MIDDLE LAYER QUICK REFERENCE

Gregory J. Portmann

This is meant to be a quick reference guide. It is not a complete explaination of the details of each function. Most of the flexibility of each function is not discused. It is designed so that one can cut/paste from this document directly into the Matlab command window to do common tasks quickly.

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. Units is not case sensitive.

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 Spear3 will be online. One can override the mode on the input to many functions. Mode is not case sensitive.

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'

3. Launch Pad GUI

srsetup is "launch pad" GUI for running commonly use functions.

4. 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

```
getx('archive');       % or  getx archive
gety('archive');       % or  gety archive
```

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

```
BPMxGolden = getx('struct');
BPMyGolden = getx('struct');
setphysdata('BPMx', Golden, BPMxGolden);
setphysdata('BPMy', Golden, BPMyGolden);
```

e. Save the present orbit as the default offset orbit

```
BPMOffset = {getx('struct'), gety('struct')};
FileName = getfamilydata('OpsData', 'BPMOffsetFile');
DirName  = getfamilydata('Directory', 'OpsData');
save([DirName FileName], ' BPMOffset ');
```

Note: Usually the offset orbit is determined from the quadrupole centering program. See the quadrupole centering section below.

f. Find and save the default BPM standard deviations

```
[BPMxSigma, BPMySigma] = measbpmsigma('Struct');
setphysdata('BPMx', 'Sigma', BPMxSigma);
setphysdata('BPMy', 'Sigma', BPMySigma);
```

g. Converting between raw control system data and calibrated data

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

Uses getphysdata to get the actual calibration numbers
Gain = getphysdata('BPMx', 'Gain');
Offset = getphysdata('BPMx', 'Offset'); % or, Offset = getoffset('BPMx');
```

5. 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*.

6. Global orbit correction

(to be written)

7. Local orbit correction

(to be written)

8. Get an orbit response matrix

(to be written)

9. 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 data
R = measbpmresp;
%    is the same as,
R = measbpmresp('BPMx', 'BPMy', 'HCM', 'VCM', [ ], [ ], 'Online', 'Bipolar',
                              'Numeric', 'Archive');

% Defaults when using the model:
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 usually saved to disk.  The 'NoArchive' flag stops 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\BPMRespMat<Date><Time>.mat

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

10. Prepare a LOCO input file

LOCO requires an orbit response matrix and the standard deviations of the BPM difference orbits. Measure a new orbit response matrix with *measbpmres* and new BPM standard deviations *measbpmsigma*. Combine the output from these functions with *makelocoinput*. ***(this function is still under development)***

11. Get/Set/Step RF frequency

(to be written)

12. 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 at Spear) by adding 'Physics' or 'Zeta' 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);
```

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

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

13. 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\TuneRespMat<Date><Time>.mat

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

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;

% Compute the delta SF and SD 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
```

14. 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' or 'Zeta' 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
```

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\ChroRespMat<Date><Time>.mat

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

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;

% 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
```

15. Save/restore

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

16. Find a quadruple center and update the offset orbit

(to be written)

17. Get/Set an insertion device position

(to be written)

18. 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
```

19. Example Script (Orbit Correction)

```
% Introduce an orbit error
setsp('HCM',rand(72,1));

% Get the proper response matrix
%Sx = getrespmat('BPMx','HCM');
Sx = measbpmresp('BPMx', 'BPMy','HCM', 'VCM', 'Struct', 'Model');
Sx = Sx(1,1).Data;

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

% Computes the SVD of the response matrix, Sx(96x94)
% Use singular vectors 1 thru 24
Ivec = 1:24;
[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]');
```