Symbolic-Numeric Computation
D. Wang and L. Zhi, Eds.
Trends in Mathematics, 269–278
© 2007 Birkhäuser Verlag Basel/Switzerland

Evaluation of Jacobian Matrices for Newton's Method with Deflation to Approximate Isolated Singular Solutions of Polynomial Systems

Anton Leykin, Jan Verschelde and Ailing Zhao

Abstract. For isolated singular solutions of polynomial systems, we can restore the quadratic convergence of Newton's method by deflation. The number of deflation stages is bounded by the multiplicity of the root. A preliminary implementation performs well in case only a few deflation stages are needed, but suffers from expression swell as the number of deflation stages grows. In this paper we describe how a directed acyclic graph of derivative operators guides an efficient evaluation of the Jacobian matrices produced by our deflation algorithm. We illustrate how the symbolic-numeric deflation algorithm can be used within *PHCmaple* interfacing Maple with PHCpack.

Mathematics Subject Classification (2000). Primary 65H10; Secondary 14Q99, 68W30.

Keywords. Deflation, evaluation, isolated singular solution, Jacobian matrix, Newton's method, numerical homotopy algorithm, polynomial system, reconditioning, symbolic-numeric computation.

1. Introduction

Newton's method slows down when approaching a singular root and convergence may even be lost if the working precision is not high enough. Moreover, using multiprecision arithmetic makes sense only if the input data is sufficiently precise. To restore the quadratic convergence of Newton's method, T. Ojika, S. Watanabe, and T. Mitsui ([11], see also [10]) developed a numerical deflation algorithm. A symbolic algorithm to restore the quadratic convergence of Newton's method was developed by Lecerf [6].

This material is based upon work supported by the National Science Foundation under Grant No. 0134611 and Grant No. 0410036.

We studied the methods proposed in [11] and [10] and – for the sake of numerical stability – implemented and experimented with two modifications to their deflation algorithm:

1. Instead of *replacing* equations of the original system by conditions from derivatives of the system, we propose to *add* equations, introducing random constants for uniform treatment.
2. Instead of using Gaussian Elimination, we propose to apply Singular Value Decomposition (SVD) to determine the numerical rank[1] of the Jacobian matrix. The threshold on the numerical rank is our only critical numerical parameter to decide whether to deflate or not to deflate.

In particular, if the numerical rank of the Jacobian matrix at the current approximation equals R, we introduce $R+1$ additional variables which serve as multipliers to selections of random combinations of columns of the Jacobian matrix (first presented in [18]). In [8] we showed that no more than $m-1$ successive deflations are needed to restore the quadratic convergence of Newton's method converging to an isolated root of multiplicity m. Once the precise location of the isolated singularity is known, numerical techniques allow the calculation of the multiplicity structure, using the methods in [1], [2], or [12] (see also [13, 16]).

The paper [2] analyzes our deflation algorithm from a different perspective, showing that our algorithm produces differentials in the dual space as a by-product. For the important special case when the Jacobian matrix has corank one, a modification of the deflation algorithm is presented in [2], which mitigates the expential growth of the size of the matrices.

Our modifications to the methods of [11] and [10] were first [18] developed in Maple, exploiting its facilities for polynomial manipulations and convenient multi-precision arithmetic, and then implemented in PHCpack [17]. While the performance of the method is promising on selected applications (such as the fourfold isolated roots of the cyclic 9-roots problem, see [8]), the method suffers from expression swell after a couple of deflations. In this paper, we describe a way to "unfold" the extra multiplier variables, exploiting the special structure of the matrices which arise in the deflation process. In the unfolding process, we naturally arrive at trees (or more precisely, directed acyclic graphs), also employed in [4, 5].

Our Jacobian matrices have a particular sparse block structure which should be also be exploited when computing the numerical rank and solving the linear system. For this, we recommend the recent rank-revealing algorithms in [9].

Our algorithm is a symbolic-numeric algorithm; perhaps, the term "numeric-symbolic" is more appropriate, as we produce by numerical means new equations which regularize or recondition the problem. In particular, our algorithm gives – in addition to more accurate values for the coordinates of the solution – a new system of polynomial equations for which the isolated singular solution is a regular root. Like [14], this paper documents the recent improvements to PHCpack [17].

[1]The determination of the numerical rank is a well studied problem in numerical linear algebra, for recent progress we refer to [3] and [9].

We encounter singular solutions when we solve polynomial systems using homotopy continuation methods, see e.g. [15]. This encounter can only happen – with probability one – at the end of the solution paths defined by the homotopy. So homotopy continuation methods deliver approximate solutions to the singularities which are then reconditioned by the deflation algorithm. One future project is the use of deflation to decide *locally* whether a solution at the end of a path is isolated or lies on a positive dimensional solution set. For this problem, deflation is needed because the solution set might be multiple.

2. Problem Statement

In this section we fix the notation. Consider $f(\mathbf{x}) = \mathbf{0}$, a system of N polynomial equations in n unknowns, with isolated multiple root $\mathbf{x}^*$. As we restrict ourselves to isolated roots, we have $N \geq n$. At $\mathbf{x} = \mathbf{x}^*$, the Jacobian matrix $A(\mathbf{x})$ of f has rank $R < n$. At stage k in the deflation, we have the system

$$f_k(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1}, \boldsymbol{\lambda}_k) = \left\{ \begin{array}{rcl} f_{k-1}(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1}) & = & 0 \\ A_{k-1}(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1}) B_k \boldsymbol{\lambda}_k & = & 0 \\ \mathbf{h}_k \boldsymbol{\lambda}_k & = & 1 \end{array} \right. \quad (1)$$

with $f_0 = f$, $A_0 = A$, and where $\boldsymbol{\lambda}_k$ is a vector of $R_k + 1$ multiplier variables, $R_k = \text{rank}(A_{k-1}(\mathbf{x}^*))$, scaled using a random vector $\mathbf{h}_k \in \mathbb{C}^{R_k+1}$. The matrix B_k is a random matrix with as many rows as the number of variables in the system f_{k-1} and with $R_k + 1$ columns. The Jacobian matrix of f_k is A_k. We have the following relations

$$\#\text{rows in } A_k: \; N_k = 2N_{k-1} + 1, \; N_0 = N, \quad (2)$$

$$\#\text{columns in } A_k: \; n_k = n_{k-1} + R_k + 1, \; n_0 = n. \quad (3)$$

The second line in (1) requires the multiplication of N_{k-1}-by-n_{k-1} polynomial matrix A_{k-1} with the random n_{k-1}-by-$R_k + 1$ matrix B_k, with the vector of $R_k + 1$ multiplier variables $\boldsymbol{\lambda}_k$.

If the evaluation of $A_{k-1} B_k \boldsymbol{\lambda}_k$ is done symbolically, i.e.: if we first compute the polynomial matrix $A_{k-1}(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1}) B_k$ before we give values to $(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1})$, the expression swell will cause the evaluation to be very expensive. In this paper we describe how to first evaluate the Jacobian matrices before the matrix multiplications are done. As this technical description forms a blueprint for an efficient implementation, it also sheds light on the complexity of the deflation.

3. Unwinding the Multipliers

We observe in (1) that the multiplier variables in $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_k$ all occur linearly. The Jacobian matrix of f_k in (1) has a nice block structure which already separates

the linearly occurring $\boldsymbol{\lambda}_k$ from the other variables:

$$A_k(\mathbf{x}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{k-1}, \boldsymbol{\lambda}_k) = \begin{bmatrix} A_{k-1} & \mathbf{0} \\ \left[\frac{\partial A_{k-1}}{\partial \mathbf{x}} \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_1} \cdots \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_{k-1}}\right] B_k \boldsymbol{\lambda}_k & A_{k-1} B_k \\ \mathbf{0} & \mathbf{h}_k \end{bmatrix}, \quad (4)$$

where the partial derivatives of an N-by-n matrix A with respect to a vector of variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ give rise to a vector of matrices:

$$\frac{\partial A}{\partial \mathbf{x}} = \left[\frac{\partial A}{\partial x_1} \; \frac{\partial A}{\partial x_2} \; \cdots \; \frac{\partial A}{\partial x_n}\right], \quad \frac{\partial A}{\partial x_k} = \left[\frac{\partial a_{ij}}{\partial x_k}\right] \text{ for } A = [a_{ij}], \; \begin{matrix} i=1,2,\ldots,N, \\ j=1,2,\ldots,n, \\ k=1,2,\ldots,n. \end{matrix} \quad (5)$$

The definition of $\frac{\partial A}{\partial \mathbf{x}}$ naturally satisfies the relation $\frac{\partial}{\partial \mathbf{x}}(A\mathbf{x}) = A$.

Notice that in (4) the evaluation of

$$\left[\frac{\partial A_{k-1}}{\partial \mathbf{x}} \; \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_1} \; \cdots \; \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_{k-1}}\right] B_k \boldsymbol{\lambda}_k \quad (6)$$

yields the matrix

$$\left[\frac{\partial A_{k-1}}{\partial \mathbf{x}} B_k \boldsymbol{\lambda}_k \; \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_1} B_k \boldsymbol{\lambda}_k \; \cdots \; \frac{\partial A_{k-1}}{\partial \boldsymbol{\lambda}_{k-1}} B_k \boldsymbol{\lambda}_k\right] \quad (7)$$

which has the same number of columns as A_{k-1}.

As we started this section observing the separation of $\boldsymbol{\lambda}_k$ in the block structure of A_k, we can carry this further through to all multiplier variables. By "unwinding" the multipliers, we mean the removal of the $\frac{\partial}{\partial \boldsymbol{\lambda}}$-type derivatives. In the end, we will only be left with derivatives of the variables $\mathbf{x}$ which are the only variables which may occur nonlinearly in the given system f.

In the k-th deflation stage, we will then need $\frac{\partial A}{\partial \mathbf{x}}, \frac{\partial^2 A}{\partial \mathbf{x}^2}, \ldots, \frac{\partial^k A}{\partial \mathbf{x}^k}$. If we execute (5) blindly, with disregard of the symmetry, we end up with n^k matrices, e.g.: for $n = 3$ and $k = 10$, we compute 59,049 3-by-3 polynomial matrices, which is quite discouraging. However, as $\frac{\partial^2 A}{\partial x_1 \partial x_2} = \frac{\partial^2 A}{\partial x_2 \partial x_1}$, we enumerate all different monomials in n variables of degree k which is considerably less than n^k, e.g.: for $n = 3$ and $k = 10$ again, we now only have 66 distinct polynomial matrices to compute. To store $\frac{\partial A}{\partial \mathbf{x}}$, a tree with n children (some branches may be empty) is a natural data structure, see [4, 5].

4. A Directed Acyclic Graph of Jacobian Matrices

In this section, we explain by means of examples, how we build our data structures.

For example, consider $k = 2$:

$$A_2(\mathbf{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = \begin{bmatrix} A_1 & \mathbf{0} \\ \left[\frac{\partial A_1}{\partial \mathbf{x}} \; \frac{\partial A_1}{\partial \boldsymbol{\lambda}_1}\right] B_2 \boldsymbol{\lambda}_2 & A_1 B_2 \\ \mathbf{0} & \mathbf{h}_2 \end{bmatrix}. \quad (8)$$

The evaluation of A_2 requires

$$A_1(\mathbf{x}, \boldsymbol{\lambda}_1) = \begin{bmatrix} A & \mathbf{0} \\ \left[\frac{\partial A}{\partial \mathbf{x}}\right] B_1 \boldsymbol{\lambda}_1 & A B_1 \\ \mathbf{0} & \mathbf{h}_1 \end{bmatrix} \tag{9}$$

and the derivatives

$$\frac{\partial A_1}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial A}{\partial \mathbf{x}} & \mathbf{0} \\ \left[\frac{\partial^2 A}{\partial \mathbf{x}^2}\right] B_1 \boldsymbol{\lambda}_1 & \left[\frac{\partial A}{\partial \mathbf{x}}\right] B_1 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \frac{\partial A_1}{\partial \boldsymbol{\lambda}_1} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \left[\frac{\partial A}{\partial \mathbf{x}}\right] * B_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{10}$$

For $\frac{\partial A}{\partial \mathbf{x}}$ as in (5), the product $\left[\frac{\partial A}{\partial \mathbf{x}}\right] B_1$ is $\left[\frac{\partial A}{\partial x_1} B_1 \ \frac{\partial A}{\partial x_2} B_1 \ \cdots \frac{\partial A}{\partial x_n} B_1\right]$. On the other hand, $\left[\frac{\partial A}{\partial \mathbf{x}}\right] * B_1$ uses the operator $*$ which treats its second argument as a vector of columns, i.e. if $B_1 = (b_1, b_2, ..., b_m)$ where b_i $(1 \le i \le m = R_1 + 1)$ are the columns, then

$$\left[\frac{\partial A}{\partial \mathbf{x}}\right] * B_1 = \left[\frac{\partial A}{\partial \mathbf{x}} b_1 \ \frac{\partial A}{\partial \mathbf{x}} b_2 \ \cdots \frac{\partial A}{\partial \mathbf{x}} b_m\right]. \tag{11}$$

One may evaluate $\left[\frac{\partial A}{\partial \mathbf{x}}\right] * B_1$ by computing the product $\left[\frac{\partial A}{\partial \mathbf{x}}\right] B_1$ followed by alternatingly permuting the columns of the result. By virtue of this operator $*$, we may write

$$\frac{\partial}{\partial \boldsymbol{\lambda}_1}\left(\left[\frac{\partial A}{\partial \mathbf{x}}\right] B_1 \boldsymbol{\lambda}_1\right) = \left[\frac{\partial A}{\partial \mathbf{x}}\right] * \frac{\partial}{\partial \boldsymbol{\lambda}_1}(B_1 \boldsymbol{\lambda}_1) = \left[\frac{\partial A}{\partial \mathbf{x}}\right] * B_1. \tag{12}$$

All matrices share the same typical structure: the critical information is in the first two entries of the first column. To evaluate A_2, the matrices we need to store are the Jacobian matrix A of the given system and its derivatives $\frac{\partial A}{\partial \mathbf{x}}$ and $\frac{\partial^2 A}{\partial \mathbf{x}^2}$. The role of the multipliers $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ in the evaluation is strictly linear, they occur only in matrix-vector products.

For $k = 3$, the evaluation of

$$A_3(\mathbf{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3) = \begin{bmatrix} A_2 & \mathbf{0} \\ \left[\frac{\partial A_2}{\partial \mathbf{x}} \ \frac{\partial A_2}{\partial \boldsymbol{\lambda}_1} \ \frac{\partial A_2}{\partial \boldsymbol{\lambda}_2}\right] B_3 \boldsymbol{\lambda}_3 & A_2 B_3 \\ \mathbf{0} & \mathbf{h}_3 \end{bmatrix} \tag{13}$$

requires the evaluation of $A_2(\mathbf{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ – which we considered above – and its partial derivatives $\frac{\partial A_2}{\partial \mathbf{x}}$, $\frac{\partial A_2}{\partial \boldsymbol{\lambda}_1}$, $\frac{\partial A_2}{\partial \boldsymbol{\lambda}_2}$ respectively listed below:

$$\frac{\partial A_2}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial A_1}{\partial \mathbf{x}} & \mathbf{0} \\ \left[\frac{\partial^2 A_1}{\partial \mathbf{x}^2} \ \frac{\partial^2 A_1}{\partial \mathbf{x} \partial \boldsymbol{\lambda}_1}\right] B_2 \boldsymbol{\lambda}_2 & \left[\frac{\partial A_1}{\partial \mathbf{x}}\right] B_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{14}$$

$$\frac{\partial A_2}{\partial \boldsymbol{\lambda}_1} = \begin{bmatrix} \frac{\partial A_1}{\partial \boldsymbol{\lambda}_1} & \mathbf{0} \\ \left[\frac{\partial^2 A_1}{\partial \boldsymbol{\lambda}_1 \partial \mathbf{x}} \ \frac{\partial^2 A_1}{\partial \boldsymbol{\lambda}_1^2}\right] B_2 \boldsymbol{\lambda}_2 & \left[\frac{\partial A_1}{\partial \boldsymbol{\lambda}_1}\right] B_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{15}$$

and

$$\frac{\partial A_2}{\partial \boldsymbol{\lambda}_2} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \left[\frac{\partial A_1}{\partial \mathbf{x}} \;\; \frac{\partial A_1}{\partial \boldsymbol{\lambda}_1}\right] * B_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{16}$$

As the multipliers occur linearly, we have that $\frac{\partial^2 A_1}{\partial \boldsymbol{\lambda}_1^2} = \mathbf{0}$. Observing $\frac{\partial^2 A_1}{\partial \boldsymbol{\lambda}_1 \partial \mathbf{x}} = \frac{\partial^2 A_1}{\partial \mathbf{x} \partial \boldsymbol{\lambda}_1}$, we have two more matrices to compute:

$$\frac{\partial^2 A_1}{\partial \mathbf{x}^2} = \begin{bmatrix} \frac{\partial^2 A}{\partial \mathbf{x}^2} & \mathbf{0} \\ \left[\frac{\partial^3 A}{\partial \mathbf{x}^3}\right] B_1 \boldsymbol{\lambda}_1 & \left[\frac{\partial^2 A}{\partial \mathbf{x}^2}\right] B_1 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \text{ and } \frac{\partial^2 A_1}{\partial \mathbf{x} \partial \boldsymbol{\lambda}_1} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \left[\frac{\partial^2 A}{\partial \mathbf{x}^2}\right] * B_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{17}$$

Thus the evaluation of $A_3(\mathbf{x}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3)$ requires $\left[A \;\; \frac{\partial A}{\partial \mathbf{x}} \;\; \frac{\partial^2 A}{\partial \mathbf{x}^2} \;\; \frac{\partial^3 A}{\partial \mathbf{x}^3}\right]$. The partial derivatives with respect to the multiplier variables in $\boldsymbol{\lambda}_1$, $\boldsymbol{\lambda}_2$, and $\boldsymbol{\lambda}_3$ do not need to be computed explicitly.

Just as a tree is a natural data structure to store the derivatives of A, a tree is used to represent the deflation matrices. For memory efficiency, the same node should only be stored once and a remember table is used in the recursive creation of the tree, which is actually a directed acyclic graph, see Fig. 1.

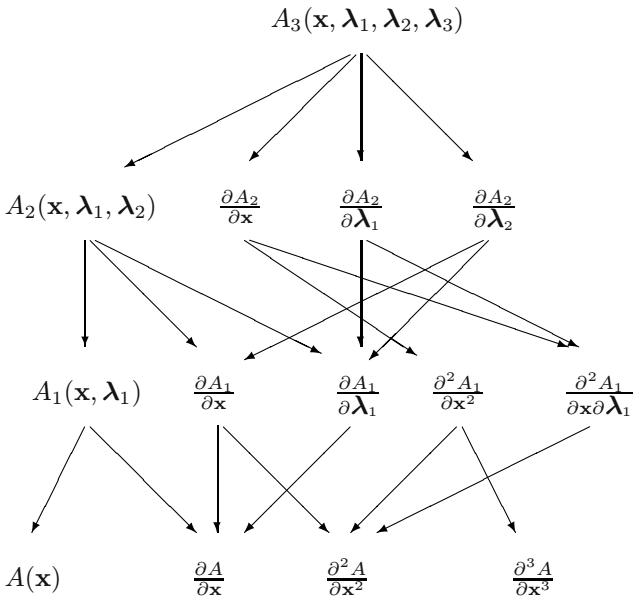

FIGURE 1. Directed acyclic graph illustrating the dependency relations in the evaluation of A_3.

The graph in Fig. 1 has 14 nodes. If we perform the deflation three times on a 10-by-10 system (10 equations in 10 variables), each time with the corank 1 (worst case scenario: maximal number of multipliers), then A_3 would be a 87-by-80 matrix and the graph of evaluated matrices would occupy about 347Kb (8 bytes for one complex number of two doubles). In case the rank would always be zero (best case: only one multiplier in each case), then A_3 would be a 87-by-13 matrix and the graph of evaluated matrices would occupy about 257Kb.

TABLE 1. Growth of the number of distinct derivative operators, as nodes in a directed acyclic graph, for increasing deflations k. The case $k = 3$ is displayed in Fig. 1.

k	1	2	3	4	5	6	7	8	9	10
#nodes	3	7	14	26	46	79	133	221	364	596

The size of the graph grows modestly, see Table 1. For example, for $k = 10$, we have 596 nodes. On the other hand, we must be careful to remove evaluated matrices when no longer needed. For $k = 10$ on a 10-by-10 system we would need at least 206Mb in the best case and 8Gb if the corank is always one.

We conclude this section with an anecdotal reference to actual timings, done on a 2.4 Ghz linux workstation, with the system cyclic 9-roots, illustrating that even for a modest number of deflations, it pays off to use a directed acyclic graph. Only to show the benefits of an efficient evaluation, we deflate twice, assuming the corank is one each time, and end up with a 39-by-36 Jacobian matrix. It takes only 30 milliseconds to evaluate this matrix using the pre-calculated directed acyclic graph, which takes less than a second to set up. Computing the symbolic representation of this Jacobian matrix in a straightforward manner and evaluating it takes 25 minutes and 40 seconds!

5. The Deflation Algorithm in *PHCmaple*

Our Maple package *PHCmaple* [7] provides a convenient interface to the functions of PHCpack. The interface exploits the benefits of linking computer algebra with numerical software. *PHCmaple* is a first step in a larger project aimed at integration of a numerical solver in a computer algebra system.

Below we give an example of using the *PHCmaple* function `deflationStep` to deflate the system of equations `Ojika1` (copied from [10] and member of our benchmark collection in [8]):

$$\begin{cases} x^2 + y - 3 = 0 \\ x + 0.125y^2 - 1.5 = 0. \end{cases} \tag{18}$$

The system has two isolated solutions: $(x, y) = (-3, -6)$ which is regular and $(x, y) = (1, 2)$ which is multiple. The function `deflationStep` takes a system and a list of solutions approximations as parameters, constructs the deflated systems

for all multiple roots in the list, and returns the list of lists of solutions. The latter are being grouped according to the rank of the Jacobian at the points, since for the points with the same rank a common deflated system may be constructed.

```
> T := makeSystem([x,y],[],[x^2+y-3, x+0.125*y^2-1.5]):
> sols := solve(T):
> printSolutions(T,sols);
```

```
    (1) [x = 1.0000+.44913e-5*I, y = 2.0000-.89826e-5*I]

    (2) [x = -3.0, y = -6.0]

    (3) [x = 1.0000+.60400e-5*I, y = 2.0000-.12080e-4*I]

    (4) [x = 1.0000-.38258e-5*I, y = 2.0000+.76515e-5*I]
```

To each group of points corresponds a table:

```
> l := map(i->table(i),deflationStep(sols,T)):
```

Solution (2) is simple (the rank of its Jacobian is full), the cluster of (1),(3),(4) approximates a multiple solution (rank $= 1 < 2$)

```
> map(g->[rank=g["rank"],num_points=nops(g["points"])],l);
```

$$[[rank = 2,\ num\_points = 1],\ [rank = 1,\ num\_points = 3]]$$

The multipliers used in the deflation step are

```
> multipliers := l[2]["multipliers"];
```

$$multipliers := [\lambda_1,\ \lambda_2]$$

The deflated system is linear in the multipliers (the matrix below is a part of (1) for k=1)

```
> DT := l[2]["deflated system"]:
> eqns :=
> map(p->p=0,DT:-polys[nops(T:-polys)+1..nops(DT:-polys)]):
> matrix(2,1,[A[0]*B[1],h[1]])=evalf[3](linalg[genmatrix](eqns,
> multipliers, b));
```

$$\begin{bmatrix} A_0\, B_1 \\ h_1 \end{bmatrix} = \begin{bmatrix} -1.51\,x - 1.31\,I\,x + 0.786 - 0.618\,I & 0.911\,x + 1.78\,I\,x - 0.0562 + 0.998\,I \\ 0.197\,y - 0.154\,I\,y - 0.755 - 0.656\,I & -0.0140\,y + 0.250\,I\,y + 0.455 + 0.890\,I \\ -0.681 + 0.732\,I & -0.115 - 0.993\,I \end{bmatrix}$$

```
> DTmu := subsVariables
      (DT,[seq(lambda[i]=mu[i],i=1..l[2]["rank"]+1)]):
> DTmu:-vars;
```

$$[x,\ y,\ \mu_1,\ \mu_2]$$

```
> g := table(deflationStep(l[2]["points"],DTmu)[1]):
> printSolutions(g["deflated system"],g["points"]);
```

```
(1) [x = 1.0-.87360e-15*I, y = 2.0+.18332e-14*I,
   mu[1] = -1.2402-.53612e-1*I, mu[2] = -.87951+.15347e-1*I,
   lambda[1] = -.85183-.21804*I, lambda[2] = -.91795+.57730*I,
   lambda[3] = .64096-.77868*I, lambda[4] = -.21656-.75758*I]
```

Get the multiplicity of the multiple solution:

```
> mult_solution := g["points"][1]: mult_solution:-mult;
```

$$3$$

Compare conditioning to that of the original approximation.

```
> mult_solution:-rco , sols[1]:-rco;
```

$$0.02542,\ 0.9301\,10^{-11}$$

Above we have seen that the powerful symbolic-numeric capabilities of Maple are useful for presenting the original and deflated systems in a compact, non-expanded form. In the future we plan to represent the sequence of consecutive deflations simply by the sequences of matrices and vectors A_k, B_k, $\mathbf{h}_k$. Combined with the approach presented here, this would lead to a more efficient evaluation, as well as provide a better control over the deflation process to the user.

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. *J. Complexity* 22(4): 475–489, 2006.

[2] B. H. Dayton and Z. Zeng. Computing the multiplicity structure in solving polynomial systems. In M. Kauers, editor, *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation*, pages 116–123. ACM, 2005.

[3] R. D. Fierro and P. C. Hansen. Utv expansion pack: Special-purpose rank-revealing algorithms. *Numerical Algorithms* 40(1): 47–66, 2005.

[4] P. Kunkel. Efficient computation of singular points. *IMA J. Numer. Anal.* 9: 421–433, 1989.

[5] P. Kunkel. A tree-based analysis of a family of augmented systems for the computation of singular points. *IMA J. Numer. Anal.* 16: 501–527, 1996.

[6] G. Lecerf. Quadratic Newton iteration for systems with multiplicity. *Found. Comput. Math.* 2: 247–293, 2002.

[7] A. Leykin and J. Verschelde. PHCmaple: A Maple interface to the numerical homotopy algorithms in PHCpack. In Q.-N. Tran, editor, *Proceedings of the Tenth International Conference on Applications of Computer Algebra* (ACA'2004), pages 139–147, 2004.

[8] A. Leykin, J. Verschelde and A. Zhao. Newton's method with deflation for isolated singularities of polynomial systems. *Theoretical Computer Science* 359(1-3): 111–122, 2006.

[9] T. Y. Li and Z. Zeng. A rank-revealing method with updating, downdating and applications. *SIAM J. Matrix Anal. Appl.* 26(4): 918–946, 2005.

[10] T. Ojika. Modified deflation algorithm for the solution of singular problems. I. A system of nonlinear algebraic equations. *J. Math. Anal. Appl.* 123, 199–221, 1987.

[11] T. Ojika, S. Watanabe and T. Mitsui. Deflation algorithm for the multiple roots of a system of nonlinear equations. *J. Math. Anal. Appl.* 96: 463–479, 1983.

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

[13] H. J. Stetter and G. T. Thallinger. Singular systems of polynomials. In O. Gloor, editor, *Proceedings of ISSAC 1998*, pages 9–16. ACM, 1998.

[14] A. J. Sommese, J. Verschelde and C. W. Wampler. Numerical irreducible decomposition using PHCpack. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, pages 109–130. Springer-Verlag, 2003.

[15] A. J. Sommese and C. W. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.

[16] G. T. Thallinger. *Zero Behavior in Perturbed Systems of Polynomial Equations.* PhD Thesis, Tech. Univ. Vienna, 1998.

[17] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software* 25(2): 251–276, 1999. Software available at `http://www.math.uic.edu/~jan`.

[18] J. Verschelde and A. Zhao. Newton's method with deflation for isolated singularities. Poster presented at ISSAC'04, 6 July 2004, Santander, Spain. Available at `http://www.math.uic.edu/~jan/Talks/poster.pdf`.

Anton Leykin, Jan Verschelde and Ailing Zhao
University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA
e-mail: `leykin@math.uic.edu, URL: www.math.uic.edu/~leykin`
`jan@math.uic.edu, URL: www.math.uic.edu/~jan`
`azhao1@math.uic.edu, URL: www.math.uic.edu/~azhao1`