ApaTools: A Software Toolbox for Approximate Polynomial Algebra

Zhonggang Zeng*

March 12, 2007

Abstract

Approximate polynomial algebra becomes an emerging area of study in recent years with a broad spectrum of applications. In this report, we present a software toolbox entitled `ApaTools` that consists of basic algorithms in approximate polynomial algebra. Those algorithms form a foundation for developing more advanced numerical and symbolic methods in computational algebra.

1 Introduction

Approximate polynomial algebra emerges as a growing area of study in recently years. With rich theories and abundant symbolic algorithms being in place for commutative algebra and algebraic geometry, numerical and hybrid computing methods appear to have a solid foundation for development. Many robust numerical and numeric-symbolic algorithms have been developed for solving polynomial systems [1, 5, 7, 9, 11], univariate factorization [15, 16], univariate GCD [13], multivariate GCD [4, 17, 18], computing the dual bases and multiplicity structure of polynomial ideals [2, 3], multivariate factorization [4, 17], and polynomial elimination [14], etc. Those algorithms have a broad spectrum of applications in scientific computing robotics, image processing, computational biology and chemistry, and so on.

We present `ApaTools`, a software toolbox for approximate polynomial algebra in this report. This toolbox includes Matlab and Maple implementations of basic algorithms, utility procedures and test suites. Those modules can be either used directly in applications or as building blocks for more advanced computing methods. Algorithms implemented in `ApaTools` rely heavily on matrix computations, particularly matrix rank-revealing.

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA, Research supported in part by NSF under Grant DMS-0412003, email: `zzeng@neiu.edu`.

One of the main difficulties for numerical computation in polynomial algebra is the frequent ill-posedness which occures when a problem has a discontinuous solution respect to data. Those ill-posed problems are not directly suitable for floating point arithmetic since the solutions are infinitely sensitive to rounding errors. This difficulty can often be overcome by seeking the *approximate solution* that is formulated based on the three principles of *backward nearness*, *maximum co-dimension* and *minimal distance* [13, 16, 19] that will be elaborated in §2.

2 Exact and approximate polynomial algebra

Conventional symbolic computation assume both data and arithmetic are *exact*. In practical applications, problem data are more often to be perturbed. As a result, exact solutions of those inexact problems may not serve the practical purposes. Alternatively, an *approximate solution* is intended to approximate the underlying exact solution before the problem is perturbed. The comparison can be seen in the following examples.

Example 1 Exact and approximate GCD. The following polynomial pair has an exact GCD $gcd(f,g) = x^2 - 2xy + 3z$:

$$\begin{aligned} f(x,y,z) &= \tfrac{513}{217}x^3z - \tfrac{127}{311}x^2z^2 - \tfrac{1026}{217}x^2yz + \tfrac{254}{311}xyz^2 - \tfrac{1539}{217}z^3, \\ g(x,y,z) &= \tfrac{213}{131}x^2yz - \tfrac{59}{77}x^4 - \tfrac{426}{131}xy^2z + \tfrac{118}{77}x^3y + \tfrac{639}{131}yz^2 - \tfrac{177}{77}x^2z. \end{aligned}$$

When polynomials are represented with 8-digit precision in coefficients in Maple

$$\begin{aligned} \tilde{f}(x,y,z) &= 2.3640553x^3z - 0.40836013x^2z^2 - 4.7281106x^2yz + 0.81672026xyz^2 - 1.2250804z^3, \\ \tilde{g}(x,y,z) &= 1.6259542x^2yz - 0.76623377x^4 - 3.2519084xy^2z + 1.5324675x^3y + 4.8778626yz^2 - 2.2987013x^2z. \end{aligned}$$

The exact GCD degrades to $gcd(\tilde{f},\tilde{g}) = 1$. The *approximate GCD within* ε, as calculated by our Maple module `MvGCD` for $10^{-7} < \varepsilon < 10^{-2}$,

$$gcd_{\varepsilon}(\tilde{f},\tilde{g}) = 0.99999999x^2 - 2.0000000xy + 3.0000000z$$

is an accurate approximation to the underlying exact GCD. □

Example 2 Exact and approximate univariate factorization. Univariate factorization in complex field $\mathbb{C}$ is equivalent to root-finding. Consider univariate polynomial

$$\begin{aligned} p(x) &= x^{200} - 400\,x^{199} + 79500\,x^{198} + \ldots + 2.04126914035338 \cdot 10^{86}\,x^{100} - 3.55467815448396 \cdot 10^{86}\,x^{99} + \\ &\quad \ldots + 1.261349023419937 \cdot 10^{53}\,x^2 - 1.977831229290266 \cdot 10^{51}\,x + 1.541167191654753 \cdot 10^{49} \qquad (1) \\ &\approx (x-1)^{80}(x-2)^{60}(x-3)^{40}(x-4)^{20} \qquad (2) \end{aligned}$$

with coefficients in hardware precision (16-digits). Using the coefficient vector of p as input, the standard Matlab root-finder outputs a cluster of 200 roots as shown in Figure 1. In contrast, our Matlab module `UvFactor` calculates an *approximate factorization*:

```
>> [F,res,cond] = UvFactor(p);

THE CONDITION NUMBER:                         2.57705
THE BACKWARD ERROR:                      7.62e-016
THE ESTIMATED FORWARD ROOT ERROR:        3.93e-015

  FACTORS

   ( x -   4.000000000000008 )^20
   ( x -   2.999999999999994 )^40
   ( x -   2.000000000000002 )^60
   ( x -   1.000000000000000 )^80
```

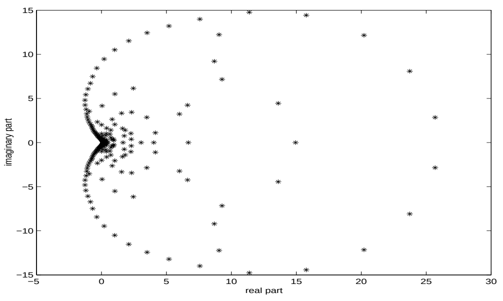

Figure 1: Matlab results for the polynomial (1)

The *approximate factors* calculated by UvFactor approximate the underlying factors in (2) with 15 correct digits. In exact sense, however, the perturbed polynomial p has only linear factors corresponding to the root cluster in Figure 1. $\square$

Both examples show the effect of ill-poseness on inexact problems. Since the solution structure can be completely altered by an infinitesimal error on data, it may not be meaningful to compute the exact solution of a perturbed ill-posed problem. Therefore, we need to formulate an *approximate solution* for the given problem in order to remove the ill-posedness. Using polynomial factorization as an example, we elaborate the formulation process below.

Associating monic polynomial $p(x) = x^4+p_1x^3+p_2x^2+p_3x+p_4$ of degree 4 with coefficient vector $[p_1, p_2, p_3, p_4] \in \mathbb{C}^4$ and defining norm $\|p\| = \sqrt{|p_1|^2 + \cdots + |p_4|^2}$, the set of all polynomials possessing a factorization $(x-z_1)^1(x-z_2)^3$ with roots $z_1 \neq z_2$ form a manifold $\Pi(1,3)$ of codimension 2. On the other hand, manifold $\Pi(1,3)$ is in the closure of manifold $\Pi(1,1,2)$ of codimension 1 since $(x-z_1)^1(x-z_2+\varepsilon)^1(x-z_2+\varepsilon')^2 \longrightarrow (x-z_1)^1(x-z_2)^3$ when $\varepsilon, \varepsilon' \longrightarrow 0$. Likewise $\Pi(1,1,2) \subset \overline{\Pi(1,1,1,1)} \equiv \mathbb{C}^4$, and the five manifolds form a *stratification* as shown in Figure 2.

When polynomial $p = (x-z_1)^1(x-z_2)^3 \in \Pi(1,3)$ is perturbed as $\tilde{p}$, the original factoring structure $(1,3)$ is lost since $\tilde{p} \in \Pi(1,1,1,1) \equiv \mathbb{C}^4$. However, structure $(1,3)$ can still be recovered from $\tilde{p}$ since $\Pi(1,3)$ is the manifold of *highest codimension* passing through the ε-neighborhood of $\tilde{p}$ for ε satisfying

$$\|p-\tilde{p}\| < \varepsilon < \inf\left\{\|\tilde{p}-q\| \;\middle|\; q \in \Pi(2,2) \cup \Pi(4)\right\}.$$

codimension

$$\begin{tikzcd}
3 & 2 & 1 & 0 \\
 & \Pi(2,2) \arrow[dr] & & \\
\Pi(4) \arrow[ur] \arrow[dr] & & \Pi(1,1,2) \arrow[r] & \Pi(1,1,1,1) \\
 & \Pi(1,3) \arrow[ur] & &
\end{tikzcd}$$

Figure 2: Stratification of manifolds of degree 4 polynomials, with "$\longrightarrow$" denoting "in the closure of"

After identifying $\Pi(1,3)$, we define the *approximate factorization* of $\tilde{p}$ within ε as the exact factorization of $\hat{p} = (x-\hat{z}_1)^1(x-\hat{z}_2)^3$ that is the nearest polynomial to $\tilde{p}$ in $\Pi(1,3)$. With this formulation, computing the approximate factorization of $\tilde{p}$ or p is a well-posed problem, and the roots $\hat{z}_1$ and $\hat{z}_2$ of $\hat{p}$ are approximation to the intended roots z_1 and z_2 of p with error bounded by a structure preserving condition number [16].

Generally, the 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 *approximate solution* of an ill-posed algebraic problem following a "three strikes" principle to remove the discontinuity:

Backward nearness: The approximate solution to the given (inexact) problem $\tilde{\mathcal{P}}$ is the exact solution of a nearby problem $\hat{\mathcal{P}}$ within ε.

Maximum codimension: The nearby problem $\hat{\mathcal{P}}$ is in the manifold Π of highest codimension among all manifolds passing through the ε-neighborhood of $\tilde{\mathcal{P}}$.

Minimum distance: Problem $\hat{\mathcal{P}}$ is the nearest point in manifold Π to the given problem $\tilde{\mathcal{P}}$

Following these principles, we can formulate the approximate polynomial GCD, the approximate matrix Jordan Canonical Form, the approximate irreducible factorization of polynomials, etc., as well-posed and often well-conditioned problems, so that it becomes feasible to calculate such approximate solutions using floating point arithmetic. More importantly, this formulation breaks the computation into two optimization process: maximizing the codimension of manifolds followed by minimizing the distance to the manifold, leading to a two-staged strategy for designing robust algorithms:

Stage I: Calculating the solution structure by finding the nearby manifold Π of highest codimension.

Stage II: Solving for the approximate solution by minimizing the distance from given problem to manifold Π.

The main mechanism at Stage I is matrix rank-revealing, while Stage II relies on solving nonlinear least squares problems.

3 Matrix computation and and matrix building tools

Finding the structure of the approximate solution at Stage I almost always involve matrix rank-revealing, whereas the minimization at Stage II can be accomplished using the Gauss-Newton iteration. For those considerations, the base level modules in `ApaTools` include matrix builders, rank-revealing tools, and the Gauss-Newton iteration routines.

Polynomials with a certain degree bound form a vector space $\mathbb{P}$ with a monomial basis $\{p_1, \cdots, p_n\}$ corresponding to a term order. The designated term order can be considered a linear mapping

$$\Psi \;:\; x_1^{j_1} x_2^{j_2} \cdots x_\ell^{j_\ell} \;\longrightarrow\; p_k$$

that forms an isomorphism between $\mathbb{P}$ and $\mathbb{C}^n$. Let $\Psi_1 : \mathbb{P}_1 \longrightarrow \mathbb{C}^m$ and $\Psi_2 : \mathbb{P}_2 \longrightarrow \mathbb{C}^n$ be isomorphisms and $\mathcal{L} : \mathbb{P}_1 \longrightarrow \mathbb{P}_2$ be a linear transformation. Then there is a matrix $A \in \mathbb{C}^{n\times m}$ that makes the following diagram commute:

$$\begin{tikzcd}
\mathbb{P}_1 \arrow[r, "\mathcal{L}"] \arrow[d, "\Psi_1"'] & \mathbb{P}_2 \\
\mathbb{C}^m \arrow[r, "A"] & \mathbb{C}^n \arrow[u, "\Psi_2^{-1}"']
\end{tikzcd}$$

Matrix $A = [\mathbf{a}_1, \cdots, \mathbf{a}_m]$ can then be genearated column-by-column in the loop as follows:

```
for $j = 1, 2, \cdots, m$ do
    – $p = \Psi_1^{-1}(\mathbf{e}_j)$
    – $q = \mathcal{L}(p)$
    – $\mathbf{a}_j = \Psi_2(q)$
end do
```

Here $\mathbf{e}_j$ is the j-th canonical vector in $\mathbb{C}^m$, namely the j-th column of the $m \times m$ identity matrix. Implementation of software modules for isomorphisms between polynomial vector spaces and $\mathbb{C}^n$ constitutes *matrix building tools* in `ApaTools`.

For instance, let $\mathbb{P}^k$ denote the vector space of polynomials with total degree less than or equal to k. For a fixed polynomimal f, polynomial multiplication with f is a linear transformation $\mathcal{L}_f : \mathbb{P}^n \longrightarrow \mathbb{P}^m$

$$\mathcal{L}_f(g) \;=\; f \cdot g \in \mathbb{P}^m \quad \text{for all } g \in \mathbb{P}^n \tag{3}$$

that corresponds to a convolution matrix $C_{n,m}(f)$. In `ApaTools`, we include a convenient module `LinearTransformMatrix` as a generic matrix builder for any linear transformation between vector spaces of polynomials. The user need provide a subroutine for the linear transformation as input for `LinearTransformMatrix`. Using the linear transformation in (3) as an example, we can write a simple Maple procedure for $\mathcal{L}_f$ with f as an input parameter:

```
> PolynMultiply := proc( g::polynom, x::{name,list}, f::polynom);
       return expand(f*g)
  end proc:
```

The convolution matrix $C_{2,3}(f)$ for $f(x, y) = x+2y+3$ can then be generated by a simple call:

```
> T := LinearTransformMatrix(PolynMultiply,[x+2*y+3],[x,y],2,3);
```

$$T := \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 \\ 0 & 2 & 1 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

In this example, the input `PolynMultiply` is the procedure name for $\mathcal{L}_f$, `[x+2*y+3]` is the list parameters for `PolynMultiply` besides `[x,y]` that stands for the list of variables, and `2, 3` are the degree bounds for the vector spaces $\mathbb{P}^n$ and $\mathbb{P}^m$ respectively.

With matrices being constructed, rank-revealing is frequently applied in approximate polynomial algebra. As a example, a polynomial pair $(f, g) \in \mathbb{P}^m \times \mathbb{P}^n$ having a nontrivial GCD can be written as $f = uv$ and $g = uw$ where $u = gcd(f, g)$ with cofactors v and w. Consequently, we have $fw - gv = 0$, or equivalently a homogeneous system of linear equations

$$\left[C_{n\text{-}k,m+n\text{-}k}(f), C_{m\text{-}k,m+n\text{-}k}(g)\right] \begin{bmatrix} \mathbf{w} \\ -\mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{v}$ and $\mathbf{w}$ are coefficient vectors of v and w respectively. As a result, finding the GCD structure is equivalent to computing the rank of the Sylvester matrix $\left[C_{n\text{-}k,m+n\text{-}k}(f), C_{m\text{-}k,m+n\text{-}k}(g)\right]$.

Another example is numerical elimination. For given polynomials f and and g in variables x and y, if there are polynomials p and q such that variable x is eliminated from polynomial $h = p \cdot f + q \cdot g$, then

$$\partial_x(pf + qg) = \left[(\partial_x f) + f \cdot \partial_x\right]p + \left[(\partial_x g) + g \cdot \partial_x\right]q = 0 \tag{4}$$

Since mapping $p \longrightarrow \left[(\partial_x f) + f \cdot \partial_x\right]p$ is a linear transformation, equation (4) can be converted to a homogeneous system of linear equations in matrix-vector form

$$M \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} = \mathbf{0}$$

that becomes a rank-revealing and kernel-finding problem for the elimination matrix M.

In *approximate* polynomial algebra, we seek the *approximate rank* and the *approximate kernel* of a matrix, in contrast to seeking exact rank and kernel in exact polynomial algebra. Following the same "three-strikes" principle, the approximate rank and kernel of a matrix $A \in \mathbb{C}^{m \times n}$ are the exact rank and kernel of a nearby matrix $B \in \mathbb{C}^{m \times n}$. This matrix B is the nearest matrix to A on the manifold Π_k that has the highest codimension among manifolds $\Pi_1, \Pi_2, \cdots$ passing through an ε-neighborhood of A, where Π_j is the set of all $m \times n$ matrices with rank j. Approximate rank/kernel can be efficiently computed using numerical rank-revealing algorithms [6, 8] that are implemented as modules of `ApaTools`.

4 Nonlinear least squares and the Gauss-Newton iteration

The minimization at Stage II is a nonlinear least squares problem that can be solved using the Gauss-Newton iteration on an overdetermined system of equations in the form of

$$\mathbf{f}(\mathbf{z}) = \mathbf{0} \quad \text{for} \ \mathbf{f} : \mathbb{C}^n \longrightarrow \mathbb{C}^m, \quad \mathbf{z} \in \mathbb{C}^n, \quad m \geq n. \tag{5}$$

We seek the least squares solution $\hat{\mathbf{z}}$ satisfying

$$\left\|\mathbf{f}(\hat{\mathbf{z}})\right\|^2 = \min_{\mathbf{z} \in \mathbb{C}^n} \left\{\left\|\mathbf{f}(\mathbf{z})\right\|^2\right\}$$

with a proper norm $\|\cdot\|$. Let $J(\mathbf{z})$ be the Jacobian of $\mathbf{f}(\mathbf{z})$. Then the minimum occurs at [16]

$$J(\hat{\mathbf{z}})^* \mathbf{f}(\hat{\mathbf{z}}) = \mathbf{0}$$

where $(\cdot)^*$ denotes the Hermitian transpose of matrix $(\cdot)$.

If $\mathbf{f}(\mathbf{z})$ is analytic, the Moore-Penrose inverse $J(\hat{\mathbf{z}})^+$ of $J(\hat{\mathbf{z}})$ exists, the minimum $\|\mathbf{f}(\hat{\mathbf{z}})\|$ is small, and the initial iterate $\mathbf{z}_0$ is close to $\hat{\mathbf{z}}$, then the Gauss-Newton iteration

$$\mathbf{z}_{k+1} \;=\; \mathbf{z}_k - J(\mathbf{z}_k)^+\mathbf{f}(\mathbf{z}_k), \quad k = 0, 1, \cdots \tag{6}$$

converges to $\hat{\mathbf{z}}$ [16].

The Gauss-Newton iteration is extensively used in `ApaTools`. A typical case is computing the approximate GCD: For given polynomial pair p and q with degrees m and n resectively, we seek a polynomial triplet $(\hat{u}, \hat{v}, \hat{w})$ such that

$$\|(p,q) - (\hat{u}\hat{v}, \hat{u}\hat{w})\|^2 \;=\; \min_{\deg(u)=k,\ \deg(uv)=\deg(p),\ \deg(uw)=\deg(q)} \left\{\|(p,q) - (uv, uw)\|^2\right\}.$$

Let $\mathbf{p}$, $\mathbf{q}$, $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ be the coefficient vectors of p, q, u, v, and w respectively, then the overdetermined system is constructed to have the least squares solution to $uv - p = 0$ and $uv - q = 0$ along with a proper scaling equation on u. This system can be written in matrix-vector form as

$$\mathbf{f}(\mathbf{u},\mathbf{v},\mathbf{w}) \;=\; \mathbf{0} \quad \text{for} \;\; \mathbf{f}(\mathbf{u},\mathbf{v},\mathbf{w}) \;=\; \begin{bmatrix} \mathbf{r}^*\mathbf{u} - 1 \\ C_{k,m}(v)\mathbf{u} - \mathbf{p} \\ C_{k,n}(w)\mathbf{u} - \mathbf{q} \end{bmatrix} \tag{7}$$

where $C_{j,l}(h)$ denotes the convolution matrices corresponding to the linear transformation $\mathcal{L}_h : g \longrightarrow h \cdot g \in \mathbb{P}^l$ on the vector space all polynomials $g \in \mathbb{P}^j$ (see §3) and $\mathbf{r}$ is a predetermined random vector. The Jacobian can be easily constructed as

$$J(\mathbf{u},\mathbf{v},\mathbf{w}) \;=\; \begin{bmatrix} \mathbf{r}^* & & \\ C_{k,m}(v) & C_{m\text{-}k,m}(u) & \\ C_{k,n}(w) & & C_{n\text{-}k,n}(u) \end{bmatrix}.$$

The Gauss-Newton iteration (6) can thus be applied accordingly.

It is essential to formulate the overdetermined system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ such that the Jacobian $J(\hat{\mathbf{z}})$ is injective by having enough equations. As a result, the Moore-Penrose inverse $J(\hat{\mathbf{z}})^+$ exists and the Gauss-Newton iteration is well-defined in a neighborhood of $\hat{\mathbf{z}}$. Moreover, the norm $\left\|J(\hat{\mathbf{z}})^+\right\|$ serves as a condition number of the approximate solution [13, 16, 19].

We provide a generic blackbox module `GaussNewton` for the general purpose Gauss-Newton iteration. This module requires the user to provide a routine for computing function $\mathbf{f}(\mathbf{z})$ in (5), a routine for computing the Jacobian $J(\mathbf{z})$ and an initial iterate $\mathbf{z}_0$ as its input. Then `GaussNewton` carries out the Gauss-Newton iteration and output the least squares solution $\hat{\mathbf{z}}$ along with the residual $\|\mathbf{f}(\hat{\mathbf{z}})\|_2$. As a convenient option, the Maple version of `GaussNewton` can forgo the requirement of providing Jacobian routine by computing $J(\mathbf{z})$ with Maple symbolic manipulation.

Using the above example of GCD computation, the generic Gauss-Newton module needs only a user defined subroutine for computing $\mathbf{f}(z)$ that can be as simple as

```
> GcdFunc := proc( z, m, n, p, q, r)
     local F, u, v, w;

     u, v, w := z[1..m], z[m+1..m+n], z[m+n+1..-1];
     F := <LinearAlgebra[DotProduct](r,u)-1, Convolution(u,v)-p, Convolution(u,w)-q>;

     return F[1..-1,1];
  end proc:
```

Here, the Maple routine `GcdFunc` returns the vector value of $\mathbf{f}(\mathbf{z})$ from input vector $\mathbf{z}$ consists of coefficients of u, v and w, along with parameters `m, n, p, q, r` representing the length of $\mathbf{u}$, the length of $\mathbf{v}$, coefficient vectors $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$, respectively. The command `Convolution(u,v)` and `Convolution(u,w)` produces coefficient vectors of polynomial products $u \cdot v$ and $u \cdot v$ respectively by `ApaTools` module `Convolution`. Then the Gauss-Newton iteration can be carried out by a simple call of `GaussNewton`:

```
> p, q := PolynomialTools[CoefficientVector](x^5-x^3+5*x^2-6*x+10,x),    # get coef. vectors p, q
          PolynomialTools[CoefficientVector](2*x^4+x^2-6,x);
> r := Vector([.4,0,.2]):                                                # define scaling vector r
> z0 := Vector([1.99,0.01,1.01,  4.99,-3.01,0,.99,  -3.01,0,1.99]):      # initial guess for u, v, w
> z,res := GaussNewton(GcdFunc,z0,[3,4,p,q,r],[1e-9,9,true]):            # Gauss-Newton iteration

        Gauss-Newton step   0,  residual =   8.00e-02
        Gauss-Newton step   1,  residual =   2.01e-04
        Gauss-Newton step   2,  residual =   3.16e-09
        Gauss-Newton step   3,  residual =   4.44e-16
```

The approximate GCD and cofactors can then be retrived from the result of the iteration:

```
> CoefficientVector2UnivariatePolynomial(z[1..3],x);
  CoefficientVector2UnivariatePolynomial(z[4..7],x);
  CoefficientVector2UnivariatePolynomial(z[8..-1],x);
```

$$2.000000000000000 + 3.1779793 \times 10^{-17}\, x + 1.000000000000000\, x^2$$

$$4.999999999999999 - 3.000000000000000\, x - 1.530197036 \times 10^{-16}\, x^2 + 0.9999999999999998\, x^3$$

$$-2.999999999999999 + 2.54808329 \times 10^{-17}\, x + 2.000000000000000\, x^2$$

that are accurate approximation to the exact GCD $u = x^2+2$, cofactors $v = x^3-3x+5$ and $w = 2x^2 - 3$. Notice that the Jacobian routine is not necessary as input for `GaussNewton`. This type of generic routines enables fast implementation of experimental algorithms.

5 `ApaTools` overview

1. **Base level tools**

   - `LinearTransformMatrix`: A generic module for constructing the matrix associated with a given linear transformation between vector spaces of polynomials, as described in §3.

   - `GaussNewton`: A generic module for carrying out the Gauss-Newton iteration as described in §4.

2. **Matrix computation tools**

- `NulVector`: The module for computing the smallest singular value and the corresponding left and right singular vectors. This module can be used to determine whether a matrix is rank deficient in approximate sense, and to serve as a submodule for `ApproxiRank` that computes the approximate rank and approximate Kernel.

- `ApproxiRank`: The module for computing the approximate rank $\text{rank}_\theta(A)$ and approximate kernel $\mathcal{K}_\theta(A)$ of a matrix A within a given threshold θ. Here the approximate rank is defined as

  $$\text{rank}_\theta(A) = \min_{\|B-A\|_2<\theta} \text{rank}(A)$$

  and the approximate kernel

  $$\mathcal{K}_\theta(A) = \mathcal{K}(B) \quad \text{for} \quad \|B-A\|_2 = \min_{\text{rank}(C)=\text{rank}_\theta(A)} \|C-A\|_2.$$

  Module `ApproxiRank` is efficient when the nullity of A is low. The standard singular value decomposition (SVD) may be more suitable if the approximate rank of A is around one half of the column dimension.

- `ApproxiJCF`: The module for computing the approximate Jordan Canonical Form (JCF) of a given matrix A within a threshold ε. Computing the exact Jordan Canonical Form is a ill-posed problem that requires exact data with symbolic computation. The approximate JCF is defined according to the same "three strikes" principle and computed with a two-staged algorithm elaborated in §2. As a result, module `ApproxiJCF` is capable of computing the JCF accurately even if the matrix is perturbed [19].

3. **Univariate polynomial tools**:

- `UvGCD`: The module for computing the approximate greatest common divisor $\mathscr{gcd}_\theta(f,g)$ of univariate polynomial pair (f,g) within a given threshold θ with definition as follows [13].

  Let $\mathbb{P}$ be the vector space of polynomial pairs (p,q) satisfying $\deg(p) \le \deg(f)$ and $\deg(q) \le \deg(g)$. Then $\Pi_j = \{(p,q) \in \mathbb{P} \mid \deg(\mathscr{gcd}(p,q)) = j\}$ is a manifold of codimension k in $\mathbb{P}$ associated with the metric topology induced by the vector 2-norm for $j = 0, 1, \cdots$. Let Π_k be the highest codimension manifold among $\Pi_0, \Pi_1, \cdots$ containing a polynomial pair within distance θ of (f,g). Then the approximage GCD $\mathscr{gcd}_\theta(f,g)$ is the exact GCD of pair (p,q), where (p,q) is the nearest polynomial pair on the manifold Π_k to the given pair (f,g).

  Module `UvGCD` accepts input polynomial pair (f,g) and threshold θ (optional) and outputs $u = \mathscr{gcd}_\theta(f,g)$, cofactors v and w along with the residual $\|(f,g) - (uv, uw)\|$, where the norm $\|\cdot\|$ is either the vector 2-norm chosen by users or the default weighted 2-norm.

- `ExtendGCD`: The module for computing a polynomial (p, q) such that $pf + qg = \mathscr{gcd}_\theta(f, g)$ for given polynomial f and g using the output of `UvGCD`. The extended GCD can be applied to transforming matrices of polynomial entries, and in particularly, to computing the Smith Normal Form [12, 18],

- `UvFactor`: The module for computing an approximate factorization

  $$p_0(x - z_1)^{m_1}(x - z_2)^{m_2} \cdots (x - z_k)^{m_k}$$

  for a given polynomial $p(x)$, as shown in Example 2. The factors with nontrivial multiplicities $m_j > 1$ can be calculated accurately by `UvFactor` without extending the machine precision even if the coefficients of $p(x)$ are perturbed [16].

4. **Multivariate polynomial tools**

- `MvGCD`: The module for computing the approximate GCD of a given pair of multivariate polynomials. The formulation of the multivariate approximate GCD is the same as the univariate case. The computation requires applying `UvGCD` repeatedly in determining the GCD structure before calling `GaussNewton` for solving a least squares problem that finding the distance from the given polynomial pair to a GCD manifold.

- `SquarefreeFactor`: The module for computing an approximate squarefree factorization of a given multivariate polynomial p. Here, again, the notion of the *approximate squarefree factorization* is formulated using the "three-strikes" principle. Module `SquarefreeFactor` produces two types of squarefree factorizations: a staircase squarefree factorization

  $$(p_1)^1(p_2)^2 \cdots (p_k)^k$$

  where $p_1, \cdots, p_k$ are coprime, or a flat-type squarefree factorization

  $$f_1 \cdot f_2 \cdots f_k$$

  where $f_j = p_j p_{j+1} \cdots p_k$. Each p_i or f_j is "squarefree", namely it has no repeated nontrivial factors of its own.

- `MultiplicityStructure`: The module for computing the multiplicity structure of a given polynomial system

  $$\left\{ \begin{array}{c} f_1(x_1, \cdots, x_s) = 0 \\ \vdots \\ f_t(x_1, \cdots, x_s) = 0 \end{array} \right. \tag{8}$$

  at a given zero $\mathbf{x}^* = (x_1^*, \cdots, x_s^*)$.

Let $\partial_{\mathbf{j}}$ denote a differential monomial

$$\partial_{\mathbf{j}} \equiv \frac{1}{j_1! \cdots j_s!} \frac{\partial^{j_1+\cdots+j_s}}{\partial x_1^{j_1} \cdots \partial x_s^{j_s}} \quad \text{for} \quad \mathbf{j} = [j_1, \cdots, j_s]\, \mathbb{N}^s .$$

A sequence of complex numbers $\mathbf{a} = \left\{ \alpha_{\mathbf{j}} \mid \mathbf{j} \in \mathbb{N}^s \right\}$ corresponds to a differential functional $a[\mathbf{x}^*]$ defined as

$$a[\mathbf{x}^*](f) = \sum_{\mathbf{j} \in \mathbb{N}^s} \alpha_{\mathbf{j}} \partial_{\mathbf{j}} f(\mathbf{x}^*)$$

for any polynomial f in the ideal $\mathcal{I} = \langle f_1, \cdots, f_t \rangle$. The vector space $\mathcal{D}_{\mathbf{x}^*}(\mathcal{I}) \equiv \left\{ \mathbf{a}[\mathbf{x}^*] \;\middle|\; \mathbf{a}[\mathbf{x}^*](f) = 0 \quad \text{for all} \quad f \in \langle f_1, \cdots, f_t \rangle \right\}$ is called the dual space of ideal $\mathcal{I}$ at $\mathbf{x}^*$. The dimension of the dual space $\mathcal{D}_{\mathbf{x}^*}(\mathcal{I})$ is the multiplicity of $\mathbf{x}^*$ as a zero to the system (8). The dual space itself constitutes the multiplicity structure of the system (8) at zero $\mathbf{x}^*$ [10].

The module `MultiplicityStructure` calculates the multiplicity, a basis for the the dual space along with other invariants such as breadth, depth and Hilbert function [3] even if the system and zero are inexact.

- `PolynomialEliminate`: The module for computing polynomials p, q and h such that $h = pf + qg$ belongs to a specified elimination ideal generated by polynomials f and g. For given f and g in variables $x_1, \cdots, x_s$, in other words, `PolynomialEliminate` eliminates a specified variable x_j in $h = pf + qg$ by solving the differential equation

$$\frac{\partial}{\partial x_j}(pf + qg) = 0$$

  via module `ApproxiKernel` on a sequence of matrices that are generated by module `LinearTransformMatrix`. Combined with module `MvGCD`, this elimination tool is particularly useful in solving polynomial systems whose solutions contain nonzero dimentional components [14].

References

[1] D. Bates, J. D. Hauenstern, A. J. Sommese, and C. W. Wampler, *Bertini: Software for Nmerical Algebraic Geometry.* http://www.nd.edu/∼sommese/bertini, 2006.

[2] D. J. Bates, C. Peterson, and A. J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, J. of Complexity, 22 (2006), pp. 475–489.

[3] B. Dayton and Z. Zeng, *Computing the multiplicity structure in solving polynomial systems.* Proceedings of ISSAC '05, ACM Press, pp 116–123, 2005.

[4] 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.

[5] T. Gao and T.-Y. Li, *MixedVol: A software package for mixed volume computation*, ACM Trans. Math. Software, 31 (2005), pp. 555–560.

[6] T.-L. Lee, T. Y. Li, and Z. Zeng, *A rank-revealing method with updating, downdating and applications, Part II.* submitted, 2006.

[7] T.-Y. Li, *Solving polynomial systems by the homotopy continuation method*, Handbook of Numerical Analysis, XI, edited by P. G. Ciarlet, North-Holand, Amsterdam (2003), pp. 209–304.

[8] T. Y. Li and Z. Zeng, *A rank-revealing method with updating, downdating and applications*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 918–946.

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

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

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

[12] J. Verschelde and Y. Wang, *Computing dynamic output feedback laws*, IEEE Trans. Automatic Control, (2004), pp. 1552–1571.

[13] Z. Zeng, *The approximate GCD of inexact polynomials. Part I.* Preprint, 2007.

[14] Z. Zeng, *A polynomial elimination method for symbolic and numerical computation.* Preprint, 2007.

[15] Z. Zeng, *Algorithm 835: Multroot – a Matlab package for computing polynomial roots and multiplicities*, ACM Trans. Math. Software, 30 (2004), pp. 218–235.

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

[17] 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).

[18] ——, *The approximate GCD of inexact polynomials. Part II.* Preprint, 2007.

[19] Z. Zeng and T. Y. Li, *A numerical method for computing the Jordan Canonical Form.* Preprint, 2007.