Lorenz Parameter Estimation Benchmarks

finmod, Chris Rackauckas, Vaibhav Dixit

March 23, 2019

1 Estimate the parameters of the Lorenz system from the dataset

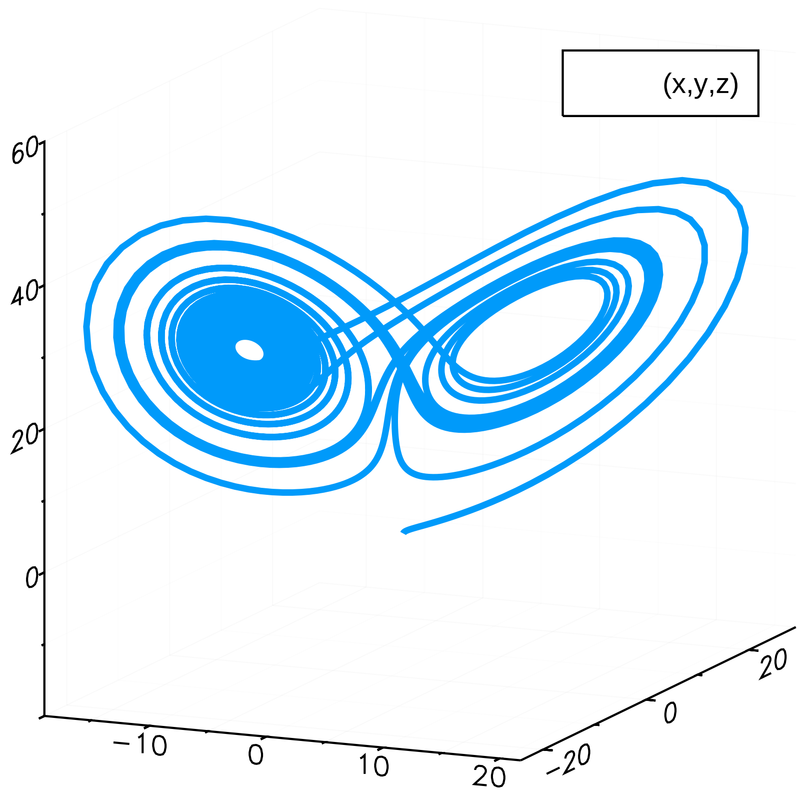
Note: If data is generated with a fixed time step method and then is tested against with the same time step, there is a biased introduced since it's no longer about hitting the true solution, rather it's just about retreiving the same values that the ODE was first generated by! Thus this version uses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

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

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)

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

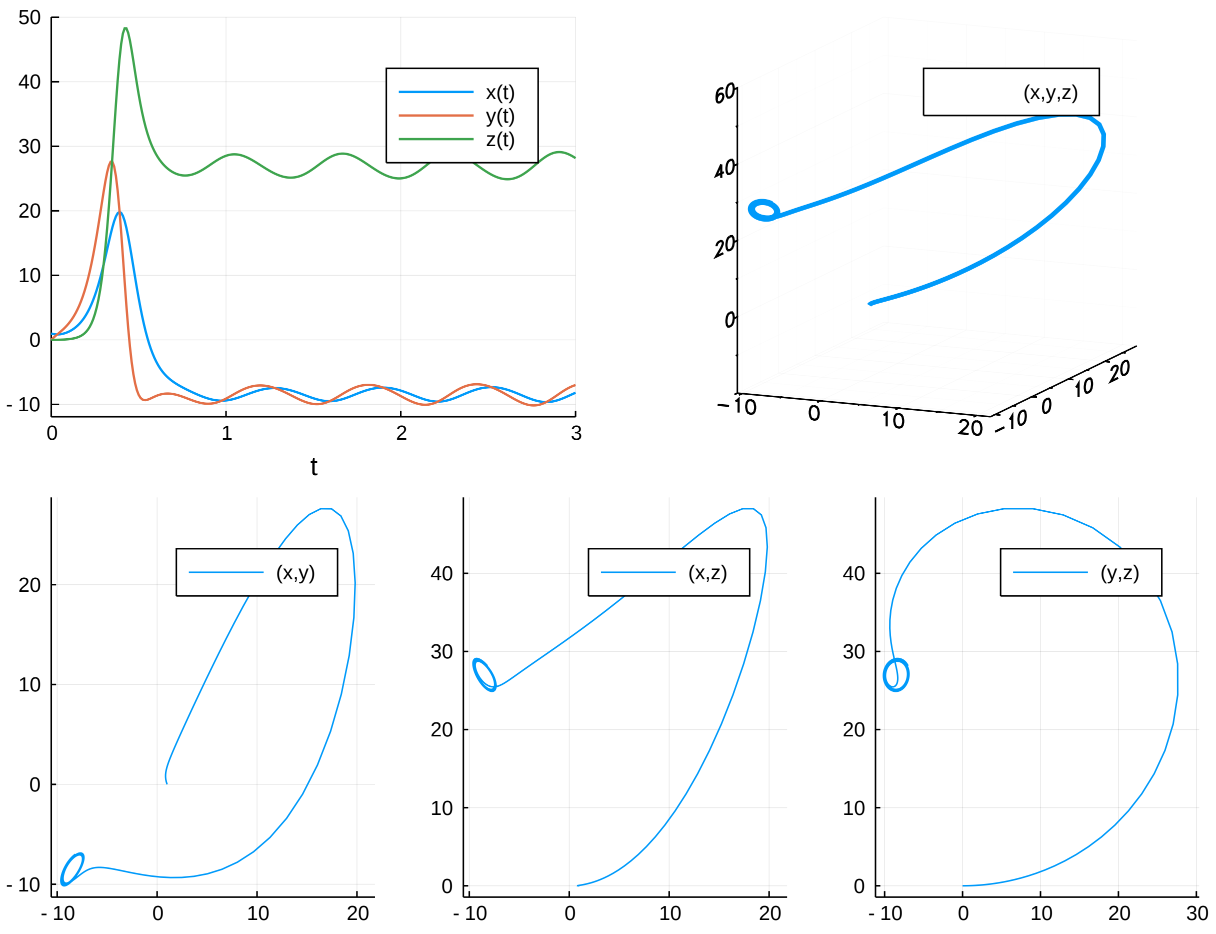
```
h = 0.01
M = 300
tstart = 0.0
tstop = tstart + M * h
tinterval_short = 0:h:tstop
t_short = collect(tinterval_short)

# 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)
```

Plot the data

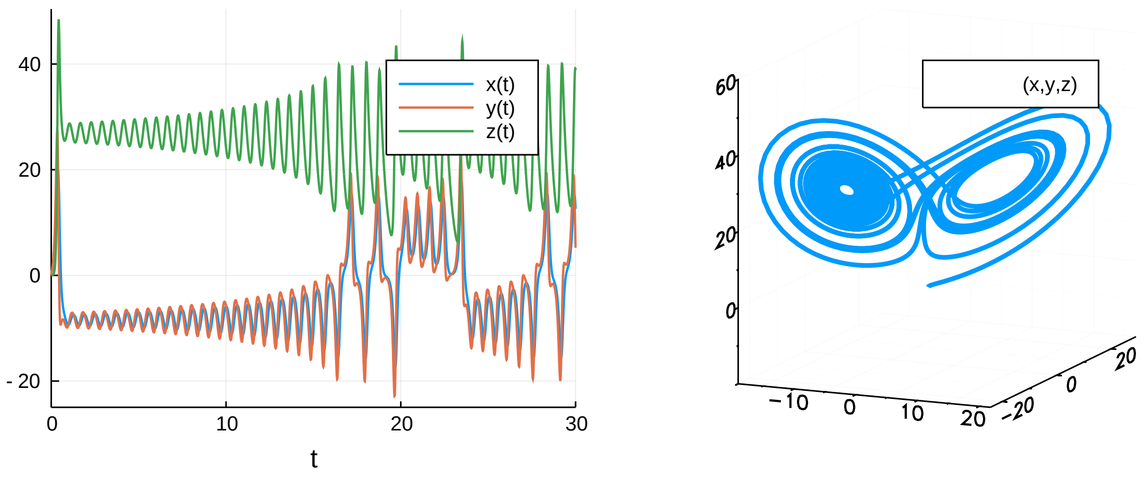
```
plot(data_sol_short,vars=(1,2,3)) # the short solution
plot(data_sol,vars=(1,2,3)) # the longer solution
interpolation_sol = solve(prob,Vern7(),saveat=t,reltol=1e-12,abstol=1e-12)
plot(interpolation_sol,vars=(1,2,3))
```

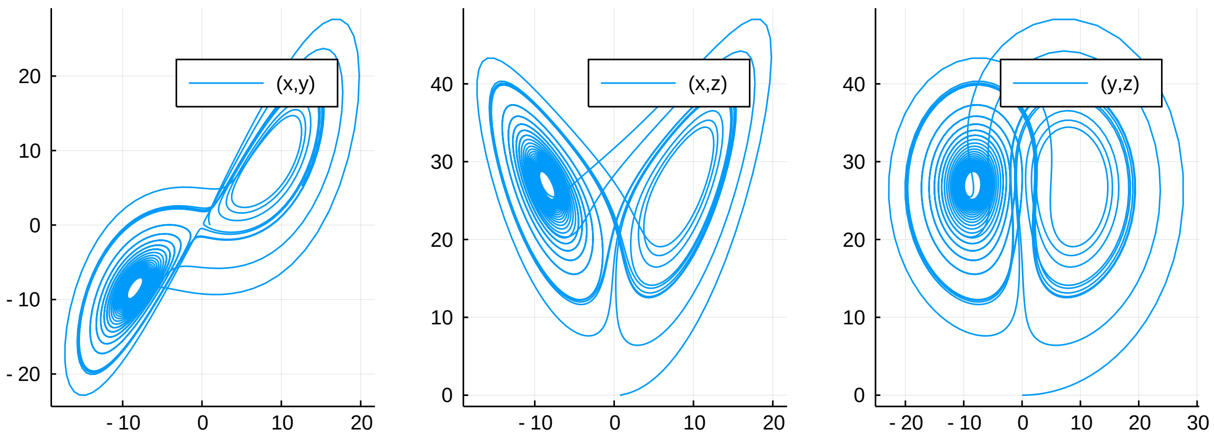


```
xyzt = plot(data_sol_short, plotdensity=10000,lw=1.5)
xy = plot(data_sol_short, plotdensity=10000, vars=(1,2))
xz = plot(data_sol_short, plotdensity=10000, vars=(1,3))
yz = plot(data_sol_short, plotdensity=10000, vars=(2,3))
xyz = plot(data_sol_short, plotdensity=10000, vars=(1,2,3))
plot(plot(xyzt,xyz),plot(xy, 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.RangePerDi
mSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 1834 evals, 1727 steps, improv/step: 0.303 (last = 0.3028), fitn
ess=0.139997853
1.00 secs, 3695 evals, 3588 steps, improv/step: 0.301 (last = 0.2988), fitn
ess=0.000288634
1.50 secs, 5525 evals, 5419 steps, improv/step: 0.301 (last = 0.3015), fitn
ess=0.000000036

Optimization stopped after 7001 steps and 1.934689998626709 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 3618.6675927251827
Function evals per second = 3673.456732109395
Improvements/step = 0.2977142857142857
Total function evaluations = 7107
```

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

Fitness: 0.000000000

# 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.RangePerDi
mSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 1282 evals, 1184 steps, improv/step: 0.327 (last = 0.3269), fitn
ess=21.765334459
1.00 secs, 2567 evals, 2470 steps, improv/step: 0.302 (last = 0.2784), fitn
ess=0.002106898
1.50 secs, 3841 evals, 3746 steps, improv/step: 0.306 (last = 0.3143), fitn
ess=0.000044440
2.00 secs, 5137 evals, 5042 steps, improv/step: 0.300 (last = 0.2832), fitn
ess=0.000000015
2.50 secs, 6429 evals, 6334 steps, improv/step: 0.295 (last = 0.2755), fitn
ess=0.000000000

Optimization stopped after 7001 steps and 2.7651290893554688 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 2531.889027152755
Function evals per second = 2566.2454701722536
Improvements/step = 0.296
Total function evaluations = 7096


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,abst
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.RangePerDi
mSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 1516 evals, 1410 steps, improv/step: 0.309 (last = 0.3085), fitn
ess=5.798501996
1.00 secs, 3079 evals, 2975 steps, improv/step: 0.293 (last = 0.2799), fitn
ess=0.023520006
1.50 secs, 4670 evals, 4566 steps, improv/step: 0.287 (last = 0.2747), fitn
ess=0.000007953
2.00 secs, 6259 evals, 6156 steps, improv/step: 0.285 (last = 0.2811), fitn
ess=0.000000007

Optimization stopped after 7001 steps and 2.2708160877227783 seconds
Termination reason: Max number of steps (7000) reached
```

```
Steps per second = 3083.032588086316
Function evals per second = 3128.39073071921
Improvements/step = 0.28685714285714287
Total function evaluations = 7104


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,abst

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

1.223574 seconds (5.22 M allocations: 497.113 MiB, 9.41% gc time)
(7.389526220984814e-18, [10.0, 28.0, 2.66], :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

1.064757 seconds (4.52 M allocations: 430.199 MiB, 9.43% gc time)
(3.152921741376366e-18, [10.0, 28.0, 2.66], :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

3.166249 seconds (13.41 M allocations: 1.247 GiB, 9.25% gc time)
(0.012735390942528997, [10.0058, 27.9966, 2.66053], :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

3.161282 seconds (13.41 M allocations: 1.247 GiB, 9.08% gc time)
(275.1514055816808, [9.54431, 28.3401, 2.54237], :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

0.038047 seconds (148.87 k allocations: 14.175 MiB, 13.18% gc time)
(2.7636309213683456e-18, [10.0, 28.0, 2.66], :XTOL_REACHED)

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

0.106511 seconds (438.52 k allocations: 41.756 MiB, 9.96% gc time)
(2.767990920967993e-18, [10.0, 28.0, 2.66], :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

0.090204 seconds (336.88 k allocations: 27.855 MiB, 7.03% gc time)
(1.1116793161130144e-15, [10.0, 28.0, 2.66], :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

0.739857 seconds (3.10 M allocations: 294.844 MiB, 9.76% gc time)
(2.8465209128971724e-18, [10.0, 28.0, 2.66], :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

0.095484 seconds (254.81 k allocations: 24.262 MiB, 5.63% gc time)
(2.514234891513546e-9, [10.0, 28.0, 2.66], :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

0.066313 seconds (273.58 k allocations: 26.050 MiB, 8.14% gc time)
(2.771606294121831e-18, [10.0, 28.0, 2.66], :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

0.241277 seconds (1.00 M allocations: 95.515 MiB, 10.69% gc time)
(2.779471170077341e-18, [10.0, 28.0, 2.66], :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

0.258433 seconds (1.11 M allocations: 105.400 MiB, 7.74% gc time)
(2.506182282152645e-16, [10.0, 28.0, 2.66], :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

0.042252 seconds (168.59 k allocations: 16.078 MiB, 13.19% gc time)
(1.116061455658539e-15, [10.0, 28.0, 2.66], :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
```

```
0.056883 seconds (215.41 k allocations: 20.545 MiB, 10.72% gc time)
(1.1161902238149476e-15, [10.0, 28.0, 2.66], :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.RangePerDi
mSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 152 evals, 93 steps, improv/step: 0.387 (last = 0.3871), fitness
=550775.094235542
1.00 secs, 309 evals, 215 steps, improv/step: 0.377 (last = 0.3689), fitnes
s=539541.337465700
1.50 secs, 465 evals, 351 steps, improv/step: 0.376 (last = 0.3750), fitnes
s=536458.693251937
2.01 secs, 622 evals, 507 steps, improv/step: 0.339 (last = 0.2564), fitnes
s=536458.693251937
2.51 secs, 778 evals, 663 steps, improv/step: 0.317 (last = 0.2436), fitnes
s=531754.058414325
3.01 secs, 934 evals, 819 steps, improv/step: 0.281 (last = 0.1282), fitnes
s=440681.199572352
3.51 secs, 1090 evals, 976 steps, improv/step: 0.259 (last = 0.1465), fitne
ss=440681.199572352
4.01 secs, 1248 evals, 1134 steps, improv/step: 0.252 (last = 0.2089), fitn
ess=440681.199572352
4.51 secs, 1405 evals, 1291 steps, improv/step: 0.233 (last = 0.0955), fitn
ess=440681.199572352
5.02 secs, 1562 evals, 1448 steps, improv/step: 0.220 (last = 0.1083), fitn
ess=440681.199572352
5.52 secs, 1721 evals, 1608 steps, improv/step: 0.213 (last = 0.1563), fitn
ess=440681.199572352
6.02 secs, 1878 evals, 1765 steps, improv/step: 0.211 (last = 0.1847), fitn
ess=440681.199572352
6.52 secs, 2035 evals, 1922 steps, improv/step: 0.202 (last = 0.1083), fitn
ess=440681.199572352
7.02 secs, 2192 evals, 2079 steps, improv/step: 0.198 (last = 0.1401), fitn
ess=440681.199572352
7.52 secs, 2354 evals, 2241 steps, improv/step: 0.188 (last = 0.0617), fitn
ess=440681.199572352
8.03 secs, 2513 evals, 2400 steps, improv/step: 0.182 (last = 0.0943), fitn
ess=440681.199572352
8.53 secs, 2673 evals, 2560 steps, improv/step: 0.176 (last = 0.0875), fitn
ess=440681.199572352
9.03 secs, 2831 evals, 2718 steps, improv/step: 0.171 (last = 0.0949), fitn
ess=440681.199572352
```

```
9.53 secs, 2992 evals, 2879 steps, improv/step: 0.167 (last = 0.0994), fitn
ess=440681.199572352
10.03 secs, 3152 evals, 3039 steps, improv/step: 0.162 (last = 0.0750), fit
ness=440681.199572352
10.54 secs, 3309 evals, 3196 steps, improv/step: 0.161 (last = 0.1401), fit
ness=440681.199572352
11.04 secs, 3469 evals, 3356 steps, improv/step: 0.159 (last = 0.1250), fit
ness=440681.199572352
11.54 secs, 3628 evals, 3515 steps, improv/step: 0.156 (last = 0.0881), fit
ness=440681.199572352
12.05 secs, 3786 evals, 3673 steps, improv/step: 0.152 (last = 0.0506), fit
ness=440681.199572352
12.55 secs, 3944 evals, 3831 steps, improv/step: 0.150 (last = 0.1076), fit
ness=440681.199572352
13.05 secs, 4103 evals, 3991 steps, improv/step: 0.147 (last = 0.0750), fit
ness=440681.199572352

Optimization stopped after 4001 steps and 13.084881067276001 seconds
Termination reason: Max number of steps (4000) reached
Steps per second = 305.7727448517745
Function evals per second = 314.3322418333788
Improvements/step = 0.14675
Total function evaluations = 4113


Best candidate found: [12.3374, 27.5316, 2.93028]

Fitness: 440681.199572352
```

```
#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
```

```
6.789238 seconds (25.97 M allocations: 2.235 GiB, 8.04% gc time)
(470298.735717923, [7.04666, 23.6661, 1.8066], :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

63.069816 seconds (243.02 M allocations: 20.915 GiB, 8.09% gc time)
(245455.26702518703, [10.1504, 26.5033, 2.53071], :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

154.538105 seconds (607.55 M allocations: 52.287 GiB, 8.17% gc time)
(378994.41008235904, [10.9487, 24.8049, 2.42306], :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

62.687756 seconds (243.02 M allocations: 20.915 GiB, 8.15% gc time)
(535448.7159269865, [2.15968, 25.4613, 1.04661], :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.

0.355921 seconds (1.36 M allocations: 119.935 MiB, 7.34% gc time)
(588113.2784194095, [9.86259, 20.5811, 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

31.801300 seconds (121.51 M allocations: 10.457 GiB, 8.12% gc time)
(404754.5095397624, [9.67892, 23.5168, 2.16107], :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
```

```
0.068410 seconds (267.27 k allocations: 23.557 MiB, 7.66% gc time)
(575377.5788505444, [9.6232, 23.116, 2.3116], :FAILURE)
```

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)
```

```
1.777075 seconds (5.67 M allocations: 413.293 MiB, 7.38% gc time)
(BoxRoot@[NaN, NaN, NaN], [5.0, 10.0, 10.0])
```

```
minimum(root)
```

```
Box1.0816479008038839e-6@[9.99994, 28.0, 2.66]
```

```
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)
```

```
2.937859 seconds (9.39 M allocations: 735.532 MiB, 7.61% gc time)
(BoxRoot@[NaN, NaN, NaN], [5.0, 15.0, 2.0])
```

```
minimum(root)
```

```
Box528522.0382483905@[26.702, 24.7704, 2.24937]
```

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.