Paramotopy: Parameter homotopies in parallel

Dan Bates[1], Danielle Brake[2], and Matt Niemerg[3]

[1] Colorado State University, USA
bates@math.colostate.edu,
www.math.colostate.edu/~bates
[2] University of Wisconsin - Eau Claire, USA
brakeda@uwec.edu,
danibrake.org
[3] research@matthewniemerg.com,
www.matthewniemerg.com

Abstract. Numerical algebraic geometry provides tools for approximating solutions of polynomial systems. One such tool is the parameter homotopy, which can be an extremely efficient method to solve numerous polynomial systems that differ only in coefficients, not monomials. This technique is frequently used for solving a parameterized family of polynomial systems at multiple parameter values. This article describes Paramotopy, a parallel, optimized implementation of this technique, making use of the Bertini software package. The novel features of this implementation include allowing for the simultaneous solutions of arbitrary polynomial systems in a parameterized family on an automatically generated or manually provided mesh in the parameter space of coefficients, front ends and back ends that are easily specialized to particular classes of problems, and adaptive techniques for solving polynomial systems near singular points in the parameter space.

1 Introduction

The methods of numerical algebraic geometry provide a means for approximating the solutions of a system of polynomials $F : \mathbb{C}^N \to \mathbb{C}^n$, i.e., those points $z \in \mathbb{C}^N$ such that $F(z) = 0$. There are many variations on these methods, but the key point is that polynomial systems of moderate size can be solved efficiently via homotopy continuation-based methods. In the case of a parameterized family of polynomial systems $F : \mathbb{C}^N \times \mathcal{P} \to \mathbb{C}^N$, where the coefficients are polynomial in the parameters $p \in \mathcal{P} \subset \mathbb{C}^M$, a particularly efficient technique comes into play: the parameter homotopy [1][4].

The process of using a standard homotopy to solve a system F begins with the construction of a polynomial system G that is easily solved. Once the system G is solved, the solutions of G are tracked numerically by predictor-corrector methods as the polynomials of G are transformed into those of F. Thanks to

[4] In fact, this technique applies when the coefficients are *holomorphic functions* of the parameters [1], but we restrict to the case of polynomials for simplicity.

the underlying geometry, discussed for example in [2] or [3], we are guaranteed to find a superset $\widehat{V}$ of the set V of isolated solutions of F. The set $\widehat{V}$ is easily trimmed down to V in a post-processing step [4].

Parameter homotopies are particularly powerful as the number of solutions to be followed is exactly equal to the number of isolated solutions of $F(z,p)$ for almost all values of $p \in \mathcal{P}$ (under the common assumption that $\mathcal{P}$ has positive volume in its ambient Euclidean space). Furthermore, the solution of a single G will work for almost all values of $p \in \mathcal{P}$, so only one round of precomputation is needed regardless of the number of polynomial systems to be solved.

Parameter homotopies are not new and have been used in several areas of application [5–8] and implemented in at least two software packages for solving polynomial systems: Bertini [9] and PHCpack [10]. These implementations allow the user to run a single parameter homotopy from one parameter value p_0 with known solutions to the desired parameter value, p_1, with the solutions at p_0 provided by the user. The software package that is the focus of this article differs from these other two implementations in the following ways:

1. Paramotopy accepts as input the general form of the parameterized family $F(z,p)$ (p given as indeterminates), chooses a random $p_0 \in \mathcal{P}$, and solves $F(z,p_0)$ via a Bertini run[5];
2. Paramotopy builds a mesh in the parameter space given simple instructions from the user (or uses a user-provided set of parameter values) and performs parameter homotopy runs from p_0 to each other p in the mesh;
3. Paramotopy carries out all of these runs in parallel, as available[6];
4. Paramotopy includes adaptive schemes to automatically attempt to find the solutions of $F(z,p)$ from starting points other than p_0 if ill-conditioning causes path failure in the initial attempt; and
5. Paramotopy is designed to simplify the creation of front ends and back ends specialized for particular applications.

The full version of this article [11] includes more background and examples.

2 Homotopies

2.1 Homotopy continuation

Given a polynomial system $F : \mathbb{C}^N \to \mathbb{C}^N$ to be solved, standard homotopy continuation consists of three basic steps:

1. Choose a *start system* $G : \mathbb{C}^N \to \mathbb{C}^N$ similar in some way to $F(z)$ that is "easy" to solve;
2. Find the solutions of $G(z)$ and form the new homotopy function $H : \mathbb{C}^N \times \mathbb{C} \to \mathbb{C}^N$ given by $H(z,t) = F(z) \cdot (1-t) + G(z) \cdot t \cdot \gamma$, where $\gamma \in \mathbb{C}$ is randomly chosen; and

[5] Bertini provides this functionality as well.

[6] Bertini and PHCpack both have parallel versions, but not for multiple parameter homotopy runs.

3. Using predictor-corrector methods (and various other numerical routines [12–14, 2]), track the solutions of G at $t = 1$ to those of F at $t = 0$.

There are many variations on this general theme, but we focus here on the basic ideas, leaving details and alternatives to the references. A discussion of the choice of an adequate start system G goes beyond the scope of this paper. It is enough to know that there are several such options [15, 16, 2, 3].

Once $G(z)$ is solved and $H(z,t)$ is formed, the solutions of $H(z,t)$ for varying values of t may be visualized as curves. Indeed, as t varies continuously, the solutions of $H(z,t)$ will vary continuously, so each solution sweeps out a curve or path (also sometimes called a *solution curve* or *solution path*) as t moves from 1 to 0. A schematic of four such paths is given in Figure 1. Predictor-corrector methods are used to follow the solutions of G to those of F along these paths. See [11] for more background.

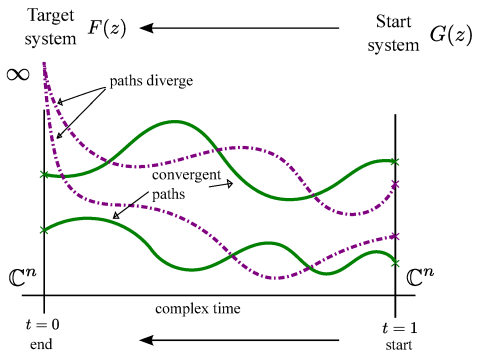

Fig. 1: A schematic depiction of a homotopy from system F to system G. There are four solutions of $G(z)$ at $t = 1$. Two solution paths diverge as $t \to 0$, while the other two lead to solutions of F at $t = 0$.

2.2 Parameter homotopies

Suppose we wish to solve a parameterized polynomial system $F(z,p)$ in variables z and parameters p at a (possibly very large) number of points in parameter space, i.e., we want to find z such that $F(z,p') = 0$ for varying values $p = p'$. If we know all isolated, finite, complex solutions at some generic point $p = p_0$ in a *convex*[7] parameter space $\mathcal{P}$, the underlying theory allows us to make use of a *parameter* or *coefficient-parameter homotopy* [1]. The usefulness of this software becomes readily apparent from the following proposition, proved in

[7] Handling non-convex parameter spaces is significantly more difficult and is described later.

somewhat different language in [2]. The proposition guarantees that we can find the isolated, finite, complex solutions of $F(z, p')$ simply by following paths through the parameter space, $\mathcal{P} \subset \mathbb{C}^M$, from the solutions of $F(z, p_0)$.

Proposition 1. *The number of finite, isolated solutions of $F(z, p)$ is the same for all $p \in \mathcal{P}$ except for a measure zero, algebraic subset $\mathcal{B}$ of $\mathcal{P}$.*

This proposition gives us a probability one guarantee that a randomly chosen path through parameter space will avoid $\mathcal{B}$. Assuming further that $\mathcal{P}$ is convex, a straight line segment through parameter space from a randomly chosen $p_0 \in \mathcal{P}$ to a prespecified target $p_1 \in \mathcal{P}$ will, with probability one, not pass through the set $\mathcal{B}$. This immediately implies a (known) technique for solving many polynomial systems from the same parameterized family with parameter space $\mathcal{P}$. First, find all finite, isolated, complex solutions for some randomly chosen $p_0 \in \mathcal{P}$. We refer to this as Step 1. Second, for each parameter value of interest, $p_i \in \mathcal{P}$, simply follow the finite, isolated, complex solutions through the simple homotopy $H(z, t) = F(z, p_0) \cdot t + F(z, p_i) \cdot (1 - t)$. We refer to this as Step 2. Notice that the randomly chosen γ from standard homotopies can be neglected in this homotopy since p_0 is chosen randomly. We describe in §3.2 how we monitor these Step 2 runs in case paths fail and also how we handle such failures.

For the cost of a single Step 1 solve at some random point p_0 in the parameter space, we may rapidly solve many other polynomial systems in the same parameterized family. Indeed, there are a minimal number of paths to follow in each Step 2 run and there is no pre-computation cost beyond the initial solve.

3 Implementation

Paramotopy is a C++ implementation of parameter homotopies, relying heavily on Bertini [9]. In this section, we provide many details about this software.

3.1 Main algorithm

We first present the main parameter homotopy algorithm that is implemented in Paramotopy. Note in particular the input value K and the while loop at the end, both included to help manage path failures during the Step 2 runs. Also, note that this algorithm assumes that $\mathcal{P} = \mathbb{C}^M$, for some M. The use of Paramotopy for other parameter spaces is described in §3.3.

Remark 1. To find all solutions for all $p \in L$, we must have that all solutions of $F(z, p_0)$ are nonsingular as we can only follow paths starting from nonsingular solutions during the parameter homotopies after the first run. Deflation [17, 18] could be used to regularize singularities in Step 1 before beginning Step 2, but this is not currently implemented.

Input : $F(z;p)$, a set of polynomial equations, variables $z \in \mathbb{C}^N$, and parameters $p \in L \subset \mathcal{P} = \mathbb{C}^M$; $\ell = \mid L \mid$ parameter values at which the solutions of $F(z;p)$ are desired; bound K on the number of times to try to find solutions for any given $p \in L$, in the case of path failures.

Output: List of solutions of $F(z;p) = 0$ for each $p \in L$.

Choose random $p_0 \in \mathcal{P}$;
Solve $F(z;p_0) = 0$ with any standard homotopy. (Step 1);
Store all nonsingular finite solutions in set S;
Set $\mathcal{F} := \varnothing$. (Beginning of Step 2.);
for *i=1 to* ℓ **do**
    Construct parameter homotopy from $F(z;p_0)$ to $F(z;p_i)$;
    Track all $|S|$ paths starting from points in S;
    Set $\mathcal{F} := \mathcal{F} \cup \{i\}$ if any path fails;
Set $k := 0$. (Beginning of path failure mitigation.);
while $|\mathcal{F}| > 0$ *and* $k < K$ **do**
    Set $\mathcal{F}' = \varnothing$;
    Choose random $p' \in \mathcal{P}$;
    Solve $F(z;p') = 0$ with a parameter homotopy from p_0;
    for *m=1 to* $|\mathcal{F}|$ **do**
        Solve $F(z;p_{\mathcal{F}[m]}) = 0$ with a parameter homotopy from p' to $p_{\mathcal{F}[m]}$;
        Set $\mathcal{F}' := \mathcal{F}' \cup \{m\}$ if any path fails;
    Set $\mathcal{F} := \mathcal{F}'$ and increment k;

Algorithm 1: Paramotopy.

3.2 Handling path failures during Step 2

If a path fails during a Step 2 run for some parameter value $p \in L$, Paramotopy will automatically attempt to find the solutions at p by tracking from a different randomly chosen parameter value $p' \neq p_0 \in \mathcal{P}$. It will repeat this process K times, with K specified by the user. This is the content of the while loop at the end of the Main Algorithm.

The idea behind this is that paths often fail for one of two reasons, either the path seems to be diverging or the Jacobian matrix becomes so ill-conditioned that either the steplength drops below the minimum allowed or the precision needed rises above the maximum allowed. For parameter homotopies, a path failure of the first type is possible for either of two reasons: either the path really is diverging or the norm of the solution is above a particular threshold. In the former case, it can happen that the nature of the solution set at target value p differs from that at a generic point in the parameter space, e.g., there could be fewer finite solutions at p. Such path failures are captured and reported by Paramotopy, but there is simply no hope for "fixing" them as this result is a natural consequence of the geometry of the solution set, i.e., p is inherently different from other points in parameter space, so Paramotopy takes the correct action in reporting it. In the latter case, it can happen that the scaling of the problem results in solutions that are large in some norm, e.g., $|z|_\infty > 10^5$ as is

the default in the current version of Bertini. If this is suspected, the user could rescale the system or adjust the threshold `MaxNorm` and run the problem again.

For the second type of path failure, the ill-conditioning is caused by the presence of a singularity $b \in \mathcal{B}$ near or on the path between p_0 and p. By choosing new starting point p' "adequately far" from p_0, it should be feasible to avoid the ill-conditioned zone around b unless b is near the target value p. In this last case, it is unlikely that choosing different starting points p' will have any value, which is why we have capped the number of new starting points allowed at K.

For now, the new point p' is chosen randomly in the unit hypercube. Future work will detect where in parameter space the failures have occurred and bound p' away from this region. Since it cannot easily be determined which paths from p' to p correspond to the failed paths from p_0 to p, there is no choice but to follow all paths from p' to p. To find all solutions at p', we simply use a parameter homotopy to move the solutions at p_0 to those at p'. Of course, if there are path failures, we must choose yet another p' and try again.

3.3 Handling parameter spaces other than $\mathbb{C}^M$

As described near the end of §2.2, Paramotopy may be used to handle parameter spaces other than the simplest parameter space, $\mathbb{C}^M$ for some M. However, some changes are needed in the algorithm.

If $\mathcal{P} \subset \mathbb{C}^M$ is a proper, convex subset of $\mathbb{C}^M$, Algorithm 1 needs only one change: p_0 must be somehow chosen within $\mathcal{P}$. To accommodate this, Paramotopy allows the user to specify p_0.

If $\mathcal{P}$ is a proper, non-convex set, more work is required. The Step 1 run would be the responsibility of the user, as in the previous paragraph, and it would be up to the user to string together subsequent Paramotopy runs to stay within $\mathcal{P}$.

3.4 Parallelization and data management

One of the features of Paramotopy that sets it apart from Bertini is the use of parallel computing for multiple parameter homotopies. Bertini includes parallel capabilities for a single homotopy run, but not for a sequence of runs. Parallelization was achieved using the head-worker paradigm, implemented with MPI. A single process controls the distribution of parameter points to the workers, which constitute the remainder of the processes. Workers are responsible for writing the necessary files for Bertini and for writing their own data to disk.

Bertini creates structures in memory by parsing an `input` file. As `input` is interpreted, several other files are created. These contain the straight line program, coefficient values, variable names, etc. Since the monomial structure of the polynomials in each Step 2 run is the same, almost all of these files are identical from one run to the next, so almost all this parsing is unnecessary. The only file that needs to be changed between runs is the file containing parameter values.

To prevent proliferation in the number of files needed to contain the data from the Paramotopy run, the Bertini output data is read back into memory, and dumped into a collective data file. The collective data files have a maximum buffer size, and once the buffer size is reached, the data in the buffer is written to the file, and the process repeats by storing the Bertini output data in memory until the buffer is full once more.

Repeated writing and reading is taxing on hard drives and clogs a LAN if the workers are using network drives. To free workers from having to physically write temporary files to electronic media storage, an option is provided to the user to exploit a shared memory location (or ramdisk), should it be available.

3.5 Front ends and back ends

Real-world problems may involve many parameters. This could be problematic when one wants to discretize a parameter space into a uniform sample as the number of parameter points of interest can easily reach into the astronomical. Hence, Paramotopy contains support for both linear uniform meshes of parameters as well as user-defined sets of parameter values stored in a text file. A generic `Matlab` interface for gathering, saving, and plotting data from an arbitrary Paramotopy run is provided on the Paramotopy website. See [11] for further details.

4 Conclusions

Paramotopy can be used to solve parameterized polynomial systems efficiently for large numbers of parameter values. This extends the reach of numerical algebraic geometry in a new direction, particularly one that might be useful for mathematicians, scientists, and engineers who would like to rapidly test a hypothesis or would like to find regions of a parameter space over which the polynomial system has the same number of solutions. While Bertini and PHCpack have some parameter homotopy capabilities, Paramotopy has been optimized for the scenario of using many-processor computers to solve at many parameter values of interest.

Acknowledgements

The authors appreciate the useful comments from several anonymous referees and Andrew Sommese as these have greatly contributed to the quality of this paper. The first author would also like to recognize the hospitality of Institut Mittag-Leffler and the Mathematical Biosciences Institute, as well as partial support from the NSF via award DMS-1719658.

References

1. Sommese, A., Morgan, A.: Coefficient-parameter polynomial continuation. Appl. Math and Comp. **29** (1989) 123–160
2. Sommese, A.J., Wampler, C.W.: The Numerical solution of systems of polynomials arising in engineering and science. World Scientific Publishing (2005)
3. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Numerical solution of polynomial systems using the software package Bertini. SIAM, Philadelphia, PA (2013)
4. Bates, D., Hauenstein, J., Peterson, C., Sommese, A.: A numerical local dimension test for points on the solution set of a system of polynomial equations. SIAM J. Numer. Anal. **47**(5) (2009) 3608–3623
5. Brake, D.A., Bates, D.J., Putkaradze, V., Maciejewski, A.A.: Illustration of numerical algebraic methods for workspace estimation of cooperating robots after joint failure. In: 15th IASTED Int. Conf. on Rob. and Appl. (2010) 461–468
6. He, Y.H., Mehta, D., Niemerg, M., Rummel, M., Valeanu, A.: Exploring the potential energy landscape over a large parameter-space. Journal of High Energy Physics **2013**(7) (2013) 1–29
7. Newell, A.J.: Transition to superparamagnetism in chains of magnetosome crystals. Geochem. Geophys. Geosy. **10**(11) (2009) Q11Z08
8. Rostalski, P., Fotiou, I.A., Bates, D.J., Beccuti, A.G., Morari, M.: Numerical algebraic geometry for optimal control applications. SIAM J Optimiz. **21**(2) (2011) 417–437
9. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.: Bertini: Software for numerical algebraic geometry (2006)
10. Verschelde, J.: Algorithm 795: Phcpack: A general-purpose solver for polynomial systems by homotopy continuation. ACM Transactions on Mathematical Software (TOMS) **25**(2) (1999) 251–276
11. Bates, D., Brake, D., Niemerg, M.: Paramotopy: Parameter homotopies in parallel. (2018) arXiv.org/abs/1804.04183.
12. Bates, D., Hauenstein, J., Sommese, A., Wampler, C.: Adaptive multiprecision path tracking. SIAM J. Numer. Anal. **46**(2) (2008) 722–746
13. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Stepsize control for path tracking. Contemp. Math. **496** (2009) 21–31
14. Bates, D.J., Hauenstein, J.D., Sommese, A.J.: Efficient path tracking methods. Numer. Algorithms **58**(4) (2011) 451–459
15. Wampler, C.W.: Bezout number calculations for multi-homogeneous polynomial systems. Appl. math. and comput. **51**(2) (1992) 143–157
16. Li, T.Y.: Numerical solution of polynomial systems by homotopy continuation methods. Handb. Numer. Anal. **11** (2003) 209–304
17. Leykin, A., Verschelde, J., Zhao, A.: Newton's method with deflation for isolated singularities of polynomial systems. Theoretical Computer Science **359** (2006) 111–122
18. Hauenstein, J., Wampler, C.: Isosingular sets and deflation. Foundations of Computational Mathematics **13** (2013) 371–403