Multiregeneration Tutorial

January 15, 2020

1 Getting Started

Say that we are given the following 2 polynomials with complex coefficient in 2 variables.

$$f_1 = (x-1)(y-3)$$
$$f_2 = (x-2)(y-4)$$

To solve the system above, let's change into the folder "getting-started", which contains the following three files.

bertiniInput_variables

```
variable_group x,y;
```

bertiniInput_equations

```
function f1,f2;
f1 = (x-1)*(y-3);
f2 = (x-2)*(y-4);
```

bertiniInput_trackingOptions *this file is empty*
inputFile.py

```
degrees = [[2], [2]]
workingDirectory = "run"
```

The first three files (those with the prefix "bertiniInput") are written in the C-like syntax used by the bertini software. The last file, "inputFile.py", contains the additional data that this program needs.

The variable "degrees" must be initialized to a list of lists, where the j'th element of the i'th list is the degree of the i'th function in the jth "variable group." For this example there is only one variable group consisting of x, y, and each function has degree two in this variable group. Therefor we use the python syntax

```
degrees = [[2], [2]]
```

to create a list of two lists, where the single element of the first list is the degree of f_1 and the single element of the second list is the degree of f_2.

The variable "workingDirectory" tell the program the name of the folder where it should write the output. If we run the program a second time, then a folder of that name will already exist, and it will be deleted before anything else happens.

To solve the system, we use python2 to run the "multiregeneration.py" script *from the "getting-started" folder*. The multiregeneration script will look for input files in the directory from which it is run, so make sure that you are in the directory with the system you wish to solve.

```
python2 ../multiregeneration.py
```

If all goes well python will print out "Done." and there should be a new directory called "run". If there was an error, then the most likely cause is that there was an error in one of the input files.

The solutions will be contained in the folder

```
run/_completed_smooth_solutions/depth_1
```

For this example, there are two files called

```
solution_tracking_depth_1_gens_1_1_dim_0_varGroup_0_regenLinear_1_pointId_326
solution_tracking_depth_1_gens_1_1_dim_0_varGroup_0_regenLinear_1_pointId_918
```

The two file contain approximate complex values for the two solutions of the initial system. For example the first file contains the following.

```
    1.999999999999996e+00  -4.107825191113079e-15
    3.000000000000000e+00  0.000000000000000e+00
```

The file can be read as

$$x = 1.999999999999996 \times 10^0 - (4.107825191113079 \times 10^{-15})i$$
$$y = 3.000000000000000 \times 10^0 - (0.000000000000000 \times 10^0)i$$

which is aproximately the solution $x = 2, y = 3$.

2 Multiple variable groups

Bézout's Theorem (one formulation of it) states that if $f_1, \ldots, f_n$ are complex polynomials in n variables of degrees $d_1, \ldots, d_n$ and $\mathcal{V}(f_1, \ldots, f_n)$ is finite, then the size of $\mathcal{V}(f_1, \ldots, f_n)$ is at most $d_1 d_2 \ldots d_n$. For now we will assume that our system has finitely many solutions. Therefore the degrees $d_1, \ldots, d_n$ give an upper bound on the size of the output, and in fact they are necessary for the algorithm itsself.

3 Projective variable groups

4 Nonsquare systems