Algorithm 776: HOMPACK90: A Suite of Fortran 90 Codes for Globally Convergent Homotopy Algorithms

LAYNE T. WATSON, MARIA SOSONKINA
Virginia Polytechnic Institute and State University
ROBERT C. MELVILLE
Lucent Technologies
ALEXANDER P. MORGAN
General Motors Research and Development Center
and
HOMER F. WALKER
Utah State University

HOMPACK90 is a Fortran 90 version of the FORTRAN 77 package HOMPACK (Algorithm 652), a collection of codes for finding zeros or fixed points of nonlinear systems using globally convergent probability-one homotopy algorithms. Three qualitatively different algorithms—ordinary differential equation based, normal flow, quasi-Newton augmented Jacobian matrix—are provided for tracking homotopy zero curves, as well as separate routines for dense and sparse Jacobian matrices. A high level driver for the special case of polynomial systems is also provided. Changes to HOMPACK include numerous minor improvements, simpler and more elegant interfaces, use of modules, new end games, support for several sparse matrix data structures, and new iterative algorithms for large sparse Jacobian matrices.

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, curve tracking, fixed point, Fortran 90, globally convergent, homotopy methods, polynomial systems, probability-one, zero

This work was supported in part by Air Force Office of Scientific Research grant F49620-92-J-0236, Department of Energy grants DE-FG05-88ER25068/A004 and DE-FG03-94ER25221, National Science Foundation grant DMS-9400217, and National Aeronautics and Space Administration grant NAG-1-1562.
Authors' addresses: M. Sosonkina, L. T. Watson, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106; R. C. Melville, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974-2070; A. P. Morgan, MC 480-106-285, Manufacturing and Design Systems Department, General Motors Research and Development Center, 30500 Mound Road, Warren, MI 48090-9055; H. F. Walker, Department of Mathematics and Statistics, Utah State University, Logan, UT 84322.

1. INTRODUCTION

Two decades ago an introduction to a paper on homotopy methods would have had to justify why homotopy methods were being considered at all, offer apologies for the inefficiency of homotopy methods relative to locally convergent methods such as quasi-Newton, and carefully circumscribe the potential applicability and advantages of homotopy algorithms. Now, after hundreds of significant applications ranging from aircraft design to statistical parameter estimation, reasonable theoretical development, and several notable attempts at algorithm and code development, such prefatory remarks seem unnecessary. In some application areas—analog circuit simulation, linkage mechanism design, geometric (CAD/CAM) modelling, H^2/H^∞ controller analysis and synthesis—homotopy methods are now the method of choice. In mechanism design, for instance, where all solutions must be found, there is no other viable alternative to homotopy methods. Surveys of homotopy methods and applications are by Watson [1986], Watson [1990], and most recently the book by Allgower and Georg [1990], which contains an extraordinary collection of references on both simplicial and continuous homotopies. The distinction between continuation, homotopy, and probability-one homotopy methods is frequently blurred in the literature, but there are fundamental differences, which are detailed by Ge et al. [1996]. The efficiency issue is now moot, since good implementations of probability-one homotopy algorithms are competitive with quasi-Newton methods in raw CPU time, and the growing emphasis on software engineering and productivity argues that the human time lost in finding good starting points and nursing locally convergent methods to physically meaningful solutions is far more valuable than the extra CPU cycles a globally convergent homotopy algorithm might take. Serious computer code development has occurred, some examples being CONSOL [Morgan 1987], CONKUB [Mejia 1986], PITCON [Rheinboldt and Burkardt 1983], FIXPT [Watson and Fenner 1980], and HOMPACK [Watson, Billups, and Morgan 1987].

The basic theory of globally convergent probability-one homotopy algorithms will be summarized here for future reference, drawing from the results of Chow, Mallet-Paret, and Yorke [1978], Watson [1979a], Watson [1979b], Watson [1979c], Watson [1980a], and Watson [1986]. The theoretical foundation of all probability-one globally convergent homotopy methods is given by the following definition and theorem from differential geometry.

DEFINITION. *Let $U \subset \mathbf{R}^m$ and $V \subset \mathbf{R}^p$ be open sets, and let $\rho : U \times [0,1) \times V \to \mathbf{R}^p$ be a C^2 map. ρ is said to be transversal to zero if the Jacobian matrix $D\rho$ has full rank on $\rho^{-1}(0)$.*

TRANSVERSALITY THEOREM. *If $\rho(a,\lambda,x)$ is transversal to zero, then for almost all $a \in U$ the map*

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

is also transversal to zero; i.e., with probability one the Jacobian matrix $D\rho_a(\lambda,x)$ has full rank on $\rho_a^{-1}(0)$.

To solve the nonlinear system of equations

$$f(x) = 0,$$

where $f : \mathbf{R}^p \to \mathbf{R}^p$ is a C^2 map, the general probability-one homotopy paradigm is to construct a C^2 homotopy map $\rho : U \times [0,1) \times \mathbf{R}^p \to \mathbf{R}^p$ such that

1) $\rho(a, \lambda, x)$ is transversal to zero,

for each fixed $a \in U$,

2) $\rho_a(0, x) = \rho(a, 0, x) = 0$ is trivial to solve and has a unique solution x_0,

3) $\rho_a(1, x) = f(x)$,

4) the connected component of $\rho_a^{-1}(0)$ containing $(0, x_0)$ is bounded.

Then (from the Transversality Theorem) for almost all $a \in U$ there exists a zero curve γ of ρ_a, along which the Jacobian matrix $D\rho_a$ has rank p, emanating from $(0, x_0)$ and reaching a zero $\bar{x}$ of f at $\lambda = 1$. This zero curve γ has no bifurcations (i.e., γ is a smooth 1-manifold), and has finite arc length in every compact subset of $[0, 1) \times \mathbf{R}^p$. Furthermore, if $Df(\bar{x})$ is nonsingular, then γ has finite arc length. The complete homotopy paradigm is now apparent: construct the homotopy map ρ_a and then track its zero curve γ from the known point $(0, x_0)$ to a solution $\bar{x}$ at $\lambda = 1$. ρ_a is called a *probability-one* homotopy because the conclusions hold almost surely with respect to a, i.e., with probability one. Since the vector a, and indirectly the starting point x_0, are essentially arbitrary, an algorithm to follow the zero curve γ emanating from $(0, x_0)$ until a zero $\bar{x}$ of $f(x)$ is reached (at $\lambda = 1$) is legitimately called *globally convergent.* Note that this is much stronger than the "global convergence" claimed for trust region quasi-Newton methods, where the guaranteed convergence is to a stationary point of some merit function, which is not even necessarily a zero of f. Local methods require assumptions such as convexity or monotonicity to ensure that the stationary point is indeed a zero of f; the global convergence of homotopy methods to a true zero holds under far weaker assumptions. Essentially, the local behavior of f is irrelevant—only the behavior of f and ρ_a "in the large" matter. One final observation is that λ need not increase monotonically along γ, a feature which distinguishes homotopy from classical continuation methods (this is illustrated by the circuit problem in Section 10).

A different version of the above result, for the special case where the homotopy map ρ_a is a simple convex combination, is [Watson 1986]

THEOREM. *Let $F : \mathbf{R}^p \to \mathbf{R}^p$ be a C^2 map such that for some $r > 0$ and $\tilde{r} > 0$, $F(x)$ and $x - a$ do not point in opposite directions for $\|x\| = r$, $\|a\| < \tilde{r}$. Then F has a zero in $\{x \in \mathbf{R}^p \mid \|x\| \le r\}$, and for almost all $a \in \mathbf{R}^p$, $\|a\| < \tilde{r}$, there is a zero curve γ of*

$$\rho_a(\lambda, x) = \lambda F(x) + (1 - \lambda)(x - a),$$

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

The conditions in the above existence theorems are difficult to verify for practical problems, but less general special cases (e.g., $\|x\| = r \Rightarrow x^t F(x) \ge 0$ for some sufficiently large r) can often be verified in practice. Furthermore, sometimes watching the behavior of divergent homotopy zero curves can suggest a homotopy map that *will* lead to convergence.

HOMPACK90 retains the same three basic curve tracking algorithms of HOMPACK: ordinary differential equation-based, normal flow, and quasi-Newton augmented Jacobian matrix. The normal flow algorithm is usually the most efficient, but there are situations where one of the other methods is preferable. A simple, easy to use driver is provided for the special case when $F(x)$ is a polynomial system. This driver makes no attempt to find common subexpressions (e.g., if x_1^3 occurs in several components of $F(x)$ it is reevaluated each time) and tracks the total degree number of paths, ignoring any structure in $F(x)$. This is well behind the current state of the art in polynomial systems [Morgan, Sommese, and Wampler 1995], [Verschelde and Cools 1993], [Verschelde et al. 1994], but developing production quality code that exploits polynomial system structure is a formidable task. The code in HOMPACK90 is, however, perfectly adequate for small polynomial systems (with total degree $< 10^4$, say). Since the curve tracking algorithms are described in detail by Watson, Billups, and Morgan [1987], this paper will concentrate on the changes and additions to the algorithms in HOMPACK [Watson, Billups, and Morgan 1987].

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

Depending on the problem, the homotopy map $\rho_a(\lambda, x)$ may be given by

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

for the Brouwer fixed point problem

$$x = f(x), \qquad f : B \to B, \tag{1b}$$

where $B \subset \mathbf{R}^p$ is the closed unit ball, or by

$$\rho_a(\lambda, x) = \lambda\ F(x) + (1 - \lambda)(x - a) \tag{2a}$$

or

$$\rho_a(\lambda, x) = \lambda\ F(x) + (1 - \lambda)G(x; a) \tag{2b}$$

for the zero finding problem

$$F(x) = 0, \qquad F : \mathbf{R}^p \to \mathbf{R}^p, \tag{2c}$$

where $G(x; a) = 0$ is a "simple" version of $F(x) = 0$. For example, in a fluids problem, G might be identical to F except with the Reynolds number $R = 0$, or in a circuit problem, G might correspond to F with the current gain of all the transistors set to zero, or in a structural mechanics problem, G might correspond to F with very stiff elements. Often the most effective homotopy maps ρ_a are nonlinear in λ, e.g., where λ parametrizes the geometry in a fluids problem, or where $\lambda = 0$ represents a simple transistor model, or where $\lambda = 0$ corresponds to simple material constitutive relations. The role of the parameter vector a in the homotopy maps above is crucial, not just for the probability-one theory, but also in practice. Think of the random a as providing "symmetry breaking," which also has desirable numerical consequences. Note that the dimension m of a need not equal p, and indeed it may be advantageous to have $m \gg p$.

The details for the fixed point, zero finding, and general homotopy map cases are similar, so for the sake of brevity, only the zero finding problem (2c) with homotopy map (2a) 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 parametrized by arc length s. Thus $\lambda = \lambda(s)$, $x = x(s)$ along γ, and

$$\rho_a(\lambda(s), x(s)) = 0 \tag{3}$$

identically in s. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s)) \begin{pmatrix} \dfrac{d\lambda}{ds} \\ \dfrac{dx}{ds} \end{pmatrix} = 0, \tag{4}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{5}$$

With the initial conditions

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

the zero curve γ is the trajectory of the initial value problem (4–6). 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 [Shampine and Gordon 1975], [Watson 1979a]. Note that (4) only implicitly defines the derivative, but Watson, Billups, and Morgan [1987] describes robust numerical linear algebra procedures to deal with this. Another subtlety is that solving (4–6) does not explicitly enforce (3), and no matter how good the ODE solver is, the computed solution will drift away from γ. This is not really a problem, though, since a computed point $(\tilde{\lambda}, \tilde{x})$ lies on some curve in the Davidenko flow [Allgower and Georg 1990], corresponding to initial point $\tilde{a}$ (which can be computed from $(\tilde{\lambda}, \tilde{x})$). Thus $\rho_{\tilde{a}}(\tilde{\lambda}, \tilde{x}) = 0$ exactly, and the trajectory of $\rho_{\tilde{a}} = 0$ is followed starting from $(\tilde{\lambda}, \tilde{x})$. There is a bundle of curves around γ all leading to the same solution point $(1, \bar{x})$, and it is only necessary to stay in this bundle [Watson 1979a]. This self-correcting property of homotopy algorithms makes them inherently stable.

Once the hyperplane $\lambda = 1$ is crossed, the point $(1, \bar{x})$ on γ is computed by interpolation from the ODE solver's mesh points and a secant method root finding iteration (precisely, subroutines INTRP and ROOT from Shampine and Gordon [1975]). The ODE codes from Shampine and Gordon [1975] used in HOMPACK are well established and reliable, so changes were confined to replacing arithmetic IF statements, converting labeled nonblock to nonlabeled block DO loops, and improving portability by inserting a USE statement to access the KIND parameter for real variables and using it in declarations and in real constants.

3. 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, alluded to in the previous section, is known as the Davidenko flow [Allgower and Georg 1990]. 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). The normal flow iteration is essentially Newton's method with a rectangular $p \times (p+1)$ Jacobian matrix, although the rectangular matrix introduces some complications that must be handled with care (see Watson [1986] and Watson, Billups, and Morgan [1987]). As before, only the zero finding case need be described—(2a) and (2c) are the relevant equations here.

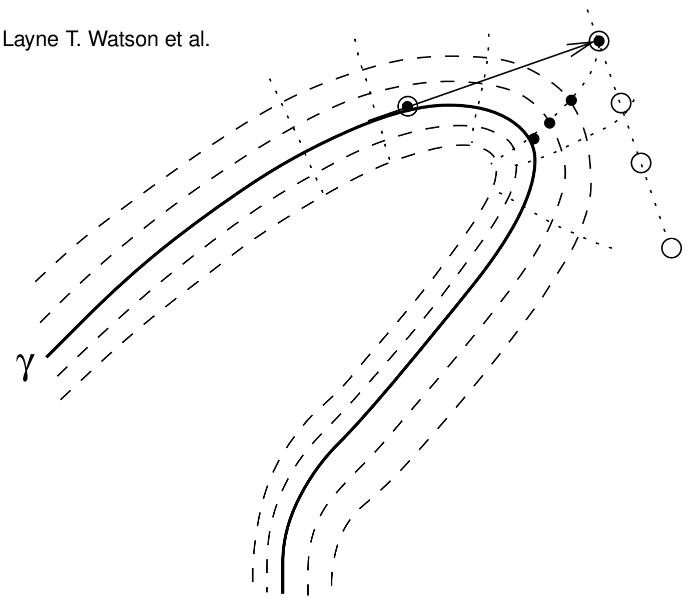

Fig. 1. Davidenko flow, normal flow iterates •, and augmented Jacobian matrix iterates ∘.

The normal flow algorithm has four phases: prediction, correction, step size estimation, and computation of the solution at $\lambda = 1$ (called the "end game"). 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

$$Z^{(0)} = p(s_2 + h), \tag{7}$$

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

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

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 mathematically is

$$Z^{(k+1)} = Z^{(k)} - \left[D\rho_a\big(Z^{(k)}\big)\right]^{\dagger} \rho_a\big(Z^{(k)}\big), \qquad k = 0, 1, \ldots \tag{8}$$

where $\left[D\rho_a(Z^{(k)})\right]^{\dagger}$ is the Moore-Penrose pseudoinverse of the $p \times (p+1)$ Jacobian matrix $D\rho_a$. Computationally the corrector step $\Delta Z = Z^{(k+1)} - Z^{(k)}$ is the unique minimum norm solution of the equation

$$\left[D\rho_a\right]\Delta Z = -\rho_a. \tag{9}$$

Small perturbations of a produce small changes in the trajectory γ. Geometrically, the iterates given by (8) return to the zero curve γ along the flow normal to the Davidenko flow (the family of trajectories γ for varying a), hence the name "normal flow algorithm" (cf. Figure 1). Robust and accurate numerical linear algebra procedures for solving (9) and for computing the kernel of $\left[D\rho_a\right]$ (required for (7)) are described in detail by Watson, Billups, and Morgan [1987].

When the iteration (8) 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 next phase, step size estimation, attempts to balance progress along γ with the effort expended on the iteration (8), and is a sophisticated blend of mathematics, computational experience, and mathematical software principles. Qualitatively, three different measures of "progress" toward $Z^* = \lim_{k\to\infty} Z^{(k)}$ are employed, and the optimal step size is computed by comparing the actual values of the measures with their ideal values. These ideal values are input arguments (see array SSPAR) to the user called subroutines, and their default values are the result of considerable computational experience with practical problems. This predicted optimal step size for the next step is adjusted to prevent chattering, and also to reflect the convergence history and achievable machine precision. This phase is identical to that in HOMPACK [Watson, Billups, and Morgan 1987].

The "end game" phase to compute the solution $(1, \bar{x})$ in the hyperplane $\lambda = 1$, which occurs when γ is tracked past $\lambda = 1$, is different from that in HOMPACK, and is described in detail here. The end game of the algorithm starts from two points $P^{(1)}$ and $P^{(2)}$ such that $P_1^{(1)} < 1$ and $P_1^{(2)} \geq 1$. Thus initially, the solution at $\lambda = 1$ lies somewhere between the previous point $P^{(1)}$ and $P^{(2)}$, i.e., $P^{(1)}$ and $P^{(2)}$ bracket the solution. When the end game is first entered, $P^{(1)}$ and $P^{(2)}$ bracket $\lambda = 1$, and the first predicted point $Z^{(0)}$ is computed as in the original normal flow algorithm in HOMPACK: using the Hermite cubic interpolation and root finding. In each subsequent step, the prediction $Z^{(k-2)}$ is generated by the secant method, namely

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

This equation results in a disastrous prediction when $\left|P_1^{(k-1)} - P_1^{(k)}\right| \ll \left|1 - P_1^{(k)}\right|$, and can also simply generate divergent approximations. To stabilize the iteration, the last point $P^{(opp)}$ on the opposite side of the hyperplane $\lambda = 1$ from $P^{(k)}$ must always be saved to bracket the solution.

In the case when $\left\|Z^{(k-2)} - P^{(k)}\right\| > \left\|P^{(k)} - P^{(opp)}\right\|$, the chord method

$$Z^{(k-2)} = P^{(k)} + \left(P^{(opp)} - P^{(k)}\right) \frac{\left(1 - P_1^{(k)}\right)}{\left(P_1^{(opp)} - P_1^{(k)}\right)}$$

is substituted for the secant method. Since the chord method has a slower rate of convergence, its application is limited to this special case.

The correction is implemented as a single Newton step

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

where $\Delta Z^{(k-2)}$ is the minimum norm solution to

$$\left[D\rho_a(Z^{(k-2)})\right] \Delta Z^{(k-2)} = -\rho_a(Z^{(k-2)}).$$

$P^{(opp)}$ is updated appropriately. This two step process (linear interpolation followed by a Newton correction) is repeated until the solution at $\lambda = 1$ is found, or a limit on the number of iterations is exceeded. This limit in HOMPACK was either taken as a constant or as

$$limit := 2\Big(\big\lfloor -\log_{10}(ansae + ansre\|P^{(2)}\|)\big\rfloor + 1\Big),$$

where *ansae* and *ansre* are absolute and relative error tolerances for the solution $\bar{x}$, respectively, supplied by the user. Depending on the problem scale, *limit* can be negative, so HOMPACK90 takes this limit as

$$limit := 2\Big(\big\lfloor \left|\log_{10}(ansae + ansre)\right| \big\rfloor + 1\Big).$$

There are numerous plausible alternatives to the end game just described: iterated Hermite cubic interpolation using bracketing points (the scheme in the HOMPACK normal flow algorithm), several Newton corrections instead of one per interpolation, projecting each Newton iterate along the tangent direction onto the hyperplane $\lambda = 1$, projecting the Newton iterate orthogonally onto $\lambda = 1$, simply scaling the Newton iterate to lie in $\lambda = 1$. All of these and other schemes have been tried, but the scheme described above is the best overall [Sosonkina, Watson, and Stewart 1996].

The normal flow algorithm for dense Jacobian matrices in HOMPACK90 is identical to that in HOMPACK, except for the end game and a few other minor improvements like the iteration limit mentioned above. This new end game is essentially the same as the end game used for the augmented Jacobian matrix algorithm in HOMPACK. Pseudocode is given for all the algorithms, including the end games, by Watson, Billups, and Morgan [1987], and so need not be repeated here since the modifications are obvious. All of the normal flow subroutines were rewritten in Fortran 90, and some LINPACK [Dongarra et al. 1979] subroutines were replaced by LAPACK [Anderson et al. 1995] routines. The changes include extensive use of array sections and intrinsics, named block DO and CASE constructs, optional arguments, interface blocks, automatic and assumed-shape arrays, and modules, resulting in code that is more elegant, easier to read, and has much simpler user interfaces.

4. AUGMENTED (DENSE) JACOBIAN MATRIX ALGORITHM

The quasi-Newton augmented Jacobian matrix algorithm is inspired by Rheinboldt and Burkardt [1983], but differs in several important respects: (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 which allows the Jacobian matrix of F to be singular at the solution $\bar{x}$ provided rank $D\rho_a(1, \bar{x}) = p$. Like the normal flow algorithm, the quasi-Newton augmented Jacobian matrix algorithm has four distinct phases: prediction, correction, step size estimation, and computation of the solution at $\lambda = 1$. Again, only the zero finding case is described here. The goal is to minimize the number of Jacobian matrix evaluations, and to use (rather complicated) step size control and quasi-Newton updates to achieve that goal. This scheme, when properly tuned, can be spectacularly efficient (in terms of number of Jacobian matrix evaluations) compared to the normal flow and ODE-based algorithms. However, it can also be erratic, and is not as robust without fine tuning as the other two algorithms [Billups 1985]. This quasi-Newton augmented Jacobian matrix algorithm is best reserved for situations where Jacobian matrix evaluation is at a premium, and some fine tuning of tracking parameters is acceptable.

The prediction phase is exactly the same as in the normal flow algorithm. Starting with 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 (7). It is assumed that $P^{(1)}$ and $P^{(2)}$ are close enough together so that the inner product $T^{(1)} \cdot T^{(2)} > 0$.

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

$$Z^{(k+1)} = Z^{(k)} - \begin{bmatrix} A^{(k)} \\ T^{(2)t} \end{bmatrix}^{-1} \begin{pmatrix} \rho_a\left(Z^{(k)}\right) \\ 0 \end{pmatrix}, \qquad k = 0, 1, \ldots \tag{10}$$

where $A^{(k)}$ is a quasi-Newton approximation to the Jacobian matrix $D\rho_a\left(Z^{(k)}\right)$. The last row of the matrix in (10) insures that the iterates lie in a hyperplane perpendicular to the tangent vector $T^{(2)}$ (cf. Figure 1). It is the augmentation of the Jacobian matrix with this additional row which motivates the name "augmented Jacobian matrix algorithm." Details for solving (10) and updating the augmented Jacobian matrix approximation are provided by Watson, Billups, and Morgan [1987], and such quasi-Newton updates are analyzed by Walker and Watson [1990].

The step size estimation algorithm, an adaptation of that by Rheinboldt and Burkardt [1983], is derived from curvature estimates and empirical convergence data. 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 ($Z^{(*)} = \lim_{k\to\infty} Z^{(k)}$). As with the normal flow algorithm, additional refinements on the optimal step size are made in order to prevent chattering and unreasonable values.

The final phase of the algorithm, computation of the solution at $\lambda = 1$, is a combination of the chord method, the secant method, and quasi-Newton corrector iterations (10). The details are very similar to, but not identical to, those for the normal flow algorithm end game. All four phases of the quasi-Newton augmented Jacobian matrix algorithm in HOMPACK90 are identical to those of the augmented Jacobian matrix algorithm in HOMPACK, for which pseudocode is given by Watson, Billups, and Morgan [1987]. The only differences in the new version are a few minor improvements in convergence criteria, more elegant argument lists, and much improved code readability, owing to Fortran 90 features such as array sections and intrinsics, named block DO and CASE constructs, optional arguments, interface blocks, automatic and assumed-shape arrays, and modules.

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

The sparse matrix codes in HOMPACK90 are a major upgrade to those in HOMPACK. HOMPACK was purposely designed so that the linear algebra subroutines could be easily replaced with others of users' choosing, and that in fact has been done (for the sparse codes) more often than not with sparse direct factorization methods [Melville et al. 1993a], [Melville et al. 1993b]. The goals of the present sparse matrix work are (1) to provide sparse iterative linear algebra algorithms that are good enough and general enough to suffice for most users, and (2) simultaneously maintain a clean interface to the sparse linear algebra subroutines so that they can be easily replaced with sparse direct factorization or sparse iterative algorithms. The only sparse matrix data structure supported by HOMPACK was the packed skyline format, and the iterative linear system solver was Craig's method [Craig 1954] for nonsymmetric matrices, using Gill-Murray preconditioning [Gill and Murray 1974]. The target application for that code was structural mechanics, where the equilibrium equations have the form

$$\rho(x, \lambda) = F(x) - \lambda\, v = 0,$$

and the tangent stiffness matrix $D_x\rho(x, \lambda) = DF(x)$, being the Hessian of a potential energy function, is symmetric. But homotopy methods involve $D\rho(x, \lambda)$, and ultimately nonsymmetric and symmetric indefinite matrices, so classical iterative methods designed for symmetric, positive definite matrices are not applicable.

However, there is a wide class of application areas (besides structural mechanics) for which $D_x\rho(x, \lambda)$ is symmetric, and a packed skyline storage format for $D_x\rho$ is appropriate. Therefore the packed skyline format for symmetric Jacobian matrices is retained as an option in HOMPACK90. Another option in HOMPACK90 is a very general sparse row storage format, intended for arbitrary sparsity patterns. Both of these sparse matrix storage formats are discussed in detail below.

Besides Gill-Murray preconditioning (for symmetric matrices stored in packed skyline format), HOMPACK90 offers an incomplete LU factorization (technically known as ILU(0)) as a preconditioner for the sparse row storage format. Both of these preconditioners are described below. Preconditioning strategies are an active research topic, and certainly preconditioners other than the two in HOMPACK90 may be better for some problems, but these two should suffice for most applications.

5.1 Packed Skyline Storage Format

For sparse problems it is convenient to write the homotopy map as

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

with the order of the arguments reversed (this is an internal matter to HOMPACK90 and causes no confusion at the user interface, since the user only specifies

$DF(x)$ in the fixed point and zero finding cases). The assumption here is that the matrix $D_x\rho_a(x,\lambda)$ is symmetric and sparse with a "skyline" structure, such as

$$\begin{pmatrix}
\bullet_1 & \bullet_3 & & & & & & & & \\
\bullet & \bullet_2 & \bullet_5 & & \bullet_{11} & \bullet_{16} & & & & \\
 & \bullet & \bullet_4 & \bullet_7 & \bullet_{10} & \bullet_{15} & & & & \bullet_{31} \\
 & & \bullet & \bullet_6 & \bullet_9 & \bullet_{14} & & & & \bullet_{30} \\
 & \bullet & \bullet & \bullet & \bullet_8 & \bullet_{13} & & & & \bullet_{29} \\
 & \bullet & \bullet & \bullet & \bullet & \bullet_{12} & \bullet_{18} & & & \bullet_{28} \\
 & & & & & \bullet & \bullet_{17} & \bullet_{20} & \bullet_{23} & \bullet_{27} \\
 & & & & & & \bullet & \bullet_{19} & \bullet_{22} & \bullet_{26} \\
 & & & & & & \bullet & \bullet & \bullet_{21} & \bullet_{25} \\
 & & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet_{24}
\end{pmatrix}.$$

The packed skyline format consists of a real array and an integer array. The upper triangle is stored in a one-dimensional real array indexed as shown above. The auxiliary integer 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 $p+1$ with the $(p+1)$st element containing the length of the packed real array plus one. $D_\lambda\rho_a(x,\lambda)$ is provided as a separate column vector. Internally HOMPACK90 may augment $D_x\rho_a$ with a row and column, also stored in skyline format (depending on the problem), but these details are transparent to the user.

In most applications where $D_x\rho_a$ is symmetric, it is also a low rank modification of a positive definite matrix. These properties, exactly as by Watson, Billups, and Morgan [1987], are exploited by matrix splitting and using a Gill-Murray factorization [Gill and Murray 1974] as a preconditioning matrix. Recall that for the ODE-based algorithm, all that is required is the kernel of the matrix $D\rho_a(x,\lambda)$. Let $(\bar{x},\bar{\lambda})$ be a point on the zero curve γ, and $\bar{y}$ the unit tangent vector to γ at $(\bar{x},\bar{\lambda})$ in the direction of increasing arc length s. Let $|\bar{y}_k| = \max_i |\bar{y}_i|$. Then the matrix

$$A = \begin{bmatrix} D\rho_a(x,\lambda) \\ e_k^t \end{bmatrix} \tag{11}$$

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

$$Ay = \bar{y}_k e_{p+1} = b. \tag{12}$$

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

$$By = (Q^{-1}A)y = Q^{-1}b = g. \tag{13}$$

The use of such a matrix is known as (left) 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 later would converge more rapidly when applied to (13) than when applied to (12). In practice B and g are never explicitly formed. Right preconditioning has the form

$$Bz = (AQ^{-1})z = A(Q^{-1}z) = Ay = b, \tag{14}$$

where the final step is to compute $y = Q^{-1}z$. HOMPACK90 uses right preconditioning because that is what the GMRES code ([Saad 1996], [Walker 1988]), on which the HOMPACK90 code is based, uses; HOMPACK used left preconditioning in conjunction with Craig's conjugate gradient method to solve (12) [Watson, Billups, and Morgan 1987].

The coefficient matrix A in the linear system of equations (12), whose solution y yields the kernel of $D\rho_a(\bar{x}, \bar{\lambda})$, has a very special structure which can be exploited if (12) is attacked indirectly as follows. Note that the leading $p \times p$ submatrix of A is $D_x\rho_a$, which is symmetric and sparse, but possibly indefinite. Write

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

where

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

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

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. Other choices for the last row of A have been thoroughly studied [Irani et al. 1991]. Thus for almost all vectors c the first p columns of M are independent, and similarly almost all $(p+1)$-vectors are independent of the first p 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

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

which requires the solution of two linear systems with the sparse, symmetric, invertible matrix M. It is the systems $Mz = u$ and $Mz = b$ which are solved by a new preconditioned adaptive GMRES(k) algorithm, described in a later section.

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 [1974]. 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 if $D_x\rho_a(x, \lambda)$ 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, which is the point of this entire scheme.

5.2 Sparse Row Storage Format

The sparse row storage format is intended for arbitrary sparsity patterns, and stores only the structural nonzeros of $DF(x)$ or $D\rho_a$ (i.e., elements that happen to be zero by accident should be treated as nonzero). The data structure for A in (11) consists of three arrays: a real one-dimensional array q holding only the structural nonzero elements of A stored by row in row order (the elements within a row need not be in column order), an integer array r with r_i giving the location within q of the beginning of the elements for the ith row, and an integer array c with c_j giving the column index of the jth element of q. By convention, the diagonal elements of A are always structurally nonzero (for invertible A there always exists a permutation matrix P such that PA has no zeros on the diagonal), and r_{p+2} is the length of q plus one. For example, the data structure for

$$A = \begin{bmatrix} 4 & 0 & 0 & 2 & 0 \\ 7 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 6 \\ 2 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 5 \end{bmatrix}$$

is (here $p = 4$)

$$\begin{aligned} q &= (4, 2, 7, 1, 3, 0, 6, 1, 2, 1, 5), \\ r &= (1, 3, 5, 8, 10, 12), \\ c &= (1, 4, 1, 2, 3, 4, 5, 4, 1, 2, 5). \end{aligned}$$

The zero in location (3,4) is stored to illustrate a structural nonzero, q and c have the same dimension, and $r_{p+2} = \dim q + 1 = 12$.

Since $D_x\rho_a(x, \lambda)$ is unstructured, there is no advantage to splitting as in (15), so that is not done. Also differing from the previous section, the last row (c^t, d) of A in (11) is not just e_k^t, but uses $d \neq 0$ as outlined by Irani et al. [1991]. The kernel of $D\rho_a$ is computed by solving the right preconditioned system (14), where

$$A = \begin{bmatrix} D_x\rho_a(x,\lambda) & D_\lambda\rho_a(x,\lambda) \\ c^t & d \end{bmatrix}, \tag{17}$$

(c^t, d) is chosen to guarantee that A is invertible [Irani et al. 1991], Q in (14) is an ILU factorization of A, and the iterative linear system solver is a new adaptive GMRES(k) algorithm described below.

Since ILU preconditioning was not used in HOMPACK, a precise description is given here. Let $N = p+1$ and Z be a set of indices contained in $\{(i,j) \mid 1 \le i, j \le N, i \neq j\}$, typically where A is known to be zero. The incomplete LU factorization is given by $Q = LU$, where L and U are lower triangular and unit upper triangular matrices, respectively, that satisfy

$$\begin{cases} L_{ij} = U_{ij} = 0, & (i,j) \in Z, \\ Q_{ij} = A_{ij}, & (i,j) \notin Z,\ i \neq j, \\ Q_{ii} = A_{ii}, & \text{whenever possible.} \end{cases}$$

The incomplete LU factorization algorithm is:

for $i = 1$ **step** 1 **until** N **do**
  for $j = 1$ **step** 1 **until** N **do**
    if $((i,j) \notin Z)$ **then**
      begin

$$s_{ij} = A_{ij} - \sum_{t=1}^{\min\{i,j\}-1} L_{it}U_{tj};$$

        if $(i \geq j)$ **then** $L_{ij} = s_{ij}$ **else** $U_{ij} = s_{ij}/L_{ii}$;
      end

It can happen that L_{ii} is zero in this algorithm. In this case L_{ii} is set to a small positive number, so that $Q_{ii} \neq A_{ii}$.

5.3 Adaptive GMRES(k) Algorithm

Among all the Krylov subspace methods for solving a linear system $Ax = b$ with a nonsymmetric invertible coefficient matrix A, the stabilized bi-conjugate gradient algorithm (BiCGSTAB) [van der Vorst 1992], the generalized minimal residual algorithm (GMRES) [Saad and Schultz 1986], and the quasi-minimal residual algorithm (QMR) [Freund and Nachtigal 1991] are considered the most robust [McQuain et al. 1994]. Similar to the classical conjugate gradient method, GMRES produces approximate solutions x_k which are characterized by a minimization property over the Krylov subspaces span$\{r_0, Ar_0, A^2r_0, \ldots, A^{(k-1)}r_0\}$, where $r_0 = b - Ax_0$ and k is the iteration number. However, unlike the conjugate gradient algorithm, the work and memory required by GMRES grow proportionately to the iteration number. In practice, the restarted version GMRES(k) is used, where the algorithm is restarted every k iterations until the residual norm is small enough. The restarted version may stagnate and never reach the solution.

QMR reduces the computational effort by employing a short-term recursion for building the Lanczos basis. An implementation of QMR based on the look-ahead Lanczos process avoids breakdowns associated with Lanczos-type algorithms [Freund and Nachtigal 1990]. However, a QMR iterate does not minimize the residual over the current Krylov subspace, which results in more iterations than unrestarted GMRES and perhaps GMRES(k) as well (that may or may not take more time than GMRES(k)). The QMR algorithm may also behave erratically.

The essence of the adaptive GMRES strategy proposed here is to adapt the parameter k to the problem, similar in spirit to how a variable order ODE algorithm tunes the order k. With Fortran 90, which provides pointers and dynamic memory management, dealing with the variable storage requirements implied by varying k is not too difficult. k can be both increased and decreased—an increase-only strategy is described below.

Though GMRES(k) cannot break down, it can stagnate. A test of stagnation developed by Walker [1996] detects an insufficient residual norm reduction in the restart number (k) of steps. Precisely, GMRES(k) is declared to have stagnated and the iteration is aborted if at the rate of progress over the last restart cycle of steps, the residual norm tolerance cannot be met in some large multiple (bgv)

of the remaining number of steps allowed ($itmax$ is a bound on the number of steps permitted). Slow progress (near-stagnation) of GMRES(k), which signals an increase in the restart value k, may be detected with a similar test. The near-stagnation test uses a different, smaller multiple (smv) of the remaining allowed number of steps. If near-stagnation occurs, the restart value k is incremented by some value m and the *same* restart cycle continues. Restarting would mean repeating the nonproductive iterations that previously resulted in stagnation, at least in the case of complete stagnation (no residual reduction at all). Such incrementing is used whenever needed if the restart value k is less than some maximum value $kmax$. When the maximum value for k is reached, adaptive GMRES(k) proceeds as GMRES($kmax$). The values of the parameters smv, bgv, and m are established experimentally and can remain unchanged for most problems.

Let $\mathbf{u}$ be the machine unit roundoff, e_j be the jth standard basis vector, and all norms the 2-norm. The rounding error of a sparse matrix-vector multiplication depends on only the nonzero entries in each row of the sparse matrix, so the error tolerance $xtol$ is proportional to the average number of nonzeros per row $avnz$ = (number of nonzeros in $A)/p$. Since GMRES convergence is normally measured by reduction in the initial residual norm, the convergence tolerance is $tol = \max\{||r_0||, ||b||\}xtol$. Pseudocode for an adaptive version of GMRES(k) with orthogonalization via Householder reflections (see Walker [1988]), called AGMRES(k) by Sosonkina, Watson, and Kapania [1998], follows.

choose $x, itmax, kmax, m$;
$r := b - Ax$; $itno := 0$; $cnmax := 1/(50\mathbf{u})$;
$xtol := \max\{100.0, 1.01 avnz\}\mathbf{u}$; $tol := \max\{||r||, ||b||\}xtol$;
while $||r|| > tol$ **do**
begin
  $r^{old} := r$;
  determine the Householder reflection P_1 such that $P_1 r = \pm||r||e_1$;
  $k_1 = 1$; $k_2 = k$;
L1: **for** $j := k_1$ **step** 1 **until** k_2 **do**
  begin
    $itno := itno + 1$;
    $v := P_j \cdots P_1 A P_1 \cdots P_j e_j$;
    determine P_{j+1} such that $P_{j+1}v$ has zero components
      after the $(j+1)$st;
    update $||r||$ as described by Saad [1996];
    compute incremental condition number estimate ICN;
    if $ICN > cnmax$ **then** abort;
    if $||r|| \leqq tol$ **then goto** L2
  end
  $test := k_2 \times \log\left[tol/||r||\right] / \log\left[||r||/\left((1.0 + 10\mathbf{u})\,||r^{old}||\right)\right]$;

if $k_2 \leqq kmax - m$ **and** $test \geqq smv \times (itmax - itno)$ **then**
$k_1 := k_2 + 1; \qquad k_2 := k_2 + m;$
goto L1
end if
L2: $e_1 := (1, 0, \cdots, 0)^T; \qquad k := k_2;$
solve $\min_y \| \, \|r\| e_1 - \bar{H}_j y \|$ for y_j where $\bar{H}_j$ is described by Saad [1996];
$q := \begin{pmatrix} y_j \\ 0 \end{pmatrix};$
$x := x + P_1 \cdots P_j q; \qquad r := b - Ax;$
if $\|r\| \leqq tol$ **then** exit;
if $\|r^{old}\| < \|r\|$ **then**
if $\|r\| < tol^{2/3}$ **then**
exit
else
abort
end if
end if
$test := k \times \log\left[tol/\|r\|\right] \, / \, \log\left[\|r\| / \left((1.0 + 10\mathbf{u})\, \|r^{old}\|\right)\right];$
if $test \geqq bgv \times (itmax - itno)$ **then**
abort
end if
end

A possible symptom of AGMRES(k) going astray is an increase in the residual norm between restarts (the residual norm is computed by direct evaluation at each restart). If the residual norm on the previous restart is actually smaller than the current residual norm, then AGMRES(k) terminates. The solution is considered acceptable if $\|r\| < tol^{2/3}$, although this loss of accuracy in the tangent vector or corrector step may cause HOMPACK90 to fail. Usually HOMPACK90 can deal gracefully with a loss of accuracy in the linear system solutions. If $\|r\| \geqq tol^{2/3}$, AGMRES(k) is deemed to have failed. In this latter case, the continuation of GMRES(k) would typically result in reaching a limit on the number of iterations allowed and a possible repetition of $\|r^{old}\| < \|r\|$ in later restarts. AGMRES(k) may exceed an iteration limit when it is affected by roundoff errors in the case of a (nearly) singular GMRES least-squares problem. As in the work of Brown and Walker [1997], the condition number of the GMRES least-squares problem is monitored by an incremental condition estimate ICN [Bischof and Tang 1991], and AGMRES(k) aborts when the estimated condition number is greater than $1/(50\mathbf{u})$.

The use of Householder reflections rather than the more standard modified Gram-Schmidt process is a subtle consequence of the well established need for high accuracy in the context of homotopy zero curve tracking. A detailed discussion of this issue and results of numerical experiments are in the work of Sosonkina, Watson, and Kapania [1998], where the algorithm AGMRES(k) is also tested on realistic large scale problems in analog circuit simulation (using the sparse row storage format and ILU preconditioning) and structural mechanics (using the packed skyline storage format and Gill-Murray preconditioning).

5.4 Sparse ODE-based Curve Tracking

For sparse Jacobian matrices, the logic of tracking the zero curve γ is exactly the same as that for the dense Jacobian matrix case. The only difference is in the linear algebra for the kernel calculation and the concomitant data structures, which are substantially more complicated for the sparse Jacobian matrix case. One significant innovation in HOMPACK90 is that storage space for temporary work arrays and for the sparse Jacobian matrix data structure itself is dynamically allocated. The sparse Jacobian matrix is made available globally via the `MODULE HOMOTOPY` rather than through subroutine call lists, which is a major change from the organization in HOMPACK. The low level details of this memory management are best left to the code, where they are thoroughly documented.

6. NORMAL FLOW ALGORITHM (SPARSE JACOBIAN MATRIX)

The logic of the predictor, corrector, step size estimation, and end game phases of this algorithm is identical to that of the normal flow algorithm for dense Jacobian matrices. 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 (9), which also involves the calculation of the kernel of $D\rho_a(x,\lambda)$. (9) is solved using the same sparse matrix data structures, matrix splitting (if appropriate), preconditioning matrices, and adaptive GMRES(k) iterative algorithm used for the sparse ordinary differential equation based algorithm (equations 11–17). For efficiency, the kernel and Newton step ΔZ are calculated together by solving

$$\begin{bmatrix} D_x\rho_a(x,\lambda) & D_\lambda\rho_a(x,\lambda) \\ c^t & d \end{bmatrix} [v\ w] = \begin{bmatrix} 0 & -\rho_a(x,\lambda) \\ \bar{y}_k & 0 \end{bmatrix} \tag{18}$$

at points (x,λ) near $(\bar{x},\bar{\lambda})$ on γ, using the unit tangent vector $\bar{y}$ to γ at $(\bar{x},\bar{\lambda})$ in the direction of increasing arc length s. The Newton step ΔZ is recovered from v and w in the usual way [Watson, Billups, and Morgan 1987].

7. AUGMENTED (SPARSE) JACOBIAN MATRIX ALGORITHM

The augmented Jacobian matrix algorithm for sparse Jacobian matrices differs from that for dense Jacobian matrices in three respects: (1) like the sparse ODE based and normal flow algorithms, the low level numerical linear algebra is changed to take advantage of the sparsity and structure of the Jacobian matrices; (2) quasi-Newton iterations are abandoned in favor of pure Newton iterations; (3) Rheinboldt's step size control [Rheinboldt and Burkardt 1983] 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 that in the algorithm for dense Jacobian matrices.

The kernel of $D\rho_a$ at the point $P^{(2)}$, needed for the unit tangent vector $T^{(2)}$ to γ at $P^{(2)}$, which in turn is required for the prediction (7), is computed from

$$\begin{bmatrix} D\rho_a(P^{(2)}) \\ T^{(1)\,t} \end{bmatrix} z = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{19}$$

Then $T^{(2)} = z/\|z\|$; note that $T^{(2)}$ automatically points in the direction of increasing arc length s because $T^{(1)\,t} z = 1 > 0$. Instead of the quasi-Newton approximation

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

used in (10), the corrector steps $\Delta Z^{(k)}$ are obtained from

$$\begin{bmatrix} D\rho_a(Z^{(k)}) \\ T^{(2)\,t} \end{bmatrix} \Delta Z^{(k)} = \begin{pmatrix} -\rho_a(Z^{(k)}) \\ 0 \end{pmatrix}. \tag{20}$$

Note that all the iterates $Z^{(k)}$ lie in the hyperplane through the initial prediction $Z^{(0)}$ and perpendicular to $T^{(2)}$—this is essentially the well known Riks-Wempner method used in mechanics.

The sparse linear algebra for the three curve tracking algorithms—ODE based, normal flow, augmented Jacobian matrix—is slightly different. The ODE based algorithm only needs the kernel of $D\rho_a(x, \lambda)$, and thus the last row $(c^t \quad d)$ in (11) or (17) can be (almost) anything. The normal flow algorithm needs both kernels and Newton corrections, but again this last row $(c^t \quad d)$ in (18) can be (almost) anything. The augmented Jacobian matrix algorithm also needs both kernels and Newton corrections, but the last row in (19) and (20) is (by definition) a tangent vector. HOMPACK had three separate subroutines: PCGDS, PCGNS, PCGQS—one for each curve tracking method and corresponding choice of $(c^t \quad d)$. HOMPACK90 is organized rather differently—essentially the same sparse linear system is solved in all three cases, with the variability coming from the storage format and preconditioner used. For example, (20) is actually solved by solving (18), and then recovering the desired Newton step ΔZ via

$$\Delta Z = w - \frac{w \cdot T^{(2)}}{v \cdot T^{(2)}} v. \tag{21}$$

This not only results in fewer subroutines in HOMPACK90, but is also more efficient for the iterative linear system solver since the row $(c^t \quad d)$ actually used has at most two nonzeros, compared to the generally full row $T^{(2)\,t}$.

The rationale for use of Newton iterations rather than quasi-Newton iterations is similar to that for HOMPACK, namely, there is still no good (comparable to Broyden or BFGS) sparse quasi-Newton update formula. The comments by Watson, Billups, and Morgan [1987] on the efficacy of deferred updating and other possibilities still apply. 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 involves implementing a more sophisticated control over the ideal starting error, described in detail by Watson, Billups, and Morgan [1987]. Except for the sparse numerical linear algebra, all the curve tracking logic of the augmented sparse Jacobian matrix algorithm in HOMPACK90 is identical to that of the corresponding algorithm in HOMPACK. A final observation is that this algorithm is more robust than the quasi-Newton version for dense matrices, because occasionally large steps produce very bad quasi-Newton approximations $A^{(k)}$ to $D\rho_a\big(Z^{(k)}\big)$, leading to erratic behavior.

8. POLYNOMIAL SYSTEMS

This section describes the POLSYS1H driver for finding all complex solutions to polynomial systems with real coefficients. A system of n polynomial equations in n unknowns,

$$F_i(x) = \sum_{j=1}^{n_i} \left[p_{ij} \prod_{k=1}^{n} {x_k}^{d_{ijk}} \right] = 0, \qquad i = 1, \ldots, n, \tag{22}$$

may have many solutions. Precisely, let

$$d_i = \max_{\substack{1 \le j \le n_i \\ 1 \le k \le n}} d_{ijk}$$

be the *degree* of the ith polynomial function $F_i(x)$, and define the *total degree* of the system $F(x)$ as

$$d = d_1 \cdots d_n .$$

Assume that (22) has finitely many solutions (this is true generically, i.e., for almost all coefficients p_{ij}). Then (22) has exactly d solutions, counting multiplicities, in complex projective space $\mathbf{P}^n$. d is called the *1-homogeneous Bezout number* of the system $F(x)$. Watson, Billups, and Morgan [1987] contains a concise description of complex projective space $\mathbf{P}^n$, how (22) is interpreted over $\mathbf{P}^n$, and references to the algebraic geometry literature.

It is possible to define a homotopy map so that all geometrically isolated solutions of (22) have at least one associated homotopy path. Generally, (22) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as λ approaches 1. However, (22) can be transformed into a new system (referred to as the *projective transformation of* $F(x) = 0$ by Watson, Billups, and Morgan [1987]) which, under reasonable hypotheses, can be proven to have no solutions at infinity and thus bounded homotopy paths. Because real problems are often poorly scaled, POLSYS1H includes a general scaling algorithm (subroutine SCLGNP) that scales both the coefficients and the variables. POLSYS1H 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 (subroutine FFUNP). The user need not code any subroutine or Fortran 90 module to be called by POLSYS1H.

Although the POLSYS1H homotopy map is defined in complex or complex projective space, the code POLSYS1H does not use complex computer arithmetic. Since the homotopy map is complex analytic, the homotopy parameter λ is monotonically increasing as a function of arc length [Garcia and Zangwill 1979]. 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 real solutions, rather than all complex projective solutions. However, the sort of generic homotopy algorithm considered here must find all solutions (with respect to the closed algebraic space over which $F(x)$ is being considered) and cannot be limited without, in essence, changing it into a heuristic. The theory of polynomial systems, algebraic geometry, is a deep and rich subject, far beyond the scope of this discussion.

The parameters n, n_i, p_{ij}, d_{ijk} defined in (22) constitute the *coefficient tableau.* The subroutine SCLGNP uses this tableau to generate scale factors for the coefficients p_{ij} and the variables x_k, and the subroutine FFUNP uses it to compute system and Jacobian matrix values for POLSYS1H. This has the advantage of being very general, but the disadvantage of usually being less efficient than a hand crafted 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, whose 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. The scaling and projective transformation algorithms in POLSYS1H are identical to those of POLSYS in HOMPACK, which are described in detail by Watson, Billups, and Morgan [1987]. Although the mathematical algorithms for POLSYS1H are the same as for POLSYS, the code POLSYS1H is significantly different from the code POLSYS in HOMPACK, mainly because POLSYS1H takes full advantage of Fortran 90 features such as automatic arrays, array sections, and array intrinsics.

POLSYS1H is constructed so that the user can 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 POLSYS1H is the coefficient tableau and a few parameters for the path tracking algorithm (normal flow) used by POLSYS1H. POLSYS1H has a parameter NUMRR (number of reruns) so that 1000*NUMRR steps will be taken before a path is abandoned. Experience on industrial problems argues for the use of both the projective transformation and scaling on most problems.

The 1-homogeneous code POLSYS1H is perfectly adequate for small problems, being routinely used for $d < 10^4$. In practice, only a handful of these d solutions are real and/or physically meaningful, and as n increases, computing the total degree number of solutions quickly becomes impractical (e.g., 20 cubic equations would have $d = 3^{20} \approx 3.5 \cdot 10^9$ solutions). While the most powerful and practical approaches still compute many nonphysical complex solutions, recent theoretical advances such as m-homogeneity, BKK theory, and product decompositions have enormously reduced the number of extraneous solutions that must be computed. These theories are complicated, and their implementation in user-friendly, robust, portable software with such capabilities as automatic differentiation and sophisticated error control is a major undertaking. The essence of all these advanced theories is to "factor out" large numbers of nonphysical solutions by exploiting the structure of F. For example, the m-homogeneous approach [Morgan and Sommese 1987], [Morgan et al. 1989] partitions the variables x_k into m groups, where the ith group has k_i variables, $k_1 + \cdots + k_m = n$. $F(x) = 0$ is then converted into a system $F'(x) = 0$ that is homogeneous with respect to the variables in *each* group, and viewed as a polynomial system over $\mathbf{P}^{k_1} \times \cdots \times \mathbf{P}^{k_m}$. The number of solutions over $\mathbf{P}^{k_1} \times \cdots \times \mathbf{P}^{k_m}$, called the *m-homogeneous Bezout number*, may be orders of magnitude less than the total degree d (the 1-homogeneous Bezout number). Many applications have a natural m-homogeneous structure, many others do not.

Much more complicated structure is exploited by the combinatorial BKK theory [Verschelde and Cools 1993], [Verschelde et al. 1994] and general product decompositions [Morgan, Sommese, and Wampler 1995]. The BKK combinatorial computation flounders on larger problems, and the product decomposition of Morgan, Sommese, and Wampler [1995] is more of an approach to exploiting structure than an algorithm, and probably can not be implemented in its full generality into computer code. Nevertheless, there are schemes between m-homogeneous and the general product decomposition that are probably implementable, and subsume all m-homogeneous formulations. A polynomial system code implementing some restricted form of product decomposition is planned for a future release of HOMPACK90.

9. ORGANIZATIONAL DETAILS

Physically, the HOMPACK90 package consists of four modules (HOMOTOPY, HOMPACK90, HOMPACK90_GLOBAL, REAL_PRECISION) and a large collection of external subroutines. An object oriented description of these components is given in later subsections. There are other useful and logical ways to view the organization of the HOMPACK90 package; some of these logical viewpoints are described next.

HOMPACK90 can be logically viewed as 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 or their reverse call versions (described below) directly to have complete control of the algorithm, or for some other task such as tracking an arbitrary parametrized curve over an arbitrary parameter range. The third subroutine level handles high level numerical linear algebra such as QR factorization, and includes some LAPACK and BLAS routines [Anderson et al. 1995]. Low level linear algebra (BLAS) is mostly done directly with Fortran 90 syntax or array intrinsics. All the high level linear algebra and sparse matrix data structure handling are concentrated in these third level routines, so a user could incorporate his own data structures by writing his own versions of these third level routines. Also, by utilizing Fortran 90 array intrinsics and by concentrating the higher level linear algebra in subroutines, HOMPACK90 can be easily adapted to a vector or parallel computer.

The organization of HOMPACK90 by algorithm/problem type is shown in Table I, which lists the driver name for each algorithm and problem type.

Table I. Taxonomy of Homotopy Subroutines

$x = f(x)$		$F(x) = 0$		$\rho(a, \lambda, x) = 0$		Algorithm
Dense	Sparse	Dense	Sparse	Dense	Sparse	
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

The naming convention is

$$FIXP \left\{ \begin{matrix} D \\ N \\ Q \end{matrix} \right\} \left\{ \begin{matrix} F \\ S \end{matrix} \right\},$$

where $D \approx$ ordinary differential equation algorithm, $N \approx$ normal flow algorithm, $Q \approx$ quasi-Newton augmented Jacobian matrix algorithm, $F \approx$ dense Jacobian matrix, and $S \approx$ sparse Jacobian matrix. The natural grouping of the HOMPACK90 routines into the three subroutine levels described above, and a list of the BLAS and LAPACK routines used, are provided in the README file with the source code.

The user written subroutines, of which exactly two must be supplied depending on the driver chosen, are F, FJAC, FJACS, RHO, RHOA, RHOJAC, and RHOJS, whose interfaces are specified in the module HOMOTOPY. The module REAL_PRECISION specifies the real numeric model with

SELECTED_REAL_KIND(13),

which will result in 64-bit real arithmetic on a Cray, DEC VAX, and IEEE 754 Standard compliant hardware.

9.1 Driver for Polynomial Systems

The special purpose polynomial system solver POLSYS1H, for which the underlying mathematical theory was described earlier, is essentially a high level interface to the driver FIXPNF in the MODULE HOMPACK90. POLSYS1H requires special versions of RHO and RHOJAC (subroutines normally provided by the user). These special versions are included in the template file distributed with HOMPACK90, so for a polynomial system the user need only call POLSYS1H, and define the problem directly to POLSYS1H by specifying the polynomial coefficients. POLSYS1H scales and computes partial derivatives on its own. Thus the user interface to POLSYS1H and HOMPACK90 is clean and simple. The only caveat is that FFUNP, which builds the polynomial system F and its Jacobian matrix DF from the coefficient tableau, cannot recognize patterns of repeated expressions in the polynomial system, and so may be less efficient than a hand crafted version. If great efficiency is required, the user can modify the default FFUNP; the sections in the code which must be changed are clearly marked. The subroutine level grouping is:

[POLSYS1H] [FIXPNF, ROOTNF, STEPNF, TANGNF] [DGEQPF, DGEQRF, DORMQR, DIVP, FFUNP, GFUNP, HFUNP, HFUN1P, INITP, MULP, OTPUTP, POWP, RHO, RHOJAC, ROOT, SCLGNP, STRPTP]

9.2 Modules HOMOTOPY, HOMPACK90, HOMPACK90_GLOBAL, and REAL_PRECISION

The user written subroutines—two of F, FJAC, FJACS, RHO, RHOA, RHOJAC, RHOJS—are external subroutines, whose interfaces are provided to HOMPACK90 via the Fortran 90 module HOMOTOPY. Since these interfaces are fixed, none of HOMPACK90 needs to be recompiled when the user subroutines F, FJAC, etc. or modules used therein change. The user written subroutines can use modules to pass global information between the main program calling HOMPACK90 and these user written subroutines called by HOMPACK90, without using COMMON or requiring extra work arrays PAR, IPAR in the argument lists of HOMPACK90 subroutines just to filter information down to F, FJAC, etc. (All the HOMPACK drivers provided arrays PAR and IPAR of arbitrary size which were passed all the way down to F, FJAC, etc. The old polynomial system code POLSYS made extensive use of these arrays.) Modules provide an effective mechanism for data hiding, and huge collections of analysis codes can be hidden as PRIVATE internal subroutines within modules used by the subroutines F, FJAC, etc. A template file for F, FJAC, FJACS, RHO, RHOA, RHOJAC, and RHOJS is provided with the HOMPACK90 distribution, intended as a starting point for the user's subroutines defining the problem to be solved. Note that skeletons for all of the subroutines among F, FJAC, FJACS, RHO, RHOA, RHOJAC, RHOJS not actually used must be provided, since they are referenced by HOMPACK90. The three sample main programs provided for verifying the installation also offer examples of F, FJAC, etc.

The module HOMPACK90_GLOBAL, used by the module HOMOTOPY, provides the repository for the sparse Jacobian matrix data structures, which are allocated and deallocated by the high level HOMPACK90 sparse routines. FJACS and RHOJS, as written by the user, simply assume that the sparse matrix data structures exist and fill them with data when called. Only two pieces of information about the data structures are given to the drivers FIXPDS, FIXPNS, or FIXPQS: the type of sparse matrix storage (integer MODE) and an upper bound LENQR on the number of nonzeros in the Jacobian matrices. There are other equally elegant organizational schemes, but this one requires no dynamic memory management by the user at all.

The new numeric model and environmental inquiry functions of Fortran 90 are fully utilized in HOMPACK90. All real variables and constants in HOMPACK90 have a KIND declared in the module REAL_PRECISION (the default is 64-bit arithmetic), which must be modified for a different numeric model. Double precision BLAS and LAPACK routines are called by HOMPACK90.

The module HOMOTOPY uses the modules REAL_PRECISION and HOMPACK90_GLOBAL, and contains interfaces for the external subroutines F, FJAC, FJACS, RHO, RHOA, RHOJAC, and RHOJS. These user written subroutines must conform to their interfaces; note especially the use of assumed shape arrays. The module HOMOTOPY is used throughout HOMPACK90 to provide global information and define interfaces with the user's subroutines F, FJAC, etc. The reason for putting only the *interfaces* to F, FJAC, etc. in HOMOTOPY rather than the routines themselves is that in the latter case, much of HOMPACK90 would have had to be recompiled every time the problem definition code in the module HOMOTOPY changed.

All of the drivers and their documentation (as comment statements) are encapsulated in a single MODULE HOMPACK90. The user's calling program would then simply contain a statement like

```
USE HOMPACK90, ONLY : FIXPNF
```

The hierarchy of the modules, in compilation order, is REAL_PRECISION, HOMPACK90_GLOBAL, HOMOTOPY, HOMPACK90. Physically, HOMPACK90 consists of four modules and external subroutines; everything can be compiled once, the external subroutines kept in an object library, and the modules kept in a module library.

9.3 Usage Details

As mentioned above, the user only needs to deal with the MODULE HOMPACK90, typically by selecting one of the drivers from that module with a USE statement. Descriptions of the argument lists, mathematical assumptions concerning appropriate use, calling trees, and specifications for the user written subroutines are given as comments in the drivers in the MODULE HOMPACK90. Three sample main programs—MAINF, MAINS, MAINP—illustrate typical usage of the drivers for dense, sparse, and polynomial systems, respectively. The special versions of RHO and RHOJAC required by POLSYS1H should be extracted verbatim from either the comments in MODULE HOMOTOPY or the template file. Installation and testing information is contained in a README file.

9.4 Reverse Call Subroutines

Many scientific programmers prefer the reverse call paradigm, whereby a subroutine returns to the calling program whenever the subroutine needs certain information (e.g., a function value) or a certain operation performed (e.g., a matrix-vector multiply). However, many users of mathematical software are unfamiliar with reverse call, it is not the consensus preference of computer scientists, and most of its advantages are obviated by the features of Fortran 90. Thus HOMPACK90 retains the forward calling (functional programming) style, but for unusual circumstances and some users' preference, two reverse call subroutines (STEPNX, ROOTNX) are provided for "expert" users.

The ODE-based (D), normal flow (N), and quasi-Newton augmented Jacobian matrix (Q) stepping routines provide complete algorithmic "coverage," but the D and Q routines are rarely used in practice, because the N routines are usually (but not always!) more efficient. Whether the Jacobian matrix is sparse or dense is irrelevant to a reverse call stepping routine, if the expert user is handling all matrix storage details and providing all linear algebra operations. Hence only one expert reverse call stepping routine, STEPNX, is needed. STEPNX is a reverse call stepping subroutine, designed to be used in lieu of any of the six standard (forward call) stepping routines STEPDF, STEPNF, STEPQF, STEPDS, STEPNS, or STEPQS (refer to the subroutine grouping in the README file and the comments in the subroutines themselves). STEPNX returns to the caller for all linear algebra, all function and derivative values, and can deal gracefully with situations such as the function being undefined at the requested steplength.

ROOTNX provides an expert reverse call end game routine. ROOTNX has the same protocol as STEPNX, and generalizes the routines ROOTNF, ROOTNS, ROOTQF by finding a point on the zero curve where $g(\lambda, x) = 0$, as opposed to just the point where $\lambda = 1$. Thus ROOTNX can find turning points, bifurcation points, and other "special" points along the zero curve. The combination of STEPNX and ROOTNX provide considerable flexibility for an expert user. The root finding algorithm in ROOTNX is the end game described by Sosonkina, Watson, and Stewart [1996], generalized to the situation $g(\lambda, x) = 0$.

10. TESTING

Watson, Billups, and Morgan [1987] give results for HOMPACK applied to some difficult concocted test problems, some mention of the range of problems in science and engineering to which HOMPACK has been applied, and caveats about the use of globally convergent homotopy methods and HOMPACK. All of those comments apply equally well to HOMPACK90, and HOMPACK90 shows only minor differences from HOMPACK (for the computed arc length and number of Jacobian matrix evaluations) on all those test problems; see Table II below for timing comparisons between HOMPACK and HOMPACK90. The exponential test problem of Watson, Billups, and Morgan [1987] deserves special mention, because numerous papers have reported a lower number of function evaluations along γ than HOMPACK, but with an incorrect arc length. Those "superior" results are obtained by missing a loop in γ, landing by accident on another loop in γ, and ultimately (again by accident) tracking a piece of γ to a solution. A recent survey of engineering applications is by Watson [1990], with some recent significant applications being the work of Rakowska et al. [1991], Melville et al. [1993b], and Melville et al. [1993a]. POLSYS1H and the new end game were extensively tested by Sosonkina, Watson, and Stewart [1996], and results for the adaptive GMRES(k) algorithm on three classes of problems were reported by Sosonkina, Watson, and Kapania [1998]. The aforementioned references show the superior performance of HOMPACK90 compared to HOMPACK, and the circuit problem described below could not have been solved at all with the sparse routines in HOMPACK.

An interesting test of the adaptive GMRES(k) algorithm is provided by the problem of computing the operating points of a bandgap voltage reference circuit with 28 transistors, fabricated with an in-house bipolar process at AT&T Bell Laboratories. A sophisticated transistor model results in $n = 198$ unknowns. Using the variable gain homotopy map described by Melville et al. [1993a] and searching

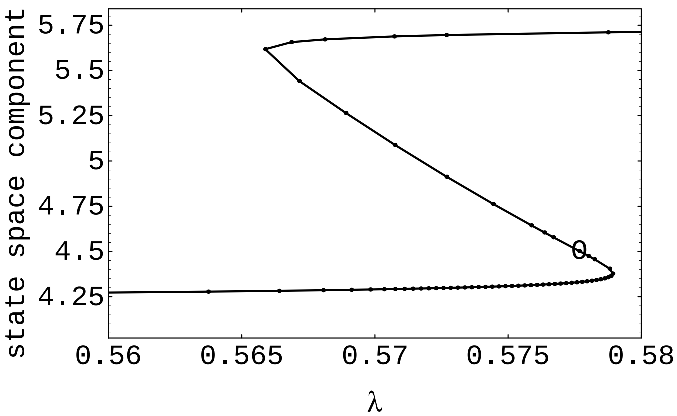

Fig. 2. Section of homotopy zero curve for bandgap voltage reference circuit.

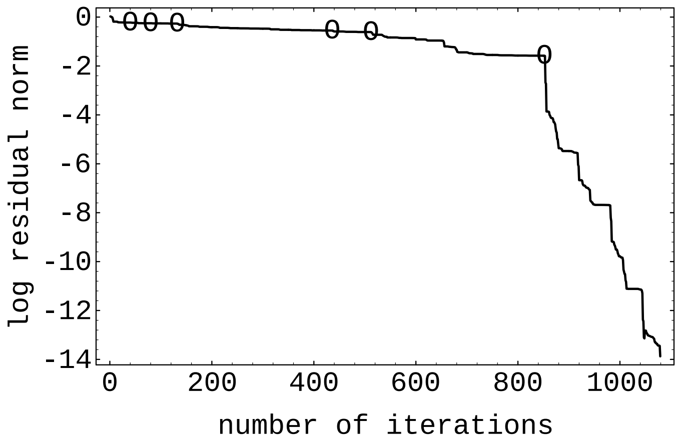

Fig. 3. Residual norm behavior for adaptive GMRES algorithm.

for multiple solutions with the program `Sframe` from Melville et al. [1993b] results in the homotopy zero curve shown in Figure 2. Note the very sharp turns in the zero curve γ, attributable to the behavior of transistors in the circuit. Reordering the unknowns so that the Jacobian matrix (900 nonzeros) is as nearly diagonally dominant as possible results in the ILU preconditioner in HOMPACK90 being reasonably effective.

The performance of the adaptive GMRES(k) algorithm on one of the Jacobian matrices (marked by the circle in Figure 2) along γ is shown in Figure 3, where a circle indicates an increase in the subspace dimension ($k = 8$ initially). The adaptive strategy is invoked for the first time rather early—to jump off the near-stagnation "plateau" after the 40th iteration. The subspace dimension is increased several more times until AGMRES(k) has enough vectors (32) to solve the system within the iteration limit. Note that a smaller iteration limit (ITMAX, set in subroutine PCGDS) than the HOMPACK90 default would have caused AGMRES(k) to increase the subspace dimension quicker, thus reducing the number of iterations, but at a considerably greater cost per iteration.

With current compiler technology, there can be a significant performance penalty for choosing Fortran 90 over FORTRAN 77. The data shown below in Table II is for the FORTRAN 77 version of subroutine FIXPNF from HOMPACK and the Fortran 90 version of subroutine FIXPNF from HOMPACK90, applied to the zero finding problem for Brown's function, using starting point $a = 0$, curve tracking tolerances $.5 \cdot 10^{-6}$, final accuracy tolerances $1.0 \cdot 10^{-10}$, and all HOMPACK(90) parameters at their default values. The compiler used was DEC Digital Fortran 2.1 with option -fpe3, and the CPU times (average of 10 runs) reported in Table II are in seconds on a DEC AlphaStation 200 4/233. There are algorithmic differences between the two versions of FIXPNF, but the *better* algorithms are in the Fortran 90 version. The numbers in parentheses after the CPU times are the numbers of Jacobian matrix evaluations—these differ because the end games for FIXPNF in HOMPACK and HOMPACK90 differ, with the new end game being more robust but occasionally more expensive. The numbers in brackets are seconds per Jacobian matrix evaluation, showing that a performance penalty for using Fortran 90 exists, but decreases with increasing n. FIXPNF does not use pointers, allocatable arrays are allocated and deallocated just once (by the driver FIXPNF), and automatic arrays are created only once, so the time difference is probably due to the overhead associated with assumed-shape arrays and with array section operations.

Table II. Timing Comparisons Between HOMPACK and HOMPACK90

Dimension n	50	100	150	200	250
FORTRAN 77 time	.350(41)	3.20(55)	10.1(54)	24.6(56)	56.4(66)
	[.00854]	[.05818]	[.18704]	[.43929]	[.85455]
Fortran 90 time	.543(45)	3.53(53)	12.6(64)	27.4(61)	50.9(59)
	[.01207]	[.06660]	[.19687]	[.44918]	[.86271]

The performance differences shown in Table II are typical for nonlinear equations whose solution cost is dominated by linear algebra and data movement. Problems dominated by the function evaluation cost, such as discretizations of boundary value problems with complicated coefficients, take essentially the same amount of time with HOMPACK90 as with HOMPACK. For sparse problems, the dynamic memory allocation required by the new adaptive GMRES algorithm is not possible in FORTRAN 77. Even for dense problems, despite an occasional performance penalty, there are compelling reasons to use Fortran 90: significantly shorter and more elegant argument lists, rigorous argument array size checking (a common source of user errors), and interfaces to user written subroutines that prevent argument type mismatches.

ACKNOWLEDGEMENT

The authors gratefully acknowledge suggestions from Nick Gould, Tim Hopkins, John Reid, and a referee for significantly improving this paper and the Fortran 90 code in HOMPACK90.

BIBLIOGRAPHY

ALEXANDER, J. C., LI, T.-Y, AND YORKE, J. A. 1983. Piecewise smooth homotopies. In *Homotopy Methods and Global Convergence*, B. C. Eaves, F. J. Gould, H.-O. Peitgen, and M. J. Todd, eds., Plenum, New York, 1–14.

ALLGOWER, E. L., CHIEN, C.-S., AND GEORG K. 1989. Large sparse continuation problems. *J. Comput. Appl. Math. 26*, 3–21.

ALLGOWER, E. L., GEORG, K., AND MIRANDA, R. 1992. Computing real solutions of polynomial systems. *SIAM J. Numer. Anal. 29*, 831–844.

ALLGOWER, E. L., GEORG, K. 1990. *Numerical Continuation Methods.* Springer-Verlag, Berlin.

ANDERSON, E., BAI, Z., BISHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1995. *LAPACK User's Guide.* SIAM, Philadelphia, PA, 2nd ed.

AUVIL, L. S., RIBBENS, C. J., AND WATSON, L. T. 1992. Problem specific environments for parallel computing. In *Proc. Scalable High Performance Computing Conference*, R. Voigt (ed.), IEEE Computer Soc. Press, Los Alamitos, CA, 149–152

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

BISCHOF, C. H., AND TANG, P. T. P. May, 1991. Robust incremental condition estimation. Tech. Rep. CS-91-133, LAPACK Working Note 33, Computer Sci. Dept., Univ. of Tennessee.

BJÖRCK, Å. 1967. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT 7*, 1–21.

BROWN, P. N. AND WALKER, H. F. 1997. GMRES on (nearly) singular systems. *SIAM J. Matrix Anal. Appl. 18*, 37–51.

BRUNOVSKÝ, P. AND MERAVÝ, P. 1984. Solving systems of polynomial equations by bounded and real homotopy. *Numer. Math. 43*, 397–418.

BUSINGER, P. AND GOLUB, G. H. 1965. Linear least squares solutions by Householder transformations. *Numer. Math. 7*, 269–276.

CHAN, T. F. AND SAAD, Y. 1985. Iterative methods for solving bordered systems with applications to continuation methods. *SIAM J. Sci. Stat. Comput. 6*, 438–451.

CHOW, S. N., MALLET-PARET, J., AND YORKE, J. A. 1978. Finding zeros of maps: Homotopy methods that are constructive with probability one. *Math. Comput. 32*, 887–899.

CHOW, S. N., MALLET-PARET, J., AND YORKE, J. A. 1979. A homotopy method for locating all zeros of a system of polynomials. In *Functional Differential Equations and Approximation of Fixed Points*, Lecture Notes in Math. #730, H. O. Peitgen and H. O. Walther, Eds., Springer-Verlag, 228–237.

CRAIG, E. J. 1954. Iteration procedures for simultaneous equations. Ph. D. thesis, MIT.

DENNIS, J. E. AND SCHNABEL, R. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, NJ.

DESA, C., IRANI, K. M., RIBBENS, C. J., WATSON, L. T., AND WALKER, H. F. 1992. Preconditioned iterative methods for homotopy curve tracking. *SIAM J. Sci. Stat. Comput. 13*, 30–46.

DONGARRA, J. J., MOLER, C. B., BUNCH, J. R., AND STEWART, G. W. 1979. *LINPACK Users' Guide.* SIAM, Philadelphia, PA.

FADEEV, D. K. AND FADEEVA, V. N. 1963. *Computational Methods of Linear Algebra.* Freeman, London.

FREUND, R. W. AND NACHTIGAL, N. M. Nov., 1990. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part II. Tech. Rep. 90.46, RIACS, NASA Ames Research Center.

FREUND, R. W. AND NACHTIGAL, N. M. 1991. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math. 60*, 315–339.

GARCIA, C. B. AND ZANGWILL, W. I. 1979. Finding all solutions to polynomial systems and other systems of equations. *Math. Programming 16*, 159–176.

GE, Y., COLLINS, JR., E. G., WATSON, L. T., AND DAVIS, L. D. 1994. An input normal form homotopy for the L^2 optimal model order reduction problem. *IEEE Trans. Automat. Control 39*, 1302–1305.

GE, Y., COLLINS, JR., E. G., AND WATSON, L. T. 1996. A comparison of homotopies for alternative formulations of the L^2 optimal model order reduction problem. *J. Comput. Appl. Math. 69*, 215–241.

GE, Y., WATSON, L. T., COLLINS, JR., E. G., AND BERNSTEIN, D. S. 1997. Globally convergent homotopy algorithms for the combined H^2/H^∞ model reduction problem. *J. Math. Systems, Estimation, Control*, to appear.

GE, Y., WATSON, L. T., COLLINS, JR., E. G., AND BERNSTEIN, D. S. 1996. Probability-one homotopy algorithms for full and reduced order H^2/H^∞ controller synthesis. *Optimal Control Appl. Methods 17*, 187–208.

GILL, P. E. AND MURRAY, W. 1974. Newton-type methods for unconstrained and linearly constrained optimization. *Math. Programming 7*, 311–350.

GOLUB, G. H. AND VAN LOAN, C. F. 1989. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 2nd ed.

HADDAD, W. M. AND BERNSTEIN, D. S. 1989. Combined L_2/H_∞ model reduction. *Internat. J. Control 49*, 1523–1535.

HASELGROVE, C. B. 1961. A solution of nonlinear equations and of differential equations with two-point boundary conditions. *Comput. J. 4*, 255–259.

HESTENES, M. R. 1956. The conjugate gradient method for solving linear systems. In *Proc. Symp. Appl. Math. 6*, Numer. Anal., AMS, New York, 83–102

HOLZER, S. M., PLAUT, R. H., SOMERS, JR., A. E., AND WHITE, W. S. 1980. Stability of lattice structures under combined loads. *J. Engrg. Mech. 106*, 289–305.

IRANI, K. M., KAMAT, M. P., RIBBENS, C. J., WALKER, H. F., AND WATSON, L. T. 1991. Experiments with conjugate gradient algorithms for homotopy curve tracking. *SIAM J. Optim. 1*, 222–251.

KAPANIA, R. K. AND YANG, T. Y. 1986. Formulation of an imperfect quadrilateral doubly curved shell element for postbuckling analysis. *AIAA J. 24*, 310–311.

KUBICEK, M. 1976. Dependence of solutions of nonlinear systems on a parameter. *ACM Trans. Math. Software 2*, 98–107.

LI, T. Y., SAUER, T., AND YORKE, J. A. 1988. Numerically determining solutions of systems of polynomial equations. *Bull. AMS 18*, 173–177.

MATTHIES, H. AND STRANG, G. 1979. The solution of nonlinear finite element equations. *Internat. J. Numer. Meth. Engrg. 14*, 1613–1626.

MCQUAIN, W. D., MELVILLE, R. C., RIBBENS, C. J., AND WATSON, L. T. 1994. Preconditioned iterative methods for sparse linear algebra problems arising in circuit simulation. *Computers Math. Appl. 27*, 25–45.

MEINTJES, K. AND MORGAN, A. P. 1985. A methodology for solving chemical equilibrium systems. Tech. Rep. GMR-4971, GM Res. Lab., Warren, MI.

MEJIA, R. 1986. CONKUB: A conversational path-follower for systems of nonlinear equations. *J. Comput. Phys. 63*, 67–84.

MELVILLE, R. C., MOINIAN, S., FELDMANN, P., AND WATSON, L. T. 1993a. Sframe: an efficient system for detailed DC simulation of bipolar analog integrated circuits using continuation methods. *Analog Integrated Circuits Signal Processing 3*, 163–180.

MELVILLE, R. C., TRAJKOVIĆ, LJ., FANG, S.-C, AND WATSON, L. T. 1993b. Artificial parameter homotopy methods for the DC operating point problem. *IEEE Trans. Computer-Aided Design 12*, 861–877.

MENZEL, R. AND SCHWETLICK, H. 1978. Zur Lösung parameterabhängiger nichtlinearer Gleichungen mit singulären Jacobi-Matrizen. *Numer. Math. 30*, 65–79.

MORGAN, A. P. 1986a. A transformation to avoid solutions at infinity for polynomial systems. *Appl. Math. Comput. 18*, 77–86.

MORGAN, A. P. 1986b. A homotopy for solving polynomial systems. *Appl. Math. Comput. 18*, 87–92.

MORGAN, A. P. 1987. *Solving polynomial systems using continuation for engineering and scientific problems.* Prentice-Hall, Englewood Cliffs, NJ.

MORGAN, A. P., AND SOMMESE, A. J. 1987. A homotopy for solving general polynomial systems that respects m-homogeneous structures. *Appl. Math. Comput. 24*, 101–113.

MORGAN, A. P., AND SOMMESE, A. J. 1989. Coefficient parameter polynomial continuation. *Appl. Math. Comput. 29*, 123–160. Errata: 51 (1992) 207.

MORGAN, A. P., SOMMESE, A. J., AND WAMPLER, C. W. 1991. Computing singular solutions to nonlinear analytic systems. *Numer. Math. 58*, 669–684.

MORGAN, A. P., SOMMESE, A. J., AND WAMPLER, C. W. 1995. A product-decomposition bound for Bezout numbers. *SIAM J. Numer. Anal. 32*, 1308–1325.

MORGAN, A. P., SOMMESE, A. J., AND WATSON, L. T. 1989. Finding all isolated solutions to polynomial systems using HOMPACK. *ACM Trans. Math. Software 15*, 93–122.

POORE, A. B. AND AL-HASSAN, Q. 1988. The expanded Lagrangian system for constrained optimization problems. *SIAM J. Control Optim. 26*, 417–427.

RAKOWSKA, J., HAFTKA, R. T., AND WATSON, L. T. 1991. Tracing the efficient curve for multiobjective control-structure optimization. *Comput. Systems Engrg. 2*, 461–472.

RHEINBOLDT, W. C. 1981. Numerical analysis of continuation methods for nonlinear structural problems. *Comput. & Structures 13*, 103–113.

RHEINBOLDT, W. C. 1986. *Numerical Analysis of Parametrized Nonlinear Equations.* Wiley-Interscience, New York.

RHEINBOLDT, W. C. AND BURKARDT, J. V. 1983. Algorithm 596: A program for a locally parameterized continuation process. *ACM Trans. Math. Software 9*, 236-241.

RICHTER, S. AND DECARLO, R. 1983. Continuation methods: theory and applications. *IEEE Trans. Aut. Control 28*, 660–665.

RICHTER, S. AND DECARLO, R. 1984. A homotopy method for eigenvalue assignment using decentralized state feedback. *IEEE Trans. Aut. Control 29*, 148–155.

RIKS, E. 1979. An incremental approach to the solution of snapping and buckling problems. *Internat. J. Solids Structures 15*, 529–551.

SAAD, Y. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput. 14*, 461–469.

SAAD, Y. 1996. *Iterative Methods for Sparse Linear Systems.* PWS, Boston, MA.

SAAD, Y. AND SCHULTZ, M. H. 1986. GMRES: a generalized minimal residual method for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput. 7*, 856–869.

SOSONKINA, M., WATSON, L. T., AND KAPANIA, R. K. 1998. A new adaptive GMRES algorithm for achieving high accuracy. *Numer. Linear Algebra Appl.*, to appear..

SOSONKINA, M., WATSON, L. T., AND STEWART, D. E. 1996. Note on the end game in homotopy zero curve tracking. *ACM Trans. Math. Software 22*, 281–287.

SHAMPINE, L. F. AND GORDON, M. K. 1975. *Computer Solution of Ordinary Differential Equations: The Initial Value Problem.* W. H. Freeman, San Francisco.

TRAJKOVIĆ, LJ., MELVILLE, R. C., AND FANG, S.-C. May, 1990. Passivity and no-gain properties establish global convergence of a homotopy method for DC operating points. In *Proc. IEEE Int. Symp. on Circuits and Systems*, New Orleans, LA, 914–917

TSAI, L.-W. AND MORGAN, A. P. 1985. Solving the kinematics of the most general six- and five-degree-of-freedom manipulators by continuation methods. *ASME J. Mechanisms, Transmissions, Aut. Design 107*, 48–57.

VAN DER VORST, H. A. 1992. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput. 13*, 631–644.

VAN DER VORST, H. A. AND VUIK, C. 1993. The superlinear convergence behaviour of GMRES. *J. Comp. Appl. Math. 48*, 327–341.

van der Vorst, H. A. and Vuik, C. 1994. GMRESR: a family of nested GMRES methods. *Numer. Linear Algebra Appl. 1*, 369–386.

van der Waerden, B. L. 1953. *Modern Algebra*. Vols. 1, 2, Frederick Ungar Pub. Co., New York.

Vasudevan, G., Watson, L. T., and Lutze, F. H. 1991. Homotopy approach for solving constrained optimization problems. *IEEE Trans. Automat. Control 36*, 494–498.

Verschelde, J. and Cools, R. 1993. Symbolic homotopy construction. *Appl. Algebra Engrg. Comm. Comput. 4*, 169–183.

Verschelde, J., Verlinden, P., and Cools, R. 1994. Homotopies exploiting Newton polytopes for solving sparse polynomial systems. *SIAM J. Numer. Anal. 31*, 915–930.

Walker, H. F. 1988. Implementation of the GMRES method using Householder Transformations. *SIAM J. Sci. Stat. Comput. 9*, 152–163.

Walker, H. F. 1996. Private communication.

Wampler, C. W., Morgan, A. P., and Sommese, A. J. 1990. Numerical continuation methods for solving polynomial systems arising in kinematics. *ASME J. Mechanical Design 112*, 59–68.

Wampler, C. W., Morgan, A. P., and Sommese, A. J. 1992. Complete solution of the nine-point path synthesis problem for four-bar linkages. *ASME J. Mech. Design 114*, 153–159.

Watson, L. T. 1979a. A globally convergent algorithm for computing fixed points of C^2 maps. *Appl. Math. Comput. 5*, 297–311.

Watson, L. T. 1979b. Solving the nonlinear complementarity problem by a homotopy method. *SIAM J. Control Optim. 17*, 36–46.

Watson, L. T. 1979c. An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems. *SIAM J. Numer. Anal. 16*, 394–401.

Watson, L. T. 1979d. Fixed points of C^2 maps. *J. Comput. Appl. Math. 5*, 131–140.

Watson, L. T. 1980a. Solving finite difference approximations to nonlinear two-point boundary value problems by a homotopy method. *SIAM J. Sci. Stat. Comput. 1*, 467–480.

Watson, L. T. 1980b. Computational experience with the Chow-Yorke algorithm. *Math. Programming 19*, 92–101.

Watson, L. T. 1981. Engineering applications of the Chow-Yorke algorithm. *Appl. Math. Comput. 9*, 111–133.

Watson, L. T. 1986. Numerical linear algebra aspects of globally convergent homotopy methods. *SIAM Rev. 28*, 529–545.

Watson, L. T. 1989. Globally convergent homotopy methods: a tutorial. *Appl. Math. Comput. 31BK*, 369–396.

Watson, L. T. 1990. Globally convergent homotopy algorithms for nonlinear systems of equations. *Nonlinear Dynamics 1*, 143–191.

Watson, L. T., Billups, S. C., and Morgan, A. P. 1987. Algorithm 652: HOMPACK: A suite of codes for globally convergent homotopy algorithms. *ACM Trans. Math. Software 13*, 281–310.

Watson, L. T., Bixler, J. P., and Poore, A. B. 1989. Continuous homotopies for the linear complementarity problem. *SIAM J. Matrix Anal. Appl. 10*, 259–277.

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

Watson, L. T., Holzer, S. M., Hansen, M. C. 1983. Tracking nonlinear equilibrium paths by a homotopy method. *Nonlinear Anal. 7*, 1271–1282.

Watson, L. T., Kamat, M. P., and Reaser, M. H. 1985. A robust hybrid algorithm for computing multiple equilibrium solutions. *Engrg. Comput. 2*, 30–34.

Watson, L. T. and Scott, L. R. 1987a. Solving Galerkin approximations to nonlinear two-point boundary value problems by a globally convergent homotopy method. *SIAM J. Sci. Stat. Comput. 8*, 768–789.

WATSON, L. T. AND SCOTT, M. R. 1987b. Solving spline collocation approximations to nonlinear two-point boundary value problems by a homotopy method. *Appl. Math. Comput. 24*, 333–357.

WEMPNER, G. A. 1971. Discrete approximations related to nonlinear theories of solids. *Internat. J. Solids Structures 7*, 1581–1599.

WRIGHT, A. H. 1985. Finding all solutions to a system of polynomial equations. *Math. Comp. 44*, 125–133.

YANG, T. Y., KAPANIA, R. K., AND SAIGAL, S. 1989. Accurate rigid-body modes representation for a nonlinear curved thin-shell element. *AIAA J. 27*, 211–218.

ŽIGIĆ, D. May, 1991. Homotopy methods for solving the optimal projection equations for the reduced order model problem. M.S. thesis, Dept. of Computer Sci., Virginia Polytechnic Institute and State Univ., Blacksburg, VA.

ŽIGIĆ, D., WATSON, L. T., AND BEATTIE, C. A. 1993. Contragredient transformations applied to the optimal projection equations. *Linear Algebra Appl. 188/189*, 665–676.

ŽIGIĆ, D., WATSON, L. T., COLLINS, JR., E. G., AND BERNSTEIN, D. S. 1992. Homotopy methods for solving the optimal projection equations for the H_2 reduced order model problem. *Internat. J. Control 56*, 173–191.

ŽIGIĆ, D., WATSON, L. T., COLLINS, JR., E. G., AND BERNSTEIN, D. S. 1993. Homotopy approaches to the H_2 reduced order model problem. *J. Math. Systems, Estimation, Control 3*, 173–205.