# A Globally Convergent Parallel Algorithm for Zeros of Polynomial Systems

Alexander P. Morgan
Layne T. Watson

TR 86-25

# A Globally Convergent Parallel Algorithm for Zeros of Polynomial Systems

Alexander P. Morgan* and Layne T. Watson†

**Abstract.**

Certain classes of nonlinear systems of equations, such as polynomial systems, have properties that make them particularly amenable to solution on distributed computing systems. Some algorithms, considered unfavorably on a single processor serial computer, may be excellent on a distributed system. This paper considers the solution of polynomial systems of equations via a globally convergent homotopy algorithm on a hypercube. Some computational results are reported.

## 1. Introduction.

Supercomputing capability can be achieved in several different ways: sheer hardware speed, algorithmic efficiency, pipelines and vector processors, or multiprocessor systems. Many (perhaps too many) different computer architectures have already been realized, and computational experience on these machines is accumulating. The reality is that very fast hardware is very expensive, and is likely to remain so, and access to huge vector computers like CRAY-2's is and will remain limited for some time to come. Many supercomputer systems developed at universities have both severely limited access and formidable programming problems. Despite federal initiatives and satellite links, using national supercomputer centers is awkward at best, and has the flavor of monolithic university central computer centers, only on a national scale.

The hypercube concept, that of cheap, independent processors connected in a reasonably efficient yet still manageable topology, seems to offer an opportunity for "supercomputing for the masses". A hypercube computer consists of $2^n$ processors (nodes), each with memory, floating-point hardware, and (possibly) communication hardware. The nodes are independent and asynchronous, and connected to each other like the corners of an n-dimensional cube.

For the purpose of discussion here, there are three classes of nonlinear systems of equations: (1) large systems with sparse Jacobian matrices, (2) small transcendental (nonpolynomial) systems with dense Jacobian matrices, and (3) small polynomial systems with dense Jacobian matrices. Sparsity for small problems is not significant, and large systems with dense Jacobian matrices are intractable, so these two classes are not counted. Of course medium sized problems are also of practical interest, and the boundaries between small, medium, and large change with computer hardware technology and algorithmic development. Depending on algorithmic efficiency, hardware capability, and the significance of sparsity, a medium sized problem is treated like it belongs to one of the above three classes anyway, so there is no need for a "medium" class.

Large sparse nonlinear systems of equations, such as equilibrium equations in structural mechanics, have two aspects: highly nonlinear and recursive scalar computations, and large matrix, vector operations. There is a great amount of parallelism in both aspects, but the nature of the

* Mathematics Department, General Motors Research Laboratories, Warren, MI 48090-9055.

† Departments of Electrical Engineering and Computer Science, Industrial and Operations Engineering, and Mathematics, The University of Michigan, Ann Arbor, MI 48109. Current address: Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061. The work of this author was supported in part by AFOSR Grant 85-0250.

parallelism is very different (or so it seems). Small dense transcendental systems of equations pose a major challenge, since they involve recursive, scalar intensive computation with a small amount of linear algebra. It has been argued that the communication overhead of hypercube machines makes them unsuited for such problems, but the issue is still open and algorithmic breakthroughs are yet possible. Polynomial systems are unique in that they have many solutions, of which several may be physically meaningful, and that there exist algorithms guaranteed to find all these meaningful solutions. The very special nature of polynomial systems is not fully appreciated, especially by those who are unfamiliar with probability-one homotopy methods.

Algorithms for solving nonlinear systems of equations can be broadly classified as (1) locally convergent or (2) globally convergent. The former includes Newton's method, various quasi-Newton methods, and inexact Newton methods. The latter includes continuation, simplicial methods, and probability-one homotopy methods. These algorithms are qualitatively significantly different, and their performance on parallel systems may very well be the reverse of their performance on serial processors. The overall purpose of this research is to study how nonlinear systems of equations might be solved on a hypercube.

Much work has been done on solving linear systems of equations on parallel computers, mostly on vector machines [4–7, 9–11, 13–15, 17, 20, 21]. Some work has been done on nonlinear equations and Newton's method [27, 24], and on finding the roots of a single polynomial equation [8, 23]. Parallel algorithms for polynomial systems have not been studied, nor have parallel homotopy algorithms for nonlinear systems of equations.

The present article considers only polynomial systems and homotopy algorithms. Large sparse nonlinear systems, small transcendental systems, and quasi-Newton algorithms will be considered in future work. Section 2 presents the mathematics behind the homotopy algorithm, and sketches a computer implementation based on ODE techniques. Section 3 briefly describes the "hypercube" computer architecture. Section 4 discusses the special case of polynomial systems in some detail. Computational results on an Intel iPSC–32 and several other machines are contained in Section 5.

## 2. Homotopy algorithm.

Let $E^p$ denote $p$-dimensional real Euclidean space, and let $F : E^p \to E^p$ be a $C^2$ (twice continuously differentiable) function. The general problem is to solve the nonlinear system of equations

$$F(x) = 0.$$

The fundamental mathematical result behind the homotopy algorithm is

**Proposition 1.** Let $F : E^p \to E^p$ be a $C^2$ map and $\rho : E^m \times [0,1) \times E^p \to E^p$ a $C^2$ map such that

1) the Jacobian matrix $D\rho$ has full rank on $\rho^{-1}(0)$;

and for fixed $a \in E^m$

2) $\rho(a,0,x) = 0$ has a unique solution $W \in E^p$;

3) $\rho(a,1,x) = F(x)$;

4) the set of zeros of $\rho_a(\lambda,x) = \rho(a,\lambda,x)$ is bounded.

Then for almost all $a \in E^m$ there is a zero curve $\gamma$ of

$$\rho_a(\lambda,x) = \rho(a,\lambda,x),$$

along which the Jacobian matrix $D\rho_a(\lambda,x)$ has full rank, emanating from $(0,W)$ and reaching a zero $\bar{x}$ of $F$ at $\lambda = 1$. Furthermore, $\gamma$ has finite arc length if $DF(\bar{x})$ is nonsingular.

The general idea of the algorithm is apparent from the proposition: just follow the zero curve $\gamma$ of $\rho_a$ emanating from $(0, W)$ until a zero $\bar{x}$ of $F(x)$ is reached (at $\lambda = 1$). Of course it is nontrivial to develop a viable numerical algorithm based on that idea, but at least conceptually, the algorithm for solving the nonlinear system of equations $F(x) = 0$ is clear and simple. A typical form for the homotopy map is

$$\rho_W(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - W), \tag{1}$$

which has the same form as a standard continuation or embedding mapping. However, there are two crucial differences. In standard continuation, the embedding parameter $\lambda$ increases monotonically from 0 to 1 as the trivial problem $x - W = 0$ is continuously deformed to the problem $F(x) = 0$. The present homotopy method permits $\lambda$ to both increase and decrease along $\gamma$ with no adverse effect; that is, turning points present no special difficulty. The second important difference is that there are never any "singular points" which afflict standard continuation methods. The way in which the zero curve $\gamma$ of $\rho_a$ is followed and the full rank of $D\rho_a$ along $\gamma$ guarantee this. Observe that Proposition 1 guarantees that $\gamma$ cannot just "stop" at an interior point of $[0, 1) \times E^p$.

The zero curve $\gamma$ of the homotopy map $\rho_a(\lambda, x)$ (of which $\rho_W(\lambda, x)$ in (1) is a special case) can be tracked by many different techniques; refer to the excellent survey [1] and recent work [30], [31]. The numerical results here were obtained with preliminary versions of HOMPACK [30], a software package currently under development at Sandia National Laboratories, General Motors Research Laboratories, Virginia Polytechnic Institute and State University, and The University of Michigan. There are three primary algorithmic approaches to tracking $\gamma$ : 1) an ODE-based algorithm, 2) a predictor-corrector algorithm whose corrector follows the flow normal to the Davidenko flow (a "normal flow" algorithm); 3) a version of Rheinboldt's linear predictor, quasi-Newton corrector algorithm [22] (an "augmented Jacobian matrix" method).

Only the ODE-based algorithm will be discussed here. Alternatives 2) and 3) are described in detail in [31] and [2], respectively. Assuming that $F(x)$ is $C^2$ and $a$ is such that Proposition 1 holds, the zero curve $\gamma$ is $C^1$ and can be parametrized by arc length $s$. Thus $\lambda = \lambda(s)$, $x = x(s)$ along $\gamma$, and

$$\rho_a(\lambda(s), x(s)) = 0 \tag{2}$$

identically in $s$. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s)) \begin{pmatrix} \dfrac{d\lambda}{ds} \\ \dfrac{dx}{ds} \end{pmatrix} = 0, \tag{3}$$

$$\left\| \left( \frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{4}$$

With the initial conditions

$$\lambda(0) = 0, \quad x(0) = W, \tag{5}$$

the zero curve $\gamma$ is the trajectory of the initial value problem (3–5). When $\lambda(\bar{s}) = 1$, the corresponding $x(\bar{s})$ is a zero of $F(x)$. Thus all the sophisticated ODE techniques currently available can be brought to bear on the problem of tracking $\gamma$ [26], [29].

Typical ODE software requires $(d\lambda/ds, dx/ds)$ explicitly, and (3), (4) only implicitly define the derivative $(d\lambda/ds, dx/ds)$. (It might be possible to use an implicit ODE technique for (3–4), but that seems less efficient than the method proposed here.) The derivative $(d\lambda/ds, dx/ds)$, which is a unit tangent vector to the zero curve $\gamma$, can be calculated by finding the one-dimensional kernel of the $p \times (p+1)$ Jacobian matrix

$$D\rho_a(\lambda(s), x(s)),$$

which has full rank by Proposition 1. It is here that a substantial amount of computation is incurred, and it is imperative that the number of derivative evaluations be kept small. Once the kernel has been calculated, the derivative $(d\lambda/ds, dx/ds)$ is uniquely determined by (4) and continuity. Complete details for solving the initial value problem (3–5) and obtaining $x(\bar{s})$ are in [28] and [29]. A discussion of the kernel computation follows.

The Jacobian matrix $D\rho_a$ is $p \times (p+1)$ with (theoretical) rank $p$. The crucial observation is that the last $p$ columns of $D\rho_a$, corresponding to $D_x\rho_a$, may not have rank $p$, and even if they do, some other $p$ columns may be better conditioned. The objective is to avoid choosing $p$ "distinguished" columns, rather to treat all columns the same (not possible for sparse matrices). There are kernel finding algorithms based on Gaussian elimination and $p$ distinguished columns [16]. Choosing and switching these $p$ columns is tricky, and based on *ad hoc* parameters. Also, computational experience has shown that accurate tangent vectors $(d\lambda/ds, dx/ds)$ are essential, and the accuracy of Gaussian elimination may not be good enough. A conceptually elegant, as well as accurate, algorithm is to compute the QR factorization with column interchanges [3] of $D\rho_a$,

$$Q\, D\rho_a\, P^t P z = \begin{pmatrix} * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ 0 & & * & * \end{pmatrix} Pz = 0,$$

where $Q$ is a product of Householder reflections and $P$ is a permutation matrix, and then obtain a vector $z \in \ker D\rho_a$ by back substitution. Setting $(Pz)_{p+1} = 1$ is a convenient choice. This scheme provides high accuracy, numerical stability, and a uniform treatment of all $p+1$ columns. Finally,

$$\left(\frac{d\lambda}{ds}, \frac{dx}{ds}\right) = \pm \frac{z}{\|z\|_2},$$

where the sign is chosen to maintain an acute angle with the previous tangent vector on $\gamma$ . There is a rigorous mathematical criterion, based on a $(p+1) \times (p+1)$ determinant, for choosing the sign, but there is no reason to believe that would be more robust than the angle criterion.

Several features which are a combination of common sense and computational experience should be incorporated into the algorithm. Since most ordinary differential equation solvers only control the local error, the longer the arc length of the zero curve $\gamma$ gets, the farther away the computed points may be from the true curve $\gamma$. Therefore when the arc length gets too long, the last computed point $(\bar{\lambda}, \bar{x})$ is used to calculate a new parameter vector $\bar{a}$ such that

$$\rho_{\bar{a}}(\bar{\lambda}, \bar{x}) = 0 \tag{6}$$

exactly, and the zero curve of $\rho_{\bar{a}}(\lambda, x)$ is followed starting from $(\bar{\lambda}, \bar{x})$. A rigorous justification for this strategy was given in [29]. If $\rho_a$ has the special form in (1), then trivially

$$\bar{a} = \left(\bar{\lambda}\, F(\bar{x}) + (1-\bar{\lambda})\,\bar{x}\right)/(1-\bar{\lambda}).$$

For more general homotopy maps $\rho_a$, this computation of $\bar{a}$ may be complicated.

Remember that tracking $\gamma$ was merely a means to an end, namely a zero $\tilde{x}$ of $F(x)$. Since $\gamma$ itself is of no interest (usually), one should not waste computational effort following it too closely. However, since $\gamma$ is the only sure way to $\tilde{x}$, losing $\gamma$ can be disastrous. The tradeoff between computational efficiency and reliability is very delicate, and a fool-proof strategy appears difficult to achieve. None of the three primary algorithms alone is superior overall, and each of the three beats the other two (sometimes by an order of magnitude) on particular problems. Since the algorithms' philosophies are significantly different, a hybrid will be hard to develop.

In summary, the algorithm is:

```
1. Set $s := 0$, $y := (0, W)$, $ypold := yp := (1, 0, \ldots, 0)$, $restart :=$ false, $error :=$ initial error
   tolerance for the ODE solver.
2. If $y_1 < 0$ then go to 23.
3. If $s >$ some constant then
   4. $s := 0$.
   5. Compute a new vector $a$ satisfying (6). If
          $\|\text{new } a - \text{old } a\| > 1 + \text{constant} * \|\text{old } a\|$,
      then go to 23.
6. $ode\ error := error$.
7. If $\|yp - ypold\|_\infty >$ (last arc length step) $*$ constant, then $ode\ error := tolerance \ll error$.
8. $ypold := yp$.
9. Take a step along the trajectory of (3–5) with the ODE solver. $yp = y'(s)$ is computed for the
   ODE solver by 10–12:
   10. Find a vector $z$ in the kernel of $D\rho_a(y)$ using Householder reflections.
   11. If $z^t\, ypold < 0$, then $z := -z$.
   12. $yp := z/\|z\|$.
13. If the ODE solver returns an error code, then go to 23.
14. If $y_1 < 0.99$, then go to 2.
15. If $restart =$ true, then go to 20.
16. $restart :=$ true.
17. $error :=$ final accuracy desired.
18. If $y_1 \geq 1$, then set $(s, y)$ back to the previous point (where $y_1 < 1$).
19. Go to 4.
20. If $y_1 < 1$ then go to 2.
21. Obtain the zero (at $y_1 = 1$) by interpolating mesh points used by the ODE solver.
22. Normal return.
23. Error return.
```

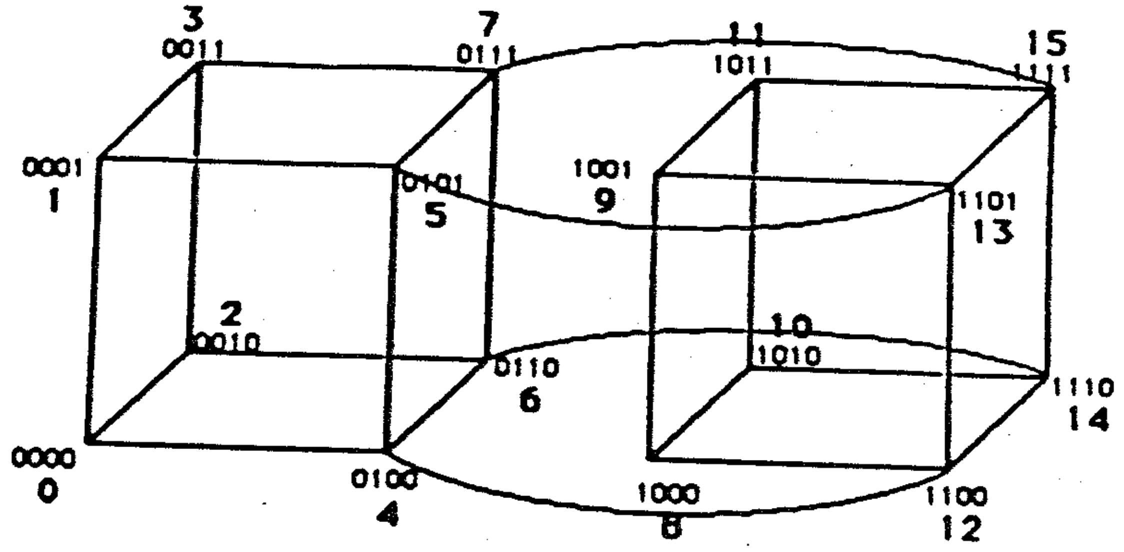

Figure 1. 4-cube structure and node labelling.

## 3. The hypercube.

The word "hypercube" refers to an $n$-dimensional cube. Think of a cube in $n$ dimensions as sitting in the positive orthant, with vertices at the points

$$(v_1, \ldots, v_n), \quad v_i \in \{0, 1\}, \quad i = 1, \ldots, n.$$

There are thus $2^n$ vertices, and two vertices $v$ and $w$ are "adjacent", i.e., connected by an edge, if and only if $v_i = w_i$ for all $i$ except one. The associated graph, also sometimes referred to as an "$n$-cube", has $2^n$ vertices (which can be labelled as above with binary $n$-tuples) and edges between vertices whose labels differ in exactly one coordinate (see Figure 1).

A "hypercube computer architecture" is a computer system with $2^n$ (node) processors, corresponding to the $2^n$ vertices (nodes), and a communication link corresponding to each edge of the $n$-cube. Thus each processor has a direct communication link to exactly $n$ other processors. The distance between any two of the $P = 2^n$ processors is at most $n = \log_2 P = \log_2(2^n)$, considered an ideal compromise between total connectivity (distance = 1) and ring connectivity (distance $\leq P/2$). Figure 1 shows how a 4-cube is built up from two 3-cubes.

Typically the node label $(v_1, \ldots, v_n)$ is viewed as a binary number $v_1 v_2 \ldots v_n$, and in this view two nodes are adjacent if and only if their binary representations differ in exactly one bit. Typically node addresses are computed in programs by a gray code, a bijective function

$$g : \{0, \ldots, 2^n - 1\} \to \{0, \ldots, 2^n - 1\}$$

such that the binary representations of $g(k \pmod{2^n})$ and $g(k+1 \pmod{2^n})$ differ in exactly one bit for all $k$ (*cf.* [12]).

Realizations of this abstract architecture have one additional feature: a "host" processor with communication links to *all* the node processors. This host typically loads programs into the nodes, starts and stops processes executing in the nodes, and interchanges data with the nodes. In current hardware implementations only the host has external I/O and peripheral storage; the nodes consists of memory, a CPU, and possibly communication and floating-point hardware.

The Intel iPSC has 32, 64, or 128 nodes. Each node is an 80286/80287 with 512K bytes of memory. The host is also an 80286/80287, but with 4 MB of memory, a floppy disk drive, a hard disk, an Ethernet connection, and Xenix. The nodes have only a minimal monitor for communication and process management.

The NCUBE has up to 1024 nodes in multiples of 64, each with 128K of memory and communication and floating-point hardware. The host is an 80286, running NCUBE's operating system, a primitive version of UNIX. The node chip is NCUBE's own design, with a unique feature being communication hardware.

## 4. Polynomial systems.

Suppose that the components of the nonlinear function $F(x)$ have the form

$$F_i(x) = \sum_{k=1}^{n_i} a_{ik} \prod_{j=1}^{n} x_j^{d_{ijk}}, \quad i = 1, \ldots, n. \tag{7}$$

The $i$th component $F_i(x)$ has $n_i$ terms, the $a_{ik}$ are the (real) coefficients, and the degrees $d_{ijk}$ are nonnegative integers. The total degree of $F_i$ is

$$d_i = \max_k \sum_{j=1}^{n} d_{ijk}.$$

For technical reasons it is necessary to consider $F(x)$ as a map $F : C^n \to C^n$, where $C^n$ is $n$-dimensional complex Euclidean space. A system of $n$ polynomial equations in $n$ unknowns, $F(x) = 0$, may have many solutions. It is possible to define a homotopy so that all geometrically isolated solutions of (7) have at least one associated homotopy path. Generally, (7) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as $\lambda$ approaches 1. However, (7) can be transformed into a new system which, under reasonable hypotheses, can be proven to have no solutions at infinity and thus bounded homotopy paths. Because scaling can be critical to the success of the method, a general scaling algorithm is applied to scale the coefficients and variables in (7) before anything else is done.

Since the homotopy map defined below is complex analytic, the homotopy parameter $\lambda$ is monotonically increasing as a function of arc length [19]. The existence of an infinite number of solutions or an infinite number of solutions at infinity does not destabilize the method. Some paths will converge to the higher dimensional solution components, and these paths will behave the way paths converging to any singular solution behave. Practical applications usually seek a subset of the solutions, rather than all solutions [18, 19]. However, the sort of generic homotopy algorithm considered here must find all solutions and cannot be limited without, in essense, changing it into a heuristic.

Define $G : C^n \to C^n$ by

$$G_j(x) = b_j {x_j}^{d_j} - a_j, \qquad j = 1, \ldots, n, \tag{8}$$

where $a_j$ and $b_j$ are nonzero complex numbers and $d_j$ is the (total) degree of $F_j(x)$, for $j = 1, \ldots, n$. Define the homotopy map

$$\rho_c(\lambda, x) = (1 - \lambda)\, G(x) + \lambda\, F(x), \tag{9}$$

where $c = (a, b)$, $a = (a_1, \ldots, a_n) \in C^n$ and $b = (b_1, \ldots, b_n) \in C^n$. Let $d = d_1 \cdots d_n$ be the *total degree* of the system.

**Theorem.** For almost all choices of $a$ and $b$ in $C^n$, $\rho_c^{-1}(0)$ consists of $d$ smooth paths emanating from $\{0\} \times C^n$, which either diverge to infinity as $\lambda$ approaches 1 or converge to solutions to $F(x) = 0$ as $\lambda$ approaches 1. Each geometrically isolated solution of $F(x) = 0$ has a path converging to it.

A number of distinct homotopies have been proposed for solving polynomial systems. The homotopy map in (9) is from [19]. As with all such homotopies, there will be paths diverging to infinity if $F(x) = 0$ has solutions at infinity. These divergent paths are (at least) a nuisance, since they require arbitrary stopping criteria. Solutions at infinity can be avoided via the following projective transformation.

Define $F'(y)$ to be the homogenization of $F(x)$:

$$F'_j(y) = {y_{n+1}}^{d_j}\, F_j(y_1/y_{n+1}, \ldots, y_n/y_{n+1}), \qquad j = 1, \ldots, n. \tag{10}$$

Note that, if $F'(y^0) = 0$, then $F'(\alpha y^0) = 0$ for any complex scalar $\alpha$. Therefore, "solutions" of $F'(y) = 0$ are (complex) lines through the origin in $C^{n+1}$. The set of all lines through the origin in $C^{n+1}$ is called complex projective $n$-space, denoted $CP^n$, and is a smooth compact (complex) $n$-dimensional manifold. The solutions of $F'(y) = 0$ in $CP^n$ are identified with the solutions and solutions at infinity of $F(x) = 0$ as follows. If $L \in CP^n$ is a solution to $F'(y) = 0$ with $y = (y_1, y_2, \ldots, y_{n+1}) \in L$ and $y_{n+1} \neq 0$, then $x = (y_1/y_{n+1}, y_2/y_{n+1}, \ldots, y_n/y_{n+1}) \in C^n$ is a solution to $F(x) = 0$. On the other hand, if $x \in C^n$ is a solution to $F(x) = 0$, then the line through $y = (x, 1)$ is a solution to $F'(y) = 0$ with $y_{n+1} = 1 \neq 0$. The most mathematically satisfying definition of *solutions to $F(x) = 0$ at infinity* is simply *solutions to $F'(y) = 0$ (in $CP^n$) generated by $y$ with* $y_{n+1} = 0$.

A basic result on the structure of the solution set of a polynomial system is the following classical theorem of Bezout:

**Theorem.** There are no more than $d$ isolated solutions to $F'(y) = 0$ in $CP^n$. If $F'(y) = 0$ has only a finite number of solutions in $CP^n$, it has exactly $d$ solutions, counting multiplicities.

Recall that a solution is *isolated* if there is a neighborhood containing that solution and no other solution. The multiplicity of an isolated solution is defined to be the number of solutions that appear in the isolating neighborhood under an arbitrarily small random perturbation of the system coefficients. If the solution is nonsingular (i.e., the system Jacobian matrix is nonsingular at the solution), then it has multiplicity one. Otherwise it has multiplicity greater than one.

Define a linear function

$$u(y_1, \ldots, y_{n+1}) = \xi_1 y_1 + \xi_2 y_2 + \cdots + \xi_{n+1} y_{n+1}$$

where $\xi_1, \ldots, \xi_{n+1}$ are nonzero complex numbers, and define $F'' : C^{n+1} \to C^{n+1}$ by

$$\begin{aligned} F''_j(y) &= F'_j(y), \qquad j = 1, \ldots, n, \\ F''_{n+1}(y) &= u(y) - 1. \end{aligned} \tag{11}$$

So $F''(y) = 0$ is a system of $n+1$ equations in $n+1$ unknowns, referred to as *the projective transformation of* $F(x) = 0$. Since $u(y)$ is linear, it is easy in practice to replace $F''(y) = 0$ by an equivalent system of $n$ equations in $n$ unknowns. The significance of $F''(y)$ is given by

**Theorem.** If $F'(y) = 0$ has only a finite number of solutions in $CP^n$, then $F''(y) = 0$ has exactly $d$ solutions (counting multiplicities) in $C^{n+1}$ and no solutions at infinity, for almost all $\xi \in C^{n+1}$.

Under the hypothesis of the theorem, all the solutions of $F'(y) = 0$ can be obtained as lines through the solutions to $F''(y) = 0$. Thus all the solutions to $F(x) = 0$ can be obtained easily from the solutions to $F''(y) = 0$, which lie on bounded homotopy paths (since $F''(y) = 0$ has no solutions at infinity).

The projective transformation functions essentially as a scaling transformation. Its effect is to shorten arc lengths and bring solutions closer to the unit sphere. The coefficient and variable scaling is different, in that it directly addresses extreme values in the system coefficients. The two scaling schemes work well together.

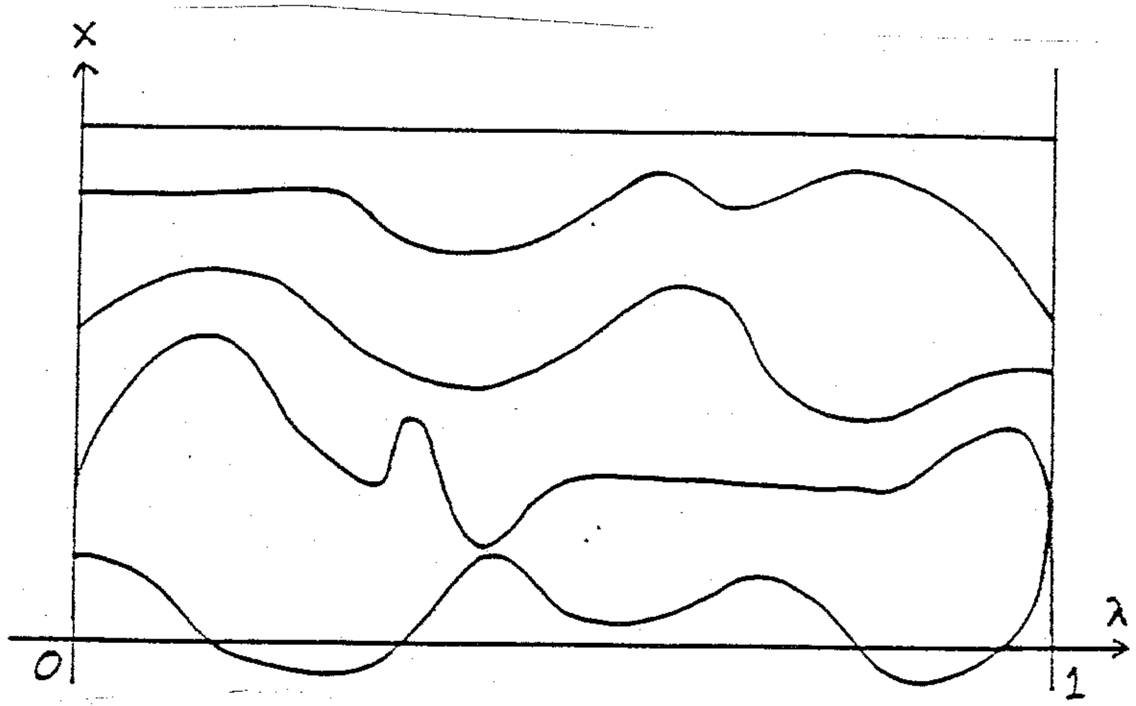

Figure 2. The set $\rho_a^{-1}(0)$ for a polynomial system.

Figure 2 shows the nature of the zero curves for a polynomial system. There are $d$ (the total degree of $F$) of them, they are monotone in $\lambda$, and have finite arc length.

## 5. Computational results.

Polynomial systems arise in such diverse areas as solid modelling, robotics, chemical engineering, mechanical engineering, and computer vision. A small problem has total degree $d < 100$ and a large problem has $d > 1000$. Given that $d$ homotopy paths are to be tracked, there are two extreme approaches using a hypercube.

The first extreme, with the coarsest granularity possible, is to assign one path to each node processor, with the host controlling the assignment of paths to the nodes, keeping as many nodes busy as possible, and post-processing the answers computed by the nodes.

The second extreme, with the finest granularity, is to track all $d$ paths on the host processor, distributing the numerical linear algebra, polynomial system evaluation, Jacobian matrix evaluation, and possibly other tasks amongst the nodes. Because of the high communication overhead (whether hardware or software) on a hypercube, this approach requires an immense amount of sophistication and analysis to prevent the communication costs from overwhelming the computational costs. Also the algorithm at this granularity would have to be a major modification of the serial algorithm. A possible advantage is that the load could be balanced better, resulting in an overall speedup over a coarser grained algorithm.

These issues are not simple, and much more research is needed on these and intermediate approaches. Neither extreme can be declared inferior *a priori*. The first approach, having a node track an entire path, has been tried, and some results are shown in Table 1. The problem number refers to an internal numbering scheme used at General Motors Research Laboratories; problem data is available on request. These problems are all real engineering problems that have arisen at GM and elsewhere.

Table 1. Execution time (secs).

| Problem number | total degree | 80286/ 80287 | iPSC–32 | VAX 11/750 | VAX 11/780 | IBM 3090 | SUN-2 /50 | SUN-3 /160 |
|---|---|---|---|---|---|---|---|---|
| 102 | 256 | 16257 | 645 | 2438 | 1248 | 77 | 10545 | 8041 |
| 103 | 625 | 34692 | 1616 | 5260 | 2656 | 163 | 22634 | 17126 |
| 402 | 4 | 255 | 54 | 41 | 18 | 2 | 158 | 111 |
| 403 | 4 | 84 | 19 | 14 | 6 | 1 | 54 | 38 |
| 405 | 64 | 3669 | 335 | 703 | 334 | 14 | 2958 | 2429 |
| 601 | 60 | 9450 | 257 | 1707 | 796 | 78 | 7417 | 5903 |
| 602 | 60 | 28783 | 2795 | 4332 | 2054 | 124 | 21897 | 16831 |
| 603 | 12 | 1200 | 243 | 325 | 152 | 9 | 1339 | 1060 |
| 803 | 256 | — | 11527 | 29779 | 16221 | 667 | 130311 | 77113 |
| 1702 | 16 | 1655 | 163 | 216 | 112 | 7 | 986 | 658 |
| 1703 | 16 | 1657 | 162 | 216 | 112 | 7 | 984 | 658 |
| 1704 | 16 | 1628 | 108 | 216 | 112 | 6 | 1005 | 667 |
| 1705 | 81 | 14336 | 378 | 1884 | 999 | 55 | 8907 | 6313 |
| 5001 | 576 | — | 11786 | 49736 | 27815 | 1997 | — | 237685 |

It is worthwhile to comment here on some realities of parallel computation. The parallel computation scheme of assigning a path to each node seems completely transparent and straightforward. Why this is not so is illustrative of the difficulty of the whole field of parallel computation. The following is a list of tasks that should be trivial, with comments as to why they were not:

*1. Assigning paths to nodes.*

There are exactly $d$ paths and starting points, completely determined by the parameters in the homotopy map $\rho_c$. Once $\rho_c$ is chosen, these starting points are fixed, and may not be changed in any way. In production use of HOMPACK, not every solution may be desired, so only some of the paths may need to be tracked. Thus the parallel computer code has to keep track of all the paths, which of those are being tracked and have been assigned to a node, and which are to be tracked but have not yet been assigned to a node. This is further complicated by the fact that the number of paths does not equal (in general) the number of nodes. Also the nodes are finishing paths at random (thus becoming available to track another path), and so must be reused in random order. Now all this bookkeeping can, of course, be done, but the point is that these considerations do not even arise for the serial program.

*2. Transmitting data to the nodes.*

In the serial version of HOMPACK, all work arrays (below the level of the main program) have variable dimensions, and there are absolutely no static arrays limiting the size of the problem. Current hardware implementations of the hypercube architecture have nodes with relatively small memories (128K or 512K), and the nodes do *not* support virtual memory. Furthermore, what executes on the nodes is a "main program", not a "subroutine". Thus memory is tight, and arrays cannot be dynamically allocated. Since the actual storage needed depends on several factors–dimension, total degree, number of terms, etc.–it is difficult to optimally fit a problem into the available memory. Another annoyance is that only short messages may be sent (on the NCUBE the node program itself must be transmitted by the host program), so the data defining the problem must be broken up and transmitted in pieces. All this communication sending, receiving, queueing, blocking, and unblocking is currently (and unfortunately) the responsibility of the application programs.

*3. Transmitting answers to the host.*

To appreciate the situation here, consider the following three facts: 1) the complete answer (point $(1, \bar{x})$ at end of path and concomitant path statistics) cannont be sent all in one message; 2) the nodes finish tracking their assigned paths asynchronously; 3) for hardware and software reasons, the host cannot just "listen" to one node. No matter how the host resolves this situation, it is going to be nontrivial.

One possibility is to maintain a huge data structure, and as the answer pieces arrive, insert them in the correct slot in the structure. This requires an identification stamp with each message, and some blocking and unblocking overhead at the ends. A second possibility is to establish a "handshake" protocol, whereby each node simply informs the host that its path is complete, and does not transmit its solution until the host requests it. Even in this case an answer reception may be punctuated by completion notices from other nodes, which must be queued. This second approach was the one used for the times reported in Table 1. There are many other reasonable approaches to this issue; more research is needed.

*4. Replicating performance measurements.*

Message transmission is a combination of hardware and software, and may involve several (intermediate) nodes between the sender and recipient. All this happens asynchronously in real time, and the temporal order of events may depend on such things as buffer status, free space list size, timer interrupts, and even random (corrected) hardware errors. The state of the node operating systems and disk file fragmentation on the host can affect durations and the temporal order of events, sometimes by as much as 10%. Obtaining performance data is difficult, and for complex, realistic codes, replicating performance results may be impossible.

These problems were also attempted on the NCUBE, but as of this writing the NCUBE f77 compiler would not correctly compile HOMPACK, which is written in ANSI standard FORTRAN 77.

## 6. References.

[1] E. Allgower and K. Georg, *Simplicial and continuation methods for approximating fixed points*, SIAM Rev., 22 (1980), pp. 28–85.

[2] S. C. Billups, *An augmented Jacobian matrix algorithm for tracking homotopy zero curves*, M.S. Thesis, Dept. of Computer Sci., VPI & SU, Blacksburg, VA, Sept., 1985.

[3] P. Businger and G. H. Golub, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.

[4] A. C. Chen and C. L. Wu, *Optimum solution to dense linear systems of equations*, Proc. 1984 Internat. Conf. on Parallel Processing, August 21–24, 1984, pp. 417–425.

[5] M. Y. Chern and T. Murata, *Fast algorithm for concurrent LU decomposition and matrix inversion*, Proc. Internat. Conf. on Parallel Processing, Computer Society Press, Los Alamitos, CA, 1983, pp. 79–86.

[6] E. Cloëte and G. R. Joubert, *Direct methods for solving systems of linear equations on a parallel processor*, Proc. 8th South African Symp. on Numerical Mathematics, Durban, South Africa, July 19–21, 1982.

[7] M. Cosnard, Y. Robert, and D. Trystran, *Comparison of parallel diagonalization methods for solving dense linear systems*, Sessions of the French Acad. of Sci. on Math., Nov., 1985, p. 781ff.

[8] G. H. Ellis and L. T. Watson, *A parallel algorithm for simple roots of polynomials*, Comput. Math. Appl., 10 (1984), pp. 107–121.

[9] D. D. Gajski, A. H. Sameh, and J. A. Wisniewski, *Iterative algorithms for tridiagonal matrices on a WSI-multiprocessor*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 82–89, 1982.

[10] W. Gentzsch and G. Schafer, *Solution of large linear systems on vector computers*, Parallel Computing 83, North Holland, Amsterdam, 1984, pp. 159–166.

[11] D. Heller, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev., 20 (1978), pp. 740–777.

[12] Intel Corporation, *iPSC Users' Manual*, 1985.

[13] Y. Kaneda and M. Kohata, *Highly parallel computing of linear equations on the matrix-broadcast-memory connected array processor system*, 10th IMACS World Congress, Vols. 1–5, pp. 320–322, 1982.

[14] J. S. Kowalik, *Parallel computation of linear recurrences and tridiagonal equations*, Proc. IEEE 1982 Internat. Conf. on Cybernetics and Society, 1982, pp. 580–584.

[15] J. S. Kowalik and S. P. Kumar, *An efficient parallel block conjugate gradient method for linear equations*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 47–52, 1982.

[16] M. Kubicek, *Dependence of solutions of nonlinear systems on a parameter*, ACM Trans. Math. Software, 2 (1976), pp. 98–107.

[17] S. Lakshmivarahan and S. K. Dhall, *Parallel algorithms for solving certain classes of linear recurrences*, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 206, Springer-Verlag, Berlin, 1985, pp. 457–477.

[18] A. P. Morgan, *A transformation to avoid solutions at infinity for polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 77–86.

[19] ——, *A homotopy for solving polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 87–92.

[20] D. Parkinson, *The solution of N linear equations using P processors*, Parallel Computing 83, North Holland, Amsterdam, 1984, pp. 81–87.

[21] D. A. Reed and M. L. Patrick, *A model of asynchronous iterative algorithms for solving large sparse linear systems*, Proc. 1984 Internat. Conf. on Parallel Processing, August 21–24, 1984, pp. 402–410.

[22] W. C. Rheinboldt and J. V. Burkardt, *Algorithm 596: A program for a locally parameterized continuation process*, ACM Trans. Math. Software, 9 (1983), pp. 236-241.

[23] T. A. Rice and L. J. Siegel, *A parallel algorithm for finding the roots of a polynomial*, Proc. Internat. Conf. Parallel Processing, Bellaire, MI, Aug. 24–27, pp. 57–61, 1982.

[24] H. Schwandt, *Newton-like interval methods for large nonlinear systems of equations on vector computers*, Computer Phys. Comm., 37 (1985), pp. 223–232.

[25] C. L. Seitz, *The cosmic cube*, Commun. ACM, 28 (1985), pp. 22–33.

[26] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

[27] H. J. Sips, *A parallel processor for nonlinear recurrence systems*, Proc. 1st Internat. Conf. on Supercomputing Systems, IEEE Computer Society Press, Los Alamitos, CA, 1984, pp. 660–671.

[28] L. T. Watson and D. Fenner, *Chow-Yorke algorithm for fixed points or zeros of $C^2$ maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.

[29] L. T. Watson, *A globally convergent algorithm for computing fixed points of $C^2$ maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.

[30] L. T. Watson, S. C. Billups, and A. P. Morgan, *HOMPACK: A suite of codes for globally convergent homotopy algorithms*, Tech. Rep. 85–34, Dept. of Industrial and Operations Eng., Univ. of Michigan, Ann Arbor, MI, 1985.

[31] L. T. Watson, *Numerical linear algebra aspects of globally convergent homotopy methods*, Tech. Report TR–85–14, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1985.