ALGORITHM 652
HOMPACK: A Suite of Codes for Globally Convergent Homotopy Algorithms

LAYNE T. WATSON
Virginia Polytechnic Institute and State University
STEPHEN C. BILLUPS
Sandia National Laboratories
and
ALEXANDER P. MORGAN
General Motors Research Laboratories

There are algorithms for finding zeros or fixed points of nonlinear systems of equations that are globally convergent for almost all starting points, i.e., with probability one. The essence of all such algorithms is the construction of an appropriate homotopy map and then tracking some smooth curve in the zero set of this homotopy map. HOMPACK provides three qualitatively different algorithms for tracking the homotopy zero curve: ordinary differential equation-based, normal flow, and augmented Jacobian matrix. Separate routines are also provided for dense and sparse Jacobian matrices. A high-level driver is included for the special case of polynomial systems.

Categories and Subject Descriptors: G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*systems of equations*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: Chow–Yorke algorithm, continuation method, curve tracking, fixed point, globally convergent, homotopy methods, nonlinear systems, polynomial systems, zero

1. INTRODUCTION

Homotopies are a traditional part of topology, and only recently have begun to be used for practical numerical computation. The algorithms described here are known as probability one globally convergent homotopy algorithms, which are related to, but distinct from, continuation, parameter continuation, incremental

L. T. Watson's work was supported in part by NSF grant MCS 8207217, Control Data Corporation grant 84V101, and AFOSR grant 85-0250. S. C. Billup's work was supported in part by NSF grant MCS 8207217 and Department of Energy contract DE-AC04-76DP00789.

Authors' current addresses: L. T. Watson, Departments of Electrical Engineering and Computer Science, Industrial and Operations Engineering, and Mathematics, University of Michigan, Ann Arbor, MI 48109; S. C. Billups, Safety Assessment Technologies Division 7233, Sandia National Laboratories, Albuquerque, NM 87185; A. P. Morgan, Mathematics Department, General Motors Research Laboratories, Warren, MI 48090-9055.

loading, displacement incrementation, invariant imbedding, and continuous Newton methods. These algorithms are also referred to as "continuous" methods, to distinguish them from the simplicial homotopy methods, whose theoretical foundations date back to the very origins of topology.

The homotopies we will consider are sometimes called "artificial-parameter generic homotopies," in contrast to natural-parameter homotopies. The distinction has to do with requiring that the homotopy zero curves obey strict smoothness conditions. These conditions generally will not hold if the homotopy parameter represents a physically meaningful quantity, but they can always be obtained via certain generic constructions using an artificial (i.e., nonphysical) homotopy parameter. The goal of using these artificial-parameter homotopies is to solve fixed-point and zero-finding problems with homotopies that avoid bifurcations and other singular and ill-conditioned behavior. This point is considered further in this section in the discussion of the simple illustrative example below. The frameworks for fixed-point and zero-finding problems are slightly different, so they will be discussed separately.

The fixed-point problem will be considered first. Let B be the closed unit ball in n-dimensional real Euclidean space E^n, and let $f: B \rightarrow B$ be a C^2 map. Define $\rho_a: [0, 1) \times B \rightarrow E^n$ by

$$\rho_a(\lambda, x) = \lambda(x - f(x)) + (1 - \lambda)(x - a). \tag{1}$$

The fundamental result [5] is that for almost all a (in the sense of Lebesgue measure) in the interior of B, there is a zero curve $\gamma \subset [0, 1] \times B$ of ρ_a, along which the Jacobian matrix $D\rho_a(\lambda, x)$ has rank n, emanating from $(0, a)$ and reaching a point $(1, \bar{x})$, where $\bar{x}$ is a fixed point of f. Thus with probability one, picking a starting point $a \in \text{int } B$ and following γ leads to a fixed point $\bar{x}$ of f. This justifies the phrase "globally convergent with probability one."

The zero-finding problem

$$F(x) = 0, \tag{2}$$

where $F: E^n \rightarrow E^n$ is a C^2 map, is more complicated. Suppose there exists a C^2 map

$$\rho: E^m \times [0, 1) \times E^n \rightarrow E^n$$

such that

(a) the $n \times (m + 1 + n)$ Jacobian matrix $D\rho(a, \lambda, x)$ has rank n on the set

$$\rho^{-1}(0) = \{(a, \lambda, x) \mid a \in E^m, 0 \leq \lambda < 1, x \in E^n, \rho(a, \lambda, x) = 0\},$$

and for any fixed $a \in E^m$,

(b) $\rho_a(0, x) = \rho(a, 0, x) = 0$ has a unique solution x_0,

(c) $\rho_a(1, x) = F(x)$,

(d) $\rho_a^{-1}(0)$ is bounded.

Then the supporting theory [5, 28, 31] says that for almost all $a \in E^m$ there exists a zero curve γ of ρ_a along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0, x_0)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. γ does not intersect

34. WATSON, L. T. Engineering applications of the Chow–Yorke algorithm. *Appl. Math. Comput. 9* (1981), 111–133.
35. WATSON, L. T. Numerical linear algebra aspects of globally convergent homotopy methods. Tech. Rep. TR-85-14, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 1985.
36. WATSON, L. T., AND FENNER, D. Chow–Yorke algorithm for fixed points or zeros of C^2 maps. *ACM Trans. Math. Softw. 6* (1980), 252–260.
37. WATSON, L. T., KAMAT, M. P., AND REASER, M. H. A robust hybrid algorithm for computing multiple equilibrium solutions. *Eng. Comput. 2* (1985), 30–34.
38. WATSON, L. T., AND SCOTT, M. R. Solving spline collocation approximations to nonlinear two-point boundary value problems by a homotopy method. Tech. Rep. TR-84-15, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 1984.
39. WRIGHT, A. H. Finding all solutions to a system of polynomial equations. *Math. Comput. 44* (1985), 125–133.

Received December 1982; revised April 1987; accepted May 1987

itself and is disjoint from any other zeros of ρ_a. The globally convergent algorithm is to pick $a \in E^m$ (which uniquely determines x_0), and then track the homotopy zero curve γ. A simple choice for ρ_a is

$$\rho_a(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - a). \tag{3}$$

This satisfies properties (a)–(c), but not necessarily (d). There are fairly general sufficient conditions on $F(x)$ so that (3) will satisfy property (d), but for some practical problems of interest the homotopy map (3) will not suffice. This is why HOMPACK is designed to handle arbitrary homotopy maps $\rho_a(\lambda, x)$ satisfying properties (a)–(d).

HOMPACK is designed for "artificial-parameter generic homotopies," also known as "probability-one globally convergent homotopies." This is in contrast to "natural-parameter homotopies." How this limits HOMPACK and what the advantages and disadvantages of this limitation are will be discussed next. Included is a simple example that illustrates the probability-one feature.

Our goal is to solve a "parameter-free" system of equations, $F(x) = 0$. This is in contrast to the problem of solving a natural-parameter homotopy, $F(\lambda, x) = 0$, where λ has a physical significance. In the case that the homotopy is given by nature, the resulting homotopy curves must be dealt with as they are, bifurcations, ill-conditioning, and all. Therefore, curve tracking becomes the main focus of the problem-solving effort. For a parameter-free problem, however, one can design the homotopy to have paths with nice properties. Thus extra attention is devoted to constructing the homotopy, and the curve-tracking algorithm can be limited to a more well-behaved class of curves. The decision to limit HOMPACK to artificial-parameter generic homotopies means that it is not suited to some natural-parameter problems (e.g., those that generate curves that bifurcate).

The Transversality Theorem from differential topology (see [20], Appendix 4) provides general conditions under which most homotopies (in a precise sense) will have smooth, nonbifurcating curves. In practice an admissible homotopy is constructed by defining artificial parameters (a) so that a partial derivative condition is satisfied and then choosing these parameters independently of the structure and coefficients of the original system (F). For example, random choices of a will generally work.

Thus the artificial homotopy $\rho(a, \lambda, x)$ might be chosen so that the jth component includes a_j and not any a_k for $k \neq j$, and so that the partial derivatives

$$\frac{\partial \rho(a, \lambda, x)}{\partial a_j}$$

for $j = 1$ to n are nonzero for $0 \leq \lambda < 1$, for all x. Here $a = (a_1, \ldots, a_n)$ and λ are *artificial*; that is, they have nothing to do with x or any other parameter of the given problem. Then a is chosen at random. The Transversality Theorem guarantees that the resulting homotopy curves will be smooth, without bifurcations or singularities. In fact, in practice they tend to be very well conditioned. This mysterious usefulness of randomly chosen a is a feature of the "probability-one" approach to constructing homotopies.

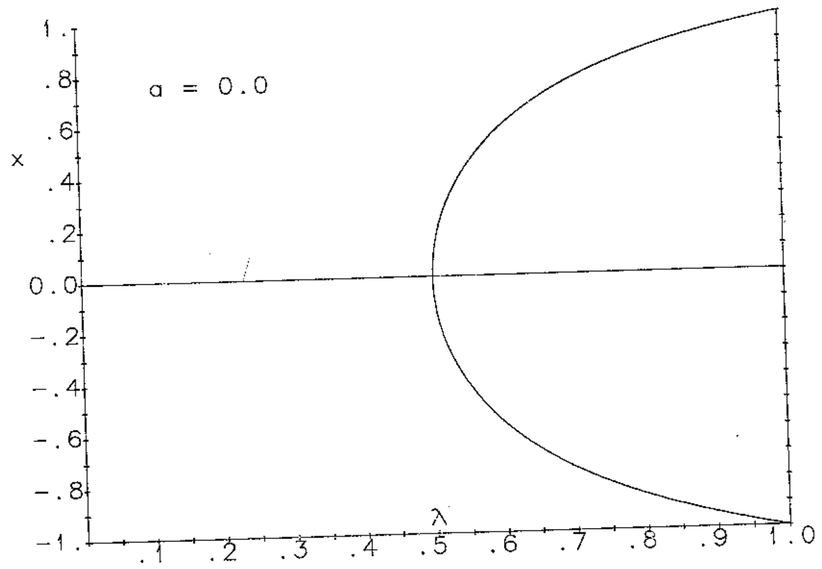


Fig. 1. $a = 0$.

Here is an example to illustrate the "probability-one" concept. Consider the single equation in one unknown

$$F(x) = x^3 - x,$$

with the homotopy

$$\rho(a, \lambda, x) = \lambda(x^3 - x) + (1 - \lambda)(x - a),$$

which is (3) in this special case. If $a = 0$, this homotopy generates a bifurcation when $\lambda = \frac{1}{2}$ (see Figure 1). However, for any other value of the artificial parameter a, no bifurcation is encountered. See Figures 2 and 3 for the cases $a = -0.01$ and $a = 0.01$. The theory of the method says that for a "randomly chosen" a, the homotopy path cannot bifurcate. In fact, in this case, bifurcation occurs for only one value of a.

Now suppose the homotopy

$$r(t, x) = t(x^3 - x) + (1 - t)x$$

were a "natural homotopy" where t goes from 0 to 1. This is ρ above, of course, with $a = 0$ and $\lambda = t$, where now t is a physical parameter and we therefore feel obligated to track the path beginning at $(t, x) = (0, 0)$. Then we would have no choice but to encounter the singularity at $t = \frac{1}{2}$. But $r(t, x) = 0$ can be solved for any fixed value of t using an artificial homotopy like the ρ above; say,

$$\tilde{\rho}(a, \lambda, x) = \lambda[t(x^3 - x) + (1 - t)x] + (1 - \lambda)(x - a),$$

which has no bifurcations and is globally convergent for almost all values of a. In many engineering applications, solving $r(t, x) = 0$ for a few important values of t is sufficient.

6. Chow, S. N., Mallet-Paret, J. and Yorke, J. A. A homotopy method for locating all zeros of a system of polynomials. In *Functional Differential Equations and Approximation of Fixed Points.* Lecture Notes in Mathematics Vol. 730. H. O. Peitgen and H. O. Walther, Eds. Springer-Verlag, New York, 1979, 228–237.
7. Craig, E. J. Iteration procedures for simultaneous equations. Ph.D. dissertation, MIT, Cambridge, 1954.
8. Dennis, J. E., and Schnabel, R. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1983.
9. Fadeev, D. K. and Fadeeva, V. N. *Computational Methods of Linear Algebra.* Freeman, London, 1963.
10. Garcia, C. B., and Zangwill, W. I. Finding all solutions to polynomial systems and other systems of equations. *Math. Program. 16* (1979), 159–176.
11. Gill, P. E., and Murray, W. Newton type methods for unconstrained and linearly constrained optimization. *Math. Program. 7* (1974), 311–350.
12. Haselgrove, C. B. A solution of nonlinear equations and of differential equations with two-point boundary conditions. *Comput. J. 4* (1961), 255–259.
13. Hestenes, M. R. The conjugate gradient method for solving linear systems. In *Proceedings of the Symposium on Applied Mathematics, 6, Numer. Anal.*, AMS, New York, 1956, 83–102.
14. Kubicek, M. Dependence of solutions of nonlinear systems on a parameter. *ACM Trans. Math. Softw. 2* (1976), 98–107.
15. Matthies, H., and Strang, G. The solution of nonlinear finite element equations. *Int. J. Numer. Math. Eng., 14* (1979), 1613–1626.
16. Meintjes, K., and Morgan, A. P. A methodology for solving chemical equilibrium systems. *Appl. Math. Comput. 22* (1987), 333–361.
17. Mejia, R. CONKUB: A conversational path-follower for systems of nonlinear equations, *J. Comput. Phys. 63* (1986), 67–84.
18. Menzel, R., and Schwetlick, H. Zur Lösung parameterabhängiger nichtlinearer Gleichungen mit singulären Jacobi–Matrizen. *Numer. Math. 30* (1978), 65–79.
19. Morgan, A. P. A transformation to avoid solutions at infinity for polynomial systems. *Appl. Math. Comput. 18* (1986), 77–86.
20. Morgan, A. P. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, Englewood Cliffs, N.J., 1987.
21. Morgan, A. P., and Sommese, A. A homotopy for solving general polynomial systems that respects *m*-homogeneous structures. *Appl. Math. Comput. 23* (1987).
22. Rheinboldt, W. C. Numerical analysis of continuation methods for nonlinear structural problems. *Comput. Struct. 13* (1981), 103–113.
23. Rheinboldt, W. C. *Numerical Analysis of Parameterized Nonlinear Equations.* Wiley-Interscience, New York, 1986.
24. Rheinboldt, W. C., and Burkardt, J. V. Algorithm 596: A program for a locally parameterized continuation process. *ACM Trans. Math. Softw. 9* (1983), 236–241.
25. Shampine, L. F., and Gordon, M. K. *Computer Solution of Ordinary Differential Equations: The Initial Value Problem.* W. H. Freeman, San Francisco, 1975.
26. Tsai, L.-W., and Morgan, A. P. Solving the kinematics of the most general six- and five-degree-of-freedom manipulators by continuation methods. *ASME J. Mech. Transmissions, Aut. Des. 107* (1985), 48–57.
27. van der Waerden, B. L. *Modern Algebra, Vols. 1 and 2.* Frederick Ungar, New York, 1953.
28. Watson, L. T. A globally convergent algorithm for computing fixed points of C^2 maps. *Appl. Math. Comput. 5* (1979), 297–311.
29. Watson, L. T. Fixed points of C^2 maps. *J. Comput. Appl. Math. 5* (1979), 131–140.
30. Watson, L. T. Solving the nonlinear complementarity problem by a homotopy method. *SIAM J. Contr. Optim. 17* (1979), 36–46.
31. Watson, L. T. An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems. *SIAM J. Numer. Anal. 16* (1979), 394–401.
32. Watson, L. T. Computational experience with the Chow–Yorke algorithm. *Math. Program. 19* (1980), 92–101.
33. Watson, L. T. Solving finite difference approximations to nonlinear two-point boundary value problems by a homotopy method. *SIAM J. Sci. Stat. Comput. 1* (1980), 467–480.

Table III. Number of Jacobian Matrix Evaluations for a Polynomial System

Projective transformation	Scaling	Path 1	Path 2	Path 3	Path 4	Total
Yes	Yes	53	37	35	46	171
No	Yes	41	37	39	1937	2054
Yes	No	141	110	132	134	517
No	No	102	101	100	21045	21348

The exact solutions (to four significant figures) are

$$\begin{aligned}(x_1, x_2) = {} & (.09089, -\ .09115),\\ & (2342, -\ .7883),\\ & (.01615 + 1.685i,\ .0002680 + .004428i),\\ & (.01615 - 1.685i,\ .0002680 - .004428i).\end{aligned}$$

Table III shows the number of Jacobian matrix evaluations for POLSYS with various options applied to this problem. The local curve tracking tolerance was 10^{-4} and the end game tolerance was 10^{-14}.

Caveats

—HOMPACK cannot always save the user from a poorly chosen homotopy map. The code was not designed for homotopies whose zero curves bifurcate, are unbounded, or oscillate rapidly.

—Singular solutions where rank $D\rho_a(1, \bar{x}) < n$ may cause HOMPACK to fail to converge, after wasting time iterating near the solution. This has been observed for polynomial systems that have multiple solutions.

—The D algorithms are the most robust, but also the most expensive. It is difficult to predict which algorithm will be best on a given problem.

—Careful use of the SSPAR parameter array (see the comments in FIXPNF or FIXPQF) will improve the efficiency and robustness of the N and Q routines.

—The POLSYS driver is simple to use, but computationally inefficient. The user should customize the subroutine FFUNP when maximum efficiency is required (see, e.g., Chapter 10 of [20]).

—HOMPACK is *not* a general curve tracking package. Using it to track zero curves of natural-parameter imbeddings should be done with extreme caution.

REFERENCES

1. BILLUPS, S. C. An augmented Jacobian matrix algorithm for tracking homotopy zero curves. Master's thesis, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., Sept. 1985.
2. BRUNOVSKÝ, P., AND MERAVÝ, P. Solving systems of polynomial equations by bounded and real homotopy. *Numer. Math. 43* (1984), 397–418.
3. BUSINGER, P., AND GOLUB, G. H. Linear least squares solutions by Householder transformations. *Numer. Math. 7* (1965), 269–276.
4. CHAN, T. F., AND SAAD, Y. Iterative methods for solving bordered systems with applications to continuation methods. *SIAM J. Sci. Stat. Comput. 6* (1985), 438–451.
5. CHOW, S. N., MALLET-PARET, J., AND YORKE, J. A. Finding zeros of maps: Homotopy methods that are constructive with probability one. *Math. Comput. 32* (1978), 887–899.

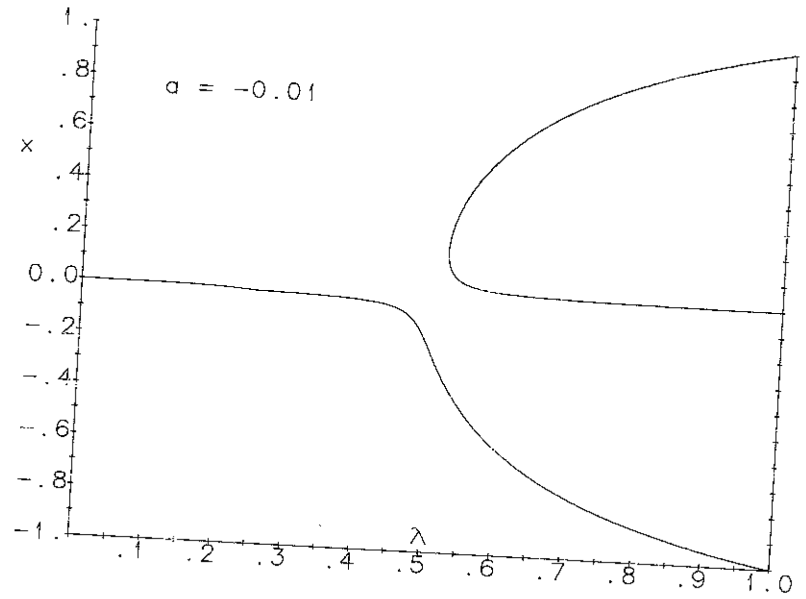


Fig. 2. $a = -0.01$.

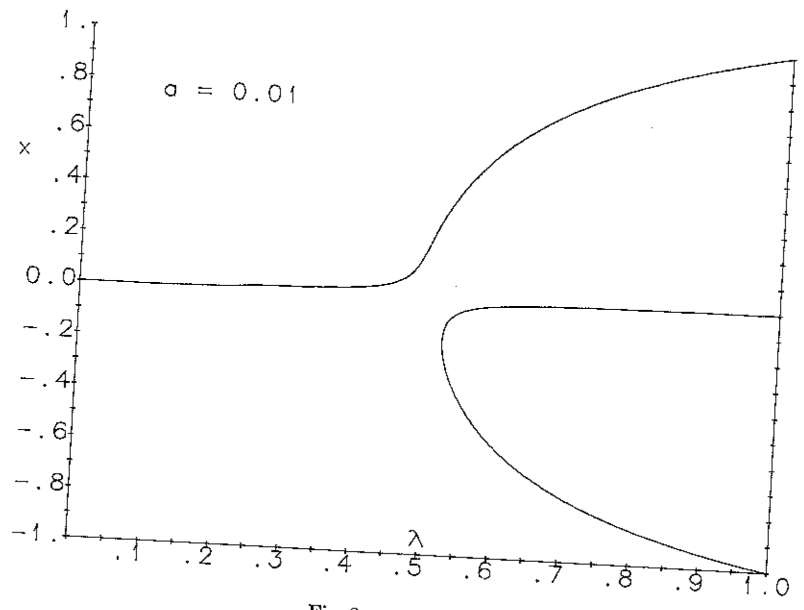


Fig. 3. $a = 0.01$.

The point is that one can often get around having to track paths with singularities, even given a natural homotopy. For a parameter-free problem, there is no reason to use anything but an artificial-parameter generic homotopy.

Probability-one globally convergent homotopy curves have no bifurcations (with probability one) for $0 \leq \lambda < 1$. However, at the end of a curve (when

$\lambda = 1$), singularities may be encountered. This happens precisely when the original problem is singular at the solution, because as $\lambda \to 1$ the homotopy becomes the original system. For some mild singularities in F, the homotopy can remain nonsingular at $\lambda = 1$, but in general this is not so.

To summarize, the purpose of probability-one globally convergent homotopy methods is to create homotopies whose zero curves are well behaved with well-conditioned Jacobian matrices and that reach a solution for almost all choices of a parameter (global convergence). Physical phenomena that are the source of numerical difficulties (such as bifurcation) occur as a function of natural parameters. Unless it is precisely these phenomena that are of interest, artificial imbeddings are preferable to natural imbeddings.

A very special and common situation is when each component of the nonlinear function $F(x)$ is a polynomial in n variables. There is an elegant algorithm for this case that finds all solutions of (2), real and complex, as well as solutions at infinity. HOMPACK provides an intelligent, easy to use, high-level driver for polynomial systems.

There are many different algorithms for tracking the zero curve γ; HOMPACK supports three such algorithms: ordinary differential equation-based, normal flow, and augmented Jacobian matrix. The ODE-based algorithm was developed by Watson [36] in 1976, and is the basis for some of the subroutines in HOMPACK. Sparse and dense Jacobian matrices require substantially different algorithms, and the development of sparse homotopy algorithms ([4], [37]) was a crucial advance. The following sections describe the three curve-tracking algorithms, their sparse-matrix versions, and the special polynomial system homotopy map.

2. ORDINARY DIFFERENTIAL EQUATION-BASED ALGORITHM (DENSE JACOBIAN MATRIX)

Depending on the problem, the homotopy map $\rho_a(\lambda, x)$ may be given by (1), (3), or something else that is even nonlinear in λ. The details for these three cases are similar, so for the sake of brevity, only the zero-finding problem (2) with homotopy map (3) will be presented. Assuming that $F(x)$ is C^2, a is such that the Jacobian matrix $D\rho_a(\lambda, x)$ has full rank along γ, and γ is bounded, the zero curve γ is C^1 and can be parameterized by arc length s. Thus $\lambda = \lambda(s)$, $x = x(s)$ along γ, and

$$\rho_a(\lambda(s), x(s)) = 0 \tag{4}$$

identically in s. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s))\begin{pmatrix} d\lambda/ds \\ dx/ds \end{pmatrix} = 0, \tag{5}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{6}$$

With the initial conditions

$$\lambda(0) = 0, \qquad x(0) = a, \tag{7}$$

Table II. Numerical Results

	FIXPDF		FIXPNF		FIXPQF		Arc
n	NFE	Time	NFE	Time	NFE	Time	length
Brown's function							
5	87(−3)	.71	17(−2)	.19	13(−2)	.59	2.7
10	85(−2)	2.12	24(−2)	.61	10(−2)	1.54	3.7
15	102(−2)	5.64	23(−2)	1.39	15(−2)	4.46	4.4
20	98(−4)	10.44	22(−2)	2.44	11(−2)	5.56	5.1
25	123(−3)	22.83	29(−2)	5.39	15(−2)	11.27	5.7
30	96(−3)	27.99	23(−2)	6.62	14(−2)	15.46	6.2
35	110(−4)	48.54	28(−2)	11.72	16(−2)	25.41	6.6
40	110(−4)	68.54	26(−2)	15.85	11(−4)	30.99	7.1
45	128(−4)	105.32	30(−3)	24.73	17(−2)	48.01	7.5
50	113(−4)	125.48	29(−2)	32.99	12(−2)	45.18	7.8
Exponential Function							
2	70(−4)	.27	12(−2)	.07	5(−2)	.14	1.6
3	270(−5)	1.42	39(−2)	.31	41(−2)	1.18	5.1
4	280(−4)	2.03	75(−2)	.87	55(−3)	2.96	6.5
5	486(−4)	4.73	213(−6)	3.38	94(−3)	7.06	14.5
6	817(−5)	10.20	293(−8)	6.16	106(−3)	9.92	16.9
7	1517(−6)	24.98	433(−8)	11.73	157(−3)	18.49	24.0
8	2931(−7)	60.50	577(−8)	20.73	228(−4)	36.65	47.6
9	4511(−8)	109.82	824(−8)	37.44	278(−4)	54.11	61.8
10	5671(−8)	165.32	1001(−9)	53.80	356(−4)	79.45	85.8

tracking tolerance as a power of 10. This tracking tolerance represents the largest tolerance that succeeded in tracking the zero curve.

Several important general conclusions about the codes can be drawn from Table II. For Brown's function, FIXPNF is more efficient in terms of time than FIXPQF, but it is equally clear from this same data that had the Jacobian matrix evaluations been very expensive, FIXPQF would have been the better code. In general FIXPNF will be faster if Jacobian matrix evaluations are cheap, and FIXPQF will be faster (as intended) if the Jacobian matrix computation is expensive. The exponential function for $n = 5$ shows that FIXPDF is more robust than FIXPNF, since FIXPDF succeeded with a tracking tolerance of 10^{-4} and FIXPNF failed at tolerances of 10^{-4} and 10^{-5}. Thus counting the time lost in failures (not shown in Table II), FIXPDF was more efficient than FIXPNF for this problem. In general, FIXPDF is the most reliable of the three codes, but it is also the most expensive, sometimes by a wide margin. Similar remarks apply to the sparse codes FIXP?S.

POLSYS was applied to the polynomial system

$$F_j(x) = a_{j1}x_1^2 + a_{j2}x_2^2 + a_{j3}x_1x_2 + a_{j4}x_1 + a_{j5}x_2 + a_{j6} = 0, \quad \text{for } j = 1, 2,$$

where

$$\begin{array}{llll} a_{11} = -.00098 & a_{14} = -235 & a_{21} = -.01 & a_{24} = .00987 \\ a_{12} = 978000 & a_{15} = 88900 & a_{22} = -.984 & a_{25} = -.124 \\ a_{13} = -9.8 & a_{16} = -1.0 & a_{23} = -29.7 & a_{26} = -.25 \end{array}$$

Table I. Taxonomy of Homotopy Subroutines

$x = f(x)$		$F(x) = 0$		$\rho(a, \lambda, x) = 0$		
Dense	Sparse	Dense	Sparse	Dense	Sparse	Algorithm
FIXPDF	FIXPDS	FIXPDF	FIXPDS	FIXPDF	FIXPDS	Ordinary differential equation
FIXPNF	FIXPNS	FIXPNF	FIXPNS	FIXPNF	FIXPNS	Normal flow
FIXPQF	FIXPQS	FIXPQF	FIXPQS	FIXPQF	FIXPQS	Augmented Jacobian matrix

RHOJAC (subroutines normally provided by the user). These special versions are included in HOMPACK, so for a polynomial system the user need only call POLSYS, and define the problem directly to POLSYS by specifying the polynomial coefficients. POLSYS scales and computes partial derivatives on its own. Thus the user interface to POLSYS and HOMPACK is clean and simple. The only caveat is that FFUNP cannot recognize patterns of repeated expressions in the polynomial system; and so may be less efficient than a handcrafted version. If great efficiency is required, the user can modify the default FFUNP; the sections in the code that must be changed are clearly marked. The grouping is:

[POLSYS] [POLYNF, POLYP, ROOTNF, STEPNF, TANGNF] [DIVP, FFUNP,
GFUNP, HFUNP, HFUN1P, INITP, MULP, OTPUTP, POWP, RHO, RHOJAC,
ROOT, SCLGNP, STRPTP]

10. TESTING

Since work was begun on HOMPACK in 1976, it has been tested on hundreds of problems. Two early versions of HOMPACK ([29] and [36]) were applied to a series of difficult engineering problems, some of which were surveyed in [34]. Numerical results on the application of homotopy methods to optimization were reported in [30] and [32], and to nonlinear two-point boundary value problems in [31], [33], and [38]. The sparse matrix components of HOMPACK have been tested on large structural mechanics problems [37].

Table II shows some results for Brown's function, which has an ill-conditioned Jacobian matrix, and an exponential function, whose zero curve γ has several sharp turns. Brown's function is

$$f_1(x) = \prod_{i=1}^{n} x_i - 1$$

$$f_k(x) = x_k + \sum_{i=1}^{n} x_i - (n+1), \quad k = 2, \ldots, n.$$

The exponential function is

$$f_k(x) = x_k - \exp\left(\cos\left(k \sum_{i=1}^{n} x_i\right)\right), \quad k = 1, \ldots, n.$$

The starting point was $a = 0$. The solutions were found with a relative error of 10^{-10}, and the CPU times are for a VAX 11/785. NFE is the number of Jacobian matrix evaluations. The number in parentheses represents the magnitude of the

the zero curve γ is the trajectory of the initial value problem (5–7). When $\lambda(\bar{s}) = 1$, the corresponding $x(\bar{s})$ is a zero of $F(x)$. Thus all the sophisticated ordinary differential equation techniques currently available can be brought to bear on the problem of tracking γ [25], [28].

Typical ordinary differential equation software requires $(d\lambda/ds, dx/ds)$ explicitly, and (5) and (6) only implicitly define the derivative $(d\lambda/ds, dx/ds)$. (It might be possible to use an implicit ordinary differential equation technique for (5–6), 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 γ, can be calculated by finding the one-dimensional kernel of the $n \times (n+1)$ Jacobian matrix

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

which has full rank according to the theory [28]. 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 (6) and continuity. Complete details for solving the initial value problem (5–7) and obtaining $x(\bar{s})$ are in [28] and [36]. A discussion of the kernel computation follows.

The Jacobian matrix $D\rho_a$ is $n \times (n+1)$ with rank n according to the supporting theory. The crucial observation is that the last n columns of $D\rho_a$, corresponding to $D_x\rho_a$, may not have rank n, and even if they do, some other n columns may be better conditioned. The objective is to avoid choosing n "distinguished" columns, rather to treat all columns the same (not possible for sparse matrices). Kubicek [14] and Mejia [17] have kernel-finding algorithms based on Gaussian elimination and n distinguished columns. Choosing and switching these n 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 $D\rho_a$,

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

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 $(Pz)_{n+1} = 1$ is a convenient choice. This scheme provides high accuracy, numerical stability, and a uniform treatment of all $n+1$ 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 γ. There is a rigorous mathematical criterion, based on a $(n+1) \times (n+1)$ determinant, for choosing the sign, but there is no reason to believe that would be more robust than the angle criterion.

Several features that 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 γ gets, the farther away the computed points may be from the true curve γ. Therefore when the arc length gets too long, the last computed point $(\bar{\lambda}, \bar{x})$ is used to calculate

$$\bar{a} = \frac{\bar{\lambda}F(\bar{x}) + (1 - \bar{\lambda})\bar{x}}{1 - \bar{\lambda}}. \tag{8}$$

Then $\rho_{\bar{a}}(\bar{\lambda}, \bar{x}) = 0$ 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 [28].

Remember that tracking γ was merely a means to an end, namely a zero $\tilde{x}$ of $F(x)$. Since γ itself is of no interest (usually), one should not waste computational effort following it too closely. However, since γ is the only sure way to $\tilde{x}$, losing γ can be disastrous [28]. HOMPACK estimates the curvature of each component of γ using finite differences, and reduces the tolerance used by the ordinary differential equation solver whenever the estimated curvature exceeds some threshold. The tradeoff between computational efficiency and reliability is very delicate, and a foolproof strategy appears difficult to achieve. This is the reason HOMPACK provides several algorithms; no single algorithm is superior overall, and each of the three beats the other two (sometimes by an order of magnitude) on particular problems.

In summary, the algorithm is:

```
1. Set $s := 0$, $y := (0, a)$, $ypold := yp := (1, 0, \ldots, 0)$, $restart :=$ 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 $a$ from (8). 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 (5–7) 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.
```

steps will be taken before a path is abandoned. The use of both the projective transformation and scaling is recommended for most problems.

9. ORGANIZATIONAL DETAILS

HOMPACK is organized in two different ways: by algorithm/problem type and by subroutine level. There are three levels of subroutines. The top level consists of drivers, one for each problem type and algorithm type. Normally these drivers are called by the user, and the user need know nothing beyond them. They allocate storage for the lower level routines, and all the arrays are variable dimension, so there is no limit on problem size. The second subroutine level implements the major components of the algorithms, such as stepping along the homotopy zero curve, computing tangents, and the end game for the solution at $\lambda = 1$. A sophisticated user might call these routines directly to have complete control of the algorithm, or for some other task such as tracking an arbitrary parameterized curve over an arbitrary parameter range. The lowest subroutine level handles the numerical linear algebra, and includes some BLAS routines. All the linear algebra and associated data structure handling are concentrated in these routines, so a user could incorporate his or her own data structures by writing his or her own versions of these low-level routines. Also, by concentrating the linear algebra in subroutines, HOMPACK can be easily adapted to a vector or parallel computer.

The organization of HOMPACK by algorithm/problem type is shown in Table I, which lists the driver name for each algorithm and problem type. The naming convention is

$$FIXP\begin{Bmatrix} D \\ N \\ Q \end{Bmatrix}\begin{Bmatrix} F \\ S \end{Bmatrix},$$

where $D \approx$ ordinary differential equation algorithm, $N \approx$ normal flow algorithm, $Q \approx$ augmented Jacobian matrix algorithm, $F \approx$ dense Jacobian matrix, and $S \approx$ sparse Jacobian matrix. Using brackets to indicate the three subroutine levels described above, the natural grouping of the HOMPACK routines is

[FIXPDF] [FODE, ROOT, SINTRP, STEPS] [DCPOSE]
[FIXPDS] [FODEDS, ROOT, SINTRP, STEPDS] [GMFADS, MFACDS, MULTDS, PCGDS, QIMUDS, SOLVDS]
[FIXPNF] [ROOTNF, STEPNF, [TANGNF]] [ROOT]
[FIXPNS] [(ROOTNS, STEPNS, TANGNS] [GMFADS, MFACDS, MULTDS, PCGDS, PCGNS, QIMUDS, ROOT, SOLVDS]
[FIXPQF] [ROOTQF, STEPQF, TANGQF] [QRFAQF, QRSLQF, R1UPQF, UPQRQF]
[FIXPQS] [ROOTQS, STEPQS, TANGQS] [GMFADS, MULTDS, PCGQS, SOLVDS]

The BLAS subroutines used by HOMPACK are DAXPY, DCOPY, DDOT, DNRM2, DSCAL, D1MACH, IDAMAX.

The user-written subroutines, of which exactly two must be supplied depending on the driver chosen, are F, FJAC, FJACS, RHO, RHOA, RHOJAC, RHOJS.

The special-purpose polynomial system solver POLSYS is essentially a high-level driver for HOMPACK. POLSYS requires special versions of RHO and

Thus, letting $L_{ij} = \log_{10}(|p_{ij}|)$,

$$
\begin{aligned}
S_i(z) &= \sum_{j=1}^{n_i} \operatorname{sgn}(p_{ij}) 10^{e_i} 10^{L_{ij}} \prod_{k=1}^{n} (10^{v_k} z_k)^{d_{ijk}} \\
&= \sum_{j=1}^{n_i} \operatorname{sgn}(p_{ij}) 10^{[e_i + L_{ij} + \sum_{k=1}^{n} v_k d_{ijk}]} \prod_{k=1}^{n} z_k^{d_{ijk}}.
\end{aligned}
\tag{57}
$$

Now choosing e_i and v_k to minimize the sum of squares of the exponents of the coefficients of S corresponds to the unconstrained minimum of

$$E(e, v) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n_i} \left[e_i + L_{ij} + \sum_{k=1}^{n} v_k d_{ijk} \right]^2. \tag{58}$$

$\nabla E = 0$ is the linear equation

$$A\ w = b \tag{59}$$

with $w = (e_1, \ldots, e_n, v_1, \ldots, v_n)^t$ and

$$
\begin{aligned}
A_{rs} &= \delta_{rs} n_r, & A_{r,n+s} &= \sum_{j=1}^{n_r} d_{rjs}, \\
A_{n+r,s} &= \sum_{j=1}^{n_s} d_{sjr}, & A_{n+r,n+s} &= \sum_{i=1}^{n} \sum_{j=1}^{n_i} d_{ijr} d_{ijs}, \\
b_r &= -\sum_{j=1}^{n_r} L_{rj}, & b_{n+r} &= -\sum_{i=1}^{n} \sum_{j=1}^{n_i} L_{ij} d_{ijr},
\end{aligned}
$$

where r and s take on all integer values from 1 to n. Observe that A does not depend on the values of the coefficients of $F(x)$ (the p's), but rather only on the exponents (the d's). If any coefficient p_{ij} is zero, it must be omitted from the above calculations. In practice whether p_{ij} is zero will be decided by a threshold test, and changing this threshold can significantly affect the scaling of the system.

The parameters n, n_i, p_{ij}, d_{ijk} defined above constitute the *coefficient tableau.* The subroutine SCLGNP uses this tableau to generate the scale factors e_i and v_k, and the subroutine FFUNP uses it to compute system and Jacobian matrix values for POLSYS. This has the advantage of being very general, but the disadvantage of usually being less efficient than a specialized FFUNP. If CPU time is an issue, the user may modify FFUNP to reduce the amount of repeated computation inherent in its generic form.

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 SCLGNP scaling is different, in that it directly addresses extreme values in the system coefficients. The two scaling schemes work well together.

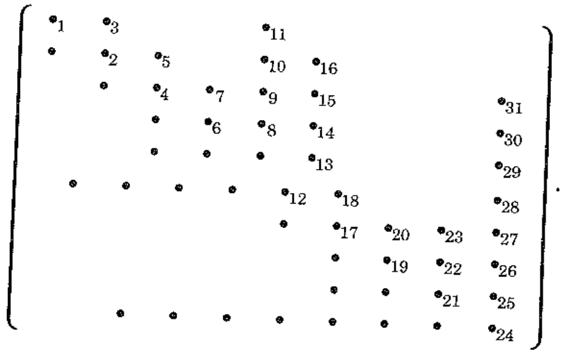
POLSYS is written so that the user can choose to evoke the projective transformation or not and evoke scaling or not. If either of these options is selected, it is transparent to the user; the solutions are returned untransformed and unscaled. The input to POLSYS is the coefficient tableau and a few parameters for the path-tracking algorithm (normal flow) used by POLSYS. POLSYS has a parameter NUMRR (number of reruns) so that 1000*NUMRR

3. ORDINARY DIFFERENTIAL EQUATION-BASED ALGORITHM (SPARSE JACOBIAN MATRIX)

Large nonlinear systems of equations with sparse symmetric Jacobian matrices occur in many engineering disciplines, and each class of problems has special characteristics. Nonlinear structural mechanics problems (see references 19–23, 26, 29, 36, 37, 42, 76, 78, 80 in [35]) will be considered here, because they are representative of many problems outside structural mechanics, and yet have enough special features to admit efficient solution. As in the previous section, the fixed-point and general cases are similar to the zero-finding case, so only the latter is described here. For technical reasons it is preferable to write the homotopy map in (3) as

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

with the order of the arguments reversed (this is an internal matter to HOMPACK and causes no confusion at the user interface, since the user only specifies $F(x)$). The matrix $D_x\rho_a(x, \lambda) = \lambda DF(x) + (1 - \lambda)I$ is symmetric (because it is the Hessian matrix of some energy potential function) and sparse with a "skyline" structure, such as



Typically such matrices are stored in packed skyline format, in which the upper triangle is stored in a one-dimensional array indexed as shown above. The auxiliary array (1, 2, 4, 6, 8, 12, 17, 19, 21, 24, 32) of diagonal indices is also required. By convention the auxiliary integer array has length $n + 1$ with the $(n + 1)$st element containing the length of the packed real array plus one.

The general theory from the previous section applies equally well here, so the zero curve γ given by $(x(s), \lambda(s))$ is the trajectory of the initial value problem

$$\frac{d}{ds}\rho_a(x(s), \lambda(s)) = [D_x\rho_a(x(s), \lambda(s)), D_\lambda\rho_a(x(s), \lambda(s))] \begin{pmatrix} dx/ds \\ d\lambda/ds \end{pmatrix} = 0, \tag{9}$$

$$\left\| \left(\frac{dx}{ds}, \frac{d\lambda}{ds} \right) \right\|_2 = 1, \tag{10}$$

$$x(0) = a, \qquad \lambda(0) = 0. \tag{11}$$

Since the Jacobian matrix has rank n along γ, the derivative $(dx/ds, d\lambda/ds)$ is uniquely determined by (9, 10) and continuity, and the initial value problem (9–11) can be solved for $x(s)$, $\lambda(s)$. As before, the problem is to solve the initial value problem (9–11), which requires calculating the implicitly defined derivative (tangent vector) $(dx/ds, d\lambda/ds)$. The difficulty now is that the first n columns of the Jacobian matrix $D\rho_a(x, \lambda)$ are definitely special, and any attempt to treat all $n + 1$ columns uniformly would be disastrous from the point of view of storage allocation. Any algorithm requires computing the kernel of the $n \times (n + 1)$ matrix $D\rho_a(x, \lambda)$, which has rank n. This can be elegantly and efficiently done for small dense matrices, but the large sparse Jacobian matrix of structural mechanics presents special difficulties. The approach taken here is to solve $D\rho_a y = 0$ using a preconditioned conjugate gradient algorithm. This conjugate gradient algorithm will now be described.

Let $(\bar{x}, \overline{\lambda})$ be a point on the zero curve γ, and $\bar{y}$ the unit tangent vector to γ at $(\bar{x}, \overline{\lambda})$ in the direction of increasing arc length s. Let $|\bar{y}_k| = \max_i |\bar{y}_i|$. Then the matrix

$$A = \begin{bmatrix} D\rho_a(x, \lambda) \\ e_k^t \end{bmatrix} \tag{12}$$

where e_k is a vector with 1 in the kth component and zeros elsewhere, is invertible at $(\bar{x}, \overline{\lambda})$ and in a neighborhood of $(\bar{x}, \overline{\lambda})$ by continuity. Thus the kernel of $D\rho_a$ can be found by solving the linear system of equations

$$Ay = \bar{y}_k e_{n+1} = b. \tag{13}$$

Given any nonsymmetric, nonsingular matrix A, the system of linear equations $Ay = b$ can be solved by considering the linear system

$$AA^t z = b.$$

Since the coefficient matrix for this system is both symmetric and positive definite, the system can be solved by a conjugate gradient algorithm. Once a solution vector z is obtained, the vector y from the original system can be computed as $y = A^t z$. An implementation of the conjugate gradient algorithm in which y is computed directly, without reference to z, any approximations of z, or AA^t, is due to Craig [7], and is described in [9] and [13]. Each iterate y^i minimizes the Euclidean error norm $\| y - y^i \|$ over the translated Krylov space

$$y^0 + span\{r^0, AA^t r^0, (AA^t)^2 r^0, \ldots, (AA^t)^{i-1} r^0\},$$

where $r^0 = b - Ay^0$. Below $\langle u, v \rangle$ denotes the inner product $u^t v$.

Craig's Method:

```
    Choose $y^0$;
    Compute $r^0 = b - Ay^0$;
    Compute $p^0 = A^t r^0$;
For $i = 0$ step 1 until convergence do
BEGIN
 $\alpha_i = \langle r^i, r^i \rangle / \langle p^i, p^i \rangle$
 $y^{i+1} = y^i + \alpha_i p^i$
 $r^{i+1} = r^i - \alpha_i A p^i$
```

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 $F'': C^{n+1} \rightarrow C^{n+1}$ by

$$\begin{aligned} F_j''(y) &= F_j'(y), \quad j = 1, \ldots, n, \\ F_{n+1}''(y) &= u(y) - 1. \end{aligned} \tag{53}$$

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 the following theorem (see [19]).

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).

There is no practical theory to guide the scaling of polynomial systems. A common sense approach is to scale the variables and equations to minimize the sum of squares of the exponents of the coefficients. An advantage of this criterion is that it leads to algorithms that effect scaling by solving a single linear system, $A\ w = b$, where A depends only on the structure of the polynomial system (the degrees of the variables) and not on the values of the coefficients.

Let

$$F_i(x) = \sum_{j=1}^{n_i} \left[p_{ij} \prod_{k=1}^{n} x_k^{d_{i,jk}} \right], \quad i = 1, \ldots, n, \tag{54}$$

where p_{ij} is a real number and d_{ijk} is a nonnegative integer. The positive integer n_i is the number of terms in equation i. Call the variable scaling factors $v_1, \ldots, v_n$ and the equation scaling factors $e_1, \ldots, e_n$. Define a new set of independent variables z_k by

$$x_k = 10^{v_k} z_k, \quad \text{for } k = 1, \ldots, n, \tag{55}$$

and a new polynomial system $S(z) = 0$ by

$$S_i(z) = 10^{e_i} F_i(10^{v_1} z_1, \ldots, 10^{v_n} z_n), \quad i = 1, \ldots, n. \tag{56}$$

Let C^n denote n-dimensional complex Euclidean space, and define $G\colon C^n \to C^n$ by

$$G_j(x) = b_j x_j^{d_j} - a_j, \quad j = 1, \ldots, n, \tag{50}$$

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

$$\rho_c(\lambda, x) = (1 - \lambda)G(x) + \lambda F(x), \tag{51}$$

where $c = (a, b)$, $a = (a_1, \ldots, a_n) \in C^n$ and $b = (b_1, \ldots, b_n) \in C^n$. Let $d = d_1 \ldots d_n$ 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 λ approaches 1 or converge to solutions to $F(x) = 0$ as λ 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, e.g., [2], [6], [10], [20], [39]. The homotopy map in (51) is from [20]. 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 $F(x)$:

$$F_j'(y) = y_{n+1}^{d_j} F_j(y_1/y_{n+1}, \ldots, y_n/y_{n+1}), \quad j = 1, \ldots, n. \tag{52}$$

Note that, if $F'(y^0) = 0$, then $F(\alpha y^0) = 0$ for any complex scalar α. 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 $y_{n+1} = 0$*. However, to avoid dealing with CP^n, these solutions are often taken to be $x \in C^n$ such that

(a) $y = (x, 0)$ is a solution to $F'(y) = 0$, and

(b) x is nonzero and the first nonzero component is 1.

This second definition is adequate for some purposes, but it is incomplete. It is hard, for example, to give a natural definition of *nonsingular solution at infinity* using it. See [20, Chap. 3] for a more complete discussion.

A basic result on the structure of the solution set of a polynomial system is the following classical theorem of Bezout [27]:

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.*

```
    $\beta_i = \langle r^{i+1}, r^{i+1} \rangle / \langle r^i, r^i \rangle$
    $p^{i+1} = A^t r^{i+1} + \beta_i p^i$
END
```

Let Q be any nonsingular matrix. The solution to the system $Ay = b$ can be calculated by solving the system

$$By = (Q^{-1}A)y = Q^{-1}b = g. \tag{14}$$

The use of such a matrix is known as preconditioning. Since the goal of using preconditioning is to decrease the computational effort needed to solve the original system, Q should be some approximation to A. Then $Q^{-1}A$ would be close to the identity matrix, and the iterative method described above would converge more rapidly when applied to (14) than when applied to (13). In the following algorithm B and g are never explicitly formed. The algorithm given above can be obtained by substituting the identity matrix for Q.

Craig's method using a preconditioner:

```
    Choose $y^0$, $Q$;
    Compute $r^0 = b - Ay^0$;
    Compute $\tilde{r}^0 = Q^{-1}r^0$;
    Compute $p^0 = A^t Q^{-t} \tilde{r}^0$;
For $i = 0$ step 1 until convergence do
BEGIN
    $\alpha_i = \langle \tilde{r}^i, \tilde{r}^i \rangle / \langle p^i, p^i \rangle$
    $y^{i+1} = y^i + \alpha_i p^i$
    $\tilde{r}^{i+1} = \tilde{r}^i - \alpha_i Q^{-1} A p^i$
    $\beta_i = \langle \tilde{r}^{i+1}, \tilde{r}^{i+1} \rangle / \langle \tilde{r}^i, \tilde{r}^i \rangle$
    $p^{i+1} = A^t Q^{-t} \tilde{r}^{i+1} + \beta_i p^i$
END
```

For this algorithm, a minimum of $5(n + 1)$ storage locations is required (in addition to that for A). The vectors y, $\tilde{r}$, and p all require their own locations; $Q^{-t}\tilde{r}$ can share with Ap; $Q^{-1}Ap$ can share with $A^t Q^{-t} \tilde{r}$. The computation cost per iteration of this algorithm is:

(a) two preconditioning solves ($Q^{-1}v$ and $Q^{-t}v$);

(b) two matrix-vector products (Av and $A^t v$);

(c) $5(n + 1)$ multiplications (the inner products $\langle p, p \rangle$ and $\langle \tilde{r}, \tilde{r} \rangle$, αp, βp, and $\alpha Q^{-1} Ap$).

The coefficient matrix A in the linear system of equations (13), whose solution y yields the kernel of $D\rho_a(\bar{x}, \bar{\lambda})$, has a very special structure which can be exploited if (13) is attacked indirectly as follows. Note that the leading $n \times n$ submatrix of A is $D_x\rho_a$, which is symmetric and sparse, but possibly indefinite. Write

$$A = M + L \tag{15}$$

where

$$M = \begin{bmatrix} D_x\rho_a(\bar{x}, \overline{\lambda}) & c \\ c^t & d \end{bmatrix},$$

$$L = ue_{n+1}^t, \qquad u = \begin{pmatrix} D_\lambda\rho_a(\bar{x}, \overline{\lambda}) - c \\ 0 \end{pmatrix}.$$

The choice of e_k^t as the last row of A to make A invertible is somewhat arbitrary, and in fact any vector (c^t, d) outside a set of measure zero (a hyperplane) could have been chosen. Thus for almost all vectors c the first n columns of M are independent, and similarly almost all $(n + 1)$-vectors are independent of the first n columns of M. Therefore for almost all vectors (c^t, d) both A and M are invertible. Assume that (c^t, d) is so chosen.

Using the Sherman-Morrison formula (L is rank one), the solution y to the original system $Ay = b$ can be obtained from

$$y = \left[I - \frac{M^{-1}ue_{n+1}^t}{(M^{-1}u)^t e_{n+1} + 1}\right]M^{-1}b, \tag{16}$$

which requires the solution of two linear systems with the sparse, symmetric, invertible matrix M. It is the systems $Mz = u$ and $Mz = b$ to which Craig's preconditioned conjugate gradient algorithm is actually applied.

The only remaining detail is the choice of the preconditioning matrix Q. Q is taken as the modified Cholesky decomposition of M, as described by Gill and Murray [11]. If M is positive definite and well conditioned, $Q = M$. Otherwise, Q is a well-conditioned positive definite approximation to M. The use of a positive definite Q is reasonable since, in the context of structural mechanics, $DF(x)$ is positive definite or differs from a positive definite matrix by a low rank perturbation. The Gill-Murray factorization algorithm can exploit the symmetry and sparse skyline structure of M; this entire scheme, Equations (13–16), is built around using the symmetry and sparse skyline structure of the Jacobian matrix $D_x\rho_a = \lambda DF + (1 - \lambda)I$. Several other possible choices for the preconditioned conjugate gradient algorithm have been studied in detail by Chan and Saad [4].

For sparse problems, the logic of tracking the zero curve γ is exactly the same as for the dense Jacobian matrix case. The only difference is in the kernel calculation and the concomitant data structures (step 10 of the algorithm in the previous section), which are substantially more complicated for the sparse Jacobian matrix case. These low-level details are best left to the code, where they are thoroughly documented.

4. NORMAL-FLOW ALGORITHM (DENSE JACOBIAN MATRIX).

As the homotopy parameter vector a varies, the corresponding homotopy zero curve γ also varies. This family of zero curves is known as the Davidenko flow. The normal-flow algorithm is so called because the iterates converge to the zero curve γ along the flow normal to the Davidenko flow (in an asymptotic sense). As before, only the zero-finding case need be described.

The normal-flow algorithm has four phases: prediction, correction, step-size estimation, and computation of the solution at $\lambda = 1$. (2) and (3) are the relevant

involves implementing a more sophisticated control over the ideal starting error δ_k used in Eq. (41). The next estimate for the ideal starting error δ_k is computed using the exact error $\tilde{\delta}_k^{(0)}$ of the last predicted point, the size of the last Newton step $\tilde{\delta}_k^{(*)}$, and the number of iterations i_* required by the correction process.

$$\delta_k = \theta\ \tilde{\delta}_k^{(0)} \tag{47}$$

where θ is a function of $\tilde{\delta}_k^{(*)}$, $\tilde{\delta}_k^{(0)}$, and i_* as described by Rheinboldt [24].

The goal behind these calculations is to keep the number of corrector iterations fixed at four. Thus θ is computed so that if the prediction error had been δ_k rather than $\tilde{\delta}_k^{(0)}$, the number of correction steps would have been approximately four, instead of i_*. Once δ_k is computed, it is used in (41) to calculate the next step size. For sharply turning curves, this δ_k is too large for convergence in four corrector iterations consistently. Numerical experiments suggest that the desired behavior (convergence in four corrector iterations) is obtained by using the formula

$$\delta_k = \theta\ \delta_{k-1} \tag{48}$$

instead of (47). HOMPACK uses (48).

8. POLYNOMIAL SYSTEMS

This section describes the POLSYS driver for finding all complex solutions to polynomial systems with real coefficients. A system of n polynomial equations in n unknowns,

$$F(x) = 0, \tag{49}$$

may have many solutions. It is possible to define a homotopy so that all geometrically isolated solutions of (49) have at least one associated homotopy path. Generally, (49) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as λ approaches 1. However, (49) can be transformed into a new system that, 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, POLSYS includes a general scaling algorithm (SCLGNP). POLSYS uses an input "tableau" of coefficients and related parameters to define the polynomial system. This tableau is used to generate function and partial derivative values (via subroutine FFUNP). The user need not code any subroutine to be called by POLSYS.

Although the POLSYS homotopy map is defined in complex space, the POLSYS code does not use complex computer arithmetic. Using algebraic geometry, it can be shown that the homotopy parameter λ is strictly increasing as a function of arc length [21]. 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 [16, 26]. However, the sort of generic homotopy algorithm considered here must find all solutions and cannot be limited without, in essence, changing it into a heuristic.

```
24. Find $\bar{s}$ such that $p(\bar{s})_1 = 1$, using $yold$, $ypold$, $y$, and $yp$ in (17). $yopp := yold$, $Z^{(0)} := p(\bar{s})$.
25. $limit := 2(\lfloor -\log_{10}(ansae + ansre \| y \|) \rfloor + 1)$. Do steps 26–33 for $k = 2, \ldots, limit + 2$.
    26. Update the augmented Jacobian matrix using (37).
    27. Take a quasi-Newton step with (46).
    28. If
            $| P_1^{(k+1)} - 1 | + \| \Delta Z^{(k-2)} \| \le ansae + ansre \| Z^{(k-2)} \|$,
        then return (solution has been found).
    29. If $| P_1^{(k+1)} - 1 | \le ansae + ansre$,
        then
            $Z^{(k-1)} := P^{(k+1)}$
        else do steps 30–33.
    30. $yold := y$, $y := P^{(k+1)}$.
    31. If $yold_1$ and $y_1$ bracket $\lambda = 1$, then $yopp := yold$.
    32. Compute $Z^{(k-1)}$ with the linear predictor (42) using $y$ and $yold$.
    33. If $\| Z^{(k-1)} - y \| > \| y - yopp \|$, then compute $Z^{(k-1)}$ with the linear predictor
        (44) using $y$ and $yopp$.
34. Return with an error flag.
```

7. AUGMENTED (SPARSE) JACOBIAN MATRIX ALGORITHM

The augmented Jacobian matrix algorithm for sparse Jacobian matrices differs from the dense algorithm in three respects: (1) like the sparse normal flow and ODE-based algorithms, the low-level numerical linear algebra is changed to take advantage of the sparsity of the problem; (2) quasi-Newton iterations are abandoned in favor of pure Newton iterations; (3) Rheinboldt's step size control [24] is implemented more faithfully because of the use of Newton iterations. Except for these three changes, the logic for tracking the zero curve γ is exactly the same as for the dense algorithm.

The change in the linear algebra affects the algorithm in two places: computing the tangent vector (Eq. 31), and computing the corrector step (Eq. 35). Rather than using a QR factorization, these two linear equations are solved by using the preconditioned conjugate gradient algorithm described for the sparse ordinary differential equation-based algorithm (Eqs. 14–16). Note however that the matrices in (31) and (35) are different than the matrix in (12), in that the final row is a tangent vector rather than a standard basis vector.

The use of Newton iterations rather than quasi-Newton iterations is necessitated by the current lack of a good (comparable to Broyden or BFGS) sparse quasi-Newton update formula. The fill-in produced by a good (dense) update formula is unacceptable, and the efficacy of deferred updating [15] is questionable (the number of applications of the Sherman-Morrison formula grows exponentially with the number of deferred updates). Also there is some evidence that, at least in the context of structural mechanics [36], a model trust region strategy with exact (expensive) Jacobian matrix evaluations is better than (cheap) quasi-Newton updating. The effect of using Newton's method in the algorithm is to replace every quasi-Newton update with the calculation of the exact augmented Jacobian matrix.

The final change for the sparse-matrix algorithm is an enhancement to the step size control, allowed by the use of Newton iterations. The enhancement

equations here. For the prediction phase, assume that several points $P^{(1)} = (\lambda(s_1), x(s_1))$, $P^{(2)} = (\lambda(s_2), x(s_2))$ on γ with corresponding tangent vectors $(d\lambda/ds(s_1), dx/ds(s_1))$, $(d\lambda/ds(s_2), dx/ds(s_2))$ have been found, and h is an estimate of the optimal step (in arc length) to take along γ. The prediction of the next point on γ is

$$Z^{(0)} = p(s_2 + h), \tag{17}$$

where $p(s)$ is the Hermite cubic interpolating $(\lambda(s), x(s))$ at s_1 and s_2. Precisely,

$$p(s_1) = (\lambda(s_1), x(s_1)), \qquad p'(s_1) = (d\lambda/ds(s_1), dx/ds(s_1)),$$
$$p(s_2) = (\lambda(s_2), x(s_2)), \qquad p'(s_2) = (d\lambda/ds(s_2), dx/ds(s_2)),$$

and each component of $p(s)$ is a polynomial in s of degree less than or equal to 3.

Starting at the predicted point $Z^{(0)}$, the corrector iteration is

$$Z^{(k+1)} = Z^{(k)} - [D\rho_a(Z^{(k)})]^\dagger \rho_a(Z^{(k)}), \quad k = 0, 1, \ldots \tag{18}$$

where $[D\rho_a(Z^{(k)})]^\dagger$ is the Moore-Penrose pseudoinverse of the $n \times (n + 1)$ Jacobian matrix $D\rho_a$. Small perturbations of a produce small changes in the trajectory γ, and the family of trajectories γ for varying a is known as the "Davidenko flow." Geometrically, the iterates given by (18) return to the zero curve along the flow normal to the Davidenko flow, hence the name "normal-flow algorithm."

A corrector step ΔZ is the unique minimum norm solution of the equation

$$[D\rho_a]\Delta Z = -\rho_a. \tag{19}$$

Fortunately ΔZ can be calculated at the same time as the kernel of $[D\rho_a]$, and with just a little more work. Normally for dense problems the kernel of $[D\rho_a]$ is found by computing a QR factorization of $[D\rho_a]$, and then using back substitution. By applying this QR factorization to $-\rho_a$ and using back substitution again, a *particular* solution v to (19) can be found. Let $u \neq 0$ be any vector in the kernel of $[D\rho_a]$. Then the minimum norm solution of (19) is

$$\Delta Z = v - \frac{v^t u}{u^t u} u. \tag{20}$$

Since the kernel of $[D\rho_a]$ is needed anyway for the tangent vectors, solving (19) only requires another $O(n^2)$ operations beyond those for the kernel. The number of iterations required for convergence of (18) should be kept small (say < 4) since QR factorizations of $[D\rho_a]$ are expensive. The alternative of using $[D\rho_a(Z^{(0)})]$ for several iterations, which results in linear convergence, is rarely cost effective.

When the iteration (18) converges, the final iterate $Z^{(k+1)}$ is accepted as the next point on γ, and the tangent vector to the integral curve through $Z^{(k)}$ is used for the tangent—this saves a Jacobian matrix evaluation and factorization at $Z^{(k+1)}$. The step-size estimation described next attempts to balance progress along γ with the effort expended on the iteration (18).

Define a contraction factor

$$L = \frac{\| Z^{(2)} - Z^{(1)} \|}{\| Z^{(1)} - Z^{(0)} \|}, \tag{21}$$

a residual factor

$$R = \frac{\| \rho_a(Z^{(1)}) \|}{\| \rho_a(Z^{(0)}) \|}, \tag{22}$$

a distance factor ($Z^* = \lim_{k\to\infty} Z^{(k)}$)

$$D = \frac{\| Z^{(1)} - Z^* \|}{\| Z^{(0)} - Z^* \|}, \tag{23}$$

and ideal values $\bar{L}$, $\bar{R}$, $\bar{D}$ for these three. Let h be the current step size (the distance from Z^* to the previous point found on γ), and $\bar{h}$ the "optimal" step size for the next step. The goal is to achieve

$$\frac{\bar{L}}{L} \approx \frac{\bar{R}}{R} \approx \frac{\bar{D}}{D} \approx \frac{\bar{h}^q}{h^q} \tag{24}$$

for some q. This leads to the choice

$$\hat{h} = (\min\{\bar{L}/L,\ \bar{R}/R,\ \bar{D}/D\})^{1/q} h, \tag{25}$$

a worst-case choice. To prevent chattering and unreasonable values, constants $h_{\min}$ (minimum allowed step size), $h_{\max}$ (maximum allowed step size), $B_{\min}$ (contraction factor), and $B_{\max}$ (expansion factor) are chosen, and $\bar{h}$ is taken as

$$\bar{h} = \min\{\max\{h_{\min},\ B_{\min}h,\ \hat{h}\},\ B_{\max}h,\ h_{\max}\}. \tag{26}$$

There are eight parameters in this process: $\bar{L}$, $\bar{R}$, $\bar{D}$, $h_{\min}$, $h_{\max}$, $B_{\min}$, $B_{\max}$, q. HOMPACK permits the user to specify nondefault values for any of these. The choice of $\bar{h}$ from (26) can be refined further. If (18) converged in one iteration, then $\bar{h}$ should certainly not be smaller than h, hence set

$$\bar{h} := \max\{h, \bar{h}\} \tag{27}$$

if (18) only required one iteration.

To prevent divergence from the iteration (18), if (18) has not converged after K iterations, h is halved and a new prediction is computed. Every time h is halved the old value h_{old} is saved. Thus if (18) has failed to converge in K iterations sometime during this step, the new $\bar{h}$ should not be greater than the value h_{old} known to produce failure. Hence in this case

$$\bar{h} := \min\{h_{\text{old}}, \bar{h}\}. \tag{28}$$

Finally, if (18) required the maximum K iterations, the step size should not increase, so in this case set

$$\bar{h} := \min\{h, \bar{h}\}. \tag{29}$$

The logic in (27–29) is rarely invoked, but it does have a stabilizing effect on the algorithm.

The final phase, computation of the solution at $\lambda = 1$, begins when a point $P^{(2)}$ on γ is generated such that $P_1^{(2)} \geq 1$. The solution lies somewhere on γ between the previous point $P^{(1)}$ and $P^{(2)}$. The endgame now consists of iterating until convergence the sequence of steps: Inverse interpolation with the Hermite cubic

An exception to these linear prediction schemes occurs with the first step of the final phase. Since the tangents $T^{(1)}$ and $T^{(2)}$ at $P^{(1)}$ and $P^{(2)}$ are available, this information is used to generate a Hermite cubic polynomial $p(s)$ for calculating the first prediction point $Z^{(0)}$. This is done by finding the root $\bar{s}$ of the equation $p_1(s) = 1$. $Z^{(0)}$ is then given by

$$Z^{(0)} = p(\bar{s}). \tag{45}$$

After the predictor $Z^{(k-2)}$ has been determined, a quasi-Newton step is taken to get the point $P^{(k+1)}$. This step is defined by

$$P^{(k+1)} = Z^{(k-2)} + \Delta Z^{(k-2)}, \tag{46}$$

where $\Delta Z^{(k-2)}$ is the solution to (35). Again, the matrix in (35) is produced by the rank-one updates (36) and (37).

The alternating process of computing a prediction and taking a quasi-Newton step is repeated until the solution is found.

In summary, the algorithm is:

```
1. $s := 0$; $y := (0, a)$; $ypold := (1, 0)$; $h := 0.1$; $failed :=$ false; $firststep :=$ true; $arcae$,
   $arcre :=$ absolute, relative error tolerances for tracking $\gamma$; $ansae$, $ansre =$ absolute,
   relative error tolerances for the answer.
2. Compute the tangent $yp$ at $y$, using (31) and (32), and update the augmented Jacobian
   matrix using (36).
3. If $firststep =$ false then
      4. Compute the predicted point $Z^{(0)}$ with the cubic predictor (17) based on $yold$,
         $ypold$, $y$, $yp$.
   else
      5. Compute the predicted point $Z^{(0)}$ using a linear predictor based on $y$ and $yp$.
6. If $failed =$ true then
      7. Compute the augmented Jacobian matrix at $Z^{(0)}$.
      8. Compute the next iterate $Z^{(1)}$ using (33).
9. $limit := 2(\lfloor -\log_{10}(arcae + arcre \| y \|) \rfloor + 1)$. Repeat steps 10–11 until either

              $\| \Delta Z^{(k)} \| \leq arcae + arcre \| Z^{(k)} \|$
                                or
                   $limit$ iterations have been performed.

      10. Update the augmented Jacobian matrix using (37).
      11. Compute the next iterate using (33).
12. If the quasi-Newton iteration did not converge in $limit$ steps, then
      13. $h := h/2$; $failed :=$ true.
      14. If $h$ is unreasonably small, then return with an error flag.
      15. Go to 3.
16. Compute the tangent at the accepted iterate $Z^{(*)}$ using (31) and (32), and update the
    augmented Jacobian matrix using (36).
17. Compute the angle $\alpha$ between the current and previous tangents by (38).
18. If $\alpha > \pi/3$, then
      19. $h := h/2$; $failed :=$ true.
      20. If $h$ is unreasonably small, then return with an error flag.
      21. Go to 3.
22. $yold := y$, $ypold := yp$, $y := Z^{(*)}$, $yp :=$ tangent computed in step 16, $firststep :=$ false,
    $failed :=$ false.
23. If $y_1 < 1$, then compute a new step size $h$ by Eqs. (26, 28, 38–41) with $\xi_{min} = 0.01$,

          $\delta_k = \min\{(arcae + arcre \| y \|)^{1/4}, \frac{1}{2} \| y - yold \| \}$,

    and go to 3.
```

Since $\hat{\xi}_k$ can be negative, use

$$\xi_k = \max(\xi_{\min}, \hat{\xi}_k) \quad \text{for some small} \quad \xi_{\min} > 0, \tag{40}$$

as the predicted curvature for the next step.

The goal in estimating the optimal step size is to keep the error in the prediction $\| Z^{(0)} - Z^{(*)} \|$ relatively constant, so that the number of iterations required by the corrector will be stable. This is achieved by choosing the step size as

$$\hat{h} = \sqrt{\frac{2\delta_k}{\xi_k}}, \tag{41}$$

where δ_k represents the ideal starting error desired for the prediction step. δ_k is chosen as a function of the tolerance for tracking the curve and is also restricted to be no larger than half of Δs_k.

As with the normal-flow algorithm, additional refinements on the optimal step size are made in order to prevent chattering and unreasonable values. In particular, $\bar{h}$ is chosen to satisfy eqs. (26) and (28). This $\bar{h}$ is then used as the step size for the next step.

The final phase of the algorithm, computation of the solution at $\lambda = 1$, is entered when a point $P^{(2)}$ is generated such that $P_1^{(2)} \geq 1$. $P^{(2)}$ is the first such point, so the solution must lie on γ somewhere between $P^{(2)}$ and the previous point $P^{(1)}$. The algorithm for finding this solution is a two-step process that is repeated until the solution is found. First, starting from a point $P^{(k)}$, a prediction $Z^{(k-2)}$ for the solution is generated such that $Z_1^{(k-2)} = 1$. Second, a single quasi-Newton iteration is performed to produce a new point $P^{(k+1)}$ close to γ, but not necessarily on the hyperplane $\lambda = 1$.

Normally, the prediction $Z^{(k-2)}$ is computed by a secant method using the last two points $P^{(k)}$ and $P^{(k-1)}$:

$$Z^{(k-2)} = P^{(k)} + (P^{(k-1)} - P^{(k)}) \frac{(1 - P_1^{(k)})}{(P_1^{(k-1)} - P_1^{(k)})}. \tag{42}$$

However, this formula can potentially produce a disastrous prediction (e.g., if $| P_1^{(k-1)} - P_1^{(k)} | \ll | 1 - P_1^{(k)} |$), so an additional scheme is added to ensure that this does not happen. In order to implement this scheme, a point $P^{(opp)}$ must be saved. This point is chosen as the last point computed from a quasi-Newton step that is on the opposite side of the hyperplane $\lambda = 1$ from $P^{(k)}$. Thus, the points $P^{(opp)}$ and $P^{(k)}$ bracket the solution. The prediction $Z^{(k-2)}$ may be bad whenever the inequality

$$\| Z^{(k-2)} - P^{(k)} \| > \| P^{(k)} - P^{(opp)} \| \tag{43}$$

is true. In this case, $Z^{(k-2)}$ is recomputed from the equation

$$Z^{(k-2)} = P^{(k)} + (P^{(opp)} - P^{(k)}) \frac{(1 - P_1^{(k)})}{(P_1^{(opp)} - P_1^{(k)})}. \tag{44}$$

This chord method, while much safer than the secant method (42), is used only in the special case (43) because it has a much slower rate of convergence than the secant method.

(17) for $\bar{s}$ such that $p(\bar{s})_1 = 1$; two iterations of (18) starting with $Z^{(0)} = p(\bar{s})$; replacing either $P^{(1)}$ or $P^{(2)}$ by $Z^{(2)}$ such that the solution on γ is always bracketed by $P^{(1)}$ and $P^{(2)}$. A precise statement of the endgame and the convergence criterion are given below.

In summary, the algorithm is:

```
 1. $s := 0$, $y := (0, a)$, $h := 0.1$, firststep := true, arcae, arcre := absolute, relative error
    tolerances for tracking $\gamma$, ansae, ansre := absolute, relative error tolerances for the
    answer.
 2. If firststep = false then
      3. Compute the predicted point $Z^{(0)}$ using (17).
         else
      4. Compute the predicted point $Z^{(0)}$ using a linear predictor based on $y = (0, a)$
         and the tangent there.
 5. Iterate with equation (18) until either

                    $\| \Delta Z^{(k)} \| \leq arcae + arcre \, \| Z^{(k)} \|$
                                        or
                        4 iterations have been performed.

 6. If the Newton iteration (18) did not converge in 4 steps, then
      7. $h := h/2$.
      8. If $h$ is unreasonably small, then return with an error flag.
      9. Go to 2.
10. firststep := false.
11. If $y_1 < 1$, then compute a new step size $h$ by (21–29) and go to 2.
12. Do steps 13–18 some fixed number of times.
      13. Find $\bar{s}$ such that $p(\bar{s})_1 = 1$, using yold, ypold, y, yp in (17).
      14. Do two iterations of (18) starting with $Z^{(0)} = p(\bar{s})$, ending with $Z^{(2)}$.
      15. If

              $| Z_1^{(2)} - 1 | + \| \Delta Z^{(1)} \| \leq ansae + ansre \, \| Z^{(1)} \|$,

          then return (solution has been found).
      16. If $Z_1^{(2)} \geq 1$, then
            17. $y := Z^{(2)}$, $yp :=$ tangent at $Z^{(2)}$.
                else
            18. $yold := Z^{(2)}$, $ypold :=$ tangent at $Z^{(2)}$.
19. Return with an error flag.
```

5. NORMAL-FLOW ALGORITHM (SPARSE JACOBIAN MATRIX).

The logic of the predictor, corrector, and step-size estimation phases of this algorithm is identical to that given in the previous section. Similar to the ordinary differential equation-based algorithm, the difference between the dense and sparse Jacobian matrix cases is the low-level numerical linear algebra. The main linear algebra problem is the solution of (19), which also involves the calculation of the kernel of $D\rho_a(x, \lambda)$. (19) is solved using the same matrix splitting, preconditioning matrix, and conjugate-gradient algorithm used for the sparse ordinary differential equation-based algorithm (equations 12–16). For efficiency, the kernel and Newton step are calculated together by solving

$$\begin{bmatrix} D_x\rho_a(\bar{x}, \bar{\lambda}) & D_\lambda\rho_a(\bar{x}, \bar{\lambda}) \\ c^t & d \end{bmatrix} [v \quad \Delta Z] = \begin{bmatrix} 0 & -\rho_a(\bar{x}, \bar{\lambda}) \\ \bar{y}_k & 0 \end{bmatrix}. \tag{30}$$

6. AUGMENTED (DENSE) JACOBIAN MATRIX ALGORITHM.

The augmented Jacobian matrix algorithm has four major phases: prediction, correction, step-size estimation, and computation of the solution at $\lambda = 1$. Again, only the zero-finding case is described here. The algorithm here is based on Rheinboldt [24], but with some significant differences: (1) a Hermite cubic rather than a linear predictor is used; (2) a tangent vector rather than a standard basis vector is used to augment the Jacobian matrix of the homotopy map; (3) updated QR factorizations and quasi-Newton updates are used rather than Newton's method; (4) different step-size control, necessitated by the use of quasi-Newton iterations, is used; (5) a different scheme for locating the target point at $\lambda = 1$ is used that allows the Jacobian matrix of F to be singular at the solution $\bar{x}$ provided rank $D\rho_a(1, \bar{x}) = n$.

The prediction phase is exactly the same as in the normal flow algorithm. Having the points $P^{(1)} = (\lambda(s_1), x(s_1))$, $P^{(2)} = (\lambda(s_2), x(s_2))$ on γ with corresponding tangent vectors

$$T^{(1)} = \begin{pmatrix} \dfrac{d\lambda}{ds}(s_1) \\ \dfrac{dx}{ds}(s_1) \end{pmatrix}, \qquad T^{(2)} = \begin{pmatrix} \dfrac{d\lambda}{ds}(s_2) \\ \dfrac{dx}{ds}(s_2) \end{pmatrix},$$

the prediction $Z^{(0)}$ of the next point on γ is given by (17).

In order to use this predictor, a means of calculating the tangent vector $T^{(2)}$ at a point $P^{(2)}$ is required. This is done by solving the system

$$\begin{bmatrix} D\rho_a(P^{(2)}) \\ T^{(1)t} \end{bmatrix} z = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{31}$$

for z, where $D\rho_a$ is the $n \times (n+1)$ Jacobian of ρ_a. Normalizing z gives

$$T^{(2)} = \frac{z}{\| z \|}. \tag{32}$$

The last row of (31) insures that the tangent $T^{(2)}$ makes an acute angle with the previous tangent $T^{(1)}$. It is the augmentation of the Jacobian matrix with this additional row that motivates the name "augmented Jacobian matrix algorithm." The solution to (31) is found by computing a QR factorization of the matrix, and then using back substitution [8].

Starting with the predicted point $Z^{(0)}$, the correction is performed by a quasi-Newton iteration defined by

$$Z^{(k+1)} = Z^{(k)} - \begin{bmatrix} A^{(k)} \\ T^{(2)t} \end{bmatrix}^{-1} \begin{pmatrix} \rho_a(Z^{(k)}) \\ 0 \end{pmatrix}, \quad k = 0, 1, \ldots \tag{33}$$

where $A^{(k)}$ is an approximation to the Jacobian matrix $D\rho_a(Z^{(k)})$. The last row of the matrix in (33) insures that the iterates lie in a hyperplane perpendicular to the tangent vector $T^{(2)}$. (33) is the quasi-Newton iteration for solving the augmented nonlinear system

$$\begin{pmatrix} \rho_a(y) \\ T^{(2)t}(y - Z^{(0)}) \end{pmatrix} = 0. \tag{34}$$

A corrector step $\Delta Z^{(k)}$ is the unique solution to the equation

$$\begin{bmatrix} A^{(k)} \\ T^{(2)t} \end{bmatrix} \Delta Z^{(k)} = \begin{pmatrix} -\rho_a(Z^{(k)}) \\ 0 \end{pmatrix}. \tag{35}$$

The matrix on the left side of this equation is produced by successive Broyden rank-one updates [8] of the matrix in (31). Precisely, letting $Z^{(-1)} = P^{(2)}$, $A^{(-1)} = D\rho_a(P^{(2)})$, and

$$M^{(k)} = \begin{bmatrix} A^{(k)} \\ T^{(2)t} \end{bmatrix},$$

the update formulas are

$$M^{(-1)} = \begin{bmatrix} A^{(-1)} \\ T^{(2)t} \end{bmatrix} = \begin{bmatrix} D\rho_a(P^{(2)}) \\ T^{(1)t} \end{bmatrix} + \hat{e}_{n+1}(T^{(2)} - T^{(1)})^t, \tag{36}$$

and

$$M^{(k+1)} = M^{(k)} + \frac{(\tilde{\Delta}\rho_a - M^{(k)}\Delta Z^{(k)})\Delta Z^{(k)t}}{\Delta Z^{(k)t}\Delta Z^{(k)}}, \quad k = -1, 0, \ldots \tag{37}$$

where

$$\tilde{\Delta}\rho_a = \begin{pmatrix} \rho_a(Z^{(k+1)}) - \rho_a(Z^{(k)}) \\ 0 \end{pmatrix}.$$

These updates can be done in QR-factored form, requiring a total of $O(n^2)$ operations for each iteration in the correction process [8]. When the iteration (33) converges within some tolerance, the final iterate $\bar{Z}^{(*)}$ is accepted as the next point on the zero curve γ.

The step-size estimation algorithm is an adaptation of a procedure developed by Rheinboldt [24]. At each point $P^{(k)}$ with tangent $T^{(k)}$ along γ, the curvature is estimated by the formula

$$\| w^{(k)} \| = \frac{2}{\Delta s_k} \, | \sin(\alpha_k/2) |, \tag{38}$$

where

$$w^{(k)} = \frac{T^{(k)} - T^{(k-1)}}{\Delta s_k}, \qquad \alpha_k = \arccos(T^{(k)t}T^{(k-1)}), \qquad \Delta s_k = \| P^{(k)} - P^{(k-1)} \|.$$

Intuitively, α_k represents the angle between the last two tangent vectors, and the curvature is approximated by the Euclidean norm of the difference between these two tangents divided by Δs_k.

This curvature data can be extrapolated to produce a prediction for the curvature for the next step

$$\hat{\xi}_k = \| w^{(k)} \| + \frac{\Delta s_k}{\Delta s_k + \Delta s_{k-1}} \left(\| w^{(k)} \| - \| w^{(k-1)} \| \right). \tag{39}$$