JLAB-TN-02-044

matbbu 2.4:
A Tool for Estimating Beam Breakup due to Higher Order Modes

K.B.Beard, L.Merminga, B.Yunn, TJNAF
28 August 2003

The matbbu, also known as MATRIXBBU, program was written by B.Yunn of TJNAF and CASA some time ago[1] (using parts of G.Krafft's tdbbu[2] [3] code), modified later by L.Merminga[4], and then by K.Beard. Its concept is outlined in CEBAF PR-87-007, "*Multipass Beam Breakup in Recirculating Linacs*" by J.Bisognano and R.Gluckstern[5], and its purpose is to predict the transverse beam breakup instability thresholds in recirculating linacs.

The tdbbu[6] and matbbu programs use the same physics[7] and input file and are very closely related, but they solve the problem in different ways. The former follows the position of the beam over time, while the latter solves for the eigenvalues of the matrix representing the system.

Very simply, matbbu reads in an input file describing the system as drift spaces, lenses, and cavities (in terms of their **h**igher **o**rder **m**odes or **HOM**s) and a recirculation matrix. This file is exactly that used by tdbbu and is very column specific.[8] The X and Y axes are treated sequentially and entirely independently. From that input, the program generates a complex matrix of size (2*n*-1)x(2*n*-1), where *n* is the number of HOMs under consideration on that axis.

That matrix is then evaluated at a single frequency; complex eigenvalues of that matrix that occur on the positive real axis correspond to a current instability at that frequency. Sweeping through the frequencies locates the current instabilities. A cutoff current (the default is 1 A) is used to discard eigenvalues whose either component's magnitude is larger than the cutoff; while not critical, it prevents wasting time on

---

1 B.Yunn, private communication
2 G.A.Krafft and J.J.Bisognano,"*Two Dimensional Simulations of Multipass Beam Breakup*," Proceedings of Particle Accelerator Conference 1987, p-1356.
3 L.Merminga, I.E.Campisi, *Higher-Order-Models and Beam Breakup Simulations in the Jefferson Lab FEL Rrecirculating Linac*, XIX International Linear Accelerator Conference, Aug 1998, ANL&FNAL, (http://accelconf.web.cern.ch/AccelConf/l98/PAPERS/TU4030.PDF)
4 L.Merminga, private communication
5 http://www.jlab.org/div_dept/admin/publications/87pub.html
6 JLAB-TN-02-045, *tdbbu 1.6: Another Tool for Estimating Beam Breakup due to Higher Order Modes*, K.B.Beard, L.Merminga, B.Yunn
7 J.J. Bisognano and R.L. Gluckstern, *Multipass Beam Breakup in Recirculating Linacs*, Proceedings of Particle Accelerator Conference 1987, p-1078
8 JLAB-TN-02-043, *TDBBU and MATBBU Input File Format*, K.B.Beard, L.Merminga, B.Yunn

solutions far from the range of interest.

Since the eigenvalues don't usually fall exactly on the positive real axis, but near it, some judgment was required of the user to go back, decrease the step size, and resweep a frequency region. In `matbbu`'s previous form, a talented user could "zero in" on a small region of interest iteratively and be reasonably confident of the solutions attained. However, in the case of the many HOMs with very high Q's, as for the Jlab 10KW FEL, this proved exceedingly difficult.

In the case of the 10KW FEL, there are 56 HOMs/axis, requiring that a complex 111x111 matrix's eigenvalues be found. This takes some significant time on any computer; in addition, the high Q's mean that the regions of interest are very narrow; only a few Hz wide. Typically, one would like to sweep a 2MHz wide space; doing that with just "brute force" by sweeping in 1Hz steps requires $2x10^6$ steps, or, very roughly for a 733MHz Pentium III, about that number of seconds (about 3 weeks) for each band (there were 11). In addition, this sweep should be repeated many times for a number of different HOM distributions representing manufacturing tolerances (typically 10). Putting that together, one would expect to use about **7 years** of CPU time.

That seemed too long a time to wait. To make matters worse, the time to solve a single matrix goes roughly as $N_{\mathrm{HOM}}^{3.6}$, so solving the case of for the CEBAF 12 GeV upgrade with 800 HOMs/axis rather than the FEL upgrade's 56 requires roughly 14,000 times the CPU time.

Fortuately, the eigenvalues' behavior is not chaotic; an infinitesimal change in the frequency should only change an eigenvalue slightly. This means that an active search can be used; the frequency can be changed up or down slightly and the eigenvalues tracked until they cross the positive real axis. The minimum frequency step size is then only limited by the numeric noise. The actual algorithm is somewhat more complicated and is discussed in more detail later.

Figures 1a and 1b below display the eigenvalues found using the same number of frequency steps, but Fig.1a uses a fixed step while Fig.1b uses the automatic hunting:

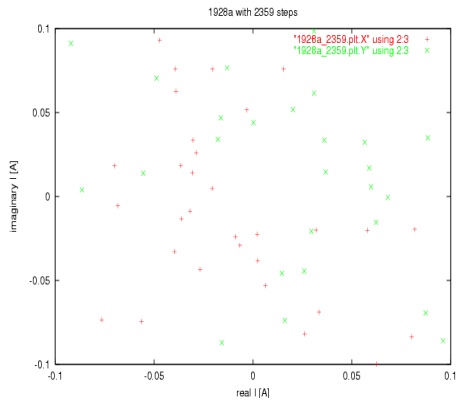

Fig.1a.

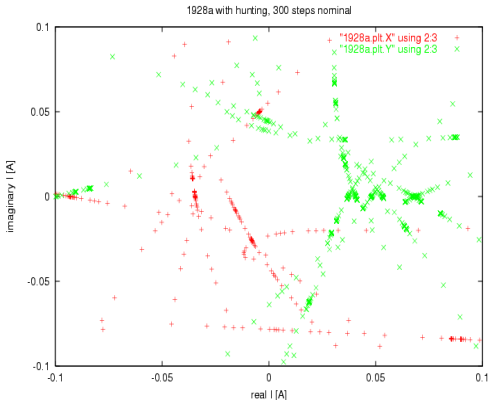

Fig.1b.

The original `matbbu` version only ran on a Cray[9] and used the IMSL[10] EVLCG routine. It was modified in a series of small steps and has reached maturity with matbbu 2.4.

matbbu 2.4 retains exactly the same algorithm to setup the matrix from the data, but has many improvements over the original:

- platform independence - can now run on nearly any UNIX
- library independence - can use either IMSL (faster, but proprietary) or the LAPACK[11] (slower, but freely available) math library
- command line interface - C style command line options
- automatic seeking - varies the frequency step and direction to find the eigenvalues of interest
- automatic Regions-of-Interest - greatly reduces the required CPU and memory by excluding HOMs outside the region of interest
- improved filenames - allows arbitrary filenames for input and output
- HOM spread - allows the HOMs to be distributed randomly according to either a uniform or Gaussian distribution
- repeating - allows many cycles to be done with different HOM distributions automatically
- allows the use of both DIMAD (m-radian) and STANDARD (cm-MeV/c) units in the recirculation matrix
- documentation - extensive documentation on the code and results

In addition, the algorithms to sort out the thresholds from among the eigenvalues are combined in the small, stand alone program `cheapseek`. The program `cheapseek` calculates no eigenvalues, it just examines the output of `matbbu`.

Input

The only input file required by matbbu is the file describing the accelerator; it is identical to the file used by tdbbu and is describe in a separate technote.[12] For historical reasons, the units used in the recirculation matrix may be either cm-MeV/c ("STANDARD) or meter-radian ("DIMAD") and are not specified in the format, so the choice must be specified on the command line (either `--STANDARD` or `--DIMAD`) and there is no default.

---

9 National Energy Research Scientific Computing Center, http://hpcf.nersc.gov/computers/J90/#retire

10 Visual Numerics, Inc., http://www.vsi.com/products/imsl/

11 LAPACK's User Guide, Third Edition, E.Anderson et al, http://www.netlib.org/lapack/

12 JLAB-TN-02-043, *TDBBU and MATBBU Input File Format*, K.B.Beard, L.Merminga, B.Yunn

The format is very column specific, and new files should be checked by examining the *.out.X and *.out.Y output file carefully. A new feature in matbbu is that it can use an exiting input file to generate new input files containing different HOM distributions by using the +MI, -MI, +MO, and -MO options. These new files can then be used by tdbbu or matbbu.

There are a number of options supported by matbbu to override the contents of the input file as well as control program execution. They greatly ease its use, allowing a single input file to be used describe a number of cases. For example, if a number of otherwise identical accelerators is to be simulated only varying in the exact values of the HOM frequencies, the -g *MHz* (Gaussian spread of *MHz* sigma), -R *N* (repeat *N* times), and -OL *I,J* (only apply to lines #*I*-#*J*) can be used to override the input file values. For each run the input file is reread and the new values substituted immediately after reading the input.

To reduce the required computation resources, matbbu generally excludes HOMs outside the region of interest by replacing them with zero length drift spaces (+S, the default), but can be forced to include them (-S). Refer to Appendix 2 for more details.

Output

By default, matbbu generates quite a number of output files. The extension describes the kind of file and its contents (Table 1). All the outputs are simple text with a header. The rootname (*) of the files is set on the command line; its default is just to use the input file name.

extension	description
*.out.X	Repeat of what was read from input file, along with any changes requested from the command line for the X axis
*.out.Y	Repeat of what was read from input file, along with any changes requested from the command line for the Y axis
*.plt.X	frequency, real current, and imaginary current for every eigenvalue calculated on the X axis
*.plt.Y	frequency, real current, and imaginary current for every eigenvalue calculated on the Y axis
*.candi	Candidates from among both X and Y eigenvalues found when crossing the positive real current axis.
*.nearest	Nearest eigenvalues to positive real current axis for both X and Y axis
*.rslt	Brief summary of results.
*.gp	Command file for GNUPLOT; plots all eigenvalues, then the nearest, and then zooms in if necessary

Table 1. matbbu output file extensions.

search algorithm

The single most important change to matbbu was the addition of the NEXT_FREQ subroutine. It searches for frequencies which produce eigenvalues on the positive real axis by examining the eigenvalues and determining the next frequency to evaluate. It is given both a nominal step size (the largest to use) and a smallest step size, below which it will stop searching and move on to just past the previous highest frequency it attained.

The list of all eigenvalues found to date are passed to the routine; only the most recent and the ones immediately preceding those are of interest here. The number of eigenvalues found can vary from 0 up to 2*n*-1 (where *n* is the number of HOMs in the axis of interest). A frequency is considered either OfInterest=YES or OfInterest=NO according to some rules...

- if positive real axis crossing was found by a previous solution (ForceOnward=YES), it steps the frequency a minimal amount past the previous highest frequency and continues the search

- if the number of eigenvalues found since the last step has changed, the frequency step is halved and the direction reversed, until the minimal frequency step is reached and declared OfInterest=YES

- if neither condition above is satisfied, current eigenvalues are matched
- with the nearest ones from the last step; if none, OfInterest=NO

- if a pair seems to have crossed the positive real axis, the step is halved and reversed and declared OfInterest=YES

- if a pair is extrapolated to cross the positive real axis on the next step, the step is halved and declared OfInterest=YES

- if neither condition is satisfied, the pairs are checked to see if their components changed by more than a certain fraction (typically 50%), and if so, declared OfInterest=YES

- if none of these conditions is satisfied, the frequency is declared OfInterest=NO and the step size doubled, but not to exceed the nominal step size, except when moving down in frequency- there, a run of frequencies (typically 5) must be OfInterest=NO before the step size is doubled (done to prevent the search from going very slowly near a point of interest).

The final threshold is found from the eigenvalues using the same BY0 and KB3

algorithms used by cheapseek and described in that section.

matbbu Example

As an example, consider the file "10kwfel_2094b.in"; it describes the Jlab 10KW FEL with 112 cavity HOMs (but only 56 in each axis). These HOMs have high Q's and hence require a very fine frequency step to identify the resonances. Here, the frequency range is specified as from 2093 to 2095 MHz with only 100 nominal steps, a cutoff current of 0.2 A, output files names to begin with 2094a, and verbose messages:

```
$> matbbu -i 10kwfel_2094a.in --DIMAD -f 2093:2095:100 -I 0.2  \
       -o 2094a -v
matrixbbuk: opened input "10kwfel_2094a.in" OK
matrixbbuk: opened output EigenFile "2094a.eigen" OK
matrixbbuk: opened output CandiFile "2094a.candi" OK
matrixbbuk: opened output ThrFile "2094a.rslt" OK
matrixbbuk: opened output NearestFile "2094a.nearest" OK
matrixbbuk: opened output GnuplotFile "2094a.gp" OK
matrixbbuk: opened output "2094a.out.X" OK
matrixbbuk: opened output "2094a.plt.X" OK
matrixbbuk: opened output "2094a.out.Y" OK
matrixbbuk: opened output "2094a.plt.Y" OK
 10kwfel_2094a.in:BY0: X_thr[mA]=    1.818 @freq[MHz]= 2094.309922  Y_thr[mA]=    1.917
@freq[MHz]=2094.409983
 10kwfel_2094a.in:KB3: Y_thr[mA]=    2.295 @freq[MHz]= 2094.749987  X_thr[mA]=    9.291
@freq[MHz]=2094.059049
```

The NEXT_FREQ routine reports when it thinks it has found a solution; these are logged into the 2094a.candi candidates file and used by matbbu's SEEKER routine. The 2094a.rslt file contains the same information as the final lines above; generally, only the KB3 algorithm need be considered and predicts a threshold of 2.3 mA on the Y axis and 9.3 mA on the X axis.

Note that the 1st estimate is provided by the BY0 original algorithm while the 2nd uses the KB3 algorithm described in the next section.

```
$> grep KB3 2094a.rslt
 10kwfel_2094a.in:KB3: Y_thr[mA]=    2.295 @freq[MHz]= 2094.749987  X_thr[mA]=    9.291
@freq[MHz]= 2094.059049  [476,64]
```

For convenience, a GNUPLOT[13] command file is written; this creates a quick display of all of the eigenvalues and those nearest (to the threshold). An example is shown in Fig. 2a and 2b.

```
$> gnuplot
gnuplot>  load "2094a.gp"
```

13 gnuplot 3.7, http://www.gnuplot.info/

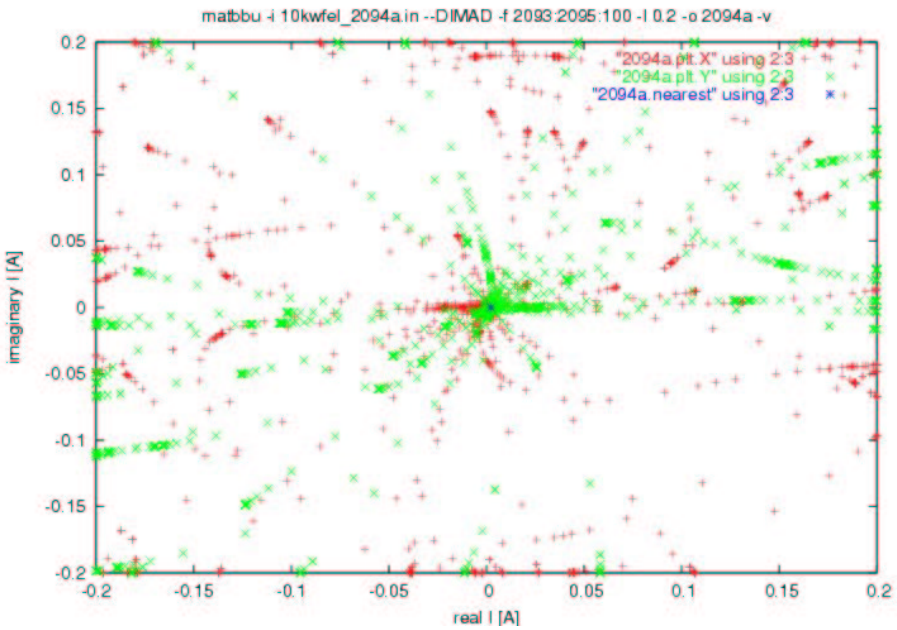

Fig.2a. Current eigenvalues.

2094a

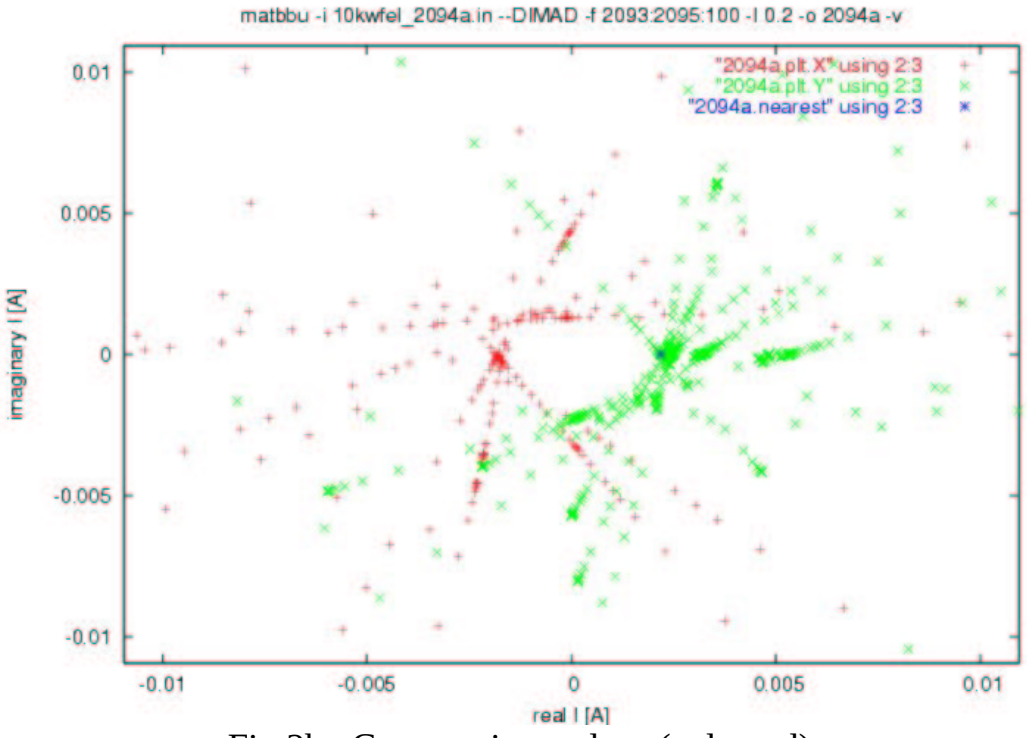

Fig.2b. Current eigenvalues (enlarged).

Additional examples are in Appendix 2.

cheapseek

The little program cheapseek can try to find the thresholds from the matbbu output. There are four algorithms to choose from in cheapseek:

- **BY0**: B.Yunn's original algorithm: Simply scans through the eigenvalues; if the absolute value of the imaginary part is a new minimum, it checks to see if the normalized eigenvalue is a new minimum too. If it is, that value is taken to be the threshold. Fast, used by `matrixbbuk`.

- **BY1**: B.Yunn's alternate algorithm: BY left a note in the code saying that perhaps the test on the normalized eigenvalue should be dropped. Fast.

- **KB2**: K.Beard's intensive algorithm: Reads in and reorders all eigenvalues by increasing frequency and attempts to find pairwise combinations that represent a crossing of the positive real axis; extrapolates to the crossing point. Very slow.

- **KB3**: K.Beard's simple algorithm: Simply scans through the list of threshold eigenvalue candidates produced by `matrixbbuk` and selects the one with the smallest real current. Very fast, used by `matrixbbuk`.

Only the BY0 and KB3 algorithms are used by matbbu. Note that the smallest threshold is printed *first*:

$> **cheapseek -i 2094a**

```
 2094a:BY0: X_thr[mA]= 1.308600 @freq[MHz]= 2094.079990 Y_thr[mA]= 1.895600 @freq[MHz]= 2094.309980
[1285,1358]
 2094a:BY1: Y_thr[mA]= 2.188300 @freq[MHz]= 2094.889918 X_thr[mA]= 2.731900 @freq[MHz]= 2094.990186
[1358,1285]
 2094a:KB2: Y_thr[mA]= 2.188290 @freq[MHz]= 2094.889918 X_thr[mA]= 19.863600 @freq[MHz]= 2094.037489 [12,1]
 2094a:KB3: Y_thr[mA]= 2.188700 @freq[MHz]= 2094.889918 X_thr[mA]= 19.866220 @freq[MHz]= 2094.037489
[1309,64]
```

Note that in this example agree on the Y axis threshold (2 mA), but strongly disagree on the X axis threshold (1 vs. 20 mA). The KB2 and KB3 algorithms closely agree and are in agreement with what is suggested by examining the plot in figure 2b. The BY0 algorithm's result is reported only for comparison with older calculations; the KB3 algorithm's result should be used for all other practical purposes.

Appendix 1. Help

Both matbbu and cheapseek have an internal help; the --help or -h option lists only the more commonly used options. The simplest way to use matbbu is to use the default values; refer to Appendix 2 for examples. For example:

$> **matbbu -i 12all --STANDARD**

This would write no messages except for errors, automatically create output filenames (of the form 12all.RoI_*n*.*), automatically select regions-of-interest, and automatically select the stepsize. In general, the order of options doesn't matter, and the last always takes preceedence.

To list only the most commonly used matbbu options, use the -h or --help option. Note that all options must be whitespace seperated; -hV is *not* the same as -h -V.

$> **matbbu --help**

```
  matbbu 2.4i1b2  15apr2003

 Calculates instability thresholds in a recirculating linac.

 form:
 $> matbbu [option [value]] [option [value]] ...
 -h  --help               - print brief list of options & quit [default *]
 +h  ++help               - print full list of options & quit
 -V  --version            - print version info & quit
 -v  --verbose            - print informational messages
     --DIMAD              - specify recirculation matrix as DIMAD (m-radian) units
     --STANDARD           - specify recirculation matrix as STANDARD (cm-MeV/c) units
 -i  --input        FILE  - input file
 -I  --Ignore       A     - ignore solutions above A amps [1.0000]
 -f  --frequency    RANGE - set fixed single frequency range in MHz
 +f  ++frequency          - choose a single frequency range automatically
 +a  ++automatic          - choose multiple frequency ranges and step size automatically *
 -o  --output       ROOT  - output file rootname [<inputfile>]
     --notes              - print additional notes on usage
     --examples           - print some examples

  also see: http://casa.jlab.org/internal/code_library/code_library.shtml
```

while ++help or +h lists all the options. It should be noted that many of these additional options are rarely, if ever used.

$> **matbbu ++help**

```
  matbbu 2.4i1b2  15apr2003

 Calculates instability thresholds in a recirculating linac.

 form:
 $> matbbu [option [value]] [option [value]] ...
 -h  --help               - print brief list of options & quit [default *]
 +h  ++help               - print full list of options & quit
 -V  --version            - print version info & quit
 +V  ++version            - print longer version info & quit
 -v  --verbose            - print informational messages
 -q  --quiet              - print no informational messages
     --DIMAD              - specify recirculation matrix as DIMAD (m-radian) units
     --STANDARD           - specify recirculation matrix as STANDARD (cm-MeV/c) units
 -i  --input        FILE  - input file
     ++DIMAD              - specify CAVMAT matricies as DIMAD (m-radian) units
     ++STANDARD           - specify CAVMAT matricies as STANDARD (cm-MeV/c) units
 -C  --CECAV        FILE  - specify CEbafCAVity input file
     -+DIMAD              - specify both recirculation & cavity matricies as DIMAD (m-radian) units
     -+STANDARD           - specify both recirculation & cavity matricies as STANDARD (cm-MeV/c) units
 -I  --Ignore       A     - ignore solutions above A amps [1.0000]
 -s  --step         N     - set nominal number of frequency steps
 +s  ++step         F     - set step by fractional change in values [0.50] (<0=>fixed step)
 -sf --stepMHz      MHz   - set nominal stepsize in frequency
 -f  --frequency    RANGE - set fixed single frequency range in MHz
 +f  ++frequency          - choose a single frequency range automatically
 +a  ++automatic          - choose multiple frequency ranges and step size automatically *
 +w  ++widthRoI     SIGMA - set multiple of cavity spread to add to RoI [2.0]
```

```
-w  --widthRoI     MHz   - set minimum full width of a RoI in MHz [0.50000]
-r  --randomseed   N     - set random seed
-R  --Repeat       N     - repeat N times (with new random #s)
    --smallestfreq Hz    - set smallest frequency step to consider [0.500](Hz)
+S  ++Suppress     SIGMA - suppress HOMs more than SIGMA outside frequency region [2.0] *
-S  --Suppress           - consider all HOMs
-o  --output       ROOT  - output file rootname [<inputfile>]
-MI --MakeInput    ROOT  - create input files ROOT.n
+MI ++MakeInput          - create input files <inputfile>.n
-MO --MakeOnly     ROOT  - only create input files ROOT.n
+MO ++MakeOnly           - only create input files <inputfile>.n
-t  --threshold    FILE  - send threshold results a file  [<root>.rslt] *
+t  ++threshold          - no threshold results file
-E  --Eigenvalues  FILE  - record sorted eigenvalues to a file [<root>.eigen] *
+E  ++Eigenvalues        - no eigenvalues file
-G  --Gnuplot      FILE  - write Gnuplot commands to a file [<root>.gp] *
+G  ++Gnuplot            - no gnuplot command file
-N  --Nearest      FILE  - record nearest to threshold in a file[<root>.gp] *
+N  ++Nearest            - no nearest file
-k  --candidates   FILE  - record candidates into a file [<root>.candi] *
+k  ++candidates         - no candidate file
-H  --HOMlog             - do not record all HOM values
+H  ++HOMlog             - record all HOM values*
-X  --Xonly              - only consider the X axis
-Y  --Yonly              - only consider the Y axis
-XY --XY                - consider both the X & Y axes *
+CR ++CavityR      R     - override all input cavity R_Ls
+CQ ++CavityQ      Q     - override all input cavity Qs
+CF ++CavityFreq   MHz   - override all input cavity frequencies
-g  --gaussian     MHz   - put a gaussian spread on cavities' HOMs
-u  --uniform      MHz   - put a uniform spread on cavities' HOMs
-OL --OnlyLine     I,J   - only apply changes to lines #I through J (multiple regions allowed)
-LO --LOCK         I,J   - lock cavities on lines #I through J together (multiple regions allowed)
    --RoI_only           - list the automatic Regions-of-Interest & exit
    --totaltime    LIST  - verify a list of recirculation times (just use zeros for testing)
    --limits             - list compiled size limits and exit
    --notes              - print additional notes on usage
    --examples           - print some examples

 also see: http://casa.jlab.org/internal/code_library/code_library.shtml
```

To prevent the list from scrolling by too fast, use the standard UNIX trick:

$> **matbbu ++help | more**

The --limits option reports the maximum sizes that can be handled by matbbu as compiled.

$> **matbbu --limits**

```
 matbbu: compiled size ("mbbu.par") limits:
    1400 MCAV - max. # of cavities/RoI
    2000 MELM - max. total # of elements
    1400 MMOD - max. # of current loading modes
       5 MPAS - max. # of passes
 1000000 MAXIMUM_NUMBER_OF_POINTS - max # of eigenvalues to track
    2000 MaxRoI - max. # of Regions-of-Interest
```

These limits may be changed in the file "mbbu.par" and the program recompiled using the provided Makefiles, but it should be noted that the size of the program goes roughly as $MPAS^2 * MCAV^2$. The above case requires ~376 Mb of memory under Linux.

Also, the --examples option just prints some simple examples of using matbbu.

In practice, cheapseek is only used to compare the various algorithms used to extract the threshold value from the eigenvalues. To list the options for cheapseek:

$> **cheapseek --help**

```
cheapseek 2.2b8b 10jul2002

Looks for thresholds in matrixbbuk output.
```

```
form:
$> cheapseek [option [value]] [option [value]] ...
-h  --help              - print this help & quit [default]
-V  --version           - print version info & quit
+V  --version           - print full version info & quit
-v  --verbose           - print informational messages
-q  --quiet             - print no informational messages
-a  --alg         LIST  - use alternative algorithm(s) [0,1,2,3]
-i  --input       BASE  - input file basename [pfile]

     also see: http://www.jlab.org/~beard/FEL/MATRIXBBU
```

An easy way to seek the appropriate option is to just use the standard UNIX techinque of searching for a keyword, for example:

$> **matbbu +h | grep -i freq**

```
-s  --step         N     - set nominal number of frequency steps  [200]
-sf --stepMHz      MHz   - set nominal stepsize in frequency []
-f  --frequency    RANGE - fixed single frequency range in MHz
+f  ++frequency          - choose single frequency range automatically
+a  ++automatic          - choose multiple frequency ranges automatically*
    --smallestfreq Hz    - set smallest frequency step to consider [0.500](Hz)
+S  ++Suppress     SIGMA - suppress HOMs more than SIGMA outside frequency region [2.0]*
+CF ++CavityFreq   MHz   - override all input cavity frequencies
```

Appendix 2. Operational Guide

Like most UNIX programs, one tells matbbu what to do via command line options. Since some of the options consist of more than a single letter, all options and any arguments for those options be separated by whitespace (blanks or tabs). All options begin with either a "+" or "-" character; and most options have longer synonyms for clarity. The arguments to an option immediately follow that option, and in general the order of the options doesn't matter. In the case where options conflict, the lattermost options rules. There is a reasonable default for nearly every option except for the recirculation matrix units (to prevent mistakes), and reasonable error messages are generated if the input syntax is incorrect.

Note that the standard release of matbbu is a very large code that requires nearly 400 Mb of memory to run. For smaller problems, matbbu can be recompiled to use less memory (see the --limits option discussion in Appendix 1 and Appendix 3).

The input file format is a legacy and is **very** exacting; refer to the TN for details[14]. The input file contains all information required for the calculation with a few exceptions; it does **not** contain information on the units used in the recirculation matrix (either --DIMAD or --STANDARD should always be specified), nor the frequency range, nor step size of interest. Typically, the user specifies those quantities on the command line (or allows matbbu to select them automatically) and the name to use in creating the output files. If no output name is specified (-o *name*), the whole input filename is used as the base output filename.

The frequency step size is specifies the maxium frequency step to use within a frequency interval; too coarse, and solutions of interest may be missed entirely, too fine, and the search may take a very long time. Usually, the automatically selected maximum step size, chosen as the smallest ($\omega/Q/2$) within the region of interest, is sufficient; matbbu adjusts the step size up and down dynamically while searching for solutions (unless given the option +s *x* option with $x < 0.0$).

The following example will scan the input file and select a frequency range spanning all the HOMs with an automatically selected maximum step size (+f) and name the output to files as acc.in.* :

```
$> matbbu   -i  acc.in  --DIMAD    +f
```

The above example produces little in the way of output to the screen; the -v option will cause various status messages to be printed; using that option twice (-v -v) will generate a very verbose account of the search, eigenvalue by eigenvalue. The disadvantage of the above is that it can waste time if the frequencies of the HOMs are widely separated.

It is important to note that adding or removing lines is very likely to change the total

14 JLAB-TN-02-043, *TDBBU and MATBBU Input File Format,* K.B.Beard, L.Merminga, B.Yunn

recirculation time; this is a very important quantity. The time may be checked either by examining the `*.out.[X-Y]` files ("`grep -i total tmp.out.X`") or by using the --totaltime option:

$> **matbbu -i acc.in --DIMAD --totaltime 0**

For convenience, matbbu supports the use of Regions of Interest (ROIs) in frequency. **The use of ROIs does nothing that cannot be done through multiple runs with different frequency ranges,** but is often much more convenient. If a fixed frequency range (-f *low:high*) is not specified, a frequency region around each HOM in the file, extending a multiple of the HOM's sigma (defined here as *freq/Q* and set by the +w *sigma* option, default is 2) is created, and then overlapping regions are merged. If the default (+a) option is set, these regions become the ROIs, and if the +f option is set, a single ROI is created that includes the full range. These ROIs are then scanned sequentially, and the results tabulated. The option --RoI_only will force matbbu to just report the ROIs, but not act on them.

An example of allowing matbbu to select the ROIs and to send the output to files of the form accm.*; each region appears in a separate file:

$> **matbbu -i acc.in --DIMAD -o accm**

In the case where a specific range is desired (-f LOW_{MHz}:$HIGH_{MHz}$), the maximum step size should be specified using either the -s N_{steps} or -sf *MHz* options. The search algorithm +s *fraction* option should seldom be changed, but if it is given a negative fraction (+s -1), matbbu will just used the maximum step size as a fixed step. This may be used simulate early versions of matbbu.

Another convenience, which also **does nothing but save much tedious editing**, is the ability to override the CAVITY values using the +CR *ohms*, +CF *MHz*, +CQ *Q* options. The (not necessarily contiguous) range of lines in which this applies may be specified using multiple -OL *low,high* options. Any values specified replaces those in the CAVITY statements, but only within the ranges specified. In addition, the frequencies may further changed from the nominal value by a randomly distributed gaussian (-g σ_{MHz}) or uniform (-u ΔF_{MHz}) distribution.

In the case of supercells, a range of neighboring cavities share a single frequency; this may be specified with the (possibly multiple) -LO *low,high* option(s).

In order to simulate a set of accelerators, it is convenient to use the -R *repeat* and -r *seed* options.

These constructions may save many hours of careful editing, but to avoid mistakes, it is often best to use the -MO *name* or +MO options to generate, but not actually run, the input files. The files can then be examined and later given to matbbu as simple input files.

For example, to replace the values in CAVITY statements in lines 91-129 in the file "generic.mode", next apply a gaussian distribution with a sigma of 5 MHz wide to those frequencies, then run that input only consider the frequency range 1861-1881 MHz with a minimum of 2000 steps:

```
$> matbbu -i generic.mode  --DIMAD  -OL 91,129  +CR 86.0  +CQ 2.6E4 \
     +CF 1871.4  -g 5.  -f 1861:1881 -s 2000 -o rchw5
```

If one wanted to only create (but not run) input files for 3 accelerators "as built":

```
$> matbbu  -R  3   -OL 91,129   -i generic.mode   --DIMAD  -g 5.  \
    +CR 86.0   +CQ 2.6E4   +CF 1871.4   -MO  case  -o tmp  -f 0:9999
```

and the input files created would be "case.1", "case.2", and "case.3". The "tmp.*" output files may be ignored except for debugging and a frequency range must be specified. To run "case.1" and only considering frequencies near the HOM specified:

```
$> matbbu  -i  case.1  --DIMAD  -f 1861:1881  -s 2000  -o case_1
```

The frequency range could be left on automatic, but that will generally also spend time on modes not of interest. To just list the modes that matbbu would examine, if left on its own:

```
$> matbbu -i js15a.1  --DIMAD  --RoI_only
matrixbbuk: opened input "js15a.1" OK

 << 10kW IR FEL 145 MeV, 18Nov02 3CMs [5,SuperCell,5]cells >>

matrixbbuk: automatic RoI: 1812.260:1812.760:19(1)
matrixbbuk: automatic RoI: 1813.040:1813.540:19(1)
matrixbbuk: automatic RoI: 1815.701:1818.926:128(7)
matrixbbuk: automatic RoI: 1819.142:1825.805:266(16)
matrixbbuk: automatic RoI: 1825.858:1829.582:148(7)
matrixbbuk: automatic RoI: 1882.580:1883.080:19(1)
matrixbbuk: automatic RoI: 1883.750:1884.250:19(1)
matrixbbuk: automatic RoI: 1885.369:1886.311:37(1)
matrixbbuk: automatic RoI: 1886.590:1888.880:91(6)
matrixbbuk: automatic RoI: 1889.031:1890.789:70(4)
matrixbbuk: automatic RoI: 1890.870:1891.370:19(1)
matrixbbuk: automatic RoI: 1891.989:1898.564:263(18)
matrixbbuk: automatic RoI: 1962.745:1964.315:62(1)
matrixbbuk: automatic RoI: 1964.698:1980.453:630(31)
matrixbbuk: automatic RoI: 2097.609:2098.629:40(40)
matrixbbuk: automatic RoI: 2102.575:2103.121:21(20)
matrixbbuk: automatic RoI: 2106.668:2107.647:39(40)
matrixbbuk: automatic RoI: 2110.634:2111.181:21(20)
```

The width of the ROIs may be specified when the cases are generated and

immediately run using the +w *Nsigma* option, and similarly, if all the HOMs are known and there is no spread in frequencies introduced by matbbu, a minimum width for the RoIs may be set by using the -w *MHz* option.

$> matbbu -i 10kwfel_2094a.in --DIMAD -w 1.0 --RoI_only

```
matrixbbuk: opened input "10kwfel_2094a.in" OK

 << 10kW IR FEL 145 MeV, April 2002 3CMs [5,7,5]cells >>

matrixbbuk: automatic RoI: 1812.010:1813.790:71(2)
matrixbbuk: automatic RoI: 1815.701:1818.926:128(7)
matrixbbuk: automatic RoI: 1819.142:1825.805:266(16)
matrixbbuk: automatic RoI: 1825.858:1829.582:148(7)
matrixbbuk: automatic RoI: 1882.330:1883.330:39(1)
matrixbbuk: automatic RoI: 1883.500:1884.500:39(1)
matrixbbuk: automatic RoI: 1885.340:1888.960:144(7)
matrixbbuk: automatic RoI: 1889.031:1891.620:103(5)
matrixbbuk: automatic RoI: 1891.989:1898.564:263(18)
matrixbbuk: automatic RoI: 1962.745:1964.315:62(1)
matrixbbuk: automatic RoI: 1964.698:1980.453:630(31)
matrixbbuk: automatic RoI: 2093.540:2095.490:77(16)
```

$> matbbu -i 10kwfel_2094a.in --DIMAD -w 5.0 --RoI_only

```
matrixbbuk: opened input "10kwfel_2094a.in" OK

 << 10kW IR FEL 145 MeV, April 2002 3CMs [5,7,5]cells >>

matrixbbuk: automatic RoI: 1810.010:1830.220:808(32)
matrixbbuk: automatic RoI: 1880.330:1898.564:729(32)
matrixbbuk: automatic RoI: 1961.030:1980.453:776(32)
matrixbbuk: automatic RoI: 2091.540:2097.490:237(16)
```

Use of multiple ROI and repeated runs produce multiple output files; the easiest way to find the resulting thresholds is to just search on the KB3 algorithm solutions:

$> grep KB3 hg[X,Y]8_q6.*.rslt

```
hgX8_q6.in.1.rslt: hgX8_q6.in.1:KB3: X_thr[mA]=    6.246032 @freq[MHz]= 2059.774791
Y_thr[mA]=    9.519763 @freq[MHz]= 2060.497149  [3515,4675]
hgX8_q6.in.2.rslt: hgX8_q6.in.2:KB3: Y_thr[mA]=    5.772718 @freq[MHz]= 2061.833026
X_thr[mA]=    6.850592 @freq[MHz]= 2061.124726  [8107,5065]
hgX8_q6.in.3.rslt: hgX8_q6.in.3:KB3: X_thr[mA]=    6.771215 @freq[MHz]= 2059.467662
Y_thr[mA]=    9.609201 @freq[MHz]= 2061.164213  [1739,5423]
hgY8_q6.in.1.rslt: hgY8_q6.in.1:KB3: X_thr[mA]=    6.246032 @freq[MHz]= 2059.774791
Y_thr[mA]=    9.519763 @freq[MHz]= 2060.497149  [3515,4675]
hgY8_q6.in.2.rslt: hgY8_q6.in.2:KB3: Y_thr[mA]=    5.772718 @freq[MHz]= 2061.833026
X_thr[mA]=    6.850592 @freq[MHz]= 2061.124726  [8107,5065]
hgY8_q6.in.3.rslt: hgY8_q6.in.3:KB3: X_thr[mA]=    6.771215 @freq[MHz]= 2059.467662
Y_thr[mA]=    9.609201 @freq[MHz]= 2061.164213  [1739,5423]
```

Often, if many HOMs are present, it is convenient to leave it up to matbbu to make reasonable choices (here with automatic selections of ROIs, a maximum frequency step of 0.01 MHz, and a minimum ROI width of 1.0 MHz):

**$> matbbu --DIMAD -i fel_nl11_case1.in +a -sf 0.01 -w 1.0 **
-o felnl11case1 -v -v

In this example, there were 81 ROIs, and so 81 felnl11case1.RoI_*n*.rslt files (where *n* ran from 1 to 81). To produce a plot (Fig.3) of threshold vs. frequency, it is only necessary to extract the thresholds and place them into a file suitable for plotting (beshuffled[15] is just one of the author's handy little utility programs):

15 http://casa.jlab.org/internal/code_library/casa_lib/MISC/DOC/index.html#beshuffled

$> **grep KB3 felnl11case1.RoI_*.rslt | beshuffled -l 5:3 > case1.rall**
$> **gnuplot**
gnuplot> **set log y**
gnuplot> **set xlabel "frequency[MHz]"**
gnuplot> **set ylabel "threshold[mA]"**
gnuplot> **set title "FEL with NL11 (case1)"**
gnuplot> **plot "case1.rall"**

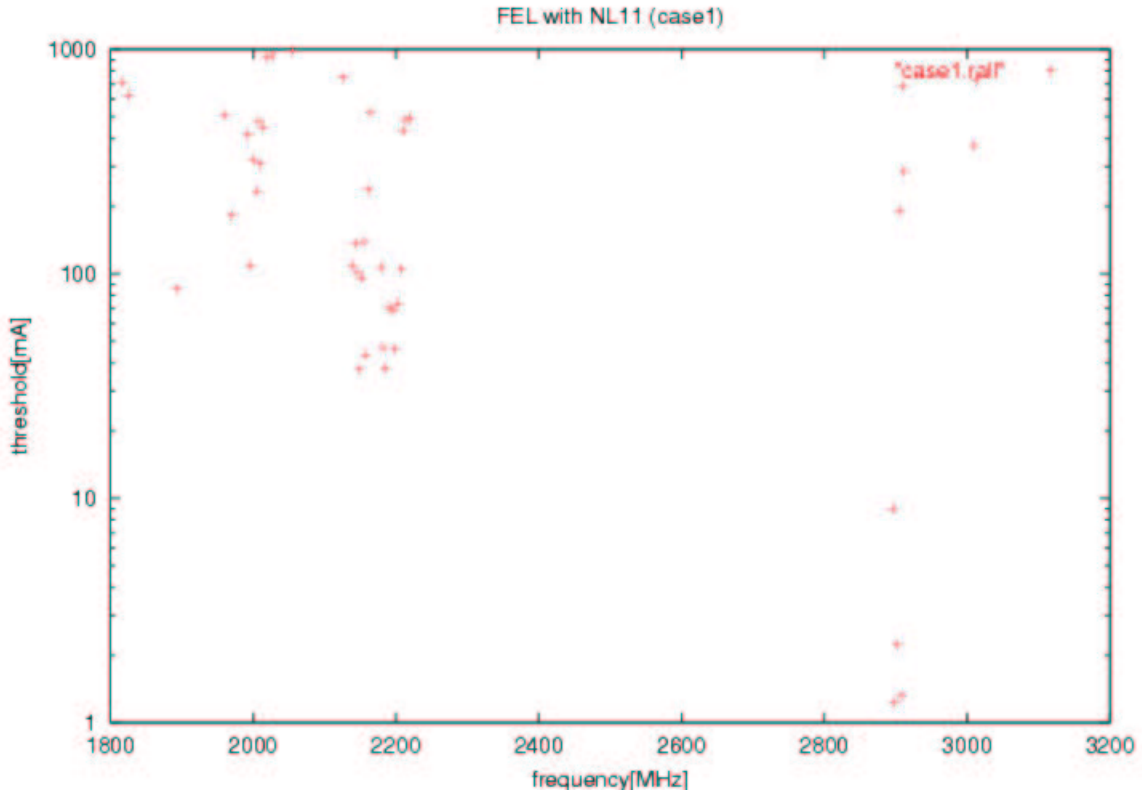

Fig.3 gnuplot output

Appendix 3. Obtaining matbbu

The matbbu and cheapseek source codes, example files, and executable binaries for a number of UNIX platforms (Linux, SunOS, HP-UX, AIX, ...) are available both onthe Jlab CUE filesystem[16] at `/group/casa/SUPPORTED/matbbu` and from the CASA Code Library site http://casa.jlab.org/internal/code_library/code_library.shtml.

To run matbbu from a CUE machine, the appropriate directory, `/group/casa/SUPPORTED/matbbu/EXE.`*ostype*, must be in the user's UNIX PATH, where *ostype* is Linux, SunOS, HP-UX, and so forth .

For use outside of the CUE system, the executable only need be downloaded, made executable (`chmod a+x` *file*), and run, but note that binaries built using the IMSL library generally require an IMSL license to run.[17] The standard version of matbbu typically requires **~400 Mb** of memory, but this may be reduced (for smaller systems of interest) by recompiling and relinking the code (refer to the --limits option discussion in Appendix 1 and the recompilation instructions in Appendix 5).

16 http://cc.jlab.org/cue/

17 Visual Numerics, Inc., http://www.vsi.com/products/imsl/

Appendix 4. Comparison of tdbbu and matbbu

Estimates of the 10KW FEL HOMs was made using both tdbbu and matbbu, and the results compared in that note.[18]

As tdbbu "sees" all frequencies of higher order modes, while matbbu only a range in frequency, tdbbu and matbbu may give very different results if there are instabilities at frequencies outside the range given matbbu.

input file	#HOMs		most	matbbu	tdbbu	matbbu	tdbbu	matbbu	tdbbu
	X	Y	significant Q	cmd#	cmd#	X [mA]	X [mA]	Y [mA]	Y [mA]
e4.in	1	1	1.00E+04	1	7	30680	30121 - 31455	-	230754 - 240970
e5.in	1	1	1.00E+05	2	8	3055.1	3012 - 3145	-	229251 - 239401
e6.in	1	1	1.00E+06	3	9	305.4	295.8 - 308.9	-	233570 - 243911
e7.in	1	1	1.00E+07	4	10	30.56	29.6 - 32.3	-	222216 - 232054
10kwfel_1724a.in	56	56	2.50E+07	5	11	197.1	176.8- 184.6	119.9	114.6- 119.7
10kwfel_1876a.in	56	56	6.20E+05	6	12	6.71	7.50 - 7.83	18.9	19.40 - 20.25

Table 2. matbbu vs. tdbbu comparison

```
 1    matbbu --DIMAD -i e4.in -o e4m -I 1.E10 -v
 2    matbbu --DIMAD -i e5.in -o e5m -I 1.E6 -v
 3    matbbu --DIMAD -i e6.in -o e6m -I 1.E6 -v
 4    matbbu --DIMAD -i e7.in -o e7m -I 1.E10 -v
 5    matbbu --DIMAD -i 10kwfel_1724a.in -v
 6    matbbu --DIMAD -i 10kwfel_1876a.in -s 300 -f 1874.5:1877.5 -o 1876a -v
 7    tdbbu --DIMAD -i e4.in -op e4t -G -j 1.E8 +S -m 30
 8    tdbbu --DIMAD -i e5.in -op e5t -G -J 1.E6 -v +S
 9    tdbbu --DIMAD -i e6.in -G -m 30 -v +S -op e6t
10    tdbbu --DIMAD -i e7.in -G -m 30 -v +S -op e7t
11    tdbbu --adjustprint --DIMAD -i 10kwfel_1724a.in -G -j 40000. +S -op t_1724a
12    tdbbu --adjustprint --DIMAD -i 10kwfel_1876a.in -G -j 40000. +S -op t_1876a
```

Table 3. matbbu and tdbbu command lines

The choice of whether to use tdbbu or matbbu to estimate thresholds depends on the characteristic time and the number of HOMs involved. The required computational time goes roughly with the Q values for tdbbu, and for very high Qs, running tdbbu may become impractical. For matbbu, the computation time grows very rapidly with the number of HOMs within close frequency proximity ($\sim N_{\mathrm{HOM}}^{3.6}$) and is relatively insensitive to the size of the Q values.

18 JLAB-TN-02-042, *Estimates of the Beam Breakup Thresholds in the 10KW FEL due to HOMs*,K.B.Beard, L.Merminga, B.Yunn

Appendix 5. matbbu code

matbbu is an old code developed with little regard for maintenance, but considerable progress has been made toward making it easier to maintain, modify and use. The current version is 2.4i5g3.

A notable feature of matbbu is that it uses C for both the main routine and for some of the lowest level routines (shown in lower case in Figure 4), while the bulk of the code is in FORTRAN (shown in upper case in Figure 4). This was done to allow the code to read options from the command line and to provide platform independence.

Rebuilding either code requires either the standard LAPACK[19] or IMSL libraries, as well as matbbu's SUPPORT and the author's KBB[20] and KWRAP[21] libraries, available from the same site as matbbu. The SUPPORT libraries are used to adapt matbbu to either the LAPACK or IMSL library (IMSLtoLAPACK) and avoid some platform dependencies in complex operations (CMPLX2) and random number generators (MISCMATH). The KWRAP and KBB libraries are used to provide general platform independence. All have their own Makefiles; either LAPACK or IMSL should be installed first, then KBB, followed by KWRAP, followed by the SUPPORT libraries, and finially matbbu itself. Very approximately, the IMSL library version seems to run about twice as fast as the LAPACK library version. At the time this is written, TJNAF only supports IMSL under SunOS.

It is strongly suggested that the author's organizational guide[22] [23] be read before attempting to rebuild the program, as the Makefiles depend on several (KBB_*) environmental variables to describe local idiosyncrasies. Once the files have been unpacked, the variables set for a supported platform, and the paths to the appropriate libraries set in the Makefile, it is only necessary to run the Makefile.

$> make help

```
make -f Makefile.HP-UX help
        #
        # make <command>
        #
        # all            - update everything
        # help           - print this help
        # show           - print current value of required symbols and their definitions
        # clean          - delete all binaries for this platform
        # distclean      - delete all binaries for all platforms
        # create_os      - create all binary directories for this platform
        # destroy_os     - delete all binary directories for this platform
        # destroy_all_os - delete all binary directories for all platforms
        # archiveall     - create a compressed tar backup of everything
        # archive        - create a compressed tar backup for export
        #
```

19 LAPACK's User Guide, Third Edition, E.Anderson et al, http://www.netlib.org/lapack/

20 KBB 7.5g Library, K.Beard, http://casa.jlab.org/internal/code_library/casa_lib/KBB/DOC/

21 KWRAP: Kevin's Wrapper 0.1h, K.Beard, http://casa.jlab.org/internal/code_library/casa_lib/KWRAP/DOC

22 http://casa.jlab.org/internal/code_library/casa_lib/KBB/DOC/bs.html

23 JLAB-TN-03-001, *My Code Development Style, Organization, amd Platform Dependency Guide,* K.B.Beard

$> **make show**

```
# make -f Makefile.HP-UX show
# --------- required symbols ------------
printenv KBB_DEV         #- location of KBB related libraries
/home/beard/DEVELOPMENT
printenv KBB_OSTYPE      #- single keyword for current operating system
HP-UX
printenv KBB_CC          #- C compiler
gcc
printenv KBB_F77         #- FORTRAN compiler
g77
printenv KBB_F2C_RULE    #- suffix on F77 objects when linking C+F77
_
printenv KBB_F2C_CASE    #- name case on F77 objects when linking C+F77
LOWERCASE
# ----------------------------------------
```

$> **make destroy_all_os; make create_os; make all**

main	MATRIXBBUK	SAY_PARAM					
		SET_OPTION					
		UNSET_OPTION					
		HOM_SET	CHECK_OPTION				
		CHECK_OPTION					
		MATBBU_PREP	CLEANR8				
			CLEANI8				
			LAST_NONBLANK				
			PRECAL	CAVITY_INRANGE	CHECK_OPTION		
				CHECK_OPTION			
				FLUSH			
				LAST_NONBLANK			
				HOM_SHIFT	ranf_		
					GAUSSIAN_CUT	ranf_	
				MOD_MAKEINPUT	FLUSH		
			SUMCHECK	WHEREAT	FLUSH		
					INSERT_PARAM		
					WHEREINDEXMAP	FLUSH	
			TRCALC	SUMCHECK	WHEREAT	FLUSH	
						INSERT_PARAM	
						WHEREINDEXMAP	FLUSH
				MATMULT			
			MMULT	SUMCHECK	WHEREAT	FLUSH	
						INSERT_PARAM	
						WHEREINDEXMAP	FLUSH
				MATMULT			
		FLUSH					
		SORT_ROI					
		ranset_					
		MATBBU_GO	CLEANZ16				
			Z_EXP	Z_IMAG			
				Z_REAL			
			REPACK	FLUSH			
				ZUMCHECK	Z_IMAG		
					Z_REAL		

Key: [$c_{routine}$] [$FORTRAN_{routine}$] [**recursive**] ...*linking: FORTRAN:* XXXX(...) => *C:* xxxx_(...)

Fig 4. matbbu structure

Figure 4 shows the dependencies of the matbbu routines generated by splitcf [24]; a hyperlinked version is included with the source distribution. The main routine is in the upper left corner; it calls the routines in the next column, and those routines call the ones next to them in the column after that, and so forth. Each routine is briefly described below:

[*SRC/cavity_inrange.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]

```
LOGICAL*4 FUNCTION CAVITY_INRANGE( FREQ, Q )
```

24 http://casa.jlab.org/internal/code_library/casa_lib/SPLITCF/DOC/

REAL*8 *FREQ* - !(input) HOM frequency [MHz]
REAL*8 *Q* - !(input) HOM Q [-]

```
*
*     Returns whether the HOM falls within the
*     the range to consider.
*
```

called by: PRECAL

calls: CHECK_OPTION

[*SRC/cheapseek.f*]
INTEGER FUNCTION CHEAPSEEK( )

```
*
*  Quickly look through a MATRIXBBU plot file
*  and find the threshold current and frequency.
*
```

called by: main

calls: FLUSH , KEEKER

[*SRC/cheapseek.f*]
SUBROUTINE KEEKER(*NRS*, *RS*, *ITHR*, *FTHR*, *INDEX*)

INTEGER*8 *NRS* -
REAL*8 *RS(FRQ:NRM,*)* -
REAL*8 *ITHR* -
REAL*8 *FTHR* -
INTEGER*8 *INDEX* -

```
*
* Looks for the threshold current in the list of
* eigenvalues...
* KBB 5/13/02
*
```

called by: CHEAPSEEK

calls: SORTLIST

[*SRC/cheapseek.f*]
SUBROUTINE SORTLIST(*DIR*, *N*, *VAL*, *IDS*)

INTEGER*4 *DIR* -
INTEGER*4 *N* -
REAL*4 *VAL(*)* -
INTEGER*4 *IDS(*)* -

```
*
*  Sort a list either dir=UP or DOWN
*
```

called by: KEEKER

[SRC/check_option.f]
```
LOGICAL*4 FUNCTION CHECK_OPTION( OPT, WHAT )

      INTEGER*4 OPT -
      INTEGER*4 WHAT -

*
*     Returns whether all the what bits are
*     set in opt.
*
```

called by: CAVITY_INRANGE , HOM_SET , MAKEGPLT , MATRIXBBUK , PRECAL , SEEKER

[SRC/cleani8.f]
```
SUBROUTINE CLEANI8( ARRAY, SIZE )

      INTEGER*8 ARRAY(*) -
      INTEGER*8 SIZE -

*
*     just zeros an I*8 array of size elements
*
```

called by: MATBBU_PREP

[SRC/cleanr8.f]
```
SUBROUTINE CLEANR8( ARRAY, SIZE )

      REAL*8 ARRAY(*) -
      INTEGER*8 SIZE -

*
*     just zeros an R*8 array of size elements
*
```

called by: MATBBU_PREP

[SRC/cleanz16.f]
```
SUBROUTINE CLEANZ16( ARRAY, SIZE )

      COMPLEX*16 ARRAY(*) -
      INTEGER*8 SIZE -

*
*     just zeros an COMPLEX*16 array of size elements
*
```

called by: MATBBU_GO

[SRC/cmplx16_ev.f]
```
SUBROUTINE CMPLX16_EV( N2, MATRIX, N1, EVS )

      INTEGER*8 N2 -
      COMPLEX*16 MATRIX(*) - !dimension (N1,N2)
      INTEGER*8 N1 -
      COMPLEX*16 EVS(*) -
```

```
*----------------------------------------------------------------------
*
* A generic routine to find the eigenvalues EVs(*)
* of a complex*16 matrix(N1 X N2) matrix
* KBB 5/17/02
*----------------------------------------------------------------------
```

called by: MATBBU_GO

calls: EVLCG

[*SRC/cross_posxaxis.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]

```
SUBROUTINE CROSSES_POSXAXIS( A, B, HAS, WILL, C )
```

REAL*8 *A(FRQ:NRM)* - !input| test points
REAL*8 *B(FRQ:NRM)* - !input| test points
LOGICAL*8 *HAS* - !output| a-> b crossed +X axis
LOGICAL*8 *WILL* - !output| b-> c will cross +X axis
REAL*8 *C(FRQ:NRM)* - !output| projected +X crossing

```
*
*     Given point a,b, decides if line connecting Has
*     or Will on the next step cross the postive X axis
*     at c
*
```

called by: NEXT_FREQ

[*SRC/evlcg.f*]

```
SUBROUTINE EVLCG( N, A, LDA, EVAL )
```

INTEGER*8 *N* -
COMPLEX*16 *A(*)* - !dimension (LDA,N)
INTEGER*8 *LDA* -
COMPLEX*16 *EVAL(*)* -

```
*
* A cheap substitute for IMSL's EVLCG that
* uses the LAPACK library... KBB 4/25/02
*
* Note that on Cray, "COMPLEX" ==> "COMPLEX*16",
* so everything explicitly double precision...
*
*----------------------------------------------------------------------
* IMSL Name:  EVLCG/DEVLCG (Single/Double precision version)
* Purpose:    Compute all of the eigenvalues of a complex matrix.
*Arguments:
*   N       - Order of the matrix A.  (Input)
*   A       - Complex matrix of order N.  (Input)
*   LDA     - Leading dimension of A exactly as specified in the
*             dimension statement in the calling program.  (Input)
*   EVAL    - Complex vector of length N containing the eigenvalues
*             of A in decreasing order of magnitude.  (Output)
*
```

called by: CMPLX16_EV

[*SRC/flush.f*]

```
SUBROUTINE FLUSH( IOCHANNEL )
```

INTEGER *IOCHANNEL* -

```
*
```

```
*  A kludge for AIX to emulate the
*  I/O FLUSH of other systems
*
```

called by: CHEAPSEEK , MAKE_MAKEINPUT , MATBBU_GO , MATRIXBBUK , MOD_MAKEINPUT , PRECAL , REPACK , WHEREAT , WHEREINDEXMAP , ZUMCHECK

[*SRC/gaussian_cut.f*]

```
REAL*8 FUNCTION GAUSSIAN_CUT( NSIGMA )
```

```
REAL*8 NSIGMA -
```

```
*
*  returns a projected gaussian distribution
*  (clamped at -Nsigma<  <+Nsigma)
*
*  From: R.Li
*  Modified K.B.Beard 27feb2001,28jun01,9may02
*
c-----------------------------------------------------------------------
```

called by: HOM_SHIFT

calls: ranf_

[*SRC/hom_set.f*]
[*mbbu.par*] [*mbbu_opt.cmn*] [*mbbu_hom.cmn*]

```
SUBROUTINE HOM_SET( OPT, WIDTH )
```

```
INTEGER*4 OPT - !option
REAL*8 WIDTH - !MHz HOM spread
```

```
************************************************
*   Specify the type and with of distribution
*   for the HOMs
*
************************************************
```

called by: MATRIXBBUK

calls: CHECK_OPTION

[*SRC/hom_shift.f*]
[*mbbu.par*] [*mbbu_opt.cmn*] [*mbbu_hom.cmn*]

```
SUBROUTINE HOM_SHIFT( FREQ )
```

```
REAL*8 FREQ - !(input&output) MHz HOM spread
```

```
************************************************
*   Shift the HOM frequency appopriately as
*   specified by HOM_SET.
************************************************
```

called by: PRECAL

calls: ranf_ , GAUSSIAN_CUT

[*SRC/insert_param.f*]
[*mbbu_parameterlist.cmn*]

```
SUBROUTINE INSERT_PARAM( STRING )
```

CHARACTER*(*) *STRING* - !(input&output) parameter list

```
*
*    Seek and replace parameters in the string.
*    Parameters previous set in say_param.
*
```

called by: WHEREAT

[*SRC/last_nonblank.f*]
SUBROUTINE LAST_NONBLANK(*STRING*, *LAST*)

CHARACTER*(*) *STRING* -
INTEGER*8 *LAST* -

```
*
*       Return the index of the last nonblank
*       character in the string (at least 1)
*
```

called by: MATBBU_PREP , PRECAL

[*SRC/make_makeinput.f*]
[*mbbu_makeinput.cmn*]
SUBROUTINE MAKE_MAKEINPUT(*IOOLD*, *IONEW*)

INTEGER *IOOLD* - !already opened input channel
INTEGER *IONEW* - !already opened output channel

```
*
* Read in and update input file from IOold;
* write it to prepared channel IOnew
*
```

called by: MATRIXBBUK

calls: FLUSH

[*SRC/makegplt.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]
SUBROUTINE MAKEGPLT(*OUT*, *CMT*, *XPLOT*, *YPLOT*, *NEAR*, *THRID*, *THR*, *SIZE*)

INTEGER *OUT* - !prepared output IO channel
CHARACTER*(*) *CMT* - !arbitrary comment
CHARACTER*(*) *XPLOT* - !X plot file name
CHARACTER*(*) *YPLOT* - !Y plot file name
CHARACTER*(*) *NEAR* - !nearest points file name
INTEGER*8 *THRID* - !id# of threshold (0=NONE)
REAL*8 *THR(FREQUENCY:NORMALIZATION_EIGENVALUE)* - !threshold point
REAL*8 *SIZE* - !frame size

```
*
*    Just writes a command file for Gnuplot;
*    gnuplot> load "file"
*
```

called by: MATRIXBBUK

calls: CHECK_OPTION

[SRC/matbbu_go.f]
[mbbu.par] [mbbu_opt.cmn] [mbbu_tm_r.cmn] [mbbu_mx.cmn] [mbbu_dload.cmn] [mbbu_go.cmn]

SUBROUTINE MATBBU_GO(AXIS, CNTR, FREQ, ICUT, NRSLT, RSLT, RSLTID)

INTEGER*8 *AXIS* - !either Xaxis or Yaxis
INTEGER*8 *CNTR* - !ID counter
REAL*8 *FREQ* - !MHz|frequency of interest
REAL*8 *ICUT* - !A| ignore currents above
INTEGER*8 *NRSLT* - !running size of result list
REAL*8 *RSLT(FREQUENCY:NORMALIZATION_EIGENVALUE,*)* - !MHz:A:A:A| eigenvalue rslt
INTEGER*8 *RSLTID(*)* -

```
C=====================================================================
C                              MATRIX BBU
C23456789012345678901234567890123456789012345678901234567890123456789012
C The latest version:
C UPDATED ON 8/1/01 - requires IMSL routine EVLCG on Cray
C Handles energy recovery and recirculation matrices in DIMAD units.
C Subroutine PRECAL can be imported from TDBBU except the CAVITY card. Also,
C last two lines in common block containing TM can be commented out.
C MCAV >= NDIM : Note that IF(LSUB) then NDIM=NCAV*NPASS-1 else NDIM=NCAV.
*
* 4/29/02 - K.B.Beard, beard@jlab.org, ported to other UNIX platforms
*           via minor changes to source; moving complex functions to
*           CMPLX package, making a IMSLtoLAPACK package to avoid dependence
*           on IMSL library
*
* Goal is to have MATRIXBBU do both X,Y planes simultaneously...
*      this version is quickly and crudely modified to do that
*
* 5/6/02 - KBB - modified to be a subroutine for use with KWRAP package;
*                moved seeking threshold elsewhere
*
* 5/10/02 - KBB - broke main "MATRIXBBU" routine into 2 pieces for
*                  efficiency
*
* 5/17/02 - KBB - reorder eigenvalues by increasing size, generic
*                  routine replaced EVLCG for machine independence
*
C=====================================================================
* -KBB- cobble platform independence...
*
```

called by: MATRIXBBUK

calls: CLEANZ16 , Z_EXP , REPACK , ZUMCHECK , CMPLX16_EV , Z_REAL , Z_IMAG , FLUSH

[SRC/matbbu_prep.f]
[mbbu.par] [mbbu_opt.cmn] [mbbu_tm_r.cmn] [mbbu_mx.cmn] [mbbu_dload.cmn] [mbbu_go.cmn]

SUBROUTINE MATBBU_PREP(AXIS)

INTEGER*8 *AXIS* - !either Xaxis or Yaxis

```
C=====================================================================
C                              MATRIX BBU
C23456789012345678901234567890123456789012345678901234567890123456789012
C The latest version:
C UPDATED ON 8/1/01 - requires IMSL routine EVLCG on Cray
C Handles energy recovery and recirculation matrices in DIMAD units.
C Subroutine PRECAL can be imported from TDBBU except the CAVITY card. Also,
C last two lines in common block containing TM can be commented out.
C MCAV >= NDIM : Note that IF(LSUB) then NDIM=NCAV*NPASS-1 else NDIM=NCAV.
*
* 4/29/02 - K.B.Beard, beard@jlab.org, ported to other UNIX platforms
*           via minor changes to source; moving complex functions to
*           CMPLX package, making a IMSLtoLAPACK package to avoid dependence
*           on IMSL library
*
```

```
* Goal is to have MATRIXBBU do both X,Y planes simultaneously...
*      this version is quickly and crudely modified to do that
*
* 5/6/02 - KBB - modified to be a subroutine for use with KWRAP package;
*                moved seeking threshold elsewhere
*
* 5/10/02 - KBB - broke main "MATRIXBBU" routine into 2 pieces for
*                 efficiency
*
C=====================================================================
* -KBB- cobble platform independence...
*
```

called by: MATRIXBBUK

calls: CLEANR8 , CLEANI8 , LAST_NONBLANK , PRECAL , SUMCHECK , TRCALC , MMULT

[*SRC/matmult.f*]

SUBROUTINE MATMULT(*M*)

REAL*8 *M(3,4)* -

```
C=====================================================================
C       GIVEN TWO MATRICES m(1,E), m(2,E),  THIS THING MULTIPLIES
C       THE TWO MATRICES TOGETHER INTO A THIRD MATRIX, m(3,E).
C=====================================================================
```

called by: MMULT , TRCALC

[*SRC/matrixbbuk.f*]
[*mbbu.par*] [*mbbu_opt.cmn*] [*mbbu_override.cmn*]

INTEGER FUNCTION MATRIXBBUK( )

```
*-----------------------------------------------------------
*  This adapts the MATRIXBBU program to the KWRAP wrapper;
*  together they gives command line functionality to the
*  code.
*  5/1/02 K.Beard
*-----------------------------------------------------------
```

called by: main

calls: SAY_PARAM , SET_OPTION , UNSET_OPTION , HOM_SET , CHECK_OPTION , MATBBU_PREP , FLUSH , SORT_ROI , ranset_ , MATBBU_GO , NEXT_FREQ , SEEKER , SCANCAND , MAKE_MAKEINPUT , MAKEGPLT

[*SRC/mmult.f*]
[*mbbu.par*] [*mbbu_opt.cmn*] [*mbbu_tm_r.cmn*] [*mbbu_mx.cmn*]

SUBROUTINE MMULT( )

```
C=====================================================================
C     GIVEN THE MATRICES BETWEEN CAVITIES, THIS SUBROUTINE
C     CALCULATES ALL POSSIBLE "T"'S, STORING THE 1,2 ENTRIES IN "TRM."
C==MPAS,MCAV,4=======================================================================
*
```

called by: MATBBU_PREP

calls: SUMCHECK , MATMULT

[SRC/mod_makeinput.f]
[mbbu_makeinput.cmn]

SUBROUTINE MOD_MAKEINPUT(*LINE_NO*, *LINE*)

INTEGER*8 *LINE_NO* - !(input) line number - where to insert line
CHARACTER*(*) *LINE* - !(input) line - to be inserted

```
*
*  Request that input file line number#line_no
*  be updated using line.
*
```

called by: PRECAL

calls: FLUSH

[SRC/next_freq.f]
[mbbu.par] [mbbu_opt.cmn]

SUBROUTINE NEXT_FREQ(*CNTR*, *NV*, *V*, *ID*, *NOM*, *FRAC*, *CUT*, *FREQ*, *MSG*, *KEEPID*)

INTEGER*8 *CNTR* - !input| current attempt counter
INTEGER*8 *NV* - !input| current number of entries
REAL*8 *V(FREQUENCY:NORMALIZATION_EIGENVALUE,*)* - !input| entries
INTEGER*8 *ID(*)* - !input| attempts counter
REAL*8 *NOM* - !input| nominal frequency step [MHz]
REAL*8 *FRAC* - !input| factional deviation
REAL*8 *CUT* - !input| cutoff current [A]
REAL*8 *FREQ* - !input&output| frequency [MHz]
CHARACTER*(*) *MSG* - !output| info message
INTEGER*8 *KEEPID* - !output| id#Keep seems to be a valid zero crossing

```
************************************************************
*
*  Decide on the next frequency to try by estimating
*  the step required to maintain a fraction change of "frac" in
*  both real and imaginary values v.  Above the  cutoff current
*  "cut" use nominal step.
*
* KBB 5/13/02
************************************************************
```

called by: MATRIXBBUK

calls: CROSSES_POSXAXIS

[SRC/precal.f]
[mbbu.par] [mbbu_tm_r.cmn] [mbbu_dload.cmn] [mbbu_opt.cmn] [mbbu_prnt.cmn] [mbbu_hom.cmn] [mbbu_override.cmn]

SUBROUTINE PRECAL(*AXIS_OF_INTEREST*)

INTEGER*8 *AXIS_OF_INTEREST* - !(input) either X or Y axis (-X,Y for only X,Y)

```
C=====================================================================
C This routine prepares all the necessary parameters
*     (and reads the input file)
C     REINJECTION POINT MARKED BY > IN MACHINE LATTICE (CAVITY,LENS,DRIFT
C     OR MATRIX)
C=====================================================================
*- 4/29/02 - KBB - crudely modified to ignore axis NOT of interest
*- 7/5/02 - KBB - modified to possibly override input file cavity values
*- 4/8/03 - KBB - enhanced CAVMAT auxillary file handling
```

```
*
* The CAVMAT file describes the transfer matrix for each cavity; each
* entry corresponds to a CECAV card in the input file.
*
*********************************************************************
```

called by: MATBBU_PREP

calls: CAVITY_INRANGE , CHECK_OPTION , FLUSH , LAST_NONBLANK , HOM_SHIFT , MOD_MAKEINPUT

[*SRC/repack.f*]

SUBROUTINE REPACK(*A*, *UA*, *SA*, *B*, *UB*, *SB*)

COMPLEX*16 *A(*)* - !(input) A(sA,sA)
INTEGER*8 *UA* - !(input) used size of A
INTEGER*8 *SA* - !(input) declared size of A
COMPLEX*16 *B(*)* - !(output) B(uB,uB) - effectively smaller matrix
INTEGER*8 *UB* - !(input) used size of B
INTEGER*8 *SB* - !(input) declared size of B

```
C=====================================================================
C The subset m X m of matrix A declared sA X sA is converted to
* a n X n subset of matrix B declared as mB X mB. This routine eliminates
C unnecessary elements in A thereby reducing the size of ithe matrix.
C=====================================================================
*
*  For example, if A(1000,1000) but only A(1,1)->A(4,4) used,
*  then pretend B is B(4,4) and ignore actual declared size.
*
```

called by: MATBBU_GO

calls: FLUSH , ZUMCHECK

[*SRC/say_param.f*]
[*mbbu_parameterlist.cmn*]

SUBROUTINE SAY_PARAM(*NAME*, *VALUE*)

CHARACTER*(*) *NAME* - !(input) name of parameter
INTEGER*8 *VALUE* - !(input) value of parameter

```
*
*    Set a parameter for later use by whereindexmap
*    routine - used for debugging variable sized
*    arrays.
*
```

called by: MATRIXBBUK

[*SRC/scancand.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]

SUBROUTINE SCANCAND(*N*, *CAND*, *NRSLT*, *RSLT*, *ITHR*, *FREQTHR*, *INDEX*)

INTEGER*8 *N* - !input| # of candidates
INTEGER*8 *CAND(*)* - !input| candidate IDs
INTEGER*8 *NRSLT* - !input| number of entries
REAL*8 *RSLT(FREQUENCY:NORMALIZATION_EIGENVALUE,*)* - !input| entries
REAL*8 *ITHR* - !output| threshold current [A]
REAL*8 *FREQTHR* - !output| threshold frequency [MHz]

INTEGER*8 *INDEX* - !output| index within list of nearest p

```
*
*  Quickly look through a MATRIXBBU plot results
*  and find the threshold current thr_I [A]  and frequency
*  thr_Freq [MHz]
*
```

called by: MATRIXBBUK

[*SRC/seeker.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]
SUBROUTINE SEEKER(*NRESULTS*, *RESULTS*, *THR_I*, *THR_FREQ*, *INDEX*)

INTEGER*8 *NRESULTS* - !input| number of entries
REAL*8 *RESULTS(FREQUENCY:NORMALIZATION_EIGENVALUE,*)* - !input| entries
REAL*8 *THR_I* - !output| threshold current [A]
REAL*8 *THR_FREQ* - !output| threshold frequency [MHz]
INTEGER*8 *INDEX* - !output| index within list of nearest point

```
*
*  Quickly look through a MATRIXBBU plot results
*  and find the threshold current thr_I [A]  and frequency
*  thr_Freq [MHz]
*
```

called by: MATRIXBBUK

calls: CHECK_OPTION

[*SRC/set_option.f*]
SUBROUTINE SET_OPTION(*OPT*, *WHAT*)

INTEGER*4 *OPT* -
INTEGER*4 *WHAT* -

```
*
*     Set the what bit pattern in opt
*
```

called by: MATRIXBBUK

[*SRC/sort_freq.f*]
[*mbbu.par*] [*mbbu_opt.cmn*]
SUBROUTINE SORT_FREQ(*NV*, *V*)

INTEGER*8 *NV* - !input| number of entries
REAL*8 *V(FREQUENCY:NORMALIZATION_EIGENVALUE,*)* - !input| entries

```
*
*  Just do a bubble sort on the results v
*  by increasing frequency
*
```

[*SRC/sort_roi.f*]
[*mbbu.par*]
SUBROUTINE SORT_ROI(*N*, *LO*, *HI*, *CNT*, *EST*)

INTEGER*8 *N* - !(input&output) #of RoI

REAL*8 *LO(*)* - !(input&output) lo edge
REAL*8 *HI(*)* - !(input&output) hi edge
INTEGER*8 *CNT(*)* - !(input&outout) #HOMs inside each RoI
REAL*8 *EST(*)* - !(input&output) estimated required step size [MHz]

```
*
*  Sort Regions-of-Interest in accending order
*  and merge overlapping regions
*
```

called by: MATRIXBBUK

[*SRC/sumcheck.f*]
SUBROUTINE SUMCHECK(*WHERE*, *WHAT*, *ARRAY*, *SIZE*, *SUM*, *DUMPTO*)

CHARACTER*(*) *WHERE* - !(input) name of calling routine
CHARACTER*(*) *WHAT* - !(input) comment (may be declaration)
REAL*8 *ARRAY(*)* - !(input) array to check
INTEGER*8 *SIZE* - !(input) total size of array
REAL*8 *SUM* - !(output) sum of absolute values in array
INTEGER *DUMPTO* - !(input) IO channel(!=0) in which to dump every value

```
*
*     simply sums up the absolute values of
*     an array of size and returns the sum -
*     just for checking for errors
*
```

called by: MATBBU_PREP , MMULT , TRCALC

calls: WHEREAT

[*SRC/trcalc.f*]
[*mbbu.par*] [*mbbu_opt.cmn*] [*mbbu_tm_r.cmn*] [*mbbu_mx.cmn*]
SUBROUTINE TRCALC(*AXIS_OF_INTEREST*)

INTEGER*8 *AXIS_OF_INTEREST* -

```
C=======================================================================
C     GIVEN THE TRANSFER MATRICES BETWEEN CAV, LENS, DRIFT,
C     & RECIRC, THIS SUBROUTINE CALCULATES THE MATRICES (TCM)
C     BETWEEN CAVITIES.
* 4/30/02 KBB - crudely modified to select axis_of_interest -
*               either Xaxis or Yaxis
C=======================================================================
```

called by: MATBBU_PREP

calls: SUMCHECK , MATMULT

[*SRC/unset_option.f*]
SUBROUTINE UNSET_OPTION(*OPT*, *WHAT*)

INTEGER*4 *OPT* -
INTEGER*4 *WHAT* -

```
*
*     Unset the what bit pattern in opt
*
```

called by: MATRIXBBUK

[*SRC/whereat.f*]
```
SUBROUTINE WHEREAT( DECLARED_AS, WHERE, STRING, LSTRING, SIZE )
```

```
CHARACTER*(*)  DECLARED_AS - !(input) variable declaration
INTEGER*4 WHERE - !(input) 1-dimensional location
CHARACTER*(*)  STRING - !(output) conventional string NAME(i,...,k)
INTEGER*4 LSTRING - !(output) length of string
INTEGER*8 SIZE - !(output) calc. size, if possible
```

```
*
*     Given a FORTRAN variable declaration string and
*     its 1D index, returns a string in the conventional
*     form.
*
*     For example declared_as="V(10,3,-1,1)" and
*     where=5 --> string="(5,1,-1)"
*
```

called by: SUMCHECK , ZUMCHECK

calls: FLUSH , INSERT_PARAM , WHEREINDEXMAP

[*SRC/whereindexmap.f*]
```
SUBROUTINE WHEREINDEXMAP( NDIM, DIMLO, DIMHI, WHERE, OK, STR )
```

```
INTEGER*4 NDIM - !(input) number of dimensions
INTEGER*4 DIMLO(*) - !(input) low end for each dimension
INTEGER*4 DIMHI(*) - !(input) high end for each dimension
INTEGER*4 WHERE - !(input) 1-dimensional location
LOGICAL*4 OK - !(output) within boundries
CHARACTER*(*)  STR - !(output) conventional string(i,...,k)
```

```
*
*     Given a variable's FORTRAN declared dimensionality
*     and its 1D index, returns a string str in the
*     conventional form - KBB
*
*     For example, if V(10,3,-1:1) -> Ndim=3
*                     dimLO(1)=1,dimLO(2)=1,dimLO(3)=-1
*                     dimHI(1)=10,dimHI(2)=3,dimHI(3)=1
*     and where= 5 --> str="(5,1,-1)"
*
```

called by: WHEREAT

calls: FLUSH

[*SRC/z_conj.f*]
```
COMPLEX*16 FUNCTION Z_CONJ( Z )
```

```
COMPLEX*16 Z -
```

```
*
*     returns the complex conjugate of Z
*
```

calls: Z_IMAG , Z_REAL

[*SRC/z_exp.f*]
```
COMPLEX*16 FUNCTION Z_EXP( Z )

      COMPLEX*16 Z -

      *
      *     returns the exponential of complex Z
      *
```

called by: MATBBU_GO

calls: Z_IMAG , Z_REAL

[*SRC/z_imag.f*]
```
REAL*8 FUNCTION Z_IMAG( Z )

      COMPLEX*16 Z -

      *
      *     returns the imaginary part of Z
      *
```

called by: MATBBU_GO , Z_CONJ , Z_EXP , ZUMCHECK

[*SRC/z_real.f*]
```
REAL*8 FUNCTION Z_REAL( Z )

      COMPLEX*16 Z -

      *
      *     returns the real part of Z
      *
```

called by: MATBBU_GO , Z_CONJ , Z_EXP , ZUMCHECK

[*SRC/zumcheck.f*]
```
SUBROUTINE ZUMCHECK( WHERE, WHAT, ARRAY, SIZE, SUM, DUMP )
```

CHARACTER*(*) *WHERE* - !(input) name of calling routine
CHARACTER*(*) *WHAT* - !(input) comment (may be declaration)
COMPLEX*16 *ARRAY(*)* - !(input) array
INTEGER*8 *SIZE* - !(input) size of array
REAL*8 *SUM* - !(output) sum of absolute values in array
LOGICAL*4 *DUMP* -

```
      *
      *     simply sums up the absolute values of
      *     an array of size and returns the sum -
      *     just for checking for errors
      *
```

called by: MATBBU_GO , REPACK

calls: Z_IMAG , Z_REAL , WHEREAT , FLUSH

[*cheapseek_main.c*]
```
int main( int argc, char *argv[] )
```

```
int argc -
char *argv[] -
```

Kevin's WRAPper
http://wims3.larc.nasa.gov/~beard/KWRAP/

calls: CHEAPSEEK

[*matrixbbuk_main.c*]
```
int main( int argc, char *argv[] )
```

```
int argc -
char *argv[] -
```

Kevin's WRAPper
http://wims3.larc.nasa.gov/~beard/KWRAP/

calls: MATRIXBBUK

[*SRC/ranf_.c*]
```
double ranf_( )
```

returns a random number 0<x<1

called by: GAUSSIAN_CUT , HOM_SHIFT

[*SRC/ranf__.c*]
```
double ranf__( )
```

returns a random number 0<x<1

[*SRC/ranset_.c*]
```
void ranset_( long *seed )
```

```
long *seed -
```
sets the random number seed

called by: MATRIXBBUK

[*SRC/ranset__.c*]
```
void ranset__( long *seed )
```

```
long *seed -
```
sets the random number seed

Appendix 6. Input File

A typical input file for the Jlab 10kW FEL. The recirculation units are those of DIMAD. Other examples are included in the distribution. *Note that the spacing is significant!*

```
1TITLE 10kW IR FEL 145 MeV, April 2002 3CMs [5,7,5]cells
 DATA
 APRTR    100000.   2.0
 REF      0.       600.0      355.00   700.00    500.0     0.0
 BEAM     10.0      2994.0     40.0     0.0       1.0       0.0
 XPRNT    2.0       203.0     1.0
 YPRNT    2.0       203.0     1.0
#CMPNT      200.0   0.0       0.0       0.0       0.0       0.0
>DRIFT  1.100.0     0.0
1DRIFT  1. 63.41    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.82    17000.    1812.510  90.0
1CAVITY    13.77    7000.     1816.220  .0
1CAVITY    22.32    120000.   1882.830  90.0
1CAVITY    22.24    8000.     1885.840  .0
1CAVITY    48.42    5000.     1963.530  90.0
1CAVITY    48.27    2600.     1966.660  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 25.      0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.64    10000.    1825.020  90.0
1CAVITY    13.60    5100.     1827.260  .0
1CAVITY    22.04    83000.    1894.660  90.0
1CAVITY    22.01    7300.     1895.980  .0
1CAVITY    47.94    2300.     1973.270  90.0
1CAVITY    47.76    1700.     1976.980  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 66.06    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.68    24000.    1821.940  90.0
1CAVITY    13.66    3100.     1823.560  .0
1CAVITY    22.12    210000.   1891.120  90.0
1CAVITY    22.08    4300.     1892.920  .0
1CAVITY    48.17    3400.     1968.680  90.0
1CAVITY    47.99    1400.     1972.210  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 25.      0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.74    45000.    1818.320  90.0
1CAVITY    13.70    2500.     1820.930  .0
1CAVITY    22.21    400000.   1887.240  90.0
1CAVITY    22.15    4300.     1889.910  .0
1CAVITY    48.29    3000.     1966.180  90.0
1CAVITY    48.09    1600.     1970.160  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 66.06    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.59    10000.    1828.330  90.0
```

```
1CAVITY    13.59    2500.     1828.120  .0
1CAVITY    22.02    160000.   1895.390  90.0
1CAVITY    21.97    3500.     1897.480  .0
1CAVITY    48.04    2600.     1971.200  90.0
1CAVITY    47.84    1800.     1975.430  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 25.      0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.70    12000.    1820.640  90.0
1CAVITY    13.66    2000.     1823.190  .0
1CAVITY    22.14    200000.   1890.370  90.0
1CAVITY    22.08    3900.     1892.960  .0
1CAVITY    48.20    3500.     1968.080  90.0
1CAVITY    48.01    1500.     1971.830  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 66.06    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.81    38000.    1813.290  90.0
1CAVITY    13.75    3700.     1817.350  .0
1CAVITY    22.29    39000.    1884.000  90.0
1CAVITY    22.18    9000.     1888.460  .0
1CAVITY    48.10    800.      1970.000  90.0
1CAVITY    47.94    1000.     1973.360  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 25.      0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.64    12000.    1825.000  90.0
1CAVITY    13.59    4700.     1828.080  .0
1CAVITY    22.04    36300.    1894.570  90.0
1CAVITY    21.98    12300.    1897.200  .0
1CAVITY    47.79    1000.     1976.500  90.0
1CAVITY    47.69    2100.     1978.460  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 63.41    0.0
1DRIFT  1. 85.1     0.0
1LENS   1.-3.597509 15.0
1DRIFT  1. 37.4     0.0
1LENS   1. 6.755323 15.0
1DRIFT  1. 37.4     0.0
1LENS   1.-3.597509 15.0
1DRIFT  1. 38.38    0.0
1DRIFT  1. 51.64    0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.890  90.0
1CAVITY    29.29    7600000.  2094.310  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.530  90.0
1CAVITY    29.29    7600000.  2094.220  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.750  90.0
1CAVITY    29.29    7600000.  2094.620  .0
1DRIFT  1. 35.      3.4375
```

```
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.260  90.0
1CAVITY    29.29    7600000.  2094.380  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.570  90.0
1CAVITY    29.29    7600000.  2094.990  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.100  90.0
1CAVITY    29.29    7600000.  2094.610  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.040  90.0
1CAVITY    29.29    7600000.  2094.060  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 30.      0.0
1DRIFT  1. 35.      3.4375
1CAVITY    29.29    7600000.  2094.410  90.0
1CAVITY    29.29    7600000.  2094.080  .0
1DRIFT  1. 35.      3.4375
1DRIFT  1. 57.02    0.0
2DRIFT  1. 38.38    0.0
2LENS   1. 1.294342 15.0
2DRIFT  1. 37.4     0.0
2LENS   1.-2.550615 15.0
2DRIFT  1. 37.4     0.0
2LENS   1. 1.294342 15.0
2DRIFT  1. 85.1     0.0
1DRIFT  1. 63.41    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.65    20700.    1824.210  90.0
1CAVITY    13.65    20700.    1824.560  .0
1CAVITY    22.06    145000.   1893.540  90.0
1CAVITY    22.06    145000.   1893.780  .0
1CAVITY    48.05    2800.     1971.030  90.0
1CAVITY    48.05    2800.     1971.230  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 25.      0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.63    17000.    1825.590  90.0
1CAVITY    13.63    17000.    1825.230  .0
1CAVITY    22.04    4300.     1894.690  90.0
1CAVITY    22.04    4300.     1894.040  .0
1CAVITY    48.02    2500.     1971.680  90.0
1CAVITY    48.02    2500.     1971.210  .0
1DRIFT  1. 25.      2.5
1DRIFT  1. 66.06    0.0
1DRIFT  1. 25.      2.5
1CAVITY    13.72    18400.    1819.610  90.0
1CAVITY    13.72    18400.    1819.340  .0
1CAVITY    22.14    106000.   1890.270  90.0
```

```
1CAVITY   22.14   106000.   1890.530  .0
1CAVITY   48.01   3000.     1971.880  90.0
1CAVITY   48.01   3000.     1971.520  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 25.     0.0
1DRIFT  1. 25.     2.5
1CAVITY   13.76   31000.    1816.550  90.0
1CAVITY   13.76   31000.    1816.430  .0
1CAVITY   22.22   189000.   1886.840  90.0
1CAVITY   22.22   189000.   1886.900  .0
1CAVITY   48.26   2600.     1966.820  90.0
1CAVITY   48.26   2600.     1966.210  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 66.06   0.0
1DRIFT  1. 25.     2.5
1CAVITY   13.65   15800.    1823.820  90.0
1CAVITY   13.65   15800.    1823.640  .0
1CAVITY   22.07   16100.    1893.390  90.0
1CAVITY   22.07   16100.    1893.900  .0
1CAVITY   48.03   4100.     1971.400  90.0
1CAVITY   48.03   4100.     1971.210  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 25.     0.0
1DRIFT  1. 25.     2.5
1CAVITY   13.62   14200.    1826.140  90.0
1CAVITY   13.62   14200.    1826.590  .0
1CAVITY   22.07   11300.    1893.350  90.0
1CAVITY   22.07   11300.    1893.920  .0
1CAVITY   48.04   3900.     1971.360  90.0
1CAVITY   48.04   3900.     1971.020  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 66.06   0.0
1DRIFT  1. 25.     2.5
1CAVITY   13.63   30000.    1825.290  90.0
1CAVITY   13.63   30000.    1825.980  .0
1CAVITY   22.06   56000.    1893.690  90.0
1CAVITY   22.06   56000.    1893.540  .0
1CAVITY   48.07   2000.     1970.760  90.0
1CAVITY   48.07   2000.     1970.030  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 25.     0.0
1DRIFT  1. 25.     2.5
1CAVITY   13.73   12300.    1818.510  90.0
1CAVITY   13.73   12300.    1818.630  .0
1CAVITY   22.20   69000.    1887.830  90.0
1CAVITY   22.20   69000.    1887.450  .0
1CAVITY   48.22   5000.     1967.640  90.0
1CAVITY   48.22   5000.     1967.980  .0
1DRIFT  1. 25.     2.5
1DRIFT  1. 63.41   0.0
2DRIFT  1.100.0    0.0
$RECIRC 1.
$CALC     0.
0.1,0.,0,0.0,0.,0
0.1,0.,0,0.0,0.,0
1093
```

0.893024 -18.6171 0.0 0.0
-0.00198 1.161135 0.0 0.0
0.0 0.0 -1.08916 18.46925
0.0 0.0 .024832 -1.33922
0.0,0.,0,0.,0.,0
0.0,0.,0,0.,0.,0