PARALLEL HOMOTOPY CURVE TRACKING ON A HYPERCUBE

by

Amal Chakraborty

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Application

APPROVED:

L. T. Watson, Chairman

C. J. Ribbens

C. Beattie

M. Abrams

D. C. S. Allison

May, 1990

Blacksburg, Virginia

ACKNOWLEDGEMENT

The author would like to thank his committee chairman, Prof. Layne Watson, for his valuable guidance and assistance during the course of this study. In particular, his patience and constant encouragement during the tedious stage of experimental work is deeply appreciated. The author would also like to acknowledge the assistance and co-operation of other committee members in the preparation of this thesis.

A special thanks is due to all the friends and colleagues who made the author's stay in Blacksburg more enjoyable.

A special tribute is paid to the author's mother Mrs. A. P. Chakraborty, for continuous encouragement and good wishes throughout this study period.

The author wishes to thank Dr. Deepak Sinha and Mrs. Aloka Sinha for their support, both financially and emotionally, throughout the author's student life. Without their support, I would not be where I am today.

Finally, I would like to dedicate this thesis to the memory of my late father Monoranjan Chakraborty and my late brother Milan Chakraborty. Their constant inspiration helped me to achieve. I hope they were as proud of me as I am of them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION.

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

Globally convergent homotopy algorithms are among the most powerful methods for solving non-linear systems of equations. While they are computationally more expensive than locally convergent methods, these algorithms can often reduce human effort by eliminating considerable work in finding a good starting point. Moreover, homotopy algorithms can handle problems which can not be solved effectively by other methods. Thus it is desirable to find faster homotopy algorithms. Serial homotopy methods, though subject to much research, are unlikely to present a breakthrough in speedup. Multiprocessors provide an opportunity to develop parallel homotopy algorithms that will be far less computationally expensive than their serial counterparts and thus will be competitive (with respect to computational time) with other methods.

Homotopy algorithms are related to some long established techniques of numerical analysis called continuation methods. These methods are related to but different from parameter continuation, incremental loading, displacement incrementation and invariant imbedding. The idea behind continuation methods is to solve a series of problems as some parameter λ is slowly varied monotonically. Thus a curve is traced out, producing a different point for each λ. Homotopy algorithms are based on this same curve tracking philosophy.

However, a major difference is that homotopy algorithms are not disturbed if the zero curve reverses direction in the λ component. The continuation methods require that λ change in only one direction. Different types of homotopy algorithms are discussed in [9] and [54].

HOMPACK [54] is a software package developed at Sandia National Laboratories, General Motors Research Laboratories, the University of Michigan, and Virginia Polytechnic Institute and State University. There are three algorithms in HOMPACK. These algorithms are known as globally convergent homotopy algorithms. Two versions of these algorithms exist – one for systems with dense Jacobian matrices and the other for sparse Jacobian matrices. The basic difference between the versions is how the linear algebra is handled. In this study we investigated how the dense Jacobian matrix algorithms can be implemented on a hypercube.

Chapter 2 briefly describes serial dense Jacobian matrix algorithms. The major steps that can be parallelized are discussed in detail. Chapter 3 includes detailed parallel algorithms to compute an element of the kernel of the Jacobian matrix of a homotopy map. Jacobian matrix evaluation is discussed in Chapter 4. Polynomial systems have a very special structure and are discussed in Chapter 5. Chapter 6 discusses the development of a parallel HOMPACK. Appendix A contains code for a parallel HOMPACK package.

Solving nonlinear systems of equations has significance for science and engineering. A special case, namely small polynomial systems of equations, occurs frequently in solid modelling, robotics, computer vision, chemical equilibrium computations, chemical process design, and mechanical engineering. There are three classes of nonlinear systems of equations: (1) large systems with sparse Jacobian matrices, (2) small transcendental (nonpolynomial) systems with dense Jacobian matrices, and (3) small polynomial systems with dense Jacobian

matrices. Because sparsity for small problems is not significant, and because large systems with dense Jacobian matrices are intractable, these two classes are not considered.

Large sparse nonlinear systems of equations, such as equilibrium equations in structural mechanics, have two aspects: highly nonlinear and recursive scalar computations, and large matrix, vector operations. There is a great amount of parallelism in both aspects, but the nature of the parallelism is very different. Small dense transcendental systems of equations pose a major challenge, since they involve recursive, scalar intensive computation with a small amount of linear algebra. It has been argued [2] that the communication overhead of hypercube machines makes them unsuited for such problems, but the issue is still open and algorithmic breakthroughs are yet possible. Polynomial systems are unique in that they have many solutions, of which several may be physically meaningful, and there exist homotopy algorithms guaranteed to find all these meaningful solutions. The special nature of polynomial systems and the power of homotopy algorithms are often not fully appreciated, perhaps because globally convergent probability-one homotopy methods have not received widespread attention.

Much work has been done on solving linear systems of equations on parallel computers, mostly on vector machines [17], [25], [28], [43]. Some work has been done on nonlinear equations and Newton's method [47], [48], [49], [50], and on finding the roots of a single polynomial equation [21], [45]. Some work has been done in nonlinear optimization on parallel computers [11], [46], [47]. Parallel algorithms for polynomial systems have been studied in [37], [38], [39]. Characteristics of large granularity parallel programs have been described in [23]. Granularity issues for solving polynomial systems on shared memory machines have been discussed in [3] and [4].

Some work has been done on parallelizing large software packages, mostly on vector computers. De Biase et al. [18] describes parallelization of an interactive software system for astronomical image processing. The frequently used time consuming operations such as convolution, geometrical normalization and geometrical correction have been split into a scalar part (typical of the operation) and a vector part (common among the whole operation set). In this way it is possible to achieve their parallelization without regard to a native or attached vector processor.

Bieterman [8] describes the application of CRAY X-MP microtasking to PLTMG [7]. PLTMG is a general purpose elliptic partial differential equation (PDE) package. It solves linear, nonlinear, and nonlinear parameterized scalar boundary value problems on general two-dimensional regions.

Some ongoing work on parallelizing ELLPACK has been reported in [29].

ITPACK ([30], [31]) is a package of subroutines for solving large sparse linear systems by adaptive iterative algorithms developed at the Center for Numerical Analysis of the University of Texas at Austin. It is a package of seven iterative algorithms for solving the linear system $Ax = b$ where A is a symmetric positive definite (or mildly nonsymmetric) matrix. The basic methods are the Jacobi, SOR, SSOR and RS method for the reduced system. Either conjugate gradient (CG) or Chebyshev (semi-iteration, SI) acceleration is applied to each of these basic methods, except for the SOR method. Kincaid and Oppe [32] discussed the development of a vector version of ITPACK to run on supercomputers such as the CDC Cyber 205 and CRI Cray 1.

Perhaps the work that is most closely related to this work is that by Byrd et al. [10], [11] and Schnabel [47], [48]. They consider local and global unconstrained optimization algorithms and how they can be executed on a network of processors. Byrd et al. [13]

presents a new parallel algorithm for the global optimization problem. The algorithm is a stochastic method related to the multi-level single linkage methods of Rinnooy Kan and Timmer [46] for sequential computers. Parallelism is achieved by partitioning the work of each of the three parts of the algorithm, sampling, local minimization, start point selection, and multiple local minimization among the processors. The parallelism is of a coarse grain type. Schnabel [47] discuss concurrent function evaluation in the context of unconstrained optimization. The basic assumption is that the function evaluation is expensive and gradients are computed by finite differences. The strategies include the use of speculative evaluation of the gradient concurrently with the evaluation of the function, before it is known whether the gradient value at this point will be required.

Byrd et al. [11], and Schnabel [48] discuss parallel function evaluation in the context of unconstrained optimization. Their approach is to compute the function and gradient completely but only part of the Hessian matrix. They let each processor compute a component of the gradient and the rest of the processors compute a single component of the Hessian matrix, assuming that the number of processors is greater than $(n+1)$ but less than $(n^2+3n+2)/2$. They do not let any processor compute more than one component. Byrd et al. [11] describes an algorithm that uses a partly computed Hessian matrix and analyzes the convergence properties of that algorithm.

1.1. Intel iPSC/1 and iPSC/2 hypercubes. The word 'hypercube' refers to an n-dimensional cube. Consider a cube in n-dimensions sitting in the positive orthant, with vertices at the points

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

There are 2^n vertices. Two vertices u and v are 'adjacent', i.e. connected by an edge, if and only if $u_i = v_i$ for all i except one. The associated graph has 2^n vertices and edges between vertices whose labels differ in exactly in one coordinate.

A n-dimensional 'hypercube computer architecture' is a multiprocessor computer architecture with 2^n processors corresponding to the 2^n vertices and a communication link corresponding to each edge of the n-cube. Thus each processor has a direct communication link to exactly n other processors. The distance (length of the shortest path) between any two processors is at most n.

Typically the nodes are labelled by a binary number $v_1 v_2 \dots v_n$. In this labelling scheme two nodes are adjacent if and only if their binary representations differ in exactly one bit.

The node addresses can be computed in programs by a gray code, a bijective function:

$$g_n : \{0, \dots, 2^n - 1\} \to \{0, \dots, 2^n - 1\}$$

defined recursively as follows.

Represent each integer in $\{0, \dots, 2^k - 1\}$ by a string of k binary digits, corresponding to its numerical value. Given a sequence $S = (s_1, \dots, s_n)$ of strings $s_i = d_i, \dots, d_{i_k}$ of binary digits $d_{i_j} \in \{0, 1\}$, let $0s_i = d_{i_1}, \dots, d_{i_k}$ and $0S = (0s_1, \dots, 0s_n)$, and similarly for $1s_i$ and $1S$. Let $S^R = (s_n, \dots, s_1)$ be the 'reverse' of S. Writing the values of g_k as strings, let $G_k = (g_k(0), g_k(1), \dots, g_k(2^k - 1))$. Then the gray code g_k is defined by

$$G_1 = (0, 1),$$

$$G_2 = (00, 01, 11, 10),$$

$$G_{k+1} = (0G_k, 1G_k^R).$$

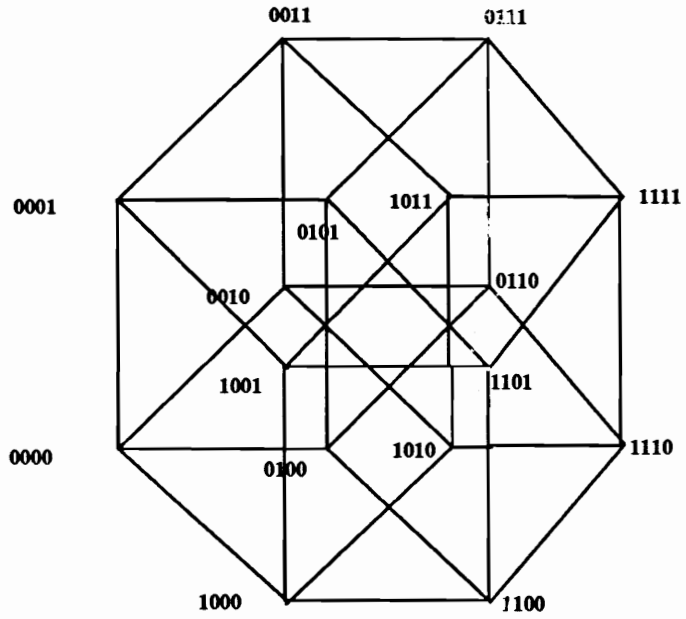

Figure 1. 4-cube structure and node labelling

Figure 1 shows the node labelling of a 4-dimensional hypercube.

Apart from these processors, each commercial hypercube computer system is equipped with an additional processor, called the 'host'. A 'host' processor has communication links to all the node processors. This host typically loads programs into the nodes, starts and stops processes executing in the nodes, and interchanges data with the nodes.

The Intel iPSC/1 is Intel's first generation hypercube computer system. It is available with 16, 32, 64 or 128 nodes. Each node is an 80286/80287 with 512K bytes of memory. Each node contains its own copy of the operating system and firmware to initialize the node. The operating system lets the node schedule and run processes within the node, make system calls to send and receive messages, and route messages through the node and to its neighbors. The message-passing algorithm use a store and forward mechanism. Store and forward message passing consumes node CPU cycles for relaying messages to other nodes

when these CPU cycles could be more profitably applied to the application. Because a single node can only relay one message at a time, the message throughput is far less than what the network can theoretically provide.

The iPSC/2 is Intel's second generation hypercube computer system. The two major differences between iPSC/1 and iPSC/2 are 1) the iPSC/2 uses 80386/80387 chips instead of 80286/80287 chips, and 2) the iPSC/2 uses a 'Direct-Connect' routing instead of a store and forward routing. In 'Direct-Connect' routing, when node A wants to send a message to node B, wherever B is in the network, the network builds a communication circuit between A and B dynamically and then transmits the complete message. At the end of the transmission the circuit is released for use in other communications. The 'Direct-Connect' network totally removes the node CPUs from the systems communication activities. Except when it is either the source of the message or its final destination, a node has nothing to do with message passing. The node's 'Direct-Connect' module (DCM) contains all the necessary hardware for building the circuits and moving the data. A single DCM can handle up to eight simultaneous communications at one time, all at full link speed.

2. SERIAL ALGORITHM.

The problem of solving a nonlinear systems of equations is to find an element x of $n-dimensional$ Euclidean space R^n such that $F(x) = 0$, where $F : R^n \to R^n$ is a C^2 map. The basic idea of the homotopy algorithm is to construct a homotopy map

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

such that

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

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

and for any fixed $a \in E^m$, with $\rho_a(\lambda,x) = \rho(a,\lambda,x)$,

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

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

4) ρ_a^{-1} is bounded.

Then the supporting theory [52], [53], [54] says that for almost all $a \in E^m$ there exists a zero curve γ of ρ_a, along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0,x_0)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. γ does not intersect itself and is disjoint from any other zeroes of ρ_a. An obvious choice for ρ_a is

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

This satisfies properties 1)–3) but not necessarily 4). There are fairly general sufficient conditions on $F(x)$ so that (3) will satisfy property 4), but for some practical problems of interest the homotopy map (3) will not suffice. This is why HOMPACK is designed to handle arbitrary homotopy maps $\rho_a(\lambda,x)$ satisfying properties 1) – 4).

The basic idea behind the homotopy algorithms is simple. Just follow the zero curve γ from $(0, x_0)$ until a point $(1, \bar{x})$ is found. $\bar{x}$ will then be the desired zero of F. This curve tracking is accomplished by the following parameterization. Assuming that $F(x)$ is C^2, a is such that the Jacobian matrix $D\rho_a(\lambda, x)$ has full rank along γ, and γ is bounded, the zero curve γ is C^1 and can be parameterized by arc length s. Thus $\lambda = \lambda(s)$, $x = x(s)$ along γ, and

$$\rho_a(\lambda(s), x(s)) = 0$$

identically in s. Therefore

$$\frac{d}{ds}\rho_a(\lambda(s), x(s)) = D\rho_a(\lambda(s), x(s)) \begin{pmatrix} \frac{d\lambda}{ds} \\ \frac{dx}{ds} \end{pmatrix} = 0, \tag{2.1}$$

$$\left\| \left(\frac{d\lambda}{ds}, \frac{dx}{ds} \right) \right\|_2 = 1. \tag{2.2}$$

With the initial conditions

$$\lambda(0) = 0, \quad x(0) = x_0, \tag{2.3}$$

the zero curve γ is the trajectory of the initial value problem (2.1–2.3). When $\lambda(\bar{s}) = 1$, the corresponding $x(\bar{s})$ is a zero of $F(x)$.

There are three qualitatively different algorithms provided in HOMPACK. They are:

1) ODE based algorithm,

2) normal flow algorithm,

3) augmented Jacobian matrix algorithm.

Algorithms (1) and (2) are suitable for parallelization and are described next. The material is repeated from [54]. For a more detailed discussion of all the algorithms see [54].

2.1. Ordinary Differential Equation Based Algorithm (dense Jacobian matrix). Equations 2.1) – 2.3) represent an initial value problem. Thus all the sophisticated ordinary differential equation techniques currently available can be used to track γ. Typical ordinary differential equation software requires $(d\lambda/ds, dx/ds)$ explicitly, and (2.1), (2.2) only implicitly define the derivative $(d\lambda/ds, dx/ds)$. (It might be possible to use an implicit ordinary differential equation technique for (2.1–2.2), but that seems less efficient than the method used in HOMPACK.) The derivative $(d\lambda/ds, dx/ds)$, which is a unit tangent vector to the zero curve γ, can be calculated by finding the one-dimensional kernel of the $n \times (n+1)$ Jacobian matrix

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

which has full rank according to the theory [29]. It is here that a substantial amount of computation is incurred, and it is imperative that the number of derivative evaluations be kept small. Once the kernel has been calculated, the derivative $(d\lambda/ds, dx/ds)$ is uniquely determined by (2.2) and continuity. Complete details for solving the initial value problem (2.1–2.3) and obtaining $x(\bar{s})$ are in [52] and [54]. A discussion of the kernel computation follows.

The Jacobian matrix $D\rho_a$ is $n \times (n+1)$ with rank n according to the supporting theory [54]. The crucial observation is that the last n columns of $D\rho_a$, corresponding to $D_x\rho_a$, may not have rank n, and even if they do, some other n columns may be better conditioned. The objective is to avoid choosing n "distinguished" columns, rather to treat all columns the same (not possible for sparse matrices). There are kernel finding algorithms based on Gaussian elimination and n distinguished columns. Choosing and switching these n columns is tricky, and based on *ad hoc* parameters. Also, computational experience has shown that accurate tangent vectors $(d\lambda/ds, dx/ds)$ are essential, and the accuracy of

Gaussian elimination may not be good enough. A conceptually elegant, as well as accurate, algorithm is to compute the QR factorization with column interchanges [2.3] of $D\rho_a$,

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

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

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

where the sign is chosen to maintain an acute angle with the previous tangent vector on γ.

Several features which are a combination of common sense and computational experience should be incorporated into the algorithm. [54] discusses these features in detail.

In summary, the algorithm [54] is:

1. Set $s := 0$, $y := (0, a)$, $ypold := yp := (1, 0, \ldots, 0)$, $restart :=$ false, $error :=$ initial error tolerance for the ODE solver.
2. If $y_1 < 0$ then go to 23.
3. If $s >$ some constant then
   4. $s := 0$.
   5. Compute a new vector a (see [54]). If

$$\|\text{new } a - \text{old } a\| > 1 + \text{constant} * \|\text{old } a\|,$$

   then go to 23.
6. $ode\ error := error$.

7. If $\|yp - ypold\|_\infty >$ (last arc length step) $*$ constant, then $ode\ error := tolerance \ll error$.

8. $ypold := yp$.

9. Take a step along the trajectory of (2.1–2.3) with the ODE solver. $yp = y'(s)$ is computed for the ODE solver by 10–12:

   10. Find a vector z in the kernel of $D\rho_a(y)$ using Householder reflections.

   11. If $z^t\, ypold < 0$, then $z := -z$.

   12. $yp := z/\|z\|$.

13. If the ODE solver returns an error code, then go to 23.

14. If $y_1 < 0.99$, then go to 2.

15. If $restart =$ true, then go to 20.

16. $restart :=$ true.

17. $error :=$ final accuracy desired.

18. If $y_1 \geq 1$, then set (s, y) back to the previous point (where $y_1 < 1$).

19. Go to 4.

20. If $y_1 < 1$ then go to 2.

21. Obtain the zero (at $y_1 = 1$) by interpolating mesh points used by the ODE solver.

22. Normal return.

23. Error return.

2.2. Normal flow algorithm (dense Jacobian matrix). As the homotopy parameter vector a varies, the corresponding homotopy zero curve γ also varies. This family of zero curves is known as the Davidenko flow. The normal flow algorithm is so called because the

iterates converge to the zero curve γ along the flow normal to the Davidenko flow (in an asymptotic sense).

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

$$Z^{(0)} = p(s_2 + h), \tag{2.4}$$

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 is

$$Z^{(k+1)} = Z^{(k)} - \left[D\rho_a(Z^{(k)})\right]^{\dagger} \rho_a(Z^{(k)}), \qquad k = 0, 1, \ldots \tag{2.5}$$

where $\left[D\rho_a(Z^{(k)})\right]^{\dagger}$ is the Moore-Penrose pseudoinverse of the $n \times (n+1)$ Jacobian matrix $D\rho_a$. Small perturbations of a produce small changes in the trajectory γ , and the family of trajectories γ for varying a is known as the "Davidenko flow". Geometrically, the iterates given by (2.5) return to the zero curve along the flow normal to the Davidenko flow, hence the name "normal flow algorithm".

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

$$[D\rho_a]\Delta Z = -\rho_a. \tag{2.6}$$

Fortunately ΔZ can be calculated at the same time as the kernel of $[D\rho_a]$, and with just a little more work. Normally for dense problems the kernel of $[D\rho_a]$ is found by computing a QR factorization of $[D\rho_a]$, and then using back substitution. By applying this QR factorization to $-\rho_a$ and using back substitution again, a *particular* solution v to (2.6) can be found. Let $u \neq 0$ be any vector in the kernel of $[D\rho_a]$. Then the minimum norm solution of (2.6) is

$$\Delta Z = v - \frac{v^t u}{u^t u} u. \tag{2.7}$$

Since the kernel of $[D\rho_a]$ is needed anyway for the tangent vectors, solving (2.6) only requires another $\mathcal{O}(n^2)$ operations beyond those for the kernel. The number of iterations required for convergence of (2.5) should be kept small (say less than 4) since QR factorizations of $[D\rho_a]$ are expensive. The alternative of using $[D\rho_a(Z^{(0)})]$ for several iterations, which results in linear convergence, is rarely cost effective.

When the iteration (2.5) converges, the final iterate $Z^{(k+1)}$ is accepted as the next point on γ, and the tangent vector to the integral curve through $Z^{(k)}$ is used for the tangent–this saves a Jacobian matrix evaluation and factorization at $Z^{(k+1)}$. The step size estimation described next attempts to balance progress along γ with the effort expended on the iteration (2.5).

Define a contraction factor

$$L = \frac{\left\| Z^{(2)} - Z^{(1)} \right\|}{\left\| Z^{(1)} - Z^{(0)} \right\|}, \tag{2.8}$$

a residual factor

$$R = \frac{\left\| \rho_a(Z^{(1)}) \right\|}{\left\| \rho_a(Z^{(0)}) \right\|}, \tag{2.9}$$

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

$$D = \frac{\left\| Z^{(1)} - Z^* \right\|}{\left\| Z^{(0)} - Z^* \right\|}, \tag{2.10}$$

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

$$\frac{\bar{L}}{L} \approx \frac{\bar{R}}{R} \approx \frac{\bar{D}}{D} \approx \frac{\bar{h}^q}{h^q} \tag{2.11}$$

for some q. This leads to the choice

$$\hat{h} = \left(\min\{\bar{L}/L, \bar{R}/R, \bar{D}/D\}\right)^{1/q} h, \tag{2.12}$$

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

$$\bar{h} = \min\left\{\max\{h_{\min}, B_{\min}h, \hat{h}\}, B_{\max}h, h_{\max}\right\}. \tag{2.13}$$

There are eight parameters in this process: $\bar{L}$, $\bar{R}$, $\bar{D}$, $h_{\min}$, $h_{\max}$, $B_{\min}$, $B_{\max}$, q. HOMPACK permits the user to specify nondefault values for any of these. The choice of $\bar{h}$ from (2.13) can be refined further. If (2.5) converged in one iteration, then $\bar{h}$ should certainly not be smaller than h, hence set

$$\bar{h} := \max\{h, \bar{h}\} \tag{2.14}$$

if (2.5) only required one iteration.

To prevent divergence from the iteration (2.5), if (2.5) has not converged after K iterations, h is halved and a new prediction is computed. Every time h is halved the old

value h_{old} is saved. Thus if (2.5) has failed to converge in K iterations sometime during this step, the new $\bar{h}$ should not be greater than the value h_{old} known to produce failure. Hence in this case

$$\bar{h} := \min\{h_{\text{old}}, \bar{h}\}. \tag{2.15}$$

Finally, if (2.5) required the maximum K iterations, the step size should not increase, so in this case set

$$\bar{h} := \min\{h, \bar{h}\}. \tag{2.16}$$

The logic in (2.14–2.16) is rarely invoked, but it does have a stabilizing effect on the algorithm.

The final phase, computation of the solution at $\lambda = 1$, begins when a point $P^{(2)}$ on γ is generated such that $P_1^{(2)} \geq 1$. The solution lies somewhere on γ between the previous point $P^{(1)}$ and $P^{(2)}$. The endgame now consists of iterating until convergence the sequence of steps: inverse interpolation with the Hermite cubic (2.4) for $\bar{s}$ such that $p(\bar{s})_1 = 1$; two iterations of (5) starting with $Z^{(0)} = p(\bar{s})$; replacing either $P^{(1)}$ or $P^{(2)}$ by $Z^{(2)}$ such that the solution on γ is always bracketed by $P^{(1)}$ and $P^{(2)}$. A precise statement of the endgame and the convergence criterion are given below.

In summary, the algorithm is:

1. $s := 0$, $y := (0, a)$, $h := 0.1$, *firststep* := true, *arcae*, *arcre* := absolute, relative error tolerances for tracking γ, *ansae*, *ansre* := absolute, relative error tolerances for the answer.
2. If *firststep* = false then
   3. Compute the predicted point $Z^{(0)}$ using (2.4).

   else

4. Compute the predicted point $Z^{(0)}$ using a linear predictor based on $y = (0, a)$ and the tangent there.

5. Iterate with equation (2.5) until either

$$\left\|\Delta Z^{(k)}\right\| \leq arcae + arcre \left\|Z^{(k)}\right\|$$

or

4 iterations have been performed

6. If the Newton iteration (2.5) did not converge in 4 steps, then

7. $h := h/2$.

8. If h is unreasonably small, then return with an error flag.

9. Go to 2.

10. $firststep :=$ false.

11. If $y_1 < 1$, then compute a new step size h by (2.8–2.16) and go to 2.

12. Do 13.–18. some fixed number of times.

13. Find $\bar{s}$ such that $p(\bar{s})_1 = 1$, using *yold*, *ypold*, y, yp in (2.4).

14. Do two iterations of (2.5) starting with $Z^{(0)} = p(\bar{s})$, ending with $Z^{(2)}$.

15. If

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

then return (solution has been found).

16. If $Z_1^{(2)} \geq 1$, then

17. $y := Z^{(2)}$, $yp :=$ tangent at $Z^{(2)}$.

else

18. $yold := Z^{(2)}$, $ypold :=$ tangent at $Z^{(2)}$.

19. Return with an error flag.

3. KERNEL COMPUTATION.

All the tracking algorithms described in the chapter 2 need a unit tangent vector at different points along the zero curve. Because of the way a homotopy map is constructed, finding a unit tangent vector amounts to finding the one dimensional kernel of the $n \times (n+1)$ Jacobian matrix $D\rho_a$, where $D\rho_a$ has (theoretical) rank n. The crucial observation is that the last n columns of $D\rho_a$, corresponding to $D_x\rho_a$, may not have rank n, and even if they do, some other n columns may be better conditioned. The objective is to avoid choosing n "distinguished" columns, but rather to treat all columns the same (which is not possible for sparse matrices). There are kernel finding algorithms based on Gaussian elimination and n distinguished columns. Choosing and switching these n columns are tricky, and based on *ad hoc* parameters. Also, computational experience has shown that accurate tangent vectors $(d\lambda/ds, dx/ds)$ are essential, and the accuracy of Gaussian elimination may not be good enough. A conceptually elegant, as well as accurate, algorithm is to compute the QR factorization (with column interchanges) $Q\, D\rho_a\, P^t = R$, where Q is a product of Householder reflections, P is a permutation matrix, and R is $n \times (n+1)$ upper triangular. We then obtain a vector $z \in \ker(D\rho_a)$ by solving $R(Pz) = 0$. Setting $(Pz)_{n+1} = 1$ is a convenient choice. This scheme provides high accuracy, numerical stability, and a uniform treatment of all $n+1$ columns.

The kernel of the Jacobian can also be found by computing an LQ factorization of $D\rho_a$. Once an LQ factorization is obtained, an element of the kernel can be found by solving $Lx = 0$ and then solving $Qz = x$. Notice that $e_{n+1} = (0, \ldots, 0, 1)^t$ is a solution of $Lx = 0$, so that $Q^t e_{n+1}$ is a solution to the system $D\rho_a z = 0$. Thus, the last column of Q^t is in the kernel of $D\rho_a$. Since $D\rho_a$ is $n \times (n+1)$ with full rank, row interchanges are not necessary.

Also, the method does not require a triangular solver. However, the matrix Q needs to be formed explicitly. This computation can be carried out simultaneously with the factorization of $D\rho_a$ without any extra communications. This will double the arithmetic computation in the factorization phase but will avoid the computation and communication associated with the triangular solver. For a small matrix ($n < 100$) this can be advantageous.

3.1. Parallel algorithms. The most time consuming parts of homotopy curve tracking are evaluating ρ_a and $D\rho_a$ and finding the unit tangent vector at different points along the zero curve γ. The three major steps for finding a unit tangent vector at a point (λ, x) on γ are: 1) compute the Jacobian matrix of the homotopy map ρ_a at (λ, x); 2) compute an orthogonal decomposition of the Jacobian matrix $D\rho_a$; 3) solve a triangular system of equations if necessary. Parallel Jacobian matrix evaluation is discussed in the next chapter. This chapter discusses parallelization of steps 2 and 3.

3.1.1. Parallel orthogonal decompositions.

Most of the literature on parallel orthogonal decompositions on distributed memory multiprocessors describes how to compute a QR factorization of a matrix (see [15], [36], [37], and [16]). Extending these algorithms to compute an LQ factorization is straight-forward. There are several parallel QR algorithms proposed in the literature. However, most of these algorithms do not consider the case when column switching is necessary. The objective here is to examine existing orthogonal factorization algorithms and incorporate necessary column switching so that they can be used to track a homotopy curve. These methods are then compared with LQ factorizations. The two algorithms described in [16] are the most suitable for our purpose. The algorithms and the necessary modifications are described briefly here. For a detailed description of the original algorithms refer to [6].

In the first algorithm of [16], the processors are mapped into a ring. The rows of the matrix are assigned to the processors in a wrap-mapping fashion. In this, the ith row is assigned to the processor numbered $i - \lfloor (i-1)/p \rfloor p - 1$, where p is the number of processors. Figure 2 shows the wrap-mapping of 9 rows to 4 processors.

$$\begin{bmatrix} P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\ P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\ P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\ P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\ P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\ P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\ P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\ P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\ P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \end{bmatrix}$$

Figure 2. The wrap mapping of 9 rows to 4 processors.

The algorithm consists of $n-1$ stages. At each stage (at the kth stage elements below the diagonal of column k are zeroed out), the algorithm performs two steps: 1) each processor independently introduces zeros in the part of the column it holds by Givens rotations or Householder reflections, 2) processors cooperate with each other in a recursive merge fashion to introduce the rest of the zeros. In this case a slight modification of the algorithm permits the switching of columns without any communication. The original algorithm performs some redundant computation in order to use a similar communication algorithm in the beginning of each stage. Because of this redundant computation, each processor has the modified pivot row (i.e., the kth row at the beginning of the $(k+1)$st stage). Each processor also holds an array $sum(j)$ $(j = 1, \ldots, n+1)$, where at the beginning of the kth stage $sum(i)$ $(i = k, \ldots, n+1)$ contains the norm of the ith subcolumn $A(k,i), \ldots, A(n,i)$. (The array A contains the $n \times (n+1)$ Jacobian $D\rho_a$). At the end of the kth stage each processor updates $sum(i)$ by executing $sum(i) := SQRT(sum(i)^2 - A(k,i)^2)$ for $i = k+1, \ldots, n+1$. At the beginning of stage 1, in a hypercube of dimension d, each processor executes the following algorithm to initialize the array sum:

for $i := 1, \ldots, n+1$ **do**

initialize $sum(i)$ **to the norm of the portion of the** i**th subcolumn it holds**

end

for $j := 1, \ldots, d$ **do**

send the array sum **to the processor whose id differs only in bit** j

receive array sum **in** t **from the processor whose id differs only in bit** j

for $i := 1, \ldots, n+1$ **do**

$sum(i) := SQRT(sum(i)^2 + t(i)^2)$

end

end

A brief description of the factorization algorithm, consisting of an independent annihilation phase (IAP) and a cooperative merging phase (CMP), follows:

Step 1. (IAP) Among all of the rows with row number $i \geq k$, each processor uses the lowest numbered row as the pivot row to eliminate all of the off-diagonal nonzero elements in the kth column of its remaining rows by a Householder transformation. Since each processor has all the necessary information to update the part of the matrix it holds, no communication is needed here.

Step 2. (CMP) $l := d$, where d is the dimension of the hypercube.

Step 3. (CMP) Every processor sends its current local pivot row to the processor whose id differs in bit b_{l-1}. It also receives a row from the other processor. Let ρ_1 and ρ_2 be the row numbers where $\rho_1 > \rho_2$.

Step 4. (CMP) Each processor computes a Givens rotation to eliminate the element $a_{\rho_2,k}$. If the row ρ_2 is originally assigned to this processor then this row is updated and saved.

Step 5. (CMP) Each processor updates row ρ_1. The updated row ρ_1 becomes the current local pivot row.

Step 6. (CMP) $l := l - 1$.

Step 7. (CMP) If $l \geq 1$ then go to Step 3.

Step 8. (CMP) Retrieve the saved row.

Step 9. (CMP) Update norms of each column.

In the second algorithm of [16], the processors are mapped into a $\lambda_1 \times \lambda_2$ rectangular grid, where $\lambda_1 = 2^{d_1}$, $\lambda_2 = 2^{d_2}$ and $d = d_1 + d_2$. Each row or column forms a subcube. Processors are arranged so that the id's of the processors in the same row differ only in the rightmost d_2 bits. Similarly, the id's of the processors in the same column differ only in the leftmost d_1 bits. Figure 3 shows the embedding of a 4×4 grid in a 16 processor hypercube. Matrix rows are assigned to subcubes corresponding to grid rows in a wrap-mapping fashion. Elements of each matrix row, in turn, are assigned to the processors in the corresponding row subcube in a wrap-mapping fashion. Thus a single processor does not hold a complete row or column of the matrix. Instead, each row subcube holds a complete matrix row, and each column subcube holds a complete matrix column. Figure 4 shows the wrap-mapping of a 8×9 matrix to a 4×4 processor grid.

$$\begin{bmatrix} P_0 & P_1 & P_2 & P_3 \\ P_4 & P_5 & P_6 & P_7 \\ P_8 & P_9 & P_{10} & P_{11} \\ P_{12} & P_{13} & P_{14} & P_{15} \end{bmatrix}$$

Figure 3. The embedding of a 4×4 matrix into a 16 processor hypercube.

$$\begin{bmatrix} P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 \\ P_4 & P_5 & P_6 & P_7 & P_4 & P_5 & P_6 & P_7 & P_4 \\ P_8 & P_9 & P_{10} & P_{11} & P_8 & P_9 & P_{10} & P_{11} & P_8 \\ P_{12} & P_{13} & P_{14} & P_{15} & P_{12} & P_{13} & P_{14} & P_{15} & P_{12} \\ P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 \\ P_4 & P_5 & P_6 & P_7 & P_4 & P_5 & P_6 & P_7 & P_4 \\ P_8 & P_9 & P_{10} & P_{11} & P_8 & P_9 & P_{10} & P_{11} & P_8 \\ P_{12} & P_{13} & P_{14} & P_{15} & P_{12} & P_{13} & P_{14} & P_{15} & P_{12} \end{bmatrix}$$

Figure 4. The wrap mapping of a 8×9 matrix into a 4×4 grid.

Algorithm 2 is based on the same Householder/Givens sequence employed by the first algorithm. However, a subcube instead of a single processor holds a row, so there is some communication involved in the independent annihilation phase. The processor holding the local pivot element must broadcast all of the computed multipliers to every other processor in its subcube. At the beginning of the kth stage, if the processors in the subcube holding the kth column have at least one more column, they can switch columns without any communication. Otherwise, they cooperate in switching columns with the subcube holding the next column. This requires one communication step. This situation will arise only in the last λ_2 stages, when each subcube holds at most one column that has not yet been treated.

The LQ versions of the above QR algorithms follow nearly the same steps. The difference is that in the case where the processors are mapped into a ring the columns are mapped to the processors in a wrap-mapping fashion. An array $\mathcal{Q}$, which accumulates Q, is initialized to I. Each processor holds exactly the same columns of $\mathcal{Q}$ as the original matrix A. At each stage while the matrix A is being multiplied by an orthogonal matrix Q_i, $\mathcal{Q}$ is also multiplied by Q_i. When two processors need to exchange information during the recursive merging phase, they can exchange some extra information about the matrix $\mathcal{Q}$. This enables the processors to compute $\mathcal{Q}$ without any extra message passing. However, the

message length will increase. This also requires each processor to store $((n+1)/p+1)(n+1)$ extra numbers for Q, where p is the number of processors.

3.1.2. Parallel triangular system solver.

A triangular system solver is needed along with QR factorization to obtain the unit tangent vector. There are two choices here. The first choice is to ship the factorization result back to the host and use a serial triangular system solver. The second choice is to keep the factorization results in the nodes and use a parallel triangular solver. Since triangular system solving is inherently serial, the relative frequency of communication with respect to the amount of computation involved is very high. Most of the literature on parallel triangular system solving on distributed memory multiprocessors assumes the size of the matrix is on the order of 1000. There are many parallel algorithms that are proposed in the literature (e.g., [22], [27], and [33]). However, the distinction between the algorithms are apparent only for medium or large size matrices. Thus, the purpose here is not to find the best parallel triangular solver but to investigate if it is at all advantageous to use a parallel algorithm. If our purpose was only to solve a triangular system it would be better to use an algorithm where each processor has full rows or columns (e.g., the processors are mapped into a ring). However, the results in Chapter 4 indicate that the factorization algorithms do much better when processors are mapped into a rectangular grid, and the savings in time are much greater than we could possibly get from solving the triangular system with a ring connected topology. Thus a different triangular solver, which assumes that the processors are mapped into a rectangular grid, is used. A simplified version of this algorithm to solve $Ax = 0$ is given below, where the $n \times (n+1)$ upper triangular matrix A is stored in the array a.

if $MYCOL(n+1)$ **then**

$x(n+1) := 1$

$b(.) := -a(., n+1)$

put $(b, LEFT)$

endif

for $k := n, n-1, \ldots, 1$ **do**

begin

if $MYCOL(k)$ **then**

if $MYROW(k)$ **then**

get $(b, RIGHT)$

$x(k) := b(k)/a(k,k)$

$b(.) := b(.) - x(k) * a(., k)$

put $(x(k), UP)$

put $(b, LEFT)$

else

get $(x(k), DOWN)$

put $(x(k), UP)$

get $(b, RIGHT)$

$b(.) := b(.) - x(k) * a(., k)$

put $(b, LEFT)$

endif

endif

end

3.2. Computational results.

Tables 1–3 show the computational results for the linear algebra part of homotopy curve tracking, obtained on a 16 node and a 32 node Intel iPSC/2 machine. All the matrices were $n \times (n+1)$ with full rank. Since the purpose of this work was to study ways to track a low dimensional homotopy path in parallel, and orthogonal factorizations are appropriate only when the Jacobian matrix is small and dense, the algorithms were tested on relatively small size matrices. The matrices used for Tables 1–3 were generated randomly. The notation used in the tables is as follows:

SER – serial QR factorization on the host;

QR1 – QR factorization, processors are mapped into a linear array, independent phase is done by Householder reflections, recursive merging phase is done by Givens rotations;

QR2 – same as QR1 except processors are mapped into a $d_1 \times d_2$ rectangular grid;

LQ1 – LQ factorization, processors are mapped into a linear array, independent phase is done by Householder reflections, recursive merging phase is done by Givens rotations;

LQ2 – same as LQ1 except processors are mapped into a $d_1 \times d_2$ rectangular grid;

TSS – triangular system is solved serially;

TSP – triangular system is solved in parallel;

QRS – QR factorization and the triangular system solution are done serially;

QRT – QR factorization (algorithm 2) is done in parallel and the triangular system is solved serially;

QRP – QR factorization (algorithm 2) and the triangular system solution are done in parallel;

It can be seen from Tables 1 and 2 that the parallel algorithms are performing better than the serial algorithm for matrices of size larger than 50. The orthogonal decomposition

Table 1. Execution time in secs (16 nodes).

n	SER	QR1	QR2	LQ1	LQ2
16	1.3	4.2	5.0	4.3	5.0
32	3.1	6.0	6.1	7.3	6.6
50	10.0	9.7	8.3	10.2	9.5
64	21.3	13.1	9.5	18.3	12.3
75	30.0	14.1	11.0	21.4	15.6
100	72.0	20.3	15.0	35.0	20.0
150	200.0	33.5	23.0	85.0	62.2

Table 2. Execution time in secs (32 nodes).

n	SER	QR1	QR2	LQ1	LQ2
16	1.3	5.7	6.4	5.4	6.3
32	3.1	7.6	7.4	8.2	8.2
50	10.0	9.7	8.6	11.9	9.7
64	21.3	10.7	9.0	15.8	9.9
75	30.0	14.4	11.4	19.4	13.7
100	72.0	22.0	14.1	27.5	17.1
150	200.0	32.0	21.9	68.0	34.2

is just one part of the larger homotopy curve tracking algorithm. In this larger context, if the functions are evaluated in parallel, it is possible that this cross-over point will be lower. The reason is that when the functions are evaluated in parallel and the factorization is done serially, the evaluated Jacobian matrix has to be shipped back to the host. This will increase the overall time taken to evaluate the functions and compute an orthogonal factorization. For a parallel orthogonal factorization this initial shipment of data is not necessary. In the case of LQ factorization, the unit tangent vector is already computed and nodes can return it instead of a much larger Jacobian matrix (in the case of a serial factorization algorithm) or resultant triangular matrix (in the case of a QR factorization).

It can also be seen from Tables 1 and 2 that LQ factorizations are doing substantially worse than QR algorithms for matrices of size larger than 64. Despite the savings due to avoiding a triangular solve in the LQ algorithms, the extra computations needed to compute

Table 3. Execution time in secs (32 nodes).

n	TSS	TSP	QRS	QRT	QRP
16	–	1.7	1.3	6.4	6.5
32	–	2.5	3.1	7.5	7.4
50	0.3	2.8	10.1	8.8	8.7
64	0.7	2.9	21.6	9.5	9.1
75	1.2	3.1	30.8	12.1	11.6
100	2.0	3.4	73.5	15.5	14.5
150	3.9	3.8	203.0	24.0	22.1

Q make these algorithms less efficient. There are other ways to compute Q which might be better than the method used here. We also see from Tables 1 and 2 that the algorithms based on a grid of processors are more efficient than those that organize the processors in a ring, for all but the smallest problems.

Table 3 shows that the serial triangular solver (TSS) is doing better than the parallel triangular solver (TSP) most of the time. However, as the size of the matrices increases the parallel algorithm becomes comparable. Though the serial triangular solver is doing better than the parallel triangular solver, it is not doing so when used in conjunction with QR factorizations (QRT versus QRP). This occurs because for QRT a large amount of data needs to be shipped back from the nodes to the host. On the other hand, when the parallel triangular solver (QRP) is used it is not necessary to ship the matrix back to the host. Only the solution to the system (which is much smaller than the triangular matrix) needs to be transmitted.

Since the granularity of computation is quite fine for small matrices, using more processors is not helpful, as shown by Tables 1 and 2. For matrices of size smaller than 50, the time taken by 16 processors is less than the time taken by 32 processors. This is because the time saved at each stage (using the additional processors) is less than the extra communication overhead. Overall QRP performs best.

4. JACOBIAN MATRIX EVALUATION.

The Jacobian matrix and function evaluations are often the most time consuming part of a homotopy algorithm. However, they are also the most easily parallelized part of the computation. Byrd et al. [11] and Schnabel [47] discuss parallel function evaluation in the context of unconstrained optimization. Their approach is to compute the function and gradient completely but only part of the Hessian matrix. Each component of the gradient is computed by a single processor and the rest of the processors compute a single component of the Hessian matrix, assuming that the number of processors is greater than $(n+1)$ but less than $(n^2+3n+2)/2$. They do not let any processor compute more than one component. Byrd [11] describes an algorithm that uses a partly computed Hessian matrix and analyzes the convergence properties of that algorithm. The present work assumes that the complete Jacobian matrix needs to be formed. Since the number of processors is generally less than (n^2+n), each processor must compute more than one component. This section examines the effect of different component complexity distributions and the size of the Jacobian matrix on the different assignments of components to the processors, and determines in what context one assignment would perform better than others.

4.1. Parallel Jacobian matrix evaluation.

For many engineering problems it is not feasible to compute the Jacobian matrix analytically. Most often the Jacobian matrix is estimated using finite difference approximations. Given that all components of the Jacobian matrix can be computed independently, a good parallel algorithm, with low communication requirements, would be to let each processor compute exactly those components that will be assigned to it during the orthogonal decomposition phase (static assignment). Since the orthogonal decomposition is most efficient when a rectangular mapping is used, this assignment will mean that a single column is

evaluated by more than one processor. In many cases, this will result in redundant computation and extra communication overhead. If d_1 and d_2 are the dimensions of the subcubes in the rectangular grid, then the maximum number of components that a processor has to compute is

$$\left\lceil \frac{n}{2^{d_1}} \right\rceil \left\lceil \frac{n+1}{2^{d_2}} \right\rceil .$$

Another approach would be to use a linear mapping by column in the evaluation phase and a rectangular mapping in the decomposition phase. In this method, the host will collect the complete matrix from the nodes and then redistribute the matrix according to a rectangular mapping. If the linear mapping by column in the evaluation phase is used, then the maximum number of components that a processor has to evaluate is

$$\left\lceil \frac{n+1}{2^{d_1+d_2}} \right\rceil n.$$

The maximum number of elements that a processor has to compute in the linear mapping case can be greater than or less than the maximum number of components that a processor has to compute in the rectangular mapping case, so there is no clear preference on these grounds. If the Jacobian matrix is not large enough to justify parallel decomposition, both the linear and rectangular mapping algorithms have to return the evaluated Jacobian matrix to the host before serial decomposition can proceed. However, because in this case there there may be fewer columns than processors, processor loads could be unbalanced. If p is the number of processors and n is the column dimension of the Jacobian matrix then $p - n$ processors will be idle in the evaluation phase. Because $n(n+1)$ components can be evaluated in parallel, a rectangular mapping scheme would utilize more processors. When $n >> p$, the load distribution for a linear mapping is not as bad as when $n \approx p$. Because the host has to collect all the columns and redistribute them to the processors, there is some

communication involved at the end of the evaluation phase. Also, all the processors have to wait for the Jacobian matrix evaluation to finish. In the case of a rectangular mapping, a significant portion of the IAP phase of the first stage could be completed before the whole Jacobian matrix is evaluated. This may or may not save any time, as all the processors have to be synchronized for the CMP phase. In any event the savings would not be enough to justify the rectangular mapping if loads are evenly balanced with the linear mapping. This is especially true because redundancy of computation may be high with the rectangular mapping case.

However, in either static assignment, if the variation in the evaluation time of the components is high, there may be uneven loading of the processors and thus inefficiency. In this situation it may be better to use the host as the master and let it assign components to the processors as they become available (dynamic assignment). However, this method has a large communication overhead. Also, when there is a large number of components to evaluate, the variation in the total evaluation time among the processors should not be very high. Thus the master/slave paradigm is advantageous only when the Jacobian matrix is relatively small, each component is very expensive to evaluate, and there is a large variation in the evaluation times.

4.2. Computational results.

The Jacobian matrix can be large or small with each component cheap or expensive to compute. If the Jacobian matrix is large enough so that parallel factorization is advantageous, then it is always better to evaluate the components of the Jacobian matrix in parallel. This is true because parallel evaluation has little overhead in addition to that already incurred by the decomposition algorithm. When the Jacobian matrix is very small but component evaluation expensive enough to justify parallel evaluation, it is better to do the factorization

Table 4. Execution time in seconds for different costs of component evaluation (32 nodes, $n = 11$).

KFLOPS	SFER	PFER
.5	9.4	9.8
.65	9.8	9.9
1	15.5	10.2
2	27.5	10.3
5	55.7	11.8
10	103.5	13.5
40	372.0	24.1

and triangular system solving serially. Tables 4–12 show timings for different situations. In the tables, R and L refers to a 4×8 rectangular and linear mapping respectively, and DYNAMIC refers to the master/slave model where the host dynamically assigns component evaluations to the nodes. The experiments were done on a 32 node Intel iPSC/1 hypercube. The notation used in the tables is as follows:

SFER – serial function and Jacobian matrix evaluation;

PFER – parallel function and Jacobian matrix evaluation;

SKER – serial function evaluation, Jacobian matrix evaluation, and unit tangent vector computation;

PFUN – function and Jacobian matrix evaluation are done in parallel but factorization and triangular solving are done serially;

PKER – parallel unit tangent vector computation (parallel function, Jacobian matrix evaluation and QR factorization using algorithm 2).

If the system has a small and cheap to compute Jacobian matrix then everything should be done serially. The crossover point at which it is better to use a parallel evaluation algorithm depends not only on the complexity of component evaluation but also on the interdependency among the components. Table 4 shows the results when component evaluation is totally independent and the cost of evaluation for each component is the same. Here parallel function evaluation is better than serial evaluation when each component requires at least 0.7 KFLOPS, and very much better for larger values of KFLOPS.

Table 5. Execution time in seconds for different costs of component evaluation

(Uniform, 32 nodes, $n = 11$).

Methods	U(0,2)	U(0,4)	U(0,10)	U(0,20)	U(3,27)	U(38,62)	U(88,112)
SERIAL	12.8, .01	21.5, .03	46.6, .06	91.0, 0.3	133, 1.6	443, 3.0	874, 9.0
STATIC(R)	12.4, 0.4	12.8, .03	14.1, .07	16.8, .4	18.1, .60	31.0, .70	49.2, 1.5
STATIC(L)	13.7, .03	15.0, .15	17.4, .22	22.1, 1.33	25.5, 2.5	48.7, 3.0	80.5, 5.5
STATIC(L20)	13.5, .03	14.8, .03	16.1, .04	20.5, .06	23.8, .66	42.3, 1.1	68.0, 1.4
STATIC(L50)	13.2, .03	13.7, .03	14.8, .05	17.4, .22	20.3, .49	33.2, 2.3	48.7, 3.0
DYNAMIC	14.5, .03	14.6, .04	14.8, .04	15.3, .04	16.4, .06	25.9, 1.56	39.1, 2.0

Table 5. Execution time in seconds for different costs of component evaluation

(Normal, 32 nodes, $n = 11$).

Methods	N(1,1)	N(2,2)	N(5,5)	N(10,7)	N(15,7)	N(50,7)	N(100,7)
SERIAL	13.1, .04	22.4, .33	52.1, 6.0	96.3, 8.0	141.5, 13.0	451, 25	879, 44
STATIC(R)	12.1, .03	12.8, .13	14.5, 0.8	16.8, 1.7	18.5, 1.8	31.2, 4.7	51.1, 6.0
DYNAMIC	13.2, .06	13.2, .14	14.8, .22	15.2, .66	17.8, 1.1	28.1, 4.7	44.5, 6.3

Experiments were done to study the effects of matrix size, cost of component evaluation, and the distribution of cost on static and dynamic assignment algorithms. The cost of component evaluation, ρ, is a random variable whose distribution is varied in these experiments. Tables 5–11 show the mean and variance of timings taken from three runs with different seeds. The cost of each component was generated randomly from the following probability distributions:

1) $U(a, b)$ - uniform distribution with lower bound a KFLOPS and upper bound b KFLOPS;
2) $N(a, b)$ - normal distribution truncated on the left at 0 with mean a KFLOPS and standard deviation b;
3) $E(a)$ - exponential distribution with mean a KFLOPS.

STATIC L20 and STATIC L50 denote algorithms which assume a linear mapping scheme where the computation is 20% and 50% less than for the rectangular mapping. It can be seen from Tables 5, 6, and 7 that when the components are not expensive (mean is less than 10

Table 7. Execution time in seconds for different costs of component evaluation

(Exponential, 32 nodes, $n = 11$).

Methods	E(1)	E(2)	E(5)	E(10)	E(15)	E(50)	E(100)
SERIAL	12.8, 1.8	22.5, 1.8	51.8, 6.0	100.3, 12.0	148, 43	502, 104	963, 1400
STATIC(R)	12.3, .06	13.1, .22	15.2, .36	19.1, .94	22.9, 1.6	49.1, 12.0	86.1, 101
DYNAMIC	14.0, 0.7	14.2, 0.7	14.9, 0.7	15.5, 1.1	17.7, 2.2	31.1, 4.3	52.5, 6.0

Table 8. Execution time in seconds for different costs of component evaluation

(Uniform, 32 nodes, $n = 50$).

Methods	U(0,2)	U(0,4)	U(0,10)	U(0,20)	U(3,27)	U(38,62)	U(88,112)
SERIAL	196.5, 1.3	377.9, 2.2	925.3, 4.3	1838, 8.0	2742, 12.0	9014, 18	18008, 70
STATIC(R)	27.7, .03	32.0, .04	47.3, .06	78.7, 1.1	104.5, 1.1	294, 2.0	563, 3.0
STATIC(L)	29.4, .04	39.1, .06	61.0, .66	94.0, 2.0	125.0, 4.7	334, 8.0	632, 13.0
STATIC(L20)	31.3, .04	36.7, .04	53.0, .06	92.6, .22	107.4, 1.3	274, 4.7	511, 6.0
STATIC(L50)	29.3, .03	29.4, .04	42.7, .06	60.0, .66	76.4, .73	195, 1.3	365, 8.0
DYNAMIC	95, .04	97.0, .04	99.4, .06	107.5, .68	109, 1.3	256, 2.2	491, 6.7

Table 9. Execution time in seconds for different costs of component evaluation

(Normal, 32 nodes, $n = 50$).

Methods	N(1,1)	N(2,2)	N(5,5)	N(10,7)	N(15,7)	N(50,7)	N(100,7)
STATIC(R)	23.5, .06	29.1, .06	44.9, .22	72.2, 1.1	97.8, 1.3	286.3, 1.7	570.7, 2.2
DYNAMIC	94, .04	96.8, .04	98.7, .06	99.8, .50	106.5, 1.1	256.1, 1.7	490.3, 4.0

Table 10. Execution time in seconds for different costs of component evaluation

(Exponential, 32 nodes, $n = 50$).

Methods	E(1)	E(2)	E(5)	E(10)	E(15)	E(50)	E(100)
STATIC(R)	26.6, .06	32.5, .22	48.9, .66	77.2, 1.3	109.6, 2.2	314, 8.0	607, 13.3
DYNAMIC	94.2, 1.1	96.4, 1.1	98.7, 1.3	100.2, 2.2	108, 6.0	269, 12.0	518, 50.0

KFLOPS) to evaluate, dynamic assignments are not as good as static assignments. However, when each component takes more than 10 KFLOPS, dynamic assignments are performing better than static assignments. This is more apparent when component evaluation times are exponentially distributed (Table 7). In this case the variation in computation time is high

Table 11. Execution time in seconds for different costs of component evaluation (Uniform, 32 nodes, $n = 98$).

Methods	U(0,2)	U(0,4)	U(0,10)	U(0,20)	U(3,27)	U(38,62)	U(88,112)
STATIC(R)	60.7, .66	82.5, 1.1	142, 1.3	240, 2.2	339, 6.6	1015, 8.0	1982, 16.0
STATIC(L)	74.0, 1.1	99.4, 1.1	167.0, 1.6	286.0, 3.0	422.0, 8.0	1204, 12.0	2397, 51.0
STATIC(L20)	70.5, .70	89.5, 1.1	149.0, 1.4	241.5, 2.2	335.5, 6.6	991, 8.0	1925, 18.0
STATIC(L50)	63.5, .66	75.5, 1.1	113.0, 1.1	170.0, 1.6	230.0, 1.7	639, 8.0	1221, 12.0
DYNAMIC	434.0, 2.2	434.6, 2.2	435.5, 2.2	438.0, 2.2	441, 3.1	803, 6.6	1836, 16.0

and thus static assignment results in a poor load balancing among the nodes. Tables 8, 9, and 10 show the results for a 50×51 Jacobian matrix. For low variation situations (Tables 8 and 9), the dynamic assignment is performing better than static assignments only when component evaluation takes at least 50 KFLOPS. This is true because there are more assigned components per processor for the 50×51 matrix than for the 11×12 Jacobian matrix. Thus the load balancing among the nodes is not as uneven. However, when component evaluation time is exponentially distributed (high variation), the dynamic assignment (Table 10) is performing better than static assignment even at the mean component cost of 15 KFLOPS.

Tables 5, 8, and 11 show the relative performances of the 4×8 rectangular mapping versus the 32-node ring linear mapping. It can be seen that as the matrix size increases the linear mapping becomes more competitive. Table 8 shows that the linear mapping will outperform the rectangular mapping if the computation can be reduced by half. Table 11 shows that for a 98×99 matrix the linear mapping will do better if the computations can be reduced by 20% and mean component complexity is more than 15 KFLOPS. This is in contrast to the 50×51 (Table 8)case where the component complexity has to be at least 50 KFLOPS for STATIC(L20) to do better than STATIC(R). These results are not exactly predictable if we consider the amount of computation that processor 0 has to perform (in either mapping processor 0 will have maximum load). When $n = 11$, processor 0 has to

Table 12. Execution time in secs (32 nodes).

n	SFER	PFER	SKER	PFUN	PKER
11	28.7	31.8	29.5	33.0	34.1
23	107.1	37.4	110.3	38.5	39.6
35	236.1	47.0	239.2	49.0	49.8
74	1053.0	76.8	1074.6	93.5	87.1
119	2850.0	130.8	3000.0	278.0	144.0

compute 6 components in the 4×8 rectangular mapping case whereas it has to compute as many as 11 components in the linear mapping by column case. When $n = 50$ and $n = 98$ these numbers are $(91, 100)$ and $(325, 392)$ respectively. This is because the computation per component is more even among the nodes when $n = 98$.

Table 12 shows some timing results for Jacobian matrix evaluation in combination with factorization and triangular solving for systems with small to medium Jacobian matrices. The matrices were obtained from a Galerkin approximation to a buoyant rotating disc fluid mechanics problem [26]. Each component was approximately 2 KFLOPS. However, they were not expensive enough to justify dynamic assignment, so a rectangular assignment was used. The Jacobian matrix was computed using central finite difference approximations. Thus there were $2(n^2+n)$ function component evaluations. Though there were no communications involved in function evaluations (except minor communication at the beginning and at the end), some computations were duplicated. This redundancy explains why a speedup close to 32 was not achieved. For this problem the function evaluation time dominated the unit tangent vector computation time and thus parallel function evaluations contributed most to the speedup.

When the system has a large enough ($n \geq 50$) Jacobian matrix, Jacobian matrix evaluation and kernel computation should all be done in parallel. If the system has a small but very expensive Jacobian matrix, then function and Jacobian matrix evaluation should

be done in parallel but the kernel computation may be done serially. This can be observed from Table 12, where for $n \leq 35$ PFUN is slightly better than PKER.

5. POLYNOMIAL SYSTEMS.

Chapter 2 described a homotopy algorithm for finding a single solution to a general nonlinear system of equations $F(x) = 0$. Proposition 1 provided the theoretical guarantee of convergence. The rich structure and multiple solutions of polynomial systems dictate that the general theory in Chapter 2 must be sharpened. This section discusses a globally convergent (with probability one) homotopy algorithm that finds *all* solutions to a polynomial system, and provides the theoretical justification for that algorithm.

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

$$F_i(x) = \sum_{k=1}^{n_i} a_{ik} \prod_{j=1}^{n} x_j^{d_{ijk}}, \quad i = 1, \ldots, n. \tag{5.1}$$

The ith component $F_i(x)$ has n_i terms, the a_{ik} are the (real) coefficients, and the degrees d_{ijk} are nonnegative integers. The total degree of F_i is $d_i = \max_k \sum_{j=1}^{n} d_{ijk}$. For technical reasons it is necessary to consider $F(x)$ as a map $F : C^n \rightarrow C^n$, where C^n is n-dimensional complex Euclidean space. A system of n polynomial equations in n unknowns may have many solutions. It is possible to define a homotopy so that all geometrically isolated solutions of (5.1) have at least one associated homotopy path. Generally, (5.1) will have solutions at infinity, which forces some of the homotopy paths to diverge to infinity as λ approaches 1. However, (5.1) can be transformed into a new system which, under reasonable hypotheses, can be proven to have no solutions at infinity and thus bounded homotopy paths. Because scaling can be critical to the success of the method, a general scaling algorithm [54] is applied to scale the coefficients and variables in (5.1) before anything else is done.

Since the homotopy map defined below is complex analytic, the homotopy parameter λ is monotonically increasing as a function of arc length [35]. The existence of an infinite

number of solutions or an infinite number of solutions at infinity does not destabilize the method. Some paths will converge to the higher dimensional solution components, and these paths will behave the way paths converging to any singular solution behave. Practical applications usually seek a subset of the solutions, rather than all solutions [34], [35]. However, the sort of generic homotopy algorithm considered here must find all solutions and cannot be limited without, in essence, changing it into a heuristic.

Define $G : C^n \to C^n$ by $G_j(x) = b_j x_j^{d_j} - a_j$, $j = 1, \ldots, n$, where a_j and b_j are nonzero complex numbers and d_j is the (total) degree of $F_j(x)$, for $j = 1, \ldots, n$. Define the homotopy map

$$\rho_c(\lambda, x) = (1 - \lambda)\, G(x) + \lambda\, F(x), \tag{5.2}$$

where $c = (a, b)$, $a = (a_1, \ldots, a_n) \in C^n$ and $b = (b_1, \ldots, b_n) \in C^n$. Let $d = d_1 \cdots d_n$ be the *total degree* of the system. The fundamental homotopy result, proved and discussed at length in [34]–[35], is:

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

A number of distinct homotopies have been proposed for solving polynomial systems. The homotopy map in (5.2) is from [35]. As with all such homotopies, there will be paths diverging to infinity if $F(x) = 0$ has solutions at infinity. These divergent paths are (at least) a nuisance, since they require arbitrary stopping criteria. Solutions at infinity can be avoided via the following projective transformation.

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

$$F'_j(y) = {y_{n+1}}^{d_j} F_j(y_1/y_{n+1}, \ldots, y_n/y_{n+1}), \qquad j = 1, \ldots, n. \tag{5.3}$$

The set of all lines through the origin in C^{n+1} is called complex projective n-space, denoted CP^n, and is a smooth compact (complex) n-dimensional manifold. The solutions of $F'(y) = 0$ in CP^n are identified with the finite solutions and solutions at infinity of $F(x) = 0$ in the usual way [54]. A basic result on the structure of the solution set of a polynomial system is the following classical theorem of Bezout [36]:

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

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

Define a linear function $u(y_1, \ldots, y_{n+1}) = \xi_1 y_1 + \xi_2 y_2 + \cdots + \xi_{n+1} y_{n+1}$ where $\xi_1, \ldots, \xi_{n+1}$ are nonzero complex numbers, and define $F'' : C^{n+1} \to C^{n+1}$ by

$$\begin{aligned} F''_j(y) &= F'_j(y), \qquad j = 1, \ldots, n, \\ F''_{n+1}(y) &= u(y) - 1. \end{aligned} \tag{5.4}$$

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

Theorem[36]. *If $F'(y) = 0$ has only a finite number of solutions in CP^n, then $F''(y) = 0$ has exactly d solutions (counting multiplicities) in C^{n+1} and no solutions at infinity, for almost all $\xi \in C^{n+1}$.*

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

The import of the above theory is that the nature of the zero curves of the projective transformation $F''(y)$ of $F(x)$ is as follows: There are exactly d (the total degree of F) zero curves; they are monotone in λ, and have finite arc length. *The homotopy algorithm is to track these d curves, which contain all isolated (transformed) zeros of F.*

5.1. Parallel algorithms.

Given that d homotopy paths are to be tracked, there are two extreme approaches when executing the homotopy algorithm in parallel. In one extreme, when the granularity is the coarsest possible, each individual processor tracks as many paths as possible until all the solutions for the polynomial system of equations have been found. The host processor reads in the data and initializes parameters (this includes the starting point for each path). It then distributes paths to each node keeping as many nodes as possible busy. When a node finishes tracking one path the host prints the result of that path and assigns a new path to that node. Since there is no *a priori* knowledge about the length of the path, the assignment is made on a first come first serve basis, i.e., paths are assigned in the order they are generated during the initialization process. However, this results in poor performance on those occasions when a few extremely long paths are tracked last. In this case most of

the processors will be sitting idle while a few processors will be tracking the long paths. If some knowledge about the length of the paths were available, the paths could be assigned in the decreasing order of their length. This would result in much better load balancing among the nodes. Omitting the tracking and initialization details of the algorithm, the coarse-grained parallel algorithm is:

FOR THE HOST:

(1) Initialize the data space and calculate a starting point for each path.

(2) SEND initializations and a starting point to a node.

(3) If the message in (2) is incomplete, go to (2).

(4) If another path needs to be assigned and a node is available, go to (2).

(5) Now wait for a message from a node.

(6) RECEIVE a "ready to transmit solution" message from a node (call it the "current" node).

(7) SEND an acknowledgement ("ready to receive" message) to the current node.

(8) If a "ready to transmit" message is received from another node, put the node identification into a queue until the current node completes transmitting a solution.

(9) RECEIVE a "solution" message from the current node.

(10) If the "solution" message is incomplete, go to (8).

(11) Process the solution sent by the current node and print it.

(12) If another path needs to be assigned, SEND initializations and a starting point to the current node.

(13) If the message in (12) is incomplete, go to (12).

(14) If any nodes are in the queue (see (8)), remove the first node from the queue, call it the current node and go to (7).

(15) If awaiting messages from any other nodes, go to (5).

(16) All paths have been assigned and all nodes have reported back, so STOP.

FOR EACH NODE:

(1) RECEIVE initializations and a starting point from the host.

(2) If the message in (1) is incomplete, go to (1).

(3) Track the path associated with the starting point.

(4) SEND a "ready to transmit solution" message to the host.

(5) RECEIVE a "ready to receive" message from the host.

(6) SEND the "solution" message to the host.

(7) If the message in (6) is incomplete, go to (6).

(8) Go to (1).

Note 1: The initialization and solution messages may be longer than permitted by the message buffer. If this is the case the information must be passed in multiple messages.

In the other extreme, where the granularity is the finest possible, the primary task of tracking the solutions is delegated to the host processor and only during the evaluation of the polynomial system and its Jacobian matrix is the work distributed among the nodes. It has been observed that in the serial version of the algorithm about 60% of the execution time is spent in evaluating these values. Thus in the finest granularity version about 60% of the serial algorithm is parallelized. However, one possible advantage of this approach is a better load balancing among the nodes. A high level description of the fine-grained algorithm is:

FOR THE HOST:

(1) SEND initializations and other parameters to all nodes.

(2) Start tracking all the paths.

(3) Continue tracking all the paths. If all paths are completed, then STOP. When a function needs to be evaluated at some point, SEND the location of the point and the index of the next row to the first available node.

(4) If all the rows have been assigned, go to (9).

(5) If all the nodes are busy, go to (7).

(6) Assign next row to next available node. Go to (4).

(7) Wait for a node to send the calculated values.

(8) RECEIVE the desired values from one node and go to (6).

(9) Wait for all the nodes to send their results back to the host.

(10) RECEIVE the desired values from all the nodes and then go to (3).

FOR EACH NODE:

(1) RECEIVE initializations and other parameters.

(2) Wait for host to send a point location and row index.

(3) RECEIVE the location of the point and the row index.

(4) Evaluate the functions and derivatives.

(5) SEND the results to the host and go to (2).

5.2. Computational results.

Polynomial systems of equations arise frequently in such diverse areas as computational geometry, robotics, chemical engineering, mechanical engineering, and computer vision. A small problem has total degree $d < 100$ and a large problem has $d > 1000$.

Table 13. Execution time (secs).

Problem number	total degree	Elxsi 6400 (serial)	Elxsi 6400 (coarse)	Elxsi 6400 (fine)	Alliant $FX/8$ (serial)	Alliant $FX/8$ (coarse)	Alliant $FX/8$ (fine)
102(4)	256	508	63	442	362	52	215
103(4)	625	1081	127	936	769	108	457
402(2)	4	10	7	11	6	2	4
403(2)	4	3	3	5	2	1	1
405(2)	64	124	26	107	96	16	55
601(2)	60	392	105	289	245	35	126
602(2)	60	769	135	558	793	148	406
603(2)	12	63	20	64	47	13	32
803(8)	256	6991	759	3045	4459	711	1642
1702(4)	16	50	15	37	36	9	16
1703(4)	16	50	15	37	36	9	16
1704(4)	16	50	11	34	35	7	15
1705(4)	81	426	53	267	308	46	134
5001(8)	576	17449	1829	9051	11579	1765	4736

Table 14. Execution time (secs).

Problem number	total degree	Encore Multimax	iPSC-16 (serial)	iPSC-16 (coarse)	iPSC-16 (fine)	iPSC-32 (coarse)
102(4)	256	1022	6480	541	11277	645
103(4)	625	2157	13753	1032	23839	1616
402(2)	4	21	108	35	198	54
403(2)	4	5	35	13	66	19
405(2)	64	287	1615	301	11277	335
601(2)	60	836	4045	383	4382	257
602(2)	60	2317	11782	1400	12283	2795
603(2)	12	133	869	195	1559	243
803(8)	256	10428	—	13750	—	11527
1702(4)	16	91	605	180	862	163
1703(4)	16	92	605	180	862	162
1704(4)	16	91	593	151	811	108
1705(4)	81	800	5302	463	5762	378
5001(8)	576	28969	—	14061	—	11786

Tables 13, 14 and 15 contain the results of a study designed to examine the granularity effects on an Intel iPSC hypercube and some other machines. The iPSC-32 was an 80286 based machine, while the iPSC-16 was a newer 80386 based system with special message

Table 15. Efficiency: [(serial time)/(parallel time)] / (number of processors used).

Problem number	total degree	Elxsi 6400 (coarse)	Elxsi 6400 (fine)	Alliant FX/8 (coarse)	Alliant FX/8 (fine)	iPSC–16 (coarse)	iPSC–16 (fine)
102(4)	256	.81	.29	.87	.42	.75	.14
103(4)	625	.85	.29	.89	.42	.83	.14
402(2)	4	.36	.40	.72	.83	.77	.27
403(2)	4	.25	.28	.65	.80	.67	.27
405(2)	64	.48	.58	.75	.87	.33	.32
601(2)	60	.37	.68	.88	.97	.66	.31
602(2)	60	.57	.69	.67	.98	.53	.24
603(2)	12	.32	.49	.45	.74	.37	.19
803(8)	256	.92	.29	.78	.34	—	—
1702(4)	16	.33	.34	.48	.54	.21	.18
1703(4)	16	.33	.34	.47	.54	.21	.18
1704(4)	16	.45	.37	.67	.57	.25	.18
1705(4)	81	.80	.40	.84	.58	.73	.23
5001(8)	576	.95	.24	.82	.31	—	—

routing hardware not available in the older system. Although this paper is mainly concerned with the results for the hypercube, the others are included for the sake of completeness. The problems are all real engineering problems in solid modelling, chemistry, and robotics that have arisen at General Motors and elsewhere. The problem number refers to an internal numbering scheme used at General Motors Research Laboratories; complete problem data is available on request. The number in parentheses is the number of equations n. The total degree refers to the number of paths d to be followed.

It can be seen from Tables 13, 14 and 15 that for the Intel iPSC the coarse grained parallel algorithm always outperforms the fine grained algorithm. It is also evident from Table 13 that for the Intel iPSC the performance of the fine grained parallel algorithm is worse than that of the serial version. This is due to the fact that the communication overhead in the fine grained version is greater than the amount of computation done in parallel. The quantum of computation done in each stage in evaluating the polynomial

system and its Jacobian matrix is small. From Table 14 it can be seen that the efficiency for the coarse grained algorithm is sometimes rather low. However, increasing the number of processors will almost always (whenever the number of processors is less than the number of paths) speedup the computation substantially. This happens because a relatively small polynomial system can have a reasonably large number of paths. All the paths can be tracked in parallel if a sufficient number of processors are available. This explains why the execution times of both the iPSC-32 and iPSC-16 are almost the same for problems 102, 103, 405, 803, 1705, and 5001 even though each node of the iPSC-32 is about half as fast as each node of the iPSC-16. In general, shared memory machines (Elxsi, Alliant) have many fewer processors than distributed memory machines. Thus for large problems the hypercube has a speedup advantage over a shared memory machine. As the total degree of the polynomial system increases, the efficiency of shared memory machines goes down significantly for the fine grained algorithm, possibly because of more memory contention. This is apparent from the results of problems 803 and 5001 which have efficiencies of .34 and .31 on the Alliant and .29 and .21 on the Elxsi. The serial version and the fine grained version on the hypercube take too long for these two problems and thus were not run.

As stated previously, the percentage of serial execution time that is spent in the evaluation of the polynomial system and its Jacobian matrix ranges from 50%-80%. The percentage depends on the complexity of the polynomial system. As the complexity increases the fraction that can be parallelized increases. This also increases the granule of parallelization and thus the ratio of communication overhead to computation carried out in parallel also decreases. This suggests that for certain classes of polynomial systems (complex function evaluation and large Jacobian matrix), the fine grained version can perform substantially

better than the serial version. In this case a mixed strategy can be employed. The coarse-grained algorithm can be used until there are no paths remaining to be tracked. Then the fine-grained algorithm can be used to finish the tracking of the uncompleted paths.

6. PARALLEL HOMPACK FOR DENSE SYSTEMS.

Chapters 3 to 5 discussed the parallelization of different components of a homotopy algorithm. This chapter integrates previous results and discusses the development of a parallel HOMPACK for systems with dense Jacobian matrices. Specifically, parallel versions of the subroutines FIXPDF, FIXPNF, and POLSYS are developed here. Hypercube versions of the routines FIXPDF, FIXPNF, and POLSYS are CFIXPDF, CFIXPNF, and CPOLSYS respectively.

Our objective is to build a software package that would let users execute different components of a homotopy algorithm either serially or in parallel depending on the problem type. The user can specify or let the program decide which components should be executed in parallel.

A variable PCODE should be set to 0 or 1 depending on whether or not the user wants full control. The routine CFIXPNF or CFIXPDF should be called with PCODE as one of its parameters. PCODE should be set to 0 if the user wants full control. When set to 0 the following integer array should be initialized.

CHOICE(1) = 0 if the Jacobian matrix should be evaluated serially,

1 if the Jacobian matrix should be evaluated in parallel,

CHOICE(2) = 0 if the factorization should be done serially,

1 if the factorization should be done in parallel.

If CHOICE (1) = 1 then

CHOICE(3) = 0 if a static rectangular mapping should be used,

1 if a static linear mapping should be used,

2 if a dynamic mapping should be used.

If PCODE = 1 then the following variables should be initialized:

AVGCST = average component complexity of the Jacobian matrix,

(measured in KFLOPS)

HIGH = .TRUE. if the variation in the component complexity is high

(approximately the ratio of standard deviation to mean is greater than .25).

CFIXPNF and CFIXPDF call the routine INTLZ (N, d_1, d_2, PCODE, CHOICE, AVGCST, HIGH, FNAME), where d_1 and d_2 are the row and column dimensions of the rectangular grid. Setting both d_1 and d_2 to zero will cause the program to use the default values $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil$, where d is the dimension of the cube. N is the dimension of the problem. FNAME is a CHARACTER*6 variable which stores the name of the node process. This should be initialized and passed to CFIXPNF and CFIXPDF. The routine INTLZ loads node processes and sets up data structures for execution. The user should link his routines FF and FJAC with file NODE.f to create an executable node program whose name is the value of FNAME. The name of the node program is passed as FNAME. The users can use shell file *link* to create both host and node executable files.

If PCODE = 1 then the program decides which components should be executed in parallel. It follows the following simple algorithm.

IF N < 50 THEN

DO factorization serially

ELSE

DO factorization in parallel

ENDIF

LOAD = n^2 * AVGCST

LOAD PER NODE = LOAD / NUMBER_OF_PROCESSORS

IF LOAD PER NODE $\geq$ 2.5 KFLOPS THEN

compute load for processor 0 for STATIC(R), STATIC(L)

and DYNAMIC assignments, estimate the efficiency

of each assignment and choose the best

ELSE

IF N $\geq$ 50 THEN

use STATIC (R) assignment

ELSE

DO serial function evaluation

ENDIF

ENDIF

If the routine CPOLSYS is called then the program always uses parallel path tracking. In this the paths are tracked independently using a master/slave paradigm (Chapter 5).

The calling structures of CFIXPNF, FF, RHO, FJAC and RHOJAC are:

CALL CFIXPNF (N,Y,IFLAG,ARCRE,ARCAE,ANSRE,ANSAE,TRACE,A,NFE,
ARCLEN,YP,YOLD,YPOLD,QR,ALPHA,TZ,PIVOT,W,WP,Z0,Z1,
SSPAR,PAR,IPAR,NUMI,NUMR,D_1,D_2,PCODE,CHOICE,AVGCST,
HIGH,WORK,IWORK)

CALL FF (N,I,X,F,IPAR,PAR)

CALL RHO (N,I,X,A,LAMBDA,F,IPAR,PAR)

CALL FJAC (N,I,J,X,DF,IPAR,PAR)

CALL RHOJAC (N,I,J,X,A,LAMBDA,DF,IPAR,PAR)

where D_1, D_2, PCODE, CHOICE, AVGCST, and HIGH are as described above. NUMI is the number of integer parameters passeed in IPAR. Similarly, NUMR is the number of

DOUBLE PRECISION parameters passed in PAR. IWORK is an work array and should be dimensioned at least max (NUMI,3). WORK is a work array and should be dimensioned at least $MAX((N^2+1)/NUMP+10+NUMP, 3*N+2)$, where NUMP is the number of processors. The rest of the variables are the same as those used in FIXPNF from HOMPACK. The routine FF evaluates the ith component of the function. RHO evaluates the ith component of the homotopy map. Similarly, FJAC and RHOJAC evaluate the (i,j)th component of the Jacobian matrix of the function and the homotopy map, respectively. FF, FJAC, RHO, and RHOJAC routines should be supplied by the user and linked with the node program. In the case of serial execution, they should be linked with the host program too. The source code for CFIXPNF is given in the appendices.

References.

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

[2] D.C.S. Allison, A. Chakraborty, and L. T. Watson, Granularity issues for solving polynomial systems via globally convergent algorithms on a Hypercube, *Proc. Third Conference on Hypercube Concurrent Computers and Applications,* Pasadena, CA (1988) 1463–1472.

[3] D. C. S. Allison, S. Harimoto, and L. T. Watson, The granularity of parallel homotopy algorithms for polynomial systems of equations, *Internat. J. Comput. Math.*, **29** (1989) 21–37.

[4] D. C. S. Allison, S. Harimoto, and L. T. Watson, The granularity of parallel homotopy algorithms for polynomial systems of equations, *Proc. 1988 Internat. Conf. on Parallel Processing,* Vol. III, D. H. Bailey (ed.), St. Charles, IL, 1988, 165–168.

[5] D. C. S. Allison, A. Chakraborty, and L. T. Watson, Granularity issues for solving polynomial systems via globally convergent algorithms on a hypercube, *J. Supercomputing,* **3** (1989) 5–20.

[6] E. Anderson and J. Dongarra, LAPACK Working Note 16: Results for the Initial Release of LAPACK, Tech. Rep. CS-89-89, University of Tennessee, 1989.

[7] R. E. Bank, PLTMG Users' Guide, Edition 4.0, Tech. Rep., University of California at San Diego, 1985

[8] M. Bieterman, Microtasking General Purpose Partial Differential Equation Software on the Cray X-MP, *J. of Supercomputing,* **2** (1988) 381–414.

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

[10] R. H. Byrd, C. L. Dert, A. H. G. Rinnooy Kan and R. B. Schnabel, Concurrent Stochastic Methods for Global Optimization, Tech. Rep. CU-CS-338-86, Dept. of Computer Science, University of Colorado, Boulder, Colorado 80309, 1986.

[11] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, Using parallel function evaluation to improve Hessian approximation for unconstrained optimization, Tech. Rep. CS-CU-361-87, Dept. of Computer Science, University of Colorado, Boulder, Colorado 80309, 1987.

[12] A. Chakraborty, D. C. S. Allison, C. J. Ribbens, and L. T. Watson, Parallel orthogonal decompositions of rectangular matrices for curve tracking on a hypercube, *Proc. Fourth Conf. on Hypercube Concurrent Computers and Applications,* J. Gustafson (ed.), ACM, Monterey, CA, 1989.

[13] A. Chakraborty, D. C. S. Allison, C. J. Ribbens, and L. T. Watson, Parallel Homotopy curve tracking on a hypercube, *Proc. 1989 Fourth SIAM Conference on Parallel Processing for Scientific Computing,* Chicago, 1989.

[14] A. Chakraborty, D. C. S. Allison, C. J. Ribbens, and L. T. Watson, Unit tangent vector computation for homotopy curve tracking on a hypercube, *Parallel Comput.,* Submitted.

[15] R.M. Chamberlain and M.J.D. Powell, QR factorization for linear least squares problems on the Hypercube, Tech. Rep. CCS 86/10, Dept. of Science and Technology, Christian Michelson Institute, Bergen, Norway, 1986.

[16] E. Chu and A. George, QR factorizations of a dense matrix on a Hypercube multiprocessor, Tech. Rep. ORNL/TM-10691, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

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

[18] G. A. De Biase, P. Ciucci and M. Cottone, Vectorized algorithms for astronomical image processing, *Parallel Computing,* **10** (1989) 339-346.

[19] J. J. Dongarra, J. J. DuCroz, I. S. Duff and S. Hammerling, A set of level 3 Basic Linear Algebra Subprograms, Tech memorandum 88, (May, revision 1), Argonne National Laboratory 1988.

[20] J. J. Dongarra, J. J. DuCroz, S. Hammerling and R. J. Hanson, ALGORITHM 656, an extended set of Basic Linear Algebra Subprograms: Model implementation and test programs, *ACM Trans. on Math. Software,* **14, 2** (1988) 18-32.

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

[22] S.C. Eisenstat, M.T. Heath, C.S. Henkel, and C.H. Romine, Modified cyclic algorithms for solving triangular systems on distributed memory multiprocessors, *SIAM J. Sci. Stat. Comput.,* **9** (1987) 589-600.

[23] R. A. Finkel, Large-grain Parallelism – Three case studies, *in The characteristics of parallel algorithms, L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds.,* (1987) 21–63.

[24] H. Fisher, Automatic differentiation: Parallel computation of function, gradient, and Hessian Matrix, *Parallel Computing,* **13** (1990) 101-110.

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

[26] S. Harimoto and L. T. Watson, The granularity of homotopy algorithms for polynomial systems of equations, *Parallel Processing for Scientific Computing,* G. Rodrigue (ed.), SIAM, Philadelphia, PA, 1989, 115–120.

[27] M.T. Heath and C.H. Romine, Parallel solution of triangular systems on distributed-memory multiprocessors, *SIAM J. Sci. Stat. Comput.,* **9** (1988) 558-588.

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

[29] E.N. Houstis, J.R. Rice, T. S. Papatheodorou, Parallel ELLPACK: An expert system for parallel processing of Partial differential equations, Tech. Rep. CSD-TR-831, Purdue University, 1988.

[30] D.R. Kincaid, J.R. Oppe, Adapting ITPACK routines for use on a vector computer, Tech. Rep. CNA-177, Center for Numerical Analysis, University of Texas, Austin, 1988.

[31] D. R. Kincaid and J. R. Oppe, ITPACK on Supercomputers, *Numerical Methods, Proceedings of the International Workshop,* V. Pereyra and A. Reinoza (ed.), Lecture Notes in Mathematics, Carcas 1982, 151–161.

[32] D. R. Kincaid, J. R. Respess, D. M. Young and R. G. Grimes, Algorithm 586 ITPACK 2C: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods, *ACM Trans. on Math. Software,* **8, 3** (1982) 302-322.

[33] G. Li and T.F. Coleman, A new method for solving triangular systems on distributed memory message-passing multiprocessors, *Nonlinear Anal.,* **13** (1989) 1339-1350

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

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

[36] ____, Solving polynomial systems using continuation for engineering and scientific problems, *Prentice-Hall, Englewood Cliffs, NJ, 1987.*

[37] A.P. Morgan, and L.T. Watson, A globally convergent parallel algorithm for zeros of polynomial systems, Tech. Rep. TR-86-25, Dept. of Computer Science, VPI&SU, Blacksburg, VA, 1986

[38] A.P. Morgan, and L.T. Watson, Solving polynomial systems of equations on a hypercube, *in Hypercube Multiprocessors 1987,* M. T. Heath, ed., SIAM, Philadelphia, PA, (1987) 501–511.

[39] A. P. Morgan and L. T. Watson, Solving nonlinear equations on a hypercube, *Super and Parallel Computers and Their Impact on Civil Engineering,* M. P. Kamat (ed.), ASCE Structures Congress '86, New Orleans, LA, 1986, 1–15.

[40] W. Pelz and L. T. Watson, Message length effects for solving polynomial systems on a hypercube *Tech. Rep. TR-86-26, Dept. of Computer Science, VPI&SU, VA, 1986.*

[41] A. Pothen, J. Somesh, and U. Vemulapati, Orthogonal factorizations on a distributed multiprocessor, *Proc. Hypercube Multiprocessors,* M. T. Heath, ed., SIAM, Philadelphia, PA, (1987) 587–596.

[42] A. Pothen and P. Raghaban, Distributed Orthogonal Factorizations, *Proc. Third Conference on Hypercube Concurrent Computers and Applications,* Pasadena, CA, (ACM, 1988) 1610–1620.

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

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

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

[46] A. H. G. Rinnooy Kan and G.T. Timmer, Stochastic global optimization methods – Part II: multi-level methods, Tech. Rep. 85401A, Econometric Institute, Erasmus University, The Netherlands, 1985.

[47] R. B. Schnabel, Concurrent function evaluations in local and global optimization, Tech. Rep. CS-CU-345-86, Dept. of Computer Science, Univ. of Colorado, Boulder, Colorado 80309, 1986.

[48] R. B. Schnabel, Sequential and parallel methods for unconstrained optimization, Tech. Rep. CU-CS-414-88, Dept. of Computer Science, Univ. of Colorado, Boulder, Colorado 80309, 1988.

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

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

[51] C.Y. Wang, Buoyant rotating disc, *manuscript and private communication.,* (1988)

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

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

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

[55] R. White, Parallel Algorithms for Nonlinear Prc blems, *SIAM J. Algebraic Discrete Methods,* **7** (1986) 137–149.

[56] R. White, A Nonlinear Parallel Algorithm with Application to the Stefan Problem, *SIAM J. Numer. Anal.,* **23** (1986) 639–652.

APPENDIX A

PARALLEL NORMAL FLOW ALGORITHM

```
      SUBROUTINE CFIXPNF(N,Y,IFLAG,ARCRE,ARCAE,ANSRE,ANSAE,TRACE,A,NFE,
     *   ARCLEN,YP,YOLD,YPOLD,QR,ALPHA,TZ,PIVOT,W,WP,Z0,Z1,SSPAR,
     *   PAR,IPAR,NUMI,NUMR,D1,D2,PCODE,CHOICE,AVGCST,HIGH
     *   ,WORK,IWORK,FNAME)
C
C SUBROUTINE  FIXPNF  FINDS A FIXED POINT OR ZERO OF THE
C N-DIMENSIONAL VECTOR FUNCTION F(X), OR TRACKS A ZERO CURVE
C OF A GENERAL HOMOTOPY MAP RHO(A,LAMBDA,X).  FOR THE FIXED
C POINT PROBLEM F(X) IS ASSUMED TO BE A C2 MAP OF SOME BALL
C INTO ITSELF.  THE EQUATION  X = F(X)  IS SOLVED BY
C FOLLOWING THE ZERO CURVE OF THE HOMOTOPY MAP
C
C  LAMBDA*(X - F(X)) + (1 - LAMBDA)*(X - A)  ,
C
C STARTING FROM LAMBDA = 0, X = A.  THE CURVE IS PARAMETERIZED
C BY ARC LENGTH S, AND IS FOLLOWED BY SOLVING THE ORDINARY
C DIFFERENTIAL EQUATION  D(HOMOTOPY MAP)/DS = 0  FOR
C Y(S) = (LAMBDA(S), X(S)) USING A HERMITE CUBIC PREDICTOR AND A
C CORRECTOR WHICH RETURNS TO THE ZERO CURVE ALONG THE FLOW NORMAL
C TO THE DAVIDENKO FLOW (WHICH CONSISTS OF THE INTEGRAL CURVES OF
C D(HOMOTOPY MAP)/DS ).
C
C FOR THE ZERO FINDING PROBLEM F(X) IS ASSUMED TO BE A C2 MAP
C SUCH THAT FOR SOME R > 0,  X*F(X) >= 0  WHENEVER NORM(X) = R.
C THE EQUATION  F(X) = 0  IS SOLVED BY FOLLOWING THE ZERO CURVE
C OF THE HOMOTOPY MAP
C
C   LAMBDA*F(X) + (1 - LAMBDA)*(X - A)
C
C EMANATING FROM LAMBDA = 0, X = A.
C
C  A  MUST BE AN INTERIOR POINT OF THE ABOVE MENTIONED BALLS.
C
C FOR THE CURVE TRACKING PROBLEM RHO(A,LAMBDA,X) IS ASSUMED TO
C BE A C2 MAP FROM E**M X [0,1) X E**N INTO E**N, WHICH FOR
C ALMOST ALL PARAMETER VECTORS A IN SOME NONEMPTY OPEN SUBSET
C OF E**M SATISFIES
C
C  RANK [D RHO(A,LAMBDA,X)/D LAMBDA , D RHO(A,LAMBDA,X)/DX] = N
C
C FOR ALL POINTS (LAMBDA,X) SUCH THAT RHO(A,LAMBDA,X)=0.  IT IS
C FURTHER ASSUMED THAT
C
C          RANK [ D RHO(A,0,X0)/DX ] = N  .
C
C WITH A FIXED, THE ZERO CURVE OF RHO(A,LAMBDA,X) EMANATING
C FROM  LAMBDA = 0, X = X0  IS TRACKED UNTIL  LAMBDA = 1  BY
C SOLVING THE ORDINARY DIFFERENTIAL EQUATION
C D RHO(A,LAMBDA(S),X(S))/DS = 0  FOR  Y(S) = (LAMBDA(S), X(S)),
C WHERE S IS ARC LENGTH ALONG THE ZERO CURVE.  ALSO THE HOMOTOPY
C MAP RHO(A,LAMBDA,X) IS ASSUMED TO BE CONSTRUCTED SUCH THAT
C
C            D LAMBDA(0)/DS > 0  .
C
C
C FOR THE FIXED POINT AND ZERO FINDING PROBLEMS, THE USER MUST SUPPLY
```

```
C A SUBROUTINE  F(X,V)  WHICH EVALUATES F(X) AT X AND RETURNS THE
C VECTOR F(X) IN V, AND A SUBROUTINE  FJAC(X,V,K)  WHICH RETURNS IN V
C THE KTH COLUMN OF THE JACOBIAN MATRIX OF F(X) EVALUATED AT X.  FOR
C THE CURVE TRACKING PROBLEM, THE USER MUST SUPPLY A SUBROUTINE
C  RHO(A,LAMBDA,X,V,PAR,IPAR)  WHICH EVALUATES THE HOMOTOPY MAP RHO AT
C (A,LAMBDA,X) AND RETURNS THE VECTOR RHO(A,LAMBDA,X) IN V, AND A
C SUBROUTINE  RHOJAC(A,LAMBDA,X,V,K,PAR,IPAR)  WHICH RETURNS IN V THE KTH
C COLUMN OF THE N X (N+1) JACOBIAN MATRIX [D RHO/D LAMBDA, D RHO/DX]
C EVALUATED AT (A,LAMBDA,X).  FIXPNF  DIRECTLY OR INDIRECTLY USES
C THE SUBROUTINES  STEPNF , TANGNF , ROOTNF , ROOT , F (OR  RHO ),
C FJAC (OR  RHOJAC ), D1MACH , AND THE BLAS FUNCTIONS  DDOT  AND
C DNRM2 .  ONLY  D1MACH  CONTAINS MACHINE DEPENDENT CONSTANTS.
C NO OTHER MODIFICATIONS BY THE USER ARE REQUIRED.
C
C
C ON INPUT:
C
C N  IS THE DIMENSION OF X, F(X), AND RHO(A,LAMBDA,X).
C
C Y  IS AN ARRRAY OF LENGTH  N + 1.  (Y(2),...,Y(N+1)) = A  IS THE
C    STARTING POINT FOR THE ZERO CURVE FOR THE FIXED POINT AND
C    ZERO FINDING PROBLEMS.  (Y(2),...,Y(N+1)) = X0  FOR THE CURVE
C    TRACKING PROBLEM.
C
C IFLAG  CAN BE -2, -1, 0, 2, OR 3.  IFLAG  SHOULD BE 0 ON THE
C    FIRST CALL TO  FIXPNF  FOR THE PROBLEM  X=F(X), -1 FOR THE
C    PROBLEM  F(X)=0, AND -2 FOR THE PROBLEM  RHO(A,LAMBDA,X)=0.
C    IN CERTAIN SITUATIONS  IFLAG  IS SET TO 2 OR 3 BY  FIXPNF,
C    AND  FIXPNF  CAN BE CALLED AGAIN WITHOUT CHANGING  IFLAG.
C
C ARCRE , ARCAE  ARE THE RELATIVE AND ABSOLUTE ERRORS, RESPECTIVELY,
C    ALLOWED THE NORMAL FLOW ITERATION ALONG THE ZERO CURVE.  IF
C    ARC?E .LE. 0.0  ON INPUT IT IS RESET TO  .5*SQRT(ANS?E) .
C    NORMALLY  ARC?E SHOULD BE CONSIDERABLY LARGER THAN  ANS?E .
C
C ANSRE , ANSAE  ARE THE RELATIVE AND ABSOLUTE ERROR VALUES USED FOR
C    THE ANSWER AT LAMBDA = 1.  THE ACCEPTED ANSWER  Y = (LAMBDA, X)
C    SATISFIES
C
C       |Y(1) - 1|  .LE.  ANSRE + ANSAE          .AND.
C
C       ||Z||  .LE.  ANSRE*||X|| + ANSAE         WHERE
C
C    (.,Z) IS THE NEWTON STEP TO Y.
C
C TRACE  IS AN INTEGER SPECIFYING THE LOGICAL I/O UNIT FOR
C    INTERMEDIATE OUTPUT.  IF  TRACE .GT. 0  THE POINTS COMPUTED ON
C    THE ZERO CURVE ARE WRITTEN TO I/O UNIT  TRACE .
C
C A(1:*)  CONTAINS THE PARAMETER VECTOR  A .  FOR THE FIXED POINT
C    AND ZERO FINDING PROBLEMS, A  NEED NOT BE INITIALIZED BY THE
C    USER, AND IS ASSUMED TO HAVE LENGTH  N.  FOR THE CURVE
C    TRACKING PROBLEM, A  MUST BE INITIALIZED BY THE USER.
C
C YP(1:N+1)  IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO
C    THE ZERO CURVE AT THE CURRENT POINT  Y .
```

```
C
C YOLD(1:N+1)  IS A WORK ARRAY CONTAINING THE PREVIOUS POINT FOUND
C    ON THE ZERO CURVE.
C
C YPOLD(1:N+1)  IS A WORK ARRAY CONTAINING THE TANGENT VECTOR TO
C    THE ZERO CURVE AT  YOLD .
C
C QR(1:N,1:N+2), ALPHA(1:N), TZ(1:N+1), PIVOT(1:N+1) , W(1:N+1) ,
C    WP(1:N+1) , ZO(1:N+1) , Z1(1:N+1)  ARE ALL WORK ARRAYS USED BY
C    STEPNF  TO CALCULATE THE TANGENT VECTORS AND NEWTON STEPS.
C
C SSPAR(1:8) = (LIDEAL, RIDEAL, DIDEAL, HMIN, HMAX, BMIN, BMAX, P)  IS
C    A VECTOR OF PARAMETERS USED FOR THE OPTIMAL STEP SIZE ESTIMATION.
C    IF  SSPAR(J) .LE. 0.0  ON INPUT, IT IS RESET TO A DEFAULT VALUE
C    BY  FIXPNF .  OTHERWISE THE INPUT VALUE OF  SSPAR(J)  IS USED.
C    SEE THE COMMENTS BELOW AND IN  STEPNF  FOR MORE INFORMATION ABOUT
C    THESE CONSTANTS.
C
C PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C NUMI : NUMBER OF PARAMETERS PASSED IN IPAR.
C NUMR : NUMBER OF PARAMETERS PASSED IN PAR.
C
C AVGCST : AVERAGE COMPONENT COMPLEXITY IN KFLOPS
C
C WORK(1..(N**+1)/NUMP+NUMP+10), DOUBLE PRECISION WORK ARRAY.
C
C IWORK(1..NUMI) INTEGER WORK ARRAY.
C
C PCODE = 0 IF EVERYTHING SHOULD BE DONE SERIALLY
C       = 1 OTHERWISE
C
C CHOICE(1..3) = CHOICE(1) = 0 IF SERIAL JACOBIAN EVALUATION
C                          = 1 IF PARALLEL JACOBIAN EVALUATION
C                CHOICE(2) = 0 SERIAL FACTORIZATION
C                          = 1 PARALLEL FACTORIZATION
C                CHOICE(3) = 0 RECTANGULAR MAPPING IN EVALUATION PHASE
C                          = 1 LINEAR MAPPING IN EVALUATION PHASE
C                          = 2 DYNAMIC EVALUATION OF JACOBIAN MATRIX
C
C FNAME : CHARACTER*6 FILE NAME OF THE EXECUTABLE NODE PROGRAM
C
C ON OUTPUT:
C
C N , TRACE , A  ARE UNCHANGED.
C
C Y(1) = LAMBDA, (Y(2),...,Y(N+1)) = X, AND Y IS AN APPROXIMATE
C    ZERO OF THE HOMOTOPY MAP.  NORMALLY LAMBDA = 1 AND X IS A
C    FIXED POINT(ZERO) OF F(X).  IN ABNORMAL SITUATIONS LAMBDA
C    MAY ONLY BE NEAR 1 AND X IS NEAR A FIXED POINT(ZERO).
C
C IFLAG =
C  -2  CAUSES  FIXPNF  TO INITIALIZE EVERYTHING FOR THE PROBLEM
C       RHO(A,LAMBDA,X) = 0 (USE ON FIRST CALL).
```

```
C
C  -1   CAUSES  FIXPNF  TO INITIALIZE EVERYTHING FOR THE PROBLEM
C       F(X) = 0 (USE ON FIRST CALL).
C
C   0   CAUSES  FIXPNF  TO INITIALIZE EVERYTHING FOR THE PROBLEM
C       X = F(X) (USE ON FIRST CALL).
C
C   1   NORMAL RETURN.
C
C   2   SPECIFIED ERROR TOLERANCE CANNOT BE MET.  SOME OR ALL OF
C       ARCRE , ARCAE , ANSRE , ANSAE  HAVE BEEN INCREASED TO
C       SUITABLE VALUES.  TO CONTINUE, JUST CALL  FIXPNF  AGAIN
C       WITHOUT CHANGING ANY PARAMETERS.
C
C   3   STEPNF  HAS BEEN CALLED 1000 TIMES.  TO CONTINUE, CALL
C       FIXPNF  AGAIN WITHOUT CHANGING ANY PARAMETERS.
C
C   4   JACOBIAN MATRIX DOES NOT HAVE FULL RANK.  THE ALGORITHM
C       HAS FAILED (THE ZERO CURVE OF THE HOMOTOPY MAP CANNOT BE
C       FOLLOWED ANY FURTHER).
C
C   5   THE TRACKING ALGORITHM HAS LOST THE ZERO CURVE OF THE
C       HOMOTOPY MAP AND IS NOT MAKING PROGRESS.  THE ERROR TOLERANCES
C       ARC?E  AND  ANS?E  WERE TOO LENIENT.  THE PROBLEM SHOULD BE
C       RESTARTED BY CALLING  FIXPNF  WITH SMALLER ERROR TOLERANCES
C       AND  IFLAG = 0 (-1, -2).
C
C   6   THE NORMAL FLOW NEWTON ITERATION IN  STEPNF  OR  ROOTNF
C       FAILED TO CONVERGE.  THE ERROR TOLERANCES  ANS?E  MAY BE TOO
C       STRINGENT.
C
C   7   ILLEGAL INPUT PARAMETERS, A FATAL ERROR.
C
C ARCRE , ARCAE , ANSRE , ANSAE  ARE UNCHANGED AFTER A NORMAL RETURN
C    (IFLAG = 1).  THEY ARE INCREASED TO APPROPRIATE VALUES ON THE
C    RETURN  IFLAG = 2 .
C
C NFE  IS THE NUMBER OF FUNCTION EVALUATIONS (= NUMBER OF
C    JACOBIAN EVALUATIONS).
C
C ARCLEN  IS THE LENGTH OF THE PATH FOLLOWED.
C
C
C
C
C
      DOUBLE PRECISION ABSERR,ANSAE,ANSRE,ARCAE,ARCLEN,ARCRE,
     1   CURSW,CURTOL,D1MACH,DNRM2,H,HOLD,RELERR,S,AVGCST
      INTEGER IFLAG,IFLAGC,ITER,JW,LIMIT,LIMITD,N,NC,NFE,NFEC,NP1,
     1   TRACE,D1,D2,PCODE,NUMI,NUMR
      CHARACTER*6 FNAME
      LOGICAL CRASH,POLSYS,START,HIGH,FIRST
C
C ***** ARRAY DECLARATIONS. *****
C
      DOUBLE PRECISION Y(N+1),YP(N+1),YOLD(N+1),YPOLD(N+1),A(N),
```

```
     *   QR(N,N+2),ALPHA(N),TZ(N+1),W(N+1),WP(N+1),ZO(N+1),
     *   Z1(N+1),SSPAR(8),PAR(*),
     *   WORK(*)
      INTEGER PIVOT(N+1),IPAR(*),CHOICE(*),IWORK(*)
C
C ***** END OF DIMENSIONAL INFORMATION. *****
C
      SAVE
C
C LIMITD  IS AN UPPER BOUND ON THE NUMBER OF STEPS.  IT MAY BE
C CHANGED BY CHANGING THE FOLLOWING PARAMETER STATEMENT:
      PARAMETER (LIMITD=1000)
C
C SWITCH FROM THE TOLERANCE  ARC?E  TO THE (FINER) TOLERANCE  ANS?E  IF
C THE CURVATURE OF ANY COMPONENT OF  Y  EXCEEDS  CURSW.
      PARAMETER (CURSW=10.0)
C
C
C
C : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
C SET LOGICAL SWITCH TO REFLECT ENTRY POINT.
      CALL INTLZ(N,D1,D2,PCODE,CHOICE,AVGCST,HIGH,FNAME)
      FIRST = .TRUE.
      POLSYS=.FALSE.
      GO TO 11
      ENTRY POLYNF(N,Y,IFLAG,ARCRE,ARCAE,ANSRE,ANSAE,TRACE,A,NFE,
     *   ARCLEN,YP,YOLD,YPOLD,QR,ALPHA,TZ,PIVOT,W,WP,ZO,Z1,SSPAR,
     *   PAR,IPAR)
      POLSYS=.TRUE.
11    CONTINUE
C
      IF (N .LE. 0  .OR.  ANSRE .LE. 0.0  .OR.  ANSAE .LT. 0.0)
     *                                              IFLAG=7
      IF (IFLAG .GE. -2  .AND.  IFLAG .LE. 0) GO TO 20
      IF (IFLAG .EQ. 2) GO TO 120
      IF (IFLAG .EQ. 3) GO TO 90
C ONLY VALID INPUT FOR  IFLAG  IS -2, -1, 0, 2, 3.
      IFLAG=7
      RETURN
C
C *****  INITIALIZATION BLOCK.  *****
C
20    ARCLEN=0.0
      IF (ARCRE .LE. 0.0) ARCRE=.5*SQRT(ANSRE)
      IF (ARCAE .LE. 0.0) ARCAE=.5*SQRT(ANSAE)
      NC=N
      NFEC=0
      IFLAGC=IFLAG
      NP1=N+1
C SET INITIAL CONDITIONS FOR FIRST CALL TO  STEPNF .
      START=.TRUE.
      CRASH=.FALSE.
      HOLD=1.0
      H=.1
      S=0.0
      YPOLD(1)=1.0
```

```
      YP(1)=1.0
      Y(1)=0.0
      DO 40 JW=2,NP1
        YPOLD(JW)=0.0
        YP(JW)=0.0
40    CONTINUE
C SET OPTIMAL STEP SIZE ESTIMATION PARAMETERS.
C LET Z[K] DENOTE THE NEWTON ITERATES ALONG THE FLOW NORMAL TO THE
C DAVIDENKO FLOW AND Y THEIR LIMIT.
C IDEAL CONTRACTION FACTOR:  ||Z[2] - Z[1]|| / ||Z[1] - Z[0]||
      IF (SSPAR(1) .LE. 0.0) SSPAR(1)= .5
C IDEAL RESIDUAL FACTOR:  ||RHO(A, Z[1])|| / ||RHO(A, Z[0])||
      IF (SSPAR(2) .LE. 0.0) SSPAR(2)= .01
C IDEAL DISTANCE FACTOR:  ||Z[1] - Y|| / ||Z[0] - Y||
      IF (SSPAR(3) .LE. 0.0) SSPAR(3)= .5
C MINIMUM STEP SIZE  HMIN .
      IF (SSPAR(4) .LE. 0.0) SSPAR(4)= (SQRT(N+1.0)+4.0)*D1MACH(4)
C MAXIMUM STEP SIZE  HMAX .
      IF (SSPAR(5) .LE. 0.0) SSPAR(5)= 1.0
C MINIMUM STEP SIZE REDUCTION FACTOR  BMIN .
      IF (SSPAR(6) .LE. 0.0) SSPAR(6)= .1
C MAXIMUM STEP SIZE EXPANSION FACTOR  BMAX .
      IF (SSPAR(7) .LE. 0.0) SSPAR(7)= 3.0
C ASSUMED OPERATING ORDER  P .
      IF (SSPAR(8) .LE. 0.0) SSPAR(8)= 2.0
C
C LOAD  A  FOR THE FIXED POINT AND ZERO FINDING PROBLEMS.
      IF (IFLAGC .GE. -1) THEN
        DO 60 JW=2,NP1
          A(JW-1)=Y(JW)
60      CONTINUE
      ENDIF
90    LIMIT=LIMITD
C
C *****  END OF INITIALIZATION BLOCK.  *****
C
C
C *****  MAIN LOOP.  *****
C
120   DO 400 ITER=1,LIMIT
      IF (Y(1) .LT. 0.0) THEN
        ARCLEN=S
        IFLAG=5
        RETURN
      ENDIF
C
C SET DIFFERENT ERROR TOLERANCE IF THE TRAJECTORY Y(S) HAS ANY HIGH
C CURVATURE COMPONENTS.
140   CURTOL=CURSW*HOLD
      RELERR=ARCRE
      ABSERR=ARCAE
      DO 160 JW=1,NP1
        IF (ABS(YP(JW)-YPOLD(JW)) .GT. CURTOL) THEN
          RELERR=ANSRE
          ABSERR=ANSAE
          GO TO 200
```

```
      ENDIF
160   CONTINUE
C
C TAKE A STEP ALONG THE CURVE.
200   CALL CSTEPNF(NC,NFEC,IFLAGC,START,CRASH,HOLD,H,RELERR,ABSERR,
     +     S,Y,YP,YOLD,YPOLD,A,QR,ALPHA,TZ,PIVOT,W,WP,Z0,Z1,SSPAR,
     +     PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
C PRINT LATEST POINT ON CURVE IF REQUESTED.
      IF (TRACE .GT. 0) THEN
WRITE (TRACE,217) ITER,NFEC,S,Y(1),(Y(JW),JW=2,NP1)
217     FORMAT(/' STEP',I5,3X,'NFE =',I5,3X,'ARC LENGTH =',F9.4,3X,
     *  'LAMBDA =',F7.4,5X,'X vector:'/1P,(1X,6E12.4))
      ENDIF
      NFE=NFEC
C CHECK IF THE STEP WAS SUCCESSFUL.
      IF (IFLAGC .GT. 0) THEN
        ARCLEN=S
        IFLAG=IFLAGC
        RETURN
      ENDIF
      IF (CRASH) THEN
C RETURN CODE FOR ERROR TOLERANCE TOO SMALL.
        IFLAG=2
C CHANGE ERROR TOLERANCES.
        IF (ARCRE .LT. RELERR) ARCRE=RELERR
        IF (ANSRE .LT. RELERR) ANSRE=RELERR
        IF (ARCAE .LT. ABSERR) ARCAE=ABSERR
        IF (ANSAE .LT. ABSERR) ANSAE=ABSERR
C CHANGE LIMIT ON NUMBER OF ITERATIONS.
        LIMIT=LIMIT-ITER
        RETURN
      ENDIF
C
      IF (Y(1) .GE. 1.0) THEN
C
C USE HERMITE CUBIC INTERPOLATION AND NEWTON ITERATION TO GET THE
C ANSWER AT LAMBDA = 1.0 .
C
C SAVE  YOLD  FOR ARC LENGTH CALCULATION LATER.
        DO 260 JW=1,NP1
          Z0(JW)=YOLD(JW)
260     CONTINUE
        CALL CROOTNF(NC,NFEC,IFLAGC,ANSRE,ANSAE,Y,YP,YOLD,YPOLD,A,QR,
     *    ALPHA,TZ,PIVOT,W,WP,PAR,IPAR,NUMI,NUMR,CHOICE,
     *    PCODE,WORK,IWORK,FIRST)
C
        NFE=NFEC
        IFLAG=1
C SET ERROR FLAG IF  ROOTNF  COULD NOT GET THE POINT ON THE ZERO
C CURVE AT  LAMBDA = 1.0  .
        IF (IFLAGC .GT. 0) IFLAG=IFLAGC
C CALCULATE FINAL ARC LENGTH.
        DO 290 JW=1,NP1
          W(JW)=Y(JW) - Z0(JW)
290     CONTINUE
        ARCLEN=S - HOLD + DNRM2(NP1,W,1)
```

```
      RETURN
     ENDIF
C
C FOR POLYNOMIAL SYSTEMS AND THE  POLSYS  HOMOTOPY MAP,
C D LAMBDA/DS .GE. 0 NECESSARILY.  THIS CONDITION IS FORCED HERE IF
C THE ENTRY POINT WAS  POLYNF .
C
     IF (POLSYS) THEN
       IF (YP(1) .LT. 0.0) THEN
C REVERSE TANGENT DIRECTION SO D LAMBDA/DS = YP(1) > 0 .
         DO 310 JW=1,NP1
           YP(JW)=-YP(JW)
           YPOLD(JW)=YP(JW)
310       CONTINUE
C FORCE  STEPNF  TO USE THE LINEAR PREDICTOR FOR THE NEXT STEP ONLY.
         START=.TRUE.
       ENDIF
     ENDIF
C
400   CONTINUE
C
C *****  END OF MAIN LOOP.  *****
C
C LAMBDA HAS NOT REACHED 1 IN 1000 STEPS.
     IFLAG=3
     ARCLEN=S
     RETURN
C
     END
     SUBROUTINE CROOTNF(N,NFE,IFLAG,RELERR,ABSERR,Y,YP,YOLD,YPOLD,
    *   A,QR,ALPHA,TZ,PIVOT,W,WP,PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,
    *   WORK,IWORK,FIRST)
C
C ROOTNF  FINDS THE POINT  YBAR = (1, XBAR)  ON THE ZERO CURVE OF THE
C HOMOTOPY MAP.  IT STARTS WITH TWO POINTS YOLD=(LAMBDAOLD,XOLD) AND
C Y=(LAMBDA,X) SUCH THAT  LAMBDAOLD < 1 <= LAMBDA , AND ALTERNATES
C BETWEEN HERMITE CUBIC INTERPOLATION AND NEWTON ITERATION UNTIL
C CONVERGENCE.
C
C ON INPUT:
C
C N = DIMENSION OF X AND THE HOMOTOPY MAP.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION IS
C    CONSIDERED TO HAVE CONVERGED WHEN A POINT Y=(LAMBDA,X) IS FOUND
C    SUCH THAT
C
C    |Y(1) - 1| <= RELERR + ABSERR             AND
C
C    ||Z|| <= RELERR*||X|| + ABSERR  ,          WHERE
C
C    (?,Z) IS THE NEWTON STEP TO Y=(LAMBDA,X).
```

```
C
C Y(1:N+1) = POINT (LAMBDA(S), X(S)) ON ZERO CURVE OF HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY MAP
C    AT  Y .
C
C YOLD(1:N+1) = A POINT DIFFERENT FROM  Y  ON THE ZERO CURVE.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C    MAP AT  YOLD .
C
C A(1:*) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:N,1:N+2), ALPHA(1:N), TZ(1:N+1), PIVOT(1:N+1), W(1:N+1),
C    WP(1:N+1)  ARE WORK ARRAYS USED FOR THE QR FACTORIZATION (IN THE
C    NEWTON STEP CALCULATION) AND THE INTERPOLATION:
C
C PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C ON OUTPUT:
C
C N , RELERR , ABSERR , A  ARE UNCHANGED.
C
C NFE  HAS BEEN UPDATED.
C
C IFLAG
C    = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C    = 4 IF A JACOBIAN MATRIX WITH RANK < N HAS OCCURRED.  THE
C       ITERATION WAS NOT COMPLETED.
C
C    = 6 IF THE ITERATION FAILED TO CONVERGE.  Y  AND  YOLD  CONTAIN
C       THE LAST TWO POINTS FOUND ON THE ZERO CURVE.
C
C Y  IS THE POINT ON THE ZERO CURVE OF THE HOMOTOPY MAP AT  LAMBDA = 1 .
C
C
C CALLS  D1MACH , DNRM2 , ROOT , TANGNF .
C
      DOUBLE PRECISION ABSERR,AERR,D1MACH,DD001,DD0011,DD01,DD011,
     *   DELS,DNRM2,F0,F1,FP0,FP1,QOFS,QSOUT,RELERR,RERR,S,SA,SB,
     *   SOUT,U
      INTEGER IFLAG,JUDY,JW,LCODE,LIMIT,N,NFE,NP1
      INTEGER NUMI,NUMR,PCODE
      LOGICAL FIRST
C
C ***** ARRAY DECLARATIONS. *****
C
      DOUBLE PRECISION Y(N+1),YP(N+1),YOLD(N+1),YPOLD(N+1),A(N),
     *   QR(N,N+2),ALPHA(N),TZ(N+1),W(N+1),WP(N+1),PAR(*)
      DOUBLE PRECISION WORK(*)
      INTEGER PIVOT(N+1),IPAR(*),CHOICE(*),IWORK(*)
C
C ***** END OF DIMENSIONAL INFORMATION. *****
```

```
C
C THE LIMIT ON THE NUMBER OF ITERATIONS ALLOWED MAY BE CHANGED BY
C CHANGING THE FOLLOWING PARAMETER STATEMENT:
      PARAMETER (LIMIT=20)
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDEI DIFFERENCES.
C
      DD01(F0,F1,DELS)=(F1-F0)/DELS
      DD001(F0,FP0,F1,DELS)=(DD01(F0,F1,DELS)-FP0)/DELS
      DD011(F0,F1,FP1,DELS)=(FP1-DD01(F0,F1,DELS))/DELS
      DD0011(F0,FP0,F1,FP1,DELS)=(DD011(F0,F1,FP1,DELS) -
     *                       DD001(F0,FP0,F1,DELS))/DELS
      QOFS(F0,FP0,F1,FP1,DELS,S)=((DD0011(F0,FP0,F1,FP1,DELS)*(S-DELS) +
     *   DD001(F0,FP0,F1,DELS))*S + FP0)*S + F0
C
C
      U=D1MACH(4)
      RERR=MAX(RELERR,U)
      AERR=MAX(ABSERR,0.0D0)
      NP1=N+1
C
C ***** MAIN LOOP. *****
C
100   DO 300 JUDY=1,LIMIT
      DO 110 JW=1,NP1
        TZ(JW)=Y(JW)-YOLD(JW)
110   CONTINUE
      DELS=DNRM2(NP1,TZ,1)
C
C USING TWO POINTS AND TANGENTS ON THE HOMOTOPY ZERO CURVE, CONSTRUCT
C THE HERMITE CUBIC INTERPOLANT Q(S). THEN USE ROOT TO FIND THE S
C CORRESPONDING TO LAMBDA = 1 . THE TWO POINTS ON THE ZERO CURVE ARE
C ALWAYS CHOSEN TO BRACKET LAMBDA=1, WITH THE BRACKETING INTERVAL
C ALWAYS BEING [0, DELS].
C
      SA=0.0
      SB=DELS
      LCODE=1
130   CALL ROOT(SOUT,QSOUT,SA,SB,RERR,AERR,LCODE)
      IF (LCODE .GT. 0) GO TO 140
      QSOUT=QOFS(YOLD(1),YPOLD(1),Y(1),YP(1),DELS,SOUT) - 1.0
      GO TO 130
C IF LAMBDA = 1 WERE BRACKETED, ROOT CANNOT FAIL.
140   IF (LCODE .GT. 2) THEN
        IFLAG=6
        RETURN
      ENDIF
C
C CALCULATE Q(SA) AS THE INITIAL POINT FOR A NEWTON ITERATION.
      DO 150 JW=1,NP1
        W(JW)=QOFS(YOLD(JW),YPOLD(JW),Y(JW),YP(JW),DELS,SA)
150   CONTINUE
C CALCULATE NEWTON STEP AT Q(SA).
      CALL CTANGNF(SA,W,WP,YPOLD,A,QR,ALPHA,TZ,PIVOT,NFE,N,IFLAG,
     *   PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
      IF (IFLAG .GT. 0) RETURN
```

```
C NEXT POINT = CURRENT POINT + NEWTON STEP.
      DO 160 JW=1,NP1
        W(JW)=W(JW)+TZ(JW)
160   CONTINUE
C GET THE TANGENT  WP  AT  W  AND THE NEXT NEWTON STEP IN  TZ .
      CALL CTANGNF(SA,W,WP,YPOLD,A,QR,ALPHA,TZ,PIVOT,NFE,N,IFLAG,
     *     PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
      IF (IFLAG .GT. 0) RETURN
C TAKE NEWTON STEP AND CHECK CONVERGENCE.
      DO 170 JW=1,NP1
        W(JW)=W(JW)+TZ(JW)
170   CONTINUE
      IF ((ABS(W(1)-1.0) .LE. RERR+AERR) .AND.
     *    (DNRM2(N,TZ(2),1) .LE. RERR*DNRM2(N,W(2),1)+AERR)) THEN
        DO 180 JW=1,NP1
          Y(JW)=W(JW)
180     CONTINUE
        RETURN
      ENDIF
C IF THE ITERATION HAS NOT CONVERGED, DISCARD ONE OF THE OLD POINTS
C SUCH THAT  LAMBDA = 1  IS STILL BRACKETED.
      IF ((YOLD(1)-1.0)*(W(1)-1.0) .GT. 0.0) THEN
        DO 200 JW=1,NP1
          YOLD(JW)=W(JW)
          YPOLD(JW)=WP(JW)
200     CONTINUE
      ELSE
        DO 210 JW=1,NP1
          Y(JW)=W(JW)
          YP(JW)=WP(JW)
210     CONTINUE
      ENDIF
300   CONTINUE
C
C ***** END OF MAIN LOOP. *****
C
C THE ALTERNATING OSCULATORY CUBIC INTERPOLATION AND NEWTON ITERATION
C HAS NOT CONVERGED IN  LIMIT  STEPS.  ERROR RETURN.
      IFLAG=6
      RETURN
      END
      SUBROUTINE CSTEPNF(N,NFE,IFLAG,START,CRASH,HOLD,H,RELERR,
     *   ABSERR,S,Y,YP,YOLD,YPOLD,A,QR,ALPHA,TZ,PIVOT,W,WP,
     *   Z0,Z1,SSPAR,PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK
     *   ,FIRST )
C
C  STEPNF  TAKES ONE STEP ALONG THE ZERO CURVE OF THE HOMOTOPY MAP
C USING A PREDICTOR-CORRECTOR ALGORITHM.  THE PREDICTOR USES A HERMITE
C CUBIC INTERPOLANT, AND THE CORRECTOR RETURNS TO THE ZERO CURVE ALONG
C THE FLOW NORMAL TO THE DAVIDENKO FLOW.  STEPNF  ALSO ESTIMATES A
C STEP SIZE H FOR THE NEXT STEP ALONG THE ZERO CURVE.  NORMALLY
C  STEPNF  IS USED INDIRECTLY THROUGH  FIXPNF , AND SHOULD BE CALLED
C DIRECTLY ONLY IF IT IS NECESSARY TO MODIFY THE STEPPING ALGORITHM'S
C PARAMETERS.
C
C ON INPUT:
```

```
C
C N = DIMENSION OF X AND THE HOMOTOPY MAP.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C START = .TRUE. ON FIRST CALL TO  STEPNF , .FALSE. OTHERWISE.
C
C HOLD = ||Y - YOLD||; SHOULD NOT BE MODIFIED BY THE USER.
C
C H = UPPER LIMIT ON LENGTH OF STEP THAT WILL BE ATTEMPTED.  H  MUST BE
C    SET TO A POSITIVE NUMBER ON THE FIRST CALL TO  STEPNF .
C    THEREAFTER  STEPNF  CALCULATES AN OPTIMAL VALUE FOR  H , AND  H
C    SHOULD NOT BE MODIFIED BY THE USER.
C
C RELERR, ABSERR = RELATIVE AND ABSOLUTE ERROR VALUES.  THE ITERATION IS
C    CONSIDERED TO HAVE CONVERGED WHEN A POINT W=(LAMBDA,X) IS FOUND
C    SUCH THAT
C
C    ||Z|| <= RELERR*||W|| + ABSERR  ,          WHERE
C
C    Z IS THE NEWTON STEP TO W=(LAMBDA,X).
C
C S = (APPROXIMATE) ARC LENGTH ALONG THE HOMOTOPY ZERO CURVE UP TO
C    Y(S) = (LAMBDA(S), X(S)).
C
C Y(1:N+1) = PREVIOUS POINT (LAMBDA(S), X(S)) FOUND ON THE ZERO CURVE OF
C    THE HOMOTOPY MAP.
C
C YP(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY MAP
C    AT  Y .
C
C YOLD(1:N+1) = A POINT BEFORE  Y  ON THE ZERO CURVE OF THE HOMOTOPY MAP.
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR TO THE ZERO CURVE OF THE HOMOTOPY
C    MAP AT  YOLD .
C
C A(1:*) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:N,1:N+2), ALPHA(1:N), TZ(1:N+1), PIVOT(1:N+1), W(1:N+1),
C    WP(1:N+1)  ARE WORK ARRAYS USED FOR THE QR FACTORIZATION (IN THE
C    NEWTON STEP CALCULATION) AND THE INTERPOLATION.
C
C Z0(1:N+1), Z1(1:N+1)  ARE WORK ARRAYS USED FOR THE ESTIMATION OF THE
C    NEXT STEP SIZE  H .
C
C SSPAR(1:8) = (LIDEAL, RIDEAL, DIDEAL, HMIN, HMAX, PMIN, BMAX, P)  IS
C    A VECTOR OF PARAMETERS USED FOR THE OPTIMAL STEP SIZE ESTIMATION.
C
C PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C ON OUTPUT:
C
```

```
C N , A , SSPAR  ARE UNCHANGED.
C
C NFE  HAS BEEN UPDATED.
C
C IFLAG
C    = -2, -1, OR 0 (UNCHANGED) ON A NORMAL RETURN.
C
C    = 4 IF A JACOBIAN MATRIX WITH RANK < N HAS OCCURRED.  THE
C       ITERATION WAS NOT COMPLETED.
C
C    = 6 IF THE ITERATION FAILED TO CONVERGE.  W  CONTAINS THE LAST
C       NEWTON ITERATE.
C
C START = .FALSE. ON A NORMAL RETURN.
C
C CRASH
C    = .FALSE. ON A NORMAL RETURN.
C
C    = .TRUE. IF THE STEP SIZE  H  WAS TOO SMALL.  H  HAS BEEN
C      INCREASED TO AN ACCEPTABLE VALUE, WITH WHICH  STEPNF  MAY BE
C      CALLED AGAIN.
C
C    = .TRUE. IF  RELERR  AND/OR  ABSERR  WERE TOO SMALL.  THEY HAVE
C      BEEN INCREASED TO ACCEPTABLE VALUES, WITH WHICH  STEPNF  MAY
C      BE CALLED AGAIN.
C
C HOLD = ||Y - YOLD||.
C
C H = OPTIMAL VALUE FOR NEXT STEP TO BE ATTEMPTED.  NORMALLY  H  SHOULD
C    NOT BE MODIFIED BY THE USER.
C
C RELERR, ABSERR  ARE UNCHANGED ON A NORMAL RETURN.
C
C S = (APPROXIMATE) ARC LENGTH ALONG THE ZERO CURVE OF THE HOMOTOPY MAP
C    UP TO THE LATEST POINT FOUND, WHICH IS RETURNED IN  Y .
C
C Y, YP, YOLD, YPOLD  CONTAIN THE TWO MOST RECENT POINTS AND TANGENT
C    VECTORS FOUND ON THE ZERO CURVE OF THE HOMOTOPY MAP.
C
C
C CALLS  D1MACH , DNRM2 , TANGNF .
C
      DOUBLE PRECISION ABSERR,D1MACH,DCALC,DD001,DD0011,DD01,
     *   DD011,DELS,DNRM2,F0,F1,FOURU,FP0,FP1,H,HFAIL,HOLD,HT,
     *   LCALC,QOFS,RCALC,RELERR,RHOLEN,S,TEMP,TWOU
      INTEGER IFLAG,ITNUM,J,JUDY,LITFH,N,NFE,NP1,PCODE
      LOGICAL CRASH,FAIL,START,FIRST
C
C ***** ARRAY DECLARATIONS. *****
C
      DOUBLE PRECISION Y(N+1),YP(N+1),YOLD(N+1),YPOLD(N+1),A(N),
     *   QR(N,N+2),ALPHA(N),TZ(N+1),W(N+1),WP(N+1),Z0(N+1),
     *   Z1(N+1),SSPAR(8),PAR(1),WORK(*)
      INTEGER PIVOT(N+1),IPAR(1),IWORK(*),CHOICE(*)
C
C ***** END OF DIMENSIONAL INFORMATION. *****
```

```
C
C THE LIMIT ON THE NUMBER OF NEWTON ITERATIONS ALLOWED BEFORE REDUCING
C THE STEP SIZE  H  MAY BE CHANGED BY CHANGING THE FOLLOWING PARAMETER
C STATEMENT:
      PARAMETER (LITFH=4)
C
C DEFINITION OF HERMITE CUBIC INTERPOLANT VIA DIVIDED DIFFERENCES.
C
      DD01(F0,F1,DELS)=(F1-F0)/DELS
      DD001(F0,FP0,F1,DELS)=(DD01(F0,F1,DELS)-FP0)/DELS
      DD011(F0,F1,FP1,DELS)=(FP1-DD01(F0,F1,DELS))/DELS
      DD0011(F0,FP0,F1,FP1,DELS)=(DD011(F0,F1,FP1,DELS) -
     *                        DD001(F0,FP0,F1,DELS))/DELS
      QOFS(F0,FP0,F1,FP1,DELS,S)=((DD0011(F0,FP0,F1,FP1,DELS)*(S-DELS) +
     *   DD001(F0,FP0,F1,DELS))*S + FP0)*S + F0
C
C
      TWOU=2.0*D1MACH(4)
      FOURU=TWOU+TWOU
      NP1=N+1
      CRASH=.TRUE.
C THE ARCLENGTH  S  MUST BE NONNEGATIVE.
      IF (S .LT. 0.0) RETURN
C IF STEP SIZE IS TOO SMALL, DETERMINE AN ACCEPTABLE ONE.
      IF (H .LT. FOURU*(1.0+S)) THEN
        H=FOURU*(1.0+S)
        RETURN
      ENDIF
C IF ERROR TOLERANCES ARE TOO SMALL, INCREASE THEM TO ACCEPTABLE VALUES.
      TEMP=DNRM2(NP1,Y,1)
      IF (.5*(RELERR*TEMP+ABSERR) .GE. TWOU*TEMP) GO TO 40
      IF (RELERR .NE. 0.0) THEN
        RELERR=FOURU*(1.0+FOURU)
        ABSERR=MAX(ABSERR,0.0D0)
      ELSE
        ABSERR=FOURU*TEMP
      ENDIF
      RETURN
 40   CRASH=.FALSE.
      IF (.NOT. START) GO TO 300
C
C *****  STARTUP SECTION(FIRST STEP ALONG ZERO CURVE.  *****
C
      FAIL=.FALSE.
      START=.FALSE.
C DETERMINE SUITABLE INITIAL STEP SIZE.
      H=MIN(H, .10D0, SQRT(SQRT(RELERR*TEMP+ABSERR)))
C USE LINEAR PREDICTOR ALONG TANGENT DIRECTION TO START NEWTON ITERATION.
      YPOLD(1)=1.0
      DO 50 J=2,NP1
        YPOLD(J)=0.0
 50   CONTINUE
      CALL CTANGNF(S,Y,YP,YPOLD,A,QR,ALPHA,TZ,PIVOT,NFE,N,IFLAG,
     *    PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
      IF (IFLAG .GT. 0) RETURN
 70   DO 80 J=1,NP1
```

```
      TEMP=Y(J) + H * YP(J)
      W(J)=TEMP
      Z0(J)=TEMP
 80   CONTINUE
     DO 200 JUDY=1,LITFH
      RHOLEN=-1.0
C CALCULATE THE NEWTON STEP  TZ  AT THE CURRENT POINT  W .
      CALL CTANGNF(RHOLEN,W,WP,YPOLD,A,QR,ALPHA,TZ,PIVOT,NFE,N,IFLAG,
    *         PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
      IF (IFLAG .GT. 0) RETURN
C
C TAKE NEWTON STEP AND CHECK CONVERGENCE.
      DO 90 J=1,NP1
        W(J)=W(J) + TZ(J)
 90     CONTINUE
      ITNUM=JUDY
C COMPUTE QUANTITIES USED FOR OPTIMAL STEP SIZE ESTIMATION.
      IF (JUDY .EQ. 1) THEN
        LCALC=DNRM2(NP1,TZ,1)
        RCALC=RHOLEN
        DO 110 J=1,NP1
          Z1(J)=W(J)
110      CONTINUE
      ELSE IF (JUDY .EQ. 2) THEN
        LCALC=DNRM2(NP1,TZ,1)/LCALC
        RCALC=RHOLEN/RCALC
      ENDIF
C GO TO WOP-UP SECTION AFTER CONVERGENCE.
      IF (DNRM2(NP1,TZ,1) .LE. RELERR*DNRM2(NP1,W,1)+ABSERR)
    *                                          GO TO 600
C
200   CONTINUE
C
C NO CONVERGENCE IN  LITFH  ITERATIONS.  REDUCE  H  AND TRY AGAIN.
     IF (H .LE. FOURU*(1.0 + S)) THEN
       IFLAG=6
       RETURN
     ENDIF
     H=.5 * H
     GO TO 70
C
C ***** END OF STARTUP SECTION. *****
C
C ***** PREDICTOR SECTION. *****
C
300   FAIL=.FALSE.
C COMPUTE POINT PREDICTED BY HERMITE INTERPOLANT.  USE STEP SIZE  H
C COMPUTED ON LAST CALL TO  STEPNF .
320   DO 330 J=1,NP1
      TEMP=QOFS(YOLD(J),YPOLD(J),Y(J),YP(J),HOLD,HOLD+H)
      W(J)=TEMP
      Z0(J)=TEMP
330   CONTINUE
C
C ***** END OF PREDICTOR SECTION. *****
C
```

```
C ***** CORRECTOR SECTION. *****
C
     DO 500 JUDY=1,LITFH
       RHOLEN=-1.0
C CALCULATE THE NEWTON STEP  TZ  AT THE CURRENT POINT  W .
       CALL CTANGNF(RHOLEN,W,WP,YP,A,QR,ALPHA,TZ,PIVOT,NFE,N,IFLAG,
     *          PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,WORK,IWORK,FIRST)
       IF (IFLAG .GT. 0) RETURN
C
C TAKE NEWTON STEP AND CHECK CONVERGENCE.
       DO 420 J=1,NP1
         W(J)=W(J) + TZ(J)
420     CONTINUE
       ITNUM=JUDY
C COMPUTE QUANTITIES USED FOR OPTIMAL STEP SIZE ESTIMATION.
       IF (JUDY .EQ. 1) THEN
         LCALC=DNRM2(NP1,TZ,1)
         RCALC=RHOLEN
         DO 440 J=1,NP1
           Z1(J)=W(J)
440      CONTINUE
       ELSE IF (JUDY .EQ. 2) THEN
         LCALC=DNRM2(NP1,TZ,1)/LCALC
         RCALC=RHOLEN/RCALC
       ENDIF
C GO TO MOP-UP SECTION AFTER CONVERGENCE.
       IF (DNRM2(NP1,TZ,1) .LE. RELERR*DNRM2(NP1,W,1)+ABSERR)
     *                                          GO TO 600
C
500   CONTINUE
C
C NO CONVERGENCE IN  LITFH  ITERATIONS.  RECORD FAILURE AT CALCULATED  H ,
C SAVE THIS STEP SIZE, REDUCE  H  AND TRY AGAIN.
      FAIL=.TRUE.
      HFAIL=H
      IF (H .LE. FOURU*(1.0 + S)) THEN
        IFLAG=6
        RETURN
      ENDIF
      H=.5 * H
      GO TO 320
C
C ***** END OF CORRECTOR SECTION. *****
C
C ***** MOP-UP SECTION. *****
C
C YOLD  AND  Y  ALWAYS CONTAIN THE LAST TWO POINTS FOUND ON THE ZERO
C CURVE OF THE HOMOTOPY MAP.  YPOLD  AND  YP  CONTAIN THE TANGENT
C VECTORS TO THE ZERO CURVE AT  YOLD  AND  Y , RESPECTIVELY.
C
600   DO 620 J=1,NP1
        YOLD(J)=Y(J)
        YPOLD(J)=YP(J)
        Y(J)=W(J)
        YP(J)=WP(J)
        W(J)=Y(J) - YOLD(J)
```

```
620   CONTINUE
C UPDATE ARC LENGTH.
      HOLD=DNRM2(NP1,W,1)
      S=S+HOLD
C
C ***** END OF MOP-UP SECTION. *****
C
C ***** OPTIMAL STEP SIZE ESTIMATION SECTION. *****
C
C CALCULATE THE DISTANCE FACTOR  DCALC .
700   DO 710 J=1,NP1
        TZ(J)=Z0(J) - Y(J)
        W(J)=Z1(J) - Y(J)
710   CONTINUE
      DCALC=DNRM2(NP1,TZ,1)
      IF (DCALC .NE. 0.0) DCALC=DNRM2(NP1,W,1)/DCALC
C
C THE OPTIMAL STEP SIZE HBAR IS DEFINED BY
C
C   HT=HOLD * [MIN(LIDEAL/LCALC, RIDEAL/RCALC, DIDEAL/DCALC)]**(1/P)
C
C     HBAR = MIN [ MAX(HT, BMIN*HOLD, HMIN), BMAX*HOLD, HMAX ]
C
C IF CONVERGENCE HAD OCCURRED AFTER 1 ITERATION, SET THE CONTRACTION
C FACTOR  LCALC  TO ZERO.
      IF (ITNUM .EQ. 1) LCALC = 0.0
C FORMULA FOR OPTIMAL STEP SIZE.
      IF (LCALC+RCALC+DCALC .EQ. 0.0) THEN
        HT = SSPAR(7) * HOLD
      ELSE
        HT = (1.0/MAX(LCALC/SSPAR(1), RCALC/SSPAR(2), DCALC/SSPAR(3)))
     *       **(1.0/SSPAR(8)) * HOLD
      ENDIF
C  HT  CONTAINS THE ESTIMATED OPTIMAL STEP SIZE.  NOW PUT IT WITHIN
C REASONABLE BOUNDS.
      H=MIN(MAX(HT,SSPAR(6)*HOLD,SSPAR(4)), SSPAR(7)*HOLD, SSPAR(5))
      IF (ITNUM .EQ. 1) THEN
C IF CONVERGENCE HAD OCCURRED AFTER 1 ITERATION, DON'T DECREASE  H .
        H=MAX(H,HOLD)
      ELSE IF (ITNUM .EQ. LITFH) THEN
C IF CONVERGENCE REQUIRED THE MAXIMUM  LITFH  ITERATIONS, DON'T
C INCREASE  H .
        H=MIN(H,HOLD)
      ENDIF
C IF CONVERGENCE DID NOT OCCUR IN  LITFH  ITERATIONS FOR A PARTICULAR
C H = HFAIL , DON'T CHOOSE THE NEW STEP SIZE LARGER THAN  HFAIL .
      IF (FAIL) H=MIN(H,HFAIL)
C
C
      RETURN
      END
      SUBROUTINE CTANGNF(RHOLEN,Y,YP,YPOLD,A,QR,ALPHA,TZ,PIVOT,
     * NFE,N,IFLAG,PAR,IPAR,NUMI,NUMR,CHOICE,PCODE,RBUFF,IBUFF,
     *  FIRST)
C
C THIS SUBROUTINE BUILDS THE JACOBIAN MATRIX OF THE HOMOTOPY MAP,
```

```
C COMPUTES A QR DECOMPOSITION OF THAT MATRIX, AND THEN CALCULATES THE
C (UNIT) TANGENT VECTOR AND THE NEWTON STEP.
C
C ON INPUT:
C
C RHOLEN < 0 IF THE NORM OF THE HOMOTOPY MAP EVALUATED AT
C    (A, LAMBDA, X) IS TO BE COMPUTED.  IF  RHOLEN >= 0  THE NORM IS NOT
C    COMPUTED AND  RHOLEN  IS NOT CHANGED.
C
C Y(1:N+1) = CURRENT POINT (LAMBDA(S), X(S)).
C
C YPOLD(1:N+1) = UNIT TANGENT VECTOR AT PREVIOUS POINT ON THE ZERO
C    CURVE OF THE HOMOTOPY MAP.
C
C A(1:*) = PARAMETER VECTOR IN THE HOMOTOPY MAP.
C
C QR(1:N,1:N+2), ALPHA(1:N), TZ(1:N+1), PIVOT(1:N+1)  ARE WORK ARRAYS
C    USED FOR THE QR FACTORIZATION.
C
C NFE = NUMBER OF JACOBIAN MATRIX EVALUATIONS = NUMBER OF HOMOTOPY
C    FUNCTION EVALUATIONS.
C
C N = DIMENSION OF X.
C
C IFLAG = -2, -1, OR 0, INDICATING THE PROBLEM TYPE.
C
C NUMI = NUMBER OF INTEGER PARAMETER (SEE IPAR())
C NUMR = NUMBER OF DOUBLE PRECISION PARAMETER
C PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C    WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN SUBROUTINES
C    RHO, RHOJAC.
C
C PCODE = 0 IF EVERYTHING SHOULD BE DONE SERIALLY
C       = 1 OTHERWISE
C
C CHOICE(1..3) = CHOICE(1) = 0 IF SERIAL JACOBIAN EVALUATION
C                          = 1 IF PARALLEL JACOBIAN EVALUATION
C                CHOICE(2) = 0 SERIAL FACTORIZATION
C                          = 1 PARALLEL FACTORIZATION
C                CHOICE(3) = 0 RECTANGULAR MAPPING IN EVALUATION PHASE
C                          = 1 LINEAR MAPPING IN EVALUATION PHASE
C                          = 2 DYNAMIC EVALUATION OF JACOBIAN MATRIX
C
C
C ON OUTPUT:
C
C RHOLEN = ||RHO(A, LAMBDA(S), X(S)|| IF  RHOLEN < 0  ON INPUT.
C    OTHERWISE  RHOLEN  IS UNCHANGED.
C
C Y, YPOLD, A, N  ARE UNCHANGED.
C
C YP(1:N+1) = DY/DS = UNIT TANGENT VECTOR TO INTEGRAL CURVE OF
C    D(HOMOTOPY MAP)/DS = 0  AT  Y(S) = (LAMBDA(S), X(S)) .
C
C TZ = THE NEWTON STEP = -(PSEUDO INVERSE OF  (D RHO(A,Y(S))/D LAMBDA ,
C    D RHO(A,Y(S))/DX)) * RHO(A,Y(S)) .
```

```
C
C NFE  HAS BEEN INCRMENTED BY 1.
C
C IFLAG  IS UNCHANGED, UNLESS THE QR FACTORIZATION DETECTS A RANK < N,
C    IN WHICH CASE THE TANGENT AND NEWTON STEP VECTORS ARE NOT COMPUTED
C    AND  TANGNF  RETURNS WITH  IFLAG = 4 .
C
C
C CALLS  DDOT , DNRM2 , F (OR  RHO ), FJAC (OR  RHOJAC ).
C
      DOUBLE PRECISION ALPHAK,BETA,DDOT,DNRM2,LAMBDA,QRKK,RHOLEN,
     *       SIGMA,SUM,YPNORM
      INTEGER I,IFLAG,J,JBAR,K,KP1,N,NFE,NP1,NP2
      LOGICAL FIRST
C
C *****  ARRAY DECLARATIONS.  *****
C
      DOUBLE PRECISION Y(N+1),YP(N+1),YPOLD(N+1),A(N),PAR(1),RBUFF(*)
      INTEGER IPAR(*),PCODE,CHOICE(*),NUMI,NUMR,IBUFF(*)
C
C ARRAYS FOR COMPUTING THE JACOBIAN MATRIX AND ITS KERNEL.
      DOUBLE PRECISION QR(N,N+2),ALPHA(N),TZ(N+1)
      INTEGER PIVOT(N+1)
C
C *****  END OF DIMENSIONAL INFORMATION.  *****
C
C
      LAMBDA=Y(1)
      NP1=N+1
      NP2=N+2
      NFE=NFE+1
      IF (PCODE.EQ. 1) THEN
        IF (CHOICE(1) .EQ. 1) THEN
C
C   OK, PARALLEL EVALUATION. SEE WHICH METHOD SHOULD BE USED
C
        IF (CHOICE(3) .EQ. 0) THEN
C
C   RECTANGULAR MAPPING
C
C       WRITE(6,*)' LAMBDA ',Y(1)
        CALL HNFVAR(QR,QR(1,N+2),Y,N,NUMI,NUMR,YP,TZ,IFLAG,A
     1            ,PAR, IPAR,CHOICE,FIRST,IBUFF,RBUFF)
        ELSE
          IF (CHOICE(3) .EQ. 1) THEN
C
C LINEAR MAPPING
C
          CALL HNFVAL(QR,QR(1,N+2),Y,N,NUMI,NUMR,YP,TZ,IFLAG,A,
     1                PAR,IPAR,CHOICE,FIRST,IBUFF,RBUFF)
          ELSE
C
C   OK, DYNAMIC MAPPING SHOULD BE USED
C
           CALL HNFVAD(QR,QR(1,N+2),Y,N,NUMI,NUMR,YP,TZ,IFLAG,A,
     1              PAR,IPAR,CHOICE,FIRST,IBUFF,RBUFF)
```

```
         ENDIF
       ENDIF
      ENDIF
      ENDIF
      IF (PCODE .EQ. 0 .OR. CHOICE(1) .EQ. 0) THEN
C
C     SERIAL EVALUATION
C
C NFE CONTAINS THE NUMBER OF JACOBIAN EVALUATIONS.
C   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
C
C COMPUTE THE JACOBIAN MATRIX, STORE IT AND HOMOTOPY MAP IN QR.
C
     IF (IFLAG .EQ. -2) THEN
C
C  QR = ( D RHO(A,LAMBDA,X)/D LAMBDA , D RHO(A,LAMBDA,X)/DX ,
C                                       RHO(A,LAMBDA,X) )  .
C
      DO 30 K=1,NP1
        DO 30 KK = 1,N
         CALL RHOJAC(N,KK,K,Y(2),A,LAMBDA,QR(KK,K),K,IPAR,PAR)
30     CONTINUE
       DO 891 KK = 1,N
891     CALL RHO(N,KK,Y(2),A,LAMBDA,QR(KK,NP2),IPAR,PAR)
     ELSE
       DO 890 KK = 1,N
890     CALL FF(N,KK,Y(2),TZ(KK),IPAR,PAR)
       IF (IFLAG .EQ. 0) THEN
C
C     QR = ( A - F(X), I - LAMBDA*DF(X) ,
C                          X - A + LAMBDA*(A - F(X)) )  .
C
        DO 100 J=1,N
          SIGMA=A(J)
          BETA=SIGMA-TZ(J)
          QR(J,1)=BETA
100      QR(J,NP2)=Y(J+1)-SIGMA+LAMBDA*BETA
        DO 120 K=1,N
          DO 121 KK = 1,N
121        CALL FJAC(N,KK,K,Y(2),TZ(KK),IPAR,PAR)
          KP1=K+1
          DO 110 J=1,N
110        QR(J,KP1)=-LAMBDA*TZ(J)
120      QR(K,KP1)=1.0+QR(K,KP1)
      ELSE
C
C   QR = ( F(X) - X + A, LAMBDA*DF(X) + (1 - LAMBDA)*I ,
C                           X - A + LAMBDA*(F(X) - X + A) )  .
C
140        DO 150 J=1,N
            SIGMA=Y(J+1)-A(J)
            BETA=TZ(J)-SIGMA
            QR(J,1)=BETA
150        QR(J,NP2)=SIGMA+LAMBDA*BETA
           DO 170 K=1,N
            CALL FJAC(Y(2),TZ,K)
```

```
            KP1=K+1
          DO 160 J=1,N
160          QR(J,KP1)=LAMBDA*TZ(J)
170         QR(K,KP1)=1.0-LAMBDA+QR(K,KP1)
        ENDIF
       ENDIF
     ENDIF
C     DO 320 KKK = 1,N
C       WRITE(6,321)(QR(KKK,JJJ),JJJ=1,N+2)
C320   CONTINUE
 321   FORMAT(1x,4(1X,E23.16))
C
C  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
C COMPUTE THE NORM OF THE HOMOTOPY MAP IF IT WAS REQUESTED.
     IF (RHOLEN .LT. 0.0) RHOLEN=DNRM2(N,QR(1,NP2),1)
C
     IF (CHOICE(2) .EQ. 0) THEN
C
C  SERIAL FACTORIZATION
C
C REDUCE THE JACOBIAN MATRIX TO UPPER TRIANGULAR FORM.
C
C THE FOLLOWING CODE IS A MODIFICATION OF THE ALGOL PROCEDURE
C DECOMPOSE  IN P. BUSINGER AND G. H. GOLUB, LINEAR LEAST
C SQUARES SOLUTIONS BY HOUSEHOLDER TRANSFORMATIONS,
C NUMER. MATH. 7 (1965) 269-276.
C
        DO 220 J=1,NP1
          YP(J)=DDOT(N,QR(1,J),1,QR(1,J),1)
220      PIVOT(J)=J
        DO 300 K=1,N
          SIGMA=YP(K)
          JBAR=K
          KP1=K+1
          DO 240 J=KP1,NP1
            IF (SIGMA .GE. YP(J)) GO TO 240
            SIGMA=YP(J)
            JBAR=J
240        CONTINUE
          IF (JBAR .EQ. K) GO TO 260
          I=PIVOT(K)
          PIVOT(K)=PIVOT(JBAR)
          PIVOT(JBAR)=I
          YP(JBAR)=YP(K)
          YP(K)=SIGMA
          DO 250 I=1,N
            SIGMA=QR(I,K)
            QR(I,K)=QR(I,JBAR)
            QR(I,JBAR)=SIGMA
250        CONTINUE
C   END OF COLUMN INTERCHANGE.
260        SIGMA=DDOT(N-K+1,QR(K,K),1,QR(K,K),1)
          IF (SIGMA .EQ. 0.0) THEN
            IFLAG=4
            RETURN
          ENDIF
```

```
270       IF (K .EQ. N) GO TO 300
          QRKK=QR(K,K)
          ALPHAK=-SQRT(SIGMA)
          IF (QRKK .LT. 0.0) ALPHAK=-ALPHAK
          ALPHA(K)=ALPHAK
          BETA=1.0/(SIGMA-QRKK*ALPHAK)
          QR(K,K)=QRKK-ALPHAK
          DO 290 J=KP1,NP2
            SIGMA=BETA*DDOT(N-K+1,QR(K,K),1,QR(K,J),1)
            DO 280 I=K,N
              QR(I,J)=QR(I,J)-QR(I,K)*SIGMA
280         CONTINUE
            IF (J .LT. NP2) YP(J)=YP(J)-QR(K,J)**2
290       CONTINUE
300     CONTINUE
        ALPHA(N)=QR(N,N)
C
C COMPUTE KERNEL OF JACOBIAN, WHICH SPECIFIES YP=DY/DS.
C
        TZ(NP1)=1.0
        DO 340 I=N,1,-1
          SUM=0.0
          DO 330 J=I+1,NP1
330         SUM=SUM+QR(I,J)*TZ(J)
340     TZ(I)=-SUM/ALPHA(I)
        YPNORM=DNRM2(NP1,TZ,1)
        DO 360 K=1,NP1
360     YP(PIVOT(K))=TZ(K)/YPNORM
        IF (DDOT(NP1,YP,1,YPOLD,1) .GE. 0.0) GO TO 380
        DO 370 I=1,NP1
370     YP(I)=-YP(I)
C YP  IS THE UNIT TANGENT VECTOR IN THE CORRECT DIRECTION.
C
C COMPUTE THE MINIMUM NORM SOLUTION OF [D RHO(Y(S))] V = -RHO(Y(S)).
C V IS GIVEN BY  P - (P,Q)Q  , WHERE P IS ANY SOLUTION OF
C [D RHO] V = -RHO  AND Q IS A UNIT VECTOR IN THE KERNEL OF [D RHO].
C
380     DO 440 I=N,1,-1
          SUM=QR(I,NP1)+QR(I,NP2)
          DO 430 J=I+1,N
430         SUM=SUM+QR(I,J)*ALPHA(J)
440     ALPHA(I)=-SUM/ALPHA(I)
        DO 450 K=1,N
450     TZ(PIVOT(K))=ALPHA(K)
        TZ(PIVOT(NP1))=1.0
      ELSE
        YPNORM=DNRM2(N+1,YP,1)
        DO 760 K=1,N+1
760     YP(K)=YP(K)/YPNORM
        IF (DDOT(NP1,YP,1,YPOLD,1) .GE. 0.0) GO TO 780
        DO 770 I=1,NP1
770     YP(I)=-YP(I)
      ENDIF
780   CONTINUE
      WRITE(6,*)' NFE ',NFE
C     WRITE(6,*)' QR '
```

```
C     WRITE(6,*)' YP '
C     WRITE(6,321)(YP(J),J=1,N+1)
C     WRITE(6,*)' TZ '
C     WRITE(6,321)(TZ(J),J=1,N+1)
C TZ NOW CONTAINS A PARTICULAR SOLUTION P, AND YP CONTAINS A VECTOR Q
C IN THE KERNEL(THE TANGENT).
      SIGMA=DDOT(NP1,TZ,1,YP,1)
      DO 470 J=1,NP1
       TZ(J)=TZ(J)-SIGMA*YP(J)
470   CONTINUE
C TZ IS THE NEWTON STEP FROM THE CURRENT POINT Y(S) = (LAMBDA(S), X(S)).
      RETURN
      END
C123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : HNFVAD (FILE HNFVAD.F)
C
C     THIS IS THE HOST SUBROUTINE TO EVALUATE THE JACOBIAN MATRIX
C     IN PARALLEL. THIS ASSUMES THAT THE JACOBIAN MATRIX ARE TO
C     IN A DYNAMIC ASSIGNMENT FASHION.
C
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE CTANGNF (FILE CTANGNF.F)
C     CALLS           : SENDMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       RECVMSG   (CUBE MANAGERS FORTRAN ROUTINE)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE HNFVAD(QR,F,Y,N,NUMI,NUMR,YP,TZ,IFLAG,A,PAR,IPAR,
     1                  CHOICE,FIRST,IBUFF,RBUFF)
C
C     INPUT VARIABLES
C       N = DIMENSION OF THE JACOBIAN MATRIX
C       A(1..N) = INITIAL RANDOM VECTOR THAT DEFINES THE HOMOTOPY
C                 MAP
C       Y(1)    = ARTIFICIAL HOMOTOPY PARAMETER
C       Y(2..N+1) = THE VALUE AT WHICH THE FUNCTION AND THE JACOBIAN
C                   ARE TO BE EVALUATED
C       PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C            WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN
C            SUBROUTINES FF AND FJAC.
C       NUMI = NUMBER OF PARAMETER IN IPAR.
C       NUMR = NUMBER OF PARAMETERS IN PAR
C       CHOICE(2) = 0 IF NODES ARE SUPPOSED TO RETURN QR
C                 = 1 IF NODES ARE SUPPOSED TO RETURN YP  AND TZ
C       FIRST  = .TRUE. IF THIS ROUTINE IS CALLED FOR THE FIRST TIME
C              = .FALSE. IF NOT
C
C     OUTPUT VARIABLES
C
C       IF CHOICE(2) = 0 THEN
C           QR(N,N+1)  = THE JACOBIAN MATRIX OF THE HOMOTOPY MAP AT Y.
C           F   = THE FUNCTION VALUE OF THE HOMOTOPY MAP AT Y.
C       IF CHOICE(2) = 1 THEN
C           YP  = AN ELEMENT OF THE KERNEL OF THE JACOBIAN MATRIX
```

```
C           TZ  = A PARTICULAR SOLUTION TO [QR].X = -F.
C
C     WORKING VARIABLES:
C
C       IBUFF(*) USED FOR BUFFERING INTEGER MESSAGES
C       RBUFF(*) USED FOR BUFFERING DOUBLE PRECISION MESSAGES
C
C     DECLARATION OF VARIABLES
C
      DOUBLE PRECISION QR(N,*),F(*),PAR(*),Y(*),A(*),YP(*),TZ(*)
     1                ,RBUFF(*)
      INTEGER IPAR(*),N,NUMI,NUMR,IFLAG,CHOICE(3),IBUFF(*)
      LOGICAL FIRST
C
C     GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A CUBE (CURRRENTLY SET TO 10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (2**MAXDIM)
C
      INTEGER DPSIZE
      PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
      PARAMETER (MAXDIM=10,FUNTYP=1,JACTYP=1)
      PARAMETER (MAXPROC=2**MAXDIM)
C
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NUMCOL,NUMROW
      INTEGER ROWNUM,COLNUM,PIVOT
      INTEGER PID,MYID,CDIM,CI,D1,D2,NUMP,TYPE,LEN,CNT
      DOUBLE PRECISION LAMBDA,ONEML
      COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1      ,NUMCOL(MAXPROC),NUMROW(MAXPROC),LCOL(MAXPROC)
      COMMON/CUBE/ PID,CDIM,CI,D1,D2,NUMP,IDIM,JDIM
C
      TYPE = 1001
      LAMBDA = Y(1)
      ONEML = 1-LAMBDA
      IF (FIRST) THEN
C
C     FIRST TIME THIS ROUTINE IS CALLED, SO SEND IPAR, AND PAR.
C     SEND Y ARRAY ANYWAY
C
        CALL SENDMSG(CI,TYPE,IPAR,NUMI*DPSIZE,NALL,NODEPID)
        CALL SENDMSG(CI,TYPE,PAR,NUMR*DPSIZE,NALL,NODEPID)
        RBUFF(1) = N
        RBUFF(2) = CHOICE(1)
        RBUFF(3) = CHOICE(2)
        RBUFF(4) = CHOICE(3)
        RBUFF(5) = IFLAG
        RBUFF(6) = D1
        RBUFF(7) = D2
        DO 10 I = 1, N
          RBUFF(I+7) = A(I)
          RBUFF(N+I+7) = Y(I)
10      CONTINUE
        RBUFF(N*2+8) = Y(N+1)
        CALL SENDMSG(CI,TYPE,RBUFF,(2*N+8)*8,NALL,NODEPID)
```

```
        FIRST = .FALSE.
      ELSE
        DO 20 I = 1,N+1
          RBUFF(I) = Y(I)
20      CONTINUE
        CALL SENDMSG(CI,TYPE,RBUFF,(N+1)*DPSIZE,NALL,NODEPID)
      ENDIF
      MAXLEN = 16000
C
C    FIRST START EVALUATING FUNCTION
C
        TYPE = 1002
        DO 100 I = 1,NUMP
          IF (I.GT. N) GOTO 200
          INDEX = I
          IBUFF(1) = I
          CALL SENDMSG(CI,TYPE,IBUFF,IPSIZE,I-1,NODEPID)
100     CONTINUE
200     CONTINUE
        DO 300 I = 1,N
          TYPE = 1501
          CALL RECVMSG(CI,TYPE,RBUFF,2*DPSIZE,CNT,NODE,NODEPID)
          JNDEX = RBUFF(1)
          F(JNDEX) = RBUFF(2)
          IF ( INDEX .GE. N) GOTO 300
          INDEX = INDEX + 1
          IBUFF(1) = INDEX
          TYPE = 1002
          CALL SENDMSG(CI,TYPE,IBUFF,IPSIZE,NODE,NODEPID)
300     CONTINUE
C
C       NOTIFY EVERYBODY THAT THE FUNCTION EVALUATION IS DONE
C
        TYPE = 1002
        IBUFF(1) = -1
        CALL SENDMSG(CI,TYPE,IBUFF,IPSIZE,NALL,NODEPID)
C
C    NOW START EVALUATING JACOBIAN MATRIX
C
C      NTOT IS THE TOTAL NUMBER OF COMPONENTS TO BE EVALUATED
C
        IF (IFLAG .EQ. -2) THEN
          NTOT = N*(N+1)
        ELSE
          NTOT = N**2
        ENDIF
        TYPE = 1003
        DO 400 I = 1,NUMP
           IF (I .GT. NTOT) GOTO 500
           INDEX = I
           JNDX = (INDEX-1)/N+1
           INDX = INDEX - (JNDX-1)*N
           IBUFF(1) = INDX
           IBUFF(2) = JNDX
           CALL SENDMSG(CI,TYPE,IBUFF,2*IPSIZE,I-1,NODEPID)
400     CONTINUE
```

```
500      CONTINUE
         DO 600 I = 1,NTOT
            INDEX = INDEX+1
            JNDX = (INDEX-1)/N+1
            INDX = INDEX - (JNDX-1)*N
            TYPE = 1502
            CALL RECVMSG(CI,TYPE,RBUFF,3*DPSIZE,CNT,NODE,NODEPID)
            KINDX = RBUFF(1)
            KJNDX = RBUFF(2)
            IF (IFLAG .EQ. -2) THEN
               QR(KINDX,KJNDX) = RBUFF(3)
            ELSE
               IF (IFLAG .EQ. 0) THEN
                 IF (KINDX .EQ. KJNDX) THEN
                    QR(KINDX,KJNDX+1) = 1.0D0 - RBUFF(3) *LAMBDA
                 ELSE
                    QR(KINDX,KJNDX+1) = LAMBDA*RBUFF(3)
                 ENDIF
               ELSE
                 IF (KINDX .EQ. KJNDX) THEN
                    QR(KINDX,KJNDX+1) = ONEML + RBUFF(3) *LAMBDA
                 ELSE
                    QR(KINDX,KJNDX+1) = LAMBDA*RBUFF(3)
                 ENDIF
               ENDIF
            ENDIF
            IF (INDEX .GT. NTOT) GOTO 600
            IBUFF(1) = INDX
            IBUFF(2) = JNDX
            TYPE = 1003
            CALL SENDMSG(CI,TYPE,IBUFF,2*IPSIZE,NODE,NODEPID)
600      CONTINUE
C
C   NOTIFY EVERYBODY THAT THE JACOBIAN EVALUATION IS DONE
C
      TYPE = 1003
      IBUFF(1) = -1
      CALL SENDMSG(CI,TYPE,IBUFF,IPSIZE,NALL,NODEPID)
C
      IF (IFLAG .NE. -2) THEN
         DO 1300 J = 1,N
           IF (IFLAG .EQ. -1) THEN
              QR(J,1) =  F(J) - Y(J+1) + A(J)
           ELSE
              QR(J,1) = A(J) - F(J)
           ENDIF
           F(J) = Y(J+1) - A(J) + LAMBDA*QR(J,1)
1300      CONTINUE
      ENDIF
      IF (CHOICE(2) .NE. 0) THEN
C
C   SEND BACK QR TO NODES FOR FACTORIZATION
C
         TYPE = 1001
         DO 75 KK = 1, NUMP
           NOFF = NUMROW(KK)*NUMCOL(KK)
```

```
          DO 65 II  =1,NUMROW(KK)
            ROWNUM = (II-1)*ROWDIM+ROWPOS(KK)
            DO 55 JJ = 1,NUMCOL(KK)
               COLNUM = (JJ-1)*COLDIM+COLPOS(KK)
               INDEX = (JJ-1)*NUMROW(KK)+II
               RBUFF(INDEX) = QR(ROWNUM,COLNUM)
55          CONTINUE
            RBUFF(NOFF+II) = F(ROWNUM)
65        CONTINUE
          CALL SENDMSG(CI,TYPE,RBUFF,(NUMCOL(KK)+1)*NUMROW(KK)*
     1                 DPSIZE,KK-1,NODEPID)
75       CONTINUE
         DO 1200 I = 1,NUMP
           TYPE = 1501
           CALL RECVMSG(CI,TYPE,RBUFF,MAXLEN,CNT,NODE,NODEPID)
           LEN = 3*DPSIZE*NUMCOL(NODE+1)
           IF (LEN .NE. CNT) THEN
              WRITE(6,*)' FROM HNFVAD, LEN .NE. CNT '
              STOP
           ENDIF
           DO 40 J = 1,NUMCOL(NODE+1)
             PIVOT = RBUFF(J)
             YP(PIVOT) = RBUFF(J+NUMCOL(NODE+1))
             TZ(PIVOT) = RBUFF(J+NUMCOL(NODE+1)*2)
40          CONTINUE
1200      CONTINUE
      ENDIF
      RETURN
      END
C123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : HNFVAL (FILE HNFVAL.F)
C
C    THIS IS THE HOST SUBROUTINE TO EVALUATE THE JACOBIAN MATRIX
C    IN PARALLEL. THIS ASSUMES THAT THE JACOBIAN MATRIX ARE TO
C    BE ASSIGNED IN A LINEAR MAPPING FASHION.
C
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE CTANGNF (FILE CTANGNF.F)
C     CALLS           : SENDMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       RECVMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       EXTRCL    (FILE : HNFVAL.F            )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
     SUBROUTINE HNFVAL(QR,F,Y,N,NUMI,NUMR,YP,TZ,IFLAG,A,PAR,IPAR,
     1                 CHOICE,FIRST,IBUFF,RBUFF)
C
C    INPUT VARIABLES
C       N = DIMENSION OF THE JACOBIAN MATRIX
C       A(1..N) = INITIAL RANDOM VECTOR THAT DEFINES THE HOMOTOPY
C                 MAP
C       Y(1)    = ARTIFICIAL HOTOPY PARAMETER
C       Y(2..N+1) = THE VALUE AT WHICH THE FUNCTION AND THE JACOBIAN
C                   ARE TO BE EVALUATED
```

```
C       PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C            WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN
C            SUBROUTINES FF AND FJAC.
C       NUMI = NUMBER OF PARAMETER IN IPAR.
C       NUMR = NUMBER OF PARAMETERS IN PAR
C       CHOICE(2)  = 0 IF NODES ARE SUPPOSED TO RETURN QR
C                  = 1 IF NODES ARE SUPPOSED TO RETURN YP, AND TZ
C       FIRST  = .TRUE. IF THIS ROUTINE IS CALLED FOR THE FIRST TIME
C              = .FALSE. IF NOT
C
C     OUTPUT VARIABLES
C
C       IF CHOICE(2) = 0 THEN
C           QR(N,N+1)  = THE JACOBIAN MATRIX OF THE HOMOTOPY MAP AT Y.
C           F   = THE FUNCTION VALUE OF THE HOMOTOPY MAP AT Y.
C       IF CHOICE(2) = 1 THEN
C           YP  = AN ELEMENT OF THE KERNEL OF THE JACOBIAN MATRIX
C           TZ  = A PARTICULAR SOLUTION TO [QR].X = -F.
C
C     WORKING VARIABLES:
C
C       IBUFF(*) USED FOR BUFFERING INTEGER MESSAGES
C       RBUFF(*) USED FOR BUFFERING DOUBLE PRECISION MESSAGES
C
C     DECLARATION OF VARIABLES
C
      DOUBLE PRECISION QR(N,*),F(*),PAR(*),Y(*),A(*),YP(*),TZ(*)
     1                 ,RBUFF(*)
      INTEGER IPAR(*),N,NUMI,NUMR,IFLAG,CHOICE(*),IBUFF(*)
      LOGICAL FIRST
C
C     GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A HYPERCUBE (CURRENTLY MAXDIM=10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (MAXPROC = 2**MAXDIM)
C
      INTEGER DPSIZE
      PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
      PARAMETER (MAXDIM=10)
      PARAMETER (MAXPROC=2**MAXDIM)
C
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NUMCOL,NUMROW
      INTEGER ROWNUM,COLNUM
      INTEGER PID,CDIM,CI,D1,D2,NUMP,TYPE,LEN,CNT,PIVOT
      COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1        ,NUMCOL(MAXPROC),NUMROW(MAXPROC),LCOL(MAXPROC)
      COMMON/CUBE/ PID,CDIM,CI,D1,D2,NUMP,IDIM,JDIM
C
      TYPE = 1001
      IF (FIRST) THEN
C
C     FIRST TIME THIS ROUTINE IS CALLED, SO SEND IPAR, AND PAR
C
        CALL SENDMSG(CI,TYPE,IPAR,NUMI*DPSIZE,NALL,NODEPID)
        CALL SENDMSG(CI,TYPE,PAR,NUMR*DPSIZE,NALL,NODEPID)
```

```
         RBUFF(1) = N
         RBUFF(2) = CHOICE(1)
         RBUFF(3) = CHOICE(2)
         RBUFF(4) = CHOICE(3)
         RBUFF(5) = IFLAG
         RBUFF(6) = D1
         RBUFF(7) = D2
         DO 10 I = 1, N
           RBUFF(I+7) = A(I)
           RBUFF(N+I+7) = Y(I)
10       CONTINUE
         RBUFF(N*2+8) = Y(N+1)
         CALL SENDMSG(CI,TYPE,RBUFF,(2*N+8)*DPSIZE,NALL,NODEPID)
         FIRST = .FALSE.
       ELSE
         DO 20 I = 1,N+1
           RBUFF(I) = Y(I)
20       CONTINUE
         CALL SENDMSG(CI,TYPE,RBUFF,(N+1)*DPSIZE,NALL,NODEPID)
       ENDIF
       MAXLEN = 16000
       DO 100 I = 1,NUMP
         TYPE = 1501
         CALL RECVMSG(CI,TYPE,RBUFF,MAXLEN,CNT,NODE,NPID)
         IF (LCOL(NODE+1) .EQ. 0) GOTO 100
         LEN = N*(LCOL(NODE+1)+1)*DPSIZE
         CALL EXTRCL(N,QR,RBUFF,F,NODE)
         IF (LEN .NE. CNT) THEN
           WRITE(6,*)' FROM HNFVAL, LEN .NE. CNT '
           STOP
         ENDIF
100    CONTINUE
C      DO 197 I=1,N
C      WRITE(6,129)(QR(I,JJ),JJ=1,N+1)
C129    FORMAT(1X,4(1X,E23.16))
C197    CONTINUE
       IF (CHOICE(2) .EQ. 1) THEN
C
C    PARALLEL FACTORIZATION
C    SO SEND QR ANF F BACK.
C
         TYPE = 1001
         DO 75 KK = 1,NUMP
           NOFF = NUMROW(KK)*NUMCOL(KK)
           DO 65 II = 1,NUMROW(KK)
             ROWNUM = (II-1)*ROWDIM + ROWPOS(KK)
             DO 55 JJ = 1,NUMCOL(KK)
               COLNUM = (JJ-1)*COLDIM+COLPOS(KK)
               INDEX = (JJ-1)*NUMROW(KK) + II
               RBUFF(INDEX) = QR(ROWNUM,COLNUM)
C               IF (II.EQ.3 .AND. JJ.EQ. 2 .AND. KK .EQ. 11)
C    1     WRITE(6,*)'  KKKKKKK ',RBUFF(INDEX),' INDEX ',INDEX
C            WRITE(6,*)' INDEX ',INDEX,' ROWNUM ',ROWNUM,' COLNUM ',
C    1                 COLNUM,' QR ',QR(ROWNUM,COLNUM),' KK ',KK
55           CONTINUE
             RBUFF(NOFF+II) = F(ROWNUM)
```

```
65          CONTINUE
C             WRITE(6,*)' NOFF ',RBUFF(7),' KK ',KK-1
            CALL SENDMSG(CI,TYPE,RBUFF,(NUMCOL(KK)+1)*NUMROW(KK)*
     1             DPSIZE,KK-1,NODEPID)
75        CONTINUE
          WRITE(6,*)' HERE OK 3 ',LEN
C
          DO 85 I = 1,NUMP
            TYPE = 1501
            CALL RECVMSG(CI,TYPE,RBUFF,MAXLEN,CNT,NODE,NODEPID)
            LEN = DPSIZE*NUMCOL(NODE+1)*3
            IF (LEN .NE. CNT) THEN
              WRITE(6,*)' FROM HNFVAL, LEN .NE. CNT '
              STOP
            ENDIF
            DO 40 II = 1,NUMCOL(NODE+1)
              PIVOT = RBUFF(II)
              YP(PIVOT) = RBUFF(II+NUMCOL(NODE+1))
              TZ(PIVOT) = RBUFF(II+2*NUMCOL(NODE+1))
40          CONTINUE
85        CONTINUE
        ENDIF
        RETURN
        END
       SUBROUTINE EXTRCL (N,QR,BUF,F,NODE)
       INTEGER INDEX,ROWNUM,NUMP,N
       INTEGER ROWPOS,COLPOS,COLDIM,ROWDIM,COLNUM
       DOUBLE PRECISION BUF(*),QR(N,*),F(*)
C
C      GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A HYPERCUBE (CURRENTLY MAXDIM=10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (MAXPROC = 2**MAXDIM)
C
        INTEGER DPSIZE
        PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
        PARAMETER (MAXDIM=10)
        PARAMETER (MAXPROC=2**MAXDIM)
C
        COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1         ,NUMCOL(MAXPROC),NUMROW(MAXPROC),LCOL(MAXPROC)
        COMMON/CUBE/ PID,CDIM,CI,B1,B2,NUMP,IDIM,JDIM
       DO 11 I = 1, N
           ROWNUM =I
           DO 10 J = 1, LCOL(NODE+1)
             INDEX = ( J - 1)*N + I
             COLNUM = (J-1)*NUMP + NODE+1
             QR(ROWNUM,COLNUM) = BUF(INDEX)
10         CONTINUE
11     CONTINUE
       NOFF = N*LCOL(NODE+1)
       DO 19 I =1,N
         F(I) = BUF(NOFF+I)
19     CONTINUE
       RETURN
```

```
      RETURN
      END
C1234567890123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : HNFVAR (FILE HNFVAR.F)
C
C     THIS IS THE HOST SUBROUTINE TO EVALUATE THE JACOBIAN MATRIX
C     IN PARALLEL. THIS ASSUMES THAT THE JACOBIAN MATRIX ARE TO
C     BE ASSIGNED IN A RECTANGULAR MAPPING FASHION.
C
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE CTANGNF (FILE CTANGNF.F)
C     CALLS           : SENDMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       RECVMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       EXTRCT    (FILE : HNFVAR.F            )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE HNFVAR(QR,F,Y,N,NUMI,NUMR,YP,TZ,IFLAG,A,PAR,IPAR,
     1               CHOICE,FIRST,IBUFF,RBUFF)
C
C     INPUT VARIABLES
C       N = DIMENSION OF THE JACOBIAN MATRIX
C       A(1..N) = INITIAL RANDOM VECTOR THAT DEFINES THE HOMOTOPY
C                 MAP
C       Y(1)    = ARTIFICIAL HOMOTOPY PARAMETER
C       Y(2..N+1) = THE VALUE AT WHICH THE FUNCTION AND THE JACOBIAN
C                   ARE TO BE EVALUATED
C       PAR(1:*) AND IPAR(1:*) ARE ARRAYS FOR (OPTIONAL) USER PARAMETERS,
C            WHICH ARE SIMPLY PASSED THROUGH TO THE USER WRITTEN
C            SUBROUTINES FF AND FJAC.
C       NUMI = NUMBER OF PARAMETER IN IPAR.
C       NUMR = NUMBER OF PARAMETERS IN PAR
C       CHOICE(2) = 0 IF NODES ARE SUPPOSED TO RETURN QR
C                 = 1 IF NODES ARE SUPPOSED TO RETURN YP, AND TZ
C       FIRST  = .TRUE. IF THIS ROUTINE IS CALLED FOR THE FIRST TIME
C              = .FALSE. IF NOT
C
C     OUTPUT VARIABLES
C
C       IF CHOICE(2) = 0 THEN
C           QR(N,N+1)  = THE JACOBIAN MATRIX OF THE HOMOTOPY MAP AT Y.
C           F   = THE FUNCTION VALUE OF THE HOMOTOPY MAP AT Y.
C       IF CHOICE(2) = 1 THEN
C           YP  = AN ELEMENT OF THE KERNEL OF THE JACOBIAN MATRIX
C           TZ  = A PARTICULAR SOLUTION TO [QR].X = -F.
C
C     WORKING VARIABLES:
C
C       IBUFF(*) USED FOR BUFFERING INTEGER MESSAGES
C       RBUFF(*) USED FOR BUFFERING DOUBLE PRECISION MESSAGES
C
C     DECLARATION OF VARIABLES
C
      DOUBLE PRECISION QR(N,*),F(*),PAR(*),Y(*),A(*),YP(*),TZ(*)
```

```
     1               ,RBUFF(*)
       INTEGER IPAR(*),N,NUMI,NUMR,IFLAG,CHOICE(*),IBUFF(*)
       LOGICAL FIRST
C
C     GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A CUBE (CURRRENTLY SET TO 10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (2**MAXDIM)
C
       INTEGER DPSIZE
       PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
       PARAMETER (MAXDIM=10)
       PARAMETER (MAXPROC=2**MAXDIM)
C
       INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NEB,NUMCOL,NUMROW
       INTEGER PID,MYID,CDIM,CI,D1,D2,NUMP,TYPE,LEN,CNT,PIVOT
       COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1       ,NUMCOL(MAXPROC),NUMROW(MAXPROC),LCOL(MAXPROC)
       COMMON/CUBE/ PID,CDIM,CI,D1,D2,NUMP,IDIM,JDIM
C
       TYPE = 1001
       IF (FIRST) THEN
C
C      FIRST TIME THIS ROUTINE IS CALLED, SO SEND IPAR, AND PAR
C
         CALL SENDMSG(CI,TYPE,IPAR,NUMI*DPSIZE,NALL,NODEPID)
         CALL SENDMSG(CI,TYPE,PAR,NUMR*DPSIZE,NALL,NODEPID)
         RBUFF(1) = N
         RBUFF(2) = CHOICE(1)
         RBUFF(3) = CHOICE(2)
         RBUFF(4) = CHOICE(3)
         RBUFF(5) = IFLAG
         RBUFF(6) = D1
         RBUFF(7) = D2
         DO 10 I = 1, N
           RBUFF(I+7) = A(I)
           RBUFF(N+I+7) = Y(I)
10       CONTINUE
         RBUFF(N*2+8) = Y(N+1)
         CALL SENDMSG(CI,TYPE,RBUFF,(2*N+8)*DPSIZE,NALL,NODEPID)
         FIRST = .FALSE.
       ELSE
         DO 20 I = 1,N+1
           RBUFF(I) = Y(I)
20       CONTINUE
         CALL SENDMSG(CI,TYPE,RBUFF,(N+1)*DPSIZE,NALL,NODEPID)
       ENDIF
       MAXLEN = 16000
       IF (CHOICE(2) .EQ. 0) THEN
         DO 100 I = 1,NUMP
           TYPE = 1501
           CALL RECVMSG(CI,TYPE,RBUFF,MAXLEN,CNT,NODE,NODEPID)
           LEN = DPSIZE *(NUMCOL(NODE+1)+1)*IDIM
           IF (LEN .NE. CNT) THEN
             WRITE(6,*)' FROM HNFVAR, LEN .NE. CNT '
```

```
            STOP
          ENDIF
          CALL EXTRCT(N,RBUFF,QR,NODE,F)
100      CONTINUE
      ELSE
         DO 200 I = 1,NUMP
          TYPE = 1501
          CALL RECVMSG(CI,TYPE,RBUFF,MAXLEN,CNT,NODE,NODEPID)
          LEN = 3*DPSIZE*NUMCOL(NODE+1)
C         WRITE(6,*)' LEN = = ',LEN,' NODE ',NODE,
C    1            ' COL ',NUMCOL(NODE+1)
          IF (LEN .NE. CNT) THEN
            WRITE(6,*)' FROM HNFVAR, LEN .NE. CNT '
            STOP
          ENDIF
          DO 40 J = 1,NUMCOL(NODE+1)
            JNDEX = COLDIM*(J-1)+COLPOS(NODE+1)
            PIVOT = RBUFF(J)
            YP(PIVOT) = RBUFF(J+NUMCOL(NODE+1))
            TZ(PIVOT) = RBUFF(J+NUMCOL(NODE+1)*2)
40         CONTINUE
200      CONTINUE
      ENDIF
      RETURN
      END
     SUBROUTINE EXTRCT (N,BUF,QR,NODE,F)
     INTEGER INDEX,BINDEX,ROWNUM,NUMP,N,OFFSET,EOFFST
     INTEGER ROWPOS,COLPOS,COLDIM,ROWDIM,COLNUM
     DOUBLE PRECISION BUF(*),QR(N,*),F(*)
C
C    GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A CUBE (CURRRENTLY SET TO 10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (2**MAXDIM)
C
      INTEGER DPSIZE
      PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
      PARAMETER (MAXDIM=10)
      PARAMETER (MAXPROC=2**MAXDIM)
C
      COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1              ,NUMCOL(MAXPROC),NUMROW(MAXPROC)
      COMMON/CUBE/ PID,CDIM,CI,B1,B2,NUMP,IDIM,JDIM
     N2 = 2*N
     DO 11 I = 1, NUMROW(NODE+1)
         ROWNUM = ROWDIM*(I-1) + ROWPOS(NODE+1)
         IF (ROWNUM .LE. N) THEN
           DO 10 J = 1, NUMCOL(NODE+1)
             INDEX = ( J - 1)*IDIM + I
             COLNUM = (J-1)*COLDIM + COLPOS(NODE+1)
             QR(ROWNUM,COLNUM) = BUF(INDEX)
10         CONTINUE
         ENDIF
11    CONTINUE
     NOFF = IDIM*NUMCOL(NODE+1)
```

```
      DO 19 I = 1,NUMROW(NODE+1)
        INDEX = ROWDIM*(I-1)+ROWPOS(NODE+1)
        F(INDEX) = BUF(NOFF+I)
19    CONTINUE
      RETURN
      END
C1234567890123456789012345678901234567890123456789012345678901234567890123456789012
C      SUBROUTINE NAME : INTLZ (FILE INTLZ.F)
C
C     THIS IS THE HOST SUBROUTINE TO INITIALIZE CONTROL AND
C     GLOBAL VARIABLES. THIS ROUTINE SHOULD BE CALLED BEFORE
C     CALLING CFIXPNF OR CFIXPDF
C
C
C      AUTHOR          : AMAL CHAKRABORTY
C      LAST UPGRADED   : 5/24/90
C      INPUT FILE      : NONE
C      OUTPUT FILE     : NONE
C      CALLED BY       : USERS MAIN ROUTINE
C      CALLS           : CUBEDIM   (CUBE MANAGERS FORTRAN ROUTINE)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE INTLZ(N,B1,B2,PCODE,CHOICE,AVGCST,HIGH,FNAME)
C
C     INPUT VARIABLES:
C                N : THE PROBLEM DIMENSION
C                B1: THE ROW DIMENSION OF THE PROCESSOR GRID
C                B2: THE COLUMN DIMENSION OF THE PROCESSOR GRID
C                PCODE = 1 IF THE USER WANTS PROGRAM TO DECIDE
C                       WHICH STEPS SHOULD BE EXECUTED IN PARALLEL
C                       0 OTHERWISE
C                CHOICE(1..3) IS INPUT VARIABLE IF PCODE = 1
C                AVGCST : AVERAGE COMPONENT COMPLEXITY OF THE
C                       JACOBIAN MATRIX
C                HIGH   : IF THERE IS A HIGH VARIATION IN COMPONENT
C                        COMPLEXITY
C
C     OUTPUT VARIABLES:
C                PCODE = 0 IF EVERY THING SHOULD BE DONE SERIALLY
C                       1, OTHERWISE
C                CHOICE(1) = 0, IF SERIAL EVALUATION
C                          1, IF PARALLEL EVALUATION
C                CHOICE(2) = 0, IF SERIAL FACTORIZATION
C                          1, IF PARALLEL FACTORIZATION
C                CHOICE(3) = 0, IF RECTANGULAR MAPPING
C                          1, IF LINEAR MAPPING
C                          2, IF DYNAMIC MAPPING
C
C     GLOBAL CUBE PARAMETERS
C
C   DPSIZE IS THE NUMBER OF BYTES REQUIRED TO STORE A DOUBLE PRECISION VALUE
C   MAXDIM = MAXIMUM DIMENSION OF A CUBE (CURRRENTLY SET TO 10)
C   MAXPROC = MAXIMUM NUMBER OF PROCESSORS (2**MAXDIM)
C
      INTEGER DPSIZE
      PARAMETER (NALL=-1,NODEPID=0,DPSIZE=8,IPSIZE=4)
      PARAMETER (MAXDIM=10)
```

```
      PARAMETER (MAXPROC=2**MAXDIM)
C
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NEB,NUMCOL,NUMROW
      INTEGER HPID,MYPID,COPEN,CHOICE(4),B1,B2
      DOUBLE PRECISION AVGCST,PLOAD,PLODPN,PLODOR,PLODOL
      LOGICAL HIGH
      INTEGER PID,CDIM,CI,D1,D2,NUMP,TYPE,LEN,CNT,CUBEDIM,PCODE
      CHARACTER*6 FNAME
      COMMON /QRFAC/ ROWDIM,COLDIM,ROWPOS(MAXPROC),COLPOS(MAXPROC)
     1       ,NUMCOL(MAXPROC),NUMROW(MAXPROC),LCOL(MAXPROC)
      COMMON/CUBE/ PID,CDIM,CI,D1,D2,NUMP,IDIM,JDIM
C
      PID = 0
      D1 = B1
      D2 = B2
      HPID = MYPID()
      CI = COPEN(HPID)
      CDIM = CUBEDIM()
      IF (D1 .EQ. 0 .AND. D2 .EQ. 0) THEN
        D1 = CDIM/2
        D2 = (CDIM-1)/2+1
      ENDIF
      IF (D1+D2 .NE. CDIM) THEN
        WRITE(6,*)' ERROR D1+D2 SHOULD BE THE DIMENSION OF THE CUBE'
        STOP
      ENDIF
      NUMP = 2 ** CDIM
      COLDIM = 2** D2
      ROWDIM = 2 ** D1
      IDIM = (N-1)/ROWDIM + 2
      JDIM = N/COLDIM+1
      IF (PCODE .EQ.0 ) THEN
         IF (CHOICE(2) .EQ. 1 .AND. CHOICE(1) .EQ. 0) THEN
            WRITE(6,*)' THIS PROGRAM ASSUMES IF FACTORIZATION IS DONE'
            WRITE(6,*)' THEN FUNCTION EVALUATION WILL ALSO BE DONE IN'
            WRITE(6,*)' PARALLEL. SO CHOICE(1) IS SET BY PROGRAM TO  '
            WRITE(6,*)' ONE '
            CHOICE (1) = 1
         ENDIF
      ENDIF
      IF (PCODE .EQ. 1) THEN
C
C PUT DECIDING LOGIC HERE
C
         IF (N .GE. 50) THEN
            CHOICE(1) = 1
            CHOICE(2) = 1
         ELSE
            CHOICE(2) = 0
         ENDIF
         PLOAD = N**2 * AVGCST
         PLODPN = LLOAD / NUMP
         IF (PLODPN .GE. 2.5) THEN
             IF (HIGH)THEN
               IF ( N .LE. 25) THEN
                   IF (AVGCST .GE. 15) THEN
```

```
              CHOICE(2) = 2
            ENDIF
          ELSE
            IF (AVGCST .GT. 50) THEN
              CHOICE(2) = 2
            ENDIF
          ENDIF
        ELSE
          IF (AVGCST .GT. 50) THEN
                CHOICE(2) = 2
          ELSE
               PLODOR = ((N-1)/ROWDIM+1) * (N/COLDIM+1)
               PLODOL = (N/CDIM + 1) * N
               IF (LOADOR .LT. LOADOL) THEN
                 CHOICE(2) = 0
               ELSE
                 CHOICE(2) = 1
               ENDIF
          ENDIF
        ENDIF
      ELSE
        IF (N .GE. 50) THEN
           CHOICE(2) = 1
           CHOICE(3) = 0
        ENDIF
      ENDIF
ENDIF
IF (PCODE .EQ. 1) THEN
   IF (CHOICE(1).EQ. 0 .AND. CHOICE(2) .EQ. 0) THEN
      WRITE(6,*)' EVERYTHING WILL BE DONE SERIALLY '
      PCODE = 0
      GO TO 1000
   ELSE
     IF (CHOICE(1) .EQ. 1) THEN
       WRITE(6,*)' PARALLEL JACOBIAN EVALUATION '
       IF(CHOICE(3).EQ.0) WRITE(6,*)' RECTANGULAR MAPPING '
       IF(CHOICE(3).EQ.1) WRITE(6,*)' LINEAR MAPPING '
       IF(CHOICE(3).EQ.2) WRITE(6,*)' DYNAMIC ASSIGNMENT '
     ELSE
       WRITE(6,*)' SERIAL JACOBIAN EVALUATION '
     ENDIF
     IF (CHOICE(2) .EQ. 1) THEN
       WRITE(6,*)' PARALLEL FACTORIZATION '
     ELSE
       WRITE(6,*)' SERIAL FACTORIZATION '
     ENDIF
  ENDIF
ELSE
   IF (CHOICE(1) .EQ. 0 .AND. CHOICE(2) .EQ. 0)THEN
       PCODE=0
   ELSE
       PCODE = 1
   ENDIF
ENDIF
IF (PCODE .EQ. 0 ) GOTO 1000
DO 100 I = 1,NUMP
```

```
      ROWPOS(I) = (I-1)/COLDIM + 1
      COLPOS(I) = (I-1) - (ROWPOS(I)-1)*COLDIM + 1
      NUMROW(I) = N / ROWDIM
      NUMCOL(I) = (N+1)/COLDIM
      IF (CHOICE(3) .EQ. 1) THEN
        LCOL(I) = (N+1)/NUMP
        IF (N+1-LCOL(I)*NUMP .GT. I-1) LCOL(I) = LCOL(I)+1
      ENDIF
      IF(N-N/ROWDIM*ROWDIM.GE.ROWPOS(I))NUMROW(I)=NUMROW(I)+1
      IF(N+1-(N+1)/COLDIM*COLDIM.GE.COLPOS(I))NUMCOL(I)=NUMCOL(I)+1
100   CONTINUE
      CALL LOAD(FNAME,NALL,NODEPID)
1000  RETURN
      END
C123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : HDCPOSE (FILE HDCPOSE.F)
C
C       THIS ROUTINE IMPLEMENTS THE QR FACTORIZATION USING GIVENS
C       ROTATIONS. THE PROCESSOR ARE MAPPED INTO A LINEAR ARRAY.
C       THE MATRIX ARE PARTITIONED IN ROW WRAPING FASHION.
C       FACTORIZATION IS OBTAINED BY REPEATING THREE STEPS N-1
C       TIMES. THEY ARE INDEPENDENT ANNHILATION, RECURSIVE MERGING
C       AND PARTIAL COLUMN SWITCHING.
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE PFJAC (FILE NODE.F)
C     CALLS           : SENDMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       RECVMSG   (CUBE MANAGERS FORTRAN ROUTINE)
C                       DOIAP     (FILE : HDCPOSE.F              )
C                       DOCMP     (FILE : HDCPOSE.F              )
C                       MODIFY    (FILE : HDCPOSE.F              )
C                       EXTRCT    (FILE : HDCPOSE.F              )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE HDCPOSE(IDIM,JDIM,N,QR,ALPHA,PIVOT,IERR,Y,SUM,
     1  SAVE1,SAVE2,CURPIV,BUF,MAXLEN,C1,S1)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C       IDIM IS THE ROW DIMENSION OF THE QR MATRIX
C       JDIM IS THE COLUMN DIMENSION OF THE QR MATRIX
C       N IS THE NUMBER OF ROWS
C       PIVOT(1..(N+1)/NUMP+1) IS THE ARRAY WHERE PERMUTATION
C               IS STORED (OUT)
C       NUMP = NUMBER OF PROCESSORS (IN)
C       ALPHA(1..(N+1)/NUMP+1) = WORK ARRAY
C       SAVE1(1..(N+1)/MIN(COLDIM,ROWDIM)+1) WORK ARRAY
C       SAVE2(1..(N+1)/MIN(COLDIM,ROWDIM)+1) WORK ARRAY
C       CURPIV(*) WORK ARRAY
C       BUF(*) WORK ARRAY (USED FOR SENDING MESSAGES ETC.)
C       QR(NDIM,*) IS THE PART OF THE MATRIX THAT IS ASSIGNED TO THIS
C           PROCESSOR
C       C1(*),C1(*) ARE WORK ARRAYS USED TO STORE MULTIPLIERS
C       SUM (*) WORK ARRAY ( COLUMN NORMS ARE STORED HERE)
C       Y(N) = RIGHT HAND SIDE OF THE EQUATION AX = Y (IN AND OUT).
```

```
C
      DOUBLE PRECISION ALPHA(*),SUM(*),SAVE1(*),SAVE2(*),BUF(*),
     1 CURPIV(*),QR(IDIM,*),Y(*)
        INTEGER IDIM,JDIM,N,PIVOT(*),NUMP,NUMROW,ROWPOS,COLPOS,
     1  NUMCOL,ROWDIM,COLDIM,ROWBEG,MAXLEN
C
C       LOCAL VARIABLES
C
        INTEGER MYID,NP1,I,K,IROW,CPROC,CI,ROW,B1,B2
        INTEGER MYNODE,CUBEDIM,MYPID,PID,CDIM,TYPE,SNODE,
     1   SPID,CNT,RNODE,RPID
        DOUBLE PRECISION  C1(*),S1(*),SAVEN,CMAX,PIVELM ,C,S
        CHARACTER*72 STRIN
        LOGICAL SAVED,WORK
        COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1         ,NUMCOL,NUMROW
        COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
119       NP1 = N + 1
        MYID = MYNODE()
        PID = MYPID()
        CDIM = CUBEDIM ()
        NUMP = 2 ** CDIM
        SAVED = .FALSE.
        ROWNUM = 1
        COLNUM = 1
        TYPE = 30 + 3*CDIM
        IF (ROWNUM .GT.ROWPOS) THEN
           ROWBEG = 1 + ROWDIM - ROWNUM + ROWPOS
        ELSE
           ROWBEG = 1 + ROWPOS - ROWNUM
        ENDIF
        ROWBEG = (ROWBEG -1)/ROWDIM + 1
        DO 10  K = 1,NUMCOL
          IF (NUMROW .NE. 0) THEN
           SUM(K) = DDOT(NUMROW,QR(1,K),1,QR(1,K),1)
          ELSE
           SUM(K) = 0.0D0
          ENDIF
10      CONTINUE
        DO 190 K = 1, NUMCOL
          PIVOT(K) =(K-1)*COLDIM + COLPOS
190     CONTINUE
C
C       COMPLETE CALCULATION OF COLUMN NORMS BY ADDING PARTIAL NORMS
C       COMPUTED BY OTHER PROCESSOR
C
          DO 20 I = 1, B1
20          CALL EXCHANGE(SUM,I+B2,BUF)
C
C       FIND THE COLUMN WITH THE LARGEST NORM.
C
        IROW = 1
        CMAX = SUM(1)
        IF (COLPOS .EQ. 1) THEN
         IF (NUMCOL .GT. 1) THEN
```

```
          IMAX = 1
          CMAX = SUM(1)
          DO 77 I = 2 , NUMCOL
              IF(CMAX .LT. SUM(I)) THEN
                 CMAX = SUM(I)
                 IMAX = I
              ENDIF
77        CONTINUE
          NSAVEN = PIVOT(IMAX)
          PIVOT(IMAX) = PIVOT(1)
          PIVOT(1) = NSAVEN
          SAVE = SUM(1)
          SUM(1) = SUM(IMAX)
          SUM(IMAX) = SAVE
          DO 22 K = 1, NUMROW
            SAVEN = QR(K,IMAX)
            QR (K,IMAX) = QR(K,1)
            QR(K,1) = SAVEN
22        CONTINUE
         ELSE
          BUF(1) = SUM(1)
          BUF(2) = PIVOT(1)
          DO 29 I = ROWBEG , NUMROW
            BUF(I+2) = QR(I,1)
29        CONTINUE
          IF (COLPOS .EQ. COLDIM) THEN
            SNODE = MYID - COLDIM + 1
          ELSE
            SNODE = MYID + 1
          ENDIF
          SPID = 0
          CALL SENDW(CI,TYPE,BUF,8*(NUMROW-ROWBEG+3),SNODE,SPID)
          CALL RECVW(CI,TYPE,BUF,8*(NUMROW-ROWBEG+3),CNT,
     1               RNODE,RPID)
          IF (SUM(1) .LT. BUF(1)) THEN
            DO 40 I =ROWBEG,NUMROW
              QR(I,1) = BUF(I-ROWBEG+3)
40          CONTINUE
            SUM(1) = BUF(1)
            PIVOT(1) = BUF(2)
          ENDIF
        ENDIF
       ELSE
        IF(COLDIM .GT. N) THEN
           IF (COLNUM .EQ. COLPOS-1) THEN
             IF (COLPOS .EQ. 1) THEN
                SNODE = MYID + COLDIM -1
             ELSE
                SNODE = MYID - 1
             ENDIF
             BUF(1) = SUM(1)
             BUF(2) = PIVOT(1)
             DO 60 I = ROWBEG , NUMROW
               BUF(I+2) = QR(I,1)
60           CONTINUE
             CALL SENDW(CI,TYPE,BUF,8*(NUMROW-ROWBEG+3),SNODE,SPID)
```

```
            CALL RECVW(CI,TYPE,BUF,8*(NUMROW-ROWBEG+3),CNT
     1           ,RNODE,RPID)
            IF (SUM(1) .GT. BUF(1)) THEN
              DO 70 I =ROWBEG,NUMROW
                QR(I,1) = BUF(I-ROWBEG+3)
70            CONTINUE
              SUM(1) = BUF(1)
              PIVOT(1) = BUF(2)
            ENDIF
          ENDIF
        ENDIF
          DO 21 K = 1, NUMCOL
            IF ( CMAX .LT. SUM(K) ) THEN
               IROW = K
               CMAX = SUM(K)
            ENDIF
21        CONTINUE
        ENDIF
C
C       SWITCH COLUMN
C
37      DO 30 K = 1, N-1
         SAVED = .FALSE.
         CALL DOIAP(N,QR,K,IDIM,ALPHA,Y,C1,S1,PIVELM,BUF,MAXLEN)
         CALL DOCMP(N,QR,K,IDIM,ALPHA,
     1      SAVED,SAVE1,SAVE2,C,S ,BUF,CURPIV,Y,PIVELM)
         CALL MODIFY(N,QR,K+1,IDIM,ALPHA,SUM,CURPIV, PIVOT,BUF)
30      CONTINUE
        RETURN
        END
        SUBROUTINE MODIFY(N,QR,K,NDIM,ALPHA,SUM,CURPIV,PIVOT
     1                    ,BUF)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        INTEGER N,PIVOT(*),K
        DOUBLE PRECISION QR(NDIM,*),ALPHA(*),SUM(*),CURPIV(*)
     1         ,BUF(*)
C
C       LOCAL VARIABLES
C
        CHARACTER*72 STRIN
        INTEGER PID,MYID,CDIM,CI,B1,B2,NUMROW,NUMCOL,TYPE
        INTEGER IMAX,CPROC,STROW,COLPOS,ROWPOS,ROWDIM,COLDIM
        INTEGER ROWBEG,COLBEG,SNODE,SPID,RNODE,RPID,COLNUM,
     1          ROWNUM
        DOUBLE PRECISION CMAX,SAVEN,SAVE
        COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1         ,NUMCOL,NUMROW
        COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2
C
       ROWNUM = K-1 - (K-1)/ROWDIM*ROWDIM
       SPID = 0
       TYPE = 100 + 3*CDIM + K
       IF (ROWNUM .EQ. 0) ROWNUM = ROWDIM
       COLNUM = K - K/COLDIM*COLDIM
       IF (COLNUM .EQ. 0) COLNUM = COLDIM
       IF (COLNUM .GT. COLPOS) THEN
```

```
          COLBEG = K + COLDIM - COLNUM + COLPOS
        ELSE
          COLBEG = K + COLPOS - COLNUM
        ENDIF
        IF (ROWNUM .GT. ROWPOS) THEN
          ROWBEG = K -1 + ROWDIM - ROWNUM +ROWPOS
        ELSE
          ROWBEG = K -1 + ROWPOS - ROWNUM
        ENDIF
         ROWBEG = (ROWBEG -1)/ ROWDIM + 1
         COLBEG = (COLBEG -1)/ COLDIM + 1
        DO 1 I = COLBEG,NUMCOL
1           SUM(I) =  SUM(I)  - CURPIV(I)*CURPIV(I)
        IF( COLPOS .EQ. COLNUM) THEN
          IF (COLBEG .LT. NUMCOL) THEN
           IMAX = COLBEG
           CMAX = SUM(COLBEG)
           DO 10 I = COLBEG+1,NUMCOL
             IF (CMAX .LT. SUM(I) ) THEN
               CMAX = SUM(I)
               IMAX = I
             ENDIF
10          CONTINUE
           NSAVEN = PIVOT(IMAX)
           PIVOT(IMAX) = PIVOT(COLBEG)
           PIVOT(COLBEG) = NSAVEN
           SAVE = SUM (COLBEG)
           SUM(COLBEG)= SUM(IMAX)
           SUM(IMAX) = SAVE
           DO 22 I = 1, NUMROW
             SAVEN = QR(I,IMAX)
             QR (I,IMAX) = QR(I,COLBEG)
             QR(I,COLBEG) = SAVEN
22          CONTINUE
          ELSE
           BUF(1) = SUM(COLBEG)
           BUF(2) = PIVOT(COLBEG)
           DO 20 I = 1 , NUMROW
             BUF(I+2) = QR(I,COLBEG)
20          CONTINUE
           IF (COLPOS .EQ. COLDIM) THEN
             SNODE = MYID - COLDIM + 1
           ELSE
             SNODE = MYID + 1
           ENDIF
           CALL SENDW(CI,TYPE,BUF,8*(NUMROW+2),SNODE,SPID)
           CALL RECVW(CI,TYPE,BUF,8*(NUMROW+2),RNODE,RPID)
           IF (SUM(COLBEG) .LT. BUF(1)) THEN
             DO 40 I =1,NUMROW
               QR(I,COLBEG) = BUF(I+2)
40           CONTINUE
             SUM(COLBEG) = BUF(1)
             PIVOT(COLBEG) = BUF(2)
           ENDIF
         ENDIF
        ELSE
```

```
      IF(K+COLDIM .GT. N+1) THEN
        IF (COLNUM .EQ. COLPOS-1 .OR. (COLPOS .EQ. 1
     1     .AND. COLNUM .EQ. COLDIM)  ) THEN
          IF (COLPOS .EQ. 1) THEN
            SNODE = MYID + COLDIM -1
          ELSE
            SNODE = MYID - 1
          ENDIF
          BUF(1) = SUM(COLBEG)
          BUF(2) = PIVOT(COLBEG)
          DO 60 I = 1 , NUMROW
            BUF(I+2) = QR(I,COLBEG)
60        CONTINUE
          CALL SENDW(CI,TYPE,BUF,8*(NUMROW+2),SNODE,SPID)
          CALL RECVW(CI,TYPE,BUF,8*(NUMROW+2),RNODE,RPID)
          IF (SUM(COLBEG) .GT. BUF(1)) THEN
            DO 70 I =1,NUMROW
              QR(I,COLBEG) = BUF(I+2)
70          CONTINUE
            SUM(COLBEG) = BUF(1)
            PIVOT(COLBEG) = BUF(2)
          ENDIF
        ENDIF
      ENDIF
C
      RETURN
      END
C
      SUBROUTINE DOCMP(N,QR,K,NDIM,ALPHA,
     1  SAVED,SAVE1,SAVE2,C,S,BUF,CURPIV,B,PIVELM)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION QR(NDIM,1),ALPHA(N),B(1),
     1  SAVE1(1),SAVE2(1),C,S,BUF(*),CURPIV(1),PIVELM,T
      INTEGER N,NUMP,NDIM,CDIM,CI,MYID,NUMROW,K,SSTROW
      INTEGER ROWPOS,COLPOS,ROWDIM,COLDIM,PID,B1,B2,CPROC,I
      INTEGER ROWBEG,COLBEG
      LOGICAL SAVED
      CHARACTER*72 STRIN
C
C     K IS THE PIVOT ROW
C     CPROC IS THE PROCESSOR HOLDING THE CURRENT ROW.
C
C     STROW IS THE IST ROW GREATER THAN EQUAL TO K THAT IS ASSIGNED
C     TO THIS NODE
C
      DOUBLE PRECISION BSAVE1,BSAVE2
      COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1       ,NUMCOL,NUMROW
      COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
      ROWNUM = K - K/ROWDIM*ROWDIM
      IF (ROWNUM .EQ. 0) ROWNUM = ROWDIM
      COLNUM = K - K/COLDIM*COLDIM
      IF (COLNUM .EQ. 0) COLNUM = COLDIM
      IF (COLNUM .GT. COLPOS) THEN
```

```
          COLBEG = K + COLDIM - COLNUM + COLPOS
        ELSE
          COLBEG = K  + COLPOS - COLNUM
        ENDIF
        COLBEG = (COLBEG -1)/ COLDIM + 1
        IF (ROWNUM .GT. ROWPOS) THEN
          ROWBEG = K + ROWDIM - ROWNUM + ROWPOS
        ELSE
          ROWBEG = K  + ROWPOS - ROWNUM
        ENDIF
        SSTROW = ROWBEG
        ROWBEG = (ROWBEG -1)/ ROWDIM + 1
        CPROC = (ROWNUM -1)*COLDIM + COLNUM -1
        DO 10 I = 1,B1
         CALL EXROWS(K,B1-I+1+B2,QR,N,NDIM,BUF,B,
     1            SSTROW,PIVELM,COLBEG,ROWBEG)
         CALL ROTATE(K,ROWBEG,SAVED,C,S,N,NDIM,QR,SAVE1,SAVE2,
     1     BSAVE1,BSAVE2,BUF,B,SSTROW,PIVELM,COLBEG,NUMCOL)
10       CONTINUE
        SSTROW = (ROWBEG -1)*ROWDIM + ROWPOS
        DO 11 I = COLBEG,NUMCOL
          CURPIV(I) = QR (ROWBEG,I)
11       CONTINUE
        IF ((ROWPOS .NE. ROWNUM).AND.(C .NE. 0 .OR. S .NE. 0))THEN
          DO 12 I = 1,NUMCOL-COLBEG+1
            QR(ROWBEG,I+COLBEG-1) = SAVE2(I)*C- SAVE1(I) * S
12         CONTINUE
            B(ROWBEG) = BSAVE2*C- BSAVE1 * S
        ELSE
          IF (ROWPOS .NE. ROWNUM) THEN
          DO 13 I = COLBEG,NUMCOL
            QR(ROWBEG,I) = SAVE2(I-COLBEG+1)
13         CONTINUE
            B(ROWBEG) = BSAVE2
          ENDIF
        ENDIF
        RETURN
        END
C
        SUBROUTINE ROTATE(K,STROW,SAVED,CSAVED,SSAVED,N,NDIM
     1   ,QR,SAVE1,SAVE2,BSAVE1,BSAVE2,BUF,B,SSTROW
     1   ,PIVELM,COLBEG,NUMCOL)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        INTEGER STROW,N,NDIM,RINDEX,SSTROW,NUMCOL,COLBEG
        LOGICAL SAVED
        DOUBLE PRECISION SAVE1(*),SAVE2(*),CSAVED,SSAVED,
     1   QR(NDIM,*),BUF(*),C,S,T,BSAVE1,BSAVE2,B(*),PIVELM
        CHARACTER*72 STRIN
C
       MYID = MYNODE()
        RINDEX = BUF(1)
        IF (PIVELM.NE.0.0D0 .OR. BUF(2).NE. 0.0D0) THEN
        IF (SSTROW .GT. RINDEX) THEN
         IF (.NOT. SAVED) THEN
           DO 10 I = 1, NUMCOL-COLBEG+1
             SAVE1(I) = BUF (2+I)
```

```
            SAVE2(I) = QR(STROW,I+COLBEG-1)
            BSAVE1 = BUF(NUMCOL-COLBEG+4)
            BSAVE2 = B(STROW)
10          CONTINUE
            IF (DABS(PIVELM) .GE. DABS(BUF(2))) THEN
              T = BUF(2)/PIVELM
              SSAVED = 1.0D0 / DSQRT ( 1.0D0 +T*T)
              CSAVED = SSAVED * T
            ELSE
              T = PIVELM/BUF(2)
              CSAVED = 1.0D0 / DSQRT ( 1.0D0 + T*T)
              SSAVED = CSAVED * T
            ENDIF
            C = CSAVED
            S = SSAVED
            SAVED = .TRUE.
          ELSE
            IF (DABS(PIVELM) .GE. DABS(BUF(2))) THEN
              T = BUF(2)/PIVELM
              S= 1.0D0 / DSQRT ( 1.0D0 + T*T)
              C= S* T
            ELSE
              T = PIVELM/BUF(2)
              C= 1.0D0 / DSQRT ( 1.0D0 + T*T)
              S= C* T
            ENDIF
          ENDIF
        ELSE
            IF (DABS(PIVELM) .GE. DABS(BUF(2))) THEN
              T = BUF(2)/PIVELM
              C= 1.0D0 / DSQRT ( 1.0D0 + T*T)
              S= C* T
            ELSE
              T = PIVELM/BUF(2)
              S= 1.0D0 / DSQRT ( 1.0D0 + T*T)
              C= S* T
            ENDIF
        ENDIF
        IF (SSTROW .GT. RINDEX) THEN
           DO 30 J = COLBEG,NUMCOL
             QR(STROW,J) = C * BUF(J-COLBEG+3) + S * QR (STROW,J)
30         CONTINUE
           B(STROW) = C * BUF(NUMCOL-COLBEG+4) + S * B (STROW)
           SSTROW = RINDEX
           PIVELM = BUF(2)*C + S * PIVELM
        ELSE
           DO 31 J = COLBEG,NUMCOL
             QR(STROW,J) = C * QR(STROW,J) + S * BUF(J-COLBEG+3)
31         CONTINUE
           B(STROW) = C * B(STROW) + S * BUF(NUMCOL-COLBEG+4)
           PIVELM = BUF(2)*S + C * PIVELM
        ENDIF
        ELSE
           IF (SSTROW .GT. RINDEX) THEN
             IF(.NOT. SAVED) THEN
               DO 80 I = 1, NUMCOL-COLBEG+1
```

```
                SAVE1(I) = BUF (I+2)
                SAVE2(I) = QR(STROW,I+COLBEG-1)
                BSAVE1 = BUF(NUMCOL-COLBEG+4)
                BSAVE2 = B(STROW)
80            CONTINUE
              SAVED = .TRUE.
              CSAVED = 0.0D0
              SSAVED = 0.0D0
            ENDIF
            DO 32 J = COLBEG,NUMCOL
32           QR(STROW,J)=  BUF(J-COLBEG+3)
            B(STROW) = BUF(NUMCOL-COLBEG+4)
            SSTROW = RINDEX
            PIVELM = BUF(2)
          ENDIF
        ENDIF
        RETURN
        END
C
        SUBROUTINE EXROWS(K,BIT,QR,N,NDIM,BUF,B,RINDEX,PIVELM
     1          ,COLBEG,ROWBEG)
C
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        INTEGER BIT,MYID,CDIM,NDIM,CI,K,RINDEX,PID,B1,B2
        INTEGER SPID,NEB,RNODE,RPID,TYPE,CNT,ROWPOS,COLPOS,
     1         ROWDIM,COLDIM,ROWBEG,COLBEG,ROWNUM,COLNUM
        DOUBLE PRECISION QR(NDIM,*),BUF(*),B(*),PIVELM
        CHARACTER*72 STRIN
C
        COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1         ,NUMCOL,NUMROW
        COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
        SPID = 0
        TYPE = BIT + 10 + CDIM
        BUF(1) = RINDEX
        BUF(2) = PIVELM
        DO 10 I = 1, NUMCOL-COLBEG+1
10         BUF(I+2) = QR(ROWBEG,I+COLBEG-1)
        BUF(NUMCOL-COLBEG+4) = B(ROWBEG)
        CALL SENDW(CI,TYPE,BUF,8*(NUMCOL-COLBEG+4),NEB(BIT),SPID)
        CALL RECVW(CI,TYPE,BUF,8*(NUMCOL-COLBEG+4),CNT,RNODE,RPID)
        IF(NEB(BIT) .NE. RNODE) THEN
           CALL SYSLOG (MYPID(),' IERR = 3')
           IERR = 3
           RETURN
        ENDIF
        IERR = 0
        RETURN
        END
C
     SUBROUTINE DOIAP(N,QR,K,NDIM,ALPHA,B,C,S,PIVELM,BUF,MAXLEN)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     DOUBLE PRECISION QR(NDIM,*),ALPHA(*),B(*),C(*),S(*),PIVELM
    1          ,BUF(*),SAVE,T,ALPHAK,QRKK,SIGMA,BETA
     INTEGER N,NDIM,K,TYPE,ROWDIM,COLDIM,ROWPOS,COLPOS,CDIM
```

```
C
C       ROWBEG IS THE IST ROW GREATER THAN EQUAL TO K THAT IS ASSIGNED
C       TO THIS NODE
C
      INTEGER ROWBEG,COLBEG,I,J,CPROC,CINDEX,SINDEX,MAXLEN
      INTEGER CI,PID,B1,B2,MYID,NUMP,CNT,ROWNUM,COLNUM,TOTNUM
      INTEGER SNODE,SPID,RNODE,RPID
      CHARACTER*80 STRIN
        COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1         ,NUMCOL,NUMROW
        COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
        SPID = 0
        ROWNUM = K - K/ROWDIM*ROWDIM
        IF (ROWNUM .EQ. 0) ROWNUM = ROWDIM
        COLNUM = K - K/COLDIM*COLDIM
        IF (COLNUM .EQ. 0) COLNUM = COLDIM
        IF (COLNUM .GT. COLPOS) THEN
          COLBEG = K + COLDIM - COLNUM + COLPOS
        ELSE
          COLBEG = K  + COLPOS - COLNUM
        ENDIF
        COLBEG = (COLBEG -1)/ COLDIM + 1
        IF (ROWNUM .GT. ROWPOS) THEN
          ROWBEG = K + ROWDIM - ROWNUM + ROWPOS
        ELSE
          ROWBEG = K  + ROWPOS - ROWNUM
        ENDIF
        ROWBEG = (ROWBEG -1)/ ROWDIM + 1
        TYPE = 51 + 5*CDIM + K
        IF (COLNUM .EQ. COLPOS) THEN
         SIGMA = DDOT(NUMROW-ROWBEG+1,QR(ROWBEG,COLBEG),1,
     1              QR(ROWBEG,COLBEG),1)
1999     IF (SIGMA .EQ. 0.0D0 .OR. NUMROW .EQ. ROWBEG) THEN
           BUF(2) = 0.0D0
           BUF(1) = 1
           PIVELM = QR(ROWBEG,COLBEG)
           BUF(3) = PIVELM
           SNODE = NEB(1)
           TOTNUM = 0
         ELSE
           QRKK = QR(ROWBEG,COLBEG)
           ALPHAK = -DSQRT(SIGMA)
           IF (QRKK .LT. 0.0D0) ALPHAK = -ALPHAK
             BETA = 1.0D0 / (SIGMA - QRKK*ALPHAK)
           BUF(1) = 1
           SNODE = NEB(1)
           BUF(2) = (NUMROW - ROWBEG)
           IF(BUF(2) .LT. 0.0D0) BUF(2) = 0.0D0
           TOTNUM = BUF(2)
           PIVELM = ALPHAK
           BUF(3) = PIVELM
           QR(ROWBEG,COLBEG) = QRKK - ALPHAK
           IF ( NUMROW .GT. ROWBEG) THEN
             BUF(7) = QR(ROWBEG,COLBEG)
             BUF(4) = SIGMA
```

```
            BUF(5) = QRKK
            BUF(6) = ALPHAK
            DO 40 I = 1, TOTNUM
              BUF(I+7) = QR(ROWBEG+I,COLBEG)
40          CONTINUE
           ENDIF
         ENDIF
         BUF(3)= PIVELM
         CALL SENDW(CI,TYPE,BUF,8*(TOTNUM+8),SNODE,SPID)
       ELSE
         CALL RECVW(CI,TYPE,BUF,8*MAXLEN,CNT,RNODE,RPID)
         PIVELM = BUF(3)
       ENDIF
       DO 50 IBIT = BUF(1)+1,B2
         BUF(1) = IBIT
         SNODE = NEB(IBIT)
         TOTNUM = BUF(2)
         CALL SENDW(CI,TYPE,BUF,8*(TOTNUM+8),SNODE,SPID)
50     CONTINUE
       IF (BUF(2) .GT. 0 .AND. COLNUM .NE. COLPOS) THEN
         SIGMA = BUF(4)
         QRKK = BUF(5)
         ALPHAK = BUF(6)
         BETA = 1.0D0/(SIGMA -QRKK*ALPHAK)
         DO 20 J = COLBEG,NUMCOL
           ALPHA(J) = BETA*DDOT(NUMROW-ROWBEG+1,BUF(7),
     1          1,QR(ROWBEG,J),1)
20       CONTINUE
         DO 30 J = COLBEG,NUMCOL
           DO 33 I = ROWBEG,NUMROW
            QR(I,J) = QR(I,J) - BUF(I-ROWBEG+7)*ALPHA(J)
33         CONTINUE
30       CONTINUE
         SIGMA = BETA*DDOT(NUMROW-ROWBEG+1,BUF(7),1,B(ROWBEG),1)
         DO 34 I = ROWBEG,NUMROW
           B(I) = B(I) - BUF(I-ROWBEG+7)*SIGMA
34       CONTINUE
       ELSE
         IF (BUF(2) .GT. 0) THEN
         DO 120 J = COLBEG+1,NUMCOL
           ALPHA(J) = BETA*DDOT(NUMROW-ROWBEG+1,BUF(7),
     1          1,QR(ROWBEG,J),1)
120      CONTINUE
         DO 70 J = COLBEG+1,NUMCOL
           DO 73 I = ROWBEG,NUMROW
            QR(I,J) = QR(I,J) - BUF(I-ROWBEG+7)*ALPHA(J)
73         CONTINUE
70       CONTINUE
         SIGMA = BETA*DDOT(NUMROW-ROWBEG+1,BUF(7),1,B(ROWBEG),1)
         DO 74 I = ROWBEG,NUMROW
           B(I) = B(I) - BUF(I-ROWBEG+7)*SIGMA
74       CONTINUE
         QR(ROWBEG,COLBEG) = ALPHAK
         DO 75 I = ROWBEG+1,NUMROW
           QR(I,COLBEG)=0.0D0
75       CONTINUE
```

```
         ENDIF
        ENDIF
129      RETURN
        END
C
C
        SUBROUTINE EXCHANGE(SNORM,BIT,BUFF)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        INTEGER BIT
        DOUBLE PRECISION BUFF(*),SNORM(*)
C
        INTEGER TYPE,SPID,SNODE,LEN,CNT,RNODE,RPID
        INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NUMCOL,NUMROW
        INTEGER CI,CDIM,PID,B1,B2
        CHARACTER*72 STRIN
        COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1         ,NUMCOL,NUMROW
        COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
        SNODE = NEB(BIT)
        SPID = 0
        LEN = 1
        TYPE = BIT + 10
        DO 22 I = 1,NUMCOL
22         BUFF(I) = SNORM(I)
        CALL SENDW(CI,TYPE,BUFF,8*(NUMCOL),SNODE,SPID)
        CALL RECVW(CI,TYPE,BUFF,8*(NUMCOL),CNT,RNODE,RPID)
        IF (RNODE .NE. SNODE) THEN
           IERR = 3
           CALL SYSLOG(MYPID(),' IERR = 3 ')
           RETURN
        ENDIF
        DO 23 I = 1, NUMCOL
          IF (SNORM(I).EQ. 0.0D0) THEN
             SNORM(I) = BUFF(I)
          ELSE
          IF (BUFF(I).NE.0) SNORM(I) = (SNORM(I)  + BUFF(I))
          ENDIF
23      CONTINUE
        RETURN
        END
C
C
C
C       THIS PROCEDURE RETURN THE NEIGHBOUR OF A PROCESSOR I BY
C       COMPLEMENTING THE BIT IN (BIT) POSITION.
C       INPUT : I -  PROCESSOR ID
C              BIT - BIT POSITION THAT WILL BE COMPLEMENTED
C       OUTPUT : IERR = -1 IF BIT IS GREATER THAN THE DIMENSION OF THE
C                HYPERCUBE.
C
C
C
        SUBROUTINE NEBOUR(PROC,NUMBIT,IERR,NDIM,NEB)
        LOGICAL LODD
        INTEGER PROC,BIT,NDIFF,IERR,NDIM,NEB(*)
```

```
C
        DO 20 BIT = 1, NUMBIT
        IF (BIT .GT. NDIM) THEN
          IERR = 1
          RETURN
        ELSE
          IERR = 0
        ENDIF
        NDIFF = 2 ** (BIT - 1)
        IF (NDIFF .EQ. 0 ) THEN
         IERR = 2
         RETURN
        ENDIF
        IF (.NOT. LODD(PROC / NDIFF)) THEN
          NEB(BIT) = PROC+ NDIFF
        ELSE
          NEB(BIT) = PROC - NDIFF
        ENDIF
20      CONTINUE
        RETURN
        END
        LOGICAL FUNCTION LODD (I)
        LODD = .TRUE.
        IF ( I - I / 2 * 2 .EQ. 0) LODD = .FALSE.
        RETURN
        END
C123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : NSOLVE (FILE NSOLVE.F)                          C
C     PURPOSE         : SOLVES AX = 0 and AX = B, WHERE 'A' IS A        C
C                       UPPER TRAPEZOIDAL (N BY N+1) MATRIX             C
C     AUTHOR          : AMAL CHAKRABORTY                                C
C     LAST UPGRADED   : 5/24/90                                         C
C     INPUT FILE      : NONE                                            C
C     OUTPUT FILE     : NONE                                            C
C     CALLED BY       : SUBROUTINE PFJAC (FILE NODE.F)                  C
C     CALLS           : NGET   (FILE NSOLVE.F)                          C
C                       NPUT   (FILE NSOLVE.F)                          C
C                       PROCNUM   (FILE NSOLVE.F)                       C
C                       MYCOL   (FILE NSOLVE.F)                         C
C                       MYROW   (FILE NSOLVE.F)                         C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE NSOLVE(IDIM,JDIM,N,B,QR,PRODUCT,BPRODUCT,
     1                  PIVOT,X,XB,BUFF)
C
C     ON INPUT:
C             IDIM : THE ROW DIMENSION OF QR. QR IS ACTUALLY A ONE
C                    DIMENSIONAL MATRIX. IDIM IS SET BY THE CALLING
C                    ROUTINE
C             JDIM : THE COLUMN DIMENSION OF QR. SET BY CALLING
C                    ROUTINE
C             N    : THE ROW DIMENSION OF 'QR'
C             QR   : THE UPPER TRAPEZOIDAL MATRIX
C             B    : THE RIGHT HAND SIDE OF THE EQUATION AX=B
C             PRODUCT : WORK ARRAY (ACCUMULATES PARTIAL PRODUCT FOR
C                       AX=0, SHOULD BE DIMENSIONED ATLEAST N/NUMP+1)
```

```
C             BPRODUCT : WORK ARRAY (ACCUMULATES PARTIAL PRODUCT FOR
C                      AX=B, SHOULD BE DIMENSIONED ATLEAST N/NUMP+1)
C             PIVOT :   PERMUTATION VECTOR FROM DECOMPOSITION PHASE.
C             BUFF  :   WORK ARRAY, USED FOR BUFFERING MESSAGES
C                      (SHOULD BE DIMENSIONED ATLEAST 2*N/NUMP+1)
C     ON OUTPUT:
C             X     : THE RESULT OF AX = 0
C                    (THE NODE NUMCOL NUMBER OF ELEMENTS, WHERE NUMCOL
C                     IS THE NUMBER OF ELEMENT THIS NODE HAS OF QR)
C             XB    : THE RESULT OF AX = B
C                    (THE NODE NUMCOL NUMBER OF ELEMENTS, WHERE NUMCOL
C                     IS THE NUMBER OF ELEMENT THIS NODE HAS OF QR)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER IDIM,JDIM,N,COLPOS,ROWPOS,NEB,NUMCOL,NUMROW,
     1        ROWDIM,COLDIM,B1,B2,NUMP,PID,MYID,TYPE,CI,
     2        PIVOT(*)
      DOUBLE PRECISION QR(IDIM,*),BUFF(*),X(*),PRODUCT(*),BPRODUCT(*)
     1                ,B(*),XB(*)
C           PARAMETERS
      INTEGER RIGHT,DOWN,LEFT,UP,ROW,COL
      INTEGER MAXLEN,BUFLEN
      PARAMETER (RIGHT=1,DOWN=2,LEFT=3,UP=4)
C
      LOGICAL MYCOL,MYROW
      INTEGER PROC(4)
C
      CHARACTER*72 STRIN
C
      COMMON/QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10),NUMCOL,NUMROW
      COMMON/CUBE/PID,MYID,CDIM,CI,B1,B2,NUMP
      COMMON/SOLVE/PROC
C
      NGETC = 0
      NPUTC = 0
      IF (NUMCOL .EQ. 0 .OR. NUMROW .EQ. 0) RETURN
C
C        INITIALIZE PROC ARRAY (PROC STORES PROCESSORS IN THE LEFT,
C           IN THE RIGHT, DOWN AND UP IN THE RECTANGULAR GRID)
C
      CALL PROCNUM(PROC,ROWPOS,COLPOS,ROWDIM,COLDIM)
      INDEX = NUMROW
      JNDEX = NUMCOL
C
      IF (MYCOL(N+1,COLPOS,COLDIM)) THEN
C
C     IF (N+1)TH COLUMN IS MINE, THEN INITIALIZE PRODUCT AND BPRODUCT
C
         DO 1 I = 1,NUMROW
           PRODUCT(I) = -QR(I,NUMCOL)
           BPRODUCT(I)= -B(I) - QR(I,NUMCOL)
1        CONTINUE
C
C     ALSO, INITIALIZE X AND XB
C
         X(NUMCOL) = 1.0D0
```

```
        XB(NUMCOL) = 1.0D0
C
C     PUT PRODUCT AND BPRODUCT ARRAY TO LEFT
C
        CALL MPUT(PRODUCT,BPRODUCT,NUMROW,LEFT)
        INDEX = INDEX - 1
        JNDEX = JNDEX - 1
      ENDIF
C
      DO 10 K = N,1,-1
       IF (MYCOL(K,COLPOS,COLDIM)) THEN
        IF (K .NE. 1) THEN
         IF (MYROW(K,ROWPOS,ROWDIM)) THEN
90         ROW = (INDEX -1)*ROWDIM + ROWPOS
           COL = (JNDEX -1)*COLDIM + COLPOS
           IF (COL .LT. ROW) THEN
               INDEX = INDEX -1
               GO TO 90
           ENDIF
C
C    IF K IS BOTH MY COLUMN AND ROW THEN GET PRODUCT AND BPRODUCT ARRAY
C    FROM RIGHT, COMPUTE X AND XB, MODIFY PRODUCT AND BPRODUCT AND
C    SEND THEM TO LEFT IF ANY PROCESSORS IN THE LEFT NEEDS THEM.
C
           CALL MGET(PRODUCT,BPRODUCT,INDEX,RIGHT)
           MGETC = MGETC+1
           X(JNDEX) = PRODUCT(INDEX)/QR(INDEX,JNDEX)
           XB(JNDEX) = BPRODUCT(INDEX)/QR(INDEX,JNDEX)
           DO 12 I = INDEX-1,1,-1
             PRODUCT(I)=PRODUCT(I)-X(JNDEX)*QR(I,JNDEX)
             BPRODUCT(I)=BPRODUCT(I)-XB(JNDEX)*QR(I,JNDEX)
12         CONTINUE
           CALL MPUT(X(JNDEX),XB(JNDEX),1,UP)
           JNDEX = JNDEX -1
           IF (INDEX.GT.1)THEN
              CALL MPUT(PRODUCT,BPRODUCT,INDEX-1,LEFT)
           ENDIF
           INDEX = INDEX -1
         ELSE
70         ROW = (INDEX -1)*ROWDIM + ROWPOS
           COL = (JNDEX -1)*COLDIM + COLPOS
C
C    IF K IS BOTH MY COLUMN AND BUT NOT MY ROW THEN GET
C    PRODUCT AND BPRODUCT ARRAY
C    FROM RIGHT, X AND XB FROM DOWN, MODIFY PRODUCT AND BPRODUCT AND
C    SEND THEM TO LEFT IF ANY PROCESSORS IN THE LEFT NEEDS THEM.
C
           IF (COL .GT. ROW) THEN
              CALL MGET(X(JNDEX),XB(JNDEX),1,DOWN)
              CALL MGET(PRODUCT,BPRODUCT,INDEX,RIGHT)
              DO 13 I = INDEX,1,-1
                PRODUCT(I)=PRODUCT(I)-X(JNDEX)*QR(I,JNDEX)
                BPRODUCT(I)=BPRODUCT(I)-XB(JNDEX)*QR(I,JNDEX)
13            CONTINUE
              IF (ROW .GT. 1) THEN
                 IF (K-ROW .LT. ROWDIM-1) THEN
```

```
                  CALL MPUT(X(JNDEX),XB(JNDEX),1,UP)
                ENDIF
              ENDIF
              CALL MPUT(PRODUCT,BPRODUCT,INDEX,LEFT)
              JNDEX = JNDEX -1
            ELSE
              INDEX = INDEX -1
              IF (INDEX .GT. 0 .AND. JNDEX .GT. 0) GO TO 70
            ENDIF
          ENDIF
        ELSE
          IF(COLPOS .EQ.1 .AND. ROWPOS .EQ.1)THEN
             CALL MGET(PRODUCT,BPRODUCT,1,RIGHT)
             MGETC = MGETC + 1
             X(1)=PRODUCT(1)/QR(1,1)
             XB(1)=BPRODUCT(1)/QR(1,1)
          ELSE
             IF (ROWPOS .EQ. 1 .AND. COLPOS .EQ. 2) THEN
               CALL MGET(X(1),XB(1),1,DOWN)
               CALL MGET(PRODUCT(1),BPRODUCT,1,RIGHT)
               MGETC = MGETC + 1
               PRODUCT(1)=PRODUCT(1)-X(1)*QR(1,1)
               BPRODUCT(1)=BPRODUCT(1)-XB(1)*QR(1,1)
               CALL MPUT(PRODUCT,BPRODUCT,1,LEFT)
             ENDIF
          ENDIF
        ENDIF
       ENDIF
10    CONTINUE
      RETURN
      END
C
      SUBROUTINE MGET(ABUFF,BBUFF,M,DIRECTION)
C
C     THIS SUBROUTINE RECEIVES 2*M ELEMENTS FROM THE PROCESSOR IN THE
C     DIRECTION 'DIRECTION', AND STORES FIRST M ELEENTS IN ABUFF AND NEXT
C     M ELEMENTS IN BBUFF
C
      DOUBLE PRECISION ABUFF(*),BBUFF(*)
      INTEGER M,DIRECTION,PID,B1,B2,CDIM,CI
      INTEGER PROC(4)
      INTEGER TYPE,COUNT
      CHARACTER*72 STRIN
C
      INTEGER DPSIZE,BUFLEN
      PARAMETER (MAXLEN=200,DPSIZE=8)
      PARAMETER (BUFLEN=2*MAXLEN)
      DOUBLE PRECISION RBUFF(BUFLEN)
C
      COMMON/CUBE/PID,MYID,CDIM,CI,B1,B2,NUMP
      COMMON/SOLVE/PROC
C
      TYPE = PROC(DIRECTION)+1100
      CALL RECVW(CI,TYPE,RBUFF,M*2*DPSIZE,COUNT,NODE.PID)
      IF (COUNT .NE. M*2*DPSIZE) THEN
         CALL SYSLOG(' PROGRAMMING ERROR, IN MSOLVE',
```

```
      ENDIF
      DO 20 I = 1,N
        ABUFF(I) = RBUFF(I)
        BBUFF(I) = RBUFF(I+N)
20    CONTINUE
      RETURN
      END
C
      SUBROUTINE NPUT(ABUFF,BBUFF,N,DIRECTION)
C
C     THIS SUBROUTINE SENDS 2*N ELEMENTS TO THE PROCESSOR IN THE
C     DIRECTION 'DIRECTION'. THE FIRST N ELEMENTS IS TAKEN FROM
C     ABUFF AND NEXT N ELEMENTS FROM BBUFF
C
      DOUBLE PRECISION ABUFF(*),BBUFF(*)
      INTEGER N,DIRECTION,PID,B1,B2,CDIM,CI
      INTEGER PROC(4)
      INTEGER TYPE,COUNT
C
      INTEGER DPSIZE,BUFLEN
      PARAMETER (MAXLEN=200,DPSIZE=8)
      PARAMETER (BUFLEN=2*MAXLEN)
      DOUBLE PRECISION RBUFF(BUFLEN)
C
      COMMON/CUBE/PID,MYID,CDIM,CI,B1,B2,NUMP
      COMMON/SOLVE/PROC
C
      TYPE = MYID + 1100
      PID=0
      DO 20 I = 1,N
        RBUFF(I) = ABUFF(I)
        RBUFF(I+N) = BBUFF(I)
20    CONTINUE
      CALL SENDW(CI,TYPE,RBUFF,N*2*DPSIZE,PROC(DIRECTION),PID)
      RETURN
      END
C
      SUBROUTINE PROCNUM (PROC,ROWPOS,COLPOS,ROWDIM,COLDIM)
C
C     ROUTINE INITIALIZES THE ARRAY 'PROC', WHICH STORES PROCESSOR NUMBERS
C     IN LEFT, RIGT, DOWN, AND UP
C
      INTEGER PROC(4),ROWPOS,COLPOS,ROWDIM,COLDIM,RIGHT,DOWN,LEFT,UP
      PARAMETER (RIGHT=1,DOWN=2,LEFT=3,UP=4)
C
      I = ROWPOS - 1
      IF (ROWPOS .EQ. 1) I = ROWDIM
      PROC(UP)= (I-1)*COLDIM+COLPOS-1
      I = ROWPOS + 1
      IF (ROWPOS .EQ. ROWDIM) I = 1
      PROC(DOWN)= (I-1)*COLDIM+COLPOS-1
      J = COLPOS-1
      IF (COLPOS .EQ. 1) J = COLDIM
      PROC(LEFT)= (ROWPOS-1)*COLDIM+J-1
      J = COLPOS+1
      IF (COLPOS .EQ. COLDIM) J = 1
```

```
      PROC(RIGHT)= (ROWPOS-1)*COLDIM+J-1
      RETURN
      END
C
      LOGICAL FUNCTION MYCOL(K,COLPOS,COLDIM)
C
C     MYCOL RETURNS TRUE IF 'K' IS MY COLUMN.
C
      INTEGER K,COLPOS,COLDIM,INDEX
      MYCOL = .FALSE.
      INDEX = K  - K/ COLDIM*COLDIM
      IF (INDEX .EQ. 0) INDEX = COLDIM
      IF (INDEX .EQ. COLPOS) MYCOL = .TRUE.
      RETURN
      END
C
      LOGICAL FUNCTION MYROW(K,ROWPOS,ROWDIM)
C
C     MYROW RETURNS TRUE IF 'K' IS MY ROW.
C
      INTEGER ROWPOS,ROWDIM,INDEX,K
      MYROW = .FALSE.
      INDEX = K  - K/ ROWDIM*ROWDIM
      IF (INDEX .EQ. 0) INDEX = ROWDIM
      IF (INDEX .EQ. ROWPOS) MYROW = .TRUE.
      RETURN
      END
C123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : NFVALD (FILE FVALD.F)
C
C     THIS SUBROUTINE EVALUATES THE JACOBIAN MATRIX. IT ACTS AS A
C     SLAVE PROCESS. IT WAITS FOR MASTER (HOST) TO SEND I, AND J  INDICES.
C     IT THEN EVALUATES AND RETURNS THE VALUE TO THE MASTER PROCESS.
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE PFJAC AND PJAC (FILE NODE.F)
C     CALLS           : FF   (FILE FEVAL.F)
C                       FJAC (FILE FEVAL.F)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE NFVALD(N,LAMBDA,A,X,DF,PAR,IPAR)
C
C     ON INPUT:
C         N  :  DIMENSION OF THE PROBLEM
C         LAMBDA : ARTIFICIAL HOMOTOPY PARAMETER
C         A      : INITIAL ARRAY THAT DEFINES THE HOMOTOPY MAP
C         X      : VECTOR AT WHICH THE F AND DF ARE TC BE EVALUATED
C         PAR    : DOUBLE PRECISION PARAMETER ARRAY (PASSED TO FF)
C         IPAR   : INTEGER PARAMETER ARRAY
C
C     ON OUTPUT: NO OUTPUT, HOWEVER, IT ACTS AS A SLAVE TO THE HOST
C               PROCESS, EVALUATES A COMPONENT OF F OR DF AND SENDS
C               BACK TO THE HOST. WHEN THE HOST INDICATES THAT THE
C               EEVALUATION OF F AND DF IS COMPLETE IT RETURNS.
```

```
C
C
      DOUBLE PRECISION X(*),DF,LAMBDA,A(*)
      DOUBLE PRECISION PAR(*)
      INTEGER IPAR(*)
      DOUBLE PRECISION RBUFF(3)
C
C           PARAMETERS
C     IPSIZE IS THE NUMBER OF BYTES TO STORE A INTEGER VALUE
C     DPSIZE IS THE NUMBER OF BYTES TO STORE A DOUBLE PRECISION VALUE
C
      INTEGER DPSIZE
      PARAMETER (IPSIZE=8,MAXPAR=2,DPSIZE=8)
C
C     CUBE VARIABLES
C
      INTEGER PID,MYID,CDIM,CI,B1,B2,NUMP,TYPE,CNT,LEN,NODE
      COMMON /CUBE/PID,MYID,CDIM,CI,B1,B2,NUMP
C
C  OTHER VARIABLES
C
      INTEGER IBUFF(MAXPAR),HOST,HPID
      LOGICAL CHECKD
      CHARACTER*72 STRIN
C
      LEN = IPSIZE
20    TYPE = 1002
      CALL RECVW(CI,TYPE,IBUFF,LEN,CNT,HOST,HPID)
      IF (IBUFF(1) .NE. -1) THEN
         INDEX = IBUFF(1)
         IF (IFLAG .NE. -2) THEN
           CALL FF(N,INDEX,X,DF,IPAR,PAR)
         ELSE
           CALL RHO(N,INDEX,X,A,LAMBDA,DF,IPAR,PAR)
         ENDIF
         RBUFF(1) = INDEX
         RBUFF(2) = DF
         TYPE = 1501
         CALL SENDW(CI,TYPE,RBUFF,DPSIZE*2,HOST,HPID)
         GOTO 20
      ENDIF
10    TYPE = 1003
      CALL RECVW(CI,TYPE,IBUFF,LEN*3,CNT,HOST,HPID)
      IF (IBUFF(1) .NE. -1) THEN
         INDEX = IBUFF(1)
         JNDEX = IBUFF(2)
         IF (IFLAG .NE. -2) THEN
           CALL FJAC(N,INDEX,JNDEX,X,DF,IPAR,PAR)
         ELSE
           CALL RHOJAC(N,INDEX,JNDEX,X,A,LAMBDA,DF,IPAR,PAR)
         ENDIF
         RBUFF(1) = INDEX
         RBUFF(2) = JNDEX
         RBUFF(3) = DF
         TYPE = 1502
         CALL SENDW(CI,TYPE,RBUFF,DPSIZE*3,HOST,HPID)
```

```
         GOTO 10
      ELSE
         RETURN
      ENDIF
      END
C12345678901234567890123456789012345678901234567890 2345678901234567890 12
C     SUBROUTINE NAME : NFVALR (FILE NFVALR.F)
C     PURPOSE         : IT EVALUATES F AND DF AT A POINT X. IT ASSUMES
C                       A RECTANGULAR MAPPING OF PROCESSORS.
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE PFJAC (FILE NODE.F)
C     CALLS           : FF   (FILE FEVAL.F)
C                       FJAC (FILE FEVAL.F)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE NFVALR(N,X,F,DF,NUMROW,NUMCOL,MYID,NUMP,ROWDIM
     1   ,ROWPOS,COLDIM,COLPOS,LAMBDA,A,PAR,IPAR,IDIM,JDIM,IFLAG)
C
C     ON INPUT:
C           N  :  DIMENSION OF THE MATRIX
C           X  :  VECTOR AT WHICH F AND DF ARE TO BE EVALUATED
C           NUMCOL : NUMBER OF COLUMN THIS NODE HOLDS
C           NUMROW : NUMBER OF ROW THIS NODE HOLDS
C           COLDIM : COLUMN DIMENSION OF THE PROCESSORS GRID
C           ROWDIM : ROW DIMENSION OF THE PROCESSORS GRID
C           COLPOS : THE COLUMN POSITION OF THIS NODE IN THE GRID
C           ROWPOS : THE ROW POSITION OF THIS NODE IN THE GRID
C           MYID   : PROCESSOR NUMBER OF THIS NODE
C           NUMP   : NUMBER OF PROCESSORS IN THE CUBE
C           LAMBDA : THE ARTIFICIAL HOMOTOPY PARAMETER
C           A      : INITIAL ARRAY THAT DEFINES HOMOTOPY MAP
C           PAR    : DOUBLE PRECISION PARAMETERS (PASSED TO FF AND FJAC)
C           IPAR   : INTEGER PARAMETERS
C           IFLAG  : TYPE OF HOMOTOPY MAPPING (SEE FIXPNF)
C           IDIM   : THE ROW DIMENSION OF DF, INTERNALLY SET BY MAIN
C           JDIM   : THE COLUMN DIMENSION OF DF, INTERNALLY SET BY MAIN
C     ON OUTPUT:
C           F      : EVALUATED FUNCTION (ONLY THOSE COMPONENTS HOLD
C                    BY THIS NODE)
C           DF     : EVALUATED JACOBIAN (ONLY THOSE COMPONENTS HOLD
C                    BY THIS NODE)
C
      DOUBLE PRECISION X(*),F(*),DF(IDIM,*),LAMBDA,ONEML,A(*)
      DOUBLE PRECISION PAR(*)
      INTEGER IPAR(*)
      INTEGER ROWDIM,COLPOS,COLDIM,ROWPOS
C           PARAMETERS
      LOGICAL CHECKD
      CHARACTER*72 STRIN
C
      ONEML = 1.0D0 - LAMBDA
      DO 90 I = 1, NUMROW
        INDEX = (I-1)*ROWDIM + ROWPOS
```

```
      IF (IFLAG .EQ. -2 ) THEN
        CALL RHO(N,INDEX,X,A,LAMBDA,F(I),IPAR,PAR)
      ELSE
        CALL FF(N,INDEX,X,F(I),IPAR,PAR)
      ENDIF
90    CONTINUE
      DO 100 I = 1, NUMROW
        INDEX = (I-1)*ROWDIM + ROWPOS
        DO 100 K = 1, NUMCOL
          KNDEX = (K-1)*COLDIM+COLPOS - 1
          IF (IFLAG .EQ. -2 ) THEN
            CALL RHOJAC(N,I,KNDEX,X,A,LAMBDA,DF(I,K),IPAR,PAR)
          ELSE
            IF (KNDEX .GT. 0) THEN
              CALL FJAC(N,INDEX,KNDEX,X,DF(I,K),IPAR,PAR)
            ENDIF
          ENDIF
100   CONTINUE
          IF (COLPOS .EQ. 1) THEN
            DO 290 KK = 1,NUMROW
              IND = (KK-1)*ROWDIM+ROWPOS
              IF (IFLAG .EQ. -1) THEN
                DF(KK,1) = F(KK) - X(IND) + A (IND)
              ELSE
                IF (IFLAG .EQ. 0) THEN
                   DF(KK,1) = A(IND) - F(KK)
                ENDIF
              ENDIF
              CHECKD = .FALSE.
              DO 290 JJ=2,NUMCOL
                DF(KK,JJ) = DF(KK,JJ)*LAMBDA
                IF (.NOT. CHECKD) THEN
                  IROW = (KK-1)*ROWDIM+ROWPOS
                  ICOL = (JJ-1)*COLDIM + COLPOS
                  IF (ICOL.GT.IROW+1) CHECKD = .TRUE.
                  IF(IROW.EQ.ICOL-1)THEN
                    IF (IFLAG .EQ. -1) THEN
                      DF(KK,JJ)=DF(KK,JJ)+ONEML
                    ELSE
                      IF (IFLAG .EQ. 0) THEN
                        DF(KK,JJ) = 1.0D0 - DF(KK,JJ)
                      ENDIF
                    ENDIF
                    CHECKD = .TRUE.
                  ENDIF
                ENDIF
290         CONTINUE
          ELSE
            DO 190 KK = 1,NUMROW
              CHECKD = .FALSE.
              DO 190 JJ=1,NUMCOL
                DF(KK,JJ) = DF(KK,JJ)*LAMBDA
                IF (.NOT. CHECKD) THEN
                  IROW = (KK-1)*ROWDIM+ROWPOS
                  ICOL = (JJ-1)*COLDIM + COLPOS
                  IF (ICOL.GT.IROW+1) CHECKD = .TRUE.
```

```
                IF(IROW.EQ.ICOL-1)THEN
                  IF (IFLAG .EQ. -1) THEN
                     DF(KK,JJ)=DF(KK,JJ)+ONEML
                  ELSE
                     IF (IFLAG .EQ. 0) THEN
                        DF(KK,JJ) = 1.0D0 - DF(KK,JJ)
                     ENDIF
                  ENDIF
                  CHECKD = .TRUE.
                ENDIF
              ENDIF
190         CONTINUE
          ENDIF
      DO 350 JJ = 1,NUMROW
       IND = (JJ-1)*ROWDIM + ROWPOS
       ONEML = X(IND) - A(IND)
       IF (IFLAG .EQ. -1) THEN
          F(JJ) =  ONEML + LAMBDA*(F(JJ)-ONEML)
       ELSE
          IF (IFLAG .EQ. 0) THEN
            F(JJ) = ONEML + LAMBDA*(A(IND) - F(JJ))
          ENDIF
       ENDIF
350    CONTINUE
      RETURN
      END
C1234567890123456789012345678901234567890123456789012345678901234567890123456789012
C     SUBROUTINE NAME : NFVALL (FILE NFVALL.F)
C     PURPOSE         : IT EVALUATES F AND DF AT A POINT X. IT ASSUMES
C                       A LINEAR MAPPING OF PROCESSORS.
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : SUBROUTINE PFJAC (FILE NODE.F)
C     CALLS           : PFJAC   (FILE NODE.F)
C                       PJAC    (FILE NODE.F)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE NFVALL(N,X,F,DF,NUMCOL,MYID,NUMP,
     1   LAMBDA,A,PAR,IPAR,IFLAG)
C
C     ON INPUT:
C           N  :  DIMENSION OF THE MATRIX
C           X  :  VECTOR AT WHICH F AND DF ARE TO BE EVALUATED
C           NUMCOL : NUMBER OF COLUMN THIS NODE HOLDS
C           MYID   : PROCESSOR NUMBER OF THIS NODE
C           NUMP   : NUMBER OF PROCESSORS IN THE CUBE
C           LAMBDA : THE ARTIFICIAL HOMOTOPY PARAMETER
C           A      : INITIAL ARRAY THAT DEFINES HOMOTOPY MAP
C           PAR    : DOUBLE PRECISION PARAMETERS (PASSED TO FF AND FJAC)
C           IPAR   : INTEGER PARAMETERS
C           IFLAG  : TYPE OF HOMOTOPY MAPPING (SEE FIXPNF)
C     ON OUTPUT:
C           F      : EVALUATED FUNCTION (ONLY THOSE COMPONENTS HOLD
C                    BY THIS NODE)
```

```
C            DF     : EVALUATED JACOBIAN (ONLY THOSE COMPONENTS HOLD
C                     BY THIS NODE)
C
      DOUBLE PRECISION X(*),F(*),DF(N,*),LAMBDA,ONEML,A(*)
      DOUBLE PRECISION PAR(*)
      INTEGER IPAR(*),IFLAG
C           PARAMETERS
      LOGICAL CHECKD
      CHARACTER*72 STRIN
C
      ONEML = 1.0D0 - LAMBDA
      DO 90 I = 1,N
      IF (IFLAG .EQ. -2) THEN
        CALL RHO(N,I,X,A,LAMBDA,F(I),IPAR,PAR)
      ELSE
        CALL FF(N,I,X,F(I),IPAR,PAR)
      ENDIF
90    CONTINUE
      DO 100 I = 1,N
      DO 100 K = 1, NUMCOL
        KNDEX = (K-1)*NUMP+MYID+1
        IF (IFLAG .EQ. -2) THEN
           CALL RHOJAC(N,I,KNDEX,X,A,LAMBDA,DF(I,K),IPAR,PAR)
        ELSE
           IF (KNDEX .GT. 1) THEN
             CALL FJAC(N,I,KNDEX-1,X,DF(I,K),IPAR,PAR)
           ENDIF
        ENDIF
100   CONTINUE
         IF (MYID .EQ. 0) THEN
           DO 290 KK = 1,N
             IF (IFLAG .EQ. -1) THEN
               DF(KK,1) = F(KK) - X(KK) + A (KK)
             ELSE
               IF (IFLAG .EQ. 0) THEN
                 DF(KK,1) = A(KK) -F(KK)
               ENDIF
             ENDIF
             CHECKD = .FALSE.
             DO 290 JJ=2,NUMCOL
               DF(KK,JJ) = DF(KK,JJ)*LAMBDA
               IF (.NOT. CHECKD) THEN
                 IROW = KK
                 ICOL = (JJ-1)*NUMP + MYID+1
                 IF (ICOL.GT.IROW+1) CHECKD = .TRUE.
                 IF(IROW.EQ.ICOL-1)THEN
                   IF (IFLAG .EQ. -1) THEN
                     DF(KK,JJ)=DF(KK,JJ)+ONEML
                   ELSE
                     IF (IFLAG .EQ. 0) THEN
                       DF(KK,JJ) = 1-DF(KK,JJ)
                     ENDIF
                   ENDIF
                   CHECKD = .TRUE.
                 ENDIF
               ENDIF
```

```
290         CONTINUE
          ELSE
            DO 190 KK = 1,N
              CHECKD = .FALSE.
              DO 190 JJ=1,NUMCOL
                DF(KK,JJ) = DF(KK,JJ)*LAMBDA
                IF (.NOT. CHECKD) THEN
                  IROW = KK
                  ICOL = (JJ-1)*NUMP + MYID+1
                  IF (ICOL.GT.IROW+1) CHECKD = .TRUE.
                  IF(IROW.EQ.ICOL-1)THEN
                    IF (IFLAG .EQ. -1) THEN
                      DF(KK,JJ)=DF(KK,JJ)+ONEML
                    ELSE
                      IF (IFLAG .EQ. 0) THEN
                        DF(KK,JJ) = 1-DF(KK,JJ)
                      ENDIF
                    ENDIF
                    CHECKD = .TRUE.
                  ENDIF
                ENDIF
190         CONTINUE
          ENDIF
      DO 350 JJ = 1,N
       ONEML = X(JJ) - A(JJ)
       IF (IFLAG .EQ. -1) THEN
         F(JJ) =  ONEML + LAMBDA*(F(JJ)-ONEML)
       ELSE
         IF (IFLAG .EQ. 0) THEN
           F(JJ) = ONEML + LAMBDA*(A(JJ)-F(JJ))
         ENDIF
       ENDIF
350    CONTINUE
      RETURN
      END
C
      PROGRAM NODEP
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C     THIS IS THE DRIVER ROUTINE FOR NODE PROCESSES
C
      INTEGER IPSIZE,DPSIZE
      PARAMETER (IPSIZE=4,DPSIZE=8)
      DOUBLE PRECISION RBUFF(1000),WBUFF(1000),PAR(1000)
      INTEGER NDIM,N,PIVOT(200),PID,CDIM,TYPE,B1,B2,MAXLEN
      INTEGER IPAR(1000)
      DOUBLE PRECISION WORK1(200),WORK2(200),X(200),XB(200)
      DOUBLE PRECISION Y(200),A(200)
      INTEGER MYID,COPEN,CI,NUMP,NUMROW,NUMCOL
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS
C
      INTEGER CUBEDIM,CNT,HOST,HPID,OFFSET,N2,EOFFST,MYPID
C
C     NUMI = NUMBER OF INTEGER PARAMETER
C     NUMR = NUMBER OF REAL PARAMETER
C
```

```
      INTEGER CHOICE(5),PCODE,NUMI,NUMR
      DOUBLE PRECISION BETA,SIGMA
      LOGICAL FIRST
      CHARACTER*72 STRIN
      COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1       ,NUMCOL,NUMROW
      COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
      MAXLEN = 150
      HOST = -32768
      MYID = MYNODE()
      PID = MYPID()
      CDIM = CUBEDIM()
      NUMP =2 ** CDIM
      TYPE = 1001
      CI = COPEN(PID)
      CALL RECVW(CI,TYPE,IPAR,1000*IPSIZE,CNT,HOST,HPID)
      NUMI = CNT/IPSIZE
      CALL RECVW(CI,TYPE,PAR,1000*DPSIZE,CNT,HOST,HPID)
      NUMR = CNT/DPSIZE
      CALL RECVW(CI,TYPE,RBUFF,1000*DPSIZE,CNT,HOST,HPID)
      N = RBUFF(1)
      CHOICE(1) = RBUFF(2)
      CHOICE(2) = RBUFF(3)
      CHOICE(3) = RBUFF(4)
      IFLAG = RBUFF(5)
      B1 = RBUFF(6)
      B2 = RBUFF(7)
      DO 10 I = 1,N
        A(I) = RBUFF(I+7)
        Y(I) = RBUFF(N+I+7)
10    CONTINUE
      Y(N+1) = RBUFF(N*2+8)
      FIRST = .TRUE.
      IF (CHOICE(2) .EQ. 1 .OR. CHOICE(3) .EQ. 0) THEN
        ROWDIM = 2  ** B1
        COLDIM = 2 ** B2
        IDIM = (N-1)/ROWDIM + 2
        JDIM = N/COLDIM+1
        ROWPOS = MYID/COLDIM + 1
        COLPOS = MYID - (ROWPOS-1)*COLDIM + 1
        NUMROW = N / ROWDIM
        NUMCOL = (N+1)/COLDIM
        IF (N - N/ROWDIM*ROWDIM .GE. ROWPOS) NUMROW = NUMROW + 1
        IF (N+1-(N+1)/COLDIM*COLDIM.GE.COLPOS)NUMCOL=NUMCOL + 1
        CALL NEBOUR(MYID,CDIM,IERR,CDIM,NEB)
        IF (CHOICE(2) .EQ. 1) THEN
          CALL PFJAC(CHOICE(3),N,NUMI,NUMR,IDIM,JDIM,IFLAG,Y,A,PAR
     1          ,IPAR,FIRST,RBUFF,WORK1,HOST,HPID,WBUFF)
        ELSE
        CALL PJAC(CHOICE(3),N,IDIM,JDIM,NUMI,NUMR,IFLAG,Y,A,PAR
     1         ,IPAR,FIRST,RBUFF,WORK1,HOST,HPID,WBUFF)
        ENDIF
      ELSE
        CALL PJAC(CHOICE(3),N,IDIM,JDIM,NUMI,NUMR,IFLAG,Y,A,PAR
     1          ,IPAR,FIRST,RBUFF,WORK1,HOST,HPID,WBUFF)
```

```
      ENDIF
      STOP
      END
C-----------------------------------------------------------------------
C     SUBROUTINE NAME : PFJAC (FILE NODE.F)
C
C     SUBROUTINE PFJAC ASSUMES THAT EVERY THING IS DONE IN PARRALEL.
C     ROUTINE IS BASICALLY AN INFINITE LOOP
C       RECEIVE X
C       EVALUATE F AND DF
C       FACTORIZE DF
C       FIND KERNEL AND MINIMUM NORM SOLN IF NEEDED
C       SEND RESULTS
C       GO BACK TO RECEIVE
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : MAIN, NODE PROCESS (FILE NODE.F)
C     CALLS           : NSOLVE   (FILE NSOLVE.F)
C                       HDCPOSE   (FILE HDCPOSE.F)
C                       NFVALL   (FILE NFVALL.F)
C                       NFVALD   (FILE NFVALD.F)
C                       NFVALR   (FILE NFVALR.F)
C-----------------------------------------------------------------------
      SUBROUTINE PFJAC(CHOICE,N,NUMI,NUMR,IDIM,JDIM,IFLAG,
     1     X,A,PAR,IPAR,FIRST,DF,F,HOST,HPID,RBUFF)
C
C    ON INPUT :
C         CHOICE = 0 IF RECTANGULAR MAPPING SHOULD BE USED IN THE
C                    EVALUATION PHASE
C                  1 IF LINEAR MAPPING SHOULD BE USED
C                  2 IF DYNAMIC ASSIGNMENT SHOULD BE USED
C         N      : DIMENSION OF THE PROBLEM
C         IPAR   : INTEGER PARAMETERS, NOT USED HERE SIMPLY PASSED
C                  TO USER WRITTEN SUBROUTINE FF() AND FJAC().
C         PAR    : DOUBLE PRECISION PARAMETERS, NOT USED HERE, SIMPLY
C                  PASSED TO USER WRITTEN SUBROUTINE FF() AND FJAC
C         NUMI   : NUMBER OF INTEGER PARAMETERS IN IPAR
C         NUMR   : NUMBER OF DOUBLE PRECISION PARAMETERS IN PAR.
C         X      : VECTOR AT WHICH THE FUNCTION AND THE JACOBIAN
C                  SHOULD BE EVALUATED.
C         A      : THE INITIAL VECTOR THAT DEFINES THE HOMOTOPY MAP.
C         HOST   : CUBE MANAGER'S ID
C         HPID   : HOST PROCESS'S PROCESS ID.
C         RBUFF(1..MAX) : WORKING ARRAY. MAX SHOULD BE LARGE ENOUGH TO
C                  HOLD BOTH DF AND F (THE PART THIS NODE HOLDS)
C         DF(IDIM,JDIM) : STORES THE EVALUATED JACOBIAN MATRIX
C         IDIM   : IS THE ROW DIMENSION OF DF
C         JDIM   : IS THE COLUMN DIMENSION OF DF
C         IFLAG  : DEFINES THE HOMOTOPY MAP
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER CHOICE,N,NUMI,NUMR,IDIM,JDIM,HOST,HPID,CNT
      DOUBLE PRECISION F(*),RBUFF(*),DF(IDIM,*),X(*),A(*)
C
```

```
C    LOCAL VARUABLES
C
      INTEGER DPSIZE
      PARAMETER (LENT=200,MAXLEN=200,DPSIZE=8,IPSIZE=4)
      DOUBLE PRECISION ALPHA(LENT),SUM(LENT),
     1  SAVE1(LENT),SAVE2(LENT),CURPIV(LENT),
     2  Y(MAXLEN),C(LENT),S(LENT),XB(LENT)
      DOUBLE PRECISION WORK1(MAXLEN),WORK2(MAXLEN)
      DOUBLE PRECISION BETA,SIGMA,LAMBDA
      CHARACTER*72 STRIN
      LOGICAL FIRST
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NEB,NUMCOL,NUMROW
      INTEGER PID,MYID,CDIM,CI,B1,B2,NUMP,PIVOT(LENT),TYPE
      COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1       ,NUMCOL,NUMROW
      COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
10    IF (.NOT. FIRST) THEN
        TYPE = 1001
        CALL RECVW(CI,TYPE,X,MAXLEN*DPSIZE,CNT,HOST,HPID)
        NUMIT = NUMIT + 1
      ELSE
        FIRST = .FALSE.
        NUMIT = 1
      ENDIF
      LAMBDA = X(1)
      IF (CHOICE .EQ. 0)  THEN
       CALL NFVALR(N,X(2),F,DF,NUMROW,NUMCOL,MYID,
     1     NUMP,ROWDIM,ROWPOS
     1     ,COLDIM,COLPOS,LAMBDA,A,PAR,IPAR,IDIM,JDIM,IFLAG)
        DO 95 II = 1,IDIM
          DO 95 JJ = 1,JDIM
            IF (JJ .GT. NUMCOL .OR. II .GT. NUMROW) THEN
              DF(II,JJ) = 0.0
            ENDIF
95       CONTINUE
      ELSE
        IF (CHOICE .EQ. 1) THEN
          LCOL = (N+1)/NUMP
          IF ((N+1)-LCOL*NUMP .GT. MYID)LCOL = LCOL + 1
          NOFF = LCOL*N
          IF (LCOL .NE. 0) THEN
            CALL NFVALL(N,X(2),RBUFF(NOFF+1),RBUFF,LCOL,MYID,NUMP,
     1         LAMBDA,A,PAR,IPAR,IFLAG)
          ENDIF
          TYPE = 1501
          CALL SENDW(CI,TYPE,RBUFF,DPSIZE*N*(LCOL+1),HOST,HPID)
          TYPE = 1001
          CALL RECVW(CI,TYPE,RBUFF,1000*DPSIZE,CNT,HOST,HPID)
          DO 35 II = 1,IDIM
            DO 35 JJ = 1,JDIM
              IF (JJ .GT. NUMCOL .OR. II .GT. NUMROW) THEN
                DF(II,JJ) = 0.0
              ELSE
                INDEX = (JJ-1)*NUMROW + II
                DF(II,JJ) = RBUFF(INDEX)
              ENDIF
```

```
35        CONTINUE
          NOFF = NUMCOL*NUMROW
          DO 145 II = 1,IDIM
            F(II) = RBUFF(NOFF + II)
145        CONTINUE
        ELSE
          CALL NFVALD(N,LAMBDA,A,X(2),DF,PAR,IPAR)
          TYPE = 1001
          CALL RECVW(CI,TYPE,RBUFF,1000*DPSIZE,CNT,HOST,HPID)
          DO 535 II = 1,IDIM
             DO 535 JJ = 1,JDIM
               IF (JJ .GT. NUMCOL .OR. II .GT. NUMROW) THEN
                 DF(II,JJ) = 0.0
               ELSE
                 INDEX = (JJ-1)*NUMROW + II
                 DF(II,JJ) = RBUFF(INDEX)
               ENDIF
535         CONTINUE
          NOFF = NUMCOL*NUMROW
          DO 645 II = 1,IDIM
            F(II) = RBUFF(NOFF + II)
645        CONTINUE
        ENDIF
      ENDIF
      N2 = 2 * N
      CALL HDCPOSE(IDIM,JDIM,N,DF,ALPHA,PIVOT,IERR,
     1   F,SUM,SAVE1,SAVE2,CURPIV,WORK1,MAXLEN
     2  ,C,S)
449    CONTINUE
      CALL NSOLVE(IDIM,JDIM,N,F,DF,WORK1,WORK2,PIVOT,
     1   X,XB,ALPHA)
      DO 120 I = 1,NUMCOL
         WORK1(I) = PIVOT(I)
         WORK1(I+NUMCOL) = X(I)
         WORK1(I+2*NUMCOL) = XB(I)
120    CONTINUE
      TYPE = 1501
      CALL SENDW(CI,TYPE,WORK1,(3*NUMCOL)*DPSIZE,HOST,HPID)
      GOTO 10
      RETURN
      END
C-----------------------------------------------------------------------
C     SUBROUTINE NAME : PJAC (FILE NODE.F)
C
C SUBROUTINE PJAC ASSUMES THAT ONLY FUNCTION EVALUATION  IS DONE IN PARRALEL.
C     THE ROUTINE IS BASICALLY AN INFINITE LOOP
C       RECEIVE X
C       EVALUATE F AND DF
C       SEND RESULTS
C       GO BACK TO RECEIVE
C
C     AUTHOR          : AMAL CHAKRABORTY
C     LAST UPGRADED   : 5/24/90
C     INPUT FILE      : NONE
C     OUTPUT FILE     : NONE
C     CALLED BY       : MAIN, NODE PROCESS (FILE NODE.F)
```

```
C     CALLS          : NFVALL   (FILE NFVALL.F)
C                      NFVALD   (FILE NFVALD.F)
C                      NFVALR   (FILE NFVALR.F)
C-------------------------------------------------------------------
C
      SUBROUTINE PJAC(CHOICE,N,IDIM,JDIM,NUMI,NUMR,IFLAG,X,A,
     *           PAR,IPAR,FIRST,DF,F,HOST,HPID,RBUFF)
C
C    ON INPUT :
C         CHOICE = 0 IF RECTANGULAR MAPPING SHOULD BE USED IN THE
C                    EVALUATION PHASE
C                  1 IF LINEAR MAPPING SHOULD BE USED
C                  2 IF DYNAMIC ASSIGNMENT SHOULD BE USED
C         N      : DIMENSION OF THE PROBLEM
C         IPAR   : INTEGER PARAMETERS, NOT USED HERE SIMPLY PASSED
C                  TO USER WRITTEN SUBROUTINE FF() AND FJAC().
C         PAR    : DOUBLE PRECISION PARAMETERS, NOT USED HERE, SIMPLY
C                  PASSED TO USER WRITTEN SUBROUTINE FF() AND FJAC
C         NUMI   : NUMBER OF INTEGER PARAMETERS IN IPAR
C         NUMR   : NUMBER OF DOUBLE PRECISION PARAMETERS IN PAR.
C         X      : VECTOR AT WHICH THE FUNCTION AND THE JACOBIAN
C                  SHOULD BE EVALUATED.
C         A      : THE INITIAL VECTOR THAT DEFINES THE HOMOTOPY MAP.
C         HOST   : CUBE MANAGER'S ID
C         HPID   : HOST PROCESS'S PROCESS ID.
C         RBUFF(1..MAX) : WORKING ARRAY. MAX SHOULD BE LARGE ENOUGH TO
C                  HOLD BOTH DF AND F (THE PART THIS NODE HOLDS)
C         DF(IDIM,JDIM) : STORES THE EVALUATED JACOBIAN MATRIX
C         IDIM   : IS THE ROW DIMENSION OF DF
C         JDIM   : IS THE COLUMN DIMENSION OF DF
C         IFLAG  : DEFINES THE HOMOTOPY MAP
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (MAXLEN=200)
      INTEGER CHOICE,N,NUMI,NUMR,IDIM,JDIM,HOST,HPID
      DOUBLE PRECISION DF(IDIM,*),X(MAXLEN),F(*),PAR(*),LAMBDA
      INTEGER IPAR(*)
      DOUBLE PRECISION RBUFF(*)
C
C     GLOBAL CUBE INFORMATION
C
      INTEGER DPSIZE
      PARAMETER (LENT=200,DPSIZE=8,IPSIZE=4)
      INTEGER ROWDIM,COLDIM,ROWPOS,COLPOS,NEB,NUMCOL,NUMROW
      INTEGER PID,MYID,CDIM,CI,B1,B2,NUMP,TYPE
      LOGICAL FIRST
      COMMON /QRFAC/ROWDIM,COLDIM,ROWPOS,COLPOS,NEB(10)
     1       ,NUMCOL,NUMROW
      COMMON /CUBE/ PID,MYID,CDIM,CI,B1,B2,NUMP
C
C
      HOST = -32768
10    IF (.NOT. FIRST) THEN
        TYPE = 1001
        CALL RECVW(CI,TYPE,X,150*DPSIZE,CNT,HOST,HPID)
      ELSE
```

```
        FIRST  = .FALSE.
      ENDIF
      LAMBDA = X(1)
      IF (CHOICE .EQ. 0)  THEN
        CALL NFVALR(N,X(2),F,DF,NUMROW,NUMCOL,MYID,NUMP,ROWDIM,ROWPOS
     1     ,COLDIM,COLPOS,LAMBDA,A,PAR,IPAR,IDIM,JDIM,IFLAG)
        TYPE = 1501
        DO 19 I = 1,NUMROW
          DF(I,NUMCOL+1) = F(I)
19      CONTINUE
        CALL SENDW(CI,TYPE,DF,DPSIZE*(IDIM*(NUMCOL+1)),HOST,HPID)
      ELSE
        IF (CHOICE .EQ. 1) THEN
          LCOL = (N+1)/NUMP
          IF ((N+1)-LCOL*NUMP .GT. MYID)LCOL = LCOL + 1
          NOFF = LCOL*N
          IF (LCOL .NE. 0) THEN
            CALL NFVALL(N,X(2),RBUFF(NOFF+1),RBUFF,LCOL,MYID,NUMP,
     1         LAMBDA,A,PAR,IPAR,IFLAG)
          ENDIF
          TYPE = 1501
          CALL SENDW(CI,TYPE,RBUFF,DPSIZE*N*(LCOL+1),HOST,HPID)
        ELSE
          CALL NFVALD(N,LAMBDA,A,X(2),DF,PAR,IPAR)
        ENDIF
      ENDIF
      GOTO 10
      RETURN
      END
```

Vita

Amal Chakraborty was born July 25, 1959 in West Bengal, India. He was educated locally and passed the Higher Secondary Examination, conducted by the West Bengal Board of Secondary Education, in 1976. Chakraborty graduated from Benares Hindu University in Mining Engineering in 1982 and got his master of science degree in Mining Engineering at Virginia Polytechnic Institute and State University in 1985. He completed his master of science degree in Computer Science in 1988

PARALLEL HOMOTOPY CURVE TRACKING ON A HYPERCUBE

by

Amal Chakraborty

Committee Chairman: Dr. Layne Watson

Department of Computer Science

(ABSTRACT)

Probability-one homotopy methods are a class of methods for solving non-linear systems of equations that are globally convergent with probability one from an arbitrary starting point. The essence of these algorithms is the construction of an appropriate homotopy map and subsequent tracking of some smooth curve in the zero set of the homotopy map. Tracking a homotopy zero curve requires calculating the unit tangent vector at different points along the zero curve. Because of the way a homotopy map is constructed, the unit tangent vector at each point in the zero curve of a homotopy map $\rho_a(\lambda, x)$ is in the one-dimensional kernel of the full rank $n \times (n+1)$ Jacobian matrix $D\rho_a(\lambda, x)$. Hence, tracking a zero curve of a homotopy map involves evaluating the Jacobian matrix and finding the one-dimensional kernel of the $n \times (n+1)$ Jacobian matrix with rank n. Since accuracy is important, an orthogonal factorization of the Jacobian matrix is computed. The QR and LQ factorizations are considered here. Computational results are presented showing the performance of several different parallel orthogonal factorization/triangular system solving algorithms on a hypercube, in the context of parallel homotopy algorithms for problems with small, dense Jacobian matrices. This study also examines the effect of different component complexity distributions and the size of the Jacobian matrix on the different assignments of components to the processors, and determines in what context one assignment would perform better than others.