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Approximately accurate within
local bounds

101.081155 seconds (154.60 M allocations: 30.178 GiB, 2.57% gc time)
(416346.70461095043, [10.099110056328932, 28.091535578943954, 2.59216967028
04673], :MAXEVAL_REACHED)

opt = Opt(:GN_ESCH, 3)
lower_bounds!(opt,[0.0,0.0,0.0])
upper_bounds!(opt,[22.0,60.0,6.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 20000)
@time (minf,minx,ret) = NLopt.optimize(opt,GloIniPar) # Approximately accurate

40.397307 seconds (61.84 M allocations: 12.071 GiB, 2.53% gc time)
(535459.5726926867, [2.1447398401747697, 25.488004292843254, 1.045521736870
3187], :MAXEVAL_REACHED)
```

This parameter estimation on the longer sample proves to be extremely challenging for the global optimizers. BlackBoxOptim is best in optimizing the objective function. All of the global algorithms produces final parameter estimates that could be used as starting values for further refinement with the local optimization algorithms.

```
opt = Opt(:LN_BOBYQA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Claims SUCCESS but does not
iterate to the true values.

0.240466 seconds (358.68 k allocations: 71.693 MiB, 5.95% gc time)
(588113.2784337488, [9.862590803788724, 20.581133879885893, 2.0], :SUCCESS)

opt = Opt(:LN_NELDERMEAD, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
```

```
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-9)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Inaccurate final values

20.222272 seconds (30.92 M allocations: 6.036 GiB, 2.54% gc time)
(404754.5095017009, [9.678915633380864, 23.516765371996325, 2.1610738070962
92], :MAXEVAL_REACHED)

opt = Opt(:LD_SLSQP, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # Inaccurate final values

0.115038 seconds (179.34 k allocations: 35.845 MiB)
(591460.7052976544, [9.237552014169724, 20.558585056149155, 2.0333441686041
98], :XTOL_REACHED)
```

No local optimizer can improve the global solution to the true values.

```
obj_short =
build_loss_objective(prob_short,Tsit5(),L2Loss(t_short,data_short),tstops=t_short)
lower = [0.0,0.0,0.0]
upper = [50.0,50.0,50.0]
splits = ([1.0,5.0,15.0],[0,10,20],[0,10,20])
@time root, x0 = analyze(obj_short,splits,lower,upper)

Error: UndefVarError: analyze not defined

minimum(root)

Error: UndefVarError: root not defined

obj = build_loss_objective(prob,Vern9(),L2Loss(t,data),tstops=t,reltol=1e-9,abstol=1e-9)
lower = [0.0,0.0,0.0]
upper = [50.0,50.0,50.0]
splits = ([0,5.0,15.0],[0,15,30],[0,2,5])
@time root, x0 = analyze(obj,splits,lower,upper)

Error: UndefVarError: analyze not defined

minimum(root)

Error: UndefVarError: root not defined
```

3 Conclusion:

1. As expected the Lorenz system is extremely sensitive to initial space values. Starting the integration from `r0 = [0.1,0.0,0.0]` produces convergence with the short sample of 300 observations. This can be achieved by all the global optimizers as well as most of the local optimizers. Instead starting from `r0= [-11.8,-5.1,37.5]`, as in PODES, with the shorter sample shrinks the number of successful algorithms to 3: `BBO`, `:GN_CRS2_LM`and `:LD_SLSQP`. For the longer sample, all the algorithms fail.

2. When trying to hit the real data, having a low enough tolerance on the numerical solution is key. If the numerical solution is too rough, then we can never actually hone in on the true parameters since even with the true parameters we will erroneously induce numerical error. Maybe this could be adaptive?

3. Excessively low tolerance in the numerical solution is inefficient and delays the convergence of the estimation.

4. The estimation method and the global versus local optimization make a huge difference in the timings. Here, BBO always find the correct solution for a global optimization setup. For local optimization, most methods in NLopt, like :LN_BOBYQA, solve the problem in <0.05 seconds. This is an algorithm that can scale a local optimization but we are aiming to scale a global optimization.

5. QuadDIRECT performs very well on the shorter problem but doesn't give very great results for the longer in the Lorenz case, more can be read about the algorithm here.

6. Fitting shorter timespans is easier... maybe this can lead to determining a minimal sample size for the optimizers and the estimator to succeed.

```
using DiffEqBenchmarks
DiffEqBenchmarks.bench_footer(WEAVE_ARGS[:folder],WEAVE_ARGS[:file])
```

3.1 Appendix

These benchmarks are a part of the DiffEqBenchmarks.jl repository, found at: https://github.com/JuliaDi

To locally run this tutorial, do the following commands:

```
using DiffEqBenchmarks
DiffEqBenchmarks.weave_file("ParameterEstimation","LorenzParameterEstimation.jmd")
```

Computer Information:

```
Julia Version 1.4.2
Commit 44fa15b150* (2020-05-23 18:35 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
```

```
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-8.0.1 (ORCJIT, skylake)
Environment:
  JULIA_DEPOT_PATH = /builds/JuliaGPU/DiffEqBenchmarks.jl/.julia
  JULIA_CUDA_MEMORY_LIMIT = 2147483648
  JULIA_PROJECT = @.
  JULIA_NUM_THREADS = 8
```

Package Information:

```
Status: `/builds/JuliaGPU/DiffEqBenchmarks.jl/benchmarks/ParameterEstimation/Project.to
[6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf] BenchmarkTools 0.5.0
[a134a8b2-14d6-55f6-9291-3336d3ab0209] BlackBoxOptim 0.5.0
[593b3428-ca2f-500c-ae53-031589ec8ddd] CmdStan 6.0.6
[ebbdde9d-f333-5424-9be2-dbf1e9acfb5e] DiffEqBayes 2.16.0
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.15.0
[ef61062a-5684-51dc-bb67-a0fcdec5c97d] DiffEqUncertainty 1.4.1
[31c24e10-a181-5473-b8eb-7969acd0382f] Distributions 0.23.4
[bbc10e6e-7c05-544b-b16e-64fede858acb] DynamicHMC 2.1.5
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.6.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.41.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 5.3.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.5.3
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 2.5.0
```