

Copyright © 1979, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

SOLVER: AN ANALYTIC FUNCTION ROOT SOLVING  
(  
AND PLOTTING PACKAGE

by

H. Stephen Au-Yeung and Alex Friedman

Memorandum No. UCB/ERL M79/55

31 August 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering  
University of California, Berkeley  
94720

## Contents

I. Introduction	1
II. Elementary SOLVER	2
III. Intermediate SOLVER	5
IV. Advanced SOLVER	8
A. Plotting Roots vs. One Parameter	8
B. Plotting a Function vs. One Argument	20
C. Plotting Roots (a Function) vs. Two Parameters (Arguments)	22
APPENDIX I. The Structure of The Library Solver	30
APPENDIX II. Listings of The Source Files	32

## Acknowledgments

We wish to acknowledge the contributions of Y.J. Chen and D. Harned, who helped test the program and offered useful suggestions, and that of Professor C.K. Birdsall, whose comments helped make this document useful. This work was supported by DOE contract DE-AS03-76SF00034, Project Agreement DE-AT03-76ET53064 and ONR contract N00014-77-C-0578.

## I. Introduction

SOLVER is a root solver that provides automatic compilation, loading, and execution. It uses a simple version of Muller's method (see Ref. 1). SOLVER is intended for the occasional user who needs roots but balks at learning to use sophisticated but complicated solvers, which usually also require considerable additional programming. The user need only supply the function to be set to zero and the number of solutions desired. SOLVER is available on the CRAY-1 computer and can be obtained by typing:

```
rfilem read 1222 .cray solver(LF)end / t v
```

This report corresponds to the SOLVER version of August 31, 1979. Later versions of SOLVER will be stored in FILEM directory .cray of user number 1222. The user should periodically check the date of this file (filem how 1222 .cray solver) to see if the program has been updated. The file SOLVER is a LIB file; it contains the latest sources as well as the latest documentation.

To obtain the documentation, type:

```
lib solver(LF)x solv/doc(LF)end / t v  
netout [usc] solv/doc box nnn solver / t v
```

A more robust but more complicated Muller subroutine is MRAF, CACM algorithm 196. Changes suggested by J. Traub and the certification appeared in CACM January 1968 p.12. A FORTRAN translation was made and some remaining glitches were removed and new features added by A.B. Langdon, UC Berkeley February 1968. This subroutine is used in the ROOTS program (see Ref. 2,3). If the demand is sufficient, this latter subroutine may be made available as an option at some future date.

Other root solvers are available on the MFECC CDC-7600; these might be transferred to the CRAY by a user. They include ABEROOT, by C.E. Seyler, which employs a global Newton method, and routines RPZERO, CPZERO for finding roots of real and complex polynomials, developed at LASL (see Ref. 5).

## II. Elementary SOLVER

To run SOLVER, the user enters "solver" followed by some commands from the terminal. In this section, some basic commands are introduced:

### <integer> -

Specify the number of roots desired (defaults to 1 and cannot exceed 1024; this is referred to as n in the rest of this report).

### f [file name] -

Enter the function to be solved. If the function is to be entered from the terminal, the user should wait for the prompt ":" to appear. The name of the function is "f" and the argument is "z". The input format is assumed to be that of FORTRAN statements. In the tty-input mode, input is terminated when "f" is the first non-blank character followed by "=" or a blank, or a slash is the first non-blank character of an input line.

### go [file name or "tty"] -

Compile, load and execute. At execution time, a number of parameters can be set to override the default values. If tty is specified, the user should wait for the prompt "-" to appear. Inputs are in the format of a NAMELIST (terminated by \$).

Some of these parameters are:

- n - Number of roots desired (defaults to <integer>).
- maxit - Maximum number of iterations per root allowed (defaults to 100).
- known - Number of Known roots (defaults to 0).
- h - Initial distance between estimated roots (defaults to 0.5).
- ep1 - Relative error tolerance on z(i) (defaults to 1.e-12 which is also the minimum one can specify).
- ep2 - Error tolerance on f(z(i)) (defaults to 1.e-20 which is also the minimum one can specify).
- of - = 0 => Output results to tty only (default).  
1 => Output results to tty and a disk file.  
2 => Output results to a disk file only.  
3 => Do not output results.  
See Section III for the name of disk file.
- y - Array for initial estimated values and result of roots (defaults to 0).
- debug - = 0 => Drop file deleted after execution (default).  
1 => Drop file retained after execution.

### xex [file name or "tty"] -

Execute the generated program again without recompiling it. See the command "go" for details.

### savef <file name> -

Save the function into the file <file name> which can be read by using the command "f" later.

kbin -

Keep the binary library "solv/b/1" in the active file area.  
If SOLVER is being using often, this step conserves computer effort (the cost of extracting "solv/b/1" from the library file each time is eliminated).

end -

Terminate SOLVER.

#### Example II.1:

Most common syntax

```

user: solver / 2 2
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: >3
routine/user: >f
routine/user: :      f = z**3 - 8.
routine/user: >go
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0065 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : roots:
routine : ( 2.000000000000e+00 , 0.000000000000e-01 )
routine : ( -1.000000000000e+00 , 1.73205080757e+00 )
routine : ( -1.000000000000e+00 , -1.73205080757e+00 )
routine/user: >end
routine : all done

```

#### Example II.2:

All commands on execute line

```

user: solver 20 f go end / 2 3
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: :      f = csin ( z )
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0061 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : roots:
routine : ( 0.000000000000e-01 , 0.000000000000e-01 )
routine : ( -3.14159265359e+00 , 9.33926735795e-21 )
routine : ( 3.14159265359e+00 , -4.28706032686e-36 )

```

```

routine : ( -6.28318530718e+00 , 1.55786371627e-25 )
routine : ( 6.28318530718e+00 , -1.81515384110e-17 )
routine : ( -1.25663706144e+01 , 6.94470731794e-23 )
routine : ( 1.25663706144e+01 , -4.06743060262e-20 )
routine : ( -9.42477796077e+00 , 3.36131166524e-27 )
routine : ( 9.42477796077e+00 , 4.33163059989e-19 )
routine : ( -2.51327412287e+01 , 2.44708281560e-24 )
routine : ( 2.51327412287e+01 , 2.46849267576e-19 )
routine : ( -2.19911485751e+01 , 2.31592585486e-23 )
routine : ( 2.19911485751e+01 , 3.54161507604e-17 )
routine : ( -3.45575191895e+01 , -9.21324826052e-27 )
routine : ( 3.45575191895e+01 , 7.42544852601e-19 )
routine : ( -1.88495559215e+01 , -4.60537034333e-22 )
routine : ( 3.14159265359e+01 , -3.34489216701e-16 )
routine : ( 1.88495559215e+01 , -1.30075028143e-26 )
routine : ( -4.71238898038e+01 , 1.40250535984e-15 )
routine : ( -4.39822971503e+01 , 5.50257927188e-20 )
routine : all done

```

If the user wants roots in some kind of order, and hopefully with none skipped, the user must provide proper initial guesses for the y values. It is also possible to ask the roots to be reordered by the user subroutine (see next section).

### III. Intermediate SOLVER

It is sometimes desirable to find different sets of roots of a function with parameters assuming different values. SOLVER provides this capability by allowing the user to supply a subroutine. The structure of the main program consists of the following:

```
40 continue
    call <the subroutine>
    if ( l(30).lt.0 ) call exit ( debug )
    call <muller to find roots>
    write <results>
    go to 40
```

There are variables which are common to the main program, the function and the user-supplied subroutine for the user to use; they are:

```
integer    l(30)
complex    c(30), p(1024,100)
```

These variables can be read in either from the terminal or from the input file during execution. As the user may have noticed, l(30) is the control variable for looping to find roots. The loop stops when l(30) < 0.

It is possible to "follow" a root as parameters change by careful programming of the option subroutine, but the user must implement this him/herself.

Several I/O units are also available in the subroutine:

- 59 - Terminal (tty).
- 05 - Input (exists only if "go <file name>" or "xeq <file name>" is given - logical unit 5 is connected to the file <file name>).
- 06 - Output (file name is "rout" with the current suffix and a letter appended. Exists only if of=1 or 2).

Libraries such as DISSPLA, TV80LIB, FORTLIB and BASELIB are linked at load time automatically. However, the user may call subroutines from other libraries; SOLVER allows the user to add and/or delete libraries at load time.

The following are additional commands available in SOLVER:

```
sub [file name] -
    Enter the subroutine. All input format and termination
    rules are similar to the command "f" in Section II.
```



saves <file name> -

Save the subroutine into the file <file name> which can be read by using the command "sub" later.

alib [lib name ...] -

Add libraries at load time. This command is terminated by the carriage return key, the line feed key, or the escape key.

dlib [lib name ...] -

Delete libraries. The syntax of this command is the same as that of the command "alib".

<else> -

If the input is the name of an executable file, SOLVER will run it as a controllee. This is useful when using a text editor to modify the function, the subroutine or the input file, or using DDT to debug the program.

In this case, the rest of the input line is passed to the controllee, i.e. any command following will be lost to SOLVER.

#### Example III.1:

```
user: solver / 2 2
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: >3 sub
routine/user: :      data      1(30) /3/, c(1) /64./
routine/user: :c
routine/user: :      1(30) = 1(30) - 1
routine/user: :      if ( 1(30).lt.0 ) return
routine/user: :      c(1) = c(1) / 2.
routine/user: :      write(59,100) c(1)
routine/user: : 100 format ( "c(1) = ", 2f14.6 )
routine/user: :/
routine/user: >f go end
routine/user: :      f = z**3 - c(1)
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0136 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : c(1) = 32.000000 0.000000
routine : roots:
routine : ( 3.17480210394e+00 , 0.00000000000e-01 )
routine : ( -1.58740105197e+00 , 2.74945927400e+00 )
routine : ( -1.58740105197e+00 , -2.74945927400e+00 )
routine : c(1) = 16.000000 0.000000
routine : roots:
routine : ( 2.51984209979e+00 , -4.17757316097e-28 )
routine : ( -1.25992104989e+00 , 2.18224727194e+00 )
```

```

routine      : ( -1.25992104989e+00 , -2.18224727194e+00 )
routine      : c(1) = 8.000000 0.000000
routine      : roots:
routine      : ( 2.00000000000e+00 , 2.44348158944e-46 )
routine      : ( -1.00000000000e+00 , 1.73205080757e+00 )
routine      : ( -1.00000000000e+00 , -1.73205080757e+00 )
routine      : all done

```

#### Example III.2:

Deleting the default graphic libraries which are not used because no plots are made.

```

user: solver / 2 1
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: >dlib disspla tv80lib
routine/user: >2 f go end
routine/user: : f = z**2 - 16.
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0063 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : ***warning-unsatisfied externals***
routine : executing
routine : roots:
routine : ( -4.00000000000e+00 , -2.01948391737e-28 )
routine : ( 4.00000000000e+00 , 6.45718332361e-42 )
routine : all done

```

#### IV. Advanced SOLVER

This section introduces the plotting capability of SOLVER.

##### A. Plotting Roots vs. One Parameter

SOLVER can plot the roots of a function vs. a user defined parameter whose values are to be stored in x, a real array of size 1024. To do so, the user has to set the variable "plotmode" to 1, 2, or 3 depending upon the destination desired for the plot (see below) and must assign values to x in the subroutine (see below). Several parameters are included for the user to set at execution time; they are:

- plotmode - = 0 => No plot (default).
  - 1 => Fr80 file.
  - 2 => Printer file (always called "disout").
  - 3 => Tektronix.If the value read for plotmode is positive, the user can vary the value of plotmode between 0 and a positive integer dynamically. The user assigns a positive number to plotmode only for those roots he/she wants to plot. The number of abscissa values plotted is the number of times the plotmode is found to be positive upon return from the subroutine.
- plotid - Depending on the destination, plotid has different interpretations. If plotmode is
  - 1 (fr80 file) => Name of this plot in not more than 8 characters.
  - 2 (printer file) => Number of characters per line (integer).
  - 3 (tektronix) => The baud rate (integer).
- id - In less than 30 characters, this is the box and id.
- mapset - = 0 => A linear-linear plot (default).
  - 1 => A linear-log plot (x linear, y log).
  - 2 => A log-linear plot (y linear, x log).
  - 3 => A log-log plot.
- xsize - The length (in inches) of the x-axis (defaults to 6.0). DISSPLA conventions for "inches" are employed, i.e. the "page" is assumed to be 8.5x11 inches.
- ysize - The length (in inches) of the y-axis (defaults to 4.5).
- grids - = 0 => No grid (default).
  - 1 => Grid.
- x - The user can also input the values of x from tty or from the input file.

The program structure now looks as follows:

```

40 continue
   call <the subroutine>
   if ( not the first time and plotmode > 0 ) call <store y>
   if ( 1(30).lt.0 ) go to 70
   call <muller to find roots>
   write <result>
   go to 40
70 continue
   if ( plot desired ) call <plot>
   call exit ( debug )

```

The graphics library used is DISSPLA. All warning or error messages, if any, will be sent to the file named "disout". The maximum number of roots which can be plotted is 25; however, since DISSPLA has only 15 different symbols to distinguish different curves, the user is advised not to plot more than 15 roots.

The plotted output includes:

- (1) The contents of the input file, if there is one.
- (2) The real part of the roots vs. the parameter (x).
- (3) The imaginary part of the roots vs. the parameter (x).

The size of a page is assumed to be 8.5"x11", regardless of output medium selected. If possible, SOLVER will place items (2) and (3) on one page. The maximum size of graphs to be plotted two-to-a-page is 6"x4.5" (or 4.5"x6" for side-by-side plots).

For an fr80 file, the user must do either of the following after finding out the name of the fr80 file:

- (1) netplot [usc] <fr80 file> [1.] [turn.] / t v or
  - (2) netout a <fr80 file> b. / t v
- (then log-off the CRAY-1 and log-on to the 7600 and view the plot file on a tektronix by FR80PLOT).

These operations (without the time and value) can be done on the same suffix under which SOLVER is running.

Example IV.A.1 (see Fig. IV.A.1):

A "back-door" way to plot  $\sin(x)$  using SOLVER

```

user: solver
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/22/79 20:19:41 001222
routine/user: >sub
routine/user: : equivalence ( ixind,1(29) )
routine/user: : data 1(30) /21/, ixind /0/
routine/user: : data pi /3.1415926535/
routine/user: :c

```

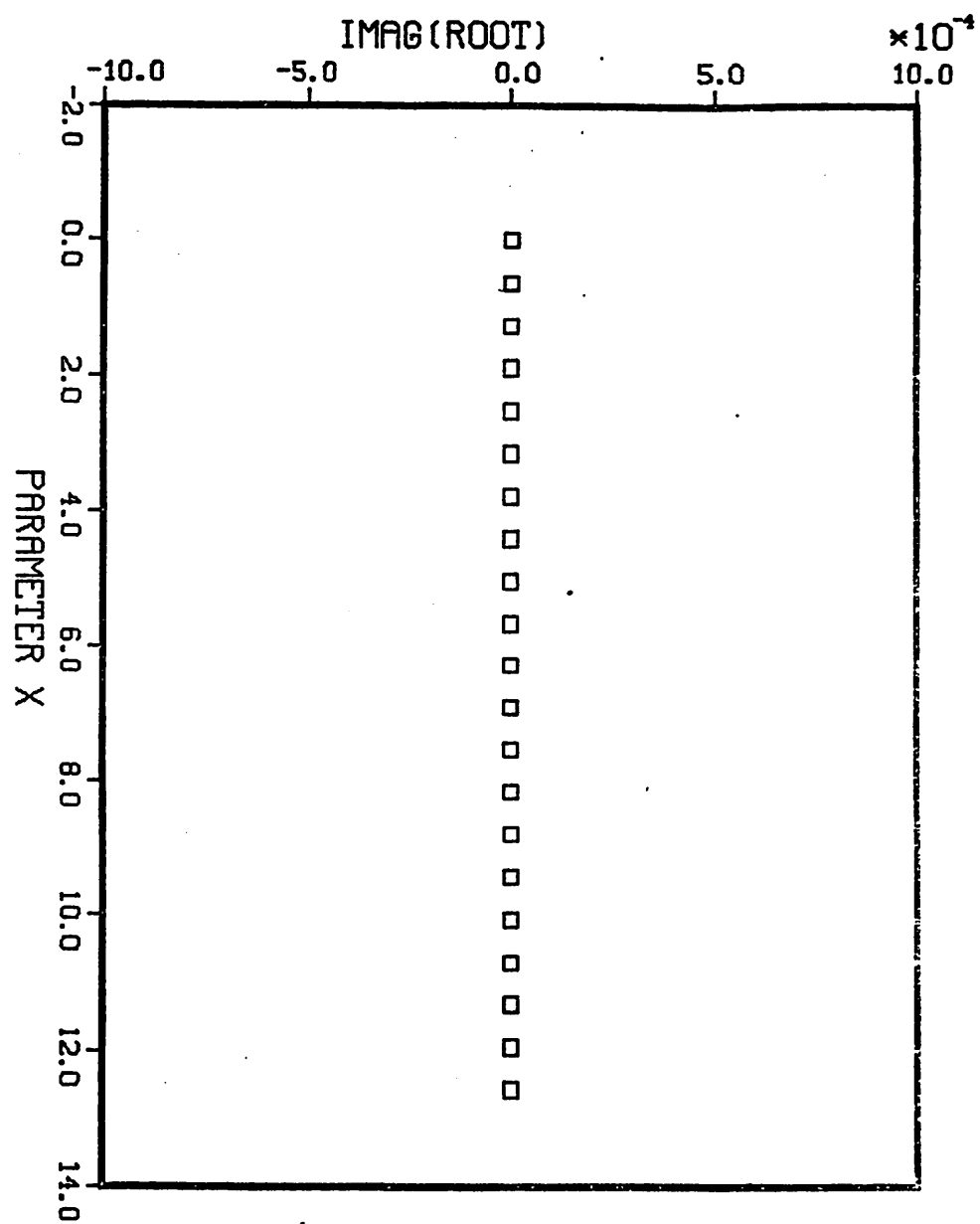
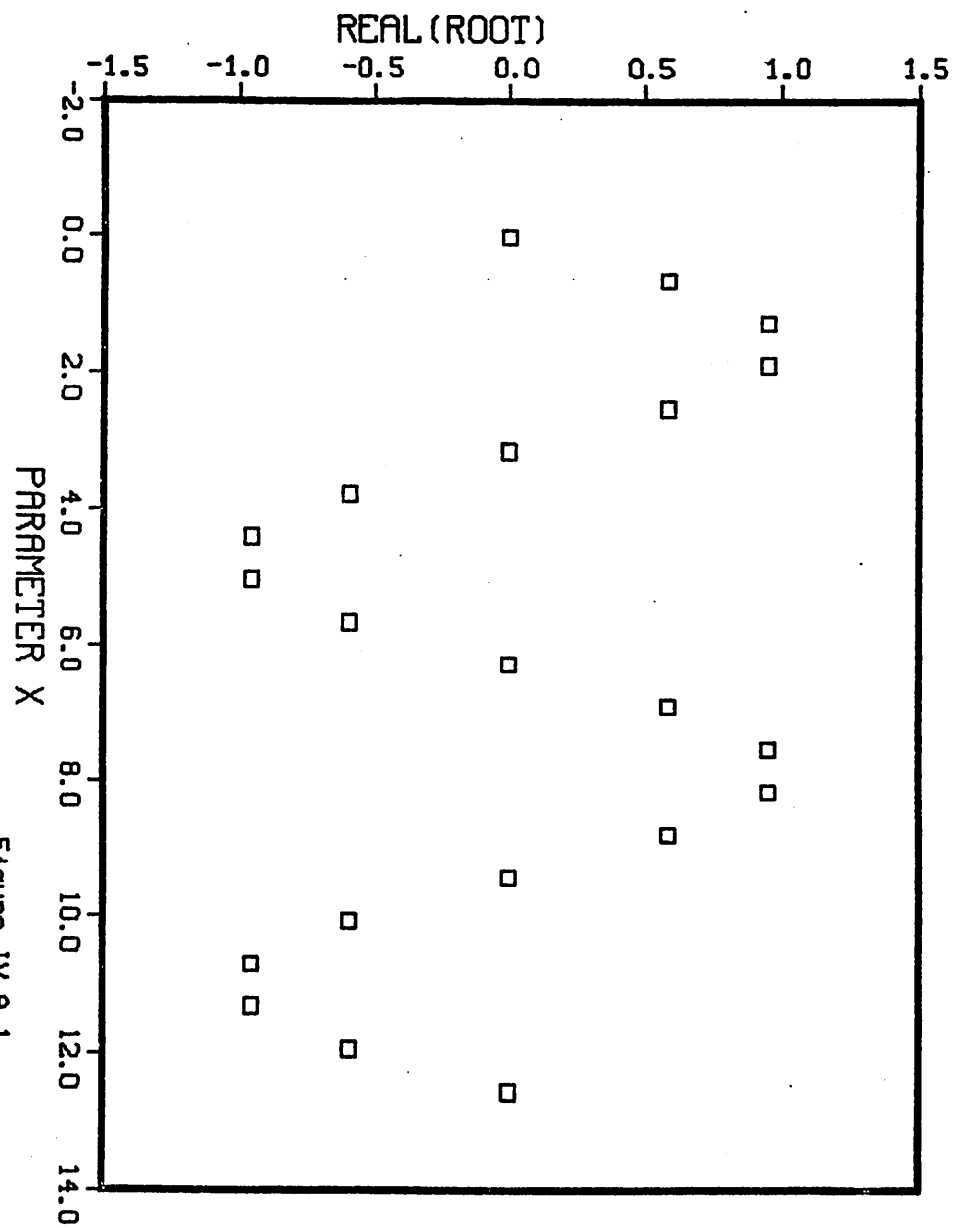


Figure IV.A.1

```

routine/user: :      1(30) = 1(30) - 1
routine/user: :      ixind = ixind + 1
routine/user: :      x(ixind) = .2 * pi * ( ixind-1 )
routine/user: :/
routine/user: >f
routine/user: :      equivalence ( ixind,1(29) )
routine/user: :c
routine/user: :      f = z - sin ( x(ixind) )
routine/user: >go tty end
routine      : compiling and loading
routine      : FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      : FT001 - COMPILE TIME =   0.0130 SECONDS
routine      : *** cray loader version - c121  07/05/79
routine      : executing
routine/user: -plotmode=1 $
routine      : roots:
routine      : ( 0.000000000000e-01 , 0.000000000000e-01 )
routine      : roots:
routine      : ( 5.87785252278e-01 , 0.000000000000e-01 )
routine      : roots:
routine      : ( 9.51056516284e-01 , 0.000000000000e-01 )
routine      : roots:
routine      : ( 9.51056516312e-01 , 0.000000000000e-01 )
routine      : roots:
routine      : ( 5.87785252351e-01 , 0.000000000000e-01 )
routine      : roots:
routine      : ( 8.97949606966e-11 , -1.89326617253e-29 )
routine      : roots:
routine      : ( -5.87785252205e-01 , 6.31088724177e-30 )
routine      : roots:
routine      : ( -9.51056516256e-01 , -2.24207754292e-44 )
routine      : roots:
routine      : ( -9.51056516340e-01 , -4.08416990527e-53 )
routine      : roots:
routine      : ( -5.87785252423e-01 , -4.21687917729e-81 )
routine      : roots:
routine      : ( -1.79589921393e-10 , 4.25795984001-109 )
routine      : roots:
routine      : ( 5.87785252133e-01 , -1.51273121674-123 )
routine      : roots:
routine      : ( 9.51056516229e-01 , 1.07486017721-137 )
routine      : roots:
routine      : ( 9.51056516367e-01 , 1.58613441594-146 )
routine      : roots:
routine      : ( 5.87785252496e-01 , 7.11282799835-161 )
routine      : roots:
routine      : ( 2.69356460379e-10 , -1.26217744835e-29 )
routine      : roots:
routine      : ( -5.87785252060e-01 , -1.26217744835e-29 )
routine      : roots:
routine      : ( -9.51056516201e-01 , -1.27447352891e-57 )
routine      : roots:
routine      : ( -9.51056516395e-01 , 1.35835186182e-71 )
routine      : roots:

```

```

routine      :      ( -5.87785252569e-01 ,   0.00000000000e-01 )
routine      :      roots:
routine      :      ( -3.59179842786e-10 ,   0.00000000000e-01 )
routine      :      all done

```

Some users may prefer the following equivalent forms for sub and f:

```

(1) sub -
    data      1(30) /21/, pi /3.1415926535/
    1(30) = 1(30) - 1
    x(21-1(30)) = .2 * pi * ( 20-1(30) )
(2) f -
    f = z - sin ( x(21-1(30)) )

```

The user may specify "of=3" to suppress the listing of roots.

Example IV.A.2 (see Fig. IV.A.2):

Another "back-door" function plot, on log-log scales this time

```

user: solver sub f go tty end
routine      :      *** root solver 79.08 ***
routine      :      extracting solv/b/1
routine      :      c 07/22/79 20:19:41 001222
routine/user: :c note that we want a log plot, so we do not
routine/user: :c plot points with negative values of the
routine/user: :c parameter.
routine/user: :c
routine/user: :      equivalence ( ixind,1(29) ), ( xnext,1(28) )
routine/user: :      data      1(30) /11/, ixind /-6/
routine/user: :c
routine/user: :      1(30) = 1(30) - 1
routine/user: :      plotmode = ixind
routine/user: :      if ( ixind.gt.0 ) x(ixind) = xnext
routine/user: :      ixind = ixind + 1
routine/user: :      xnext = ixind**3
routine/user: :/
routine/user: :      equivalence ( xnext,1(28) )
routine/user: :c
routine/user: :      f = z - xnext
routine      :      compiling and loading
routine      :      FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      :      FT001 - COMPILE TIME =   0.0153 SECONDS
routine      :      *** cray loader version - c121 07/05/79
routine      :      executing
routine/user: : -plotmode=1 mapset=3 xsize=4.5 ysize=6.0
routine/user: : -plotid="test" id="b22" "stephen au-yeung" $
routine      :      roots:
routine      :      ( -1.25000000000e+02 ,   1.37001788954e-24 )
routine      :      roots:
routine      :      ( -6.40000000000e+01 ,   -8.01829413028e-50 )
routine      :      roots:
routine      :      ( -2.70000000000e+01 ,   -3.11734638189e-67 )
routine      :      roots:

```

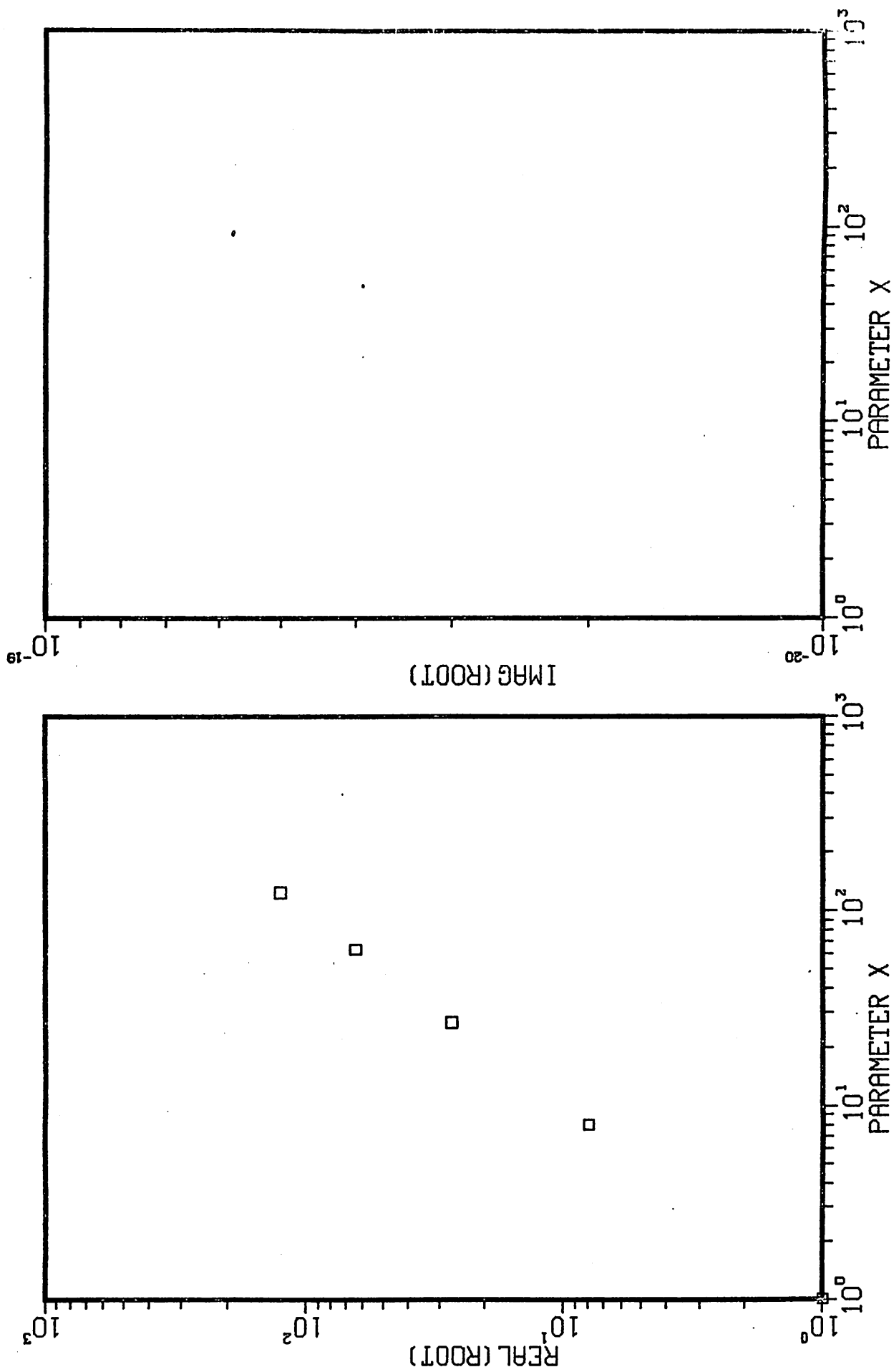


Figure IV.A.2



```

routine      :      ( -8.00000000000e+00 ,   7.67045853953e-93 )
routine      :      roots:
routine      :      ( -1.00000000000e+00 ,   6.16896886431-110 )
routine      :      roots:
routine      :      (   3.02922587605e-28 ,  -1.61229026582-137 )
routine      :      roots:
routine      :      (   1.00000000000e+00 ,   2.17066284129-165 )
routine      :      roots:
routine      :      (   8.00000000000e+00 ,  -1.75344747921-191 )
routine      :      roots:
routine      :      (   2.70000000000e+01 ,  -3.39941662484-218 )
routine      :      roots:
routine      :      (   6.40000000000e+01 ,   5.63600517340-231 )
routine      :      roots:
routine      :      (   1.25000000000e+02 ,  -3.99599503859-256 )
routine      :      all done
user:        files f
routine      :      1375 rwe f105ro0x
routine      :      all done
user:        netplot usc f105ro0x 1. box b22 steve
routine      :      destination:   ucb
routine      :      4 frames of output processed
routine      :      f105ro0x ; file id is: steve - rxucb/rz
routine      :      all done

```

Example IV.A.3 (see Fig. IV.A.3), by Y.J. Chen:

In a 1d particle-hybrid simulation of the lower-hybrid drift instability (LHDI) in a uniform magnetic field, the quiet start Maxwellian loader was used to set up the warm, unmagnetized ion particles. Dispersion relations for the LHDI and the multi-beam instabilities were obtained from SOLVER. Initial guesses for roots were made in order to determine the approximate frequencies at which the desired roots should be obtained.

```

user: solver alib lhdmbi(LF)f disper sub mbilhd go imbilhd
routine      : *** root solver 79.08 ***
routine      : extracting solv/b/1
routine      : c 07/22/79 20:19:41 001222
routine      : compiling and loading
routine      : FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      : FT001 - COMPILE TIME =   0.0129 SECONDS
routine      : *** cray loader version - c121 07/05/79
routine      : executing
routine      : routba is the output file
routine/user: >netplot f105ro0x 1. box b11 yjchen
routine      : destination:   ucb
routine      :      6 frames of output processed
routine      : f105ro0x ; file id is: yjchen - rxucb/1a
routine/user: >end
routine      :      all done

```

The contents of the various files are:

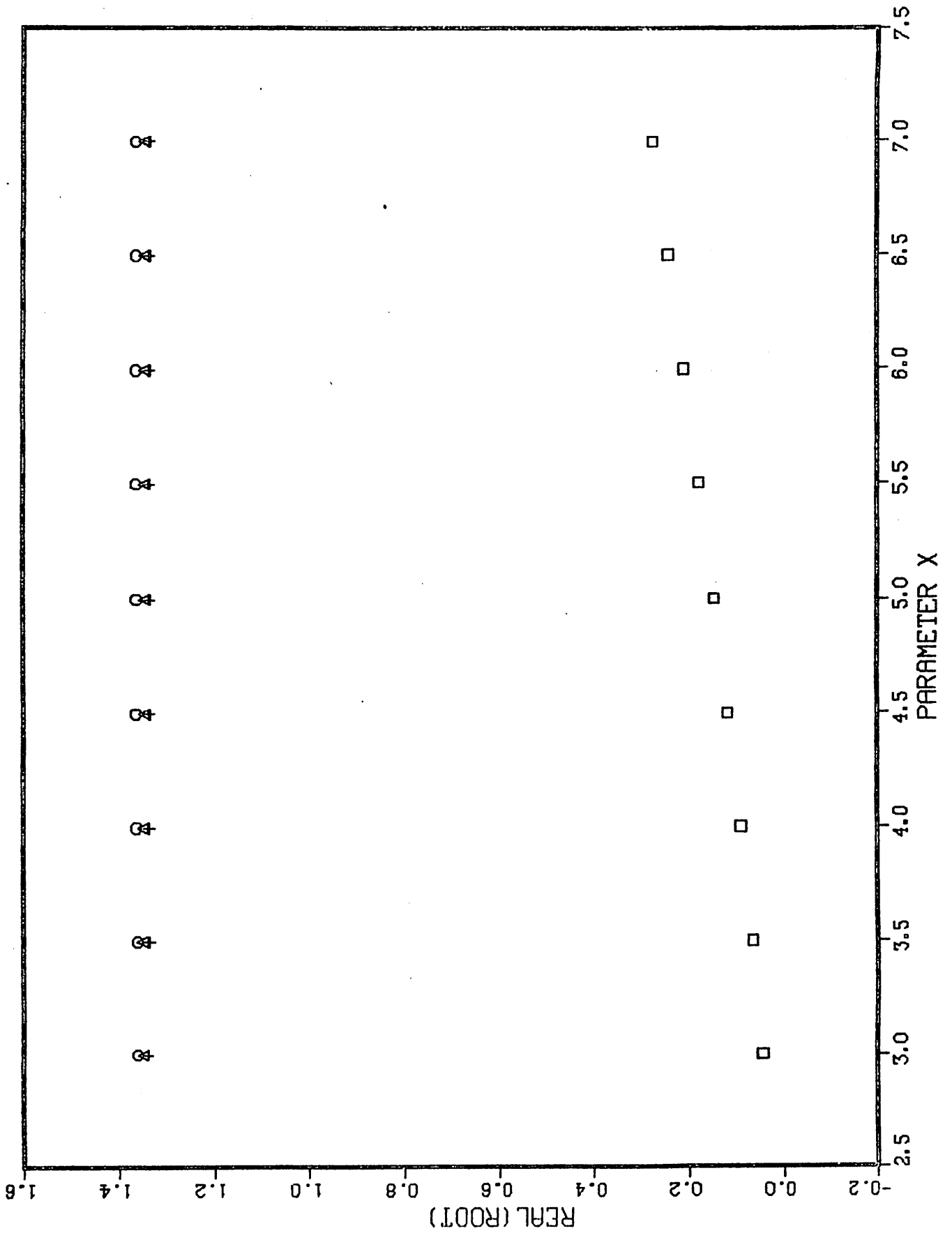


Figure IV.A.3 (a)

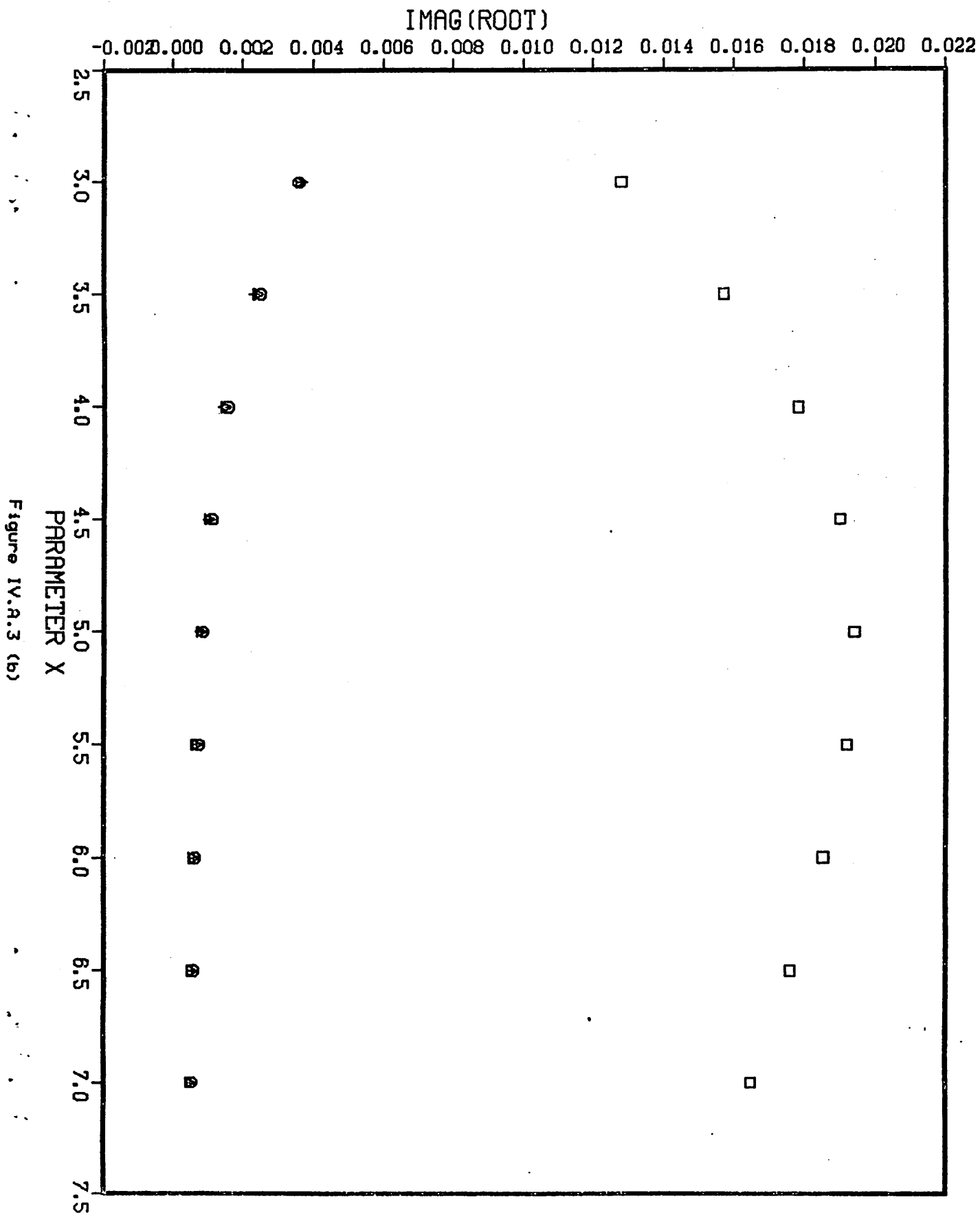


Figure IV.A.3 (b)

```

(1) disper -
    real v(1),lambda
    complex fsum(1)
    equivalence (v(1),p(1,33)), (fsum(1),p(1,49))
    equivalence (xe1,c(8)), (xe2w,c(9)), (vek,c(10))
    equivalence (ngr,l(2))
c
    fsum(1)=-ngr*(1.+xe1+xe2w/(z-vek))
    ngr2=ngr/2
    do 10 i=2,ngr2+1
        fsum(i)=fsum(i-1)+1./((z-v(i-1))*(z-v(i-1)))
10 continue
    do 20 i=ngr2+2,ngr+1
        fsum(i)=fsum(i-1)+1./((z-v(i-1))*(z-v(i-1)))
20 continue
    f=fsum(ngr+1)

(2) mbilhd -
c appears as the subroutine "uvsetup" in the "solver".
c calculate the dispersion of the lower hybrid drift instability
c including effects of the multibeam instability due to the quiet
c start.
    real xx(1),vx(1),v(1),k,kmin,klast
    real lni, lti, lsi
    real lambda
    equivalence ( xx(1),p(1,1) )
    equivalence ( vx(1),p(1,17) )
    equivalence ( v(1),p(1,33) )
    equivalence ( vti,c(1) )
    equivalence ( vd,c(3) )
    equivalence ( wpc,c(4) )
    equivalence ( vte,c(5) )
    equivalence ( ve,c(6) )
    equivalence ( rm,c(7) )
    equivalence ( xe1,c(8) )
    equivalence ( xe2w,c(9) )
    equivalence ( vek,c(10) )
    equivalence ( b,c(11) )
    equivalence ( dk,c(12) )
    equivalence ( kmin,c(13) )
    equivalence ( klast,c(14) )
    equivalence ( lni,c(15) )
    equivalence ( lti,c(16) )
    equivalence ( lsi,c(17) )
    equivalence ( re,c(18) )
    equivalence ( nn,l(1) )
    equivalence ( ngr,l(2) )
    equivalence ( npts,l(27) )
c
    if( l(29).ne.0 ) go to 28
    l(30)=(klast-kmin)/dk+1.5
    npts = l(30)
    write(6,1) nn,ngr,vti,wpc,vte,rm,dk,kmin,klast,
        lni,lti,lsi

```

```

1 format(//,"ion:",//," nn=",i15," ngr=",i15," vti=",e14.5,/,
.      "electron:",//,
.      " wpc=",e14.5," vte=",e15.5,/,
.      " mass ratio, rm=",e22.5,///,
.      " dk=",e15.5," kmin=",e14.5," klast=",e12.5,/,
.      " lni=",e14.5," lti=",e15.5," lsi=",e15.5,///)
c
c set up the multibeam
vmax=5.*vti
dv=vmax/(nn-1)
xx(1)=0.
do 10 i=2,nn
vv=(i-1.5)*dv
fv=exp(-.5*(vv/vti)**2)
10 xx(i)=xx(i-1)+fv
c
df=xx(nn)/ngr
j=1
i1=ngr/2+1
i2=i1-1
do 12 i=1,ngr,2
fv=i*df
13 if( fv.lt.xx(j+1) ) go to 14
j=j+1
if( j.gt.nn-1 ) go to 25
go to 13
14 vv=dv*( j-1+( fv-xx(j) )/( xx(j+1)-xx(j) ) )
vx(i1)=vv
vx(i2)=-vv
i1=i1+1
12 i2=i2-1
25 continue
c
ngr2=ngr/2
c change vte to the electron larmor radius
27 continue
wcei=sqrt(wpc/rm)
wcii=wcei*rm
re=vte*wcei
ve=vti*vti*(lni+lti)*wcii
write(6,100)wcii,wcei,re,ve
100 format(" wcii=",e13.5," wcei=",e13.5," re=",e16.5,/,
.      " ve=",e15.5,///)
28 continue
l(29) = l(29) + 1
l(30) = l(30) - 1
if ( l(30).lt.0 ) go to 36
c
c calculate coefficients of the dispersion relation
k=klast-l(30)*dk
write(6,29) k
29 format(//,"k=",e12.5)
x(l(29)) = k
do 30 i=1,ngr

```

```

      v(i)=vx(i)*k
30  continue
      vk=vk*k
      bk=vte*k
      b=bk*bk
      mm=n
c
      if( b.eq.0. ) go to 32
      bi0=bessei(0,b)
      bi1=bessei(1,b)
      lambda=bi0*exp(-b)
      a=( 1.-lambda )/b
      fle=b*( 1.-bi1/bi0 )
      go to 33
32  lambda=1.0
      a=1.0
      fle=0.0
33  xel=a*wpc
      rli=lni - lsi*(1.-fle) - lti*fle
      xe2w=wpc*lambda*rli/(k*wcei)
c  guess the roots for the first time only
      if ( l(29).gt.1 ) go to 36
c
      j=2
35  continue
      if ( j.gt.min0(ngr2,mm) ) return
      j1=ngr+1-j-10
      y(j)=v(j1)+cmplx(0.,.001)
      j=j+1
      go to 35
c
36  continue
      do 40 j=1,mm
      y(j)=cmplx(real(y(j)),abs(aimag(y(j))))
40  continue

(3) imbilhd -
n=4 maxit=300 h=.00002 of=2
l(1)=16384 l(2)=16384
c(1)=.141421 c(3)=0.0 c(4)=1.0
c(5)=0.00 c(6)=0.0 c(7)=1600.0 c(12)=0.5
c(13)=3.0 c(14)=7.0
c(15)=.075 c(16)=0.0 c(17)=0.0
y(1)=(.002,.001)
plotmode=1 plotid="lhdi" id="box b11" "y j chen"
xsize=9. ysize=6.75
$
l(1)=nn, l(2)=ngr for giving y(j) in input file
c(1)=vt, c(3)=vd, c(4)=wpc
c(5)=vte, c(6)=ve, c(7)=rm, c(12)=dk
c(13)=kmin, c(14)=klast
c(15)=lni, c(16)=lti, c(17)=lsi

```

## B. Plotting a Function vs. One Argument

SOLVER can also plot a function vs.  $x$ , the real array described in Section IV.A. To tell SOLVER that the user wants a plot of the function rather than the roots of the function, the user need only assign the value 4, 5, or 6 to plotmode from the terminal or from the input file. The destination of the output when plotmode equals 4, 5, or 6 is the same as that when plotmode equals 1, 2, or 3, respectively. All other variables but "of" retain the same meaning and usage. The only output is a plot in this case.

Also note that within the function  $f$ ,  $z$  is equivalent to  $x(i)$  where  $i$  corresponds to the number of times the function has been called. The structure can be viewed as follows:

```
40 continue
   call <the subroutine>
   if ( not the first time ) call <store y(1)>
   if ( l(30).lt.0 ) go to 70
   i = i + 1
   y(1) = f ( x(i) )
   go to 40
70 continue
   call <plot>
   call exit ( debug )
```

Example IV.B.1 (see Fig. IV.B.1):

```
user: solver f f1/s sub s1/s go g1/d end
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/22/79 20:19:41 001222
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0129 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : all done
```

The contents of the various files are:

```
(1) f1/s -
   equivalence ( ixind,1(29) )
c
   f = sin ( x(ixind) )
c
c the above lines for the function f are equivalent
c to the following alternative line
c   f = csin ( z )
```

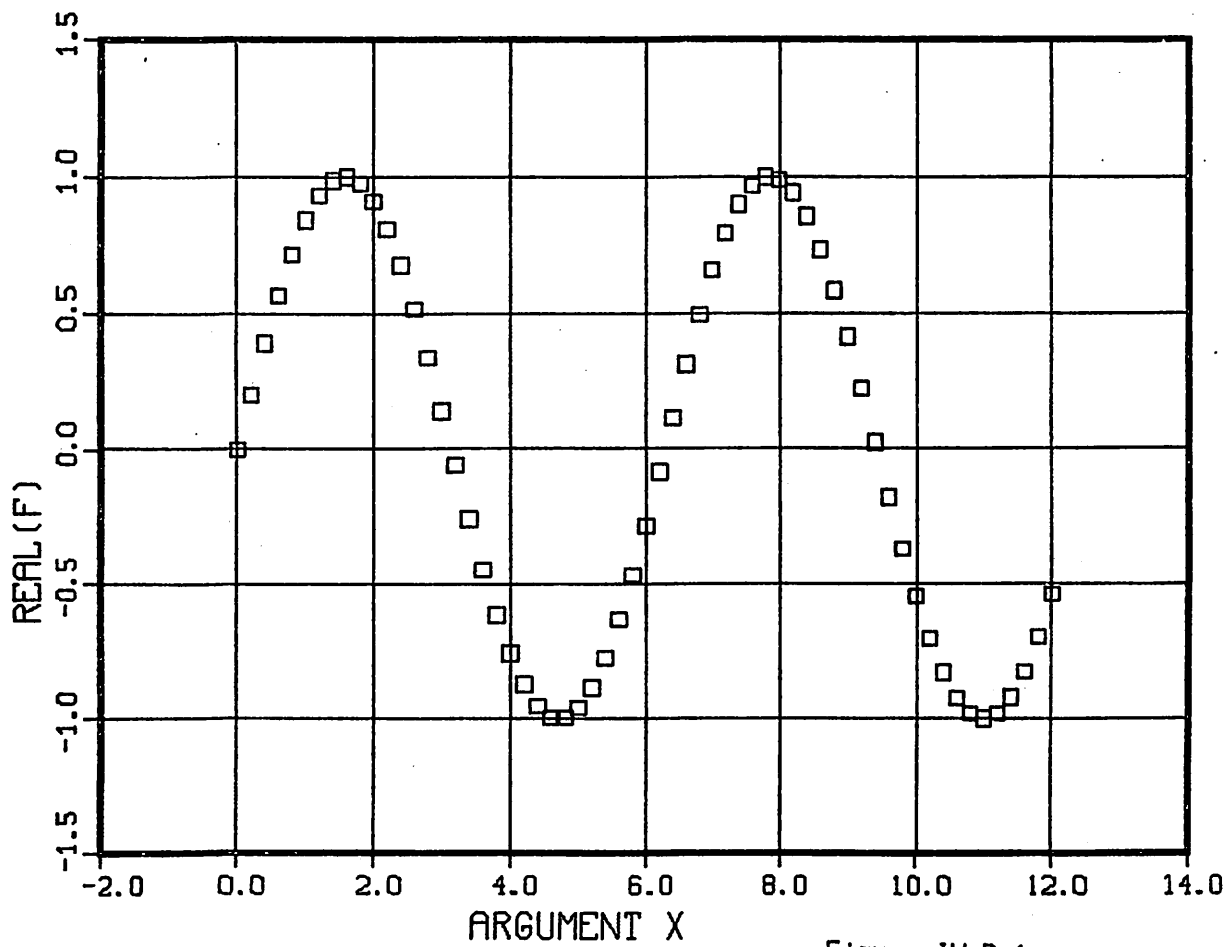
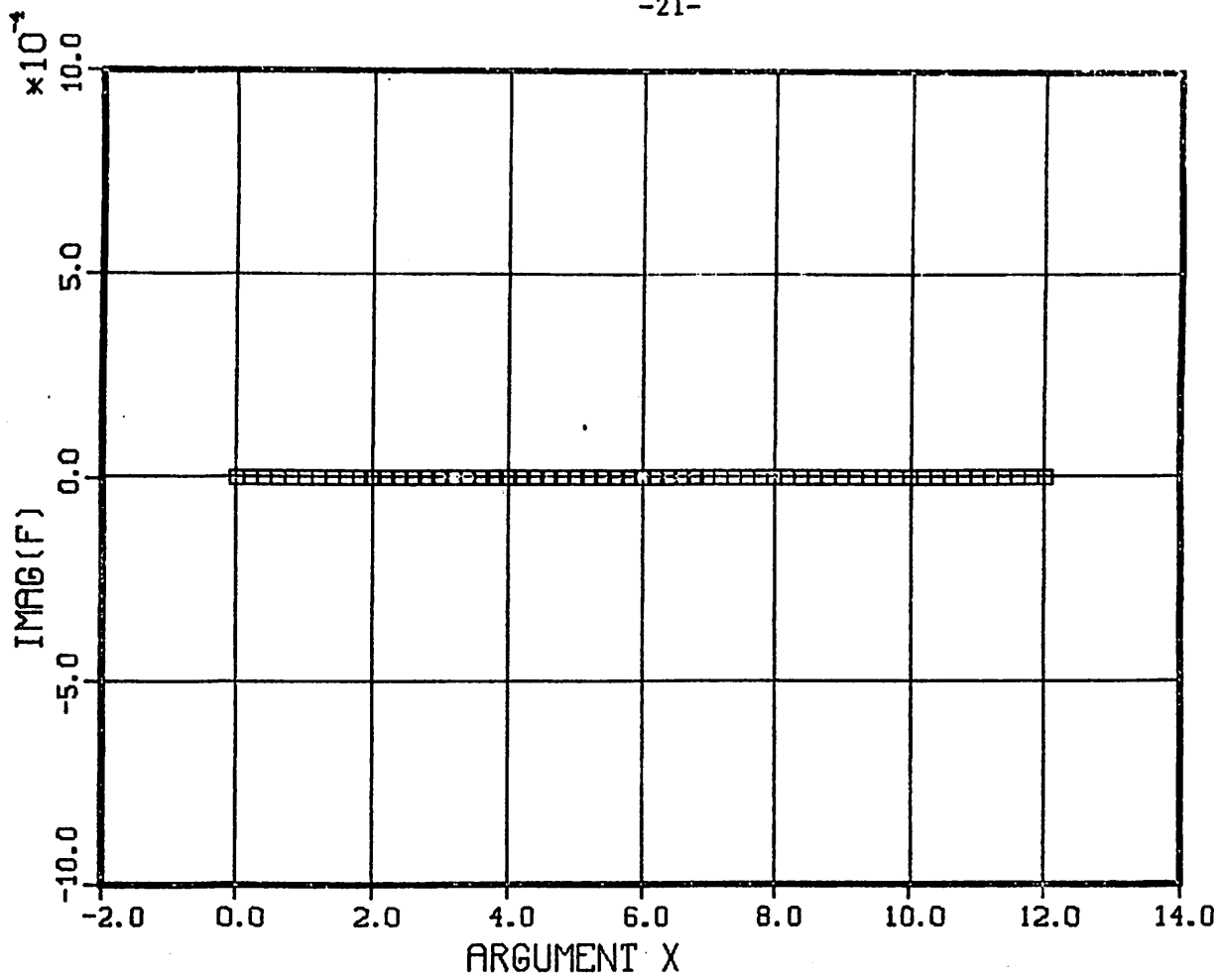


Figure IV.B.1



```

(2) s1/s -
    equivalence ( ixind,1(29) )
    data      1(30) /21/, ixind /0/
c
    1(30) = 1(30) - 1
    ixind = ixind + 1
    x(ixind) = .2 * (ixind-1 )

(3) g1/d -
1(30)=61
plotmode=4 grids=1
of=3
$

```

### C. Plotting Roots (a Function) vs. Two Parameters (Arguments)

A pair of contour plots will be produced in this case. The second parameter (argument) is to be stored in w, a real array of size 1024. The user should store values into x and w in the subroutine. In addition to x and w, two more variables have to be set at execution time; they are:

nx - Number of elements in x (defaults to 1024).

nw - Number of elements in w (defaults to 25).

If the product of nx and nw is greater than  $1024 \times 25 = 25600$ , nx and nw will be set back to their default values. It is important that w be the more rapidly varying parameter of the two (x is the slowly varying parameter). This point will become clear upon examination of the examples which follow.

#### Example IV.C.1 (see Fig. IV.C.1):

In this example, the fastest growing "tilting mode" root of a weak ion ring - plasma system is identified and contours of its real frequency and growth rate normalized to the ion cyclotron frequency are plotted as functions of two parameters. These are the ring strength "qr" and the background plasma density "zni". Logarithmic scales are used, and (in the stable region at the lower right) the frequency is set to zero when the growth rate is essentially zero, since there is then no "fastest growing mode". Note that the roots are reordered since the contour plotter can only plot one root, and we have not arranged to "follow" individual roots as parameters change. The constant "cay" depends upon the differencing scheme being used. (A. Friedman, unpublished)

```

user: solver f f6/s sub s6/s go g6/d
routine : *** root solver 79.08 ***
routine : extracting solv/b/1

```

REAL (ROOT)

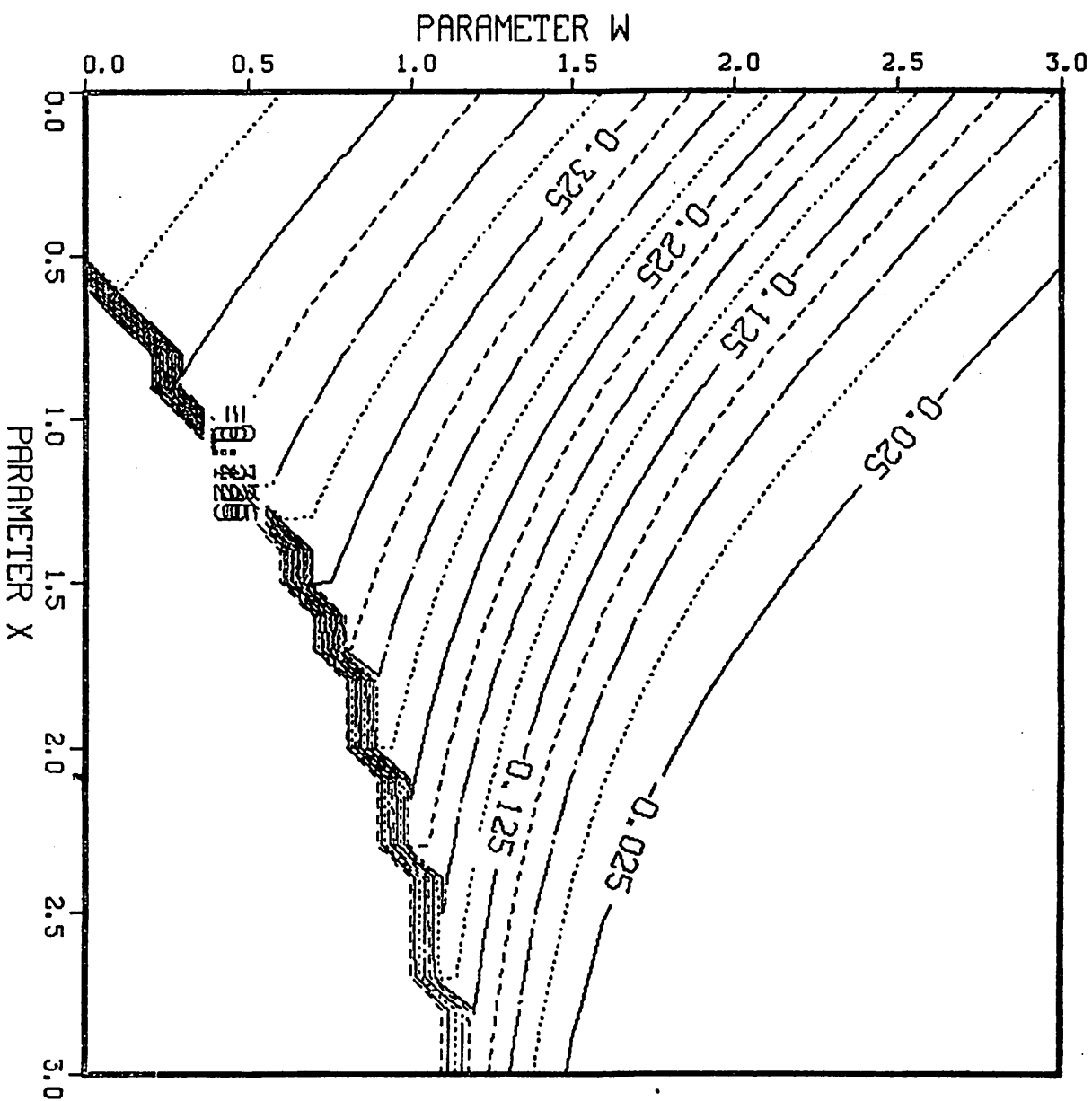


Figure IV.C.1 (a)

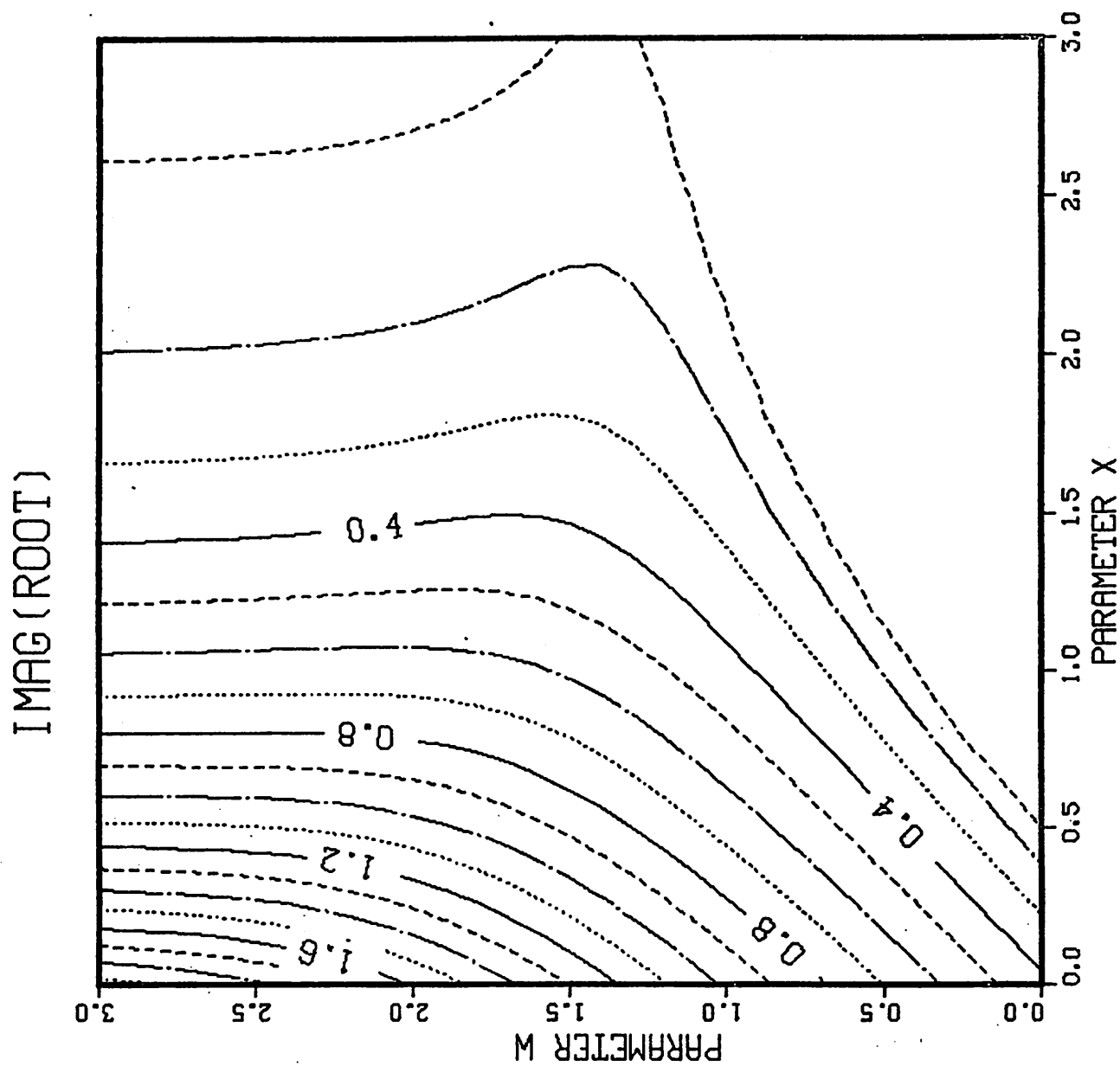


Figure IV.C.1 (b)

```

routine      : c 07/28/79 11:21:15 001222
routine      : compiling and loading
routine      : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine      : FT001 - COMPILE TIME = 0.0245 SECONDS
routine      : *** cray loader version - c121 07/05/79
routine      : executing
routine/user: >netplot f105ro0x 1.
routine      : box number: b22
routine      : destination: ucb
routine      : 6 frames of output processed
routine      : f105ro0x ; file id is: f105ro0x - rxucb/4K
routine/user: >end
routine      : all done

```

The contents of the various files are:

```

(1) f6/s -
    equivalence ( qr,1(26) ), ( cay,1(27) ), ( zni,1(28) )
c
    f = z * z * ( 1.+z )**2 - qr * ( z*z/8.+2.*cay/zni )

(2) s6/s -
    equivalence ( qr,1(26) ), ( cay,1(27) ), ( zni,1(28) )
    equivalence ( ipoint,1(29) )
    data      cay/.25/, ipoint/0/, 1(30)/-1000/
c
    if ( 1(30).eq.-1000 ) 1(30) = nx * nw
    1(30) = 1(30) - 1
    ipoint = ipoint + 1
c
    ixind = ( ipoint-1 ) / nx + 1
    znilog = ( ixind-1 ) / 10.
    zni = 10.**znilog
    x(ixind) = znilog
c
    iwind = mod ( ipoint-1, nw ) + 1
    qrlog = ( iwind-1 ) / 10.
    qr = .36 * 10.**qrlog
    w(iwind) = qrlog
c
c sort the roots so that the fastest growing mode appears first
    n = 4
    c(1) = y(1)
    do 2 iroot = 2, 4
        if ( aimag(y(iroot)).le.aimag(c(1)) ) go to 2
        c(1) = y(iroot)
        y(iroot) = y(1)
        y(1) = c(1)
    2 continue
c
c set root to zero if growth rate is below a reasonable threshold
    if ( aimag(y(1)).lt.0.01 ) y(1) = 0.

(3) g6/d -
nx=31 nw=31
plotmode=1 mapset=4
xsize=6. ysize=6.
of=3
$

```

Example IV.C.2 (see Fig. IV.C.2):

```

      user: solver f f5/s sub s5/s go g5/d
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/28/79 11:21:15 001222
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPILE TIME = 0.0245 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine/user: >netplot f105ro0x 1.
routine : box number: b22
routine : destination: ucb
routine : 6 frames of output processed
routine : f105ro0x ; file id is: f105ro0x - rxucb/\;
routine/user: >end
routine : all done

```

The contents of the various files are:

```

(1) f5/s -
      equivalence ( ixind,1(29) ), ( iwind,1(28) )
c
      f = 0
      rsq = ( abs(x(ixind))- .5 )**2 + .25*w(iwind)**2
      if ( rsq.ge.0.25 ) return
      f = cmplx ( sqrt(.25-rsq), sqrt(rsq) )
      if ( x(ixind).gt.0. ) f = -f

(2) s5/s -
      equivalence ( ixind,1(29) ), ( iwind,1(28) )
      equivalence ( denx,1(27) ), ( denw,1(26) )
      data      1(30)/-1000/, ixind/0/, iwind/25600/
c
      if ( 1(30).ne.-1000 ) go to 10
      1(30) = nx * nw
      denx = nx / 2
      denw = nw / 2
c
      10 continue
      1(30) = 1(30) - 1
      if ( iwind.lt.nw ) go to 20
      ixind = ixind + 1
      x(ixind) = 1.2 * ( ixind-1-denx ) / denx
      iwind = 0
      20 continue
      iwind = iwind + 1
      w(iwind) = 1.2 * ( iwind-1-denw ) / denw

(3) g5/d -
      plotmode=4
      mapset=4
      nx=81 nw=81
      xsize=7. ysize=7.
      of=3
      $

```



IMAG(F)

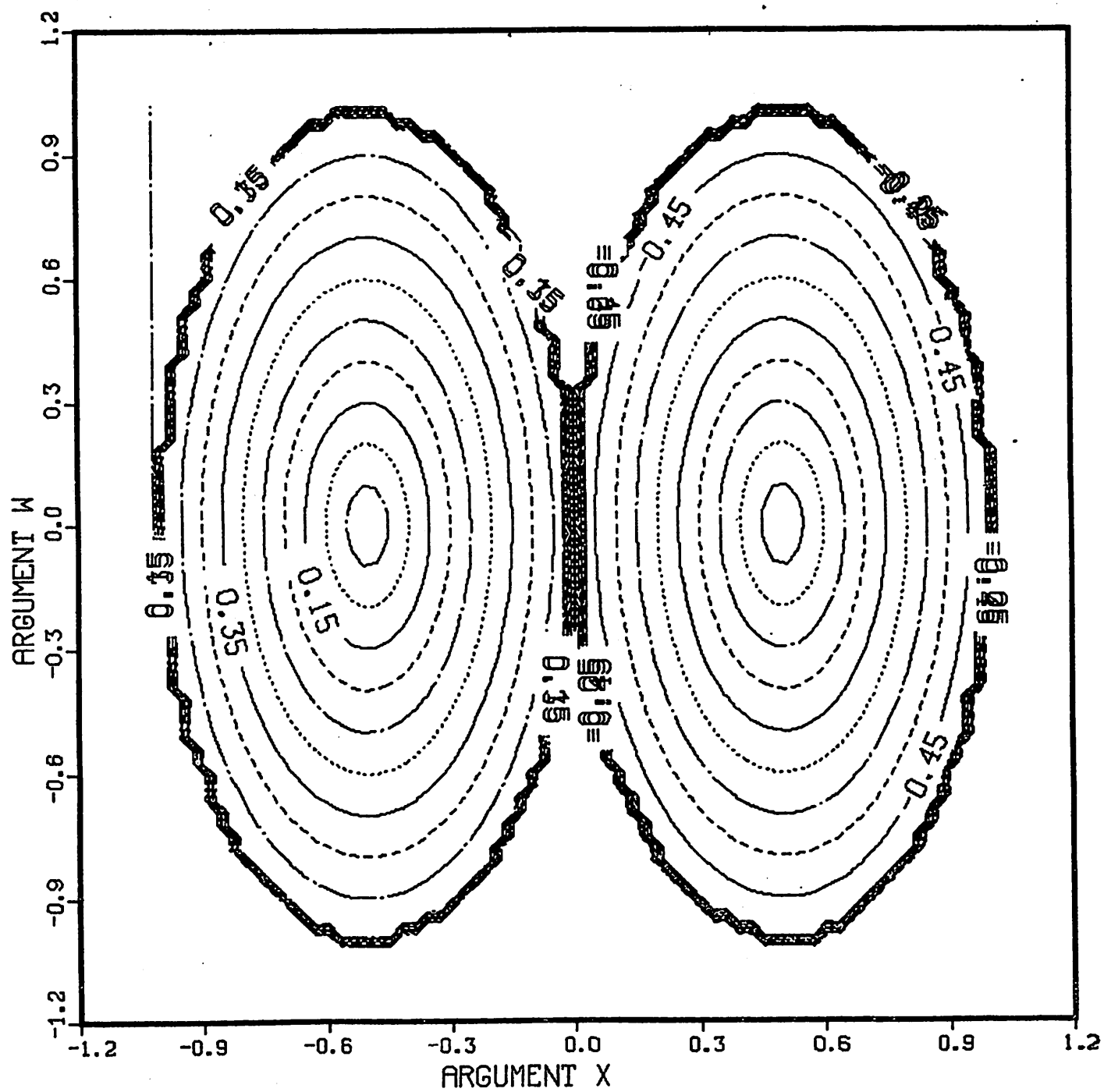


Figure IV.C.2 (b)

## References

1. S.D. Conte and C. de Boor, "Elementary Numerical Analysis: An Algorithm Approach," Ed. 2, pp. 74-83, McGraw-Hill, New York (1972).
2. M.J. Gerver, C.K. Birdsall, A.B. Langdon and D. Fuss, "Normal Modes of A Loss Cone Plasma Slab with Steep Density Gradient," Memorandum No. ERL-M502, Electronics Research Laboratory, College of Engineering, UC Berkeley, (24 September 1976). See Appendix I for a description and listing of ROOTS which includes MRAF and FOLLOW, which traces roots, including branching.
3. M.J. Gerver, "Roots, A Dispersion Equation Solver," Memorandum No. ERL-M77-27, Electronics Research Laboratory, College of Engineering, UC Berkeley (31 October 1976). This report complements Ref. 2 and contains instructions to the user of ROOTS (MRAF, FOLLOW, etc.).
4. "DISSPLA User's Manual," Version 8.0, ISSCO, San Diego (1978).
5. "Third Quarter Progress Report on Plasma Theory and Simulation," ERDA Contract EY-76-S-03-0034, Project Agreement No. 128, Electronics Research Laboratory, College of Engineering, UC Berkeley (1 October 1977).



## APPENDIX 1. The Structure of The Library Solver

As mentioned in Section I, SOLVER is a library that contains all files needed by the SOLVER program.

Entering "solver / t v" from the terminal causes "solver/x", the first entry in SOLVER, to be executed. A dropfile named +solver<s>, where <s> is the current suffix (channel) that SOLVER is running under, is created. The next step is extracting the binary library "solv/b/1" from "solver" (if "solv/b/1" is not yet in the user's active file area). The prompt ">" will then appear and the user proceeds to enter commands.

The rest of the files in the SOLVER library are: the latest version of this documentation and the source files. Some of these files are part of the SOLVER controller and the rest are used in the generated program. A list of routines and corresponding source files follows:

### (1) The SOLVER controller program -

file name	routine name
-----	-----
solver/s	main. (the main program)
cmds/s	iscmd
digit/s	digit
fexist/s	fexist
func/s	getfct
inite/s	inite
letter/s	letter
prog/s	putprog
readf/s	readfct
remove/s	remove
savef/s	savef
scncmd/s	scancmd
sfmsgc/s	shiftmsg
symove/s	symove

### (2) The SOLVER generated program -

file name	routine name
-----	-----
driver/s	driver (the control routine of the generated program)
curves/s	curves
dest/s	destin
maptyp/s	maptype
root/s	muller
ncyc/s	ncyc
plot/s	rootplot
setf/s	setfile
step/s	stepsz
wvset/s	wvsetup

To modify SOLVER, the user need only recompile those source files that are modified and use BUILD to update "solv/b/1". If all the modified files are in item (2) above, the user need only update the SOLVER library "solv/b/1"; otherwise, the user must generate a new "solver/x" by using LDR as follows:

```
ldr i=solver/b,x=solver/x,lib=solv/b/1:fortlib
```

"solver/b" is the binary obtained by compiling main. (in file "solver/s") using CFT. If "solver/s" is not modified, "solver/b" can be generated by:

```
build ol. solv/b/1(LF)g. solver/b(LF)xg. main.(LF)end
```

## APPENDIX II. Listings of The Source Files

The following is a listing of all source programs used by SOLVER; files are current as of August 31, 1979, and are listed in the same order as in the table above:

```

1  c  main program of solver
2  c
3      implicit integer(a-z)
4      common /dabk/ Keep, initprm
5      data Keep /0/
6      data termin /0/
7  c
8      integer msg(10), symb(4), type(4), len
9  c
10     integer libuf(20), nlib, buf(22)
11     data buf /" i=#b,x", "=#x,lib=", "solv/b/1", ":displa",
12           ":tv80lib", ":fortlib"/
13     data nlib /32/
14     equivalence ( libuf(1), buf(3) )
15  c
16  c  initialization
17     call dropfile ( 0 )
18     call msgflag ( msgset, 2 )
19     call msglink ( 59, 1 )
20     call mprompt ( ">", 1 )
21     call msgtor ( "*** root solver 79.08 ***", 32 )
22  c
23  c  extract binary modules from the library
24     if ( fexist(libuf(1)).eq.0 ) go to 10
25     call msgtor ( "extracting solv/b/1", 20 )
26     call inite ( "lib", 600000000, 0., ires )
27     call msgtoe ( "solver", 6, 1, 0, ires )
28     call msgfre ( msg, len, 26, ires )
29     call msgtor ( msg, len )
30     call msgtoe ( "x solv/b/1", 10, 1, 0, ires )
31     call msgfre ( msg, len, 32, ires )
32     if ( len.gt.5 ) call msgtor ( msg, len )
33     call bypass ( 1, 1 )
34     call msgtoe ( "end", 3, 1, 0, ires )
35     call msgfre ( msg, len, 16, ires )
36  c
37     10 continue
38         call msgtor ( " ", 1 )
39  c  scan commands
40         call scancmd ( "#i", libuf, nlib, termin )
41         go to ( 20, 50, 40, 12, 40 ), termin+1
42  c  compile root solver
43     12 continue
44         termin = 4
45     20 continue
46         call msgtor ( "compiling and loading", 21 )
47         call inite ( "cft", 6000000000, 0., ires )
48         call destroy ( "#b" )
49         call bypass ( 1, 1 )
50         call msgtoe ( "i=#i,b=#b,l=0", 13, 0, 0, ires )
51         call msgfre ( msg, len, 80, ires )
52  c  load root solver
53     30 continue
54         call inite ( "ldr", 6000000000, 0., ires )

```

```

55      call destroy ( "#x" )
56      call bypass ( 1, 1 )
57      call msgtoe ( buf, nlib+16, 0, 0, ires )
58      call msgfre ( msg, len, 80, ires )
59 c   execute root solver
60      call msgtor ( "executing", 10 )
61      40 continue
62      call inite ( "#x", 6000000000, 0., ires )
63      if ( ires.ne.0 ) go to 45
64      call bypass ( 1, 1 )
65      nc = -1
66      if ( termin.eq.4 ) nc = 8
67      call msgtoe ( initprm, nc, 0, 0, ires )
68      call msgfre ( msg, len, 80, ires )
69      call getsymb ( symb, type, nc, msg, len, 4 )
70      if ( symb(3).ne."all" .or. symb(4).ne."done" )
71      -   call msgtor ( msg, len )
72      go to 10
73      45 continue
74      call msgtor ( "program not yet generated", 30 )
75      go to 10
76 c
77 c   destroy temporary files
78      50 continue
79      call destroy ( "#i" )
80      call destroy ( "#b" )
81      call destroy ( "#x" )
82      call destroy ( "1#b" )
83      call destroy ( "$out" )
84      call destroy ( "disout" )
85      if ( keep.eq.0 ) call destroy ( "solv/b/1" )
86      call exit
87 c
88      end

```

```

1      logical function digit ( ch )
2      integer  ch
3      c
4      c digit returns true if ch is a digit
5      c
6      integer  zero, nine
7      data     zero / 60b /, nine / 71b /
8      c
9      digit = .true.
10     if ( ch.ge.zero .and. ch.le.nine ) return
11     digit = .false.
12     return
13     end

```

```

1      integer function fexist ( name )
2      implicit integer(a-z)
3      c
4      c check if name is an existant file
5      c fexist returns    0 => file exists
6      c                    else => file not exists
7      c
8      call freeioc ( ioc )
9      fexist = izopen ( ioc, name, len, acs )
10     if ( fexist.eq.0 ) call izclose ( ioc, acs )
11     return
12     end

```

```

1      subroutine getfct ( buffer, nl )
2      integer buffer(9,1), nl
3      c
4      c gets the function or the subroutine and stores it in buffer.
5      c terminates when the first non-blank character is f followed by
6      c a blank or a equal sign, or is a slash
7      c
8      logical digit
9      integer line(80)
10     integer blank, equ, f, slash
11     data blank / 40b /
12     data equ / 75b /, f / 146b /
13     data slash / 57b /
14     c
15     call mprompt ( ":", 1 )
16     nl = 0
17     20 continue
18         nl = nl + 1
19         call msgfrr ( buffer(1,nl), len, 72 )
20         call zrjcharz ( line, buffer(1,nl), len )
21         i = 0
22     25 continue
23         i = i + 1
24         if ( line(i).eq.slash ) go to 40
25         if ( line(i).eq.blank .or. digit(line(i)) ) go to 25
26         if ( line(i).ne.f ) go to 20
27         if ( line(i+1).ne.blank .and. line(i+1).ne.equ ) go to 20
28     c
29     30 continue
30     call mprompt ( ">", 1 )
31     return
32     c
33     40 continue
34         nl = nl - 1
35         go to 30
36     c
37     end

```



```

1      subroutine inite ( name, time, prio, result )
2      implicit integer(a-z)
3      integer name, time, result
4      real    prio
5      c
6      c  initiates the controllee - name
7      c
8      parameter ( nbeta=8 )
9      integer beta(nbeta)
10     data      beta(1) /20000b/
11     data      beta(5) /0/
12     data      beta(7) /0/
13     data      beta(8) /0/
14     c
15     real      priority
16     equivalence ( priority, beta(5) )
17     c
18     beta(3) = name
19     beta(4) = time
20     priority = prio
21     result = izcray ( beta, nbeta )
22     return
23     end

```

```

1      logical function letter ( ch )
2      integer:  ch
3
4      c letter returns true if ch is a letter
5      c
6      integer    a, z
7      data      a / 141b /, z / 172b /
8      c
9      letter = .true.
10     if ( ch.ge.a .and. ch.le.z ) return
11     letter = .false.
12     return
13     end

```

```

1      subroutine putprog ( ious, itext, buffer, nl, w )
2      implicit integer(a-z)
3      integer ious, itext, buffer(9,1), nl, w
4  c
5  c writes the function f or the subroutine wvsetup onto the disk
6  c file - itext
7  c
8      real h
9      common /miscbk/ n
10     data n /1/
11  c
12     integer fun(9), sub(9), fors(9,2)
13     equivalence ( fun(1),fors(1,1) ), ( sub(1),fors(1,2) )
14     data fun /" co", "mplex fu", "nction ", "f (z)", 5*" " /
15     data sub /" su", "broutine", " wsetu", "p", 5*" " /
16  c
17     if ( w.gt.1 ) go to 10
18     call freeus ( ious )
19     call create ( ious, itext, 2, -1 )
20  c
21  c write program on disk
22     write(ious,101)
23     10 continue
24     write(ious,102) (fors(i,w),i=1,9), n
25     write(ious,103) ((buffer(j,i), j=1,9), i=1,nl)
26     write(ious,104)
27     return
28  c
29     101 format ( " call driver" /
30     " end" )
31     102 format ( 9a8 /
32     " complex z" /
33     " integer of, id(10), plotmode, plotid" /
34     " common /miscbk/ n, maxit, h, epl, ep2, of " /
35     " data n /", i4, " " /
36     " real x(1024), w(1024)" /
37     " common /plotbk/ id, mapset, plotmode," /
38     " plotid, x, w, xsize, ysize, inlen," /
39     " grids, nx, nw" /
40     " complex c(30), p(1024,100), y(1024)" /
41     " integer l(30)" /
42     " common /parmbk/ l, c, p, y" )
43     103 format ( 9a8 )
44     104 format ( " return" /
45     " end" )
46  c
47     end

```

```

1      subroutine readfct ( ious, name, funct, nl, ierr )
2      implicit integer(a-z)
3      integer ious, name, funct(9,1), nl, ierr
4      data first /1/
5      c
6      c reads function or subroutine from a file
7      c
8          if ( first.eq.1 ) call freeus ( ious )
9          first = 0
10         ierr = 0
11         call open ( ious, name, 0, ierr )
12         if ( ierr.le.0 ) return
13         nl = 0
14     10 continue
15         nl = nl + 1
16         call rdline ( ious, funct(1,nl), len )
17         if ( len.gt.-1 ) go to 10
18         nl = nl - 1
19         call close ( ious )
20         return
21     c
22     end

```

```

1      subroutine remove ( name, str, len )
2      implicit integer(a-z)
3      integer name, str(1), len
4      c
5      c remove name from str. len is the number of character in str.
6      c
7      parameter ( cpw=8 )
8      parameter ( blank=40b )
9      integer buf(25)
10     c
11     call zmovechr ( buf, 0, str, 0, len )
12     c
13     c compute the length of name
14     c
15     n = zscanrne ( name, 0, cpw, blank ) + cpw
16     c
17     c remove name
18     c
19     loc = zskeybyt ( buf, 0, len, name, n )
20     if ( loc.eq.0 ) go to 30
21     if ( loc.eq.len ) go to 40
22     call zmovechr ( str, 0, buf, 0, loc-1 )
23     len = len - n - 1
24     call zmovechr ( str, loc-1, buf, n+loc, len-loc+1 )
25     return
26     c
27     30 continue
28     n = n + 1
29     len = len - n
30     call zmovechr ( str, 0, buf, n, len )
31     return
32     c
33     c not found
34     c
35     40 continue
36     write(59,101) name
37     return
38     c
39     101 format ( "not found: ", a8 )
40     c
41     end

```

```

1      subroutine savef ( name, funct, nl )
2      implicit integer(a-z)
3      integer name, funct(9,1), nl
4      c
5      c save funct to the file "name"
6      c
7          if ( fexist(name).ne.0 ) go to 5
8          answer = "n"
9          call mprompt ( "destroy existence file? ", 24 )
10         call msgfrr ( answer, len, 1 )
11         call mprompt ( ">", 1 )
12         if ( answer.ne."y" ) return
13     5 continue
14         call freeus ( ious )
15         call create ( ious, name, 2, -1, ierr )
16         if ( ierr.ne.0 ) go to 20
17         write(ious,101) ((funct(j,i), j=1,9), i=1,nl)
18         call close ( ious )
19         return
20     c
21     20 continue
22         call msgtor ( "error creating file", 25 )
23         return
24     c
25     101 format ( 9a8 )
26     c
27     end

```

```

1      subroutine scancmd ( itext, libuf, nlib, termin )
2      implicit integer(a-z)
3      integer itext, libuf(1), nlib, termin
4      c
5      c this subroutine scans the input line and executes the
6      c given commands
7      c
8      parameter ( ncmd=10 )
9      integer command(ncmd)
10     data      command /
11     .          "sub",
12     .          "kbin",
13     .          "dlib",
14     .          "xeq",
15     .          "save",
16     .          "savef",
17     .          "alib",
18     .          "end",
19     .          "go",
20     .          "f" /
21     common      /miscbk/ n
22     c
23     common      /dabk/ keep, initprm
24     c
25     parameter ( maxsym=80 )
26     integer symb(maxsym), type(maxsym), buffer(10)
27     c
28     integer i, numsym
29     data      i / 0 /, numsym / 0 /
30     c
31     integer funct(9,300), n1
32     data      funct / "      f=", "0.", "7*" " /
33     data      n1 / 1 /
34     integer prog(9,300), npl
35     data      prog / "c", "8*" " /
36     data      npl / 1 /
37     c
38     integer buf(20)
39     c
40     if ( i.lt.numsym ) go to 20
41     c
42     c main loop to interpret commands
43     c
44     10 continue
45     i = 0
46     call msgfrr ( buffer, len, maxsym )
47     call getnumb ( symb, type, numsym, buffer, len, maxsym )
48     type(numsym+1) = 7
49     20 continue
50     i = i + 1
51     go to ( 100, 400, 20, 420, 420, 420, 420, 10 ), type(i)+1
52     c
53     100 continue
54     go to ( 120, 300, 280, 260, 240, 230, 220, 200, 180,

```

```

55      1          160, 140 ), iscmd(symb(i),command,ncmd)+1
56 c
57 c  controllee assumed
58 c
59      120      continue
60          call inite ( symb(i), 60000000000, 0., ires )
61          if ( ires.ne.0 ) go to 130
62          call bypass ( '1, 1 )
63          ns = -1
64          if ( type(i+1).eq.7 ) go to 125
65          call shiftmsg ( buffer, len )
66          ns = len
67      125      continue
68          call msgtoe ( buffer, ns, 0, 0, ires )
69          call msgfre ( buffer, len, maxsym, ires )
70          call getsymb ( symb, type, numsym, buffer, len, 4 )
71          if ( symb(3).eq."all" .and. symb(4).eq."done" ) go to 10
72          call msgtor ( buffer, len )
73          go to 10
74      130      continue
75          call msgtor ( "invalid command disregarded", 30 )
76          go to 20
77 c
78 c  f command
79 c
80      140      continue
81          if ( type(i+1).ne.0 ) go to 155
82          call readfct ( ious, symb(i+1), funct, nl, ires )
83          if ( ires.lt.0 ) go to 155
84          i = i + 1
85          go to 20
86      155      continue
87          call getfct ( funct, nl )
88          go to 20
89 c
90 c  go command
91 c
92      160      continue
93          termin = 0
94          call putprog ( iou, itext, funct, nl, 1 )
95          if ( icall.eq.1 ) call putprog ( iou, itext, prog, npl, 2 )
96          call close ( iou )
97          icall = 0
98          if ( type(i+1).ne.0 ) return
99          if ( iscmd(symb(i+1),command,ncmd).gt.0 ) return
100         i = i + 1
101         initprm = symb(i)
102         termin = 3
103         return
104 c
105 c  end command
106 c
107      180      continue
108         termin = 1

```



```

109          return
110 c
111 c  alib command
112 c
113 200      continue
114          call zmovechr ( buf, 0, libuf, 0, nlib )
115          newlib = 0
116 205      continue
117          if ( type(i+1).gt.1 ) go to 210
118          i = i + 1
119          call symove ( libuf, newlib, symb(i) )
120          if ( type(i+1).eq.1 ) go to 205
121          call zmovechr ( libuf, newlib, ":", 0, 1 )
122          newlib = newlib + 1
123          go to 205
124 210      continue
125          call zmovechr ( libuf, newlib, buf, 0, nlib )
126          nlib = nlib + newlib
127          go to 20
128 c
129 c  savef command
130 c
131 220      continue
132          if ( type(i+1).ne.0 ) go to 225
133          i = i + 1
134          call savef ( symb(i), funct, nl )
135          go to 20
136 225      continue
137          call msgtor ( "file name required", 20 )
138          go to 20
139 c
140 c  saves command
141 c
142 230      continue
143          if ( type(i+1).ne.0 ) go to 225
144          i = i + 1
145          call savef ( symb(i), prog, npl )
146          go to 20
147 c
148 c  xeq command
149 c
150 240      continue
151          termin = 2
152          if ( type(i+1).gt.0 ) return
153          if ( iscmd(symb(i+1),command,ncmd).gt.0 ) return
154          i = i + 1
155          initprm = symb(i)
156          termin = 4
157          return
158 c
159 c  dlib command
160 c
161 260      continue
162          if ( type(i+1).ne.0 ) go to 20

```

```

163             i = i + 1
164             call remove ( symb(i), libuf, nlib )
165             go to 260
166 c
167 c   kbin command
168 c
169     280      continue
170             keep = 1
171             go to 20
172 c
173 c   sub command
174 c
175     300      continue
176             ical1 = 1
177             if ( type(i+1).ne.0 ) go to 305
178             call readfct ( ious, symb(i+1), prog, npl, ires )
179             if ( ires.lt.0 ) go to 305
180             i = i + 1
181             go to 20
182     305      continue
183             call getfct ( prog, npl )
184             go to 20
185 c
186 c   symbol with more than 8 characters
187 c
188     400      continue
189             call msgtor ( "symbol too long", 15 )
190             go to 20
191 c
192 c   set number of solutions
193 c
194     420      continue
195             n = symb(i)
196             go to 20
197 c
198             end

```

```

1      subroutine shiftmsg ( line, len )
2      implicit integer(a-z)
3      integer line(1), len
4      c
5      c shiftmsg shifts off the first symbol from line. len is the
6      c number of characters in line.
7      c
8      c parameter ( blank=40b )
9      c
10     sbk = zscanlne ( line, 0, len, blank )
11     loc = zscanleq ( line, sbk, len-sbk, blank )
12     nbk = zscanlne ( line, sbk+loc, len-sbk-loc, blank )
13     len = len - sbk - loc - nbk
14     call zmovechr ( line, 0, line, sbk+loc+nbk, len )
15     return
16     c
17     end

```

```

1      subroutine symove ( destin, loc, source )
2      implicit integer(a-z)
3      integer  destin(1), loc, source
4      c
5      c move characters from source to destin(loc) until a blank is
6      c detected. loc is the offset in characters of destin
7      c
8      parameter ( blank=40b )
9      parameter ( cpw=8 )
10     c
11         n = zscanleq ( source, 0, cpw, blank )
12         call zmovechr ( destin, loc, source, 0, n )
13         loc = loc + n
14         return
15     c
16     end

```

```

1      subroutine driver
2      c
3      c the driver of the solver generated program
4      c
5      external f
6      complex f, y(1024)
7      integer l(30), id(10), inlen, mapset, plotmode, plotid, of
8      integer grids, debug, first
9      complex c(30), p(1024,100)
10     real x(1024), w(1024)
11     common /parmbk/ l, c, p, y
12     common /plotbk/ id, mapset, plotmode, plotid, x, w,
13     . xsize, ysize, inlen, grids, nx, nw
14     common /miscbk/ n, maxit, known, h, ep1, ep2, of
15     namelist /initbk/ n, maxit, known, h, ep1, ep2, of, l, c, p, y,
16     . mapset, plotmode, plotid, x, w, xsize, ysize,
17     . id, grids, debug, nx, nw
18     parameter ( npmax=1024, nrmax=25 )
19     real rootre(npmax,nrmax), rootai(npmax,nrmax)
20     c
21     c default values
22     c
23     data maxit /100/
24     data known /0/
25     data h /.5/
26     data ep1 /0./
27     data ep2 /0./
28     data of /0/
29     data mapset /0/
30     data plotmode /0/
31     data plotid /0/
32     data xsize /6.0/
33     data ysize /4.5/
34     data id /"solver"/
35     data grids /0/
36     data debug /0/
37     data nx /npmax/
38     data nw /nrmax/
39     c
40     integer line(10), symb(1), type(1)
41     integer namef(2)
42     data namef / 5r+root, 4rrout /
43     c
44     c initializations
45     c
46     call setfile ( namef, 2 )
47     call dropfile ( namef(1) )
48     call msgflag ( msg, 2 )
49     c
50     c read input, if any
51     c
52     if ( msg.eq.0. ) go to 15
53     call msgfrr ( line, len, 80 )
54     call getnumb ( symb, type, ns, line, len, 1 )

```

```

55         if ( symb(1).ne."tty" ) go to 20
56         call mprompt ( "-", 1 )
57         read ( 59, initbk )
58         call mprompt ( 0, 0 )
59         go to 15
60     20    continue
61         call open ( 5, symb(1), 0, inlen )
62         if ( inlen.lt.1 ) go to 35
63         read ( 5, initbk )
64     c
65     c create and write to output file, if of=1 or 2
66     c
67     15 continue
68         if ( of.lt.1 .or. of.gt.2 ) go to 35
69         namef(2) = 400b*namef(2) + 140b
70     10 continue
71         namef(2) = namef(2) + 1
72         call open ( 1, namef(2), 0, len )
73         call close ( 1 )
74         if ( len.gt.0 ) go to 10
75         call create ( 6, namef(2), 2, -1 )
76         write ( 59, 103 ) namef(2)
77         write ( 6, 104 ) n, maxit, Known, h, ep1, ep2, (y(i), i=1,n)
78     c
79     c check information
80     c
81     35 continue
82         n = min0 ( n, 1024 )
83         if ( of.gt.2 .and. plotmode.eq.0 ) plotmode = 1
84         iplot = plotmode
85         if ( iplot.gt.3 .or. mapset.gt.3 ) n = 1
86         np = nx
87         nr = nw
88         if ( np*nr.le.25600 .and. mapset.gt.3 ) go to 40
89         np = npmax
90         nr = nrmax
91     c
92     c main loop
93     c
94     40 continue
95         call vvsetup
96         if ( plotmode.gt.0 .and. first.eq.1 )
97             . call rootplot ( rootre, rootai, np, nr )
98         if ( 1(30).lt.0 ) go to 70
99         first = 1
100        if ( iplot.gt.3 ) go to 60
101        num = n
102        call muller ( Known, num, y, maxit, h, ep1, ep2, f, Kount )
103        if ( of.gt.2 ) go to 40
104        if ( of.eq.2 ) go to 50
105        write ( 59, 101 ) ( y(i), i = 1, num+Known )
106        if ( Kount.gt.maxit ) write ( 59, 102 ) Kount
107        if ( of.eq.0 ) go to 40
108    50 continue

```

```

109      write ( 6, 101 ) ( y(i), i = 1, num+known )
110      if ( kount.gt.maxit ) write ( 6, 102 ) kount
111      go to 40
112 c
113 c  evaluating function instead of finding roots
114 c
115      60 continue
116          ixind = ixind + 1
117          y(1) = f ( x(ixind) )
118          go to 40
119 c
120 c  terminating
121 c
122      70 continue
123          if ( iplot.le.0 ) call exit ( debug )
124          plotmode = iplot
125          call rootplot ( rootre, rootai, np, nr )
126          call donepl
127          call exit ( debug )
128 c
129      101 format ( "roots:" / ( " ( ", e19.11, " , ", e19.11, " )" ) )
130      102 format ( "does not converge after ", i5, " iterations" )
131      103 format ( a8, " is the output file" )
132      104 format ( "n      =", i4 / "maxit =", i4 / "known =", i4 /
133          .      "h      =", f15.7 / "ep1  =", f15.7 / "ep2  =", f15.7 /
134          .      "y      =", 2(2f15.7, " , ") / ( 7x, 2(2f15.7, " , ") ) )
135      end

```

```

1      subroutine curves ( x, y, np, mark, nc )
2      real      x(1), y(1024,1)
3      integer   np, mark, nc
4      c
5      c draw curves
6      c
7      do 10 i = 1, nc
8          call curve ( x(1), y(1,i), np, mark )
9      10 continue
10     return
11     end

```



```

1      subroutine destin ( plotmode, plotid )
2      integer plotmode, plotid
3      c
4      c plotmode = 1 => fr80
5      c           = 2 => prtplt
6      c           = 3 => tk
7      c plotid for <fr80> => id
8      c           for <prtplt> => line width
9      c           for <tk> => baud rate
10     c
11     mode = plotmode
12     if ( mode.lt.1 ) return
13     if ( mode.gt.3 ) mode = mod ( mode, 4 ) + 1
14     go to ( 10, 20, 30 ), mode
15     c
16     c fr80 file
17     c
18     10 continue
19         call keep80 ( "roots", 3 )
20         call fr80id ( plotid, 1, 1 )
21         call plts
22         return
23     c
24     c printer
25     c
26     20 continue
27         call prtplt ( plotid )
28         return
29     c
30     c tektronix
31     c
32     30 continue
33         call tk ( 0, .1*plotid, 59 )
34         return
35     c
36     end

```

```

1      subroutine maptype ( mapset, npoint, xmin, xmax, ymin, ymax,
2                          xsize, ysize )
3      integer  mapset, npoint
4      real    xmin, xmax, ymin, ymax, xsize, ysize
5  c
6  c  mapset = 0  => linear-linear
7  c            = 1  => linear-log
8  c            = 2  => log-linear
9  c            = 3  => log-log
10 c            = 4  => contour
11 c
12     parameter ( nwsp=16384 )
13     common      work(nwsp)
14     data        scale /"scale"/
15  c
16     map = mapset
17     if ( map.lt.0 ) map = 0
18     if ( map.gt.4 ) map = 0
19     nxsize = ( xsize+1 ) / 2
20     nysize = ( ysize+1 ) / 2
21     xinc = stepsz ( xmin, xmax, 2.*nxsize, xd )
22     yinc = stepsz ( ymin, ymax, 2.*nysize, yd )
23     xcyc = xsize / ncyc ( xmin, xmax, xdec )
24     ycyc = ysize / ncyc ( ymin, ymax, ydec )
25     go to ( 10, 20, 30, 40, 50 ), map+1
26  c
27 10 continue
28     call graf ( xmin-xd, scale, xmax+xd, ymin-yd, scale, ymax+yd )
29     return
30  c
31 20 continue
32     call ylog ( xmin-xd, xinc, ydec, ycyc )
33     return
34  c
35 30 continue
36     call xlog ( xdec, xcyc, ymin-yd, yinc )
37     return
38  c
39 40 continue
40     call loglog ( xdec, xcyc, ydec, ycyc )
41     return
42  c
43 50 continue
44     xinc = xinc - 2. * xd / max0 ( 2*nysize,4 )
45     yinc = yinc - 2. * xd / max0 ( 2*nysize,4 )
46     call graf ( xmin, xinc, xmax, ymin, yinc, ymax )
47     call bcomon ( nwsp )
48     call conlin ( 0, "solid", "labels", 1, 10 )
49     call conlin ( 1, "dash", "nolabels", 1, 8 )
50     call conlin ( 2, "chndot", "nolabels", 1, 5 )
51     call conlin ( 3, "dot", "nolabels", 1, 3 )
52     call conang ( 60. )
53     return
54  c
55     end

```

```

1      subroutine muller (Kn, n, rts, maxit, step, ep1, ep2, fn, Kount)
2      complex rts(1)
3      complex fn
4      c
5      c this subroutine uses the muller's method to find the roots of
6      c a function
7      c
8      c parameters:
9      c   Kn      - number of roots previously computed, normally zero
10     c   n       - input: number of roots desired
11     c             output: number of roots found
12     c   rts     - an array of initial estimates of all desired roots
13     c             set to zero if no better estimates are available
14     c   maxit   - maximum number of iterations per root allowed
15     c   step    - initial distance between x's
16     c   ep1     - relative error tolerance on x(i)
17     c   ep2     - error tolerance on f(x(i))
18     c   fn      - fn(z) is a external complex function which, for given
19     c             z, returns f(z)
20     c   Kount   - number of iterations of the last root.  if Kount
21     c             is equal to maxit, the routine will stop at once
22     c
23     c
24     complex rt, h, delfpr, frtdef, lambda, delf, dfprlm, num, den
25     complex g, sqr, frt, frtprv
26     c
27     c initialization
28     eps1 = amax1 (ep1, 1.e-12)
29     eps2 = amax1 (ep2, 1.e-20)
30     ibeg = Kn + 1
31     iend = Kn + n
32     c
33     do 100 i = ibeg, iend
34         Kount = 0
35     c compute first three estimates for root as
36     c rts(i)+step, rts(i)-step, and rts(i)
37         1   h = step
38             rt = rts(i) + h
39             index = 1
40             go to 70
41         10   delfpr = frtdef
42             rt = rts(i) - h
43             index = 2
44             go to 70
45         20   frtprv = frtdef
46             delfpr = frtprv - delfpr
47             rt = rts(i)
48             index = 3
49             go to 70
50         30   index = 4
51             lambda = 1.
52     c compute next estimate for root
53         40   delf = frtdef - frtprv
54             dfprlm = delfpr * lambda

```

```

55      num = -fritdef * (1. + lambda) * 2.
56      g = (1. + lambda * 2.) * delf - lambda * dfprim
57      sqr = g * g + 2. * num * lambda * (delf - dfprim)
58      sqr = csqrt(sqr)
59      den = g + sqr
60      if (real(g) * real(sqr) + aimag(g) * aimag(sqr) .lt. 0.)
61          1      den = g - sqr
62      if (cabs(den) .lt. epl) den = 1.
63      lambda = num / den
64      fritprv = fritdef
65      delfpr = delf
66      h = h * lambda
67      rt = rt + h
68      if (kount .gt. maxit) go to 200
69  c
70      70      kount = kount + 1
71      frit = fn (rt)
72      fritdef = frit
73      do 71 j = 1, i-1
74          den = rt - rts(j)
75          if (cabs(den) .lt. eps2) go to 79
76          fritdef = fritdef / den
77      71      continue
78      75      go to (10, 20, 30, 80), index
79      79      rts(i) = rt + .001
80      go to 1
81  c
82  c      check for convergence
83      80      if (cabs(h) .lt. eps1 * cabs(rt)) go to 100
84      if (amax1(cabs(frit), cabs(fritdef)) .lt. eps2) go to 100
85  c
86  c      check for divergence
87      if (cabs(fritdef) .lt. 10. * cabs(fritprv)) go to 40
88      h = .5 * h
89      lambda = .5 * lambda
90      rt = rt - h
91      go to 70
92      100 rts(i) = rt
93      return
94      200 continue
95      n = i
96      return
97      end

```

```

1      integer function ncyc ( vmin, vmax, vdec )
2      real      vmin, vmax, vdec
3      c
4      c  return the number of cycles needed for log scale
5      c
6      parameter ( one=.999999999999 )
7      parameter ( maxcyc=10 )
8      parameter ( base=1.e-20 )
9      c
10     delta = 0.
11     if ( vmin.lt.1. ) delta = 1.
12     min = alog10 ( amax1(vmin,base) ) - delta
13     max = alog10 ( amax1(vmax,base) ) + one
14     ncyc = max0 ( max-min, 1 )
15     vdec = 10. ** min
16     if ( vdec.gt.base*10. .or. ncyc.lt.maxcyc ) return
17     ncyc = maxcyc
18     vdec = 10. ** ( max-maxcyc )
19     return
20     c
21     end

```

```

1      subroutine rootplot ( rootre, rootai, npnew, nrnew )
2      real    rootre(npnew,nrnew), rootai(npnew,nrnew)
3      c
4      c  plots rootre vs. x and/or w and rootai vs. x and/or w
5      c  rootre and rootai are the real and imaginary part of the
6      c  roots (or the value of the function. if so, stored in
7      c  rootre(1) and rootai(1)). npnew is the maximum number of
8      c  points and nrnew is the maximum number of roots.
9      c
10     integer  l(30), id(10), inlen, mapset, plotmode, plotid, of
11     integer  grids
12     real     x(1024), w(1024)
13     complex  y(1024)
14     complex  c(30), p(1024,100)
15     common   /parmbk/  l, c, p, y
16     common   /plotbk/  id, mapset, plotmode, plotid, x, w,
17     .         xsize, ysize, inlen, grids
18     common   /miscbk/  n, maxit, known, h, ep1, ep2, of
19     c
20     real     psize(2)
21     parameter ( big=10.**100, small=-10.**100 )
22     integer  first, npoint, line(11)
23     data     remin, aimin, xmin, wmin /4*big/
24     data     remax, aimax, xmax, wmax /4*small/
25     data     line(11) /1h$/
26     data     psize /11.,8.5/
27     data     nx /0/, nw /25600/
28     c
29     integer  labfre(2), labfai(2), labtre(2), labtai(2)
30     integer  labwre(2), labwai(2), labx(2)
31     integer  labpw(2), labax(2), labaw(2)
32     data     labfre /"real(f)", " "/
33     data     labfai /"imag(f)", " "/
34     data     labpw  /"paramete", "r w"/
35     data     labax  /"argument", "x"/
36     data     labaw  /"argument", "w"/
37     data     labx   /"paramete", "r x"/
38     data     labwre /"real(roo", "t)"/
39     data     labwai /"imag(roo", "t)"/
40     data     labtre /" ", " "/
41     data     labtai /" ", " "/
42     c
43     c  initializations
44     c
45     if ( first.ne.0 ) go to 5
46     first = 1
47     nr = min0 ( n+known, nrnew )
48     xsz = xsize
49     ysz = ysize
50     c
51     c  check if page size should be 8.5"x11" or 11"x8.5"
52     c
53     if ( xsz.eq.ysz ) go to 5
54     if ( xsz.lt.ysz .and. ysz.le.psize(2)-1.5 ) go to 5

```

```

55         if ( xsz.ge.ysz .and. xsz.gt.psize(2)-1.5 ) go to 5
56         temp = psize(1)
57         psize(1) = psize(2)
58         psize(2) = temp
59     c
60     c store the roots of the function value, y
61     c
62         5 continue
63         if ( 1(30).lt.0 ) first = first + 1
64         if ( first.gt.2 ) go to 20
65         if ( mapset.gt.3 ) go to 12
66         nx = nx + 1
67         if ( nx.gt.npnew ) return
68         do 10 i = 1, nr
69             rootre(nx,i) = real ( y(i) )
70             rootai(nx,i) = aimag ( y(i) )
71             if ( remin.gt.rootre(nx,i) ) remin = rootre(nx,i)
72             if ( remax.lt.rootre(nx,i) ) remax = rootre(nx,i)
73             if ( aimin.gt.rootai(nx,i) ) aimin = rootai(nx,i)
74             if ( aimax.lt.rootai(nx,i) ) aimax = rootai(nx,i)
75         10 continue
76         return
77     c
78         12 continue
79         if ( nw.lt.nrnew ) go to 14
80         nx = nx + 1
81         nw = 0
82         14 continue
83         nw = nw + 1
84         rootre(nx,nw) = real ( y(1) )
85         rootai(nx,nw) = aimag ( y(1) )
86         if ( remin.gt.rootre(nx,nw) ) remin = rootre(nx,nw)
87         if ( remax.lt.rootre(nx,nw) ) remax = rootre(nx,nw)
88         if ( aimin.gt.rootai(nx,nw) ) aimin = rootai(nx,nw)
89         if ( aimax.lt.rootai(nx,nw) ) aimax = rootai(nx,nw)
90         return
91     c
92     c find the minimum and maximum values of x and w
93     c
94         20 continue
95         npoint = min0 ( nx, npnew )
96         do 30 j = 1, npoint
97             if ( xmin.gt.x(j) ) xmin = x(j)
98             if ( xmax.lt.x(j) ) xmax = x(j)
99         30 continue
100         if ( mapset.lt.4 ) go to 40
101         do 35 j = 1, nw
102             if ( wmin.gt.w(j) ) wmin = w(j)
103             if ( wmax.lt.w(j) ) wmax = w(j)
104         35 continue
105         if ( remax-remin.lt.1.e-25 ) nore = 1
106         if ( aimax-aimin.lt.1.e-25 ) noai = 1
107         remin = wmin
108         aimin = wmin

```

```

109          remax = wmax
110          aimax = wmax
111 c
112 c figure out the title and name for the axes
113 c
114 40 continue
115     if ( plotmode.lt.4 .and. mapset.lt.4 ) go to 60
116     if ( plotmode.lt.4 ) go to 50
117         call move ( labx, labax, 2 )
118         call move ( labure, labfre, 2 )
119         call move ( labwai, labfai, 2 )
120         if ( mapset.lt.4 ) go to 60
121         call move ( labure, labaw, 2 )
122         call move ( labwai, labaw, 2 )
123         call move ( labtre, labfre, 2 )
124         call move ( labtai, labfai, 2 )
125         go to 60
126 50     continue
127         call move ( labtre, labure, 2 )
128         call move ( labtai, labwai, 2 )
129         call move ( labure, labpw, 2 )
130         call move ( labwai, labpw, 2 )
131 c
132 c getting ready to plot
133 c
134 60 continue
135     call setbox ( id, 80 )
136     call gfsiz ( 3, 2000000b )
137     call destin ( plotmode, plotid )
138     call nobrdr
139 c
140 c plot the input file, if there is one
141 c
142     if ( inlen.le.0 ) go to 90
143     call page ( psize(1), psize(2) )
144     call tablet ( "ltset", "long" )
145     rewind 5
146 70     continue
147         read(5,1010) ( line(i), i=1,10 )
148         if ( ieof(5).ne.0 ) go to 80
149         call ltline ( line )
150         go to 70
151 80     continue
152         call endtab ( 0 )
153         call dendpl ( 0 )
154 c
155 90 continue
156     xsz = amin1 ( xsz, psize(1)-1.5 )
157     ysz = amin1 ( ysz, psize(2)-1.5 )
158     call page ( psize(1), psize(2) )
159 c
160 c plot the real(root) vs. x
161 c
162     xsor = ( psize(1)*.5-xsz )*.75

```



```

163      if ( xsor.lt..75 ) xsor = ( psize(1)-xsz )*.5
164      ysor = ( psize(2)*.5-ysz )*.5
165      if ( ysor.lt..5 ) ysor = ( psize(2)-ysz )*.5
166      call physor ( xsor, ysor )
167 c
168      call title ( labtre, 16, labx, 16, labwre, 16, xsz, ysz )
169      call maptype ( mapset, npoint, xmin, xmax, remin, remax,
170                  xsz, ysz )
171      call dframe
172      if ( grids.ne.0 ) call grid ( -grids, -grids )
173      if ( mapset.lt.4 ) go to 100
174      if ( nore.eq.1 ) go to 120
175      call conmak ( rootre, npnew, nrnew, "scale" )
176      call contour ( 4, "labels", "draw" )
177      go to 110
178 100      continue
179      call curves ( x, rootre, npoint, -1, nr )
180 110      continue
181      call endgr ( 0 )
182 c
183 c plot imaginary(root) vs. x
184 c
185 120 continue
186      xsor = psize(1)*.5 + ( psize(1)*.5-xsz )*.5
187      if ( xsor.lt..5+psize(1)*.5 ) xsor = ( psize(1)-xsz )*.5
188      ysor = psize(2)*.5 + ( psize(2)*.5-ysz )*.5
189      if ( ysor.lt..5+psize(2)*.5 ) ysor = ( psize(2)-ysz )*.5
190      call physor ( xsor, ysor )
191      if ( xsor.lt.psize(1)*.5 .and. ysor.lt.psize(2)*.5 )
192          call dendpl ( 0 )
193 c
194      call title ( labtai, 16, labx, 16, labwai, 16, xsz, ysz )
195      call maptype ( mapset, npoint, xmin, xmax, aimin, aimax,
196                  xsz, ysz )
197      call dframe
198      if ( grids.ne.0 ) call grid ( -grids, -grids )
199      if ( mapset.lt.4 ) go to 130
200      if ( noai.eq.1 ) go to 140
201      call conmak ( rootai, npnew, nrnew, "scale" )
202      call contour ( 4, "labels", "draw" )
203      go to 140
204 130      continue
205      call curves ( x, rootai, npoint, -1, nr )
206 140      continue
207      call endgr ( 0 )
208      call dendpl ( 0 )
209 c
210      return
211 c
212 1010 format ( 10a8 )
213 c
214      end

```

```

1      subroutine setfile (names, n)
2      c
3      c  subroutine setfile appends the current suffix (channel) to
4      c  all the given files.
5      c
6      c  parameters:
7      c      names - an one-dimensional integer array with file names
8      c               in it. all names are limited to 7 characters.
9      c      n      - an integer that indicates the number of files.
10     c
11     c  library used:
12     c      fortlb
13     c
14     c
15     c      integer names(n), n
16     c      integer userno, account, df, suffix, i
17     c
18     c  set unit 59 to be the controller or terminal
19     c
20     c      call msglink (59, 1)
21     c
22     c  get the suffix
23     c
24     c      call userinfo (userno, account, df, suffix)
25     c
26     c  change the left-justified suffix to right-justified
27     c
28     c      suffix = suffix / 4000000000000000000b
29     c
30     c  append the suffix to all file names
31     c
32     c      do 10 i = 1, n
33     c          names(i) = 400b * names(i) + suffix
34     c  10 continue
35     c
36     c      return
37     c      end

```

```

1      real function stepsz ( vmin, vmax, size, delta )
2      real      vmin, vmax, size, delta
3
4      c find step size for a graph
5      c
6      delta = amax1 ( 0.001, (vmax-vmin)*.05 )
7      stepsz = ( vmax-vmin+2.*delta ) / max0 ( int(size), 4 )
8      return
9      end

```

```

1      subroutine vvsetup
2      complex c(30), p(1024,100), y(1024)
3      integer i(30)
4      common /parmbk/ l, c, p, y
5      data    i(30) /1/
6  c
7      i(30) = i(30) - 1
8      return
9      end

```