CSE/ECE 848 Introduction to Evolutionary Computation

Module 2, Lecture 7, Part 2b
Nelder-Mead and Evolutionary Computation for Function Optimization

Erik D. Goodman, Executive Director
BEACON Center for the Study of Evolution in Action
Professor, ECE, ME, and CSE

So, let's make the function a little harder

- Let's have 10 "design" or independent variables,

  so we're minimizing a 10-D hypersphere
- Let's again compare the default GA in Pymoo with the default Nelder-Mead
- The GA code is on the next two slides
- This time, we're specifying the problem locally, not using a "canned" Pymoo function
- We're using elementwise evaluation; if wanted to run in parallel, could use a vector evaluation format provided in Pymoo
- (See the Pymoo documentation, www.pymoo.org)

```
import autograd.numpy as anp
from pymoo.algorithms.so_genetic_algorithm import GA
from pymoo.optimize import minimize
from pymoo.model.problem import Problem


class MyProblem(Problem):
    def __init__(self):
        # define lower and upper bounds -  1d array with length equal to number
of variables
        xl = -5 * anp.ones(10)
        xu = 5 * anp.ones(10)
        super().__init__(n_var=10, n_obj=1, n_constr=0, xl=xl, xu=xu,
evaluation_of="auto", elementwise_evaluation=True)

    def _evaluate(self, x, out, *args, **kwargs):
        f = anp.sum(anp.power(x, 2))
        out["F"] = f
```

```
problem = MyProblem()
algorithm = GA(
    pop_size=100,
    eliminate_duplicates=True)

res = minimize(problem,
               algorithm,
               termination=('n_gen', 300),
               seed=1,
               verbose=True)

print("Best solution found: \nX = %s\nF = %s" % (res.X, res.F))
```

GA on 10-D sphere function

...

293	29300	1.20393E-06	1.20415E-06
294	29400	1.20367E-06	1.20410E-06
295	29500	1.20328E-06	1.20405E-06
296	29600	1.20303E-06	1.20397E-06
297	29700	1.20303E-06	1.20390E-06
298	29800	1.20299E-06	1.20380E-06
299	29900	1.20299E-06	1.20369E-06
300	30000	1.20299E-06	1.20353E-06

Best solution found:

X = [-1.96555105e-06 6.50299341e-06 -2.29020756e-06 1.70299072e-06
-1.08051584e-05 -1.09644576e-04 -4.88548339e-06 1.05006710e-03
9.55270706e-06 -2.96712477e-04]

F = [1.20298731e-06]

THAT was harder: 30,000 evaluations to find this solution!

Setup of Nelder-Mead for 10-D sphere function

Import …

```
class MyProblem(Problem):
    def __init__(self):
        xl = -5 * anp.ones(10)
        xu = 5 * anp.ones(10)
        super().__init__(n_var=10, n_obj=1, n_constr=0, xl=xl, xu=xu,
evaluation_of="auto", elementwise_evaluation=True)

    def _evaluate(self, x, out, *args, **kwargs):
        f = anp.sum(anp.power(x, 2))
        out["F"] = f
```

```
problem = MyProblem()
algorithm = NelderMead()

res = minimize(problem,
               algorithm,
               seed=1,
               verbose=True)

print("Best solution found: \nX = %s\nF = %s" % (res.X, res.F))
```

How does this program do on the 10-D sphere function?

...

311 | 582 | 3.36123E-06 | 4.79421E-06

312 | 584 | 3.18665E-06 | 4.50355E-06

313 | 586 | 3.18665E-06 | 4.24182E-06

314 | 588 | 3.18665E-06 | 3.99305E-06

315 | 589 | 3.18665E-06 | 3.79748E-06

316 | **590** | 3.18665E-06 | 3.63426E-06

Best solution found:

X = [6.95873765e-05 -9.61112127e-05 1.98275704e-04 1.80328707e-04
-8.00314633e-04 1.31184335e-03 1.52014831e-04 -4.80700280e-04
-1.79236617e-04 -6.73048171e-04]

F = [3.18664884e-06]

SO, the Nelder-Mead still won easily, because the function is so easy, even though the design space is now somewhat larger! It got similar accuracy in 590 evaluations to what the GA got in 30,000.

ATTENTION: Problem Size and Scaling

- Many rapidly converging methods (like Nelder-Mead) do not scale well with problem dimension
- Usually, in real-world problems, almost all computer time is spent evaluating the fitness function of each individual, BUT
- As number of design variables increases, many methods begin to take a lot of time (and memory, sometimes) to take each step
- Let's see how this works by increasing the sphere function to 200 design variables, instead of 10
- We'll use the default GA parameters, except setting population size to 300 for this more difficult problem

Nelder-Mead on 200-D sphere function

- 2.07952736e-04 -1.53824512e-04 4.70911797e-05 1.65847875e-04
- -7.32534522e-05 -6.50292693e-06 -7.19644428e-06 -6.87565468e-05
- 1.24738909e-04 -3.54438781e-05 -2.05630363e-04 -7.09836421e-06
- -2.06858891e-06 -3.48157469e-05 -8.11902632e-05 3.05884108e-05
- -7.84850801e-05 3.72672375e-05 -8.93091189e-05 -1.96583393e-04
- 9.07998247e-05 -2.76919509e-05 -6.95309305e-05 1.99867322e-05]
- F = [1.42098525e-06]

- Process finished with exit code 0

This run took Nelder-Mead **3 hours to do 180,645 function evaluations**… almost all of that time is spent in calculating the points to sample, since the fitness function is very fast to calculate.

GA on 200-D sphere function

- With population size 300, GA reached same fitness level, <1.42E-6, in 385,200 function evaluations, **in 90 seconds**
- So although Nelder-Mead still required fewer evaluations, it took 120 times as much clock time to reach the solution
- If the fitness function were more time-consuming to compute, Nelder-Mead might still win, with fewer evaluations
- But the **GA is scaling better (often sublinear)**, with smaller increase in evaluations with the problem size:

# Vars	GA	N-M
10	30,000	590
200	385,000	180,645
(20 X)	(~13 X)	(~300 X)

- Imagine the comparison if we ran a 1000-D sphere!

What's the Lesson Here?

- Even though a GA can be made to work on many problems, you shouldn't use it if the problem is one that can be solved efficiently by a sequential algorithm (not using a population)… that will probably be faster, as long as it scales well enough to handle YOUR problem
- Characteristics that mean "a GA might be better" include:
  - Many design variables
  - Multimodality
  - Non-Convexity
  - Discontinuity
  - Non-differentiability
  - Mixed real/integer design variables
  - Combinatorial (finite # of choices)