ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

NAClab: A Matlab Toolbox for Numerical Algebraic Computation

(extended abstract)

Zhonggang Zeng* Tien-Yien Li†

1 Introduction

We present a Matlab toolbox NAClab for numerical algebraic computation. This toolbox includes Matlab implementations of the basic numerical algorithms in algebraic computations and utility functions. Those functions can be used either directly in applications or as building blocks for implementing advanced computing methods. NAClab is a result of collective effort and its contributors include Liping Chen, Tianran Chen, Wenrui Hao, Tsung-Lin Lee, Andrew Sommese and Wenyuan Wu.

Numerical algebraic computation, particularly numerical polynomial algebra, emerges as a growing area of study in recent years with a solid foundation in place for building numerical and hybrid computing methods [8, 9]. Many robust numerical and numeric-symbolic algorithms have been developed for solving polynomial systems, polynomial factorizations, polynomial GCD, computing dual bases and multiplicity structures of polynomial ideals, etc, with implementations such as in [1, 2, 3, 5, 6, 10] Those algorithms have a broad spectrum of applications in scientific computing such as robotics, control, image processing, computational biology and chemistry, and so on.

One of the main difficulties for numerical algebraic computation is the ill-posedness that frequently occurs when the solution of a problem does not possess Lipschitz continuity with respect to data. Those ill-posed problems are not directly suitable for floating point arithmetic since the solutions are infinitely sensitive to rounding errors. However, it has been shown in recent studies that such a difficulty can be overcome by seeking a regularized numerical solution with a proper formulation. Algorithms implemented in NAClab are designed to regularize the problem for removing ill-posedness rather than extending machine precision for accurate computation. The package also includes generic routines for matrix building and Gauss-Newton iteration that are the main engines for handling ill-posed algebraic computaiton.

NAClab is an on-going project as a major upgrade and expansion from its predecessor ApaTools [13]. At this stage, the main objective is to provide researchers in numerical algebra with generic and versatile tools that simplify and accelerate algorithm development, experimentation, and implementation. We emphasize on achieving the highest possible accuracy and robustness in algorithm design and implementation.

NAClab is maintained at its permanent website[1] and freely accessible to academic researchers and educators for the foreseeable future.

2 Differences between symbolic and numerical algebra

Conventional symbolic computation assumes both data and arithmetic to be exact. In practical applications, however, problem data are likely to be empirical. As a result, exact solutions of those inexact problems may not serve the practical purposes. Using the polynomial

$$p = 81\,x^4 + 16\,y^4 - 648.001\,z^4 + 72x^2y^2 + .002x^2z^2 + .001y^2z^2 - 648x^2 - 288y^2 - .007z^2 + 1296 \qquad (1)$$

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA, email: zzeng@neiu.edu.
†Department of Mathematics, Michigan State University, E. Lansing, MI 48824, USA, email: li@math.msu.edu
[1] http://www.neiu.edu/∼naclab

in [7] as an example, Kaltofen proposed the Open Problem 1 in challenges of symbolic computation: Is there factorable polynomial nearby? While polynomial p in (1) is not factorable in conventional sense, it is a perturbed data representation of a factorable polynomial $\tilde{p}$. The objective of numerical factorization is to calculate the factorization of the hidden underlying polynomial $\tilde{p}$ using the imperfect data p. The function `PolynomialFactor` in `NAClab` is built to carry out this computation as follows. First of all, `NAClab` allows polynomials to be entered intuitively into Matlab as strings:

```
>> p = '81*x^4+16*y^4-648.001*z^4+72*x^2*y^2+.002*x^2*z^2+.001*y^2*z^2-648*x^2-288*y^2-.007*z^2+1296';
```

Then the numerical factorization can be obtained using the known relateve data error bound 10^{-5} as the error tolerance:

```
>> F = PolynomialFactor(p,1e-5,'row')     % factor p within error tolerance 1e-5, showing result in a row
F =
1296 * (1-0.25000013640167*x^2-0.11111232357041*y^2+0.707104626181296*z^2) * (1-0.249999863598339*x^2-0.111110989865191*y^2-0.707110027415863*z^2)
```

This result is an accurate approximation to the factorization of the hidden polynomial $\tilde{p}$ and reveals the graph of $p = 0$ is the union of an ellipsoid and a hyperboloid of one sheet.

In contrast to symbolic computation, another subtle issue in numerical algebraic computation is that the given empirical data may have different underlying solutions depending on the error tolerance. This phenominon can be further illustrated in polynomial factorization on

$$f \quad = \quad 0.47619031y + 0.5714288y^2 + 0.55555493x + 1.3809278yx + 0.8571143y^2x + 0.8333328x^2 + yx^2$$

Two numerical factorizations within different error tolerances can be computed by `NAClab` function `PolynomialFactor`:

```
>> PolynomialFactor(f,1e-4,'row')  % within tolerance 1e-4
ans =
0.999996404282881 * (0.833329772039985 + y)  * (0.666677452701414 + x)  * (0.85712030479082*y + x)

>> PolynomialFactor(f,1e-6,'row')  % within tolerance 1e-6
ans =
1.00000001100681 * (0.833332777619492 + y)  * (0.571428774473131*y + 0.666666352807009*x + 0.857114290319387*y*x + x^2)
```

They are accurate approximations to the factorizations of two nearby polynomials from the same data.

Many other algebraic computations follow a similar pattern: An accurate solution of an algebraic problem is to be computed but the problem data are imperfect so that the exact solution is meaningless since the solution is infinitely sensitive to data perturbations. The objective of the numerical algebraic computation is to solve the problem using the slightly perturbed data within an error tolerance similar to the factorization example above. `NAClab` is built for this purpose.

Algebraic problems are often ill-posed because the set of problems whose solutions possessing a distinct structure form a manifold of positive codimension, and perturbations generically pushes a given problem away from the manifold. Our strategy starts with formulating the *numerical solution* of an ill-posed algebraic problem following a "three strikes" principle consisting of *backward nearness, maximum codimension* and *minimum distance* [13] for removing the ill-posedness. Based on those formulation principles, computing the numerical solution can be carried out in two optimization processes: maximizing the codimension of manifolds followed by minimizing the distance to the manifold, leading to a two-staged strategy for designing robust algorithms.

The main mechanism at Stage I is matrix rank-revealing, while Stage II relies on solving nonlinear least squares problems. In `NAClab`, we provide matrix building/computation tools for Stage I and nonlinear least squares tools for Stage II.

3 NAClab overview

NAClab originated from its predecessor ApaTools [13]. Among many improvement areas, we implemented a user-friendly mechanism for direct polynomial manipulations and included a major package in numerical solution of polynomial systems by homotopy continuation method. For example, solving the polynomial system

$$x^5 - y^5 + 3y + 1 \quad = \quad 5y^4 - 3 \quad = \quad 20x - y + z \quad = \quad 0$$

can now be carried out in a simple call:

```
>> P = {'x^5-y^5+3*y+1','5*y^4-3','20*x-y+z'};    % enter the polynomial system directly and intuitively
>> [Solutions, variables] = psolve(P)             % call the polynomial system solver and obtain all the isolated solutions
   Solutions =
     Column 1
     0.778497746685646 + 0.893453081179308i
    -0.000000000000000 + 0.880111736793394i
   -15.569954933712914 -16.988949886792764i
     ... ...
     Column 20
     0.778497746685646 - 0.893453081179308i
     0.000000000000000 - 0.880111736793393i
   -15.569954933712925 +16.988949886792767i

   variables =
    'x'    'y'    'z'
```

Compared to Maple and Mathematica, Matlab has an advantage in efficient numerical matrix computations with an disadvantage in user interface. Cumbersome representations are required for carrying out polynomial and other algebraic computations. Using NAClab, polynomials can be entered and displayed as character strings in Matlab in a way similar to Maple.

```
>> f = '3*x^2 - (2-5i)*x^3*y^4 - 1e-3*z^5-6.8'
>> g = '-2*y^3 - 5*x^2*z + 8.2'
```

Using such an intuitive polynomial representation, users can now perform common polynomial operations such as addition, multiplication, power, evaluation, differentiation, factorization, extracting coefficients, finding greatest common divison (GCD), etc, by calling NAClab functions, such as

```
>> p = pplus('2*x^5-3*y','4+x*y')           % add any number of polynomials
>> q = ptimes(f,g,h)                        % multiply any number of polynomials
>> v = PolynomialEvaluate(f,{'x','z'},[3,4]) % evaluate f(x,y,z) for x=3, z=4

>> % and a lot more.  For example, to compute a greatest common divisor:
>> f = '10 - 5*x^2*y + 6*x*y^2 - 3*x^3*y^3';
>> g = '30 + 10*y + 18*x*y^2 + 6*x*y^3'
>> u = PolynomialGCD(f,g)
u =
33.5410196624968 + 20.1246117974981*x*y^2
```

In summary, NAClab is developed for numerical algebraic computations in Matlab including solving polynomial systems, polynomial factorizations, polynomial greatest common divisors, multiplicity and dual spaces of nonlinear systems at isolated zeros, numerical Jordan Canonical Forms, and numerical rank revealing. The package also contains a comprehensive library of programming utilities for building further algorithms for numerical algebraic computations.

NAClab is an on-going project. While continuing to refine the existing functions, we shall expand the package by developing more algorithms and their implementations for numerical algebraic computations.

References

[1] D.J. Bates, C. Peterson and A.J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese and C.W. Wampler II, *Software for numerical algebraic geometry: A paradigm and progress towards its implementation*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 1–14.

[3] R. M. Corless, S. M. Watt, and L. Zhi, *QR factoring to compute the GCD of univariate approximate polynomials*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[4] B. Dayton, T.Y. Li and Z. Zeng, *Multiple zeros of nonlinear systems* Mathematics of Computation, Vol. 80, pp. 2143-2168, 2011

[5] S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi, *Approximate factorization of multivariate polynomials via differential equations.* Proc. ISSAC '04, ACM Press, pp 167-174, 2004.

[6] C.-P. Jeannerod and G. Labahn, *The SNAP package for arithemetic with numeric polynomials.* In International Congress of Mathematical Software, World Scientific, pages 61-71, 2002.

[7] E. Kaltofen, *Challenges of symbolic computation: My favorite open problems*, J. Symb. Comput., 29, pp.161-168, 2000.

[8] A.J. Sommese and C.W. Wampler II, *The Numerical Solution of Systems of Polynomials*, World Scientific Pub., Hackensack, NJ. 2005

[9] H. J. Stetter, *Numerical Polynomial Algebra*, SIAM, 2004.

[10] J. Verschelde, *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. on Math. Software, 25(1999), pp. 251–276.

[11] Z. Zeng, *A polynomial elimination method for symbolic and numerical computation.* 409(2008) pp. 318-331.

[12] Z. Zeng, *Computing multiple roots of inexact polynomials*, Math. Comp., 74 (2005), pp. 869–903.

[13] Z. Zeng, *ApaTools: A Maple and Matlab toolbox for approximate polynomial algebra*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 149–167.

[14] Z. Zeng and B. Dayton, *The approximate GCD of inexact polynomials. II: A multivariate algorithm.* Proceedings of ISSAC'04, ACM Press, pp 320-327. (2006).