CSE/ECE 848
Introduction to Evolutionary Computation

Module 2, Lecture 7, Part 2a
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

Practical Use of Optimization Algorithms

- Today, we'll use an example package, Pymoo, for some work on optimization. Find Pymoo and its documentation at www.pymoo.org
- Although called Pymoo (Python Multi-Objective Optimization), it also provides many single-objective optimization algorithms and example problems
- It is open source, so Python programmers can easily replace any operators or algorithms with their own designs
- It was written by MSU Ph.D. student Julian Blank, whom we thank for releasing and supporting it!

Nelder-Mead and Evolutionary Computation for Function Optimization

- Prof. Deb introduced the Nelder-Mead search algorithm in Lecture 2
- Here, we'll use it, as implemented in the Python package Pymoo, to compare with a genetic algorithm running in the same package, on a variety of function optimization problems
- We'll begin with a "classical" optimization problem, minimization of a sphere function, first with 2 variables (really a sphere), then with 10 variables (a hypersphere)
- (I ran it in the PyCharm environment, installed with Anaconda)

Setup for GA Run in Pymoo

```
from pymoo.algorithms.so_genetic_algorithm import GA
from pymoo.factory import get_problem
from pymoo.optimize import minimize

problem = get_problem("go-sphere")
```

Getting test problem from library

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

Specifying algorithm to use and two parameters set to other than default

```
res = minimize(problem,
            algorithm,
            seed=1,
            verbose=False)
```

Specifying solution technique as using above-defined problem and algorithm; giving seed for random number generator

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

What’s Happening Behind the Scenes?

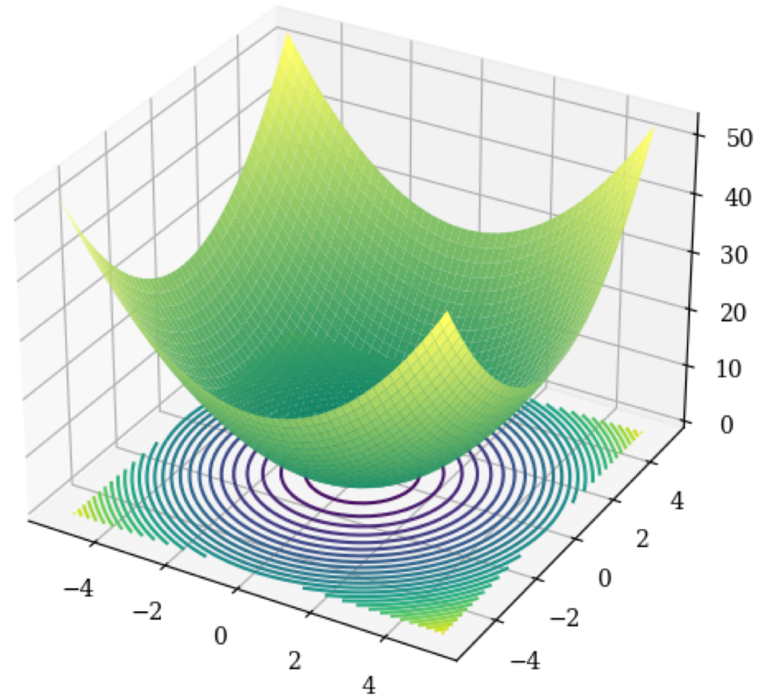
- Could have specified lots more parameters for the GA, including:
  - sampling=
  - selection=
  - crossover=
  - mutation=
  - survival=
  - n_offsprings=
  - display=
  - **kwargs (the keyword ”dictionary”)

GA **Defaults** in Pymoo

- Initialization: random
- Tournament selection, default tourneysize 2 (fairly weak selection pressure)
- Survival: **μ** + **λ** (top **μ** of (**μ** + **λ**) (parents + children) survive)
- Crossover: SBX on reals, at 90% crossovers, eta = 20
- Mutation: Polynomial mutation, at 100%

- You can change any of these easily in the calls

Fitness Landscape of the 2-D Sphere Function We Are to Minimize:



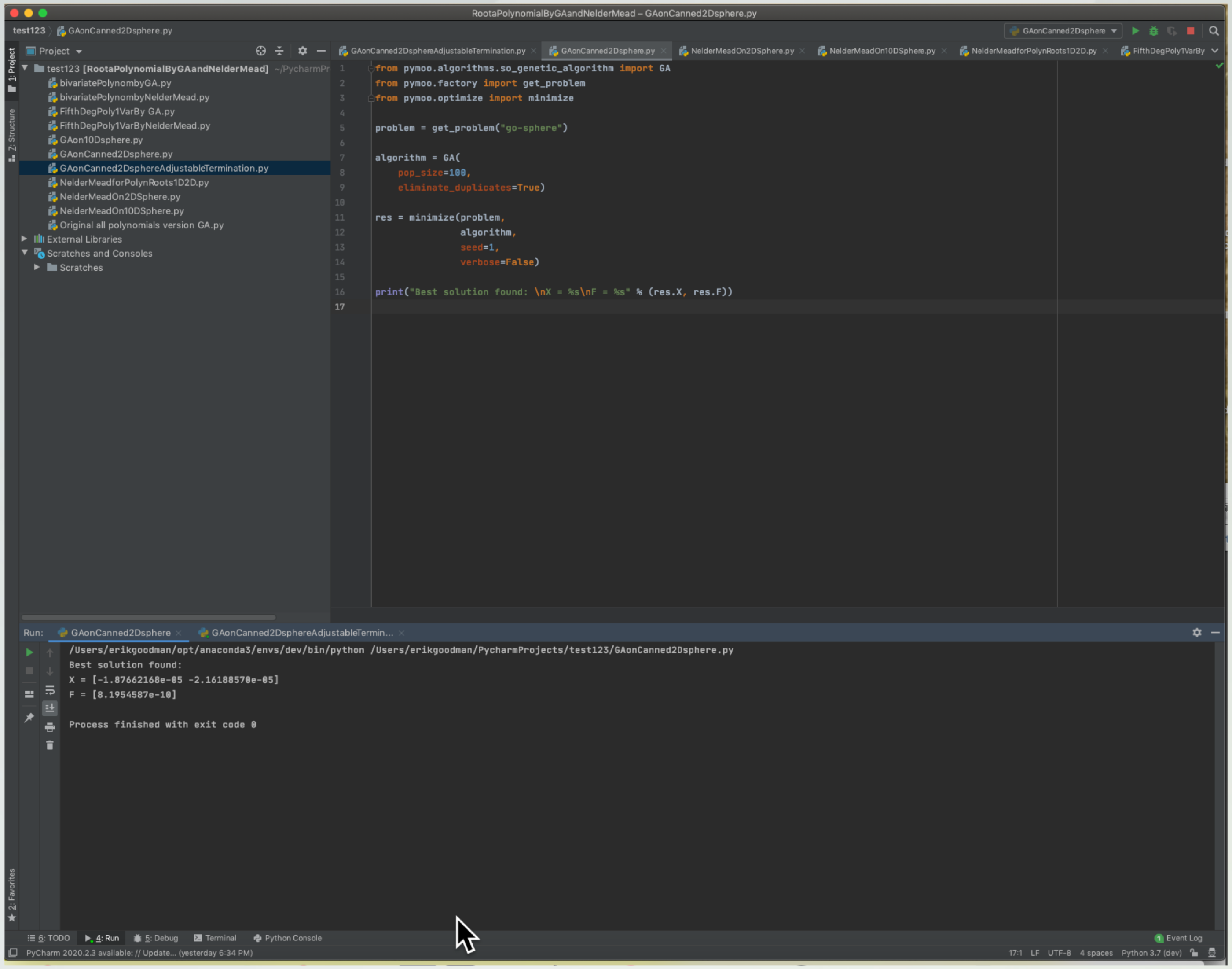
Output from GA run, default termination, population size 100, 2-D sphere function:

```
/Users/erikgoodman/opt/anaconda3/envs/dev/bin/python    /Users/erikgoodman/PycharmProjects/test123/GAonCanned2Dsphere.py
Best solution found:
X = [-1.87662168e-05 -2.16188570e-05]
F = [8.1954587e-10]


Process finished with exit code 0
```

(see screenshot next page)

Screenshot (what it looks like in PyCharm)


RootaPolynomialByGAandNelderMead – GAonCanned2Dsphere.py
test123
GAonCanned2Dsphere.py
Project
test123 [RootaPolynomialByGAandNelderMead] ~/PycharmPr
bivariatePolynomByGA.py
bivariatePolynombyNelderMead.py
FifthDegPoly1VarBy GA.py
FifthDegPoly1VarByNelderMead.py
GAon10Dsphere.py
GAonCanned2Dsphere.py
GAonCanned2DsphereAdjustableTermination.py
NelderMeadforPolynRoots1D2D.py
NelderMeadOn2DSphere.py
NelderMeadOn10DSphere.py
Original all polynomials version GA.py
External Libraries
Scratches and Consoles
Scratches
GAonCanned2DsphereAdjustableTermination.py
GAonCanned2Dsphere.py
NelderMeadOn2DSphere.py
NelderMeadOn10DSphere.py
NelderMeadforPolynRoots1D2D.py
FifthDegPoly1VarBy
Run:
GAonCanned2Dsphere
GAonCanned2DsphereAdjustableTermin...
6: TODO
4: Run
5: Debug
Terminal
Python Console
Event Log
PyCharm 2020.2.3 available: // Update... (yesterday 6:34 PM)
17:1 LF UTF-8 4 spaces Python 3.7 (dev)

```python
from pymoo.algorithms.so_genetic_algorithm import GA
from pymoo.factory import get_problem
from pymoo.optimize import minimize

problem = get_problem("go-sphere")

algorithm = GA(
    pop_size=100,
    eliminate_duplicates=True)

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

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

```
/Users/erikgoodman/opt/anaconda3/envs/dev/bin/python /Users/erikgoodman/PycharmProjects/test123/GAonCanned2Dsphere.py
Best solution found:
X = [-1.87662168e-05 -2.16188570e-05]
F = [8.1954587e-10]

Process finished with exit code 0
```

And if we ran it again?

- Identical output, as the code provided the seed for the random number generator (“seed=1”)
- If we want to let the run use a new seed every run, we can just say “seed=None” and accomplish that
- Now, another run yields:

```
Best solution found:
X = [-1.19789909e-05  1.75317458e-04]
F = [3.08797074e-08]

Process finished with exit code 0
```

If want to watch progress every gen, set:

“verbose=True”

That will cause program to print the generation number, the number of fitness evaluations done to date, the best fitness achieved so far, and the average fitness of the current population

```
 …
 47 |   4700 | 1.72760E-07 |  2.34369E-06
 48 |   4800 | 1.72760E-07 |  2.10954E-06
 49 |   4900 | 1.72760E-07 |  1.84637E-06
 50 |   5000 | 1.72760E-07 |  1.55333E-06
 51 |   5100 | 1.72760E-07 |  1.33895E-06
 52 |   5200 | 1.45396E-07 |  9.99768E-07
 53 |   5300 | 1.39991E-07 |  7.70167E-07
 54 |   5400 | 1.39991E-07 |  6.67964E-07
 55 |   5500 | 1.29023E-07 |  5.86499E-07
Best solution found:
X = [-0.00018878 -0.00030559]
F = [1.29023431e-07]
```

What if we want more evaluations, for more accurate minimization? Do this:

```
problem = get_problem("go-sphere")

algorithm = GA(
    pop_size=100,
    eliminate_duplicates=True)

termination = get_termination("n_eval", 10000)

res = minimize(problem,
               algorithm,
               termination,
               seed=1,
               verbose=False)

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

With 10,000 function evaluations on the 2-D sphere function, we get:

```
Best solution found:
X = [-3.68042531e-07 -1.71604751e-07]
F = [1.64903495e-13]
Normal termination
```

- So we can get more accuracy if we run longer, but note that the “fine-tuning” is relatively slow… Let’s compare with the Nelder-Mead algorithm on the same problem

Here's the code for the Nelder-Mead run:

```
from pymoo.algorithms.so_nelder_mead import NelderMead
from pymoo.factory import get_problem
from pymoo.optimize import minimize

problem = get_problem("go-sphere")

algorithm = NelderMead()

res = minimize(problem,
            algorithm,
            seed=None,
            verbose=True)

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

And here's what Nelder-Mead produces, with the default parameters:

```
…
20 |     48 |  0.000015053 |  0.000029017
21 |     50 |  8.54232E-06 |  0.000015380
22 |     52 |  3.55716E-06 |  9.05082E-06
23 |     54 |  9.37344E-07 |  4.34561E-06
24 |     56 |  9.37344E-07 |  2.17193E-06
25 |     58 |  9.06120E-07 |  1.28825E-06
26 |     60 |  4.05807E-07 |  7.49757E-07
Best solution found:
X = [3.28583969e-05 6.36181820e-04]
F = [4.05806982e-07]

Process finished with exit code 0
```

What do you notice?

More accuracy with a lot FEWER evaluations!

Bad problem for using a GA!

- Nelder-Mead got acceptable accuracy with only 60 total evaluations (although we ran the GA longer than necessary)
- If we ran N-M longer, it would continue making more progress per evaluation than the GA
- That's because this function is convex, unimodal, bivariate (only two independent variables to optimize), and way too simple to use a GA on, if you care at all about efficiency
- Of course, if the GA is lying around and the function you're to optimize is fast enough to compute, it doesn't matter… you can use almost anything!