Adaptive Data Distribution for Concurrent Continuation

E. F. Van de Velde
J. Lorenz

CRPC-TR89013
August, 1989

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

CRPC-89-4-revised August 30, 1989

Adaptive Data Distribution for Concurrent Continuation*

Eric F. Van de Velde and Jens Lorenz

Applied Mathematics 217-50
California Institute of Technology
Pasadena, California 91125

* *This work is supported in part by Department of Energy Grant No. DE-AS03-76ER72012. This material is based upon work supported by the NSF under Cooperative Agreement No. CCR-8809615. The government has certain rights in this material.*

Abstract

Continuation methods compute paths of solutions of nonlinear equations that depend on a parameter. This paper examines some aspects of the multicomputer implementation of such methods. The computations are done on a mesh connected multicomputer with 64 nodes.

One of the main issues in the development of concurrent programs is load balancing, achieved here by using appropriate data distributions. In the continuation process, many linear systems have to be solved. For nearby points along the solution path, the corresponding system matrices are closely related to each other. Therefore, pivots which are good for the LU-decomposition of one matrix are likely to be acceptable for a whole segment of the solution path. This suggests to choose certain data distributions that achieve good load balancing. In addition, if these distributions are used, the resulting code is easily vectorized.

To test this technique, the invariant manifold of a system of two identical nonlinear oscillators is computed as a function of the coupling between them. This invariant manifold is determined by the solution of a system of nonlinear partial differential equations that depends on the coupling parameter. A symmetry in the problem reduces this system to one single equation, which is discretized by finite differences. The solution of the discrete nonlinear system is followed as the coupling parameter is changed.

1 Introduction

Consider a system of M equations:

$$G(\mathbf{u}, \lambda) = \mathbf{0} \tag{1}$$

for $\mathbf{u} \in \mathbf{R}^M$, which depends on a parameter $\lambda \in \mathbf{R}$. Here,

$$G : \mathbf{R}^M \times \mathbf{R} \longrightarrow \mathbf{R}^M$$

is a smooth map. By a solution branch we mean a one parameter family

$$(\mathbf{u}(s), \lambda(s)) \in \mathbf{R}^M \times \mathbf{R}, \quad s_a \leq s \leq s_b \tag{2}$$

of solutions of (1) depending smoothly on some parameter $s \in [s_a, s_b]$. Because of the importance for applications, many numerical methods have been devised and investigated to compute such branches; see, e.g., [3,8,10].

In this paper, we discuss some aspects of the implementation of such methods on multicomputers. In particular, we address the question of good data distributions for the linear systems that need to be solved during the continuation procedure. Assuming that the branch (2) contains only regular points and simple folds, one has to solve linear systems whose matrices have the form:

$$\begin{bmatrix} A & \mathbf{b} \\ \mathbf{c}^T & \delta \end{bmatrix} \in \mathbf{R}^{(M+1)\times(M+1)}$$

where A is $M \times M$.

These bordered systems can be solved efficiently once an LU-decomposition of A is known. In general, for reasons of numerical stability, it is necessary to compute such a decomposition with pivoting. To do this on multicomputers, the algorithm described in [13] is used. As already noted in [13], the efficiency of the decomposition depends crucially on an interplay between the pivot locations and the distribution of the matrix entries over the concurrent processes. For continuation problems, it is reasonable to believe that the pivot locations can be kept constant along a whole piece of the branch. This is indeed confirmed by our experience. Hence, the pivot search cost is eliminated for the majority of the LU-decompositions. In addition, with the pivots known in advance, the data distribution can be chosen to achieve optimal load balance. Because the pivots can be kept constant along a whole piece of the branch, an adaptation of the data distribution to new pivot locations is necessary only occasionally.

In continuation methods, it is thus possible to reduce a dynamic to a static pivoting strategy without significant loss of numerical stability. Here, we use this technique for an example with linear systems whose sparsity structure we do not exploit. Nonetheless, the technique of projecting a dynamic to a static pivoting strategy is valuable also in a sparse context: At the start of the continuation process, a dynamic sparse strategy is applied which achieves numerical stability while limiting (or minimizing) the amount of fill. In this step, the issue of load balance is ignored. Then, for the subsequent LU-decompositions, the predetermined pivot locations and the predetermined sparse matrix structure are used together with a corresponding data distribution which achieves good load balance. Numerical stability is monitored for these decompositions. Only occasionally, when numerical stability is not achieved, the dynamic routine is applied again. The present paper investigates this approach, ignoring the issue of sparsity, however.

An outline of the paper follows. In Section 2, we describe the mathematical aspects of the concurrent LU-decomposition algorithm. In Section 3, we explain the terminology concerning data distributions on multicomputers and apply the concepts to the LU-decomposition algorithm. Sections 4 and 5 contain some details on the continuation and bordering algorithms which are used to test the strategy.

Sections 6 and 7 give performance and numerical stability results for a test problem, namely the numerical calculation of the invariant manifold of a dynamical system. We have chosen such a — rather involved — example, because it leads to matrices in the continuation process which have no apparent symmetry or diagonal dominance properties. Therefore, the pivot locations do not follow easily from structure properties of the matrices. The concept of invariant manifold is explained in Section 8, and the dynamical system for which we compute it is given in Section 9. The numerical results of the computations are given in Section 10.

In this paper, we make essential use of concepts from computer science (multicomputers, data distribution) and from scientific computing (continuation for nonlinear equations, LU-decomposition). Some necessary background material and explanations of terminology are included to make this paper accessible to numerical analysts as well as computer scientists.

2 LU-Decomposition

To clarify terminology, we describe here (a version of) the LU-decomposition algorithm. For its implementation on multicomputers we refer to [13]. The main characteristic of the presented version is the following: though arbitrary pivoting strategies can be incorporated, an explicit interchange of rows and columns is not required. The main advantage of this formulation is that it allows an implementation which is, to a large extent, *independent of the data distribution*; see [13]. A second advantage is that *communication costs are reduced* if explicit row and column interchanges are avoided.

Let A be an $M \times M$ matrix with entries $a_{m,n}$, $0 \leq m < M$ and $0 \leq n < M$. Let r_k, $0 \leq k < M$, be the row index and c_k, $0 \leq k < M$, the column index of the pivot of the k-th elimination step. The permutations r_k and c_k may either be known in advance or may be determined during the course of the elimination. In Figure 1, pseudo-code is given for the LU-decomposition algorithm with implicit pivoting, i.e., without explicit interchanges of rows and columns. The "pivot search" in this code refers to the computation necessary to determine the pivot if the pivots are not preset.

The next theorem follows from a well known result by permutation:

Theorem 1 *Let R and C denote the permutation matrices corresponding to the permutations r_k and c_k, i.e., for $0 \leq k < M$ and for all vectors $\mathbf{x} \in \mathbf{R}^M$:*

$$(R\mathbf{x})_k = x_{r_k} \text{ and } (C\mathbf{x})_k = x_{c_k}.$$

If the algorithm of Figure 1 runs to completion, then it overwrites the original matrix $A = A^{(0)}$ by the matrix:

$$A^{(M-1)} = (L - R^T C) + U,$$

where one defines the matrices L and U by their entries $\ell_{m,n}$ and $u_{m,n}$, $0 \leq m < M$ and $0 \leq n < M$, as follows:

$$\ell_{r_m,c_n} = \begin{cases} a^{(M-1)}_{r_m,c_n} & \text{if } n < m \\ 1 & \text{if } n = m \\ 0 & \text{otherwise} \end{cases}$$

$$u_{r_m,c_n} = \begin{cases} a^{(M-1)}_{r_m,c_n} & \text{if } n \geq m \\ 0 & \text{otherwise.} \end{cases}$$

With these definitions, the original matrix A has the decomposition:

$$A = LC^T RU. \tag{3}$$

```
$\mathcal{M} := \{m : 0 \le m < M\}$ ;
$\mathcal{N} := \{n : 0 \le n < M\}$ ;
for $k = 0, 1, \ldots, M-1$ do begin
    {Pivot Strategy and Bookkeeping.}
    do pivot search and find $a_{rc}, r[k], c[k]$ ;
    $\mathcal{M} := \mathcal{M} \setminus \{r[k]\}$ ;
    $\mathcal{N} := \mathcal{N} \setminus \{c[k]\}$ ;
    if $a_{rc} = 0.0$ then terminate ;

    {Calculation of the Multiplier Column.}
    for all $m \in \mathcal{M}$ do
        $a[m, c[k]] := a[m, c[k]]/a_{rc}$ ;

    {Elimination.}
    for all $(m, n) \in \mathcal{M} \times \mathcal{N}$ do
        $a[m, n] := a[m, n] - a[m, c[k]]a[r[k], n]$
end
```

Figure 1: LU-Decomposition with implicit pivoting.

The matrix $\hat{L} = RLC^T$ is unit lower triangular and $\hat{U} = RUC^T$ is upper triangular. The matrices L and U are called *permuted triangular*. As is well known, once the decomposition (3) is computed, the linear system $A\mathbf{x} = \mathbf{b}$ is easily solved.

It remains to discuss the strategy for the choice of the pivot indices r_k and c_k. There are two aspects of importance, namely numerical stability and computational costs. In this paper, we restrict ourselves to only two different pivoting strategies, which are, in some sense, opposite to each other. The first strategy, called *complete pivoting*, requires to determine r_k, c_k before each elimination step in such a way that the pivot element a_{r_k,c_k} is as large as possible in absolute value. This strategy is usually good from the perspective of numerical stability, but the computational costs are high. The second strategy, called *preset pivoting*, is the trivial strategy to accept the permutations r_k, c_k as known (from previous calculations) before the LU-decomposition algorithm for A starts. Clearly, there is the danger of numerical instability or even breakdown of the algorithm. On the other hand, there are obviously no search costs; more importantly, with preset pivoting one can choose an optimal data distribution, as will be explained in Section 3.

For reasons of reliability, one would like to have a convenient measure to evaluate the numerical stability of the algorithm with preset pivoting. If $a_{m,n}$ denotes entries of the original matrix A and $a_{m,n}^{M-1}$ entries of the overwritten matrix $A^{(M-1)} = (L - R^TC) + U$, then we compute the *growth factor* γ as the ratio given by:

$$\gamma = \frac{\max_{0 \leq m,n < M} |a_{m,n}^{M-1}|}{\max_{0 \leq m,n < M} |a_{m,n}|}. \tag{4}$$

If γ is large, there is the danger for numerical instability. A theoretically sounder, but — for applications — prohibitively expensive growth factor is given in [14]. For a recent discussion of the significance of growth factors to numerical stability; see [7].

3 Data Distribution

3.1 Terminology

On multicomputers, the prevalent paradigm for concurrency is to use a number of sequential processes that communicate with each other. For the purposes of the following discussion, the number of processes is kept fixed. Each process has a unique identifier, which is supplied by the multicomputer operating system, and which is used as an address in the exchange of messages. No a priori assumptions are made about the physical location of the processes on multicomputer nodes. It is often convenient to map the system-supplied identifier into a user-defined identification. E.g., for vector calculations the processes are organized as a one-dimensional process grid. The user identification for each process is then a number p between 0 and $P-1$, where P is the number of processes. A multicomputer program operating on a vector, say of dimension M, must distribute the vector entries over the P processes. A distribution allocates each vector entry to a particular process, e.g., it maps the m-th entry to process $p = p(m)$. The collection of entries allocated to one process form a local vector. Each entry of this local vector corresponds to exactly one entry of the global vector, e.g., the i-th local entry in process p is the m-th global entry. A vector distribution μ is thus a map from the global index m, where $0 \le m < M$, to an index pair $\mu(m) = (p, i)$ consisting of a process number $p = p(m)$ and a local index $i = i(m)$. Two often used maps are the *linear distribution* given by:

$$\left\{ \begin{array}{lcl} p(m) & = & \max(\left\lfloor \frac{m}{L+1} \right\rfloor, \left\lfloor \frac{m-R}{L} \right\rfloor) \\ i(m) & = & m - p(m)L - \min(p(m), R) \end{array} \right. \tag{5}$$

and the *scatter distribution* given by:

$$\left\{ \begin{array}{lcl} p(m) & = & m \bmod P \\ i(m) & = & \lfloor \frac{m}{P} \rfloor, \end{array} \right. \tag{6}$$

where $L = \left\lfloor \frac{M}{P} \right\rfloor$ and $R = M \bmod P$. (For the linear distribution, it is assumed in (5) that $L \neq 0$ or, equivalently, $P \le M$. If $P > M$ then $p(m) = m$ and $i(m) = 0$.) The notation $\lfloor x \rfloor$ means the greatest integer less than or equal to x. An example is displayed in Table 1: the index range $0 \le m < 10$ is distributed over 4 processes.

For matrix calculations, the processes are organized in a rectangular grid such that each process is identified by two coordinates (p, q), where $0 \le p < P$ and $0 \le q < Q$. The total number of processes is thus $P \times Q$. All processes with coordinates (p, q) where $0 \le q < Q$ form the *p-th process row* and all processes with coordinates (p, q) where $0 \le p < P$ form the *q-th process column*. A matrix distribution is defined as the Cartesian product of two vector distributions μ and ν. The rows of the matrix are distributed by

	Linear		Scatter	
Global	Process	Local	Process	Local
0	0	0	0	0
1	0	1	1	0
2	0	2	2	0
3	1	0	3	0
4	1	1	0	1
5	1	2	1	1
6	2	0	2	1
7	2	1	3	1
8	3	0	0	2
9	3	1	1	2

Table 1: Four fold linear and scatter distribution.

μ over the P process rows. Similarly, the columns are distributed by ν over the Q process columns. Thus, if $\mu(m) = (p, i)$ and $\nu(n) = (q, j)$, the matrix entry with global row and column indices m and n is found in process (p, q) as local matrix entry $a_{i,j}$.

Using only the linear and the scatter distributions, a matrix can be distributed in a variety of ways. Consider, e.g., the distribution of a matrix over four processes. They may be organized as a 4×1, a 2×2, or a 1×4 process grid. Then, if either the linear or the scatter distribution to rows and columns is applied, the following 8 distributions are possible:

4×1 linear-,	2×2 linear-linear,
4×1 scatter-,	2×2 scatter-linear,
1×4 -linear,	2×2 linear-scatter,
1×4 -scatter,	2×2 scatter-scatter.

Here, we list $P \times Q$, the type of row, and the type of column distribution. (Note that the linear and the scatter distribution are the same when $P = 1$ or $Q = 1$.) A 5×7 matrix $A = [a_{m,n}]$ distributed over a 2×2 process grid with a linear row and a scatter column distribution is stored according to:

$$\begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix} = \left[\begin{matrix} \begin{bmatrix} a_{0,0} & a_{0,2} & a_{0,4} & a_{0,6} \\ a_{1,0} & a_{1,2} & a_{1,4} & a_{1,6} \\ a_{2,0} & a_{2,2} & a_{2,4} & a_{2,6} \end{bmatrix} & \begin{bmatrix} a_{0,1} & a_{0,3} & a_{0,5} \\ a_{1,1} & a_{1,3} & a_{1,5} \\ a_{2,1} & a_{2,3} & a_{2,5} \end{bmatrix} \\ \begin{bmatrix} a_{3,0} & a_{3,2} & a_{3,4} & a_{3,6} \\ a_{4,0} & a_{4,2} & a_{4,4} & a_{4,6} \end{bmatrix} & \begin{bmatrix} a_{3,1} & a_{3,3} & a_{3,5} \\ a_{4,1} & a_{4,3} & a_{4,5} \end{bmatrix} \end{matrix} \right]$$

where $A_{p,q}$ is the submatrix of A stored in process (p, q). The local indices (i, j) corresponding to the global indices (m, n) are determined by the position of the entry $a_{m,n}$ in its submatrix.

3.2 Data Distribution for LU-Decomposition

Our concurrent LU-decomposition program is valid for any distribution of the matrix represented with functions like μ and ν, not necessarily linear or scatter. As shown in [13], the choice of distribution influences the efficiency of the execution considerably. Fine tuning the data distribution is usually crucial to obtain an effective concurrent program. Distribution functions like the linear and scatter distributions (5) and (6) are *static*: they depend only on the dimension of the problem and the number of processes (i.e., on M, N, P, and Q, assumed fixed). *Adaptive or dynamic* data distributions also depend on the actual computation itself.

Adaptive distributions may increase the concurrent efficiency by increasing the load balance. To make this plausible, consider the LU-decomposition algorithm in somewhat greater detail. Each decomposition step deactivates the pivot row and the pivot column of the step. Consequently, once a row (column) has been chosen as a pivot row (column) it is not accessed for the remainder of the decomposition. Therefore, a bad load balance results if successive pivot rows (columns) fall into the same process row (column). How can one obtain a good load balance? Suppose the number of process rows P and the number of process columns Q is kept fixed, and the load balance of LU-decompositions for all possible distributions of the type $\mu \times \nu$ is compared. Then, for full matrices, the best load balance is obtained if:

1. within divisibility constraints, an equal number of matrix rows (columns) is allocated to each process row (column),

2. the order in which matrix rows (columns) become inactive is such that always after P (Q) steps, one row (column) has become inactive in each process row (column).

The first condition is easily fulfilled by many static distributions, e.g., by the linear and scatter distributions discussed earlier. The second condition, however, connects the desired data distribution to the order of the pivots. It can be satisfied if the ***locations of the pivots*** are available prior to the LU-decomposition. To explain this, assume that the location of the k-th pivot is known to be at the global row index r_k and the global column index c_k. Then, the second condition is satisfied by choosing the row and column distribution given by:

$$\mu(r_k) = (k \bmod P, \lfloor k/P \rfloor) \quad \text{and} \quad \nu(c_k) = (k \bmod Q, \lfloor k/Q \rfloor). \tag{7}$$

For LU-decomposition without pivoting, i.e., $r_k = k$ and $c_k = k$, these coincide with the scatter distribution (6).

3.3 Remark on Vectorization

When the pivot locations are known in advance, the matrix distribution (7) is also optimal from the perspective of vectorization. Consider the set of feasible rows in one arbitrary but fixed process during the course of the LU-decomposition. (A row is feasible as long as some of its entries are still allowed to be pivots.) The set of feasible row indices starts out as an interval, i.e., $\mathcal{I} = \{i : 0 \leq i < I\}$. Note that this is a set of local indices. Each elimination step makes one global row infeasible. If this row is represented in the process under consideration, its local row index is deleted from the index set $\mathcal{I}$. In general, there is no a priori information about the order in which local indices are removed. For the given data distribution, however, it will be the first index in the set. The feasible row index set thus *remains an interval.* The same holds for the feasible column index set. As a result, all computations throughout the LU-decomposition can be done on contiguous blocks of data, and vectorization is straightforward. The Symult Series 2010 does not have vector hardware, and the full effect of this observation has not been assessed at this time. On current nodes, the only effect of the above observation is a reduction in administrative overhead because loop traversal is simplified.

3.4 Vector Distributions in Matrix Calculations

Scalars and vectors that occur in the same operations as a matrix must be distributed compatibly with the given matrix distribution. Scalars are easily taken care of by duplication, i.e., their values are stored in every process.

Vector distributions compatible with matrix distributions are more complicated, because the desired distribution depends on the operation which will be performed. E.g., consider the implementation of the following assignment, which requires the evaluation of a matrix vector product:

$$\mathbf{b} := A\mathbf{x}.$$

Entry x_n of the vector $\mathbf{x}$ multiplies the n-th column of A. Hence, x_n must be known in every process where a segment of the n-th column of A is stored. This implies that $\mathbf{x}$ must be duplicated in each process row, and within each process row, it must be distributed according to the column distribution ν of the matrix A. Similarly, it is seen that storage space for the vector $\mathbf{b}$ must be assigned in a dual fashion, i.e., duplicated in each process column and distributed across process rows. Although conflicting with linear algebra conventions, we refer below — for brevity — to the vector $\mathbf{x}$ as a *row vector* (since it appears in each process row) and to $\mathbf{b}$ as a *column vector* (since it appears in each process column). These two types of vector distributions are sufficient to implement all matrix vector operations necessary for the whole continuation algorithm. In particular, the back-solve algorithm, which generally follows an LU-decomposition, uses a column vector as right hand side and produces the solution in a row vector.

3.5 Data Redistribution

In a program with adaptive data distribution, it is, in general, necessary to perform some data redistribution when switching distributions. In the continuation program, e.g., successive iterations add corrections to the current estimate of the solution vector. If the correction and the solution are identically distributed, summing them is easy and trivially concurrent. After changing distributions, the solution vector is redistributed to keep its distribution identical to that of future corrections.

Because row vectors are duplicated in every process row, the row distribution μ can be changed without affecting data stored in row vectors. Similarly, data in column vectors are not affected by changes in the column distribution ν. Thus, before changing the column distribution, row vectors are copied into temporary column vectors. Once ν has been changed, the vectors are copied back from the temporary column vectors to the original row vectors. A similar strategy is applied to column vectors when the row distribution is changed.

To copy a row into a column vector (the reverse operation is analogous), the column vector is initialized to zero. This is followed by a *local assignment phase*: all local row vector entries that can be assigned to local column vector entries are assigned. I.e., in process (p, q) local row vector entry j is assigned to local column vector entry i if and only if the corresponding global vector indices are equal, i.e., if and only if:

$$\nu^{-1}(q,j) = \mu^{-1}(p,i).$$

This is achieved by running through all local row vector entries j, computing their global index $n = \nu^{-1}(q,j)$, computing $(\hat{p}, i) = \mu(n)$, and assigning the j-th row vector entry to the i-th column vector entry if $p = \hat{p}$. Consider now only processes of one arbitrary but fixed process row. Every row vector entry is stored exactly once in the process row. Only a subset of the column vector entries is stored, however, and those column vector entries are duplicated in each process of the process row. During the local assignment phase, the process row as a whole considered each row vector entry exactly once and made the corresponding assignment to the column vector section if appropriate. Thus, within the process row, each assignment to the column vector section was made exactly once. Each of the individual processes, however, has only a partially initialized column vector section. In the *global assignment phase*, these partially initialized sections of the column vector are combined by a standard recursive doubling procedure over the process columns. In such a procedure, pairs of process columns are recursively combined. This requires $\log_2 Q$ sequential steps; each step consists of $Q/2$ concurrent data exchanges.

3.6 Choice of Process Grid

The discussion so far left the number of processes and the choice of process grid $P \times Q$ open. Here, we examine some factors that influence practical choices for P and Q.

For computations of medium and large grain concurrency on multicomputers like the Symult Series 2010, the total number of processes is usually fixed and equal to the number of processors in the multicomputer. For computations with a high communication overhead on architectures with low cost context switching, increasing the process to processor ratio may be an effective way to hide the communication cost. For the computations discussed here, the overall communication cost is too small and the context switching overhead too large for such strategies to be considered. Hence, the total number of processes is kept fixed, and we examine how best to organize them.

The choice of process grid has a bearing on the communication cost and the memory requirements of the program. The *fixed communication cost*, i.e., the part that depends only on the number of messages sent, is reduced when either $P = 1$ or $Q = 1$ because the broadcast of the pivot row (if $P = 1$) or the pivot column (if $Q = 1$) is avoided in each step. The *marginal communication cost*, i.e., that part of the communication cost that increases with the message length, is minimized by choosing a process grid in which $P \approx Q$, see [6,13], because such a choice minimizes the message length of pivot row and column broadcasts. Here, we take $P \approx Q$ to mean equal within divisibility constraints. *Memory usage* is minimized when choosing $P \approx Q$ because this leads to a minimum amount of duplication in vector storage, at least in programs with about an equal number of row and column vectors. To see this, consider a program with one row and one column vector, each of dimension M. Because a row vector is duplicated P times and a column vector Q times, the total memory usage is $(P+Q)M$ words. With the total number of processes $P \times Q$ fixed, this is minimized if $P \approx Q$. On the Symult Series 2010 the fixed communication cost is so small that it may be omitted from consideration, and hence, two dimensional distributions with $P \approx Q$ are preferred.

4 The Continuation Procedure

In this section, we describe briefly the continuation process that we have implemented. The basic ideas are well-established; see [8,10]. In actual calculations, one has to specify — among others — the stopping criterion for Newton's method and the strategy for determining the arc-length of the continuation steps. To avoid ambiguity, we shall also give details on these issues below.

We consider a smooth map

$$G : \mathbf{R}^M \times \mathbf{R} \longrightarrow \mathbf{R}^M$$

and assume that the system of equations

$$G(\mathbf{u}, \lambda) = \mathbf{0}, \tag{8}$$

has a smooth solution branch

$$(\mathbf{u}(s), \lambda(s)) \in \mathbf{R}^M \times \mathbf{R}, \quad s_a \leq s \leq s_b, \tag{9}$$

consisting of regular points and simple folds only. We also assume that a scalar product of the form

$$\langle(\mathbf{v},\mu),(\mathbf{u},\lambda)\rangle = \omega\mathbf{v}^T\mathbf{u} + \mu\lambda, \quad \omega > 0, \tag{10}$$

is chosen in $\mathbf{R}^M \times \mathbf{R}$. The corresponding norm is denoted by $\| \cdot \|$. (The freedom in the choice of ω in (10) is convenient; e.g., if the system (8) arises by discretization, then ω will change naturally with the grid spacing.) For definiteness, we assume that the branch (9), is parametrized by its arc-length s with respect to the scalar product (10). Then, the tangent vectors

$$(\dot{\mathbf{u}},\dot{\lambda}) = (\frac{d\mathbf{u}}{ds},\frac{d\lambda}{ds}), \quad s_a \leq s \leq s_b,$$

are normalized, i.e.,

$$\| (\dot{\mathbf{u}},\dot{\lambda}) \| = 1.$$

Clearly, for each $s_a \leq s \leq s_b$, the tangent vector satisfies the M linear equations:

$$G_u(\mathbf{u},\lambda)\dot{\mathbf{u}} + G_\lambda(\mathbf{u},\lambda)\dot{\lambda} = 0.$$

4.1 Computation of the Normalized Directed Tangent

Suppose two nearby points on the branch (9) were computed previously:

$$\begin{aligned} (\mathbf{u}_{-1},\lambda_{-1}) &= (\mathbf{u}(s_{-1}),\lambda(s_{-1})), \\ (\mathbf{u}_0,\lambda_0) &= (\mathbf{u}(s_0),\lambda(s_0)), \qquad s_{-1} < s_0, \end{aligned}$$

and we also know the tangent vector $(\dot{\mathbf{u}}_{-1},\dot{\lambda}_{-1})$ at $(\mathbf{u}_{-1},\lambda_{-1})$. Since the two points on the branch are assumed to be close, one can expect that (see Figure 2):

$$\omega\dot{\mathbf{u}}_{-1}^T\dot{\mathbf{u}}_0 + \dot{\lambda}_{-1}\dot{\lambda}_0 > 0.$$

Therefore, the normalized directed tangent $(\dot{\mathbf{u}}_0,\dot{\lambda}_0)$ can be obtained by solving the system of equations:

$$\begin{bmatrix} G_u(\mathbf{u}_0,\lambda_0) & G_\lambda(\mathbf{u}_0,\lambda_0) \\ \omega\dot{\mathbf{u}}_{-1}^T & \dot{\lambda}_{-1} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mu \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{11}$$

and normalizing:

$$\dot{\mathbf{u}}_0 = \frac{\mathbf{v}}{\| (\mathbf{v},\mu) \|}, \quad \dot{\lambda}_0 = \frac{\mu}{\| (\mathbf{v},\mu) \|}. \tag{12}$$

The linear system (11) is solved with the bordering algorithm of Section 5.

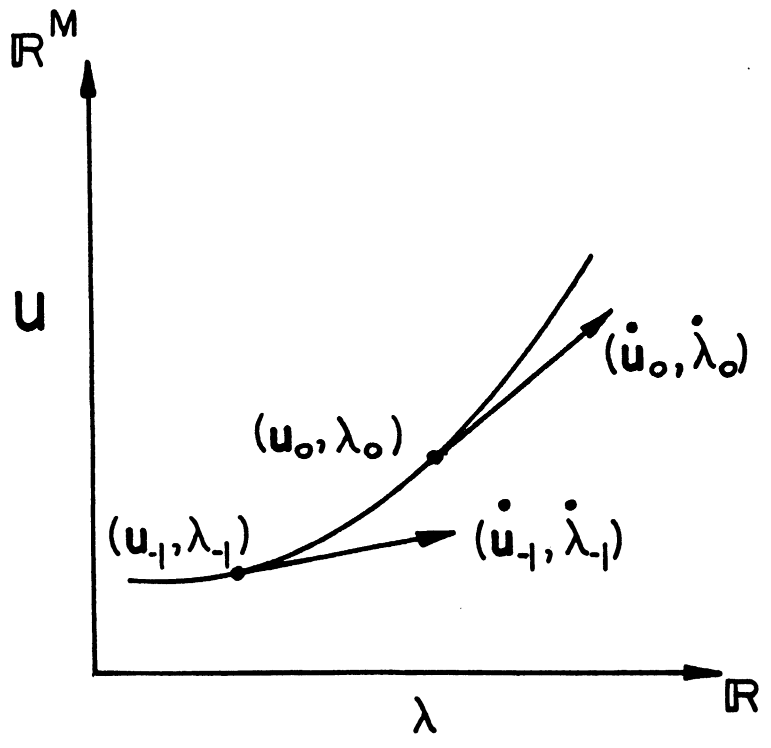


Figure 2: Two nearby solutions and their tangent vectors on a solution branch.

4.2 Newton Iteration in a Hyperplane

After determining the tangent $(\dot{\mathbf{u}}_0, \dot{\lambda}_0)$, a new point on the branch is computed. To this end, assume that an arc-length distance $\Delta s > 0$ is chosen; see Figure 3. The initial approximation

$$(\mathbf{u}^{(0)}, \lambda^{(0)}) = (\mathbf{u}_0, \lambda_0) + \Delta s(\dot{\mathbf{u}}_0, \dot{\lambda}_0)$$

is improved by a Newton iteration in the hyperplane H through $(\mathbf{u}^{(0)}, \lambda^{(0)})$ and perpendicular to $(\dot{\mathbf{u}}_0, \dot{\lambda}_0)$. One obtains the iteration for $k \geq 0$:

$$\begin{aligned} \mathbf{u}^{(k+1)} &= \mathbf{u}^{(k)} + \Delta\mathbf{u}^{(k+1)} \\ \lambda^{(k+1)} &= \lambda^{(k)} + \Delta\lambda^{(k+1)}, \end{aligned}$$

where the correction $(\Delta\mathbf{u}^{(k+1)}, \Delta\lambda^{(k+1)})$ solves the linear system:

$$\begin{bmatrix} G_u(\mathbf{u}^{(k)}, \lambda^{(k)}) & G_\lambda(\mathbf{u}^{(k)}, \lambda^{(k)}) \\ \omega\dot{\mathbf{u}}_0^T & \dot{\lambda}_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u}^{(k+1)} \\ \Delta\lambda^{(k+1)} \end{bmatrix} = \begin{bmatrix} -G(\mathbf{u}^{(k)}, \lambda^{(k)}) \\ 0 \end{bmatrix}. \tag{13}$$

Again, the bordering algorithm of Section 5 is used to solve the above system.

The two basic procedures outlined above have to be supplemented by starting procedures, stopping criteria, and a strategy for changing Δs.

4.3 Stopping Criterion for Newton's Iteration

In the above iteration, we compute the norm of the correction:

$$\epsilon^{(k+1)} = \| (\Delta\mathbf{u}^{(k+1)}, \lambda^{(k+1)}) \|,$$

and stop the iteration when

$$\epsilon^{(k+1)} < \tau = 10^{-5}. \tag{14}$$

The last approximation $(\mathbf{u}^{(k+1)}, \lambda^{(k+1)})$ is accepted as a new point on the branch (9). We stop the iteration as unsuccessful if $\epsilon^{(k+1)} > \epsilon^{(k)}$ or if it takes more than 10 iterations before (14) is satisfied.

4.4 Strategy for Changing Δs

Let Δs_0 denote the arc-length for the previous step, i.e.,

$$(\mathbf{u}_{-1}, \lambda_{-1}) + \Delta s_0(\dot{\mathbf{u}}_{-1}, \dot{\lambda}_{-1})$$

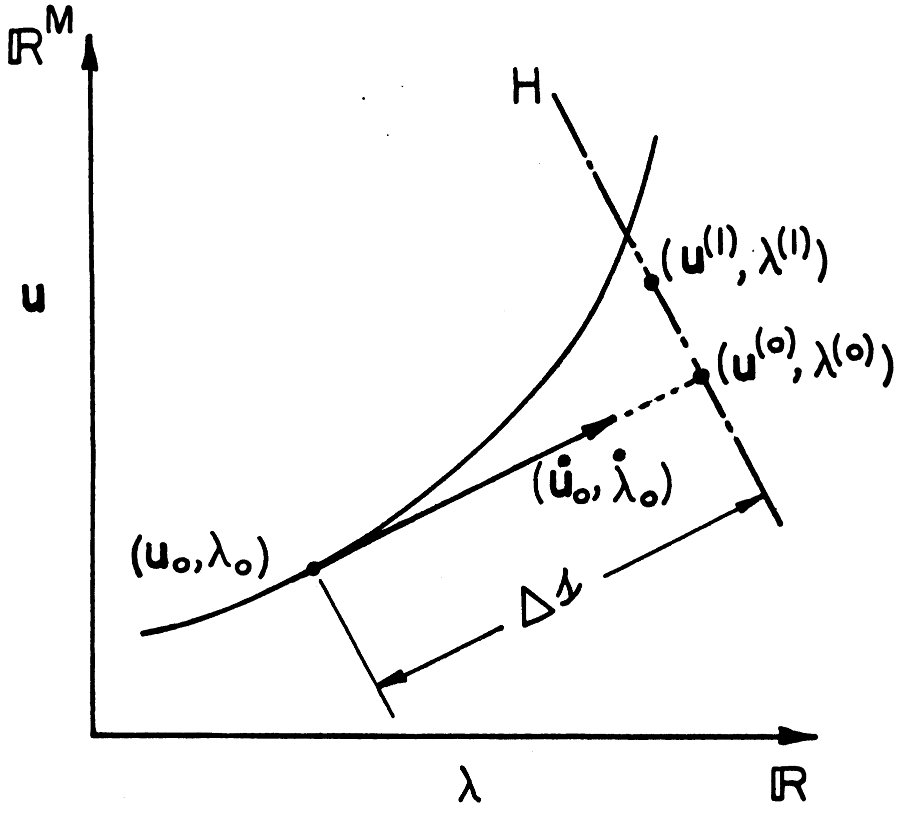

$\mathbb{R}^M$
H
$(u^{(1)}, \lambda^{(1)})$
$(u^{(0)}, \lambda^{(0)})$
$(\dot{u}_0, \dot{\lambda}_0)$
(u_0, λ_0)
Δs
u
λ
$\mathbb{R}$

Figure 3: Initial approximation for new solution point.

was the starting point of the Newton iteration leading to the latest solution $(\mathbf{u}_0, \lambda_0)$. Let N_0 denote the number of Newton iterations that was required to satisfy the stopping criterion (14) for this iteration. The norms of the corrections are denoted by $\epsilon_0^{(n)}$, where $1 \leq n \leq N_0$. (We identify the "solution" $(\mathbf{u}_0, \lambda_0)$ and the latest Newton iterate, though, of course, the latest iterate solves the system (8) only up to $O(\tau^2)$.) We want to determine a new step length Δs_1, which will lead to the starting point

$$(\mathbf{u}_0, \lambda_0) + \Delta s_1(\dot{\mathbf{u}}_0, \dot{\lambda}_0). \tag{15}$$

With $\epsilon_1^{(n)} = \epsilon_1^{(n)}(\Delta s_1)$, we denote the norms of the corresponding corrections if we should start the Newton process at (15).

We assume quadratic convergence; then, to leading order, we have that:

$$\epsilon_i^{(n)} = C(\epsilon_i^{(n-1)})^2 = C^{2^{n-1}-1}(\epsilon_i^{(1)})^{2^{n-1}} \tag{16}$$

for $i = 0$ and $i = 1$. We also make the assumption that for $i = 0$ and $i = 1$:

$$\epsilon_i^{(1)} = \alpha \Delta s_i. \tag{17}$$

(If Δs_i is very small, a theoretical analysis to leading order suggests the assumption:

$$\epsilon_i^{(1)} = \alpha(\Delta s_i)^2. \tag{18}$$

The assumptions (17) and (18) lead to slightly different strategies for changing Δs. It is our experience, however, that the strategy based on (17) is slightly better than the one suggested by (18). It seems that (17) is more realistic than (18) if Δs_i is not extremely small.)

Our aim is to choose Δs_1 such that the stopping condition:

$$\epsilon_1^{(K)} < \tau \tag{19}$$

is satisfied after K steps. Here, the number K is chosen by heuristic arguments; we always perform at least 3 Newton iterations, and our aim is to choose the parameters such that 3 iterations are sufficient to satisfy the stopping criterion. Thus, we assume $N_0 \geq 3$, and take

$$K = \begin{cases} 3 & \text{if } N_0 = 3, \\ N_0 - 1 & \text{if } N_0 > 3. \end{cases}$$

We replace (19) by the requirement $\epsilon_1^{(K)} = \theta\tau$, $\theta = 0.1$, and obtain from (16) and (17) that:

$$\epsilon_1^{(K)} = \epsilon_0^{(K)} \left(\frac{\Delta s_1}{\Delta s_0}\right)^{2^{K-1}} = \theta\tau.$$

Therefore, the suggested value for Δs_1 is:

$$\Delta s_1 = \left(\frac{\theta\tau}{\epsilon_0^{(K)}}\right)^{2^{-K+1}} \Delta s_0.$$

As usual, some precautions must be taken if the suggested value for Δs_1 is too large or too small, but we omit technical details. For alternative approaches to selecting an arc-length Δs, the reader is referred to [3,8], e.g..

4.5 Remarks on the Starting Procedure

The first point on the solution branch, $(\mathbf{u}_0, \lambda_0) = (\mathbf{u}(s_a), \lambda(s_a))$, cannot be computed as outlined above. In general, a special starting procedure is needed. However, the problem at $\lambda = \lambda_0$ is often substantially simpler. The exact solution might be known or easily constructed. If a good estimate for the solution is available, the straight Newton iteration given by:

$$\begin{aligned} G_u(\mathbf{u}^{(k)}, \lambda_0)\mathbf{v}^{(k+1)} &= -G(\mathbf{u}^{(k)}, \lambda_0) \\ \mathbf{u}^{(k+1)} &= \mathbf{u}^{(k)} + \mathbf{v}^{(k+1)}. \end{aligned}$$

can be used. If the initial problem is linear, as is the case for the example discussed in Section 9, one Newton iteration is sufficient to find the solution.

To start the extended Newton iteration, also the tangent to the solution branch $(\dot{\mathbf{u}}_0, \dot{\lambda}_0)$ is necessary. To compute it, use (12) with $\mu = 1$ and $\mathbf{v}$ the solution of the system:

$$G_u(\mathbf{u}_0, \lambda_0)\mathbf{v} = -G_\lambda(\mathbf{u}_0, \lambda_0).$$

This determines the tangent except for an easily resolved ambiguity in sign. To complete the starting procedure, the arc-length step Δs must be initialized. This value is determined heuristically.

5 The Bordering Algorithm

In every step of the continuation algorithm, a system of linear equations is constructed and solved. Most of the computation time is taken up by the solution of these linear systems. For continuation methods, these systems have special structure, and a specialized algorithm to solve them is outlined below. As seen from (11) and (13), the linear systems have the form given by:

$$\begin{bmatrix} A & \mathbf{b} \\ \mathbf{c}^T & \delta \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \eta \end{bmatrix} = \begin{bmatrix} \mathbf{r} \\ \sigma \end{bmatrix}, \tag{20}$$

where A is an M by M matrix, the vectors $\mathbf{b}$, $\mathbf{r}$, and $\mathbf{c}$ are of dimension M, and δ and σ are scalars. The unknowns consist of the vector $\mathbf{x}$ and the scalar η. According to our assumption, the solution branch (9) consists of regular points and simple folds only. Then, the bordered matrix is not singular, i.e., has rank $M+1$. The matrix A has rank M at regular points and rank $M-1$ at simple folds.

In principle, the system (20) could be solved by a straightforward LU-decomposition of the bordered matrix. An accurate solution is found if a numerically stable pivoting strategy is used and the bordered system is sufficiently well conditioned. LU-decomposition of the bordered matrix is not favored because the sign of the determinant of A detects simple folds and potential bifurcation points. By treating the matrix A and the border vectors separately and computing the LU-decomposition of A instead of the bordered matrix, the sign of the determinant of A is easily monitored from the LU-decomposition of A. In applications where the matrix A is sparse there is also another reason why one does not favor factoring the bordered matrix: The vectors $\mathbf{b}$ and $\mathbf{c}$ are, in general, dense. Unless they are excluded from the pivot search until the final phase of the LU-decomposition, a factorization of the bordered matrix can generate an unacceptable amount of fill.

If A is well-conditioned, then (20) can be solved as follows:

1. Solve: $A\mathbf{w} = \mathbf{b}$, $A\mathbf{z} = \mathbf{r}$.

2. From the first M equations of (20) it follows that:

$$\mathbf{x} = -\eta\mathbf{w} + \mathbf{z}. \tag{21}$$

3. Here, the scalar η is obtained from the last equation of (20), i.e., from:

$$(\delta - \mathbf{c}^T\mathbf{w})\eta = \sigma - \mathbf{c}^T\mathbf{z}.$$

Nonsingularity of the matrix (20) implies that $\delta - \mathbf{c}^T\mathbf{w} \neq 0$. This strategy is proposed by Keller [9]. To solve the systems $A\mathbf{w} = \mathbf{b}$ and $A\mathbf{z} = \mathbf{r}$, one can, of course, use the LU-decomposition of A.

For the case that A is singular or almost singular, numerous modifications to the bordering algorithm have been suggested; see [2]. We describe here a version that is easily implemented in a concurrent environment. This version — under certain reasonable assumptions — works well if A is nonsingular, singular, or almost singular. (We always assume that the bordered matrix is well conditioned.)

Let the matrix A be factored with implicit pivoting, i.e., $A = LC^TRU$. Multiplying the first M equations of (20) with R^TCL^{-1}, we obtain:

$$\begin{bmatrix} U & R^TCL^{-1}\mathbf{b} \\ \mathbf{c}^T & \delta \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \eta \end{bmatrix} = \begin{bmatrix} R^TCL^{-1}\mathbf{r} \\ \sigma \end{bmatrix}. \tag{22}$$

In practice, this transformation is achieved by taking the vectors $\mathbf{b}$ and $\mathbf{r}$ along in the LU-decomposition of A. The procedure overwrites the vectors $\mathbf{b}$ and $\mathbf{r}$ with $R^TCL^{-1}\mathbf{b}$ and $R^TCL^{-1}\mathbf{r}$, respectively. For notational convenience, assume this overwriting is done so that (22) becomes:

$$\begin{bmatrix} U & \mathbf{b} \\ \mathbf{c}^T & \delta \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \eta \end{bmatrix} = \begin{bmatrix} \mathbf{r} \\ \sigma \end{bmatrix}. \tag{23}$$

Here, the matrix U is permuted triangular. Let $p_r[k]$ and $p_c[k]$, where $0 \leq k < M$, be the arrays of row and column pivot indices used by the decomposition. Let $\nu = u[p_r[m], p_c[m]]$ be the smallest pivot in absolute value. For notational convenience and clarity of the exposition only, assume that an explicit row and column permutation in (23) brings row $p_r[m]$ to row $M-1$ and column $p_c[m]$ to column $M-1$, i.e., the last row and column of U. We now separate the last row and column in U, and apply a corresponding partitioning to the vectors $\mathbf{c}$, $\mathbf{b}$, $\mathbf{x}$, and $\mathbf{r}$. We obtain:

$$\begin{bmatrix} U_0 & \mathbf{u}_0 & \mathbf{b}_0 \\ \mathbf{v}_0^T & \nu & \beta \\ \mathbf{c}_0^T & \gamma & \delta \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \xi \\ \eta \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \rho \\ \sigma \end{bmatrix}. \tag{24}$$

Here, the matrix U_0 is a nonsingular permuted triangular matrix, and $\nu = 0$ if and only if A is singular. Proceeding analogously to (21), we write:

$$\mathbf{x}_0 = -\xi\mathbf{t}_0 - \eta\mathbf{w}_0 + \mathbf{z}_0, \tag{25}$$

where $U_0\mathbf{t}_0 = \mathbf{u}_0$, $U_0\mathbf{w}_0 = \mathbf{b}_0$, and $U_0\mathbf{z}_0 = \mathbf{r}_0$. (In actual computations, the matrix U_0 is not explicitly formed, and the back-solves are done using the matrix U. The standard back-solve algorithm with U is modified slightly so that it ignores row $p_r[m]$ and column $p_c[m]$.) Substitution of (25) into the last two equations of (24) leads to the following 2×2 system for ξ and η:

$$\begin{bmatrix} \nu - \mathbf{v}_0^T \mathbf{t}_0 & \beta - \mathbf{v}_0^T \mathbf{w}_0 \\ \gamma - \mathbf{c}_0^T \mathbf{t}_0 & \delta - \mathbf{c}_0^T \mathbf{w}_0 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} = \begin{bmatrix} \rho - \mathbf{v}_0^T \mathbf{z}_0 \\ \sigma - \mathbf{c}_0^T \mathbf{z}_0 \end{bmatrix}.$$

This 2×2 system is solved with complete pivoting. The computation of ξ and η is followed by an application of (25) to find the solution of the bordered system. Often, the smallest pivot and the last pivot are the same, i.e., $m = M - 1$. In this case, the above 2×2 system simplifies because the vector $\mathbf{v}_0$ vanishes.

In summary, the underlying assumptions for this algorithm to work reliably are

1. that the system (20) is well-conditioned, and

2. that the factorization $A = LC^T RU$ (the LU-decomposition with pivoting) contains at most one small pivot, i.e., if there is a small pivot, then it is well separated from the next smallest.

6 Computational Results: Pivoting

In this section and in Section 7, we present some computational results for a test problem of the form (8). The test problem itself is explained in detail in Section 9, whereas numerical output is presented in Section 10. Here, it is sufficient to note that the dimension M of the system (8) depends on a step size $h = 2\pi/N$, and that $M = N^2$. The tests described in this section are concerned with

1. pivoting and numerical stability and

2. relations of complete pivots along a branch.

In Section 7 we present timings for matrix LU-decompositions calculated with different data distributions.

Test 1: Preset Pivoting and Numerical Stability

One of the heuristic assumptions underlying this work can loosely be formulated as follows: Suppose a matrix A_0 is factored using complete pivoting.

(Here, we always refer to the decomposition of Section 2.) Suppose further that the matrix A is near A_0 in some sense. Then, if we use the pivot locations of A_0 as preset, the resulting decomposition of A is acceptable from the point of view of numerical stability.

To test validity and usefulness of this "principle," we have computed a solution branch by continuation for the example (33) with $h = 2\pi/25$, i.e., $M = 625$. We always tried to use preset pivoting and calculated the growth factor γ defined in (4) to detect numerical instability. If $\gamma < 100$, the LU-decomposition was considered numerically stable; the next decomposition used the same preset pivots. (The limit of 100 on the growth factor is somewhat arbitrary.) If $\gamma \geq 100$ or if a change in the determinant sign was detected, the next LU-decomposition used complete pivoting. In the third column of Table 2, we present the factors γ. Horizontal lines in the table separate different Newton iterations. Since the start-up is special (linear problem for $\lambda = 0$), the table begins with the first extended Newton iteration. Excluding the start-up calculation, only one more time complete pivoting is necessary in the presented part of the branch; this includes the solution of 34 linear systems. This part of the branch was typical, and throughout the computation preset pivoting was almost always acceptable. In this calculation the determinant sign of matrices 30 and 31 differed, and hence, the LU-decomposition of matrix number 32 was done with complete pivoting.

For comparison, we also have computed all decompositions with complete pivoting, and we list the resulting growth factor in the second column of Table 2. As expected, these growth factors are usually slightly smaller. The resulting gain in numerical stability is negligible, however, and the discretization error is several orders of magnitude larger than the gain in accuracy of the solution vector. (Note that for matrices number 14 through 18, 20, and 21 preset pivoting has a slightly better growth factor than complete pivoting. Although this may seem somewhat surprising, complete pivoting does not guarantee a minimal growth factor.)

Test 2: Relation between Complete Pivots along Branch

The previous test confirms that preset pivoting is most often acceptable from the point of view of numerical stability. Of course, to make a stronger claim, many more examples should be investigated. (Also a better quantitative understanding of the relationships between pivot locations, step size strategy, and numerical stability in a continuation process is needed.) It

LU Number	Complete	Preset	
-1	1.626	1.626	*
0	1.626	1.626	*
1	1.436	1.436	*
2	1.444	1.444	
3	1.444	1.444	
4	1.334	1.386	
5	1.335	1.392	
6	1.335	1.392	
7	1.293	1.357	
8	1.325	1.363	
9	1.326	1.363	
10	1.286	1.357	
11	1.287	1.358	
12	1.287	1.358	
13	1.261	1.753	
14	1.445	1.278	
15	1.414	1.284	
16	1.420	1.286	
17	1.420	1.286	
18	1.447	1.422	
19	1.478	1.480	
20	1.478	1.475	
21	1.478	1.475	
22	1.431	1.579	
23	1.431	1.578	
24	1.431	1.578	
25	1.447	2.388	
26	1.444	2.281	
27	1.443	2.279	
28	1.225	2.489	
29	1.225	2.465	
30	1.225	2.465	
31	1.139	35.550	
32	1.149	1.149	*
33	1.148	1.148	
34	1.148	1.148	

Table 2: Comparison of growth in matrix entries between complete pivoting and a preset pivoting strategy.

is, in principle, possible that our example is such that the pivot locations, if always determined by complete pivoting, remain almost unchanged along the branch. To refute this (remote) possibility, we ran the continuation code for our standard example, i.e., $h = 2\pi/25$ and $M = 625$, but using complete pivoting for every LU-decomposition encountered. Let $A_0, A_1, A_2, \ldots$ be the matrices that are factored successively in the continuation process. Let $r_k^{(i)}, c_k^{(i)}$ be the pivot indices of A_i. We counted the numbers:

$$R^{(i)} = \#\{k : 0 \leq k < 625 \text{ and } r_k^{(i-1)} = r_k^{(i)}\},$$
$$C^{(i)} = \#\{k : 0 \leq k < 625 \text{ and } c_k^{(i-1)} = c_k^{(i)}\},$$

briefly called the number of row hits and the number of column hits. The results are presented in Table 3 This test confirms that, with complete pivoting, the pivot locations vary considerably from step to step.

7 Computational Results: Data Distributions and Timings

The calculations were performed on a Symult Series 2010 multicomputer with up to 64 nodes. In our first study we investigate the dependence of the execution time on the data distribution for one LU-decomposition. Here, we used 64 nodes and an 8×8 process grid. As expected, the adapted data distribution turned out to be superior. In the second study we consider, for each fixed strategy, the dependence of the execution time on the number of nodes. We used 2, 4, 8, 16, 32, and 64 nodes, and obtained excellent speed-up for each strategy. For absolute performance, we made a comparison of the sequential version of our code with a fully optimized C-version of the LINPACK benchmark. In our last study we timed the redistribution procedures of vector data. As explained in Section 3.5, a data redistribution of vectors is necessary whenever a new matrix distribution is introduced. Our study shows that the redistribution times are negligible as compared with the time for the LU-decomposition.

Study 1: LU-Decomposition Timings

We consider the example (33) with $h = 2\pi/25$, i.e., $M = 625$. In Table 4, we present timings for one (typical) LU-decomposition using complete pivoting and preset pivoting in combination with different data distributions

LU Number	Row Hits	Column Hits
1	1	1
2	184	184
3	622	622
4	10	10
5	143	143
6	623	623
7	6	6
8	84	84
9	612	612
10	58	58
11	332	333
12	625	625
13	3	2
14	2	2
15	15	16
16	463	463
17	625	625
18	7	6
19	39	39
20	583	583
21	625	625
22	68	69
23	593	593
24	625	625
25	17	13
26	161	159
27	425	424
28	126	123
29	593	591
30	625	625
31	18	17
32	82	80
33	468	467
34	625	625

Table 3: Number of times k-th pivot index of current LU-decomposition is equal to k-th pivot index of the previous. The basis problem is a discretization on a 25×25 grid and has 625 unknowns.

	Pivoting	Distribution	Time (s)	MegaFLOPS
1	Complete	8 x 8 Linear-Linear	75.308	1.081
2	Complete	8 x 8 Random-Random	63.669	1.278
3	Complete	8 x 8 Scatter-Scatter	62.760	1.297
4	Complete	8 x 8 Adapted-Adapted	51.277	1.587
5	Preset	8 x 8 Linear-Linear	48.921	1.664
6	Preset	8 x 8 Scatter-Scatter	40.335	2.018
7	Preset	8 x 8 Adapted-Adapted	33.808	2.407
8	Fast Preset	8 x 8 Adapted-Adapted	29.741	2.736

Table 4: LU-Decomposition times for a 25 x 25 grid problem on 64 node Symult Series 2010. Number of megaFLOPS is based on $M^3/3$ floating point operations, where $M = 25^2$ is the number of unknowns.

for the factored matrix. The linear and scatter distributions were defined in (5) and (6), respectively. The adapted distribution uses the pivot locations of the previous LU-decomposition to distribute the current matrix with the distribution functions (7). In the version "Fast Preset" of preset pivoting, certain administrative overhead is eliminated; see Section 3.3. All calculations were done on a 64 node machine using 64 processes, one process running on each node. The process grid was partioned into $P = 8$ process rows and $Q = 8$ process columns.

Assuming — for each calculation — $M^3/3$ floating point operations, the execution times are also translated into megaFLOPS.

Study 2: LU-Decomposition Speed-Ups

To test the concurrent performance of our code, we determine the execution time as a function of the number of nodes. The same example as in Study 1 is computed successively using 2, 4, 8, 16, 32, and 64 nodes, and always choosing the number of processes equal to the number of nodes, one process running on each node. The numbers P and Q of process rows and columns were chosen equal within divisibility constraints; see Section 3.5. When the logarithm of the execution time is plotted as a function of the logarithm of the number of processes, ideal speed-up is characterized by a straight line with slope -1 if appropriate scales are used. Figure 4 shows that, for each strategy, the execution-time plot is almost parallel to the line characterizing ideal speed-up. Table 4 can be used to identify the individual timing plots.

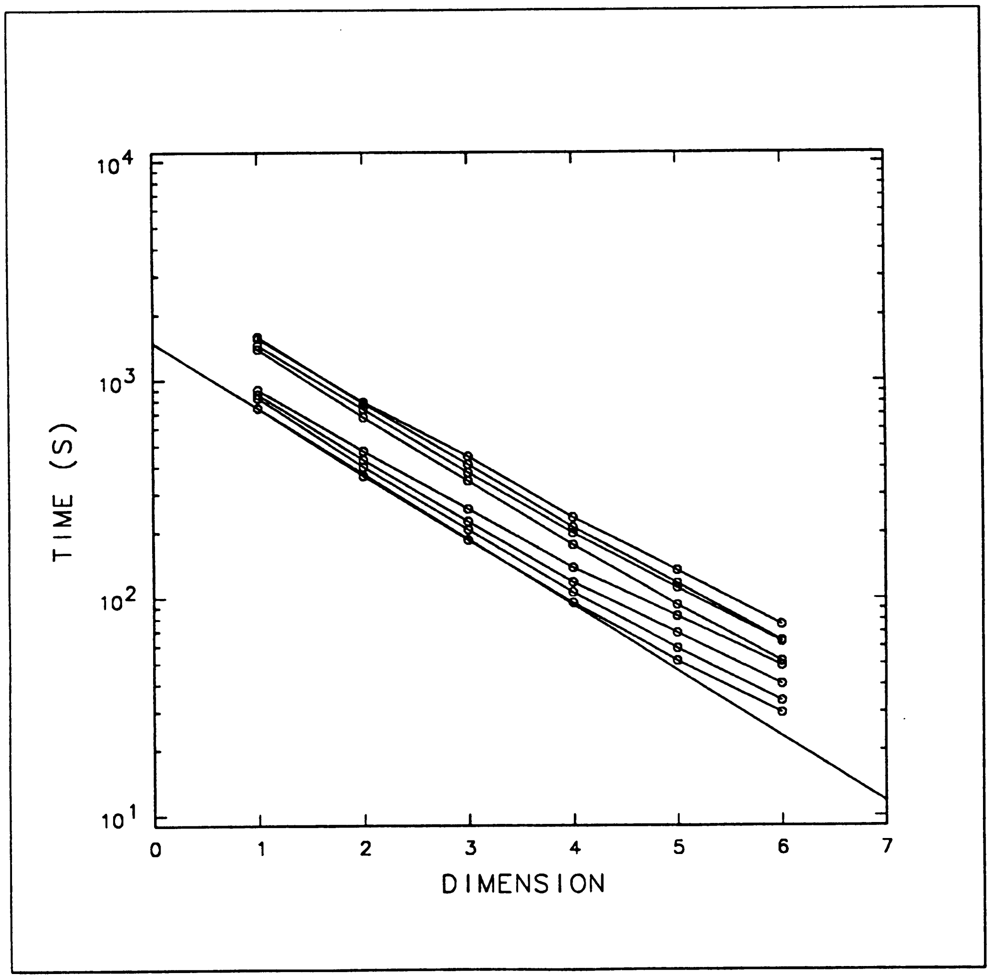


Figure 4: LU-Decomposition times for a 25 × 25 grid problem as a function of number of nodes on a Symult Series 2010.

The problem was too big to run on a one-node machine. Precise speed-ups could thus not be calculated. In Table 5, we give speed-ups and efficiencies with respect to two-node timings, i.e., the real speed-up is estimated by:

$$S_{PQ} \approx 2 * T_2/T_{PQ},$$

and the real efficiency is estimated by:

$$\epsilon_{PQ} \approx 2 * T_2/(PQT_{PQ}).$$

Here, T_2 is the two-node timing and T_{PQ} is the timing with $P \times Q$ nodes. Speed-up and efficiency are good measures for the overhead due to communication and load imbalance.

When varying the data distribution and keeping the pivoting strategy fixed, it is clear that the adapted data distribution is the most efficient. This is easily explained by the increased load balance of the adapted data distribution. This observation holds for both complete pivoting and preset pivoting.

When comparing efficiencies for the same distribution but for different pivoting strategies (i.e., in Table 5 compare lines 1 and 5, 3 and 6, 4 and 7), it is seen that complete pivoting is more efficient. This is because the pivot-search cost leads to a higher ratio of computation to communication time for complete pivoting than for preset pivoting.

Another interesting observation, which follows from Tables 4 and 5, is that complete pivoting with the random distribution (line 2) is more efficient than complete pivoting with the scatter distribution (line 3). The execution time, however, is lower for the scatter distribution. The random distribution is better than the scatter distribution for load balancing, and hence, has higher efficiency. The random distribution leads to very irregular memory access patterns, however, and that causes the absolute execution time to be larger.

The fast-preset algorithm is both faster and more efficient than the preset algorithm. The absolute performance gain is due to overall reduction in administrative overhead. The communication calls were simplified to a larger degree than the computational parts of the code; hence, the efficiency gain.

For absolute performance considerations, we compared a sequential version of our fast-preset code with a LINPACK benchmark program [12]. Due to memory restrictions, this comparison was done with a random 300×300 matrix. A sequential version of the fast preset pivoting algorithm ran about

	Pivoting	Distribution	Speed-Up	Efficiency (%)
1	Complete	8×8 Linear-Linear	41.4	64.7
2	Complete	8×8 Random-Random	49.9	78.0
3	Complete	8×8 Scatter-Scatter	46.2	72.2
4	Complete	8×8 Adapted-Adapted	54.2	84.7
5	Preset	8×8 Linear-Linear	36.9	57.7
6	Preset	8×8 Scatter-Scatter	42.6	66.6
7	Preset	8×8 Adapted-Adapted	48.9	76.4
8	Fast Preset	8×8 Adapted-Adapted	50.0	78.2

Table 5: LU-Decomposition speed-up and efficiency estimates for a 25×25 grid problem on a 64 node Symult Series 2010.

5% slower than LINPACK. (These 5% result from the fact that we have not implemented a number of low level optimizations used by LINPACK.)

Study 3: Redistribution Timings

In the infrequent case that preset pivoting is not successful in the continuation process, a data redistribution of vectors (not of matrices) is necessary. See Section 3.5 for details. Table 6 gives the time in milliseconds needed for the data redistribution. These times are three orders of magnitude smaller than those of the LU-decomposition, and hence, they are negligible. It follows from Table 6 that, for a 64 node computation, the process grid with smallest redistribution time is given by $P = Q = 8$. This is, of course, not important in view of the small absolute execution times for redistribution.

8 Invariant Manifolds

To explain our example in Section 9, we introduce here — in a very specialized form — the concept of an invariant manifold of a dynamical system. We consider autonomous systems in the partitioned form:

$$\begin{bmatrix} \dot{\boldsymbol{\theta}} \\ \dot{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\boldsymbol{\theta}, \mathbf{r}) \\ \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}) \end{bmatrix}, \qquad (26)$$

where $\boldsymbol{\theta} \in T^K$ and $\mathbf{r} \in \mathbf{R}^L$. (Here $T^K = (\mathbf{R} \bmod 2\pi)^K$ denotes the standard torus of dimension K.) The functions:

$$\mathbf{f} \ : \ T^K \times \mathbf{R}^L \longrightarrow \mathbf{R}^K$$

Redistribution Times	
Process Grid	Time (ms)
1×1	183
2×1	93
2×2	80
4×2	77
4×4	67
8×4	65
64×1	158
32×2	93
16×4	65
8×8	58
4×16	66
2×32	92
1×64	158

Table 6: Overhead time due to data redistribution, 25×25 grid.

$$\mathbf{g} \; : \; T^K \times \mathbf{R}^L \longrightarrow \mathbf{R}^L$$

are assumed to be smooth. The solution of (26) to initial data $\boldsymbol{\theta}(0) = \boldsymbol{\theta}^0, \mathbf{r}(0) = \mathbf{r}^0$ is denoted by $S^t(\boldsymbol{\theta}^0, \mathbf{r}^0)$.

We consider manifolds of the form:

$$\mathcal{M} = \{(\boldsymbol{\theta}, \mathbf{R}(\boldsymbol{\theta})) : \boldsymbol{\theta} \in T^K\} \subset T^K \times \mathbf{R}^L, \tag{27}$$

where $\mathbf{R} : T^K \longrightarrow \mathbf{R}^L$ is a C^1-function. (These manifolds are all diffeomorphic to T^K.) $\mathcal{M}$ is called an invariant manifold for (26) if, given any initial data $(\boldsymbol{\theta}^0, \mathbf{r}^0)$ in $\mathcal{M}$, the solution $S^t(\boldsymbol{\theta}^0, \mathbf{r}^0)$ stays in $\mathcal{M}$ for all $t \in \mathbf{R}$. (Since $\mathcal{M}$ is compact, the solution $S^t(\boldsymbol{\theta}^0, \mathbf{r}^0)$ exists for all $t \in \mathbf{R}$ if $\mathcal{M}$ is invariant.)

What are the conditions on $\mathbf{R}$ so that $\mathcal{M}$ is invariant? If $\mathcal{M}$ is invariant, then the solutions in $\mathcal{M}$ have the form $(\boldsymbol{\theta}(t), \mathbf{r}(t)) = (\boldsymbol{\theta}(t), \mathbf{R}(\boldsymbol{\theta}(t)))$. Thus (26) implies:

$$\begin{aligned} \dot{\boldsymbol{\theta}} &= \mathbf{f}(\boldsymbol{\theta}, \mathbf{R}(\boldsymbol{\theta})), \\ \dot{\mathbf{r}} &= \sum_{k=0}^{K-1} \frac{\partial \mathbf{R}}{\partial \theta_k} \dot{\theta}_k = \mathbf{g}(\boldsymbol{\theta}, \mathbf{R}(\boldsymbol{\theta})). \end{aligned}$$

Therefore, $\mathbf{R} : T^K \longrightarrow \mathbf{R}^L$ satisfies the first order system:

$$\sum_{k=0}^{K-1} f_k(\boldsymbol{\theta},\mathbf{R})\frac{\partial \mathbf{R}}{\partial \theta_k} = \mathbf{g}(\boldsymbol{\theta},\mathbf{R}), \quad \boldsymbol{\theta} \in T^K. \tag{28}$$

Conversely, it is not difficult to show that any C^1-solution of (28) defines a manifold $\mathcal{M}$ which is invariant for (26).

Clearly, the dynamics on an invariant manifold (27) is governed by the equation:

$$\dot{\boldsymbol{\theta}} = \mathbf{f}(\boldsymbol{\theta},\mathbf{R}(\boldsymbol{\theta})), \tag{29}$$

which has only K variables. In general, knowledge of an invariant manifold often reduces the dimensionality of a problem; here it is assumed, of course, that the phenomenon of interest is captured in $\mathcal{M}$.

9 Example: A System of Two Coupled Oscillators

A simple example of a single oscillator is described by the two scalar equations

$$\left\{\begin{array}{rcl} \dot{\theta} & = & \omega \\ \dot{r} & = & r(1-r^2), \end{array}\right.$$

where ω is a fixed constant (for computations presented below $\omega = -0.55$). From the sign of $\dot{r}$, it follows that

$$\lim_{t\to\infty} r(t) = 1$$

if $r(0) > 0$. In this case, the one-dimensional manifold given by:

$$\left\{(\theta,1) : \theta \in T^1\right\}$$

is invariant and it describes an attracting limit cycle.

As in [1], a system of two coupled oscillators of the above form is studied. Let the subscript j be used to identify the oscillator, i.e., j is either 0 or 1. Let (θ_j, r_j) be polar coordinates for each oscillator; then the equations considered read:

$$\begin{array}{rcl} \dot{\theta}_j & = & \omega + \lambda C_j \\ \dot{r}_j & = & r_j(1-r_j^2) + \lambda C_j', \end{array} \tag{30}$$

where

$$\begin{aligned} C_0 &= -\cos 2\theta_0 + \frac{r_1}{r_0}(\cos(\theta_0+\theta_1) - \sin(\theta_0-\theta_1)) \\ C_0' &= r_1(\sin(\theta_0+\theta_1) + \cos(\theta_0-\theta_1)) - r_0(1+\sin 2\theta_0) \end{aligned}$$

and λ is the coupling parameter. The coupling terms C_1 and C_1' are obtained by interchanging the indices 0 and 1 in the right hand sides of the above expressions for C_0, C_0' . A motivation for this specific form of coupling is found in [1]. Some isolated invariant manifolds of this system were computed in [4].

For the coupling parameter $\lambda = 0$, the system has the attracting invariant 2-torus

$$\mathcal{M}(\lambda = 0) = \left\{(\boldsymbol{\theta}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}) : \boldsymbol{\theta} \in T^2\right\}.$$

One can show that the torus persists for small coupling.. More precisely, general theory, given in [5] and [11], yields the following result:

Theorem 2 *For any $k \geq 1$ there exists a $\lambda_k > 0$ and a C^k function*

$$\mathbf{R} : T^2 \times (-\lambda_k, \lambda_k) \longrightarrow \mathbf{R}^2$$

such that

$$\mathcal{M}(\lambda) = \left\{(\boldsymbol{\theta}, \mathbf{R}(\boldsymbol{\theta}, \lambda)) : \boldsymbol{\theta} \in T^2\right\},$$

with $-\lambda_k < \lambda < \lambda_k$, is invariant for (30).

Here, $\mathbf{R}(\boldsymbol{\theta}, \lambda = 0) \equiv \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and the invariant manifold $\mathcal{M}(\lambda)$ is locally unique. As the absolute value of λ is increased, the torus looses more and more derivatives, and — according to the calculations of [4] — the torus disappears at about $\lambda = 0.2527$. (It is not clear if one can attach an exact λ-value to the "disappearance"; the bifurcation of tori is not governed by Fredholm theory.)

For the example (30) the partial differential equation (28) with $K = L = 2$ depends on λ. It is convenient to write:

$$\mathbf{R}(\boldsymbol{\theta}, \lambda) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \lambda \mathbf{S}(\boldsymbol{\theta}, \lambda).$$

Then (28) becomes in our example:

$$(\omega + \lambda C_0)\frac{\partial \mathbf{S}}{\partial \theta_0} + (\omega + \lambda C_1)\frac{\partial \mathbf{S}}{\partial \theta_1} = \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix}, \qquad (31)$$

with

$$\psi_j = -(1 + \lambda S_j)(2S_j + \lambda S_j^2) + C_j', \quad j = 0, 1.$$

Using the symmetries:

$$\begin{aligned} C_1(\theta_0, \theta_1, r_0, r_1) &= C_0(\theta_1, \theta_0, r_1, r_0), \\ C_1'(\theta_0, \theta_1, r_0, r_1) &= C_0'(\theta_1, \theta_0, r_1, r_0) \end{aligned}$$

it is not difficult to prove: if S is a solution, then

$$\tilde{\mathbf{S}}(\boldsymbol{\theta}, \lambda) = \begin{bmatrix} S_1(\theta_1, \theta_0, \lambda) \\ S_0(\theta_1, \theta_0, \lambda) \end{bmatrix}$$

is also a solution. Therefore, as long as the solution is unique, it obeys the symmetry:

$$S_0(\theta_0, \theta_1, \lambda) = S_1(\theta_1, \theta_0, \lambda). \qquad (32)$$

To arrive at a discrete system, we choose a grid spacing $h = 2\pi/N$ and replace the torus T^2 by a grid:

$$T_h^2 = \{(hk_0, hk_1) : 0 \le k_j < N\}.$$

As usual, for notational convenience, grid functions on T_h^2 are identified with grid functions on:

$$\{(hk_0, hk_1) : k_j \in \mathbf{Z}\}$$

which are 2π-periodic in θ_0 and θ_1. If $\mathbf{v} : T_h^2 \longrightarrow \mathbf{R}^2$ is a grid function, one defines the centered divided differences:

$$\begin{aligned} D_0\mathbf{v}(\theta_0, \theta_1) &= \frac{1}{2h}(\mathbf{v}(\theta_0 + h, \theta_1) - \mathbf{v}(\theta_0 - h, \theta_1)) \\ D_1\mathbf{v}(\theta_0, \theta_1) &= \frac{1}{2h}(\mathbf{v}(\theta_0, \theta_1 + h) - \mathbf{v}(\theta_0, \theta_1 - h)). \end{aligned}$$

The leap-frog discretization for (31) is obtained by replacing $\frac{\partial \mathbf{S}}{\partial \theta_j}$ with $D_j\mathbf{S}^h$. This system has $2N^2$ scalar unknowns. Finally, we impose the symmetry (32) on the discrete system to obtain an equation:

$$G_h(\mathbf{u}, \lambda) = 0. \qquad (33)$$

Here, $\mathbf{u}$ is a vector of $M = N^2$ scalar unknowns, identified with the grid function S_0^h. The continuation process is applied to the system (33): thus symmetry is enforced. Of course, by enforcing the symmetry on the solution, we exclude a study of bifurcations breaking the symmetry. Initial studies indicated that higher-order singularities (different from simple folds) do occur in the discrete system if one does not enforce the symmetry.

In our computations, the grid function S_0^h and the vector $\mathbf{u}$ are identified according to a lexicographic ordering of the grid. Then, linearization of (33) leads to a coefficient matrix which has the characteristic sparsity structure displayed in Figure 5. The figure shows the matrix structure for an 8×8 grid. The dense lines parallel to the main diagonal are due to the periodic leap-frog discretization. The remaining anomalous sparse fill is caused by the enforcement of the symmetry. The anomalous fill makes it difficult for the LU-decomposition algorithm to exploit the sparsity of the matrix.

10 Numerical Results

We have applied our continuation algorithm to the system (33) with $h = 2\pi/25$ and $h = 2\pi/50$. In Figures 6 and 7, we present the solution paths by plotting

$$\| S_0^h(\cdot,\lambda) \|_{T_h^2} = h \| \mathbf{u}(\lambda) \|_2$$

as a function of the coupling parameter λ. Though the discretization theory for $h \to 0$ is not complete (see [4] for some discussions), we believe that the discrete branches are fairly good approximations to the continuous branch of invariant tori, but only away from the fold points. In our calculation, the first fold in the discrete branch occurs at λ-values approximately given by 0.243 for $h = 2\pi/25$ and 0.249 for $h = 2\pi/50$. This is in fairly good agreement with the value $\lambda = 0.2527$ reported in [4] as the last value where a torus could be obtained. (The calculations in [4] reach higher resolution, but do not give a solution branch.) The continuation of the 25×25-grid problem was terminated when the arc-length steps became too small. From comparison of the 25×25-grid and the 50×50-grid calculations, it is reasonable to expect that the discrete solutions approximate continuous solutions up to the first fold.

In Figure 8, we present the functions $R_0^h(\boldsymbol{\theta},\lambda)$ for $h = 2\pi/50$ and two different λ-values. Beyond the fold, the surfaces show crinkling. We want to emphasize that the discrete solutions beyond the first fold are not necessarily related to any invariant manifold of the dynamical system.

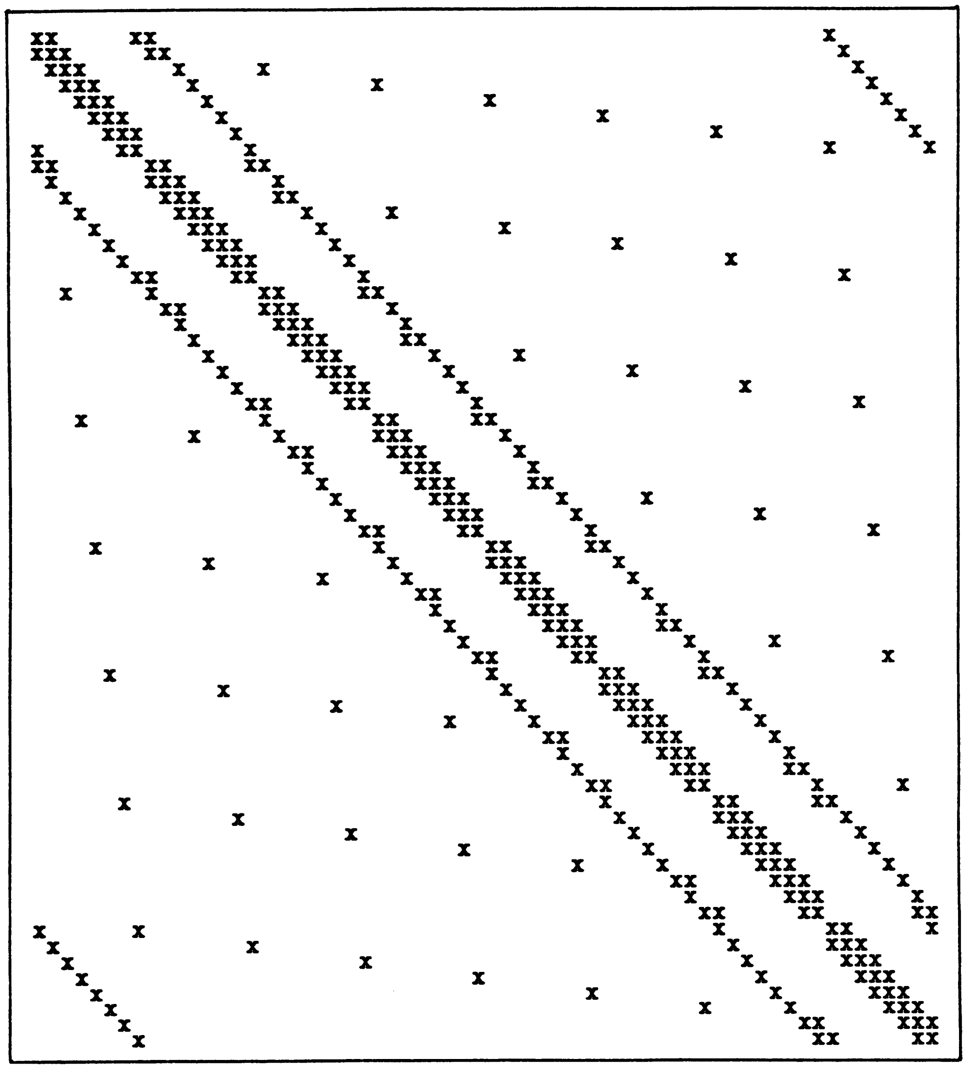


Figure 5: Fill structure of coefficient matrix for 8 × 8 grid.

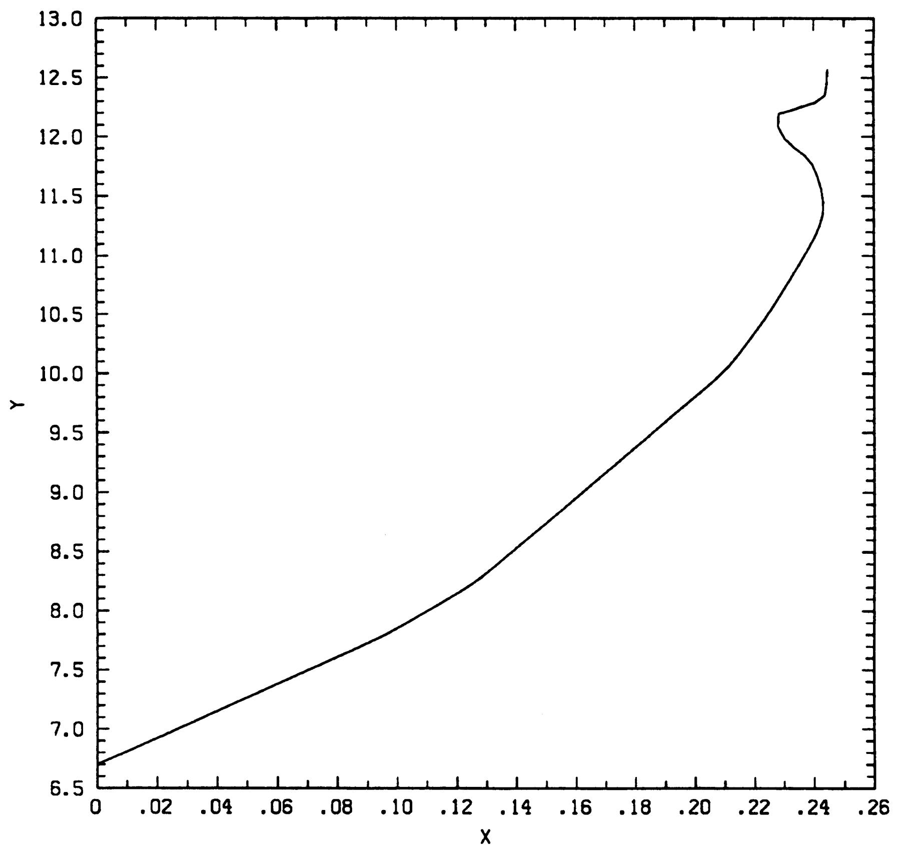


Figure 6: Solution norm as a function of the coupling parameter; $h = 2\pi/25$.

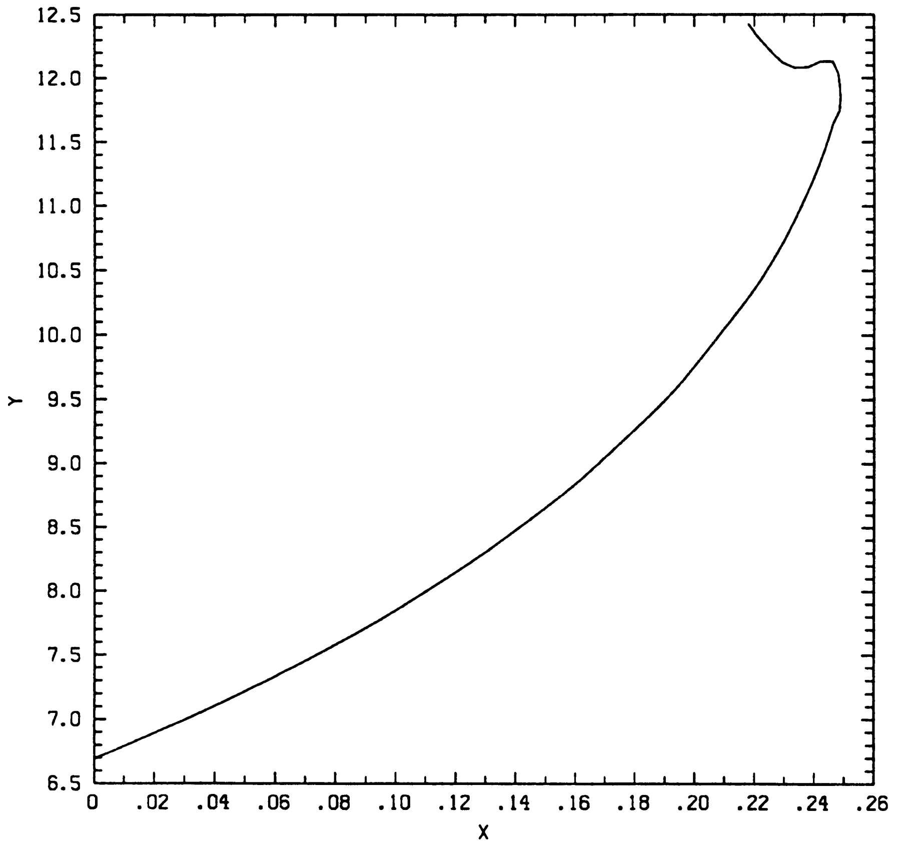
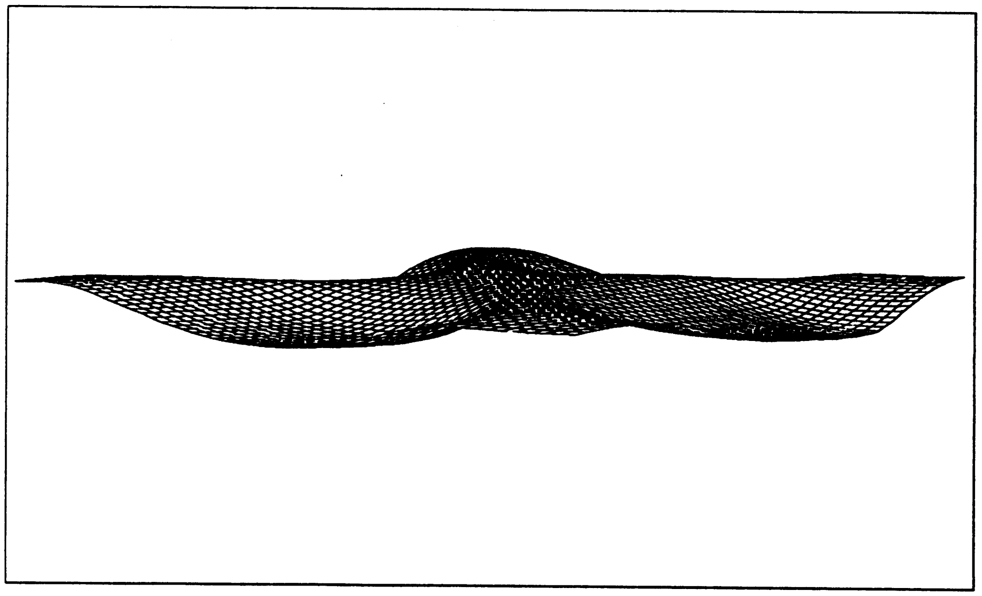


Figure 7: Solution norm as a function of the coupling parameter; $h = 2\pi/50$.



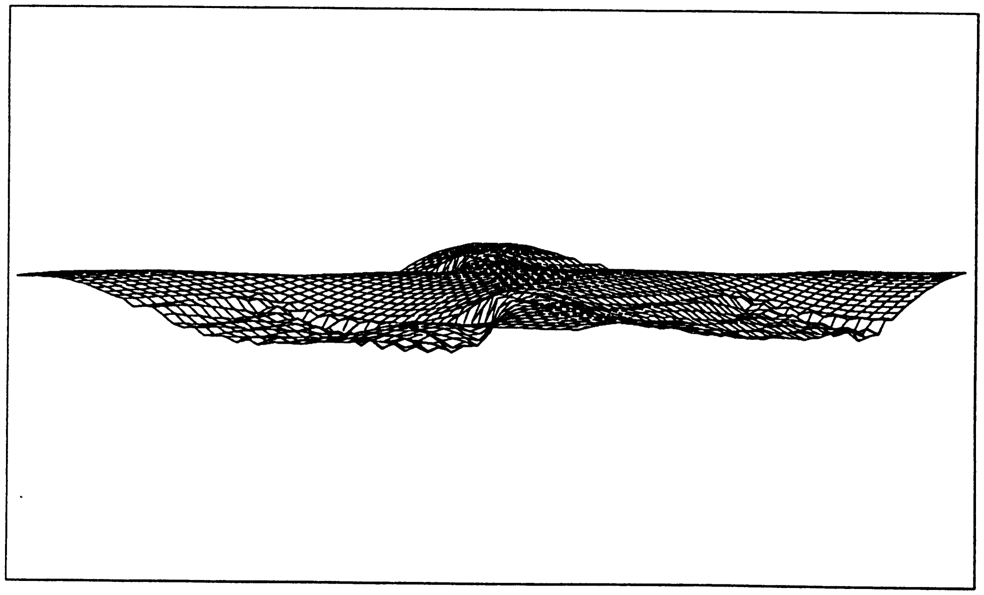


Figure 8: Function $r_0(\theta_0, \theta_1)$ defining the invariant manifold for $\lambda = 0.245$ (before the fold) and $\lambda = 0.247$ (beyond the fold).

We also present the streamlines of the system (29), which governs the dynamics on the computed invariant torus; see Figure 9. Clearly visible are the "in-phase" attracting periodic orbit and the instability of the out-of-phase periodic orbit; see [1].

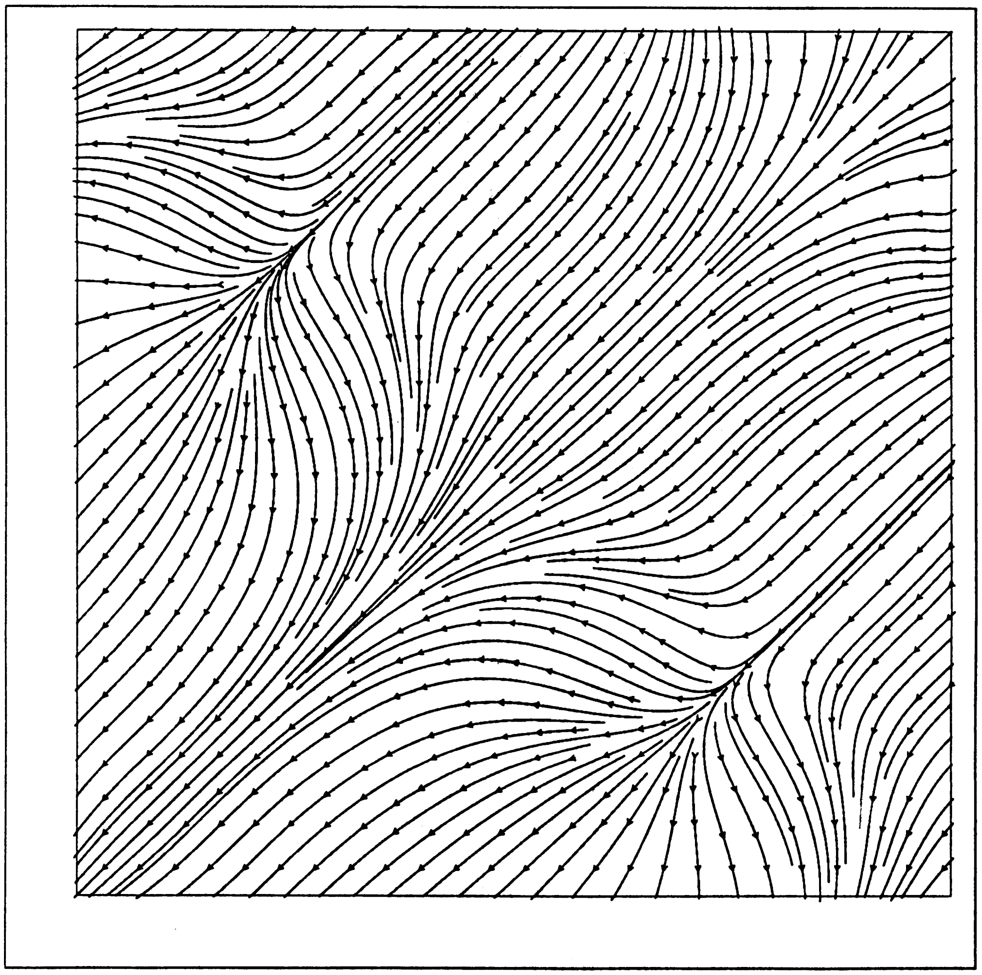


Figure 9: Flow on the invariant torus corresponding to $\lambda = 0.245$.

References

[1] D. G. Aronson, E. J. Doedel, and H. G. Othmer. An analytical and numerical study of the bifurcations in a system of linearly-coupled oscillators. *Physica*, 25D:20–104, 1987.

[2] T. F. Chan. Deflation techniques and block-elimination algorithms for solving bordered singular systems. *SIAM Journal on Scientific and Statistical Computing*, 5(1):121–134, March 1984.

[3] B. Deuflhard, B. Fiedler, and P. Kunkel. Efficient numerical pathfollowing beyond critical points. *SIAM Journal on Numerical Analysis*, 24(4):912–927, August 1987.

[4] L. Dieci, J. Lorenz, and R. D. Russel. Numerical calculation of invariant tori. 1989. Submitted.

[5] N. Fenichel. Persistence and smoothness of invariant manifolds for flows. *Indiana University Mathematics Journal*, 21:193–226, 1971.

[6] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors*. Prentice Hall, 1988.

[7] N.J. Higham and D.J. Higham. Large growth factors in Gaussian elimination with pivoting. *SIAM Journal on Matrix Analysis*, 10(2):155–164, 1989.

[8] H.B. Keller. *Numerical Methods in Bifurcation Problems*. Tata Institute of Fundamental Research, Bombay, 1987.

[9] H.B. Keller. Practical procedures in path following near limit points. In R. Glowinski and J.L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, North-Holland, 1982.

[10] W.C. Rheinboldt. *Numerical Analysis of Parametrized Nonlinear Equations*. Wiley, New York, NY, 1986.

[11] R. Sacker. A perturbation theorem for invariant manifolds and Hölder continuity. *Journal Mathematical Mechanics*, 18:705–762, 1969.

[12] B. Toy. Private Communication.

[13] E. F. Van de Velde. *Experiments with Multicomputer LU-Decomposition.* report CRPC-89-1, Center for Research in Parallel Computing, 1989. To appear in Concurrency: Practice and Experience.

[14] J.H. Wilkinson. Error analysis of direct methods of matrix inversion. *Journal of the Association for Computing Machinery*, 8:281–330, 1961.