Lorenz Parameter Estimation Benchmarks

finmod, Chris Rackauckas, Vaibhav Dixit

June 15, 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.

```julia
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
.
```

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

```
Plots.GRBackend()
```

```julia
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)
```

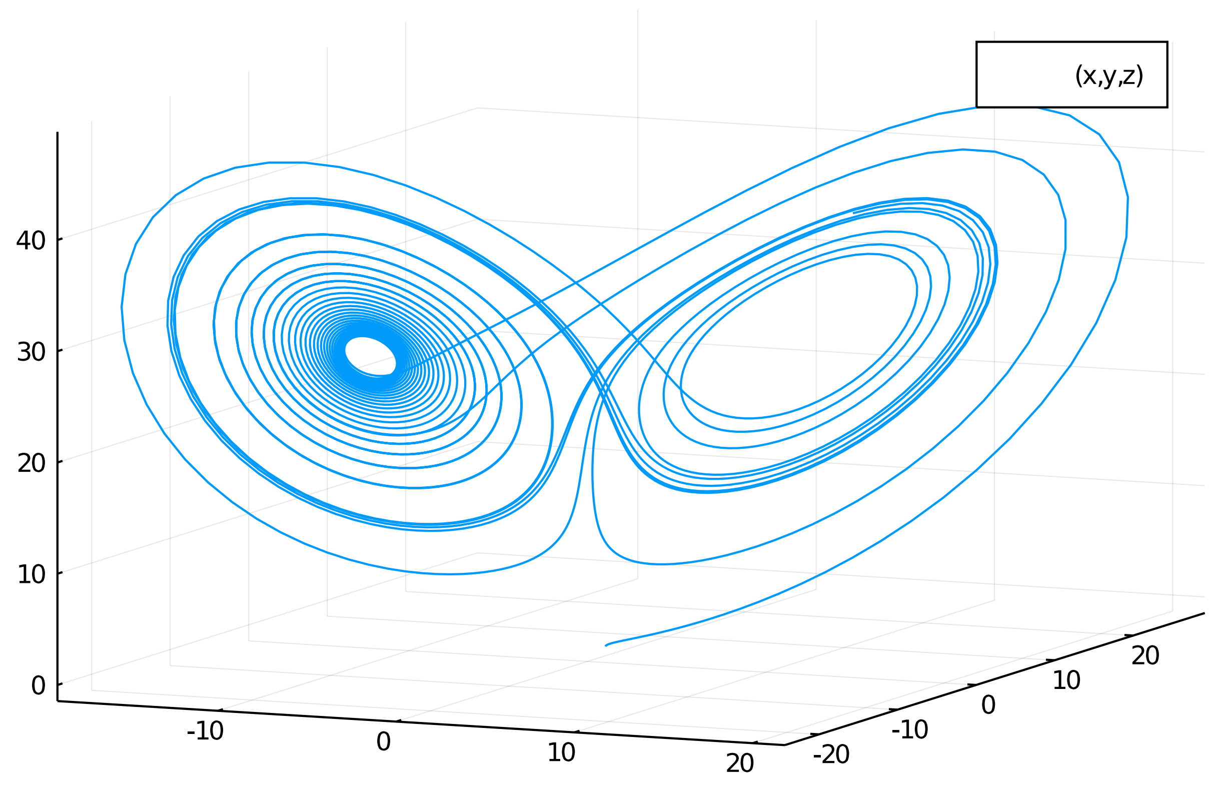
```
301-element Array{Float64,1}:
 0.0
 0.01
 0.02
 0.03
 0.04
 0.05
 0.06
 0.07
 0.08
 0.09
 ⋮
 2.92
 2.93
 2.94
 2.95
 2.96
 2.97
 2.98
 2.99
 3.0
```

```
# 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)
```

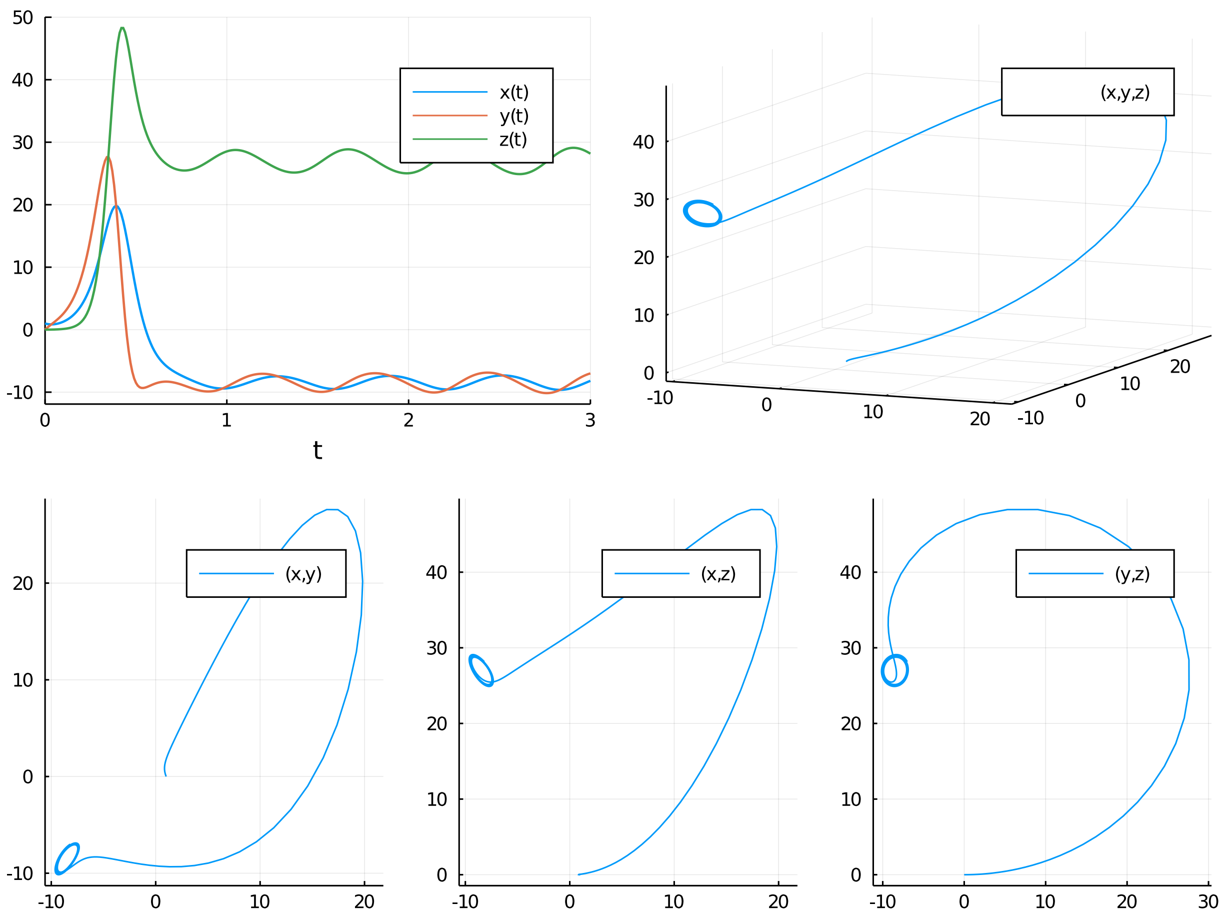
```
3×3001 Array{Float64,2}:
 1.0  0.917924    0.867919    0.84536     …  13.8987    13.2896   12.5913
 0.0  0.26634     0.51174     0.744654        8.31875    6.7199    5.22868
 0.0  0.00126393  0.00465567  0.00983655     39.19      39.1699   38.904
```

Plot the data

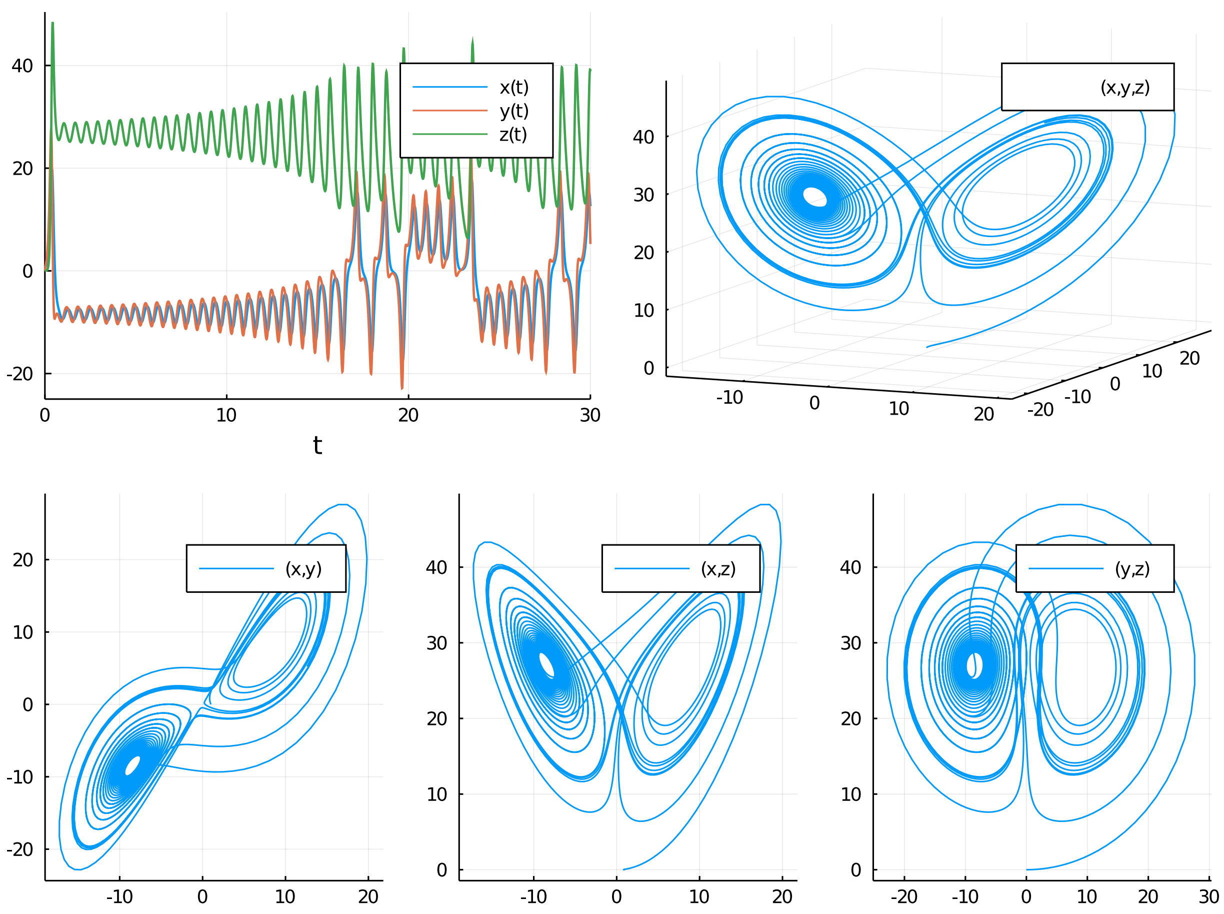
```
plot(data_sol_short,vars=(1,2,3)) # the short solution
plot(data_sol,vars=(1,2,3)) # the longer solution
interpolation_sol = solve(prob,Vern7(),saveat=t,reltol=1e-12,abstol=1e-12)
plot(interpolation_sol,vars=(1,2,3))
```



```
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
0.51 secs, 1883 evals, 1767 steps, improv/step: 0.314 (last = 0.3135), fitn
ess=0.637659424
1.01 secs, 4037 evals, 3922 steps, improv/step: 0.303 (last = 0.2942), fitn
ess=0.000163831
1.51 secs, 6182 evals, 6068 steps, improv/step: 0.289 (last = 0.2628), fitn
ess=0.000000025

Optimization stopped after 7001 steps and 1.70 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 4127.46
Function evals per second = 4194.67
Improvements/step = 0.28686
Total function evaluations = 7115
```

```
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.Continuous
RectSearchSpace}}
0.00 secs, 0 evals, 0 steps
0.50 secs, 1413 evals, 1285 steps, improv/step: 0.301 (last = 0.3012), fitn
ess=46.142208898
1.00 secs, 2759 evals, 2632 steps, improv/step: 0.304 (last = 0.3073), fitn
ess=0.019130338
1.50 secs, 4111 evals, 3985 steps, improv/step: 0.302 (last = 0.2986), fitn
ess=0.000044336
2.00 secs, 5630 evals, 5505 steps, improv/step: 0.300 (last = 0.2947), fitn
ess=0.000000040
2.50 secs, 7120 evals, 6995 steps, improv/step: 0.296 (last = 0.2785), fitn
ess=0.000000000

Optimization stopped after 7001 steps and 2.50 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 2797.43
Function evals per second = 2847.38
Improvements/step = 0.29557
Total function evaluations = 7126


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, 1560 evals, 1445 steps, improv/step: 0.281 (last = 0.2810), fitn
ess=23.265615558
1.00 secs, 3217 evals, 3102 steps, improv/step: 0.275 (last = 0.2698), fitn
ess=0.022385838
1.50 secs, 4732 evals, 4617 steps, improv/step: 0.278 (last = 0.2838), fitn
ess=0.000017798
2.00 secs, 6259 evals, 6145 steps, improv/step: 0.282 (last = 0.2925), fitn
ess=0.000000032

Optimization stopped after 7001 steps and 2.31 seconds
Termination reason: Max number of steps (7000) reached
Steps per second = 3029.81
Function evals per second = 3078.72
Improvements/step = 0.28357
Total function evaluations = 7114


Best candidate found: [10.0, 28.0, 2.66]

Fitness: 0.000000002
```

```
# 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.WeaveSandBox
1.LorenzExample{Main.WeaveSandBox1.var"###ParameterizedDiffEqFunction#334",
Main.WeaveSandBox1.var"###ParameterizedTGradFunction#335",Main.WeaveSandBox
1.var"###ParameterizedJacobianFunction#336",Nothing,Nothing,ModelingToolkit
.ODESystem},Base.Iterators.Pairs{Union{},Union{},Tuple{},NamedTuple{(),Tupl
e{}}},DiffEqBase.StandardODEProblem},OrdinaryDiffEq.Vern9,DiffEqParamEstim.
L2Loss{Array{Float64,1},Array{Float64,2},Nothing,Nothing,Nothing},Nothing},
DiffEqParamEstim.var"#47#53"{DiffEqParamEstim.var"#43#48"{Nothing,Bool,Int6
4,typeof(DiffEqParamEstim.STANDARD_PROB_GENERATOR),Base.Iterators.Pairs{Sym
bol,Any,Tuple{Symbol,Symbol,Symbol},NamedTuple{(:tstops, :reltol, :abstol),
Tuple{Array{Float64,1},Float64,Float64}}},DiffEqBase.ODEProblem{Array{Float
64,1},Tuple{Float64,Float64},true,Array{Float64,1},Main.WeaveSandBox1.Loren
zExample{Main.WeaveSandBox1.var"###ParameterizedDiffEqFunction#334",Main.We
aveSandBox1.var"###ParameterizedTGradFunction#335",Main.WeaveSandBox1.var"#
##ParameterizedJacobianFunction#336",Nothing,Nothing,ModelingToolkit.ODESys
tem},Base.Iterators.Pairs{Union{},Union{},Tuple{},NamedTuple{(),Tuple{}}},D
iffEqBase.StandardODEProblem},OrdinaryDiffEq.Vern9,DiffEqParamEstim.L2Loss{
Array{Float64,1},Array{Float64,2},Nothing,Nothing,Nothing},Nothing}}}) (gen
eric 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
```

```
1.286890 seconds (1.50 M allocations: 295.220 MiB, 4.37% 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
```

```
1.077990 seconds (1.28 M allocations: 252.282 MiB, 4.90% gc time)
(2.8002377596252798e-18, [10.000000000042073, 28.000000000026894, 2.6600000
000069355], :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
```

```
3.215308 seconds (3.86 M allocations: 758.057 MiB, 4.16% gc time)
(0.0002277071209408747, [10.000090994771682, 28.000429012614745, 2.65995251
5572831], :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
```

```
3.137167 seconds (3.86 M allocations: 758.057 MiB, 4.85% gc time)
(191.28763873861587, [10.83940823581734, 27.772247025268822, 2.709226351139
285], :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
```

```
0.032594 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
```

```
0.091877 seconds (121.60 k allocations: 23.879 MiB)
(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
```

```
0.108481 seconds (169.47 k allocations: 19.028 MiB)
(1.1112284907903994e-15, [9.999999999755289, 28.000000001197094, 2.66000000
00042696], :XTOL_REACHED)
```

```julia
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
```

```
0.595512 seconds (745.37 k allocations: 146.381 MiB, 4.94% gc time)
(2.9382755008911024e-18, [10.000000000027239, 28.000000000027903, 2.6600000
000065447], :XTOL_REACHED)
```

```julia
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
```

```
0.152335 seconds (98.44 k allocations: 19.331 MiB)
(2.0977630616285726e-8, [10.000007614669004, 28.000000291860562, 2.66000050
93015358], :SUCCESS)
```

```julia
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
```

```
0.574705 seconds (906.70 k allocations: 46.961 MiB, 5.32% gc time)
(22152.114358269442, [10.914427589375805, 22.252775444390906, 2.12705929148
4556], :SUCCESS)
```

```julia
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
```

```
0.212613 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
```

```
0.265918 seconds (318.73 k allocations: 62.578 MiB, 11.19% 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
```

```
0.039004 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
```

```
0.046710 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, 169 evals, 99 steps, improv/step: 0.374 (last = 0.3737), fitness
=562234.553677235
1.00 secs, 347 evals, 238 steps, improv/step: 0.391 (last = 0.4029), fitnes
s=555881.935013463
1.51 secs, 516 evals, 398 steps, improv/step: 0.359 (last = 0.3125), fitnes
s=535101.906269624
2.01 secs, 687 evals, 566 steps, improv/step: 0.337 (last = 0.2857), fitnes
s=529003.425134825
2.51 secs, 860 evals, 739 steps, improv/step: 0.317 (last = 0.2486), fitnes
s=528950.576668170
3.01 secs, 1041 evals, 920 steps, improv/step: 0.304 (last = 0.2541), fitne
ss=528950.576668170
3.51 secs, 1214 evals, 1093 steps, improv/step: 0.306 (last = 0.3121), fitn
ess=526436.440728998
4.01 secs, 1386 evals, 1265 steps, improv/step: 0.292 (last = 0.2035), fitn
ess=526436.440728998
4.52 secs, 1555 evals, 1434 steps, improv/step: 0.276 (last = 0.1598), fitn
ess=515365.250623169
5.02 secs, 1728 evals, 1607 steps, improv/step: 0.264 (last = 0.1618), fitn
ess=515365.250623169
5.52 secs, 1913 evals, 1793 steps, improv/step: 0.252 (last = 0.1452), fitn
ess=515365.250623169
6.02 secs, 2078 evals, 1958 steps, improv/step: 0.240 (last = 0.1152), fitn
ess=515365.250623169
6.52 secs, 2251 evals, 2131 steps, improv/step: 0.230 (last = 0.1156), fitn
ess=507396.182019638
7.03 secs, 2423 evals, 2303 steps, improv/step: 0.222 (last = 0.1279), fitn
ess=507396.182019638
7.53 secs, 2606 evals, 2486 steps, improv/step: 0.216 (last = 0.1311), fitn
ess=507396.182019638
8.03 secs, 2781 evals, 2661 steps, improv/step: 0.211 (last = 0.1486), fitn
ess=507396.182019638
8.53 secs, 2953 evals, 2833 steps, improv/step: 0.209 (last = 0.1802), fitn
ess=507396.182019638
9.03 secs, 3124 evals, 3004 steps, improv/step: 0.206 (last = 0.1462), fitn
ess=507396.182019638
9.53 secs, 3298 evals, 3178 steps, improv/step: 0.205 (last = 0.1897), fitn
ess=501141.301532982
10.03 secs, 3481 evals, 3361 steps, improv/step: 0.202 (last = 0.1475), fit
ness=501141.301532982
10.54 secs, 3653 evals, 3533 steps, improv/step: 0.198 (last = 0.1279), fit
ness=501141.301532982
11.04 secs, 3824 evals, 3704 steps, improv/step: 0.197 (last = 0.1754), fit
ness=490412.130280500
11.54 secs, 3998 evals, 3878 steps, improv/step: 0.194 (last = 0.1264), fit
ness=490412.130280500
```

```
Optimization stopped after 4001 steps and 11.87 seconds
Termination reason: Max number of steps (4000) reached
Steps per second = 337.17
Function evals per second = 347.28
Improvements/step = 0.19325
Total function evaluations = 4121


Best candidate found: [6.43452, 23.442, 1.7564]

Fitness: 486670.785425578
```

```
#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
```

```
6.133466 seconds (6.61 M allocations: 1.290 GiB, 3.65% 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
```

```
57.469358 seconds (61.84 M allocations: 12.071 GiB, 3.57% gc time)
(78220.13384518395, [10.621690765207925, 27.406365084365717, 2.716678720992
908], :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
```

```
143.965091 seconds (154.60 M allocations: 30.178 GiB, 3.53% gc time)
(509355.94460218796, [8.872456535588045, 22.716494992132247, 2.191472433058
737], :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
```

```
57.221938 seconds (61.84 M allocations: 12.071 GiB, 3.52% gc time)
(532374.5181280716, [10.71850867833877, 21.98483896477884, 2.21970430174583
9], :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.
```

```
0.320784 seconds (358.68 k allocations: 71.693 MiB)
(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
```

```
29.430585 seconds (30.92 M allocations: 6.036 GiB, 3.48% 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
```

```
0.179333 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.

Using QuadDIRECT

```
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) Xeon(R) CPU @ 2.30GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-8.0.1 (ORCJIT, haswell)
Environment:
  JULIA_CUDA_MEMORY_LIMIT = 536870912
  JULIA_DEPOT_PATH = /builds/JuliaGPU/DiffEqBenchmarks.jl/.julia
  JULIA_PROJECT = @.
  JULIA_NUM_THREADS = 4
```

Package Information:

```
Status: `/builds/JuliaGPU/DiffEqBenchmarks.jl/Project.toml`
[28f2ccd6-bb30-5033-b560-165f7b14dc2f] ApproxFun 0.11.14
[a134a8b2-14d6-55f6-9291-3336d3ab0209] BlackBoxOptim 0.5.0
[a93c6f00-e57d-5684-b7b6-d8193f3e46c0] DataFrames 0.21.2
[2b5f629d-d688-5b77-993f-72d75c75574e] DiffEqBase 6.38.3
[ebbdde9d-f333-5424-9be2-dbf1e9acfb5e] DiffEqBayes 2.15.0
[eb300fae-53e8-50a0-950c-e21f52c2b7e0] DiffEqBiological 4.3.0
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.21.0
[c894b116-72e5-5b58-be3c-e6d8d4ac2b12] DiffEqJump 6.9.1
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.14.1
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.8.0
[ef61062a-5684-51dc-bb67-a0fcdec5c97d] DiffEqUncertainty 1.4.1
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.14.0
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.21.2
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.6.1
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.6.0
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.8.0
[54ca160b-1b9f-5127-a996-1867f4bc2a2c] ODEInterface 0.4.6
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.7.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.41.0
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 5.3.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.4.0
[b4db0fb7-de2a-5028-82bf-5021f5cfa881] ReactionNetworkImporters 0.1.5
[f2c3362d-daeb-58d1-803e-2bc74f2840b4] RecursiveFactorization 0.1.2
[9672c7b4-1e72-59bd-8a11-6ac3964bc41f] SteadyStateDiffEq 1.5.1
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 4.2.3
[a759f4b9-e2f1-59dc-863e-4aeb61b1ea8f] TimerOutputs 0.5.6
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.4
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils
```

```
[d6f4376e-aef5-505a-96c1-9c027394607a] Markdown
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg
[9a3f8284-a2c9-5f02-9a11-845980a1fd5c] Random
```