CSE/ECE 848 Introduction to Evolutionary Computation

Module 2, Lecture 7, Part 2c
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

Let's look at some additional issues arising when optimizing:

- Let's find the real roots of some polynomials
- We'll start with a 5th-degree polynomial in one variable (so might be up to 5 real roots, and WILL be at least one):

  $f(x) = x^5 - 17x^3 - 12x^2 + 52x + 48$
- We're looking for values of x that will make f(x) = 0.
- To make it a MINIMIZATION problem, let's evaluate |f(x)| or abs(f(x)), the absolute value. It will have a minimum at the roots of the polynomial
- Let's look at Pymoo code for a GA to do this, for illustration, on the next two slides:

```
import sys
from pymoo.algorithms.so_genetic_algorithm import GA
from pymoo.model.problem import Problem
from pymoo.optimize import minimize

class MyProblem(Problem):
    # Find a root of a polynomial between -10 and 10, if there is one.
    def __init__(self):
        # define lower and upper bounds
        xl = -10
        xu = 10

        super().__init__(n_var=1, n_obj=1, n_constr=0, xl=xl, xu=xu,
evaluation_of="auto")
        # store custom variables needed for evaluation

    def _evaluate(self, x, out, *args, **kwargs):
        f = abs(x[:, 0]**5 - 17*x[:, 0]**3-12*x[:, 0]**2+52*x[:, 0]+48)
        # above polynomial has 5 real roots, at -1, -2, -3, 2, and 4.
        out["F"] = f
```

```
problem = MyProblem()

algorithm = GA(
    pop_size=100,
    seed=None,
    eliminate_duplicates=True)

res = minimize(problem,
            algorithm,
            termination=('n_gen', 100),
            verbose=True)


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

What do you observe?
Best vs average… ?

…

92	9200	0.011604807	0.265608279
93	9300	0.011604807	0.265608279
94	9400	0.011604807	0.260107714
95	9500	0.011604807	0.260107714
96	9600	0.011604807	0.258886657
97	9700	0.011604807	0.258886657
98	9800	0.011604807	0.255329448
99	9900	0.011604807	0.255329448
100	10000	**0.011604807**	**0.255329448**

Best solution found:

X = [-2.00048344]

F = [0.01160481]

Process finished with exit code 0

The Best value is not moving quickly, but the average value is staying MUCH worse, though slowly moving. Could some solutions near OTHER roots be remaining in the population?

Let's run it again, see if get something similar…

…

94	9400	0.000127921	0.239842729
95	9500	0.000127921	0.238744048
96	9600	0.000127921	0.238744048
97	9700	0.000127921	0.237353743
98	9800	0.000127921	0.237346722
99	9900	0.000127921	0.232482274
100	10000	0.000127921	0.232021759

Best solution found:

X = [-1.00000426]

F = [0.00012792]

Process finished with exit code 0

Aah! We got a DIFFERENT root, but that's okay, isn't it… a 5th-degree polynomial could have up to 5 real roots!

Lesson: Know what to expect before you run any optimizer! After several runs, GA found all the roots, at: -1, -2, -3, 2, and 4.

Let's try Nelder-Mead on this 5th-degree polynomial

95	190	0.00000E+00	0.00000E+00
96	192	0.00000E+00	0.00000E+00
97	194	0.00000E+00	0.00000E+00
98	196	0.00000E+00	0.00000E+00
99	198	0.00000E+00	0.00000E+00
100	200	0.00000E+00	0.00000E+00

Best solution found:

X = [2.]

F = [0.]

Running many times (it picks a different random starting point for each run),

It finds roots:

-1, -2, 2, and 4, **but never found -3 in over 100 runs…**

And it often fails, returning, for example:

Best solution found:

X = [-1.22783439]

F = [5.26098548]

Maybe we could reduce this problem by changing the termination conditions… it stopped after 200 evaluations

Let's look at another polynomial—a bivariate polynomial (2 independent variables)

This 4th-degree bivariate polynomial in x, y is defined by:

$$F(x,y) = x^2y^2 - 5xy^2 + 6y^2 + x^2y - 5xy + 6y - 2x^2 + 10x - 12$$

As before, to find the roots, we will minimize abs(f(x)).

If all roots are real, we expect to find 4 roots, 2 for x and 2 for y.

Let's see what the GA tells us, if we set 30 generations as the termination condition:

27	2700	1.59514E-07	0.000014622
28	2800	4.64680E-08	9.51944E-06
29	2900	4.64680E-08	5.47866E-06
30	3000	2.09969E-08	3.49869E-06

Best solution found:

X = [1.99999565 0.99838938]

F = [2.09968505e-08]

We can get as much accuracy as we want by running it more generations…

So is this the answer, x=2, y=1?

- Let's run the GA again, with random seed:

```
…
29 |   2900 |  3.75742E-09 |  1.64300E-07
30 |   3000 |  3.75742E-09 |  1.15826E-07
```

Best solution found:

X = [2.00234271 -2.00000054]

F = [3.75741926e-09]

So this says that x=2, y=-2 are also roots

Running again, we get x=2, y=1. After more runs, the GA comes up with a total of 4 answers, the pairs:

X	Y
2	-2
2	1
3	-2
3	1

Let's see what Nelder-Mead does on this problem

99 | 204 | 1.70530E-13 | 2.01913E-13

100 | 206 | 1.19016E-13 | 1.58688E-13

Best solution found:

X = [2.00000006 -1.99999935]

F = [1.19015908e-13]

And running it again:

Best solution found:

X = [3. -1.706434]

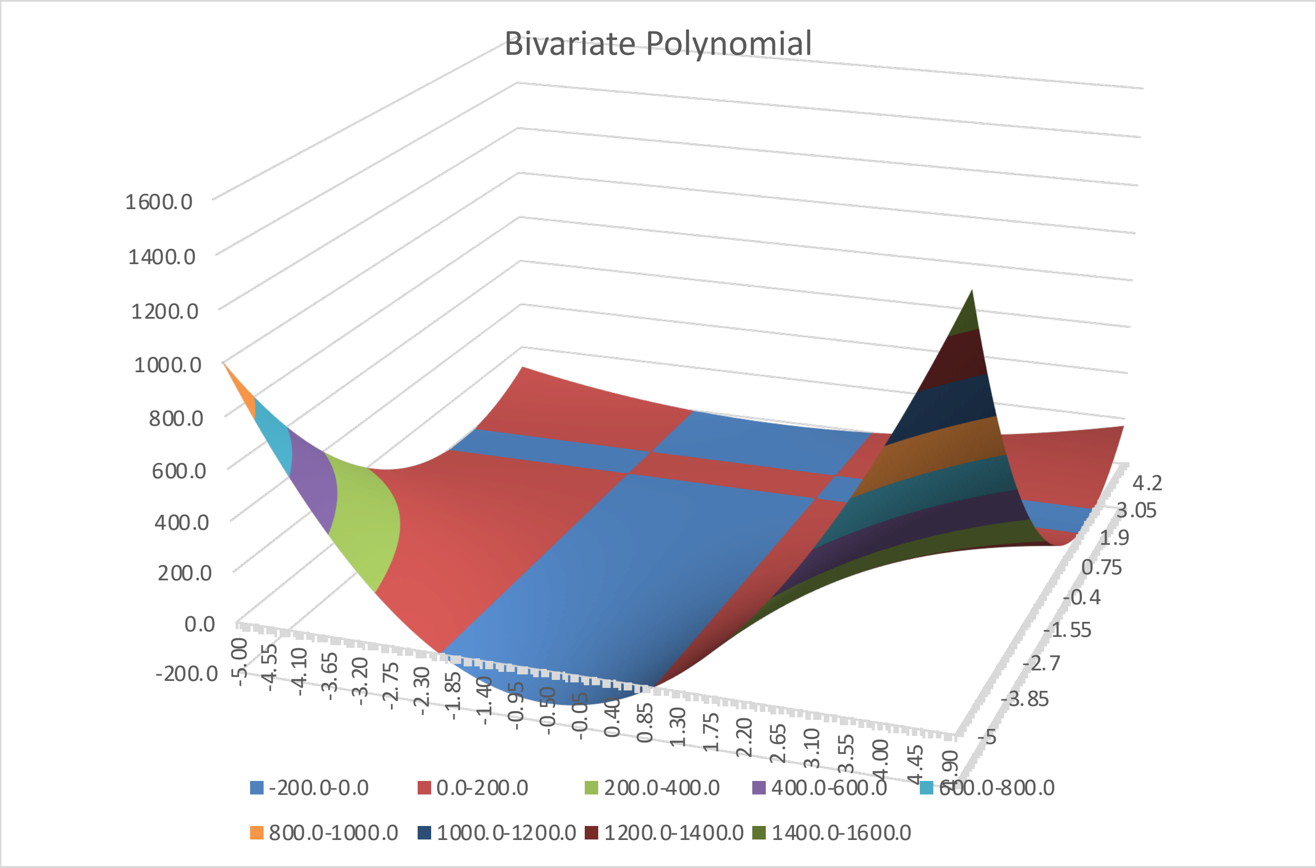
F = [0.]

I never got an answer like this from the GA…

1) Is it correct?
2) Why did the GA always minimize both x and y roots?

Here's the fitness landscape for that problem:

Note that function is 0 all along the intersections between blue (<0) and red (>0) areas, NOT only at the four "corner" points the GA found. Nelder-Mead's answers were okay!



What can we learn from this example?

- You need to understand the problem domain before you know if the answers your optimizer is giving you make sense and are the only good answers
- Because the GA never found the exact roots for either x or y, both x and y errors affected the overall error, so the GA worked at reducing BOTH. Nelder-Mead's formulaic moves allowed it to concentrate on driving one variable to its root, not both at once
- We don't NEED both x and y at roots for this problem; all of the four roots are valid independent of each other!

Lessons from today

- Don't use a GA if a less stochastic, greedier method will work well—it will be faster, **as long as it scales well enough for the problem size at hand**
- Know what form you expect your answer to take, so you will be able to recognize when the algorithm gives you something surprising or inappropriate
- Get an understanding of a number of different algorithm types (including several types of evolutionary algorithms) in your "bag of tricks" if you want to be an expert at optimization