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, CA 94720
SOLVER: AN ANALYTIC FUNCTION ROOT SOLVING
AND PLOTTING PACKAGE

by

H. Stephen Au-Yeung Alex Friedman

Memorandum No. UCB/ERL-M79/55
31 Aug 1979

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley 94720

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited
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.
1. Introduction

SOLVER is a root solver that provides automatic compilation, loading, and execution. It uses a simple version of Muller's method (see Ref. 19). SOLVER is intended for the occasional user who needs root solutions but balks at learning to use sophisticated but complicated solvers, which usually also require considerable additional programming. The user needs 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:

```
rfilém 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 [uscl 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 "solve" followed by some commands from the terminal. In this section, some basic commands are introduced:

\(<\text{integer}>\) - 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).

\(f\ [\text{file name}]\) - Enter the function to be solved. If the function is to be entered from the terminal, the user should wait for the prompt "f:" 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 "e" or a blank, or a slash is the first non-blank character of an input line.

\(\text{go}\ [\text{file name or "tty"}]\) - 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 \(<\text{integer}>\)).
- \(\text{maxit}\) - Maximum number of iterations per root allowed (defaults to 100).
- \(\text{Known}\) - Number of known roots (defaults to 0).
- \(\text{h}\) - Initial distance between estimated roots (defaults to 0.5).
- \(\text{ep1}\) - Relative error tolerance on \(z(i)\) (defaults to 1.e-12 which is also the minimum one can specify).
- \(\text{ep2}\) - Error tolerance on \(f(z(i))\) (defaults to 1.e-20 which is also the minimum one can specify).
- \(\text{of}\) - = 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).

\(\text{debug}\) - = 0 => Drop file deleted after execution (default).
  1 => Drop file retained after execution.

\(\text{xeq}\ [\text{file name or "tty"}]\) - Execute the generated program again without recompiling it. See the command "go" for details.

\(\text{save}\ [\text{file name}]\) -
Save the function into the file (file name) which can be read by using the command "f" later.

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

end -
Terminate SOLVER.

Example 11.1:
Most common syntax

user: solver / 2 2
routine : *** root solver 79.08 ***
routine : extracting solw/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 : FIO04 - CFT VERSION - 07/14/79 SCHEDULER
routine : FTO01 - COMPILE TIME = 0.0065 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : roots:
routine : ( 2.0000000000e+00 , 0.0000000000e-01 )
routine : ( -1.0000000000e+00 , 1.7320508075e+00 )
routine : ( -1.0000000000e+00 , -1.7320508075e+00 )
routine/user: >end
routine : all done

Example 11.2:
All commands on execute line

user: solver 20 f go end / 2 3
routine : *** root solver 79.08 ***
routine : extracting solw/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: f = csin ( z )
routine : compiling and loading
routine : FIO04 - CFT VERSION - 07/14/79 SCHEDULER
routine : FTO01 - COMPILE TIME = 0.0061 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine : roots:
routine : ( 0.0000000000e-01 , 0.0000000000e-01 )
routine : ( -3.1415925354e+00 , 9.3392673575e+21 )
routine : ( 3.1415925354e+00 , -4.2870605268e-36 )
<table>
<thead>
<tr>
<th>Routine</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-6.28318530718e+00, 1.55786371627e-25)</td>
</tr>
<tr>
<td>2</td>
<td>(6.28318530718e+00, -1.8151384110e-17)</td>
</tr>
<tr>
<td>3</td>
<td>(-1.25663706144e+01, 6.9470731794e-23)</td>
</tr>
<tr>
<td>4</td>
<td>(1.25663706144e+01, -4.06743868262e-20)</td>
</tr>
<tr>
<td>5</td>
<td>(-9.42477796877e+00, 3.36131166524e-27)</td>
</tr>
<tr>
<td>6</td>
<td>(9.42477796877e+00, 4.33163959599e-19)</td>
</tr>
<tr>
<td>7</td>
<td>(-2.51327412287e+01, 2.44708281568e-24)</td>
</tr>
<tr>
<td>8</td>
<td>(2.51327412287e+01, 2.46949267576e-19)</td>
</tr>
<tr>
<td>9</td>
<td>(-2.19911485751e+01, 2.31592585486e-23)</td>
</tr>
<tr>
<td>10</td>
<td>(2.19911485751e+01, 3.54161507684e-17)</td>
</tr>
<tr>
<td>11</td>
<td>(-3.45575191895e+01, -9.21324826052e-27)</td>
</tr>
<tr>
<td>12</td>
<td>(3.45575191895e+01, 7.42544852601e-19)</td>
</tr>
<tr>
<td>13</td>
<td>(-1.68495559215e+01, -4.60537034333e-22)</td>
</tr>
<tr>
<td>14</td>
<td>(3.14159265359e+01, -3.34489216761e-16)</td>
</tr>
<tr>
<td>15</td>
<td>(1.80495559215e+01, -1.30075028143e-26)</td>
</tr>
<tr>
<td>16</td>
<td>(-4.71238898638e+01, 1.40250535984e-15)</td>
</tr>
<tr>
<td>17</td>
<td>(4.398222971503e+01, 5.50257927188e-20)</td>
</tr>
</tbody>
</table>

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 himself.

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

```
59 - Terminal (tty).
05 - Input (exists only if "go <file name>" or "seq <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.

delib [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 controller. 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 controller, 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: : l(30) = l(30) - 1
routine/user: : if ( l(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: : go end
routine/user: : f = z**3 - c(1)
routine: compiling and loading
routine: FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine: FT004 - 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: ( 3.1740218394e+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: ( -1.25992104909e+00 , 2.18224727194e+00 )

-6-
Example III.2:
Deleting the default graphic libraries which are not used because no plots are made.

user: solver / 2
routine : *** root solver 79.08 ***
routine : extracting solv/b/1
routine : c 07/20/79 12:39:32 001222
routine/user: >dlib display tv00lib
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 - COMPILIE TIME = 0.0063 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : ***warning-unsatisfied externals***
routine : executing
routine : roots:
routine : ( -4.00000000000e+00 , -2.01640391737e-29 )
routine : ( 4.00000000000e+00 , 6.45710332361e+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 => Printer file (always called "disout").
  2 => 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 (printer 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 (x linear, y log).
  3 => A log-log plot.

xsize - The length (in inches) of the x-axis (defaults to 6.8). DISPLA 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)
    if ( 1(30).lt.8 ) 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 6.5"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 [use] <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 teletype 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 solu/b/1
routine:  c 07/22/79 20:19:41 001222
routine/user:  >sub
routine/user:  equivalence ( xind,1(29) )
routine/user:  data 1(30) /21/, xind /0/
routine/user:  data pi /3.1415926535/
routine/user:  c
routine/user: 1(30) = 1(30) - 1
routine/user: ixind = ixind + 1
routine/user: x(ixind) = .2 * pi * (ixind-1)
routine/user: if
routine/user: t = z - sin (x(ixind))
routine/user: endif
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 : ( 0.0000000000e-01 , 0.0000000000e-01 )
routine : roots:
routine : ( 5.8778525278e-01 , 0.0000000000e-01 )
routine : roots:
routine : ( 9.51056516284e-01 , 0.0000000000e-01 )
routine : roots:
routine : ( 9.51056516312e-01 , 0.0000000000e-01 )
routine : roots:
routine : ( 5.8778525235e-01 , 0.0000000000e-01 )
routine : roots:
routine : ( 8.9794606966e-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 , -4.2168917729e-81 )
routine : roots:
routine : ( 1.79589921933e-10 , 4.25795964001e-109 )
routine : roots:
routine : ( 5.87785252133e-01 , -1.51273121674e-123 )
routine : roots:
routine : ( 9.51056516229e-01 , 1.07486017721e-137 )
routine : roots:
routine : ( 9.51056516367e-01 , 1.59613441594e-146 )
routine : roots:
routine : ( 5.87785252496e-01 , 7.11282799835e-161 )
routine : roots:
routine : ( 2.69356460379e-10 , -1.25217744835e-29 )
routine : roots:
routine : ( -5.87785252860e-01 , -1.25217744835e-29 )
routine : roots:
routine : ( -9.51056516201e-01 , -1.27447352891e-57 )
routine : roots:
routine : ( -9.51056516395e-01 , 1.35835186182e-71 )
routine : roots:
Some users may prefer the following equivalent forms for sub and f:

(1) sub -
   data 1(30) /21/, pi /3.1415926535/
   l(30) = l(30) - 1
   x(21-l(30)) = .2 * pi * (20-l(30))
(2) f -
   f = z - sin( x(21-l(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 solutions
routine : c 07/22/79 20:19:41 001222
routine/user: ic note that we want a log plot, so we do not
routine/user: ic plot points with negative values of the
routine/user: ic parameter.
routine/user: ic
routine/user: equivalence (ixind.1(29)).(xnext.1(28))
routine/user: equivalence (ixind.1(29)).(xnext.1(28))
routine/user: c
routine/user: data 1(30) /11/, ixind /-6/
routine/user: c
routine/user: l(30) = l(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: c
routine/user: c
routine/user: equivalence (xnext.1(28))
routine/user: c
routine/user: f = z - xnext
routine : compiling and loading
routine : FTOO4 - CFT VERSION - 07/14/79 SCHEDULER
routine : FTOO1 - COMPIL ETAIME = 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 , 1.3700178954e-24 )
routine : roots:
routine : ( -6.40000000000e+01 , -3.01829413028e-50 )
routine : roots:
routine : ( -2.70000000000e+01 , -3.11734638189e-67 )
routine : roots:

-12-
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.

The contents of the various files are:
(1) disp -
  real v(1), lambda
  complex fsum(1)
equivalence (v(1),p(1,1)), (fsum(1),p(1,49))
equivalence (xel.c(8)), (xe2w.c(9)), (vek.c(10))
equivalence (ngr,1(2))
c
  fsum(1) = ngr*(1.+xel+xel2w/(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))*(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))*(z-v(i-1)))
  20 continue
  f = fsum(ngr+1)

(2) mbilhd -
  c appears as the subroutine "setup" 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), kmin, klast
  real ini, iti, lsi
  real lambda
equivalence (xx(1),p(1,1))
equivalence (vx(1),p(1,17))
equivalence (v(1),p(1,33))
equivalence (uti.c(1))
equivalence (ud.c(3))
equivalence (utc.c(4))
equivalence (ute.c(5))
equivalence (ve.c(6))
equivalence (rm.c(7))
equivalence (xel,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 (ini,c(15))
equivalence (iti.c(16))
equivalence (lsi,c(17))
equivalence (re,c(18))
equivalence (nn,1(1))
equivalence (ngr,1(2))
equivalence (npts,1(27))
c
  if(1(29).ne.0) go to 20
  1(30)=(klast-kmin)/ak+1.5
  npts = 1(30)
  write(6,1) nn, ngr, vti, upc, vte, rm, dk, kmin, klast,
  ini, iti, lsi

-17-
! format(,,"ion: ",nn="i15," ngr="i15," vti="e14.5/",
  "electron: ",
  "wpc="e14.5," vle="e15.5/",
  "mass ratio: rm="e22.5/",
  "dk="e14.5," kmin="e14.5," klast="e12.5/",
  "lni="e14.5," lli="e15.5," lsi="e15.5/")

  c set up the multibeam
  vmax=5.*vti
dv=vmax/(nn-1)
xv(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( vv.it.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) ) )
   xv(i1)=vv
   xv(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
   ucei=sqrt(wpc/rm)
exi=ucei*rm
   re=vte*ucei
   ve=vte=vti=(lni+lli)*exi
   write(6,180)uci;ucei;re;ve
180 format(" ucei="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
\begin{verbatim}
\nu(i) = u_x(i) * \star$
30 continue
vek = v_e * \star
bk = u_t * \star
b * bk * ek
mm = n
c
if ( b .eq. 0. ) go to 32
bi0 = bessel(0, b)
b1 = bessel(1, b)
lambda = bi0 * exp(-b)
a = (1. - lambda) / b
flo = b * (1. - bi1/bi0)
go to 33
32 lambda = 1.0
a = 1.0
flo = 0.0
33 x0 = a * wpc
rl1 = ini - lsi*(1. - flo) - lti * flo
xa2 = wpc * lambda * rl1 / (k * wkei)
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(ngr, mm) ) return
j1 = ngr + j - 10
y(j) = v(j) + cmplx(0., 0.01)
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
\end{verbatim}
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(i)>      
  if ( l(30).lt.0 ) go to 70
    i = i + 1
  y(i) = f ( x(i) )
  go to 40
70 continue
  call <plot>
  call exit ( debug )

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

\[ \text{nx} - \text{Number of elements in } x \text{ (defaults to 1024).} \]
\[ \text{nw} - \text{Number of elements in } w \text{ (defaults to 25).} \]

If the product of nx and nw is greater than 1024x25=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 "gr" 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
```
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/0. + 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 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 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 07/20/79 11:21:15 001222
routine : compiling and loading
routine : FT004 - CFT VERSION - 07/14/79 SCHEDULER
routine : FT001 - COMPIL E TIME = 0.0245 SECONDS
routine : *** cray loader version - c121 07/05/79
routine : executing
routine/user: >netplot f105ro0x 1.
routine : box number: 22
routine : destination: ucb
routine : 6 frames of output processed
routine : f105ro0x file id j 5: 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 = complex ( 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

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 nu=81
xsize=7. ysize=7.
of=3
$
References


3. M.J. Gerber, "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.).


APPENDIX I. 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 / tv" from the terminal causes "solver/x", the first entry in SOLVER, to be executed. A dropfile named \( +\text{solver}<s> \), 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 -

<table>
<thead>
<tr>
<th>file name</th>
<th>routine name</th>
</tr>
</thead>
<tbody>
<tr>
<td>solver/s</td>
<td>main. (the main program)</td>
</tr>
<tr>
<td>cmds/s</td>
<td>iscmd</td>
</tr>
<tr>
<td>digit/s</td>
<td>digit</td>
</tr>
<tr>
<td>fexist/s</td>
<td>fexist</td>
</tr>
<tr>
<td>funct/s</td>
<td>getfct</td>
</tr>
<tr>
<td>inito/s</td>
<td>inito</td>
</tr>
<tr>
<td>letter/s</td>
<td>letter</td>
</tr>
<tr>
<td>prog/s</td>
<td>putprog</td>
</tr>
<tr>
<td>readf/s</td>
<td>readfct</td>
</tr>
<tr>
<td>remove/s</td>
<td>remove</td>
</tr>
<tr>
<td>savef/s</td>
<td>savef</td>
</tr>
<tr>
<td>scncmd/s</td>
<td>scancmd</td>
</tr>
<tr>
<td>sfmsge/s</td>
<td>shiftmsg</td>
</tr>
<tr>
<td>symove/s</td>
<td>symove</td>
</tr>
</tbody>
</table>

(2) The SOLVER generated program -

<table>
<thead>
<tr>
<th>file name</th>
<th>routine name</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver/s</td>
<td>driver (the control routine of the generated program)</td>
</tr>
<tr>
<td>curves/s</td>
<td>curves</td>
</tr>
<tr>
<td>dest/s</td>
<td>destin</td>
</tr>
<tr>
<td>mauptyp/s</td>
<td>mauptype</td>
</tr>
<tr>
<td>root/s</td>
<td>muller</td>
</tr>
<tr>
<td>ncyc/s</td>
<td>ncyc</td>
</tr>
<tr>
<td>plot/s</td>
<td>rootplot</td>
</tr>
<tr>
<td>self/s</td>
<td>selffile</td>
</tr>
<tr>
<td>step/s</td>
<td>stepsz</td>
</tr>
<tr>
<td>vvset/s</td>
<td>vvsetup</td>
</tr>
</tbody>
</table>
To modify SOLVER, the user need only recompile those source files that are modified and use BUILD to update "solver/b/1". If all the modified files are in item (2) above, the user need only update the SOLVER library "solver/b/1": otherwise, the user must generate a new "solver/x" by using LDR as follows:

```plaintext
ldr i=solver/b,x=solver/x,lib=solver/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:

```plaintext
build ol, solver/b/1(LF)g, solver/b(LF)xg, main.(LF)end
```
APPENDIX I. 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/l" from "solver" (if "solv/b/l" 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 -

<table>
<thead>
<tr>
<th>file name</th>
<th>routine name</th>
</tr>
</thead>
<tbody>
<tr>
<td>solver/s</td>
<td>main. (the main program)</td>
</tr>
<tr>
<td>cmds/s</td>
<td>iscmd</td>
</tr>
<tr>
<td>digit/s</td>
<td>digit</td>
</tr>
<tr>
<td>fexist/s</td>
<td>fexist</td>
</tr>
<tr>
<td>funct/s</td>
<td>getfct</td>
</tr>
<tr>
<td>init/s</td>
<td>init</td>
</tr>
<tr>
<td>letter/s</td>
<td>letter</td>
</tr>
<tr>
<td>prog/s</td>
<td>putprog</td>
</tr>
<tr>
<td>readf/s</td>
<td>readfct</td>
</tr>
<tr>
<td>remove/s</td>
<td>remove</td>
</tr>
<tr>
<td>savef/s</td>
<td>savef</td>
</tr>
<tr>
<td>scancmd/s</td>
<td>scancmd</td>
</tr>
<tr>
<td>sfmsg/s</td>
<td>shiftmsg</td>
</tr>
<tr>
<td>symove/s</td>
<td>symove</td>
</tr>
</tbody>
</table>

(2) The SOLVER generated program -

<table>
<thead>
<tr>
<th>file name</th>
<th>routine name</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver/s</td>
<td>driver (the control routine of the generated program)</td>
</tr>
<tr>
<td>curves/s</td>
<td>curves</td>
</tr>
<tr>
<td>dest/s</td>
<td>destin</td>
</tr>
<tr>
<td>moptyp/s</td>
<td>moptype</td>
</tr>
<tr>
<td>root/s</td>
<td>muller</td>
</tr>
<tr>
<td>ncy/s</td>
<td>ncy</td>
</tr>
<tr>
<td>plot/s</td>
<td>rootplot</td>
</tr>
<tr>
<td>self/s</td>
<td>setfile</td>
</tr>
<tr>
<td>step/s</td>
<td>stepsz</td>
</tr>
<tr>
<td>vset/s</td>
<td>vsetup</td>
</tr>
</tbody>
</table>
To modify SOLVER, the user need only recompile those source files that are modified and use BUILD to update "solv/b/l". If all the modified files are in item (2) above, the user need only update the SOLVER library "solv/b/l"; otherwise, the user must generate a new "solv/x" by using LDR as follows:

```
ldr i=solver/b.x=solver/x,lib=solv/b/l:fort\lib
```

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

```
build al, 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.
c main program of solver

implicit integer(a-z)
common /dabk/ keep, initprm
data keep /=0/
data termin /=0/

integer msg(10), symb(4), type(4), len
integer libuf(20), nlib, buf(22)
data libuf ="lib", buf ="lib", "solvb/lib", "displab",
data nlib =32/
equivalence (libuf(1), buf(3))

a initialization
  call dropfile ( 0)
call msgflag ( magset, 2)
call msglink ( 59, 1)
call mprcmt ("*", 1)
call msgtor ( "### root solver 79.08 ", 32)
c
extract binary modules from the library
  if ( exist(libuf(1)).eq.0 ) go to 10
call msgtor ( "extracting solvb/lib", 20)
call init ( "lib", 6000000000, 0, ires)
call msgtor ( "solver", 6, 1, 0, ires)
call msgfre ( msg, len)
call msgtor ( msg, len)
call msgtor ( msg, len)
call msgtor ( msg, len)
call msgtor ( msg, len)
call bypass ( 1, 1)
call msgtor ( "end", 3, 1, 0, ires)
call msgfre ( msg, len, 16, ires)
c
continue
  call msgtor ( "*", 1)
c scan commands
  call scancmd ( "*i", libuf, nlib, termin)
go to ( 20, 50, 40, 12, 40), termin+1
c compile root solver
  12 continue
  term = 4
  20 continue
  call msgtor ( "compiling and loading", 21)
call init ( "cft", 6000000000, 0, ires)
call destroy ( "m5b")
call bypass ( 1, 1)
call msgtor ( "i=lib=b=lib=b=lib=b=lib=b=lib=", 13, 0, 0, ires)
call msgtor ( msg, len, 10, ires)
c load root solver
  30 continue
  call init ( "ldr", 6000000000, 0, ires)
55  call destroy ("ex")
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 init ("ex", 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  45 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 ("ei")
80    call destroy ("eb")
81    call destroy ("ex")
82    call destroy ("eb")
83    call destroy ("sout")
84    call destroy ("disout")
85    if (keep.eq.0) call destroy ("solv/b/1")
86    call exit
87  c
88  end
logical function digit ( ch )
integer ch

digit returns true if ch is a digit

digit = .true.
if ( ch.ge.zero .and. ch.le.nine ) return
digit = .false.
return
end
call destroy ( "ex" )
call bypass ( 1, 1 )
call msgtoe ( buf, nlib+16, 0, 0, ires )
call msgfre ( msg, len, 80, ires )
c execute root solver
    call msgtor ( "executing", 10 )
40 continue
call init ( "ex", 6000000000, 0., ires )
if ( ires.ne.0 ) go to 45
call bypass ( 1, 1 )
c = -1
if ( termin.eq.4 ) nc = 8
call msgtoe ( initprm, nc, 0, 0, ires )
call msgfre ( msg, len, 80, ires )
call getsymb ( symb, type, nc, msg, len, 4 )
if ( symb(3), ne, "all", or, symb(4), ne, "done" )
    call msgtor ( msg, len )
go to 10
45 continue
call msgtor ( "program not yet generated", 30 )
go to 10
c
c destroy temporary files
50 continue
call destroy ( "ei" )
call destroy ( "eb" )
call destroy ( "ex" )
call destroy ( "ieb" )
call destroy ( "$out" )
call destroy ( "$disout" )
if ( keep.eq.0 ) call destroy ( "solv/b/1" )
call exit
c
end
logical function digit ( ch )
   integer ch
   digit returns true if ch is a digit
   integer zero, nine
   data zero / 60b /, nine / 71b /
   digit = .true.
   if ( ch.ge.zero .and. ch.le.nine ) return
   digit = .false.
   return
end
integer function fexist ( name )
implicit integer(a-z)
ccheck if name is an existant file
fc exist returns 0 => file exists
celse => file not exists
c
call freioic ( ioc )
fexist = izopen ( ioc, name, len, acs )
if ( fexist.eq.0 ) call izclose ( ioc, acs )
return end
subroutine getfat (buffer, nl)
integer buffer(9,1), nl

c gets the function or the subroutine and stores it in buffer.
c terminates when the first non-blank character is f followed by
a blank or a equal sign, or is a slash

logical digit
integer line(80)
integer blank, equ, f, slash
data blank / 40b /
data equ / 75b /, f / 145b /
data slash / 57b /

call mprompt (":", 1 )

nl = 0
20 continue
18 nl = nl + 1
19 call msgfrr (buffer(1,nl), len, 72 )
20 call zrjcharz ( line, buffer(1,nl), len )
i = 0
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
30 continue
30 call mprompt (">", 1 )
31 return

40 continue
34 nl = nl - 1
35 go to 30
36 c
37 end
subroutine init (name, time, prio, result)
implicit integer(a-z)
ingTEGER name, time, result
real prio

c
\* initiates the controller - name

c
parameter (nbeta=8)
ingTEGER beta(nbeta)
data beta(1) /20000b/
data beta(6) /0/
data beta(7) /0/
data beta(8) /0/
c
real priority
equivalence (priority, beta(5))
c
beta(3) = name
beta(4) = time
priority = prio
result = izcray (beta, nbeta)
return
end
1   logical function letter ( ch )
2       integer   ch
3       c
4       letter returns true if ch is a letter
5       c
6       integer   a, z
7       data      a / 14b /, z / 172b /
8       c
9
10      letter = .true.
11      if ( ch.ge.a .and. ch.le.z ) return
12      letter = .false.
13      return
14      end
subroutine putprog (ious, itext, buffer, nl, w)
implicit integer(a-z)
integer ious, itext, buffer(9,1), nl, w

! writes the function for the subroutine vvsetup on the disk
! file - itext
!
! real h
common /miscbk/ n
data n /1/
!
! integer fun(9), sub(9), fors(9,2)
equivalence ( fun(1), fors(1,1) ), ( sub(1), fors(1,2) )
data fun /* co".,"mplex fu","nction ","f (z)","5" */
data sub /* su"."broutine","vvsetu","p","5" */
!
if ( u.gt.1 ) go to 10
call freeus (ious)
call create (ious, itext, 2, -1)
!
write program on disk
!
write(ious,101)
10 continue
write(ious,102) (fors(i,w),i=1,9), n
write(ious,103) ((buffer(j,i), j=1,9), i=1,n1)
write(ious,104)
return
!
101 format ( "call driver" / )
"end"
!
102 format ( 9a8 /
"complex z" /
"integer of, id(10), plotmode, plotid" /
"common /miscbk/ n, maxit, h, ep1, ep2, of " /
"data n /", i4, "/ /
"real x(1024), w(1024)" /
"common /plotbk/ id, mapset, plotmode," /
". plotid, x, w, xsize, ysize, inlen," /
". grids, nx, nw" /
"complex c(30), p(1024,100), y(1024)" /
"integer i(30)" /
"common /parmbk/ l, c, p, y")
!
103 format ( 9a3 )
104 format ( "return" /
"end" )
!
end
subroutine readfct (ious, name, funct, nl, ierr)

implicit integer(a-z)
integer ious, name, funct(9,1), nl, ierr
data first /1/

C reads function or subroutine from a file

if (first.eq.1) call freeus (ious)
first = 0
ierr = 0
call open (ious, name, 0, ierr)
if ( ierr.le.0 ) return
nl = 0
10 continue
nl = nl + 1
call rdline (ious, funct(1,nl), len)
if ( len.gt.-1 ) go to 10
nl = nl - 1
call close (ious)
return
C
end
subroutine remove (name, str, len)

implicit integer(a-z)
integer name, str(1), len

remove name from str. len is the number of character in str.

parameter (cpw=8)
parameter (blank=40b)
integer buf(25)

call zmovechr (buf, 0, str, 0, len)

compute the length of name

n = zscanrne (name, 0, cpw, blank) + cpw

remove name

loc = zskkeybyt (buf, 0, len, name, n)
if (loc.eq.0) go to 30
if (loc.eq.len) go to 40

call zmovechr (str, 0, buf, 0, loc+1)
len = len - n - 1

call zmovechr (str, loc+1, buf, n+loc, len-loc+1)
return

30 continue

n = n + 1
len = len - n

call zmovechr (str, 0, buf, n, len)
return

33 not found

40 continue

write(59,101) name
return

101 format ("not found: ", a9)
subroutine savof (name, funct, ni)

implicit integer(a-z)
integer name, funct(9,1), ni

c
save funct to the file "name"

if (exist(name).ne.0) go to 5
answer = "n"
call mprompt ("destroy existence file? ", 24)
call msgfrr (answer, len, 1)
call mprompt ("y", 1)
if (answer.ne."y") return

5 continue

call freeus (ious)
call create (ious, name, 2, -1, ierr)
if (ierr.ne.0) go to 20
write(ious,101) ((funct(j,i), j=1,9), i=1,ni)
call close (ious)
return

20 continue

call msgtor ("error creating file", 25)
return

25 format (9a8)

end
subroutine scan cmd (i text, libuf, nlib, termin)
implicit integer(a-z)
integer i text, libuf(!), nlib, termin

this subroutine scans the input line and executes the
given commands

parameter (ncmd=10)
integer command(ncmd)
data command / "sub", "kbin", "dlib", "xeq", "saves", "savef", "alib", "end", "go", "f" /
common /miscbk/ n
common /dab k/ keep, initprm

parameter (maxsym=80)
integer symb(maxsym), type(maxsym), buffer(10)
data i, numsym
data i / 0 /, numsym / 0 /
data funct(9,300), nl
data funct / "f=", "0", "#*" /
data nl / 1 /
data prog(9,300), npl
data prog / "c", "0", "#*" /
data npl / 1 /
data buf(20)
if (i.lt.num sym) go to 20
main loop to interpret commands

10 continue
i = 0
call msgfrr (buffer, len, maxsym)
call getnumb (symb, type, numsym, buffer, len, maxsym)
type(numsym+1) = 7
20 continue
i = i + 1
51 go to (100, 400, 20, 420, 420, 420, 420, 420, 19), type(i)+1
53 continue
100 go to (120, 300, 280, 260, 240, 230, 220, 200, 180,

-44-
160. 140), iscmd(symb(i).command,ncmd)+1

55  c
56  c controllee assumed
57  c
58  120  continue
59      call initc ( symb(i), 60000000000.0, 0., irec )
60      if ( irec.ne.0 ) go to 130
61      call bypass (’1, ’)
62      ns = -1
63      if ( type(i+1).eq.7 ) go to 125
64      call shiftn ( buffer, len )
65      ns = len
66      125  continue
67      call msgtoe ( buffer, ns, 0, 0, irec )
68      call msgfre ( buffer, len, maxsym, irec )
69      call getsymb ( symb, type, numsym, buffer, len, 4 )
70      if ( symb(3).eq.”all” .and. symb(4).eq.”done” ) go to 10
71      call msgtor ( buffer, len )
72      go to 10
73      continue
74      call msgtor ( ”invalid command disregarded”, 30 )
75      go to 20
76 77  c f command
78  c
79  140  continue
80      if ( type(i+1).ne.0 ) go to 155
81      call readfct ( ious, symb(i+1), funct, ni, irec )
82      if ( irec.lt.0 ) go to 155
83      i = i + 1
84      go to 20
85      155  continue
86      call getfct ( funct, ni )
87      go to 20
88  c
89  150  continue
90      if ( type(i+1).ne.0 ) return
91      if ( iscmd(symb(i+1).command,ncmd).gt.0 ) return
92      i = i + 1
93      initprm = symb(i)
94      return
95  c
96  160  continue
97      term = 0
98      call putprog ( iou, itext, funct, ni, 1 )
99      if ( icall.eq.1 ) call putprog ( iou, itext, prog, npl, 2 )
100     call close ( iou )
101     icall = 0
102     if ( type(i+1).ne.0 ) return
103     if ( iscmd(symb(i+1).command,ncmd).gt.0 ) return
104     i = i + 1
105     initprm = symb(i)
106     return
107  c
108  180  continue
109     term = 1
109 c return
110 c alib command
111 c
112 c
113 200 continue
114 call zmovchr (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 zmovchr (libuf, newlib, ":", 0, 1)
122 newlib = newlib + 1
123 go to 205
124 210 continue
125 call zmovchr (libuf, newlib, buf, 0, nlib)
126 nlib = nlib + newlib
127 go to 20
128 c
129 c saves command
130 c
131 220 continue
132 if (type(i+1).ne.0) go to 225
133 i = i + 1
134 call saves (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 saves (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   260  continue
170   keep = 1
171   go to 20
172 c
173 c sub command
174 c
175   300  continue
176   icall = 1
177   if ( type(i+1).ne.0 ) go to 305
178   call readfct ( ious, symb(i+1), prog, npl, ilres )
179   if ( ilres.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 c end
subroutine shiftmsg (line, len)
implicit integer(a-z)
integer line(1), len

shiftmsg shifts off the first symbol from line. len is the number of characters in line.

parameter (blank=48)

sbk = zscanline (line, 0, len, blank)
loc = zscanleq (line, sbk, len-sbk, blank)
nbk = zscanline (line, sbk+loc, len-sbk[loc], blank)
len = len - sbk - loc - nbk
call zmovechr (line, 0, line, sbk+loc+nbk, len)
return
end
subroutine symove (destin, loc, source)
implicit integer(a-z)
integer destin(1), loc, source

move characters from source to destin(loc) until a blank is detected. loc is the offset in characters of destin

parameter ( blank=40b )
parameter ( cpu=8 )
n = zscanleq (source, 0, cpu, blank)
call zmovechr (destin, loc, source, 0, n)
loc = loc + n
return
end
subroutine driver

c the driver of the solver generated program

c external f
complex f, y(1024)
integer l(30), id(10), inlen, mapset, plotmode, plotid, of
integer grids, dbwg, first
complex c(30), p(1024, 100)
real x(1024), w(1024)
common /parmbook/ i, c, p, y
common /plotbook/ id, mapset, plotmode, plotid, x, w
common /miscbook/ n, maxit, known, h, epl, ep2, of
name list /initbook/ n, maxit, known, h, epl, ep2, of, 1, c, p, y,
common /miscbook/ n, maxit, known, h, epl, ep2, of
parameter (nmax=1024, nrmax=25)
real rootre(nmax, nrmax), rootai(nmax, nrmax)

c default values

21
data maxit /100/
data known /0/
data h /0.5/
data epl /0.1/
data ep2 /0.1/
data of /0/
data mapset /0/
data plotmode /0/
data plotid /0/
data xsize /6.0/
data ysize /4.5/
data grids /0/
data dbug /0/
data nx /nmax/
data nu /nrmax/

40
t nteger line(10), symb(1), type(1)
t nteger namef(2)
data namef /Srinout, 4rout/

c initializations
45
call setfile (namef, 2)
call dropfile (namef(1))
call msgflag (msg, 2)

c read input, if any
50
call msgfrr (line, len, 88)
call getnumb (symb, type, ns, line, len, 1)
if ( symb(1).ne."tty" ) go to 20
  call mprompt ( ":", 1 )
  read ( 59, initbk )
  call mprompt ( 0, 0 )
  go to 15
20 continue
  call open ( 5, symb(1), 0, inlen )
  if ( inlen.lt.1 ) go to 35
  read ( 5, initbk )
c create and write to output file, if of=1 or 2
15 continue
  if ( of.lt.1 .or. of.gt.2 ) go to 35
  namef(2) = 400b+namef(2) + 140b
10 continue
  namef(2) = namef(2) + 1
  call open ( 1, namef(2), 0, len )
  call close ( 1 )
  if ( len.gt.0 ) go to 10
  call create ( 6, namef(2), 2, -1 )
  write ( 59, 103 ) namef(2)
  write ( 6, 104 ) n, maxit, known, h, ep1, ep2, (y(i), i=1,n)
c check information
35 continue
  n = min0 ( n, 1024 )
  if ( of.gt.2 .and. plotmode.eq.0 ) plotmode = 1
  iplot = plotmode
  if ( iplot.gt.3 .or. mapset.gt.3 ) n = 1
  np = nx
  nr = nw
  if ( np*nr.le.25600 .and. mapset.gt.3 ) go to 40
  np = npmax
  nr = nrmax
c main loop
40 continue
  call wsetup
  if ( plotmode.gt.0 .and. first.eq.1 )
    call rootplot ( rootre, rootai, np, nr )
  if ( I(30).lt.0 ) go to 70
  first = 1
100 if ( iplot.gt.3 ) go to 60
  num = n
  call muller ( known, num, y, maxit, h, ep1, ep2, f, kount )
  if ( of.gt.2 ) go to 40
  if ( of.eq.2 ) go to 50
  write ( 59, 101 ) ( y(i), i = 1, num+known )
  if ( kount.gt.maxit ) write ( 59, 102 ) kount
  if ( of.eq.0 ) go to 40
write ( 6, 101 ) ( y(i), i = 1, num+known )

if ( kount.gt.maxit ) write ( 6, 102 ) kount

continue
evaluating function instead of finding roots

ixind = ixind + 1

y(1) = f ( x(ixind) )

go to 40

to 40

c

60 continue

c evoluo ' ir- 3

funCti On

instc -cn of finding roots

c

70 continue

if ( iplot.eq.0 ) call exit ( dbug )

plotmode = iplot

call rootplot ( rootre, rootai, np, nr )

call donepl

call exit ( dbug )

c

101 format ( "roots:" / ( " (",e19.11,"",e19.11,"",)* ) )

102 format ( "does not converge after", i5, " iterations" )

103 format ( "a8, "is the output file" )

104 format ( "n =", i4 / "maxit =", i4 / "known =", i4 / 

h =", f15.7 / "ep1 =", f15.7 / "ep2 =", f15.7 / 

y =", 2(f15.7, *, *) / ( 7x, 2(f15.7, *, *)) )

end
subroutine curves ( x, y, np, mark, nc )
real x(1), y(1:24,1)
integer np, mark, nc
c
draw curves

do 10 i = 1, nc
   call curve ( x(1), y(1,i), np, mark )
10 continue
return
end
subroutine destin (plotmode, plotid)
integer plotmode, plotid
plotmode = 1 => fr60
   = 2 => prtpit
   = 3 => tk
plotid for <fr60>  => id
   for <prtpit>  => line width
   for <tk>  => baud rate
mode = plotmode
if ( mode.lt.1 ) return
if ( mode.gt.3 ) mode = mod ( mode, 4 ) + 1
go to ( 10, 20, 30 ), mode
fr60 file
10 continue
call keep60 ("roots", 3)
call fr60id (plotid, 1, 1)
call prtpit
return
printer
20 continue
call prtpit (plotid)
return
Tektronix
30 continue
call tk (0,1,plotid, 59)
return
end
subroutine motype (mapset, npoint, xmin, xmax, ymin, ymax, xsize, ysize)

integer mapset, npoint
real xmin, xmax, ymin, ymax, xsize, ysize

mapset = 0 => linear-linear
1 => linear-log
2 => log-linear
3 => log-log
4 => contour

parameter (nusp=16384)
common work(nusp)

map = mapset
if (map.lt.0) map = 0
if (map.gt.4) map = 0
nxsizc = (xsize+l)/2
yysize = (ysize+l)/2
xinc = stepsz (xmin, xmax, 2*xsize, xd)
yinc = stepsz (ymin, ymax, 2*ysize, yd)
xcyc = xsize / ncyc (xmin, xmax, xdec)
yycyc = ysize / ncyc (ymin, ymax, ydec)
go to (10, 20, 30, 40, 50), map+1

10 continue
   call graf (xmin-xd, scale, xmax+xd, ymin-yd, scale, ymax+yd)
   return

20 continue
   call ylog (xmin-xd, xinc, ydec, yycyc)
   return

30 continue
   call xlog (xdec, xcyc, ymin-yd, yinc)
   return

40 continue
   call loglog (xdec, xcyc, ydec, yycyc)
   return

50 continue
   xinc = xinc - 2.*xd / max0 (2*xsize, 4)
yinc = yinc - 2.*xd / max0 (2*ysize, 4)
call graf (xmin, xinc, xmax, ymin, yinc, ymax)
call bcomon (nusp)
call conlin (0, "solid", "labels", 1, 10)
call conlin (1, "dash", "nolabels", 1, 8)
call conlin (2, "chndot", "nolabels", 1, 5)
call conlin (3, "dot", "nolabels", 1, 3)
call conang (60.)
return

end
subroutine muller (Kn, n, rts, maxit, step, ep1, ep2, fn, kount)

complex rts(1)
complex fn

c this subroutine uses the muller's method to find the roots of a function

c parameters:

Kn — number of roots previously computed, normally zero
n — input: number of roots desired
output: number of roots found
rts — an array of initial estimates of all desired roots
set to zero if no better estimates are available
maxit — maximum number of iterations per root allowed
step — initial distance between x's
ep1 — relative error tolerance on x(i)
ep2 — error tolerance on f(x(i))
fn — fn(z) is a external complex function which, for given
z, returns f(z)
kount — number of iterations of the last root. if kount
is equal to maxit, the routine will stop at once

complex rt, h, delipr, frtdef, lambda, delf, dfprlm, num, den
complex g, sqr, frt, frtprv

initialization
eps1 = amax1 (ep1, 1.e-12)
eps2 = amax1 (ep2, 1.e-20)
ibeg = Kn + 1
iend = Kn + n
do 100 i = ibeg, iend
kount = 0
compute first three estimates for root as
rts(i)+step, rts(i)-step, and rts(i)
1 h = step
rt = rts(i) + h
index = 1
10 go to 70
delfpr = frtdef
rt = rts(i) - h
index = 2
40 go to 70
50 go to 70
20 frtprv = frtdef
delfpr = frtprv - delfpr
rt = rts(i)
index = 3
49 go to 70
50 index = 4
lambda = 1.
compute next estimate for root
40 delf = frtdef - frtprv
dfprlm = delfpr * lambda
num = -frtdef * (1. + lambda) * 2.
g = (1. + lambda * 2.) * delf - lambda * dfprim
sqr = g * g + 2. * num * lambda * (delf - dfprim)
sqr = csqrt(sqr)
den = g + sqr

if (real(g) * real(sqr) + aimag(g) * aimag(sqr) .lt. 0.)
den = g - sqr

if (cabs(den) .lt. eps1) den = 1.
lambda = num / den
frtprv = frtdef
delfpr = delf
h = h * lambda
rt = rt + h
if (kount .gt. maxit) go to 200

70 70  kount = kount + 1
f0 = fn (rt)
frtdef = f0
do 71 j = 1, i-1
    den = rt - rts(j)
    if (cabs(den) .lt. eps2) go to 79
    f0 = frtdef / den
77  71  continue
79  75  go to (10, 20, 30, 80), index
    rts(i) = rt + .001
80  80  go to 1
81  c
82  c  check for convergence
83  80  if (cabs(h) .lt. eps1 * cabs(rt)) go to 100
84  c  if (amax1(cabs(f0), cabs(frtdef)) .lt. eps2) go to 100
85  c
86  c  check for divergence
87  c  if (cabs(frtdef) .lt. 10. * cabs(frtprv)) go to 40
88  c  h = .5 * h
     lambda = .5 * lambda
89  c  rt = rt - h
91  91  go to 70
92  100  rts(i) = rt
93  return
94  200  continue
95  c
96  c  return
97  end
integer function ncyc ( vmin, vmax, vdec )
real vmin, vmax, vdec

return the number of cycles needed for log scale

parameter ( one=.999999999999 )
parameter ( maxcyc=10 )
parameter ( base=1.8-20 )

delta = 0.
if ( vmin.lt.1. ) delta = 1.
min = alog10 ( amax1(vmin,base) ) - delta
max = alog10 ( amax1(vmax,base) ) + one
ncyc = max0 ( max-min, 1 )

vdec = 10. ** min
if ( vdec.gt.base**10. or. ncyc.lt.maxcyc ) return
ncyc = maxcyc
vdec = 10. ** ( max-maxcyc )
return

end
subroutine rootplot ( rootre, rootai, npnew, nrnew )
real rootre(npnew,nrnew), rootai(npnew,nrnew)
!
plots rootre vs. x and/or w and rootai vs. x and/or w
rootre and rootai are the real and imaginary part of the
roots (or the value of the function. if so, stored in
rootre(1) and rootai(1)). npnew is the maximum number of
points and nrnew is the maximum number of roots.
!
integer 1(30), id(10), inlen, mapset, plotmode, plottid, of
integer grids
real x(1024), w(1024)
complex y(1024)
complex c(30), p(1024:100)
common /parmbk/ l, c, p, y
common /plotbk/ id, mapset, plotmode, plottid, x, w,
common xsize, ysize, inlen, grids
common /miscbk/ n, maxit, known, h, epl, ep2, of
!
real psize(2)
parameter ( big=1.0e10, small=-1.0e10 )
integer first, npoint, line(11)
data remin, amin, xmin, umin /4*big/
data remax, amax, xmax, umax /4*small/
data line(11) /in$/
data psize /11.,0.5/
data nx /0., nw /25600/
!
integer labfre(2), labfa(i), labre(2), labai(2)
integer labuwe(2), labuai(2), labx(2)
integer labpu(2), labax(2), labau(2)
data labfre /"real(f)", "/
data labfa /"imag(f)", "/
data labre /"real(a)", "/
data labai /"imag(a)", "/
data labuwe /"real(w)", "/
data labx /"real(x)", "/
data labuai /"imag(a)", "/
data labuwai /"imag(w)", "/
data labfre /"", "/
data labai /"", "/
!
initializations
!
if ( first.ne.0 ) go to 5
first = 1
nr = min(0, n+known, nrnew )
xsz = xsize
ysz = ysize
!
check if page size should be 8.5"x11" or 11"x8.5"
!
if ( xsz.eq.ysz ) go to 5
if ( xsz.lt.ysz .and. ysz.le.psize(2)-1.5 ) go to 5
if ( xsz.ge.ysz .and. xsz.gt.psize(2)-1.5 ) go to 5
  temp = psize(1)
  psize(1) = psize(2)
  psize(2) = temp

store the roots of the function value, y

5 continue
  if ( 1(30).lt.0 ) first = first + 1
  if ( first.gt.2 ) go to 20
  if ( mopset.gt.3 ) go to 12
  nx = nx + 1
  if ( nx.gt.pnnew ) return
  do 10 i = 1, nx
    rootre(nx,i) = real ( y(i) )
    rootai(nx,i) = imag ( y(i) )
    if ( remin.gt.rootre(nx,i) ) remin = rootre(nx,i)
    if ( remax.lt.rootre(nx,i) ) remax = rootre(nx,i)
    if ( amin.gt.rootai(nx,i) ) amin = rootai(nx,i)
    if ( amax.lt.rootai(nx,i) ) amax = rootai(nx,i)
  10 continue
return

12 continue
  if ( nw.gt.nrnew ) go to 14
  nx = nx + 1
  nw = 0
14 continue
  nw = nu + 1
  rootre(nx,nw) = real ( y(i) )
  rootai(nx,nw) = imag ( y(i) )
  if ( remin.gt.rootre(nx,nw) ) remin = rootre(nx,nw)
  if ( remax.lt.rootre(nx,nw) ) remax = rootre(nx,nw)
  if ( amin.gt.rootai(nx,nw) ) amin = rootai(nx,nw)
  if ( amax.lt.rootai(nx,nw) ) amax = rootai(nx,nw)
return

find the minimum and maximum values of x and w

20 continue
  npoint = min0 ( nx, npnew )
  do 30 j = 1, npoint
    if ( xmin.gt.x(j) ) xmin = x(j)
    if ( xmax.lt.x(j) ) xmax = x(j)
  30 continue
20 continue
  if ( mopset.gt.4 ) go to 40
  do 35 j = 1, nw
    if ( wmin.gt.w(j) ) wmin = w(j)
    if ( wmax.lt.w(j) ) wmax = w(j)
  35 continue
  if ( remax.-remin.lt.1.e-25 ) nrea = 1
  if ( amin.-aimin.lt.1.e-25 ) nrea = 1
  remin = wmin
  amin = wmin
109 \text{remax} = \text{umax} \\
110 \text{aimax} = \text{umax} \\
111 \text{c} \\
112 \text{c figure out the title and name for the axes} \\
113 \text{c} \\
114 \text{40 continue} \\
115 \text{if ( plotmode.lt.4 .and. mapset.lt.4 ) go to 50} \\
116 \text{if ( plotmode.lt.4 ) go to 50} \\
117 \text{call move ( labx, labax, 2 )} \\
118 \text{call move ( labre, labfre, 2 )} \\
119 \text{call move ( labui, labfai, 2 )} \\
120 \text{if ( mapset.lt.4 ) go to 60} \\
121 \text{call move ( labre, labw, 2 )} \\
122 \text{call move ( labui, labw, 2 )} \\
123 \text{call move ( labre, labfre, 2 )} \\
124 \text{call move ( labfai, labfai, 2 )} \\
125 \text{go to 60} \\
126 \text{50 continue} \\
127 \text{call move ( labre, labw, 2 )} \\
128 \text{call move ( labfai, labw, 2 )} \\
129 \text{call move ( labw, labw, 2 )} \\
130 \text{call move ( labw, labw, 2 )} \\
131 \text{c getting ready to plot} \\
132 \text{c} \\
133 \text{60 continue} \\
134 \text{call setbox ( id, 80 )} \\
135 \text{call gfsizq ( 3, 2000000b )} \\
136 \text{call destins ( plotmode, plotid )} \\
137 \text{call nobdr} \\
138 \text{c} \\
139 \text{c plot the input file. if there is one} \\
140 \text{c} \\
141 \text{c} \\
142 \text{if ( inlen.le.0 ) go to 90} \\
143 \text{call page ( psize(1), psize(2) )} \\
144 \text{call tablet ( "tset", "long" )} \\
145 \text{rewind 5} \\
146 \text{70 continue} \\
147 \text{read(5,10;0) ( line(i), i=1,10 )} \\
148 \text{if ( ieof(5).ne.0 ) go to 80} \\
149 \text{call itline ( line )} \\
150 \text{go to 70} \\
151 \text{80 continue} \\
152 \text{call endtabl ( 0 )} \\
153 \text{call endapl ( 0 )} \\
154 \text{c} \\
155 \text{90 continue} \\
156 \text{xsz = amin1 ( xsz, psize(1)-1.5 )} \\
157 \text{ysz = amin1 ( ysz, psize(2)-1.5 )} \\
158 \text{call page ( psize(1), psize(2) )} \\
159 \text{c} \\
160 \text{c plot the real(root) vs. x} \\
161 \text{c} \\
162 \text{xstor = ( psize(1)***.5-xsz )***.75}
if ( xsort.lt.75 ) xsort = (psize(1)-xsz)*.5
ysort = (psize(2)*.5-ysz)*.5
if ( ysort.lt.5 ) ysort = (psize(2)-ysz)*.5
call physor ( xsort, ysort )
call title ( labtr, 16, labx, 16, labyre, 16, xsz, ysz )
call maptype ( mapset, npoint, xmin, xmax, amin, amax, ksz, ysz )
call dfmame
if ( grids.ne.0 ) call grid ( -grids, -grids )
if ( mapset.lt.4 ) go to 100
   if ( nora.eq.1 ) go to 120
   call contur ( 4, "labels", "draw" )
go to 110
100 continue
call curves ( x, rootre, npoint, -1, nr )
110 continue
call endgr ( 0 )
c c plot imaginary(root) vs. x
120 continue
   xsort = (psize(1)*.5 + (psize(1)*.5-xsz)*.5
   if ( xsort.lt.5+psize(1)*.5 ) xsort = (psize(1)-xsz)*.5
   ysort = (psize(2)*.5 + (psize(2)*.5-ysz)*.5
   if (ysort.lt.5+psize(2)*.5 ) ysort = (psize(2)-ysz)*.5
call physor ( xsort, ysort )
   if ( xsort.lt.psize(1)*.5 .and. ysort.lt.psize(2)*.5 )
      call dendpl ( 0 )
c c call title ( labtr, 16, labx, 16, labyre, 16, xsz, ysz )
call maptype ( mapset, npoint, xmin, xmax, amin, amax, ksz, ysz )
call dfmame
if ( grids.ne.0 ) call grid ( -grids, -grids )
if ( mapset.lt.4 ) go to 130
   if ( nora.eq.1 ) go to 140
   call contur ( 4, "labels", "draw" )
go to 140
130 continue
call curves ( x, rootai, npoint, -1, nr )
140 continue
call endgr ( 0 )
call dendpl ( 0 )
c return
1010 format ( 10a8 )
c end
subroutine setfile (names, n)

subroutine setfile appends the current suffix (channel) to all the given files.

parameters:
  names — an one-dimensional integer array with file names in it. all names are limited to 7 characters.
  n — an integer that indicates the number of files.

library used:
  forlib

integer names(n), n
integer userno, account, df, suffix, i

set unit 59 to be the controller or terminal

call msglink (59, 1)

c get the suffix

call userinfo (userno, account, df, suffix)

c change the left-justified suffix to right-justified

suffix = suffix / 40000000000000000b

append the suffix to all file names

do 10 i = 1, n
   names(i) = 400b * names(i) + suffix
10 continue

return

end
real function stepsz ( vmin, vmax, size, delta )
real vmin, vmax, size, delta
c find step size for a graph
delta = aminl( 0.001, (vmax-vmin)*.05 )
stepsz = (vmax-vmin+2.*delta) / max0 ( int(size), 4 )
return
end
subroutine vvsetup
complex c(30), p(1024,100), y(1024)
integer i(30)
common /parmbk/ l, c, p, y
data i(30) /1/

c
l(30) = l(30) - 1
return
end