under-determined systems), and characterize the components or manifolds of solutions of such systems. This is part of the rapidly developing area of Numerical Algebraic Geometry initiated in [23]. This yields new methods for problems which have been traditionally approached with symbolic methods from Computer Algebra, such as factorization [3], Gröbner bases and the completion of systems of partial differential equations. We have extended this work to systems of differential equations in [14], and initiated a study of Numerical Jet Geometry, using homotopy methods.

Despite the availability of very well developed implementations for homotopy continuation methods [26,12] surprisingly little has been implemented in the context of computer algebra systems for numerical solutions for polynomial systems. We note that, for example, Gröbner bases in *Maple* are limited to polynomials with rational coefficients. In *Maple*, the existing solvers focus on univariate equations. Even when working with a powerful Polynomial Homotopy Continuation package (in our case we have extensively used Verschelde's PHCpack [26]) we found the ability to perform experiments and try out ideas in a rich environment such as *Maple* to be a valuable asset. The work we discuss here represents a starting point for *Maple*, since many of the other standard algorithms of Numerical Algebraic Geometry (such as the computation of mixed volumes) still are not implemented in that context.

Existing Homotopy implementations in *Maple* include the univariate program of Fee [6]. In that work Fee truncates the Riemann zeta function, and uses a very efficient homotopy method he has developed for analytic functions to find roots of this truncated function in a given domain. Root counts are verified by using Cauchy's integral formula, using numerical quadrature, around the boundary of the domain. Kotsireas [11] has developed a multivariate fixed step homotopy method in *Maple*.

We have implemented a variable step homotopy continuation method in *Maple*, both for second and third orders, using the code of Smith [18] as a starting point. We compare the methods, and apply them to a variety of problems arising in polynomial system solving. For scalar functions, higher-order schemes are often called Halley methods [7], because of Halley's discovery in Newton's era. Higher-order schemes allow more rapid convergence and larger step sizes in processes such as homotopy solution techniques.

In this paper, we first present the higher order method for solving a single scalar equation. Then in the next section we apply it to systems, and extend it to a homotopy method. In the applications section we apply

it to some well-known examples having finitely many roots. Finally we give the first published example of a method using homotopy continuation to identify the missing constraints in a nonlinear system of PDE.

2. Iterative schemes

Newton's method to find solutions of a single nonlinear equation $f(x) = 0$ is well known; it is also well known that the method is second order and that higher-order methods have been derived [25,7]. Here we start by giving a uniform treatment of the higher-order scalar schemes, as a preparation for the vector case.

Consider solving the scalar equation $f(x) = 0$, given an initial estimate x_0 for the solution. We expand $f(x)$ as a Taylor series around x_0

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \tfrac{1}{2}(x - x_0)^2 f''(x_0) + \cdots. \tag{2.1}$$

Setting $\Delta = x - x_0$ and assuming $f(x) = 0$, we can solve for Δ by series reversion. Abbreviating $f(x_0)$ to f for clarity, gives

$$\Delta = -\frac{1}{f'}f - \frac{f''}{2(f')^3}f^2 + \frac{3(f'')^2 - f'f'''}{6(f')^5}f^3 + \cdots. \tag{2.2}$$

The series is written as shown to emphasize that it is a series in powers of $f(x_0)$, where $f(x_0)$ will be small in some sense when x_0 is close to the root being sought. The classical Newton iteration is obtained by taking one term of this series; taking two terms gives the third-order scheme

$$\Delta = -\frac{f}{f'} - \frac{f''f^2}{2(f')^3}, \tag{2.3}$$

which has been called Chebyshev's method. The Halley form of (2.3) is

$$\Delta = -\frac{f}{f' - \frac{1}{2}ff''/f'}. \tag{2.4}$$

One derivation of this form solves (2.1) by writing $0 = f + f'\Delta + \frac{1}{2}f''$

and solve this equation for Δ. None of the methods above can be applied at a point x_0 where $f'(x_0) = 0$, and Halley's method cannot be used for a function satisfying $2(f')^2 - ff'' = 0$, which means any function of the form $f(x) = 1/(Ax + B)$.

For the vector case, we use Cartesian-tensor notation [9]. When applying these results to homotopy methods, we shall give equivalent results in vector-matrix notation. Let $f : \mathbb{R}^m \to \mathbb{R}^m$ be a vector function, with component functions f_i. Let f depend upon the vector x, which in turn has components x_j. We wish to solve $f_i(x) = 0$, starting from an initial estimate $x^{(0)}$. We direct the reader to the literature where a multivariate Halley method of the type below is given [5].

The Taylor series for f about $x^{(0)}$ can be written using $\Delta_j = x_j - x_j^{(0)}$:

$$f_i(x^{(0)}) = f_i(x^{(0)}) + f_{i,k}(x^{(0)})\Delta_k + \tfrac{1}{2} f_{i,kh}(x^{(0)})\Delta_k \Delta_h + \cdots. \qquad (2.5)$$

Let $\check{f}_{ki}$ be the inverse of $f_{i,k}$, defined by $\check{f}_{ki} f_{i,j} = \delta_{kj}$, where δ_{kj} is the

basin of attraction. If an estimate $x^{(0)}$ is sufficiently close to an isolated non-singular root $x^{(e)}$, then with each iteration a second-order method will approximately double the number of digits that are correct, while a third-order method will triple that number [8]. Thus, for example, an estimate that is correct to 1 digit can be improved to 8 digits in 3 second-order steps or 2 third-order steps. Since the computation of the second derivative term is often expensive, in the scalar case, a higher-order method is usually not an advantage. However, in the vector case, an iteration requires a matrix inverse, making the iteration more expensive. In addition higher order derivatives for polynomial systems can be cheaply obtained (e.g. by automatic differentiation [4]) and this opens the possibility that the third-order method will be more efficient.

If an initial estimate $x^{(0)}$ is further away from the root, and the convergence theorems do not apply, then we must consider the basins of attraction of the root. Graphical presentations of how particular basins of attraction change with the iterative scheme have been published recently [25]. We expect the basin of attraction to be larger for higher-order methods, but have yet to investigate this.

3. Homotopy Method

Consider a system of equations $p(x) = 0$, which we wish to solve. Both p and x are vectors. Suppose we possess a system of equations q

An appropriate start system is now described for the homotopy. Let $p(x) = (p_1(x), \ldots, p_n(x)) = 0$ denote the system of n polynomial equations in n unknowns that we wish to solve. Let d_j denote the total degree of the jth equation (that is, the degree of the highest order monomial in the

a small nonzero value, and the next step will involve solving a slightly perturbed problem $H = \delta, x(\Delta t) = \bar{a}$. This can lead to an accumulation of error, unless residuals are carefully monitored.

ODE integrators usually adapt their step length according to relative error, whereas corrector methods for Newton's method usually use a combination of residual and relative errors. Specifically the residual error $H(x, t)$ at time t and the error, Δx , where Δx is the difference between successive values of x, are compared with a working tolerance ε. A challenge then is to reflect the additional residual control in ODE integrators.

Another view of homotopy solving is that of solving a differential equation on a manifold (that is, solving a differential-algebraic equation). Both Visconti [28] and Arponen [24] have implemented DAE solving methods in *Maple*, and it would be of interest to use these in homotopy solving.

Differentiating the ODE (3.4) yields expressions for the higher order derivatives:

H_x

which as usual will be directed to linear solvers, instead of the more expensive computational approach of inverting H_x. Now expanding to third order gives:

$$H(x + \Delta x, t) = H(x, t) + H_x \Delta x + (\Delta x)^T H_{xx}(\Delta x)/2 + O(\Delta x^3), \quad (3.8)$$

where $(\Delta x)^T$ is the transpose of the column vector Δx. If the i-th component of H is denoted by H^i, then in the equation above, H^i_{xx} is an $n \times n$ matrix with entries $H^i_{x_j x_k}$. Suppose $\Delta x = \Delta\tilde{x} + \Delta\hat{x}$. Then $\Delta\tilde{x}$ is of order 2 and $\Delta\hat{x}$ satisfies (3.8) to order 3. Substitution of this expression into (3.8), using (3.7), ignoring terms in $\Delta x \Delta\tilde{x}$, and the above, then shows that $\Delta\hat{x}$ satisfies to third order the following:

$$H_x \Delta\hat{x} = -(\Delta\tilde{x})^T H_{xx}(\Delta\tilde{x})/2. \quad (3.9)$$

From a computational point of view, notice the difference between the above expression (3.9) and (2.6). Here the number of computations has been reduced. Also notice that two linear systems must be solved: (3.7) and (3.9). The coefficient matrix is H_x in both cases, so naturally a gain in performance can be realized by computation of an LU factorization, which can then be used twice.

3.3. *Implementation of the Homotopy Algorithm*

We were guidein

Again for efficiency, the problem $H(x, t) = 0$ is solved to a lower accuracy along the path, and then when a possible solution is obtained at $t = 1$, the "end game" is entered and the solutions are obtained to a greater accuracy. Typically the maximum number of iterative improvements is increased.

Having an environment such as *Maple* opens up the possibilities to use automatic differentiation to calculate the higher order derivatives. Specifically the derivatives are encoded as programs for which good complexity estimates are known [4].

4. Application to some polynomial systems

The algorithms above were coded in *Maple* with a parameter that allowed us to turn the third-order terms on and off. In comparing performance of the codes, we can select between many different metrics. In complicated systems such as *Maple*, there is a particular difficulty of separating the efficiency of the mathematical method from the details of the programming. For this reason, we have selected to test the average size of a step and the number of iterative loops used by the corrector code.

We apply the code to the following simple problems: the intersection of 2 curves given by

$$x^2 + y^2 = 1, \quad x + 2y - 6 = 0\,; \tag{4.1}$$

a univariate problem similar to the well-known Wilkinson polynomial

(

the iterative solver was called (abbreviated as Iters below). In all cases, the third-order method used fewer steps and fewer iterations. These statistics are presented in the table below, using N for second order and H for third order.

Problem	N-Time	N-Steps	N-Iters	H-Time	H-Steps	H-Iters
Wilkinson5						

```
# The initial conditions corresponding to the known solutions
idata :=[seq(seq(seq({xr(0)=2*i-1+Re(sx),xi(0)=Im(sx),
 yr(0)=2*j-1+Re(sy),yi(0)=Im(sy), zr(0)=2*k-1+Re(sz),zi(0)=Im(sz)},
          k=0..1),j=0..1),i=0..1)]:
# Loop through the data, and obtain the solution for each
for id in idata do
   # Construct the dsolve/numeric procedure
   dsn := dsolve(dsys union id, numeric, implicit=true):
   # Obtain the solution at t=1 and print it.
   sol := dsn(1);
   print(evalf[7](
        eval([x=xr(t)+I*xi(t),y=yr(t)+I*yi(t),z=zr(t)+I*zi(t)],sol)
   ));
end do:
```

The output from running this script is

```
[x = 1.384601 + 0.5873485 I,
 y = -2.516459 - 0.1233784 I,
 z = -1.181569 + 0.7662005 I]
[x = 1.041660 + 2.196067 I,
 y = -0.1078828 - 3.715860 I,
 z = 1.630095 - 1.988129 I]
[x = 0.1122859 + 0.3201893*10^(-6) I,
 y = 0.1227375 + 0.1602351*10^(-6)I,
 z = -0.8616124 - 0.3224805*10^(-7)I]
[x = 0.4493258 + 0.8280515*10^(-6)I,
 y = 1.316899 - 0.7538583*10^(-6)I,
 z = 1.685232 - 0.1398326*10^(-6)I]
 [x = 1.384601 - 0.5873477 I,
 y = -2.516459 + 0.1233783 I,
 z = -1.181569 - 0.7661998 I]
[x = 1.041658 - 2.196065 I,
 y = -0.1078817 + 3.715856 I,
 z = 1.630094 + 1.988127 I]
[x = -2.656328 + 5.385065 I,
 y = -1.330291 + 4.515593 I,
 z = 1.681109 + 4.152297 I]
[x = -2.656328 - 5.385073 I,
 y = -1.330290 - 4.515598 I,
 z = 1.681113 - 4.152299 I]
```

The computation took around 1 second on a 1.5GHz machine. Of course a Newton improvement could be done at the end to increase accuracy. Evaluation of the original quadratic equations Eqn. (5.1) yields residuals of magnitude less than 10^{-4}. This reflects the working tolerances of the default method (10^{-7}). Tightening of these tolerances to 10^{-10} provides

Here D_x and D_y are the usual formal total derivatives so that $D_x\varphi^2 = (2u_x + 1)u_{xx} - u_x = 0$, etc. Thus we have 4 equations in the 6 unknowns $(u, u_x, u_y, u_{xx}, u_{xy}, u_{yy})$. Now regarded as a submanifold of J^2, the dimension of $V(R)$ satisfies $\dim V(R) \leq 4$ since we already have 2 obviously independent PDEs $\varphi^1 = 0$ and $\varphi^2 = 0$, and $\dim J^2 = 6$. To check if $V(R)$ has components of dimension 4 in J^2, we intersect it with a random 2 dimensional linear subspace of $\mathbb{C}^6$. This linear space is the solution set of 4 random linear equations of the form:

$$\psi^j := a_{j0} + a_{j1}u + a_{j2}u_x + a_{j3}u_y + a_{j4}u_{xx} + a_{j5}u_{xy} + a_{j6}u_{yy} = 0, \qquad (6.5)$$

where $j = 1, 2, 3, 4$ and the a_{jk} are random complex floating point numbers. The equations (6.5) together with those in (6.4) form a system of 8 equations for 6 variables for the intersection of $V(R)$ with this subspace. Following the

this problem does not occur (this is a generalization of the algebra-geometry correspondence to PDE), and is achieved in the exact case for our example by constructing representations for radicals of algebraic ideals occurring in the computation. In the approximate case the interpolation dependent methods play the same role. However constructing an interpolation-free method in the higher multiplicity case remains an open problem, which is important because of the higher complexity of the interpolation dependent methods.

7. Acknowledgements

Two of the authors (GR and KH) thank Jan Verschelde for helpful discussions. GR thanks Ilias Kotsireas, and Chris Smith for discussions.

References

1. E. L. Allgower, K. Georg. Numerical path following. In P. G. Ciarlet, J. L. Lions, eds. *Scientific Computing* (*Part 2*), 3–203. Volume 5 of *Handbook of Numerical Analysis*, North-Holland, 1997.
2. D. N. Bernstein. The number of roots of a system of equations. (Russian) *Functional Anal. Appl.* 9(3) (1975), 183–185 (English Translation, 1976).
3. R. M. Corless, A. Galligo, I. S. Kotsireas, S. M. Watt. A geometric-numeric algorithm for absolute factoriz342(19(Tw)2ul0(for)-21-30c66Tfan9(e)-1(ds.)TJF38.966Uu5orless)h9589(Tw)2p

11. I. S. Kotsireas. Homotopies and polynomial system solving I. Basic Principles. *SIGSAM Bulletin* 5(1) (2001), 19–32.
12. T. Y. Li. Numerical solution of multivariate polynomial systems by homo-