MIDDLE LAYER QUICK REFERENCE

Gregory J. Portmann

This is meant to be a quick reference guide. It is not a complete explanation of the details of each function. Most of the flexibility of each function is not discussed. It is designed so that one can cut/paste from this document directly into the Matlab command window to do common tasks quickly. Most of these commands (and more) are accessible from *plotfamily*.

1. Machine State

a. Units: Hardware or Physics

```
switch2hw
switch2physics
```

Each family can be put in either hardware or physics units but it is not recommended to do this. Spear3 will operate in hardware units for all applications. One can override the units on the input to many functions.

b. Mode: Online or Simulate

```
switch2online
switch2sim
```

Each family can be put in either online or simulate mode but it is not recommended to do this. The default mode for Spear3 is online. One can override the mode on the input to many functions.

2. Common flags

a. 'Hardware' or 'Physics'
b. 'Online', 'Simulator', 'Model', or 'Manual'
c. 'Struct' or 'Numeric'
d. 'Archive' or 'NoArchive'
e. 'Display' or 'NoDisplay'

Flag are not case sensitive (familynames are case sensitive). Note: in the future Matlab on the PC will be case sensitive on filename, just like Unix.

3. Get/Save Orbits

a. Get an orbit

```
x = getx;       % or  x = getam('BPMx');
y = gety;       % or  y = getam('BPMy');
Note 1:  Use an optional 'Struct' input to return a data structure
Note 2:  To graphically view orbits use plotorbit or plotfamily.
```

b. Save orbit data to the default directory for orbits

```
getbpm('Archive');
getbpm('Archive', FileName);       % to save data to FileName
```

c. Get the default Golden and Offset orbits

```
Xgolden = getgolden('BPMx');       % Horizontal golden orbit
Ygolden = getgolden('BPMy');       % Vertical golden orbit

Xoffset = getoffset('BPMx');       % Horizontal offset orbit
Yoffset = getoffset('BPMy');       % Vertical offset orbit
```

d. Save the present orbit as the default golden orbit
How this is done depends on the accelerators. Ultimately the AO.BPMx.Golden field needs to be set properly. At Spear, the golden orbit is saved in the spear3physdata file and loaded into the AO by spear3init. Hence, at Spear to change to a new golden orbit run the following code then run spear3init..

```
setphysdata('BPMx', 'Golden', BPMxGolden, DeviceList);
setphysdata('BPMy', 'Golden', BPMyGolden, DeviceList);
```

At ALS, the golden and offset orbits are changed by directly editing the setoperationalmode file. CLS make the appropriate changes to the AO in clsinit.

Keeping track of past golden and offset orbits can be cumbersome. I usually save them by date on the day the new orbit is made. The functions savegoldenorbit and saveoffsetorbit are a simple way to do it.

e. Find and save BPM data (used for standard deviations, etc)

```
monbpm;

% To make this file the default BPM sigma file
copybpmsigmafile;
```

f. Converting between raw control system data and calibrated data

```
RealData = raw2real(Family, RawData, DeviceList)
RawData = real2raw(Family, RealData, DeviceList)

To get the actual calibration numbers:
Gain = getgain('BPMx');
Offset = getoffset('BPMx');
```

4. Get/Set magnet settings

a. Get all the horizontal corrector setpoint and monitor values

```
SP = getsp('HCM');
AM = getam('HCM');
```

b. Set HCM(2,1)=1 and HCM(2,4)=12

```
setsp('HCM', [1;12], [2 1; 2 4]);
```

c. Set all horizontal corrector to zero
```
setsp('HCM', 0);
```

Note 1: *getfamilylist* returns the names of all families.
Note 2: To graphically view magnets setpoints and monitors use *plotfamily*.

5. Global orbit correction

(to be written, but use setorbit* or *setorbitdefault)

6. Local orbit correction

(to be written, but use setorbitbump)

7. Retrieve an orbit response matrix

```
% Default response matrix
R = getbpmresp;

% Retreive response matrix from file
R = getbpmresp(Filename);
R = getbpmresp('');                    % For a browser

% Model response matrix
R = measbpmresp('Model');
```

8. Measure an orbit response matrix (and set as the default)

measbpmresp measures the orbit response matrix. The units for the orbit response matrix at Spear is always [mm/amp] (hardware units). See *help measbpmresp* for more details.

```
% Defaults for online response matrices
R = measbpmresp;
%   is the same as,
R = measbpmresp('BPMx', 'BPMy', 'HCM', 'VCM', 'Online', 'Bipolar',
                'Numeric', 'Archive');

% Defaults for model response matrices
R = measbpmresp('Model');
%   is the same as,
R = measbpmresp('BPMx', 'BPMy', 'HCM', 'VCM', 'Model', Bipolar',
                'Numeric', 'NoArchive', 'FixedPathLength', 'Full');

% Get a response matrix for 3 BPMs and 2 Correctors w/o saving data.
% (the data is saved to disk by default if 'Online', use the 'NoArchive'
% flag suppress this).
```

```
R1 = measbpmresp('BPMx', [1 1;2 2;6 3], 'BPMy', [1 1;2 2;6 3], 'HCM', [1 1; 1 3],
'VCM', [1 1; 1 2], 'NoArchive');

% Get a the same data from the model
% The default for the model is not to save data to disk.
R2 = measbpmresp('BPMx', [1 1;2 2;6 3], 'BPMy', [1 1;2 2;6 3], 'HCM', [1 1; 1 3],
'VCM', [1 1; 1 2], 'Model');

% Compare a column (ie, a corrector magnet response)
plot(R2(:,1) - R1(:,1));

% Get a the same data from the model w/ a FixedMomentum, Linear model
R2 = measbpmresp('BPMx', 'BPMy', 'HCM', [1 1; 1 3], 'VCM', [1 1; 1 2],
                 'FixedMomentum', 'Linear', 'Model');
```

The default filename and directory is,
<DataRoot>\Response\BPM\< BPMRespFile><Date><Time>.mat

To make this file the default operational file, copy it to:
<OpsData>\< BPMRespFile>.mat

It is best to use the following function to do the copy (in plotfamily).
copybpmrespfile;

9. Get/Set/Step RF frequency

(see help getrf, setrf, steprf in Matlab)

10. Get/Save/Plot Dispersion

The dispersion is measured with the *measdisp* function. The orbit shift verses RF or momentum will be plotted. The default units for dispersion are change in orbit per change in RF frequency [mm/MHz] (hardware units at Spear). However, one can select units of change in orbit per change in momentum [meters/(dp/p)] (physics units) by adding 'Physics' to the input line. A structure output is selected by adding 'Struct' to the input line. The fields of the structure are similar to a response matrix structure (delta orbit for a delta change in RF frequency). Use *plotdisp* to plot a past measurement. See help *measdisp* for more details.

```
% Measure the dispersion (vector output)
[Dx, Dy] = measdisp;

% Measure the dispersion (structure output)
[Dx, Dy] = measdisp('Struct');

% Measure the dispersion in physics units [1/(dp/p)]
[Dx, Dy] = measdisp ('Struct', 'Physics');
```

```
% Measure the dispersion and archive (output is optional)
measdisp ('Archive');

% Measure and plot the dispersion
% No outputs or 'Display' on the input line will automatically plot
[Dx, Dy] = measdisp('Display');
measdisp;

% Plot using plotdisp function
 [Dx, Dy] = measdisp('Struct');
plotdisp(Dx, Dy);
plotdisp('');                          % Plot data from a file

% Model dispersion (override the Mode to Simulate)
[Dx, Dy] = measdisp('Simulator');

% Model dispersion (calls AT directly)
[Dx, Dy] = measdisp('Model');          % calls modeldisp
```

11. Tune

Get/Set/Step tunes

```
% Measure the tune and return a 2x1 vector
Tune = gettune;

% Measure the tune and return a structure
Tune = gettune('Struct');
```

Measure a tune response matrix (and set as the default)

meastuneresp measures the tune response matrix. The units for the tune response matrix at Spear is [Fractional Tune/Amp] (hardware units). See help *meastuneresp* for more details.

The default filename and directory is,
<DataRoot>\Response\Tune\< TuneRespFile><Date><Time>.mat

To make this file the default operational file, copy it to:
<OpsData>\< TuneRespFile>.mat

It is best to use the following function to do the copy (in plotfamily).

```
copytunerespfile;
```

Step the tune and get the tune response matrix

To change the tune by [-.05; .05], simple use the *steptune* command.

```
steptune([-.05; .05]);
```

The *steptune* function can easily be done manually:

```
% Measure the tune (just to check the result)
Tune1 = gettune;

% Get the chromaticity response matrix for SF and SD
m = gettuneresp;                % Default (used by settune, steptune)
m = gettuneresp('Model');       % Model

% Compute the delta QF and QD and apply the correction
DeltaAmps = inv(m) * [-.05; .05];
setsp({'QF', 'QD'}, {getsp('QF')+DeltaAmps(1), getsp('QD')+DeltaAmps(2)});

% Measure the chromaticity and check result
Tune2 = gettune;
DeltaTune = Tune2 - Tune1
```

12. Chromaticity

Measure a chromaticity

Measure the chromaticity using the *measchro* command. The tune shift verses RF or momentum will be plotted. Chromaticity being the linear term in the curve fit. Verify that the curve fit looks "accurate." Note: when the chromaticity is close to zero small tune errors can produce an inaccurate chromaticity measurement. The default units for chromaticity are change in tune per change in RF frequency [1/MHz] (hardware units). However, one can select units of change in tune per change in momentum [1/(dp/p)] (physics units) by adding 'Physics' to the input line. The default output is the 2x1 chromaticity vector, however, a structure output is selected by adding 'Struct' to the input line. The fields of the structure are similar to a response matrix structure. *measchro* automatically plots the results Use *plotchro* to plot a past measurement. See help *measchro* for more details.

```
% Measure the chromaticity and return a 2x1 vector (units [1/MHz])
Chro = measchro;

% Measure the chromaticity and return a structure
ChroStruct = measchro('Struct');

% Measure the chromaticity is physics units [1/(dp/p)]
ChroStruct = measchro('Struct', 'Physics');

% Measure the chromaticity and archive to the appropriate directory
measchro('Archive');                % Output is optional
```

```
% Plot the chromaticity measurement
C = measchro('Struct');
plotchro(C);
plotchro('');                          % Plot data from a file
```

Measure a chromaticity response matrix (and set as the default)
measchroresp measures the chromacity response matrix for the default sextupole families. The units for the chromacity response matrix at Spear is [1/MHz] (hardware units). See help *measchroresp* for more details.

The default filename and directory is,
<DataRoot>\Response\Chromaticity\< GoldenChroResp><Date><Time>.mat

To make this file the default operational file, copy it to:
<OpsData>\< GoldenChroResp>.mat

It is best to use the following function to do the copy (in plotfamily).
```
copychrorespfile;
```

Step the chromaticity and get the chromaticity response matrix
To change the chromaticity by [-.25; .25] [1/MHz], simple use the *stepchro* command.
```
stepchro([-.25; .25]);
```

If the chromaticity was measured in physics units and the response matrix was measured in hardware units then convert it before passing it to *stepchro*. For RF=476.3 and MCF=.0011,
```
stepchro([.1346; -.1346] / -RF / MCF);
```
However, it is best to stick with one set of units for all measurements.

The *stepchro* function can easily be done manually:

```
% Measure the chromaticity (just to check the result)
figure(1);
Chro1 = measchro;

% Get the chromaticity response matrix for SF and SD
m = getchroresp;               % Default (use by stepchro)
m = getchroresp('Model');      % Model

% Compute the delta SF and SD and apply the correction
DeltaAmps = inv(m) * [-.25; .25];
setsp({'SF', 'SD'}, {getsp('SF')+DeltaAmps(1), getsp('SD')+DeltaAmps(2)});

% Measure the chromaticity and check result
```

figure(2);
Chro2 = measchro;
DeltaCrho = Chro2 - Chro1

13. Save/restore

- Archive a lattice
- Make the default magnet lattice for operations

The default filename and directory for archiving is,
<DataRoot>\< ConfigData>\ <LatticeFile><Date><Time>.mat

To make this file the default operational file, copy it to:
<OpsData>\< LatticeFile>.mat

It is best to use the following function to do the copy (in plotfamily).
copymachineconfigfile;

14. Prepare a LOCO input file

LOCO requires an orbit response matrix, dispersion, and the standard deviations of the BPM difference orbits. Measure a new orbit response matrix with *measbpmresp*, dispersion with *measdisp('Archive'),* and new BPM standard deviations *monbpm*. Or use *measlocodata* to do it all at once. Combine the output from these functions with *buildlocoinput*. The accelerator specific part of the function in broken out as *buildlocofitparameters*.

15. Finding quadrupole centers

To find the offset between the orbit at the BPM and quadrupole:

```
% One quadrupole center
quadcenter;          % Data is automatically saved to a subdirectory
                     % of DataRoot>\QMS named by date and time

% Find all quadrupole centers
quadcenterall;       % Data saved to <DataRoot>\QMS with filename
                     % determined by date and time
```

% Save the new centers to the default offset orbit
How this is done depends on the accelerators. Ultimately the AO.BPMx.Golden field needs to be set properly. quadplotall is often used to plot the offset information after the whole accelerator is measured. Use quadplot to check any individual quadrupole center. At Spear the offset orbit is saved in the spear3physdata file and loaded into the AO by spear3init. Hence, at Spear to

change to a new offset orbit put all the quadrupole center files in one directory and run:

```
[x, y] = quadplotall([]);
setphysdata('BPMx', 'Offset', x(:,3), x(:,1:2));
setphysdata('BPMy', 'Offset', y(:,3), y(:,1:2));
```

Then rerun spear3init.

At the ALS changes to the golden and offset orbits are made in setoperationalmode. At the CLS the golden and offset are stored in a file and loaded from clsinit.

16. Model only functions

A number of function have been written only to get or set model parameters.

```
% Model dispersion function
modeldisp;                    % Plots with units mm/MHz
modeldisp('BPMx', 'BPMy');    % Plots at 'BPMx', 'BPMy' families [mm/MHz]
modeldisp('Physics');         % Plots with units meters/(dp/p)
[Dx, Dy] = modeldisp;         % Returns Dx, Dy with units mm/MHz

% Model beta function
modeltwiss('beta');                    % Plot beta
modeltwiss('beta', 'BPMx');            % Plot beta at the BPMx family
[Betax, Betay] = modeltwiss('beta', 'BPMx');   % Returns beta at BPMx

% Model closed orbit
[x, y] = modeltwiss('ClosedOrbit');    % Closed orbit at all AT elements
[x, y] = modeltwiss('x');              % Closed orbit at all AT elements
y = modeltwiss('y', 'BPMy');           % Vertical orbit at BPMy family

% RF Cavity and Radiation
setcavity on;
setcavity off;

setradiation on;      % It is interesting to check the orbit changes
setradiation off;     % due to energy changes due to radiation
```

17. Example Script (Orbit Correction)

```
% Introduce an orbit error (10 microradian)
setsp('HCM', 10e-6*randn(size(family2dev('HCM'),1),1), 'Physics');

% Get the horizontal response matrix
% Use the Model
Sx = measbpmresp('Struct', 'Model');
```

```
Sx = Sx(1,1).Data;
% Or use the Default
Sx = getrespmat('BPMx','HCM');

% Gets all horizontal BPMs (vector)
X = getx;

% Computes the SVD of the response matrix
% Use singular vectors 1 thru 60
Ivec = 1:60;
[U, S, V] = svd(Sx);

% Find the corrector changes (vector)
DeltaAmps = -V(:,Ivec)*((U(:,Ivec)*S(Ivec,Ivec))\X) ;

% Changes the current in all horizontal corrector magnets
stepsp('HCM', DeltaAmps);

% Plot new orbit
plot(getspos('BPMx'), X, 'b', getspos('BPMx'), getx, 'r');
xlabel('BPM Position [meters]');
legend('Before', 'After');
```

18. Error Handling

In general, error handling in Matlab is done with try;catch statements. However, some middle layer users prefer Setpoint – Monitor errors to behave differently. Hence a middle layer family variable can be set to change what happen on a SP-AM error.

```
AD.ErrorWarningLevel =  0 SP-AM errors are a Matlab errors {Default}
                       -1 SP-AM errors are a Matlab warnings
                       -2 SP-AM errors prompt a dialog box
                       -3 SP-AM errors are ignored
```

To get the warning level:

```
ErrorWarningLevel = getfamilydata('ErrorWarningLevel');
```

To set the warning level:

```
setfamilydata(ErrorWarningLevel , 'ErrorWarningLevel');
```

Note that many functions have try;catch statements to catch errors. Only when the ErrorWarningLevel is 0 will these statement be used to catch SP-AM errors.

Changing the default warning level for an accelerator is usually done in the setoperationalmode function.

19. Setpoint – Monitor Tolerance

Determining whether or not a magnet (or other devices) are ramping is often done by comparing the setpoint to the monitor value and seeing if it's within a tolerance band. Some

To get the tolerance:

```
Tol = getfamilydata(Family, 'Setpoint', 'Tolerance', DeviceList);
  or
Tol = family2tol(Family, DeviceList);
```

To set the tolerance:

```
setfamilydata(Tol, Family, 'Setpoint', 'Tolerance', DeviceList);
```

Permanently changing the tolerance field is done in the initialization file.