Tracking Solution Paths with phcpy

Konrad Kadzielawa

April 28, 2016

Abstract

Solving polynomial systems for maximum number of real solutions arise quite frequently in a variety of scientific fields. It has been the subject of this study to explore the parameter space of two polynomial systems: system derived from two intersecting circles and a system defined by tangent lines to four spheres. The discriminant variety of the two intersecting circles was completely found and described, while the random walk optimization algorithm for finding the maximum number of tangent lines to four spheres was studied.

1 Statement of Purpose

This is the report of the study conducted in the Mathematical Computing Lab at University of Illinois at Chicago for Spring 2016, entitled "Tracking solution paths with phcpy" supervised by Prof. J. Verschelde and mentored by Nathan Bliss.

2 Introduction

The overall goal of this project was to develop Python scripts in order to explore the parameter space of systems arising in applications using the sweep homotopies implemented in *phcpy*. Nevertheless, the feature most extensively used in this project was the blackbox solver feature implemented in *PHCpack*, which allowed us to heuristically explore the parameter space of various polynomial systems.

3 Motivation

Solving polynomial systems for maximum number of real solutions arise quite frequently in a variety of scientific fields.
One of the motivating studies behind this project came from the paper published by P.Dietmaier from the Institut für Mechanik, Technische Universität Graz, entitled "The Stewart-Gough Platform of General Geometry can have 40 real postures" where he showed that Stewart-Gough platform actually possesses 40 real assembly modes or postures.

4 Discriminant

To understand what it takes to find the maximum number of real solutions, I'll need to very briefly touch upon three concepts relevant here: the **resultant**, the **discriminant** and the **discriminant variety**.
Given two polynomials, P and Q, the resultant is the determinant of the Sylvester matrix associated to P and Q, let's call it f. To determine the discriminant, we need to further compute the resultant of f and f'. Lastly, once the discriminant for the system has been found, its solution set constitutes the discriminant variety.
Why look for discriminant variety you might ask? In very simple terms - in the parameter space, maximally many real roots areas are very rare and they lie in extremely small chambers which are determined by the discriminant variety. Once we find it, we can choose from various regions defined by the discriminant variety in search of maximum # of solutions.
Nevertheless, the following two problems will illustrate an important issue: while the discriminant variety offers a theoretical solution to the problem of finding the maximum number of real solutions, for almost all nontrivial problems the complexity of the discriminant variety is too prohibitive for the theoretical solution to work in practice.

5 Two Circles Problem

As an example problem of explicitly finding the discriminant variety we considered a simple system consisting of a unit circle and circle defined by its 2 parameters: its radius and center located on the x-axis. This was described by the polynomial system:

$$f(x,y) = \begin{cases} x^2+y^2-1=0 \\ (x-c)^2+y^2-r^2=0 \end{cases}$$

To arrive at the solution, we performed a systematic loop through the combination of centers and radii inputs, given a step size (loop increment).

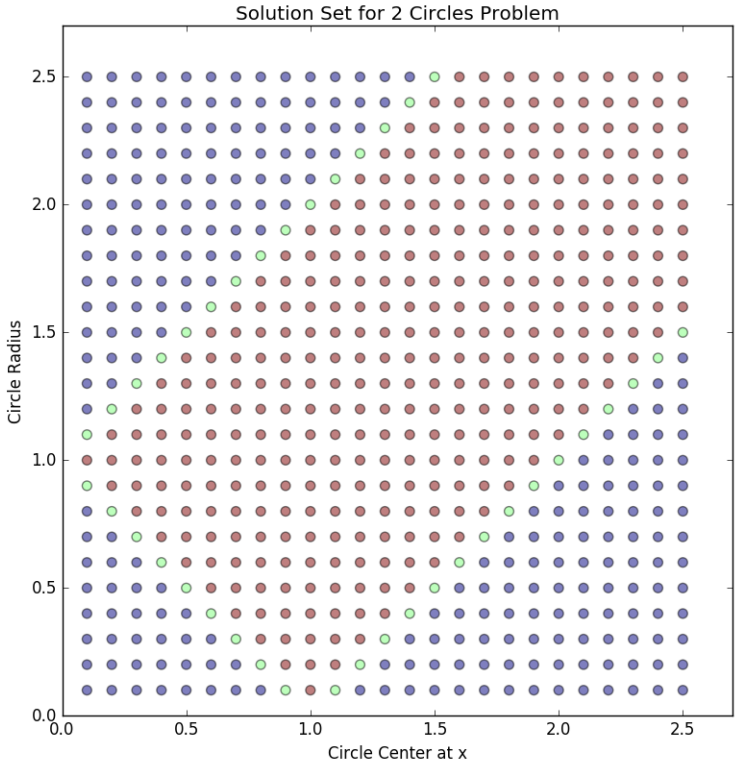
Listing 1: Function returning # of all solutions given center/radius.

```
from phcpy.solver import solve
from sympy import symbols
from numpy import arange

def return_solns(c_i,c_f,r_i,r_f,step):
        my_sols_list = []
        my_tuple_list = ()
        for i in arange(c_i,step+c_f,step):
                for j in arange(r_i,step+r_f,step):
                        x, y = symbols('x y')
                        coeff = float(i)
                        rad = float(j)
                        f1 = str(x^2 + y^2 - 1) +';'
```

```
            f2 = str(x^2 - 2*coeff*x + coeff
                ^2 + y^2 - rad^2) + ';'
            f = [f1, f2]
            s = solve(f, silent = True)
            my_tuple_list = (i,j,s)
            my_sols_list.append(my_tuple_list
                )
    return my_sols_list
```

The output of this function was a list of tuples which was further modified to arrive at the # of real solutions. Each pair of radii and x-axis centers was exported to JSON format and subsequently plotted in the *matplotlib*, with lime colored circles representing the discriminant variety. Our final output consisted of the following figure:



6 Four Spheres Problem

As a more advanced example of investigating the parameter space of polynomial systems, we considered the following geometric problem: **Given four spheres, how many**

real lines are tangent to all four spheres? The system of polynomial equations representing the problem contained 6 equations, 6 variables and total of 9 parameters all defining the centers and radii of 3 spheres (1 was a unit sphere).
As mentioned previously, the complexity of the parameter space for almost all nontrivial problems is prohibitively complex. Therefore, we performed a random walk through the parameter space using two approaches: "wiggle" approach where we varied each parameter individually around a good solution and "box minimizing" approach where we zoned in/minimized the range of parameters upon successive runs. Sample output from the "wiggle" approach is given:

[1.2298563009936108, 1.5877174064656896, 1.2510163838483237, 1.1698149762829906, 0.61428882846887711, 0.66700970876612264, 8]
[1.3298563009936109, 1.5877174064656896, 1.2510163838483237, 1.1698149762829906, 0.61428882846887711, 0.66700970876612264, 8]
[1.1518321516923646, 0.64248128111568947, 1.5657561296056171, 0.88083453863158423, 0.61156443017219031, 0.52009454251661402, 12]
I have found all 16 solutions with the parameters being: [1.1518321516923646, 0.64248128111568947, 1.465756129605617, 0.88083453863158423, 0.61156443017219031, 0.52009454251661402, 16]

The output above was taken from an instance when only 6 parameters were varied (all radii were set to $\frac{1}{2}$) and 6 parameters in order $[a1, a2, a3, b2, b3, c3]$ represent the values of spheres' centers at $(a1, 0, 0), (a2, b2, 0)$ and $(a3, b3, c3)$. Last element in each list represents total number of real solutions.
First list came from a random guess, second represented a "wiggle". "Wiggle" showed no improvement, hence new random guess was generated. Third list was again a random guess, while the fourth list showed the "wiggle" (in this case the third parameter decreasing by .1 yielded higher # of real solutions, hence the new list was kept).

The approach of "box minimizing", where upon successful runs, we decreased the range parameters were randomly chosen from, yielded much poorer results, with much longer time required to find all 16 real solutions. Sample output from "box" approach:

[1.4953553356286367, 0.57391064220933552, 1.2462095414234216, 0.953361181894123, 0.68226098447409278, 0.51743542020598154, 16]
[1.4826077817136394, 0.84827506029075073, 1.419849649609402, 0.64240241111165153, 0.77036583674361392, 0.55912122836549705, 12]
I have found all 16 solutions with the parameters being: [1.4953553356286367, 0.57391064220933552, 1.2462095414234216, 0.953361181894123, 0.68226098447409278, 0.51743542020598154]

In this case, we actually ran into all 16 real solutions through a random search first and minimizing the range only decreased the number of real solutions.
On average however, "box minimizing" approach performed slightly better than a purely random walk. The running time comparison between all 3 approaches can be seen seen on the poster presented along this written report.
A snippet of the "wiggle" python code in form of two functions (random list generator

and max. solution finder) can be seen below (breaklines for readability purposes):

Listing 2: Random List Generator Function

```
#given range low to high, and size = # of random params,
    trial = # of random list, return list of random lists
def random_nums_list_generator(low, high, size,trials):

        list_of_lists_of_nums = []
        for i in xrange(trials):
                list_of_lists_of_nums.append([random.
                    uniform(low,high) for _ in xrange(size
                    )])
        return list_of_lists_of_nums
```

Listing 3: Max. Solutions Function of "wiggle" approach

```
# this function takes the best of 100 random lists,
    x_list contains ``wiggled" lists (.1 = "wiggle" factor
    )
#further_updated_list contains best list of the x_list
def take_big_list_rerun(better_list):

        updated_better_list = max_first_list_printout(
            pop_list_of_lists(better_list))

        local_better_list = updated_better_list

        x_list = vary_args_first_max_list(
            local_better_list, .1)

        further_updated_list = max_first_list_printout(
            pop_list_of_lists(x_list))

        if local_better_list[6] == 16:
                print "I␣have␣found␣all␣16␣solutions␣with
                    ␣the␣parameters␣being:␣" +str(
                    local_better_list[0:6])
        elif further_updated_list[6] == 16:
                print "I␣have␣found␣all␣16␣solutions␣with
                    ␣the␣parameters␣being:␣" +str(
                    further_updated_list[0:6])
        elif further_updated_list[6] <= local_better_list
            [6]:
                main()
        else:
```

```
        further_updated_list =
            take_big_list_rerun(
            vary_args_first_max_list(
            further_updated_list ,.1))
    return  further_updated_list
```

Lastly, one of our solutions with return values plugged into "Grapher" application on OS X and the full solution of 12 real tangent lines (provided by Prof. Verschelde) is shown below:

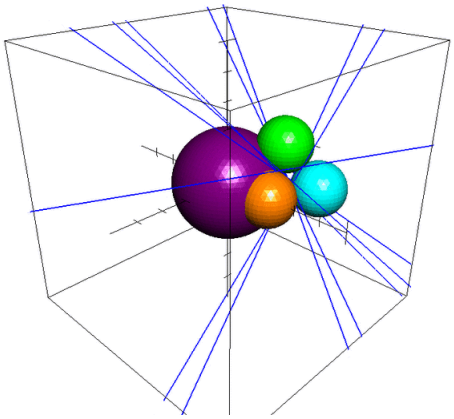


Figure 1: One of our 8 real tangent lines solutions.

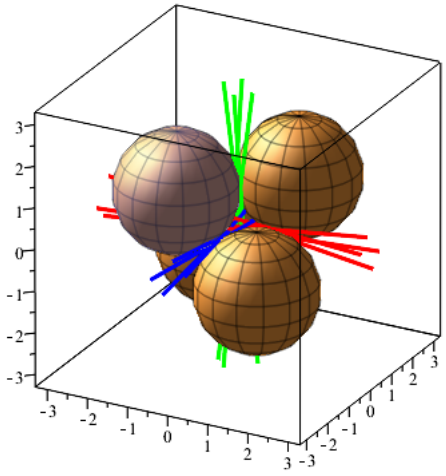


Figure 2: Fully real solution: 12 real tangent lines

References

[1] D. Cox, J. Little, and D. OShea. *Ideals, Varieties and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra* . Undergraduate Texts in Mathematics. SpringerVerlag, Fourth Edition, 2015.

[2] J. Verschelde Lecture 2, Elimination Methods & Lecture 24, Newton's Method with Deflation in *MCS 563: Analytic Symbolic Computation Lecture Notes*

[3] P. Dietmeier. "The Stewart-Gough platform of general geometry can have 40 real postures", in *Advances in Robot Kinematics: Analysis and Control*, Kluwer Academic Publishers, 1998, pp.1-10