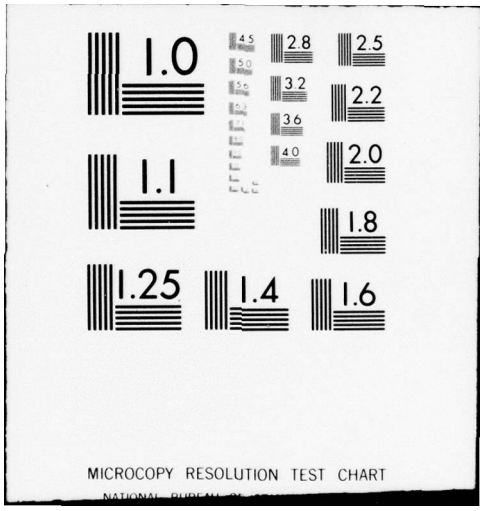
AD-A061 561 ARMY RESEARCH OFFICE RESEARCH TRIANGLE PARK N C F/G 12/1
PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS C--ETC(U)
OCT 78

UNCLASSIFIED ARO-78-3 NL

1 OF 4
AD
A061 561




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART

ADA061561

ARO Report 78-3

PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS CONFERENCE

LEVEL II



DDC
RECEIVED
NOV 27 1978

DDC FILE COPY

Approved for public release; distribution unlimited.
The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

SPONSORED BY
THE ARMY MATHEMATICS STEERING COMMITTEE ON BEHALF OF

THE OFFICE OF
THE CHIEF OF RESEARCH, DEVELOPMENT AND
ACQUISITION

78 11 24 080

(9) Interim technical rept.

(12)

U. S. ARMY RESEARCH OFFICE

Report No. 78-3

(11) October 1978 (12) 329p.

PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS CONFERENCE

Sponsored by the Army Mathematics Steering Committee

HOST (14) ARO-78-3

U. S. ARMY MISSILE RESEARCH AND DEVELOPMENT COMMAND

Redstone Arsenal, Alabama

1-2 March 1978

(6) Proceedings of the 1978 Army Numerical Analysis and Computers Conference held at U. S. Army Missile Research and Development Command, Redstone Arsenal, AL. 1-2 March 1978.

Approved for public release; distribution unlimited.
The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, North Carolina

040 900 78 11 24 000

FOREWORD

Are four different colors all that are needed in making a map? This is known as the four color problem, and it has challenged mathematicians for many years. Recently, with the aid of a computer, this problem has been solved in the affirmative. This certainly points out the fact that computers are powerful tools, and one might be led to the wrong conclusion about their ability to solve all the problems faced by scientists. David Hilbert, a famous German mathematician, suggested that the consistency of any set of axioms could be proven by step-by-step analysis. In 1931, the Austrian mathematician, Kurt Gödel, proved his celebrated imcompleteness theorem, which showed that not all problems can be handled in this manner. While this fact may be a little discouraging, still it hasn't slowed individuals in the process of finding applications of numerical analysis and the use of computers. One has only to scan the program listed on the next pages to realize that step-by-step analysis has become an indispensable tool for treating scientific data.

The 1978 Army Numerical Analysis and Computers Conference had as its host the U. S. Army Missile Research and Development Command at Redstone Arsenal, Alabama. The Army Mathematics Steering Committee (AMSC), which sponsors these conferences, was pleased to have one of its members, Dr. Siegfried H. Lehnigk serve as chairman on local arrangements. He has served in a similar capacity for two other Army-wide conferences sponsored by the AMSC. In 1969, Dr. Lehnigk took on the responsibility for the 15th Conference on the Design of Experiments in Army Research, Development and Testing; and in 1971, he guided the local arrangements for the 17th Conference of Army Mathematicians. Those in attendance at each of these meetings were certainly fortunate to have such an able gentlemen looking after their needs while away from their home offices.

This conference continues a series of meetings held annually since 1959 for the purposes of providing a forum for the exchange of information among the scientific and technical staffs of Army computation centers, and to provide the AMSC with information on the current and future needs of

the Army on numerical analysis and related fields of mathematics and computing.

The responsibility for organizing these conferences rest in the hands of the AMSC Subcommittee on Numerical Analysis and Computers. Members of this committee selected "Software Reliability" as the theme for the meeting at Redstone Arsenal. Many of the contributed papers as well as three of the four invited speakers listed below stressed this area of scientific endeavor.

Speaker and Institution	Title of Address
Stephen S. Yau Northwestern University	On Error Resistant Software Design
Carl de Boor Mathematics Research Center	Best Nodes for Polynomial Interpolation
Lloyd Fosdick University of Colorado	Some Algorithms for the Analysis of Computer Programs
Sol Greenspan University of Toronto	Software Structuring: Concepts and Methods

Members of the AMSC are taking this opportunity to express their thanks to the U. S. Army Missile Research and Development Command for serving as host of this conference. They realize that all the speakers spent a lot of time and effort in preparing and presenting their papers, and would like to let them know their efforts were appreciated by all those in attendance at this meeting. Many of these papers appear in these Proceedings so that the important results they contain can be studied and be used by members of the scientific community.

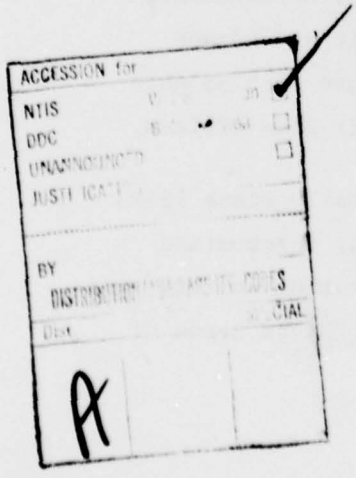


TABLE OF CONTENTS*

Title	Page
Foreward	iii
Table of Contents	v
Agenda	vii
On Error-Resistant Software Design Stephen S. Yau	1
Primary Program Descriptions: Why We Need a New Approach to Correctness Leon S. Levy and Paul Broome	21
The BRL Bessel Function Subroutine Kathleen L. Zimmerman and Alexander S. Elder	37
A Semantic Updating System for Repairing Software Morton A. Hirschberg, Edward F. Miller, Jr., and John S. Praninskas	45
Software Restyling in Graphics and Programming Languages Eric Grosse	79
Best Nodes for Polynomial Interpolation Carl de Boor	109
Comparing Digital Filters Which Produce Derivative Approximations Charles K. Chui, Philip W. Smith, and Joseph D. Ward	111
Computable Error Bounds for the Nystrom Method J. W. Hilgers and L. B. Rall	117
Numerical Solutions to the Lateral Stability of a Missile Julian J. Wu	143
Computation of Transonic Flow Past Slender Bodies at Angle of Attack R. P. Reklis, W. B. Sturek and F. R. Bailey	155

next page

*This Table of Contents lists only the papers that are published in this technical manual. For a list of all of the papers presented at the 1977 Army Numerical Analysis and Computers Conference see the copy of the Agenda.

Numerical Simulation of Electro-Chemical Machining
Gunter H. Meyer 165

The CEMA Ryad Computer Family
S. E. Goodman 171

Some Algorithms for the Analysis of Computer Programs
Lloyd D. Fosdick 181

Solving Differential Equations on a Hand Held Programmable Calculator
J. Barkley Rosser 207

The Mechanical Train Analog: A Proposed Software Evaluation Tool
Peter Beck and Raymond Chuvala 249

A Comparison of Nastran Code and Exact Solution to an Elastic-Plastic Deformation Problem
Peter C. T. Chen 261

On Block Relaxation Techniques
D. Boley, B. L. Buzbee and S. V. Parter 277

Software Structuring: Concepts and Methods.
Clement L. McGowan and Sol J. Greenspan 297

List of Attendees 319

AGENDA
1978
ARMY NUMERICAL ANALYSIS AND COMPUTERS CONFERENCE

US ARMY MISSILE RESEARCH AND DEVELOPMENT COMMAND
REDSTONE ARSENAL, ALABAMA

WEDNESDAY, 1 MARCH

0800 **Registration** *Lobby, Hilton Inn*

0830 **Opening of the Conference**

0845 **Keynote Address**

Chairperson: Ronald Uhlig, Hq, US Army Materiel Development and Readiness Command, Alexandria, Virginia

Speaker: Stephen S. Yau, Northwestern University, Evanston, Illinois

Title: **On Error Resistant Software Design**

0945 **Break**

1000 **TECHNICAL SESSION I**

Chairperson: Stanley Taylor, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland

Why We Need a New Approach to Correctness
Leon S. Levy, University of Delaware – Newark
Paul Broome, Chemical Systems Laboratory, Aberdeen Proving Ground, Maryland

The BRL Bessel Function Subroutine
Kathleen L. Zimmerman and Alexander S. Elder
Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland

A Semantic Updating System for Repairing Software
Morton A. Hirschberg, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland
Edward F. Miller, Jr., Software Research Associates, Waltham, Massachusetts

Implementation of the Generalized Reduced Gradient Method for Nonlinear Programming Problems
Gary A. Gabriele, US Army Computer Systems Command, Fort Belvoir, Virginia

Software Restyling in Graphics and Programming Languages
Eric Grosse, Stanford University, Stanford, California

1140 Lunch

1300 **GENERAL SESSION I**

Chairperson: Paul Boggs, Army Research Office, Research Triangle Park, North Carolina

Speaker: Carl de Boor, Mathematics Research Center, University of Wisconsin – Madison
Title: **Best Nodes for Polynomial Interpolation**

1400 Break

Note: **Technical Sessions II and III will be held concurrently.**

1415 **TECHNICAL SESSION II**

Chairperson: William Shepherd, White Sands Missile Range, New Mexico

Numerical Differentiation by Means of Recursive Digital Filters
C. K. Chui, P. W. Smith, and J. D. Ward, Texas A&M University, College Station, Texas

Sinc-Galerkin Method of Solution of Maxwell's Equations for Geologic Sounding
W. Petrick and F. Stenger, University of Utah, Salt Lake City, Utah

Computable Error Bounds for the Nystrom Method
J. W. Hilgers, Michigan Technological University
L. B. Rall, Mathematics Research Center, University of Wisconsin – Madison

Temporal Fortran: Flags
R. D. Scanlon, Benet Weapons Laboratory, Watervliet Arsenal, Watervliet, New York

1415 TECHNICAL SESSION III

Chairperson: Norman Banks, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland

Numerical Solutions to the Lateral Stability of a Missile
Julian J. Wu, Benet Weapons Laboratory, Watervliet Arsenal, Watervliet, New York

Computation of Transonic Flow Past Slender Bodies at Angle of Attack
R. P. Reklis and W. B. Sturek, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland
F. R. Bailey, NASA Ames Research Center, Moffett Field, California

The Method of Lines for Free Boundary Problems
Gunter H. Meyer, Georgia Institute of Technology, Atlanta, Georgia

The CEMA RYAD Computer Family
S. E. Goodman, The Woodrow Wilson School of Public and International Affairs, Princeton University, Princeton, New Jersey

1535 Break

1550 GENERAL SESSION II

Chairperson: A. H. Curry, Computer Systems Command, AIRMICS, Atlanta, Georgia

Speaker: Lloyd Fosdick, University of Colorado, Boulder, Colorado
Title: **Some Algorithms for the Analysis of Computer Programs**

THURSDAY, 2 MARCH

Note: Technical Sessions IV and V will be held concurrently.

0830 TECHNICAL SESSION IV

Chairperson: Shirley Smith, Hq, US Army Aviation Systems Command, St. Louis, Missouri

Solving Differential Equations on a Hand-Held Programmable Calculator
J. Barkley Rosser, Mathematics Research Center, University of Wisconsin – Madison

Computer-Aided Design of Warheads by the Application of Hydrodynamic Computer Code Calculations
John T. Harrison, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland

The Mechanical Train Analog – A Proposed Software Evaluation Tool
Peter Beck and Raymond Chuvala, US Army Armament Research and Development Command, Dover, New Jersey

0830 TECHNICAL SESSION V

Chairperson: Julian J. Wu, Benet Weapons Laboratory, Watervliet Arsenal, Watervliet, New York

Eigensystem Problems Arising in the Computation of the Response of Large-Scale Dynamical Systems
Ben Noble, Mathematics Research Center, University of Wisconsin – Madison

A Comparison of Nastran Code and Exact Solution to an Elastic-Plastic Deformation Problem
Peter C. T. Chen, Benet Weapons Laboratory, Watervliet Arsenal, Watervliet, New York

On Block Relaxation Techniques
D. Boley, Computer Science Department, Stanford University, Stanford, California
B. L. Buzbee, Los Alamos Scientific Laboratory, Los Alamos, New Mexico
S. V. Parter, Mathematics Research Center, University of Wisconsin – Madison

0930 Break

0945 **GENERAL SESSION III**

Chairperson: Donna Brock, US Army Missile Research and Development Command, Redstone Arsenal, Alabama

Speaker: Sol Greenspan, University of Toronto and SofTech, Inc., Waltham, Massachusetts
Title: **Software Structuring: Concepts and Methods**

1045 **OPEN MEETING**

Subcommittee on Numerical Analysis and Computers, AMSC.

1200 Adjournment

ON ERROR-RESISTANT SOFTWARE DESIGNS*

Stephen S. Yau
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois 60201

ABSTRACT. Error-resistant software is a piece of software which can resist the adverse effects of errors. It can provide high reliability and continuous availability of computing resource, even in the presence of software errors and hardware faults. This feature is very important for many real-time systems, such as missile defense systems, military communication systems, air traffic control systems, and telephone switching systems. In this paper, the concept of error-resistant software designs is introduced, and various design approaches will be presented. Further research and development in this area will also be discussed.

1. INTRODUCTION. Error-resistant software is a piece of software which can resist the adverse effects of errors [1]. In order to accomplish this goal, the program must possess the capabilities to detect errors, to locate and contain the propagation of errors, and to recover from the errors. Many existing software systems contain errors but reliable in terms of producing correct results. High reliability and continuous availability, even in the presence of software errors and hardware faults, are very important for many real-time systems, such as missile defense systems, military communication systems, air-traffic control systems, and telephone switching systems.

The capabilities of error detection, error containment and recovery can be implemented by self-checking software [2]. A piece of self-checking software contains software redundancy in the program to check the dynamic behavior for proper operation during its execution. When an abnormal behavior is detected, it will be interpreted as an error and the error will be isolated to minimize its propagation. A recovery procedure is then attempted to correct the abnormal behavior. It will involve the repair of damage caused by the error and to correct the error if possible. Some transient errors can be corrected by repeating the operation. Other permanent damage can be repaired by reconstructing the value of the mutilated items from redundant information stored in the program or from a safe backup copy stored at an earlier state. Some rollback and retry operations are usually involved. Some errors can be corrected by a duplicated copy of hardware for hardware faults or a different version of algorithm for software errors. Human intervention may be necessary sometimes.

* This work was supported by the U.S. Army Research Office
Grant No. DAAG-29-76-G-0183.

From the reliability point of view, a piece of well-designed error-resistant software should detect an error at the "earliest" time, contain the damaging effects of an error to the "smallest" domain, and perform the most "reliable" recovery. This requires a great deal of software redundancy (code and data). Although a fast recovery is guaranteed, this capability will also cost very large overhead during the normal operation of the program. There is a tradeoff between the overhead of validation checks and the cost of recovery. However, with the rapid improvement of hardware cost and speed, many of the validation checks can be performed concurrently and economically with the normal data processing operations in a multiprocessor system. Error-resistance can then be achieved economically with very little cost in terms of real-time overhead.

In this paper, we will present various approaches to error-resistant software design, and discuss further research and development in this area.

2. TYPES OF ERRORS. Software can be roughly partitioned into two types, the application programs and the operating system programs. The application programs perform the processing of data required by the users. The operating system acts as an interface between the hardware and the application programs. It provides the user with a more usable virtual machine. It is responsible for the allocation of resources, the synchronization of the physical progress of the processes, and the protection of the integrity of the processes as well as data stored in the computer system. All these tasks have to be performed efficiently in terms of time and the amount of resources required.

The behavior of a program may be considered at three levels: the function performed by each module (module level), the flow of control and data between modules (program level), and the interactions that a program has with other software components of the system (system level). If the normal behavior of an application program at each of these three levels can be specified, then its execution can be monitored to check whether it is performing properly. At the module level, the function of a module is to perform a transformation from its input data to its output data [3]. This transformation can be specified by describing the legitimate values of the input data objects, and the corresponding output values that a correctly functioning module would produce. At the program level, the normal flow of control and data between modules in a program can be specified by indicating how control can be transferred between modules, under what circumstances each of the control transfer may be taken, and what data should be passed at each interface. (We assume that global data will be passed as parameters, at least conceptually.) At the system level, the interactions that a program is allowed to have with its external environment can be specified by indicating which objects the program is authorized to interact with, under what conditions those interactions may occur, and what information should be communicated in those interactions. It also includes other aspects of an operating system, such as resource allocation, job management and memory management. Deviations from the specified behavior at one of the levels will be detected by the system as a software error or hardware fault at that level. If possible, recovery procedures will then be initiated to allow continuation

of normal execution. Complete recovery may be impossible, but it may be possible to continue in a degraded mode of operation and still produce useful results.

Since errors are detected as deviations from the normal behavior of the program, the error detection code is designed to detect both software errors and hardware faults. In other words, the error detection routines do not make a sharp distinction among the origins of the error, whether it is caused by a hardware or software failure. There are several reasons for this consideration. As hardware becomes more complicated, it is plagued by design errors as well as component failures. Under these circumstances, hardware and software errors are indistinguishable. We have to try to cope with the error situation without knowing whether the error is caused by a program bug or a hardware fault. Since software errors are very general in nature, many of the precautions taken against software errors will be useful for many types of hardware faults, such as transient faults and multiple faults, for which there exists no effective hardware diagnosis technique at present. It is not always advisable to depend solely on hardware redundancy to achieve hardware fault-tolerance. A piece of error-resistant software should resist the adverse effects of any error, regardless whether it is caused by software or hardware.

We will assume that a separate set of hardware diagnostics are available to verify the integrity of the system hardware after an error is detected and that hardware redundancy is available to enable hardware fault-tolerant operation. In general, we may be concerned with software errors of the following categories in the three levels:

Module Level Errors which cause a module to perform the function specified by the programmer improperly. These errors occur when

- a module does not start execution because the input data to the module do not meet its input specifications,
- a module starts execution, but does not complete execution because of an error condition detected by the hardware, e.g., division by zero, illegal opcode or address out of range,
- a module does complete execution, but the output data of the module do not meet its output specifications.

Program Level Errors which cause the flow of control and data between modules in a program to deviate from the specifications of the programmer. These errors occur when

- there is a wrong transfer of control between modules.
- there is an infinite internal loop (within a module),
- there is an infinite external loop (between modules).

System Level Errors which cause the program not to interact with objects in the system in the manner specified by the programmer. These errors occur when

- the program is not authorized to interact in the manner specified by the programmer,
- the program is authorized to interact in the manner specified, but the interaction failed because
  a. the program itself has an error,
  b. the program it was interacting with has an error,
  c. the data it was interacting with have an error.
- the operating system does not operate properly because
  a. it has resource control errors,
  b. it has memory access right errors,
  c. it has decision errors,
  d. it has time dependent synchronization errors.

In this paper, we will present some of the available methods dealing with these types of errors and also discuss future work in this area.

3. USE OF SELF-CHECKING SOFTWARE. Many self-checking techniques can be incorporated in software design to verify the correct operation of the system during execution. Here, we will discuss some of the self-checking techniques. For more detailed discussion, the reader is referred to [2]. Basically, all the self-checking techniques can be classified according to the three aspects of a process that the technique is checking:

- the function of a process
- the control sequence of a process
- the data of a process.

3.1 Functional Checking. The functional aspects of a process can be checked by verifying the reasonableness of the output for a given set of input. This can be implemented easily if the function of the process is well defined mathematically. For example, if the output is the solution of a set of mathematical equations, its correctness can be verified by substituting the solution to the equations and checking for consistency. In some cases, there may exist a simple relationship among the output variables, and the correctness of

the output can be verified by checking this relationship, e.g., a sorting program. Other obviously incorrect results, such as arithmetic overflow, will also indicate an error. However, in most cases, the correctness of the output can only be verified by an algorithm which is just as complicated, and therefore unreliable, as the program to be tested, e.g., a compiler, a scheduler, etc. In these cases, a rigorous check on the correctness of the output is infeasible and we can only check the reasonableness of the output to determine its reliability. For example, if $\bar{x}$ is the solution of an optimization problem, the reasonableness of $\bar{x}$ can be validated by comparing the value of the objective function corresponding to $\bar{x}$ and other values of x. An unreasonable output usually indicated the presence of errors, but not vice versa. The formulation of an effective set of reasonableness checks is a difficult task. A common technique used is to check the consistency rather than the correctness of the output as the reasonableness criterion. For example, when a record corresponding to a key is located by a search, we can perform the reverse translation to verify that the record will produce that key. The consistency and validity of the input variables can also be checked before they are accepted. It is then hoped that no abnormal behavior of the program will be resulted when the input variables are processed.

3.2 Control Sequence Checking. The control sequence of a process refers to the sequence of computations executed. In terms of the program graph, this corresponds to a path from the entry node to the exit node of the graph. When each node represents a well-defined operation, the particular sequence in which these operations are executed will determine the outcome of a process. Any deviation from the correct execution sequence will lead to an unreliable result. Such an error in the control sequence is called a control fault and it can be roughly classified into one of the following groups:

- an infinite loop is executed
- a loop is executed an incorrect number of times
- an illegal branch (non-existent branch in the program graph) is taken
- a wrong branch is taken.

When the control sequence of a process is known and fixed, it is relatively simple to monitor the execution sequence to verify its correctness against any control fault. Infinite loops can be checked by defining an upper bound on the maximum number of times a loop can be executed. When the execution sequence is not fixed, we can check if the sequence is a valid (though not necessarily correct) execution sequence [4]. In this method, all valid control sequences can be derived from the program graph. At the entry to each node, a check is made to insure that the sequence up to this point is valid. Illegal branches are readily checked this way. An illegal branch can also be checked by a scheme called relay-runner [5]. In this scheme, a "baton", which is similar to a password, is carried along with the transfer of control and checked at appropriate check points. When an illegal branch is taken, control will not possess the valid up-to-date baton value and the error will be detected at the next check point, as illustrated in Fig. 1. Illegal branches can also be partially checked by defining

legal entry points as the only valid destinations of control from one part of the program to another. By limiting the area to which control can be transferred, a range check can also detect illegal branches partially. Other illegal branches can be avoided by using some defensive programming techniques.

There are, however, few effective techniques for checking wrong branching and incorrect loop termination. When a branch condition can be checked by an independent set of logical variables, some wrong branching can be detected. In the control fault detection method developed by Kane and Yau [4], wrong branches are detected by placing the decision logic in each successor node rather than the conventional practice of placing the decision logic in the predecessor node. In this way, each control decision is checked several times, leading to multiple simultaneous execution sequences in the case of an error in a single decision computation. When the number of times a loop should be executed can be calculated from the value of some variables, a consistency check can be performed on this number with the termination condition of the loop.

3.3 Data Checking. Although data usually refer to the information items to be processed by the program, they may be generalized to include all information stored within the computer system. In a stored-program control system, these include both program instructions and data. The mutilation of such information stored in the computer can be caused by residual software errors as well as hardware errors. Checking can be performed on the following aspects of the data:

- the integrity of data value
- the integrity of data structure
- the nature of data value.

The integrity of data value can be protected easily by maintaining a check sum of the content of a piece of code (instruction and data). The check sum can be checked periodically or at the time before a piece of code is executed to validate its integrity [5,6]. It is important to recompute the check sum if some legitimate modifications are made on the data during the execution. A simple check sum is the modular sum of the data values, as shown in Fig. 2. The integrity of state data stored in the system tables can also be checked by verifying their consistency with the actual states of the resources. Error-detecting codes can be employed to code the state of flags.

There are a number of techniques to check the integrity of the data structures, especially for linked lists. We can add an additional pointer to the end of the linked list and periodically trace the list to verify that the list is intact [7]. Redundant state information of the items can be kept to verify that items linked together satisfy the condition corresponding to the list [8]. Redundant linking can also be performed by using a bi-directional linked list place of a uni-directional linked list [6]. When an item is located, inserted, or deleted, a check can be made on the link in the opposite direction to verify its integrity after the corresponding operation.

The correct operation of the system can also be checked by monitoring the behavior of critical data variables during execution time. Data filter can be inserted to check that a piece of data does not fall outside its maximum and minimum value [9], as shown in Fig. 3. The mean and variance of important parameters can also be measured during execution time to detect abnormal behavior. Assertions on the relationship between variables can be inserted at appropriated check-points of the program to detect incorrect computations. This technique is especially effective if a computation can be divided into stages and intermediate results can be checked easily. The assertions formulated will be similar to those used in proving program correctness.

Self-checking techniques can be used at different levels: system level, program level, module level, instruction level, and data level. Software redundancy at different levels is employed to check the system at that level. The lower is the level, the narrower is the scope or error detection, but the finer is the error resolution. They can also be implemented at different levels of languages, such as design languages, high level languages, machine languages and microprograms. However, these techniques are of ad hoc nature and their error detection and correction capabilities vary greatly, so are the cost of implementing and using them. In order to use them effectively in error-resistant software design, overall consideration must be given to the entire systems design. In the following sections, we are going to discuss some of such system design techniques.

4. SYSTEM DESIGN OF ERROR-RESISTANT SOFTWARE. There have been many methods dealing with some specific aspect of error-resistant software system design, such as operating systems and system structures. Most of these techniques deal with the design of error-resistant operating systems. Because two survey papers have appeared recently [10,11] our discussions here will emphasize the system structures for error-resistance software.

The first comprehensive system structure we would like to discuss was proposed by Randell [12]. This structure is based on recovery block scheme of Horning et al.[13]. Basically, a recovery block consists of a conventional block with error detection capability (an acceptance test) and additional alternates. A possible format for simple recovery blocks is as follows:

```
ENSURE      ⟨acceptance test⟩
      BY    ⟨primary alternate⟩
ELSE BY     ⟨2nd alternate⟩
              :
              :
ELSE BY     ⟨Nth alternate⟩
ELSE ERROR
```

The primary alternate corresponds exactly to the block of the equivalent convention program, and is probably the most efficient one to perform the desired function. The acceptance test, which is a logical expression without side effects, is evaluated on exit from any alternate to determine whether the alternate has performed satisfactorily. A further alternate, if one exists, is entered if

the preceding alternate fails to complete or fails the acceptance test. Before an alternate is so entered, the state of the process is restored to that just before entry to theprimary alternate. If the acceptance test is passed, further alternates are ignored, and the statement following the recovery block is the next to be executed. However, if the last alternate fails, then the entire recovery block is regarded as having failed so that the block in which it is embedded fails to complete and recovery should be performed at that level. It should be pointed out that nested recovery blocks are often used.

In using the recovery block scheme to design error-resistant software, we have to consider the following items. First, we need effective acceptance tests for the individual alternates. Although some of the self-checking techniques discussed before among others [14], may be used as acceptance tests, methods for generating simple and powerful tests must be further developed in order to make this approach practical. This is probably the most crucial part for further development of this approach. Secondly, regarding various alternates, the primary one is the most efficient one to perform the function, and other alternates might perform the same function in some different manner, presumably less economically (one source of the other alternates may be earlier releases of the primary alternate). Thirdly, the automatic restoring of the system state is necessary to simplify error recovery. Whenever a process has to be backed up, the state that it had reached just before entry to the primary alternate must be saved. Therefore, the only values that have to be reset are those of nonlocal variables that have been modified. Since no explicit restart information is given, it is not known beforehand which nonlocal variables should be saved. Various versions of a mechanism which arranges that nonlocal variables are saved in the so-called "recursive cache" have been developed. This is accomplished by detecting, at execution time, assignments to nonlocal variables, and in particular, by recognizing when an assignment to a nonlocal variable is the first to have been made to that variable within the current alternate. Thus, precisely sufficient information can be preserved. Finally, error recovery among interacting processes must be carefully considered. When a process is implemented by a series of recovery blocks, the point to which the process must roll back in case of a error is not necessarily the starting point of the recovery block containing such an error, if this process interacts with other processes. An example is illustrated in Fig. 4, where each of the three interacting processes has a series of four recovery blocks. Should Process A fail at the end, it will be backed up to its latest, the fourth recovery point, but the other processes will not be affected. If Process B fails at the end, it will be backed up to the third recovery point of Process A. However, if Process C fails at the end, it has to be backed up all the way to the starting point of the three processes. This type of domino effect can occur when two particular circumstances exist in combination.

- The recovery block structures of the various processes are uncoordinated, and take no account of process interdependencies caused by their interactions.

- The processes are symmetrical with respect to error propagation - either of two interacting processes can cause the other to back up.

By removing either of these circumstances, the danger of the domino effect can be avoided. To avoid the first circumstances, process interactions should be properly structured. To avoid the second circumstance, the concept of multilevel processes can be used. For more detailed discussion of these results, the reader is referred to [12]. A further study on the coordination of recovery interacting processes has been made by Kim [15].

Recognizing the fact that a very large portion of the errors normally occurs in application programs, and the fact that error-resistant operating systems may be designed using different methods, Yau, Cheung and Cochrane [1] proposed a error-resistant software system structure with the emphasis on dealing with errors in the execution of applications programs. This approach uses the module level as the basic level of software error detection and recovery, and adopts the concept of the recovery block. The main distinction of this approach is that in order to protect the integrity of the error detection and recovery code against possible damage caused by the occurrence of errors, they are executed in a differene protection domain from that of application program modules. All the reliability codes are separated from the programs and placed under the jurisdiction of a piece of software, called the System Monitor. The System Monitor will then be responsible for monitoring the application programs to make sure that they are executed reliably. In a system with a hierarchical protection structure such as Multies [16], it will operate at a protection level between the level of the operating system and the application programs. It contains some user defined code, but also shares some of the capabilities of the operating system, such as communication with other operating processes and handling of errors detected by hardware. During execution, the System Monitor has the responsibility of ensuring that all interactions that a module has with other objects in the system are reliable. As each module in a program ends execution, regardless of whether it is because the module finished its processing or because the hardware detected an error, it will return control to the System Monitor. If the processing was completed, the System Monitor will then evaluate the functional reliability of the module's execution. If the module performs the specified function with acceptable performence, it will determine which module should be given control next and also what data values should be passed to the module's successors. It will then give control to the selected module. If an error is detected at any point, the System Monitor will initiate recovery procedures. This scheme guarantees that the reliability code will always be executed, and furthermore, it ensures that it will be executed in a reliable environment.

The internal structure of the System Monitor is shown in Fig. 5. There are five types of compononts in the System Monitor: the Internal Process Supervisors (IPS), the External Process Supervisors (EPS), the Interaction Supervisor (IS), the System Monitor Kernel (SMK), and the Maintenance Program (MP). The term Internal Process here refers to the execution of a program, and the term External Process refers to the access of global data shared among modules or programs.

There is one IPS for each internal process. It is responsible for checking whether the execution of its associated program is reliable. Its components are the Program Supervisor and the Module Supervisor. There is one Program Supervisor per program and one Module Supervisor per module.

The Program Supervisor is responsible for monitoring the flow of control and data between modules. The Module Supervisor is responsible for checking the functional reliability of the modules. Each module performs a function on the input parameters. Each module in a user's program may have the format as shown in Fig. 6. It is essentially a simple recovery block, except that it has both the entrance block containing validation tests to determine if the input data satisfying the specifications, and the acceptance block containing acceptance tests to determine if the module has successfully processed the input data and to validate the output before their values are stored back to the global data structures. However, both the entrace block and acceptance block are stored in the Module Supervisor, and only the functional block which contains the alternate versions performing the processing function is actually considered as in the module of the internal process shown in Fig. 5. Note that a module acts like an input-output transformer and all global data shared among modules have to be passed as input and output parameters.

There is one EPS created by the programmer for each global data structure. It is responsible for checking whether reliable data are available to the program modules. A global data structure may be shared among modules of a program or among several concurrent processes corresponding to different programs. The EPS contains the error detection routines as well as the information and code needed for recovery. In addition, the EPS may also include facilities that support abstract data types, such as operation clusters [17]. If an error is detected by the EPS, it may retry the access again from the storage device; it may use some error correcting codes in conjunction with the data store; or it may attempt to repair the damaged data by reconstructing them from redundant information stored by the EPS or from a safe backup copy. The EPS's of the data used by a program must cooperate with the IPS of that program during its recovery. For data structures shared by several concurrent internal processes, the synchronization requirements will be enforced by the next component of the System Monitor, the Interaction Supervisor.

The Interaction Supervisor (IS) is a portion of the operating system which is responsible for ensuring the reliability of interactions among processes. When an internal process attempts to interact with an object (another internal process or external process) in its external environment, it must do so via the IS. The IS will first determine whether the process is authorized to engage in the interaction. If the interaction is permissable, the IS will provide the facilities required for the interaction, such as process synchronization, message buffers, and then supervise the interaction to make sure that it is completed reliably. The IS may be implemented as an operating system monitor [18,19]. In the event that an interacting process fails, the IS will coordinate recovery among the processes that the faulty process has interacted with.

If the System Monitor is to be able to guarantee the reliable operation of the application software, it must have some capability of ensuring its own integrity. The System Monitor has a component in the operating system, called the System Monitor Kernel, which has the responsibility of checking the integrity of the Process Monitor. There will be a Maintenance Program stored in the operating system portion of the System Monitor. It will be

run during periods of light processing loads to attempt to "repair" faulty processes in the system.

The detailed description of the operation of the System Monitor and handling of various errors is given in [1]. The practicality of this approach, similar to that proposed by Randell [12], depends largely on the development of effective entrance tests and acceptance tests, which depends strongly on the application (function of the program) and modularization. Various schemes for implementing the individual components of the System Monitor must be developed.

5. SPECIFICATION OF ERROR-RESISTANT SOFTWARE. The development of a large-scale software system starts from the analysis of the user's requirements, and the results of this analysis will produce some requirement documents [20,21]. The requirements of a system, including functional, performance, and cost requirements, can be stated in a more formal format, which can be used as a set of guidelines during the design stage, such as a blueprint for deriving the system specification. It is well known that the high quality of the specification of the software system is the key to a successful design [22]. The specification of a software system consists of the control structure, which includes the data flow, and data types. Following certain guidelines, such as those suggested by Parnas [23], the system can be decomposed into subsystems and modules, and then the control structure of the system can be established. Although some specification techniques of the control structure have been developed [24,25], little work in the area of specification of data types has been done for error-resistent software design.

The concept of abstract data types was originally developed for high level programming languages, the word abstract indicates that it is implementation independent. The main purpose of data abstract in a programming language is to support the capability for constructing a more reliable program. An abstract data type consists of a set of data objects, a set of operations that acts upon the objects, and a set of descriptions that characterizes the operations. For example, a stack with elements being integers and five stack operations, namely, POP, TOP, PUSH, NEWSTACK, and ISNEWSTACK can be specified, using algebraic specification approach [26], as follows:

```
TYPE Stack [item]
   DECLARE  NEWSTACK → Stack
            PUSH (Stack,item) → Stack
            POP (Stack) → Stack
            TOP (Stack) → item
            ISNEWSTACK (Stack) → Boolean
   FOR ALL    s ∈ Stack, i ∈ item  LET
         ISNEWSTACK (NEWSTACK) = true
         ISNEWSTACK (PUSH (s,i)) = false
         POP (NEWSTACK) = NEWSTACK
         POP (PUSH (s,i)) = s
         TOP (NEWSTACK) = UNDEFINED
         TOP(PUSH (s,i)) = i
   END
END STACK
```

In this example, it only gives an implicit definition on the possible values of data objects, i.e. the stack itself, integers, the definitions of operations and their interactive behavior. However, the details on how these data objects and operations are implemented need not be considered at the design stage, and hence the designer will have more flexibility on developing the correct structure of the program during the early part of the design stage. Such a facility in programming language will give the designer more convenience and control over his development of the program, and also more confidence in the reliability of the final software product because this idea is consistent with the concept of step-wise refinement. For the development of large-scale software systems, the same strategy can also be adopted in the design methodology to reduce the development cost and still retain the reliability of the system. Moreover, the abstract data types can be treated as system resources, such as disks and other I/O devices; and be controlled and protected under some mechanism, so that the overall protection problem in the developing cycle can be simplified.

Although the concept of abstract data types has been applied to the development of software systems, as in HOS [25], the lack of a formal specification technique of abstract data types makes the concept still not an intrinsic one to the design methodology [27]. Here, the formality of the specification techniques means the specification can be constructed following certain rigorous disciplines and the resultant specification can be proved correct. Some work has been done in this area [26,28,29], but the results have to be made more practical for the designer to use [29] and effective methods for checking the correctness of the abstract data type specification need to be developed [26,27]. Furthermore, the effects of possible errors that may be associated with the abstract data types must also be studied. For example, requesting the top element of a stack will be reasonable only if the stack is not empty; otherwise, it will be an undefined condition as shown in the previous example. Although we need not to know the details on how to recover from error conditions during the design stage,they will become more clear during the implementation stage if there are some error detection and processingmechanisms in the specification. For example, in the previous example of stack before executing the operation POP, a checking on the emptiness of the stack will avoid an illegal execution. With some modifications of the specification, the example we just mentioned can be specified as follows:

```
TYPE: Stack [n,item]
ASSERTIONS: n ≤ 100
OPERATIONS:
            .
            .
            .
            POP:
                IF (n .EQ. Ø) THEN ERROR 1
                ELSE   s = POP (s,n-1)
            .
            .
            .
            BLOCK ERROR 1
                BEGIN
                     MESSAGE */Stack is empty/*;
                END
            END ERROR 1;
            .
            .
            .
```

Here, n represents the number of elements in the stack s, and we use an assertion to limit it to at most 100 elements. This is just the same as performing an input validation test on the data object n as suggested in [1].

In order to have an effective error-resistant software design methodology it is necessary to have an integrated specification technique that can unambiguously and clearly specify the control structure as well as the desired design data type. The error-resistant software specification generated should be proved to be correct. Features such as error detection and processing should be an integral part of the specification technique to guarantee the completeness and correctness of the specification. As discussed before, much work must be done in this area.

6. CONCLUSIONS AND FUTURE WORK. Error-resistant software can provide ultra reliability and continuous availability of computing resources, which are vital in many critical real-time applications. From the discussion of this paper, we can see that substantial progress has been made in this area during the past several years. However, in order to have a complete and cost-effective methodology for error-resistant software design, much further research and development have to be done in this area. Many specific problems that need to be solved, ranging from system requirements, specifications, structures, and testing (checking), have also been discussed in this paper. It is our hope that this paper will stimulate the interest of more researchers to work in this important area so that cost-effective methodologies for large-scale error-resistant software design can soon be developed.

ACKNOWLEDGEMENT. The author would like to express his thanks to Messrs. Mehmet Caglayan and Chen-Chau Yang for their assistance provided for this work.

REFERENCES

[1] S. S. Yau, R. C. Cheung, and D. C. Cochrane, "An Approach to Error-Resistant Software Design," Proc. Second International Conf. on Software Engineering, October, 1976. San Francisco; pp. 429-436.

[2] S. S. Yau and R. C. Cheung, "Design of Self-Checking Software," Proc. 1975 International Conf. on Reliable Software, March, 1975. Los Angeles; pp. 450-457.

[3] G. J. Meyers, Reliable Software Through Composite Design. Petrocelli/Charter, 1975.

[4] J. R. Kane and S. S. Yau, "Concurrent Software Fault Detection," IEEE Trans. on Software Engineering, Vol. 1. March, 1974. pp. 87-99.

[5] C. V. Ramamoorthy and R. C. Cheung, "Integrity of Large Software Systems," Proc. 3rd Annual Texas Conf. on Computing Systems. October, 1974.

[6] B. Randell, "Operating Systems: The Problems of Performance and Reliability," Information Processing 71, pp. 281-290.

[7] R. S. Fabry, "Dynamic Verification of Operating System Decisions," Comm. ACM, Vol. 16, No. 11, Nov, 1973; pp. 659-668.

[8] J. R. Connet, E. J. Pasternak and B. D. Wagner, "Software Defenses in Real-Time Control Systems," Proc. 1972 Int. Sym. on Fault-Tolerant Computing," pp. 94-99.

[9] C. V. Ramamoorthy, R. E. Meeker and J. Turner, "Design and Construction of an Automated Software Evaluation System," IEEE Sym. on Computer Software Reliability, 1973.

[10] P. J. Denning, "Fault-Tolerant Operating Systems," ACM Computing Survey, Vol. 8, No. 4, December, 1976; pp. 359-389.

[11] T. A. Linden, "Operating System Structures to Support Security and Reliable Software," ACM Computing Survey, Vol. 8, No. 4, December, 1976; pp. 409-445.

[12] B. Randell, "System Structure for Software Fault Tolerance," IEEE Trans. on Software Engineering, Vol. SE-1, No. 2, June, 1975; pp. 220-232.

[13] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell, "A Program Structure for Error Detection and Recovery," Proc. Conf. Operating Systems: Theoretical and Practical Aspects, IRIA, April 23-25, 1974, pp. 177-193.

[14] T. Anderson and R. Kerr, "Recovery Blocks in Action: A System Supporting High Reliability," Proc. 2nd International Conf. on Software Engineering, October, 1976, pp. 447-457.

[15] K. H. Kim, "Programmer-Transparent Coordination of Recovering Parallel Processes: Philosophy and Rules for Efficient Implementation," submitted for publication.

[16] J. H. Saltzer, "Protection and Control of Information in Multics," Comm. ACM, Vol. 17, No. 7, July, 1974, pp. 388-402.

[17] B. Liskov and S. Zilles, "Programming with Abstract Data Types," Proc. ACM Conf. on Very High Level Languages, SIGPLAN Notices, Vol. 9, April 1974, pp. 50-59.

[18] B. Hansen, Operating System Principles, Prentice-Hall, 1973.

[19] C. A. R. Hoare, "Monitors: An Operating System Structuring Concept," Comm. ACM, Vol. 17, No. 10, October, 1874, pp. 549-557.

[20] M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," IEEE Trans. on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 60-69.

[21] C. G. Davis and C. R. Vick, "The Software Development System," IEEE Trans. on Software Engineering, Vol. SE-3, No. 1, Jan. 1977, pp. 69-84.

[22] P. C. Belford, A. F. Bond, D. G. Henderson and L. S. Sellers, "Specification: A Key to Effective Software Development," Proc. 2nd International Conf. on Software Engineering, Oct. 1976, pp. 71-79.

[23] D. L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," Comm. ACM, Vol. 15, No. 12, Dec. 1972, pp. 1053-1058.

[24] P. E. Lauer and R. H. Campell, "Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes," Acta Informatica, Vol. 5, 1975, pp. 297-332.

[25] M. Hamilton and S. Zeldin, "High Order Software - A Methodology for Defining Software," IEEE Trans. on Software Engineering, Vol. SE-2, No.1, March 1976, pp. 9-32.

[26] J. Gutlag, E. Horowitz, and D. Musser, "The Design of Data Type Specification," Proc. 2nd International Conf. on Software Engineering, Oct. 1976, pp. 414-420.

[27] B. H. Liskov and S. N. Zelles, "Specification Techniques for Data Abstractions," IEEE Trans. on Software Engineering, Vol. SE-1, No.1, March 1975, pp. 7-19.

[28] J. A. Gognen, J. W. Thatcher, and E. G. Wagner, "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Type," IBM Thomas J. Watson Research Center Research Report 6487, Oct. 1976.

[29] S. Cushing, "Algebraic Specification of Data Types in High Order Software," HOS, Inc. Technical Report 13, Jan. 1977.

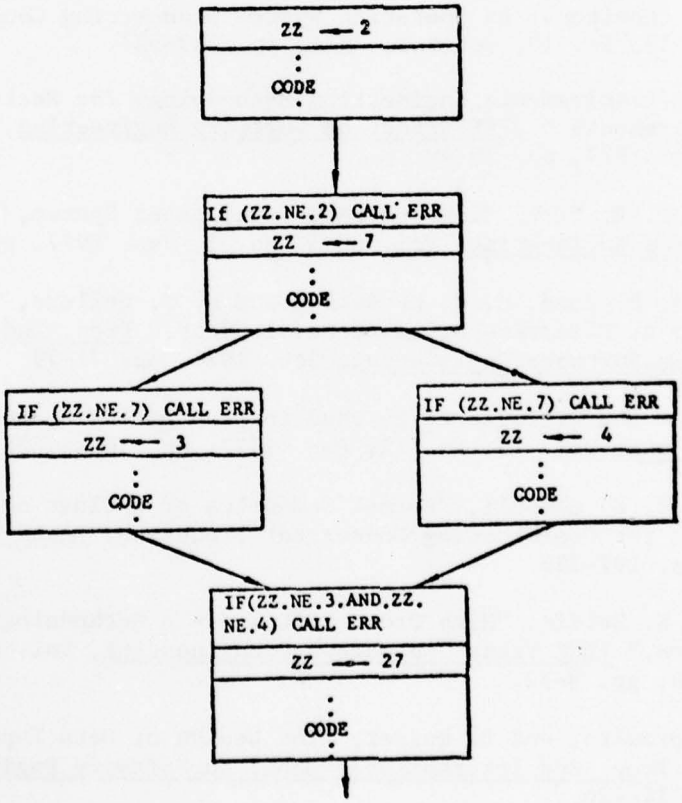


Fig. 1. An example to illustrate the relay runner scheme (for detecting illegal branches in a control structure during execution).

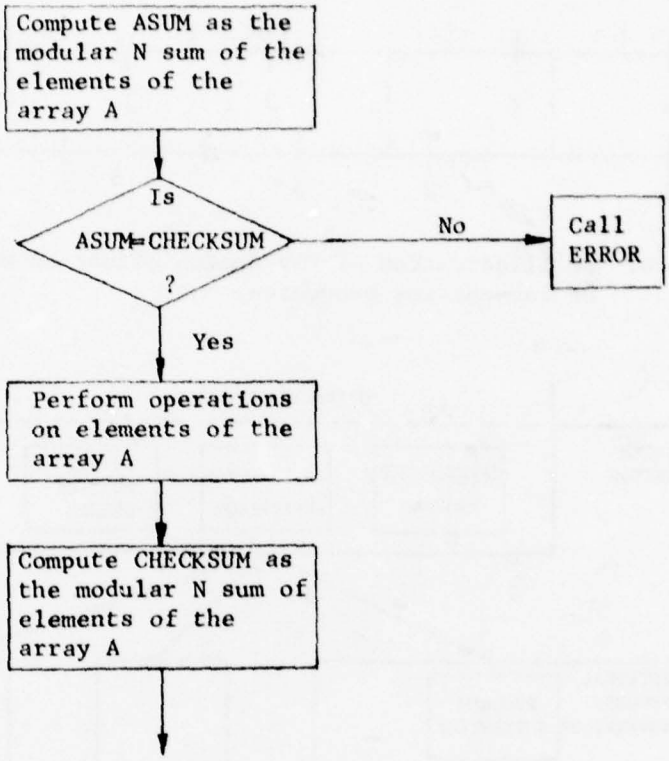


Fig. 2. A simple checksum scheme for checking the integrity of data value.

An upper bound XMAX and a lower bound XMIN for the value of a variable X are specified by the programmer. The following statements are inserted after each assignment of X:

```
X = ...
IF (X.GT.XMAX) CALL ERROR (X)
IF (X.LT.XMIN) CALL ERROR (X)
```

Fig. 3. A data filter for checking the range of values of a data variable.

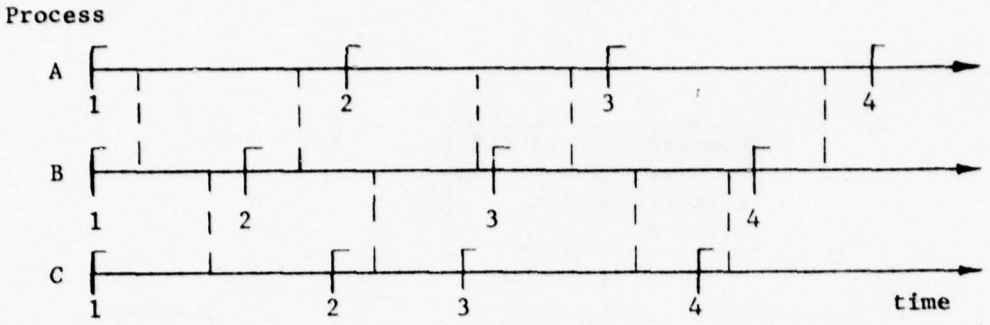


Fig. 4. An illustration of the domino effect on error recovery of interacting processes.

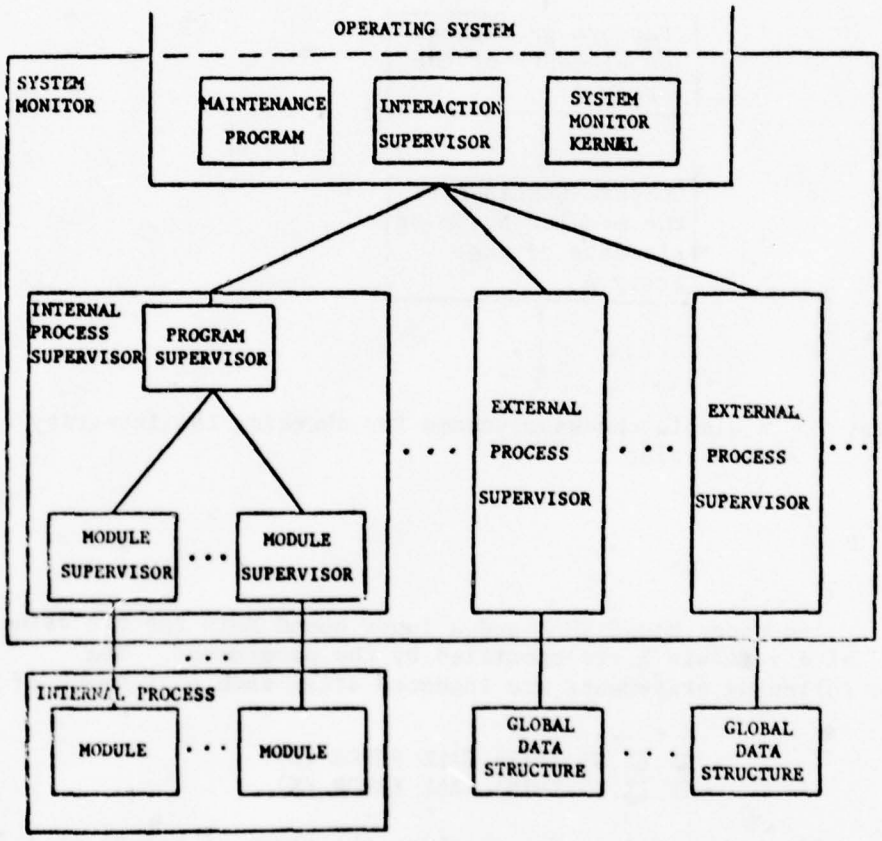


Fig. 5. The structure of the System Monitor used in the error-resistant software design [1].

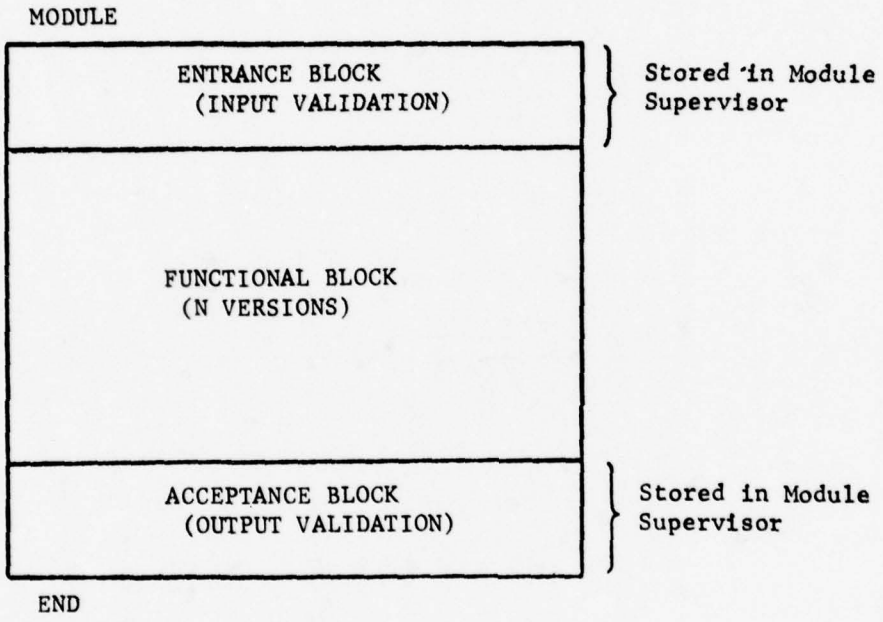


Fig. 6. A user program module used in the error-resistant software design given in [1].

PRIMARY PROGRAM DESCRIPTIONS: WHY WE NEED A NEW APPROACH TO CORRECTNESS

Paul Broome
Chemical Systems Laboratory
Research Laboratory
US ARRADCOM
Aberdeen Proving Ground, MD 21010

and

Leon S. Levy
Department of Statistics and Computer Science
University of Delaware
Newark, Delaware 19711

ABSTRACT. A new synthesis of correctness methodology is needed to provide reasonable explanations of even simple programs as illustrated by example. The example suggests that current approaches to correctness must be modified to present understandable and convincing arguments that a program accomplishes its intended objectives.

Our approach, as illustrated by example, defines a class of primary program descriptions. A primary program description is to meet the following criteria:

a. It is understood by the intended audience.

b. It is algorithmically transferred to the object program.

c. Its correctness is convincing to the intended audience.

In the present paper we focus on some linguistic issues in primary program descriptions in so-called pidgin-ALGOL programs.

1. INTRODUCTION. The software problem is now, and for the foreseeable future will be, one or the major technical problems with which we will have to contend. The scope of the problem has been documented by Boehm [1]:

"The annual cost of software in the U.S. is approximately 20 billion dollars. Its rate of growth is considerably greater than that of the economy in general .... A recent study by Fisher [2] found that DoD's software expenditure during FY 1973 was roughly $2.9-3.6 billion."

The following data of Brandon [3] accentuate the gravity of the problem:

PRECEDING PAGE BLANK

"	1976	1980	1986
Total expenditure on computing*	$ 35 billion	$90 billion	$139 billion
% of GNP accounted for by the Computing Industry	3%	6%	7%

*Includes hardware, programming, supplies, etc: as a comparison, the 1975 figure for car sales in the U.S.A. was $38 billion."

Clearly with a problem of such magnitude, an extensive scientific endeavor at different levels is appropriate to try to control the problem. Indeed, a variety of approaches is being tried, and we shall briefly survey these in Section 2.

In Section 3, we give an example of a relatively simple program which illustrates, in miniature, the software problem. This program is much smaller and less complex than typical operational programs, and thus allows us to deal with it in this paper. Still even this miniature program is quite difficult enough.

Section 4 starts with an analysis of the difficulties of programming, as exemplified in Section 3, from both the theoretical and practical points of view. We then describe some of the criteria for a solution to the problem, in what we call a primary program description.

2. APPROACHES TO THE SOFTWARE PROBLEM. The literature on approaches to the software problem is so extensive already that a complete bibliography or survey would be an encyclopedic work. Even an incomplete and selective annotated bibliography in 1975 [4] contained over 300 items, approximately 3/4 of these from the 1972-1974 period. Further, the rate at which articles and books on the software problem appear seems to still be increasing with new journals and conferences.

In view of the vast literature, we shall only try to give a brief sampler of some of the major approaches to the software problem. For our purposes, we classify the approaches in four categories: management approaches, formal correctness, mechanized verification, and documentation. This certainly does not exhaust the possibilities. Table 1 contains a listing of the headings under which papers at a recent software engineering conference [5] were organized and is instructive. (This conference alone had over 100 papers). Many of the approaches described in the literature combine two or more of our four categories. Still others concentrate either on a particular class of software, or the requirements of a specific group of users.

The first of our four categories is management. Here, two approaches in particular, chief programmer team [6] and egoless programming [7] indicate the diversity. Essentially, the chief programmer team concept can be

Table 1. Categories of Papers in the Software Engineering Conference

Requirements Definition
Program Synthesis Techniques
Operating Systems
Requirements Engineering
Education
Operating Systems and Networks
Performance Evaluation
Programmer's Workbench
Software Design and Development
Design of Large Programs
Programming Languages and Systems
Software Design
Software Engineering in the Department of Defense
Software Verification and Validation
Program Proving and Verification
Theoretical Aspects of Software Engineering
Software Fault Tolerance
Validation and Testing
Data Bases
Case Studies
Software Automated Tools
Programming Languages
Software Modeling
Design Specification and Management

understood as structurally similar to a surgical team. The chief programmer is aware of and responsible for all of the programming process, although some of the details are delegated to other members of the team. In the egoless programming approach, a less hierarchical management structure is postulated, and the emphasis is placed on information exchange among programmers.

Another category is formal correctness which has ben especially emphasized in research based approaches. Here the papers which marked the beginning of the major activity in proving correctness were Floyd's [8] and a variant later introduced by Hoare [9]. The basic idea in these approaches is that a specification of the program requirement is written in a formal language, typically a first order logic, and one then attempts to prove, using the deduction rules of the formal system, that the program satisfies the specification.

A variant of this approach is developed by Dijkstra [10]. In Dijkstra's method, a specification of the program is written in a formal language, and the program is developed from the specification.

A number of experimental implementations of programs to perform verification have been developed. King [11] was the earliest of these, implementing Floyd's method for a (decideable) subset of Algol. Other systems have concentrated on other languages with a different style; e.g. Boyer and Moore [12] on LISP. Experimental systems intended for program development, rather than verification, have also been described, such as Burstall and Darlington's interactive system [13].

The approaches based on formal logic have been described in a number of textbooks, of which the most widely used is Manna [14]. More recently, Manna and Waldinger [15] and London [16] have surveyed both the theory of proofs that programs meet their specifications, and experimental systems.

The logic based approach has also been the subject of some controversy. Demillo, Lipton, and Perlis [17] and Dijkstra [18] is a lively exchange on the subject.

Finally, in the area of documentation, Teicherow and Hershey [19] is perhaps the best known. PSL/PSA (problem statement language/problems statement analysis) uses the computer to store and analyze requirements (this is presumably synonymous with specification) and assists in preparing documentation. A less ambitious documentation program that is also well-known is Mills [21]. Mills starts with the program text, and uses an interactive system to develop documentation to annotate the program.

In the next section we will discuss a "toy" program and return to consider, in the light of this, what should be the nature of a solution to the software problem.

3. EXAMPLE. The program in this section is not large, but it is also not easy to understand. We will examine this program in detail and then attempt a presentation of this program so that its design is displayed.

The example is a program for inverting a permutation without using an additional array. This program appeared in 1964, but was revised and simplified to appear again the next year. Since 1965 we have seen other presentations of this program (Burstall, [22], Knuth [23]), but they are not unlike the 1965 version which is shown:

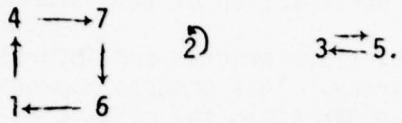
```
1   For m: = n step -1 until 1 do
2   begin i: = A [m];
3         if i < o then A [m]: = -i
4         else if i ≠ m then
5          begin k: = m;
6                while i ≠ m do
7                      begin j: = A [i]; A [i]: = -k;
8                            k: = i; i: = j
9                      end;
10               A [m]: = k;
11        end
12  end
```

Program 1: A program that inverts a permutation, A, in situ.

Before examining this program we must agree on some definitions. A permutation is an arrangement of n distinct objects in a row. If the objects are the integers {1,2, ..., n} and A [1] A [2] ... A [n] is a permutation, then the inverse permutation B [1] B [2] ... B [n] is the rearrangement that undoes the effect of A. Thus if i goes to j under A then j goes to i under B.

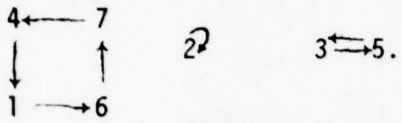
Every permutation consists of one or more disjoint cycles. An example of a permutation along with its cycle structure is:

```
     i:  1 2 3 4 5 6 7
A [i]:   4 2 5 7 3 1 6
```



The corresponding inverse permutation, B, and its cycle structure is:

```
     i:  1 2 3 4 5 6 7
B [i]:   6 2 5 1 3 7 4
```



The cycle structure of B is similar to that of A except that the arrows have reversed directions.

There is a very simple method for computing the inverse of A: set B[A[i]] := i for $1 \leq i \leq n$. Then B[1] B[2] ... B[n] is the desired inverse. This method uses 2 n memory cells, n for each array.

The concern for memory storage requirements can be reduced by computing the inverse without using an additional array, so that when the program is finished A [1] A [2] ... A [n] is the inverse of the original permutation.

An outline of the algorithm is:

```
For each m between 1 and n do

    if cycle beginning at m has not been inverted then
       invert and mark each element of the cycle (except for m)

    else
         remove the marker from this element (m).
```

This algorithm inverts one cycle at a time, marking each element (with a negative sign (-)). When the algorithm later encounters one element with a marker, the marker is removed.

Program 1 has few variables. The idea of the algorithm is simple, but the program causes us difficulty. How do we determine correctness for this program? How can we retain the link between the algorithm and the program so that when the program is written the design is not thrown away?

The communication medium between the programmer and the computer is many times the source of the problem. When we must contort our ideas to fit the syntax of a particular programming language, the idea is sometimes lost. Luckily we are beginning to realize that a programs' purpose is not to instruct the computer but that the computer is there to execute our program.

Nonetheless, even if we accept the constraints that machines and programming languages impose on us, something can still be done to better represent the idea of Program 1.

One major cause of errors in a program is the concern for (machine) efficiency. Only if we first know that a program is correct can we then be concerned about efficiency. If a program is not correct, it matters little how fast it runs.

In line 4 of Program 1 there is a test for $i \neq m$ so that we can avoid inverting a singleton cycle, even though the cycle inversion works for a singleton cycle. But the average number of singleton cycles in a permutation of size n is just 1 (Knuth [23] , p. 178). So this attempt to improve efficiency has actually decreased efficiency.

Because of the early introduction of the variable, i, it is not readily apparent that the test $i \neq m$ is a test for a singleton cycle. An attempt such as this to avoid reevaluation of $A[m]$ is usually unnecessary because most compilers can recognize a duplicate expression and avoid a recalculation for us.

In order to obtain a better program we should return to the design stage. Another outline of the algorithm is:

```
For m: = 1 to n do

    if cycle at m has not been inverted then

       invert and mark every element of the cycle

    remove the marker from this element.
```

It is more natural to take m from 1 to n than to go the opposite way. We can avoid special cases by marking (with a minus sign (-)) every element of the cycle, whereas Program 1 leaves element m unmarked.

The most difficult part of Program 1 is that for chasing around a cycle, especially the part for moving from one element to the next. All ways of moving around a cycle are similar in that they have the following outline:

```
start at m
while not done do
      begin
      process the current element
      move to the next element
      end.
```

As an example, consider the problem of finding an element x which we know to be somewhere in the cycle. The program part:

```
i: = m;
while A [i] ≠ x do
      i: = A [i];
```

will return i such that A[i] = x. Other than the array, A, and the value, x, this program part requires one other variable, the variable i. As we shall see, the number of additional variables indicates the complexity of the process.

As a second example of loop chasing, assume that we want to break the cycle into all singleton cycles. The program part:

```
i: = m;
j:= A[i];
while A[i] ≠ i do
      begin
      A [i]: = i;
      i: = j;
      j: = A[i];
      end;
```

requires two additonal variables, i and j. Before we modify A[i] to be i we must have a way of getting to the next element, hence the need for previously saving this next element in j.

The program part for finding an element x does not modify the cycle so there is no statement comparable to A[i] := i as in the last example; yet, both examples have a similar pattern of movement.

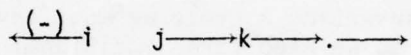
In both examples the conditions for loop termination accurately represent the functions of the programs parts. In the first program part we desire to have A[i] = x. When this happens we leave the loop. In the second program part we must have A[i] = i. When this happens we leave the loop because this element (i) was previously visited.

When inverting a cycle we must nave knowledge of three (adjacent) elements, so we need three additional variables; the current element, the last element (where we want the current one to point), and the next element (where the current one now points). The program part:

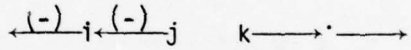
```
i := m;
j := A[i];
k := A[j];
while A[j] ≠ -i do
    begin
    A[j] := -i;
    i := j;
    j := k;
    k := A[j]
    end
```

inverts a cycle and negates each element to indicate the visit. Note that the terminating condition (A[j] = -i) and the first statement of the loop (A[j] :=i;) are very similar.

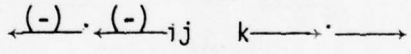
The part of the loop body for moving to tne next element is reminiscent of a centipede that is crawling. Note the effect of the loop body on a segment of a cycle.



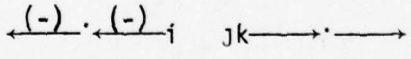

```
A[j] := -i;
```




```
i := j;
```




```
j := k;
```




```
k := A[j];
```



We finally arrive at the program shown as Program 2.

```
1    For m: = 1 to n do
2        begin if A[m] > 0 then
3                begin i: = m;
4                      j: = A[i];
5                      k: = A[j];
6                      while A[j] ≠ -i do
7                            begin A[j]: = -i;
8                                  i: = j;
9                                  j: = k;
10                                 k: = A[j]
11                           end
12               end;
13               A[m]: = -A[m]
14       end.
```

Program 2: Invert a permutation, A, in situ.

This program examines each element m. If the element at m is positive then we have found a loop that has not yet been inverted. So we invert it. Therefore at line 13, A[m] will always be negative.

The assignment statements at lines 3-5 and at lines 8-10 point out that i, j, and k are adjacent elements of the cycle (the last, current and next elements). The test in line 6 (A[j] ≠ -i) points out that the value A[j] is not what we wish it to be. The situation is thus remedied in line 7.

Not only is Program 2 easier to understand than Program 1, but it also executes faster. The execution time of Program 2 can be improved even more (by inverting A[m] outside the while loop), but it will then be a larger (more difficult to understand) program.

4. PRIMARY PROGRAM DESCRIPTIONS. The example of the previous section indicates that even very simple programs are difficult to understand and, consequently, will be hard to certify with any degree of assurance. Still we should like to suggest what we think the major difficulties are in constructing correct programs.

Undecideability of Equivalence

Given two programs, or a program and a specification, it is known that there is no algorithm to decide whether the two are equivalent. The consequence of this theoretical result is that we should find it difficult to show that a program and its specification, or that two programs independently developed, are equivalent. For each of the programs given in the preceding section, a proof that the program meets its specification is indeed somewhat difficult.

The Competence-Performance Dilemma

Another reason that it is hard to write correct programs is that we are often tempted to write programs that are about as complicated as we can manage. Precisely because these programs are at the limits of human performance, they are also difficult to debug. Of course, the syntax of programming languages only deals with the competence aspect of language and ignores this "human engineering" dimension. Restricting program text to a single page, or eliminating goto's can only be partial solutions.

What you see is not what you get

The program text presented as the solution to a problem does not indicate its derivation, or the underlying principles used to arrive at the program. This program deep structure [25] is essential to an understanding of the program, and the correction of bugs almost invariably requires such an understanding. As noted in [25] and [26], a program developed rationally at the level of deep structure and then optimized via transformations is perfectly acceptable, providing that the deep structure and transformations are part of the documentation.

We have commented elsewhere [27] on the particular difficulties of developing and understanding programs in systems that require the programmer to do optimizing transformations. An example which is generally familiar, is the one-pass vs. two-pass assembler. The two pass assembler is much clearer than the one-pass assembler, since the one-pass assembler is essentially a transformed and optimized version of the two-pass assembler.

Primary Program Descriptions

If our analysis is correct, then a proper methodology for program correctness must be based on the following principles:

i) There must be a document completely describing the program.

ii) The program description must be understood by its intended audience at a level well within the limits of their performance.

iii) The documentation must include all of the deep structural information, and all of the relevant transformations.

We will call a description satisfying these criteria a primary program description (PPD). No particular syntax is prescribed for the PPD as the following examples illustrate:

a) A payroll program may use an annotated decision table as a PPD.

b) A lexical scanner [28] may use an annotated finite state transducer as a PPD.

c) A depth first search algorithm [29] may use an annotated pidgin-Algol program as a PPD.

d) Various flowcharts or structured flowcharts with appropriate commentary can serve as PPD [30].

The important point is that if different forms of PPD are used with the same program, then the correspondence between the PPD's must be established algorithmically. Moreover, there will generally be a syntax-directed translation between PPD's using algebraic methods as described in [31].

Conclusion

We do not claim that what we have presented here will solve or even significantly reduce the software problem. But it seems to us that an approach to correctness should include the concept of a primary program description as a basis for program management and for all levels of documentation.

References

1. Boehm, B.W. Software Engineering: R&D Trends and Defense Needs. Presented at the Conference on R&D Problems in Software. Brown University, Oct. 1977.

2. Fisher, D.A. Automatic Data Processing Costs in the Defense Department. Institute for Defense Analysis, Paper page P-1046. Oct. 1974.

3. Brandon, D.H. Commercial Software in Software Portability: An Advanced Course. Ed. P.J. Brown, Cambridge University Press, 1977, pages 203-212.

4. Newton, G.E. A Partially Annotated Bibliography of Top-Down and Goto-less Programming. Proceedings IRIA Colloquium on Proving and Improving Programs. Rocquecourt, France, July 1975, pages 435-473.

5. Proceedings 2nd International Conference on Software Engineering. San Francisco, Cal., Oct. 1976.

6. Baker, F.T. Chief Programmer Team Management of Production Programming. IBM Systems Journal, Vol. 11, No. 1, pages 56-73, 1972.

7. Weinberg, G.M. The Psychology of Computer Progragming. Van Nostrand Reinhold Co., 1971.

8. Floyd, R.W. Assigning Meaning to Programs in Proceedings of a Symposium in Applied Mathematics. Vol.19, J. Schwartz, Ed. AMS, pages 19-32, 1967.

9. Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Communications ACM, pages 576-580, Oct. 1969.

10. Dijkstra, E.W. A Discipline of Programming. Prentice Hall, 1976.

11. King, J.C. A Program Verifier. Ph.D. Thesis, CMU, 1969.

12. Boyer, R.S. and Moore, J.S. Proving Theorems about LISP Functions. JACM, Vol. 22, No. 1, pages 129-144, Jan. 1975.

13. Burstall, R.M. and Darlington, J. A Transformation System for Developing Recursive Programs. JACM, Vol. 24, No. 1, pages 44-67, Jan. 1977.

14. Manna, Z. Mathematical Theory of Computation. McGraw Hill, 1974.

15. Manna, Z. and Waldinger, R. The Logic of Computer Programming. Aug. 1977, Stanford Report #STAN-CS-77-611.

16. London, R.L. Program Verfication. In Conference on R&D Problems in Software, Brown University, 1977.

17. DeMillo, R.A., Lipton, R.J., and Perlis, A.J. Social Processes and Proofs of Theorems. Proceedings 4th Symposium on the Principles of Programming Languages, Los Angeles, pages 144-154, 1977.

18. Dijkstra, E.W. A Political Pamphlet from the Middle Ages. Unpublished Response to [16].

19. Teicherow, D. and Hershey, III, E.A. PSL/PSA, A Computer Aided Technique for Structured Documentation and Analysis of Information Systems. Abstract in Proceedings 2nd International Conference on Software Engineering, San Francisco, Cal., 1976, page 2.

20. Teicherow, D. and Hershey, III., E.A. Documentation Methodology. In Proceedings, Conference R&D Problems in Software, Brown University, 1977.

21. Mills, H.D. Syntax-Directed Documentation for PL 360. Communications of the ACM, Vol. 13, pages 216-222, April 1970.

22. Burstall, R.M. Proving Programs as Hand Simulation with a Little Induction. Proceedings, IFIP Congress, Stockholm, pages 308-312, 1974.

23. Knuth, D.E. Fundamental Algorithms. Addison Wesley, 1969.

24. Gries, D. An Exercise in Proving Parallel Programs Correct. Communications, ACM, Vol. 20, No. 12, pages 921-930.

25. Levy, L.S. and Melville, R. The Algebraic Anatomy of Programs. The Computer Journal, Nov. 1977.

26. Knuth, D.E. Programming with GOTO's. Computing Surveys, Dec. 1974.

27. Leathrum, J. and Levy, L.S., unpublished memo, July 1977.

28. Aho, A.V. and Ullman, J.D. Principles of Compiler Design. Addison Wesley, 1977.

29. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. The Design and Analysis of Algorithms. Addison Wesley, 1974.

30. Carberry, S., Khalil, H., Leathrum, J.F., and Levy, L.S. General Computer Science. Charles E. Merrill, 1976.

31. Levy, L.S. Discrete Structures. Wiley, forthcoming.

THE BRL BESSEL FUNCTION SUBROUTINE

Kathleen L. Zimmerman (Speaker)
Alexander S. Elder (Co-Author)
Propulsion Division
U.S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005
Autovon 283-3083

ABSTRACT. A subroutine for calculating Bessel functions of integral order and complex argument has been developed at the BRL and has been run on the UNIVAC 1108, the BRLESC, and the CDC 7600 computers. The requirements for accuracy, simplicity, generality, and reduction of round-off error have been satisfied by the computational methods used. While the input can be given in either cartesian or polar coordinates, all of the calculations are carried out in terms of cartesian coordinates (x+iy=z, where z is complex).

Three methods of calculation were chosen to compute the ordinary and modified Bessel functions of the first and second kind:

1. Weber-Schlafli series for $|z| \leq 2.5$,
2. Gauss continued fractions and recurrence formulas for $2.5 < |z| \leq 21.0$, and
3. Hankel asymptotic series for $|z| > 21.0$.

The computation of the Weber-Schlafli series for small values of z is straightforward. Terms of the infinite series are summed from the smallest term to the largest term in order to avoid cancellation error. Bessel functions for large values of z are calculated using Hankel asymptotic functions in the right half-plane. Analytic continuation is used to compute corresponding functions in the left half-plane.

Bessel functions of the first kind for moderate values of z are calculated by selecting an order that is much larger than $|z|$ and applying a downward recurrence relation using a Weber-Schlafli series approximation for the starting value. Although this series can be used to start a similar recurrence for modified Bessel functions of the second kind, functions of lower order cannot be calculated accurately as the difference of two nearly like numbers occurs in the course of the computations. The innovative use of Gauss continued fractions has yielded accurate results for the computation of $K_n(z)$. The ordinary Bessel functions of the second kind are calculated in terms of Hankel functions which are linear combination of ordinary Bessel functions.

The results of the subroutine are generally good to at least 13 or 14 significant figures for small and moderate orders; the accuracy falls off, however for the higher orders. The accuracy has been verified for selected ranges of argument and order.

1. INTRODUCTION. The development of this subroutine began in the late 1960's when Mr. Elder, the principal investigator, realized that Bessel functions of complex argument were needed to solve a general class of problems involving the Laplace and biharmonic equations in cylindrical coordinates. In particular, the evaluation of Fourier integrals by the theory of residues required complex eigenvalues to a high degree of accuracy. Although tables of Bessel functions in complex argument were available at that time, they were limited in scope and accuracy. Interpolation for intermediate values caused a further loss in accuracy.

The mathematical analysis was essentially completed in 1968 by Mr. Elder. Mrs. Alene Depue completed the programming shortly thereafter. Due to a serious illness which eventually caused her early retirement, Mrs. Depue did not document the program. As too often happens, the computer code was used and the report ignored. In 1976 I was assigned the task of writing the report on this subroutine. Since there were virtually no notes around, it was necessary to reconstruct the entire derivation before beginning to write the report. The text was completed last summer, and the report is nearly ready for publication.

2. OBJECTIVES AND SCOPE. During the early seventies more interest was shown in this work. Other good codes have been developed, generally for real variables, but it is felt that this one still meets the requirements for accuracy, simplicity, generality, and reduction of round-off error when calculations involving complex arguments are required. The program calculates ordinary and modified Bessel functions of complex argument and integral order from 0 to 40. Pairs of orders, m and n=m+1, and the functions of both the first and second kinds are calculated for either the ordinary or modified Bessel functions in any given call to the subroutine.

3. STRUCTURE. There are four main sections in the code:

a. Preliminary calculations

b. Evaluation for small z: $|z|<2.5$

c. Evaluation for moderate z: $2.5<|z|\leq 21.0$

d. Evaluation for large z: $|z|>21.0$

4. COMPUTATIONAL METHODS. The first section converts the input, if it was given in polar coordinates, to cartesian coordinates. If cartesian coordinates are entered, the calculation of the polar coordinate θ is computed using the half-angle formula for ARCTAN since its principal values lie between $\pi/2$ and $-\pi/2$. This should make the calculations independent of any predefined ARCTAN subroutine. A few quantities which appear regularly throughout the program are defined here. Tests are performed to determine which method of calculation will be used according to the value of rho.

Calculation of ordinary and modified Bessel functions of the first kind for small values of the argument involve the evaluation of an infinite series and are accurate when $|z|$ is small or the other is very large compared to $|z|$.

$$J_n(z) = \frac{\left(\frac{z}{2}\right)^n}{n!}\left[1 - \frac{\left(\frac{z}{2}\right)^2}{1!(n+1)} + \frac{\left(\frac{z}{2}\right)^4}{2!(n+1)(n+2)} - \frac{\left(\frac{z}{2}\right)^6}{3!(n+1)(n+2)(n+3)} + \cdots\right]$$

$$= \frac{\left(\frac{z}{2}\right)^n}{n!}\sum_{k=o}^{\infty} AA_k \left(\frac{z}{2}\right)^{2k} \quad \text{where } AA_k = (-1)^k \frac{n!}{k!(n+k)!}$$

$$I_n(z) = \frac{\left(\frac{z}{2}\right)^n}{n!}\left[1 + \frac{\left(\frac{z}{2}\right)^2}{1!(n+1)} + \frac{\left(\frac{z}{2}\right)^4}{2!(n+1)(n+2)} + \frac{\left(\frac{z}{2}\right)^6}{3!(n+1)(n+2)(n+3)} + \cdots\right]$$

$$= \frac{\left(\frac{z}{2}\right)^n}{n!}\sum_{k=o}^{\infty} BB_k \left(\frac{z}{2}\right)^{2k} \quad \text{where } BB_k = \frac{n!}{k!(n+k)!}$$

Functions of the second kind are calculated by the Weber-Schlafli formulas which contain a logarithmic term, an infinite series term, and a finite series term. In both cases, the terms of the series are summed from the smallest to the largest to avoid round-off error. The procedure used is standard.

$$Y_n(z) = \frac{2}{\pi}\left[J_n(z)\left\{\gamma - \ln 2 + \ln z\right\} - \frac{1}{2}\sum_{n=u}^{n-1}\frac{(n-r-1)!}{r!}\left(\frac{z}{2}\right)^{2r-n}\right.$$

$$\left. - \frac{1}{2}\sum_{r=o}^{\infty}\frac{(-1)^r}{r!(n+r)!}\left(\frac{z}{2}\right)^{n+2r}\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{r} + 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n+r}\right)\right]$$

$$K_n(z) = (-1)^{n+1} I_n(z)\left\{\gamma - \ln 2 + \ln z\right\} + \frac{1}{2}\sum_{r=o}^{n-1}\frac{(-1)^r(n-r-1)!}{r!}\left(\frac{z}{2}\right)^{2r-n}$$

$$+ \frac{(-1)^n}{2}\sum_{r=o}^{\infty}\frac{1}{r!(n+r)!}\left(\frac{z}{2}\right)^{n+zr}\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{r} + 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n+r}\right)$$

Ordinary and modified Bessel functions of the first kind for moderate values of the argument are calculated using downward recurrence formulas:

$$J_{n-1}(z) = \frac{2n}{z} J_n(z) - J_{n+1}(z)$$

$$I_{n-1}(z) = \frac{2n}{z} I_n(z) + I_{n+1}(z)$$

To start the recurrence, a sufficiently large order is chosen so that the series computation for the initial high order J or I function will be accurate. Recall that the infinite series is accurate if the order is much larger than the argument. Miller has shown that cancellation does not occur when stable recurrence relations are used.

An analagous procedure cannot be used for functions of the second kind because Miller has shown the recurrence relations are not stable: the difference of two nearly like numbers occurs in the calculations of functions of lower order.

The use of the Gauss continued fraction to calculate modified Bessel functions of the second kind was Mr. Elder's contribution to this work. The equation

$$K_n(z) = \left(\frac{\pi}{2z}\right)^{1/2} \frac{e^{-z}}{\Gamma(n+1/2)} \int_0^\infty e^{-u} u^{n-1/2} \left(1+\frac{u}{2z}\right)^{n-1/2} du$$

is related to the hypergeometric equation

$$f(a, b; v) = \frac{1}{\Gamma(a)} \int_0^\infty \frac{e^{-u} u^{a-1}}{(1+vu)^b} du$$

discussed by Wall and others. It can be seen immediately that the modified K function can be expressed in terms of the equation given by Wall.

$$K_n(z) = \left(\frac{\pi}{2}\right)^{1/2} e^{-z} f(a, b; v)/\sqrt{z}$$

$$K_{n-1}(z) = \left(\frac{\pi}{2}\right)^{1/2} e^{-z} f(a-1, b+1; v)/\sqrt{z}$$

Now if a quotient function Q_n is defined in terms of K_n and K_{n-1}, then the exponential and constant factors will be eliminated, yielding a ratio of two hypergeometric functions:

$$Q_n = \frac{K_{n-1}(z)}{K_n(z)} = \frac{f(a-1,\ b+1;\ v)}{f(a,\ b;\ v)}$$

After some algebra, the quotient function can be reduced to

$$Q_n = F_1(a,\ b;\ v)\ \left[1 + v(b+1)\ G_1(a,\ b;\ v)\right]$$

The functions F_1 and G_1 are forms of the Gauss continued fraction

$$\frac{f(c,\ d;\ v)}{f(c,\ d-1;\ v)} = \cfrac{1}{1 + \cfrac{cv}{1 + \cfrac{dv}{1 + \cfrac{(c+1)v}{1 + \cfrac{(d+1)v}{1 + \cfrac{(c+2)v}{1 + \ldots}}}}}}$$

Since it is of the form 1 + , we need not worry about cancellation. The computation of these fractions is accomplished by an iterative procedure.

Calculations of the modified function of the second kind are made by substituting the quotient function in the Wronskian

$$I_n(z)K_{n-1}(z) + I_{n-1}(z)K_n(z) = \frac{1}{z}$$

$$K_{n-1}(z) = K_n(z)Q_n(z)$$

$$K_n(z) = \frac{1}{\left[z\ I_{n-1}(z) + I_n(z)Q_n(z)\right]}$$

Analytic continuation must be used to calculate the correct functional value for values of z lying in the left half-plane.

Ordinary Bessel functions of the second kind are calculated in terms of Hankel functions which are linear combinations of ordinary Bessel functions:

$$H_n^{(1)}(z) = J_n(z) + iY_n(z)$$

$$H_n^{(2)}(z) = J_n(z) - iYn(z).$$

The J's have already been calculated by recurrence; a method for evaluating the H functions must be developed. Looking at the equations for the Hankel functions,

$$H_n^{(1)}(z) = \left(\frac{2}{\pi z}\right)^{1/2} \frac{e^{i(z - n\pi/2 - \pi/4)}}{\Gamma(n + 1/2)} \int_0^\infty e^{-u}\, u^{n - 1/2} \left(1 + \frac{iu}{2z}\right)^{n - 1/2} du$$

$$= \left(\frac{2}{\pi z}\right)^{1/2} e^{i(z - n\pi/2 - \pi/4)}\, f(a, b; v)$$

$$= \frac{-2i}{\pi} e^{-n\pi i/2} K_n(-iz)$$

$$H_n^{(2)}(z) = \left(\frac{2}{\pi z}\right)^{1/2} \frac{e^{-i(z - n\pi/2 - \pi/4)}}{\Gamma(n + 1/2)} \int_0^\infty e^{-u}\, u^{n - 1/2} \left(1 - \frac{iu}{2z}\right)^{n - 1/2} du$$

it can be seen that they can be rewritten in terms of the modified functions just calculated. Substituting similar quotient functions into Wronskians gives the Hankel functions in terms of the ordinary functions of the first kind and the quotient function:

$$H_n^{(1)}(z) = -\frac{4}{\pi} \frac{i}{2z} \left[\frac{1}{J_{n-i}(z) - iJ_n(z)Q_n(iz)}\right]$$

$$H_n^{(2)}(z) = \frac{4}{\pi} \frac{i}{2z} \left[\frac{1}{J_{n-1}(z) + iJ_n(z)Q_n(iz)}\right] .$$

Using the values now available for the Hankel functions makes the computation of ordinary Bessel functions of the second kind a simple matter.

$$Y_n(z) = i \left[J_n(z) - H_n^{(1)}(z)\right] \quad \text{for Im } z < 0$$

$$Y_n(z) = -i \left[J_n(z) - H_n^{(2)}(z)\right] \quad \text{for Im } z \leq 0$$

Bessel functions for large values of z are evaluated using Hankel asymptotic formulas. If the given argument does not lie in the right half-plane, it is rotated ± 180 degrees to either quadrant I or IV so that it lies within the range of all the formulas used.

$$J_n(z) = \frac{1}{2}\left[H_n^{(1)}(z) + H_n^{(2)}(z)\right]$$

$$Y_n(z) = -\frac{i}{2}\left[H_n^{(1)}(z) - H_n^{(2)}(z)\right]$$

$$H_n^{(1)}(z) = \frac{e^{-\rho \sin\theta}}{\sqrt{\frac{\pi\rho}{2}}} e^{i(\rho\cos\theta - \frac{\theta}{2} - \frac{m\pi}{2} - \frac{\pi}{4})}(P_n + iQ_n)$$

$$H_n^{(2)}(z) = \frac{e\ \rho \sin\theta}{\sqrt{\frac{\pi\rho}{2}}} e^{-i(\rho\cos\theta + \frac{\theta}{2} - \frac{m\pi}{2} - \frac{\pi}{4})}(P_n - iQ_n)$$

$$P_n = 1 - \frac{(4n^2 - 1^2)(4n^2 - 3^2)}{2!(8z)^2} + \frac{(4n^2 - 1^2)(4n^2 - 3^2)(4n^2 - 5^2)(4n^2 - 7^2)}{4!(8z)^4} - \ldots$$

$$Q_n = \frac{4n^2 - 1^2}{1!(8z)} - \frac{(4n^2 - 1^2)(4n^2 - 3^2)(4n^2 - 5^2)}{3!(8z)^3} + \ldots$$

The evaluation of the series expansions P and Q in the asymptotic formulas is straightforward. Direct substitution of the Hankel values yields the desired result for the ordinary Bessel functions.

The modified Bessel functions are calculated from equations

$$I_n(z) = \frac{e^z}{\sqrt{2\pi z}} \sum_{k=o}^{\infty} \frac{(-1)^k (n,k)}{(2z)^k} + \frac{e^{-z}}{\sqrt{2\pi z}} e^{(n + 1/2)\pi i} \sum_{k=o}^{\infty} \frac{(n,k)}{(2z)^k}$$

$$= \frac{e^z}{\sqrt{2\pi z}}(P_n + Q_n) + \frac{e^{-z}}{\sqrt{2\pi z}}(P - Q_n)\ ,\ -\frac{\pi}{2} < \arg\ z < \frac{3\pi}{2}$$

$$K_n(z) = \sqrt{\frac{\pi}{2z}}\, e^{-z} \sum_{k=o}^{\infty} \frac{(m,k)}{(2z)^k} = \sqrt{\frac{\pi}{2z}}\, e^{-z}\ (P_n - Q_n)$$

involving the same series expansions used in the ordinary function calculations. The methods are standard and certainly do not need detailed explanation.

Analytic continuation formulas are used to obtain correct functional values for arguments that were rotated to the right half-plane in order to use these formulas.

5. FUTURE WORK AND CHECKING METHODS. There are regions of overlap between the methods of calculation. Taking advantage of this allowed some checking of results between the methods. Double precision runs on the UNIVAC were used to check for round-off error on BRLESC. Results agree to at least 13 significant figures on the CDC. UNIVAC, and BRLESC. We are proceeding on a time available basis to check using other methods of calculation.

We are preparing a CDC double precision version of the subroutine. The region in which the Gauss continued fraction is used will be extended past 21.0 since the Hankel asymptotic formulas will not give 20 significant figures at 21.0. It is expected that similar adjustments will have to be made in other constants.

We would like to extend the program to include real orders; this will be done if any interest is expressed. Eventually the work will include Kummer and Whittaker functions of complex argument.

A more detailed description of the coding and formulas used is contained in the ARRADCOM Special Publication "User's Manual for the BRL Subroutine to Calculate Bessel Functions of Integral Order and Complex Argument." The report is expected to be published this Spring and will be available through the Defense Documentation Center.

A SEMANTIC UPDATING SYSTEM FOR REPAIRING SOFTWARE

Edward F. Miller, Jr., and John S. Praninskas
Software Research Associates, San Francisco, CA 94126
and
Morton A. Hirschberg
U. S. Army Ballistic Research Laboratory, Aberdeen, MD 21005

ABSTRACT

Recent estimates show that 50% or more of all software costs accrue after the software has been written. It is, therefore, important to be able to repair software easily and efficiently.

This paper details the interior command structure to perform a semantic update for FORTRAN software. Semantic updating is oriented towards systems of programs which are interrelated. The effects of a semantic update are felt through an entire module (sub-program) and often throughout an entire system (program). Semantic updating operates on an entire statement (command) rather than a line (card image) treated by classical (existing, traditional) updating systems. A semantic updating system tends towards a program understanding system (although it is not in the class of a full-blown artificial intelligence program), as opposed to text editors which tend toward generalized word processors which are applicable to any file. The current design covers seventy-two commands. It can deal with side effects. No exterior command system has been designed; however, an implementation is being considered for a Control Data Corporation Cyber System. It is estimated that once implemented, a semantic updating system would realize an order of magnitude or greater savings over update systems currently in use.

A SEMANTIC UPDATING SYSTEM

FOR REPAIRING SOFTWARE

1 INTRODUCTION

In a 1973 article[1] Boehm showed that roughly 70 percent of all computer costs were software related and predicted software costs would continue to rise to roughly 90 percent of all computer costs by 1985 (Figure 1). To date there has been no evidence presented to contradict Boehm's estimates. Moreover, recent articles have estimated software maintenance costs to be between 40 and 60 percent of all software costs.[2,3,4] Mills predicted that "unless radical new methods are found, maintenance will go even higher in its demands and will very nearly stifle further development" (Ref. 3, page 80).

Structured programming,[5] now in its second decade, has not been the panacea needed to significantly reduce maintenance costs. In fact, Kraft claims structuring techniques have so routinized programming that we have developed a work force of white-collar clerks who in general are not as skilled as programmers were before structuring.[6] Similarly, top-down programming may be very good for design but in practice is not used in coding by sophisticated programmers.[7]

---

[1] Boehm, B.W., "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp. 48-59.

[2] Boehm, B.W., "The High Cost of Software," In...Practical Strategies for Developing Large Software, Addison Wesley, 1975 (12 pages).

[3] Mills, H.D., "Software Development," Procedings 2nd International Conference on Software Engineering Supplement, October 1976, pp. 79-86.

[4] Putnam, L.H. and Wolverton, R.W.,"Quantitative Management: Software Cost Estimating," IEEE Computer Society, Publication EHO-129-7, November 1977.

[5] Yourdon,E. Techniques of Program Structure and Design, Prentice-Hall, 1975.

[6] Kraft, P. Programmers and Managers, Heidelberg Science Library, New York, 1977.

[7] Parnas, D.L., Personal Communication, ARO Workshop, February, 1978.

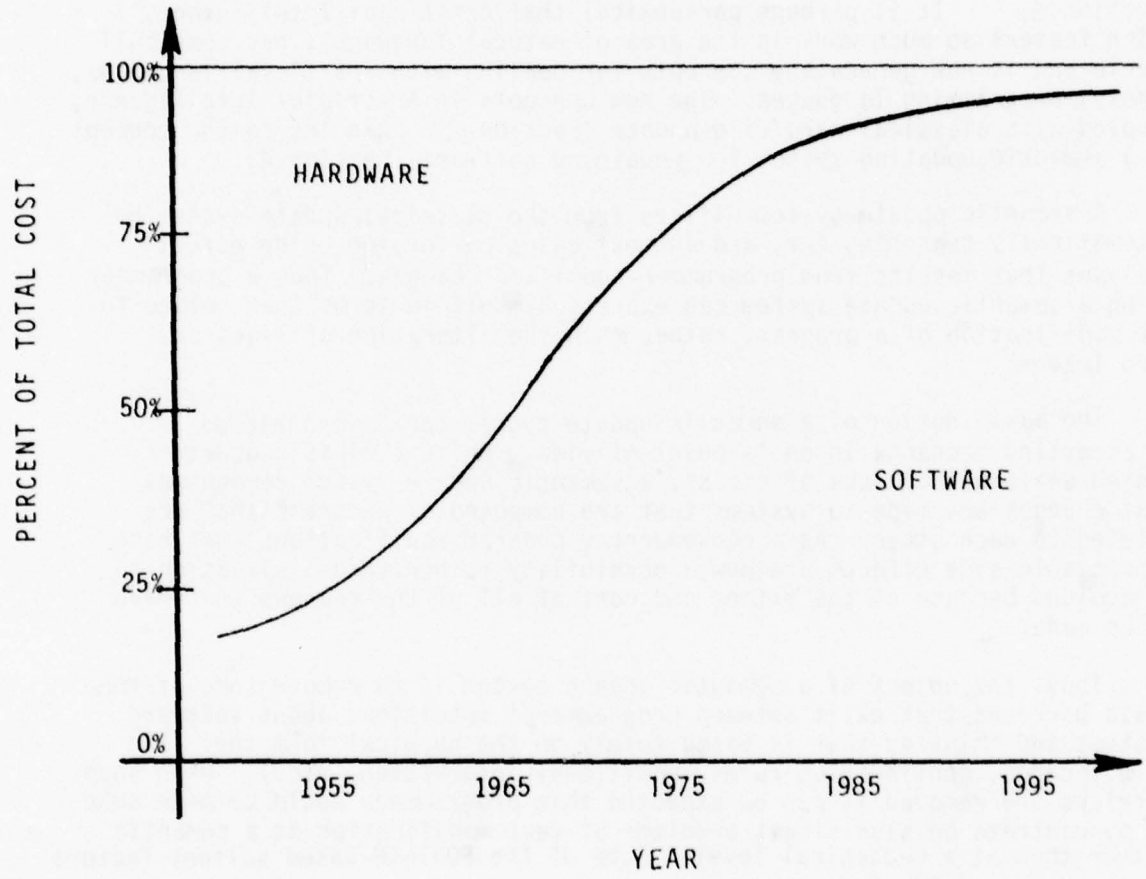


Figure 1: Hardware/Software Cost Trends

There are, however, new ideas now emerging in Artificial Intelligence (AI) which can significantly reduce software costs and software maintenance (Section 3). It is perhaps paradoxical that Artificial Intelligence, which fosters so much work in the area of natural languages, has come full circle and is now generating concepts for dealing with artificial languages, namely, programming languages. The new concepts in Artificial Intelligence, coupled with classical card/file update (Section 2). has led to the concept of a semantic updating system for repairing software (Section 4).

A semantic update system differs from the classical update system by automatically searching for, and in most cases performing, side-effect analyses that results from programmer-specified changes. Thus a programmer using a semantic update system can express himself in terms that relate to the modification of a program, rather than the alteration of lines or card images.

The basic notion of a semantic update system can be understood by accepting a change in one's point of view. While a classic update system deals with "decks of cards", a semantic update system recognizes that changes are made to systems that are composed of programs that are related to each other. As a consequence, program modifications that have appreciable side effects are now a possibility rather than a situation to be avoided because of the extent and cost of all of the changes that have to be made.

Thus, the object of a semantic update system is to remove some of the basic barriers that exist between programmers' intuitions about software systems and thinking that is based solely on the physical form they take (i.e., cards, continuation rules, positional information, etc.). When such barriers are removed it can be expected that programmers would be more able to concentrate on significant problems of text modification at a semantic rather than at a mechanical level. Some of the FORTRAN-based salient factors of semantic updates are:

- It allows the user to think in terms of modifications to programs, rather than in terms of changes to "lines" or card-images.
- It is oriented towards large software systems with many interrelated modules.
- Its fundamental units are FORTRAN statements, not card-images.
- It applies to programs written in ANSI FORTRAN although later versions may extend to other languages.

- It employs a static analysis of the program; i.e., it avoids executing the program.
- It applies to an entire program or to selected parts of a program.
- Trial updates can be performed with side-effects reported to the user.
- Programmer knowledge about the software is supported by a series of reports thus providing automated documentation.

This paper describes the characterization of the semantic update and its expected gain as a maintenance tool. More than that, a semantic updating system is a realizable effort which can be completed in the near term, providing insights for several of the Artificial Intelligence proposals which are now paper studies and likely to be a decade away.

2 CLASSICAL UPDATING SYSTEMS

There are two large classes of maintenance problems which have been handled with classical card/file updating systems:

a. Improved designs

b. Error correction

In the case of improved design, an operational program is modified to account for new algorithms, different parameterizations, or implementation on a new machine. On the other hand, errors may have been produced because of design (logic and algorithm errors), or coding (syntax and semantic errors).

The tool which has been used for both improved design and error correction for nearly twenty years is the classical card/file system (Figure 2).

The history of update systems (here meant in a general way to include any system used to assist in management of a large source program) is rooted in the very earliest generalized batch processing operating systems. In that enviornment it was first possible to maintain large software systems on magnetic tape as card images rather than explicitly as physical card records. In many cases a "hard copy" of the tape was kept as back-up.

Vestiges of the earliest use of update systems could still be seen in the late 1970's when interest in systematizing source code control processes with configuration control procedures first became of wide interest. Needed configuration control usually was achieved by mixing manual record keeping procedures with controlled use of a basic update system.

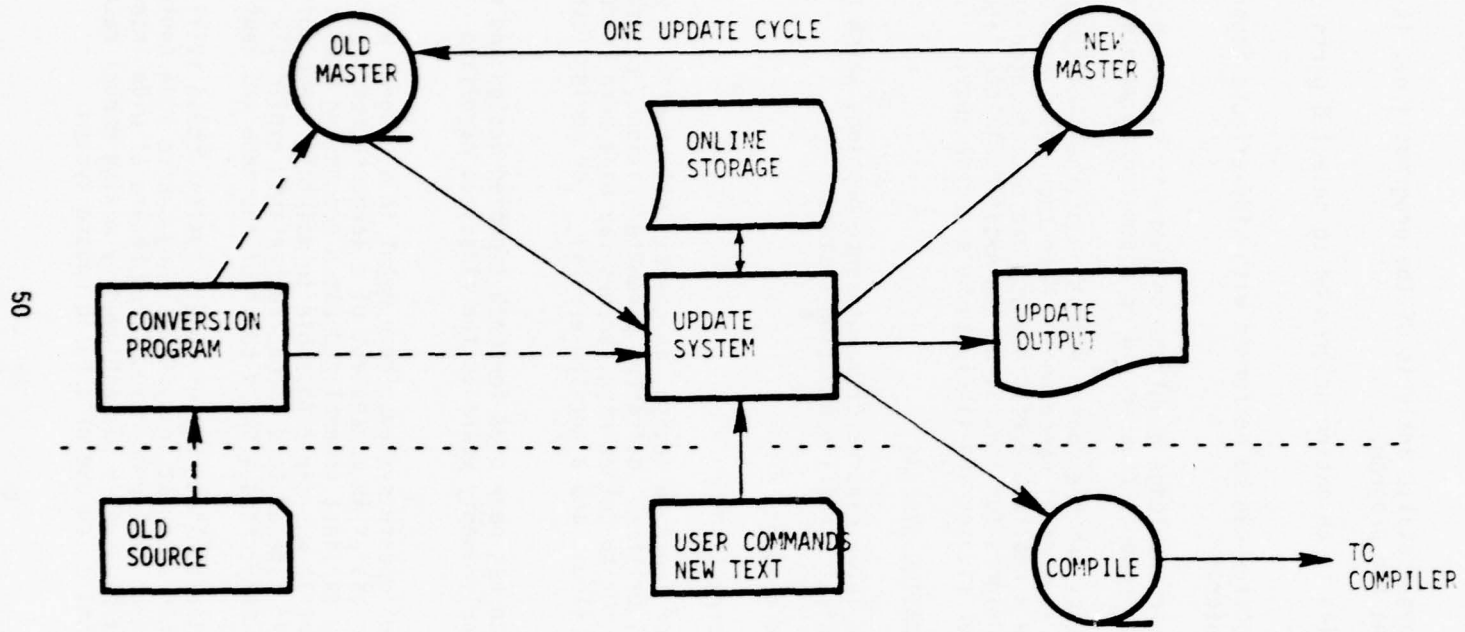


Figure 2: Basic Structure of a Classical Card/File Update System

The classic update system characterized by a number of relatively standard features:

1. Systems were designed to operate in a batch environment and were specifically intended to process card images (i.e., either source programs or source data card images).

2. Systems processed only an elementary "include facility" (e.g., the *COMPKG/*CALL facility of CDC Update[8]) to be used to incorporate defined text.

In the earliest operating environments object code was produced fresh each time the source program was compiled. Later, tools specifically designed to minimize the overhead of recompilation were developed so that it was no longer necessary to provide a complete copy of the updated software system source text each time a change was made.

Standard operating modes for a classical batch-oriented update system are described first in general terms and then in terms of generally accepted ground rules of operation. The ground rules take the form of a series of assumptions about the way the update system's capabilities are employed in managing the source form of the program set (or software system), and also relating to the normal way in which object-code versions of the program text is handled.

2.1 Classical Update Facility

Figure 2 illustrates a conventional batch oriented update system. There is an OLD MASTER, which stores the previous version of the software system. In operation, a user provides update commands intermixed with new source text statements to produce: (1) a COMPILE file that is used in the current run, and (2) an optional NEW MASTER that incorporates all of the changes made to the system in a way that would ultimately permit backing up if that were found necessary.

The update system consumes some online working storage as it processes the set of user commands and associated program text against the OLD MASTER file. In addition to compilable code there is update system output that reports on the actions taken and identifies any exceptions (warnings or errors) that were found during the update process.

Periodically the OLD MASTER is replaced by the NEW MASTER; this action constitutes one update cycle. A software system may proceed through many update cycles between the start of the development activity and release of the final version of the program. Normally the most recent OLD MASTER

---

8 Control Data Corporation, Update Manual, Form 84000016, March 1976.

contains the backup master copy of the software system.

In certain situations when it is necessary to create an OLD MASTER from a source program in some other format a special conversion program is used that converts a program from the COMPILE file format to whatever is needed for the update system to operate.

2.2 Operating Ground Rules

It is important to characterize the typical operating environment that surrounds an update system. What follows is a set of statements that represent good procedures and good judgment in use of a classical update system.

(1) It is usually assumed that the NEW MASTER replaces the OLD MASTER only infrequently, namely whenever the set of changes made to the source program on the OLD MASTER becomes large enough to justify the cycle.

(2) There must be no capacity limits on the OLD MASTER/NEW MASTER file formats so that the update system can be used for large applications.

(3) Usually, the number of changes is a small percentage (typically around 5%) of the total content of the OLD MASTER.

(4) There must be an option in the update system that generates a full set of newly updated programs for the COMPILE file. In normal operation, however, only the modified programs are issued to the COMPILE file for compilation and selective replacement of previously compiled object programs.

(5) A change to a program that does not pass muster when compiled (from the COMPILE file) ordinarily does not cause loss of the run since the compiler typically issues no object code and the load process continues forward without the missing module (using, typically, the previous version). Ultimately, of course, all changes specified by the user must make sense to the compiler.

(6) The update system must not fail to produce something regardless of the sense of the user input (or even the lack of an appropriate OLD MASTER). Otherwise, there is no place to start.

(7) Changes to any module maintained in the OLD MASTER are either:
(1) small modifications to individual statement/lines, or
(2) a complete replacement/deletion of a module or modules.

(8) Most update systems have incompatibility problems that would be solved if the basic format for updating could be generated automatically from a full COMPILE file version of the program (the conversion program box). Most classical update systems are deficient in this regard.

2.3 Object Library

In addition to maintaining an accurate copy of the source text of a software system in many situations it is quite desirable to keep a related copy of the relocatable object text -- one that corresponds to the source. This need is usually based on the realization that it may be too expensive to regenerate all of the object code for a software system each time a change is made. Instead, only those elements of the software system that are changed are actually recompiled.

Figure 3 shows the way this is accomplished, using a library edit program. The object texts of each recompiled module are merged with the OLD LIBRARY to produce a NEW LIBRARY. The merge takes place with the rule that an object module on the OBJECT FILE is always added to the NEW LIBRARY; if there is a previous copy on the OLD LIBRARY that copy is ignored. The result is a NEW LIBRARY that contains all of the most-recently changed object code.

In normal operation the combination of the contents of the OLD MASTER and the OLD LIBRARY represent the best source of an executable software system. The Update system and the Library Edit system are used in concert to make modifications as necessary. In some environments it is possible to save the actual bound object text (called the absolute binary file, or the executable object file) in addition; this would be done for a frequently used system to avoid the expense of repeated link/edit/load activities.

2.4 Archiving Procedures

A final topic necessary to understanding the operating environment for update systems and their allied tools centers on the procedures used to assure a secure backup in case any of the previously described procedures fail. For example, coherent knowledge about a software system would be irretrievably lost if the OLD MASTER and NEW MASTER were assigned as the same file (this would require write-access to the file since the NEW MASTER is generated by the update system) and if a fatal error occurred during the update activity.

The situation is similar, but not nearly so dangerous, during the library update step. At least in this case if there are errors the OLD MASTER will contain a fairly recent program text that can be used to

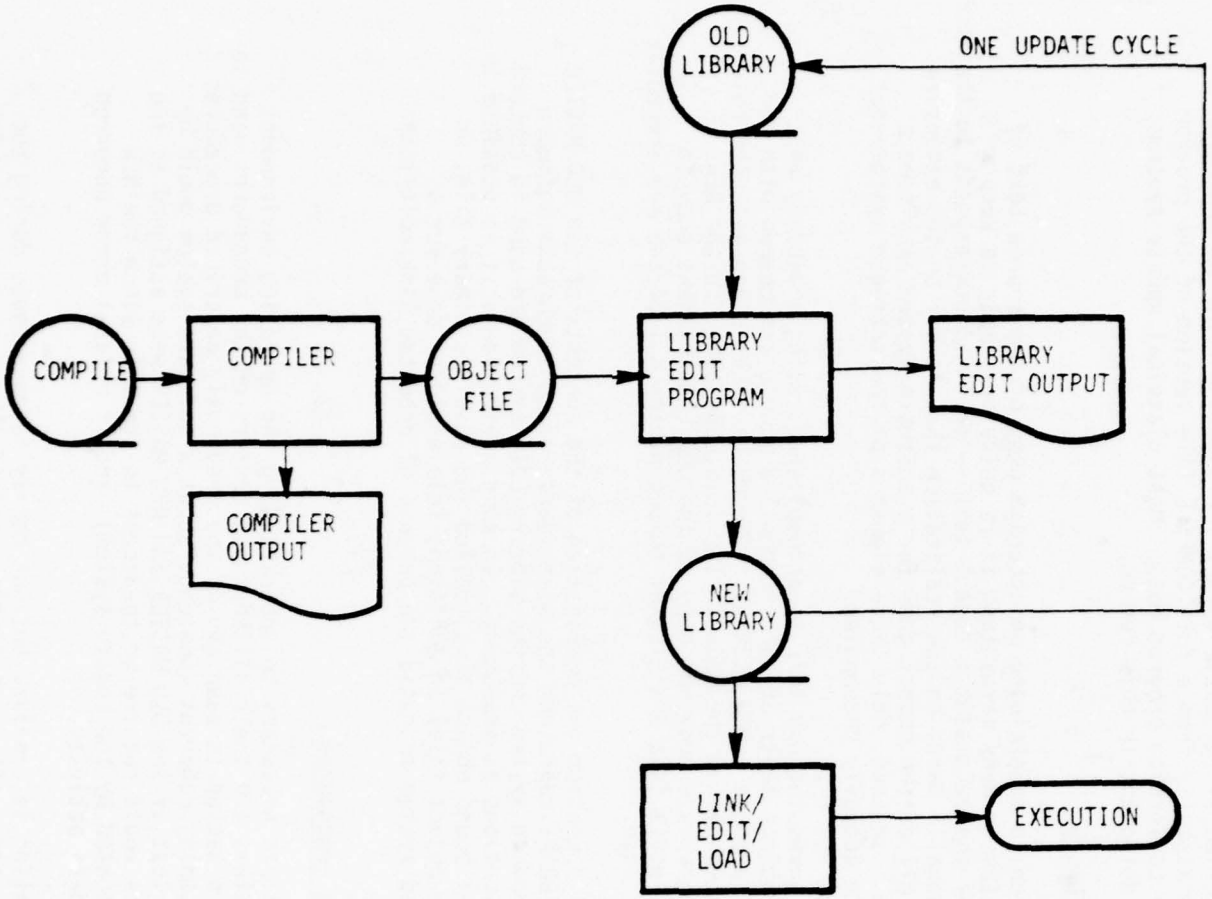


Figure 3: Maintenance of an Object Library File

generate a complete set of object modules; in turn, they can be used to regenerate the OLD LIBRARY.

Typical operating environments minimize any impact of a failure in one or more of the tools by (a) assuring there is a backup copy of the OLD MASTER and/or the OLD LIBRARY stored in a safe place, and (b) keeping the complete sequency of update operations on file. This typically requires keeping a copy of each of the OLD MASTER files in the archive; normally only the most recently generated OLD LIBRARY is archived. This assures that a catastrophic failure will "cost" only whatever effort is involved in moving from the current OLD MASTER to the current NEW MASTER.

3 ARTIFICIAL INTELLIGENCE AND PROGRAM MAINTENANCE

Among the many activities in Artificial Intelligence (AI) research has been the design of software which is error-resistant and can repair software[9] and the development of systems which exhibit understanding about computer programs.

In a program understanding system, deduction is performed by the system in order to provide a knowledge base from which questions asked by the user can be answered. This situation is analogous to a theorem-proving system in the following way: in a theorem-proving system the axioms represent the knowledge required for theorem-proving.

The work of Waters[10] is the development of an automatic program understanding system oriented to the processing of FORTRAN programs. Waters investigates the structure and function of a system that "understands" individual programs by operating within a plan for them. The plans are developed by analysis of the source code of the program, plus assertions about the program.

The proposed system (which at this writing is not known to have been implemented) is intended to answer users' questions about candidate programs; some of the questions that could be answered take the following form:

- "What" questions, which relate to the problem-independent role individual statements, segments and/or modules have in a large software system.

---

9 Yau, S.S., Cheung, R.C., and Cochrane, D.C., "An Approach to Error-Resistant Software Design," Procedings 2nd International Conference on Software Engineering, IEEE Catalog No. 76CH1125-4C, pp. 429-436.

10 Waters, R.C., "A System for Understanding Mathematical FORTRAN Programs," MIT, Artificial Intelligence Laboratory, AIM-368, August 1976.

- "How" questions that deal with explanations of the internal workings of individual pieces of a program.

- "Why" questions which require answers stating the reasons program particles were included in the program text.

Waters' arguments for this kind of question-answering system are lengthy and persuasive. The early investigations of this kind of system are important because they will potentially lead in the direction of identifying useful program processing functions that can be implemented in practical systems, as well as examining in detail the underlying notions of generalized program analyzers.

A more general thrust of AI research is exemplified by the work of Bobrow and Winograd[11]. They examine the current state of program understanding systems (such systems are not necessarily limited to treating computer programs). Ultimately, such systems will provide for understanding of natural language; they are consequently critically important as the prototype for man/machine interface systems.

A typical knowledge representation language treats understanding in terms of sub-domains such as:

Task domains that describe the nature of activities a system being described is supposed to perform.

Linguistic domains, i.e., syntactic and semantic analyses of descriptive statements.

Common sense domains, which include primitive statements (like axioms) that would be universal relative to some systems.

Strategy domains, which include descriptions of primitive operating procedures for intermingling of task/linguistic/common-sense domain constructs.

Although the AI-related projects just described are quite abstract, they provide encouragement that in the long term there will be fairly sophisticated methods for handling knowledge about programs in a useful way. Second, they suggest the possibility that limited-capability systems which can be brought into reality now would have a very significant effect on the ease of managing complex software systems.

---

[11] Bobrow, D.G., and Winograd, T., "An Overview of KRL, A Knowledge Representation Language," Stanford University, Artificial Intelligence Laboratory, AIM-293, November 1976.

The key to making this work, of course, is adopting the appropriate context of knowledge. In the remainder of this report, the knowledge base will generally be taken as meaning the body of information about a FORTRAN software system that is large and complex. This knowledge base is thus founded in these two facets of a FORTRAN software system:

(1) The semantic content of actual FORTRAN programs arranged into a software system.

(2) The auxiliary information about the software system design that is probably not included in the actual semantics (but which may be included in the documentation).

As subsequent sections will demonstrate, the relationship between fairly naive (but certainly non-trivial) forms of semantic update capability and more-general knowledge-based processing of source programs is a very close one.

Figure 4 illustrates a final point about the role of a semantic update system. The figure shows two major routes of a growth for update systems: in the direction of sophisticated editors, and in the direction of program understanding systems. The ultimate evolution for editor-based systems is a fully generalized word processor.

4 THE SEMANTIC UPDATING SYSTEM

Figure 5 characterizes the primary differences between a classical update system, a semantic update system, and a general program understanding system. The salient features will be described below.

The atomic element for a semantic update system is a whole statement, or a command to the system (expressed as a whole statement also). By comparison, the input to the classical update system is a single card-image, or line.

A second factor concerns the extent to which it is possible in a semantic update system for individual commands to affect other parts of the text being processed. For example, in a classical update system a command can refer only to an individual line, but in a semantic update system a command could affect the entire software system.

A third area involves the facilities provided by the semantic update system to controlling the structure of a large software system. Classical update systems provide almost no support (the exception is the define/include facility for blocks of card images). A semantic update system would make it possible to define major elements of a software system and base subsequent commands on such definitions.

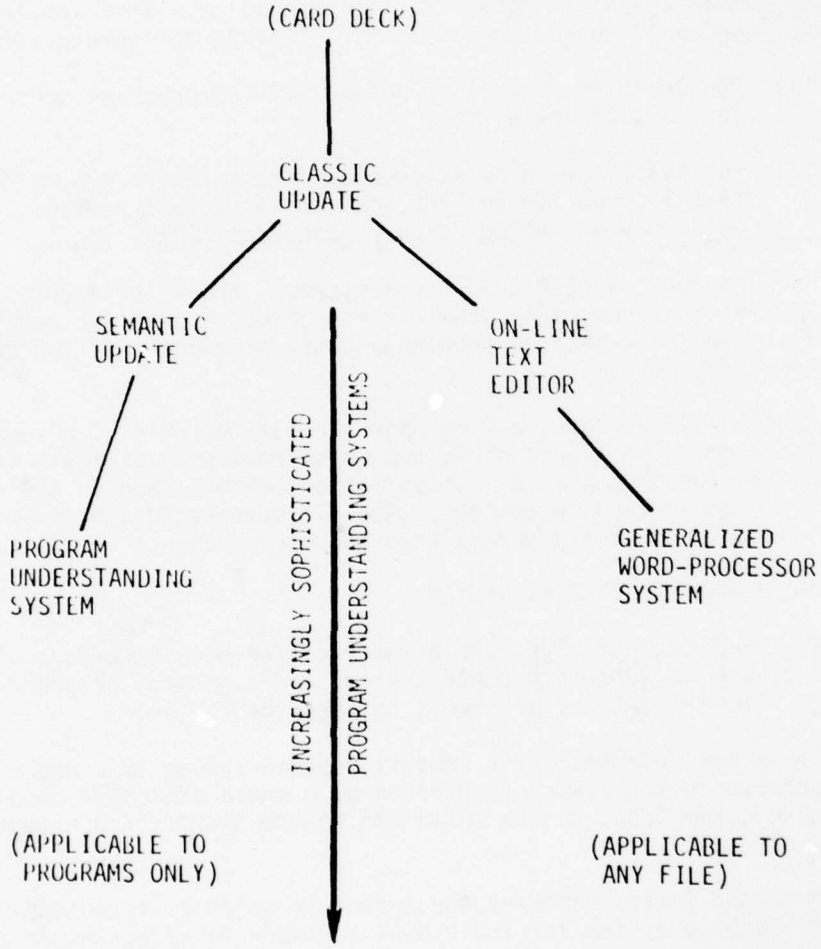


Figure 4: Classes of Source Text Processors

Increasing Sophistication →

FEATURE	CONVENTIONAL UPDATE	SEMANTIC UPDATE	GENERAL PROGRAM UNDERSTANDING SYSTEM
ATOMIC ELEMENT	LINE (Card Image)	STATEMENT COMMAND	STATEMENT COMMAND REQUEST
USER COMMAND DESCRIBES	REPLACED LINE NEW LINE INTERNAL COMMAND	REPLACE FUNCTION NEW FUNCTION	REPLACE FUNCTION NEW FUNCTION
EXTENT OF EFFECT OF COMMAND	SOMETIMES PART OF A MODULE	MODULE SYSTEM	VARIES
PROGRAM ORGANIZATION ASSISTANCE	DEFINE/INCLUDE ONLY	EXTENSIVE AND EXTENDABLE	FULLY AUTOMATIC

Figure 5: Comparison of Program Manipulations

4.1 Information Flow

It is now possible to discuss the general structure of a semantic update system, in terms of the information flow its use would represent for a user. Figure 6 shows the basic information flow relationship for the conceptual system; a user operates in a context defined by the following forms of information:

- System Status Reports, which tell the user about the current state of the software system. These reports provide fairly general and statistical data about the system such as the number of modules, the total number of statements, the number of defined entities.

- System Structure Reports, which provide information about the imposed and natural structures within the software system.

- Definitions reports, which provide current copies of all defined entities.

- Annotated listings of the programs which are currently being modified, or are otherwise of special interest. This set of programs could include all of the modules known to the update system.

In addition to the reports provided by the semantic update system, the user also has access to the normal compiler output. (It may not be necessary to use all of the compiler output when using semantic updating.)

Thus, a user's role is to examine this information and, combined with his knowledge of the intended form of the software system, construct (or modify) commands that bring the current system "closer" to that desired. It should be clear that the success of a semantic update system is based wholly on its ability to fully satisfy the information needs of the user.

4.2 Ground Rules

Clearly the notion of semantic update is machine and programming language dependent; it is possible that semantic update systems could be built that are application dependent also. It is important at the outset to identify some ground rules that will keep the thinking specific enough to permit considering real systems. Some of these are:

- We will assume a FORTRAN environment throughout. This means that the "semantics" of the underlying programming language will be those of the FORTRAN programming language. It is to

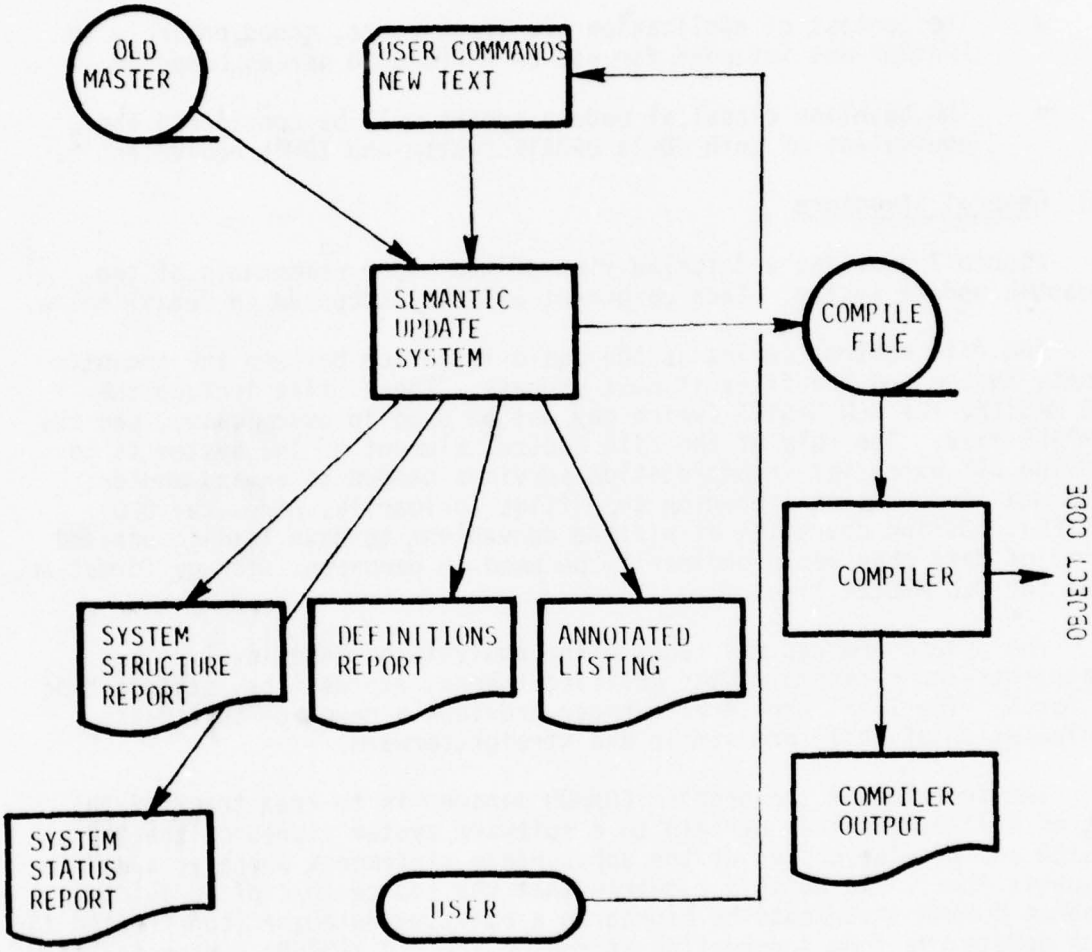


Figure 6: Information Flow in a Semantic Update System

be expected that at least some of the conceptualizations will reflect the flavor of FORTRAN directly.

- The context of application are large codes, coded entirely in FORTRAN and intended for use on a CDC 6000 series computer.

- The baseline classical update system will be considered the equivalent of both CDC's UPDATE system and IBM's equivalent[12].

4.3 General Structure

Figure 7 provides a detailed view of the major components of the semantic update system. Each component will be discussed in detail below.

The file control serves as the basic interface between the semantic update system and the files it must process. These files include the OLD MASTER, the NEW MASTER (which may not be used in every case), and the COMPILE file. The role of the file control element of the system is to provide all essential interpretation services needed to expand and/or contract information stored on such files (primarily, here, the OLD MASTER). During operation it will be convenient to have less-compressed forms of data than would ordinarily be used in permanent storage format on, say, the OLD MASTER file.

The update process may require the analysis of individual statements and reassembly with modified tokens, expressions, strings, and so forth. The local work area manager provides a resource that makes manipulation of that form simple and straightforward.

The function of the program COMMON manager is to keep track of the set of definitions that pertain to a software system stored on the OLD MASTER and provide copies of the appropriate statements whenever a module requests them. To do this requires that the source text of individual program COMMON statements be stored on a run-time database (constructed for each update run from information stored on the OLD MASTER). Whenever a particular COMMON is required the text is copied into the appropriate locations.

The COMMON manager also analyzes new COMMON statements for consistency with existing definitions. This is done in concert with the FORTRAN statement recognizer (see below).

As previously mentioned, the semantic update system has a conditional assembly feature that permits a user to parameterize the inclusion (or exclusion) of individual statements, blocks of statements, or even whole

---

[12] IBM Corporation, IBM OS/VS Utilities, IEBUPDTE User's Manual, Form GC35-0005-4, November 1975.

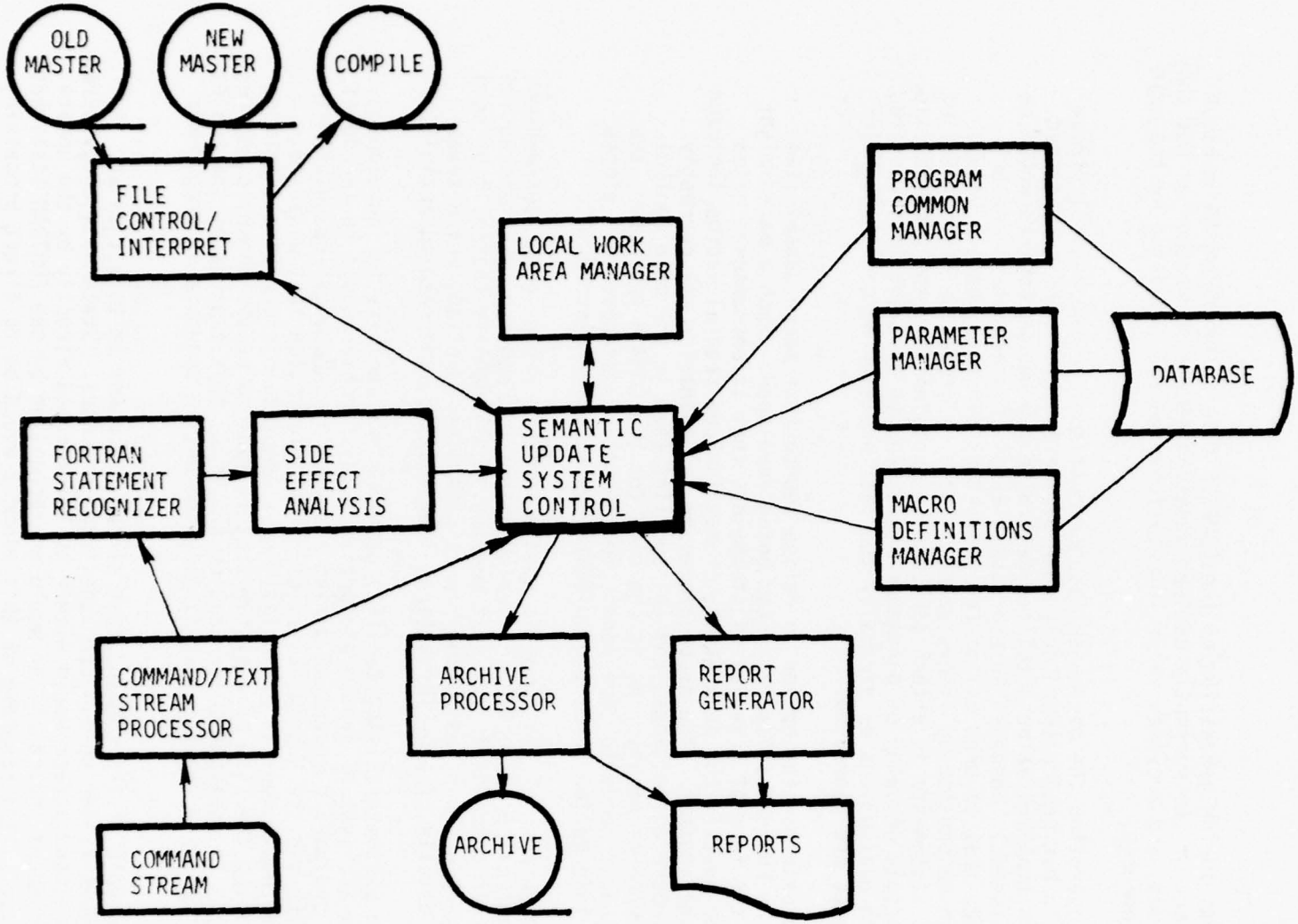


Figure 7: Overview of Semantic Update System Structure

subsystems during generation of the COMPILE file. The parameter manager keeps track of the currently defined update parameters, whether or not they have values, and provides value information whenever requested by the main control program.

In operation the parameter manager must monitor the control stream in detail, but need respond only when parameter values are encountered. (It is an implementation detail to determine how the parameters would be distinguished.) Whenever an expression involving parameters can be evaluated, that is when all of the parameters currently have a value, then such an expression is evaluated. Based on its value the controlled block of statements is either included or excluded from the COMPILE file. The principle of delay of parameter evaluation is assumed; this requires that such evaluations be virtually the last activity before the COMPILE file images are generated.

The user has the option to define source-text macro capabilities within the FORTRAN software system being developed. Such a macro might specify the form of a series of statements that is encountered often enough to justify the definition, or may include several macros included within one another. The macro processor (a) maintains all currently defined macro definitions, and (b) supplies them to the main control program when necessary. As is the case for the COMMON processor, the macro manager needs to have access to the online database that stores current information about the software system being altered.

The purpose of the semantic update system's report generator/manager is to centralize production of various kinds of reports. (See discussion of Class 10 commands in the next section.) All reports that will be seen by the user are handled by the report generator function, which takes care of details of page formatting, line editing, and related activities.

The system archiving facility provides a capability for developing detailed archives of system production. This archive would be in addition to that ordinarily generated during a series of update cycles between the OLD MASTER and the NEW MASTER files. The information that would be preserved would consist essentially of a complete history of all actions taken during the life of a system that is managed by the semantic update facility. In addition to generating the file, this element of the system also can produce reports that display the entire sequence of programmer actions.

The role of the command/text stream processor is to act as an intelligent filter on all user-specified commands. Statements encountered on the command stream would either be interpreted directly by the update system command processor, or would be recognized by the FORTRAN statement recognizer. The response of this element would be to direct processing to a state that corresponds to one of the forms that appears as a

command class; in other words, the exterior syntax scanning for semantic update commands, as well as control of the FORTRAN statement recognizer, is vested in the input processor.

Because the input stream may involve FORTRAN statements, and because certain other semantic update actions will require the recognition (but not a full parse), there must be a facility for type classification and lexical analysis of FORTRAN statements. This facility is closely allied in capability with the side-effect analyzer (discussed next). The input to the statement recognizer is a FORTRAN statement. The output would typically be (a) a partially "cracked" version of the statement in which the major elements of the text stream were decomposed at least in part, and (b) the value of the statement type. Note that some FORTRAN statements involve parts of more than one "type" so the "type" that is returned may have more than one component value.

The purpose of the side-effect analyzer is to determine the extent of side effects that a particular semantic update command would have under the assumptions that (1) the command is the only one being processed for the given module, and (2) there is sufficient information about the module to let the side-effect analyzer operate normally. The information that is required is a function of the complexity of the module: the more complex the module the more difficult the side-effect analysis is going to be.

In general, the side-effect analyzer will need to have access to rather complete structural and partial-semantic information about the module in question. This requires a level of analysis somewhat greater than that needed simply for program common analysis and/or parameter analysis. However, the information does not approach that needed by a compiler. It is important to note that the amount of information is also a function of the specific update command. For example, if the update command specified the alteration of a program label then only label information would be necessary to decide the extent of side effects. For other commands different information would be needed. It would seem reasonable to organize the side effect analyzer so that it generated the information it needed on an ad hoc basis, developing only that which is necessary. Such an approach implies that the information would be volatile in the sense it would not be saved from update run to update run but would be regenerated as necessary.

All the components illustrated in Figure 7 and discussed in the prior paragraphs operate around the major control loop. This loop is shown in the flow diagram of Figure 8. Individual commands are received and analyzed, and included FORTRAN statements to statement fragments are recognized. Next, the appropriate statements that are going to be analyzed are retrieved, either from the database of definitions and related information, or from the OLD MASTER copy of the module embryo, and then recognized (if necessary). The retrieval/recognition process produces

(1) small modifications to individual statement/lines, or
(2) a complete replacement/deletion of a module or modules.

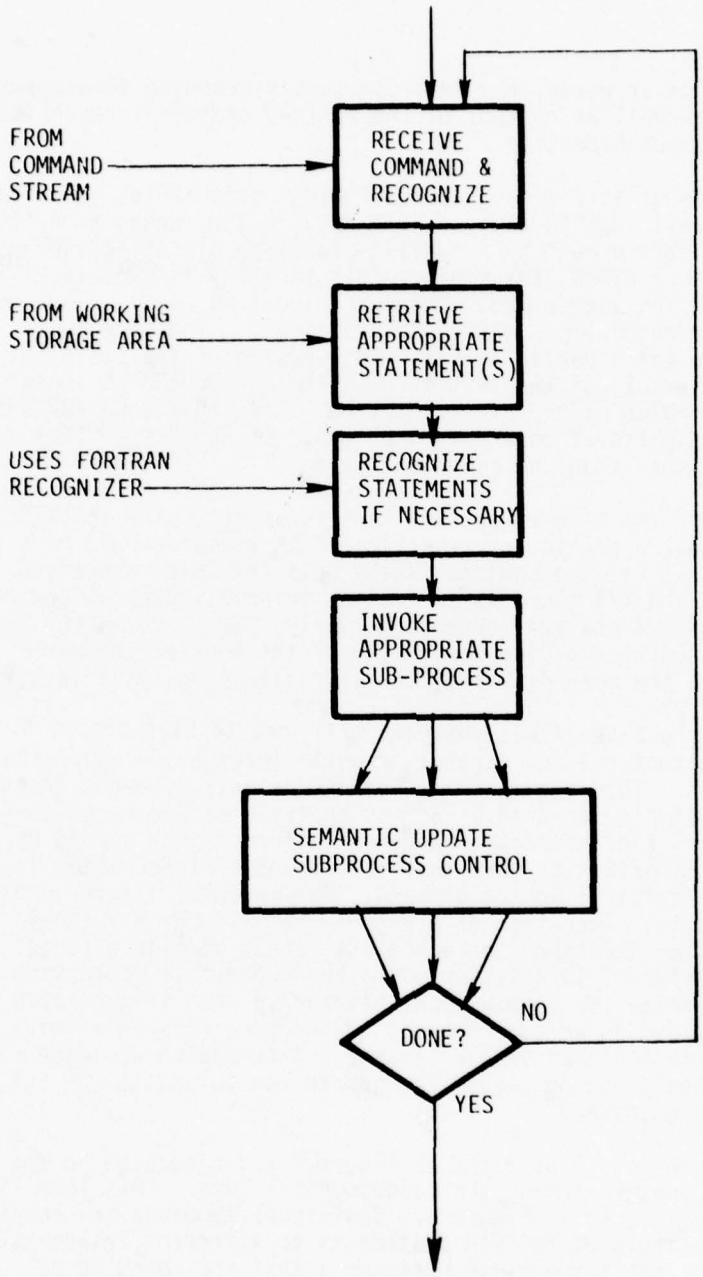


Figure 8: Basic Control Loop for Semantic Update System

MASTER will contain a fairly recent program text that can be used to

enough information to permit selection of the appropriate sub-process within the semantic update system; after the sub-process finishes executing the operations required based on the user's command, a check is made to determine whether there are remaining commands. This cycle continues until all of the user's commands have been processed. Note that this may or may not require accessing the complete OLD MASTER file; only those modules which are affected need be treated in detail.

4.4 Command Classes

Figure 9 gives the major classes of commands, and points out the main identifying feature of each class. Some of the classes affect the overall behavior of the semantic update process, while other commands are directly related to system-level structures for the software maintained on the OLD MASTER. The overall relationships between the commands on a by-class basis is depicted in Figure 10.

Class 1 commands control the type of processing to be done by the semantic update system, and include such data as the following:

- Identification of the OLD MASTER FILE.
- Identification of the programmer so his access rights can be checked.
- Control what output is created by the system. This includes the NEW - MASTER FILE, compiler output, reports, and whether the update is a temporary or permanent one.
- Establish new systems.

Class 2 and 3 commands control modifications made to modules known within the semantic update system. Class 2 commands are those which affect a single module and which have side-effects that are limited to that module alone. Class 3 commands also affect a single module, but have side effects that extend beyond the module at which the change originates.

The commands in Class 2 and Class 3 involve the following primitive kinds of actions:

- Add text to the program.
- Delete text from the program.
- Change the program.

In some cases a "change" is the equivalent of a combined add-delete pair, and in other cases a change operation is more complicated than the combination.

CLASS NUMBER	DESCRIPTION	NUMBER OF COMMANDS IN CLASS
1	COMMANDS TO THE SEMANTIC UPDATE SYSTEM	7
2	COMMANDS THAT AFFECT A SINGLE MODULE AND HAVE NO GLOBAL SIDE EFFECTS	25
3	COMMANDS THAT AFFECT A SINGLE MODULE AND HAVE GLOBAL SIDE EFFECTS	
4	COMMANDS THAT AFFECT PROGRAM COMMON DECLARATIONS	9
5	COMMANDS THAT AFFECT MODULE DEFINITIONS, INCLUDING THE NAME AND PARAMETER LIST	6
6	COMMANDS THAT AFFECT GLOBAL DEFINITIONS (EXCLUDING PROGRAM COMMON) (GENERAL MACRO)	2
7	COMMANDS THAT AFFECT SYSTEM STRUCTURE	7
8	COMMANDS THAT AFFECT CONDITIONAL PROCESSING	3
9	COMMANDS THAT AFFECT SYSTEM ARCHIVING FEATURES	3
10	COMMANDS THAT REPORT SYSTEM STATUS	10

Figure 9: Major Classes of Semantic Update Commands

Waters, R.C., "A System for Understanding Mathematical FORTRAN Programs,"
MIT, Artificial Intelligence Laboratory, AIM-368, August 1976.

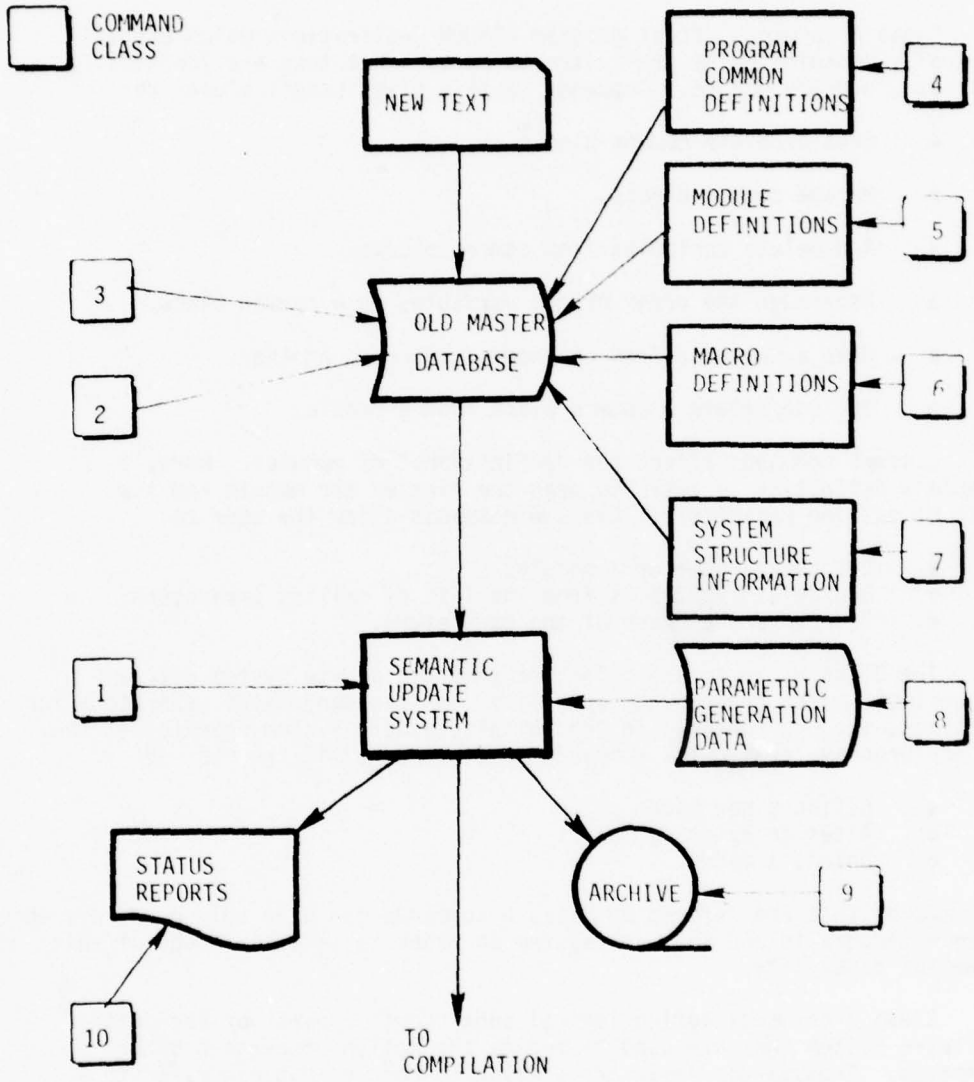


Figure 10: Impact of Commands on Semantic Update System

Bobrow, D.G., and Winograd, T., "An Overview of KRL, A Knowledge Representation Language," Stanford University, Artificial Intelligence Laboratory, AIM-293, November 1976.

Class 4 commands affect program COMMON declarations which are all centrally coordinated by the system to assure that they are identically the same in every module. Commands in this class permit a user to:

- Create/delete common blocks.
- Rename common blocks.
- Add/delete variables from common blocks.
- Rearrange the order of the variables in a common block.
- Move a variable from one common block to another.
- Include/delete a common block from a module.

Class 5 commands affect the "definitions" of modules. Here, a module's definition is taken to mean the name of the module and its list of calling parameters. Class 5 commands allow the user to:

- Change the name of a module.
- Add/delete variables from the list of calling parameters.
- Rearrange the order of the parameters.

The Class 6 commands provide the semantic update system with a generalized macro processing capability. Unlike many macro generators for programs, the one included in the semantic update system operates at the source-program level. The commands in Class 6 permit the user to:

- Define a new macro.
- Alter an existing macro.
- Delete a macro.

For macros that are defined by Class 6 commands the user only need reference them elsewhere in the software system in order to take advantage of this powerful capability.

Class 7 commands define logical subsystems or parts of the entire software system that are used to modify the action of certain other commands. Subsystem's names are similar to modules but can refer to a set of several modules. Class 7 commands permit a user to:

- Define/delete and sub-system.
- Add/delete a module from a sub-system.
- Add modules to a subsystem until it contains all the modules it calls.

would make it possible to define major elements of a software system and base subsequent commands on such definitions.

- Apply subsequent commands to a subsystem.

The Class 8 commands provide the user with the capability to control the conditional text generation features of the semantic update system. Basically, a user has parametric control[13] over selection of source text at the final COMPILE-file generation point. Based on the current values of run-time parameters the system selects previously identified statements for inclusion and/or exclusion. Class 8 commands include the capability to:

- Set the actual value of a generation-time parameter.
- Declare a new parameter
- Delete a previously used parameter

Class 9 commands deal with the archiving function of the semantic update facility. The archive produced contains a complete record of all transactions made; this record is likely to be more comprehensive than the changes that would reflect through an OLD MASTER/NEW MASTER cycle, where the primary concern is generation of the "new" software system representation. The user has the capability, with Class 9 commands, to indicate the beginning and end of epochs in the history archive, as well as to instruct the system to update the history archive.

Class 10 commands specify which system-level reports are to be generated during the current semantic update run. A variety of reports, intended to be of use to the programmer, can be generated. Primarily these involve reference-format listings of various system tables, definitions, etc.

4.5 A Sample Command

Figure 11 illustrates a sample command in the Class 3 command class which would be used to increase or decrease the maximum values that a variables subscript could assume.

5 DESIGN CONSIDERATIONS

This section presents some considerations that are important in the design of the semantic update facility. The considerations relate both to features of FORTRAN software systems that can be exploited effectively by the semantic update system, and to a fairly general characterization of the FORTRAN environment.

It is important to appreciate the simplifications that can result from assuming that the software will be written in a specific programming language. While it would be inappropriate to argue the merits and shortcomings of FORTRAN, it certainly can be considered to be a language that

[13] Kruskal, V.J., "An Editor for Parametric Programs," IBM Research Report, RC-6070, June 1976.

Figure 4: Classes of Source Text Processors

SYNTAX:

SIZE-SUBSCRIPT VARIABLE (SUBSCRIPT-LIST)

GENERAL EXPLANATION:

INCREASE OR DECREASE MAXIMUM VALUES SUBSCRIPTS ARE ALLOWED TO TAKE.

ERRORS AND WARNINGS:

W1 NO INSTANCE OF VARIABLE FOUND

W2 NO SUBSYSTEM NAMED (ASSUMED TO APPLY TO THE TOTAL SYSTEM)

W3 A PROGRAM ERROR MAY HAVE BEEN CREATED

E1 THE SUBSCRIPT LIST CONTAINS THE WORNG NUMBER OF ENTRIES

SIDE EFFECTS:

A DECREASE COULD AFFECT ALL VALUES WHERE HIGHER LIMITS ARE USED. THESE ARE FLAGGED FOR PROGRAMMER ATTENTION.

EXAMPLE AND EXPLANATION:

SIZE SUBSCRIPT A (6,6)

VARIABLE A (3,12) IS CHANGED TO A (6,6) IN ALL DECLARATION STATEMENTS. STATEMENTS SUCH AS A (2,9) = 10 or A (2,I) = I + 1 ARE FLAGGED.

Figure 11: A Sample Command with SIDE EFFECTS

is widespread. In addition, there is a substantial wealth of applications software already coded in FORTRAN. Hence, the basic design of a semantic update system on the semantics of FORTRAN may be important for practical reasons, even if it is not an elegant notion.

One of the areas in which there can be significant savings is in the realm of statement recognition. As the command structures have suggested, it will ordinarily be necessary to know something about the statement that is submitted to the system. This knowledge would vary with the type of statement: some statement types would require more than others. For example, if an insert command is applied to an assignment statement then it is likely there are not side effects; the semantic update system needs only to know that the statement is an assignment, since subsequently it can ignore any conflict analysis.

The continued references made to commands applying to statements deserves some comment. Figure 12 illustrates a hierarchy for FORTRAN software systems and illustrates an important point. As the hierarchical structure shows there is a very orderly progression from "software system" down to module, but sub-module partitioning yields two alternative routes to the next-joined primitive, the statement. Furthermore, the relationship between the logical entity "statement" and its physical constituents (i.e., the real world implementations) is more complex. There seems to be no advantage for a semantic update system to operate in a disorderly structure; hence, as shown in Figure 12, semantic-based operations are restricted to apply only to the statement level and above. Making this restriction imposes only the difficulty of dealing with blocks vs. segments. A segment is a logically inseparable sequence of statements that is always executed as a unit; a block is a structurally contiguous sequence of statements whose execution is controlled by the same predicate. (The interior of if...else...end if constructions are blocks.) The difficulty arises because segments need not involve statements that are contiguous; for this reason, semantic update commands that alter control flow will be particularly difficult to deal with effectively, since the system would have to deal with unconnected sequences of source text.

The hierarchy shown in Figure 12 is also of importance in helping determine the extent to which conflict resolution would have to proceed. For example, we have already seen that commands that change the names of single variables within a module do not have side-effects elsewhere. However, modifications to the parameter lists of one module can, depending on the structure of the software system, affect a very large number of other modules. The hierarchy will help determine the limits to such side-effect analyses.

Because the semantic update system provides a different view of FORTRAN programs it is important to have a framework in which to visualize the effects the system would have under certain programming circumstances. This framework must be based on the "typical" structure of FORTRAN programs.

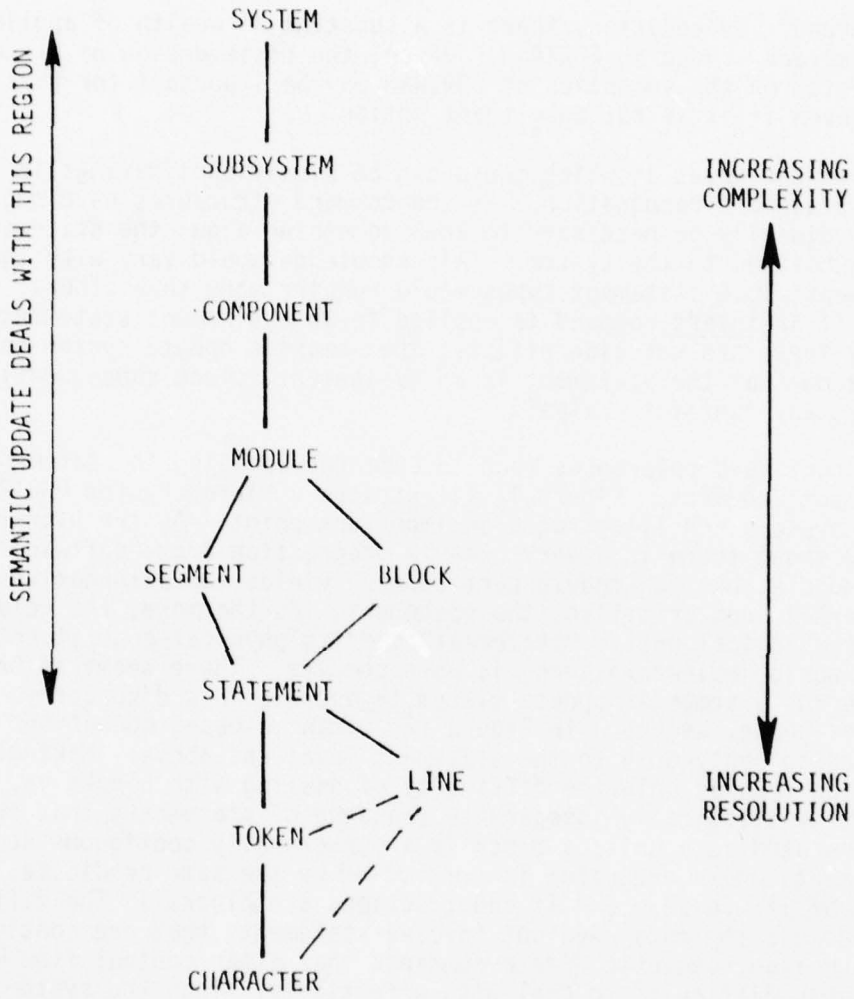


Figure 12: A Hierarchy for FORTRAN Software Systems

Figure 13 illustrates some of the attributes of a FORTRAN module in a generic format that will serve as the needed framework. As the Figure indicates it is possible to impose a rough classification on the statements in a FORTRAN module. Note that it is not necessary in the FORTRAN language that all of the declaration statements be in the particular order implied in Figure 13; the ordering presented there however, does represent good programming practice.

The FORTRAN module opens with a definition of its name and formal parameters. This defines the kind of module and specifies the globally understood information about it: the number of parameters it expects when it is invoked. Note that the FORTRAN language, like many others, can impose no specific restrictions on the correspondence between actual and formal parameters. This fact obtains primarily because the compilers for FORTRAN are incremental in nature; that is, they encounter only one subroutine or function at a time and thus cannot tell whether a particular invocation is valid or not.

Next come the definitions of the locally-understood types, dimensionality, and extent of the actual parameters. These definitions apply, of course, only to the current module. It is important to note that the actual definitions made cannot be in conflict with either a program COMMON declaration or with an internal function definition.

The next part of the module consists of references to one or more program COMMON statements, thereby establishing intercommunication between the module and others which also access the same COMMON statements. After the COMMON statements come the local definitions/declarations that apply throughout the module. As for formal parameters, such declarations cannot be in conflict with COMMON variables.

The next feature is a set of initializations of local variables. (It is not legal to initialize either a COMMON variable, except in a BLOCK DATA program, or a formal parameter.) All of the initializations must preceed the first executable statement. Normally internal function definitions are stated just prior to the first executable statement. The remaining parts of the FORTRAN module are the live or executable statements; they form the body of the module.

A final design-related note concerns the logical processing structure of the semantic update system. Because it is likely that only a very small number of modules will be altered during each session, or within each command set, it will be important to design the semantic update so that the largest file is accessed only once. This means that the system must be built so that it makes a single pass through the OLD MASTER file, which is likely to be many hundreds of times larger than the complete command file. This means that all of the processing for a single module must be completed

```
      SUBROUTINE or FUNCTION name (formal-parameter-list)
C     THIS STATEMENT DEFINES THE MODULE NAME AND CALLING STRUCTURE
C     declarations regarding the formal parameters
C           (i) name of formal parameter
C          (ii) type of formal parameter
C         (iii) extent of formal parameter
C
      COMMON /name/ list
C     THIS STATEMENT DEFINES THE COMMON BLOCK(S) TO BE USED BY THE
C     MODULE
C     each common block provides name, type, and extent information
C     for a number of program variables
C
      DECLARATIONS name(extent...)
C     THIS STATEMENT DEFINES LOCAL VARIABLES (NEITHER COMMON NOR
C     FORMAL PARAMETERS)
C     each local name is typed, and has extent and dimensionality
C     information as well
C
      INITIALIZATION...
C     THESE DATA STATEMENTS PROVIDE FOR INITIALIZATION OF ALL LOCAL
C     VARIABLES
C
      INTERNAL FUNCTION DEFINITION(s)...
C     THIS STATEMENT PROVIDES FOR DEFINITION OF LOCAL FUNCTIONS (IF
C     ANY ARE USED
C
      ...BODY
C     THIS IS THE BODY OF THE MODULE
C
C
      END
```

Figure 13: Typical Contents of a FORTRAN Module

for each module in turn before the fully-generated version is emitted to the COMPILE file. Consequently, the semantic update system will probably have to make several passes over each module as it is presented from the OLD MASTER file for processing.

6 DISCUSSION

We have presented the flavor of the semantic update system. Complete documentation appears in Miller and Praninskas [14,15].

We described a system being developed to modify complex software with ease and reliability. It is designed to examine a program and make modifications to it based on instructions it receives. It reports extensively on what it did and the problems and conflicts it encountered. It differs from the traditional update system in its knowledge of the programming language, in this case FORTRAN, and its ability to make multiple and varied changes to the software based on a single user command. Traditional updating systems do not perform the most elementary form of checking, not even to see if the line being replaced is exactly the same as the replacement line.

Cost savings using semantic updating compared with conventional methods are expected to be significant. The simplifications arise primarily through the automatic identification of side-effects. Because maintenance represents 40-60% of the life-cycle cost, a 10:1 reduction in maintenance translates into an approximate 50% savings overall. (There is reason to believe a 10:1 reduction is low based on early manual estimates of the number of steps saved through semantic updating.) An inportant aspect of future work will be detailed quantification of benefit/cost ratios.

The semantic update system is one of a genre of program understanding systems[16] which has evolved from recent ideas in artificial intelligence. It is a realizable project for program modifications and software repair. No assertions need accompany modification[17]. Furthermore, a semantic update system can be operational within a year or so and provide insights for some of the systems mentioned which may take a decade to develop if indeed they are even undertaken.

---

[14] Miller, E.F.Jr., and Praninskas, J.S., "Semantic Update Systems - A Conceptual Analysis," RP-104, Software Research Associates, San Francisco, California, October 1977.

[15] Miller, E.F.Jr., and Praninskas, J.S., "Semantic Update Systems - A System Specification," RP-105, Software Research Associates, San Francisco, California November 1977.

[16] Green, C.C., " Progress Report on Program Understanding Systems", Stanford University, AIM 240, August 1974.

[17] Dershowitz, N. and Manna, Z., "The Evolution of Programs: A System for Automatic Program Modification", Stanford University, Artificial Intelligence Laboratory, AIM-294, December 1976.

REFERENCES

1. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment", Datamation, May 1973, pp. 48-59.

2. Boehm, B. W., "The High Cost of Software", In. . .Practical Strategies for Developing Large Software, Addison Wesley, 1975 (12 pages).

3. Mills, H. D., "Software Development", Proceedings 2nd International Conference on Software Engineering Supplement, October 1976, pp. 79-86.

4. Putnam, L. H., and Wolverton, R. W., "Quantitative Management: Software Cost Estimating", IEEE Computer Society, Publication EHO-129-7, November 1977.

5. Yourdon, E., Techniques of Program Structure and Design, Prentice-Hall, 1975.

6. Kraft, P., Programmers and Managers, Heidelberg Science Library, New York, 1977.

7. Parnas, D. L., Personal Communication, ARO Workshop, February 1978.

8. Control Data Corporation, Update Manual, Form 84000016, March 1976.

9. Yau, S. S., Cheung, R. C., and Cochrane, D. C., "An Approach to Error-Resistant Software Design", Proceedings 2nd International Conference on Software Engineering, IEEE Catalog No. 76CH1125-4C, pp. 429-436.

10. Waters, R. C., "A System for Understanding Mathematical FORTRAN Programs", MIT, Artificial Intelligence Laboratory, AIM-368, August 1976.

11. Bobrow, D. G., and Winograd, T., "An Overview of KRL, A Knowledge Representation Language", Stanford University, Artificial Intelligence Laboratory, AIM-293, November 1976.

12. IBM Corporation, IBM OS/VS Utilities, IEBUPDTE User's Manual, Form GC35-0005-4, November 1975.

13. Kruskal, V. J., "An Editor for Parametric Programs", IBM Research Report, RC-6070, June 1976.

14. Miller, E. F. Jr., and Praninskas, J. S., "Semantic Update Systems - A Conceptual Analysis", RP-104, Software Research Associates, San Francisco, California, October 1977.

15. Miller, E. F. Jr., and Praninskas, J. S., "Semantic Update Systems - A System Specification," RP-105, Software Research Associates, San Francisco, California, November 1977.

16. Green, C. C., "Progress Report on Program Understanding Systems", Stanford University, AIM-240, August 1974.

17. Dershowitz, N., and Manna, Z., "The Evolution of Programs: A System for Automatic Program Modification", Stanford University, Artificial Intelligence Laboratory, AIM-294, December 1976.

SOFTWARE RESTYLING IN GRAPHICS AND PROGRAMMING LANGUAGES

Eric Grosse
Computer Science Department
Stanford University
Stanford CA 94305

ABSTRACT. The value of large software products can be cheaply increased by adding restyled interfaces that attract new users. As examples of this approach, a set of graphics primitives and a language precompiler for scientific computation are described. These two systems include a general user-defined coordinate system instead of numerous system settings, indention to specify block structure, a modified indexing convention for array parameters, a syntax for n-and-a-half-times-'round loops, and engineering format for real constants; most of all, they strive to be as small as possible.

0.0 PHILOSOPHY. Kernighan and Plauger [1976] describe explicitly and by example three precepts of the Software Tools philosophy:

- trim out the inessentials
- build it adaptively
- let someone else do the hard part

Two more examples, driven by the same philosophy, are given below. The basic idea is to obtain high leverage by taking an existing, powerful piece of software and make it useful to more people by designing a new interface. Webster's calls this process facelifting: "a restyling intended to increase comfort or salability."

1.0 JUSTIFICATION FOR STILL ANOTHER PROGRAMMING LANGUAGE. Fortran will no doubt remain for many years the most important programming language for scientific computation. When used carefully and with discipline, it yields remarkably portable codes; this is its greatest virtue. But, as programmers have complained for years, it also has many faults:

- awkward syntax for statements, strings, names
- primitive control structures
- DO loop restrictions
- no macros

Fortran preprocessors, such as MORTRAN [Cook+Shustek 1975], have eliminated many of these disadvantages and therefore have become very popular. Unfortunately, they reduce portability somewhat, since either the preprocessor must be installed at the new site

or illegible 'object' Fortran sent there. More importantly, such preprocessors have only a minor effect on inherent problems of Fortran:

- dynamic allocation is either unavailable or requires the use of rather confusing tricks
- no PROCEDURE VARIABLE type
- no STRUCTURE type
  (Labelled common blocks, since they do not use the combinatorial possibilities of procedure parameterization, are less flexible.)
- no 0-origin indexing
- array bound information is not autoratically passed
- no vector operations
- no recursion

The PORT library makes such heavy use of dynamic allocation that it has become one of the most advertised features: "We have found that use of dynamic storage allocation in PORT leads to more clearly structured programs, cleaner calling sequences, improved memory utilization, and better error detection." [Fox+Hall+Schryer 1977] Adding a stack to Fortran is a messy affair, however, as shown in figure 1, which contains two alternate methods in PORT for allocating an INTEGER and REAL array.

```
      SUBROUTINE LBB(A,N)
C
      COMMON /CSTAK/DSTAK(500)
C
      DOUBLE PRECISION DSTAK
      INTEGER ISTAK(1000)
      REAL A(1)
      REAL RSTAK(1000)
C
      EQUIVALENCE (DSTAK(1),ISTAK(1))
      EQUIVALENCE (DSTAK(1),RSTAK(1))
C
      II = ISTKGT(2*N,2)
      IR = ISTKGT(N,3)


      { code referring to RSTAK(IR+n) and ISTAK(II+m)
        probably ending with code to store the stuff
        from the real scratch storage into array A }


      CALL ISTKRL(2)
C
      RETURN
      END
```

```
      SUBROUTINE LBB(A,N)
C
      COMMON /CSTAK/DSTAK(500)
C
      DOUBLE PRECISION DSTAK
      INTEGER ISTACK(1000)
      REAL A(1)
      REAL RSTAK(1000)
C
      EQUIVALENCE (DSTAK(1),ISTAK(1))
      EQUIVALENCE (DSTAK(1),RSTAK(1))
C
      II = ISTKGT(2*N,2)
      IR = ISTKGT(N,3)
C
      CALL L1BB(A,ISTAK(II),RSTAK(IR),N)
C
      CALL ISTKRL(2)
      RETURN
      END
```

figure 1

Other proposals are even more complicated. (After a 7 page description of DYNOSOR, Huybrechts[1977] states: "This paper gives only the basic features of the DYNOSOR system. A more sophisticated use allows the user, once he is familiarized with the system, to improve greatly the speed of programs using it."

PL/I, which is now becoming fairly widely available in some form, overcomes all these difficulties. However, so huge a language tends to overwhelm people, and because of tricky precision rules, silent type conversions (as in I=J=0;), and the like, learning only part of the language is dangerous.

Other languages, while beautifully designed, have their own flaws. For example, Algol W does not have a robust interface to Fortran; in addition to this [Mohilner 1977], Pascal places painful restrictions on arrays.

1.1 T. Thus another approach seems warranted, which can combine the needed features of PL/I, the deliberate syntax of ALGOL, and

the low implementation cost of the Fortran preprocessors. Such an approach has produced the language T, intended to assist in the implementation and documentation of algorithms for scientific computation. The principal aims have been ease of reading and writing, low implementation cost, and reasonable efficiency.

Appendix T gives the formal language proposal, where syntax is specified using Wirth's proposal [1977]. Since T is similar to Fortran, Algol 60, and PL/I, a complete specification of the semantics may be omitted without confusion. To provide the heuristics behind the design choices and to give an overview of the language, various aspects of the following example will be discussed.

```
TRIPEAK
   # example of T and G systems;
   # various views of the sum of three Gaussian peaks;
   #   Eric Grosse   Stanford University

   REAL: AZIM, ELEV,      # VIEWING ANGLES FOR SURFACE PLOT
         RELERR, ABSERR,# ERROR TOLERANCES FOR ODE
         T, TOUT,         # INDEPENDENT VARIABLES OF TRAJECTORY
         NORMYP           # 2 NORM OF THE GRADIENT
   REAL(2): LL, UR,       # CORNERS OF RECTANGULAR DOMAIN OF FUNCTION
            ORIGIN,       # FOCAL POINT FOR SURFACE PLOT
            XO, SCALE,    # COORDINATE TRANSFORMATION PARAMETERS
            Y, YP         # LOCATION AND GRADIENT FOR TRAJECTORY
   REAL(142): ODEWORK
   INTEGER(5): ODEIWORK
   DEFINE(P,20)           # density of F samples;
   REAL(-P:P,-P:P): F TABLE
   REAL(3): LEVEL         # CONTOUR LEVELS
   INTEGER: I, J,
            IFLAG         # DIAGNOSTICS FLAG FOR ODE
   STRUCTURE: PARAM       # LOCATIONS, HEIGHTS, AND WIDTHS OF PEAKS
      REAL(3,2): X
      REAL(3): H, W
   STRUCTURE: PF          # PLOT FILE
      INTEGER(500): WORK
   PROCEDURE: GOPEN, GCLOSE, GPICT, GCONT, GSURF, GLTYPE,
        GJUMP, GDRAW, GTRAN1
   FORTRAN PROCEDURE: ODE, DF, STASH
   PROCEDURE () REAL: F

   # SET UP PARAMETERS
   BLANK SEPARATION (2)
   REAL DIGITS(3)
   GET DATA(AZIM,ELEV)
   PUT DATA(AZIM,ELEV)
   X(1,1) := 0
   X(1,2) := 0.5
   X(2,1) := -0.43301 2702
```

```
X(2,2) := -0.25
X(3,1) := -X(2,1)
X(3,2) := X(2,2)
PUT DATA ARRAY(X)
GET ARRAY(H)
PUT DATA ARRAY(H)
GET ARRAY(W)
PUT DATA ARRAY(W)
STASH(X,H,W)
FOR( -P <= I <= P )
   Y(1) := FLOAT(I) / P
   FOR( -P <= J <= P )
      Y(2) := FLOAT(J) / P
      F TABLE(I,J) := F(Y,PARAM)
# SURFACE PLOT
GOPEN('VEP12FF',PF)
GPICT(PF)
LL := -1
UR := 1
ORIGIN := 0.5
GSURF(LL,UR,FTABLE,AZIM,ELEV,ORIGIN,0.25,PF)

# CONTOUR PLOT
GPICT(PF)
SCALE := 0.3333
X0 := -0.5/SCALE(1)
GTRAN1(X0,SCALE,PF)
GET ARRAY(LEVEL)
PUT DATA ARRAY(LEVEL)
GCONT(LL,UR,FTABLE,LEVEL,PF)
GLTYPE('DOT',PF)
GET ARRAY(LEVEL)
PUT DATA ARRAY(LEVEL)
GCONT(LL,UR,FTABLE,LEVEL,PF)

# COMPUTE AND PLOT TRAJECTORY
RELERR := 10(-6)
GLTYPE('SOLID',PF)
ABSERR := 10(-6)
WHILE( ¬ END OF INPUT )
   GET ARRAY( Y )
   PUT DATA ARRAY( Y )
   T := 0
   GJUMP(Y,PF)
   IFLAG := 1
   WHILE( NORMYP > 1(-3) & 1<=IFLAG & IFLAG<=3 )
      TOUT := T + 10(-3)/NORMYP
      ODE(DF,2,Y,T,TOUT,RELERR,ABSERR,IFLAG,ODEWORK,ODEIWORK)
      CASE
         2 = IFLAG
            GDRAW(Y,PF)
         3 = IFLAG
            PUT('ODE DECIDED ERROR TOLERANCES WERE TOO SMALL.')
            PUT('NEW VALUES:')
```

```
            PUT DATA(RELERR,ABSERR)
         ELSE
            PUT('ODE RETURNED THE ERROR FLAG:')
            PUT DATA(IFLAG)
      FIRST
      DF(T,Y,YP)
      NORMYP := NORM2(YP)
   GCLOSE(PF)

F ( Y, PARAM ) Z
   REAL(): Y
   REAL: Z, NORMSQ
   STRUCTURE: PARAM
      REAL(3,2): X
      REAL(3): H, W
   INTEGER: I
   Z := 0
   FOR( 1 <= I <= 3 )
      NORMSQ := (Y(1)-X(I,1))**2 + (Y(2)-X(I,2))**2
      Z := Z + H(I)*EXP(-0.5*W(I)*NORMSQ)
```

1.2 CONTROL AND OTHER SYNTAX. Perhaps the most striking feature the Algol veteran sees in this example is the complete absence of BEGINs and ENDs. Not only is the text indented, but the indention actually specifies the block structure of the program. Such a scheme was apparently first proposed by Landin [1966]. Except for an endorsement by Knuth [1974], the idea seems to have been largely ignored.,

Ideally, the text editor would recognize tree-structured programs. [Hansen 1971] In practice, text editors tend to be line oriented so that moving lines about in an indented program requires cumbersome manipulation of leading blanks. Therefore the current implementation of T uses BEGIN and END lines, translating to indention on output. Thus the input

```
      STRUCTURE: PARAM
      ((
      REAL(3,2): X
      REAL(3): H, W
      ))
```

produces the output

```
      STRUCTURE: PARAM
         REAL(3,2): X
         REAL(3): H, W
```

Whatever the implementation, the key idea is to force the block structure and the indention to be automatically the same, and to reduce clutter from redundant keywords.

Blanks are insignificant outside of strings. Mathematical tables have long used blanks inside numeric constants, as in

```
      PI := 3.14159 26535 89793
```

for readability. Blanks in identifiers also can improve readability, while reducing the chance of misspelling and easing the pain of name length restrictions imposed by the local operating system.

In accordance with the recommendations of Scowen+Wichmann [1973], comments start with a special character, #, and run to the end of the physical line.

The small reserved word list eliminates the need for a stropping convention. The psychological advantages of this approach have been elaborated by Hansen [1973].

The form of the assignment and procedure call statements follows the clean, clear style of Algol 60. To make macros more understandable, their syntax and semantics match those of procedures as closely as possible.

In addition to normal statement sequencing and procedure calls, three control structures are provided. The CASE and WHILE statements are illustrated in this typical program segment:

```
WHILE( NORMYP > 1(-3) & 1<=IFLAG & IFLAG<=3 )
    TOUT := T + 10(-3)/NORMYP
    ODE(DF,2,Y,T,TOUT,RELERR,ABSERR,IFLAG,ODEWORK,ODEIWORK)
    CASE
       2 = IFLAG
          GDRAW(Y,PF)
       3 = IFLAG
          PUT('ODE DECIDED ERROR TOLERANCES WERE TOO SMALL.')
          PUT('NEW VALUES:')
          PUT DATA(RELERR,ABSERR)
       ELSE
          PUT('ODE RETURNED THE ERROR FLAG:')
          PUT DATA(IFLAG)
    FIRST
    DF(T,Y,YP)
    NORMYP := NORM2(YP)
```

The CASE statement is modelled after the conditional expression of LISP; the boolean expressions are evaluated in sequence until one evaluates to YES, or until ELSE is encountered. The use of indention makes it easy to visually find the relevant boolean expression and the end of the statement.

One unusual feature of the WHILE loops is the optional FIRST marker, which specifies where the loop is to be entered. In the example above, the norm of the gradient, NORMYP, is computed before the loop test is evaluated. Thus the loop condition, which often provides a valuable hint about the loop invariant, appears prominently at the top of the loop, and yet the common n-and-a-half-times-'round loop can still be easily expressed.

The FOR statement adheres as closely as practical to common mathematical practice.

```
FOR( 1 <= I <= 3 )
    NORMSQ := (Y(1)-X(I,1))**2 + (Y(2)-X(I,2))**2
    Z := Z + H(I)*EXP(-0.5*W(I)*NORMSQ)
```

Several years experience with these control constructs has demonstrated them to be adequately efficient and much easier to maintain than the alternatives.

Procedure nesting is not used for two reasons. First, textual nesting that extends over many pages is difficult for a human to keep track of. Second, programs typically contain several high level procedures calling a single primitive, so a tree representation is inappropriate anyway.

By removing the nesting of procedures, however, we worsen the problem of entry point hiding that arises when combining programs from many sources into a single library. A solution to this problem is to have an official name for each procedure, coded along the lines of IMSI, and also a more mnemonic nick name (which users can pick for themselves if they like). The macro processor which is built into T can then be used to change all occurences of the nick names into the corresponding official names.

1.3 DECLARATIONS. The fundamental scalar types are INTEGER, REAL, and COMPLEX, from which arrays and structures may be built up. As the example

```
REAL(-P:P,-P:P)
```

illustrates, general upper and lower bounds are allowed.

The upper bound expression is omitted for a formal array parameter, so that an appropriate value can be taken from the length of the corresponding actual array argument. The origin of an actual array argument need not match the origin of the corresponding formal array parameter. For example, if the actual argument A was declared REAL(0:7): A and the formal parameter B was declared REAL(): B, then B(8) will correspond to A(7). Most languages, when they allow lower bounds at all, do not permit this flexibility, which is used in the example program when a matrix with lower bound -P is passed to a general purpose library routine which assumes a lower bound of 0.

Structures of arbitrary depth may be declared. As the examples

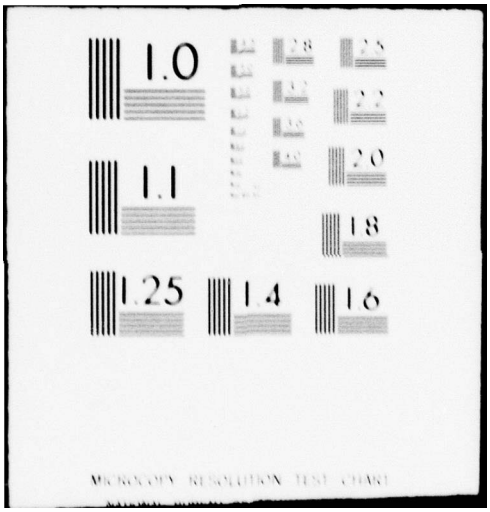
```
STRUCTURE: PARAM
    REAL(3,2): X
    REAL(3): H, W
STRUCTURE: PF
    INTEGER(500): WORK
```

AD-A061 561 ARMY RESEARCH OFFICE RESEARCH TRIANGLE PARK N C F/G 12/1
PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS C--ETC(U)
OCT 78

UNCLASSIFIED ARO-78-3 NL

2 OF 4
AD
A061 561




1.0
1.1
1.25
1.4
1.6
2.0
2.2
2.5
2.8
3.2
3.6
4.0
18
MICROCOPY RESOLUTION TEST CHART

suggest, structures are useful passing collections of related data, without the need for long parameter lists. This makes feasible the prohibition of global variables in a drastic attempt to narrow and make more explicit the interface between procedures. Euclid [Popek+others 1977] has emphasized the importance of visibility of names.

The graphics procedures which use the WORK vector of the example are able to divide up the space into convenient units. This capability, which would be possible in PL/I only through the use of pointers, encourages information hiding and abstraction.

PROCEDURE VARIABLEs allow the names of procedures to be saved, an essential feature for applications like the user-specified coordinate transformation described in the graphics system below.

The importance of existing Fortran software is recognized by providing for FORTRAN PROCEDUREs as an integral part of the language. The current implementation of T performs this linkage in a more efficient way than the naive user of PL/I would be likely to discover.

A novel syntax is introduced for function returns. Since procedures may be recursive, Fortran's convention of using the function name as variable cannot be followed. Instead, the procedure header declares a return variable just like any other parameter:

```
F ( Y, PARAM ) Z
   REAL(): Y
   REAL: Z
   ...
```

1.4 INPUT/OUTPUT. Beginners often find Fortran's input/output the most difficult part of the language, and even seasoned programmers are tempted to just print unlabelled numbers, often to more digits than justified by the problem, because formatting is so tedious. PL/I's list and data directed I/O is so much easier to use that it was wholeheartedly adopted in T. By providing procedures for modifying the number of decimal places and the number of separating blanks to be output, no edit-directed I/O is needed. Special statements are provided for array I/O so that, unlike PL/I, arrays can be printed in orderly fashion without explicit formatting.

Since almost as much time is spent in scientific computation staring at pages of numbers as at pages of program text, much thought was given to the best format for displaying numbers.

In accordance with the "engineering format" used on Hewlett-Packard calculators and with standard metric practice [GM Service Section 1977], exponents are forced to be multiples of 3. As

figure 2, an excerpt from the example program's output, shows, this convention has a histogramming effect that concentrates the information in the leading digit, as opposed to splitting it between the leading digit and the exponent, which are often separated by 14 columns. The use of parentheses to surround the exponent, like the legality of imbedded blanks, was suggested by mathematical tables. This notation separates the exponent from the mantissa more distinctly than the usual E format.

53.519(-03)	5.35186E-02
51.311(-03)	5.13109E-02
46.721(-03)	4.67211E-02
40.651(-03)	4.06514E-02
33.764(-03)	3.37636E-02
26.491(-03)	2.64908E-02
18.981(-03)	1.89808E-02
11.346(-03)	1.13461E-02
3.635(-03)	3.63508E-03
-4.129(-03)	-4.12944E-03
-11.912(-03)	-1.19123E-02
-19.709(-03)	-1.97092E-02
-27.525(-03)	-2.75248E-02
-35.324(-03)	-3.53243E-02
-43.118(-03)	-4.31176E-02
-50.907(-03)	-5.09068E-02
-58.684(-03)	-5.86841E-02
-66.448(-03)	-6.64483E-02
-74.197(-03)	-7.41973E-02
-81.930(-03)	-8.19297E-02
-89.644(-03)	-8.96443E-02
-97.340(-03)	-9.73401E-02
-105.016(-03)	-1.05016E-01
-112.670(-03)	-1.12670E-01
-120.302(-03)	-1.20302E-01
-127.910(-03)	-1.27910E-01
-135.493(-03)	-1.35493E-01
-143.050(-03)	-1.43050E-01
-150.576(-03)	-1.50576E-01

figure 2

1.5 DISCUSSION.

Following Kernighan+Plauger [1976], the initial implementation is unsophisticated [Comer 1978]. Nevertheless, the preprocessing is less costly than the PL/I compile, so the overall results are quite satisfactory. (The evaluation looks even better if one compares PL/I + T against PL/I + PL/I's macro preprocessor.) Most of the processor cost lies in basic I/O; by integrating the

macro processor with the language translator, this cost has been minimized. [Kantcrowitz 1976] Much of the two-man-months spent in implementation were spent in understanding nooks and crannies of PL/I.

T is not intended to replace any existing languages. For distributing mathematical software, Fortran remains the only practical medium; for character processing, something like PL/I or SNOBOL should be used. Still, for the bulk of scientific computation, T ought to be the easiest to use, particularly since it coexists comfortably with Fortran and PL/I. On the other hand, one can imagine ways that T might be improved, as well. Features omitted for ease of implementation include:

- trimmed arrays, like X(2:N)
- procedure results of general type
- conditional boolean operators that do not evaluate their arguments when it is possible to avoid doing so
- a swap operator

For other features, no entirely satisfying design was apparent:

- strings
- more general procedure calls (such as indefinite number and type of arguments)
- a means of constructing arrays directly from components, as a string constant constructs a string from individual characters
- a means of specifying the invocation graph of who calls whom

Perhaps the most fundamental though unavoidable flaw is that, unlike LISP, the language is not trivial, and therefore programs cannot be trivially manipulated.

2.0 JUSTIFICATION FOR STILL ANOTHER SET OF GRAPHICS PRIMITIVES.

The next example of restyling is a simple but reasonably complete interface for noninteractive device-independent graphics. In addition to the basic line drawing primitives, higher level procedures are provided for displaying functions of one or two variables. This interface has been implemented as a library of PL/I procedures which call the SLAC Unified Graphics package written by Robert Beach [1978]

Unified Graphics, with its emphasis on the ability to drive displays like the IBM 2250, is troublesome to use directly for function plots and the like. In contrast, Top Drawer, another graphics system at SLAC, allows for function plots but little else. The collection described in detail in Appendix G is meant to strike a useful balance between these two extremes, and contains most of the features of DISSPLA important for scientific computation.

2.1 ESTABLISHING THE ENVIRONMENT. The following excerpt from the example program given in section 1.1 above illustrates typical preparation for plotting:

```
STRUCTURE: PF          # PLOT FILE
   INTEGER(500): WORK
REAL(2): LL, UR,       # CORNERS OF RECTANGULAR DOMAIN
         ORIGIN,       # FOCAL POINT FOR SURFACE PLOT
         X0, SCALE     # COORDINATE TRANSFORMATION PARAMETERS
GOPEN('VEP12FF',PF)
GPICT(PF)
SCALE := 0.3333
X0 := -0.5/SCALE(1)
GTRAN1(X0,SCALE,PF)
```

The plot area PF is used to remember various options and to buffer low level plotter instructions. This work area is initialized by the GOPEN call, which specifies the output device. (In the current implementation, no corresponding JCL changes are necessary.) The ease with which devices may be changed is very useful in tuning a plot for publication.

For compatibility with numerical procedures, REAL variables are in full precision, not short. At the start of each new picture, which might be a screenful on a CRT or an 8.5 by 11" page on an electrostatic plotter, GPICT is called.

All plotting is done relative to a user coordinate system, which is specified by calling

```
GTRAN( F, PF )
```

where F is the name of a procedure which, when called in the form

```
F( X, W, PF )
```

with

```
REAL(N): X        N<=10
REAL(2): W
```

will map the point X in user coordinates into a point W in the unit square [0,1]x[0,1]. Normally W(1) is thought of as horizontal and W(2) as vertical. By extending PF, the user can pass parameters to F. For convenience, the default transformation maps

```
W := SCALE * ( X - X0 )
```

2.2 DRAWING, DIMENSIONING, AND FUNCTION GRAPHING. The basic drawing commands are GJUMP, GDRAW, and GTEXT for drawing lines and adding text. If a nonlinear coordinate system has been specified, GDRAW produces a piecewise linear approximation to the implied curve.

A procedure GGRAPH is provided which automatically samples function values, sets up an appropriate scaling, graphs the function, and dimensions the graph using round numbers in a style consistent with the format used by T. Figure 3, taken from Chan [1978], is a typical plot.

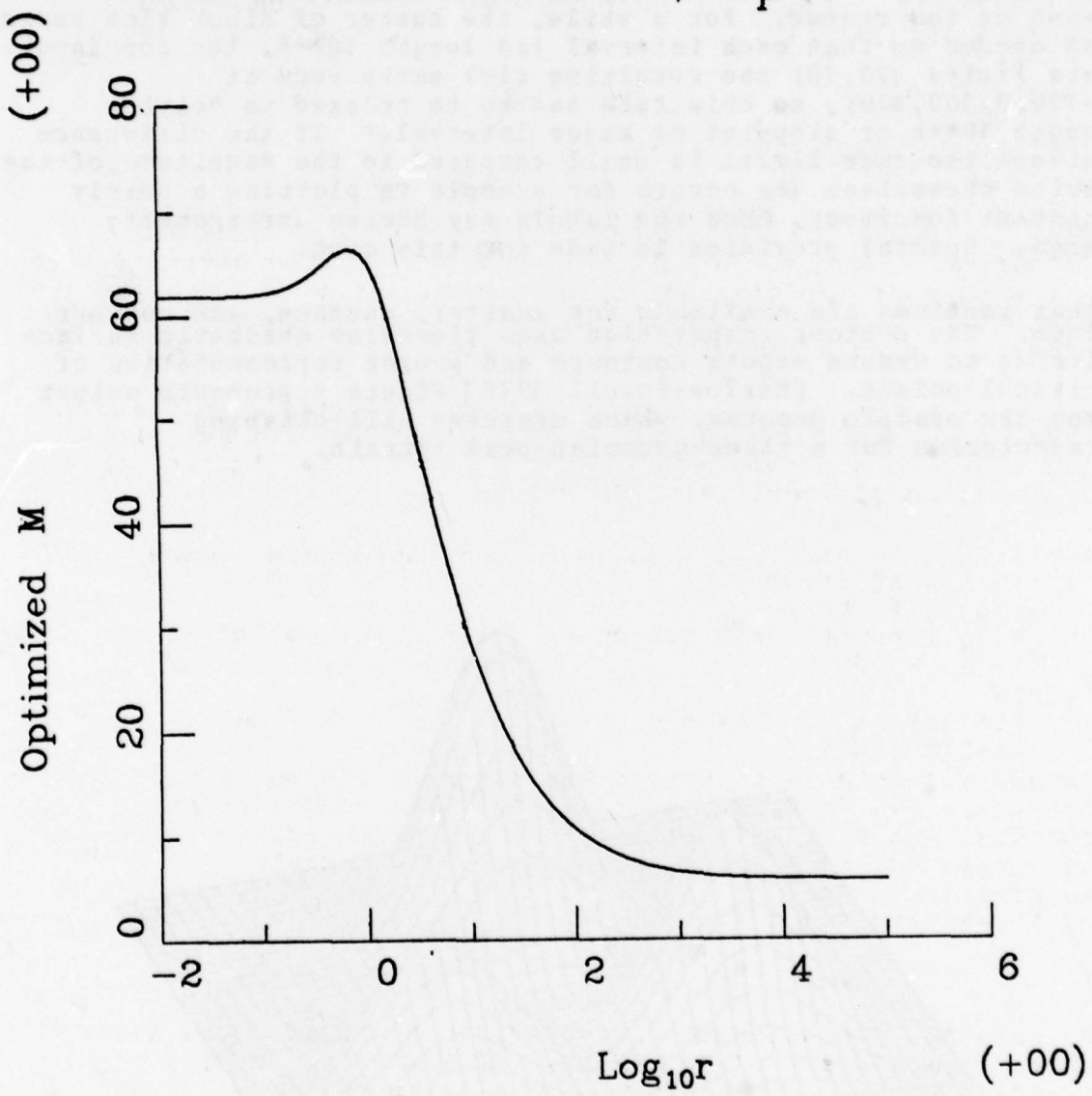
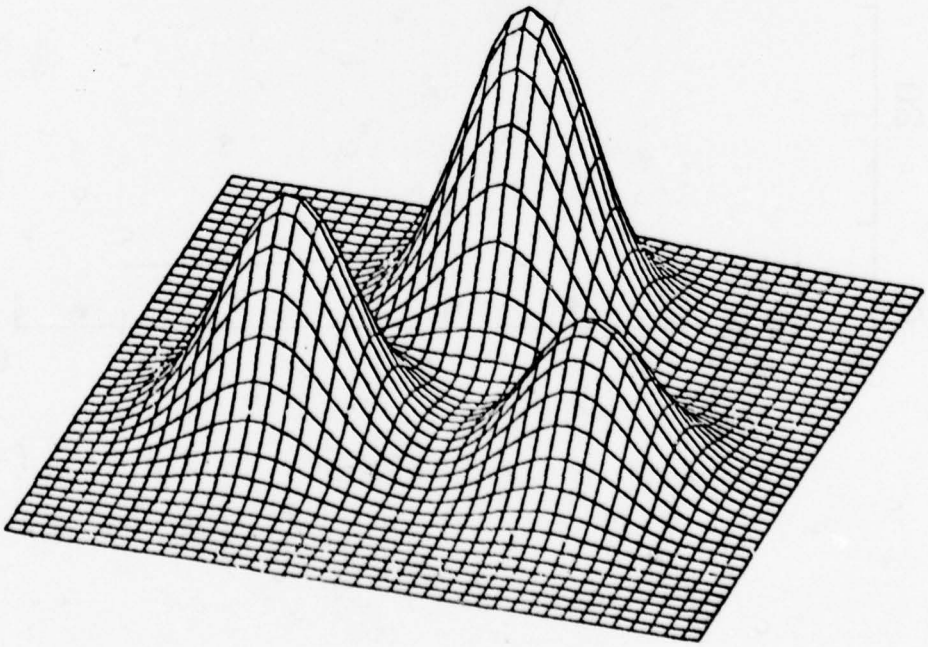

Scheme LF2DF2, E_p = 0.01
Optimized M
(+00)
80
60
40
20
0
−2
0
2
4
6
$Log_{10}r$
(+00)

figure 3

The scheme for choosing round numbers is based on the algorithm by Dixon+Kronmal [1965]. Experience and an informal survey of what people would accept as being "round numbers" led to various refinements. As in Unified Graphics, the choice is optimized over a reasonable number of major tick marks. The total number of tic marks, major and minor, is not allowed to be either too dense or too sparse. For a while, the number of minor tick marks was chosen so that each interval had length 10**k, but for input data limits (20,70) the resulting tick marks were at (-100,0,100,200), so this rule had to be relaxed to "either length 10**k or midpoint of major interval." If the difference between the data limits is small compared to the magnitude of the limits themselves (as occurs for example in plotting a nearly constant function), then the labels may become unreasonably large. Special provision is made for this case.

Other routines are available for scatter, surface, and contour plots. The contour computation uses piecewise quadratic surface fitting to ensure smooth contours and proper representation of critical points. [Marlow+Powell 1976] Figure 4 presents output from the example program, which computes hill-climbing trajectories for a three-gaussian-peak terrain.



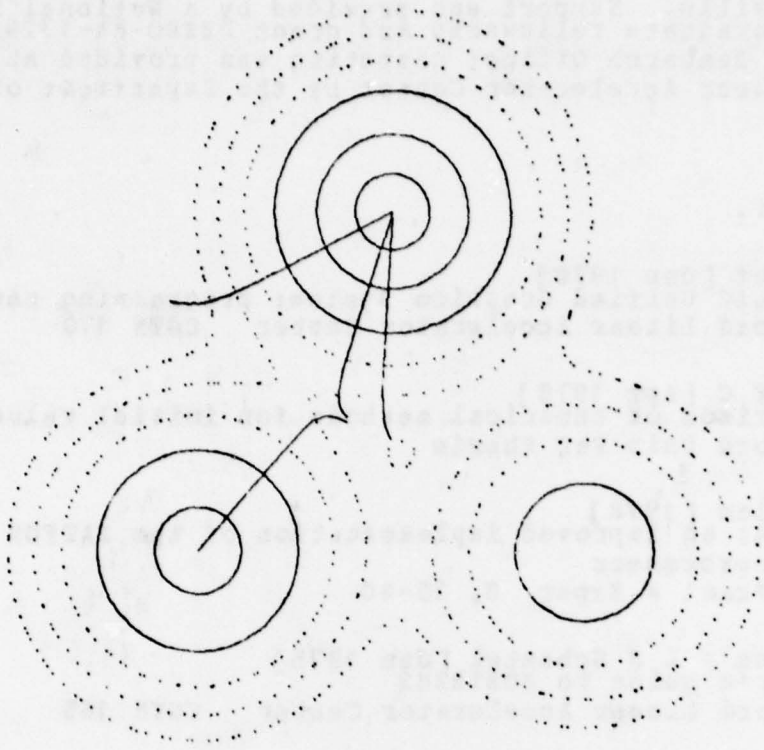


figure 4

CONCLUSION. With a level of effort comparable to writing a Fortran preprocessor, we have created, by compiling into PL/I, a language substantially better than Fortran or its derivatives. Since PL/I problems cannot be altogether avoided by this approach, further work on a language like T could be useful. Perhaps the effort would be better spent on making LISP a practical language for scientific compuation by building on the research in symbolic computation.

Like PL/I, Unified Graphics is good for a wide range of applications. But in practice, many people won't use either. For languages, they stick to Fortran; for graphics, they plot by hand or not at all. In both cases it has proven possible to cheaply restyle the existing system, via a preprocessing phase or driver routines, in order to create more agreeable tools.

ACKNOWLEDGEMENTS. Special thanks go to Bill Coughran for discussions of this report and help with T's realization in a PL/I precompiler. Helpful comments were made by Petter Bjorstad, Dan Boley, Tony Chan, Hector Garcia, Mike Heath, Randy Leveque,

and Bob Melville. Support was provided by a National Science Foundation graduate fellowship and grant DAAHO-MA-13292-M from the US Army Research Office; computing was provided at the Stanford Linear Accelerator Center by the Department of Energy.

BIBLIOGRAPHY.

Beach, Robert [Jun 1978]
The SLAC Unified Graphics System: programming manual
Stanford Linear Accelerator Center CGTM 170

Chan, Tony F C [Apr 1978]
Comparison of numerical methods for initial value problems
Stanford Univ PhD thesis

Comer, Douglas [1978]
MOUSE4: an improved implementation of the RATFOR preprocessor
Soft Pract + Exper 8, 35-40

Cook, A James + L J Schustek [Jun 1975]
A user's guide to MORTRAN2
Stanford Linear Accelerator Center CGTM 165

Dixon, W J + B A Kronmal [Apr 1965]
The choice of origin and scale for graphs
J ACM 12, 259-261

Fox, P A + A D Hall + N L Schryer [May 1977]
The PORT mathematical subroutine library
Bell Laboratories Murray Hill Comp Sci Tech Rep 47

GM Service Section [1977]
Metric usage guide
Detroit MI

Hansen, Wilfred J [Jul 1971]
Creation of hierarchic text with a computer display
Argonne National Laboratory ANL 7818

Hansen, Wilfred J [Nov 1973]
A revised Algol 68 hardware representation for ISO-CODE and EBCDIC
Univ Illinois Urbana UIUCDCS R 73 607

Huybrechts, M [Apr 1977]
DYNOSOR a set of subroutines for dynamic memory organization
SIGPLAN Notices Apr 1977, 67-74

Kantorovitz, E [Feb 1976]
Macro processors for efficient program production

European Research Office, US Army

Kernighan, Brian W + P J Plauger [1976]
Software Tools
Addison Wesley

Knuth, Donald E [Dec 1974]
Structured programming with go to statements
Computing Surveys 6, 261-301

Landin, P J [Mar 1966]
The next 700 programming languages
Comm ACM 9, 157-166

Marlow, S + M J D Powell [Apr 1976]
A Fortran subroutine for plotting the part of a conic that is inside a given triangle
UK AERE Harwell R8336

Mohilner, Patricia R [1977]
Using PASCAL in a FORTRAN environment
Soft Pract + Exper 7, 357-362

Popek, G J + J J Horning + B W Lampson + J G Mitchell + R L London [Mar 1977]
Notes on the design of EUCLID
SIGPLAN Notices Mar 1977, 11-18

Scowen, R S + B A Wichmann [May 1974]
The definition of comments in programming languages
Soft Pract + Exper 4, 181-188

Wirth, Niklaus [Nov 1977]
What can we do about the unnecessary diversity of notation for syntactic definitions?
Comm ACM 20,822-823

APPENDIX T

Report on the programming language T

"trim: free from anything extraneous;
having clean lines or proper proportion;
the state of readiness for action or use;"
Webster's Third New International

"Everything should be as simple as possible,
but no simpler."
Einstein

"What it lies in our power to do,
it lies in our power not to do."
Aristotle

"In all spheres, the true craftsman is
the one who thoroughly understands his tools."
Hoare

"Let someone else do the hard part."
Kernighan + Plauger

TOKENS. Program text is made up of the following tokens:

keyword - one of the following:

CASE	NO
COMPLEX	PROCEDURE
ELSE	PROCEDURE VARIABLE
FIRST	PUT
FOR	PUT ARRAY
FORTRAN PROCEDURE	PUT DATA
FORTRAN PROCEDURE VARIABLE	PUT DATA ARRAY
GET	REAL
GET ARRAY	STRUCTURE
GET DATA	WHILE
INTEGER	YES

identifier - a letter optionally followed by more letters and digits.

integer-constant - one or more digits

real-constant - one or more digits , a ".", possibly more digits, a "(", possibly a "-", one or more digits, and a ")". Either the decimal point and succeeding digits or the parentheses and exponent, but not both, may be omitted. Thus 1., 0.23, 6.22(-23), 1(-6), and 3.14159(+00) are all legal real-constants.

string-constant - a sequence of characters enclosed in apostrophes. Apostrophes are not allowed in the string proper.

delimiter - one of the following:
eor ( ) $ @ : . , & | ¬ + - * ** / := = ¬= < <= > >=
(The last six are called relational-operators.)

Except in string-constants, blanks are insignificant and may be used freely for clarity. Text from a "#" up to the next "eor" (end-of-record) is treated as a comment.

PROCEDURES.

```
program   =  {procedure} .
procedure =  identifier [ "(" identifiers ")" [result] ] "eor"
             begin
             {declaration}
             {statement}
             end    .
identifiers =  {identifier,} identifier
result  =  identifier   .
begin   =  "((" "eor"   .
end     =  "))" "eor"   .
```

Parameters are passed using call-by-reference. "result" may be used just like any other formal parameter, in particular as a destination, but must be a scalar.

If the parameter list is omitted, the procedure is assumed to be the top level main program.

DECLARATIONS.

```
declaration  =  scalar-type [ "(" bounds ")" ] ":" identifiers "eor"
             |  "STRUCTURE"  ":" identifiers "eor"
                    begin
                    {declaration}
                    end
             |  ["FORTRAN"] "PROCEDURE" ["VARIABLE"]
                    ["()" scalar-type ] ":" identifiers "eor"
scalar-type  =  "INTEGER"
             |  "REAL"
             |  "COMPLEX"
bounds  =  [bounds,] [expression:] [expression] .
```

No global variables are allowed: communication occurs only through parameter lists. Declarations reserve storage on a stack; the variables are undefined until first assigned to.

If the <expression>: part of a bound is omitted, 1: is assumed. The second <expression> should be omitted for a formal array parameter, and an appropriate value will be taken from the length of the corresponding actual array argument. The origin of an actual array argument need not match the origin of the corresponding formal array parameter. For example, if the actual argument A was declared REAL(0:7): A and the formal parameter B was declared REAL(): B, then B(8) will correspond to A(7).

STRUCTURES are data areas assumed to be decomposed as indicated in the subdeclarations. Thus if actual parameter A is declared

```
      STRUCTURE: A
         INTEGER: I, K
         REAL: X
         COMPLEX: Z
```

and corresponding formal parameter F is declared

```
      STRUCTURE: F
         INTEGER: J, L
         COMPLEX: W
```

then A.I and F.J correspond, but A.Z and F.W do not. (If F.W is assigned to, both A.X and A.Z may be destroyed.)

A PROCEDURE VARIABLE is a variable that may refer to various actual procedures; in contrast, a PROCEDURE is literally the name of a procedure.

STATEMENTS.

```
statement  =  procedure-call "eor"
  |  destination ":=" expression  "eor"
  |  "GET(" identifiers ")"
  |  "GET ARRAY (" identifer ")"
  |  "GET DATA (" identifiers ")"
  |  "PUT(" arguments ")"
  |  "PUT ARRAY (" destination ")"
  |  "PUT DATA (" arguments ")"
  |  "PUT DATA ARRAY (" destination ")"
  |  "CASE" "eor"
        begin
        {boolean-expression "eor"
          begin
          {statement}
          end }
        ["ELSE" "eor"
          begin
          {statement}
          end  ]
        end
  |  "WHILE(" boolean-expression ")" "eor"
        begin
        {statement}
        ["FIRST" "eor"
        {statement} ]
```

```
      end
  |   "FOR("
      (( expression ("<"|"<=") destination ("<"|"<=") expression)
      |( expression (">"|">=") destination (">"|">=") expression) ) "eor"
          begin
          {statement}
          end    .
```

GET reads from the next record of the input data a sequence of constants, separated by commas. For GET DATA, each value should be prefixed with "identifier :="; the input values need not appear in the same order as the corresponding identifiers in the GET DATA, and if a value is omitted, the variable is left unchanged. PUT DATA and PUT write out the current values of the identifiers, labelled or unlabelled, in an intelligent fashion.

In other languages,

```
      CASE
         cond1
            case1
         cond2
            case2
         ELSE
            case3
```

might be written as:

```
      IF( cond1 ) THEN
         case1
      ELSE IF( cond2 ) THEN
         case2
      ELSE
         case3
```

Similarly,

```
      WHILE cond
         part a
         FIRST
         part b
```

might be translated as:

```
      GOTO FIRST
   TOP:  part a
   FIRST: part b
      IF( cond ) THEN GOTO TOP
```

If the FIRST line is omitted, it is assumed to be at the end of the loop; that is, part b is empty.

Finally,

```
      FOR( LOW <= I <= HIGH )
         loop
```

would be translated as

```
      FOR I = LOW TO HIGH
         loop
```

and

```
      FOR( HIGH > I >= LOW )
         loop
```

```
as
      FOR I = HIGH-1 BY -1 TO LOW
         loop
```
The destination and expressions controlling a FOR loop may not be modified inside the loop.

```
EXPRESSIONS.
destination  =  {identifier "."} identifier
             |  identifier "(" subscripts ")"
             |  @ procedure-identifier  .
subscripts  =  [subscripts,] [expression]
procedure-call  =  procedure-identifier [ "(" arguments ")" ]  .
procedure-identifier  =  identifier  .
arguments = [arguments,] ( expression | string-constant
                         | YES | NO  ) .
expression  =  arithmetic-expression
            |  boolean-expression  .
arithmetic-expression  =  ["-"] term
  |  arithmetic-expression "+" term
  |  arithmetic-expression "-" term  .
term  =  factor
      |  term "*" factor
      |  term "/" factor  .
factor  =  primary
        |  primary "**" primary
        |  primary "**-" primary  .
primary  =  integer-constant
         |  real-constant
         |  destination
         |  procedure-call
         |  "(" arithmetic-expression ")"  .

boolean-expression  =  [boolean-expression "|"] boolean-factor  .
boolean-factor  =  [boolean-factor "&"] boolean-secondary .
boolean-secondary  =  ["¬"] boolean-secondary
                   |  boolean-primary
boolean-primary  =  YES | NO
   |  destination
   |  procedure-call
   |  arithmetic-expression  relational-operator arithmetic-expression .
```

If a subscript is empty, it is assumed that an entire row or column is being referenced. If all the subscripts are empty, the parentheses and commas may also be omitted. Array expressions are performed elementwise.

To refer to a procedure without actually invoking it, put a @ before the procedure identifier.

If the operands are mixed INTEGER and REAL the result is REAL; if either is COMPLEX, the result is COMPLEX. Dividing an INTEGER by an INTEGER and raising an INTEGER to a power are illegal.

PRIMITIVES AND MACROS. The following macros are predefined:

```
LENGTH(array,i)  size of array in the i-th dimension
DADD, DMUL, DDIV   extended precision arithmetic
CEIL, FLOOR, SIGN, ABS, FIOAT, RE, IM
MAX, MIN ( x1, x2, x3, ... )
MOD ( x1, x2 )     smallest nonnegative r such that
        ( x1 - r ) / x2  is integral
PI, EPS, MAXREAL, MAXINTEGER
ACOS, ASIN, ATAN, COS, EXP, LN or LOG,
     LOG10, SIN, SQRT, TAN
```

The following procedures are predefined:

```
NEXT LINE[ (n) ]        skip to start of next line,
                        then put out n-1 blank lines
                        (if no argument then n=1)
NEXT PAGE               page eject
END OF INPUT            BOOLEAN expression
BLANK SEPARATION(n)     number of blanks to be left
                        between output values (default 1)
INTEGER DIGITS(n)       number of digits for integer
                        output  (default 5)
REAL DIGITS(n)          ... for real output  (default 5)
REAL LEADING DIGITS (n)     0<=n<=3   (default 3)
DATE AND TIME     is replaced by: 'dd mmm 19yy   hh:mm'
```

Macros are evaluated much like procedure calls. First, each argument is evaluated; then the macro is applied to its arguments by inline expansion according to the macro definition; finally, the replacement text is regarded as fresh input. During the evaluation, tokens beginning with "_" have the first "_" stripped off. (This allows macros to be temporarily "hidden.")

DEFINE(identifier, replacement text) defines a macro. The replacement text is a sequence of tokens, possibly containing eor and matched parentheses. Places in the replacement text where arguments are to be inserted during expansion are indicated by $ followed by a digit or letter.
IFELSE(a,b,c,d) is replaced by c if the token a is the same as b, otherwise by d. Either a or b may be empty. For example, the text

```
      DEFINE(VERBOSE,YES)
      IFELSE(VERBOSE,YES,PUT DATA(X),)
```

would be replaced by

```
      PUT DATA(X)
```

A macro defined before the first procedure applies globally; other macros, which may temporarily redefine the global ones, apply only to the procedure in which they are defined.

APPENDIX I

T implementation notes

FILE FORMAT. The Report mentions only an abstract end-of-record delimiter and not any particular file format. This is because the assumptions of local text editors are of overwhelming importance.

The current implementation provides for card image files, in which columns 73 through 80 are ignored. In order to obtain text records longer than 72 characters, continuation lines, flagged by a blank in column 1, may be used. Comments, starting with a "#" end in column 72.

More technically, each card has an end-of-line character EOL appended to it and each record has an EOR appended after the last EOL. Thus

```
  REAL:     # constants
    PI, EPS
```

is translated to

```
 REAL:  #constants  EOL  PI, EPS   EOL EOR
```

The T processor discards blanks and text from a # up to the next EOL. The runtime I/O package discards blanks, comments, and begins and ends. On the other hand, the PRINT program makes use of EOLs to intelligently format its listing.

The input/output procedures effectively append an infinite string of EOF characters to the end of the file. The characters EOL, EOR, and EOF are represented internally as the ASCII control characters US, RS, and FS.

OTHER REMARKS ON USING T. Strings may be passed through a procedure, for example from a high level routine to a graphics primitive, by declaring a formal paramter as INTEGER(1).

Various options may be invoked by a macro call of the form:

```
      option( ? )
```

where

```
      ? is either ON or OFF
```

and option is [default value is in brackets]

```
      RECURSIVE    [OFF]      recursive procedures
      SUBCHECK     [OFF]      subscript checking
      SHORT        [OFF]      short precision
      FORTRAN FACADE [OFF]    procedure looks to the
                               outside world like Fortran
      UNDERFLOW    [ON]       turn off underflow error messages
                   in the current procedure and its descendants
                   (reals that underflow are set to 0)
```

Arrays may have at most 5 subscripts.

For the purposes of STRUCTURE alignment, the sizes of the scalar types are:

INTEGER	1 unit
REAL	2 units
COMPLEX	4 units

REALs and COMPLEXes should start an even number of units into the STRUCTURE for fastest access.

JCL AT SLAC. In order to invoke the PRINT program (designed to intelligently list a set of T procedures), use the PRINT catalogued procedure stored in WYI.CG,EHG.E. An example of a PRINT run:

```
//  JOB
//PROCLIB DD DSN=WYL.CG.EHG.E,DISP=SHR
//P EXEC PRINT
//INPUT DD *
  ... T procedures ...
```

The PRINT program recognizes three commands, which look like comments:

#%E	causes the next line to appear on the next page
#%Ttitle	causes the characters immediately appearing after "#%T" to appear in the title line, and the next line will appear on the next page
#%Ln	causes only identions of at most level n to appear, e.g. #%L0 will cause indention to be suppressed

In order to invoke the precompiler and compile the resulting PL/I, use the TC catalogued procedure stored in WYL.CG.EHG.E. An example of a compile-and-go run:

```
//  JOB
//PROCLIB DD DSN=WYL.CG.EHG.E,DISP=SHR
//C EXEC TC
//INPUT DD *
  ... T procedures ...
//G EXEC PLIG
```

Provision has been made for a PL/I dump. To get this, add the JCL card:

```
//PLIDUMP DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
```

APPENDIX G

Report on the graphics interface G

> "In improving exploratory data analysis, we need to find new questions to ask of the data (probably the hardest task), and new ways to ask old questions. Throughout, arithmetic as a basis for preparing pictures is likely to be the keynote. It is most important that we see in the data those things we do not expect -- pictures help us in this far more than numbers, though we can gain a lot just by what numbers we use."
>
> John Tukey

ESTABLISHING THE ENVIRONMENT. In order to remember various options and settings, such as character size and line type, and as a buffer for lowlevel plotting commands, a work area PLOT is provided to the graphics procedures. This may be declared as:

```
STRUCTURE: PLOT
   INTEGER(500): WORK
   STRUCTURE: USER
   ...
```

The workspace USER, which may be as large or small as desired, allows parameters to be passed to user procedures called by G.

To initialize PLOT at the start of a run, call

```
GOPEN( DEVICE, PLOT )
```

where

DEVICE is a string containing one of the codes:

CALFICH	microfiche
PDS4013	Tektronix, Hewlett-Packard
VEP12FF	Versatec

Note that if VEP12FF,EXTSORT is specified, then the reordering of the plot commands necessary for the Versatec will not be done by a main memory sort (greatly reducing the run-time memory requirements). In this case an external sort step must be supplied.

Before each new picture, including the first, call

```
GPICT( PLOT )
```

Finally, at the end of the run, call

```
GCLOSE( PLOT )
```

Omitting this may cause the last picture to be lost.

For convenience, the initial transformation is GTRAN1A, which performs the mapping

```
U := SCALE * ( X - X0 )
```

where

```
      REAL(2): W, X, X0, SCALE
```
and X0 = (0,0), SCALE = (1,1). Tc change these values, which are saved in PLOT.WORK, call
```
      GTRAN1( X0, SCALE, PLOT )
```

Line types, character sizes, and character angles are specified by calling
```
      GLTYPE( LTYPE, PLOT )
      GCSIZE( CSIZE, PLOT )
      GCANGL( CANGL, PLOT )
   where
      LTYPE is a string containing one of the codes:
         SOLID, DOT, DASH, or DOT-DASH
      REAL: CSIZE,  # character spacing; initially 1;
            CANGL   # character angle, in radians
                    #  counterclockwise frcm horizontal;
                    #  initially 0;
```

BASIC DRAWING. To jump straight to the point X,
```
      GJUMP( X, PLOT )
   where
      REAL(): X    # destination (in user coordinates)
```
To draw a line from the current pcsition tc X,
```
      GDRAW( X, PLOT )
```
If the coordinate transformation is curvilinear, this produces a piecewise linear approximation to the implied curve. Following a change cf coordinate system, GJUMP should be called before GDRAW.

To write out text, call
```
      GTEXT( X, PRI, SEC, PLOT )
   where
      REAL(): X       # location for center of first character
      PRI, SEC  are strings of length at most 255
```

In order to obtain a large alphabet, text is presented to GTEXT using a pair of strings. Every pair of corresponding characters in the primary and secondary strings denotes one character in the extended alphabet.

The seccndary character for

lower Roman	upper Roman	lower Greek	upper Greek
L		G	H

is

For common special characters the secondary character is also a space. The primary character for Greek letters is the first letter of its English name or one of the special cases:

H	eta	F	phi	W	omega
Q	theta	Y	psi		

Additional special and control characters are available:

```
0C begin superscript
1  end
2  begin subscript
3  end
4  save position1
5  restore
6  save position2
7  restore
8  save position3
9  restore
E  increase size
F  decrease

1V half up
2     down
3  third up
4     down
5  sixth up
6     down

0U one back
1  half forward
2     back
3  third forward
4     back
5  sixth forward
6     back

MX is an element of
N  is not an element or
E  there exists
A  for all
I  intersect
U  union
<  is strictly contained in
>  strictly contains
L  is contained in
R  contains

UA up arrow
D  down
L  left
R  right
B  bidirectional

IS integral
J  contour integral
P  partial-d
D  del
S  plus-or-minus
X  times
:  divided by
+  abstract plus
*  abstract times
2  radical
0  infinity
/  back slash
(  left square bracket
)  right square bracket
L  left angle bracket
R  right angle bracket
A  left curly bracket
Z  right curly bracket
<  less or equal
=  not equal
E  equivalent to
Q  proportional to
>  greater equal
O  degree
N  section
G  dagger
F  double dagger
H  h bar
W  lambda bar
U  underscore
V  overscore
0P cross
1  diagonal cross
2  diamond
3  box
4  star
5  diagonal start
6  cross with serifs
7  diagonal cross with serifs
8  compass rose
9  octagon
```

For example, the definition of the gamma function is given by:

```
GTEXT(X,'(N-1)!=G(N)=I40052F05 E0-T1T0N-11DT',
        ' L     H L  SCCSCCC C LC LCLCL  CLL',PLOT)
```

DIMENSIONING. Given limits on the data range, suitable values for plotting the data may be obtained by calling

```
      GSCALE( DATA MIN, DATA MAX,                          # input
         LABEL MIN, LABEL MAX, EXP, MAJOR, MINOR )   # output
```

The data will then be bracketed by LABEL MIN*10**EXP and LABEL MAX*10**EXP, which are round numbers in engineering format.

GTIC acts somewhat like a ruler, drawing an unlabelled axis with large and small tic marks:

```
      GTIC( L, H, MAJOR, MINOR, OFF, PLOT )
   where
      REAL(): L, H,  # endpoints of axis;
              OFF    # offset coordinates of major tic mark
                     #  endpoints, which determine the size
                     #  and direction of the tic marks;
      REAL: MAJOR, MINOR  # number of major and minor tic marks,
                     #  counting endpoints;  thus a yardstick
                     #  might have MAJOR=4, MINOR=13;
```

GLAB adds integer labels to the tic marks produced by GTIC:

```
      GLAB( L, H, MAJOR, OFF, LOW, HIGH, PLOT )
   where
      INTEGER: LOW, HIGH    # label values at endpoints;
      REAL(): OFF           # offset of first character of label
```

L, H, MAJOR are as in GTIC.

GFORM1 lays out a form suitable for scatter plots and function graphs:

```
      GFORM1( GPRI, GSEC,
              XPRI, XSEC, XLOW, XHIGH,
              YPRI, YSEC, YLOW, YHIGH,  PLOT )
   where
      GPRI, GSEC, XPRI, XSEC, YPRI, YSEC are pairs of strings
            defining the general title, X- and Y-axis labels;
      REAL: XLOW,XHIGH,YLOW,YHIGH  # specifies the data limits
```

GFORM1 automatically sets up a coordinate system so that the plot will fill the screen. To add a function curve, just use GJUMP and GDRAW.

PLOTTING. GSCAT provides scatter plots:

```
      GSCAT( GPRI, GSEC,
             XPRI, XSEC, X,
             YPRI, YSEC, Y,  PLOT )
   where
      REAL(): X, Y      # data points
```

GPRI, ... , YSEC are as in GFORM1.

GGRAPH provides graphs of functions of one variable:

```
      GGRAPH( GPRI, GSEC,
              XPRI, XSEC, A, B,
              FPRI, FSEC, F,  PLOT )
   where
```

```
      REAL: A, B              # endpoints of interval for graphing
      PROCEDURE()REAL: F      # function to be plotted, which
                              # is called in the form:
                              #   Y = F( X, PLOT )
```

To plot contours of the surface passing through the points
(X(I), Y(J), F(I,J))
call

```
      GCONT( LL, UR, F, LEVELS, PLOT )
```

where

```
      REAL(2): LL, UR         #  coordirates of lower left (most
                              #  negative) and upper right (most
                              #  positive) corners of the rectangle
                              #  in the X Y plane on which data
                              #  is given
      REAL(,): F              #  data values
      REAL(): LEVELS          #  contour levels to be plotted
```

A uniform grid is assumed.

To draw a transect surface plot with hidden lines removed, call

```
      GSURF( LL, UR, F,
         AZIMUTH, ELEVATION, ORIGIN, SCALE,  PLOT )
```

where

```
      REAL: AZIMUTH, ELEVATION, SCALE
      REAL(2): ORIGIN
```

The coordinate transformation used in GSURF maps P = (X,Y,F) into

```
   ORIGIN  +  (  -S1     C1      0    )  (P-C)*SCALE
              (  -S2*C1  -S2*S1  C2   )
```

where C1 = COS(AZIMUTH), ..., S2 = SIN(ELEVATION)
C = ((LL(1)+UR(1))/2, (LL(2)+UR(2))/2, 0)
LL, UR, and F are as in GCONT.

BEST NODES FOR POLYNOMIAL INTERPOLATION

Carl de Boor
Mathematics Research Center
University of Wisconsin-Madison
Madison, Wisconsin 53706

ABSTRACT

Facts about polynomial interpolation conjectured some time ago by Bernstein and Erdös and proved recently by Kilgore, Pinkus and the speaker are discussed. These facts concern the choice of interpolation points which make the norm of the interpolation map as small as possible. It is then pointed out that such optimal nodes offer little improvement over readily available "good" nodes whose use is encouraged. Finally, the question of "good" interpolation points for polynomial approximation in several variables is raised.

COMPARING DIGITAL FILTERS WHICH PRODUCE DERIVATIVE APPROXIMATIONS*

Charles K. Chui, Philip W. Smith, and Joseph D. Ward
Department of Mathematics
Texas A&M University
College Station, Texas 77843

ABSTRACT. A method for comparing digital filters which produce derivative approximations is proposed. A judicious use of the Peano Kernel Theorem allows one to compare these digital filters rather cheaply.

1. INTRODUCTION. Digital filtering is often used in a variety of situations for smoothing data and approximating derivatives. In this article, we propose a method by which two competing filters may be judged. For simplicity, we will consider only digital filters of the type: $f'_j = D_j(\underline{\beta},\underline{f})$ where $\underline{\beta} = (\beta_0, \ldots, \beta_{r+p})$, $\underline{f} = (f_0, \ldots, f_N)$, and

$$D_j(\underline{\beta},\underline{f}) = \sum_{k=0}^{p} \beta_k f_{j-k} + \sum_{k=1}^{r} \beta_{k+p} f'_{j-k}, \tag{1}$$

$j = 0, \ldots, N$. Throughout, f_{-n}, f'_{-n} are considered to be zero if $n > 0$, and f'_j will be an approximation of $f'(j/N)$. The reader can see that the more general situation including the use of higher order derivatives can be treated in an analogous manner.

A related problem is to choose the best filter coefficients β_i in the sense of Sard [6]. This would also produce the optimal estimate analogous to the ideas of Golomb-Weinberger [5]. Basically, this entails choosing a class of functions B so that the best filter coefficients $\underline{\beta}^*$ satisfy the mini-max property

$$\inf_{\underline{\beta}} \left(\sup_{f\in B} |D_N(\underline{\beta},\underline{f}) - f'(1)| \right) = \sup_{f\in B} |D_N(\underline{\beta}^*,\underline{f}) - f'(1)|.$$

*This research was supported by the U.S. Army Research Office under Grant No. DAHC 04-75-G-0186.

PRECEDING PAGE BLANK

In section 2 we develop some theoretical results and in section 3 we discuss numerical implementation of a specific comparison method. In particular, it is shown that relatively cheap comparisons may be made.

2. DISCUSSION OF RESULTS. The type of filters to be considered may be described as follows. Let

$$H_2^n = \{f: \int_0^1 |f^{(n)}|^2 dt < \infty\}$$

and let $\{f_i\}_{i=0}^N$ be data samples taken at i/N, $i=0, \ldots, N$. We assume that there is a function $f \in H_2^n$ and

$$f_i = f(i/N), \qquad i=0, \ldots, N.$$

The type of recursive filter considered here is given by $f_j' = D_j(\underline{\beta},\underline{f})$, for $j=0, \ldots, N$, where $\underline{f} = (f_0, \ldots, f_N)$, $\underline{\beta} = (\beta_0, \ldots, \beta_{p+r})$, and $D_j(\underline{\beta},\underline{f})$ is as in (1). f_j' is then an approximation of $f'(j/N)$. The problem is to choose the β_k's in the above formula so that the derivative approximants are "best possible".

There is a classical method to attack this problem dating back to Golomb-Weinberger [5] and Sard [6]. Assume that (1) is exact for polynomials of degree less than or equal to $n-1$ and let B_n be the unit ball of H_2^n.

Proposition 1.2. The following inequality holds for $j=0, \ldots, N-1$.

$$\sup_{f \in B_n} |D_j(\underline{\beta},\underline{f}) - f'(j/N)| \leq \sup_{f \in B_n} |D_{j+1}(\underline{\beta},\underline{f}) - f'(j+1/N)|.$$

The proof is derived from considerations of the Peano Kernel Theorem and we omit the details. Note that the "worst" error must occur at $j=N$.

Definition 1.3. A (p+r)-tuple $\underline{\beta}^*$ is said to be optimal if

$$\sup_{f\in B_n} |D_N(\underline{\beta}^*,\underline{f}) - f'(1)| \le \sup_{f\in B_n} |D_N(\underline{\beta},\underline{f}) - f'(1)|$$

for all (p+r)-tuples $\underline{\beta}$.

Example 1.4. Consider the recursive filter given by

$$f'_j = (1-\alpha)[f_j - f_{j-1}]N + \alpha f'_{j-1} = D_j(\alpha,\underline{f}).$$

This filter is exact for polynomials of degree less than or equal to one. The optimal $\alpha^* \equiv \alpha^*(N)$ can be computed by minimizing the formula

$$\sup_{f\in B_2} |D_N(\alpha,\underline{f}) - f'(1)| = \sqrt{(1-\alpha^{2N})(1+\alpha+\alpha^2)/[3N(1-\alpha^2)]}$$

for $\alpha \neq \pm 1$. In particular, $\alpha^*(N)$ converges to $-2 + \sqrt{3}$ as $N \to \infty$.

The following may be observed about the above example:

i) The optimization problem is nonlinear.
ii) For each N, there is a unique solution α_N^*.
iii) The α_N^* converges as N goes to infinity.
iv) The rate of decrease is of order $O(1/N^{1/2})$.

For the general problem, we can prove

Theorem 1.5. a) (1) has an optimal solution $\underline{\beta}^*$.

b) If (1) is exact for all polynomials of degree less than or equal to n-1 and if p is larger than n-2, then there exist positive constants C_1 and C_2 so that

$$C_1 \le \sup_{f\in B_n} N^{-n+3/2} |D_N(\beta^*(N),\underline{f}) - f'(1)| \le C_2$$

Sketch of proof: Part a) is proved in a standard fashion by assuming that a minimizing sequence is unbounded and then deriving a

contradiction. For the proof of part b), consider the non-recursive filter

$$f'_j = \sum_{k=0}^{p} \beta_k f_{j-k}.$$

From Lemma 2.3 in [4], it is known that

$$\sup_{f \in B_n} |D_N(\underline{\beta}^*(N), \underline{f}) - f'(1)|$$

is bounded above by the reciprocal of the norm of a certain fundamental spline. The norm estimates of de Boor ([1, p. 38] and [2, p. 115]) for such splines now yield the upper bound. To obtain the lower estimate, note that

$$D_N(\underline{\beta}, \underline{f}) = \sum_{j=0}^{N} \gamma_j f(j/N).$$

The best estimator of this type is $S'_N(1)$. This follows from [5] where S_N is the natural spline of order $2n$ interpolating the data. The results of Golomb [4] and de Boor [1] and [2] now allow us to obtain a positive constant $C_1 > 0$ such that

$$C_1 N^{n-3/2} \leq \max_{f \in B_n} |S'_N(1) - f'(1)|.$$

This completes the sketch of our proof.

3. COMPUTATIONAL ASPECTS OF COMPARING FILTERS. As in the previous section we consider filters of the type $f'_j = D_j(\underline{\beta}, \underline{f})$ where $D_j(\underline{\beta}, \underline{f})$ is given in (1). We also assume that $f'_j = D_j(\underline{\beta}, \underline{f})$, $j=0, \ldots, N$, is exact for polynomials of degree $\leq n-1$. We may obtain an integral representation of the error over B_n via the Peano Kernel Theorem [3] of the form

$$\sup_{f \in B_n} |D_N(\underline{\beta}, \underline{f}) - f'(1)| = \sqrt{\int_0^1 [K^N_{\underline{\beta}}(t)]^2 dt} \tag{2}$$

where $K^N_{\underline{\beta}}(t) = \left(D_N(\underline{\beta}, (\cdot-t)^{n-1}_+) - (n-1)(1-t)^{n-2}\right)/(n-1)!$.

Thus, if one wants to know whether $\underline{\beta}^1$ or $\underline{\beta}^2$ produces a better differentiation formula $f'_j = D_j(\underline{\beta}^i, \underline{f})$ $(i=1, 2)$ it is only necessary to compare the numbers

$$\int_0^1 [K^N_{\underline{\beta}^i}(t)]^2 dt, \quad i=1, 2. \tag{3}$$

Computing the integral in (3) is made easier by noticing that for $t \in (j/N, (j+1)/N)$, $K^N_{\underline{\beta}}(t)$ is a polynomial of degree less than n. Hence, one could use Gauss quadrature formula with n points in each interval of the form $(j/N, (j+1)/N)$. This means that one must evaluate $K^N_{\underline{\beta}}$ at nN points on $[0,1]$, and off-hand one would suspect that the filter $f'_j = D_j(\underline{\beta}, \underline{f})$ would have to be applied to nN functions of the type $(x-\tau_j)^{n-1}_+$ where the τ_j would be the properly scaled Gaussian points.

However, since the digital filter is recursive we will indicate below that we need only apply it to n functions! This is of course a considerable saving since one would like to choose N large in order to guage the long term effects of the filter. Note that $f(x) \equiv (x-t)^{n-1}_+$ vanishes for all $x < t$ so that

$$D_N(\underline{\beta}, \underline{g}) = D_{N-j}(\underline{\beta}, \underline{f}) \tag{4}$$

where $g(x) = (x-(j/N + t))^{n-1}_+$. In particular, (4) indicates that all the information needed to compute $K^N_{\underline{\beta}}$ at the scaled Gaussian points can be obtained by applying the filter to the n functions

$$\left\{(x-\tau_j)^{n-1}_+\right\}_{j=1}^{n}$$

where the τ_j are the scaled Gaussian points in the interval $[0, 1/N]$.

We conclude this section with the following observation. Our original comparison criteria led us to consider

(5) $$\sup_{f \in B_n} |D_N(\underline{\beta}, f) - f'(1)|$$

as a measure of a filter's performance. But as we just indicated above we actually only need to apply the filter to n test functions in order to recover (5) as opposed to all the functions in B_n. This appears to be a remarkable savings.

References

1. C. de Boor, Odd-degree spline interpolation at a biinfinite knot sequence, in Approximation Theory, edited by R. Schaback and K. Scherer, Springer-Verlag Lecture Notes #556, New York 1976, 30-53.

2. C. de Boor, How small can one make the derivatives of an interpolating function?, J. Approx. Theory, 13(1975), 105-116.

3. P. J. Davis, Interpolation and Approximation, Blaisdell Publishing Co., Waltham, Mass, 1963.

4. M. Golomb, Interpolation operators as optimal recovery schemes for classes of analytic functions, in Optimal Estimation in Approximation Theory, edited by C. A. Micchelli and T. J. Rivlin, Plenum Press, New York, 1976, 93-138.

5. M. Golomb and H. F. Weinberger, Optimal approximation and error bounds, in On Numerical Approximation, edited by R. E. Langer. The University of Wisconsin Press, Madison (1959), 117-190.

6. A. Sard, Best approximate integration formulae; best approximation formulae. Amer. Jour. of Math. 71(1949), 80-91.

COMPUTABLE ERROR BOUNDS FOR THE NYSTRÖM METHOD

J. W. Hilgers
Department of Mathematics
Michigan Technological University
Houghton, Michigan 49931

and

L. B. Rall
Mathematics Research Center
University of Wisconsin-Madison
Madison, Wisconsin 53706

Dedicated to Professor J. Barkley Rosser on his 70th birthday.

ABSTRACT. The classical Nyström method for the numerical solution of Fredholm integral equations of second kind consists of numerical integration, collocation, and interpolation. The approximate solution obtained by this procedure is shown to be identical to the solution of certain finite-rank integral equations with kernels belonging to a specified class $\{K_n\}$, and thus has minimal error with respect to approximation of the original equation over this class. A computable (but in general nonoptimal) error bound for the Nyström approximate solution can be obtained on the basis of how well a specific finite-rank integral operator with kernel in $\{K_n\}$ approximates the integral operator in the Fredholm equation being solved numerically.

1. THE NYSTRÖM METHOD. The linear Fredholm integral equation of second kind,

$$x(s) - \lambda \int_0^1 K(s,t)x(t)dt = y(s), \qquad 0 \le s \le 1 \tag{1.1}$$

arises in the solution of boundary value problems for ordinary and partial differential equations, and other important applications. Given the function $y(s)$ and the kernel $K(s,t)$, one problem in connection with (1.1) is to obtain the solution function $x(s)$, at least for values of the parameter λ for which it is unique. Another problem, usually posed for the homogeneous equation ($y(s) \equiv 0$), is to find the eigenvalues and eigenfunctions of the kernel $K(s,t)$, that is, values λ^* of the parameter λ for which the homogeneous

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

equation has corresponding nontrivial solutions $x^*(s) \neq 0$. Fredholm [6] obtained solutions to these problems in terms of infinite series in λ, with coefficients expressed as repeated integrals of larger and larger determinants. As in the case of the analogous Cramer's rule for linear algebraic systems, Fredholm's formulas are satisfactory from a theoretical standpoint, but are usually unsuitable for practical computation. To remedy this situation, various numerical methods to obtain approximate solutions have been developed [4], including the simple and effective procedure due to Nyström [13]. Nyström's method consists of three steps: (i) numerical integration, which replaces the integral in (1.1) by a finite sum to obtain an approximating functional equation; (ii) collocation, which gives a finite linear algebraic system for values $z_1, z_2, \ldots, z_n$ of an approximate solution of (1.1) at n points $t_1, t_2, \ldots, t_n$; and (iii) interpolation, which uses the values found by collocation to construct a function $z(s)$ on the entire interval $0 \leq s \leq 1$ which is an approximate solution of (1.1) such that $z(t_i) = z_i$, $i = 1,2,\ldots,n$. These steps will be explained in greater detail in the following sections.

The simplicity of the Nyström method stems from the fact that it is basically an interpolatory procedure; only values of the kernel $K(s,t)$ and the function $y(s)$ are needed at the nodes of the numerical integration formula being used. In terms of accuracy, on the other hand, it will be shown that the Nyström method is optimal with respect to a certain class of approximation procedures. Estimates will be derived for the norm $\|x - z\|$ of the error in a normed linear function space. In this connection, it will be convenient to use operator notation, in terms of which (1.1) may be written in the form

$$(I - \lambda K)x = y \,, \tag{1.2}$$

where I denotes the identity operator, and K the linear integral operator with kernel $K(s,t)$.

2. NUMERICAL INTEGRATION. A rule for numerical integration with nodes $t_1, t_2, \ldots, t_n$ and weights $w_1, w_2, \ldots, w_n$ may be expressed in terms of the linear functional

$$R_n = R_n\begin{pmatrix} t_1 & t_2 & \cdots & t_n \\ w_1 & w_2 & \cdots & w_n \end{pmatrix} \tag{2.1}$$

where

$$R_n[f] = \sum_{j=1}^{n} f(t_j)w_j \,. \tag{2.2}$$

Thus, one has the quadrature formula

$$\int_0^1 f(t)dt = R_n[f] + E_n[f] \tag{2.3}$$

for the integral of a function $f(t)$ over the interval $0 \le t \le 1$, where E_n denotes the (linear) error functional associated with the rule R_n. Attention will be restricted here to quadrature formulas of order m and interpolation type, for which $E_n[f] = 0$ if $f(t)$ is a polynomial in t of degree $m - 1$ or less [10, p. 162]. In addition, it will be assumed that the rules for numerical integration considered have positive weights $w_i > 0$, $i = 1,2,\ldots,n$. This will guarantee that the quadrature formula of interpolation type is convergent in the sense that $\lim_{n\to\infty} E_n[f] = 0$, at least for continuous integrands $f(t)$ [10, p. 186].

Applying the quadrature formula (2.3) to the integral equation (1.1) gives the equivalent equation

$$x(s) - \lambda \sum_{j=1}^{n} K(s,t_j)w_j x(t_j) = y(s) + \lambda E_n[Kx](s) \,, \tag{2.4}$$

where the error term has been moved to the right-hand side. The form of (2.4) suggests that an approximate solution $z(s)$ of the integral equation can be obtained by solving the functional equation

$$z(s) - \lambda \sum_{j=1}^{n} K(s,t_j)w_j z(t_j) = y(s) \,, \tag{2.5}$$

which will be called the Nyström equation resulting from the application of the rule of numerical integration R_n to the Fredholm integral equation (1.1).

3. COLLOCATION. The left-hand side of equation (2.4) involves the values $x(t_j)$ of the solution $x(s)$ of the integral equation (1.1) at the nodes $t_1,t_2,\ldots,t_n$ of the rule of numerical integration R_n. Setting $s = t_i$ in equation (2.4) gives the system of equations

$$x(t_i) - \lambda \sum_{j=1}^{n} K(t_i,t_j)w_j x(t_j) = y(t_i) + \lambda E_n[Kx](t_i) \,, \tag{3.1}$$

$i = 1,2,\ldots,n$, for these values. Approximations z_i to $x(t_i)$ may be obtained by discarding the error term in (3.1) and solving the resulting collocation equations,

$$z_i - \lambda \sum_{j=1}^{n} K(t_i,t_j)w_j z_j = y_i, \qquad i = 1,2,\ldots,n , \tag{3.2}$$

where $y_i = y(t_i)$. Given the rule of numerical integration R_n, setting up and solving the finite linear algebraic system (3.2) (or the corresponding eigenvalue-eigenvector problem) can be carried out readily with the aid of an electronic computer, even for moderately large values of n.

Some other formulations of the system (3.2) may be useful in particular instances. For example, the quantities $\xi_i = w_i z_i$ may be needed for numerical or theoretical purposes. Multiplying the equations (3.2) by $w_1, w_2, \ldots, w_n$ in turn and setting $\eta_i = w_i y_i$ gives the system

$$\xi_i - \lambda \sum_{j=1}^{n} w_i K(t_i,t_j)\xi_j = \eta_i, \qquad i = 1,2,\ldots,n , \tag{3.3}$$

which can be solved directly for $\xi_1, \xi_2, \ldots, \xi_n$. Another case would be that the kernel $K(s,t)$ of the integral equation (1.1) is symmetric, $K(s,t) = K(t,s)$ (or Hermitian, $K(s,t) = \overline{K(t,s)}$), and it is desired to carry this property over to the coefficient matrix of the finite system, which might be particularly convenient when calculating approximate eigenvalues and eigenfunctions. One way to do this is to use numerical integration rules of Chebyshev type with equal weights $w_i = \frac{1}{n}$, $i = 1,2,\ldots,n$ [10, pp. 213-216]. A symmetrization procedure which works for arbitrary rules with positive weights is to multiply the ith equation of (3.2) by $\sqrt{w_i}$. In terms of $\zeta_i = \sqrt{w_i} z_i$, $\theta_i = \sqrt{w_i} y_i$, the resulting system is

$$\zeta_i - \lambda \sum_{j=1}^{n} \sqrt{w_i} K(t_i,t_j) \sqrt{w_j} \zeta_j = \theta_i, \qquad i = 1,2,\ldots,n , \tag{3.4}$$

which has a symmetric (or Hermitian) coefficient matrix if $K(s,t)$ has the corresponding property.

4. INTERPOLATION. In order to keep the system of collocation equations small, Nyström [13] recommended the use of highly accurate rules of numerical integration, such as Gaussian quadrature. Even when using electronic computers,

there are advantages in speed and accuracy to be gained in this way. However, the Gaussian nodes (see [9, p. 288] for a table based on the interval $0 \leq t \leq 1$) are not always the points at which approximate values of the solution of the integral equation are desired. In other applications, what is needed is not just a finite set of values $z_1, z_2, \ldots, z_n$, but rather a function $z(s)$ which is an approximate solution on the entire interval $0 \leq s \leq 1$. These requirements can be met by some method for interpolation (or extrapolation) from the values computed at the collocation points to other points of the interval. Frequently used procedures for interpolation are based on piecewise linear functions, polynomials, or spline functions, which lead to various representations for an approximate solution of (1.1). Nyström's interpolation formula is simply

$$z(s) = y(s) + \lambda \sum_{j=1}^{n} K(s,t_j) w_j z_j \tag{4.1}$$

in terms of the solutions $z_1, z_2, \ldots, z_n$ of the system of equations (3.2). It follows from the Nyström equation (2.5) that $z(t_i) = z_i$, $i = 1,2,\ldots,n$; in fact, (4.1) is the unique solution of (2.5) which interpolates the values computed from (3.2) [4, pp. 88-89]. Formula (4.1) is natural from the standpoint of simplicity and the fact that it makes use of information from the original integral equation (values of $y(s)$ and $K(s,t)$) at all points of the interval, while an arbitrary interpolation formula would only use the values $z_1, z_2, \ldots, z_n$. This suggests that the Nyström method should be fairly accurate, as is usually observed in actual practice.

Other forms of (4.1) may be useful if equations (3.3) or (3.4) are used instead of (3.2). In terms of the solutions $\xi_1, \xi_2, \ldots, \xi_n$ of (3.3), the Nyström interpolation formula becomes simply

$$z(s) = y(s) + \lambda \sum_{j=1}^{n} K(s,t_j) \xi_j \, . \tag{4.2}$$

If equations (3.4) are solved for $\zeta_1, \zeta_2, \ldots, \zeta_n$ instead, then (4.1) may be written

$$z(s) = y(s) + \lambda \sum_{j=1}^{n} K(s,t_j) \sqrt{w_j} \zeta_j \, . \tag{4.3}$$

5. SOME METHODS OF ERROR ESTIMATION. In addition to computational results as furnished by application of Nyström's method, some indication of their reliability is desired in many instances. Such error estimates may range from heuristic to rigorous, and be pointwise, usually for $|x(t_j) - z_j|$, $j = 1,2,\ldots,n$, or global, in which case a bound is given for the norm $\|x - z\|$ of the function $x(s) - z(s)$ in some appropriate space. Among the possible techniques for error estimation are (i) recovery of known solutions; (ii) analysis of the error functional E_n; (iii) use of the theory of collectively compact operator approximation [1]; and (iv) approximation of the integral equation by an equation of finite rank. The method presented in this paper falls into the last category; before going into details, a brief description of the other procedures will be given.

5.1. Recovery of known solutions. For a given kernel $K(s,t)$, it may be that equation (1.1) has known solutions $x(s)$ corresponding to particular choices of the function $y(s)$. In this situation, the approximate solution $z(s)$ obtained by the Nyström method can be compared directly with the exact solution. If the observed accuracy is good, then the numerical results computed for right-hand sides $y(s)$ corresponding to unknown solutions $x(s)$ may be viewed with some confidence. As pointed out by Nyström [13], integral equations with known solutions may be constructed using functions $x(s)$ for which the transformed function $Kx(s)$ can be calculated explicitly.

The computational effort required for this type of error estimation is not particularly great; with a given coefficient matrix, the system (3.2) (or one of the alternative forms (3.3) or (3.4)) may be solved simultaneously for several right-hand sides, corresponding to known and unknown solutions of the integral equation. The accuracy of the approximation obtained for the known solutions may then be used as an indication of reliability of the results calculated for the unknown solutions. It should be emphasized that this procedure is entirely heuristic, it being possible that a certain choice of R_n would work well for some manufactured equations, but not give accurate results for an actual problem. It is comforting to note that the illustrations given by Nyström [13] show good performance of his method applied to boundary-value problems arising in mathematical physics, rather than just to some contrived examples.

5.2. Analysis of the error functional E_n. Essentially, the error in the approximate solution $z(s)$ obtained by the Nyström method is due to neglect of terms involving the error functional E_n; that is, the replacement of (2.4) by (2.5) and using (3.2) instead of (3.1). (More precisely, this is the truncation error of the method; in this discussion, roundoff error in the actual computation and errors in the data $K(s,t)$ and $y(s)$ will be ignored.) An expression for the truncation error $x(s) - z(s)$ will now be obtained. From (2.4) and (2.5),

$$x(s) - z(s) = \lambda\left\{E_n[Kx](s) + \sum_{j=1}^{n} K(s,t_j)w_j[x(t_j) - z(t_j)]\right\} . \tag{5.1}$$

For simplicity of notation, set

$$E_n(s) = E_n[Kx](s), \quad 0 \le s \le 1 . \tag{5.2}$$

The errors $x(t_i) - z(t_i)$ at the collocation points satisfy the linear system of equations

$$[x(t_i) - z(t_i)] - \lambda \sum_{j=1}^{n} K(t_i,t_j)w_j[x(t_j) - z(t_j)] = \lambda E_n(t_i) , \tag{5.3}$$

$i = 1,2,\ldots,n$, by (3.1) and (3.2). Suppose that the coefficient matrix A of (5.3),

$$A = (\delta_{ij} - \lambda K(t_i,t_j)w_j) , \tag{5.4}$$

where δ_{ij} is the Kronecker delta ($\delta_{ij} = 0$ if $i \neq j$, $\delta_{ii} = 1$), has the inverse

$$B = (\beta_{ij}) = A^{-1} . \tag{5.5}$$

In terms of the coefficients of B, the solutions of (5.3) may be written

$$x(t_i) - z(t_i) = \lambda \sum_{j=1}^{n} \beta_{ij}E_n(t_j), \quad i = 1,2,\ldots,n , \tag{5.6}$$

and (5.1) becomes

$$x(s) - z(s) = \lambda\left\{E_n(s) + \lambda \sum_{j=1}^{n}\sum_{k=1}^{n} K(s,t_j)w_j\beta_{jk}E_n(t_k)\right\} . \tag{5.7}$$

On the basis of some assumption about the magnitude of $E_n(s) = E_n[Kx](s)$, for example, $|E_n(s)| \leq M$, $0 \leq s \leq 1$, (5.6) may be used to derive pointwise error bounds at the collocation points $t_1, t_2, \ldots, t_n$, and (5.7) will furnish a global error estimate, or pointwise bounds at points other than the nodes of the rule of numerical integration.

If the integrand $f(t)$ is smooth enough, then the error term $E_n[f]$ of a formula of order m and interpolation type can be expressed as

$$E_n[f] = C_m(n) \cdot f^{(m)}(\tau_n), \qquad 0 < \tau_n < 1, \tag{5.8}$$

where $C_m(n)$ is a known function of n for which $\lim_{n\to\infty} C_m(n) = 0$ in the case of convergent rules, and the point τ_n is unknown [9, pp. 108-116; 5, pp. 217-223]. Thus, the problem of bounding $E[Kx](s)$ can be reduced to estimating $\frac{\partial^m}{\partial t^m} K(s,t)x(t)$. This has the obvious drawback of requiring guesses for the size of derivatives of the unknown function $x(t)$. Another hindrance to this method for obtaining error bounds is the possibility that the integrand $K(s,t)x(t)$ of the integral transform has a low degree of continuity in t, as occurs, for example, in the important applications in which $K(s,t)$ is a Green's function. Thus, even when $x(t)$ is known to be fairly smooth, one may have to settle for a small value of m in (5.8), and a correspondingly large majorant function $C_m(n)$ [5, pp. 257-260].

To avoid some of these difficulties, use may be made of a device which gives good results in many cases, although it is not completely rigorous. If the same quadrature formula is applied with two different values of n, say p and q, then

$$E_p[f]/E_q[f] = [C_m(p)/C_m(q)] \frac{f^{(m)}(\tau_p)}{f^{(m)}(\tau_q)} . \tag{5.9}$$

Assuming that $f^{(m)}(\tau_p)$ and $f^{(m)}(\tau_q)$ are approximately equal, the ratio of the errors $E_p[f]$ to $E_q[f]$ can be estimated by the computable value of $C_m(p)/C_m(q)$. For example, commonly

$$C_m(n) = c\left(\frac{1}{n}\right)^m , \tag{5.10}$$

where c is a known constant, and

(5.11) $$C_m(q)/C_m(p) = (p/q)^m .$$

A frequent choice is $q = 2p$, which leads to the rule of thumb that doubling the nodes of the rule of numerical integration reduces the truncation error by a factor of 2^{-m} or, in other words, gives m additional binary digits of accuracy. To apply this or other estimates obtained from (5.11), one may simply compare the number of digits which agree in the two answers computed, or apply a suitable extrapolation formula [11, pp. 231-237]. In order to use this method of error estimation, the system of equations (3.2) (or one of its equivalent forms) must be set up and solved for at least two values of n. This will involve considerably more labor than the corresponding operation for integration of a function of a single variable. In addition, more than two numerical solutions of the integral equation for different values of n may be required to provide assurance that a theoretical convergence rate as predicted, for example, by (5.10), is actually being observed empirically.

5.3. Collectively compact operator approximation theory. Error estimates for the Nyström method can be obtained on the basis of the theory of collectively compact operator approximation developed by Anselone [1]. As this application of the general theory is also explained clearly and in detail in the book by Atkinson [4, pp. 88-104], only the essential features of this approach will be summarized here. The setting for this technique, which provides rigorous results, is some Banach space of functions $x(s)$ defined on the interval $0 \leq s \leq 1$. Usual examples are the space $C[0,1]$ of continuous functions with the maximum norm, or $L_2[0,1]$, the Hilbert space of functions with Lebesgue integrable squares and the norm defined in the ordinary way as the square root of the integral of the square of the function. The norm of an element x of the space will be denoted in the customary fashion by $\|x\|$. In order to avoid possible confusion, a different notation will be used for the operator norm of a linear operator L that maps the space into itself, which is defined by

(5.12) $$M(L) = \sup_{\|x\|=1} \|Lx\| .$$

(Often $\|L\|$ is written for $M(L)$, as in the literature cited [1,4], it being clear from the context whether the element or the operator norm is meant.) For example, if K is a linear integral operator in the space $C[0,1]$ with

continuous kernel $K(s,t)$, then

$$M(K) = \max_{0 \le s \le 1} \int_0^1 |K(s,t)|dt \ . \tag{5.13}$$

In $L_2[0,1]$, if $K(s,t)$ is symmetric and positive definite with eigenvalues $0 < \lambda_1 \le \lambda_2 \le \ldots$, then

$$M(K) = \frac{1}{\lambda_1} \ . \tag{5.14}$$

For arbitrary integral operators K in $L_2[0,1]$, one has

$$M(K) \le \left(\int_0^1 \int_0^1 |K(s,t)|^2 dsdt \right)^{\frac{1}{2}} , \tag{5.15}$$

which is computationally more tractable than expressions of the form (5.14); however, the right-hand side of inequality (5.15) may overestimate $M(K)$ grossly.

Fundamental to the Anselone theory is the definition of the numerical integration operator Q_n by

$$Q_n x = R_n[Kx] \ , \tag{5.16}$$

or

$$Q_n x(s) = \sum_{j=1}^{n} K(s,t_j) w_j x(t_j) \ , \tag{5.17}$$

for the given nodes and weights of the rule of numerical integration considered. A straightforward application of approximation theory is not possible at this point, as the operator Q_n does not approximate K in the operator norm; in fact [4, p. 90]

$$M(K - Q_n) \ge M(K) \tag{5.18}$$

independently of the value of n. However, provided that the set of numerical integration operators $\{Q_n\}$ has a technical property known as collective compactness [1, pp. 3-4], error estimates may be obtained in terms of the operator norm of

(5.19) $$\Delta_n = (Q_n - K)K = Q_nK - K^2$$

which has the kernel

(5.20) $$\Delta_n(s,t) = \sum_{j=1}^{n} K(s,t_j)w_jK(t_j,t) - \int_0^1 K(s,r)K(r,t)dr \ .$$

In this formulation, the Nyström approximation $z(s)$ is obtained by solving the equation

(5.21) $$(I - \lambda Q_n)z = y \ .$$

In the nonsingular case, $(I - \lambda Q_n)^{-1}$ exists, and

(5.22) $$z = (I - \lambda Q_n)^{-1}y \ .$$

Furthermore [1, pp. 11-12], if

(5.23) $$\delta_n = M((I - \lambda Q_n)^{-1})M(\lambda^2\Delta_n) < 1 \ ,$$

then $(I - \lambda K)^{-1}$ exists (that is, the original integral equation (1.1) has a unique solution $x(s)$), and

(5.24) $$\|z - x\| \leq \frac{M((I - \lambda Q_n)^{-1})\,\|\lambda Q_n y - \lambda K y\| + \delta_n\|z\|}{1 - \delta_n} \ .$$

As the property of collective compactness of the set of operators $\{Q_n\}$ and the fact that $\lim_{n\to\infty} \delta_n = 0$ (including a rate of convergence) may be verified fairly easily for convergent numerical integration rules commonly used in practice, inequality (5.24) provides an error bound which is both rigorous and computable. Some difficulty may be encountered in obtaining precise bounds, particularly if the transformed function Ky and the kernel of the iterated operator K^2 cannot be calculated explicitly; however, satisfactory overestimates for the quantities on the right-hand side of inequality (5.24) can usually be obtained. It is worth noting that if $K(s,t)$ is symmetric (or Hermitian), then it follows directly from (5.20) that $\Delta_n(s,t)$ has the same property. This may be useful in $L_2[0,1]$, as sharper bounds for the operator norm $M(\Delta_n)$ than given by (5.15) may be computable in terms of approximate eigenvalues of $\Delta_n(s,t)$.

In order for the error bound given by (5.24) to be small, the function $Q_n y$ must be a good approximation to the transformed function Ky, as measured by the norm of their difference $\|Q_n y - Ky\|$, and the finite rank operator $Q_n K$ must be close to the iterated operator K^2 in the operator norm topology. It will be shown below that the Nyström method can also be formulated in terms of finite rank operators K_n which approximate K directly, leading to simpler error bounds than (5.24).

5.4. Approximation by equations of finite rank. A standard procedure in the numerical analysis of the integral equation (1.1) is to approximate it by an equation

$$z(s) - \lambda \int_0^1 L(s,t)z(t)dt = y(s), \qquad 0 \leq s \leq 1 , \tag{5.25}$$

which can be solved for $z(s)$ [15]. Here, one looks for estimates of $\|x - z\|$ in terms of $M(K - L)$, $M((I - \lambda L)^{-1})$, and perhaps $\|z\|$, it being assumed that these values or upper bounds for them are computable. For example, in the nonsingular case that $(I - \lambda K)^{-1}$ and $(I - \lambda L)^{-1}$ exist, it is easy to verify the identity

$$(I - \lambda K)^{-1} - (I - \lambda L)^{-1} = \lambda(I - \lambda L)^{-1}(K - L)(I - \lambda K)^{-1} . \tag{5.26}$$

Operating on y with both sides of (5.26) gives

$$x - z = \lambda(I - \lambda L)^{-1}(K - L)x , \tag{5.27}$$

and thus

$$\frac{\|x - z\|}{\|x\|} \leq |\lambda| M((I - \lambda L)^{-1}) M(K - L) \tag{5.28}$$

gives a computable bound for the relative error $\|x - z\|/\|x\|$. By symmetry,

$$\frac{\|x - z\|}{\|z\|} \leq |\lambda| M((I - \lambda K)^{-1}) M(K - L) , \tag{5.29}$$

which requires an estimate for $M((I - \lambda K)^{-1})$ to be computable. However, if

$$\theta(K - L) = M((I - \lambda L)^{-1}) M(K - L) < \frac{1}{|\lambda|} , \tag{5.30}$$

then $(I - \lambda K)^{-1}$ exists, and

$$M((I - \lambda K)^{-1}) \leq \frac{M((I - \lambda L)^{-1})}{1 - |\lambda|\theta(K - L)} \tag{5.31}$$

[16, pp. 50-57]. Substitution of (5.31) into (5.29) yields

$$\|x - z\| \leq \frac{|\lambda|\theta(K - L)}{1 - |\lambda|\theta(K - L)} \|z\| , \tag{5.32}$$

which provides a computable bound for the absolute error $\|x - z\|$. The error bounds (5.28) and (5.32) are simpler than (5.24).

The error analysis of the Nyström method conducted here will be based on approximation of (1.1) by equations (5.25) with kernels of finite rank, that is, $L(s,t) = F_n(s,t)$, where

$$F_n(s,t) = \sum_{j=1}^{n} u_j(s)v_j(t) , \tag{5.33}$$

and $\{u_1(s),u_2(s),\ldots,u_n(s)\}$, $\{v_1(t),v_2(t),\ldots,v_n(t)\}$ are sets of linearly independent functions. More particularly, the choice $u_j(s) = K(s,t_j)w_j$, $j = 1,2,\ldots,n$ will be made for the Nyström method, which leads to approximate kernels of the form

$$K_n(s,t) = \sum_{j=1}^{n} K(s,t_j)w_jv_j(t) , \tag{5.34}$$

and error bounds corresponding to (5.28) and (5.32) with $L = K_n$.

Error analysis of a number of methods for the numerical solution of integral equations can be carried out on the basis of approximation by finite rank equations with kernels (5.33) or (5.34), including collocation-interpolation procedures which have the Nyström method as a special case. The papers by Phillips [14], Noble [12], and Sloan [17] describe various methods for which the present approach is suitable. One of the principal results of this paper is to show that the Nyström method is of optimal accuracy with respect to a class of approximations of the kernel $K(s,t)$ by finite rank kernels of the form (5.34).

6. SOLUTION OF FINITE RANK EQUATIONS. In 1907, Goursat [8] gave the recipe for solving Fredholm integral equations of second kind with finite rank kernels (5.33), that is, equations of the form

$$z(s) - \lambda \int_0^1 \sum_{j=1}^{n} u_j(s)v_j(t)z(t)dt = y(s) . \tag{6.1}$$

In terms of the ordinary inner product

$$\langle f,g\rangle = \int_0^1 f(t)g(t)dt \ , \tag{6.2}$$

which will be assumed to be defined whether or not the function space considered is a Hilbert space, equation (6.1) may be written

$$z(s) - \lambda \sum_{j=1}^{n} u_j(s)\langle v_j,z\rangle = y(s) \ . \tag{6.3}$$

By taking the inner products of equation (6.3) with $v_1(s),v_2(s),\ldots,v_n(s)$ in turn, one obtains the finite system of linear algebraic equations

$$\langle v_i,z\rangle - \lambda \sum_{j=1}^{n} \langle v_i,u_j\rangle\langle v_j,z\rangle = \langle v_i,y\rangle \ , \tag{6.4}$$

$i = 1,2,\ldots,n,$ for the unknown inner products $\langle v_1,z\rangle,\langle v_2,z\rangle,\ldots,\langle v_n,z\rangle$. In terms of the solutions of (6.4), the solution of (6.3) and hence of the finite rank integral equation (6.1) may be written as

$$z(s) = y(s) + \lambda \sum_{j=1}^{n} u_j(s)\langle v_j,z\rangle \ . \tag{6.5}$$

There is an obvious similarity between (6.3) and the Nyström equation (2.5), the linear algebraic system (6.4) and the collocation equations (3.2), and finally between the solution (6.5) and the interpolation formula (4.1). These similarities will be exploited below to obtain error bounds. The principal difference between the two sets of equations is that the Nyström method makes use of interpolation data, that is, values of the functions involved at specified points, while the inner products (6.2) involved in the Goursat equations (6.3)-(6.5) require values of the functions considered on the entire interval $0 \leq t \leq 1$ (except possibly for sets of measure zero), which will be called approximation data.

In the nonsingular case, it is customary to express the solution of the integral equation (1.1) as

$$x(s) = y(s) + \lambda \int_0^1 \Gamma(s,t;\lambda)y(t)dt \ , \tag{6.6}$$

where $\Gamma(s,t;\lambda)$ is called the resolvent kernel of the kernel $K(s,t)$. The resolvent kernel $G_n(s,t;\lambda)$ of the finite rank kernel $F_n(s,t)$ defined by (5.33) may be expressed in terms of the inverse matrix $B = (\beta_{ij}) = A^{-1}$ of the coefficient matrix

$$A = (\delta_{ij} - \lambda\langle v_i, u_j\rangle) \tag{6.7}$$

of the linear system (6.4) as

$$G_n(s,t;\lambda) = \sum_{j=1}^{n}\sum_{k=1}^{n} u_j(s)\beta_{jk}v_k(t) \ . \tag{6.8}$$

Equation (6.8) follows directly by substitution of the solutions

$$\langle v_j, z\rangle = \sum_{j=1}^{n} \beta_{jk}\langle v_k, y\rangle, \qquad j = 1,2,\ldots,n \ , \tag{6.9}$$

of the system (6.4) into (6.5). There are a number of ways to express the resolvent kernel (6.8) as a kernel of rank n in the form (5.33). For example, defining

$$V_j(t) = \sum_{k=1}^{n} \beta_{jk}v_k(t), \qquad j = 1,2,\ldots,n \ , \tag{6.10}$$

one obtains

$$G_n(s,t;\lambda) = \sum_{j=1}^{n} u_j(s)V_j(t) \ . \tag{6.11}$$

Other expressions may be obtained by summing over j first, or by manipulation of the matrix B such as reduction to a canonical form, LU or singular value decomposition, etc., in order to write the double sum in (6.8) as a single summation of products of functions $U_j(s)V_j(t)$, $j = 1,2,\ldots,n$. In some applications, one of these alternative forms may be more useful than (6.11).

In the singular case, λ is an eigenvalue of the kernel $F_n(s,t)$, and the homogeneous system

$$\langle v_i, z\rangle - \lambda \sum_{j=1}^{n} \langle v_i, u_j\rangle\langle v_j, z\rangle = 0, \qquad i = 1,2,\ldots,n \ , \tag{6.12}$$

corresponding to (6.4) has d linearly independent sets of solutions $\langle v_1,z\rangle_k,\langle v_2,z\rangle_k,\ldots,\langle v_n,z\rangle_k$, $k = 1,2,\ldots,d$, where $d = n - r$ is called the defect of the matrix A of coefficients of (6.12), r being its rank. In terms of these solutions of (6.12), d linearly independent (right) eigenfunctions $z_1(s),z_2(s),\ldots,z_d(s)$ of $F_n(s,t)$ are

$$z_k(s) = \lambda \sum_{j=1}^{n} u_j(s)\langle v_j,z\rangle_k, \qquad k = 1,2,\ldots,d . \tag{6.13}$$

Any given right eigenfunction of $F_n(s,t)$ may be expressed as a linear combination of the functions (6.13).

By the Fredholm theory [6], if λ is an eigenvalue, then the transposed homogeneous system

$$\langle p,u_j\rangle - \lambda \sum_{i=1}^{n} \langle p,u_i\rangle\langle v_i,u_j\rangle = 0, \qquad j = 1,2,\ldots,n \tag{6.14}$$

has d linearly independent sets of solutions $\langle p,u_1\rangle_k,\langle p,u_2\rangle_k,\ldots,\langle p,u_n\rangle_k$, $k = 1,2,\ldots,d$, corresponding to which

$$p_k(t) = \lambda \sum_{i=1}^{n} \langle p,u_i\rangle_k v_i(t), \qquad k = 1,2,\ldots,d , \tag{6.15}$$

form a complete set of linearly independent left eigenfunctions of the kernel $F_n(s,t)$; that is, any solution $p(t)$ of the transposed homogeneous integral equation

$$p(t) - \lambda \int_0^1 p(s) \sum_{i=1}^{n} u_i(s)v_i(t)dt = 0, \qquad 0 \le t \le 1 , \tag{6.16}$$

can be expressed as linear combinations of the functions (6.15). In this case, the inhomogeneous system (6.4) has solutions if and only if

$$\sum_{i=1}^{n} \langle p,u_i\rangle_k\langle v_i,y\rangle = 0, \qquad k = 1,2,\ldots,d , \tag{6.17}$$

or, in integral form,

$$\int_0^1 \left[\sum_{i=1}^{n} \langle p,u_i\rangle_k v_i(t)\right] y(t)dt = \int_0^1 p_k(t)y(t)dt = 0 , \tag{6.18}$$

$k = 1,2,\ldots,d$. In other words, the necessary and sufficient condition that (6.1) be solvable in case λ is an eigenvalue is that $y(t)$ be orthogonal to all solutions of the transposed homogeneous integral equation (6.16). Assuming that this holds, given a <u>particular solution</u> $z_0(s)$ of (6.1), the <u>general solution</u> may be written

$$z(s) = z_0(s) + \sum_{k=1}^{d} \alpha_k z_k(s) , \tag{6.19}$$

where $\alpha_1,\alpha_2,\ldots,\alpha_d$ are arbitrary.

Thus, approximation of the integral equation (1.1) by a finite rank equation (6.1) provides a way to construct an approximate solution and resolvent kernel in the nonsingular case, and approximate eigenvalues and left and right eigenfunctions from the corresponding homogeneous equations. To handle the inhomogeneous equation in the singular case, it may be necessary to approximate also the right-hand side of (1.1) by a function which satisfies (6.18).

<u>7. IDENTIFICATION OF THE NYSTRÖM METHOD WITH FINITE RANK APPROXIMATIONS</u>. On the basis of the above, it can be seen that the results of the Nyström method are identical to those obtained by approximation of (1.1) by finite rank equations with kernels $K_n(s,t)$ of the form (5.34), provided that the functions $v_1(t),v_2(t),\ldots,v_n(t)$ are chosen to satisfy the $n^2 + n$ conditions

$$\langle v_i,K_j\rangle = \int_0^1 v_i(t)K(t,t_j)dt = K(t_i,t_j) , \tag{7.1}$$

$i,j = 1,2,\ldots,n$, where $K_j(t) = K(t,t_j)$, and

$$\langle v_i,y\rangle = \int_0^1 v_i(t)y(t)dt = y(t_i) , \tag{7.2}$$

$i = 1,2,\ldots,n$.

There are many ways to find functions $v_i(t)$, $i = 1,2,\ldots,n$, which satisfy (7.1) and (7.2). For example, suppose that a reproducing kernel $R(s,t)$ [3; 7, pp. 146-160] is known for a space containing the functions $K_j(s)$, $j = 1,2,\ldots,n$, (or, more generally, the functions $K_t(s) = K(s,t)$ for $0 \le t \le 1$) and $y(s)$, then one may take

$$v_i(t) = R(t_i,t), \quad i = 1,2,\ldots,n . \tag{7.3}$$

A less exotic way to determine suitable $v_1(t), v_2(t), \ldots, v_n(t)$ would be as linear combinations of functions for which the integrals in (7.1) and (7.2) can be calculated explicitly. The space Ω of functions $\omega(t)$ orthogonal to $K_1(t), K_2(t), \ldots, K_n(t)$, and $y(t)$ is infinite, with codimension at most $n + 1$, and if $v_i(t)$, $i = 1,2,\ldots,n$, satisfy the linear constraints (7.1) and (7.2), then so do the functions $v_i(t) + \omega_i(t)$ for arbitrary $\omega_i \in \Omega$, $i = 1,2,\ldots,n$. Thus, given the integral equation (1.1) and the rule of numerical integration R_n, the class of finite rank kernels for which (7.1) and (7.2) hold will be denoted by

$$\{K_n(s,t)\} = \{K(s,t), R_n, y(s)\} . \tag{7.4}$$

A special notation will be used for the homogeneous case $y(s) \equiv 0$, namely

$$\{K_n(s,t)\}_0 = \{K(s,t), R_n, 0\} , \tag{7.5}$$

for which $K_n(s,t)$ has to satisfy only the conditions (7.1). The notations $\{K_n\} = \{K, R_n, y\}$ and $\{K_n\}_0 = \{K, R_n, 0\}$ will be used for the corresponding classes of finite rank linear integral operators K_n with kernels $K_n(s,t)$.

8. THE NYSTRÖM CONSTANT AND ERROR BOUNDS. As approximation of the integral equation (1.1) by any finite rank equation

$$z(s) - \lambda \int_0^1 K_n(s,t) z(t) dt = y(s), \qquad 0 \le s \le 1 , \tag{8.1}$$

with kernel $K_n(s,t)$ chosen from $\{K_n(s,t)\}$ gives precisely the same results as the Nyström method with the rule of numerical integration R_n, the accuracy of the Nyström solution can be studied in terms of how well the integral operator K can be approximated by finite rank operators K_n with kernels $K_n(s,t)$ belonging to $\{K_n(s,t)\}$. To this end, define

$$\nu = \nu(K, R_n, y) = \inf_{K_n \in \{K_n\}} M(K - K_n) \tag{8.2}$$

to be the Nyström constant for the given integral equation and rule of numerical integration. In the homogeneous case, the notation

$$\nu_0 = \inf_{K_n \in \{K_n\}_0} M(K - K_n) \tag{8.3}$$

will be used. As the constraints (7.2) are automatically satisfied in the homogeneous case, one has $\{K_n\} = \{K,R_n,y\} \subset \{K_n\}_0$ for arbitrary y, and thus

$$\nu_0 \leq \nu = \nu(K,R_n,y) \ . \tag{8.4}$$

The minimal Nyström constant ν_0 is appropriate for the eigenvalue-eigenfunction problem for $K(s,t)$, and estimation of the accuracy with which the resolvent kernel $\Gamma(s,t;\lambda)$ of $K(s,t)$ can be approximated by resolvent kernels

$$\Gamma_n(s,t;\lambda) = \sum_{j=1}^{n} \sum_{k=1}^{n} K(s,t_j) w_j \beta_{jk} v_k(t) \tag{8.5}$$

of $K_n(s,t)$ belonging to $\{K_n(s,t)\}_0$, as the function $y(s)$ is not involved in this calculation.

In the nonsingular case, for

$$B_0 = M((I - \lambda K)^{-1}) \ , \tag{8.6}$$

the inequality

$$\frac{\|x - z\|}{\|z\|} \leq |\lambda| \nu B_0 \tag{8.7}$$

follows from (5.29) and (8.2). Hence, the accuracy of the Nyström method is optimal with respect to approximation of the integral operator K by finite rank operators K_n belonging to the class $\{K_n\}$. This supports the observation that good results are usually obtained in practice, as in the examples cited by Nyström [13] and Atkinson [4, pp. 102-104].

It is also interesting to note that either $(I - \lambda K_n)^{-1}$ exists for all $K_n \in \{K_n\}_0$ or λ is an eigenvalue of all the kernels $K_n(s,t)$ belonging to $\{K_n(s,t)\}_0$. This is true because the invertibility of $I - \lambda K_n$ is equivalent by construction to that of the matrix A given by (5.4) for all kernels $K_n(s,t) \in \{K_n(s,t)\}_0$ (and hence for all $K_n(s,t) \in \{K(s,t),R_n,y(s)\}$ for arbitrary $y(s)$). If $A^{-1} = B$ exists, then the classes $\{K_n\}_0$ and $\{K_n\}$ are said to be nonsingular. A sufficient condition for nonsingularity of these classes is that $(I - \lambda K)^{-1}$ exist, and

$$|\lambda| \nu_0 B_0 < 1 \ . \tag{8.8}$$

The Nyström method determines the approximate solution $z(s)$ uniquely, but not the approximate resolvent kernel (8.5), as the functions $v_1(t), v_2(t), \ldots, v_n(t)$ are only required to satisfy (7.1). In terms of the resolvent operators Γ of K and Γ_n of K_n, the identity (5.26) may be written

$$\Gamma - \Gamma_n = (I + \lambda\Gamma_n)(K - K_n)(I + \lambda\Gamma) \tag{8.9}$$

in the nonsingular case. Supposing that (8.8) holds, choose $\varepsilon > 0$ and $K_n^{\varepsilon} \in \{K_n\}_0$ such that

$$M(K - K_n^{\varepsilon}) \leq \nu_0 + \varepsilon < \frac{1}{|\lambda| B_0} . \tag{8.10}$$

Then,

$$M(I + \lambda\Gamma_n^{\varepsilon}) = M((I - \lambda K_n^{\varepsilon})^{-1}) \leq \frac{B_0}{1 - |\lambda|(\nu_0 + \varepsilon)B_0} , \tag{8.11}$$

and, from (8.9),

$$M(\Gamma - \Gamma_n^{\varepsilon}) \leq \frac{(\nu_0 + \varepsilon)B_0^2}{1 - |\lambda|(\nu_0 + \varepsilon)B_0} . \tag{8.12}$$

Thus, the distance γ_0 from Γ to the class $\{\Gamma_n\}_0$ of resolvent operators of the finite rank operators K_n belonging to $\{K_n\}_0$ satisfies

$$\gamma_0 = \inf_{\Gamma_n \in \{\Gamma_n\}_0} M(\Gamma - \Gamma_n) \leq \frac{\nu_0 B_0^2}{1 - |\lambda|\nu_0 B_0} \tag{8.13}$$

in the operator norm. The use of a resolvent kernel (8.5) selected from $\{\Gamma_n(s,t;\lambda)\}_0$ to solve equations (8.1) for various choices of $y(s)$ actually amounts to a modification of the Nyström method by the use of approximation data

$$y_i = \langle v_i, y \rangle, \quad i = 1,2,\ldots,n , \tag{8.14}$$

on the right-hand side of (3.2) instead of the interpolation data $y_i = y(t_i)$, $i = 1,2,\ldots,n$. The two sets of data will be identical, of course, if $\Gamma_n(s,t;\lambda)$ is the resolvent kernel of a kernel $K_n(s,t)$ belonging to the class $\{K_n(s,t)\} = \{K(s,t), R_n, y(s)\}$.

As noted above, the Nyström method may also be applied to the homogeneous integral equation

$$x(s) - \lambda \int_0^1 K(s,t)x(t)dt = 0, \qquad 0 \le s \le 1 , \tag{8.15}$$

to obtain approximate eigenvalues and eigenfunctions of $K(s,t)$ by solving the homogeneous system

$$z_i - \lambda \sum_{j=1}^{n} K(t_i,t_j)w_j z_j = 0, \qquad i = 1,2,\ldots,n , \tag{8.16}$$

and using the interpolation formula

$$z(s) = \lambda \sum_{j=1}^{n} K(s,t_j)w_j z_j \tag{8.17}$$

for the corresponding right eigenfunctions. By construction, all the kernels $K_n(s,t)$ in the class $\{K_n(s,t)\}_0$ have the same sets of eigenvalues $\lambda_1^{(n)},\lambda_2^{(n)},\ldots,\lambda_n^{(n)}$ and corresponding right eigenfunctions $z_1^{(n)}(s)$, $z_2^{(n)}(s),\ldots,z_n^{(n)}(s)$, including the possibilities of multiplicity of eigenvalues and the existence of generalized eigenfunctions. This is because the functions $v_1(t),v_2(t),\ldots,v_n(t)$ do not enter into these calculations explicitly.

It is also possible to obtain $O(\nu_0)$ error estimates for $|\lambda - \lambda^{(n)}|$ and $\|z - z^{(n)}\|$, for example, by setting up the eigenvalue-eigenfunction problem as a nonlinear operator equation, and use of the Kantorovich theorem [2] or some other technique [12, pp. 228-231]. As these are outside the scope of this paper, explicit error bounds will not be given here.

The calculation of approximate left eigenfunctions $p_1^{(n)}(t),p_2^{(n)}(t)$, $\ldots,p_n^{(n)}(t)$ is carried out on the basis of the transposed homogeneous system

$$p_j - \lambda \sum_{i=1}^{n} p_i K(t_i,t_j)w_j = 0, \qquad j = 1,2,\ldots,n , \tag{8.18}$$

and the interpolation formula

$$p(t) = \lambda \sum_{i=1}^{n} p_i v_i(t) \tag{8.19}$$

for the left eigenfunctions of $K_n(s,t)$, which are hence linear combinations of $v_1(t), v_2(t), \ldots, v_n(t)$, and depend on the particular kernel chosen, unlike (8.17). In this case, the inhomogeneous equation (8.1) will be solvable without further approximation of $y(s)$ if

$$\langle v_i, y \rangle = 0, \quad i = 1,2,\ldots,n , \tag{8.20}$$

or, if $K_n(s,t) \in \{K(s,t), R_n, y(s)\}$, one has

$$y(t_i) = 0, \quad i = 1,2,\ldots,n . \tag{8.21}$$

Conditions (8.20) and (8.21) are sufficient that $\langle p,y \rangle = 0$ for all solutions $p(t)$ of the transposed homogeneous equation

$$p(t) - \lambda \int_0^1 p(s)K_n(s,t)ds = 0, \quad 0 \leq t \leq 1 , \tag{8.22}$$

for the eigenvalue λ.

9. COMPUTABLE ERROR BOUNDS. The error bounds depending on the Nyström constant given by (8.7) or (8.13), for example, are theoretical in character, as the unknown (but fixed) quantity B_0 is involved. However, for some given operator $K_n \in \{K_n\}$, the quantities

$$\nu_n = M(K - K_n), \quad B_n = M((I - \lambda K_n)^{-1}) \tag{9.1}$$

may be computed in the nonsingular case, and, if $|\lambda|\nu_n B_n < 1$, then $(I - \lambda K)^{-1}$ exists, and B_0 may be estimated by

$$B_0 \leq \frac{B_n}{1 - |\lambda|\nu_n B_n} , \tag{9.2}$$

as follows from (5.31). The known value ν_n may, of course, be used as an upper bound for ν or ν_0. One has also

$$\frac{\|x - z\|}{\|x\|} \leq |\lambda|\nu_n B_n , \tag{9.3}$$

directly from (5.28).

Finding the Nyström constant ν (or ν_0) requires the solution of a nonlinear optimization problem subject to linear constraints (7.1) and (7.2) (or (7.1) only). This could be of comparable or greater difficulty than the

original task of solving the linear integral equation (1.1). In some circumstances, however, it might be desired to estimate the Nyström constant directly, rather than use a particular kernel $K_n(s,t)$ belonging to $\{K_n(s,t)\}$ or $\{K_n(s,t)\}_0$. It would also be useful if the estimation process furnishes information concerning good choices of the functions $v_1(t),v_2(t),\ldots,v_n(t)$. This can be done in $L_2[0,1]$ by classical calculus of variations technique applied to the upper bound (5.15) for the functional $M(K)$. Setting $v = (v_1,v_2,\ldots,v_n)$, this approximating optimization problem may be posed as

$$(9.4) \qquad \textit{minimize } F[v] = \int_0^1 \int_0^1 |K(s,t) - K_n(s,t)|^2 ds dt ,$$

with v subject to (7.1) and (7.2). Here, the fully constrained case will be treated in detail, with appropriate modifications indicated for the problem in $\{K_n(s,t)\}_0$ corresponding to only the constraints (7.1), and also the unconstrained problem of minimizing $F[v]$ over all v with $v_i \in L_2[0,1]$, $i = 1,2,\ldots,n$. Introducing Lagrange multipliers Λ_{ij} for the constraints (7.1) and Λ_i for (7.2), the kernels of the Gâteaux derivatives of the functional

$$(9.5) \qquad \Phi[v] = F[v] + \sum_{i=1}^n \sum_{j=1}^n \Lambda_{ij}(\langle v_i,K_j\rangle - K(t_i,t_j)) + \sum_{i=1}^n \Lambda_i(\langle v_i,t\rangle - y(t_i))$$

with respect to $v_1,v_2,\ldots,v_n$ will vanish at the solution of the optimization problem. This necessary condition is equivalent to the system of equations

$$(9.6) \qquad -2\int_0^1 K_i(s)K(s,t)w_i ds + 2\sum_{j=1}^n \left(\int_0^1 K_i(s)K_j(s)w_i w_j ds\right)v_j(t) +$$

$$+ \sum_{j=1}^n \Lambda_{ij}K_j(t) + \Lambda_i y(t) = 0, \qquad i = 1,2,\ldots,n .$$

This can be simplified somewhat by writing

$$(9.7) \qquad \psi_i(t) = \int_0^1 K_i(s)K(s,t)w_i ds, \qquad i = 1,2,\ldots,n ,$$

and

(9.8) $$c_{ij} = \int_0^1 K_i(s)K_j(s)w_i w_j ds, \qquad i,j = 1,2,\ldots,n .$$

Thus, (9.6) becomes

(9.9) $$\sum_{j=1}^{n} c_{ij}v_j(t) = \psi_i(t) - \frac{1}{2}\sum_{j=1}^{n} \Lambda_{ij}K_j(t) - \frac{1}{2}\Lambda_i y(t) ,$$

$i = 1,2,\ldots,n$. If now for $C = (c_{ij})$, one has that $C^{-1} = D = (d_{ij})$ exists, then

(9.10) $$v_i(t) = \sum_{j=1}^{n} d_{ij}\psi_j(t) - \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} d_{ij}\Lambda_{jk}K_k(t) -$$

$$- \frac{1}{2}\sum_{j=1}^{n} d_{ij}\Lambda_j y(t), \qquad i = 1,2,\ldots,n ,$$

which expresses the functions $v_1(t),v_2(t),\ldots,v_n(t)$ as linear combinations of the unknown Lagrange multipliers, with known functions as coefficients. The unconstrained solutions $\hat{v}_i(t)$ of (9.4), $i = 1,2,\ldots,n$, may be read directly from (9.10) as

(9.11) $$\hat{v}_i(t) = \sum_{j=1}^{n} d_{ij}\psi_j(t) = \sum_{j=1}^{n} d_{ij}\int_0^1 K_j(s)K(s,t)w_j ds ,$$

which does not involve the Lagrange multipliers. The kernel

(9.12) $$\hat{K}_n(s,t) = \sum_{j=1}^{n} K(s,t_j)w_j\hat{v}_j(t)$$

is the best approximation to $K(s,t)$ in the sense of (9.4), not necessarily in the operator norm. To use (9.12), one must set up and solve the linear system (6.4), which requires approximation data on both sides, and gives results which will differ from the Nyström solution, except in special cases. Linear systems of equations for the Lagrange multipliers may be obtained by substituting (9.10) into the constraint equations (7.1) and (7.2). From (7.1),

(9.13) $$K(t_h,t_i) = \sum_{j=1}^{n} d_{hj}\langle\psi_j,K_i\rangle - \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} d_{hj}\langle K_k,K_i\rangle\Lambda_{jk} -$$

$$- \frac{1}{2}\sum_{j=1}^{n} d_{hj}\langle y,K_i\rangle\Lambda_j, \qquad h,i = 1,2,\ldots,n ,$$

and (7.2) becomes

$$(9.14)\qquad y(t_i) = \sum_{j=1}^{n} d_{ij}\langle \psi_j, y\rangle - \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} d_{ij}\langle K_k, y\rangle \Lambda_{jk} -$$

$$- \frac{1}{2}\sum_{j=1}^{n} d_{ij}\langle y, y\rangle \Lambda_j, \qquad i = 1,2,\ldots,n\ .$$

If one sets $\Lambda_1 = \Lambda_2 = \ldots = \Lambda_n = 0$ in (9.13) and solves for $\Lambda_{hi} = \Lambda^0_{hi}$, $h,i = 1,2,\ldots,n$, the corresponding functions

$$(9.15)\qquad v^0_i(t) = \sum_{j=1}^{n} d_{ij}\int_0^1 K_j(s)K(s,t)w_j ds - \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} d_{ij}\Lambda^0_{jk}K_k(t)$$

give the kernel $K^0_n(s,t)$ which minimizes the functional $F[v]$ over $\{K_n(s,t)\}_0$. By solving (9.13) and (9.14) for the Lagrange multipliers, one can find $v_1(t), v_2(t), \ldots, v_n(t)$ for the completely constrained problem. By substitution, the estimates $\nu_0 \leq F[v^0]$ and $\nu \leq F[v]$ can be obtained for the respective Nyström constants.

REFERENCES

1. Anselone, P. M., Collectively Compact Operator Approximation Theory, Prentice-Hall, Englewood Cliffs, N. J., 1971.
2. Anselone, P. M. and Rall, L. B., The solution of characteristic value-vector problems by Newton's method, Numer. Math. 11 (1968), 38-45.
3. Aronszajn, N., Theory of reproducing kernels, Trans. Amer. Math. Soc. 68 (1950), 337-404.
4. Atkinson, K. E., A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind, SIAM, Philadelphia, 1976.
5. Davis, P. J. and Rabinowitz, Philip, Methods of Numerical Integration, Academic Press, New York, 1975.
6. Fredholm, I., Sur une classe d'équations fonctionnelles, Acta Math. 27 (1903), 365-390.
7. Golomb, M. and Weinberger, H. F., Otpimal approximation and error bounds, On Numerical Approximation, Ed. by R. E. Langer, pp. 117-190, Univ. of Wisconsin Press, Madison, 1959.
8. Goursat, É., Sur un cas élémentaire de l'équation de Fredholm, Bull. Soc. Math. France 35 (1907), 163-173.

9. Milne, W. E., Numerical Calculus, Princeton Univ. Press, Princeton, N. J., 1949.

10. Mysovskih, I. P., Lectures on Numerical Methods, tr. from Russian by L. B. Rall, Wolters-Noordhoff Publishing, Groningen, The Netherlands, 1969.

11. Noble, B., Numerical Methods II: Differences, Integration and Differential Equations, Oliver and Boyd, London, 1964.

12. Noble, B., Error analysis of collocation methods for solving Fredholm integral equations, Topics in Numerical Analysis, Ed. by John J. H. Miller, pp. 211-232, Academic Press, New York, 1973.

13. Nyström, E. J., Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben, Acta Math. 54 (1930), 185-204.

14. Phillips, J. L., Error analysis for direct linear integral equation methods, Math. Comp. 27 (1973), 849-859.

15. Rall, L. B., Error bounds for iterative solution of Fredholm integral equations, Pacific J. Math. 5 (1955), 977-986.

16. Rall, L. B., Computational Solution of Nonlinear Operator Equations, John Wiley & Sons, New York, 1969.

17. Sloan, I. H., Error analysis for a class of degenerate-kernel methods, Numer. Math. 25 (1976), 231-238.

NUMERICAL SOLUTIONS TO THE LATERAL STABILITY OF A MISSILE

Julian J. Wu
U. S. Army Armament Research and Development Command
Benet Weapons Laboratory, LCWSL
Watervliet Arsenal, Watervliet, NY 12189

ABSTRACT. Numerical data are presented in this paper on the stability behavior of a free flying column subjected to an axial thrust, the direction of which can be adjusted to improve the stability characteristics. This is a basic problem of missile design for lateral stability and has not been fully understood prior to this date. The results obtained by the finite element method are plotted showing clearly the effects on the structure in various types of instability due to different thrust directional control parameters. These data have been substantiated by a recent analysis to be published separately.

1. INTRODUCTION. Numerical results are presented in this paper on the lateral stability of a free-flying Euler-Bernoulli column subjected at one end to an axial thrust, the direction of which can be rotated through a small angle about the tangent of the column. This is a basic problem of a flexible missile, many aspects of which have not been fully explored. Recently, several elusive questions on the solution of this problem have been resolved by the use of asymptotic expansions [1]. A brief history of this problem and associated difficulties have been discussed in [1]. The purpose of this paper is to present some additional numerical data on this fundamental and interesting problem, to recapitulate the solution formulations and to provide some physical interpretations of the solutions.

2. GOVERNING EQUATIONS AND STABILITY PARAMETERS. The differential equation of the lateral motion of an Euler-Bernoulli beam can be written as the following

$$(EIu'')'' + (Pu')' + \rho A\ddot{u} = 0 \tag{1a}$$

The boundary conditions associated with a free-flying column subjected to a constant thrust with directional control are

$$EIu'' = (EIu'')' = 0 \quad \text{at} \quad x = 0 \tag{1b,1c}$$

$$\left.\begin{aligned} EIu'' &= 0 \\ (EIu'')' - PK_\theta u' &= 0 \end{aligned}\right\} \text{at} \quad x = \ell \tag{1d,1e}$$

In Eqs. (1), $u = u(x,t)$ is the lateral disturbance of the column from its equilibrium position, a prime (') denotes differentiation with respect to the spatial coordinates s, a dot ($\cdot$), differentiation with respect to time t, E is the Young's Modulus, ρ, density of the material, I, second moment, A the area of the cross-section, ℓ length of the column, P is the axial thrust acted at end $x = \ell$ and K_θ is a nondimensional design parameter indicating the amount of rotation the thrust is to have. It will be shown in this paper that the value of K_θ has great effect on the stability behavior of the column under consideration (Figure 1).

To simplfy our discussion, we shall consider a uniform column and Eqs. (1) will be nondimensionalized. The quantities in length will be nondimensionalized through a division by ℓ and those in time, through a division by a constant c, where

$$c = \left(\frac{\rho A \ell^4}{EI}\right)^{1/2} \tag{2}$$

which has a unit of real time. Thus Eqs. (1) become

$$u'''' + Q(xu')' + \ddot{u} = 0 \tag{3a}$$

$$u''(0) = 0, \quad u'''(0) = 0, \quad u''(1) = 0 \tag{3b,3c,3d}$$

$$u'''(1) - K_\theta Q u'(1) = 0 \tag{3e}$$

where in Eqs. (3), the nondimensionalized thrust Q is given by

$$Q = \frac{P\ell^2}{EI} \tag{4}$$

For vibrations and quasi-dynamic stability problems, one can eliminate the time variable by assuming that

$$u(x,t) = u(x)e^{\lambda t} \tag{5}$$

Eq. (1a) then becomes

$$u'''' + Q(xu')' + \lambda^2 u = 0 \tag{3a'}$$

and the initial conditions do not enter into the problem. "Eqs. (3')" will be used to refer to Eqs. (3) with Eq. (3a) replaced by Eq. (3a'). For most of the results of this paper Eqs. (3') will be used. However, in the case of repeated eigenvalues with identical eigenfunctions, Eqs. (3) must be used to find another independent solution. This point has been discussed in reference [1].

It is clear from Eq. (5) that the parameter λ dictates the stability nature of the problem. A purely imaginary λ indicates stable vibrations; a purely real and positive λ, instability of divergence and a complex λ with positive real part, instability of flutter. Since λ appears only as λ^2 in Eqs. (3'), it is equally true that a real negative λ^2 indicates stable vibrations; a real positive λ^2, instability of divergence; and a complex λ^2, instability of flutter.

3. FINITE ELEMENTS WITH UNCONSTRAINED VARIATIONAL FORMULATIONS. The method used to obtain the numerical data is based on a mathematical analysis concerning nonself-adjoint boundary value problems [2]. Here in this section, we shall apply it to this special problem missile stability. The key to this solution is an unconstrained, adjoint variational principle. First, we shall show that such a principle can be constructed and is equivalent to the differential equations together with the boundary conditions. Second, we shall show that this principle can be routinely discretized and lead to the matrix equation required for the numerical solutions.

In conjunction with Eqs. (3'), an adjoint variable $v(x)$ is introduced in the following variational statement

$$\delta J = 0 \tag{6a}$$

where

$$J = J(u,v) = \int_0^1 (u''v'' - Qxu'v' + \lambda^2 uv)dx + Q(1 + K_\theta)u'(1)v(1) \tag{6b}$$

Through integrations by parts, one can easily establish that the necessary and sufficient condition for Eqs. (6) is the original boundary value problem of Eq. (3') and the following adjoint boundary value problem

$$v'''' + Q(xv')' + \lambda^2 v = 0 \tag{7a}$$

$$v''(0) = 0, \quad v'''(0) = 0 \tag{7b,7c}$$

$$v''(1) + Q(1 + K_\theta)v(1) = 0 \tag{7d}$$

$$v'''(1) + Qv'(1) = 0 \tag{7e}$$

However, in so far as the original problem is linear, $u(x)$ and $v(x)$ are quite independent of each other. The solution of $u(x)$ can be obtained without solving for $v(x)$. Thus, to proceed for the finite element matrix equation, one shall take variation of $v(x)$ alone and keep $u(x)$ fixed.

$$(\delta J)_u = 0 = \int_0^1 (u''\delta v'' - Qxu'\delta v' + \lambda^2 u\delta v)dx$$

$$+ Q(1 + K_\theta)u'(1)\delta v(1) \tag{8}$$

In the process of descritization, the column is divided into many segments (elements). Again for the sake of simplicity, these segments are taken to be of equal length. Let L be the number of elements and the local coordinate

$$\xi = \xi^{(i)} = L[x - (i - 1)/L] \tag{9}$$

where the superscript i indicates the i-th element. In terms of $\xi^{(i)}$, Eq. (8) becomes

$$\sum_{i=1}^{L} \int_0^1 \{L^3 u^{(i)}{}'' \delta v^{(i)}{}'' - Q[\xi + i - 1]u^{(i)}{}' \delta v^{(i)}{}' + \frac{\lambda^2}{L} u^{(i)}\delta v^{(i)}\}d\xi$$

$$+ Q(1 + K_\theta)u^{(L)}{}'(1)\delta v^{(L)}(1) = 0 \tag{10}$$

Now, introduce the generalized coordinate vectors

$$\underset{\sim}{u}^{(i)T} = \{u_1^{(i)} \quad u_2^{(i)} \quad u_3^{(i)} \quad u_4^{(i)}\} \tag{11a}$$

$$\underset{\sim}{v}^{(i)T} = \{v_1^{(i)} \quad v_2^{(i)} \quad v_3^{(i)} \quad v_4^{(i)}\} \tag{11b}$$

and the displacement function vector

$$\underset{\sim}{a}^T(\xi) = \{1 - 3\xi^2 + 2\xi^3 \quad \xi - 2\xi^2 + \xi^3 \quad 3\xi^2 - 2\xi^3 \quad -\xi^2 + \xi^3\} \tag{12}$$

such that

$$u^{(i)}(\xi) = \underset{\sim}{a}^T(\xi)\underset{\sim}{U}^{(i)}, \qquad v^{(i)}(\xi) = \underset{\sim}{a}^T(\xi)\underset{\sim}{V}^{(i)} \tag{13}$$

The superscript T indicates "transpose of a matrix". Eqs. (13) are substituted into (10) to yield

$$\sum_{i=1}^{L} \delta\underset{\sim}{V}^{(i)T}\{L^3\underset{\sim}{C} - Q[\underset{\sim}{D} + (i - 1)\underset{\sim}{B}] + \frac{\lambda^2}{L}\underset{\sim}{A}\}\underset{\sim}{U}^{(i)}$$

$$+ \delta\underset{\sim}{V}^{(L)T}\underset{\sim}{E}\,\underset{\sim}{U}^{(L)} = 0 \tag{14}$$

where

$$\underset{\sim}{A} = \int_0^1 \underset{\sim}{a}\underset{\sim}{a}^T d\xi, \quad \underset{\sim}{B} = \int_0^1 \underset{\sim}{a}'\underset{\sim}{a}'^T d\xi, \quad \underset{\sim}{C} = \int_0^1 \underset{\sim}{a}''\underset{\sim}{a}''^T d\xi$$

$$\underset{\sim}{D} = \int_0^1 \xi \underset{\sim}{a}\underset{\sim}{a}^T d, \quad \underset{\sim}{E} = \underset{\sim}{a}(1)\underset{\sim}{a}'^T(1) \tag{15}$$

Through the use of requirement that the displacement and slope be continuous at the nodes, i.e.,

$$\begin{aligned} U_3^{(i-1)} &= U_1^{(i)} \\ U_4^{(i-1)} &= U_2^{(i)} \end{aligned} \qquad i = 2,3,\ldots,L \tag{16}$$

Eq. (14) can be written as

$$\delta\underset{\sim}{V}^T(\underset{\sim}{K} + \lambda^2\underset{\sim}{M})\underset{\sim}{U} = 0 \tag{17}$$

where

$$\begin{aligned} \underset{\sim}{U}^T &= \{U_1^{(1)} \quad U_2^{(1)} \quad U_3^{(1)} \quad U_4^{(1)} \quad U_3^{(2)} \quad U_4^{(2)} \quad \ldots\ U_3^{(L)} \quad U_4^{(L)}\} \\ \underset{\sim}{V}^T &= \{V_1^{(1)} \quad V_2^{(1)} \quad V_3^{(1)} \quad V_4^{(1)} \quad V_3^{(2)} \quad V_4^{(2)} \quad \ldots\ V_3^{(L)} \quad V_4^{(L)}\} \end{aligned} \tag{18}$$

The global stiffness matrix $\underset{\sim}{K}$ and inertia matrix $\underset{\sim}{M}$ are assembled from the local matrices of Eqs. (15) through standard procedures [3].

As remarked earlier in this section, δv is unconstrained and is independent of u. Thus $\delta\underset{\sim}{V}$ is unconstrained and independent of $\underset{\sim}{U}$. Eq. (17) then reduces to

$$(\underset{\sim}{K} + \lambda^2\underset{\sim}{M})\underset{\sim}{U} = 0 \tag{19}$$

which is the final matrix equation to be solved.

4. NUMERICAL RESULTS AND INTERPRETATIONS. Numerical results obtained by the formulations described in the previous section have been plotted in Figures 2 through 6. We shall discuss their significance of structural stability in several categories.

4.1 The Case of Zero Thrust. The two lowest eigenvalues of bending mode are 22.37 and 61.70 as shown in Figures 2 through 6. There are two zero eigenvalues (in terms of λ^2). One corresponds to the rigid body translation, the other, rigid body rotation. As the thrust increases from zero, the rigid body translation mode is going to remain as a valid mode of motion regardless of the value of K_θ. However, the rigid body rotation will cease to be a mode. It evolves into various types of stable and unstable motion depending on the design value of K_θ as it will be described in sequel.

4.2 The Case $K_\theta = 0$. The thrust in this case is a follower force. As it increases from zero, the rigid body rotation has ceased to be a mode. In other words, it is no more a valid eigenfunction of the boundary value problem defined by Eqs. (3'). Instead it degenerates into a rigid body translation. However, the rigid body rotation does exist, not as a mode, but together with a parametric term as a solution to the partial differential equation and boundary conditions of Eqs. (3). This has been discussed in [1]. The elastic instability occurs only as the third and the fourth eigenvalue branches coalesce as Q has reached a value of $11.126\pi^2$ (Figure 2). The eigenvalue λ^2 becomes complex from there on, meaning the onset of flutter instability.

4.3 The Case $K_\theta > 0$. The direction of the thrust is controlled so that the thrust is rotated in the same direction as the slope of the column. As the magnitude of the thrust increases from zero, the eigenvalue λ corresponding to the rigid body rotation at zero thrust becomes imaginary, meaning a vibratory mode (Figures 3 and 4). This new mode becomes the lowest mode of bending, and it will stay as a vibratory mode before the thrust Q reaches the value of $Q_1 = 2.598\pi^2$. At Q_1, the eigenvalue becomes zero again, and the mode shape degenerates into rigid body translation (not rotation). The rigid body rotation motion exists only with a parametric excitation as the case mentioned in Subsection 4.2. Thus from zero thrust to Q_1, there is a range of elastic stability. Beyond Q_1, the eigenvalue λ becomes real and positive, meaning divergence instability. In other words, the column will buckle.

4.4 The Case $K_\theta < 0$. The direction of the thrust is now rotated in the opposite sense to that of the column's slope. As the thrust Q increases from zero, one of zero eigenvalues, corresponding to the rigid body rotation at zero thrust, immediately becomes real and positive - a case of buckling under arbitrarily small thrust! The physical implication of this buckling instability is somewhat unconventional. It appears to be a unique feature to the stability problem capable of rigid body motions. (Figures 5 and 6.)

REFERENCES

1. D. A. Peters and J. J. Wu, 1978 Journal of Sound and Vibration (to appear). Asymptotic solutions to a stability problem.

2. J. J. Wu, 1975 Journal of Sound and Vibration 39, 195-206. On adjoint operators associated with boundary value problems.

3. J. J. Wu, 1975 Developments in Mechanics 8, 279-294. A unified finite element approach to column stability problems.

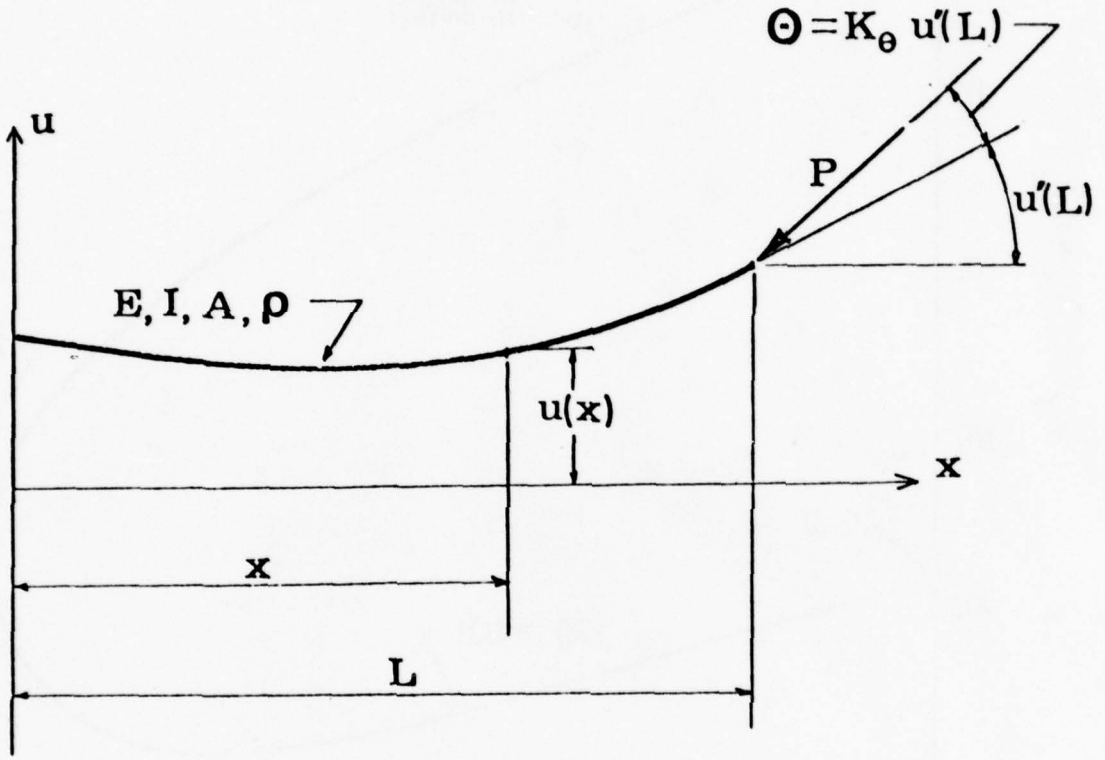


FIGURE 1. Problem Configuration.

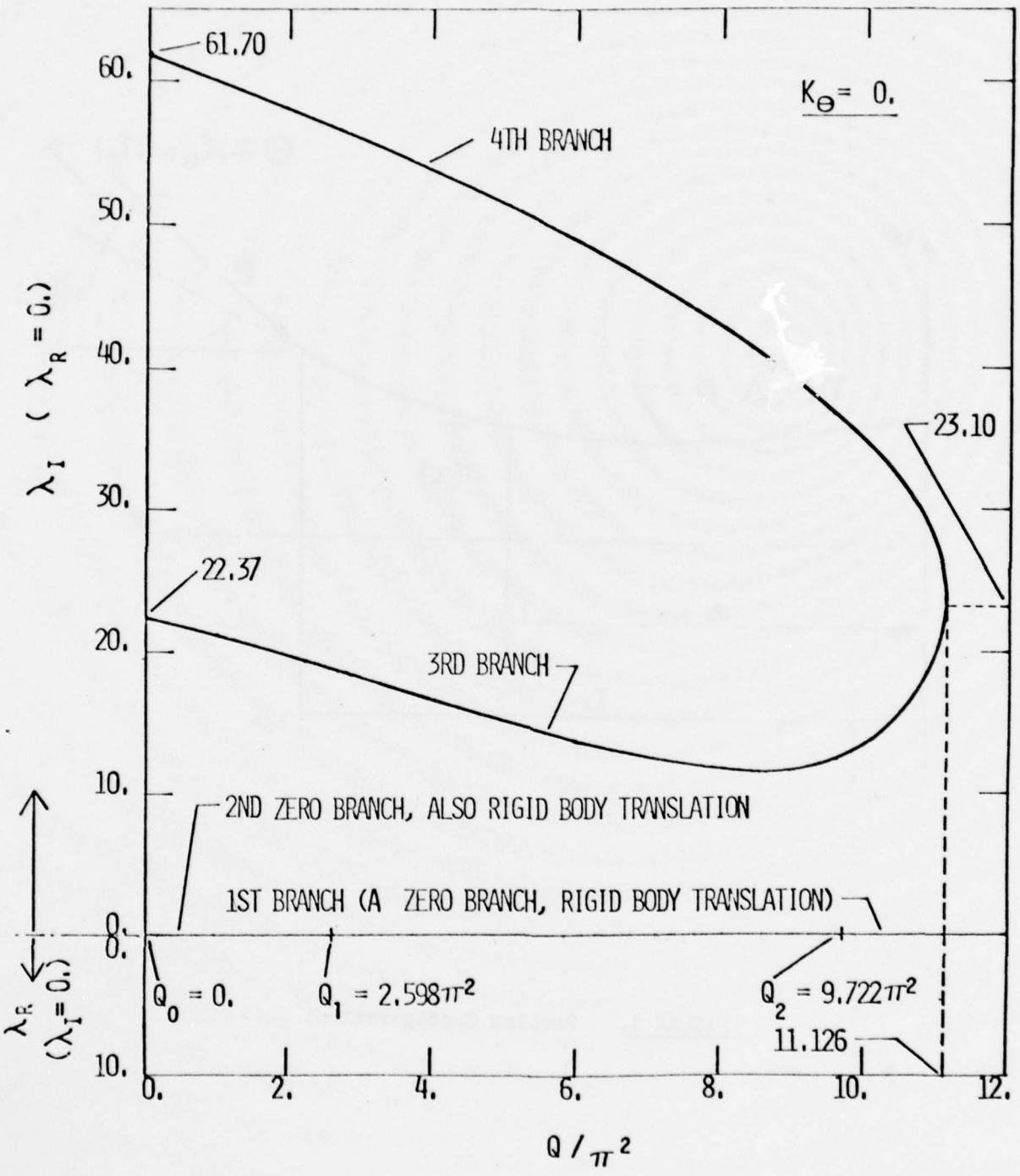


FIGURE 2. Four Lowest Branches of Eigenvalues, $K_\theta = 0.$

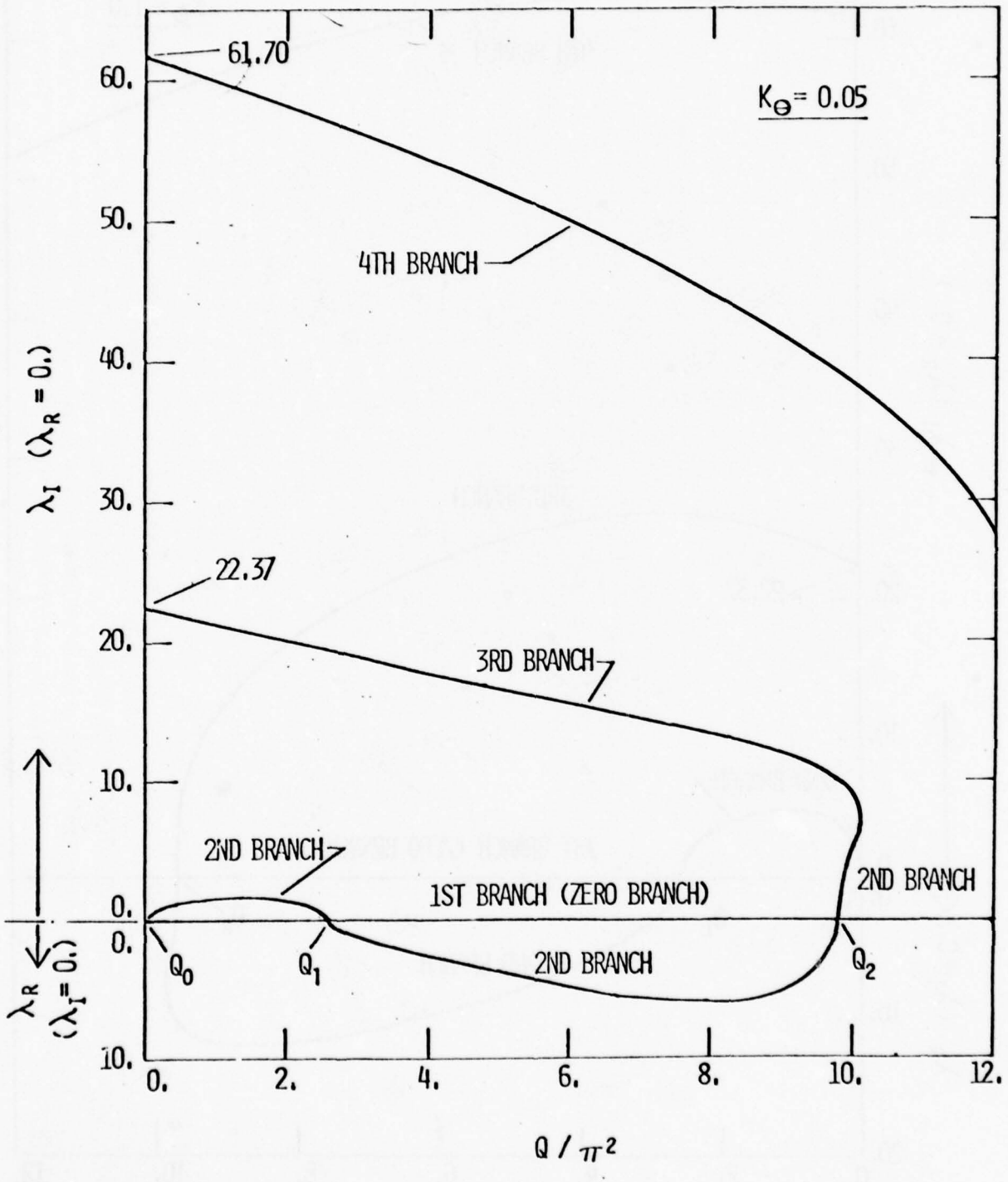


FIGURE 3. Four Lowest Branches of Eigenvalues, K_θ = 0.05.

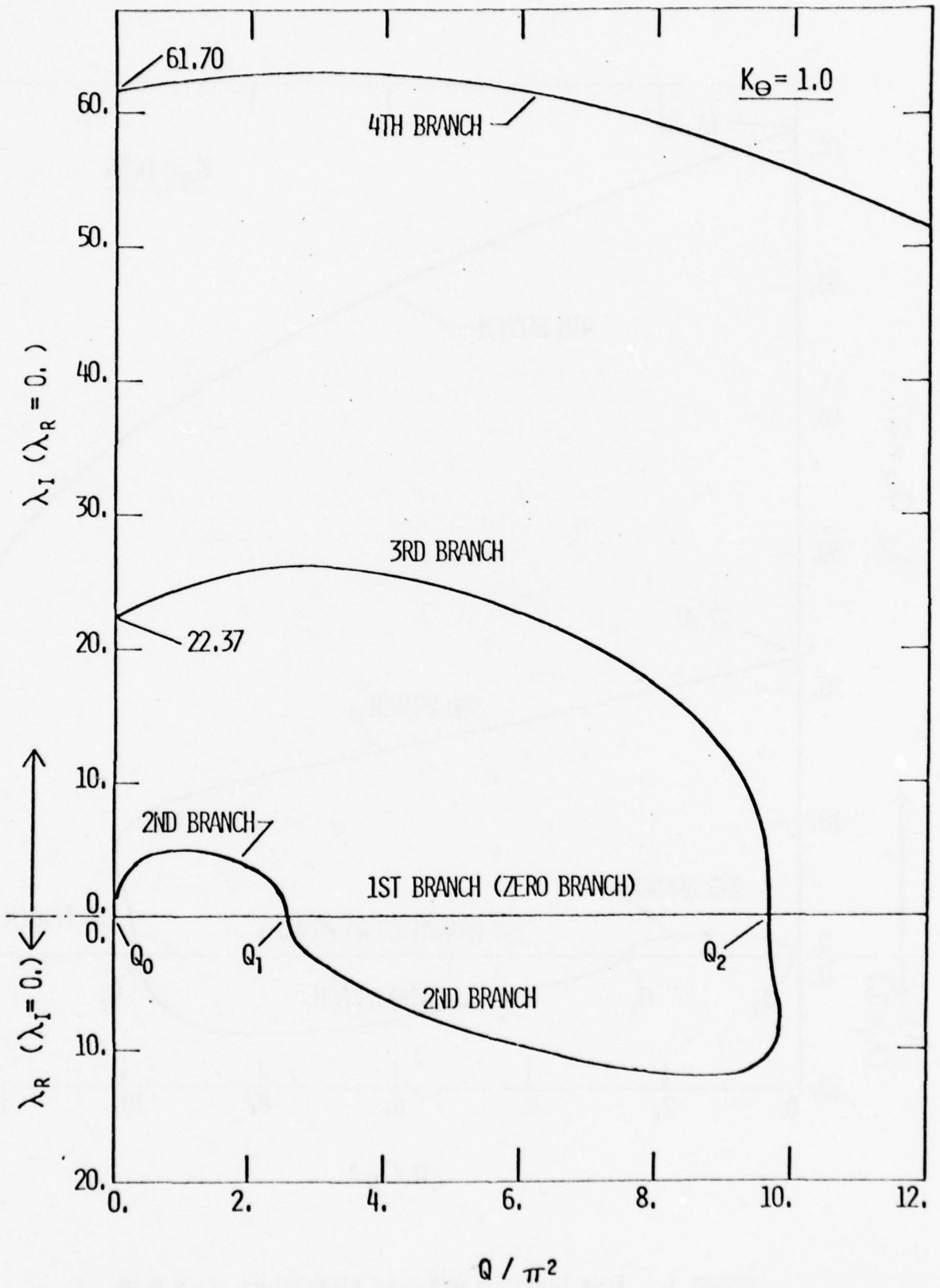


FIGURE 4. Four Lowest Branches of Eigenvalues, $K_\theta = 1.0$.

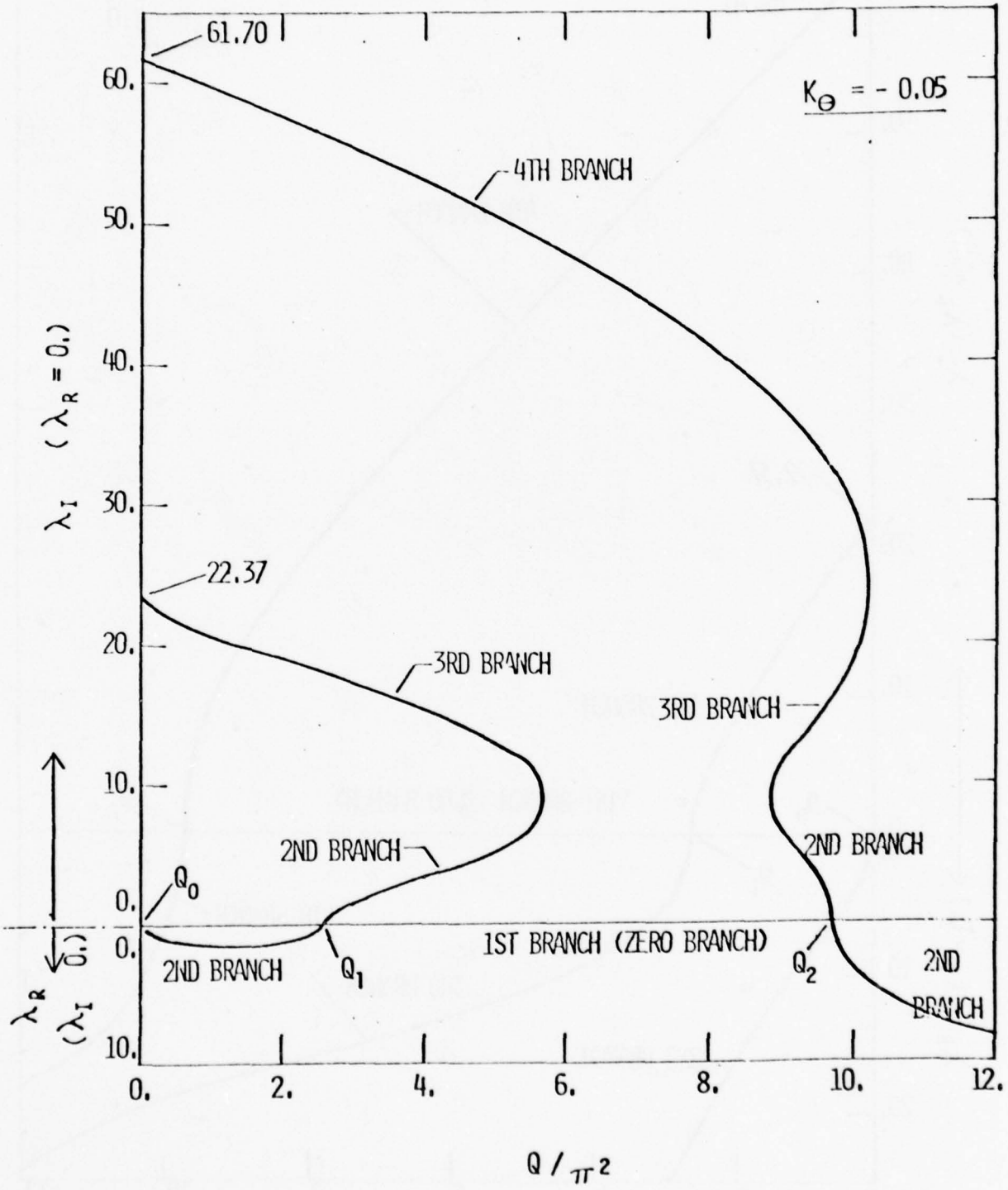


FIGURE 5. Four Lowest Branches of Eigenvalues, K_θ = -0.05.

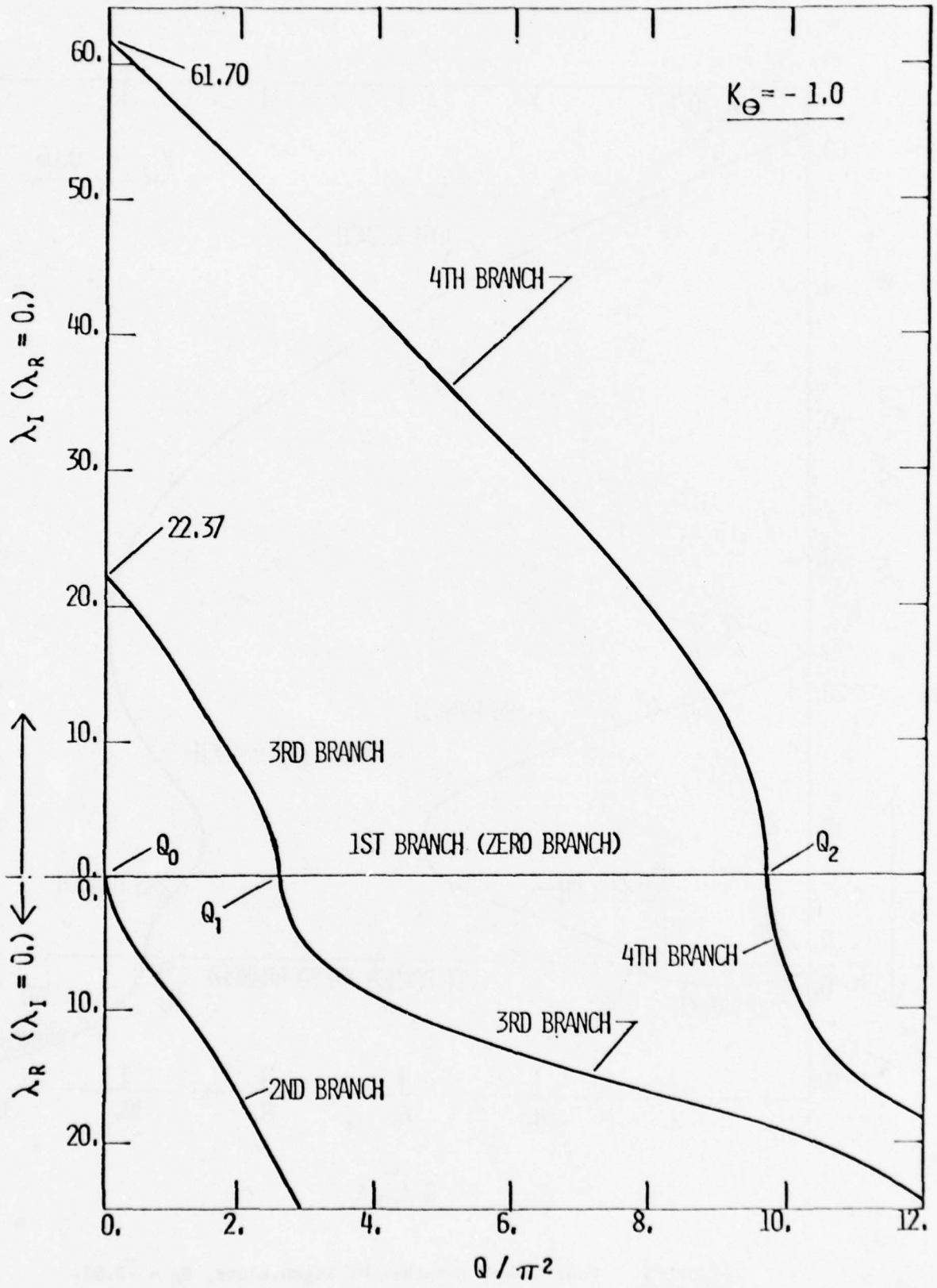


FIGURE 6. Four Lowest Branches of Eigenvalues, K_θ = -1.0.

COMPUTATION OF TRANSONIC FLOW PAST SLENDER BODIES AT ANGLE OF ATTACK

R. P. Reklis and W. B. Sturek
U.S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

and

F. R. Bailey
NASA Ames Research Center
Moffett Field, California 94035

ABSTRACT. Aerodynamic properties of artillery shell such as normal force and pitching moment reach peak values in a narrow transonic Mach number range. In order to compute these quantities, numerical techniques have been developed to obtain solutions to the three-dimensional transonic small disturbance equation about slender bodies at angle of attack. The computation is based on a plane relaxation technique involving Fourier transforms to partially decouple the three-dimensional difference equations. Particular care is taken to assure accurate solutions near corners found in shell designs. Computed surface pressures are compared to experimental measurements for circular arc and cone cylinder bodies which have been selected as test cases. Computed pitching moments are compared to range measurements for a typical projectile shape.

1. INTRODUCTION. When designing an artillery shell, it is necessary to develop a vehicle which will fly with stability under a wide variety of aerodynamic conditions. A range of propellant charges may be used giving the shell launch velocities covering a spectrum from subsonic to supersonic. The shell will also slow in flight, particularly near the apex of its trajectory. It is, therefore, important that the shell fly with stability in subsonic, transonic, and supersonic flight regimes.

Difficulties are often experienced by projectiles at transonic velocities. Aerodynamic properties such as drag and the pitching moment, which is critical to stability, will reach peak values at some transonic Mach number. This peak can form in a Mach number range which may be very limited as, for example, between $.92 < M < .94$. The sharpness of this critical behavior as well as the value of the critical Mach number are very sensitive to body geometry. A slight change in boattail length may make the difference between a successful shell and one whose behavior is unpredictable.

Aerodynamic range and wind tunnel testing are difficult and expensive, particularly at transonic velocities. Therefore, it is of great importance for artillery projectile design to develop a computational capability which can provide guidance in choosing shell configurations and reduce aerodynamic testing requirements. Techniques have been established and computers are now available which should make possible the development of useful computational design tools, particularly for the limited and usually simple geometries found in artillery projectile shapes.

The basic approach used by BRL to compute flow over supersonic projectiles[1] has been to solve for the inviscid flow field around the body and to then compute the flow in the boundary layer using turbulence modeling. This approach seems quite feasible for use in the transonic problem considering the results shown by Schmidt, Stock, and Fritz[2] who have coupled integral boundary layer calculations to transonic inviscid solutions. The major thrust of the transonic work reported here involves the development of numerical techniques for the computation of three-dimensional transonic inviscid flow past artillery projectiles. A three-dimensional, compressible, turbulent, boundary layer code using finite difference techniques has been developed by BRL[1] for use with supersonic flow and it's carry over should be straight forward.

Complications in the body geometry such as the rotating band and rifling have been neglected. The resulting simplified shape, however, exhibits the basic aerodynamic properties of the projectile. One complication which cannot be ignored is the presence of corners at the junction of the ogive and cylindrical portions of the shell and at the junction of the cylindrical portion and the boattail. These corners are responsible for the critical transonic aerodynamic behavior of the shell.

2. COMPUTATION TECHNIQUES. Transonic techniques based on type dependent differencing have come into wide acceptance since they were developed by Murman and Cole[3]. There are now several schemes like that developed by Bailey and Ballhaus[4] that will solve three-dimensional potential flow over wings and wing body combinations. Considerable simplification, however, may be achieved in the projectile problem if the code is restricted to axially symmetric bodies. Further, by using a cylindrical coordinate system which fits the body surface, no vertical or horizontal preferred directions are established. This provides an important increase in accuracy. The axially symmetric problem at angle of attack, although it is simpler than the wing body problem, does not have the range of application and has not seen as wide an interest. There has been, however, some recent numerical work by Reyhner[5] who has studied axisymmetric inlets.

Even though techniques are available to solve the full inviscid potential equation directly[6], the approach chosen for this study has been to solve the transonic small disturbance equation

$$[(1 - M^2) - M^2 (\gamma + 1) \phi_z] \phi_{zz} + \phi_{rr} + \phi_r/r + \phi_{\theta\theta}/r^2 = 0 \quad , \qquad (1)$$

which is an approximation to the full equation. This is a nonlinear partial differential equation of mixed elliptic-hyperbolic type written in a cylindrical coordinate system (z,r,θ) as shown in Figure 1. The free stream Mach number is given by M in this equation and the ratio of specific heats (1.4 for air) is represented by γ. The solution to this equation has been shown by Bailey[7] to give good results for the slender body case at zero angle of attack. This equation has also been studied both numerically and analytically for many years and it is simple enough that much valuable insight may be gained from it, particularly in regions near the body where it reduces to

$$\phi_{rr} + \phi_r/r + \phi_{\theta\theta}/r^2 = 0 \quad . \tag{2}$$

This last equation is a Laplace equation that may be solved analytically.

In his two dimensional solution of equation (1) Bailey[7] used successive line over relaxation in which difference equations were solved along lines through the flow extending radially from the body axis. This line relaxation procedure has been carried a step further in the technique used here in three dimensions. The difference equations for whole planes of flow cutting the body axis have been solved simultaneously. The method thus obtained treats coupling in the r and θ directions with a direct solution technique which closely matches the physical coupling found in the flow. Flow disturbances propagate much more strongly in the r and θ directions than along the body in the z direction. This may be seen from a look at the inner equation (2). There is no z coupling in this equation. This lack of disturbance propagation along the body is a well known property of transonic flow.

The matrix equations obtained from a line relaxation procedure are in tridiagonal form which may readily be solved. The matrix equations for plane relaxation are not. It is, however, possible to make use of the periodic nature of the axisymmetric problem in such a way as to transform these matrix equations into tridiagonal form. This transformation is accomplished by a basis change which is equivalent to a Fourier transformation in θ. Reyhner[5] has pointed out that the solution around the body is very nearly

$$\phi_1 (r,z) + \phi_2 (r,z) \cos \theta \quad .$$

This result appears very clearly in the Fourier transform approach and allows a considerable saving in both computer time and storage since only a few Fourier components need be treated. The use of Fourier transforms does increase, to some extent, the problem of obtaining a stable relaxation scheme. A simple stable scheme can be obtained, however, which reduces, at zero angle of attack, to the usual line relaxation algorithm.

3. DISCUSSION OF RESULTS. The results for computations of the surface pressure coefficient for bodies with circular arc profiles can be seen in Figures 2 and 3. Figure 2 shows a comparison of computed and wind tunnel[8] pressure coefficients along the surface of a 1/10 fineness ratio body at zero angle of attack in a Mach number .99 free stream. The location of a shock can be clearly seen just aft of the center of the body.

The solid line shows the results of computations for a body generated by a perfect circular arc. The wind tunnel model, however, was supported from the rear by a sting. The effect of the sting was modeled and the resulting computed pressure coefficients are shown by the dashed line in this figure. As the angle of attack is zero in the case shown in Figure 2, the computation is two dimensional. This same case was computed by Bailey[7] in his earlier two-dimensional work and the results are identical.

Figure 3 shows a comparison of computed and wind tunnel[9] pressures for a slightly more slender body of fineness ratio 1/12. The Mach number in this case was .90 which is too low to allow development of a large supersonic region with strong shocks. The figure is presented to show the result of a three-dimensional computation. Unfortunately, wind tunnel data for cases showing strong shock development were unavailable at angle of attack. Discrepancies between computed and wind tunnel pressure over the aft portion of the body are the result of the presence of the wind tunnel sting.

The results presented in these two figures confirm the ability of the three-dimensional code to predict surface pressures over smooth bodies. There is little difference between the nose of a typical artillery shell which is an ogive and the front portion of these circular arc bodies. Artillery shell, however, often exhibit corners, particularly at the junction between the ogive and cylinder portions and between the cylinder portion and the boattail. Strong shocks are formed by the collapse of supersonic regions which are generated by the expansion of the flow over these corners when the shell is flown at a slightly subsonic velocity ($.8 < M < 1$). This phenomena is demonstrated in Figure 4 which shows the shadowgraph of a shell at the critical Mach number ($M = .926$). Note that the shock on the upper surface of the boattail is nearly off the end. The high pressure on the lower surface behind the shock on the boattail provides a large lift on the tail creating a powerful overturning moment. The flow pattern generated by the corner at the beginning of the boattail is largely responsible for the critical behavior of the overturning moment. Thus, the accurate treatment of corner flow is of prime consideration.

Corners create singularities in the potential. The well known incompressible result is that flow obtains infinite velocity over a corner. The speed of the flow, however, will clearly become supersonic before it becomes infinite so that an incompressible calculation is unacceptable. The corner problem is, therefore, by nature transonic and can be treated by transonic techniques. Also, the boundary layer which is well developed over the corner at the start of the boattail will tend to effectively round this corner so that a strict mathematical singularity does not exist.

The ability of the present theory to predict flow over a corner can be seen in Figure 5. Figure 5 shows a comparison of surface pressure results with wind tunnel experiments[10] for flow over a cone cylinder model at Mach number 1.1. The theory shows reasonable behavior near the corner of the cone and cylinder sections. In order to achieve these results it was necessary to use care in applying boundary conditions at the body surface. An approach that is often taken is to use solutions of the simpler inner equation (2) to extrapolate the boundary conditions from the body surface to the body axis or to some other convenient location. In Bailey's two dimensional paper the boundary conditions were extrapolated to the axis where a series of sources and sinks were placed. The source and sink strengths were obtained from the solution to the inner equation.

This procedure is not feasible if accurate corner flow is to be obtained. The inner equation, which is obtained by dropping the nonlinear term from equation (1), does not apply near corners where, in fact, the nonlinear term may be large close to the body surface. Boundary conditions must be applied directly at the surface without extrapolation. Further improvement in the application of boundary conditions may also be obtained. The usual boundary condition which is applied at the body surface is given by

$$\phi_r\Big|_{\text{surface}} = dR/dz \quad ,$$

where the left hand side is the radial derivative of the potential evaluated at the body surface and the right hand side is the slope of the body surface. This is a first order approximation to the body surface boundary condition. A second order formula is more appropriate and is given by

$$\phi_r\Big|_{\text{surface}} = (1 + \phi_z\Big|_{\text{surface}})\, dR/dz \quad .$$

The first order formula works well as long as ϕ_z remains small in comparison to 1. Near a corner ϕ_z may become large enough that it produces a noticeable effect as seen in Figure 5. Because of the iterative relaxation procedure used in solving the potential equation ϕ_z may be obtained at an old iteration. The right hand side of the second order formula may thus be evaluated. The effect of using a finer grid spacing may also be seen in Figure 5. The grid in all cases shown in this figure was made up of 64 points. In the case labeled fine grid in this figure these points were clustered so as to give twice the density near the corner. Subsequent calculations have been carried out with 128 points so as to achieve this same fine density when more than one corner is present.

The absolute necessity of the care taken with corner flow may be seen when computations of normal force and pitching moment are made. It is possible to use the inner solution found from equation (2) to predict both normal force and pitching moment. It should be noted that there is no Mach number dependence in equation (2). Because of this, both normal force and pitching moment will be constants in Mach number and will depend only on the body shape. Since it is precisely the large transonic variations of normal force and pitching moment that are desired and since the use of equation (2) will predict no Mach number dependence of these quantities, equation (2) must not be valid everywhere on the surface and it is not possible to use equation (2) to supply boundary conditions. A corollary to this argument is that it is the areas where equation (2) breaks down which are of interest in obtaining variations of normal force and pitching moment with Mach number. Such breakdowns occur near corners and it is for this reason that they are of such critical interest. Breakdowns also occur around shocks. As seen in Figure 2, accurate shock location is also vital to the computation of aerodynamic quantities because of the large pressure difference between the upper and lower surfaces in the neighborhood of the shock on the boattail.

It has been shown in the above discussion that the transonic techniques, that have been developed, will predict flow over both smooth bodies and bodies with corners. It should, therefore, be possible to obtain the solution over a body that closely resembles an artillery shell. The lift loading computed for a typical artillery shell shape is plotted in Figure 6. This graph shows the normal force per unit length plotted as a function of the position along the shell. It is felt that the features of this curve, particularly the large downward spike in the boattail region, gives an accurate representation of the aerodynamic forces on this body. Comparison of the pitching moment coefficient computed for this body and range measurements[11] of a similar shell are given as a function of Mach number in Figure 7. The peak shown in the computed results falls a few hundredths of a Mach number higher than the peak in the range measurements and is not as pronounced.

It is felt that this situation will improve with the inclusion of the boundary layer, the inclusion of the rotating band, and a better modeling of the wake. These improvements will be added in the very near future. At present it is felt that the techniques described above will be capable of accurate prediction of both normal force and pitching moment for practical shell configurations. With the inclusion of a boundary layer, Magnus forces may also be predicted.

REFERENCES

1. Sturek, W. B., et al, "Computations of Turbulent Boundary Layer Development Over a Yawed, Spinning Body of Revolution With Application to the Magnus Effect," BRL Report No. 1985, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, Maryland, May 1977. AD A041338.

2. Schmidt, W., Stock, H. W., and Fritz, W., "Numerical Simulation of Three Dimensional Transonic Flow Including Wind Tunnel Wall Effects," AGARD Specialists Meeting, "Numerical Methods and Wind Tunnel Testing," VKI, Brussels, 1976.

3. Murman, E. M., and Cole, J. D., "Calculation of Plane Steady Transonic Flows," *AIAA Journal*, Vol. 9, No. 1, January 1971, pp. 114-121.

4. Bailey, F. R., and Ballhaus, W. F., "Comparisons of Computed and Experimental Pressures for Transonic Flows About Isolated Wings and Wing Fuselage Configurations," NASA SP-347, Vol. 2, March 1975, pp. 1213-1226.

5. Reyhner, T. A., "Transonic Potential Flow Around Axisymmetric Inlets and Bodies at Angle of Attack," *AIAA Journal*, Vol. 15, No. 9, September 1977, pp. 1299-1306.

6. South, J. C., Jr., and Jameson, Anthony, "Relaxation Solutions for Inviscid Axisymmetric Transonic Flow Over Blunt or Pointed Bodies," AIAA Computational Fluid Dynamics Conference, Palm Springs, California, July 1973, pp. 8-17.

7. Bailey, F. R., "Numerical Calculation of Transonic Flow About Slender Bodies of Revolution," NASA TN-D-6582, December 1971.

8. Taylor, R. A., and McDevitt, J. B., "Pressure Distribution at Transonic Speeds for Parabolic-Arc Bodies of Revolution Having Fineness Ratios of 10, 12, and 14," NACA TN-4234, March 1958.

9. McDevitt, J. B., and Taylor, R. A., "Force and Pressure Measurements at Transonic Speeds for Several Bodies Having Elliptical Cross Sections," NACA TN-4362, September 1958.

10. Page, W. A., "Experimental Study of the Equivalence of Transonic Flow About Slender Cone-Cylinders of Circular and Elliptic Cross Section," NACA TN-4233, April 1958.

11. McCoy, Robert L., U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, Maryland. Private communication.

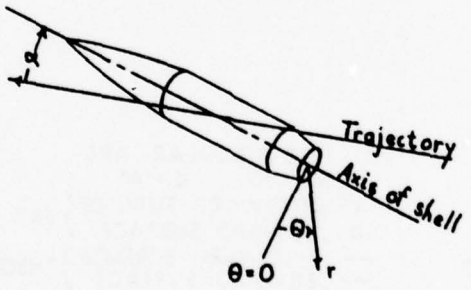


Figure 1. Coordinate System

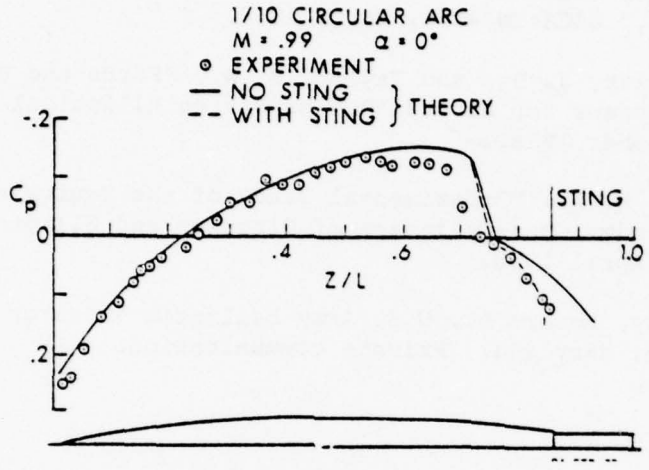


Figure 2. Comparison of Calculated Pressure Coefficients With Wind Tunnel Data for a Fineness Ratio 1/10 Circular Arc Body, M = 0.99

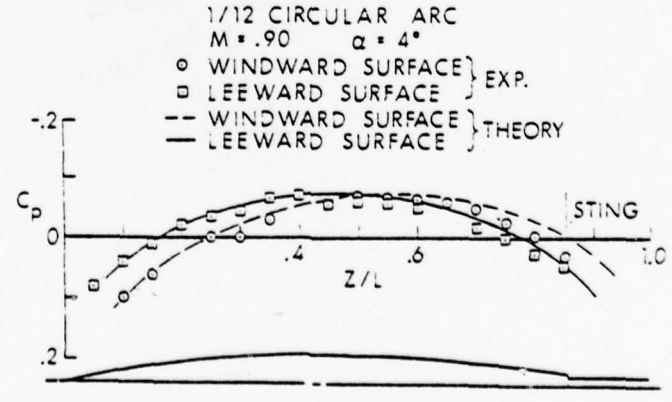


Figure 3. Comparison of Calculated Pressure Coefficients With Wind Tunnel Data for a Fineness Ratio 1/12 Circular Arc Body at Angle of Attack, $\alpha = 4^\circ$, M = 0.90

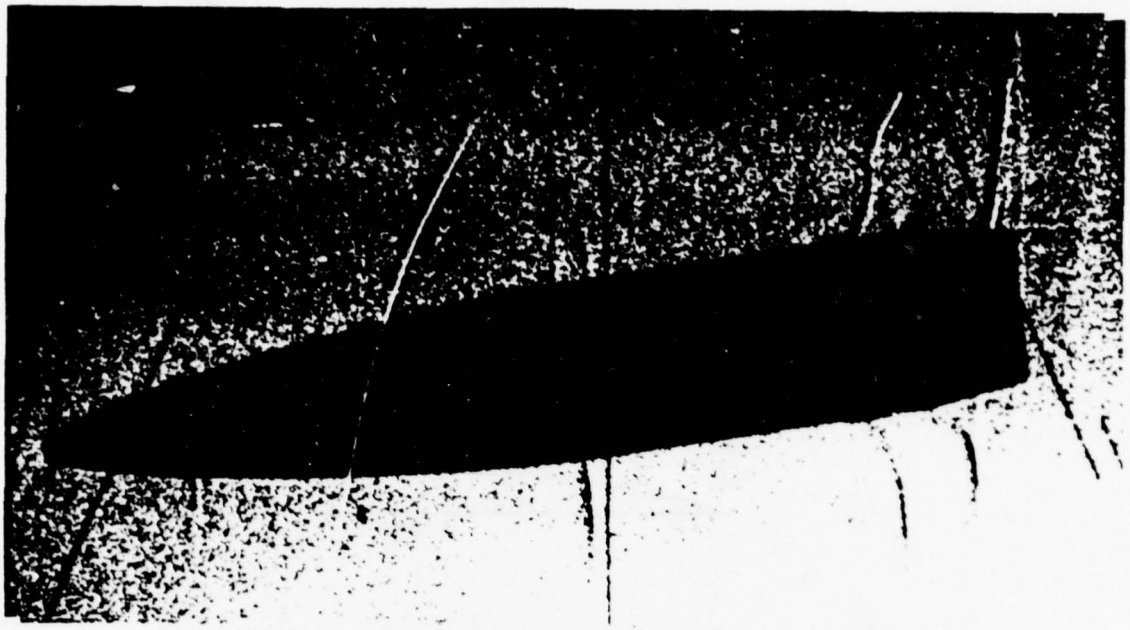


Figure 4. Spark Shadowgraph of a Typical Projectile at Critical Mach Number, M = .926, $\alpha = 8^{\circ}$

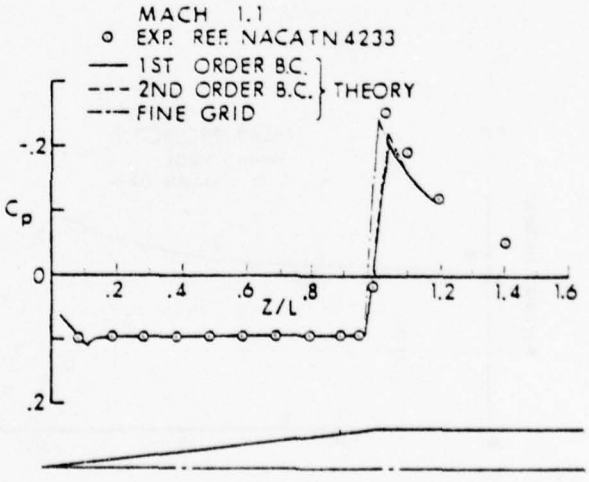


Figure 5. Comparison of Calculated Pressure Coefficient With Wind Tunnel Data for a 7° Half Angle Cone Cylinder, M = .99

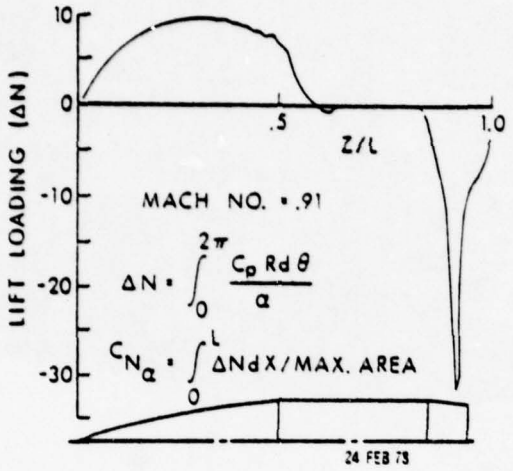


Figure 6. Computed Normal Force Loading Along a Typical Projectile

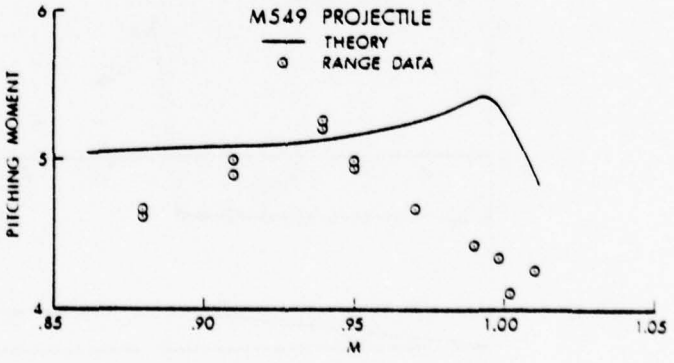


Figure 7. Comparison of Computed Pitching Moment With Range Data for A Typical Projectile

NUMERICAL SIMULATION OF ELECTRO-CHEMICAL MACHINING

Gunter H. Meyer
School of Mathematics
Georgia Institute of Technology
Atlanta, Georgia 30332

ABSTRACT. Anode erosion in an electrolytic cell can be described by a free boundary problem for the potential equation. It is shown that the method of lines solution technique allows the numerical simulation of electro-chemical machining of anodes for a variety of tool and work piece shapes.

1. INTRODUCTION. While electro-chemical machining has become a common metal forming process, its numerical simulation remains a difficult and largely unsolved problem. Yet full scale simulation can aid in choosing tool shapes and feed rates and thus have a direct influence on the design and utilization of electro-forming machinery. The lack of simulation appears to be due not so much to the difficulty of building a mathematical model for the process but to the difficulty of solving the model equations for realistic applications.

Ideally, one would like to determine the tool shape necessary to produce a specified work piece. Unfortunately, such prediction would have to be based on an initial value problem for the potential equation which is mathematically unstable. To date such formulation remains practically unsolvable. On the other hand, the prediction of the work piece shape for a given tool leads to a stable mathematical boundary value problem with a moving surface for which numerical methods are now becoming available.

It is the purpose of this note to indicate how a fairly simple numerical method for the solution of the potential equation can be used to predict the shape of the work piece under dynamic working conditions. The method is implicit in time and yields simultaneously the potential in the electrolyte and the anode surface. Implicit methods as a rule allow larger time steps and more irregular anode shapes than competing explicit methods where the anode is predicted and the field equation is solved in the predicted gap between the electrodes. Moreover, neither special scaling of the geometry is required to avoid the spurious

Research sponsored by the U. S. Army Research Office under Grant DA-AG29-76-G-0261.

solutions possible in the integral equation method of [1], nor is the method restricted to nearly circular electrodes as required for a perturbation approach [4]. In fact, quite arbitrary electrodes can be examined with the same computer program so that digital simulation appears to be considerably more economical and flexible than the analog simulation outlined in [2].

2. THE MODEL EQUATIONS AND THE NUMERICAL METHOD. In order to demonstrate the numerical technique and to compare our results with those obtained recently by two different methods we shall consider the case of a circular cathode surrounding an irregular anode. Electrolyte flows in the axial direction at sufficiently high velocity to be considered of constant conductivity. The electrodes are taken to be equipotentials. Following [1] and [4] we write

(2.1a) $\nabla\cdot\nabla V = 0$ in the electrolyte

(2.1b) $V = 0$ on the circular cathode $r = R$

(2.1c) $V = 1$ on the anode

(2.1d) $(\frac{dx}{dt}, \frac{dy}{dt}) = \nabla V$ on the anode.

where t is a dimensionless variable scaled according to

$$t = \frac{MV}{\alpha^2 C^2}\tau .$$

Here τ is the actual time, V is the anode potential, $M = \frac{e\sigma}{\rho}$ is the machining constant depending on the electro-chemical equivalent e and the density ρ of the anode and on the conductivity σ of the electrolyte, and C_1 is the cathode radius. α is a numerical scale factor of no importance to our method. An initial anode shape $r = s(\theta,0)$ is given. It is assumed that throughout the machining process the anode can be expressed unambiguously in polar coordinates as $r = s(\theta,t)$.

The method of lines algorithm introduced in [3] is suggested for the solution of the free surface problem (2.1). We shall advance the solution $\{V(r,\theta), s(\theta,t)\}$ in discrete time steps of length Δt by discretizing θ and solving (2.1) as a function of r along the rays $\theta = \theta_i = \frac{i2\pi}{N}$, $i = 1,\cdots,N$. As is common in the method of lines we write

(2.2a) $$\frac{1}{r}(rV_i')' + \frac{1}{\Delta\theta^2 r^2}[V_{i+1} + V_{i-1} - 2V_i] = 0$$

(2.2b) $$V_i(R) = 0,\ V_i(s_i) = 1$$

where V_i is the approximate potential along the ray $\theta = \theta_i$. In order to find the motion of the free surface in the radial direction we write (2.1d) as

$$(2.2c) \qquad \left(\frac{\partial V}{\partial r}, \frac{1}{r}\frac{\partial V}{\partial \theta}\right) = \left(\frac{dr}{dt}, r\frac{d\theta}{dt}\right).$$

Since $r = s(\theta,t)$ denotes the anode surface the chain rule yields

$$\frac{\partial V}{\partial r} = \frac{dr}{dt} = \frac{\partial s}{\partial \theta}\frac{d\theta}{dt} + \frac{\partial s}{\partial t} = \frac{\partial s}{\partial \theta}\frac{1}{r^2}\frac{\partial V}{\partial \theta} + \frac{\partial s}{\partial t}.$$

It is convenient (but not essential) to replace $\frac{\partial V}{\partial \theta}$. Since $V = 1$ it follows that the tangential derivative

$D_t V \equiv \left\langle \left(\frac{1}{r}\frac{\partial s}{\partial \theta}, 1\right), \left(\frac{\partial V}{\partial r}, \frac{1}{r}\frac{\partial V}{\partial \theta}\right)\right\rangle$ vanishes on the anode. Solving for $\frac{\partial V}{\partial \theta}$ and substituting into (2.2c) we obtain the following expression for the movement of the anode along the ith ray

$$\frac{\partial s}{\partial t} = \left(1 + \left(\frac{1}{s}\frac{\partial s}{\partial \theta}\right)^2\right)\frac{\partial V}{\partial r}(r,\theta_i)$$

which after time and angle discretization leads to

$$(2.2d) \qquad V_i'(s_i) = (s_i - s_i(t-\Delta t))/\Delta t\left(1 + \left\{\frac{1}{s_i}\left(\frac{s_{i+1} - s_{i-1}}{2\Delta\theta}\right)\right\}^2\right)$$

The numerical solution of (2.2a,b,d) is based on the algorithm described in [3] which is included here for the sake of completeness. An initial guess is chosen for the potential $V_i(r)$ and the initial anode location s_i where $i = 1,\cdots,N$. This guess is usually the solution from the preceding time level, but any reasonable choice will work. Then the one dimensional equation

$$(2.3a) \qquad \tilde{V}_i'' + \tilde{V}_i' + \frac{1}{(r\Delta\theta)^2}\left[V_{i+1} + V_{i-1} - 2\tilde{V}_i\right] = 0$$

$$(2.3b) \qquad \tilde{V}_i(R) = 0, \quad \tilde{V}_i(s_i) = 1$$

$$(2.3c) \qquad \tilde{V}_i'(s_i) = \frac{s_i - s_i(t-\Delta t)}{\Delta t}\Bigg/\left(1 + \left[\frac{s_{i+1} - s_{i-1}}{2\Delta\theta s_i}\right]^2\right)$$

on the interval $[s_i,R]$ is solved for $\tilde{V}_i(r)$ and s_i with the sweep method given below. Once $\tilde{V}_i$ is found it is used to replace $V_i(r)$ according to the formula

$$V_i(r) \leftarrow V_i(r) + \omega[\tilde{V}_i(r) - V_i(r)],$$

where ω is a relaxation factor, while the old value of s_i is replaced by that just found. We cycle through the rays until V_i and s_i remain stationary for $i = 1,\cdots,N$.

The solution of (2.3a,b,c) is based on the so-called method of invariant imbedding. It is well known that $\tilde{V}_i$ and $\tilde{V}_i'$ are related through the Riccati transformation

$$\tilde{V}_i = U(r)\tilde{V}_i' + w(r) \tag{2.4}$$

where $U(r)$ and $w(r)$ are the solutions of the following well defined initial value problems

$$U' = 1 + \frac{1}{r}U - \frac{2}{(r\Delta\theta)^2}U^2 \ , \ U(R) = 0 \tag{2.5}$$

$$w' = \frac{U(r)}{(r\Delta\theta)^2}[2w - V_{i+1}(r) - V_{i-1}(r)], \ w(R) = 0. \tag{2.6}$$

It follows from (2.4) and (2.3b,c) that the boundary s_i must be chosen so that

$$1 = U(s_i)\frac{s_i - s_i(t-\Delta t)}{\Delta t} \Big/ \left(1 + \left[\frac{s_{i+1} - s_{i-1}}{2\Delta\theta s_i}\right]^2\right) + w(s_i) \ .$$

Hence, as the equations (2.5) and (2.6) are integrated we monitor the functional

$$\phi(r) = U(r)\left[\frac{r - s_i(t-\Delta t)}{\Delta t} \Big/ \left(1 + \left[\frac{s_{i+1} - s_{i-1}}{2\Delta\theta r}\right]^2\right)\right] + w(r) - 1 \ .$$

Where it passes through zero the anode surface s_i is placed.

3. A NUMERICAL EXAMPLE. When the electrodes are concentric circles a closed form solution is available for the position of the anode surface [2]. Sample computations with the data of [1] gave answers within fractions of one percent of the analytic result. For example, advancing the anode from an initial position $s(\theta,0) = 9.0$ to a final position $s(\theta,2.5) = 7.4374$ in 10 equal time steps gave a relative error of .34%. In this one dimensional case the number of rays does not influence the final result. In order to present a more complex application the notching of an initially circular anode due to a circular cathode was computed.

It is assumed in the scaled variables of (2.1) that $R = 10$, $s(\theta,0) = 9.5$ and that $t \in [0,2]$. It is further assumed that the anode surface is masked everywhere except for $\theta \in (\frac{3\pi}{8}, \frac{5\pi}{8})$ and $\theta \in (\frac{11\pi}{8}, \frac{13\pi}{8})$ so that it can erode only in these two segments. No movement across the bounding rays such as $\theta = \frac{3\pi}{8}$ was permitted and symmetry about $\theta = 0$ and $\theta = \frac{\pi}{2}$ was used to restrict the calculation to the first quadrant. Fig. 1 shows a typical result when the equations (2.5,6) are solved exactly as described in [3]. In order to accentuate the erosion of the anode in the plot, the radius is scaled according to $r_{plot} = \ell n(\frac{r_{actual}}{7.0})$. The computed depth of the notch at $\theta = \frac{\pi}{2}$ and $t = 2.0$ was $s(\frac{\pi}{2}, 2) = 7.88$.

4. EXTENSIONS. The algorithm was presented for the model equations (2.1) to allow a simple description of the method of lines approach. However, the algorithm applies to general field equations subject to nearly arbitrary boundary conditions. Thus it is possible to use electrodes which are not necessarily equipotentials, to account for electrolytes with variable conductivity, and to model in more detail the erosion of the anode. The method is also applicable in three dimensions. These extensions are obtainable for moderate cost in program complexity but will significantly affect the computation times.

References

1. S. Christiansen and H. Rasmussen, Numerical solutions for two-dimensional annular electrochemical machining problems, J. Inst. Maths. Applics. 18 (1976), pp. 295-307.

2. A. E. De Barr and D. A. Oliver, edts., Electrochemical Machining, MacDonal, London 1968.

3. G. H. Meyer, The method of lines for Poisson's equation with nonlinear or free boundary conditions, Numer. Math. 29 (1978), pp. 329-344.

4. H. Rasmussen and S. Christiansen, A Perturbation Solution for a Two-dimensional Annular Electrochemical Machining Problem, J. Inst. Maths. Applics. 18 (1976), pp. 149-153.

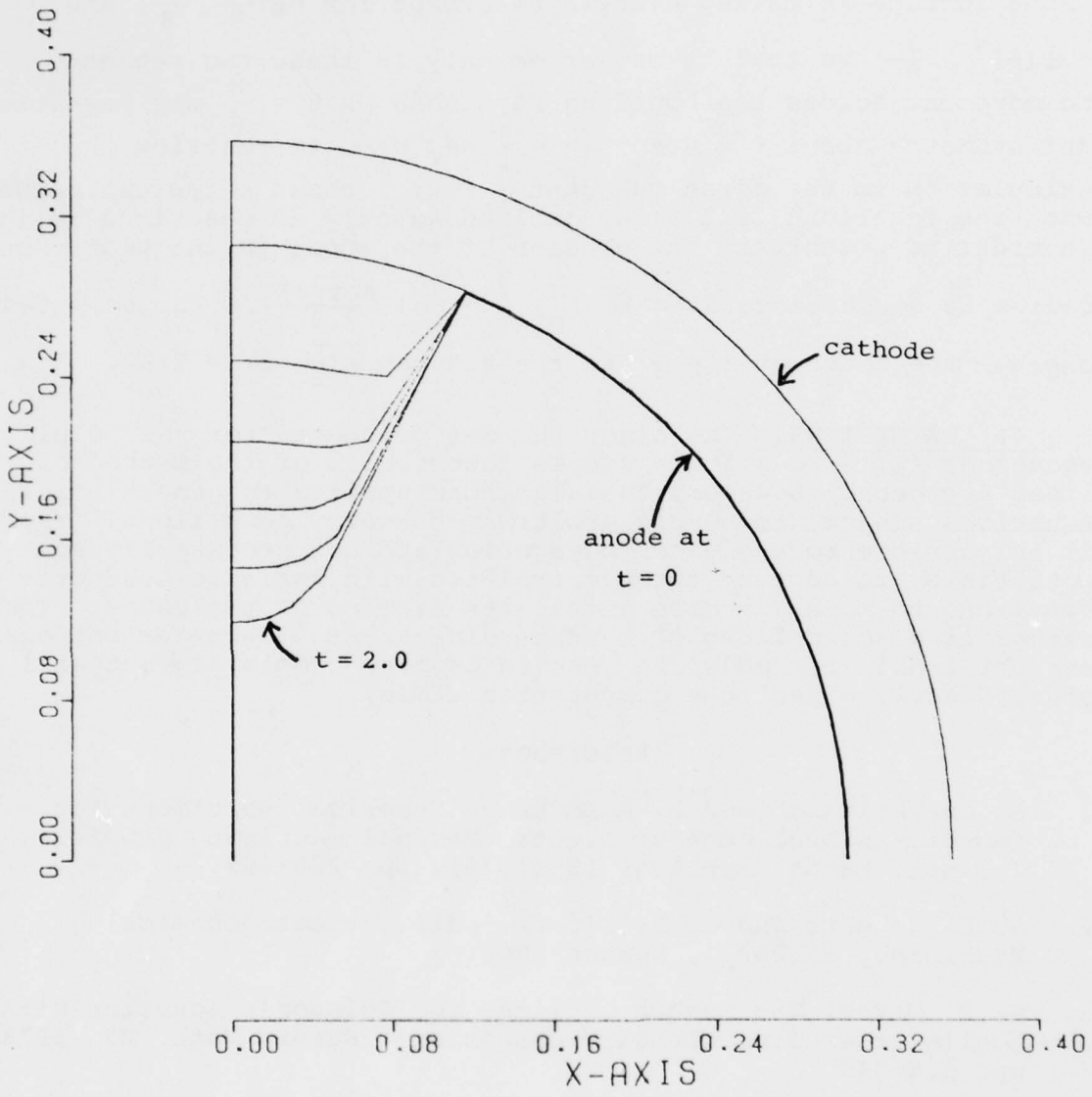


Fig. 1. Scaled plot of the anode and cathode surfaces during electro-chemical erosion. The anode is plotted after every ten time steps.
$\Delta r = 0.025$, $\Delta\theta = \pi/40$, $\Delta t = 0.04$, $\omega = 1.3$. After about 10 SOR iterations per time step the anode surface and the potential in the electrolyte changed by less than 10^{-6}.
CDC Cyber 74 computer time: 100 secs.

The CEMA Ryad Computer Family

S. E. Goodman

Science Division
U. S. Army Foreign Science and Technology Center

Department of Mathematics and the Woodrow Wilson School of
Public and International Affairs
Princeton University

Department of Applied Mathematics and Computer Science and the
Center for Russian and East European Studies
University of Virginia

ABSTRACT. Ryad is the popular name for the Unified System, the Soviet Bloc's upward compatible family of third generation mainframes that are an effective functional reverse engineering of the IBM S/360. Although backward by current Western and Japanese standards, the Unified System is of considerable technological and economic importance. This paper is a brief introduction to the Ryad project.

I am a consultant for the U. S. Army Foreign Science and Technology Center on the subject of Soviet computing. This brief introduction to Ryad is part of a more comprehensive study of the Unified System project that I have been working on with Norman Davis of the CIA [Davis and Goodman 78]. The views expressed here do not necessarily reflect official opinion or policy of the US Army or the CIA.

During the last 10 years the USSR and its CEMA* allies have designed, developed and put into production a family of upward compatible third generation computers known as the Unified System (ES) or Ryad**. This system is an effective functional reverse engineering of the IBM S/360 series, and provides Soviet Bloc users with unprecedented quantities of reasonably good general purpose hardware and software.

In keeping somewhat with the theme of this conference, I will try to direct this short paper a bit towards software considerations.

---

*Council for Economic Mutual Assistance: Bulgaria, Czechoslovakia, German Democratic Republic (GDR), Hungary, Poland and the USSR. Cuba, Mongolia and Romania have weaker affiliations.

**ES is a transliterated abbreviation of the Russian for Unified System. The Cyrillic abbreviation, EC, and an alternate transliteration, YeS, are also commonly used. Language differences among the participating countries produce other variants; for example, the Polish abbreviation is JS. Ryad (alternate transliteration: Riad) is the Russian word for "row" or "series". The prefix R is sometimes used to designate computer models.

The poor state of Soviet and East European hardware before Ryad made it difficult to develop large, modern software systems for general purpose computing. Of several major handicaps, three are particularly important:

(1) Small primary memories. Most machines were provided with no more, and frequently much less, than 32K words of poor quality core memory.
(2) For all practical purposes, disk storage was not widely available until 1973. Secondary storage was generally on poor quality magnetic tape units.
(3) There was a lack of suitable and reliable peripherals. Card readers and alphanumeric printers were not generally available untilthe mid-to-late 1960s. The units that were later produced, and their associated paper products, were of notoriously poor quality and reliability.

This situation was made worse by hardware vendor practices. They delivered nearly empty machines. Users had to write all but the most basic utility programs. Furthermore, the users had to maintain the hardware themselves. This eventually led to local engineering modifications that made it difficult or impossible for users with the same basic CPU model to share software.

Because of these and other factors, the pre-Ryad software situation could be summarized as follows:

(a) Software existed in the form of many isolated pockets of machine language programs. There was very little portability.
(b) Computer centers were essentially on their own once the hardware was delivered.
(c) Many applications, especially those relating to non-numeric computing, were out of the range of the hardware.
(d) Little experience had been built up in the development of large, modern software systems.
(e) Computers were not accessible to users who had not had much technical training.

During the early 60s the Soviets began to appreciate the need for an upward compatible family of general purpose computers for the management of assorted hierarchies of personnel files and transportation networks. A Soviet designed series and a first attempt to duplicate the S/360 architecture had both essentially failed by the late 1960s.

The Unified System represents a much more serious attempt. When first announced in 1967, Ryad was a Soviet project that was almost certainly to be based

on a Soviet design. The USSR had good political, economic, technical and military reasons to want to turn this into a joint CEMA effort that would provide the Soviet Bloc with a uniform hardware and systems software base. By 1969 the Soviet Union had succeeded in coercing and cajoling most of the rest of CEMA into joining the project. The Bulgarians and East Germans were the most cooperative. The Poles, Hungarians, and Czechoslovakians were less than thrilled with the plan. Romania remains a holdout; it has little more than a superficial relationship to the Ryad project.

The decision to copy the S/360 architecture was made shortly thereafter. The GDR was the major advocate of this course of action. This decision was primarily a reflection on the East-West software gap. These countries had not had much experience in developing large, modern software systems nor had they accumulated a large, economically significant collection of applications software. The plan was to expropriate the IBM S/360 operating systems. Then with this base of systems software, they would be in a position to borrow the huge quantities of other systems and applications software that had been developed over the years by IBM and its customers. This plan has been followed with considerable success and represents the most impressive technology transfer in Soviet history.

The effort put into the development and production of the Unified System was considerable. Together the CEMA countries invested the efforts of 70-80 research and production enterprises and over 300,000 scientists, engineers and skilled workers.

Ryad production started in early 1972. The first group of ES models, the Ryad-1s, are listed below along with their IBM S/360 equivalents. Further details are provided in Table I.

ES-1010 (Hungary)	- French Mitra 15
ES-1021 (Czechoslovakia)	- Production model of EPOS-2
ES-1020 (Bulgaria, USSR)	- 360/30
ES-1030 (Poland, USSR)	- 360/40
ES-1040 (GDR)	- 360/50 to 65
ES-1050 (USSR)	- 360/65 to 75
ES-1060 (USSR)	- 360/75 to 85

The 1010 and 1021 are not part of the "real" upward compatible ES family. The Hungarian 1010 is the French Mitra 15 minicomputer produced under license (the Mitra 15 is itself a licensed version of the SDS Sigma 5). The Czech 1021 is based on a local design and none are known to be used outside of Czechoslovakia. The inclusion of these two machines represents a political compromise that was necessary to get some kind of non-trivial Ryad participation from these two countries.

Model	ES-1010	ES-1021 (1020A)	ES-1020	ES-1030	ES-1040	ES-1050	ES-1060
Responsible country	Hungary	Czech.	Bulgaria USSR	Poland USSR	GDR	USSR	USSR
Processor							
Operating speed*(K opns/sec)	10	40	20	100	320	500	1500
Selected performance times (μsec)							
Short operations	1.0-3.0	15-30	20-30	5-11	0.9-1.8	.65-2.0	0.25-0.30
Floating point add/sub.	2.6-3.6	n/a	50-70	10-16	2.6-3.5	1.4-2.4	0.8-1.0
Fixed point multiply	4.0-38	80-120	220-350	35	5.5-13.1	2.0-2.4	1.5-1.8
Floating point divide	n/a	n/a	400	50	10.4-20.3	7.2	3.0-4.0
Instruction set	Special Instr. Set 55 (86) Instruct.	Partial Compatibility Special Instr. 66 (71) Instruct.	----------Complete Program Compatibility----------				
			----------IBM S/360 Instruction Set----------				
Principle of processor control	----------Microprogram----------				Microprog. Hardware	Hardware	Hardware
Primary memory							
Capacity (K bytes)	8-64	16-64	64-256	128-512	128-1024	128-1024	256-2048
Cycle time (μsec)	1.0	1.5	2.0	1.25	1.35	1.25	0.6
Length of accessed word (bytes)	1	1	2	4	8	8	8
Channels							
Selector channels							
Number	1	2	2	3	6	6	6
Transmission rate (K byte/sec)	240	120-300	120-300	600	1200	1300	1300
Multiplexor channel							
Transmission rate in multiplex mode (K byte/sec)	40	35	10-16	40	110	110	150

Basic peripheral configurations**							
Magnetic tape units			4	4	8	8	8
Magnetic tape control units			1	1	1	1	1
Magnetic disk units	1	2	2	2	6	5	7
Magnetic disk control units	1	1	1	1	1	1	2
Punched card readers		1	1	1	1	2	2
Punched tape readers	1		1	1	1	2	2
Card punches			1	1	1	2	2
Tape punches	1		1	1	1	2	2
Printers		1	1	1	1	2	2
Typewriters	1	1	1	1	1	2	2

* This is a common Soviet measure of overall performance that has not been precisely defined. It appears to be based on a mix weighted heavily towards the fastest instructions. Performance for several explicit mixes are given in [GDR 76].

**These configurations were used during acceptance testing. The ES-1060 configuration is projected. CRT units, plotters, terminals and other devices are available on a very limited basis.

Sources: [ES EVM 73, UCS 73, SCR 74, Mayorov 75, Bratukhin 76, GDR 76]. There were some significant differences among the numbers given by these sources.

Selected Characteristics of Ryad-1 Computer Systems

Table 1

With the possible exception of the 1040, all of the ES models had severe birth pains and their introduction was spread out over a longer period than their S/360 counterparts. Initial production rates were about 10% of those for the S/360 and later rates rose to around 15-20%. Not surprisingly, like IBM the CEMA countries had their worst problems at the high end of the line. The 1050 did not appear in a viable form until 1976 and, after much delay, production of the 1060 was finally announced at the end of 1977 (although there are no known installations as of February, 1978).

This first group of Ryads has since been followed by several interim models. These are shown in Table II. All are essentially straightforward upgrades of Ryad-1 models by their design and production plants. The Poles had never really developed a production version of their 1030; they dragged their heels and continued to devote their attention to the ICL-like ODRA project. The 1032 is compatible with the main group of ES models and represents Poland's real entry into the Unified System mainframe business.

The CEMA countries are now developing a new group of Ryad-2 models that will have much the same relationship to the earlier Ryads as the IBM S/370 has to the S/360. Selected design parameters for the new machines are given in Table III. New features to be available in the new members of the Unified System include much larger primary memory, semiconductor primary memory, virtual storage, block multiplexor channels, relocatable control storage, improved peripherals, and expanded system timing and protection facilities. There are also plans for dual processor systems and extended teleprocessing capabilities. By early 1977 most of the new models were well into the design stage. The appearance of prototypes and the initiation of serial production will probably be scattered over the next five years. Prototypes for the 1035 and 1055 may appear during 1978. The Hungarians plan to continue with the development of new minicomputers under the Unified System program.

The secondary storage and peripherals available with the Ryad-1 and interim models were similar to those produced by IBM in the mid-to-late 60s. Peripheral reliability is still not up to S/360 standards. Not surprisingly, disk technology has been a major problem. Until around 1976, 7.25M byte disk packs were the only units available. Now 30M byte units are slowly becoming available, but are hardly in widespread use. Efforts are in progress to master 100M byte unit production. Surprisingly, the Bulgarians have concentrated on disk technology and have emerged as the Communist world's disk specialists. Even the East Germans have essentially terminated their disk development effort in favor of the Bulgarian units. The USSR is the only country to maintain an indigenous capability in all areas of secondary storage and peripheral devices; this apparently reflects the usual conservative paranoia of the Soviet military. Characteristics of commonly used secondary storage and peripheral

Model	ES-1022	ES-1032	ES-1033	ES-1012
Responsible Country	Bulgaria USSR	Poland	USSR	Hungary
Processor				
Operating speed* (K opns/sec)	80	200	200	
Selected performance times (μ sec)				
Short operations	9	2.5-4.0	1.4-2.7	2.6
Floating point add/sub.	30	4.5	4.5	n/a
Fixed point multiply	80	9.0	8.5	8.5
Floating point divide	100	14.0	17.7	n/a
Instruction set	-----IBM S/360 Instruction Set--			Special 109 Instr.
Principle of processor control	----Rigid Microprogram----------			
Primary memory				
Capacity (K bytes)	128-512	128-1024	256-512	8-64
Cycle time (μ sec)	2.0	1.2	1.2	1.0
Length of accessed word (bytes)	4	4	4	2
Channels				
Selector channels				
Number	2	3	3	
Transmission rate (K byte/sec)	500	1100	800	
Multiplexor channel				
Transmission rate in multiplex mode (K byte/sec)	40	110	70	40

*See Table I

Sources: [Kamburelis 75, Bratukhin 76, GDR 76, Budapest 77]. There were some significant differences among the numbers given by these sources.

Selected Characteristics of Interim Ryad Computer Systems

Table II

Model	ES-1025	ES-1035	ES-1045	ES-1055 (without buffer)	ES-1055 (with buffer)	ES-1065***
Responsible Country	Czech	Bulgaria USSR	Poland	GDR	GDR	USSR
Processor						
Operating speed* (K opns/sec)	30-40	100-140	400-500	450	750	4000-5000
Selected performance times (μ sec)						
Short operations	5-18	2.6-4.5	0.6-2.2	0.6-3.9	0.3-2.2	0.12
Floating point add/sub	50-55	9.7	1.9-2.3	1.6-3.6	1.3-1.6	0.24
Fixed point multiply	95-220	23	2.8-3.4	3.4-5.2	3.1	0.6
Floating point divide	225-235	32	8.4-11	4.1-6.0	3.9	1.2
Instruction set	----------IBM S/360 Instruction Set----------					
	----------Some additional IBM S/370-like commands----------					
Principle of processor control	----------Microprogram----------					
Working memory**						
Primary memory capacity (K bytes)	128-256	256-512	256-3072	256-4096	256-4096	1-16 M byte
Virtual memory	----------up to 16 M bytes----------					
Buffer memory	----------8 K bytes available----------					

*See Table I

**Peripheral configurations that we have seen in print closely resemble those given in Table I [Bratukhin 76]. The Czech and GDR models do not include papertape equipment.

***The ES-1060 has gravitated to the Ryad-2 group. See Table I .

Source : [Bratukhin 76]

Selected Characteristics of Ryad-2 Computer Systems

Table III

devices may be found in [CSTAC I 78]. New devices projected for the Ryad-2 models are described in [CSTAC II 78].

In spite of these negative comparisons with respect to IBM, the fact remains that the Soviets and their CEMA partners are doing well with respect to their own past. They have done much to correct the major data processing hardware deficiencies mentioned earlier.

The CEMA countries did not have an easy time adapting the IBM operating systems to the Ryad hardware. They consistently underestimate the difficulties associated with the development and maintenance of software even more than we do. They got DOS up reasonably quickly, but had a lot more trouble with OS. The ES hardware is not as fully family compatible as the S/360 hardware and this may have necessitated the separate adaption of DOS and OS to each of the Ryad models (excluding the 1010 and 1021 of course). In any case, the effort seems to have taken them more time than it took IBM to build these operating systems in the first place. Continued maintenance is nowhere near Western standards. The East Germans and Hungarians seem to do the best job of taking care of their systems software, the Soviets the worst.

The basic Ryad plan was very conservative. It was to functionally duplicate S/360 and to make the hardware useful to the general economy as quickly as possible. Lots of resources were committed and priorities came down from the highest Party and government levels. As far as we can tell, the Soviet Bloc made no effort to do any more than this -- for example, to incorporate S/370-like features into the early ES models. It is moot to speculate as to whether they could have done more.

Actually, it is amazing that they were able to succeed as well as they have. The problems that had to be overcome were enormous. There were language barriers, the difficulty of trying to duplicate sophisticated foreign technology, a small computer industry and weak support industries, poor telecommunications and long physical distances, assorted international bad feelings, and an untested control structure supervising many development and production facilities that had not worked together before.

The effective use of the Ryad computers is quite another matter that's the subject of other papers. It will suffice to note here that assorted systemic factors (i.e. those that relate to institutional arrangements and economic practices) make the Soviets their own worst enemy.

BIBLIOGRAPHY

[Bratukhin 76]: Bratukhin, P. I., V. N. Kvasnitskiy, V. G. Lisitsyn, V. I. Maksimenko, Yu. A. Mikheyev, Yu. N. Cherkasov, and A. L. Shchers, Fundamentals of Construction of Large-Scale Information - Computer Networks (D. G. Zhimerin and V. I. Maksimenko, eds.) Statistika, Moscow, 1976.

[Budapest 77]: "The First R-22 Has Been Started Up. Favorable Experiences at the ELGAV", Szamitastechnika, Budapest, March 1977, 3.

[CSTAC I 78]: "COMECON Ryad I Report", Foreign Availability Subcommittee, Computer Systems Technical Advisory Committee (CSTAC), Jan. 10, 1978.

[CSTAC II 78]: "COMECON Ryad II Report", Draft Version, Foreign Availability Subcommittee, CSTAC, Jan. 25, 1978.

[Davis and Goodman 78]: Davis, N. C. and S. E. Goodman, "The Soviet Bloc's Unified Computer System", to appear. Copies are available from the author.

[ES EVM 73]: Collection of several score marketing brochures put together by the Scientific Research Center for Electronic Computing Technology, Moscow, 1973.

[GDR 76]: Ryad Overview, GDR publication Rechentechnik Datenverarbeitung distributed by Memorex Corp., McLean, Va., 1976.

[Kamburelis 75]: Kamburelis, T. and A. Zasada, "The ODRA 1300 and JS 1032 Computer Systems for 1975", Informatyka, Warsaw, Sept. 1975, 1-5.

[Mayorov 75]: Mayorov, S. A., S. A. Krutovskikh, and A. A. Smirnov, Computer Handbook of Engineering, Sovetskaye Radio, Moscow, 1975.

[SCR 74]: "Unified System Compendium", Special Issue, Soviet Cybernetics Review, May-June 1974, 1-57.

[UCS 73]: "Unified Computer System, ESZR" (Catelog of Hardware Equipment), Budapest, Statisztikai Kiado Vallalat, 1973.

SOME ALGORITHMS FOR THE ANALYSIS OF COMPUTER PROGRAMS*

Lloyd D. Fosdick
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

ABSTRACT. The analysis of computer programs is an important part of program translation, error detection, optimization, and documentation. It consists of two distinct activities: the construction of an abstract model of a program, given the program itself in some language such as FORTRAN, and the extraction of information from the program by examination of the model. A labeled, directed graph is a model that is often used. In recent years workers in theoretical computer science have constructed and analyzed algorithms for solving problems on labeled, directed graphs which are directly related to important problems arising in the analysis of computer programs. Some of these algorithms and their applications are described. The discussion does not assume a knowledge of graph theory.

1. INTRODUCTION. The analysis of computer programs is important in error detection, optimization, documentation, translation and many other activities associated with the design, construction, and maintenance of software. It is difficult because of the size and complexity of programs. Generally speaking, program analysis has been an unorganized subject, consisting of a variety of ad hoc techniques with little unifying structure, but this situation is changing. One reason for this change is the development of good algorithms for recognizing the implicit relationships in directed graphs. I will describe some of these algorithms and how they can be used on problems in program analysis. In doing so I hope to provide an indication of how we can begin to organize the subject of program analysis.

I want to make a careful distinction between the analysis of algorithms [1] and the analysis of programs. The analysis of an algorithm starts with the algorithm and focuses on the problem of obtaining time and memory space bounds or expectation values in terms of a few parameters of the underlying problem. The analysis of a program starts with a program, say in FORTRAN or COBOL, and proceeds to some abstract model of it on which the analysis is performed. The "program" is expressed exactly as it will be, or has been, read into the store of the computer. The construction of the abstract model of the program is a critical step in the analysis of programs, critical because decisions must be made about which information is to be discarded and which is to be retained, and the analysis will suffer if too much or too little is discarded. Program analysis is concerned with time and memory space requirements but it is also concerned with far more detailed information than is algorithm analysis; for example, program analysis is concerned with which variables are assigned values in certain areas of the program, which subroutines are called by which other subroutines, which variables depend on which other variables, and so forth. Finally, because of the large amount of information that arises in program analysis, most of the work, if not all, is done by computers whereas the analysis of algorithms is done by humans.

*Supported in part by ARO Grant # DAAG29-78-G-0046.

2. PROGRAM ABSTRACTIONS. The basic structure for a program abstraction is a directed graph [2]. The subject of graphs dates back to Euler and the famous Königsberg bridge problem [3]. Graphs appeared in automatic computing at an early date [4] and since then they have been used extensively to represent machines, programs, data structures and problems attacked by computers such as network flow problems. A directed graph is shown in Fig. 2.1. It is denoted by G(N,E) where N is a set of points called nodes, joined by a set, E, of directed lines, called edges. Letters or numbers are used to identify nodes as in

$$N = \{a, b, c, d, e, f\} \quad \text{or} \quad N = \{1, 2, 3, 4, 5, 6\}$$

and ordered pairs of nodes are used to represent edges, as in

$$E = \{(a,b), (b,c), (b,d), (d,e), (e,d), (d,f), (c,f)\}$$

to represent the set of edges of the graph in Fig. 2.1. The edge (a,b) is said to leave a and enter b. The notation $|S|$ is used to represent the number of elements in the set S; $|N| = 6$ and $|E| = 7$ for the graph shown in Fig. 2.1. A path in a graph is a sequence of nodes joined by edges: for example $a \to b \to d \to e$ and $b \to c \to f$ are paths in Fig. 2.1. Sometimes intermediate nodes along the path are suppressed and the notation $a \overset{*}{\to} e$ is used to denote a path from node a to node e. The length of a path is the number of edges on the path: the length of $a \to b \to c \to f$ is three. A path in which the first and last nodes are the same is a cycle; for example, $d \to e \to d$ and $e \to d \to e \to d \to e$ are cycles in Fig. 2.1. A path such as $a \to b \to d \to e \to d$ in Fig. 2.1 is said to contain a cycle. If no path in the graph is a cycle the graph is called acyclic. If every node, save one called the root, has exactly one edge entering it and the root has no edges entering it, then the graph is a tree. A tree is an acyclic graph but the converse is not true. A picture of a tree and an acyclic graph which is not a tree are shown in Fig. 2.2. If (i,j) is an edge in a tree then node i is called the parent of node j.

It is common practice to represent control flow with a graph called a flow graph. In such a representation the nodes identify statements and the edges identify the order of execution of the statements. When one considers creating such an abstraction mechanically it becomes immediately evident that a simple one-to-one mapping of statements to nodes is not always adequate. An example of the kind of problem that can be encountered is illustrated with the DO statement in FORTRAN which actually consists of three, more elementary, statements separated by a group of statements following the DO. This is illustrated in Fig. 2.3. Even though a node in a flow graph may not represent a statement exactly because of such complications it is convenient, and should cause no confusion here, to speak of nodes in flow graphs as representing statements. A node in a flow graph with no edges entering it is called an entry node, and a node with no edges leaving it is called an exit node. A subroutine in ANS FORTRAN would be represented by a flow graph with one entry node, corresponding to the first executable statement and one or more exit nodes corresponding to RETURN statements. It is customary to restrict flow graphs to have a single entry node and to represent them with the notation $G(N,E,n_0)$ where n_0, an element of the set N, is the single entry node.

The nodes of a flow graph may represent larger structures than individual statements: they may represent statement blocks, cycles, subroutines and so

forth. Each of these is a different program abstraction retaining some aspects of control flow and suppressing others. A statement block is a sequence of statements such that control always flows from one immediately to the next; thus a branch statement can only appear at the end of a block and only the first statement can be branched to by some other statement. The transformation of a flow graph in which nodes represent statements to one in which nodes represent statement blocks is illustrated in Fig. 2.4. The resulting graph is more compact but still shows all branches in control flow and all cycles. The transformation of such a graph into one in which cycles are suppressed by collecting them into single nodes is illustrated in Fig. 2.5. Here cycle information is lost but connectivity relationships between cycles are retained. In graph terminology the nodes of the resulting flow graph represent the strongly connected components of the original flow graph. When the nodes of a flow graph represent subroutines or procedures and the edges represent one or more procedure calls, the graph is named a call graph. This is illustrated in Fig. 2.6. Since FORTRAN prohibits recursion a call graph for a FORTRAN program must be acyclic. The use of a call graph together with flow graphs for individual subroutines naturally partitions the abstract representation of a program into more manageable form for analysis.

Given a flow graph the following questions may be asked about its structure:

1) Is it acyclic?

2) What nodes lie on some path from a given node?

3) Is it possible to construct a path which includes a given set of nodes (edges)?

4) Can you find a path from the entry node to an exit node which does not include both members of certain pairs of edges?

5) Which sets of nodes have the property that there is a path from any node to any other node of the set?

These questions and others are related to important questions that may be asked about a program:

1') Is it possible for this program to loop indefinitely during execution?

2') Can statement s_i be executed before statement s_j?

3') What is the smallest set of test data needed to execute every statement (traverse every edge) once?

4') Can you find an executable sequence of statements?

5') Can statement s_i and s_j be in a cycle?

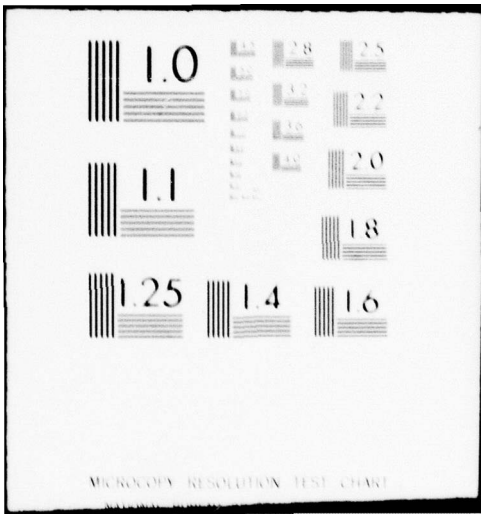
The questions about graph structure clearly can be answered by enumeration since the graphs are finite, but graphs derived from programs may be large and enumeration impractical. Therefore it is important to find algorithms which are significantly faster than enumeration and it is important to know about the complexity of these problems. It is known, for example, that question 4 is a problem that is called NP-complete [5]: this means that the worst case execution time of any algorithm for solving this problem on

AD-A061 561 ARMY RESEARCH OFFICE RESEARCH TRIANGLE PARK N C F/G 12/1
PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS C--ETC(U)
OCT 78

UNCLASSIFIED ARO-78-3 NL

3 OF 4
AD
A061 561




1.0
1.1
1.25
1.4
1.6
2.8
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

arbitrary graphs is almost certainly going to be exponential in the size of the graph as measured by $|N|$ and $|E|$. Other questions in this group can be answered by very fast algorithms; for instance question 5 can be answered by an algorithm that has time complexity $O(|N|+|E|)$.*

Association of information describing data actions with the nodes of a flow graph makes it possible to analyze sequences of data actions, called data flow. For example, consider a particular variable, say X, and label each node with the symbol r, d, or e according as X is referenced (its value is fetched from memory as in Y ← X+1), defined (a value is assigned to X as in X ← Y+1), or X is not referenced or defined. If more than one action on the variable takes place as in X ← X+1 then an appropriate sequence of action symbols, rd, is attached to the node. An example is shown in Fig. 2.7: every path in this labeled graph corresponds to a sequence of data actions as illustrated below for the variables X and Y.

Path	Data Actions
1→2→3→5→7	e d r e (X)
	e e r e e (Y)
1→2→3→4→6→4→7	e d r r r d r e (Y)
	e e r r e r e (Y)

Looking at the data flow described on the right it is evident that Y must be assigned a value before entry to the program since there are paths on which the first action is r. It also appears that X need not be assigned a value before entering the path since it is defined before any reference; further consideration shows that X does not need to be assigned a value before entering any path starting at n_0.

Analysis of data flow can provide answers to the following questions about a program:

1) Are undefined variables referenced?
2) Are there unnecessary definitions of values?
3) Which parameters in a procedure call need to be assigned values before entry?
4) Which parameters in a procedure call may have altered values upon return?

---

*Let $f(|N|,|E|)$ represent the time to execute the algorithm as a function of $|N|$ and $|E|$, then time complexity $O(|N|+|E|)$ means

$$\lim_{|N|+|E|\to\infty} \left(\frac{f(|N|,|E|)}{|N|+|E|} \right) = k, \quad k \text{ a constant unequal to zero.}$$

Time complexity $O(|N|+|E|)$ implies that the approximation $f(|N|,|E|) \cong k \times (|N|+|E|)$ may be used for large $|N|+|E|$.

Analysis of data flow can detect programming errors. If the first data action on a path is r then it is likely that a data initialization has been omitted, or the reference action is incorrect because the variable was misspelled, or the referencing statement is in the wrong place, and so forth. Similarly if a d is followed by a d without an intervening r then a variable may have been misspelled causing the redundant definition or causing the omission of the intervening reference. Analysis of data flow can also assist in the global optimization of programs, in providing documentation aids, and in program modification.

It should be evident from this that algorithms for manipulating graphs and for extracting implied relationships in graphs have many applications in program analysis. However, not all useful abstractions require graphs. In some applications, for example, sets of variables are sufficient: the set of variables declared as formal parameters to a procedure should be a subset of the set of variables used in a procedure; sets of variables which are equivalent in the sense that all variables in the set represent the same memory location are important for a consideration of aliasing. Thus algorithms for set operations are also important for program analysis. Here, however, our attention is directed at graph algorithms.

3. ALGORITHMS. A flow graph contains explicit and implicit information. Explicit information is information associated with a node or edge which is independent of information at other nodes or edges. Thus it is local information about the program, determined at a cost that is independent of the program size. Examples of explicit information are: the statement type; the list of variables appearing in a statement; the branch condition on an edge, for example the fact that $A \geq 0$ is true if the edge is traversed during execution. Implicit information may be associated with a node or edge also, or it may be associated with a larger structure including the entire flow graph: it is dependent on information at more than one node or edge, perhaps all of them. Thus implicit information is global information that may, and usually does, depend on the program's size. Examples of implicit information are: the set of statements which can be reached on all paths from a given statement; the set of variables on which the first data action will be definition on all paths from a given statement; the set of statements which are on all paths to a particular statement. Explicit information is collected at the time the abstract model of the program is created. It is, in fact, part of the model itself. Implicit information is derived from the model. It is the derivation of this implicit information that is the focus of attention in the subsequent discussion.

Two mechanisms are frequently used for deriving information from a flow graph: one consists of performing graph transformations, [6,7] the other consists of performing a search on a graph [8,9]. When graph transformations are used a flow graph $G(N,E,n_0)$ is transformed into another flow graph $G(N',E',n_0')$: the transformation is not only one of structure but also one of information attached to nodes and edges. This approach generally consists of a sequence of transformations, each being an elementary transformation in some sense. Through an appropriately chosen sequence of transformations global information about the program can be collected and distributed to appropriate nodes. A search consists of moving over the graph in a systematic manner dictated by the relationships between nodes implied by the edges. During the search global information about relationships can be collected. Transformations may be used to assist the search, and transformations may be used in place of a search.

Two kinds of search on a flow graph are common: depth-first and breadth-first. A depth-first search is defined by the following algorithm:

Algorithm (DFS)

1. Push the entry node on a stack and mark it (this is the first node visited, nodes are marked to prevent visiting them more than once).
2. While the stack is not empty do
   2.1 If there is an edge from the node at the top of the stack to an unmarked node then push the unmarked node on the stack and mark it else pop the stack.
3. Stop.

A breadth-first search is defined by the following algorithm:

Algorithm (BFS)

1. Put the entry node on a queue and mark it.
2. While the queue is not empty do
   2.1 If there is an edge from the node at the head of the queue to an unmarked node then add the unmarked node to the end of the queue and mark it else remove the node at the head of the queue
3. Stop.

A search defines a numbering of the nodes determined by the order in which they are visited. Two numberings of importance associated with DFS are preorder numbering and postorder numbering: preorder numbering corresponds to the order in which the nodes are first visited in a depth-first search and postorder numbering corresponds to the order in which they are last visited in a depth-first search.* These numberings are illustrated in Fig. 3.1. When a graph is a tree preorder numbering assures that every node has a higher number than its parent and postorder numbering assures that every node has a lower number than its parent. This is illustrated in Fig. 3.2. When an arbitrary graph has a preorder numbering the presence of an edge (v_i,v_j) such that $v_i > v_j$ implies the graph is not a tree but the converse is not true. When an arbitrary graph has a postorder numbering the presence of an edge (v_i,v_j) such that $v_i < v_j$ implies the presence of a cycle; removal of all edges with this property transforms the graph into an acyclic directed graph, but not necessarily a tree. It is important to recognize that a preorder or postorder numbering can be done quickly. The time required for a depth-first search increases linearly with the number of edges in a connected graph: each edge is traversed once in a forward direction and once in the backward direction, thus the time complexity for the DFS algorithm is $O(|E|)$.

---

*There has been some confusion in the literature with this terminology. Our definition corresponds to recent usage [1] but differs from Knuth [10].

It is useful for a number of applications to know whether a pair of nodes can be on a path. This gives rise to the following problems: given a node, u, determine all nodes v such that there is a path from u to v (symbolically this is expressed as the set $S(u) = \{v | u \overset{*}{\rightarrow} v\}$); and given a node, u, determine all nodes v such that there is a path from v to u (symbolically this is expressed as $P(u) = \{v | v \overset{*}{\rightarrow} u\}$). Any algorithm for solving the first problem can be used to solve the second problem simply by reversing the directions of the edges. The reflexive and transitive closure of a graph is defined as the set of all ordered pairs of nodes (u,v) such that there is a path (including a path of zero length) from u to v (symbolically this is $\{(u,v) | u \overset{*}{\rightarrow} v\}$). If paths of zero length are excluded then the set is called the transitive closure (symbolically this is $\{(u,v) | u \overset{+}{\rightarrow} v\}$).

An algorithm by Warshall [11] for computing transitive closure is based on matrix multiplication and has time complexity $O(|N|^3)$. It is not difficult to see that the transitive closure could be computed by performing $|N|$ depth-first searches, one search from each node. This approach would require a time proportional to $|N||E|$. Of course $|E| \leq |N|^2$ so in the worst case the time required would also be proportional to $|N|^3$. But for programs it is more common that $|E| \leq k|N|$ where k is a constant and so the time would be proportional to $|N|^2$.

A clever idea described by Schnorr [12] may allow further improvement of the computation time for transitive closure. Consider a node v of $G(N,E,n_0)$ and suppose $v \overset{*}{\rightarrow} u_1$, $v \overset{*}{\rightarrow} u_2, \ldots, v \overset{*}{\rightarrow} u_k$ where $k = \lceil |N|/2 + 1 \rceil$. Now consider a graph G'(N,E') derived from G by simply reversing the direction of all of the edges. Let w be a node and suppose in G: $w \overset{*}{\rightarrow} r_1$, $w \overset{*}{\rightarrow} r_2, \ldots, w \overset{*}{\rightarrow} r_k$ where k is defined as before. It is easy to see that the sets $\{u_1,u_2,\ldots,u_k\}$ and $\{r_1,r_2,\ldots,r_k\}$ must have at least one node in common since $2 \times \lceil |N|/2 + 1 \rceil$ equals $|N| + 2$ or $|N| + 3$ according as $|N|$ is even or odd. This idea is the basis of an algorithm which uses breadth-first search to compute transitive closure but terminates the search from a node after $\lceil |N|/2 + 1 \rceil$ nodes have been reached, unless it terminates earlier because no more nodes can be reached. Then the idea just described is applied to complete the computation of the transitive closure. The interesting property of this algorithm is that its expected time for execution is $O(|N| + |E|^*)$ where $|E|^*$ is the expected number of edges. The model used for computing the expected values consists of an ensemble of random graphs with $|N|$ nodes and for any pair of nodes u and v there is constant probability p for an edge (u,v). Unfortunately, this ensemble probably does not match the ensemble for real programs very well.

A determination of strongly connected components is useful in some aspects of program analysis. A strongly connected component, SCC, of a flow graph $G(N,E,n_0)$ is a subset of N defined as follows: for every pair of nodes u, v in the SCC there is a path from u to v and v to u in G, and no more nodes may be added to SCC preserving this property. In general a graph may have more than one SCC. Once the SCC's of G have been determined it is possible to make a transformation $G(N,E,n_0) \rightarrow G'(N',E',n_0)$ where the elements of N' are the SCC's of G and the elements of E' represent paths between components implied

by the elements of E: the idea is illustrated in Fig. 2.5. The flow graph G' is acyclic and so a partial ordering of the nodes is possible. In particular, if the nodes of G' are numbered in postorder then on every path, $n_0 \overset{*}{\rightarrow} n$, $n \in N'$, the numbering of the nodes will be in decreasing order. This partial ordering is useful in data flow analysis. A fast algorithm for computing SCC's has been described by Tarjan [13]. In this algorithm a pair of numbers, preorder(n) and lowlink(n), is associated with each node n; preorder(n) is the preorder number defined earlier, lowlink(n) is defined by

$$\text{lowlink}(n) = \min_{S(n)}[\text{preorder}(v)],\ S(n) = \{v \mid n \overset{*}{\rightarrow} v\}.$$

This numbering is illustrated in Fig. 3.3. Tarjan shows that a pair of nodes are in the same SCC if and only if they have the same lowlink number and that the lowlink numbers can be computed using a depth-first search. The time complexity for Tarjan's algorithm applied to a flow graph is $O(|N|+|E|)$.

In testing a program the notion of coverage or completeness of a set of tests is important [14,15]. One measure of test coverage is the percentage of nodes executed or edges traversed in the flow graph. Randomly selected input data for a set of tests will give poor coverage, therefore some care in choosing test data is necessary. In attempting to choose the test data carefully the question of whether it is possible to execute a given set of nodes in a test arises. This question is closely related to the following one about a flow graph of the program: given the flow graph $G(N,E,n_0)$ and a set N', $N' \subseteq N$, is there a path $n_0 \overset{*}{\rightarrow} n$, $n \in N$, which includes every node in N'. Note that if the answer to the graph question is no, then the answer to the question about the program is certainly no; however, if the answer to the graph question is yes one cannot infer that the answer to the question about the program is yes because it may not be possible to satisfy all of the branch conditions as illustrated in Fig. 3.4. Gabow, Maheshwari, and Osterweil [5] have described an algorithm for answering the flow graph question. The idea is based on a consideration of an acyclic directed graph which would result from an arbitrary flow graph if one were to replace all SCC's by single nodes. Note that if any node v in N', the set of nodes to be included in the path, is in a SCC then a path to any member of that SCC can be extended to include v. Thus the original graph question only needs to be asked about an acyclic directed graph. In presenting the algorithm we use the word "frontier" for the set of entry nodes (initially the frontier is $\{n_0\}$) removing a node from the graph implies all edges leaving the node are also removed. Here is the algorithm:

```
until the frontier is empty or the frontier contains
      more than one node in N' do
   if the frontier is a singleton then remove this node
   else remove a frontier node that is not in N';
stop.
```

If the graph is empty when this algorithm terminates then there is a path $n_0 \overset{*}{\rightarrow} n$ which includes all nodes of N', and if the graph is not empty then there is no such path. This algorithm is illustrated in Fig. 3.5. The time complexity for this algorithm is $O(|E|)$ and since the time complexity for getting

the SCC's by Tarjan's algorithm is also $O(|E|)$ it follows that the overall time complexity for answering the original graph question is $O(|E|)$.

Data flow analysis has applications in global program optimization [16,17], error detection [9], discovery of unexecutable paths [18,19], and detection of deadlock [20]. A basic problem in data flow analysis is the so-called live variable problem. This problem can be stated as follows: given a flow graph $G(N,E,n_0)$ with the nodes labeled to show the reference (r) actions and definition (d) actions on a variable x, and given n, $n \in N$, does there exist a path from n such that the first action on x is r. If the answer to this question is yes, then x is said to be live at n, otherwise it is dead at n. If only one variable and one node were involved the easiest way to answer this question would be to conduct a depth-first search from n. However, the normal situation is that this question is to be answered for many variables at all nodes of the flow graph. A number of algorithms for treating this problem have been described in the literature [6,7,8,16,21]. A particularly simple and effective algorithm is the one due to Hecht and Ullman [8]. The main ideas are as follows. We associate three sets with every node: at node n these sets are ref(n), def(n), live(n). The sets are initialized as follows for all n, $n \in N$:

1. ref(n) = {V| V a variable referenced at n}
2. def(n) = {V| V a variable defined at n}

   (for simplicity we assume here that a variable is not referenced and defined at the same node.)
3. live(n) = $\emptyset$ (the empty set)

After this initialization ref(n) and def(n) are not changed, but live(n) is modified by applying the following formula iteratively to the nodes of $G(N,E,n_0)$:

$$\text{live}(n) = \bigcup_{k \in S(n)} \left(\left(\text{live}(k) \cap \neg\, \text{def}(k) \right) \cup \text{ref}(k) \right)$$

where $S(n) = \{j | (n,j) \in E\}$, the set of successors of n. Two examples of the application of this formula are shown in Fig. 3.6. Hecht and Ullman have shown [8] that if this formula is applied iteratively to the nodes of $G(N,E,n_0)$ in postorder, the convergence is quite rapid. In practice the number of iterations can be expected to be less than four or five. The application of this algorithm to the detection of uninitialized variables in entire programs has been thoroughly discussed by Fosdick and Osterweil [9].

There is an interesting connection between an important problem in program analysis and non-linear optimization. I mentioned earlier that a path in a program flow graph may not represent a sequence of statements that could actually be executed: Fig. 3.4 illustrates this situation. Suppose that we are given a path $n_0 \overset{*}{\rightarrow} n$ in a flow graph and we wish to determine whether the path can be executed. To do this imagine moving along the path writing down the predicates that must be satisfied at every branch. In doing this we must take into account changes in values of variables caused by assignments so that a particular variable name represents the same value in every predicate. A necessary and sufficient condition for the path to be executable is that the

system of predicates written down is consistent; that is, there must exist an assignment of values to variables appearing therein such that every predicate is satisfied. Now this issue of consistency is exactly the same one that arises in trying to determine whether there is a feasible region defined by the constraints in a non-linear (or linear) optimization problem. This problem is difficult and there are no really good algorithms for dealing with it. Clarke has discussed the problem and some experience in attempting to solve it in a recent paper [22].

4. CONCLUSION. Recent work has provided a number of algorithms which have important applications in the analysis of computer programs. While much work remains to be done in the development of these algorithms, we are now in a position to build some important program analysis tools based on these algorithms. Such tools could be used for program optimization, error detection, testing, and documentation.

These tools should be organized into a library that is easy to use. This will take careful planning. Consistent patterns of use must be developed, and portability, and adaptability to various language dialects must be taken into account. It is a large and difficult challenge, but one which we should accept.

References:

1. Aho, Alfred V.: Hopcroft, John E.; and Ullman, Jeffrey D. The Design and Analysis of Computer Algorithms. Addison-Wesley (1974).

2. Berge, Claude. Graphs and Hypergraphs, North-Holland (1973).

3. Biggs, Norman L.; Lloyd, E. Keith; and Wilson, Robin J. Graph Theory 1736-1936. Clarendon Press, Oxford (1976).

4. Goldstine, Herman H. and von Neumann, John. Planning and coding problems for an electronic computing instrument. In John von Neumann Collected Works, Volume 5, Abraham H. Taub, Ed., Pergamon Press (1961).

5. Gabow, Harold N.; Maheshwari, Sachindra N.; and Osterweil, Leon J. On two problems in the generation of program test paths. Proc. IEEE Trans. on Soft. Eng. SE-2,3 (Sept. 1976), 227-231.

6. Graham, Susan L.; and Wegman, Mark. A fast and usually linear algorithm for global flow analysis. JACM 23,1 (Jan. 1976), 172-202.

7. Allen, F.E.; and Cocke, J. A program data flow analysis procedure. CACM 19,3 (March 1976),137-147.

8. Hecht, Matthew S.; and Ullman, Jeffrey D. A simple algorithm for global data flow analysis. SIAM J. Computing 4 (Dec. 1975), 519-532.

9. Fosdick, Lloyd D. and Osterweil, Leon J. Data flow analysis in software reliability. ACM Comp. Surveys 8,3 (Sept. 1976), 305-330.

10. Knuth, Donald E. The Art of Computer Programming, Vol. 1. Addison-Wesley (1968).

11. Warshall, S. A theorem on boolean matrices. JACM 9,1 (Jan. 1962), 11-12.

12. Schnorr, C.P. An algorithm for transitive closure with linear expected time. In Lecture Notes in Computer Science, v. 48, G. Goos and J. Hartmanis, Eds., Springer-Verlag (1977).

13. Tarjan, Robert E. Depth-first search and linear graph algorithms. SIAM J. Computing 1,2 (June 1972), 146-160.

14. Brown, John R. Getting better software cheaper and quicker, in Practical Strategies for Developing Large Software Systems, Ellis Horowitz, Editor. Addison-Wesley (1975).

15. Huang, J.C. An approach to program testing. ACM Comp. Surveys 7,3 (Sept. 1975), 113-128.

16. Schaefer, J.C. A Mathematical Theory of Global Program Optimization. Prentice-Hall (1973).

References cont'd.

17. Ullman, Jeffrey D. and Aho, Alfred V. The Theory of Parsing, Translation, and Computing: Volume II. Prentice-Hall (1973).

18. Osterweil, Leon J. The detection of unexecutable program paths through static data flow analysis. Tech. Rept. 110, Dept. of Computer Science, University of Colorado, Boulder, CO. 80309 (May 1977).

19. Bollacker, Lee A. An algorithm for detecting unexecutable paths through program flowgraphs. Tech. Rept. 112, Dept. of Computer Science, University of Colorado, Boulder,CO. 80309 (Jan. 1978).

20. Saxena, Ashok. Static detection of deadlocks. Tech. Rept. 122, Dept. of Computer Science, University of Colorado, Boulder,CO. 80309 (Dec. 1977).

21. Kennedy, Kenneth. Node listings applied to data flow anlaysis. Proc. 2nd ACM Symposium on Principal of Programming Languages. Palo Alto, CA. (Jan. 1975).

22. Clarke, Lori A. A system to generate test data and symbolically execute programs. IEEE Trans. on Software Engineering SE-2 (Sept. 1976), 215-222.

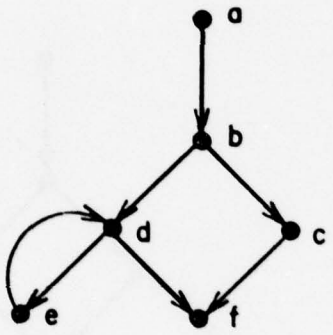


Figure 2.1

A directed graph. The dots represent nodes, the directed lines represent edges. The node set is N = {a, b, c, d, e, f}, the edge set is E = {(a,b), (b,c), (b,d), (d,e), (e,d), (d,f), (c,f)}.



Figure 2.2

Both of these graphs are acyclic but only the graph on the left is a tree. The graph on the right is not a tree because it has a node with two edges entering it.

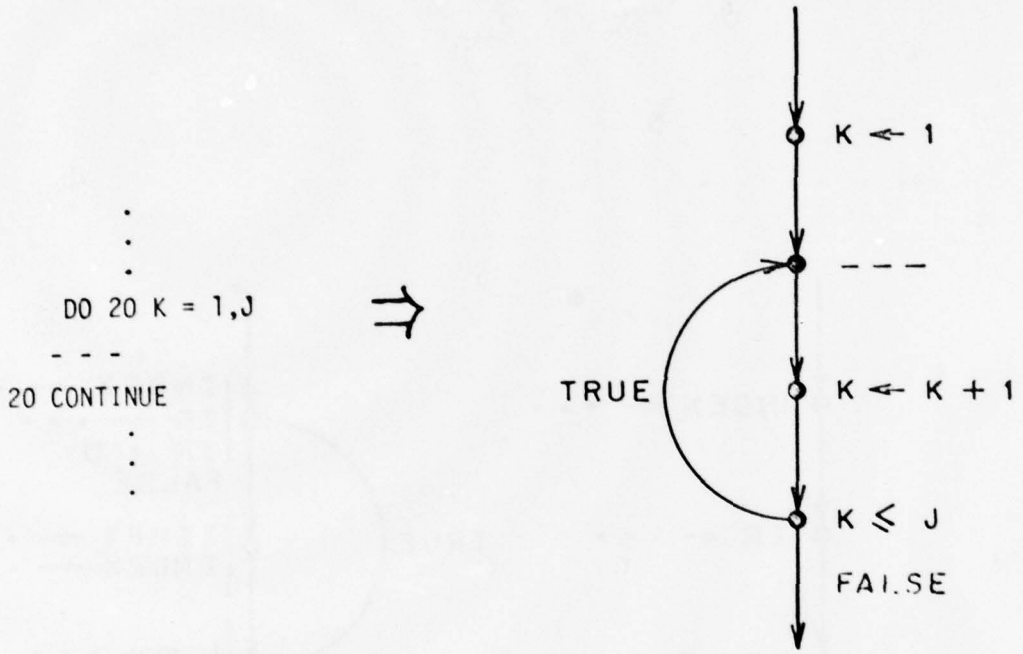
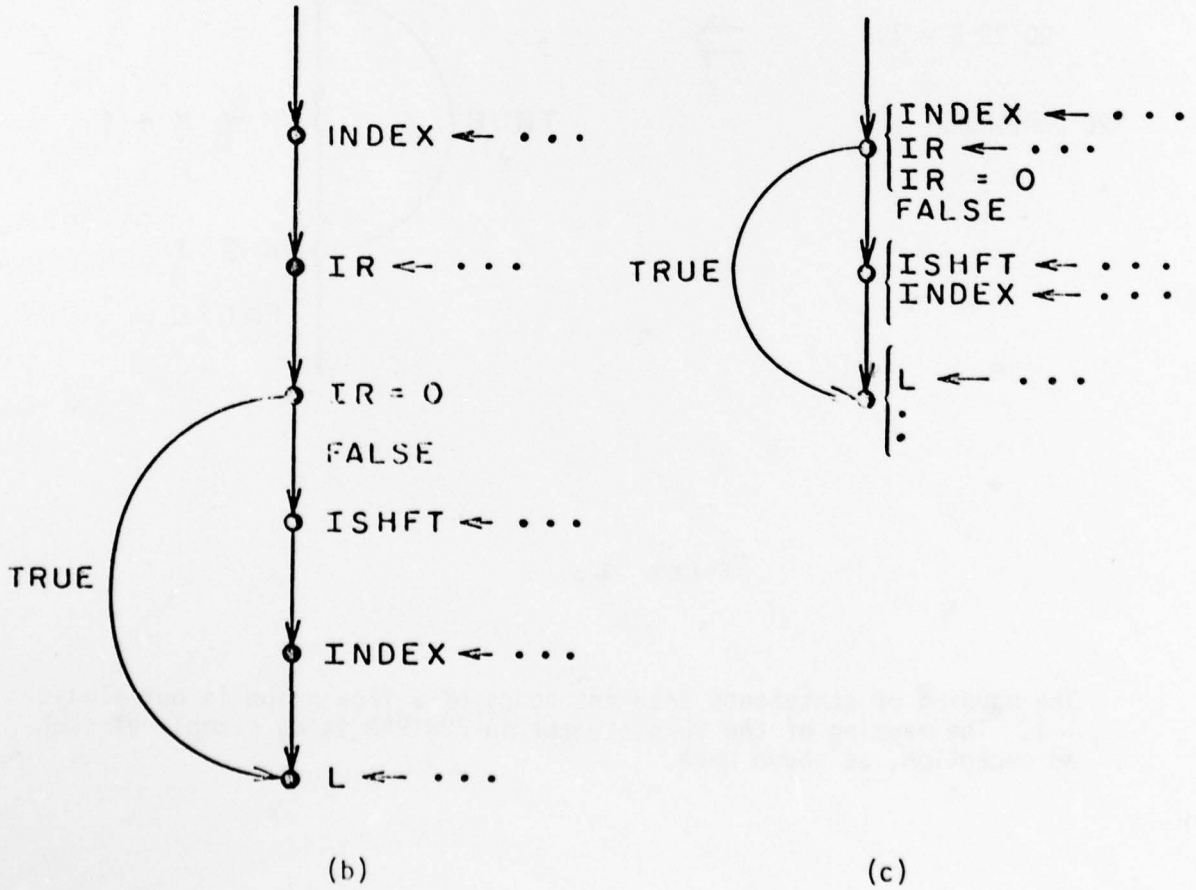


Figure 2.3

The mapping of statements onto the nodes of a flow graph is not always 1-1. The mapping of the DO statement in FORTRAN is an example of such an exception, as shown here.

```
      IF ( - - ) GO TO 90
           .
           .
           .
90    INDEX = J/K
      IR = J - INDEX * K
      IF (IR.EQ.0) GO TO 95
      ISHFT = K - IR
      INDEX = INDEX + 1
95    L = M (INDEX)
           .
           .
           .
```

(a)



(b) (c)

Figure 2.4

(a) Segment of a FORTRAN program; (b) Segment of a flow graph derived from the program segment, with nodes representing individual statements; (c) Segment of a flow graph, derived from the graph in (b), in which nodes represent blocks.

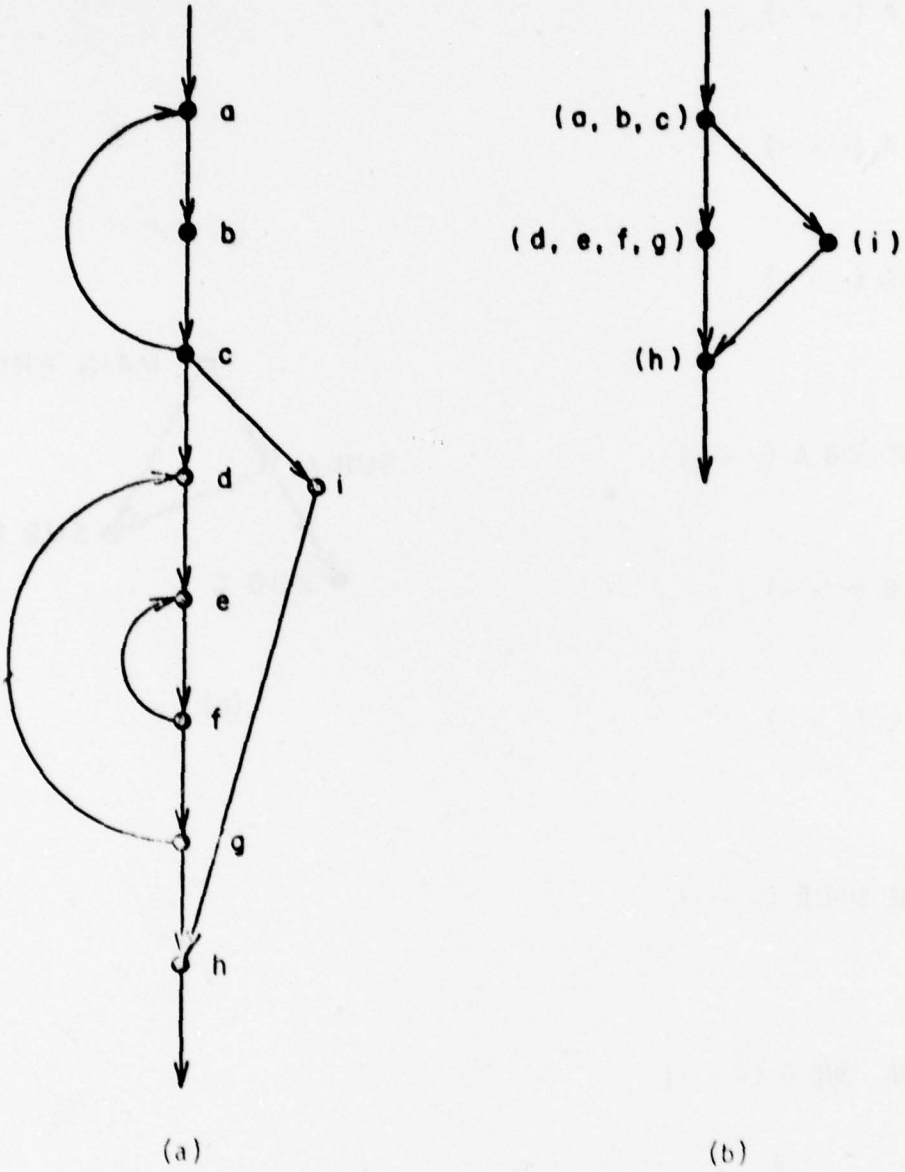
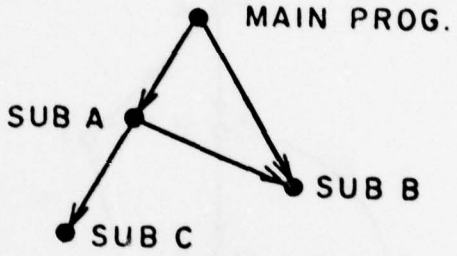


Figure 2.5

(a) Segment of a flow graph with cycles; (b) Segment of a flow graph, derived from (a), in which nodes represent strongly connected components.

```
(MAIN PROGRAM)
   .
   .
   .
 CALL SUB A (- - -)
   .
   .
   .
 CALL SUB B (- - -)
   .
   .
   .
 CALL SUB A (- - -)
   .
   .
   .
END
SUBROUTINE SUB A (- - -)
   .
   .
   .
CALL SUB B (- - -)
   .
   .
   .
CALL SUB C (- - -)
   .
   .
   .
END
SUBROUTINE SUB B (- - -)
   .
   .
   .
END
SUBROUTINE  SUB C (- - -)
   .
   .
   .
END
```

(a)



(b)

Figure 2.6

(a) FORTRAN program with subroutine calls; (b) Call graph for program in (a).

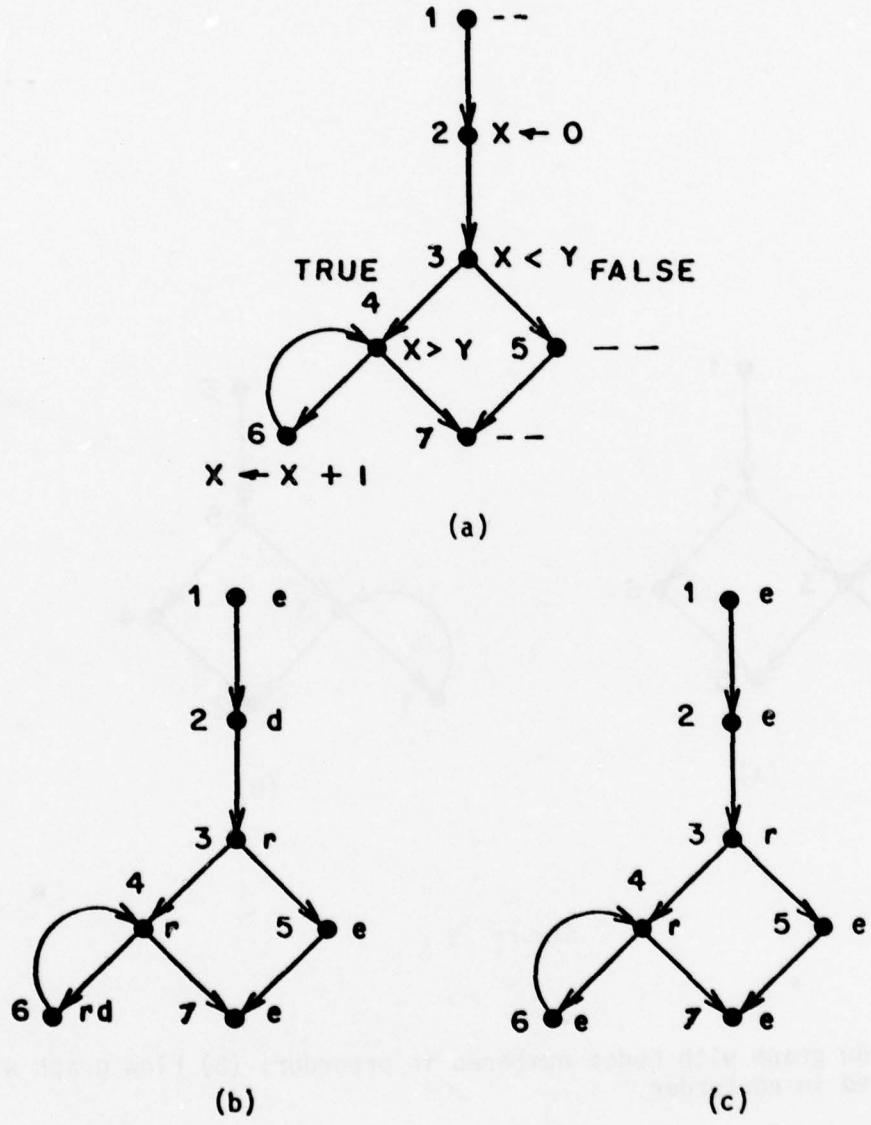


Figure 2.7

(a) Flow graph with nodes numbered for identification and relevant statements corresponding to nodes indicated; (b) Flow graph, derived from (a), with data actions on X indicated; (c) Flow graph derived from (a) with data actions on Y indicated.

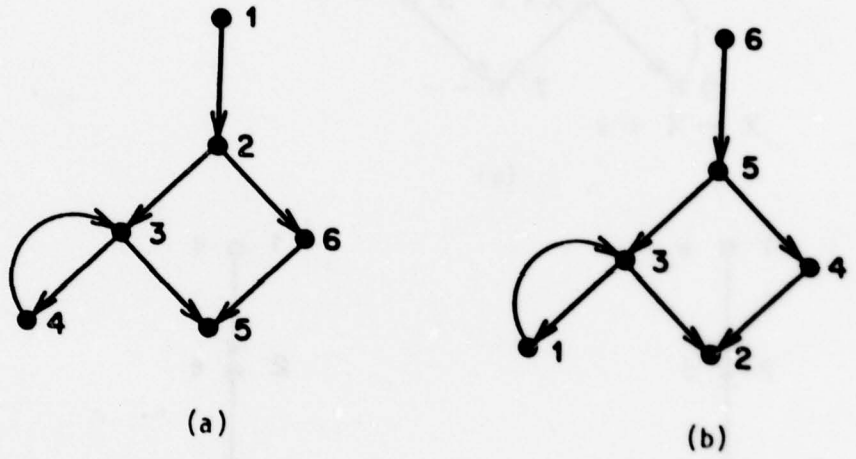


Figure 3.1

(a) Flow graph with nodes numbered in preorder; (b) Flow graph with nodes numbered in postorder.

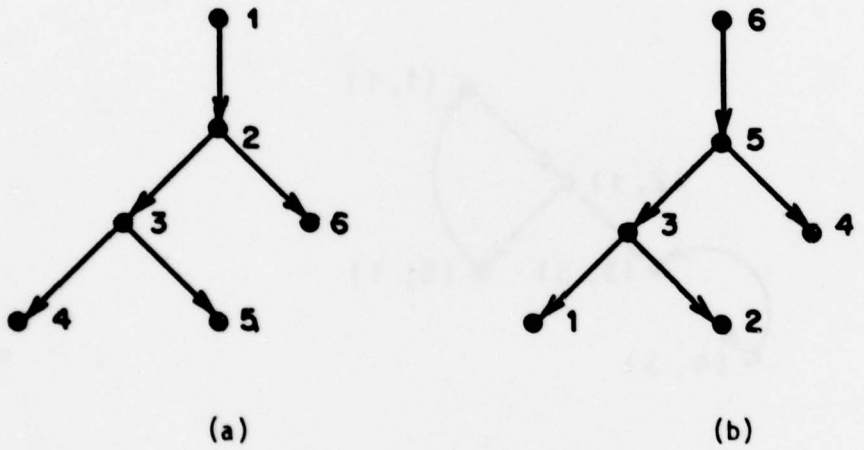


Figure 3.2

(a) Tree with nodes numbered in preorder, each node has a higher number than its parent; (b) Tree with nodes numbered in postorder, each node has a lower number than its parent.

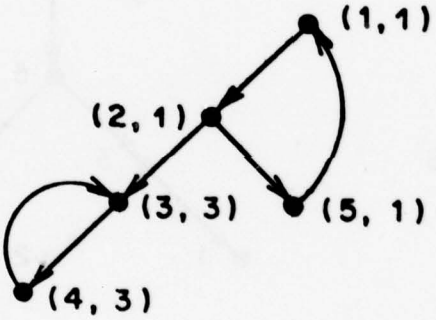


Figure 3.3

Flow graph with nodes labeled (i,j) where i is the preorder number and j is the lowlink number. All nodes with the same lowlink number are in the same strongly connected component and nodes with different lowlink numbers are in different strongly connected components.

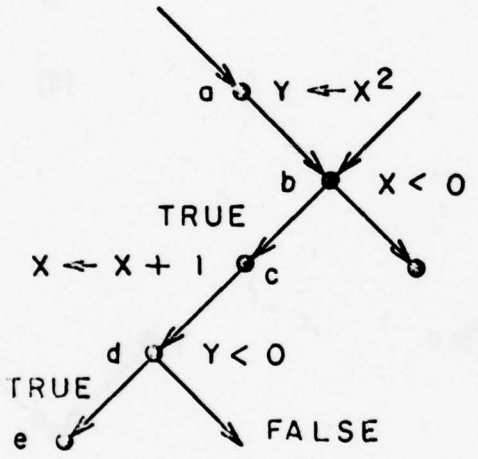


Figure 3.4

Flow graph with nodes labeled, and associated program statements indicated. The path a → b → c → d → e is unexecutable since the computation at a implies $y \geq 0$ and traversing the edge (d,e) implies $y < 0$.

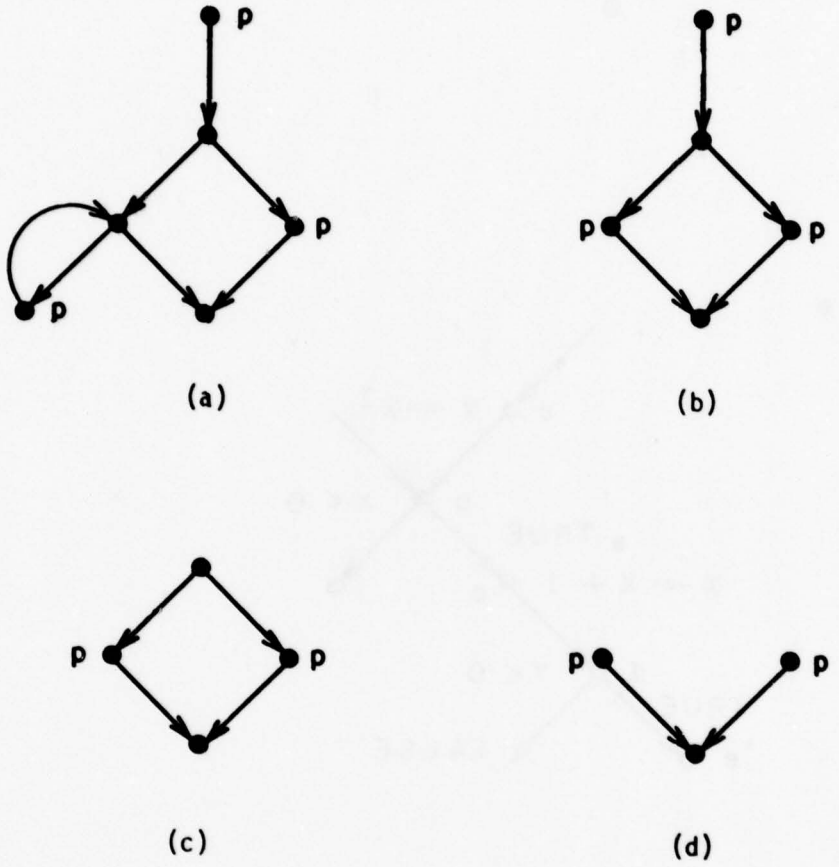
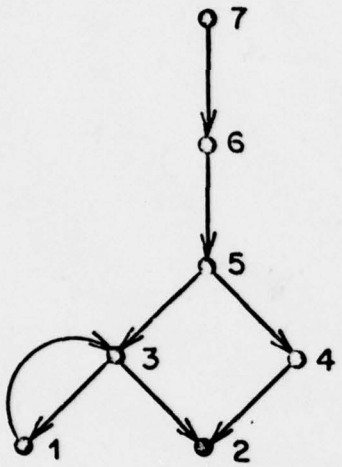


Figure 3.5

Illustration of steps in the Maheshwari, Gabow, Osterweil algorithm to determine existence of a path through a set of nodes. Nodes to be on the path are marked by p in (a). Nodes in the same SCC are collapsed into a single node resulting in (b). The frontier node is removed resulting in (c). The frontier node in (c) is removed resulting in (d). Now there are two nodes on the frontier both of which are to be on the path and the algorithm stops. Since the graph is not empty at this point the conclusion is there is no path in the graph (a) which includes all nodes marked p.



n	ref(n)	def(n)	live(n) (initial)	live(n) (final)
1	x	∅	∅	x,y
2	y	∅	∅	∅
3	x,y	∅	∅	x,y
4	z	∅	∅	y
5	x	∅	∅	x,y,z
6	∅	y	∅	x,y,z
7	∅	∅	∅	x,z

(a)



n	ref(n)	def(n)	live(n) (initial)	live(n) (after 1 iter.)	live(n) (final)
1	x	∅	∅	z	x,y,z
2	z	∅	∅	x,y,z	x,y,z
3	y	∅	∅	x,y,z	x,y,z

(b)

Figure 3.6

(a) Illustration of live sets, for given sets ref(n) and def(n) on a graph. Nodes are numbered in postorder and just one application of the live formula to the nodes in postorder produces the final live sets. (b) Another illustration of live sets. In this case two applications of the live formula to the nodes in postorder is required to obtain the final live sets.

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

SOLVING DIFFERENTIAL EQUATIONS ON A HAND HELD PROGRAMMABLE CALCULATOR

J. Barkley Rosser

Technical Summary Report

ABSTRACT

Most scientists who occasionally have to solve numerically a differential equation now own a hand held programmable calculator which will very often be adequate. Since hand held calculators are slow, there is particular need to keep the number of function evaluations to a minimum. At first thought, this would seem to rule out use of Runge-Kutta methods, but recent developments, such as those by Fehlberg (mostly unknown except to specialists), may make them competitive after all. In the area of predictor-corrector methods, some variations make excessive use of memory locations for a hand held calculator. An analysis of such matters is made in order to advise as to good procedures to follow, including alerting the solver to methods that are seldom taught in numerical analysis courses (where the emphasis is on the use of large fast computers).

AMS(MOS) Subject Classification: 34-02, 34A50, 65L05

Key Words: Runge-Kutta
predictor-corrector
numerical stability

Work Unit No. 7 - Numerical Analysis

PRECEDING PAGE BLANK

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

Significance and Explanation

Most scientists who occasionally have to solve numerically a differential equation now own a hand held programmable calculator which will very often be adequate for the purpose. Since hand held calculators are slow, there is particular need to keep the number of calculations of the function appearing in the differential equation to a minimum. At first thought, this would seem to rule out use of what are called Runge-Kutta methods, but some recent developments may make them competitive after all for certain types of problems. These types are identified, and a discussion, with numerical examples, is given how best to use the Runge-Kutta methods.

Some other methods which call for much fewer calculations of the function require more memory locations than are available on many hand held calculators. There still remain some methods which are modest both in the number of calculations and in memory requirements. How best to use these on a hand held calculator is explained, with numerical examples.

SOLVING DIFFERENTIAL EQUATIONS ON A HAND HELD PROGRAMMABLE CALCULATOR

J. Barkley Rosser

Dedicated to Prof. Dr. Johannes Weissinger on his 65th birthday.

1. Preliminaries. The present discussion is limited to initial value ordinary differential equations. One wishes to solve a system of equations

$$
\begin{aligned}
y_1' &= f_1(x, y_1, y_2, \ldots, y_p) \\
y_2' &= f_2(x, y_1, y_2, \ldots, y_p) \\
&\cdots\cdots\cdots\cdots \\
y_p' &= f_p(x, y_1, y_2, \ldots, y_p),
\end{aligned} \tag{1.1}
$$

being given the values of $y_1, y_2, \ldots, y_p$ at $x = x_0$; here y_i' denotes dy_i/dx. We will discuss only the special case

$$y' = f(x,y) , \tag{1.2}$$

being given the value of y at $x = x_0$; here y' denotes dy/dx. The discussion of (1.2) can easily be generalized to a system of equations. See Conte and de Boor, 1972, pp. 365-366. Incidentally, sets of higher order equations can be reduced to a system of first order equations, such as (1.1). See Conte and de Boor, 1972, p. 365.

The person who only occasionally has to solve numerically a differential equation may never have had a course in numerical analysis, or may have forgotten much of it. So it seems necessary to make the present paper reasonably self-contained, presupposing little previous experience. Even if the reader has previous experience, it was presumably on a large fast computer. For the hand held calculator, considerations are sufficiently different that one cannot rely too much on such previous experience. The discussion to follow is comprehensive enough to cover the important differences.

Also, there is disagreement between reputable texts on various points. An adjudication between them, with explanations, is included, in spite of the fact that this adds to the bulk of the present paper.

The overall procedure for solution is the classic one, namely to try step by step to calculate approximations $y_1, y_2, y_3, \ldots$ for y corresponding to the values $x_1, x_2, x_3, \ldots$, where $x_0 < x_1 < x_2 < x_3 < \ldots$. We take y_0 as the value given for y at $x = x_0$.

We denote $x_{n+1} - x_n$ generically by h. Usually h will be taken the same for a considerable succession of steps. A change in h is occasionally called for, but not uncommonly this involves some travail, and in the main one tries to avoid it. On the other hand, all authorities agree that at each step (or at least frequently), tests should be made to see if a change of step length is needed, shorter to keep errors under control, or longer to avoid unduly extended calculation that would result from taking h smaller than necessary.

2. The Euler method. If one has proceeded to a good approximation y_n for the value of y at $x = x_n$, a Taylor's expansion will give

$$y_{n+1} = y_n + h\, y_n' + \frac{h^2}{2} y''(\xi), \tag{2.1}$$

where $x_n < \xi < x_{n+1}$. We evaluate y_n' by (1.2), and get

$$y_{n+1} = y_n + h\, f(x_n, y_n) + \frac{h^2}{2} y''(\xi) . \tag{2.2}$$

Unless one already knows the solution of (1.2), one has no way to calculate the final term on the right of (2.2). Certainly, if h is taken to be small enough, the final term will be quite small, and the approximation

$$y_{n+1} \cong y_n + h\, f(x_n, y_n) \tag{2.3}$$

will give a sufficiently accurate value for y_{n+1}, after which one gets y_{n+2} by a similar formula, then y_{n+3}, etc. A problem with which we must cope is being sure that we have taken h small enough that repeated use of (2.3) instead of (2.2) does not lead to serious error in the end.

Suppose we wish to solve

$$y' = -y, \tag{2.4}$$

given that $y = 1$ when $x = 0$. Let us try to approximate the value of y when $x = 6$. As the answer for (2.4) is obviously $y = e^{-x}$, we can say that for $0 < \xi \leq 6$ we have $|y''(\xi)| < 1$. Hence the error in (2.3) will be less than $h^2/2$. If we take steps of constant length h, we will require $6/h$ steps to get from $x = 0$ to $x = 6$. With an error less than $h^2/2$ at each step, and $6/h$ steps, the final error will add up to less than

$$\left(\frac{h^2}{2}\right)\left(\frac{6}{h}\right) = 3h .$$

From this, it is tempting to conclude that the method is first order; that is, the overall error is roughly proportional to h. For example, if we decrease h by a factor of 2, we would expect to decrease the error at $x = 6$ by approximately a factor of 2.

Within bounds, the conclusion is correct. However, the argument given above to support it is fallacious. To see this, let us look at a specific example. Take $h = 0.1$. By (2.3), we will get

$$y_1 = 0.9.$$

This is too small by about

$$0.0048374.$$

(In accordance with (2.2), this error is close to $h^2/2$.) However, the value of y at $x = 6$ is

(2.5) $$e^{-6} \cong 0.0024888.$$

Thus our final answer is less than the error on the first step. To suggest that we can approximate the overall error at $x = 6$ by adding up such errors as shown above for each of the 60 steps required to get from 0 to 6 is simply not sound.

In fact, for the equation (2.4), use of (2.3) with $h = 0.1$ gives

$$y_{n+1} = (0.9)y_n .$$

Using this 60 times would give an estimate for y at $x = 6$ of

$$(0.9)^{60} \cong 0.0017970.$$

This is in error by less than 28%. For a procedure in which the error on the first step was almost twice the total final answer, this is not bad.

By (2.3), we get

$$y_{n+1} \cong (1-h)y_n \tag{2.6}$$

for the equation (2.4). This gives

$$\begin{aligned} y_{n+1} &\cong \{e^{-h} - \frac{h^2}{2!} + \frac{h^3}{3!} - \ldots\} y_n \\ &= \{e^{-h} - \frac{h^2}{2}(1 - \frac{h}{3} + \frac{h^2}{12} - \ldots)\} y_n \\ &= e^{-h}\{1 - \frac{h^2}{2} e^{h}(1 - \frac{h}{3} + \frac{h^2}{12} - \ldots)\} y_n . \end{aligned}$$

The factor

$$e^{h}(1 - \frac{h}{3} + \frac{h^2}{12} - \ldots) \tag{2.7}$$

tends to remain constant and close to unity for $|h|$ small, since as h increases the first factor increases and the second decreases. So we replace (2.7) by unity, getting

$$y_{n+1} \cong e^{-h}\{1 - \frac{h^2}{2}\} y_n . \tag{2.8}$$

So at $x = 6$, we get

$$\begin{aligned} y &= \left[e^{-h}\left\{1 - \frac{h^2}{2}\right\}\right]^{6/h} \\ &= e^{-6}\left[\left\{1 - \frac{h^2}{2}\right\}^{-\frac{2}{h^2}}\right]^{-3h} \\ &\cong e^{-6} e^{-3h} . \end{aligned}$$

For $h = 0.1$, we get

$$y = e^{-6}\{0.74082\}.$$

Hence, by the above analysis, we expect the calculated value to be too low by about 26%: it was actually too low by about 28%.

The final formula,

(2.9) $$y \doteq e^{-6}\, e^{-3h}$$

shows that (for a reasonable range of h) the relative error is indeed of order h. Actually for $h = 0.1$ and the equation (2.4), we see by (2.2) that we have about 0.5% relative error at each step. Accumulating such relative errors for 60 steps could give an overall error of 30%, which is about what we got.

So sometimes it is absolute error that one should accumulate from step to step, but other times it can be relative error. For programmers who try to write general purpose differential equation solvers for large computers with error control built in, the question of how to handle error accumulation poses formidable difficulties. See Hull, et al., 1972, pp. 607-608. When one is solving on a hand held calculator, one sees the progress of the solution, step by step. In regions where it is best to reckon by accumulating relative errors, one can do so. However, when it would be better to accumulate absolute errors (for example, if one is going to pass through a zero value of y), the change to accumulating absolute errors is easily made. Such flexibility is hard to arrange in a preset program for a large computer.

By (2.9), if we should wish to get to $x = 6$ with 0.1% accuracy, we should have to take h about 1/3000. Thus we would require 18,000 steps. This would not be too bad on a large fast computer, but on a hand held calculator it could require hours, especially if our equation to be solved were more complicated than (2.4). So we need something better than the Euler method.

Actually, there were two reasons for including the present section on the Euler method. One is to demonstrate the importance of maintaining flexibility about whether one is accumulating relative errors or absolute errors. With a hand held calculator, such flexibility is easily maintained. With a preset program for a large fast computer, such flexibility is almost impossible.

Thus, in Hull, et al., 1972, the decision was made to lock one's self into accumulating absolute errors (see their p. 607). To get to $x = 6$ with a 0.1% error by this scheme would require far more than 18,000 steps. On the other hand, the large computers are so very fast that it is still a practical scheme.

The other reason for mentioning the Euler method is that it is useful for "looking ahead." Using a few steps with large h takes very little time and gives at least a general idea of what one might expect to encounter for some distance ahead. If one takes such a "look ahead" every so often it can help in planning how best to arrange the upcoming part of the integration.

3. Runge-Kutta methods. The text Henrici, 1977, explains how to do many calculations on the HP-25 programmable calculator. On p. 182, he suggests that for solving the differential equation (1.2) one might use the second order Runge-Kutta method

$$y_{n+1} \doteq y_n + \frac{h}{2}\{f(x_n,y_n) + f(x_{n+1},y_n + h\, f(x_n,y_n))\}. \tag{3.1}$$

Applied to

$$y' = k\, y\ , \tag{3.2}$$

this gives

$$y_{n+1} \doteq \left(1 + hk + \frac{(hk)^2}{2}\right) y_n\ .$$

An analysis like that in the previous section shows that if we set $y = 1$ at $x = 0$, then at $x = X$ we would get approximately

$$y \doteq e^{kX}\ e^{-h^2k^3X/6}\ . \tag{3.3}$$

The first factor on the right is what y should be, and the second factor shows about how far off we are from the true value. So, if we take $k = -1$ (as in (2.4)) and $X = 6$, and ask for 0.1% accuracy, we need to take h about 1/32. Thus it will take 192 steps to get to $x = 6$. However (note (3.1)), each step requires TWO evaluations of the function $f(x,y)$. So we require 384 function evaluations.

Actually, we require more than that. As stated above, one should make frequent checks to see if h is about the right size; this is a point that we failed to come to grips with in the previous section. Analogous to (2.2), there is a formula (see pp. 192-200 of Ralston, 1965) that gives the error of (3.1) as

$$h^3K + h^4L + \ldots ,$$

where K, L, etc., are complicated functions of derivatives of y and partial derivatives of $f(x,y)$. For small h, we may say that the error of (3.1) is about h^3K. If we take two successive steps, from y_n to y_{n+2}, and if K does not change much from one step to the next, then we could accumulate a local error of $2h^3K$ (over and above whatever error we had at y_n) in getting to y_{n+2}. Now apply (3.1) with $2h$ in place of h to get from y_n to y_{n+2} in a single step. Assuming that K is not flucuating badly, we would make an error of about $(2h)^3K$ in getting to y_{n+2}, or four times the error we made in getting to y_{n+2} in two steps. So now we have two approximations for y_{n+2}, one with an error about four times the other. From this, we can estimate about how far off our better approximation is. If it is close enough, we take it as the value of y_{n+2}. If it is not close enough, we have to go back to y_n and try over again with a smaller value of h. See Hull, et al., 1972, bottom of p. 616. How close is "close enough" is a sticky question for which there seems not to be a very good answer. After all, this involves only two steps, and one has to worry about the accumulation of errors over many steps. And, as noted in the previous section, there is the question if one should be worried about accumulation of absolute errors or of relative errors. But at least one has to have some sort of estimate of the step by step errors.

As pointed out above, if we take two steps of length h each to get from y_n to y_{n+2}, we probably have an estimate that is in error by about one fourth of what we would get if we went from y_n to y_{n+2} in one step of length $2h$. If the error were EXACTLY one fourth, we could write a formula for the true

value of y. It is tempting to take this formula as the value for y_{n+2}. It most likely gives a better estimate for y_{n+2}. However, there is no way to say how much better; an estimate for the step by step error would then not be available. Also, one is left with no improved value for y_{n+1}. So we had best consider that this formula merely provides an estimate of the error after each pair of steps.

For the pair of steps of length h, we required two function evaluations per step. For the step of length $2h$, we also require two evaluations, except that the evaluation of $f(x,y)$ at $x = x_n$ has already been made. So, for the evaluation of y_{n+1} and y_{n+2}, with an estimate of error at x_{n+2}, we require altogether five function evaluations. So for the 192 steps to get from $x = 0$ to $x = 6$, that is, 96 pairs of steps, we require 480 function evaluations. While this is a great improvement over the 18,000 function evaluations required by the Euler method, it could be rather time consuming if $f(x,y)$ is at all complex.

So we wish for something better. We cannot manage anything better on the HP-25. It has only 50 program steps, and implementation of (3.1) uses 39 of these, leaving only 11 program steps for the calculation of $f(x,y)$. In many cases, 11 program steps will not be adequate. So, if we are to do much with differential equations, we need a more capable calculator than the HP-25.

4. Alternatives to Runge-Kutta. Let us suppose that we have a calculator at least as capable as the HP-65. Most programmable calculators now on the market are appreciably more capable than the HP-65, but it sufficed for the calculations recorded in this paper. With it, one can carry out higher order Runge-Kutta methods than (3.1). Suppose we use the classical fourth order Runge-Kutta. See (5.6-48) on p. 200 of Ralston, 1965, or (6.37) on p. 338 of Conte and de Boor, 1972. Incidentally, if we have two equations

(4.1) $$y' = f(x,y,z)$$

(4.2) $$z' = g(x,y,z),$$

then the classical fourth order Runge-Kutta method consists of

(4.3) $$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

(4.4) $$z_{n+1} = z_n + \frac{1}{6}(\ell_1 + 2\ell_2 + 2\ell_3 + \ell_4)$$

where

$$k_1 = h\, f(x_n, y_n, z_n)$$

$$\ell_1 = h\, g(x_n, y_n, z_n)$$

$$k_2 = h\, f\left(x_n + \frac{h}{2},\ y_n + \frac{k_1}{2},\ z_n + \frac{\ell_1}{2}\right)$$

$$\ell_2 = h\, g\left(x_n + \frac{h}{2},\ y_n + \frac{k_1}{2},\ z_n + \frac{\ell_1}{2}\right)$$

$$k_3 = h\, f\left(x_n + \frac{h}{2},\ y_n + \frac{k_2}{2},\ z_n + \frac{\ell_2}{2}\right)$$

$$\ell_3 = h\, g\left(x_n + \frac{h}{2},\ y_n + \frac{k_2}{2},\ z_n + \frac{\ell_2}{2}\right)$$

$$k_4 = h\, f(x_n + h,\ y_n + k_3,\ z_n + \ell_3)$$

$$\ell_4 = h\, g(x_n + h,\ y_n + k_3,\ z_n + \ell_3).$$

See Conte and de Boor, 1972, pp. 365-366. Extension to a system of equations such as (1.1) is obvious. Incidentally, the error in (4.3) is approximately h^5K for a suitable K, and the error in (4.4) correspondingly.

If we wish to solve (2.4) out to $x = 6$ with 0.1% accuracy, we must take h slightly less than 3/8. Taking $h = 3/8$ is probably close enough, which would require 16 steps. There are four function evaluations per step. To accomplish the procedure suggested above for estimating the step by step error of y_{n+2} after a pair of steps, we must also make a single step of length $2h$ from y_n to y_{n+2}. This also takes four evaluations, but one has been done before. So we require 11 function evaluations for each pair of steps. So we need 88 function evaluations.

Can we do better? In Enright and Hull, 1976, testing is reported of four sorts of methods:

1. Runge-Kutta-Fehlberg methods.
2. Rational extrapolation methods.
3. Adams type predictor-corrector methods.
4. Milne type predictor-corrector methods.

The Runge-Kutta-Fehlberg methods are Runge-Kutta type methods improved according to ideas in Fehlberg, 1968, 1969, and 1970, to give easier ways of estimating step by step errors. This is purported to reduce somewhat the number of function evaluations. In view of certain advantages of Runge-Kutta methods, the Fehlberg improvements make the Runge-Kutta methods fairly competitive when the function evaluations can be done quickly. However on p. 949 of Enright and Hull, 1976, some disadvantages with the Fehlberg methods are reported. By phone (March, 1978) T. E. Hull of Toronto informed the present writer that new improvements had eliminated certain of these disadvantages, and that IMSL has recently embodied these new improvements in its Runge-Kutta software package. However, at best these methods are competitive only when the function evaluations can be done quite quickly.

The rational extrapolation methods derive from ideas of Gragg, 1965, and were developed carefully in Bulirsch and Stoer, 1966. In Enright and Hull, 1976, these methods are given a fairly good rating when the function evaluations can be done quickly. They seem complex to program for a hand held calculator, and so we will say no more about them.

Enright and Hull, 1976, give their highest ratings to certain Adams type predictor-corrector methods. However, their conclusions do not necessarily hold for hand held calculators because of the very limited number of memory locations of hand held calculators. So we will make a study of Adams type and Milne type predictor-corrector methods for hand held calculators.

Incidentally, one cannot merely take one of the programs tested in Enright and Hull, 1976, and put it on his hand held calculator. In order to accomodate the wide diversity of vagaries that can arise in solutions of differential equations, these programs have a large amount of "overhead", and are beyond the capabilities of present hand held calculators, besides being so long as to be very time consuming at the slow speed of hand held calculators. However, unless one has a very large system of equations to solve, one can proceed step by step on a hand held calculator. One "looks ahead" periodically as an aid to planning the calculation, one monitors the accumulation of errors as one goes, and one is alert for idiosyncracies that might arise. If the latter do arise, one can try shifting methods; if worse comes to worst, one can try a change of variables, or more complex stratagems.

The classical predictor-corrector method is that of Milne (see (28.1) and (28.2) on p. 65 of Milne, 1953, or (5.5-12) on p. 182 of Ralston, 1965):

$$y^{p}_{n+1} = y_{n-3} + \frac{4h}{3}(2y'_n - y'_{n-1} + 2y'_{n-2}) \tag{4.5}$$

$$y_{n+1} = y_{n-1} + \frac{h}{3}(f(x_{n+1}, y^{(p)}_{n+1}) + 4y'_n + y'_{n-1}). \tag{4.6}$$

This uses two function evaluations per step. There is the obvious one in (4.6), and when one comes to use (4.5) for the next step, one needs y'_{n+1}, which calls for a second function evaluation. Incidentally, it is required that

$$x_{n+1-i} - x_{n-i} = h \quad \text{for} \quad i = 0,1,2,3.$$

To use this method, we must have y_{n-3}, y_{n-1}, y'_{n-2}, y'_{n-1}, and y'_n stored, and for the next step we will have to have y_{n-2} and y_n. So our storage requirements amount to four values of y and three of y'. In addition, one has to carry the current value of x_n. This uses up eight memory locations, which for all practical purposes are all there are on the HP-65. This leaves no memory locations to use in evaluating $f(x,y)$. So use of the classical Milne predictor-corrector would often not be possible on the HP-65. For

calculators with more memory locations, the large requirement for memory locations could preclude use of the Milne predictor-corrector if one wishes to solve a system of even as few as three equations. A more compelling reason not to use the Milne method is that it is now known to be unstable in certain circumstances which arise not too infrequently. We shall produce predictor-corrector methods that have much more modest memory requirements and are stable besides. Meanwhile the Milne predictor-corrector will be used to illustrate typical features of predictor-corrector methods.

As the name would indicate, a predictor-corrector method embodies a predictor and a corrector. The predictor is (4.5) and the corrector is

$$y_{n+1}^{(c)} = y_{n-1} + \frac{h}{3}\,(y'_{n+1} + 4y'_n + y_{n-1}), \tag{4.7}$$

which is nothing more than Simpson's rule for integrating y' approximately. The obvious disadvantage of (4.7) is that one needs the value of y'_{n+1} to get the value of $y_{n+1}^{(c)}$, whereas by (1.2) one needs the value of $y_{n+1}^{(c)}$ to get the corresponding value of y'_{n+1}. Actually, if h is reasonably small and $f(x,y)$ is reasonably well behaved, one can get out of this impasse by an iteration scheme; after making a guess for y'_{n+1}, successively substitute the current y'_{n+1} into (4.7) to get a $y_{n+1}^{(c)}$ and substitute $y_{n+1}^{(c)}$ into (1.2) to get a better value for y'_{n+1}. In the usual case that will arise, this will converge to a y'_{n+1} and $y_{n+1}^{(c)}$ that satisfy both (4.7) and (1.2). Unfortunately, it is prodigal with function evaluations.

So we adopt a compromise. A predictor is given, in this case (4.5), which produces a reasonably close approximation for y_{n+1}. This is substituted into (1.2) to get a guess for y'_{n+1}. This is then substituted into the corrector; the net result is embodied in (4.6). There the process is stopped. We do not have as much accuracy as we could get with a few more iterations, but we have held the total number of function evaluations to two per step.

Since we need four values of y to use the predictor (and at equally spaced values of x), these four values must somehow be obtained before we can start to use this process. On pp. 61-64 of Milne, 1953, is given a scheme to get four starting values. It is now generally agreed that the best way to get started is to calculate y_1, y_2, and y_3 by Runge-Kutta.

One attractive feature of predictor-corrector methods is the ease with which one gets an estimate of the step by step error. The error for (4.5) is

$$\frac{14}{45} h^5 y^{(v)}(\xi)$$

and that for (4.7) is

$$-\frac{1}{90} h^5 y^{(v)}(\xi),$$

provided that one has values of y'_{n+1} and $y^{(c)}_{n+1}$ that satisfy both of (4.7) and (1.2). Then the error in (4.7) will be about

$$-\frac{1}{29} \{y^{(c)}_{n+1} - y^{(p)}_{n+1}\} . \tag{4.8}$$

Actually, the value of y_{n+1} given by (4.6) is close enough to what one would get by iterating with (4.7) that the error in y_{n+1} from (4.6) is approximately what one would get by using y_{n+1} from (4.6) in place of $y^{(c)}_{n+1}$ in (4.8).

Since one needs four consecutive values of y at equal step sizes for the Milne method, a change of step size (should it be required) is not easy. If one has as many as seven preceding consecutive steps of equal length, one can double the step size by picking every other value from the present and preceding y's. However this would require recovering the values of y_{n-6}, y_{n-4}, and y'_{n-4} besides the values that one usually stores. Or one could carry on for three more steps before doubling, being careful to save the key values.

If one wishes to halve the step size, one can use

$$y_{n-\frac{1}{2}} = \frac{1}{128}\{45y_n + 72y_{n-1} + 11y_{n-2} + h(-9y'_n + 36y'_{n-1} + 3y'_{n-2})\} \tag{4.9}$$

$$y_{n-\frac{3}{2}} = \frac{1}{128}\{11y_n + 72y_{n-1} + 45y_{n-2} - h(3y'_n + 36y'_{n-1} - 9y'_{n-2})\} \tag{4.10}$$

(see p. 208 of Hamming, 1962, or (A57) and (A58) on p. 451 of Rosser, 1967).

Unfortunately, doubling or halving the step size is often not the most efficient change to make. For a change of a different size, one can use interpolation formulas, of which (4.9) and (4.10) are samples, but it might be simpler just to make a fresh start, generating the next three values of y by Runge-Kutta.

5. Adams predictor-correctors. For the Adams method of order r, the predictor (which is commonly called an Adams-Bashforth formula) is

$$y_{n+1}^{(p)} = y_n + h \sum_{i=0}^{r-1} \alpha_i \, y'_{n-i} + K_p \, h^{r+1} \, y^{(r+1)}(\xi), \tag{5.1}$$

and the corrector (which is commonly called an Adams-Moulton formula) is

$$y_{n+1}^{(c)} = y_n + h \sum_{i=0}^{r-1} \beta_i \, y'_{n+1-i} - K_c \, h^{r+1} \, y^{(r+1)}(\xi). \tag{5.2}$$

The h that appears is

$$h = x_{n+1-i} - x_{n-i},$$

which is required to be the same for $i = 0, \ldots, r-1$.

The error term on the right of (5.2) is based on the assumption that

$$y'_{n+1} = f(x_{n+1}, y_{n+1}^{(c)}). \tag{5.3}$$

As indicated with (4.7), values of $y_{n+1}^{(c)}$ and y'_{n+1} that satisfy both (5.2) and (5.3) can usually be found by an iterative process. That is, if one chooses a $\bar{y}_{n+1}$ that is near the limiting $y_{n+1}^{(c)}$, and then forms

$$\bar{y}'_{n+1} = f(x_{n+1}, \bar{y}_{n+1}),$$

and substitutes this $\bar{y}'_{n+1}$ for y'_{n+1} on the right of (5.2), one will get a value nearer to $y_{n+1}^{(c)}$ than $\bar{y}_{n+1}$. Repetition of this converges to $y_{n+1}^{(c)}$. However, this is costly in function evaluations, and what is mostly done is to define

$$y_{n+1}^{(t)} = y_n + h \, \beta_0 \, f(x_{n+1}, y_{n+1}^{(p)}) + h \sum_{i=1}^{r-1} \beta_i \, y'_{n+1-i}, \tag{5.4}$$

after which one takes $y_{n+1} = y_{n+1}^{(t)}$. This holds the number of function

evaluations to two per step. (One may think of the superscript "t" as standing for "traditional.")

The requirements for memory locations are one for y_n, r for y'_{n-i} ($i = 0,\ldots,r-1$), and one for x_n. Actually, on a calculator like the HP-65, one can reduce the memory requirements to one fewer by judicious use of the stack. Suppose we are at x_n, and have y_n, but have not yet calculated y'_n. Let us have y'_{n-i} stored at memory location i, for $i = 1, \ldots, r-1$, with y_n at location r, and x_n at location $r+1$. We calculate

$$\sum_{i=1}^{r-1} \alpha_i \, y'_{n-i} ,$$

and store it in location 1, having for $i = r-2, r-3, \ldots, 1$ successively stored y'_{n-i} in location $i+1$. Now calculate

$$y'_n = f(x_n, y_n) .$$

Push an extra copy of this up in the stack to hold momentarily. Now add $\alpha_0 \, y'_n$ to what was stored in location 1, put the extra copy of y'_n into location 1, and proceed with the calculation of $y^{(p)}_{n+1}$ by (5.1). We now have in storage or in the stack everything needed for calculation of $y^{(t)}_{n+1}$ by (5.4).

In the above, it is assumed that the α's and β's are part of the program. The α's and β's are fairly simple numbers, so that this does not seem to overload the program, unless one has a very large value of r, in which case one hopes that additional memory locations will be available to store the α's and β's.

The Adams formulas of any order, with an error term, can be derived by use of the Newton backward difference formula. Details are given on pp. 340-342 and 350-351 of Conte and de Boor, 1972, where the Adams method of order 4 is derived. A rather diffuse explanation is scattered through a number of pages of Milne, 1953, but on p. 50 is a table from which the coefficients can be calculated up to order 9. Henrici, 1962, gives coefficients of the predictors on p. 194 and

of the correctors on p. 199, both up to order 6. Neither the tables of Milne nor of Henrici give the error terms, but these can be derived easily by taking $y = x^{r+1}$ in (5.1) and (5.2), which will give the values of K_p and K_c. Correctors up to order 9, with error terms, can be got by a trivial modification of formulas (A2), (A4), (A7), (A12), (A20), (A26), (A34), and (A42) on pp. 446-449 of Rosser, 1967.

In order to get started, we need the r values $y_0, y_1, \ldots, y_{r-1}$.

For the Adams method of order two, we have the predictor and corrector

$$(5.5) \qquad y_{n+1}^{(p)} = y_n + \frac{h}{2}(3y'_n - y'_{n-1}) + \frac{5h^3}{12} y'''(\xi)$$

$$(5.6) \qquad y_{n+1}^{(c)} = y_n + \frac{h}{2}(y'_{n+1} + y'_n) - \frac{h^3}{12} y'''(\xi) .$$

We recognize (5.6) as the trapezoid rule. As noted for the general case, we define

$$(5.7) \qquad y_{n+1}^{(t)} = y_n + \frac{h}{2}(f(x_{n+1}, y_{n+1}^{(p)}) + y'_n)$$

and take $y_{n+1} = y_{n+1}^{(t)}$.

As y_0 is given, we need to obtain a value only for y_1 to get started. If one wishes to avoid use of Runge-Kutta, the value of y_1 can be obtained by iterating with (5.6). The same applies if one needs to make a restart after changing the length of the step.

While the values of ξ in the error terms of (5.5) and (5.6) will scarcely ever be the same, if y''' is slowly varying then the error of $y_{n+1}^{(c)}$ will be about one fifth that of $y_{n+1}^{(p)}$ if we have iterated on (5.6) until both (5.6) and (5.3) are satisfied. In practice, the value of $y_{n+1}^{(t)}$ is close enough to this limiting value of $y_{n+1}^{(c)}$ that one can say that the error of $y_{n+1}^{(t)}$ is about one fifth that of $y_{n+1}^{(p)}$. In other words the step by step error of the new y_{n+1} is about

$$(5.8) \qquad -\frac{1}{6}(y_{n+1}^{(t)} - y_{n+1}^{(p)}) .$$

If it were not so, then one is probably using too large a value of h.

One is tempted to try for more accuracy by defining

$$y_{n+1}^{(e)} = y_{n+1}^{(t)} - \frac{1}{6}\left(y_{n+1}^{(t)} - y_{n+1}^{(p)}\right), \tag{5.9}$$

and then taking y_{n+1} to be $y_{n+1}^{(e)}$. (We use the superscript "e" to denote "extrapolated".) This does indeed seem to give more accuracy. If one would use

$$y_{n+1}^{(c)} - \frac{1}{6}\left(y_{n+1}^{(c)} - y_{n+1}^{(p)}\right),$$

with the limiting value of $y_{n+1}^{(c)}$, one would have a third order method. See Ralston, 1965, p. 186. As $y_{n+1}^{(t)}$ is close to $y_{n+1}^{(c)}$, one has close to a third order method. So one should have more accuracy, but one has no way to tell how much more; there is no way to estimate the step by step error. Indeed, as we shall shortly show, numerical examples disclose a somewhat erratic behavior of $y_{n+1}^{(e)}$.

The doctrine on use of $y_{n+1}^{(e)}$ is not clear. In Ralston, 1965, on p. 186, such use is not favored. Indeed, it is there stated that it affects the stability properties, and this is also affirmed on p. 210 in Hamming, 1962. However, in Hamming, 1962, there is advocacy of more complicated schemes which make use of an equivalent of $y_{n+1}^{(e)}$. For the Adams methods, which are strongly stable, the present writer has found no evidence that use of $y_{n+1}^{(e)}$ causes stability to deteriorate, though it does indeed for some other types of predictor-corrector methods. However, the lack of any way to estimate the step by step error if one uses $y_{n+1}^{(e)}$ seems a strong point against it.

Since we have brought up the question of stability, we should note that usage of the term is not uniform. Many people, following Dahlquist, 1956, say that stability requires that all roots of a certain difference equation should be less than unity in absolute value. In Hamming, 1962, on p. 191, it is pointed out that according to that definition no method for solving $y' = ky$ could be

stable for $k > 0$. So Hamming introduces (on p. 191) the notion of relative stability, in which roots can be larger than unity in absolute value provided they are less in absolute value than a certain selected root. In Definition 5.2 on p. 176 of Ralston, 1965, we find Ralston taking this as the definition of stability; what Hamming calls "relative stability" is called "stability" by Ralston. We concur with Ralston in our use of the word "stable." If a method is stable in this sense, one can safely carry out a numerical integration of indefinite extent by means of it.

It turns out that to achieve stability for predictor-corrector methods, one must maintain certain bounds for

$$h \frac{\partial f(x,y)}{\partial y} \quad . \tag{5.10}$$

The same is claimed to be true for some Runge-Kutta methods. See Shampine and Watts, 1977, p. 270. Writers of texts on numerical analysis have been very lax about calling this to the attention of their readers, and most people probably harbor the illusion that Runge-Kutta methods are stable under all circumstances. Fortunately, the Runge-Kutta's of orders two, three, and four given in this paper happen to be relatively stable (in the sense of Hamming) in all circumstances, and hence give no trouble about stability. Far and away most people who use a Runge-Kutta use oneof these three, which is probably why almost no cases of instability have been reported when using Runge-Kutta.

A discussion of why one needs to set bounds for (5.10) to avoid instability is given in Hamming, 1962, on pp. 189-190 and in Ralston, 1965, on pp. 169-178.

Certainly, when one is using predictor-corrector methods, one should make routine estimates of $\partial f(x,y)/\partial y$ to check whether (5.10) is remaining in bounds. A numerical example of what can happen if one fails to do this will be given in Section 6. If one has two values y and $y + \epsilon$, and ϵ is fairly small, then we get a good estimate by

$$\frac{\partial f(x,y)}{\partial y} \cong \frac{f(x, y+\epsilon) - f(x, y)}{\epsilon} \quad . \tag{5.11}$$

For the calculations proposed, we can take $x = x_{n+1}$, $y + \varepsilon = y_{n+1}^{(t)}$, and $y = y_{n+1}^{(p)}$. Thus we can get our estimate by (5.11) without having to perform any additional function evaluations, since $f(x_{n+1}, y_{n+1}^{(t)})$ must be calculated to provide the value of y'_{n+1} required in the predictor for the next step.

The reason that $y_{n+1}^{(t)}$ is close to the $y_{n+1}^{(c)}$ that one would get by iterating the corrector, (5.6), is because $y_{n+1}^{(p)}$ is already fairly close to that $y_{n+1}^{(c)}$; the use of $y_{n+1}^{(p)}$ in (5.7) makes $y_{n+1}^{(t)}$ even closer to $y_{n+1}^{(c)}$ than $y_{n+1}^{(p)}$ If we could find a still closer approximation than $y_{n+1}^{(p)}$ to use in (5.7), we could do still better. A closer approximation is

$$y_{n+1}^{(p)} + (y_{n+1}^{(t)} - y_{n+1}^{(p)}), \tag{5.12}$$

since this is $y_{n+1}^{(t)}$. Obviously we cannot use this until we have calculated $y_{n+1}^{(t)}$. However, with h fairly small, $y_n^{(t)} - y_n^{(p)}$ does not vary hugely from step to step. So if we define

$$y_{n+1}^{(a)} = y_{n+1}^{(p)} + (y_n^{(t)} - y_n^{(p)}), \tag{5.13}$$

then $y_{n+1}^{(a)}$ should be closer to $y_{n+1}^{(c)}$ than $y_{n+1}^{(p)}$. (We may consider the "a" as standing for "adjusted.") This suggestion is made at the bottom of p. 201 of Hamming, 1962, but its use is there discouraged although later a slight variation of it is strongly advocated. (We shall discuss this Hamming variation later.)

If we use $y_{n+1}^{(a)}$ as an improved predictor, we would define

$$y_{n+1}^{(ta)} = y_n + \frac{h}{2}\,(f(x_{n+1}, y_{n+1}^{(a)}) + y'_n)\,, \tag{5.14}$$

and then use $y_{n+1}^{(ta)}$ for y_{n+1} instead of $y_{n+1}^{(t)}$.

Since $y_{n+1}^{(ta)}$ is closer to $y_{n+1}^{(c)}$ than $y_{n+1}^{(p)}$ or $y_{n+1}^{(t)}$, it would be better to define $y_{n+1}^{(a)}$ by

$$y_{n+1}^{(a)} = y_{n+1}^{(p)} + (y_n^{(ta)} - y_n^{(p)})\,. \tag{5.15}$$

Actually, there is trouble getting started with (5.15). If one has only y_0 and y_1, there is no way to get $y_1^{(p)}$, and so (5.15) cannot be used to get $y_2^{(a)}$. We will discuss this later. However, once one has got started, we will define

$y_{n+1}^{(a)}$ by (5.15) rather than (5.13), and use this in (5.14).

Though $y_{n+1}^{(ta)}$ is closer to $y_{n+1}^{(c)}$ than $y_{n+1}^{(t)}$ is, it is not necessarily closer to the true value of y_{n+1}. Indeed, numerical results which will be given shortly seem to support the proposition that use of $y_{n+1}^{(ta)}$ rather than $y_{n+1}^{(t)}$ results in somewhat poorer results about half of the time. However it results in appreciably better results the other half of the time. Also, its behavior is quite a bit less erratic than for $y_{n+1}^{(t)}$. So, on the whole, it seems a good idea.

To implement this, we would have to allocate an extra memory location to store $y_n^{(ta)} - y_n^{(p)}$ for use in calculating $y_{n+1}^{(a)}$ for the next step. If one is pressed for memory space, one may have to forego use of $y_{n+1}^{(a)}$. Also, there is difficulty about calculating the first instance of $y_n^{(ta)}$. If one has y_0 given, and y_1 estimated by some means, one can follow the suggestion of Hamming, 1962, on p. 206, that one would simply set $y_2^{(a)} = y_2^{(p)}$. That is, in (5.15), we take $y_1^{(ta)} - y_1^{(p)} = 0$. What this amounts to is that we use $y_2^{(t)}$ as an approximation for y_2. It may indeed be as good an approximation for y_2 as we had earlier got for y_1. Once we have done this, we can continue on by (5.14) and (5.15). Alternatively, one can generate an approximation for y_2 by the same means that we got an approximation for y_1. Then we can get $y_3^{(a)}$ by (5.13), which is close to (5.15), after which one can use (5.15) for subsequent steps.

It is again tempting to try for more accuracy by defining

$$y_{n+1}^{(ea)} = y_{n+1}^{(ta)} - \frac{1}{6}\left(y_{n+1}^{(ta)} - y_{n+1}^{(p)}\right), \tag{5.16}$$

and then taking y_{n+1} to be $y_{n+1}^{(ea)}$. This has the same advantages and disadvantages as using $y_{n+1}^{(e)}$, but appears to be less erratic.

If we are taking y_{n+1} to be $y^{(ta)}$, then the two function evaluations that would be available to put on the right side of (5.11) would be with

$y + \varepsilon = y_{n+1}^{(ta)}$ and $y = y_{n+1}^{(a)}$. For the second order Adams method that we are considering, this should work well. However, in Hamming, 1962, on p. 207, concern is expressed that, for a high order Adams method, $y_{n+1}^{(ta)}$ and $y_{n+1}^{(a)}$ might be so close together that ε and the numerator on the right of (5.11) could be appreciably altered by round off error, so that use of (5.11) would give a poor estimate of $\partial f(x,y)/\partial y$. One should not be working close enough to the bounds for (5.10) that a sharp estimate for $\partial f(x,y)/\partial y$ is required. However, one should investigate the round off properties of his calculator to see if there is danger of a really poor answer from (5.11). If there is, one will occasionally have to use an extra function evaluation to get an extra value $f(x_{n+1},y)$ from which to estimate $\partial f(x,y)/\partial y$ by (5.11).

To give some assessment of the merits of the preceding procedures, we have calculated the relative errors that would result at $x = 6$. This has been done for the three equations $y' = y$, $y' = -y$, and $y' = -2xy^2$, all with $y(0) = 1$, and for three cases $h = 0.1$, 0.2, and 0.3. Needless to say, for $y' = -2xy^2$, the solution is $y = (1+x^2)^{-1}$. The row labelled R-K in Table 1 gives the result of the Runge-Kutta of order two that we discussed earlier. The rows labelled "t", "e", etc. refer to the cases where $y_{n+1}^{(t)}$, $y_{n+1}^{(e)}$, etc. are used for y_{n+1}. The rows labelled "tH" and "eH" will be explained shortly.

Reference to Table 1 will verify some comments which were made before. The behavior of $y_{n+1}^{(e)}$ is distinctly erratic, the most extreme case being for $y' = y$ and $h = 0.3$, where $y_{n+1}^{(e)}$ gives a poorer result than $y_{n+1}^{(t)}$. It will be noted that results using $y_{n+1}^{(ta)}$ are poorer than $y_{n+1}^{(t)}$ for $y' = y$ and better for $y' = -y$, but on the whole less erratic than for $y_{n+1}^{(t)}$.

In both Hamming, 1962, and Ralston, 1965, there is advocacy of using $y_{n+1}^{(H)}$ rather than $y_{n+1}^{(a)}$, where we define

$$y_{n+1}^{(H)} = y_{n+1}^{(p)} + \frac{5}{6}\,(y_n^{(tH)} - y_n^{(p)}), \tag{5.17}$$

TABLE 1

Relative error at x = 6 for second order Adams.

		$y' = y$	$y' = -y$	$y' = -2xy^2$
h = 0.1	R-K	9.24×10^{-3}	-1.08×10^{-2}	-1.52×10^{-3}
	t	-3.65×10^{-3}	6.65×10^{-3}	6.12×10^{-4}
	ta	-4.88×10^{-3}	4.83×10^{-3}	4.55×10^{-4}
	tH	-4.67×10^{-3}	5.12×10^{-3}	4.81×10^{-4}
	e	7.10×10^{-4}	8.84×10^{-4}	8.91×10^{-5}
	ea	-3.26×10^{-4}	-6.12×10^{-4}	-4.23×10^{-5}
	eH	-1.47×10^{-4}	-3.74×10^{-4}	-2.09×10^{-5}
h = 0.2	R-K	3.39×10^{-2}	-4.76×10^{-2}	-6.56×10^{-3}
	t	-1.02×10^{-2}	3.46×10^{-2}	3.21×10^{-3}
	ta	-1.83×10^{-2}	1.71×10^{-2}	1.89×10^{-3}
	tH	-1.69×10^{-2}	1.97×10^{-2}	2.12×10^{-3}
	e	5.09×10^{-3}	7.90×10^{-3}	7.63×10^{-4}
	ea	-1.73×10^{-3}	-6.29×10^{-3}	-3.26×10^{-4}
	eH	-5.11×10^{-4}	-4.16×10^{-3}	-1.50×10^{-4}
h = 0.3	R-K	6.96×10^{-2}	-1.19×10^{-1}	-1.60×10^{-2}
	t	-1.47×10^{-2}	9.89×10^{-2}	9.47×10^{-3}
	ta	-3.73×10^{-2}	2.90×10^{-2}	4.61×10^{-3}
	tH	-3.31×10^{-2}	3.88×10^{-2}	5.40×10^{-3}
	e	1.54×10^{-2}	2.98×10^{-2}	2.81×10^{-3}
	ea	-3.40×10^{-3}	-2.64×10^{-2}	-9.35×10^{-4}
	eH	4.47×10^{-5}	-1.85×10^{-2}	-3.39×10^{-4}

having defined

$$y_{n+1}^{(tH)} = y_n + \frac{h}{2}\,(f(x_{n+1},\, y_{n+1}^{(H)}) + y_n') \;; \tag{5.18}$$

we take y_{n+1} to be $y_{n+1}^{(tH)}$. (The superscript "H" stands for "Hamming.") This appears at the appropriate place in Table 1. Hamming further proposes taking

$$y_{n+1}^{(eH)} = y_{n+1}^{(tH)} - \frac{1}{6}\,(y_{n+1}^{(tH)} - y_{n+1}^{(p)}) \tag{5.19}$$

and using $y_{n+1}^{(eH)}$ for y_{n+1}, though Ralston frowns on this (see p. 186 of Ralston, 1965). The results of this also are shown in Table 1.

Hamming's rationale seems to be as follows. Subtracting $y_{n+1}^{(p)}$ from both sides of (5.19) gives

$$y_{n+1}^{(eH)} - y_{n+1}^{(p)} = \frac{5}{6}\,(y_{n+1}^{(tH)} - y_{n+1}^{(p)}) \;. \tag{5.20}$$

So we can write (5.17) as

$$y_{n+1}^{(H)} = y_{n+1}^{(p)} + (y_n^{(eH)} - y_n^{(p)}) \;. \tag{5.21}$$

If there is very little change of $y_n^{(eH)} - y_n^{(p)}$ from n to $n+1$, then we are very nearly taking $y_{n+1}^{(H)}$ to be $y_{n+1}^{(eH)}$. As expressed on p. 206 of Hamming, 1972, we "mop up" the error of $y_{n+1}^{(p)}$. Indeed, later on the same page there is the suggestion that one might just take $y_{n+1}^{(H)}$ to be y_{n+1}, and so cut down the number of function evaluations to one per step. Presumably one still has to go through the evaluation of $y_{n+1}^{(eH)}$ at each step, to have it available for use in (5.21) at the next step, but for y_{n+1}' in the next predictor we would use

$$f(x_{n+1},\, y_{n+1}^{(H)}) \;.$$

Actually, one cannot quite hold it down to one evaluation per step, since one will occasionally need two evaluations in a step for use in (5.11). Incidentally, Hamming does not make an analysis of the stability of this proceeding. It has the disadvantage that there is no way to estimate the step by step error.

Actually, if (5.19) is to give a third order method, then $y_{n+1}^{(tH)}$ should really be $y_{n+1}^{(c)}$. For this, we should like $y_{n+1}^{(H)}$ in (5.18) to be as close as

possible to $y_{n+1}^{(c)}$. But the definition (5.17), being equivalent to (5.21), is liable to make $y_{n+1}^{(H)}$ closer to $y_{n+1}^{(eH)}$ than to $y_{n+1}^{(c)}$, and the earlier procedure of using $y_{n+1}^{(ta)}$ would seem preferable. However, according to Table 1, $y_{n+1}^{(eH)}$ seems to do a shade better than $y_{n+1}^{(ea)}$. The very erratic value at $y' = y$, $h = 0.3$ for $y_{n+1}^{(eH)}$ is disquieting.

If one just stops at $y_{n+1}^{(tH)}$, and takes this to be y_{n+1}, instead of going on to $y_{n+1}^{(eH)}$ (this is what is proposed in Ralston, 1965, on p. 189), it seems better to use the simpler $y_{n+1}^{(ta)}$. From what numerical evidence we have, $y_{n+1}^{(ta)}$ is better than $y_{n+1}^{(tH)}$ in half of the cases, and worse in half.

To get started, we calculated y_1 and y_2 from the known solutions of the equations, and then (as suggested earlier) used (5.13) to calculate $y_3^{(a)}$.

If we use these methods to solve $y' = k\,y$, we get stability in the ranges shown in Table 2. This means that for stability the bounds shown in Table 2 must be satisfied by (5.10). The fact that one has stability for all positive values is surprising. This fact is not particularly useful, since for large hk the step by step errors would be so large as to render the method of little value.

TABLE 2

Stability ranges for $y' = ky$ for Adams second order.

t	$-0.6 \leq hk$	e	$-0.8 \leq hk$
ta	$-0.7 \leq hk$	ea	$-0.7 \leq hk$
tH	$-0.7 \leq hk$	eH	$-0.7 \leq hk$

Observe that in Table 2, use of extrapolated values does not diminish the region of stability. If anything, the reverse is true.

In Section 6 we will indicate how the values in Table 2 were derived.

We now turn to third order methods. We will use the Runge-Kutta

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) , \tag{5.22}$$

where

$$k_1 = h\, f(x_n, y_n)$$
$$k_2 = h\, f(x_n + \frac{h}{2},\ y_n + \frac{k_1}{2})$$
$$k_3 = h\, f(x_n + h,\ y_n - k_1 + 2k_2) ;$$

see (5.6-46) on p. 199 of Ralston, 1965, or 25.5.8 on p. 896 of Abramowitz and Stegun, 1964.

The third order Adams is given by

$$y_{n+1}^{(p)} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + \frac{9h^4}{24} y^{(iv)}(\xi) \tag{5.23}$$

$$y_{n+1}^{(c)} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}) - \frac{h^4}{24} y^{(iv)}(\xi) . \tag{5.24}$$

Analogously to the second order Adams, we set

$$y_{n+1}^{(t)} = y_n + \frac{h}{12}(5f(x_{n+1}, y_{n+1}^{(p)}) + 8y'_n - y'_{n-1}) \tag{5.25}$$

$$y_{n+1}^{(e)} = y_{n+1}^{(t)} - \frac{1}{10}(y_{n+1}^{(t)} - y_{n+1}^{(p)}) \tag{5.26}$$

$$y_{n+1}^{(a)} = y_{n+1}^{(p)} + (y_n^{(ta)} - y_n^{(p)}) \tag{5.27}$$

$$y_{n+1}^{(ta)} = y_n + \frac{h}{12}(5f(x_{n+1}, y_{n+1}^{(a)}) + 8y'_n - y'_{n-1}) \tag{5.28}$$

$$y_{n+1}^{(ea)} = y_{n+1}^{(ta)} - \frac{1}{10}(y_{n+1}^{(ta)} - y_{n+1}^{(p)}) \tag{5.29}$$

$$y_{n+1}^{(H)} = y_{n+1}^{(p)} + \frac{9}{10}(y_n^{(tH)} - y_n^{(p)}) \tag{5.30}$$

$$y_{n+1}^{(tH)} = y_n + \frac{h}{12}(5f(x_{n+1}, y_{n+1}^{(H)}) + 8y'_n - y'_{n-1}) \tag{5.31}$$

$$y_{n+1}^{(eH)} = y_{n+1}^{(tH)} - \frac{1}{10}(y_{n+1}^{(tH)} - y_{n+1}^{(p)}) . \tag{5.32}$$

TABLE 3

Relative error at $x = 6$ for third order Adams.

		$y' = y$	$y' = -y$	$y' = -2xy^2$
h = 0.1	R-K	2.31×10^{-4}	2.71×10^{-4}	3.31×10^{-5}
	t	-1.50×10^{-4}	-3.80×10^{-4}	-2.06×10^{-5}
	ta	-2.32×10^{-4}	-2.47×10^{-4}	-1.82×10^{-5}
	tH	-2.24×10^{-4}	-2.60×10^{-4}	-1.85×10^{-5}
	e	5.90×10^{-5}	-7.98×10^{-5}	-9.36×10^{-7}
	ea	-1.50×10^{-5}	3.82×10^{-5}	1.31×10^{-6}
	eH	-7.38×10^{-6}	2.69×10^{-5}	1.04×10^{-6}
h = 0.2	R-K	1.70×10^{-3}	2.35×10^{-3}	3.03×10^{-4}
	t	-5.96×10^{-4}	-4.38×10^{-3}	-1.27×10^{-4}
	ta	-1.62×10^{-3}	-1.71×10^{-3}	-1.76×10^{-4}
	tH	-1.51×10^{-3}	-1.95×10^{-3}	-1.76×10^{-4}
	e	8.14×10^{-4}	-1.49×10^{-3}	9.04×10^{-6}
	ea	-1.17×10^{-4}	8.73×10^{-4}	-2.77×10^{-5}
	eH	-1.85×10^{-5}	6.58×10^{-4}	-2.78×10^{-5}
h = 0.3	R-K	5.30×10^{-3}	8.56×10^{-3}	1.19×10^{-3}
	t	-4.58×10^{-4}	-2.06×10^{-2}	-3.48×10^{-5}
	ta	-4.49×10^{-3}	-3.70×10^{-3}	-8.99×10^{-4}
	tH	-4.06×10^{-3}	-5.15×10^{-3}	-4.05×10^{-4}
	e	3.56×10^{-3}	-8.84×10^{-3}	2.52×10^{-4}
	ea	-1.33×10^{-4}	6.04×10^{-3}	-4.05×10^{-4}
	eH	2.67×10^{-4}	4.77×10^{-3}	-3.97×10^{-4}

To get started, we can estimate y_1 and y_2 by some means and then take $y_3^{(a)} = y_n^{(p)}$; this amounts to approximating y_3 by $y_3^{(t)}$, and is what was done to get the values in Table 3. Alternatively one can estimate y_1, y_2, and y_3 by some means, and then take either

$$y_4^{(a)} = y_4^{(p)} + (y_3^{(t)} - y_3^{(p)})$$

or

$$y_4^{(H)} = y_4^{(p)} + \frac{9}{10}(y_3^{(t)} - y_3^{(p)}) ;$$

subsequently one uses either (5.27) or (5.30).

Some numerical results are given in Table 3. The column headed $y' = -2xy^2$ is quite erratic. This bears out the remark at the top of p. 210 in Hamming, 1962, that the corresponding solution is often troublesome to approximate by polynomials.

For the equation $y' = ky$, we get stability in the ranges shown in Table 4. These bounds should be satisfied by (5.10). For the case $h = 0.3$ for $y' = -2xy^2$, the bounds are not satisfied by (5.10) for a region near $x = 1$.

TABLE 4

Stability ranges for $y' = ky$ for Adams third order.

t	$-0.8 \leq hk$	e	$-0.9 \leq hk$
ta	$-0.5 \leq hk$	ea	$-0.5 \leq hk$
tH	$-0.5 \leq hk$	eH	$-0.5 \leq hk$

However, we got through the region of instability in two or three steps, which were not enough for the instability to build up appreciably. For most of the range of integration, (5.10) was well within the bounds of stability.

As with the second order Adams method, use of extrapolated values does not diminish the region of stability.

To get some feeling how erratic some of the values are in Table 3, note that R-K, t, ta, and tH are supposed to be third order methods. Thus the error for $h = 0.2$ should be 8 times that for $h = 0.1$, and the error for $h = 0.3$ should be 27 times that for $h = 0.1$. This works out reasonably well except for t.

As pointed out on p. 186 of Ralston, 1965, use of

$$y_{n+1} = y_{n+1}^{(c)} - \frac{1}{10}\left(y_{n+1}^{(c)} - y_{n+1}^{(p)}\right) \tag{5.33}$$

should give a fourth order method. As $y_{n+1}^{(t)}$, $y_{n+1}^{(ta)}$, and $y_{n+1}^{(tH)}$ are close to $y_{n+1}^{(c)}$, use of (5.26), (5.29), and (5.32) means that e, ea, and eH should be close to fourth order methods. So we look for the error for $h = 0.2$ to be 16 times that for $h = 0.1$, and the error for $h = 0.3$ to be 81 times that for $h = 0.1$. None of e, ea, or eH comes very close to such behavior. Nor can one count on a striking increase in accuracy from using extrapolated values. In two cases eH is only barely better than tH, and in one case ea is poorer than ta. Considering that there is no way to estimate step by step error when one is using extrapolated methods, they should probably not be used.

For the fourth order Runge-Kutta, consult (4.3) and (4.4), which are generalized versions. For the fourth order Adams method (see Conte and de Boor, 1972, p. 342 and p. 351), we set

$$y_{n+1}^{(p)} = y_n + \frac{h}{24}\left(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}\right) + \frac{251h^5}{720}y^{(v)}(\xi) \tag{5.34}$$

$$y_{n+1}^{(c)} = y_n + \frac{h}{24}\left(9y'_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}\right) - \frac{19h^5}{720}y^{(v)}(\xi). \tag{5.35}$$

These give

$$y_{n+1}^{(t)} = y_n + \frac{h}{24}\left(9f(x_{n+1}, y_{n+1}^{(p)}) + 19y'_n - 5y'_{n-1} + y'_{n-2}\right) \tag{5.36}$$

$$y_{n+1}^{(e)} = y_{n+1}^{(t)} - \frac{19}{270}\left(y_{n+1}^{(t)} - y_{n+1}^{(p)}\right) \tag{5.37}$$

$$y_{n+1}^{(a)} = y_{n+1}^{(p)} + \left(y_n^{(ta)} - y_n^{(p)}\right) \tag{5.38}$$

TABLE 5

Relative error at $x = 6$ for fourth order Adams.

		$y' = y$	$y' = -y$	$y' = -2xy^2$
h = 0.1	R-K	4.62×10^{-6}	-5.44×10^{-6}	-8.10×10^{-7}
	t	-7.65×10^{-6}	2.73×10^{-5}	9.87×10^{-7}
	ta	-1.39×10^{-5}	1.61×10^{-5}	1.02×10^{-6}
	tH	-1.35×10^{-5}	1.69×10^{-5}	1.02×10^{-6}
	e	5.11×10^{-6}	7.50×10^{-6}	-5.63×10^{-7}
	ea	-8.10×10^{-7}	-2.88×10^{-6}	-5.41×10^{-7}
	eH	-4.32×10^{-7}	-2.17×10^{-6}	-5.40×10^{-7}
h = 0.2	R-K	6.77×10^{-5}	-9.46×10^{-5}	-1.37×10^{-5}
	t	-2.98×10^{-5}	6.93×10^{-4}	-5.46×10^{-5}
	ta	-1.81×10^{-4}	2.18×10^{-4}	5.65×10^{-6}
	tH	-1.70×10^{-4}	2.48×10^{-4}	2.11×10^{-6}
	e	1.34×10^{-4}	2.94×10^{-4}	-4.97×10^{-5}
	ea	-7.66×10^{-6}	-1.44×10^{-4}	4.80×10^{-6}
	eH	2.98×10^{-6}	-1.15×10^{-4}	1.60×10^{-6}
h = 0.3	R-K	3.16×10^{-4}	-5.21×10^{-4}	-7.19×10^{-5}
	t	1.71×10^{-4}	5.30×10^{-3}	-9.46×10^{-4}
	ta	-6.85×10^{-4}	5.39×10^{-4}	5.59×10^{-4}
	tH	-6.20×10^{-4}	8.28×10^{-4}	4.55×10^{-4}
	e	8.43×10^{-4}	2.74×10^{-3}	-6.92×10^{-4}
	ea	3.70×10^{-5}	-1.61×10^{-3}	6.37×10^{-4}
	eH	9.81×10^{-5}	-1.35×10^{-3}	5.48×10^{-4}

$$y_{n+1}^{(ta)} = y_n + \frac{h}{24}\left(9f(x_{n+1}, y_{n+1}^{(a)}) + 19y'_n - 5y'_{n-1} + y'_{n-2}\right) \tag{5.39}$$

$$y_{n+1}^{(ea)} = y_{n+1}^{(ta)} - \frac{19}{270}\left(y_{n+1}^{(ta)} - y_{n+1}^{(p)}\right) \tag{5.40}$$

$$y_{n+1}^{(H)} = y_{n+1}^{(p)} + \frac{251}{270}\left(y_n^{(tH)} - y_n^{(p)}\right) \tag{5.41}$$

$$y_{n+1}^{(tH)} = y_n + \frac{h}{24}\left(9f(x_{n+1}, y_{n+1}^{(H)}) + 19y'_n - 5y'_{n-1} + y'_{n-2}\right) \tag{5.42}$$

$$y_{n+1}^{(eH)} = y_{n+1}^{(tH)} - \frac{19}{270}\left(y_{n+1}^{(tH)} - y_{n+1}^{(p)}\right) . \tag{5.43}$$

The matter of getting started can be handled as in the second and third order Adams methods.

Some numerical results are given in Table 5. The extrapolated values are distinctly erratic, as is also $y_{n+1}^{(t)}$. However, $y_{n+1}^{(ta)}$ behaves fairly well. The excellent results from the Runge-Kutta are worth remarking.

For the equation $y' = ky$, we get stability in the ranges shown in Table 6. These bounds should be satisfied by (5.10). For $h = 0.3$ for $y' = -2xy^2$, this was not the case for a short interval, not long enough to cause any trouble.

TABLE 6

Stability ranges for $y' = ky$ for Adams fourth order.

t	$-0.6 \leq hk$	e	$-0.6 \leq hk$
ta	$-0.4 \leq hk$	ea	$-0.4 \leq hk$
tH	$-0.4 \leq hk$	eH	$-0.4 \leq hk$

Use of extrapolated values does not diminish the region of stability.

6. Milne type predictor-correctors. In Southard and Yowell, 1952, is given

$$y_{n+1}^{(p)} = -4y_n + 5y_{n-1} + h(4y'_n + 2y'_{n-1}) + \frac{h^4}{6} y^{(iv)}(\xi) \tag{6.1}$$

$$y_{n+1}^{(c)} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}) - \frac{h^4}{24} y^{(iv)}(\xi) \ . \tag{6.2}$$

One will recognize that the corrector is the same as for the third order Adams, to wit (5.24). So the formula for $y_{n+1}^{(t)}$ would be the same as (5.25). We would have

$$y_{n+1}^{(e)} = y_{n+1}^{(t)} - \frac{1}{5}(y_{n+1}^{(t)} - y_{n+1}^{(p)}) \ , \tag{6.3}$$

and the other formulas analogously.

This predictor-corrector has some good features. It is third order, but has modest memory requirements, and is easy to get started, or get restarted after changing the length of the step. However, it behaves poorly as to stability, as can be seen from Table 7 and Table 8. The fact that all positive values of hk are excluded for ea and eH is startling.

Because the range of stability for $y_{n+1}^{(t)}$ is so limited, one finds (3.10) outside that range in the case $h = 0.3$ and $y' = -2xy^2$ for an extended period, and the solution really blows up. Already at $x = 3$, one gets a negative value for y, after which the errors compound catastrophically. Before reaching $x = 6$, the calculator stops on an overflow, because the numbers are too large for its capacity.

If one had been monitoring (5.10), this instability could have been avoided by using a smaller h through the critical region.

The analysis of the stability for $y_{n+1}^{(t)}$ and $y_{n+1}^{(e)}$ would be worthy of special attention by anyone interested in stability. Neither the usual Dahlquist criterion of stability nor the Hamming-Ralston criterion for relative stability is applicable, and a special definition had to be contrived. Never mind the details. The reader can trust that if (5.10) stays too long outside the bounds

TABLE 7

Relative error at $x = 6$ for Southard and Yowell.

		$y' = y$	$y' = -y$	$y' = -2xy^2$
$h = 0.1$	R-K	2.31×10^{-4}	2.71×10^{-4}	3.31×10^{-5}
	t	-1.64×10^{-4}	-4.18×10^{-4}	-1.50×10^{-5}
	ta	-2.32×10^{-4}	-2.48×10^{-4}	-2.01×10^{-5}
	tH	-2.17×10^{-4}	-2.73×10^{-4}	-2.09×10^{-5}
	e	1.18×10^{-5}	-1.52×10^{-5}	-1.82×10^{-7}
	ea	unstable	9.43×10^{-6}	5.77×10^{-7}
	eH	unstable	5.05×10^{-6}	4.16×10^{-7}
$h = 0.2$	R-K	1.70×10^{-3}	2.35×10^{-3}	3.03×10^{-4}
	t	-8.99×10^{-4}	-7.16×10^{-3}	4.54×10^{-3}
	ta	-1.63×10^{-3}	-1.75×10^{-3}	7.17×10^{-5}
	tH	-1.45×10^{-3}	-2.20×10^{-3}	-1.34×10^{-4}
	e	1.70×10^{-4}	-2.84×10^{-4}	4.00×10^{-6}
	ea	unstable	1.79×10^{-4}	4.03×10^{-5}
	eH	unstable	1.09×10^{-4}	1.05×10^{-5}
$h = 0.3$	R-K	5.30×10^{-3}	8.56×10^{-3}	1.19×10^{-3}
	t	-2.09×10^{-3}	-2.79×10^{-1}	unstable
	ta	-4.66×10^{-3}	unstable	1.30×10^{-1}
	tH	-4.00×10^{-3}	unstable	3.45×10^{-2}
	e	7.79×10^{-4}	-1.72×10^{-3}	1.59×10^{-4}
	ea	unstable	unstable	unstable
	eH	unstable	unstable	1.04×10^{-1}

TABLE 8

Stability ranges for $y' = ky$ for Southard and Yowell.

t	$-0.31 \leq hk$	e	$-0.7 \leq hk$
ta	$-0.25 \leq hk$	ea	$-0.25 \leq hk < 0$
tH	$-0.28 \leq hk$	eH	$-0.28 \leq hk < 0$

given in Table 8, there will be trouble; see the case $h = 0.3$ for $y' = -2xy^2$ for example.

In Hamming, 1959, on p. 47, concern is expressed about the stability of Southard and Yowell, and it is proposed to remedy the situation by using a different predictor

$$(6.4) \qquad y_{n+1}^{(p)} = y_n + y_{n-1} - y_{n-2} + 2h(y'_n - y'_{n-1}) + \frac{h^4}{3} y^{(iv)}(\xi) .$$

This appreciably increases the number of memory locations needed and requires more starting values to get started or restarted, thus cancelling out the two good features of Southard and Yowell. It does appear to improve the stability; for $y_{n+1}^{(t)}$ it is increased to $-0.5 \leq hk$. However, this is considerably less than for the Adams method of order three. If we compare the Adams of order three with what results from (6.4) we find that both are of order three, both need two extra starting values to get started or restarted, but the Adams is considerably more stable and requires fewer memory locations.

In Hamming, 1959, it was proposed to render the classical Milne predictor-corrector stable by using a different corrector, namely

$$(6.5) \qquad y_{n+1}^{(c)} = \frac{1}{8}(9y_n - y_{n-2} + 3h(y'_{n+1} + 2y'_n - y'_{n-1})) - \frac{h^5}{40} y^{(v)}(\xi) .$$

the old predictor, (4.5), was retained. The results of this are shown in Table 9 and Table 10. In Table 9 there is no column $y' = -2xy^2$ because the memory requirements exceeded the capacity of the HP-65 with which the calculations of this paper were performed. Although there is a region of stability, it is less than for the Adams method of comparable order, namely order four. All in all, the Hamming method is not a contender for use with hand held calculators.

From Ralston, 1965, one would get the impression that Hamming's method is very superior. It is $y_{n+1}^{(tH)}$ that gets the stamp of approval on p. 189 of Ralston, 1965. Actually, $y_{n+1}^{(ta)}$ would be simpler, and gives about the same results. We do not believe that $y_{n+1}^{(tH)}$ is enough superior to $y_{n+1}^{(ta)}$ to be

TABLE 9

Relative error at $x = 6$ for Hamming.

		$y' = y$	$y' = -y$
$h = 0.1$	R-K	4.62×10^{-6}	-5.44×10^{-6}
	t	-8.63×10^{-6}	4.55×10^{-5}
	ta	-1.68×10^{-5}	2.10×10^{-5}
	tH	-1.61×10^{-5}	2.24×10^{-5}
	e	4.24×10^{-6}	7.52×10^{-6}
	ea	-8.10×10^{-7}	-2.55×10^{-6}
	eH	-4.32×10^{-7}	-1.92×10^{-6}
$h = 0.2$	R-K	6.77×10^{-5}	-9.46×10^{-5}
	t	-3.87×10^{-5}	2.37×10^{-3}
	ta	-2.07×10^{-4}	2.82×10^{-4}
	tH	-1.92×10^{-4}	3.35×10^{-4}
	e	1.07×10^{-4}	3.73×10^{-4}
	ea	-8.49×10^{-6}	-1.15×10^{-4}
	eH	1.60×10^{-6}	-9.44×10^{-5}
$h = 0.3$	R-K	3.16×10^{-4}	-5.21×10^{-4}
	t	1.05×10^{-4}	unstable
	ta	-7.53×10^{-4}	6.78×10^{-4}
	tH	-6.75×10^{-4}	1.08×10^{-3}
	e	6.63×10^{-4}	5.90×10^{-3}
	ea	2.32×10^{-5}	-1.10×10^{-3}
	eH	8.02×10^{-5}	-9.56×10^{-4}

TABLE 10

Stability ranges for $y' = ky$ for Hamming.

t	$-0.26 \leq hk$	e	$-0.38 \leq hk$
ta	$-0.37 \leq hk$	ea	$-0.38 \leq hk$
tH	$-0.38 \leq hk$	eH	$-0.39 \leq hk$

worth the extra trouble of programming and the longer running time that it entails.

Despite Ralston's endorsement of Hamming, Enright and Hull, 1976, give it a very low rating on pp. 954-955. Interestingly enough, though Hamming invented the method, and devotes a lot of space discussing it in Hamming, 1962, he finally (in pp. 206-210) seems to favor something related to an Adams method.

On p. 46 of Hamming, 1959, the stability for $y_{n+1}^{(eH)}$ is claimed to be good down to about -0.65. As this disagrees with the value given in Table 10, we will justify the value in Table 10.

For Hamming's method, we have

$$y_{n+1}^{(H)} = y_{n+1}^{(p)} + \frac{112}{121}\,(y_n^{(tH)} - y_n^{(p)}) \tag{6.6}$$

$$y_{n+1}^{(eH)} = y_{n+1}^{(tH)} - \frac{9}{121}\,(y_{n+1}^{(tH)} - y_{n+1}^{p)}) \,. \tag{6.7}$$

Subtracting $y_{n+1}^{(p)}$ from both sides of (6.7) gives

$$y_{n+1}^{(eH)} - y_{n+1}^{(p)} = \frac{112}{121}\,(y_{n+1}^{(tH)} - y_{n+1}^{(p)}) \,. \tag{6.8}$$

So, by (6.6)

$$y_{n+2}^{(H)} = y_{n+2}^{(p)} + (y_{n+1}^{(eH)} - y_{n+1}^{(p)}) \,. \tag{6.9}$$

As the entries that are accepted are those with a superscript (eH), we shall simplify the notation by omitting this. Then by (4.5) and (6.9), we see that for the equation $y' = ky$, we have

$$y_{n+2}^{(H)} = \{y_{n-2} + \frac{4hk}{3}(2y_{n+1} - y_n + 2y_{n-1})\} + (y_{n+1} - \{y_{n-3} + \frac{4hk}{3}\,(2y_n - y_{n-1} + 2y_{n-2})\}). \tag{6.10}$$

This simplifies to

$$y_{n+2}^{(H)} = y_{n+1} + y_{n-2} - y_{n-3} + \frac{4hk}{3}(2y_{n+1} - 3y_n + 3y_{n-1} - 2y_{n-2}). \tag{6.11}$$

We have, by (6.5),

$$(6.12)\qquad y_{n+2}^{(tH)} = \frac{1}{8}\,(9y_{n+1} - y_{n-1} + 3hk(y_{n+2}^{(H)} + 2y_{n+1} - y_n)).$$

So

$$(6.13)\qquad y_{n+2}^{(tH)} = \frac{1}{8}\,(9y_{n+1} - y_{n-1} + hk(9y_{n+1} - 3y_n + 3y_{n-2} - 3y_{n-3})$$

$$+ (hk)^2(8y_{n+1} - 12y_n + 12y_{n-1} - 8y_{n-2})).$$

By (6.7)

$$(6.14)\qquad y_{n+2} = \frac{112}{121}\,y_{n+2}^{(tH)} + \frac{9}{121}\,y_{n+2}^{(p)}\ .$$

By (4.5) and (6.13), this gives

$$(6.15)\qquad y_{n+2} = \frac{1}{121}\,(126y_{n+1} - 14y_{n-1} + 9y_{n-2}$$

$$+ hk\,(150y_{n+1} - 54y_n + 24y_{n-1} + 42y_{n-2} - 42y_{n-3})$$

$$+ (hk)^2(112y_{n+1} - 168y_n + 168y_{n-1} - 112y_{n-2})).$$

The solution of this difference equation is

$$(6.16)\qquad y_n = \sum_{i=1}^{5} C_i\,\rho_i^n\ ,$$

where the ρ_i are the roots of the equation

$$(6.17)\qquad 121\rho^5 - (126 + 150hk + 112(hk)^2)\rho^4 + (54hk + 168(hk)^2)\rho^3$$

$$+ (14 - 24hk - 168(hk)^2)\rho^2 - (9 + 42hk - 112(hk)^2)\rho + 42hk = 0\ .$$

If we take $hk = -0.4$, we get

$$(6.18)\qquad 121\rho^5 - 83.92\rho^4 + 5.28\rho^3 - 3.28\rho^2 + 25.72\rho - 16.8 = 0\ .$$

The polynomial has the factors

$$(6.19)\qquad \rho - 0.67053\ \ 10701$$

$$(6.20)\qquad \rho^2 + 0.92764\ 94248\rho + 0.45383\ \ 23357$$

$$(6.21)\qquad \rho^2 - 0.95067\ \ 20739\rho + 0.45625\ \ 70288.$$

The zeros of (6.20) have absolute value

0.67367 07918

and the zeros of (6.21) have absolute value

$$0.67546\ \ 80072\ .$$

For large n, the powers of these quantities will predominate, and the values of y_n will jump about very erratically.

For $hk = -0.39$, the factor corresponding to (6.19) will produce the dominant ρ, and (6.16) will approximate e^{hkn}, as it should.

Amongst the disadvantages of the Hamming method for hand held calculators is its excessive requirement for memory locations. To improve this, one could change the predictor to

$$(6.22) \qquad y_{n+1}^{(p)} = -9y_n + 9y_{n-1} + y_{n-2} + h(6y'_n + 6y'_{n-1}) + \frac{h^5}{10} y^{(v)}(\xi)\ .$$

When used with the Hamming corrector, (6.5), this still gives a fourth order method. However, it requires two fewer memory locations. Also, it requires fewer starting values to get a start or restart, being in this respect also superior to the Adams method of order four.

This predictor is one of a set which, in a footnote on p. 171 of Hamming, 1962, is dismissed as not being worth consideration. The combination of (6.22) and (6.5) does turn out to have very poor stability characteristics. For $y_{n+1}^{(t)}$ one must have $-0.13 \leq hk$, and for $y_{n+1}^{(ta)}$ one must have $-0.19 \leq hk \leq 0.28$. At this point, we became disheartened, and did not check out the other cases.

7. Conclusions. Despite heroic efforts to remedy various faults of the Milne type predictor-correctors, they are still decidedly inferior to the Adams type. One would prefer if the Adams type results were less erratic than they are, but if the user is diligent to keep (5.10) within bounds, it appears that one can use the Adams methods safely, and without running into excessive calculation times. An occasional "look ahead" to plan the progress of the calculation is advisable.

Runge-Kutta methods have many advantages and, if it happens that $f(x,y)$ can be evaluated quickly, they should be given serious consideration. Although the three Runge-Kutta's given in this paper are relatively stable under all circumstances, so that one can use them without any concern about stability, one must still monitor the size of h carefully, since if one allows the step by step errors to become excessive the final results can be almost without meaning.

REFERENCES

M. Abramowitz and I. A. Stegun, "Handbook of mathematical functions," Applied Mathematics Series 55, Natl. Bureau of Standards, 1964.

R. Bulirsch and R. Stoer, "Numerical treatment of ordinary differential equations by extrapolation methods," Numer. Math., vol. 8 (1966), pp. 1-13.

S. D. Conte and Carl de Boor, "Elementary numerical analysis," second edition, McGraw-Hill Book Co., 1972.

G. Dahlquist, "Convergence and stability in the numerical integration of ordinary differential equations," Math. Scand., vol. 4 (1956), pp. 33-53.

W. H. Enright and T. E. Hull, "Test results on initial value methods for non-stiff ordinary differential equations," SIAM Jour. Num. Anal., vol. 13 (1976), pp. 944-961.

E. Fehlberg, "Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control," NASA Tech. Report No. 287, 1968, Huntsville.

E. Fehlberg, "Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle," Computing, vol. 4 (1969), pp. 93-106.

E. Fehlberg, "Klassische Runge-Kutta-Formeln vierter und niedriger Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme," Computing, vol. 6 (1970), pp. 61-71.

W. B. Gragg, "On extrapolation algorithms for ordinary initial value problems," SIAM Jour. Num. Anal., vol. 2 (1965), pp. 384-403.

R. W. Hamming, "Stable predictor-corrector methods for ordinary differential equations," Jour. of ACM, vol. 6 (1959), pp. 37-47.

R. W. Hamming, "Numerical methods for scientists and engineers," McGraw-Hill Book Co., 1962.

Peter Henrici, "Discrete variable methods in ordinary differential equations," John Wiley and Sons, 1962.

Peter Henrici, "Computational analysis with the HP-25 pocket calculator," John Wiley and Sons, 1977.

T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick, "Comparing numerical methods for ordinary differential equations," SIAM Jour. Num. Anal., vol. 9 (1972), pp. 603-637.

W. E. Milne, "Numerical solution of differential equations," John Wiley and Sons, 1953.

Anthony Ralston, "A first course in numerical analysis," McGraw-Hill Book Co., 1965.

J. Barkley Rosser, "A Runge-Kutta for all seasons," SIAM Review, vol. 9 (1967), pp. 417-452.

L. F. Shampine and H. A. Watts, "The art of writing a Runge-Kutta Code, Part I," in "Mathematical Software III," ed. by John R. Rice, Academic Press, 1977.

T. H. Southard and E. C. Yowell, "An alternative 'predictor-corrector' process," Math. Tables and Other Aids to Comp., vol. 6 (1952), pp. 253-4.

JBR/db

THE MECHANICAL TRAIN ANALOG A PROPOSED SOFTWARE EVALUATION TOOL

Peter Beck
Raymond Chuvala
US Army Armament R&D Command
Product Assurance Directorate
Dover, New Jersey 07801

ABSTRACT - The current method of evaluating and qualifying software is accomplished by having a panel of experts exercise the software system. This is a nonanalytical, subjective procedure, since it is impossible to expose a specific software system to a sufficiently widespread scrutiny, by sufficiently diverse interests, to assure that enough significant points of potential weakness have been exposed. This shortcoming of existing software evaluation tools (SET) is the reason for proposing the mechanical train analog (MTA).

1. INTRODUCTION

The automated battlefield is a new and unique commodity where, by definition, the functioning is controlled by devices that replace the human elements of observation, effort, and data reduction. This system has two major elements: hardware and software. Hardware product assurance is reasonably well understood, while for the associated software system product assurance is an emerging specialty.

The software problem in need of clarification is how to translate the user/customer quality notions of the software into quantitative terms (i.e. scoring criteria); and to describe, in the broadest sense, those activities that must be performed by the manufacturing organizations to assure uniformity of the product and customer satisfaction. The method of quantifying the customers' quality notions must proceed according to a formalized methodology consistent with the scientific method, if customer acceptance is to be achieved. This methodology is commonly called product assurance. The product assurance task, then, is to identify and quantify customer safisfaction measures for all phases of the life cycle. Thus, the question of scoring criteria for initial qualification, replication, and followon procurements must be addressed. The preliminary evidence is that for software systems the lack of common or at least consistent standards for system specifications, test plans, and procedures (supportive manuals, handbooks, and guides) contributes to confusion and a lack of understanding of the scoring criteria applicable to the software system.

How can we tell when the product assurance discipline is sufficiently developed to assure customer satisfaction with a software system? Lord Kelvin said "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind. It may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be." This desire to have standarized quantitative measures is still valid today. It assures widespread understanding of the commodity and therefore, has the potential for achieving widespread customer satisfaction.

2. MISSION AREA RESPONSIBILITIES

It is the responsibility of the Selected Armaments System Division (SASD) to perform for mine systems the following tasks, which include mines and mine dispensing systems: to design, develop, and evaluate test programs for determining the safety and reliability, availability and Maintainability (S/RAM) engineering characteristics for demonstration of the system and subsystems' achievements. In addition, SASD performs the required safety engineering functions including the safety statement for these same materials.

The first mine system integratable into the automated battlefield is currently under development and is called the Family of Scatterable Mines (FASCAM). The introduction of FASCAM into the Army's inventory will provide the field commander with a revolutionary capability, e.g., the use of mine field barriers as a tactical weapon if the compliance with the safety and RAM requirements can be adequately demonstrated to the user. In utilizing mine fields in this new role, however, we must first be aware of the extensive safety precautions and techniques traditionally employed to control mine fields seeded with conventional, nonscatterable mines. Then, we must recognize the need to address new and better techniques for achieving a comparable or improved level of control and command over mine fields that are no longer handemplaced, but remotely scattered and sown in large quantities from high speed delivery systems at very rapid rates. The consideration of these command and control aspects of scatterable mines has led to the formulation of a concept which has been given the hypothetical designation of Minefield Information System (MIS).

To permit the greatest possible utility of the FASCAM mining systems by the field commander, the MIS shall be comprised of an interactive graphics display, automatic data processing equipment, computer programs, and appropriate communications links. The MIS will facilitate the commander's decision making by providing him with uptotheminute intelligence processed in realtime from all data vital and pertinent to the conduct of mining operations.

The information processed by the MIS command and control center and displayed on command at division headquarters will include, in part: terrain, target and weather information; a record of types and quantities of mines and delivery systems in the division's supply system and immediately available for deployment; the recommended number of minefields, minefield sizes, mixes densities and optimum locations of minefields to meet a particular threat and/or to accomplish an offensive objective; exact times of emplacing the various minefields; and an automatic armed life countdown to facilitate subsequent safe passage of friendly troops or to facilitate reseeding requirements. Additionally, MIS will have the capability to accurately map and display minefield locations and will provide a semiautomated communications link between the commander and his tactically dispersed minefields for the purpose of complete command and control of tactical minefields.

An MIS functional flow chart for a command and control barrier system might be represented as in figure 1. Based on intelligence received from the tactical operating system (TOS) or directly through the army communications system and based on the mine warfare data base stored in the computer, the MIS will recommend optimum mine fields and provide minefield mapping. In addition, it will keep an uptotheminute graphically displayed status of the minefields for the field commander's use in his overall tactical planning and, upon action by the field commander, the MIS command and control center will signal the minefields to destruct. Further, each critical MIS activity will be recorded on a hardcopy printout at the division tactical operations center. Although the hypothetical MIS function of TOS will be capable of independent operation (i.e. only interfacing with TOS), its design, as depicted in figure 2, will provide for the direct functional interface through the field army communications network with other TOS subsystems. This option is important if the field commander is to maintain a timely response to mining operations under all circumstances.

It is planned that the MIS will ultimately facilitate the integration of FASCAM mining capabilities into ideal command and control barriers for optimum use by the field commander of the future automated battlefield. This on-line, real time data processing system will provide information which is more accurate, more timely, more complete, more concise, and more relevant to decision making than the present manual technique of mine field record keeping and marking maps. Accordingly, a safety/RAM engineering effort, which will demonstrate the safe, effective use and control of FASCAM mines and their associated quick response delivery systems in realtime combat operations is imperative.

3. PROPOSED SOFTWARE EVALUATION TOOL

The complexity of future munition systems, such as FASCAM with MIS, has prompted SASD's review of software reliability to assure data security, user protection from catastrophic system failures, friendly troop safety, and the reliability of decision support information. This review indicates that there is a nonlinear relationship between the number of assigned tasks, with their respective priorities and the number of system states possible. In fact, the problem is inherent and is derived from the intricate structure of precedence constraints and the complicated relationships among the execution times. Therefore, it can be shown that the possible states of a useful software system grow so rapidly that there is no hope of examining even a small fraction of them.

Philosophy divides the understanding of truth into two broad classes; the metaphysical and the physical. Metaphysics is the science of understanding based upon intuition, function, duration, i.e., getting inside of the phenomenon. The physical understanding is usually called analysis and has been well defined by Henri Bergson as "the operation which reduces the object to elements already known, that is, to elements common both to it and other objects. To analyze, therefore, is to express a thing as a function of something other than itself. All analysis is thus a translation, a development into symbols, a representation taken from successive points of view from which we note as many resemblances as possible between the new object which we are studying and others which we believe we know already." It, therefore, stands to reason that existing SETs follow this same philosophical classification where Monte Carlo simulation, interactive gaming and deterministic evaluation are metaphysical in nature (i.e. you play with the system until you get bored) and algorithmic interrogation techniques comprise the analytical approach. The authors believe that the analytical approach is the only viable method of evaluating the inherent workability (correctness) of a system. We also believe that mathematical proofs of correctness are not well enough developed to have universally accepted standards. Therefore, this new analytical approach comprised of a standarized hardware analog is proposed.

This standarized hardware analog is called a Mechanical Train Analog. It is a model railroad system (i.e. a miniature model train set) of a sufficiently complex configuration to perform software analysis. The synthesis of the MTA was stimulated by two facts: 1) the enormous number of possible states and/or functions MIS might perform, and 2) the observation that 80-90 percent of human cognition is achieved through visual perception. We, therefore, reasoned that if a visual gage, the MTA, could be developed to display the cognitive associations of a software system and if comparative standards could also be developed, the tools necessary for software analysis would be realized.

4. MTA DESCRIPTION

The search for a standarized hardware analog led us to the world of the model builder. It became obvious immediately, that the most developed miniaturization of human activity is the domain of the model railroader. It must be emphasized that model railroading is a diversified pursuit. The traditional railroader attempts to design his layout to include all facets of wordly interactions. Therefore, consideration is given to scenery, buildings and other offtrack activities of both a static and dynamic nature. The nontraditional railroader is investigating the model world as a teaching tool for slow learners, as a tool for real time computer evaluation, as a dynamic distributed artificial intelligence array, etc. This broad based experience with model railroads, by individuals of widely differing backgrounds, confirmed our hunch of the practicability of using a model railroad as a SET.

Model railroad layouts, because of their physical existence, are rather intolerant of human oversights. The experimenter must address the problems of mixed hardware, series and parallel operation, and the randomness of human error and hardware malfunction. The hardware mixture can be as simple as varying roadbed design and/or locomotives, or as complex as transferring cargo from truck to train to truck. The multiple operating subsystems that comprise a given layout may include switch yards, subway systems, trolleys, automated loading of coal cars, etc. The diversity of hardware has caused many human misunderstandings and maintenance difficulties for complex model layouts, that require more than one person for model operation. This imperfect coordination is the human element of the MTA and is of value in evaluating the robustness of our analysis.

In addition to the realistic hardware complexity model railroads encompass an extensive supporting model railroad infrastructure is also avaliable. This infrastructure includes the standardization of scale sizes, the availability of hardware manufactured to these standards, magazines, clubs, contests, conventions, system review standards, and an everrenewing source of trained and skilled experimenters. Therefore, we can see that the MTA can serve as a visual evaluation gage where the inspector/evaluator can easily see if one operation is better than another. In addition, since the MTA operation incorporates the software operation that controls it, an analytical judgement of the quality of the software under test is available.

5. MTA APPLICATION

In the hope of clarifying the MTA's usefullness for evaluating a system, a typical application will be addressed. The example dicusses some parallels between an electric utility and the MTA. The example also discusses how the MTA might be used to evaluate a specific electric utility performance criteria. The reader should keep in mind that this illustrative application has not been run on the MTA and, therefore, is not a complete procedure.

A cursory review of electric utilities' terminology uncovers a similarity with railroad terminology. The primary elements are generating stations, both major and minor; power lines interconnecting the stations and customers (via substations and feeder lines); and switching complexes which facilitate changing the systems' routing. The utility system is stressed by equipment malfunction, weather, customer demand, cost, maintenance schedules and other time varying events. The above parallels in terminology between the MTA and an electric utility indicate that each are composed of similar elements: i.e. fixed assets which can be connected at terminals or switching stations via interconnecting lines to provide an almost infinite variety of system configurations.

The electric utility performance criteria we have chosen to evaluate is the prevention of the catastrophic system crash called a blackout. A simple definition of a blackout is a system wide malfunction which physically damages the system and also requires a significant time to restore service to the customers. Because of the costs involved, blackout prevention cannot be assured by system over capacity and/or redundancy. Therefore, most utilities have developed load shedding schemes to prevent the malignant spread of system malfunctions. This scheme has the intended purpose of containing and isolating malfunctions. The MTA can be used to visualize how the translated electric utilities system, including its load shedding scheme, are affected by random malfunctions, one analysis that might be performed is a hardware failure

modes, effects and criticality analysis where every MTA element incorporated in the translated electric utility system is intentionally malfunctioned and its effect on the system is noted. It will be immediately obvious which malfunctions propagate throughout the system causing a total stoppage of trains (blackout). It will allow us to investigate the effects of fault isolation schemes and of the procedures used to restart the system.

Therefore, it can be seen, that the MTA will perform the task of a gage, displaying the entire dynamic system under study: hardware and software defects will show up as nonnormal system functioning. Experience has shown that the human mind and eye can detect irregularities, when the entire systems interrelationships are observed, that are normally not statistically noteworthy and, therefore, are considered insignificant. This ability to watch the system run will allow us to find bottlenecks and critical paths which may have the potential for causing system instabilities leading to catastrophic system crashes.

6. MTA INVESTIGATION PLAN

The intuitively appealing hypothesis of the MTA must find a mode of expression and of application which conforms to the habits of accepted thought, and one which furnishes us, in the shape of well defined concepts, with solid points of support if it is to be accepted. This procedure of review and analysis has three primary interrelated tasks: The first task is to develop a standardized translation procedure to assure that all problems of the same class will be reviewed by the same standards; the second task requires the establishment of accepted review standards for the purpose of evaluation of the MTA's performance when it is inflicted with the translated procedures generated by the first task; and finally, the third task is the establishment of a referee with the wisdom to evaluate and bless tasks one and two.

In order, to establish the analytical confidence in the MTA, the investigation plan we are proposing, incorporates the three tasks described above. The most important task is that of the referee because it is necessary to verify the other aspects of the investigation. This is too large a task for a single individual, and a review by committee would tend to blur the critique. Therefore, we have selected an automated production facility as the vehicle for verifying the proposed methodology. The use of a production facility will permit the investigation of the analytical insight and results generated by the MTA on a full scale hardware system. Our investigation plan requires the selection of a production facility which is in the process of being debugged. After the facility has been selected, a software model of the facility will be generated from the facilities documentation package and debugged by standard software methodology.

The next step in the investigation is the review and analysis of the debugged software model with the MTA. The critical part of the investigation is then ready to be performed: The software model and its MTA analysis will be used to describe and characterize the chosen facility. This assessment will be used to identify which statistics of the facility are descriptive and their respective magnitudes. The investigation then proceeds to the floor of the production facility to see if in fact the computer simulation (software system under test) and its respective MTA analysis have any value. It should be obvious, that the design plan for the production facility is the mother of the software model and the actual production facility will be the referee that evaluates the MTA analysis of the software model.

7. CONCLUSION

We feel that existing SETs are insufficient for the Army's future munition systems. Therefore, we have analysized the SET difficiencies and hypothesized the MTA to meet the Army's needs. Experimentation with the MTA is practicable, affordable, and analytical. The MTA can provide a unique visualization of the software which is open to widespread scrutiny. The MTA can display potential system weaknesses and strengths which are typically of a synergistic nature.

It should be kept in mind that: 1) the MTA is a proposed solution to a perceived need and has not been investigated rigorously and 2) the authors did not intend to offend any of the readers by what may be deemed as a nonrigorous useage of some of the terminology contained herein. In addition, the authors would welcome communication regarding the proposed MTA as a new SET.

Selected References

Beecher, W.X.: Automated Warfare is Foreseen by Westmoreland After Vietnam; The NY Times, October 14, 1970.

Bergson, H.: An Introduction to Metaphysics; The Liberal Arts Press, Inc., N.Y., N.Y., 1955.

Dijkstra, E.W.: Programming as a Discipline of Mathematical Nature; The American Mathematical Monthly; Volume 81, number 6, 1974.

Graham, R.L.: The Combinatorial Mathematics of Scheduling; Scientific American, March 1978.

Land, G.T.L.: Grow or Die; Delta Books, Dell Publishing Co., Inc. 1974.

NY Times Editorial: Little Light from ConEd; NYC Blackout of July 13, 1977.

Picatinny Arsenal: REMIDS - Remote Mine Field Identification of Display System; D.A. Project No.: 1X663606D006, January 1976; Confidential.

Schmucker, K.J. and Tarr, R.M.: The Computers of Star Trek; Byte, December 1977.

HYPOTHETICAL FASCAM EMPLACEMENT FLOW DIAGRAM

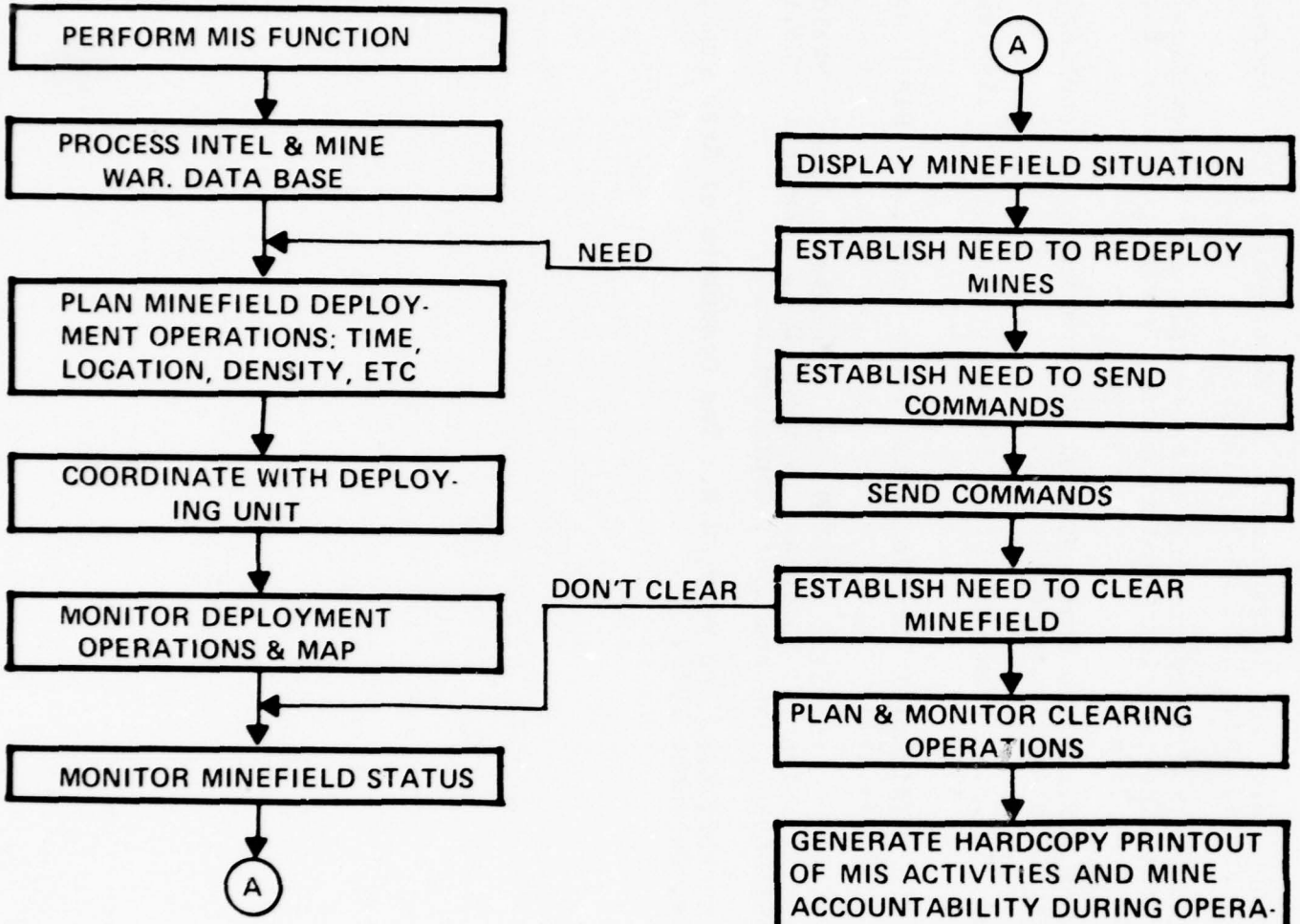


FIGURE 1

HYPOTHETICAL MIS FUNCTIONAL INTERFACES

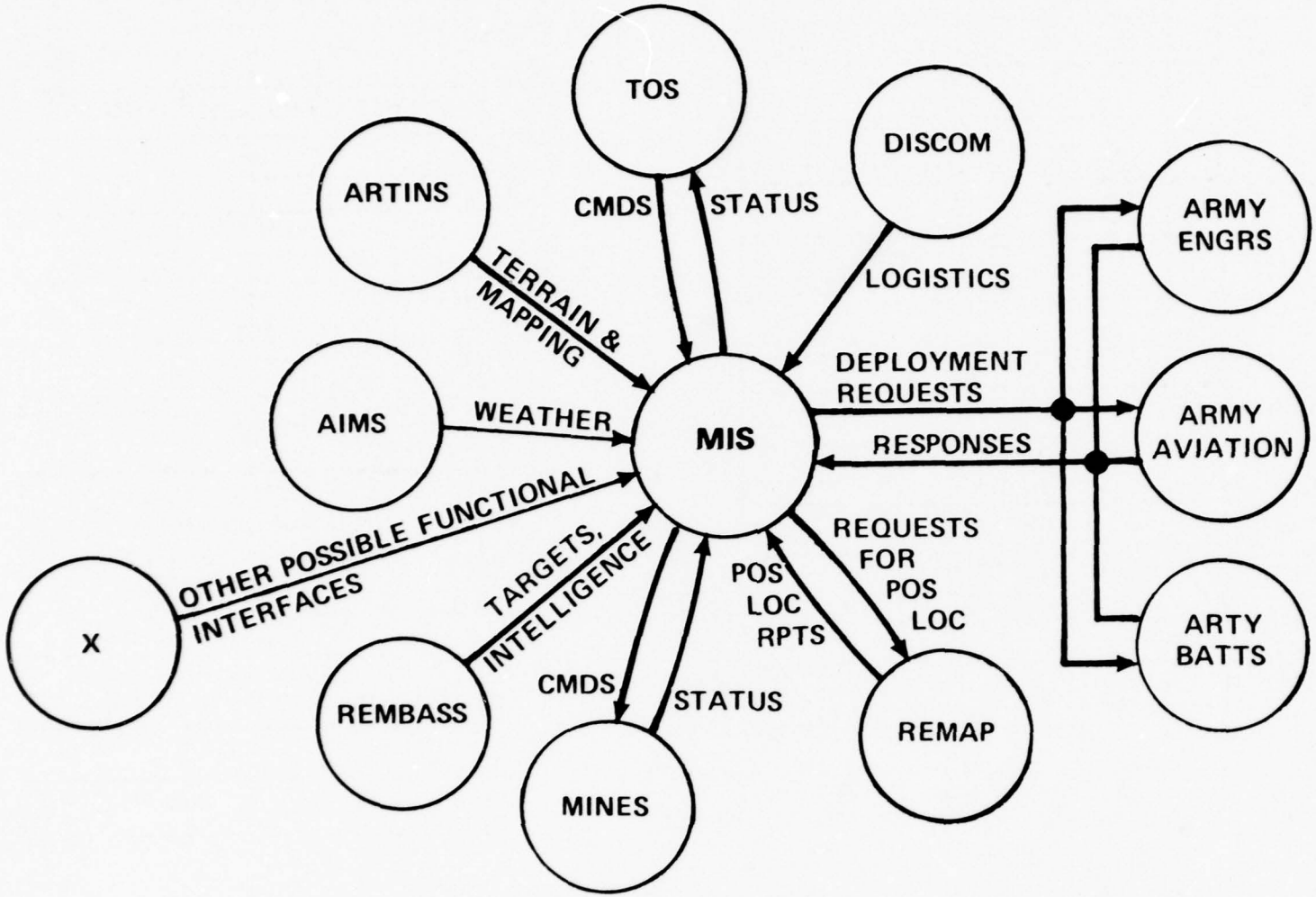


FIGURE 2

A COMPARISON OF NASTRAN CODE AND EXACT SOLUTION TO AN ELASTIC-PLASTIC DEFORMATION PROBLEM

Peter C. T. Chen
U. S. Army Armament Research and Development Command
Benet Weapons Laboratory, LCWSL
Watervliet Arsenal, Watervliet, N. Y. 12189

ABSTRACT. The plate elements of the NASTRAN code are used to solve a radially-stressed elastic-plastic annular plate problem for which an exact solution was reported recently. A comparison of two approaches together with an assessment of the NASTRAN code will be given.

1. INTRODUCTION. NASTRAN developed under NASA sponsorship is a large, general-purpose computer program for structural analysis using a finite-element displacement method approach [1]. The computer program has been operational on IBM 360 Model 44 at this Arsenal since May 1972. The piece-wise linear analysis option of this code can be used to analyze general plane stress problems in the plastic range. The reliability of this code has been well demonstrated in the linear range but not in the nonlinear range of loadings. One major reason is because exact analytical solutions for elastic-plastic problems are usually not available for comparison with approximate NASTRAN solutions.

In this paper, the plate elements of the NASTRAN code are used to solve an elastic-plastic deformation problem for which exact solutions were reported recently. The problem considered is an elastic-plastic annular plate radially stressed by uniform internal pressure. For ideally plastic materials, the stress solution for this statically determinate problem was first obtained by Mises [2] and the corresponding two strain solutions were obtained by the present author on the basis of both J_2 deformation and flow theories [3]. For elastic-plastic strain-hardening materials, an exact solution was recently reported in [4] for the partially plastic deformation and in [5] for the fully plastic deformation. Analytical expressions were derived on the basis of the J_2 deformation theory, the Hill's yield criterion and a modified Ramberg-Osgood law. The validity of the above solution has been established by satisfying the Budiansky's criterion.

In the following, the theory of elastic-plastic plate elements as used in NASTRAN is briefly reviewed. In its present form, NASTRAN can not be used for problems involving ideally plastic materials. It is shown that this limitation can be easily removed by making minor changes. For an elastic-plastic strain-hardening material, the NASTRAN solution is reported here and compared with the exact solution. The results are presented graphically and an assessment of the NASTRAN code is made.

PRECEDING PAGE BLANK

2. PLATE ELEMENTS. The theoretical basis of two dimensional plastic deformation as used in NASTRAN is that developed by Swedlow [6]. In the development, a unique relationship between the octahedral stress, τ_o, and the plastic octahedral strain, ε_o^p, is assumed to exist and the use of ideally plastic materials is excluded. The total strain components (ε_x, ε_y, ε_z, and γ_{xy}) are composed of the elastic, recoverable deformations and the plastic portions ($\varepsilon_x p$, $\varepsilon_y p$, $\varepsilon_z p$, and $\gamma_{xy} p$). The rates of plastic flow, ($\dot{\varepsilon}_x p$, etc.), are independent of a time scale and are simply used for convenience instead of incremental values. The definitions of the octahedral stress and the octahedral plastic strain rate for isotropic materials are:

$$\tau_o = \sqrt{(s_{11}^2 + 2s_{12}^2 + s_{22}^2 + s_{33}^2)/3} \quad , \tag{1}$$

$$\dot{\varepsilon}_o^p = \sqrt{[(\dot{\varepsilon}_{11}^p)^2 + 2(\dot{\varepsilon}_{12}^p)^2 + (\dot{\varepsilon}_{22}^p)^2 + (\dot{\varepsilon}_{33}^p)^2]/3} \quad , \tag{2}$$

where

$$\begin{aligned} s_{11} &= \frac{1}{3}(2\sigma_x - \sigma_y) \quad , & \varepsilon_{11}^p &= \varepsilon_x^p \quad , \\ s_{22} &= \frac{1}{3}(2\sigma_y - \sigma_x) \quad , & \varepsilon_{22}^p &= \varepsilon_y^p \quad , \\ s_{33} &= -\frac{1}{3}(\sigma_x + \sigma_y) \quad , & \varepsilon_{33}^p &= \varepsilon_z^p \quad , \\ s_{12} &= T_{xy} \quad , & \varepsilon_{12}^p &= \frac{1}{2}\gamma_{xy}^p \quad . \end{aligned} \tag{3}$$

The s_{ij} is called the deviator of the stress tensor; σ_x, σ_y and T_{xy} are the cartesian stresses. The isotropic material is assumed to obey the Mises yield criterion and the Prandtl-Reuss flow rule. The matrix relationship for the plastic flow is

$$\begin{Bmatrix} \dot{\varepsilon}_x^p \\ \varepsilon_y^p \\ \gamma_{xy}^p \end{Bmatrix} = \frac{1}{6\tau_o^2 M_T(\tau_o)} \begin{bmatrix} s_{11}^2 & s_{11}s_{22} & 2s_{11}s_{12} \\ s_{11}s_{22} & s_{22}^2 & 2s_{22}s_{12} \\ 2s_{11}s_{22} & 2s_{22}s_{12} & 4s_{12}^2 \end{bmatrix} \begin{Bmatrix} \dot{\sigma}_x \\ \dot{\sigma}_y \\ \dot{T}_{xy} \end{Bmatrix} , \tag{4}$$

where

$$2M_T(\tau_o) = \dot{\tau}_o / \dot{\varepsilon}_o^p \quad . \tag{5}$$

The plastic modulus, $M_T(\tau_o)$, can be related to the slope, E_T, of the effective stress-strain curve by

$$\frac{1}{3M_T(\tau_o)} = \frac{1}{E_T} - \frac{1}{E} \quad . \tag{6}$$

The total strain increments, obtained by adding the plastic and linear elastic parts, are:

$$\{\Delta\varepsilon\} = ([D^p] + [G]^{-1})\{\Delta\sigma\} = [G_p]^{-1}\{\Delta\sigma\} \quad , \tag{7}$$

where [G] is the normal elastic material matrix and $[G_p]$ is the equivalent plastic material matrix. The matrices $[D^p]$ and $[G_p]^{-1}$ exist for finite values of M_T (or E_T) and $[G_p]$ can be obtained numerically. Because this procedure is chosen in developing NASTRAN program, only strain hardening materials can be considered for applications. However, it should be noted that even the matrix $[G_p]^{-1}$ does not exist when M_T or E_T is equal to zero, the matrix $[G_p]$ may still exist. In fact, the closed form of $[G_p]$ has been given in [7]. We can express as

$$[G_p] = \frac{E}{Q}\begin{bmatrix} s_{22}^2+2A & & \text{SYM.} \\ -s_{11}\,s_{22}+2\nu A & s_{11}^2+2A & \\ -\frac{s_{11}+\nu s_{22}}{1+\nu}\,s_{12} & -\frac{s_{22}+\nu s_{11}}{1+\nu}\,s_{12} & \frac{B}{2(1+\nu)} \quad \frac{(1-\nu)E_T}{E-E_T}\,\tau_o^2 \end{bmatrix} \tag{8}$$

where

$$A = \frac{E_T}{E-E_T}\,\tau_o^2 + \frac{s_{12}^2}{1+\nu} \quad , \quad B = s_{11}^2 + 2\nu s_{11}\,s_{22} + s_{22}^2 \quad ,$$

$$Q = 2(1-\nu^2)A + B \quad . \tag{9}$$

If we want to remove the limitation that the use of ideally plastic materials is excluded, we have to make minor changes in subroutines PSTRM and PKTRM of the NASTRAN program.

3. PROBLEM DEFINITION. The finite element model, shown in Figure 1, utilizes the condition of axial symmetry so only a sector of the annular plate is modeled. All membrane elements use stress dependent materials. The effective stress-strain curve is defined by

$$\sigma = E\varepsilon \quad \text{for} \quad \sigma \le \sigma_o \quad ,$$

$$(\sigma/\sigma_o)^n = (E/\sigma_o)\varepsilon \quad \text{for} \quad \sigma \ge \sigma_o \quad , \tag{10}$$

and a lack of understanding of the scoring criteria applicable to the software system.

where

$$\sigma = (3/\sqrt{2})\tau_o \quad , \quad \varepsilon = \sigma/E + \sqrt{2}\,\varepsilon_p \quad , \tag{11}$$

n is the strain hardening parameter and the initial yield surface is defined by the ellipse $\sigma = \sigma_o$. The input parameters for the problem are

a = 1.0 inch (inner radius) , b = 2.0 inch (outer radius) ,

θ_o = 90/17 degree (sector) , t = 0.1 inch (thickness) ,

$E = 10.5 \times 10^6$ psi , ν = 0.3 (Poisson's ratio) ,

$\sigma_o = 5.5 \times 10^4$ psi , n = 9 .

All grid points are constrained in the tangential direction. The applied load is the internal pressure p. If p = 1000 psi, the equivalent nodal force at each of the two interior grids is Q = apt tan(θ/2) = 4.62329 pound in the radial direction. The true pressure corresponding to initial yielding for this problem is p_o = 23,571.35 psi. Four sets of load factors are used. The load increments for three of them are uniform with $\Delta p/p_o$ = 0.20, 0.10, 0.05, respectively. The load factors for the fourth set are obtained from the analytical solution and each of the first nine load factors is supposed to cause one more element to become plastic. A complete input deck acceptable by the NASTRAN program is shown in Table 1. This input deck defines a NASTRAN problem.

4. RESULTS AND DISCUSSIONS. The outputs of the NASTRAN program are the displacements and stresses for each load factor. The information for the strains is not available. The stress results are σ_x,σ_y and T_{xy} at the centroid of each element. The stress components in polar coordinates can be calculated by

$$\sigma_r = \frac{1}{2}(\sigma_x+\sigma_y) + \frac{1}{2}(\sigma_x-\sigma_y)\cos 2\theta + T_{xy}\sin 2\theta \quad ,$$

$$\sigma_o = \frac{1}{2}(\sigma_x+\sigma_y) - \frac{1}{2}(\sigma_x-\sigma_y)\cos 2\theta - T_{xy}\sin 2\theta \quad ,$$

$$\sigma_{ro} = -\frac{1}{2}(\sigma_x-\sigma_y)\sin 2\theta + T_{xy}\cos 2\theta \tag{12}$$

The NASTRAN results will depend on the user's choices of element sizes and load increments. In general, a user can get better results at a higher cost by using more elements and smaller load increments. But there are still other sources of built-in errors in the program.

In the elastic range, the NASTRAN results based on two finite-element models were compared with the exact solution. For a ten-element model, the maximum error is 0.36% in displacements and 0.95% in stresses. For a twenty-element model, the maximum errors are reduced to 0.09% and 0.24%, respectively. Since we are satisfied with 1% error, the ten-element model as shown in Figure 1 is chosen.

In the plastic range, the user has to choose the load increments properly in order to obtain good results at reasonable cost. The values of the load factors can be normalized so its unit value corresponds to the limit of elastic solution, i.e., p_o = 23,571 psi. The load increments can be uniform or nonuniform. It seems that the size of load increments depend on the material properties and sizes of elements. In order to determine the influence of load factors on the displacements and stresses in the plastic range, four sets of load factors are chosen. The load increments for three of them are uniform with $\Delta p/p_o$ = 0.20, 0.10, 0.05, respectively. The influence of load factor, p/p_o, on the inside radial displacement, u_1, is shown in Figure 2. The effect of load factors on the major principal stresses in element 1 is shown in Figure 3. We also show in these two figures the corresponding analytical results. On the basis of these comparisons, we can make the following conclusions. In the earlier stages of plastic deformation, larger load increments can be used to give very good results. As plastic deformation becomes bigger, smaller load increments should be used in order to get reasonably good answer. However, for very large plastic deformation, it seems that we cannot improve the NASTRAN results much better by chosing even smaller increments. This is because there are other built-in errors in the NASTRAN program, e.g., the linear displacement function is assumed.

If uniform load increments are used, a direct comparison of the analytical and NASTRAN results is not available. The solid curves shown in Figures 2 and 3 were obtained indirectly since the analytical results of the displacements, stresses and pressure were represented as functions of elastic-plastic boundary [4,5]. We have obtained the analytical results when the elastic-plastic boundary is located at a radial distance of 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95 inch from the inside surface. The corresponding values of the pressure factor (p/p_o) are 1.094776, 1.265848, 1.413734, 1.540074, 1.646427, 1.734275, 1.805022, 1.860089, 1.900829, 1.928619, respectively. This set of values is used as load factors in the input deck of the NASTRAN program. Some of the NASTRAN results together with the corresponding analytical results are shown in Figures 4 to 7. A direct comparison of the two approaches in the plastic range can be seen. The effect of load factors on the distribution of radial displacements is shown in Figure 4. The stress results of the NASTRAN program are σ_x, σ_y, T_{xy}, two principal stresses, principal stress angle and maximum shear at the centroids of all elements. The distributions of major and minor principal stresses for three load factors are shown in Figures 5 and 6, respectively. The effect of the load factors on the errors in principal stress angles is shown in Figure 7. If there is no error in the principal stress angle, the major and minor principal stresses will be the tangential and radial stresses, respectively. As can be seen in Figures 4 to 7, a direct comparison of two approaches will support the following conclusion: Even if the results in the elastic range are in excellent agreement, the differences in the plastic range can be quite big for large values of load factors. This suggests more research efforts should be given to large plastic deformation. It also remains a question how much improvement over the present model in the plastic range by using more elements. An alternative approach is to use higher order isoparametric elements.

In this paper, emphasis has been given on the limitations of the NASTRAN program so we can make some suggestions for improvements and additions of new capability. However, this code in its present form is still a valuable tool because it can be used to solve quite complicated plane stress problems provided the plastic deformation involved is not too large. Furthermore, the NASTRAN program is widely used and well documented so we can expect more improvements and additions in the future either by us or others.

REFERENCES

1. "NASTRAN Theoretical Manual," NASA SP-221 (01), 1972.
"NASTRAN User's Manual," NASA SP-222 (01), 1972.
"NASTRAN Programmer's Manual," NASA SP-223 (01), 1972.
"NASTRAN Demonstration Problem Manual," NASA SP-224 (01), 1972.

2. R. V. Mises, "Three Remarks on the Theory of Ideal Plastic Body," Reissner Anniversary Volume, pp. 415-429, 1949.

3. P. C. T. Chen, "A Comparison of Flow and Deformation Theories in a Radially Stressed Annular Plate," Journal of Applied Mechanics, Vol. 40, No. 1, Trans. ASME, Vol. 95, pp. 283-287, 1973.

4. P. C. T. Chen, "An Exact Solution to an Elastic-Plastic Deformation Problem in a Radially-Stressed Annular Plate," Trans. of the Twenty-Second Army Mathematicians, ARO Report 77-1, pp. 227-238, 1977.

5. P. C. T. Chen, "Fully Plastic Deformation in Anisotropic Annular Plate Under Internal Pressure," Trans. of the Twenty-Third Army Mathematicians, ARO Report 78-1, to appear in 1978.

6. J. L. Swedlow, "The Thickness Effect and Plastic Flow in Cracked Plates," ARL 65-216, Aerospace Research Laboratories, Office of Aerospace Research, October 1965.

7. Y. Yamada, N. Yoshimura, and T. Sakurai, "Plastic Stress-Strain Matrix and its Application for the Solution of Elastic-Plastic Problems by the Finite Element Method," Int. J. Mech. Sci., Vol. 10, pp. 345-354, 1968.

analytical approach is the only viable method of evaluating the inherent workability (correctness) of a system. We also believe that mathematical proofs of correctness are not well enough developed to have universally accepted standards. Therefore, this new analytical approach comprised of a standarized hardware analog is proposed.

TABLE 1. LIST OF INPUT DECK FOR A RADIALLY STRESSED ANNULAR PLATE

```
ID ANNULAR PLATE UNDER UNIFORM INTERNAL PRESSURE
APP DISPLACEMENT
SOL 6,0
TIME 30
CEND
TITLE = ELASTIC-PLASTIC ANNULAR PLATE
 SUBT = SECTOR OF 90/17 DEGREE WITH GEOMETRIC RATIO 2.
LABEL = RAMBERG-OSGOOD LAW
 LOAD #200
 SPC# 300
 MPC # 350
PLCOEFF # 400
 STRESS#ALL
 DISP # ALL
 SPCF #ALL
BEGINBULK
CORD2C  1000            0.000   0.0     0.0     0.000   0.0     1.000   EC2C
EC2C    1.000   0.0     1.000
CQDMEM          1       111         1       3       4       2
CQDMEM          2       111         3       5       6       4
CQDMEM          3       111         5       7       8       6
CQDMEM          4       111         7       9      10       8
CQDMEM          5       111         9      11      12      10
CQDMEM          6       111        11      13      14      12
CQDMEM          7       111        13      15      16      14
CQDMEM          8       111        15      17      18      16
CQDMEM          9       111        17      19      20      18
CQDMEM         10       111        19      21      22      20
FORCE         201         11000     4.62329 1.
FORCE         202         21000     4.62329 1.
GRDSET                                                  3456
GRID    1       1000    1.0     0.
GRID    2       1000    1.0     5.294118
GRID    3       1000    1.1     0.
GRID    4       1000    1.1     5.294118
GRID    5       1000    1.2     0.
GRID    6       1000    1.2     5.294118
GRID    7       1000    1.3     0.
GRID    8       1000    1.3     5.294118
GRID    9       1000    1.4     0.
GRID    10      1000    1.4     5.294118
GRID    11      1000    1.5     0.
GRID    12      1000    1.5     5.294118
GRID    13      1000    1.6     0.
GRID    14      1000    1.6     5.294118
```

TABLE 1. LIST OF INPUT DECK FOR A RADIALLY STRESSED ANNULAR PLATE

(continued)

```
GRID     15       1000     1.7      0.
GRID     16       1000     1.7      5.294118
GRID     17       1000     1.8      0.
GRID     18       1000     1.8      5.294118
GRID     19       1000     1.9      0.
GRID     20       1000     1.9      5.294118
GRID     21       1000     2.0      0.
GRID     22       1000     2.0      5.294118
LOAD     200      23.57135 1.           201  1.          202
MAT1           5 1.05E7          .3
MATS1          5      500
MPC            351        2        2.995734         2        1-.092268
MPC            352        4        2.995734         4        1-.092268
MPC            353        6        2.995734         6        1-.092268
MPC            354        8        2.995734         8        1-.092268
MPC            355       10        2.995734        10        1-.092268
MPC            356       12        2.995734        12        1-.092268
MPC            357       14        2.995734        14        1-.092268
MPC            358       16        2.995734        16        1-.092268
MPC            359       18        2.995734        18        1-.092268
MPC            360       20        2.995734        20        1-.092268
MPC            361       22        2.995734        22        1-.092268
MPCADD         350      351      352      353      354      355      356      357&MPC1
&MPC1    358      359      360      361
PLFACT         4001.0947761.2658481.4137341.5400741.6464271.7342751.805022&PL1
&PL1     1.86C0891.9C08291.95     2.        2.05    2.1
PODMEM         111        5  .1
SPC1           300   2            1        3        5        7        9       11&PC1
&PC1     13       15       17       19       21
TABLES1  500                                                                  &TAB 0
&TAB 0   .0              0.0.0052381 55000.0.0056826 55500.0.0061603 56000.0&TAB 1
&TAB 1   .0066733 56500.0.0072241 57000.0.0078148 57500.0.0084481 58000.0&TAB 2
&TAB 2   .0091266 58500.0.0098532 59000.0.0106306 59500.0.0114623 60000.0&TAB 3
&TAB 3   .0123511 60500.0.0133008 61000.0.0143147 61500.0.0153969 62000.0&TAB 4
&TAB 4   .0165511 62500.0.0177817 63000.0.0190928 63500.0.0204893 64000.0&TAB 5
&TAB 5   .0219758 64500.0.0235575 65000.0.0252393 65500.0.0270274 66000.0&TAB 6
&TAB 6   .0289289 66500.0.0309443 67000.0.0330857 67500.0.0353581 68000.0&TAB 7
&TAB 7   .0377678 68500.0.0403227 69000.0.0430298 69500.0.0458976 70000.0&TAB 8
&TAB 8   .0489338 70500.0.0521474 71000.0.0555471 71500.0.0591426 72000.0&TAB 9
&TAB 9   .0629432 72500.0.0669597 73000.0.0712020 73500.0.0756821 74000.0&TAB10
&TAB10   .0804105 74500. .0854002 75000. .0962122 76000. .1082241 77000. &TAB11
&TAB11   ENDT
ENDDATA
```

$E = 10.5 \times 10^6$ psi , $\nu = .3$, $\underline{(\sigma/\sigma_0)^n = (E/\sigma_0)\epsilon}$

$\sigma_0 = 5.5 \times 10^4$ psi , $n = 9$.

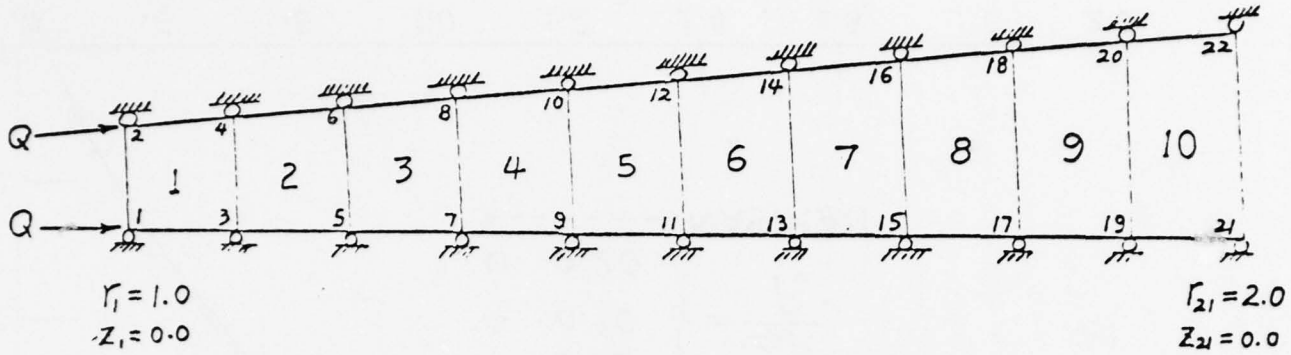


FIGURE 1. TEN-ELEMENT ANNULAR PLATE UNDER UNIFORM PRESSURE

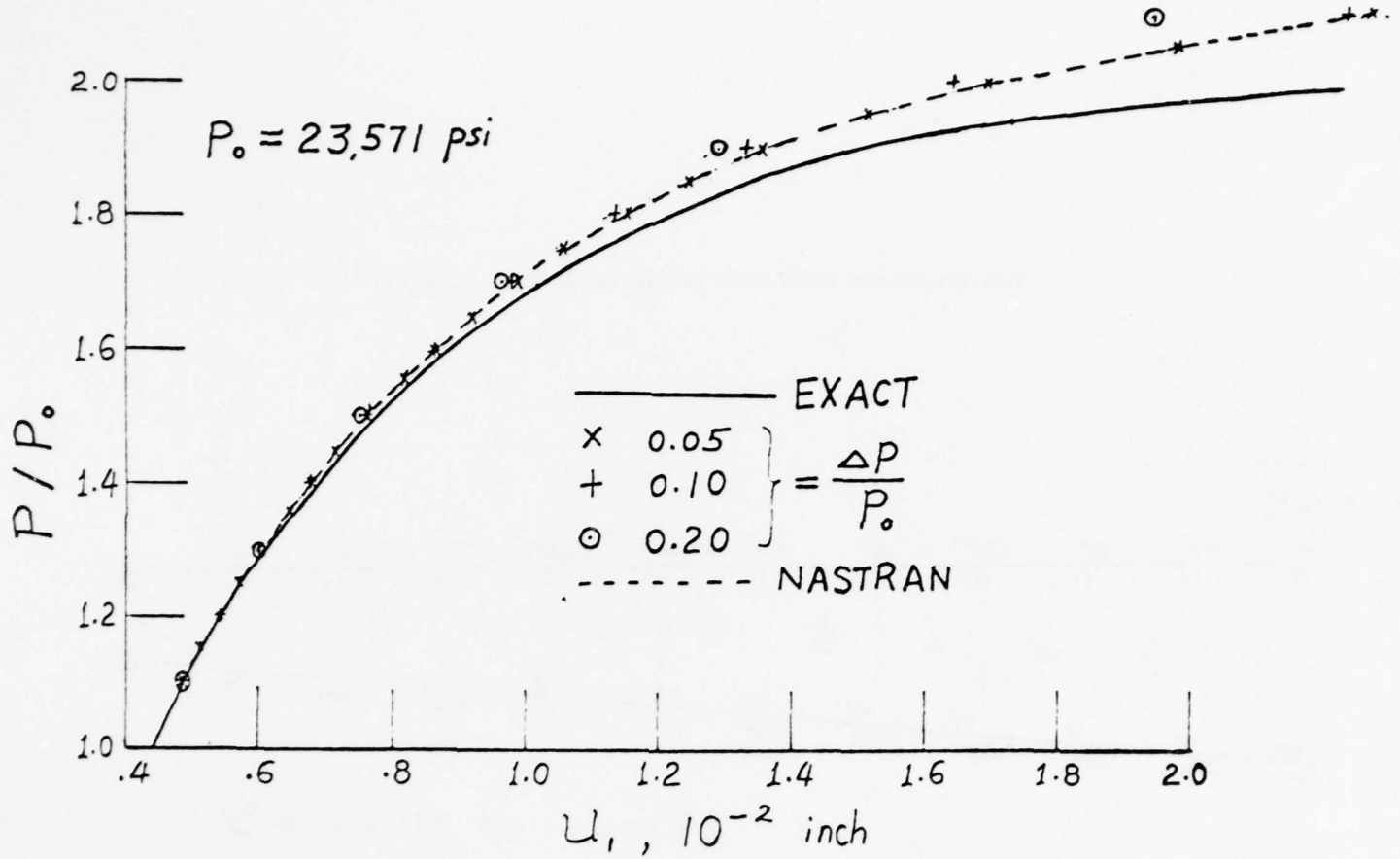


FIGURE 2. INFLUENCE OF LOAD FACTOR ON THE INSIDE DISPLACEMENTS

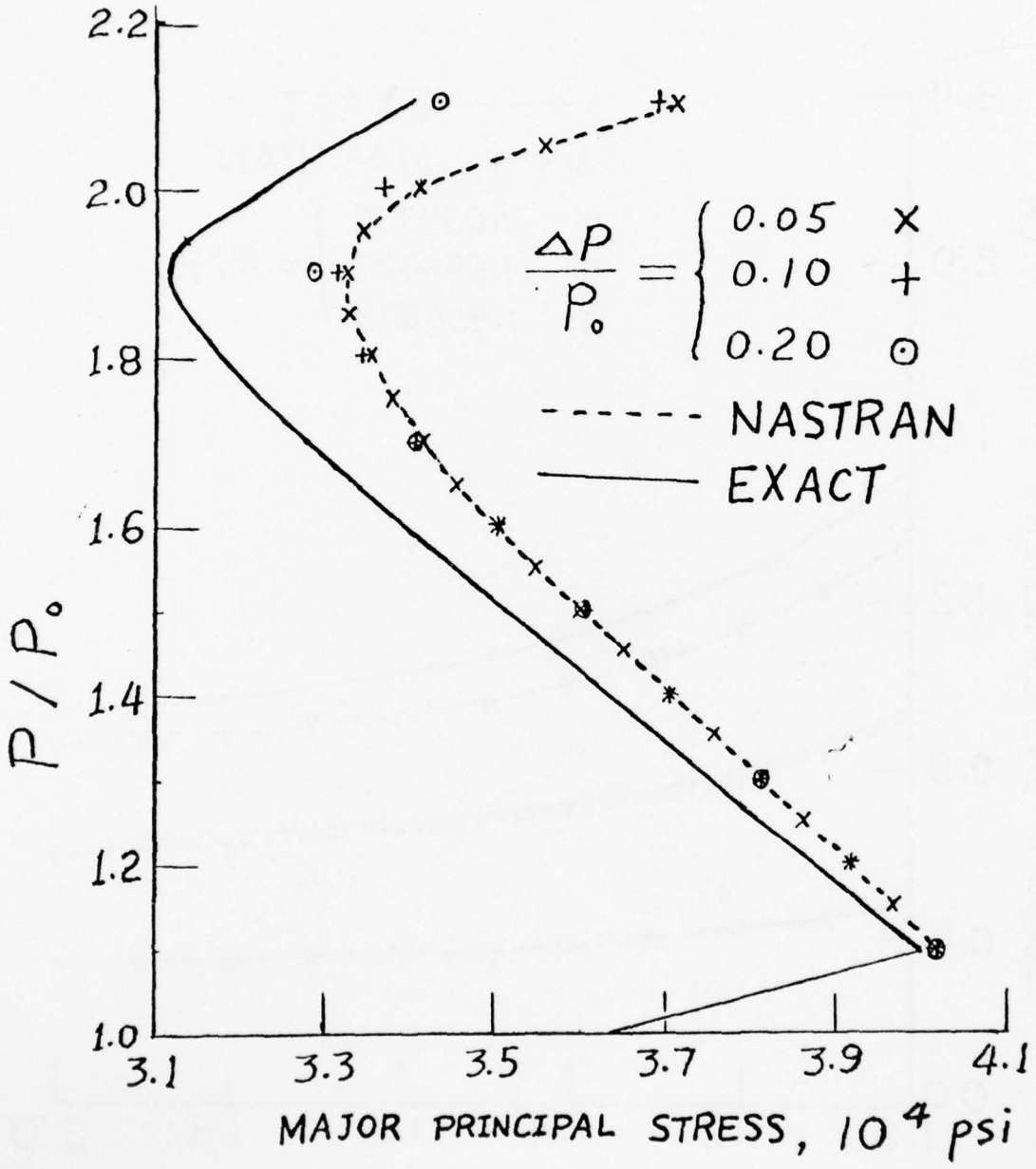


FIGURE 3. INFLUENCE OF LOAD FACTOR ON THE MAJOR PRINCIPAL STRESSES IN ELEMENT 1

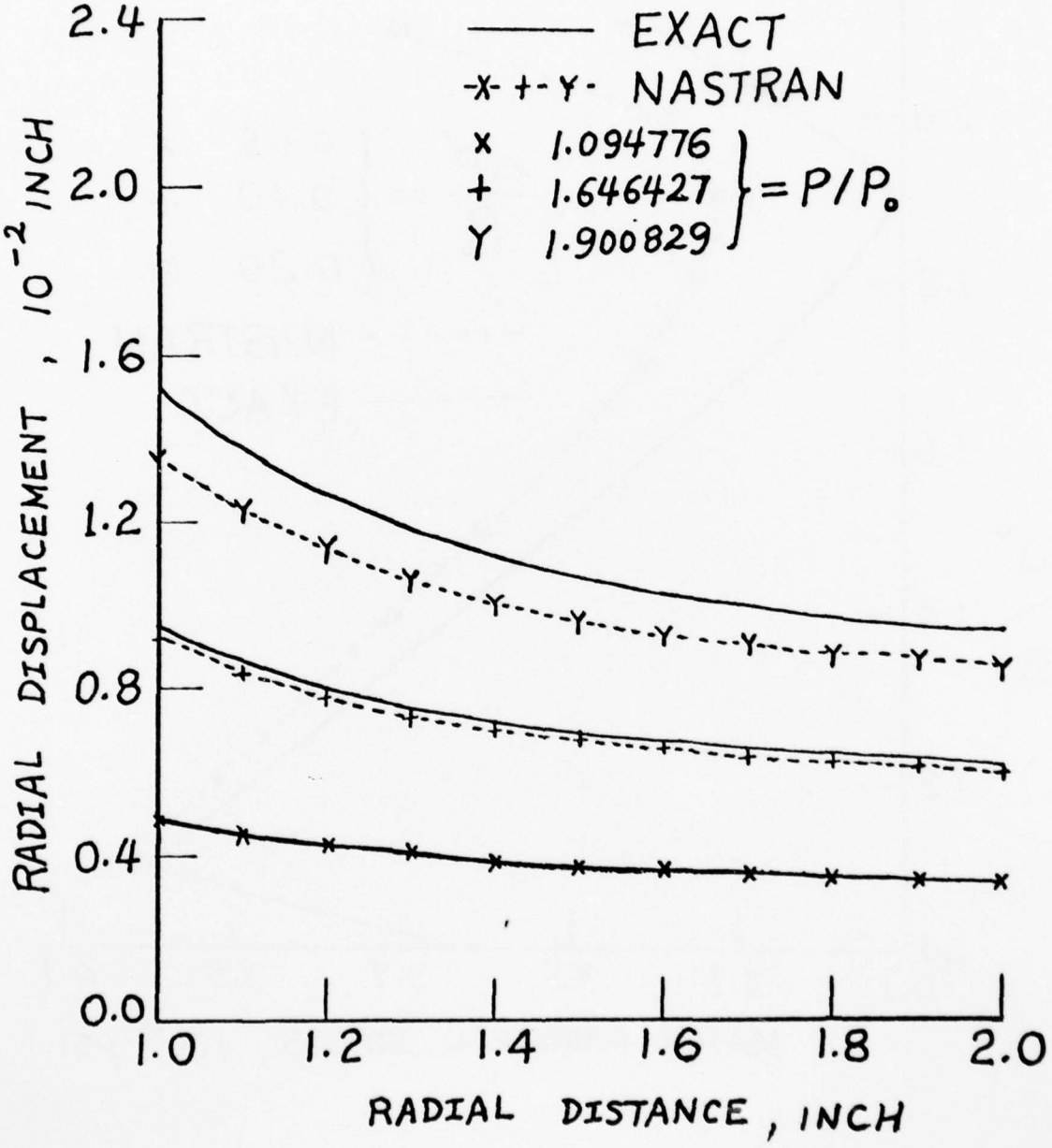


FIGURE 4. DISTRIBUTION OF RADIAL DISPLACEMENTS

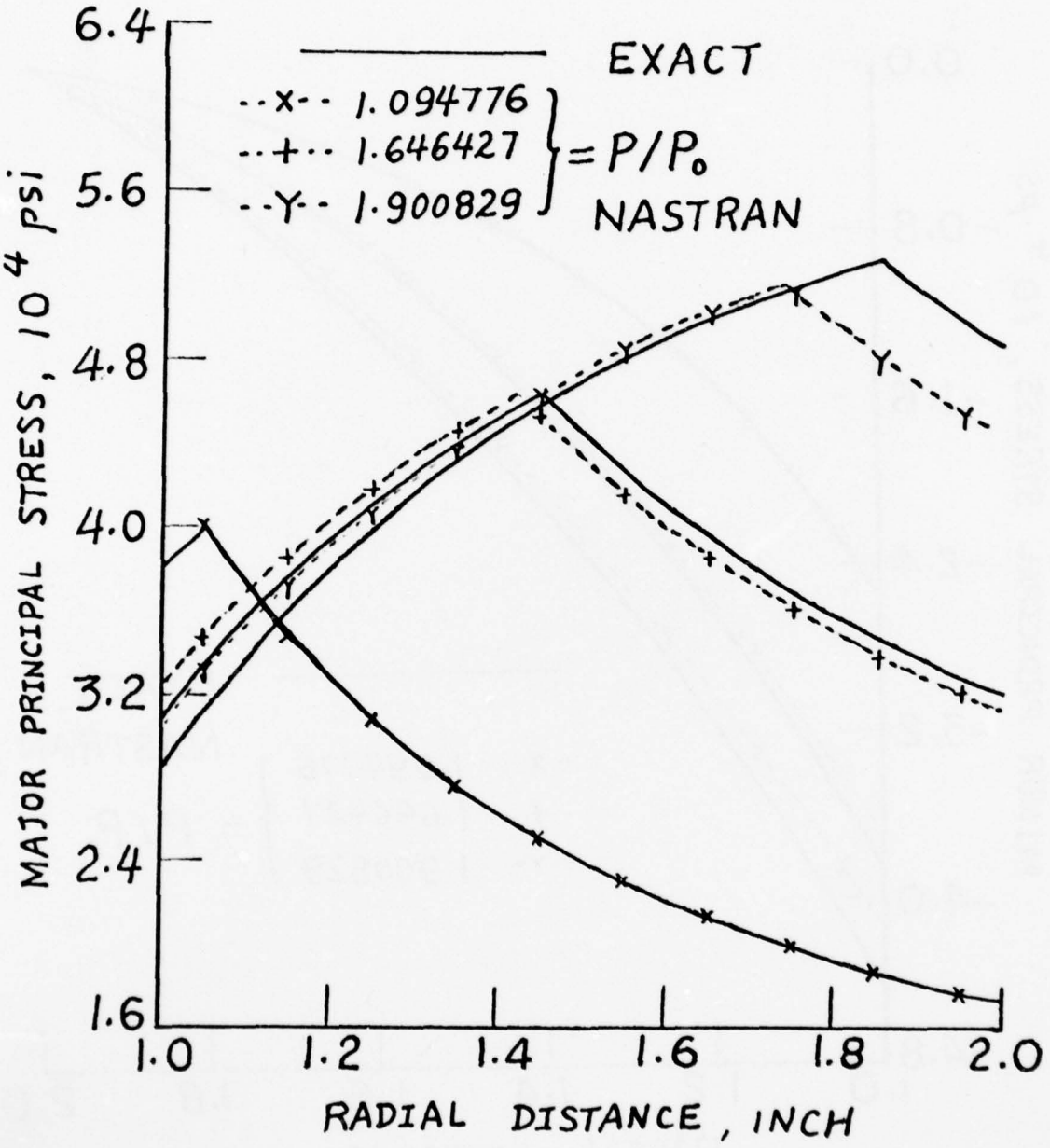


FIGURE 5. DISTRIBUTION OF MAJOR PRINCIPAL STRESSES

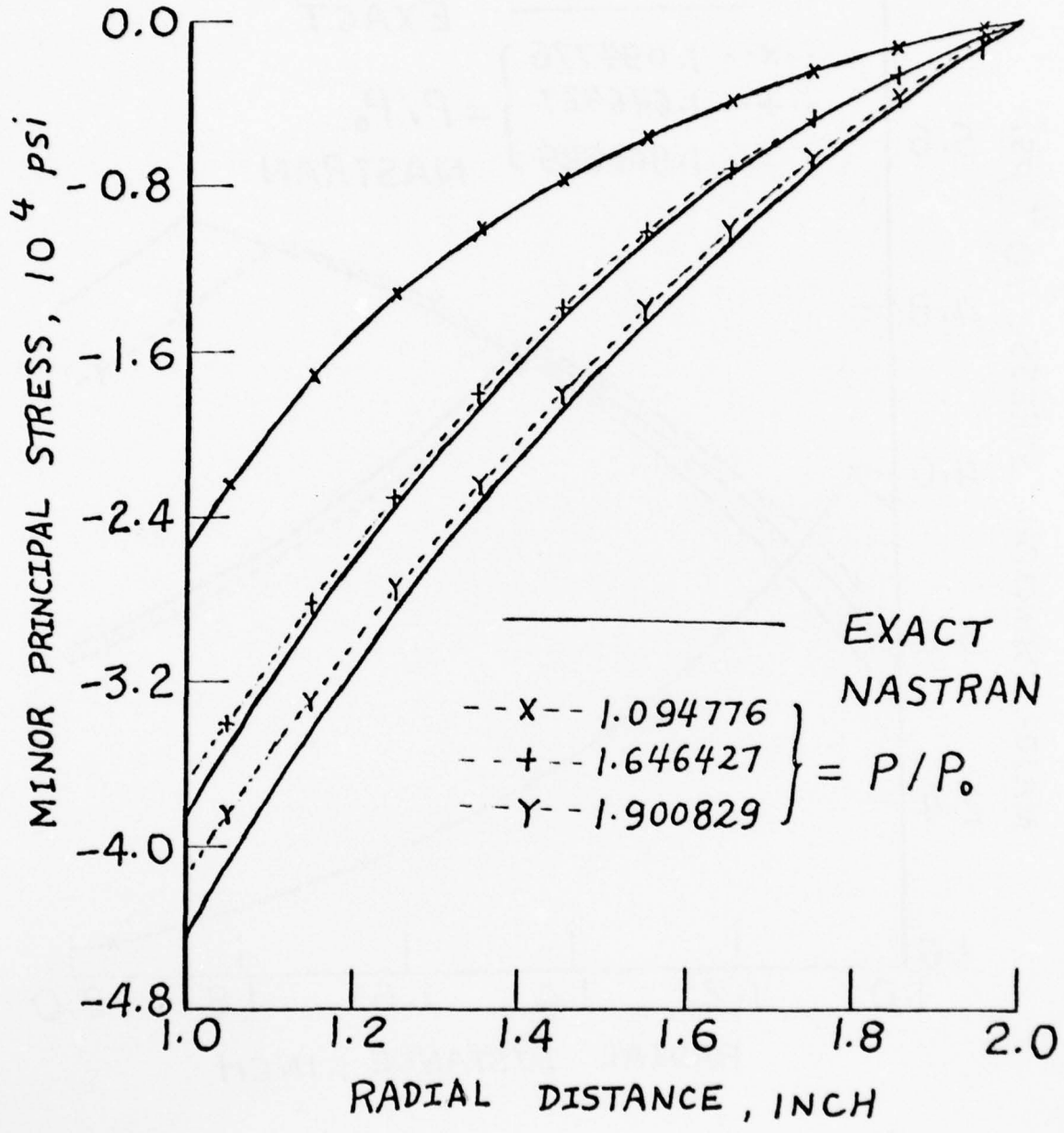


FIGURE 6. DISTRIBUTION OF MINOR PRINCIPAL STRESSES

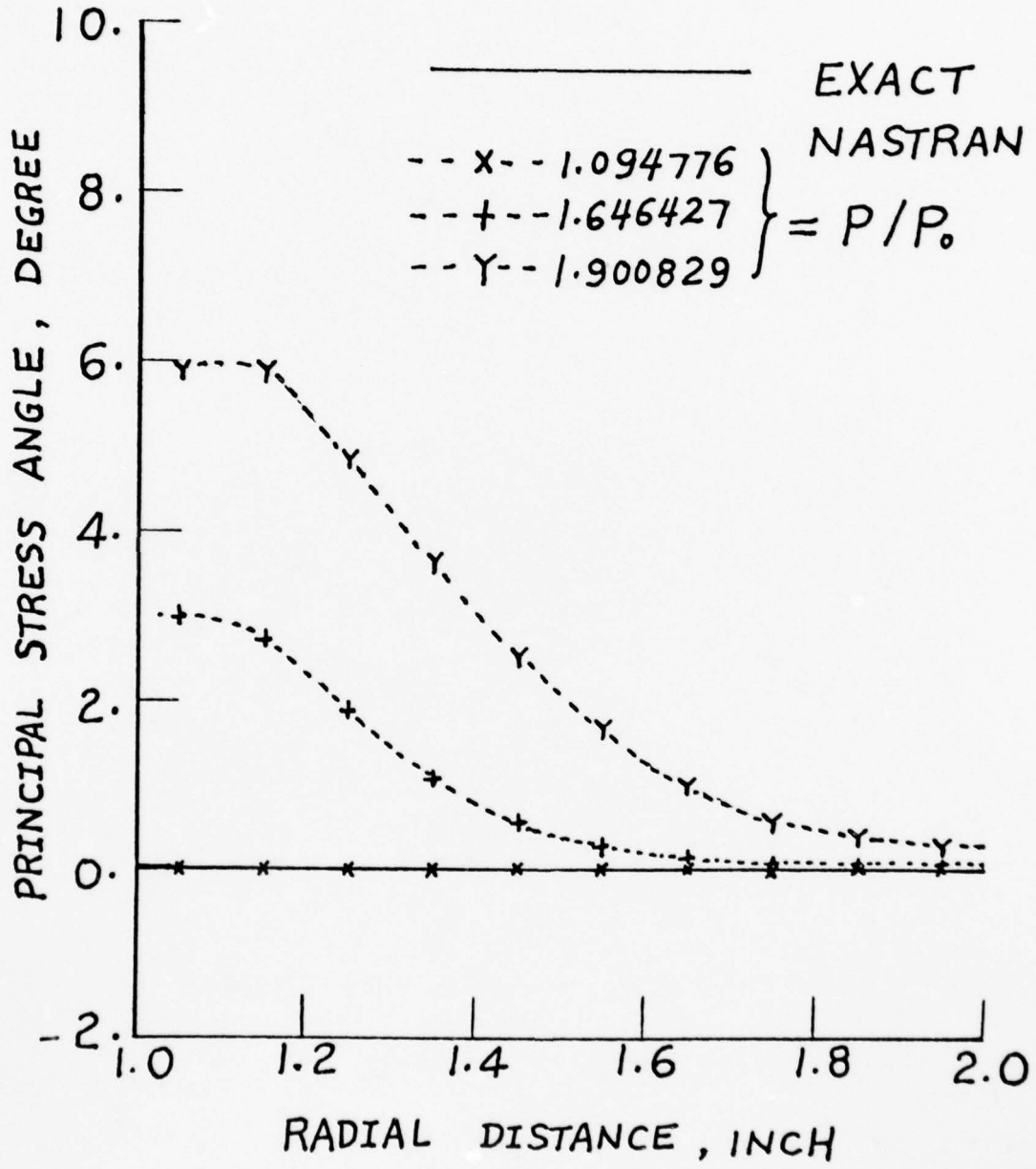


FIGURE 7. ERRORS IN PRINCIPAL STRESS ANGLES

ON BLOCK RELAXATION TECHNIQUES

D. Boley
Department of Computer Science
Stanford University
Palo Alto, California 94305

B. L. Buzbee
Los Alamos Scientific Laboratory
Los Alamos, New Mexico 87544

and

S. V. Parter
Department of Mathematics and
Mathematics Research Center
University of Wisconsin-Madison
Madison, Wisconsin 53706

ABSTRACT. In connection with efforts to utilize the CRAY-1 computer efficiently, we present some methods of analysis of rates of convergence for block iterative methods applied to the model problem. One of the more interesting methods involves relaxing an $p \times p$ blocks of points. A Cholesky decomposition is used for that smaller problem. One of the basic methods of analysis is a modification of a method discussed earlier by Parter. This analysis easily extends to more general second order elliptic problems.

1. INTRODUCTION. Some 15-20 years ago there was a great interest in iterative methods for elliptic difference equations - see [13], [14], [15], [7], [9], [10]. More recently there has been a greater emphasis on direct methods for these sparse matrices - see [5], [6], [11], [12].

However, with the advent of "vector machines" and "parallel processors" we have found it necessary to return to a consideration of certain iterative methods.

The CRAY-1 computer can perform up to 250 million floating point operations per second [2]. Algorithms that execute with high arithmetic efficiency on this computer must "fit the architecture" of it and be carefully programmed in assembly language. Thus in using this computer, we seek computational modules that can be implemented efficiently and that can be used in solving diverse problems. The solution of banded positive definite linear systems is such a module, and the Cholesky decomposition algorithm for it can be implemented on the CRAY-1 such that its execution proceeds at the rate of about 100 million floating point operations per second. Since the vector registers of the CRAY-1 can hold at most 64 numbers, implementation of banded Cholesky is simplified if vector lengths do not exceed 64. Block Relaxation techniques for solving elliptic difference approximations require the solution of banded positive definite linear systems. These facts led us to investigate the convergence rate of block successive overrelaxation for the model problem using $p \times p$ blocks (preferably $p \leq 64$).

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024; by the Los Alamos Scientific Laboratory under Contract No. W-7405-ENG-36; and by the Office of Naval Research under Contract No. N00014-76-C-0341.

PRECEDING PAGE BLANK

In [9] one of us developed a fairly general theory for obtaining such estimates on the rates of convergence of iterative methods for elliptic difference equations. However, partly because of the generality of that work (variable coefficients, general domains, etc.) it is by no means a transparent discussion. On the other hand, in the case of the model problem it is relatively easy to develop this general approach. This is partly due to the strong estimates of [1] and [8].

In section 2 we describe the model problem and iterative methods for its solution. In section 3 we develop the general theory (for this special case). In section 4 we obtain the rates of convergence estimates for the $p \times p$ block method mentioned earlier. In section 5 we apply the theory to the multi-line methods (these methods have been studied earlier [9], [10]).

Finally, because it is worthwhile for the practical worker to have available many methods for getting information on rates of convergence (some work here - others there), in section 6 we return to multi-line methods. Using a completely different technique we obtain upper bounds (unfortunately: not sharp bounds) on the rates of convergence.

2. THE MODEL PROBLEM. Let

$$\Omega \equiv \{(x,y);\ 0 < x,y < 1\} . \tag{2.1}$$

Let P be a fixed integer and set

$$h = \frac{1}{P+1} .$$

Consider the set of interior mesh points

$$\Omega(h) = \{(x_k,y_j) = (kh,jh)\}, \quad 1 \le k,j \le P \tag{2.2}$$

as well as the boundary mesh points

$$\partial\Omega(h) \equiv \{(x_k,y_j);\ (k = 0 \text{ or } P+1) \text{ or } (j = 0 \text{ or } P+1)\} . \tag{2.3}$$

Let $U = \{u_{kj}\}$ be a vector defined on the set of all grid points: $\Omega(h) \cup \partial\Omega(h)$ that is, u_{kj} is the value of U at (x_k,y_j). We call U a "grid vector". In differing circumstances we will choose different orderings of the components of U.

As usual, we define the discrete Laplace operator by

$$(\Delta_h U)_{kj} = \frac{u_{k+1,j} - 2u_{kj} + u_{k-1,j}}{h^2} + \frac{u_{k,j+1} - 2u_{kj} + u_{k,j-1}}{h^2}, \quad 1 \le k,j \le P . \tag{2.4a}$$

Note: While U is defined on the entire mesh region, $\Delta_h U$ is defined only on $\Omega(h)$, the interior. Also, we define the difference operators

2.4b) $$[\nabla_x U]_{k,j} = \frac{u_{k,j} - u_{k-1,j}}{h}, \qquad 1 \le j \le P, \quad 1 \le k \le P+1,$$

2.4c) $$[\nabla_y U]_{k,j} = \frac{u_{k,j} - u_{k,j-1}}{h}, \qquad 1 \le j \le P+1, \quad 1 \le k \le P.$$

The basic problem is: Given grid vectors F and G, find a grid vector U such that

2.5a) $$\Delta_h U = F, \quad \text{in} \quad \Omega(h),$$

2.5b) $$U = G, \quad \text{on} \quad \partial\Omega(h).$$

After an ordering of the points (x_k, y_j) is determined we let A be the matrix representation of $-h^2\Delta_h$; symbolically, we write

2.6) $$A \sim -h^2\Delta_h .$$

As we have already remarked Δ_h maps vectors with $P^2 + 4P$ components into vectors with P^2 components. The matrix A actually is a square P^2 by P^2 matrix. The known boundary values, G, are put on the right-hand-side. In this way the difference equations (2.5a), (2.5b) take the form

2.7) $$AV = \tilde{F}$$

where the $\tilde{}$ over F is meant to indicate both the result of ordering the components of $-h^2F$ and the necessary modifications of F required by the G terms. In any case, <u>every vector V with P^2 components may be thought of as a grid vector which also satisfies</u>

2.8) $$V = 0 \quad \text{on} \quad \partial\Omega(h).$$

An iterative method for the solution of (2.7) is determined by a "splitting"

2.9a) $$A = M - N .$$

Equation (2.7) is then

2.9b) $$MV = NV + \tilde{F} .$$

After choosing a first guess V^0, one obtains $V^1, V^2, \ldots, V^k, \ldots$ from

2.10) $$MV^{k+1} = NV^k + \tilde{F} .$$

Let

2.11) $$\rho = \max\{|\lambda|;\ \det(\lambda M - N) = 0\} .$$

It is well known that the iterates V^k converge to the unique solution V of (2.7) if and only if (independently of V^0)

2.12) $$\rho < 1 .$$

The problem studied in this report is: find the asymptotic behaviour of ρ as $h \to 0$.

Remark: Of course, for every λ which is a generalized eigenvalue, (i.e. $\det(\lambda M - N) = 0$) there is a vector $U \neq 0$ such that

2.13) $$\lambda MU = NU \ .$$

3. A GENERAL APPROACH. We make some assumptions about the splitting (2.9a).

A.1) $$M = M^* \quad \text{and is positive definite}$$

A.2) $$\rho = \max_{x \neq 0} \frac{\langle Nx, x \rangle}{\langle Mx, x \rangle}$$

where

$$\langle x, y \rangle = x^T \bar{y} = \Sigma \, x_{kj} \bar{y}_{kj} \ .$$

Note: Since $A = A^*$, $M = M^*$ then $N = N^*$; and, as is well-known [4] the generalized eigenvalues are all real and

$$\rho = \max_{x \neq 0} \frac{|\langle Nx, x \rangle|}{\langle Mx, x \rangle} \ .$$

Thus, the force of the assumption (A.2) is that $\max|\lambda|$ occurs for a positive eigenvalue $\lambda = \rho$.

A.3) There is a positive constant N_0, independent of h, such that

$$\|N\|_\infty \leq N_0 \ .$$

Here

$$\|N\|_\infty = \sup\{|(NU)_{kj}|; \ |u_{kj}| \leq 1\} \ .$$

Finally we come to the main new concept.

A.4) There are positive constants q, K, independent of h, such that: if U is a grid vector satisfying

(i) $$U = 0 \quad \text{on} \quad \partial\Omega(h)$$

and

(ii) $$|\nabla_x U| + |\nabla_y U| \leq B$$

for some constant B, then

3.1) $$\langle NU, U \rangle = q\langle U, U \rangle + E$$

where

3.1a) $$|E| \leq KB/h \ .$$

Remark: As one might imagine, the determination of q and the verification of (A.4) is the important technical aspect of this analysis when applied to any particular case. However, as we shall see in sections 4 and 5, it is not too difficult.

Lemma 3.1: Suppose the splitting (2.9a) satisfies (A.1) and (A.2). Then the method is convergent. That is;

3.2) $$\rho < 1 \ .$$

Proof: Let U be the eigenvector associated with ρ. Then $\langle NU,U\rangle \geq 0$. Since $M = A + N$ and A is positive definite, we have

$$0 \leq \rho = \frac{\langle NU,U\rangle}{\langle MU,U\rangle} = \frac{\langle NU,U\rangle}{\langle AU,U\rangle + \langle NU,U\rangle} < 1 \ .$$

The basic result of this section is

Theorem 3.1: Suppose the splitting (2.9a) satisfies the conditions (A.1), (A.2), (A.3) and (A.4). Then

3.3) $$\rho = 1 - \frac{2\pi^2}{q} h^2 + O(h^3) \ .$$

Proof: Let U be the grid vector

3.4) $$u_{kj} = (\sin k\pi h)(\sin j\pi h) \ .$$

Then U satisfies conditions (i), (ii) of (A.4). In particular, because of (i) we may speak of $\langle NU,U\rangle$ and $\langle U,U\rangle$. The constant B of (ii) is 2π. The following facts are well known (see [13] particularly page 202).

3.5) $$h^2\langle U,U\rangle = \frac{1}{4}\left(\frac{P}{P+1}\right)^2$$

3.6) $$\frac{h^2\langle AU,U\rangle}{h^2\langle U,U\rangle} = 4(1 - \cos\pi h) = 2\pi^2h^2[1 - \frac{1}{12}(\pi h)^2 + O(h^4)] \ .$$

For all V which are zero on $\partial\Omega(h)$ and $V \neq 0$,

3.7) $$\frac{-h^2\langle \Delta_h V,V\rangle}{h^2\langle V,V\rangle} = \frac{1}{h^2}\left[\frac{h^2\langle AV,V\rangle}{h^2\langle V,V\rangle}\right] \geq 2\pi^2[1 - \frac{1}{12}(\pi h)^2 + O(h^4)] \ .$$

Since $M = A + N$

$$\rho \geq \frac{\langle NU,U\rangle}{\langle MU,U\rangle} = \frac{h^2\langle NU,U\rangle}{h^2\langle AU,U\rangle + h^2\langle NU,U\rangle} \quad .$$

Applying (A.4) we have

$$h^2\langle NU,U\rangle = q[h^2\langle U,U\rangle] + h^2E \quad .$$

And, using (3.5) we have

$$h^2\langle NU,U\rangle = [q + O(h)][h^2\langle U,U\rangle] \quad .$$

Thus

$$\rho \geq \frac{1}{1 + \dfrac{h^2\langle AU,U\rangle}{[q + O(h)][h^2\langle U,U\rangle]}} \quad .$$

Using (3.6) we obtain

3.8) $$\rho \geq 1 - \frac{2\pi^2h^2}{q} + O(h^3) \quad .$$

In order to obtain the reverse inequality we require some basic estimates of [1] and [8]. These are

Lemma 3.2: Let V be a grid vector which is zero on $\partial\Omega(h)$. Then

3.9) $$|v_{kj}| \leq \frac{1}{4}\sqrt{1+\pi}\,\{h^2\langle\Delta_h V,\Delta_h V\rangle\}^{1/2} \quad .$$

Proof: See lemma 8, page 304 of [8].

Lemma 3.3: Let V be a grid vector which is zero on $\partial\Omega(h)$. Then

3.10a) $$|\nabla_x V| \leq \max|\Delta_h V| \quad ,$$

3.10b) $$|\nabla_y V| \leq \max|\Delta_h V| \quad .$$

Proof: This result is contained in Theorem 5, page 488 of [1].

For convenience of notation, for every grid vector V, restricted to the interior $\Omega(h)$, we write

3.11) $$\|V\|_g = \{h^2\langle V,V\rangle\}^{1/2} \quad .$$

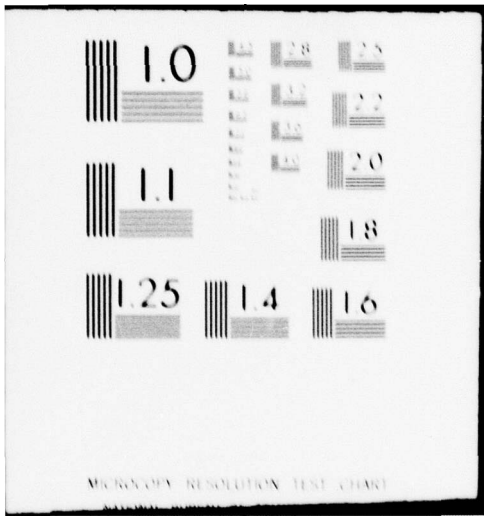
AD-A061 561 ARMY RESEARCH OFFICE RESEARCH TRIANGLE PARK N C F/G 12/1
PROCEEDINGS OF THE 1978 ARMY NUMERICAL ANALYSIS AND COMPUTERS C--ETC(U)
OCT 78

UNCLASSIFIED ARO-78-3 NL

4 OF 4
AD
A061 561



END
DATE
FILMED
2-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

Returning to the proof of the theorem, let U be the eigenvector associated with ρ and normalized so that

3.12) $$\|U\|_g = 1 .$$

Then

$$\rho MU = NU$$

$$\rho AU = \rho(M - N)U = (1 - \rho)NU .$$

That is

3.13a) $$-\Delta_h U = \mu NU$$

where

3.13b) $$\mu = (1 - \rho)/\rho h^2 .$$

From lemma 3.1 and (3.8) we see that

3.14a) $$0 < \mu$$

and

3.14b) $$\limsup_{h \to 0} \mu \leq \frac{2\pi^2}{q} .$$

Moreover, the theorem will be proven if we show that

$$\mu = \frac{2\pi^2}{q} + O(h) .$$

We write (3.13a) as

$$-\Delta_h U = \psi$$

where, if h is small enough

$$\|\psi\|_g \leq \frac{4\pi^2}{q} N_0 = N_1 .$$

Applying lemma 3.2 we see that

$$|u_{kj}| \leq \frac{1}{4}\sqrt{1 + \pi}\, N_1 .$$

Thus

$$|\psi_{kj}| \leq \frac{4\pi^2}{q} N_0 [\frac{1}{4}\sqrt{1 + \pi}\, N_1] = N_2 .$$

Applying lemma 3.3 we have (ii) of (A.4) with $B = 2N_2$. Hence, using (A.4) we have

$$h^2(NU,U) = q[h^2(U,U)] + h^2E .$$

Or, making use of (3.12)

3.15) $$h^2(NU,U) = [q + O(h)][h^2(U,U)] .$$

From (3.13a) we have

$$-h^2(\Delta_h U,U) = \mu h^2(NU,U)$$
$$= \mu[q + O(h)][h^2(U,U)] .$$

Hence, from (3.7)

$$2\pi^2[1 + O(h^2)] \leq \mu[q + O(h)] .$$

Thus, combining this result with (3.14b), the theorem is proven.

4. $p \times p$ BLOCKS. Let p be a fixed integer and assume that

4.1) $$P = pQ .$$

Of course, as $P \to \infty$ (i.e. $h \to 0$) $Q \to \infty$ and vice-versa.

The interior grid vector is arranged into sub grid vectors $\{U_{rs}\}$ of p^2 entrees as follows

4.2) $$U_{rs} = \{u_{(r-1)p+\sigma,(s-1)p+\mu},\ 1 \leq \sigma,\mu \leq p\}, \quad 1 \leq r,s \leq Q .$$

Within U, the U_{rs} are ordered as follows

4.3) $$U = \{U_{11},U_{21}, \cdots U_{Q1},U_{12},U_{22}, \cdots U_{Q2}, \cdots U_{1Q}, \cdots U_{QQ}\}^T .$$

That is; we start at the bottom row of $p \times p$ blocks and count off from left to right; then to the next (second) row of $p \times p$ blocks - again from left to right, etc. Within each block (U_{rs}) the subgrid vector is ordered in the same manner. To be specific, let $G(r,s,\mu)$, $\mu = 1,2,\ldots,p$ be the p vector of grid values associated with the μ^{th} horizontal line within the (r,s) block. That is

4.4) $$G(r,s,\mu) = \begin{bmatrix} u_{(r-1)p+1,(s-1)p+\mu} \\ u_{(r-1)p+2,(s-1)p+\mu} \\ \vdots \\ u_{(r-1)p+\sigma,(s-1)p+\mu} \\ \vdots \\ u_{rp,(s-1)p+\mu} \end{bmatrix},$$

then

4.5) $$U_{rs} = \begin{bmatrix} G(r,s,1) \\ \vdots \\ G(r,s,\mu) \\ \vdots \\ G(r,s,p) \end{bmatrix}.$$

The discrete Poisson equation (2.5a), (2.5b) takes the form

4.6) $$TU_{rs} - A_{-1}U_{r-1,s} - A_1U_{r+1,s} - B_{-1}U_{r,s-1} - B_1U_{r,s+1} = F_{rs}$$

where T, A_{-1}, A_1, B_{-1}, B_1 are $p^2 \times p^2$ matrices. Each is block tridiagonal of "pseudo order" p and each block is a $p \times p$ matrix. Specifically

$$T = [-I_p, R_p, -I_p] \quad \text{"block tridiagonal"}$$

$$R_p = [-1,4,-1] \qquad \text{tridiagonal .}$$

If $E_{\alpha\beta}$ is the $p \times p$ matrix with "1" in the (α,β) position and zero elsewhere, then

4.7a) $$A_{-1} = \text{diagonal}[E_{1p}, E_{1p}, \ldots, E_{1p}],$$

4.7b) $$A_1 = \text{diagonal}[E_{p1}, E_{p1}, \ldots, E_{p1}].$$

Notice that

$$E_{1p} = \begin{bmatrix} & & 1 \\ & \mathbf{0} & \end{bmatrix}, \quad E_{p1} = E_{1p}^T.$$

The matrix B_{-1} is the "block" E_{1p} while B_1 is the "block" E_{p1}. That is

4.8) $$B_{-1} = \begin{bmatrix} & I_p \\ O & \end{bmatrix}, \quad B_1 = \begin{bmatrix} & O \\ I_p & \end{bmatrix}.$$

We rewrite (4.6) as

4.9) $$MU = NU + F$$

where $M = \text{diagonal}[T,T,\ldots,T]$ and N is made up of A_{-1}, A_1, B_{-1}, B_1.

We see at a glance that M is positive definite and (A.1) is satisfied. Furthermore, we are dealing with a "block" five point star, that is, the equations have the same block structure as the original problem. Therefore, our splitting has "block property A". Thus, we have the basic result, if λ is an eigenvalue of $\det\{\lambda M - N\} = 0$, so is $-\lambda$ (see [14], [15]). Therefore (A.2) is satisfied.

Now, N only includes the coefficients in Δ_h which relate points in the (r,s) block with neighboring points in the four blocks $(r+1,s)$, $(r-1,s)$, $(r,s+1)$, $(r,s-1)$. We see that each row of N not corresponding to a corner point of the (r,s) block has at most one "1" and all other entrees are "0". The rows corresponding to corners lead to exactly two "1"'s. Thus

4.10) $$\|N\|_\infty = 2 ,$$

and (A.3) is satisfied with $N_0 = 2$.

Finally we turn our attention to the determination of q and the verification of (A.4).

Lemma 4.1: Suppose U is a grid vector which satisfies (i), (ii) of (A.4). Then

4.11a) $$\langle NU,U\rangle = \frac{4}{p}\langle U,U\rangle + E$$

where

4.11b) $$|E| \leq \frac{16B^2}{h} .$$

That is, (A.4) is satisfied with

4.12a) $$q = \frac{4}{p}$$

and

4.12b) $$K = 16B^2 .$$

Proof: We have

4.13) $$\langle NU,U\rangle = \sum_{r,s} U_{rs}^T (NU)_{rs} .$$

Consider a term

$$4.14)\quad U^T_{rs}(NU)_{rs} = U^T_{rs}A_{-1}U_{r-1,s} + U^T_{rs}A_1U_{r+1,s} + U^T_{rs}B_{-1}U_{r,s-1} + U^T_{rs}B_1U_{r,s+1} .$$

It is easy to see that

$$4.15)\qquad U^T_{rs}A_{-1}U_{r-1,s} = \sum_{\mu=1}^{p} u_{(r-1)p,(s-1)p+\mu} \cdot u_{(r-1)p+1,(s-1)p+\mu} .$$

Fix σ, $1 \le \sigma \le p$. We use (ii) to write

$$F_\mu = u_{(r-1)p,(s-1)p+\mu} \cdot u_{(r-1)p+1,(s-1)p+\mu}$$

$$= [u_{(r-1)p+\sigma,(s-1)p+\mu} + \sigma h\beta_1][u_{(r-1)p+\sigma,(s-1)p+\mu} + \sigma h\beta_2]$$

where

$$|\beta_j| \le B, \qquad j = 1,2 .$$

Thus

$$F_\mu = [u_{(r-1)p+\sigma,(s-1)p+\mu}]^2 + 2\varepsilon_1 h + \varepsilon_2 h^2$$

where

$$|\varepsilon_j| \le (pB)^2, \qquad j = 1,2 .$$

Therefore, we may replace F_μ by the average over σ, $1 \le \sigma \le p$. Thus

$$U^T_{rs}A_{-1}U_{r-1,s} = \sum_{\mu=1}^{p} F_\mu = \frac{1}{p}\sum_{\mu=1}^{p}\sum_{\sigma=1}^{p} [u_{(r-1)p+\sigma,(s-1)p+\mu}]^2 + E_1$$

where

$$|E_1| \le 2(pB)^2 h(1 + h) .$$

Each of the other terms in the right-hand-side of (4.14) may be treated in a similar manner. We obtain

$$U^T_{rs}(NU)_{rs} = \frac{4}{p} U^T_{rs}U_{rs} + E_2$$

where

$$|E_2| \le 16(pB)^2 h .$$

Finally, using (4.13) we see that

$$\langle U, NU \rangle = \frac{4}{p} \langle U, U \rangle + E$$

where

$$|E| \leq 16(pB)^2 Q^2 h \leq (16B^2)\frac{1}{h} .$$

Corollary: If one considers the $p \times p$ block Jacobi iterative method described by (4.6)-(4.9) then

$$\rho = 1 - \left(\frac{\pi^2}{2} p\right) h^2 + O(h^3) .$$

Proof: Apply Theorem 3.1.

We close this section with a consideration of the successive over-relaxation iterative method based on this splitting.

Let a parameter ω be chosen. Then the successive over relaxation method based on (4.6) is given by

$$\frac{1}{\omega} TU^{k+1}_{r,s} = A_{-1}U^{k+1}_{r-1,s} + B_{-1}U^{k+1}_{r,s-1} + A_1 U^k_{r+1,s} + B_1 U^k_{r,s+1} + (\frac{1}{\omega} - 1)U^k_{rs} + \tilde{F}_{rs} .$$

Because the basic splitting satisfies block property A the number $\rho(\omega)$ which is the related spectral radius satisfies the equation (see [15])

$$(\rho(\omega) + \omega - 1)^2 = \omega^2 \rho^2 \rho(\omega) .$$

Thus, having determined ρ, we know $\rho(\omega)$.

5. p LINE METHOD: I. Again, let P be a fixed integer and assume that (4.1) holds.

The interior grid vector is arranged into subgrid vectors $\{U_j\}$ of pP entrees as follows

5.1) $$U_j = \{u_{\sigma,(j-1)p+\mu};\ 1 \leq \sigma \leq P,\ 1 \leq \mu \leq p\} .$$

That is, U_j consists of the values associated with the j^{th} block of p lines. We now have

5.2) $$U = \{U_1, U_2, \ldots, U_Q\}^T .$$

Within each U_j the ordering is the same. That is, let $G(j,\mu)$ be the P vector associated with the μ^{th} horizontal line within the j^{th} block of p horizontal lines, i.e.

$$G(j,\mu) = \begin{bmatrix} u_{1,(j-1)p+\mu} \\ u_{2,(j-1)p+\mu} \\ \vdots \\ u_{\sigma,(j-1)p+\mu} \\ \vdots \\ u_{p,(j-1)p+\mu} \end{bmatrix},$$

then

$$U_j = \begin{bmatrix} G(j,1) \\ G(j,2) \\ \vdots \\ G(j,p) \end{bmatrix}.$$

The discrete Poisson equation (2.5a), (2.5b) now takes the form

5.3) $$TU_j = RU_{j-1} + R^T U_{j+1} + \tilde{F}_j$$

where T and R are pP by pP matrices. In fact, T is block tridiagonal with

5.4a) $$T = [-I_p, T_p, -I_p]_p \qquad T_p = [-1,4,-1]_p$$

and

5.4b) $$R = \begin{bmatrix} & I_p \\ O & \end{bmatrix}.$$

This decomposition is used to make the splitting

5.5) $$A = M - N$$

where

5.6a) $$M = \text{diagonal}(T)$$

5.6b) $$N = [R,0,R^T].$$

It is immediately clear that $M = M^*$ and is positive definite since each T is positive definite. Once more, this splitting satisfies block property A (see [9], [10]). Thus (A.1) and (A.2) are satisfied. From the structure of N we see that (A.3) is satisfied with $N_0 = 2$.

Once more we seek to determine an appropriate q and verify (A.4).

Lemma 5.1: Suppose U is a grid vector which satisfies (i) and (ii) of (A.4). Then

5.7a) $$\langle NU,U\rangle = \frac{2}{p}\langle U,U\rangle + E$$

where

5.7b) $$|E| \leq 8B^2p .$$

That is, (A.4) is satisfied with

5.8a) $$q = 2/p$$

and

5.8b) $$K = 8B^2p .$$

Proof: We have

5.9) $$\langle NU,U\rangle = \sum_{j=1}^{Q} U_j^T(NU)_j .$$

Consider a term

5.10) $$U_j^T(NU)_j = U_j^TRU_{j-1} + U_j^TR^TU_{j+1} .$$

Now

$$U_j^TRU_{j-1} = \sum_{\sigma=1}^{P} u_{\sigma,(j-1)p+1} \cdot u_{\sigma,(j-1)p} .$$

Fix μ, $1 \leq \mu \leq p$. Then

$$[u_{\sigma,(j-1)p+1}][u_{\sigma,(j-1)p}] = [u_{\sigma,(j-1)p+\mu} + \varepsilon_1][u_{\sigma,(j-1)p+\mu} + \varepsilon_2]$$

where

$$|\varepsilon_j| \leq Bph .$$

Therefore, averaging once more, we have

$$U_j^TRU_{j-1} = \frac{1}{p}\left[\sum_{\mu=1}^{p}\sum_{\sigma=1}^{P}[u_{\sigma,(p-1)j+\mu}]^2 + 4hp^2B^2\right] .$$

Thus, as in section 4

$$\langle U,NU\rangle = \frac{2}{p}\langle U,U\rangle + E$$

where

$$|E| \leq 8B^2p .$$

Corollary: If we consider the p line iterative method described by (5.3)-(5.6b), then

$$\rho = 1 - p\pi^2h^2 + O(h^3) .$$

Proof: Apply theorem 3.1.

Remark: A careful look at this section shows that $K = 8B^2ph$ and hence we easily obtain

$$\rho = 1 - p\pi^2h^2 + O(h^4) .$$

6. p-LINE METHOD II. In this section we approach the p-line method of section 5 with another method of analysis. The results obtained are weaker, but the approach may well have applications in cases where the analysis of section 3 does not apply.

Lemma 6.1: Let $u_n(x)$ denote the Chebychev polynomial of the second kind of order n, i.e., $u_0(x) = 1$, $u_1(x) = 2x$ and $u_{n+1}(x) = 2xu_n(x) - u_{n-1}(x)$.

If $x \geq 1$, then

6.1a) $$u_n(x) \geq u_{n-1}(x) + 1, \qquad n \geq 1$$

6.1b) $$u_n(x) \geq n + 1, \qquad n \geq 1$$

and

6.1c) $$\frac{d}{dx} u_n(x) \geq \frac{d}{dx} u_{n-1}(x) + 2n .$$

Proof: Apply induction.

Corollary: $u_n'(x) \geq 2$, $n \geq 1$ and $x \geq 1$.

Lemma 6.2: Let $B = [-I,2S,-I]_m$ where S and I are $n \times n$. Let

$$U_j(S) = u_j(S) ,$$

that is, $U_j(S)$ is an $n \times n$ matrix obtained by evaluating u_j, the Chebychev polynomial, at S. If $U_m(S)$ is nonsingular, then

6.2) $$B^{-1}_{ij} = \begin{cases} U_m^{-1}(S)U_{j-1}(S)U_{m-i}(S), & j \leq i , \\ U_m^{-1}(S)U_{i-1}(S)U_{m-j}(S), & i \leq j . \end{cases}$$

Proof: See Theorem 1 of [3].

The quantity of interest, ρ, is the spectral radius of $M^{-1}N$. Since

$$M^{-1}N = [T^{-1}R, O, T^{-1}R^T]$$

(see section 5) we may apply Lemma 6.2 to obtain T^{-1} and hence $M^{-1}N$. We find that

$$M^{-1}N = [D, O, E]$$

where

6.3a)
$$D = \begin{bmatrix} \bigcirc & \begin{matrix} U_p^{-1}U_{p-1} \\ U_p^{-1}U_{p-2} \\ \vdots \\ U_p^{-1}U_0 \end{matrix} \end{bmatrix},$$

6.3b)
$$E = \begin{bmatrix} \begin{matrix} U_p^{-1}U_0 \\ U_p^{-1}U_1 \\ \vdots \\ U_p^{-1}U_{p-1} \end{matrix} & \bigcirc \end{bmatrix},$$

and

6.3c)
$$U_j = U_j(\tfrac{1}{2}T_p) .$$

If Q denotes the unitary matrix which diagonalizes T_p (and hence $\frac{1}{2}T_p$), then

(6.4a)
$$\begin{cases} Q^{-1}U_j(\frac{1}{2}T_p)Q = U_j(\frac{1}{2}Q^{-1}T_pQ) \\ \qquad\qquad = \text{diag}\{u_j(\lambda_r)\}, \quad r = 1,2,\ldots,P , \end{cases}$$

and

6.4b)
$$\begin{cases} Q^{-1}U_p^{-1}(\frac{1}{2}T_p)Q = U_p^{-1}(\frac{1}{2}Q^{-1}T_pQ) \\ \qquad\qquad = \text{diag}\{u_p^{-1}(\lambda_r)\}, \quad r = 1,2,\ldots,P , \end{cases}$$

where

6.4c) $$\lambda_r = 2 - \cos(r\pi h) > 1$$

is the r'th eigenvalue of $\frac{1}{2} T_p$.

Let

$$\hat{Q} = \text{diag}\{Q, Q, \ldots, Q\}$$

we see that

6.5) $$\hat{Q}^{-1} M^{-1} N \hat{Q} = [Q^{-1} D Q, O, Q^{-1} E Q] .$$

Thus, applying the Gershgorin circle theorem

6.6) $$\rho \leq B_p \equiv \max_{\substack{r \\ 1 \leq j \leq p}} \{u_p^{-1}(\lambda_r)[u_{p-j}(\lambda_r) + u_{j-1}(\lambda_r)]\} .$$

Lemma 6.3: If $x \geq 1$, then

6.7) $$u_0(x) + u_{p-1}(x) \geq u_{j-1}(x) + u_{p-j}(x), \qquad j = 1, 2, \ldots, p .$$

Proof: For $x \geq 1$ and $i \geq 2$, from the basic recursion formula and (6.1b) we have

$$u_i(x) - u_{i-1}(x) \geq u_{i-1}(x) - u_{i-2}(x) .$$

Thus, if $n > m > 0$ we have

$$u_n(x) - u_{n-1}(x) \geq u_m(x) - u_{m-1}(x) ,$$

or

6.8) $$u_n(x) + u_{m-1}(x) \geq u_m(x) + u_{n-1}(x) .$$

Of course (6.7) is true for $j = 1$ and $j = p$. Assume that (6.7) is true for a value of $j = \sigma$, $1 \leq \sigma \leq p - 2$. Then

6.9) $$u_0(x) + u_{p-1}(x) \geq u_{\sigma-1}(x) + u_{p-\sigma}(x) .$$

We may assume $\sigma < p - \sigma$. Then applying (6.8) with $n = p - \sigma$ and $m = \sigma$ we find

$$u_{p-\sigma}(x) + u_{\sigma-1}(x) \geq u_{p-(\sigma+1)}(x) + u_{\sigma}(x) .$$

That is

$$u_{p-\sigma}(x) + u_{\sigma-1}(x) \geq u_{j}(x) + u_{p-(j+1)}(x) .$$

Substitution of this result into (6.9) gives (6.7) for the larger value of j and the lemma is proven.

Theorem 6.1: With B_p defined by (6.6) we have

6.10) $$\rho(M^{-1}N) \leq B_p = 1 - \frac{p}{2}\pi^2 h^2 + O(h^4) .$$

Proof: From Lemma 6.3 we see that

$$B_p = \max_{\lambda_r} \frac{1 + u_{p-1}(\lambda_r)}{u_p(\lambda_r)} ,$$

where the λ_r are given by (6.4c). It is not difficult to see that

$$\hat{B}_p(\lambda) = \frac{1 + u_{p-1}(\lambda)}{u_p(\lambda)}$$

is a monotone non-decreasing function of λ for $\lambda \geq 1$. Thus

$$B_p = \frac{1 + u_{p-1}(2 - \cos\pi h)}{u_p(2 - \cos\pi h)} .$$

Expansion of $\hat{B}_p(\lambda)$ about $\lambda = 1$ yields (6.10).

Final Remark: Since it is better to slightly overestimate the relaxation parameter in successive overrelaxation than to underestimate it, it might appear that the estimate of this section is preferable to that of section 5 for coarse meshs. However, numerical experiments contradict this idea.

REFERENCES

[1] Brandt, Achi, "Estimates for difference quotients of solutions of Poisson type difference equations", Math. of Comp., 20 (1966), pp. 473-499.

[2] Buzbee, B. L., Golub, G. H., and Howell, J. A., "Vectorization for the CRAY-1 of some methods for solving elliptic difference equations", High Speed Computer and Algorithm Organization, Eds. David J. Kuck, Duncan H. Lawrie, and Ahmed H. Sameh, pp. 255-272, Academic Press, New York, 1977.

[3] Fischer, C. F. and Usmani, R. A., "Properties of some tridiagonal matrices and their application to boundary value problems", SINUM 6 (1969), pp. 127-142.

[4] Franklin, J. L., Matrix Theory, Prentice-Hall, Englewood Cliffs, N.J., 1968.

[5] George, Alan, "Nested dissection of a regular finite element mesh", SINUM 10 (1973), pp. 345-363.

[6] _____, "Numerical experiments using dissection to solve n by n grid problems", SINUM 14 (1977), pp. 161-179.

[7] Keller, H., "On some iterative methods for solving elliptic difference equations", Quart. Appl. Math. 16 (1958), pp. 209-226.

[8] Nitsche, J. and Nitsche, J. C. C., "Error estimates for the numerical solution of elliptic difference equations", Archive for Rat. Mech. and Anal. (1960), pp. 293-306.

[9] Parter, S. V., "On estimating the 'Rates of Convergence' of iterative methods for elliptic difference equations", Trans. of the A.M.S. 114 (1965), pp. 320-354.

[10] _____, "Multi-line iterative methods for elliptic difference equations and fundamental frequencies", Numerische Math. 3 (1961), pp. 305-319.

[11] Rose, D. J., "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", Graph Theory and Computing, R. C. Read ed. Academic Press, New York, 1972.

[12] Rose, D. J. and Willoughby, R. A. - Editors, Sparse Matrices and their Applications, Plenum Press, N.Y., 1972.

[13] Varga, R. S., "Matrix Iterative Analysis", Prentice-Hall, Englewood Cliffs, N.J., 1962.

[14] Young, David M., Iterative Solution of Large Linear Systems, Academic Press, New York, 1971.

[15] _____, "Iterative methods for solving partial difference equations of elliptic type", Trans. Amer. Math. Soc. 76 (1954), pp. 92-111.

SOFTWARE STRUCTURING: CONCEPTS AND METHODS

Clement L. McGowan
SofTech, Inc.
460 Totten Pond Road
Waltham, MA 02154

Sol J. Greenspan
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 1A7

ABSTRACT. A major premise of this paper is that the high cost and low reliability of software are due to lack of precision and systematic procedures in the early stages of the software development cycle. The greater part of the paper is devoted to a survey of some techniques for Software Analysis and Design. Some underlying principles and commonalities are noted, and some directions for needed research are discussed. Since the paper is tutorial in nature, a large set of references is included.

1. INTRODUCTION. Software development can be viewed as consisting of three phases: Analysis, Design, and Implementation (Figure 1). Analysis produces a functional architecture, which describes what the system must do. Design produces a system architecture, which describes how it will be done by determining structure (parts and interfaces) and choosing algorithms. Implementation produces an operational system.

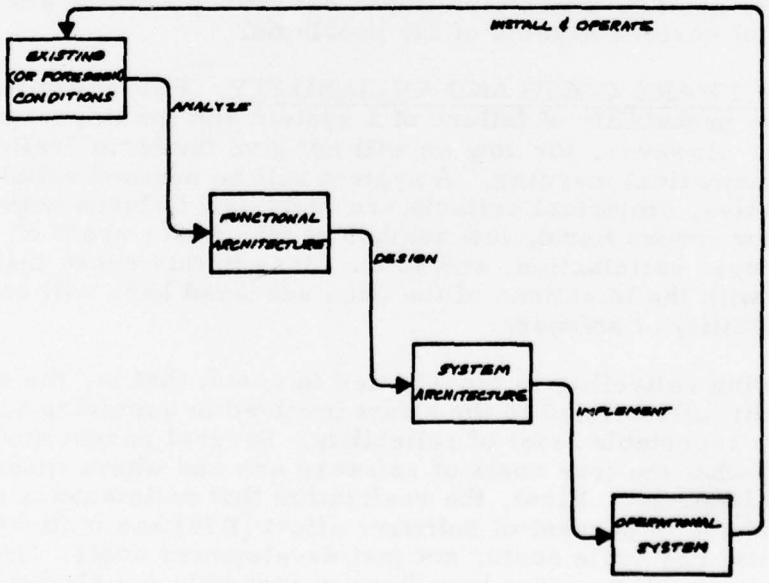


Figure 1. The System Development Cycle

PRECEDING PAGE BLANK

Analysis is part of the broader activity of Requirements Definition. Requirements Definition also consists of context analysis and the specification of design constraints [RS77]. However, we will use both terms, Analysis and Requirements Definition, to refer to essentially the same stage of development. Similarly, Implementation incorporates Programming, as well as other activities, but we will use these two terms more or less interchangeably.

Much attention has been paid to the programming phase, including research and development efforts in structured programming [D72, McGK75, McG75], language design for reliable software [Wo77], and programming methodology [Gr77]. Much less work has been done toward improving the analysis and design phases. This paper discusses a few analysis and design methods that have emerged over the past five or so years. This survey should serve as an introduction for those wishing an overview of the area.

Before beginning with the discussion of the methods, we present some facts concerning software costs; the evidence concerning life cycle costs explains our emphasis on the early stages of software development. Then, the following section of the paper discusses the following methods:

- Jackson Design Technique
- Structured Design Method
- HIPO
- SADT
- PSL/PSA
- SREM

Although none of these methods covers the full range of software development problems, each method offers important concepts which aid our understanding of various aspects of the problems.

2. SOFTWARE COSTS AND RELIABILITY. Reliability is concerned with the probability of failure of a system and the impact of the failures [G78]. However, for now we will not give the term "reliability" a precise mathematical meaning. A system will be deemed reliable if certain qualitative, empirical criteria are met: few failures with only low impact, few errors found, low number of man hours spent on maintenance, high user satisfaction, and so on. It is in this sense that we hope methods with the intentions of the ones surveyed here will contribute to the reliability of software.

Improving reliability is closely tied to costs; that is, the costs incurred are directly related to the effort involved in achieving and maintaining an acceptable level of reliability. Several recent studies have indicated what the true costs of software are and where these costs are primarily incurred. First, the realization that maintenance costs can consume about 75 percent of software effort [B76] has motivated a concern for total life cycle costs, not just development costs. Second, detecting and correcting errors have been responsible for almost half of life cycle costs [A76]. Third, analysis and design errors are, by

far, the most costly and crucial types of errors [B75]. The conclusion we draw from these studies (and from our own experience) is that a larger proportion of the development effort should be spent on analysis and design in order to reduce life cycle costs. For preventing errors is much cheaper than correcting errors that have been built in. And a well structured and documented solution to a clearly defined problem is the legacy system operation needs to understand and to change the existing system with confidence.

3. SOME ANALYSIS AND DESIGN TECHNIQUES. Each of the techniques we will discuss can be characterized with respect to several properties:

- structuring principles, which provide a way of conceptualizing the system as parts and the relationships between them.
- graphical description technique, a language of boxes and arrows that makes explicit the notions of structuring being used.
- procedure for creating descriptions, which vary in how prescriptive or formal they are.
- emphasis on analysis, design, or implementation.
- adequacy or "goodness" criteria, for example, how to judge the goodness of a design, or how to tell whether a set of requirements is "complete" or "consistent."
- pragmatics which deal with how well and on what range of projects the various techniques have worked in practice; this involves not only where in a phased approach a technique was especially applicable but also how easily managed and transferred to new group they proved to be.

4. THE JACKSON TECHNIQUE. The Jackson Technique has many adherents, especially in the United Kingdom and Europe. There is even a Jackson Method User Group with regular meetings. The primary reference is [J75].

The technique's basic structuring principle is stated by Jackson as follows: "program structure should mirror problem structure." Although we often hear such a principle espoused, Jackson means something very simple and specific by it in practice.

By the "structure of the problem," Jackson means the structure of the input and output data. The technique is often used for data processing programs for which the data structures (or file structures) are well understood and easily stated. Structuring is based on a context-free grammar extended with the Kleene star (*) notation. Three primitive constructs are used to represent data structure (Figure 2a).

The first construct represents the decomposition of a data item into a sequence of sub-items. In the example in Figure 2a, A consists of 'B followed by C followed by D'. The iteration construct shows zero or more repetitions: A consists of a sequence of zero or more B's. Selection is denoted by a superscript zero: A consists of either B or C or D. Arbitrarily complex hierarchical structures can be constructed using the three basic notations (Figure 2b).

A program structure can be represented in the same notation as can data structure. In fact, the three constructs correspond to the three basic structured programming constructs, SEQUENCE, DO-WHILE, IF-THEN-ELSE (or CASE). Therefore, any program structure represented with Jackson's constructs has a direct analog in structured code.

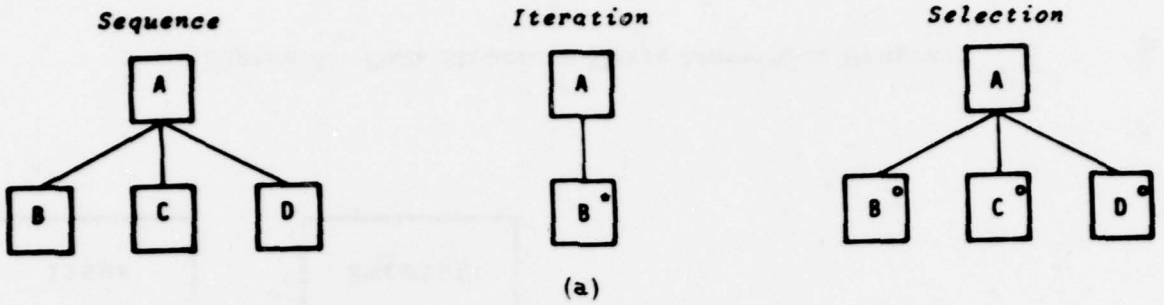
Figure 3 shows the input and output data structures for a system that keeps a record of the movement of parts through a company's stockroom. The input file consists of a series of part groups, each part group consists of a series of movement records, and a movement record is either an "issue" or "receipt" record. The output is a report consisting of a heading, followed by a body which consists of several lines of output, each line corresponding to a Part Group.

Program design involves finding a program whose structure is compatible with both the input and output structures. A program structure compatible with the files of Figure 3 is shown on Figure 4. Discovering the program is the creative step in the design process, like the crafting of an apt loop invariant in structured programming [D72, D76, McGK75]. It is clear that the program can treat both the input and output data structures as sequential files and perform its function in a straightforward manner.

Of course, it is not always possible to find a program structure that is compatible with both the input and output structures. This situation, called a 'structure clash,' is resolved by the creation of an intermediate file. Then it is sufficient to find two subprograms, one that is compatible with the input and the intermediate file, and one that is compatible with the intermediate file and the output. For an example in more detail, see [J75].

5. STRUCTURED DESIGN. The Structured Design method resulted from ten years of research at IBM by Constantine and others [SMC74]. Its most notable characteristic is the provision of specific qualitative criteria for judging the "goodness" of a design.

A "module" is defined as a named set of contiguous program statements. A "good" module is cohesive. Cohesion is a measure of intramodule strength. The possible levels of cohesion, in decreasing order of cohesiveness, and therefore in order of diminishing desirability, are shown in Figure 4. The most desirable level is functional; the least desirable is coincidental.



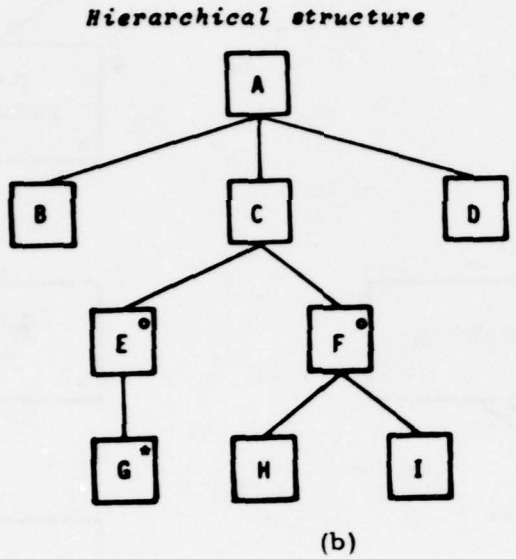


Figure 2. Structure Notations

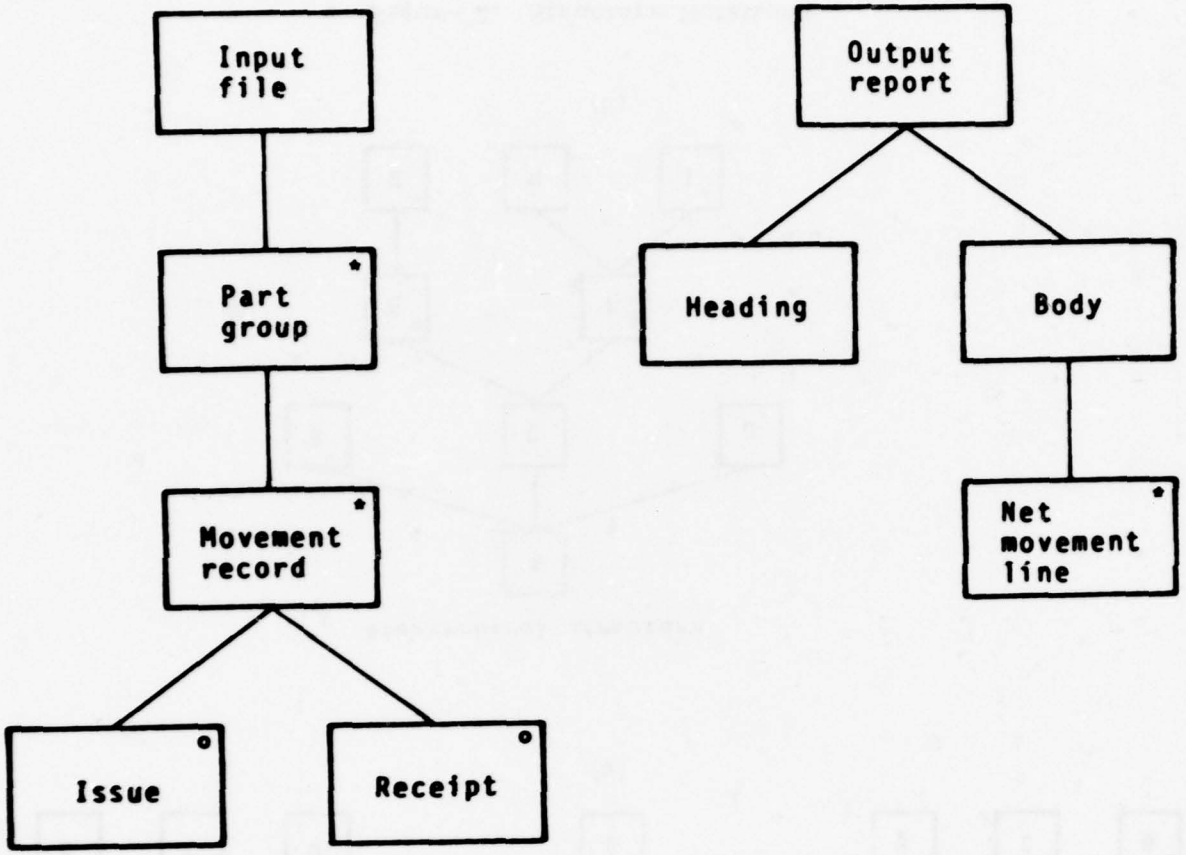


Figure 3. Data Structure Parts Movement Program

1. Functional: module performs a single specific function — "write a record to output file"

2. Clustered: module is a group of functions sharing a data structure usually to hide its representation from the rest of the system; only one function is performed per invocation — "symbol table with insert and look-up functions"

3. Sequential: module action comprises several functions that pass the data along — "update and write a record"

4. Communicational: module action consists of several logical functions operating on some data — "print and punch a file"

5. Procedural: module elements are grouped for algorithmic reasons — "loop body"

6. Temporal: module functions are all related in time — "initialization"

7. Logical: module can perform a general function (i.e., several logically related functions); an invoking parameter value determines the specific function — "general-error-routine" called with an error-number

8. Coincidental: no real relationship between module elements that are grouped for packaging considerations "common subexpression"

Figure 4. Cohesion Levels

A complementary criterion is coupling. Modules are formed so as to minimize the connections between modules. The levels of coupling are shown in Figure 5, in order of increasing module interdependency. Data coupled modules are the most loosely coupled and, hence, most desirable. Content coupled modules are the least desirable.

Using the Structured Design criteria, a design can be assessed and reworked to improve its quality, to enhance its expected performance, or to package it better for its target environment. Some design quality might be sacrificed as a result of such engineering tradeoff analyses, but at least Structured Design has made these decisions and their qualitative costs more explicit.

Besides an evaluative framework, Structured Design prescribes a methodology for creating a design (i.e., a system architecture). The first step is to describe the data flow characteristics of the system using data flow charts, in which the nodes represent functions and the arrows represent the flow of data.

The next step is to decide where the "top" of the system should be. If the program had to be specified as three steps, 1) obtain input, 2) perform tranformation, 3) emit output, how should the data flow diagram be partitioned? This process is called "finding the most abstract form of the input and output," shown here:

(Two modules are ——— coupled if . . .)	
1.	Data: all communication between them is via arguments that are data elements
2.	Stamp: their communication includes an argument that references a data structure (some of whose fields are not needed)
3.	Control: an argument from one knowingly influences the flow-of-control of the other, e.g., flag
4.	External: they reference an externally declared data element
5.	Common: they reference an externally declared (i.e., command data structure (some of whose fields are not needed)
6.	Content: one references the contents of the other

Figure 5. Coupling Levels

The methodology hereafter consists of applying this procedure (recursively) to the Obtain, Perform, and Emit parts in order to obtain a hierarchical program structure. At each level, each of the three (abstract) modules is treated as a subproblem. Hierarchical structure charts are used to document the resulting design.

Structured Design is eminently transferable, especially since the morphology for some standard designs has already been identified [YC75]. Hughes Aircraft has extended the method to obtain additional guidelines for designing real-time military systems such as software for radar systems [Je75].

6. HIPO. HIPO (hierarchy plus input-process-output) was originally developed as a software design aid and documentation technique [H74, Ka76, S76]. It has become a documentation standard in some of IBM's program logic manuals. Recently there have been attempts to use HIPO in developing requirements specifications [Jo76].

A HIPO package consists of a visual table of contents (Figure 6), overview HIPO diagrams, and detail HIPO diagrams. The visual table of contents is a tree structure that gives the hierarchy of the HIPO diagrams. Overview diagrams form the upper levels and are concerned with general functions, inputs (files), and outputs. Detail diagrams form the lower levels and are concerned with subfunctions, specific inputs (records, fields), outputs, and internal data flow. Detail diagrams often have an extended description section which allows for additional notes, details, etc., and contains references to the corresponding code (module names, labels, etc.). The diagrams are suitably cross-referenced.

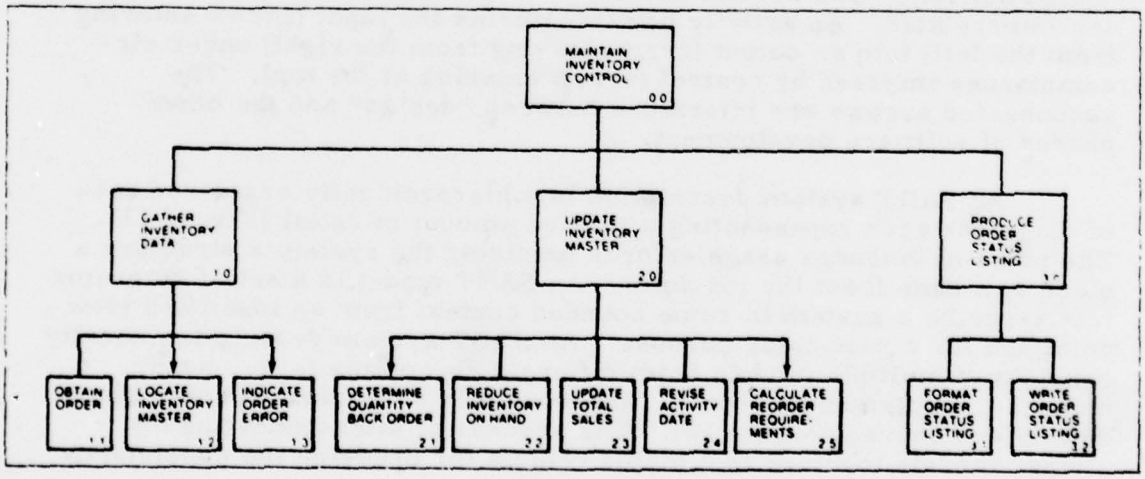


Figure 6. HIPO Table of Contents Showing Hierarchy of System Functions

A HIPO diagram (Figure 7) has, in left to right order, an input, a process, and an output section. Each diagram refines a single function into subfunctions, which are listed as steps in the process section. Inputs and outputs are placed in the appropriate sections. Arrows connect process steps with their inputs/outputs. A HIPO diagram primarily shows function and data flow, but the process section, especially of detail diagrams, can show a limited amount of control flow. A recurring problem in the use of HIPO has been in the specification of interfaces between functions in the hierarchy. The notation does not mandate the matching of interfaces or even prescribe a specific standard form for them.

7. SADT. SADT® — Structured Analysis and Design Technique — is an integrated methodology for requirements definition and the design and specification of software systems. It has been applied to a variety of complex system problems [BM77] including military training [ST76], a government agency's financial management system, software design for a large data base system and for a PABX telephone system, and computer-aided manufacturing [AF74]. "ITT Europe, for example, has used SADT since early 1974 for analysis and design of both hardware/software systems (telephonic and telegraphic) and nonsoftware people-oriented problems (project management and customer engineering)" [RS77].

SADT is built on a graphical language [R77] whose primary structuring principle is hierarchical decomposition (Figure 8). The basic unit of expression in SADT is the box (Figure 9) which represents an activity. Arrows representing data interfaces connect boxes to form diagrams. A typical SADT diagram is shown in Figure 10, which describes the design phase of the software development process. Each box in the diagram represents an activity that is part of the containing design activity. The arrows are all data that show how the activities are interrelated. An activity box transforms the input (arrow entering from the left) into an output (arrow leaving from the right) under circumstances imposed by control (arrow entering at the top). The unconnected arrows are interfaces between "design" and the other phases of software development.

An SADT system description is a hierarchically organized set of diagrams each representing a limited amount of detail (Figure 11). The rules of language usage enforce unfolding the system's structure a piece at a time from the top down. An SADT model is a set of diagrams that describe a system in some bounded context from an identified viewpoint and for a particular purpose. An SADT system description usually consists of multiple models from different viewpoints (e.g., user, manager, implementor, etc.). These models are interconnected where details are shared (Figure 12). The process of interconnecting or "tying" models together is a major vehicle for checking the consistency and completeness of the system description.

[SADT is a trademark of SofTech, Inc.]

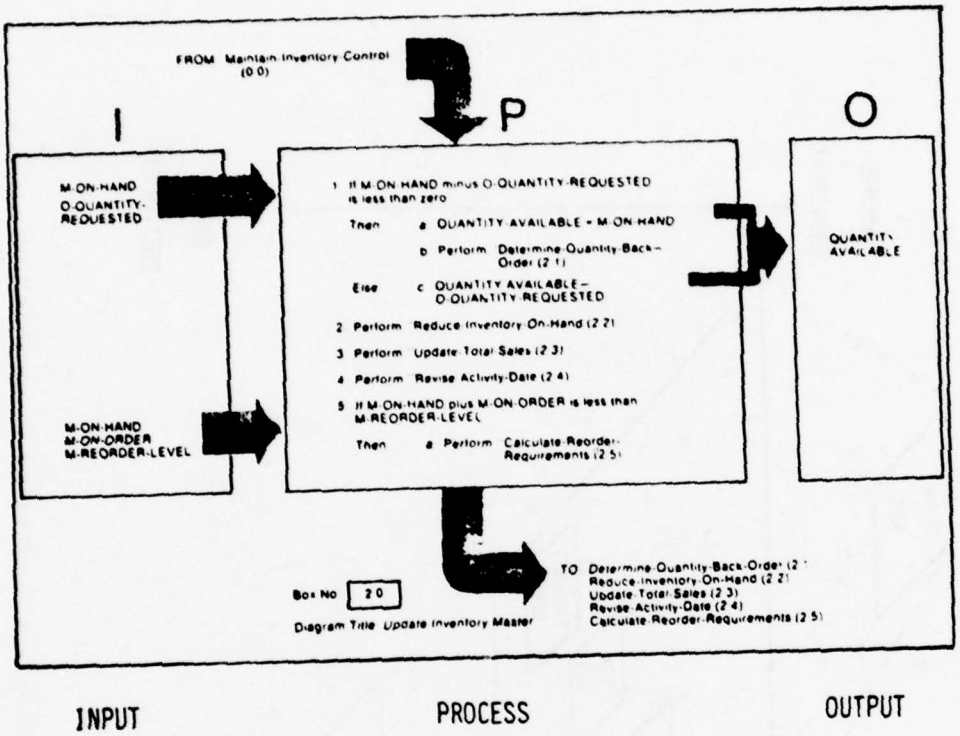


Figure 7. Functional HIPO

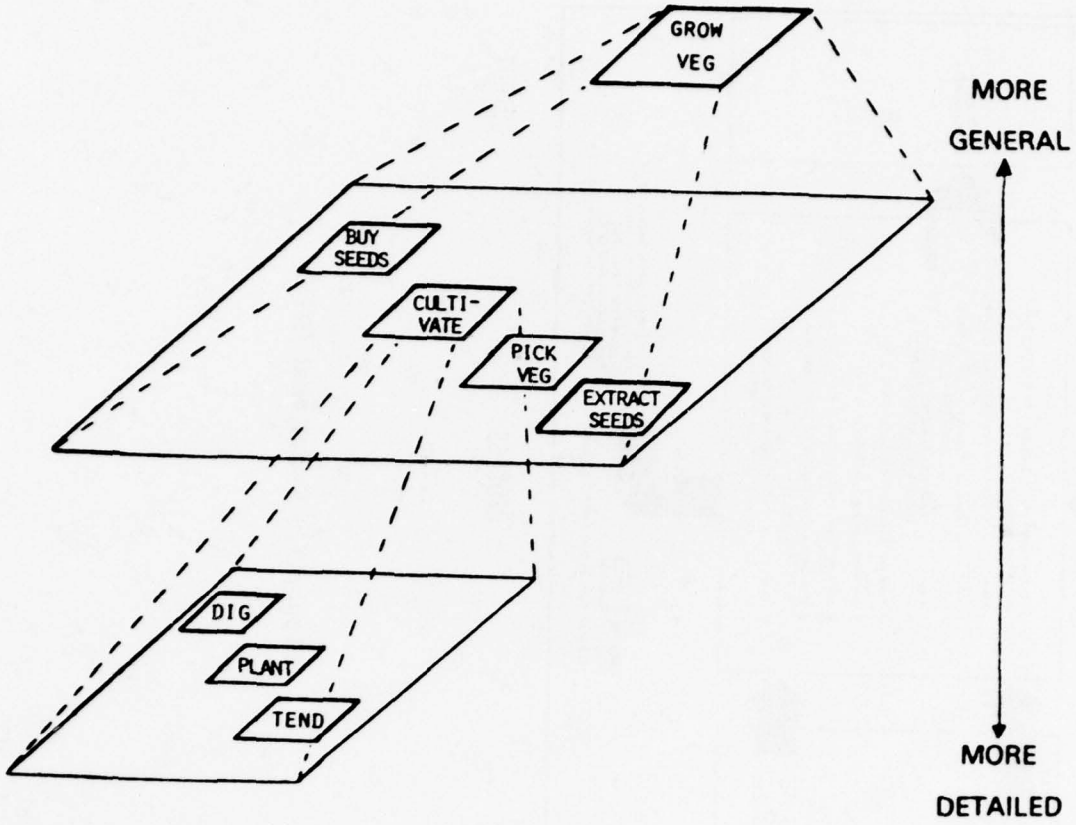


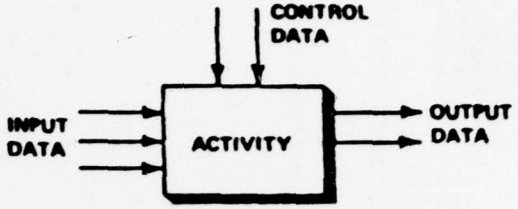
Figure 8. Model Building Via Structured Decomposition

BOXES

SHOW ACTIVITIES

ARROWS

SHOW THE INTERFACES OF A BOX AND CLARIFY ITS ACTION



INPUT DATA

- CONVERTED INTO OUTPUT

CONTROL DATA

- CONSTRAINS HOW INPUT IS MODIFIED TO PRODUCE OUTPUT
- CAN BE USED WITHOUT MODIFICATION TO PRODUCE OUTPUT

Figure 9. The SADT Unit of Expression

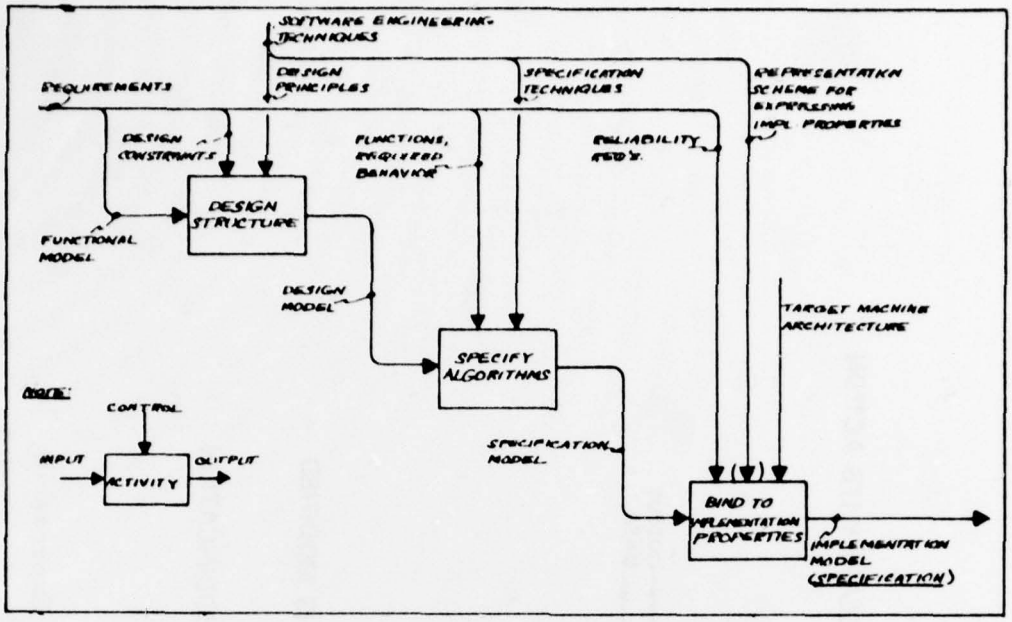


Figure 10. The Design Process

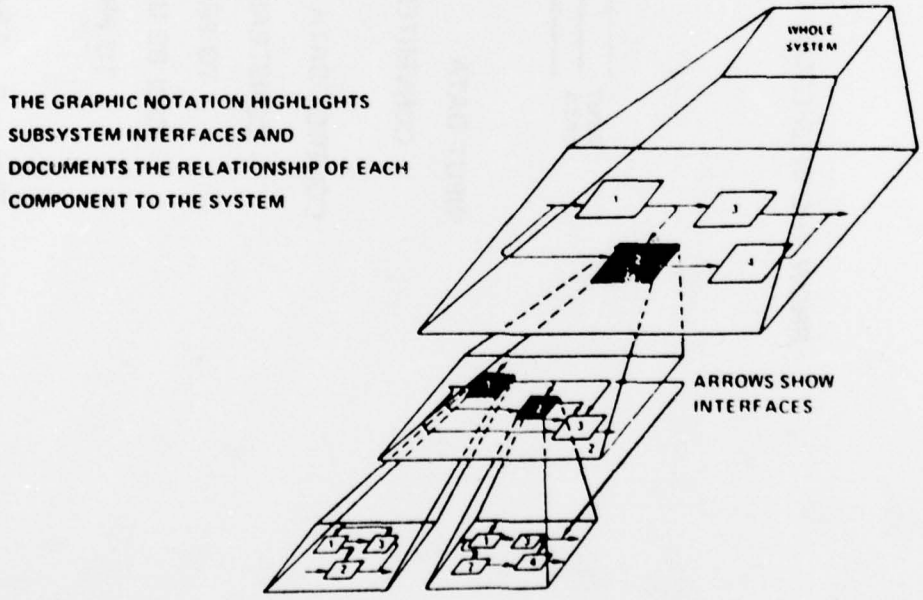


Figure 11. SADT Model

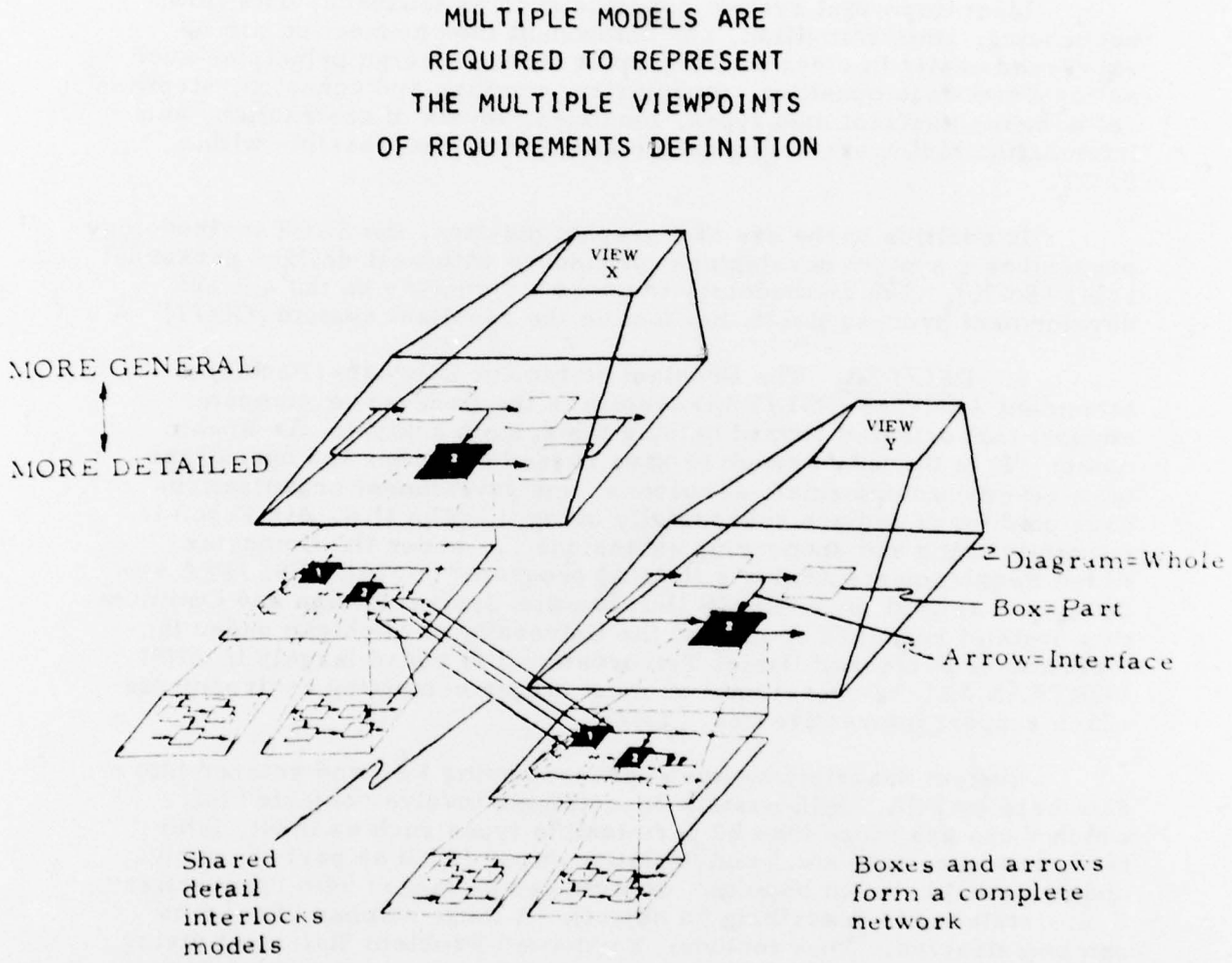


Figure 12. Interconnected SADT Models

The discussion of activity modeling in SADT is only half of the story. Data models are also part of the SADT methodology. A data model shows the top-down decomposition of the data aspects of a system. The notation for data modeling is exactly dual to that for activities: the boxes represent data and the arrows represent activities that create, consume, and use the data. Tying together activity and data models enforces further checks on the system's integrity.

Most important system concepts such as feedback, data flow, sequencing, state transition, and component interconnection can be expressed easily in clear SADT graphic form. Design principles such as top-down decomposition, modularity, coupling and cohesion, stepwise refinement, abstract data types, monitors, levels of abstraction, and information hiding are recognizable and directly expressible within SADT.

In addition to the use of a graphic notation, the SADT methodology prescribes a system development procedure with well-defined personnel roles [Sof76]. The methodology imposes a structure on the system development process itself, not just on the resultant system [GM77].

8. PSL/PSA. The Problem Statement Language/Problem Statement Analyzer (PSL/PSA) system is the first major computer support tool oriented toward helping the system analyst. As Boehm notes: "It is the only system to have passed a market and operations test; several commercial, aerospace, and government organizations have paid for it and are successfully using it. The U.S. Air Force is currently using and sponsoring extensions ... under the Computer Aided Requirements Analysis (CARA) program" [Bo76]. PSL/PSA was developed as part of the ISDOS (Information System Design and Optimization System) research project at the University of Michigan under the direction of Professor Daniel Teichroew. It is coded largely in ANSI FORTRAN and "is operational on most larger computing environments which support interactive use" [TH77].

System descriptions are expressed using PSL and entered into a data base by PSA. PSL system descriptions involve "objects" (of which there are more than 20 permissible types such as input, interface, process, set, etc.) and "relationships" (such as part of, derive, update, etc.) between objects. Objects can be tagged with "properties" (i.e., statements describing an object). A large number of reports can be extracted. They include: Formatted Problem Statement giving all properties and relationships for a specific object, Extended Picture depicting data flows in graphical form, Structure Report presenting hierarchical composition information, plus other summary, change, and data base analysis reports.

9. SREM. TRW working for the U.S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) has developed a significant extension to the PSL/PSA approach [AlB76, A177, Be77, DV77] called Software Requirements Engineering Methodology, SREM. It is addressed explicitly to the real-time process control types of problems represented by ballistic missile defense. The system has

been implemented on an Advanced Scientific Computer and is being transferred to a CDC 7600. It supports multi-color graphics I/O, does requirements consistency checks, has a general query capability, and allows for system simulation executions to be extracted.

As an approach to analysis, SREM regards messages and entities as the system outside (which is equated with system "top"). After these are identified, the necessary processing path for each message is defined. Paths connected to a common interface are combined into a requirements network, R-net. System level requirements are mapped onto the various processing paths. To guarantee testable requirements, validation points are located appropriately along processing paths. SREM then attempts to demonstrate analytic feasibility by simulation. The input Requirements Statement Language, RSL, is based on 4 primitives: Elements (such as message, interface, etc.), Attributes (such as maximum value, entered by, etc.), Relationships (such as composes, connects to, etc.), and Structures (representing the 2-dimensional R-nets).

10. SUMMARY AND CONCLUSIONS. Figure 13 shows where in the system development process the focus of each method is. In our survey we started toward the "detailed design" end of the scale and moved toward methods more applicable in the early requirements definition phases.

Techniques for design and implementation are concerned more with the world of concrete objects, machines, details, and mathematical models. The other end of the scale is concerned with the world of more abstract objects, people, generality and imprecise concepts. The requirements definition area sometimes seems to defy technical analysis and even tends toward the philosophical. This raises a challenge to software engineering researchers of how to represent "high level" knowledge about systems in precise and useful ways. Such research is in progress but much more is needed.

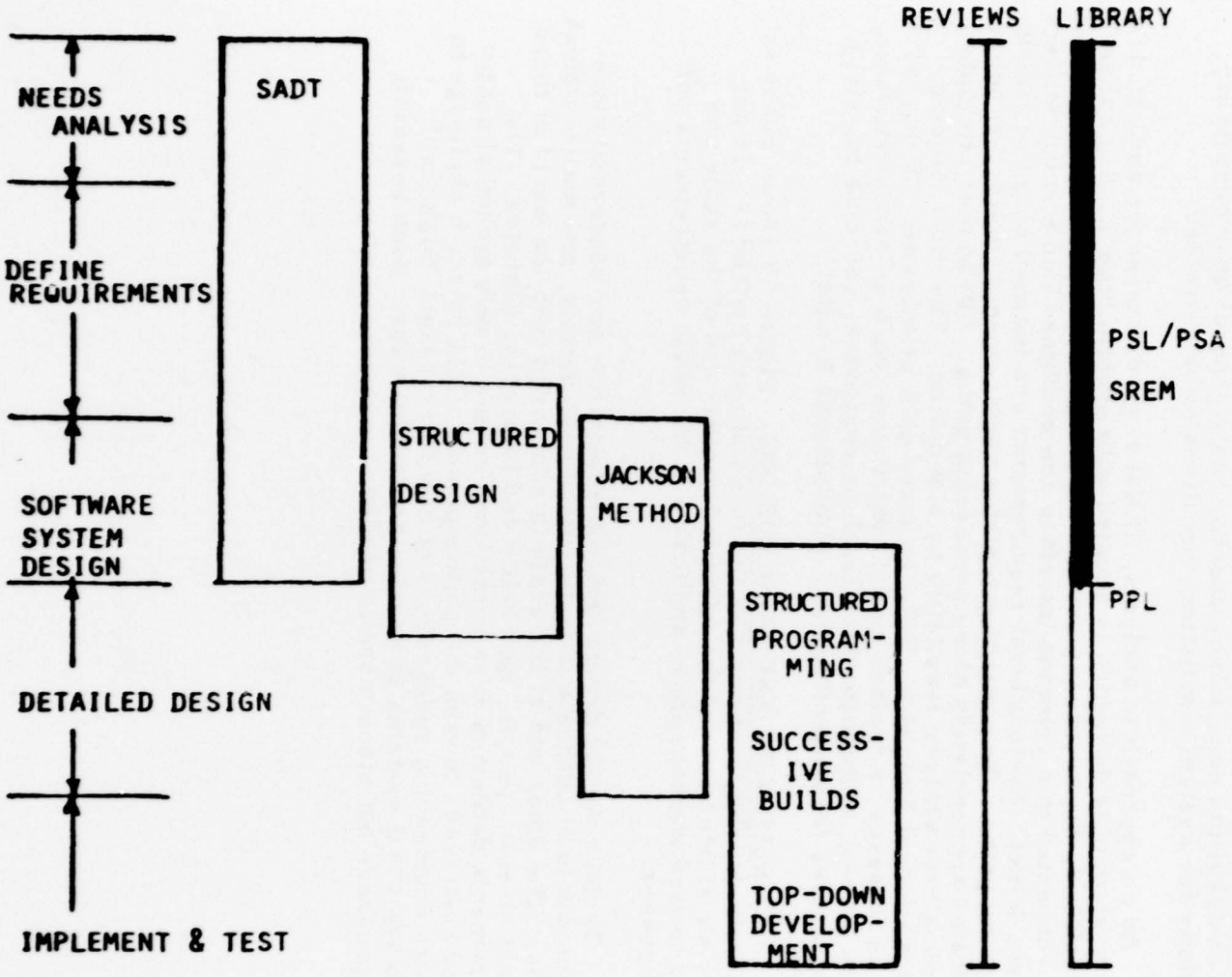


Figure 13. Applicability of Methods

REFERENCES

AF74 Air Force Materials Laboratory, AFSC, WPAFB, Air Force Computer-Aided Manufacturing (AFCAM) Master Plan (Vol. II, App. A, and Vol. III) (Report no. AFML-TR-74-104 available from DDC as AD 922-041 L and 922-171 L), July 1974.

A76 Alberts, D., "The economics of software quality assurance," Proceedings of the National Computer Conference, AFIPS Press, Vol. 45, pp. 433-442, 1976.

Al77 Alford, M., "A requirements engineering methodology for real-time processing requirements," IEEE Trans. Software Eng., 3, 1, pp. 60-68, 1977.

AlB76 Alford, M. W. and I. F. Burns, "R-Nets: A Graph Model for Real-Time Software Requirements," Proc. Sym. on Computer Software Eng., Poly. Inst. of N. Y., pp. 97-108, April 1976.

Be77 Bell, T. E. et al., "An extendable approach to computer-aided software requirements engineering," IEEE Trans. Software Eng., 3, 1, pp. 49-59, 1977.

B76 Boehm, B., "Software Engineering Education: Some Industry Needs," in Software Engineering Education, A. Wasserman and P. Freeman (ed.), Springer-Verlag, New York, 1976.

Bo76 Boehm, B., "Software Engineering," IEEE Trans. Computers, Vol. C-25, no. 12, pp. 1226-1241, 1976.

B75 Boehm, B., R. McClean, and D. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," Proceedings of the International Conference on Reliable Software, pp. 105-113, 1975.

BM77 Brackett, J. W. and C. L. McGowan, "Applying SADT to Large System Problems," Proc. Sym. on System Development Technology, U. of Md., April 1977.

DV77 Davis, C. G. and C. R. Vick, "The software development system," IEEE Trans. Software Eng., 3, 1, pp. 69-84, 1977.

DMR77 Dickover, M. E., C. L. McGowan, and D. T. Ross, "Software Design Using SADT," Proc. Nat. ACM Conf., 1977 (to appear).

D72 Dijkstra, E. W., "Notes on structured programming," in O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Structured Programming, Academic Press, London, 1972.

D76 Dijkstra, E. W., A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.J., 1976.

GM77 Greenspan, S. J., and C. L. McGowan, "Structuring Software Development for Reliability," Proceedings of the Canadian SRE Reliability Symposium, in Microelectronics and Reliability, Vol. 17, no. 1, pp. 75-83, January 1978.

G78 Goodenough, J. B., "A Survey of Program Testing Issues," in Research Directions in Software Technology, MIT Press, 1978 (to appear).

Gr77 Greenspan, S. J., and J. J. Horning, "Programming Methodology: An Annotated Bibliography for IFIP Working Group 2.3," First Edition, Technical Report CSRG-81, University of Toronto, Computer Systems Research Group, May 1977.

HZ76 Hamilton, M. and S. Zeldin, "Higher order software - A methodology for defining software," IEEE Trans. Software Eng., pp. 9-32, March 1976.

H74 HIPO - A Design Aid and Documentation Techniques, IBM Corp., Form GC20-1851, 1974.

J75 Jackson, M. A., Principles of Program Design, Academic Press, London, 1975.

Je75 Jensen, E. Hughes IR&D Structured Design Methodology, Vol. II, Hughes Aircraft Co., 1975.

Jo76 Jones, M. N., "HIPO for developing specifications," DATAMATION 22, 3, pp. 112-125, 1976.

Ka76 Katzan, H., System Design and Documentation, Van Nostrand, New York, 1976.

McG75 McGowan, C., "Structured Programming: A Review of Some Practical Concepts," COMPUTER, Vol. 8, no. 6, pp. 25-30, 1975.

McGK75 McGowan, C. and J. Kelly, Top-Down Structured Programming Techniques Mason/Charter, New York, 1975.

MK77 McGowan, C. L. and J. R. Kelly, "A review of decomposition and design methodologies," Proc. of Infotech Conf. on Structured Design, pp. 117-143, 1977.

P72 Parnas, D. L., "On the criteria to be used in decomposing systems into modules," CACM 15, 12, pp. 1053-1058, 1972.

Ra75 Randell, B., "System structure for software fault tolerance," IEEE Trans. Software Eng., 1, 2, pp. 220-232, 1975.

R77 Ross, D. T., "Structured Analysis: A language for communicating ideas," IEEE Trans. Software Eng., 3, 1, pp. 16-33, 1977.

RS77 Ross, D. T. and K. E. Schoman, "Structured analysis for requirements definition, "IEEE Trans. Software Eng., 3, 1, pp. 6-15, 1977.

ST77 SofTech, Inc., TRAIDEX Needs and Implementation Study (Final Report) DARPA Contract no. MDA 903-75-C-0224, May 1976 (available through NTIS AD-A024 861).

Sof76 "An Introduction to SADT — Structured Analysis and Design Technique," SofTech, Inc., Waltham, MA, November 1976.

S76 Stay, J., "HIPO and integrated program design," IBM Systems Journal, 15, 2, pp. 143-154, 1976.

SMC74 Stevens, W., G. Myers, and L. Constantine, "Structured Design," IBM Systems Journal 13, 1, 114-139, 1974.

TH77 Teichroew, D. and E. A. Hershey, "PSL/PSA: a computer-aided technique for structured documentation and analysis of information processing systems," IEEE Trans. Software Eng., 3, 1, pp. 41-48, 1977.

Wi75 Wilson, M. L., "The Information Automat approach to design and implementation of computer-based systems," IBM FSD Information Automat Tech. Rpt., June 1975.

Wo77 Wortman, D. B. (ed.), "Proceedings of an ACM Conference on Language Design for Reliable Software," March 28-30, 1977, in SIGPLAN Notices 13, 3 (March 1977), and in Operating Systems Review 11, 2 (April 1977), and in Software Engineering Notes 2, 2, March 1977.

YC75 Yourdon, E. and L. L. Constantine, Structured Design, Yourdon, Inc., New York, 1975.

OUT OF TOWN ATTENDEES

BANKS, Norman	APG, BRL
BECK, Peter	ARRADCOM
BEVELOCK, James	ARRADCOM
BOGGS, Dr. Paul	ARO
BRINCKA, George A.	ARRADCOM
BROOME, Paul	APG
CHANDRA, Jagdish	ARO
CHUVALA, Ray	ARRADCOM
CURRY, Allen H.	Computer Systems Command
DOUGLAS, Dr. A. J.	Dept. of National Defense
DE BOOR, Carl	University of Wisconsin
FOSDICK, Lloyd	University of Colorado
GABRIELE, Gary A.	AIRMICS
GOODMAN, S. E.	Princeton University
GREENSPAN, Sol	SofTech, Inc.
GROSSE, Eric	Stanford University
HARRISON, John	APG, BRL
HAUSNER, Arthur	HDL
HIRSCHBERG, Morton	APG, BRL
LACHER, Edward B.	ARRADCOM
LAUNER, Dr. Robert	ARO
LEVY, Leon S.	University of Delaware
MCGOWAN, Clement L.	SofTech, Inc.
MEYER, Gunter H.	Georgia Tech
MILLER, Edward F.	Software Research Associates
NEVANLINNA, Olavi	MRC
NOBLE, Ben	University of Wisconsin
OGDEN, Carl	Ft. Huachuca
PARTER, Seymour V.	University of Wisconsin
RALL, Louis B.	University of Wisconsin
REKLIS, Robert	BRL-LED, APG
ROSSER, J. Barkley	University of Wisconsin
SCANLON, Raymond D.	Watervliet Arsenal

SHEPHERD, William	White Sands Missile Range
SIEGEL, Andrew F.	University of Wisconsin
SMITH, Philip W.	Texas A&M University
SMITH, Shirley	Aviation System Command
STENGER, Frank	University of Utah
STUREK, Walter B.	ARRADCOM
TAYLOR, S. M.	APG, BRL
UHLIG, Ronald P.	HQ USA, DARCOM
WANG, Dr. Chia Ping	US Army Natick R&D Command
WILLIAMS, Robert H.	USA COMISA
WILTSE, Gary K.	USAEPG
WU, C. F.	University of Wisconsin
WU, Julian J.	Watervliet Arsenal
YAU, Stephen S.	Northwestern University
ZIMMERMAN, Kathleen	APG, BRL

LOCAL ATTENDEES

AICHELE, David G.	NASA MSFC
ANDERSON, Archie	MSFC-EF 31
AUER, CPT Joseph A. Jr.	DRDMI-TER
BOYD, D. G.	DRDMI-QRT
BOYD, Helen	DRDMI-HRA
BROCK, Donna L.	DRDMI-TGG
BROWN, Jerry H.	DRDMI-YRP
BRYAN, Ferrell	DRDMI-TGG
BURROWS, Roger R.	EL-23, MSFC
BURT, Jill	DRDMI-CGA
CHRISTIAN, Jim	DRSMI-MMM
COLLINS, Jim	DRSMI-WS
COPELAND, D. E.	DRDMI-TGG
CRAIG, Bill	DRDMI-TGG
CURRY, David M.	DRSMI-WSP
DICKSON, Richard E.	DRDMI-TDW
DIHM, Henry A.	DRDMI-TDW
DOLLMAN, Thomas	NASA-EDY3
DUNCAN, Reynolds	NASA-EL23
FAGAN, J. J.	DRDMI-T
FERNANDEZ, Ken	NASA
GIBBS, Billy G.	DRSMI-WSE
GRAHAM, Maurice	DRDMI-DCM
GRANT, Lynnon F.	NASA/MSFC
HALIJZK, Charles A.	UAH
HALLOWES, John P.	DRDMI-XS
HAN, S. M.	UAH
HARTMAN, R. L.	DRDMI-HR
HSIA, Pei	UAH
JONES, Wayne L.	DRSMI-WSP
LADD, Tom	DRDMI-TGG
LEHNIGK, Siegfried H.	DRDMI-HR

MATHEWS, David	DRDMI-EAA
MCAULEY, Van A.	NASA-AH 33
MILLER, Jim	DRSMI-MM
MITCHELL, Kermit	DRDMI-TKP
MOUTAIN, John B.	DRSMI-WT
MULLIN, Albert A.	BMDATC
NUTT, Richard L.	DRDMI-QRT
ODDO, Joseph A.	BMDSC-HD
PELIY, John M.	DRSMI-WT
PINER, Charles R.	MIRADCOM
PURINTON, Steve	NASA
RAGLAND, James	DRDMI-YRP
ROACH, M. D.	UAH
SCOTT, D. L.	DRSMI-MMM
SIMS, Robert C.	DRSMI-LP
SMITH, Jerry D.	DRSMI-WSE
SOFFERIS, John	DRSMI-WSP
SPRUELL, Wayne	DRSMI-WSP
STONE, Bill	ROHM & HAAS Company
STREET, Troy A.	DRDMI-TDK
TAYLOR, Lyle A.	DRCPM-MD-T-S
TAYLOR, S. M.	DRDAR-BLB
TUBBS, Lawrence A.	BMDATC-P
WATSON, Max	DRSMI-WT

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO REPORT 78-3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Proceedings of the 1978 Army Numerical Analysis and Computers Conference		5. TYPE OF REPORT & PERIOD COVERED Interim Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Army Mathematics Steering Committee on behalf of the Chief of Research, Development and Acquisition		12. REPORT DATE October 1978 13. NUMBER OF PAGES 322
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Army Research Office ATTN: DRXRO-MA P. O. Box 12211 Research Triangle Park, NC 27709		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited. The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This is a technical report resulting from the 1978 Army Numerical Analysis and Computers Conference. It contains papers on computer aided design and engineering as well as papers on numerical analysis.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

error-resistant software design
program descriptions
Bessel functions
repairing software
graphics
programming languages
polynomial interpolation
digital filters
Nystrom method
stability of a missile
transonic flow
numerical simulation of electro-chemical machining
the CEMA Ryad Computer family
algorithms for analysis of computer programs
differential equations and hand held programmable calculators
software equation tool
elastic-plastic deformation
block relaxation techniques
software structuring

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

☆U.S. GOVERNMENT PRINTING OFFICE: 1978-747-784