The W7-AS data acquisition and automatic antenna matching system

Marc Ballico

IPP 4/264 October 1993

MAX–PLANCK–INSTITUT FÜR PLASMAPHYSIK

85748 GARCHING BEI MÜNCHEN

MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

The W7-AS data acquisition and automatic antenna matching system

Marc Ballico

IPP 4/264 October 1993

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

The W7AS data acquisition and automatic antenna matching system.

27/10/1993

by Dr Mark Ballico.

Contents:

Abstract:

An automatic system for measuring the complex antenna impedance and calculating the optimal tuning stub positions for the ICRH antennas on W7AS is presented.

Introduction:

Present ICRH antennas consist of a poloidal current strap fed from a coaxial feed line. The radiation resistance of this strap to the plasma is typically very low, of order 1Ω/m, leading to a high VSWR in the transmission lines feeding the antenna. Since the high power RF generators supplying the power can only operate into low VSWR lines (50 ohms), some form of matching is required. This is usually achieved by a so-called double stub tuner, consisting of adjustable transmission lines terminated in short-circuits, connected in parallel to the tranmission line at two points about a quarter wavelength apart. By appropriate choice of length of the two short-circuited lines or *stub tuners* an arbitary antenna load can be *matched* so that is appears as 50 ohms to the RF generator [Cheng p431]. When the antenna loading is low, resulting in a high VSWR, the positions of the stubs must be very accurately determined.

Previous solution:

Previously on W7AS the procedure for matching the antenna was to firstly determine by trial and error at low power the stub positions resulting in matching for an empty torus, the so-called *vacuum matching*. During the plasma, the antenna loading, both resistive and reactive, is very different and the antenna is again mis-matched, resulting in large reflected power to the RF generator. On a shot by shot basis, the matching would then be improved. Since only the magnitude of the forward and reflected power was measured, this effectively meant determining the minimum position of a function $|\Gamma(S_1, S_2)|$, varying alternately stub 1 and stub 2, and converging to the matching point. This procedure would usually take 6 or more shots to achieve a few % reflected power. When the plasma conditions changed, due for example to a change in plasma density or position, this shot-by-shot matching procedure must be repeated. Since during a typical experimental programme electron density scans (changing the density on a shot by shot basis) are frequently used, this can substantially impair the effectiveness of ICRH. On W7AS operating at full toroidal field, the shot interval is 15 minutesso these *matching* shots represent wasted experimental time. Additionally, with 2 ICRH systems, keeping track of the matching positions was an additional load to the experimentalist.

Proposed solution:

Previously on W7AS a program using measurements of the complex reflection coefficient was used to calculate the correct matching position for the stubs was developed by A. Murphy and G. Cattanei. This system however, used data on the CMS central computer and so the data was available only just before the next shot. Further, the program required user input of the existing stub positions and manual control of the stub tuners.

It was decided to develop a quicker, automated matching system. This consisted of several parts.

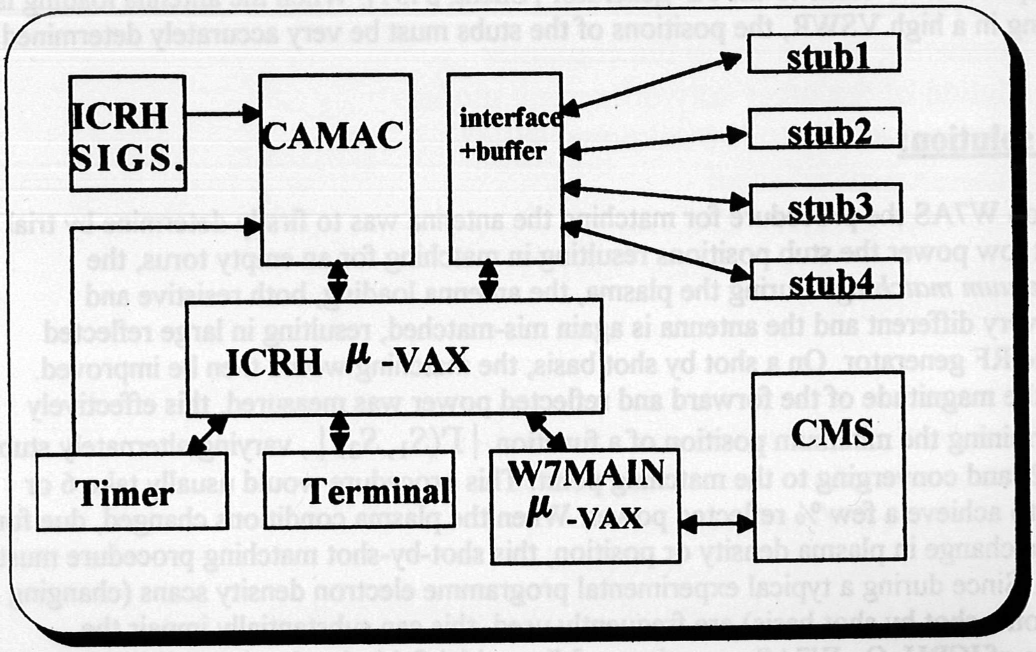
(1) A new ICRH data acquisition system, based on a $\mu-$VAX on the W7AS local area nework.

(2) Interfacing of the stub tuners to the new computer system.

(3) Determination of an accurate circuit model for the double stub tuner system.

(4) A computer program and user interface for measuring the antenna impedance and calculating an appropriate position for the stub tuners.

This system has been in use on W7AS since the beginning of 1992.



A new data acquisition system:

The previous data acquisition system of W7AS was based on a PDP-11 located in the old ASDEX generator control room, and connected by optical fibre to a CAMAC crate in the W7AS experiment control room. Rectified power and voltage signals were connected to this CAMAC. The ICRH data on the PDP-11 was not accesible to the W7AS computers, and hence was not incorporated in the shotfile, and not accessible to other users.

It was decided to update the data processing facilities for ICRH on W7AS, and so a DEC MICROVAX was purchased (by the W7AS group) and connected to the existing CAMAC digitizer crate. This had the advantage of full compatibility to the data acquisition system on W7AS, which is based on these machines. The data acquisition software UDAS developed by the W7AS group was then simply ported to this machine. UDAS allows the ICRH computer to operate either in *local* or *online* mode. In *local* mode, ICRH test shots can be made and the data analysed at any time, allowing the flexibility to test the whole system and perform calibrations etc. on non-shot days. In *online* mode the ICRH computer stores data during a shot, and then delivers this data to a host computer to be incorporated into the total shot-file. In this way ICRH data is (i) available to programs on the ICRH computer within seconds of the shot end, and (ii) to other users about 90 seconds later.

The Configuration file:

It was decided to also incorporate the calibration factors for the various ICRH signals, within the shotfile itself. This was necessary in order to ensure that (i) ICRH data for old shots, which may no longer be stored on the local ICRH computer, can still be evaluated. Also, it (ii) allows other users to easily evaluate the ICRH signals if necessary, and (iii) flexibly allows change to the ICRH configuration, detectors, patching and calibrations. This was achieved by extending the *configuration* file facility offered by the UDAS system. This is a text file UD$SYS:CONFIGURATION.OLD containing information on the settings of the digitizers, together with text comments from the user. It is stored along with signals recorded by the digitizer, in each shotfile. Information stored in the configuration file, is easily changed during an experiment using a standard text editor and is accessible to user-programs, so a table of calibrations or even the patching of the signals can be changed during an experiment, and this information can be used by the analysis program later to properly evaluate the data. The data within the configuration file consists of 5 parts;

(i) whether the experiment is in online or local mode.(supplied by the W7AS group).
(ii) comments from the user, specifying the ICRH system configuration.
(iii) a patch table (ICRH specific).
(iv) calibration arrays (ICRH specific).
(v) digitizer and timer settings (supplied by the W7AS group).

A sample configuration file is given in the appendix.
The comments section stores important information about the ICRH system that the automatic matching system needs to know, such as the operating frequency and the lengths of transmission lines in the stub tuner system. The system is currently configured for 2 antennas, each with a double stub tuner, so each of the following variables has 2 values, one for each antenna.

stub1_vac	vacuum matching position for stub 1.
stub2_vac	vacuum matching position for stub 2.
stub1_stub2	distance between stub tuners in mm.
stub1_offset	zero offset to stub 1 tuner readout in mm.
stub2_offset	zero offset to stub 2 tuner readout in mm.
stub2_dc	distance between stub 1 and the directional coupler in mm.
freq	frequency in MHz

The patch table has one line for each ICRH signal, and has the following form,

NAME DIGITIZER TYPE:DETECTOR ATTENUATION UNITS

where,

NAME is a text string, specifying the name of the particular signal.
DIGITIZER specifies which digitizer channel the signal comes from.
TYPE (either RAW, DET or CAL) indicates what sort of calibration table is to be used.
DETECTOR is a text string specifying the name of detector used for the signal.
ATTENUATION specifies the attenuation between the signal being measured and the detector, it can have one the following two forms, illustrated by example here;

(a) dB67.4 multiply the signal by $10^{67.4/20}$
(b) x104.3 multiply the signal by 1004.3

Prefixing the attenuation factor by "s" (eg. sx10.4) takes the square root of the data after calibration, which is typically used for converting powers to voltages.

UNITS is a text string giving the units of the signal eg. *V*, *W*, *Torr* etc.

the usual ICRH signals defined are:

pfwd1 pref1 pfwd2 pref2	forward and reflected power signals for system 1 and 2
vmax1 vmax2	maximum voltage in the 50 ohm line for system 1 and 2
sin1 cos1 sin2 cos2	sin and cos of the phase between forward and reflected signals for system 1 and 2

I_top_1 , I_bot_1 , I_top_r , I_bot_r : current signals for the 4 ICRH antenna ports.
However further signals can be defined and their calibrations specified simply by adding a line to the patch table.

Calibration arrays store the calibration factors for possibly non-linear detectors. Two different types of calibration curves are supported, depending on what value *TYPE* is assigned.

(i) RAW
No calibration table is used, only the attenuation factor given.

(ii) DET
Suitable for power detectors (eg. power and voltage signals). The corresponding calibration table has the following form.

*NAME** Pmax Pstep Npoints
NAME-1 v_1 v_2 v_3 ......
NAME-2 v.......
NAME-3 ....

where *NAME* is the name of the calibration curve refered to in the patch table, Pmax is the maximum power in dBmW used in the calibration, Pstep is the power step in dB between calibration points, and Npoints is the number of calibration points. v_1, v_2.... etc. is the output voltage of the detector from the detector in decreasing order.

(iii) CAL
General calibration curve (e.g. gas pressure or phase signals). The calibration table shold have the following form.

*NAME** Npoints
*NAME*x-1 x_1 x_2 ....
*NAME*x-2 ....
*NAME*y-1 y_1 y_2 ....

NAMEy-2 ....

where Npoints is the number of data points, and $y_i(x_i)$ is the physical signal y_i for a measured digitizer voltage x_i. Note that the x_i must be in decreasing order.

Data retrival software:

A function ICDATA(name,n,t,y) was written (M. Ballico) and incorporated by J. Saffert into the W7AS standand W7AS library W7FU , and is available on all microvax systems on W7AS and on the IBM mainframe system. Its is also incorporated into the general data plot package NMUL run by the W7AS group.

Inputs:

name	a character string containing the name of the ICRH signal required (eg. pfwd1)
n	an integer containing the number of data points required.
t	real array t(n) containing the times at which data is requested.

outputs:

y	real array y(n) with the data y(t(i))
dim	character array containing the dimensions of data returned in y
icdata	integer success flag (0=no error).

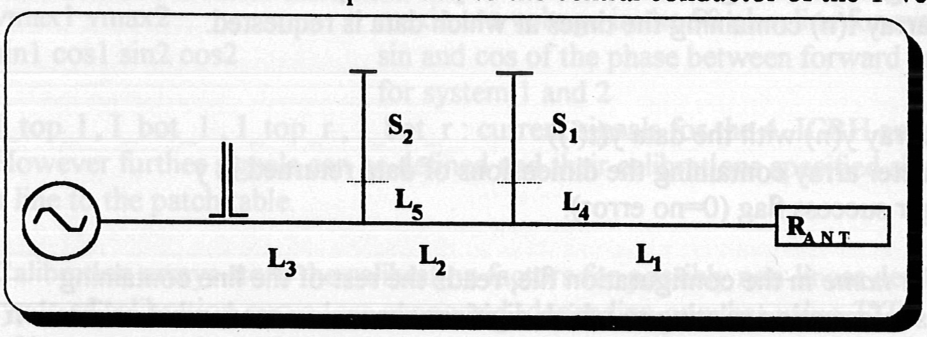
This routine looks for *name* in the configuration file, reads the rest of the line containing it and interprets it as information relating to which digitizer channel to read, what calibration factor to use and which calibration table to use. It the reads an array of data from the specified channel, searches the configuration file for the appropriate calibration table and calibrates the signal.

Interfacing of the stub tuners to the computer.

It was desired that the optimization of matching be entirely automatic, so it was necessary to for the computer to be able to both read the stub tuner positions and to set new positions. The stub tuners code the values for both input and output as two 8 bit bytes at 9600 baud, with a bit flag in each byte telling whether it is the high or low byte. The current tuner position is given as a continuous data stream, and immediately on reciept of valid input data, the tuner moves to the new position. Each tuner is simply connected to a spare serial port on the ICRH Microvax. Fortran subroutines were written to (i) initialize the ports *OPEN_RL100* (ii) write new stub information in the correct form *WRITE_RL100* , and (iii) read the current stub position and decode it *READ RL100*. It was not possible to make the UDAS data acquisition system read the stub positions directly, so an indirect technique was used. The automatic matching system, *MATCH*, which is left running during the shot day, reads the stub positions once per second, and writes the information to a file *UD$SYS:STUB.STATUS* . UDAS provides a facility to read data from a file and store it as shot data, which can then be recalled in the same way as data from a CAMAC module.

Determination of the tuner network circuit model.

The electrical circuit model for the stub tuners is at first glance trivial, however, the generally high VSWR in the unmatched section of line requires very accurate determination of the lengths of the lines involved, typically a stub movement of a few cm is sufficient to move to a complete mismatch. The mechanical lengths of the lines proved to be insufficiently accurate, and a more refined model was required. Some reasons for the discrepancy between the mechanical and electrical line lengths include
(i) the presence of ceramic spacers, where the high epsilon of the spacer results in a lower wave speed.
(ii) the presence of 90 degree bends, with a radius of curvature of the same order as the coaxial line diameter.
(iii) The physical size of the T-junction connecting the stub tuners to the line, and which constituted a discontinuity in the coaxial line fields, e.g. the hole in the outer conductor of the main line and the field disruption due to the central conductor of the T-vertical.



The manufacturers of coaxial lines have as a main concern keeping the VSWR in matched lines low, and so *compensate* the discontinuities caused by (i) and (ii) by appropriate tailoring of the contours of the inner conductor, so that the impedance of the line remains 50 ohms, and so no reflections occur at the discontinuity. It is appropriate then, to replace the mechanical lengths by effective electrical lengths. The discontinuity due to (iii) is somewhat harder to consider. One possibility is to consider it as an ideal T with possible uncompensated lumped elements such as excess inductance. Because the impedance due to the plasma, transformed to the middle of the antenna, is typically $R_{ANT} \approx X_{ANT} \approx 1\Omega << 50\ \Omega$, the standing wave minima and maxima do not change position appreciably during a plasma, it is equivalent to replace this lumped element by an appropriate short section of transmission line ($l<\lambda$). The model used for the stub tuner network then , is simply that effective electrical lengths are to be used instead of the mechanical lengths. The length difference in the cables from the directional coupler to the detector is incorporated into L_3. The problem, then is to determine these unknown lengths to sufficient accuracy. The obvious method, of measuring the individual parts directly, was considered not feasable for several reasons.
(i) it would require substantial build down and build up of the rather physically large and heavy transmission line systems.
(ii) the discontinuities introduced by the conversion form 6" line to N-Type connectors was considered to introduce excessive uncertainties.
The solution chosen was to measure the complex reflection coefficient Γ at the directional coupler on the generator side of the stub tuners, as a function of the stub tuner positions, around the vacuum matching point. The unknown lengths can then be determined from solution of the resulting set of equations.
The initial approach used was to (i) vary stub 2 up and down, which gives unknowns L_3 and L_5 and then (ii) vary stub 1 , which gives unknown L_2 and L_4. This approach proved to be unstable, since the errors in (i) made the solution of the second set of equations subject to too

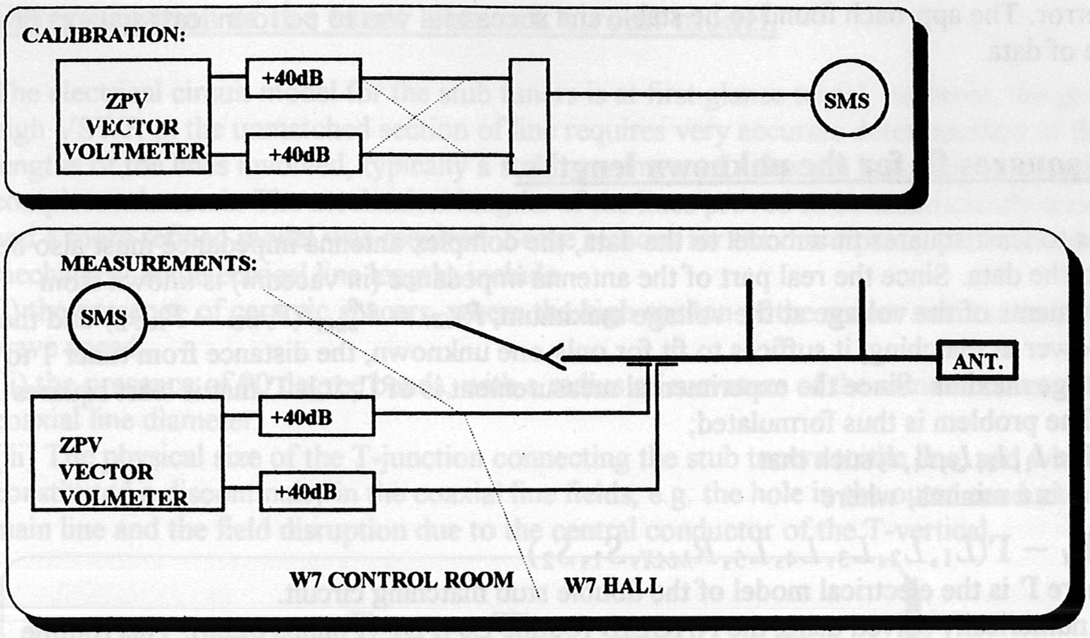
much error. The approach found to be stable and successful was to perform least squares fit to a range of data.

Least squares fit for the unknown lengths.

In order to least squares fit a model to the data, the complex antenna impedance must also be fitted to the data. Since the real part of the antenna impedance (in vaccum) is known from measurements of the voltage at the voltage maximum, $R_{MIN} = V^2_{MAX}/(P_{FWD} - P_{REFL})$ and the input power at matching, it suffices to fit for only one unknown, the distance from tuner 1 to the voltage maxima. Since the experimental measurement is of Γ, it is Γ that is least squares fitted. The problem is thus formulated;
determine l_1, l_2, l_3, l_4, l_5 such that
$\sum_i |R_i|^2$ is a minima, where
$$R_i = \Gamma_i - \Gamma(L_1, L_2, L_3, L_4, L_5, R_{MAX}, S_1, S_2)$$
and where Γ is the electrical model of the double stub matching circuit.
This is numerically solved using the NAGLIB routine E04FDF (Phillips p228). This routine requires an initial guess in $L_1 ... L_5$ space to start the minimization, and there is the danger that this initial will lead to the minimization routine *falling into* a local minimum, instead of the desired global minimum. This is a standard problem in least squares minimizations, the solution is (i) to start as close as possible to the minimum position and (ii) to randomly vary the start position and ensure the the routine always finds the global minimum. Another problem is that so-called outlying data or data with an abnormally large error, will tend to have a unfairly dominant role in the weighting in the least squares fit. This is usually solved by looking at the statistical distribution of residuals and deleting those data with residuals more than 3 times the standard deviation, and once more minimizing.
i.e. delete point i if $|R_i|^2 > \sigma_i = \sum_i |R_i|^2$
This procedure is followed by the fortran program CALCLEN (see appendix).

Measurement protocoll.

The ZPV Vector-Voltmeter is used for the measurements. Since the directional coupler has a very low coupling factor (-65dB), and the calibrations must take place at very low power, typically 20mW (between 0.5W and several kW multipactering in the antenna makes the antenna impedance change), an amplifier is required in the signal lines. Since the amplitude and phase reponse of the two ampifiers will not be identical, a calibration value, "Γ =1 at 0 deg" is required, this is achieved using a signal split and sent to the two ampifiers, the splitter outputs are then exchanged and a second value is taken, any inequality in the splitter is then cancelled:
$$|\Gamma_0|^2 = |\Gamma_A||\Gamma_B|$$
$$\arg(\Gamma_0) = \tfrac{1}{2}(\arg(\Gamma_A) + \arg(\Gamma_B))$$
Gamma is then measured for stub positions around vacuum matching, typically varying first stub 1 in steps in both directions until gamma is about 0.9 and then repeating with stub 2. Usually about 30 data points are taken. The program CALCLEN is then used to evaluate the unknown network parameters.



Description of the use of program CALCLEN.

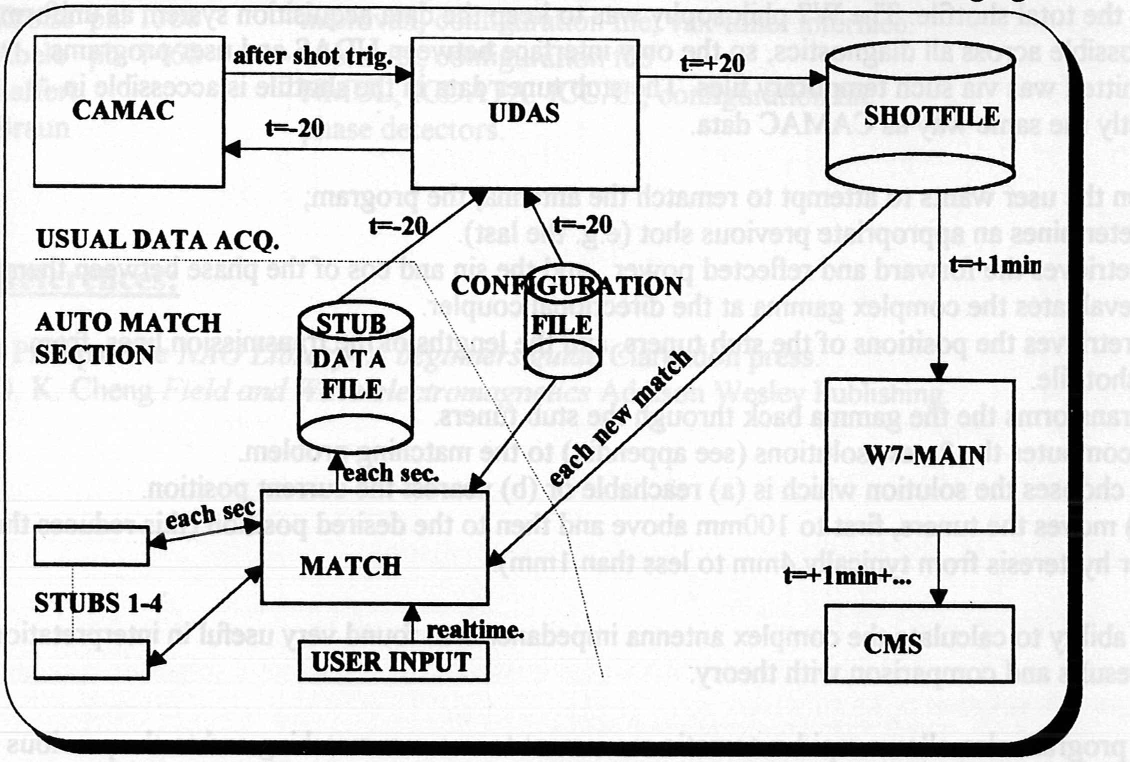
The program source is written in standard fortran. It requires as input a data file with the following format, (sample given in appendix).

f(Hz) , R_{MAX} (Ω)
$|\Gamma_0|$ (dimensionless) , arg (Γ_0) (degrees)
L_1, L_2, L_3, L_4, L_5 (m)
S_1 (mm) , S_2 (mm), $|\Gamma_1|$ (dim.less) , arg (Γ_1) (degrees)
S_1 (mm) , S_2 (mm), $|\Gamma_2|$ (dim.less) , arg (Γ_2) (degrees)
....
S_1 (mm) , S_2 (mm) , $|\Gamma_N|$ (dim.less) , arg (Γ_N) (degrees)

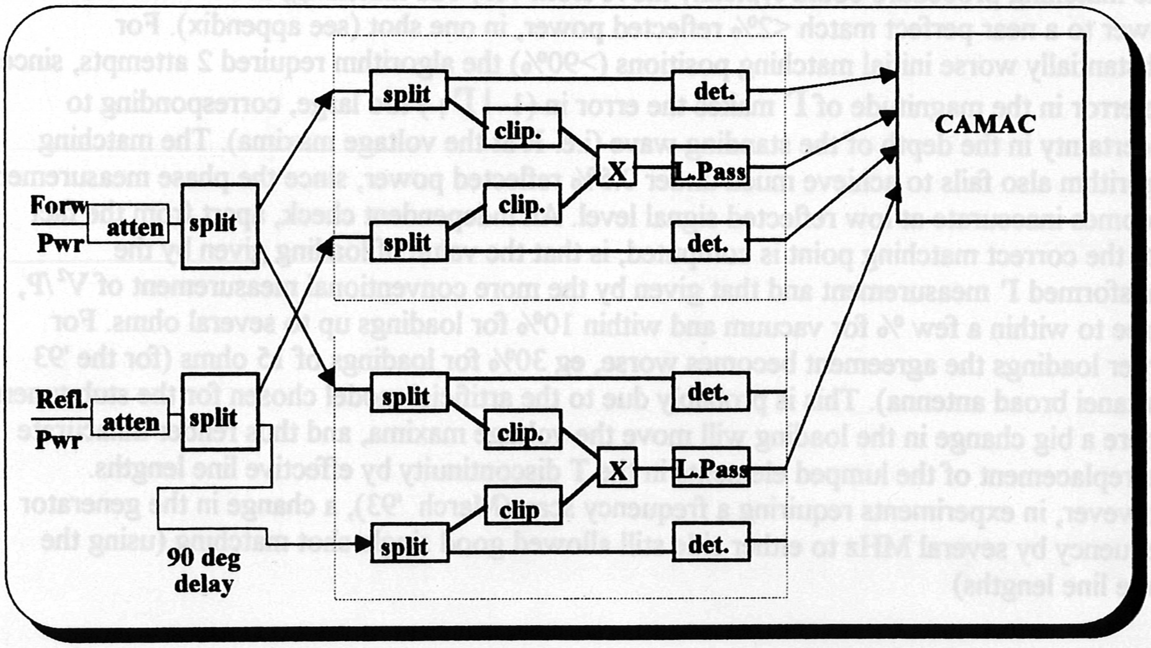
When the program is run it will ask the user for (i) the name of the data file containing the information about the stub tuners, (ii) a cutoff gamma above which data will be ignored (eg. 0.9) and (iii) how many random starts and how far from the initial guess to try (eg. 50 and λ/10)

The automatic matching program.

The information and dependencies for the program are shown in the following figure.



The complex gamma was measured using phase detector boxes built by F. Braun (ICRH group). These detectors use diode detectors for the signal amplitude, limiting amplifiers and a mixer to measure the cosine of the phase between the channel $V_{OUT} = k \cos \phi$. Since a 4-quadrant measurement of gamma was required, two such boxes were used, together with a 90 degree delay line, to get sin and cosine. This also overcome an inherent disadvantage of this type of phase detector, that they are inaccurate near 0 or 180 degrees, due to the flat charactersitic there ($\frac{dV_{OUT}}{d\phi} = 0$). The phase detector signal nearest to 90 degrees is used for the principal part of the phase and the other detector for the quadrant of the phase.



The MATCH program reads the current tuner positions each second and stores the current position in a temporary file. The UDAS data acquisition system then stores this data along with the CAMAC and configuration data file in the ICRH shotfile, which is then combined with the total shotfile. The W7 philosophy was to keep the data acquisition system as uniform as possible across all diagnostics, so the only interface between UDAS and user programs permitted was via such temporary files. The stub tuner data in the shotfile is accessible in exactly the same way as CAMAC data.

When the user wants to attempt to rematch the antenna, the program;
(i) determines an appropriate previous shot (e.g. the last).
(ii) retrieves the forward and reflected power , and the sin and cos of the phase between them.
(iii) evaluates the complex gamma at the directional coupler.
(iv) retrieves the positions of the stub tuners, and the lengths of the transmission lines, from the shot file.
(v) transforms the the gamma back through the stub tuners.
(vi) computes the 2 new solutions (see appendix) to the matching problem.
(vii) chooses the solution which is (a) reachable or (b) nearest the current position.
(viii) moves the tuners, first to 100mm above and then to the desired position (this reduces the tuner hysteresis from typically 4mm to less than 1mm).

The ability to calculate the complex antenna impedance was found very useful in interpretation of results and comparison with theory.

The program also allows rapid automatic movement to vacuum matching and to the previous shot's matching, which was found very useful for antenna conditioning between shots.

An interface to DEC-WINDOWS with submenus is also provided by the program. In usual operation one would open a new window on the terminal for the program to use.

Typical results:

The matching procedure could typically move from very bad matching, of 80% reflected power to a near perfect match <2% reflected power, in one shot (see appendix). For substantially worse initial matching positions (>90%) the algorithm required 2 attempts, since the error in the magnitude of Γ makes the error in $(1-|\Gamma|)$ too large, corresponding to uncertainty in the depth of the standing wave (i.e. R at the voltage maxima). The matching algorithm also fails to achieve much under 0.5% reflected power, since the phase measurement becomes inaccurate at low reflected signal level. An independent check, apart from the fact that the correct matching point is computed, is that the value of loading given by the transformed Γ measurement and that given by the more conventional measurement of V^2/P, agree to within a few % for vacuum and within 10% for loadings up to several ohms. For larger loadings the agreement becomes worse, eg 30% for loadings of 15 ohms (for the '93 Cattanei broad antenna). This is probably due to the artificial model chosen for the stub tuners, where a big change in the loading will move the voltage maxima, and thus render innacurate the replacement of the lumped elements in the T discontinuity by effective line lengths. However, in experiments requiring a frequency scan (March '93), a change in the generator frequency by several MHz to either side still allowed good single shot matching (using the same line lengths)

Contact people:

Kneidl	- stub tuners + tuner-vax interface.
Kroiss ph. 1308	- microvax, configuration file, vax-tuner interface.
Abele ph. 1462	- microvax, configuration file
Saffert	- NMUL, ICDATA, ICCAL, configuration file
Braun	- phase detectors.

References:

J Phillips *The NAG Library- A beginners guide* Clarendon press.
D. K. Cheng *Field and Wave electromagnetics* Addison Wesley Publishing.

Appendix (1) transformation through the tuner network:

The transformation from the load impedance Z to the measured reflection Γ coefficient can be expressed as follows , defining

$$f(Z) = \frac{1-Z}{1+Z}$$

we have

$$\Gamma(Z) = C_3^{-1} f\left[-Y_2 + f\left[C_2^{-1} f\left[-Y_1 + f\left[C_1^{-1} f(Z_0/Z) \right] \right] \right] \right]$$

where

$$Y_1 = i \tan^{-1} \beta(S_1 + L_4) \quad Y_2 = i \tan^{-1} \beta(S_2 + L_5)$$

$$C_1 = e^{i\beta L_1} \quad C_2 = e^{i\beta L_2} \quad C_3 = e^{i\beta L_3}$$

$$Z_0 = 50\Omega$$

noting that the operator f transforms between admittance and reflection coefficient in both directions. Transforming the other way, we have;

$$Z = Z_0 / f(C_1 f(Y_1 + f(C_2\, f(Y_2 + f(C_3 \Gamma)))))$$

Appendix (2) : Solution for the stub tuner positions:

Given an admittance Y at the stub tuner nearest the antenna (stub 1), the equation for determining the correct stub tuner admittances is quadratic, giving either 2 solutions, a single degenerate solution or no solutions.

$$A = C_2^{-1} \left[1 \pm \sqrt{Y_r \left[1 + C_2^2 (1 - Y_r) \right]} \right]$$

$$Y_1 = Y_i - A$$

$$Y_2 = \frac{(A + C_2)(1 - AC_2) - Y_r^2 C_2}{(1 - AC_2)^2 + (Y_r C_2)^2}$$

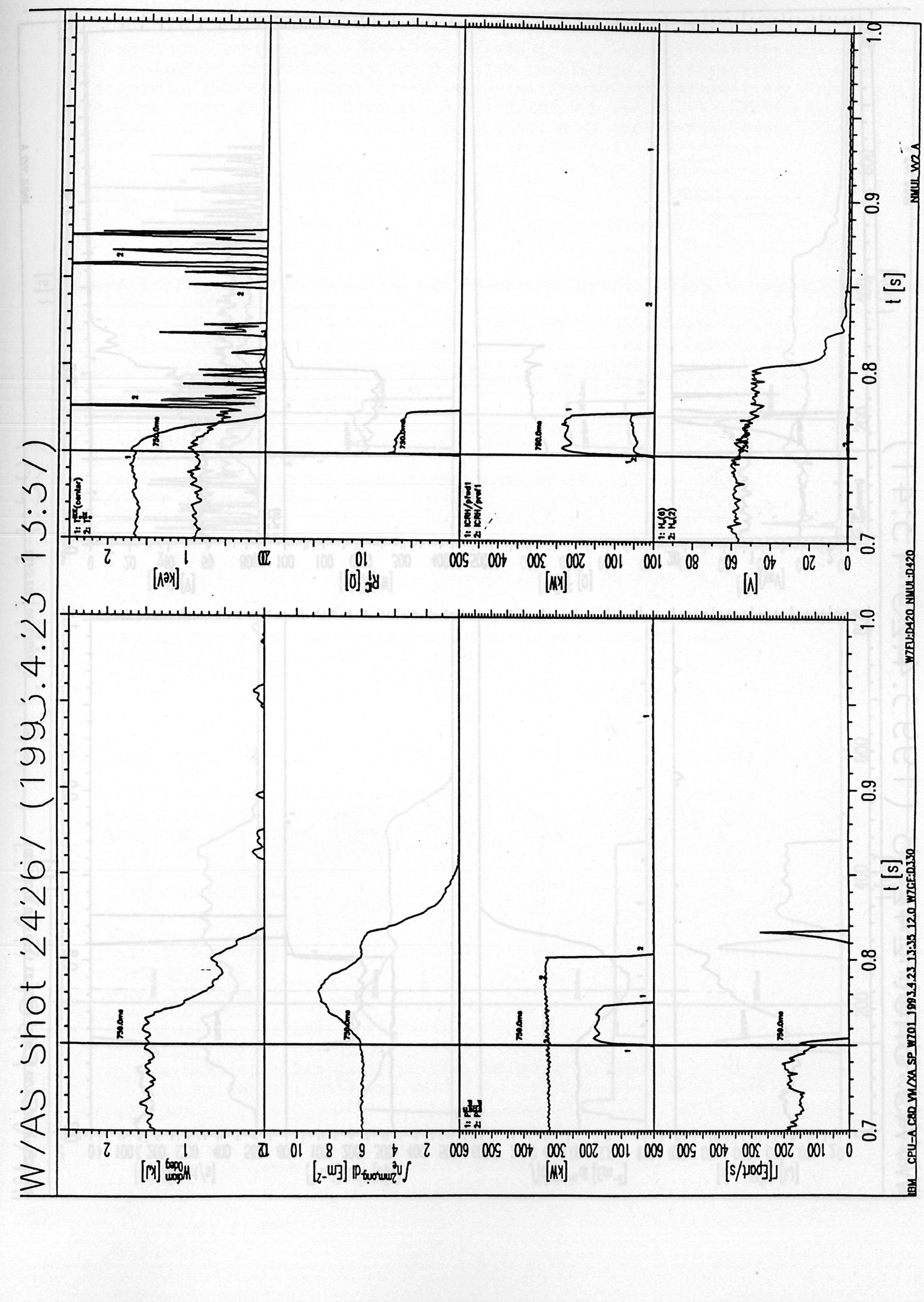
and the corresponding tuner positions are;

$$S_1 = \beta^{-1} \tan^{-1}(Y_1) \;\; - L_4$$

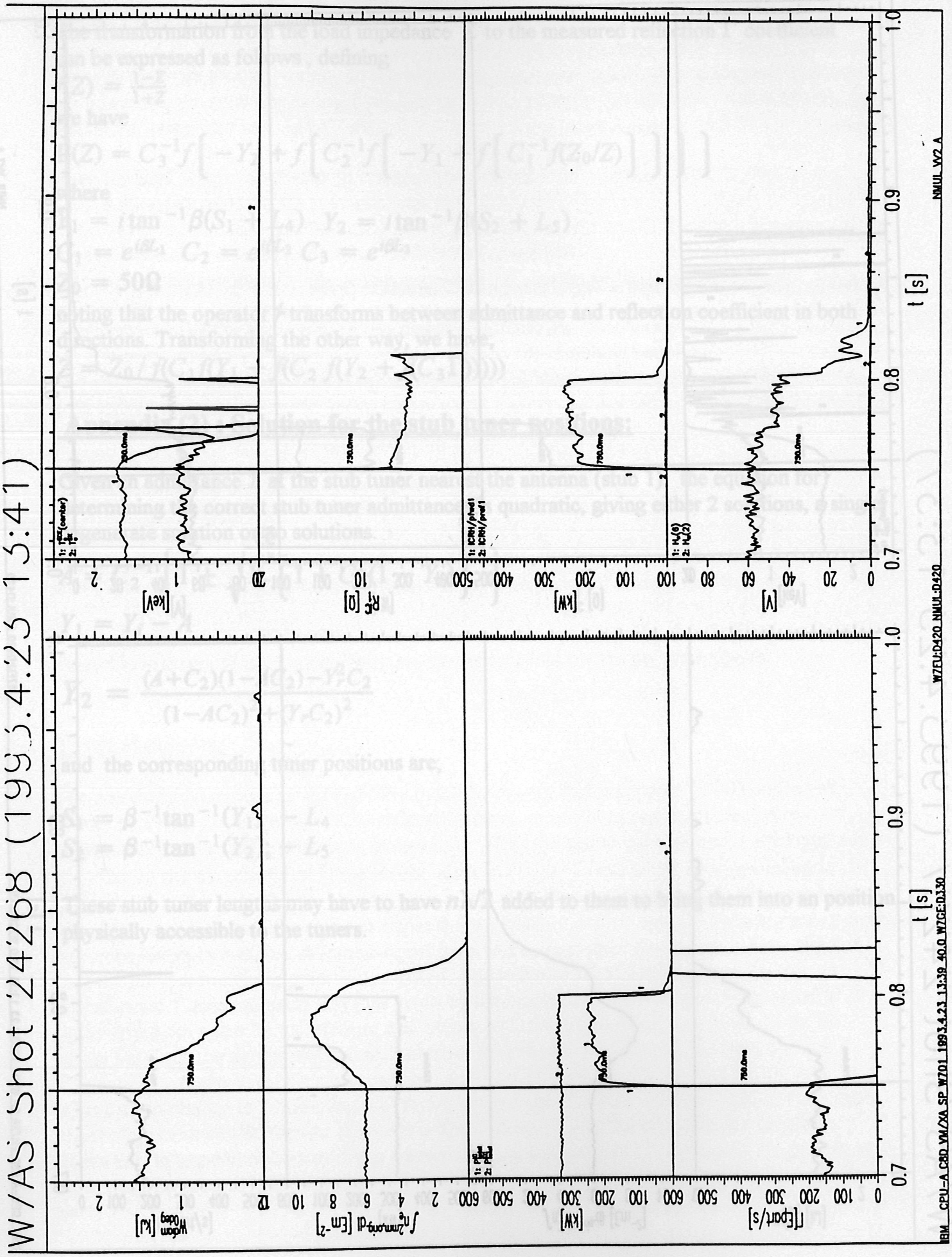
$$S_2 = \beta^{-1} \tan^{-1}(Y_2); - L_5$$

These stub tuner lengths may have to have $n\lambda/2$ added to them to bring them into an position physically accessible to the tuners.

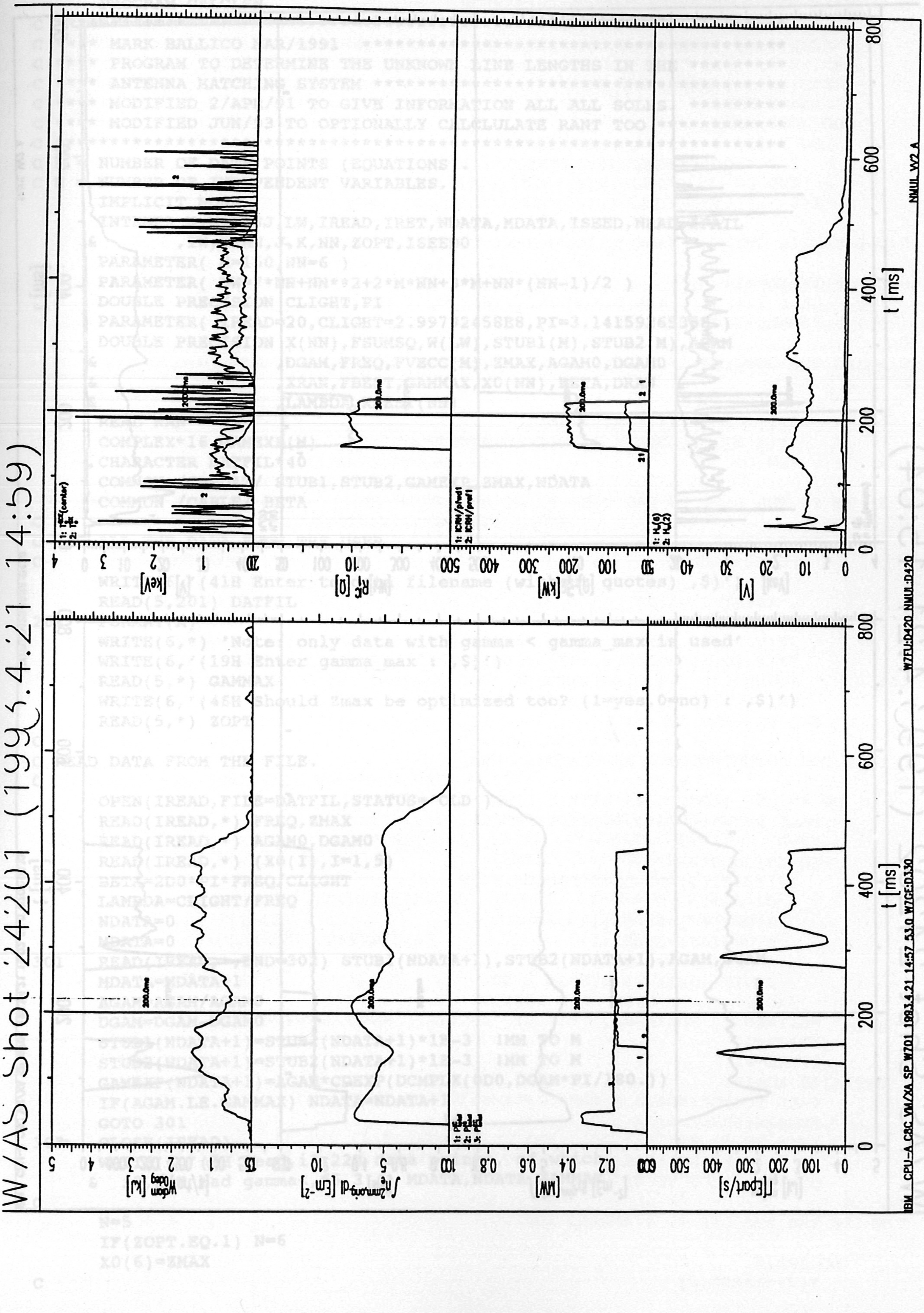
W7AS SHOT 24267 21/4/93
Initial poor matching Pref/Pfwd=30%



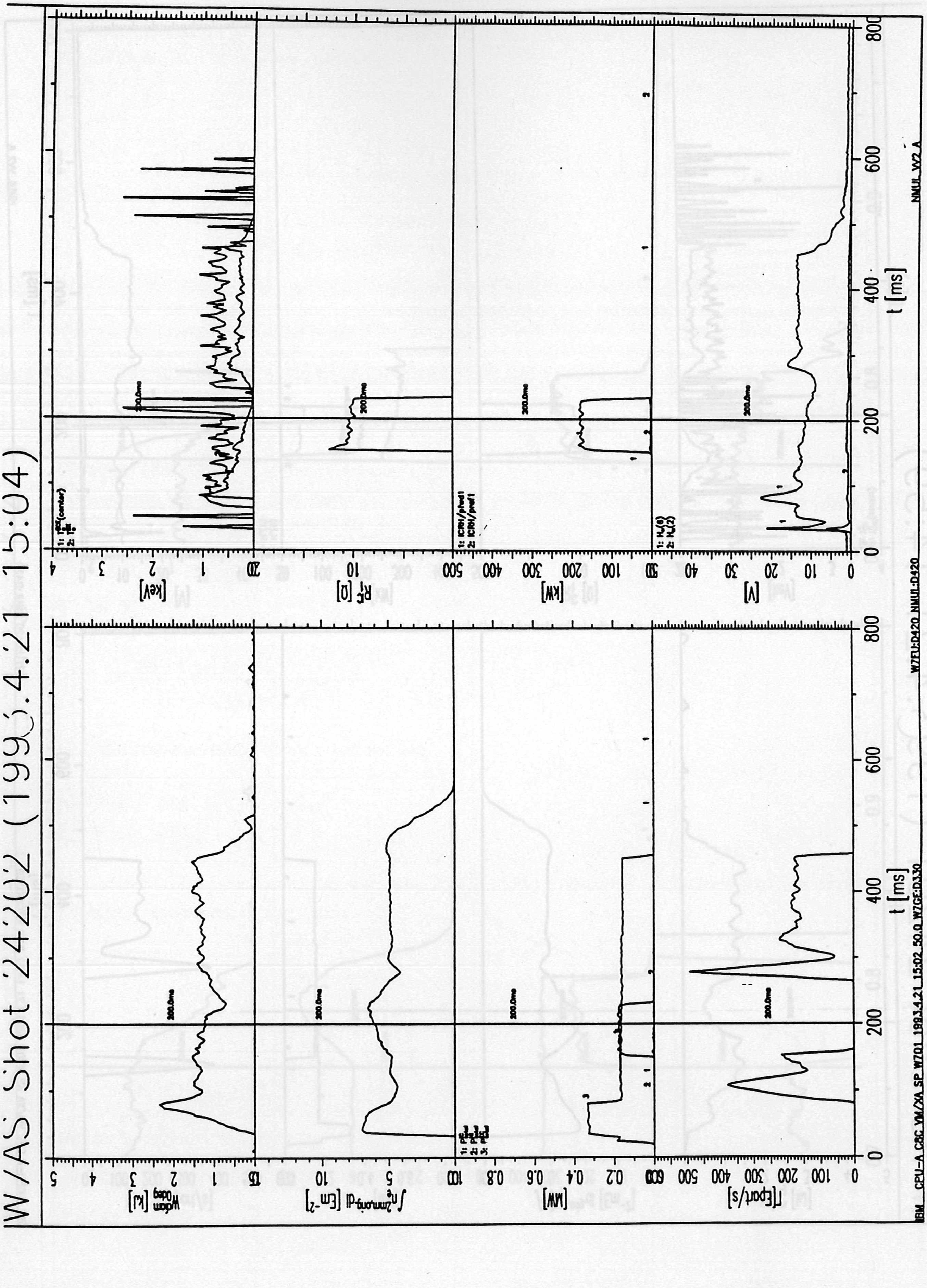
W7AS SHOT 24268 21/4/93
Next shot : Pref/Pfwd < 1%



W7AS SHOT 24201 21/4/93
Initial poor matching Pref/Pfwd=30%



W7AS SHOT 24202 21/4/93
Next shot : Pref/Pfwd < 1%



Program source for CALCLEN


```
      PROGRAM CALCLEN
C ************************************************************************
C **** MARK BALLICO MAR/1991 **********************************************
C **** PROGRAM TO DETERMINE THE UNKNOWN LINE LENGTHS IN THE *********
C **** ANTENNA MATCHING SYSTEM ********************************************
C **** MODIFIED 2/APR/91 TO GIVE INFORMATION ALL ALL SOLNS. *********
C **** MODIFIED JUN/93 TO OPTIONALLY CALCLULATE RANT TOO ************
C ************************************************************************
C M = NUMBER OF DATA POINTS (EQUATIONS).
C N = NUMBER OF INDEPENDENT VARIABLES.
      IMPLICIT NONE
      INTEGER N,M,I,JJ,LW,IREAD,IRET,NDATA,MDATA,ISEED,NRAD,IFAIL
     &       ,IW,NRAN,J,K,NN,ZOPT,ISEED0
      PARAMETER( M=150,NN=6 )
      PARAMETER( LW=7*NN+NN**2+2*M*NN+3*M+NN*(NN-1)/2 )
      DOUBLE PRECISION CLIGHT,PI
      PARAMETER( IREAD=20,CLIGHT=2.99792458E8,PI=3.1415926535 8 )
      DOUBLE PRECISION X(NN),FSUMSQ,W(LW),STUB1(M),STUB2(M),AGAM
     &       ,DGAM,FREQ,FVECC(M),ZMAX,AGAM0,DGAM0
     &       ,XRAN,FBEST,GAMMAX,X0(NN),BETA,DRAN
     &       ,LAMBDA,XBEST(NN)
      REAL RAN
      COMPLEX*16 GAMEXP(M)
      CHARACTER DATFIL*40
      COMMON /GAMDAT/ STUB1,STUB2,GAMEXP,ZMAX,NDATA
      COMMON /CABLE/ BETA
C
C GET ALL THE DATA FROM THE USER.
C
      WRITE(6,'(41H Enter to data filename (without quotes) ,$)')
      READ(5,201) DATFIL
201   FORMAT(A)
      WRITE(6,*) 'Note: only data with gamma< gamma_max is used'
      WRITE(6,'(19H Enter gamma_max : ,$)')
      READ(5,*) GAMMAX
      WRITE(6,'(46H Should Zmax be optimized too? (1=yes,0=no) : ,$)')
      READ(5,*) ZOPT
C
C READ DATA FROM THE FILE.
C
      OPEN(IREAD,FILE=DATFIL,STATUS='OLD')
      READ(IREAD,*) FREQ,ZMAX
      READ(IREAD,*) AGAM0,DGAM0
      READ(IREAD,*) (X0(I),I=1,5)
      BETA=2D0*PI*FREQ/CLIGHT
      LAMBDA=CLIGHT/FREQ
      NDATA=0
      MDATA=0
301   READ(IREAD,*,END=302) STUB1(NDATA+1),STUB2(NDATA+1),AGAM,DGAM
      MDATA=MDATA+1
      AGAM=AGAM/AGAM0
      DGAM=DGAM-DGAM0
      STUB1(NDATA+1)=STUB1(NDATA+1)*1E-3   !MM TO M
      STUB2(NDATA+1)=STUB2(NDATA+1)*1E-3   !MM TO M
      GAMEXP(NDATA+1)=AGAM*CDEXP(DCMPLX(0D0,DGAM*PI/180.))
      IF(AGAM.LE.GAMMAX) NDATA=NDATA+1
      GOTO 301
302   CLOSE(IREAD)
      WRITE(6,'(5H Read,i3,22H data points, of which,
     &  i3,11H had gamma<,f5.3)') MDATA,NDATA,GAMMAX
C
      N=5
      IF(ZOPT.EQ.1) N=6
      X0(6)=ZMAX
C
```

IPP print formatter for mjb on uts DATE: Tue Sep 14 14:17:22 1993

```
C INITIAL POINT TO START THE OPTIMIZATION.
C
      WRITE(6,*) 'Random perturbation of initial guess:'
      WRITE(6,'(42H Enter iseed, # perts. , max pert. in m : ,$)')
      READ(5,*) ISEED0,NRAN,DRAN
      ISEED=ISEED0
      DO JJ=1,NRAN
        DO I=1,5
          X(I)=X0(I)+DRAN*(RAN(ISEED)*2-1)
          END DO
        X(6)=X0(6)
C
C USE NAGLIB OPTIMIZATION TO FINE TUNE THESE VALUES
C
        IFAIL=+1
        CALL E04FDF(NDATA,N,X,FSUMSQ,IW,1,W,LW,IFAIL)
        FSUMSQ=FSUMSQ/NDATA
C
C BRING THE LENGTHS INTO RANGE 0 TO LAMBDA/2.
C
        DO I=1,5
          X(I)=MOD(X(I),LAMBDA/2D0)
          IF(X(I).LT.0D0) X(I)=X(I)+LAMBDA/2D0
          END DO
C
C IF IT IS THE BEST SO FAR THEN PRINT AND STORE IT.
C
        IF(JJ.EQ.1.OR.FSUMSQ.LT.FBEST) THEN
          FBEST=FSUMSQ
          DO J=1,N
            XBEST(J)=X(J)
            END DO
          IF(N.EQ.5) WRITE(6,102) FSUMSQ,(1D3*X(J),J=1,5)
          IF(N.EQ.6) WRITE(6,102) FSUMSQ,(1D3*X(J),J=1,5),X(6)
102       FORMAT(1H ,F9.6,6F10.2)
          ENDIF !IF THE BEST SO FAR.
        END DO  !DO JJ
C
C LOOK FOR OUTLYING DATA AND DELETE THEM.
C
      DO K=1,5  !LOOP OVER DATA 5 TIMES FOR OUTLIERS.
        CALL LSFUN1(NDATA,N,XBEST,FVECC)
        JJ=0
        DO I=1,NDATA
          IF(FVECC(I)**2.LE.3*FBEST) THEN
            JJ=JJ+1
            STUB1(JJ)=STUB1(I)
            STUB2(JJ)=STUB2(I)
            GAMEXP(JJ)=GAMEXP(I)
            ENDIF !DATA IS WITHIN 3 SD
          END DO !LOOP OVER DATA POINTS.
        WRITE(6,*) NDATA-JJ,' PTS OF ',NDATA,' WERE OUTSIDE 3 SD'
        NDATA=JJ
        IFAIL=+1
C OPTIMIZE AGAIN.
        CALL E04FDF(NDATA,N,XBEST,FBEST,IW,1,W,LW,IFAIL)
        FBEST=FBEST/NDATA
        IF(N.EQ.5) WRITE(6,102) FBEST,(1D3*XBEST(J),J=1,5)
        IF(N.EQ.6) WRITE(6,102) FBEST,(1D3*XBEST(J),J=1,5),XBEST(6)
        END DO !LOOP TO GET RID OF OUTLYERS.
C
C WRITE THE RESULTS TO STANDARD OUTPUT.
C
      DO J=1,N
        X(J)=XBEST(J)
```

```
      END DO
901   FORMAT(1H ,A20,1H=,F9.2,7H MM OR ,F9.2,3H MM)
902   FORMAT(1H ,6HISEED=,I8,6H NRAN=,I4,6H DRAN=,F6.3)
903   FORMAT(1H ,A20,1H=,F10.4,5H OHMS)
      WRITE(6,*) 'DATA FILE = <',DATFIL,'>'
      WRITE(6,*) 'MAXIMUM GAMMA USED =      ',GAMMAX
      WRITE(6,902) ISEED0,NRAN,DRAN
      WRITE(6,*) 'RMS GAMMA ERROR     =     ',SQRT(FBEST)
      WRITE(6,901) 'STUB1 TO MAX',1000*X(1),1000*(X(1)+LAMBDA/2)
      WRITE(6,901) 'STUB SEP    ',1000*X(2),1000*(X(2)+LAMBDA/2)
      WRITE(6,901) 'STUB2 TO REF',1000*X(3),1000*(X(3)+LAMBDA/2)
      WRITE(6,901) 'STUB1 OFFSET',1000*X(4),1000*(X(4)+LAMBDA/2)
      WRITE(6,901) 'STUB2 OFFSET',1000*X(5),1000*(X(5)+LAMBDA/2)
      IF(ZOPT.EQ.1) WRITE(6,903) 'R AT V MAX  ',X(6)
C
      END

      SUBROUTINE LSFUN1(NDATA,N,XC,FVECC)
      IMPLICIT NONE
      INTEGER NMAX,NDUM,N,I,NDATA
      PARAMETER( NMAX=150 )
      DOUBLE PRECISION XC(N),FVECC(NDATA),STUB1(NMAX),STUB2(NMAX)
     &               ,ZMAX,BETA,TMP,A1,A2
      COMPLEX*16 GAMEXP(NMAX),C1,C2,C3,CZ,CZ0
      COMMON /CABLE/ BETA
      COMMON /GAMDAT/ STUB1,STUB2,GAMEXP,ZMAX,NDUM
C TRANSFORMATION FACTORS FOR THE 3 LINE SEGMENTS.
      TMP=-2*BETA*XC(1)
      C1=DCMPLX(COS(TMP),SIN(TMP))
      TMP=-2*BETA*XC(2)
      C2=DCMPLX(COS(TMP),SIN(TMP))
      TMP=-2*BETA*XC(3)
      C3=DCMPLX(COS(TMP),SIN(TMP))
C NORMALIZED ADMITTANCE ON THE ANTENNA SIDE OF STUB 1.
      IF(N.EQ.5) CZ0=DCMPLX((ZMAX -50)/(ZMAX +50)) !GAMMA AT V MAX.
      IF(N.EQ.6) CZ0=DCMPLX((XC(6)-50)/(XC(6)+50)) !GAMMA AT V MAX.
      CZ0=CZ0* C1                                 !GAMMA AFTER STUB 1
      CZ0=(1-CZ0)/(1+CZ0)                         !NORM. ADMITTANCE AFTER STB1
C CALCULATE THE RESIDUALS FOR EACH DATA POINT:
      DO I=1,NDUM
        A1=1/DTAN(BETA*(XC(4)+STUB1(I)))   !NORM. J*ADMIT. STUB 1
        A2=1/DTAN(BETA*(XC(5)+STUB2(I)))   !NORM. J*ADMIT. STUB 2
        CZ=CZ0 - DCMPLX(0D0,A1)            !ADMIT. BEFORE STUB 1
        CZ=(1-CZ)/(1+CZ) * C2              !GAMMA AFTER STUB 2
        CZ=(1-CZ)/(1+CZ) - DCMPLX(0D0,A2)  !ADMIT. BEFORE STUB 2
        CZ=(1-CZ)/(1+CZ) * C3              !GAMMA AT DIRECT. COUPLER
        FVECC(I)=CDABS(CZ-GAMEXP(I))
      END DO
      RETURN
      END
```

IPP print formatter for mjb on uts DATE: Tue Sep 14 14:17:26 1993

```
38e6 1867.0 25
1.00 0.00
5.0 5.0 5.0 5.0 5.0
893 2190 0.068 +3.0
903 2190 0.208 +42.0
913 2190 0.359 +43.4
923 2190 0.489 +38.0
933 2190 0.593 +32.0
943 2190 0.676 +27.3
953 2190 0.740 +22.5
963 2190 0.780 +19.3
883 2190 0.160 -76.0
863 2190 0.455 -72.6
853 2190 0.570 -67.4
843 2190 0.653 -62.1
833 2190 0.724 -57.4
823 2190 0.777 -53.3
893 2130 0.638 -78.9
893 2140 0.583 -82.3
893 2150 0.518 -85.5
893 2160 0.425 -90.0
893 2170 0.299 -96.1
893 2180 0.148 -93.9
893 2190 0.068 +6.0
893 2200 0.261 +30.4
893 2210 0.453 +25.7
893 2220 0.617 +16.4
893 2230 0.739 +8.1
```

IPP print formatter for mjb on uts DATE: Tue Sep 14 14:17:30 1993

```
76.00e6 1000.0 30
0.656 -23.1
3.725 2.791 2.315 0.610 1.582

1966 2237 0.006 -68.3
1940 2237 0.255 +45.0
1920 2237 0.411 +60.0
1900 2237 0.514 +22.1
1880 2237 0.523 +80.1
2060 2237 0.531 +154.4
2040 2237 0.469 +160.4
2020 2237 0.392 +168.0
2000 2237 0.288 -179.0
1980 2237 0.132 -164.0

1966 2300 0.570 +114.3
1966 2280 0.440 +131.5
1966 2260 0.245 +131.6
1966 2240 0.030 +179.7
1966 2220 0.146 +2.1
1966 2200 0.264 +14.0
1966 2180 0.345 +21.9
1966 2160 0.399 +27.5
1966 2140 0.437 +31.6
1966 2120 0.464 +34.8
1966 2100 0.485 +37.3

1930 2300 0.418 +112.2
1930 2280 0.247 +99.7
1930 2260 0.243 +66.1
1930 2240 0.321 +54.0
1930 2220 0.385 +52.2
1930 2200 0.430 +52.7
1930 2180 0.461 +53.6
1930 2160 0.483 +54.6
1930 2320 0.572 +103.7

2000 2320 0.653 +100.3
2000 2300 0.620 +110.9
2000 2280 0.558 +125.4
2000 2260 0.453 +145.6
2000 2240 0.309 +174.3
2000 2220 0.177 -136.0
2000 2200 0.173 -65.1
2000 2180 0.230 -27.1
2000 2160 0.325 -2.9
2000 2140 0.380 -3.2
2000 2120 0.421 +10.9
2000 2100 0.452 +16.6

2080 2270 0.620 +122.9
2060 2270 0.607 +124.8
2040 2270 0.589 +127.2
2020 2270 0.560 +130.0
2000 2270 0.514 +134.8
1980 2270 0.534 +139.2
1960 2270 0.303 +140.2
1940 2270 0.183 +109.2
1920 2270 0.306 +72.6
1900 2270 0.462 +75.0
1880 2270 0.553 +81.6
1860 2270 0.601 +87.1

2080 2200 0.431 -165.5
2060 2200 0.374 -154.1
```

IPP print formatter for mjb on uts DATE: Tue Sep 14 14:17:30 1993

```
2040 2200 0.305 -138.0
2020 2200 0.226 -112.0
2000 2200 0.173 -66.7
1980 2200 0.204 -11.4
1960 2200 0.294 +22.7
1940 2200 0.387 +44.1
1920 2200 0.464 +59.6
1900 2200 0.520 +70.7
1880 2200 0.560 +79.3
1860 2200 0.598 +86.4
```

Sample configuration file for experiments in1993

```
$ ICRH
Level-State     test
Level-Type      DIAG
System          'W7AS ICRH'
Data-Limit      200192
User            'Ballico,Cattanei'
Version 1

******* ICRH SYSTEM LENGTHS ********

stub1_vac       320      0
stub2_vac       2162     0
stub1_offset    610.54   0.0
stub2_offset    1582.14  0.0
stub1_stub2     2791.68  0.0
stub2_dc        2315.62  0.0
freq            38.00    0.0

******* SHOT COMMENTS  ********************************

comment: direction couplers are -69.8dB at 38MHz
comment: voltage probes are -80dB
comment: current probes are -80dB
comment: cables for voltage and current probes assumed 1.2dB
comment: cable from dir.coupler to icrh room is 1.2dB
comment:+1.7dB for power signals: maybe calibration abs value wrong?
comment: power splitter in pfwd and pref lines is 3dB
comment: vmax=sqrt(pmax+17dB), pmax=vmax**2/50 => vmax=sqrt(50*pmax)
comment: imax=sqrt(pmax-17dB), pmax=imax**2*50 => imax=sqrt(pmax/50)

****** PATCH TABLE ********************************************

pfwd1     ICRH/ADC-1(1)   det:box1U     dB75.7      W
pref1     ICRH/ADC-1(3)   det:box1I     dB75.7      W
vmax1     ICRH/ADC-1(5)   det:A1ohne    sdB109.9    Vrms
vmax2     ICRH/ADC-1(6)   det:A2ohne    sdB109.9    Vrms
sin1      ICRH/ADC-1(4)   cal:box2P     x1.0        deg
cos1      ICRH/ADC-1(2)   cal:box1P     x1.0        deg
I_top_l   ICRH/ADC-1(7)   det:A3ohne    sdB75.9     Arms
I_top_r   ICRH/ADC-1(8)   det:A4ohne    sdB75.9     Arms
I_bot_l   ICRH/ADC-1(9)   det:A5ohne    sdB75.9     Arms
I_bot_r   ICRH/ADC-1(10)  det:A6ohne    sdB75.9     Arms
LH_vmag   ICRH/ADC-2(1)   raw           x-4e3       V
LH_imag   ICRH/ADC-2(2)   raw           x1.0        A
LH_ref    ICRH/ADC-2(3)   det:three     dB60.1      W
LH_fwd    ICRH/ADC-2(4)   det:one       dB60.1      W
prb1      ICRH/ADC-2(5)   det:box3U     dB-35.3     W
cosprb1   ICRH/ADC-2(6)   det:box3P     x1.0        deg
refprb1   ICRH/ADC-2(7)   det:box3I     x1.0        a.u.
sinprb1   ICRH/ADC-2(8)   det:box4P     x1.0        deg

***** CALIBRATIONS FOR DETECTORS ***************************

A1mit*   26.00 1.00 23.
A1mit-1  4.347 3.540 2.895 2.350 1.908 1.539 1.243 0.995
A1mit-2  0.792 0.638 0.507 0.404 0.321 0.259 0.203 0.161
A1mit-3  0.128 0.102 0.081 0.064 0.052 0.040 0.032
A2mit*   26.00 1.00 23.
A2mit-1  4.340 3.536 2.872 2.322 1.870 1.502 1.203 0.959
A2mit-2  0.760 0.604 0.479 0.382 0.302 0.239 0.190 0.150
A2mit-3  0.121 0.095 0.074 0.059 0.048 0.036 0.028
A3mit*   26.00 1.00 23.
```

```
A3mit-1 4.180 3.402 2.750 2.227 1.795 1.440 1.154 0.921
A3mit-2 0.729 0.581 0.460 0.367 0.293 0.230 0.182 0.145
A3mit-3 0.117 0.091 0.072 0.056 0.045 0.037 0.029
A4mit* 26.00 1.00 23.
A4mit-1 4.574 3.729 3.034 2.460 1.983 1.595 1.280 1.022
A4mit-2 0.810 0.644 0.512 0.407 0.323 0.256 0.204 0.162
A4mit-3 0.129 0.102 0.080 0.063 0.052 0.041 0.031
A5mit* 26.00 1.00 23.
A5mit-1 4.336 3.537 2.880 2.335 1.888 1.522 1.226 0.981
A5mit-2 0.780 0.621 0.495 0.395 0.314 0.248 0.197 0.157
A5mit-3 0.126 0.100 0.079 0.063 0.051 0.041 0.033
A6mit* 26.00 1.00 23.
A6mit-1 4.478 3.656 2.978 2.416 1.954 1.576 1.271 1.018
A6mit-2 0.810 0.646 0.515 0.413 0.328 0.260 0.205 0.163
A6mit-3 0.130 0.106 0.082 0.065 0.052 0.041 0.033
A7mit*  26.00 1.00 23.
A7mit-1 4.100 3.328 2.694 2.172 1.747 1.402 1.125 0.895
A7mit-2 0.710 0.562 0.448 0.356 0.282 0.223 0.178 0.140
A7mit-3 0.112 0.090 0.071 0.056 0.045 0.035 0.028
A8mit*  26.00 1.00 23.
A8mit-1 4.300 3.570 2.910 2.362 1.912 1.544 1.244 0.997
A8mit-2 0.793 0.633 0.505 0.403 0.321 0.254 0.201 0.160
A8mit-3 0.128 0.101 0.080 0.063 0.050 0.040 0.032
A1ohne*  8.00 1.00 26.
A1ohne-1 7.827 6.556 5.462 4.532 3.739 3.046 2.483 2.016
A1ohne-2 1.633 1.315 1.049 0.840 0.671 0.537 0.428 0.339
A1ohne-3 0.270 0.215 0.171 0.136 0.106 0.084 0.066 0.053
A1ohne-4 0.042 0.034
A2ohne* 8.00 1.00 26.
A2ohne-1 6.554 5.451 4.502 3.705 3.021 2.437 1.965 1.581
A2ohne-2 1.269 1.013 0.802 0.638 0.508 0.405 0.320 0.252
A2ohne-3 0.200 0.158 0.127 0.100 0.076 0.061 0.048 0.037
A2ohne-4 0.029 0.023
A3ohne*  8.00 1.00 26.
A3ohne-1 6.727 5.593 4.626 3.812 3.115 2.515 2.302 1.637
A3ohne-2 1.316 1.051 0.833 0.661 0.528 0.420 0.333 0.263
A3ohne-3 0.208 0.165 0.133 0.103 0.082 0.065 0.051 0.042
A3ohne-4 0.032 0.025
A4ohne*  8.00 1.00 26.
A4ohne-1 6.720 5.593 4.628 3.817 3.118 2.517 2.307 1.639
A4ohne-2 1.320 1.056 0.836 0.667 0.530 0.423 0.335 0.269
A4ohne-3 0.211 0.168 0.132 0.105 0.089 0.065 0.052 0.043
A4ohne-4 0.033 0.025
A5ohne* 8.00 1.00 26.
A5ohne-1 6.760 5.633 4.672 3.862 3.169 2.566 2.083 1.685
A5ohne-2 1.359 1.090 0.868 0.694 0.553 0.442 0.353 0.278
A5ohne-3 0.221 0.177 0.140 0.111 0.088 0.071 0.055 0.044
A5ohne-4 0.036 0.027
A6ohne* 8.00 1.00 26.
A6ohne-1 6.620 5.519 4.564 3.766 3.083 2.497 2.025 1.635
A6ohne-2 1.317 1.056 0.839 0.670 0.536 0.428 0.341 0.269
A6ohne-3 0.215 0.169 0.135 0.108 0.085 0.067 0.054 0.043
A6ohne-4 0.033 0.028
A7ohne* 8.00 1.00 26.
A7ohne-1 6.668 5.540 4.579 3.773 3.078 2.485 2.007 1.615
A7ohne-2 1.298 1.037 0.823 0.655 0.520 0.414 0.328 0.259
A7ohne-3 0.206 0.163 0.130 0.103 0.081 0.065 0.052 0.041
A7ohne-4 0.033 0.027
A8ohne* 8.00 1.00 26.
A8ohne-1 6.764 5.632 4.666 3.851 3.151 2.550 2.065 1.667
A8ohne-2 1.343 1.076 0.855 0.682 0.543 0.433 0.343 0.271
A8ohne-3 0.216 0.171 0.136 0.108 0.087 0.069 0.055 0.042
A8ohne-4 0.034 0.027
```

```
Burst-3          0 0 STOP

$ Timer-2
Driver          TIMER
Unit-State      online
ID-Code         2
Address         0  3
Mode            No_Signal
Data-Type       BYTE
Data-Count      0
Start            Extern
Burst-1          1 1  Delay
Burst-2          500 8000  Continue
Burst-3          0 0  STOP

$ ADC-1
Driver           LC8212
Unit-State       offline
ID-Code         39
Address          1       6
Mode            Dynamic
Data-Type       INT2
Data-Count       14352
Timebase         Timer-1  0
Channels         16
Pre-Trigger      0
Post-Trigger     897
Conversion       2.442e-3  4096 2048

$ ADC-2
Driver           LC8212
Unit-State       online
ID-Code          40
Address          1       14
Mode             Dynamic
Data-Type        INT2
Data-Count       16384
Timebase         Timer-2  0
Channels         8
Pre-Trigger      0
Post-Trigger     2048
Conversion       2.442e-03  4096  2048

$ STUB-TUNER
Driver          FILEDRV
Unit-State      online
ID-Code         0
Address         0 0
Mode            On_Init
Data-Type       INT4
Data-Count      8
File-Name       USR:STUB.STATUS

$ End.
```

```
box3U*  13.00 2.00 14.
box3U-1 8.673 6.104 4.223 2.852 1.899 1.238 0.7992 0.5130
box3U-2 0.3242 0.2065 0.1298 0.0819 0.0517 0.0322
box3I*  13.00 2.00 14.
box3I-1 8.978 6.392 4.484 3.079 2.087 1.385 0.9109 0.5954
box3I-2 0.3842 0.2503 0.1624 0.1071 0.0720 0.0494
box3P* 20.
box3Py-1 2.7 10. 20. 30. 40. 50. 60. 70. 80. 90. 100. 110.
box3Py-2 120. 130. 140. 150. 160. 170. 180. 184.
box3Px-1 1.897 1.669 1.388 1.158 0.823 0.531 0.290 0.034
box3Px-2 -0.233 -0.558 -0.861 -1.103 -1.403 -1.802 -2.070
box3Px-3 -2.252 -2.397 -2.551 -2.637 -2.645
box4P* 19.
box4Py-1 0. 10. 20. 30. 40. 50. 60. 70. 80. 90. 100. 110.
box4Py-2 120. 130. 140. 150. 160. 170. 180.
box4Px-1 1.426 1.219 0.979 0.840 0.548 0.263 0.027 -0.139
box4Px-2 -0.298 -0.503 -0.713 -0.883 -1.146 -1.445 -1.624
box4Px-3 -1.749 -1.871 -1.985 -2.037

box1U*  13.00 2.00 14.
box1U-1 6.329 4.299 2.904 1.907 1.242 0.7910 0.5015
box1U-2 0.3163 0.1964 0.1222 0.0747 0.0453 0.0270 0.0154
box1I* 13.00 2.00 14.
box1I-1 6.046 4.106 2.732 1.762 1.129 0.7093 0.4446
box1I-2 0.2782 0.1715 0.1061 0.0643 0.0386 0.0226 0.0124
box1P* 19.
box1Py-1 1. 11. 21. 31. 41. 51. 61. 71. 81. 91. 101. 111.
box1Py-2 121. 131. 141. 151. 161. 171. 181. 184.
box1Px-1 2.608 2.517 2.258 1.950 1.611 1.286 0.941 0.615
box1Px-2 0.302 0.004 -0.292 -0.578 -0.871 -1.156 -1.461
box1Px-3 -1.733 -2.066 -2.353 -2.530 -2.542

box2P* 19.
box2Py-1 -2.6 6. 16. 26. 36. 46. 56. 66. 76. 86. 96. 106.
box2Py-2 116. 126. 136. 146. 156. 166. 176.
box2Px-1 2.690 2.586 2.299 1.991 1.655 1.302 0.974
box2Px-2 0.641 0.304 -0.004 -0.311 -0.623 -0.938
box2Px-3 -1.224 -1.527 -1.851 -2.161 -2.435 -2.615

one*    16. 1. 36.
one-1   1.912 1.753 1.556 1.379 1.223 1.084 0.961 0.850 0.746
one-2   0.659 0.581 0.513 0.452 0.398 0.349 0.306 0.269 0.235
one-3   0.203 0.177 0.154 0.133 0.115 0.100 0.085 0.073 0.062
one-4   0.053 0.044 0.037 0.031 0.026 0.021 0.017 0.014 0.012

three* -19. -1. 36.
three-1 -.010 -.012 -.015 -.018 -.022 -.026 -.032 -.038 -.045
three-2 -.054 -.063 -.075 -.088 -.102 -.118 -.137 -.158 -.183
three-3 -.211 -.241 -.276 -.316 -.361 -.410 -.466 -.530 -.602
three-4 -.684 -.765 -.866 -.978 -1.106 -1.251 -1.422 -1.641 -1.986

***** UDAS SET-UP FOR CAMAC DIGITIZERS ***********************

$ Timer-1
Driver          TIMER
Unit-State      online
ID-Code         1
Address         0  3
Mode            No_Signal
Data-Type       BYTE
Data-Count      0
Start            Extern
Burst-1          350000  1 Delay
Burst-2          500     4000 Continue
```

Command script for compilation of MATCH

```
$ set verify
$ for match,open_rl100,read_rl100,write_rl100,read_timeout
$ link match,open_rl100,read_rl100,write_rl100,read_timeout-
,ud$lib:w7fu/lib,ud$lib:w7ge/lib
$ delete match.obj;*,open_rl100.obj;*,write_rl100.obj;*-
 ,read_rl100.obj;*,read_timeout.obj;*
$ set noverify
```

Program source for MATCH

```
      program match
c written by Mark Ballico 18/8/92
c variable declarations.
      implicit none
      integer nmax
      parameter(nmax=1000)
      real new_stub1(2),new_stub2(2),best,rmin,dmin,a,b,pi,c_light
     &  ,y1(nmax),y2(nmax),y3(nmax),y4(nmax),xr,beta,disc,c5,xi,tmp
     &  ,stub1_stub2(2),stub2_dc(2),stub1_offset(2),stub2_offset(2)
     &  ,freq(2),stub1,stub2,t(nmax),ti,tf,d,mbcot,mbacot,czsd,pthresh
      parameter(pi=3.1415926535,c_light=3.00e8,pthresh=50e-3)
      complex cz,czav,c1,c2,c3,c4
      character class*2,result*80,cvtsht*22,fname*22,select*80
     &  ,getdat*80,openf*80,closef*80,mode*1,text6*6,text*80
     &  ,lastsh*80,system*1,getpar*80
      integer isys(2),ishot,lunit,stubs(4),i,ierr,setstubx,w1,w2,w3
     &    ,line,status,err,pos(4),itmp,w4,isystem,jbest,j,icdata,n,nok
     &  ,stub1_vac(2),stub2_vac(2)
     &  ,w5,idum,w6
     &  ,pbid
     &  ,smg$put_chars
     &  ,smg$put_with_scroll
     &  ,smg$set_cursor_abs
     &  ,smg$create_pasteboard
     &  ,smg$create_virtual_display
     &  ,smg$paste_virtual_display
     &  ,smg$erase_display
     &  ,smg$unpaste_virtual_display
     &  ,smg$change_pbd_characteristics
      include'($smgdef)'                    !windows definitions.
      logical active(4),mb_select,mb_getpar,mb_getdat,mb_openf
      common pbid,w1,w4                     !window common block.
      call udsetm('AUTOLOG','OFF')          !disable udas-error printing.
c
c set up the array of time values.
      n=nmax
      ti=0.
      tf=1.
      do i=1,n
        t(i)=(ti*(n-i)+tf*(i-1))/(n-1)
        enddo
c
c defaults for user input.
      isys(1)=1
      isys(2)=0
      class='DN'
      goto 3    !*******bypass********
c
c user input.
1     write(*,*) 'Is system 1 in use ? (0=no,1=yes)'
      read(*,*,err=1) isys(1)
2     write(*,*) 'Is system 2 in use ? (0=no,1=yes)'
      read(*,*,err=2) isys(2)
      write(*,*) 'Using test(''DT'') or real(''DN'') shots'
      read(*,*,err=3) class
c
c decide which of the 4 stubs are in use.
3     active(1)=isys(1).eq.1
      active(2)=isys(1).eq.1
      active(3)=isys(2).eq.1
      active(4)=isys(2).eq.1
c
```

```
c open stub tuner I/O channels.
        do i=1,4
          if(active(i)) call open_rl100(i)
          end do
c
c define windows.
        status=smg$create_pasteboard(pbid)      !GET pasteboard id.
        status=smg$change_pbd_characteristics(pbid,43,idum,26,idum
     &    ,smg$c_color_black,idum)
        status=smg$create_virtual_display(3,18,w1,smg$m_border)
        status=smg$create_virtual_display(9,27,w2,smg$m_border)
        status=smg$create_virtual_display(1,40,w3,smg$m_border)
        status=smg$create_virtual_display(1,18,w4,smg$m_border
     &      ,smg$m_bold.or.smg$m_blink)
        status=smg$create_virtual_display(10,40,w5,smg$m_border)
        status=smg$create_virtual_display(2,30,w6,smg$m_border
     &  ,smg$m_bold) ! .or.smg$m_blink)
c
c text for stub window and menu window.
        status=smg$put_with_scroll(w1,'       tuner1 tuner2')
        status=smg$put_with_scroll(w1,'sys1     off    off ')
        status=smg$put_with_scroll(w1,'sys2     off    off ')
        status=smg$put_with_scroll(w2,'0...set to new values')
        status=smg$put_with_scroll(w2,'1...set to old shot values')
        status=smg$put_with_scroll(w2,'2...set to last shot values')
        status=smg$put_with_scroll(w2,'3...calc. from old shot')
        status=smg$put_with_scroll(w2,'4...calc. from last shot')
        status=smg$put_with_scroll(w2,'5...set to vacuum matching')
        status=smg$put_with_scroll(w2,'6...change time window')
        status=smg$put_with_scroll(w2,'7...exit')
        status=smg$put_with_scroll(w6,'ICRH-ANTENNA MATCHING PROGRAM')
        status=smg$put_with_scroll(w6,'by Mark Ballico')
c
c main loop.
4       status=smg$paste_virtual_display(w1,pbid,5,6)
        status=smg$paste_virtual_display(w6,pbid,1,1)
        status=smg$paste_virtual_display(w2,pbid,10,2)
        status=smg$put_with_scroll(w2,'enter 0-5 ? ')
5       status=smg$set_cursor_abs(w2,9,12)
        call read_timeout(' ',mode,i,5)
c
c
c
c update usr:stub.status and screen display.
        if(i.eq.0) then
          do line=1,4
            if(active(line)) then
              call read_rl100(err,line,pos(line))
              write(text6,'(i4,2hmm)') pos(line)
              status=smg$put_chars(w1,text6,(line+3)/2,13-7*mod(line,2))
              endif
            enddo
          call update(active,pos,stubs)
          goto 5
          endif
c
c
c
c redraw main menu and stub display.
         if(mode.lt.'0'.or.mode.gt.'7') then
          status=smg$unpaste_virtual_display(w1,pbid)
          status=smg$unpaste_virtual_display(w2,pbid)
          goto 4
          endif
```

```
c
c
c
c change time window.
      if(mode.eq.'6') then
        status=smg$paste_virtual_display(w3,pbid,20,3)
16      status=smg$put_with_scroll(w3,
     &  'enter ti and tf:____________________')
        status=smg$set_cursor_abs(w3,1,17)
        call read_timeout(' ',text,itmp,10)
        if(itmp.eq.0) goto 17
        read(text(1:itmp),*,err=16) ti,tf
        do i=1,n
          t(i)=(ti*(n-i)+tf*(i-1))/(n-1)
        enddo
17      status=smg$unpaste_virtual_display(w3,pbid)
        goto 5
        endif
c
c
c
c program end.
      if(mode.eq.'7') then
        status=smg$unpaste_virtual_display(w1,pbid)
        status=smg$unpaste_virtual_display(w2,pbid)
        status=smg$unpaste_virtual_display(w6,pbid)
        stop
        endif
c
c
c
c enter new values by hand.
      if(mode.eq.'0') then
        status=smg$paste_virtual_display(w3,pbid,16,3)
        do i=1,2
          if(isys(i).ne.0) then
6           write(text,'(18H Enter new system ,i1,8H values:
     &           ,9H_________)') i
            status=smg$put_with_scroll(w3,text)
            status=smg$set_cursor_abs(w3,1,28)
            call read_timeout(' ',text,itmp,10)
            if(itmp.eq.0) goto 7
            read(text(1:itmp),*,err=6) stubs(2*i-1),stubs(2*i)
            endif
          enddo
        ierr=setstubx(active,stubs)
7       status=smg$unpaste_virtual_display(w3,pbid)
        goto 5
        endif
c
c
c
c get shot number from the user.
      if(mode.eq.'1'.or.mode.eq.'3') then
        status=smg$paste_virtual_display(w3,pbid,16,3)
8       status=smg$put_with_scroll(w3,'Enter shot number:_________')
        status=smg$set_cursor_abs(w3,1,20)
        call read_timeout(' ',text,itmp,10)
        status=smg$unpaste_virtual_display(w3,pbid)
        if(itmp.eq.0) goto 5
        read(text(1:itmp),*,err=8) ishot
        fname=cvtsht(ishot,class,'ICRH')
        endif
```

```
c
c
c
c get the last shot number
        if(mode.eq.'2'.or.mode.eq.'4'.or.mode.eq.'5') then
          result=lastsh(fname,ishot)
          if(result(1:1).ne.' ') then
            status=smg$paste_virtual_display(w3,pbid,16,3)
            status=smg$put_with_scroll(w3,'error finding last shot')
            call ttwait(2)
            status=smg$unpaste_virtual_display(w3,pbid)
            endif
          endif
c
c
c
c read vacuum matching from the shotfile and set tuners.
        if(mode.eq.'5') then
          lunit=3
          if(mb_openf (w5,fname,lunit)) goto 101
          if(mb_select(w5,'ICRH','Timer-1')) goto 102
          if(mb_getpar(w5,'DIAG','stub1_vac',stub1_vac)) goto 102
          if(mb_getpar(w5,'DIAG','stub2_vac',stub2_vac)) goto 102
          stubs(1)=stub1_vac(1)
          stubs(2)=stub2_vac(1)
          stubs(3)=stub1_vac(2)
          stubs(4)=stub2_vac(2)
          endif
c
c
c
c open shotfile and retrieve tuner values.
        if(mode.eq.'1'.or.mode.eq.'2'.or.mode.eq.'3'.or.mode.eq.'4') then
          status=smg$erase_display(w5)
          status=smg$paste_virtual_display(w5,pbid,16,3)
          write(text,'(16HCurrent shot is ,a22)') fname
          status=smg$put_with_scroll(w5,text)
          lunit=3
          if(mb_openf (w5,fname,lunit)) goto 101
          if(mb_select(w5,'ICRH','STUB-TUNER')) goto 102
          if(mb_getdat(w5,stubs,1,4)) goto 102
          endif
c
c
c
c set tuners to retrieved values.
        if(mode.eq.'1'.or.mode.eq.'2'.or.mode.eq.'5') then
          ierr=setstubx(active,stubs)
          endif
c
c
c
c calculate the load impedance and work out new stub positions.
        if(mode.eq.'3'.or.mode.eq.'4') then
c get icrh system data from parameter file.
          if(mb_select(w5,'ICRH','Timer-1')) goto 102
          if(mb_getpar(w5,'DIAG','stub1_offset',stub1_offset)) goto 102
          if(mb_getpar(w5,'DIAG','stub2_offset',stub2_offset)) goto 102
          if(mb_getpar(w5,'DIAG','stub1_stub2',stub1_stub2)) goto 102
          if(mb_getpar(w5,'DIAG','stub2_dc',stub2_dc)) goto 102
          if(mb_getpar(w5,'DIAG','freq',freq)) goto 102 !MHz
c loop over the systems.
          do isystem=1,2                  !loop over system 1 & 2.
            if(isys(isystem).eq.1) then !system is active.
```

```
c get RF signals.
        write(system,'(i1)') isystem
        ierr=icdata('pfwd'//system,n,t,y1,text)
   &      +icdata('pref'//system,n,t,y2,text)
   &      +icdata('sin' //system,n,t,y3,text)
   &      +icdata('cos' //system,n,t,y4,text)
        if(ierr.ne.0) then
          status=smg$put_with_scroll(w5,'error: can''t get data')
          goto 102
          endif
c calculate transmission line factors.
        stub1=stubs(2*isystem-1)+stub1_offset(isystem)
        stub2=stubs(2*isystem  )+stub2_offset(isystem)
        beta=2e3*pi*freq(isystem)/c_light   !rad/mm
        c1=exp(cmplx(0.,2*beta*stub2_dc(isystem)))
        c2=cmplx(0.,-mbcot(beta*stub2))
        c3=exp(cmplx(0.,2*beta*stub1_stub2(isystem)))
        c4=cmplx(0.,-mbcot(beta*stub1))
        c5=1.0/mbcot(beta*stub1_stub2(isystem))
c calculate average gamma at the directional coupler.
        nok=0
        czav=(0.,0.)
        czsd=0.
        do i=1,n         !data points.
          if(y1(i).gt.pthresh) then !data ok.
c choose the phase channel closest to 90 deg for best accuracy.
c and use the other channel to determine the sign.
            if(abs(y4(i)-90.).lt.abs(y3(i)-90.)) then
              if(y3(i).le.90.) tmp=+y4(i)
              if(y3(i).gt.90.) tmp=-y4(i)
            else
              if(y4(i).ge.90.) tmp=+y3(i)+90.0
              if(y4(i).lt.90.) tmp=-y3(i)+90.0
            endif
            if(y2(i).lt.0.) stop 'negative reflected power!!'
            cz=sqrt(y2(i)/y1(i))*cmplx(cosd(tmp),sind(tmp))
            czav=czav+cz
            czsd=czsd+abs(cz)**2
            nok=nok+1
            endif                    !data ok.
          enddo      !data points.
        if(nok.le.1) then        !error
          status=smg$put_with_scroll(w5,'error: too few good data points')
          goto 102
          endif                  !error.
        czav=czav/nok
        tmp=czsd-nok*abs(czav)**2
        if(tmp.lt.0.) status=smg$put_with_scroll(w5,'sd<zero?')
        tmp=max(tmp,0.)
        czsd=sqrt(tmp)/(nok-1)
        write(text,*) 'number of data points=',nok
        status=smg$put_with_scroll(w5,text)
        write(text,*) 'abs(gamma)=',abs(czav)
        status=smg$put_with_scroll(w5,text)
        if(czav.ne.(0.,0.)) write(text,*) 'arg(gamma)='
   &      ,atan2(aimag(czav),real(czav))*180.0/pi
        status=smg$put_with_scroll(w5,text)
        write(text,*) 'sd(gamma)=',czsd
        status=smg$put_with_scroll(w5,text)
c calculate the complex impedance on the antenna side of stub 1
        cz=czav                      !gamma at d.c.
        cz=cz*c1                     !gamma at/with stub 2
        cz=(1-cz)/(1+cz) - c2        !y at/without stub 2
        cz=(1-cz)/(1+cz) * c3        !gamma at/with stub 1
```

```
      cz=(1-cz)/(1+cz) - c4             !y at/without stub 1
      cz=(1-cz)/(1+cz)                  !gamma at/without stub 1
      rmin=50.0*(1-abs(cz))/(1+abs(cz))
      dmin=atan2(aimag(cz),real(cz))/beta
      write(text,*) 'rmin=',rmin,'ohms'
      status=smg$put_with_scroll(w5,text)
      write(text,*) 'dmin=',dmin,'mm'
      status=smg$put_with_scroll(w5,text)
c calculate the two possible new matching positions.
      cz=(1-cz)/(1+cz)                  !y
      xr=real(cz)                       !Re(Y)
      xi=aimag(cz)                      !Im(Y)
      disc=xr*(1+c5**2*(1-xr))
      if(disc.lt.0.) then               !error.
        status=smg$put_with_scroll(w5,'matching is not possible')
        goto 102
        endif                           !error.
      do j=1,2      !2 solutions.
        a= (1 + (2*j-3)*sqrt(disc)) / c5
        b=-((a+c5)*(1-a*c5)-xr**2*c5)/((1-a*c5)**2+(xr*c5)**2)
        new_stub1(j)=mbacot(-a+xi)/beta-stub1_offset(isystem)
        new_stub2(j)=mbacot(-b   )/beta-stub2_offset(isystem)
        if(new_stub1(j).lt.0.) new_stub1(j)=new_stub1(j)+pi/beta
        if(new_stub2(j).lt.0.) new_stub2(j)=new_stub2(j)+pi/beta
        write(text,'(6H soln.,i2,3H is,2f7.1)')
     &   j,new_stub1(j),new_stub2(j)
        status=smg$put_with_scroll(w5,text)
        enddo       !2 solutions.
c decide which, if any, matching point is more appropriate.
      jbest=0
      best=1e20
        do j=1,2    !loop over the 2 solutions.
        if(new_stub1(j).ge.0.and.new_stub1(j).le.2500..and.
     &     new_stub2(j).ge.0.and.new_stub2(j).le.2500.) then
          d=(new_stub1(j)-stubs(2*isystem-1))**2
     &     +(new_stub2(j)-stubs(2*isystem   ))**2
          if(d.lt.best) then            !better.
            best=d
            jbest=j
            endif                       !better.
          endif !if allowed.
        enddo   !2 solutions.
c move stubs.
       if(jbest.eq.0) then
         status=smg$put_with_scroll(w5,
     &   'neither matching point is accessible')
         goto 102
       else
         write(text,*) 'matching point',jbest,' is closer'
         status=smg$put_with_scroll(w5,text)
         stubs(2*isystem-1)=nint(new_stub1(jbest))
         stubs(2*isystem  )=nint(new_stub2(jbest))
         ierr=setstubx(active,stubs)
       endif
c end of calculating matching routine.
       endif !system is active.
       enddo !system 1 & 2.
     endif !calculate new matching from rf data.
c
c
c
c exits.
100   goto 103                    !normal exit.
101   call ttwait(6)
```

```
      goto 104
102   call ttwait(6)
103   result=closef()              !close shot file.
104   status=smg$unpaste_virtual_display(w5,pbid)
      goto 5                       !return to main menu.
      end


      function mb_openf(window,fname,lunit)
c redirect errors to a window & return true or false.
      implicit none
      character fname*(*),result*80,openf*80
      integer window,status,smg$put_with_scroll,lunit
      logical mb_openf
      mb_openf=.false.
      result=openf(fname,lunit)
      if(result(1:1).ne.' ') then
        status=smg$put_with_scroll(window,result)
        mb_openf=.true.
      endif
      return
      end


      function mb_getdat(window,datbuf,start,amount)
c redirect errors to a window & return true or false.
      implicit none
      character result*80,getdat*80
      integer window,status,smg$put_with_scroll,start,amount,datbuf
      logical mb_getdat
      mb_getdat=.false.
      result=getdat(datbuf,start,amount)
      if(result(1:1).ne.' ') then
        status=smg$put_with_scroll(window,result)
        mb_getdat=.true.
        endif
      return
      end


      function mb_select(window,diagn,modn)
c redirect errors to a window & return true or false.
      implicit none
      character diagn*(*),modn*(*),result*80,select*80
      integer window,status,smg$put_with_scroll
      logical mb_select
      mb_select=.false.
      result=select(diagn,modn)
      if(result(1:1).ne.' ') then
        status=smg$put_with_scroll(window,result)
        mb_select=.true.
        endif
      return
      end
```

```
      function mb_getpar(window,level,pname,pbuf)
c redirect errors to a window & return true or false.
      implicit none
      character level*(*),pname*(*),result*80,getpar*80
      integer window,status,smg$put_with_scroll,pbuf
      logical mb_getpar
      mb_getpar=.false.
      result=getpar(level,pname,pbuf)
      if(result(1:1).ne.' ') then
        status=smg$put_with_scroll(window,result)
        mb_getpar=.true.
        endif
      return
      end

      function mbacot(x)
c version of arc-contangent that fails softly for x=0
      real mbacot,x,pi,small
      parameter(small=1e-10,pi=3.14159265)
      if(abs(x).gt.small) then
        mbacot=atan(1.0/x)
      else
        if(x.ge.0) then
          mbacot=pi/2.0
        else
          mbacot=-pi/2.0
        endif
      endif
      return
      end

      function mbcot(x)
c version of cotangent that wont return 0 or infinity.
      implicit none
      real mbcot,x,s,c,small
      parameter(small=1e-10)
      s=sin(x)
      c=cos(x)
      if(abs(s).gt.small) goto 1
        if(s.lt.0.) s=-small
        if(s.ge.0.) s=+small
1     if(abs(c).gt.small) goto 2
        if(c.lt.0.) c=-small
        if(c.ge.0.) c=+small
2     mbcot=c/s
      return
      end


      function setstubx(active,stubs)
c move those active stub tuners to 100mm above the requested position
c then move them up to the position requested (avoid hysteresis)
c wait for all tuners to arrive.
c timeout(20sec) => return false.
      implicit none
      integer setstubx,setstub,stubs(4),i,k,w1,w4,pbid,status
     &    ,smg$paste_virtual_display,stubs_under(4)
     &    ,smg$unpaste_virtual_display
     &    ,smg$put_with_scroll
      logical active(4)
      common pbid,w1,w4
```

```
      status=smg$paste_virtual_display(w4,pbid,9,5)
      status=smg$put_with_scroll(w4,'Moving Tuners now:')
      do i=1,4
        stubs_under(i)=min(2500,max(0,stubs(i)+100))
        enddo
      if(stubs(i).le.2500.and.stubs(i).ge.0)
   &    setstubx=setstub(active,stubs_under,5)
   &            +setstub(active,stubs       ,1)
      call smg$unpaste_virtual_display(w4,pbid)
      return
      end

      function setstub(active,stubs,maxerr)
c set those stub tuners that are active to their new positions.
c poll continuously, waiting up to 40 sec for them all to move.
c tuner are deemed to have arrived if they are within <maxerr>
c of the requested value.
c return .false. if they are not moved by then
      implicit none
      real secnds,time
      integer setstub,stubs(4),pos(4),i,itry,err
   &    ,jtry,w1,status,pbid,w4,maxerr
   &    ,smg$put_chars
   &    ,smg$put_with_scroll
      character text6*6,text*72
      logical active(4),arrived
      common pbid,w1,w4
      setstub=0
      time=secnds(0.)
c set those stubs that need setting.
      do i=1,4
        if(active(i)) then
          itry=0
1         itry=itry+1
          call write_rl100(err,i,stubs(i))
            if(err.ne.0) then
            write(text,'(1x,i2,16H retries on stub,i1)') itry,i
            status=smg$put_with_scroll(w4,text)
            if(itry.le.10) goto 1
            setstub=-1
            endif
          endif
        enddo
c wait for all stubs to arrive.
2     arrived=.true.
      do i=1,4
        pos(i)=-99999
        if(active(i)) then
          jtry=0
4         jtry=jtry+1
          call read_rl100(err,i,pos(i))
          if(err.ne.0.and.jtry.le.4) goto 4
          if(err.ne.0) pos(i)=10000+pos(i)
          if(err.eq.0) write(text6,'(i4,2hmm)') pos(i)
          if(err.ne.0) write(text6,'(4Herr=,i1,1H )') err
          status=smg$put_chars(w1,text6,(i+3)/2,13-7*mod(i,2))
          arrived=(arrived.and.(abs(pos(i)-stubs(i)).le.maxerr))
          endif
        enddo
3     if(.not.arrived) then
        if(secnds(time).le.20.0) then
          goto 2
```

```
          endif
        status=smg$put_with_scroll(w4,'TUNERS STUCK!!')
        setstub=-1
        endif
      call update(active,pos,stubs)
      return
      end

      subroutine update(active,pos,stubs)
c update status file.
      implicit none
      character*80 status,write_file
      integer i,buffer(8),pos(4),stubs(4)
      logical active(4),change
      data buffer /8*-9999/
      save buffer
      change=.false.
      do i=1,4
        if(active(i)) then
          if(buffer(i).ne.pos(i)) then
            change=.true.
            buffer(i)=pos(i)
            endif
          if(buffer(i+4).ne.stubs(i)) then
            change=.true.
            buffer(i+4)=stubs(i)
            endif
          endif
        enddo
      if(change) status=write_file('usr:stub.status',buffer,8,'INT4')
      return
      end
```

```
      SUBROUTINE READ_RL100(ERR,LINE,POS)
C
C WRITTEN 21/8/92 BY MARK BALLICO
C READ TWO BYTES OF DATA FROM THE STUB CONTROL LINE <LINE>
C AND DECODE THE ACTUAL STUB POSITION IN MM.
C NOTE: 20 RETRIES FOR THE READ ARE PERMITTED GIVING AN ERROR OTHERWISE.
C NBB: PARITY ERROR ON READ IS IGNORED. WE SIMPLY TRY AGAIN FOR THE
C LOST BYTE.
C REFER TO : "MICROVMS PROGRAMMING SUPPORT MANUAL" QIO.1 (PP QIO-3)
C AND TO : "MICROVMS PROGRAMMERS MANUAL" 8.4.1 (PP 8-57)
C
      INTEGER ERR,LINE,POS,STATUS,i,I1,I2,M1,M2
     2   ,CODE,INPUT_BUFF_SIZE,INPUT_SIZE,SYS$QIOW
      INTEGER*2 INPUT_CHAN(4),IINPUT(5)
      PARAMETER(INPUT_BUFF_SIZE=10)
      CHARACTER*10 INPUT
      EQUIVALENCE(IINPUT,INPUT)
      COMMON /RL_100/ INPUT_CHAN
      INCLUDE '($IODEF)'
      STRUCTURE /IOSTAT_BLOCK/
        INTEGER*2 IOSTAT,
     2            TERM_OFFSET,
     2            TERMINATOR,
     2            TERM_SIZE
      END STRUCTURE
      RECORD /IOSTAT_BLOCK/ IOSB
      CODE=IO$_READVBLK.OR.IO$M_NOECHO
      I=0
1     STATUS=SYS$QIOW(,
     2                %VAL (INPUT_CHAN(LINE)),
     2                %VAL (CODE),
     2                IOSB,
     2                ,,
     2                %REF (INPUT),
     2                %VAL (INPUT_BUFF_SIZE),
     2                ,,,)
      IF(.NOT.STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
c     IF(.NOT.IOSB.IOSTAT) CALL LIB$SIGNAL(%VAL(IOSB.IOSTAT))
      INPUT_SIZE=IOSB.TERM_OFFSET
C
      IF(INPUT_SIZE.EQ.0) THEN
        IF(I.LE.20) THEN
          I=I+1
          GOTO 1
        ELSE
          POS=-99999
          ERR=-1
          RETURN
        ENDIF
        ENDIF
C
      I1=IINPUT(1)/256
      I2=MOD(IINPUT(1),256)
      M1=MOD(I1,64)
      M2=MOD(I2,64)
      IF(MOD(I1,128)/64.EQ.1) THEN
        POS=M1*64+M2-200
      ELSE
        POS=M2*64+M1-200
      ENDIF
      ERR=0
      END
```

```
      SUBROUTINE WRITE_RL100(ERR,LINE,POS)
C
C WRITTEN 25/8/92 BY MARK BALLICO
C SEND NEW STUB POSITION TO THE STUB-CONTROLLER OVER THE LINE <LINE>
C ENCODED AS 2 BYTES AS SPECIFIED IN SPINNER'S DEVICE SPECIFIECATION
C MANUAL.
C FOR MORE INFORMATION ON I/O HANDLING :
C REFER TO: "MICROVMS PROGRAMMING SUPPORT MANUAL"  QIO.2 (PP QIO-10)
C
      INTEGER ERR,LINE,POS,STATUS,I1,I2,CODE
    2  ,OUTPUT_BUFF_SIZE,OUTPUT_SIZE,SYS$QIOW
      INTEGER*2 OUTPUT_CHAN(4),IOUTPUT
      PARAMETER(OUTPUT_BUFF_SIZE=2)
      CHARACTER*2 OUTPUT
      EQUIVALENCE(OUTPUT,IOUTPUT)
      COMMON /RL_100/ OUTPUT_CHAN
      INCLUDE '($IODEF)'
      STRUCTURE /IOSTAT_BLOCK/
        INTEGER*2 IOSTAT,
    2             TERM_OFFSET,
    2             TERMINATOR,
    2             TERM_SIZE
      END STRUCTURE
      RECORD /IOSTAT_BLOCK/ IOSB
C ENCODE THE POSITION IN MM INTO 2 BYTES.
      IF(POS.LT.0.OR.POS.GT.2500) THEN         !ERROR.
        ERR=-1
        RETURN
        ENDIF                                  !ERROR.
      I1=MOD(POS,64)             !LOW BYTE + FLAG
      I2=POS/64+64               !HIGH BYTE + FLAG
      IOUTPUT=256*I2+I1          !EQUIVALENCED TO OUTPUT BUFFER.
C SEND THE DATA TO THE I/O LINE.
      CODE=IO$_WRITEVBLK.OR.IO$M_NOFORMAT
      STATUS=SYS$QIOW(,
    2         %VAL (OUTPUT_CHAN(LINE)),       !I/O LINE
    2         %VAL (CODE),                    !FUNCTION CODE.
    2         IOSB,                           !I/O STATUS BLOCK
    2         ,,
    2         %REF (OUTPUT),                  !BUFFER.
    2         %VAL (OUTPUT_BUFF_SIZE),        !BUFFER LENGTH.
    2         ,,,)
      IF(.NOT.STATUS) CALL LIB$SIGNAL(%VAL(STATUS))             !ERROR?
      IF(.NOT.IOSB.IOSTAT) CALL LIB$SIGNAL(%VAL(IOSB.IOSTAT))   !ERROR?
      ERR=0
      END
```

```
      SUBROUTINE READ_TIMEOUT(PROMPT,INPUT,INPUT_SIZE,TIMEOUT_COUNT)
C
C WRITTEN BY MARK BALLICO 20/8/92
C PROMPT SYS$INPUT WITH THE TEXT STRING <PROMPT> AND
C WAIT FOR A TEXT STRING <INPUT> IF NO INPUT FOR <TIMEOUT_COUNT>
C SECONDS THEN FALL THRU.
C NUMBER OF CHARACTERS READ IS RETURNED IN <INPUT_SIZE>
C IF <PROMPT> IS EMPTY THEN NO PROMPT IS WRITTEN.
C
      INTEGER*2 INPUT_CHAN
      INTEGER CODE,INPUT_BUFF_SIZE,PROMPT_SIZE,INPUT_SIZE
    2 ,TIMEOUT_COUNT,STATUS
      CHARACTER INPUT*(*),PROMPT*(*)
      INCLUDE '($IODEF)'
      INCLUDE '($SSDEF)'
C REFER TO "MICROVMS PROGRAMMING SUPPORT MANUAL" QIO-9
      STRUCTURE /IOSTAT_BLOCK/
        INTEGER*2 IOSTAT,
    2           TERM_OFFSET,
    2           TERMINATOR,
    2           TERM_SIZE
      END STRUCTURE
      RECORD /IOSTAT_BLOCK/ IOSB
C
      INTEGER*4 SYS$ASSIGN,SYS$QIOW
      INPUT_BUFF_SIZE=LEN(INPUT)
      PROMPT_SIZE=LEN(PROMPT)
      STATUS=SYS$ASSIGN('SYS$INPUT',INPUT_CHAN,,)
      IF(.NOT.STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
      IF(PROMPT.EQ.' ') CODE=IO$_READVBLK.OR.IO$M_TIMED
      IF(PROMPT.NE.' ') CODE=IO$_READPROMPT.OR.IO$M_TIMED
C REFER TO "MICROVMS PROGRAMMING SUPPORT MANUAL" QIO
      STATUS=SYS$QIOW(,                               !EFN.RLU.V
    2                 %VAL (INPUT_CHAN),              !CHAN.RWU.V
    2                 %VAL (CODE),                    !FUNC.RWU.V
    2                 IOSB,                           !IOSB.WQU.R
    2                 ,                               !ASTADR.RZEM.RP
    2                 ,                               !ASTPRM.RZ
    2                 %REF (INPUT),                   !P1.RZ
    2                 %VAL (INPUT_BUFF_SIZE),         !P2.RZ
    2                 %VAL (TIMEOUT_COUNT),           !P3.RZ
    2                 ,                               !P4.RZ
    2                 %REF (PROMPT),                  !P5.RZ
    2                 %VAL (PROMPT_SIZE))             !P6.RZ
C
      IF(.NOT.STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
      IF(IOSB.IOSTAT.NE.SS$_NORMAL
    2   .AND.IOSB.IOSTAT.NE.SS$_TIMEOUT)
    2   CALL LIB$SIGNAL(%VAL(IOSB.IOSTAT))
      INPUT_SIZE=IOSB.TERM_OFFSET
      RETURN
      END
```

```
        SUBROUTINE  OPEN_RL100(LINE)
C
C WRITTEN 21/8/92 BY MARK BALLICO.
C ALLOCATE AN I/O CHANNEL TO THE PORT TXA:<LINE> TO BE
C USED FOR STUB MATCHING CONTROL, AND LEAVE IT IN A COMMON BLOCK
C TO BE USED BY READ_RL100 AND WRITE_RL100.
C REFER TO: "MICROVMS PROGRAMMING SUPPORT MANUAL"  SYS-16
C AND TO  : "MICROVMS PROGRAMERS MANUAL" SECTION 8.4 (PP 8-56)
C
        INTEGER*2 INPUT_CHAN(4)
        CHARACTER*5 DEVICE
        INTEGER SYS$ASSIGN,STATUS,LINE
        COMMON /RL_100/ INPUT_CHAN
        WRITE(DEVICE,'(3HTXA,I1,1H:)') LINE
        STATUS=SYS$ASSIGN(DEVICE,INPUT_CHAN(LINE),,)
        IF(.NOT.STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        RETURN
        END
```