# Error setting and correction using Accelerator Toolbox 2.0

ESRF, Grenoble

9th November 2016



## Contents

**This is a DRAFT Document.**
**Functions names, description and expected output may change!**

# 1 Errors

A not complete and not exhaustive, often probably not correct implementation of errors in AT is presented in this document. work is in progress to complete this document. Examples of code are given. Most functions need to have as first input the AT lattice structure (THERING in the globally diffused, global variable approach to AT). This backward compatibility feature is being set up.

## 1.1 Nomenclature and conventions

WORK IN PROGRESS
x,y,z, rotations, bpm errors

### Magnets modeled by several elements (sliced)

magnum, or errors only at beginning and end

## 1.2 Alignment errors

To model magnets misalignment in AT the particle trajectory is displaced to enter off axis in the magnet, and brought back to the initial after the magnet.

```
/*  misalignment at entrance */
if (T1) ATaddvv(r6, T1);
if (R1) ATmultmv(r6, R1);
/* magnet */
...
/* Check physical apertures */
...
/* Misalignment at exit */
if (R2) ATmultmv(r6, R2);
if (T2) ATaddvv(r6, T2);
```

### Generic alignment errors

In the most general way, AT provides T1,T2 and R1,R2 field in most *PassMethods* to describe translation or rotations of the 6D coordinates. Those fields are set in higher level functions such as *atsetshift* and *atsettilt*. In AT, T2 is the net misalignment of the magnet and T1=-T2. In AT, R2 is the net rotation of the magnet and R1=inv(R2).

### Generic field errors

Concerning magnetic field errors, the structures *PolynomB* and *PolynomA* provide full access to all magnetic components. The field *maxorder* describes up to which multipole AT is going to interpret *PolynomB* and *PolynomA*.

### Apertures with alignment errors

Physical apertures do not move with the displaced elements. The apertures are set about zero and if there is a large misalignment then the particles will be in fact closer to the aperture. Drift space and markers can be displaced. In the case of drift spaces, the displacement can be interpreted as the beam moving off axis in the vacuum chamber.

In the following we describe in more detail several of the functions for errors setting provided in AT, together with some additional functions developed for more advanced errors setting.

### Horizontal and vertical

Below an example of usage of *atsetshift* to displace a group of magnets (Quadrupoles, in the example), with random offsets of 1 micro meter standard deviation. Figure 1 shows an example of the effect of this function on the AT lattice: the introduction of alignment errors has generated as expected closed orbit and dispersion distortions.

```
% get indexes
indq=find(atgetcells(ring,'Class','Quadrupole'));

% define alignemnt errors
dx=1e-6*randn(size(indq)); % random errors of 1um
dy=1e-6*randn(size(indq));

% set errors
ringerr=atsetshift(ring,indq,dx,dy);
```

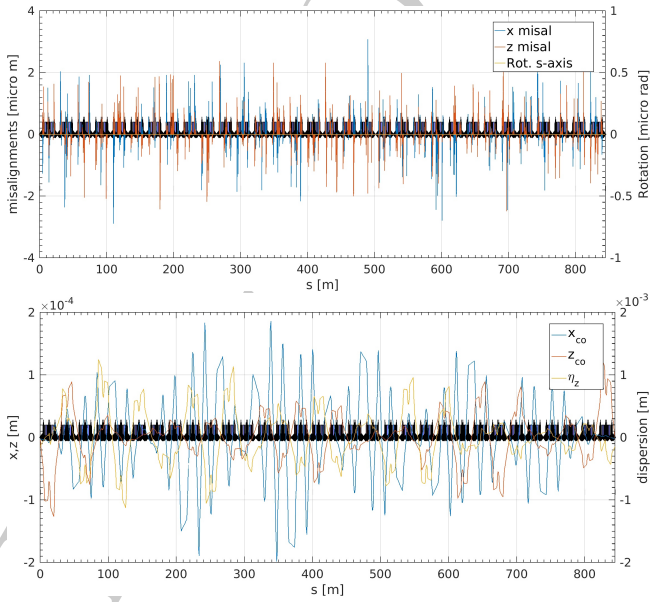



Figure 1: top: random errors from a Gaussian distribution with $\sigma = 1\,\mu\text{m}$, bottom: closed orbit and vertical dispersion with errors

### Rotation about the s-axis

Below an example of rotation of quadrupole magnets using the function *atsettilt*. Figure 2 presents the dispersion distortion obtained in this case.

```
% get indexes
indq=find(atgetcells(ring,'Class','Quadrupole'));

% define s-axis rotation errors
dt=1e-6*randn(size(indq)); % random errors of 1um

% set errors
ringerr=atsettilt(ring,indq,dt);
```

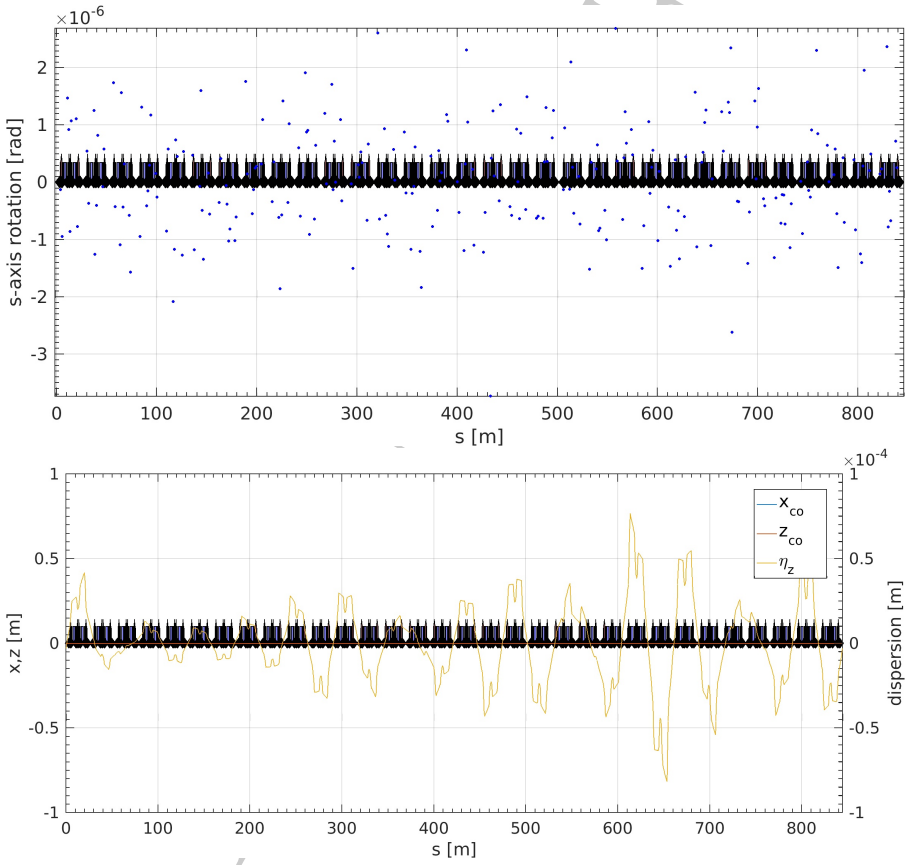


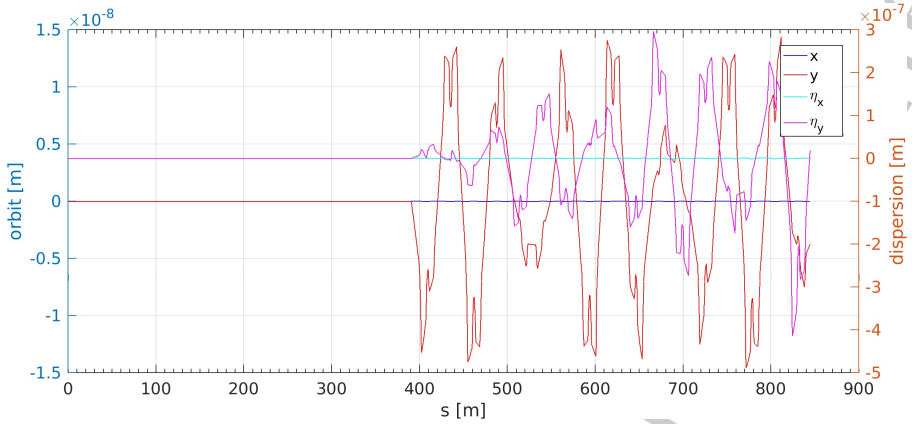

Figure 2: top: random s-axis rotation errors from a Gaussian distribution with $\sigma = 1\,\mu\text{rad}$,
bottom: closed orbit and vertical dispersion with s-axis rotation errors

**Rotation of a dipole about the s-axis**

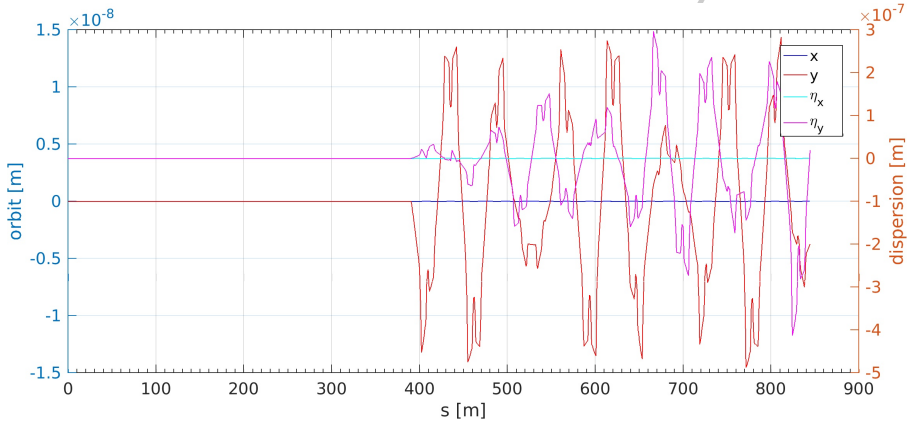
Dipoles change the reference frame but their field is usually ignored. To evidence the effect of dipole rotation and field errors those must be expressed in terms of *PolynomA* and *PolynomB*. The function *atsettiltdipole* implements this feature, and sets the field PolynomB and PolynomA in a dipole to represent the distortion introduced by the rotated bending magnet. In the figures below the effect of the rotation of a dipole are shown in three cases: rotation of a straight multipole using *atsettilt* (Fig.3a), rotation of a dipole using *atsettilt* (Fig.6b), rotation of a dipole using **??** (Fig.3b).

There is no difference between 3a and 3b, while the tilt implemented rotating the reference system does not show the expected orbit distortion.

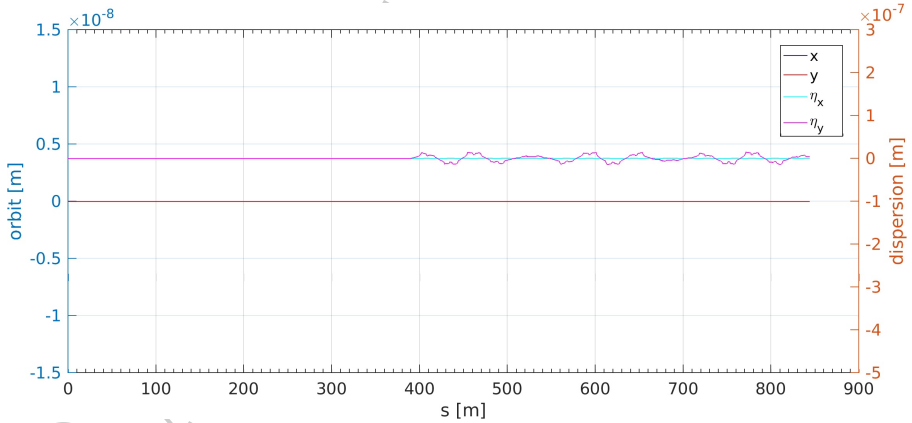
The same considerations are ture for the rotation of a combined function dipole-quadrupole in figure 5.




(a) Multipole kick rotated using *atsettilt*




(b) Dipole rotated using *atsettiltdipole*




(c) Dipole rotated using *atsettilt*

Figure 3: Orbit and dispersion variation when tilting by $100\,\mu rad$ a $10\mu rad$ horizontal kick at about $390\,m$.

### Rotation about x and y axis

These two rotation are implemented using the fields *T1* and *T2*.

### Longitudinal alignment errors

Longitudinal alignment errors require to change the length of adjacent drift spaces. For Dipoles also a change of trajectory needs to be considered (see Fig. 4). the length of the lattice changes when displacing longitudinally dipoles.

```
% get indexes
inddip=find(atgetcells(ring,'Class','Dipole'));

% define longitudinal displacement
DS=-1e-2;

% set fixed -1cm displacement at first 2 dipoles in the lattice
ringerr=atset_s_shift(ring,inddip([1,2]),DS);
```

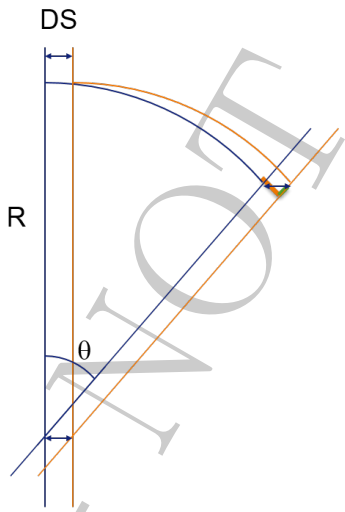
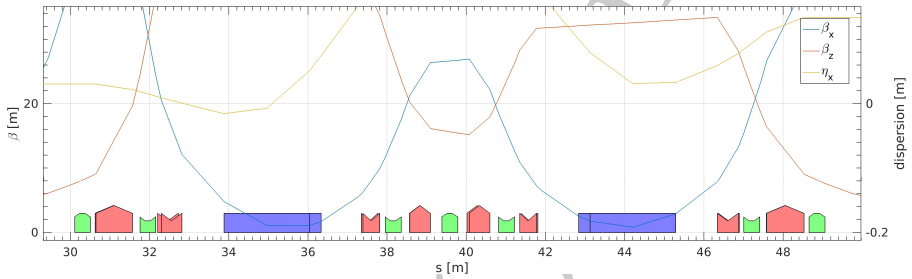



Figure 4: Longitudinal displacements of dipole.

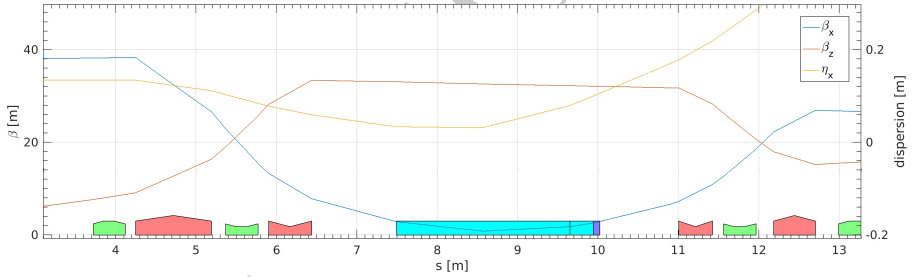
## functions mentioned in the above section

- atsetshift
- atsettilt
- atsettiltdipole

- atset$_{ss}hift$




(a) Quadrupoles longitudinally displaced




(b) Dipole longitudinally displaced.

Figure 5: Longitudinal displacements.

## 1.3 BPM errors

BPM errors are: offset, rotation, gain and reading precision (random). Those are set in the lattice with the function *atsetbpmerr*.

```
% get indexes
indm=find(atgetcells(ring,'Class','Monitor'));

% define bpm offset and rotation errors
ox=1e-5*randn(size(indm)); % random offset errors of 10um
oy=1e-5*randn(size(indm));
gx=1e-3*randn(size(indm)); % random gain errors of 0.1%
gy=1e-3*randn(size(indm));
rx=1e-6; % reading error sigma of 1um (can also be a vector)
ry=1e-6;

% set errors in the lattice
ringerr=atsetbpmerr(ringerr,indm,ox,oy,gx,gy,rx,ry,rot);
```

To obtain BPM readings including the errors the function *findorbit4Err* and *findorbit6Err* are used. Below the simple function *findorbit6Err* where the bpm errors are recovered and set on the orbit obtained by *findorbit6Err*. Future AT versions will try to implement this in a dedicated PassMethod in C, to increase the speed of this transformation.

```
function orbit = findorbit6Err(RING, indbpm,varargin)
% findorbit6 with bpm reading errors
%
%see also findorbit6 bpm_process bpm_matrices

orbit = findorbit6(RING, indbpm, varargin{:});

% read errors stored in BPM elements
useind=1:length(indbpm);
[rel,tel,trand] = bpm_matrices(RING(indbpm(useind)));

% modify x and y coordinate to consider the errors
bpmreading = bpm_process(orbit([1,3],:),rel,tel,trand);
orbit(1,:)=bpmreading(1,:);
orbit(3,:)=bpmreading(2,:);
```

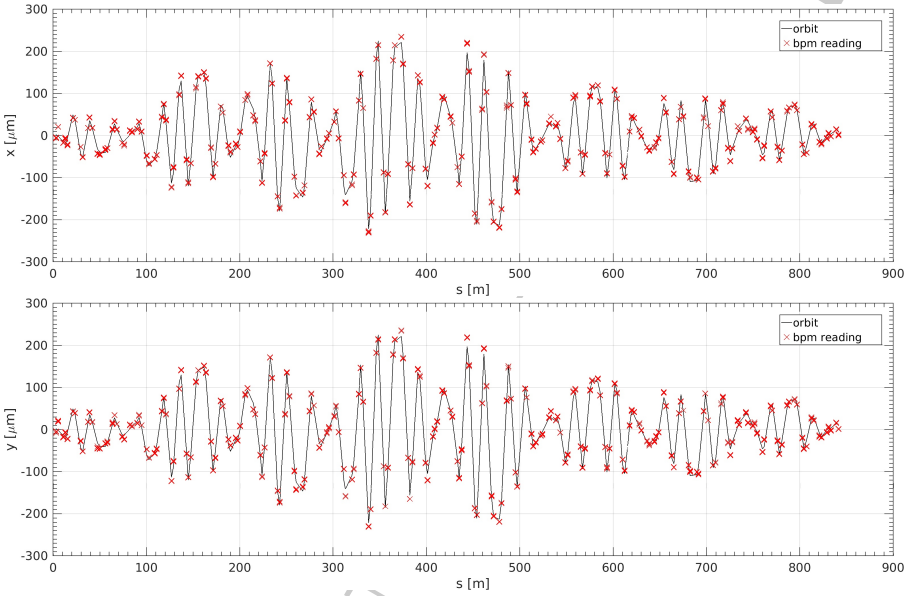
Below a complete script to display the effect of BPM errors.

```
% get indexes
indm=find(atgetcells(ring,'Class','Monitor'));
indq=find(atgetcells(ring,'Class','Quadrupole'));
%ring=atsetfieldvalues(ring,indm,'PassMethod','MonitorPass');

% define quadrupole alignment and rotation errors
dx=1e-6*randn(size(indq)); % random errors of 1um
dy=1e-6*randn(size(indq)); % random errors of 1um
dt=1e-6*randn(size(indq)); % random errors of 1urad

% define bpm offset and rotation errors
```

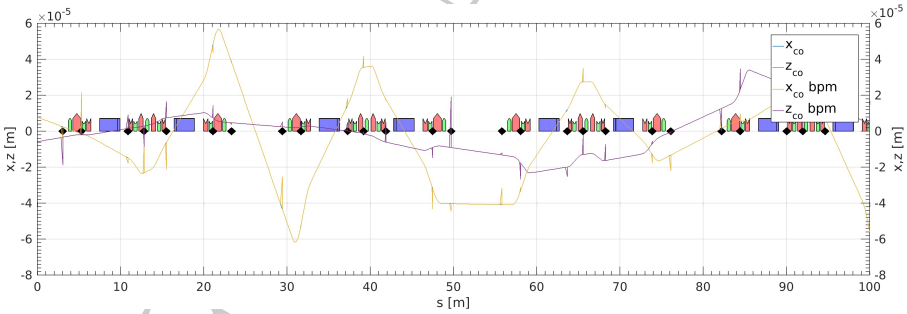
```
ox=1e-5*randn(size(indm)); % random offset errors of 10um
oy=1e-5*randn(size(indm));
gx=1e-3*randn(size(indm)); % random gain errors of 0.1%
gy=1e-3*randn(size(indm));
rx=1e-6; % reading error sigma of 1um (can also be a vector)
ry=1e-6;
rot=1e-5*randn(size(indm)); % random rotation errors of 10urad

% set errors
ringerr=ring;
ringerr=atsetshift(ringerr,indq,dx,dy);
ringerr=atsettilt(ringerr,indq,dt);
ringerr=atsetbpmerr(ringerr,indm,ox,oy,gx,gy,rx,ry,rot);

% plots
figure('units','normalized','position',[0.1 0.4 0.65 0.35]);
s=findspos(ringerr,indm);
% no bpm errors
o=findorbit4(ringerr,0,indm);
plot(s,o(1,:)'*1e6,'k'); hold on;
% with bpm errors
oe=findorbit4Err(ringerr,0,indm);
plot(s,oe(1,:)'*1e6,'rx');

legend('orbit','bpm reading');
xlabel('s [m]');ylabel('x [\mum]')
```

## functions used above

- atsetbpmerr
- findorbit4Err
- findorbit6Err




(a) Simulated orbit and 5 BPM orbit reading




(b) Difference between orbit and orbit reading displaied by *atplot*

Figure 6: Orbit and BPM orbit reading.

## 1.4 Several errors in the lattice

In most cases several errors need to be set in the lattice simultaneously. The function *atsetrandomerrors* accepts a more structured input to simplify the specification of long lists of errors. A common seed is specified in order to have deterministic error sets. Below a sample of code and output errors, where several errors are specified and set simultaneously. The functions also considers splitted elements grouped using the *MagGroup* flag in the AT structure fields.

```
% sextupoles
inds=findcells(r0,'Class','Sextupole');
errstruct(1).indx=inds;
errstruct(1).type='psi'; % roll
errstruct(1).sigma=200*1e-6;

% quadrupoles
indqm=[findcells(r0,'Class','Quadrupole')];
errstruct(2).indx=indqm;
errstruct(2).type='x';
errstruct(2).sigma=150*1e-6;
errstruct(3).indx=indqm;
errstruct(3).type='y';
errstruct(3).sigma=170*1e-6;

% girders
indg=[findcells(r0,'FamName','GS')];
errstruct(4).indx=indg;
errstruct(4).type='gx.gy';
errstruct(4).sigma=500*1e-6;

% set errors
magindex=arrayfun(@(a)a.indx,errstruct,'un',0);
type=arrayfun(@(a)a.type,errstruct,'un',0);
sigma=arrayfun(@(a)a.sigma,errstruct,'un',0);

rerr=atsetrandomerrors(...
    r0,...
    magindex,... % cell array of indexes
    findcells(r0,'Class','Monitor'),...
    123456,... % common seed
    sigma,...  % cell array of sigmas
    2.5,...    % truncation
    type);     % cell array of type
```

The errors that can be set are coded as follow:

- individual magnets: x, y, s, psi (roll), theta (yaw), phi (pitch), x.y, x.y.psi, x.y.s.psi, x.y.s.psi.theta.phi
- bpm : bpm.offset, bpm.scale, bpm.read
- girders: gx, gy, gpsi, gtheta, gphi, gx.gy, gx.gy.gpsi, gx.gy.gpsi.x.y.psi
- main field components: dpb1, dpb2, dpb3, dpb4

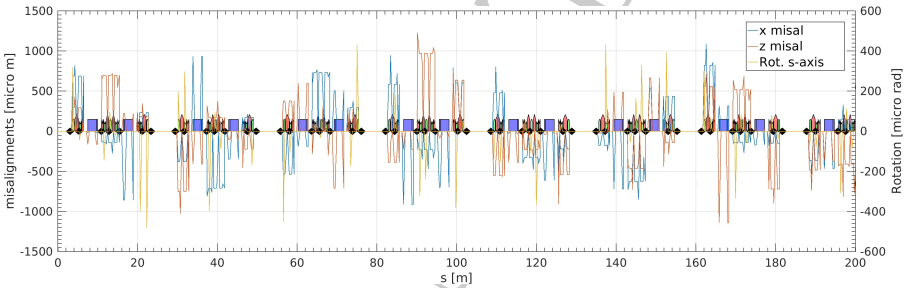



Figure 7: List of errors for different elements set using *atsetrandomerrors*

## 1.5 Girder errors

Girder errors can be implemented in two ways: moving by the same amount all elements that belong to a girder or setting the T1 and R1 field for the first elements and T2 and R2 field for the last one. The markers GS and GE define the start and end of Girders in the lattice and are used to specify the errors. The first case is implemented in the function *atsetrandomerrors* with error types *gx, gy, gpsi, gx.gy ...*

```
% get indexes
indm=find(atgetcells(ring,'Class','Monitor'));
% girders are defined by GS and GE markers (start and end of girder)
indg=find(atgetcells(ring,'FamName','GS'));

% example: set pitch errors
rerr=atsetrandomerrors(...
    ring,... % lattice
    indg,... % indexes
    indm,... % bpm indexes
    1,...   % seed
    1e-5,... % sigma or random error
    2.5,... % number of sigmas for truncation
    'gphi'); % error kind

rerr2=atsetrandomerrors(...
    ring,... % lattice
    indg,... % indexes
    indm,... % bpm indexes
    1,...   % seed
    1e-5,... % sigma or random error
    2.5,... % number of sigmas for truncation
    'gx.gy.gpsi'); % error kind
```

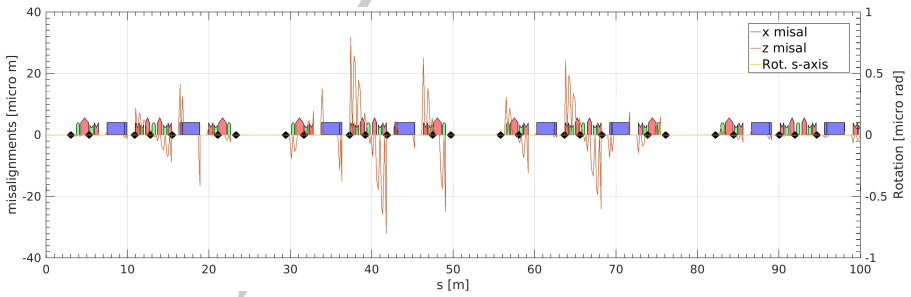



Figure 8: Girders pitch errors set using *atsetrandomerrors*

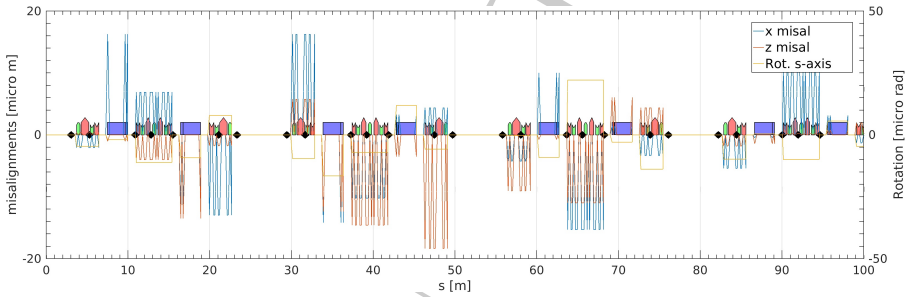



Figure 9: Girders horizontal, vertical and rotation about s errors set using *atsetrandomerrors*

## 1.6 Survey errors

Measurements of the positions of the accelerators are frequently performed. Knowing these measurement and their errors, it is also possible to simulate survey curves. This curves are a better estimation of the global alignment errors, and can eventually replace the girder-to-girder errors. If no measurements are available, studies of errors as shown in Sec. 1.7 can be performed.

These errors can be very large, but as they are smoothly varying along the lattice, their effect is limited. BPM alignment errors (T1,R1,...) are used in *findorbit4Err* and *findorbit6Err* to offset the BPM reading. This defines a reference closed orbit that follows the global Survey curves.

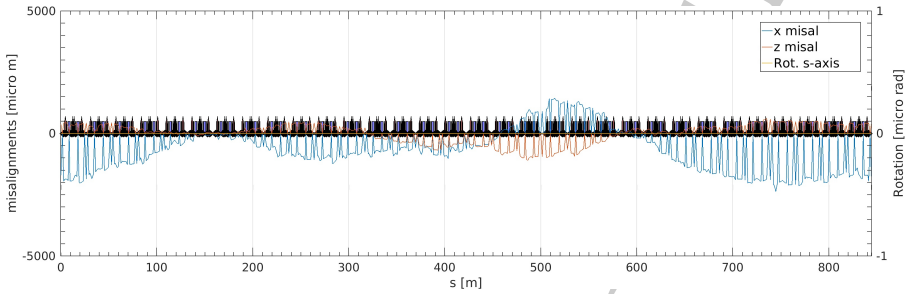



Figure 10: Survey Errors

## 1.7 Errors in frequency domain

Errors can be also studied in terms of frequency and amplitude. The function *atsetwaveerrors* allows this kind of study. Below a sample of code and the resulting errors.

```
ie=1;

wltouse=1:0.5:3;
amplx=0.6e-3;
amplY=0.6e-3;
amplpsi=0.6e-3;

W=findspos(r0,length(r0)+1)./wltouse;

A=amplx/length(W)*randn(size(W));
errwavestruct(ie).indx=1:length(r0);%findcells(r0,'Class','Quadrupole');
errwavestruct(ie).type='x';
errwavestruct(ie).A=A(end:-1:1);
errwavestruct(ie).W=W;
ie=ie+1;

A=amplY/length(W)*randn(size(W));
errwavestruct(ie).indx=1:length(r0);%findcells(r0,'Class','Quadrupole');
errwavestruct(ie).type='y';
errwavestruct(ie).A=A(end:-1:1);
errwavestruct(ie).W=W;
ie=ie+1;
```

```
A=amplpsi/length(W)*randn(size(W));
errwavestruct(ie).indx=1:length(r0);%findcells(r0,'Class','Quadrupole');
errwavestruct(ie).type='psi';
errwavestruct(ie).A=A(end:-1:1);
errwavestruct(ie).W=W;
ie=ie+1;

magindex=arrayfun(@(a)a.indx,errwavestruct,'un',0);
type=arrayfun(@(a)a.type,errwavestruct,'un',0);
A=arrayfun(@(a)a.A,errwavestruct,'un',0);
W=arrayfun(@(a)a.W,errwavestruct,'un',0);

rerr=atsetwaveerrors(...
    r0,...
    magindex,...
    findcells(r0,'Class','Monitor'),...
    W,...
    A,...
    type);
```

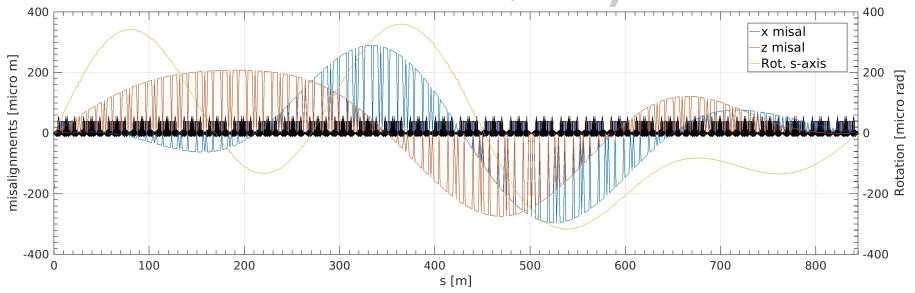



Figure 11: List of low frequency errors (sum of several frequency and amplitudes) for different errors set using *atsetwaveerrors*

## 1.8 Main field errors

To set main field errors the options of *atsetrandomerrors*, *dpb1, dpb2, dpb3, dpb4* can be used.

```
% get indexes
indm=find(atgetcells(ring,'Class','Monitor'));
indq=find(atgetcells(ring,'Class','Quadrupole')); % girders are defined
    by GS and GE markers (start and end of girder)

rerr=atsetrandomerrors(...
    ring,... % lattice
    indq,... % quad indexes
    indm,... % bpms
    1,...   % seed
    1e-1,... % errors of 10%
    2.5,... % truncation
    'dpb2'); % error type: dpb1, dpb2, dpb3, dpb4
```

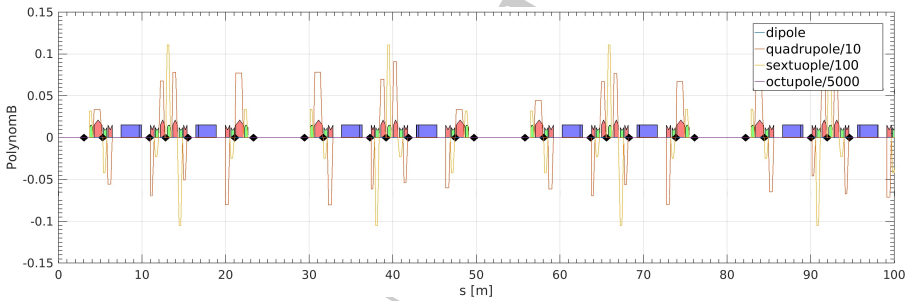



Figure 12: Quadrupole field errors set using *atsetrandomerrors*

## 1.9 Multipole errors

Multipole errors may be systematic or random. To set on a given magnet a series of multipole errors, in AT is sufficient to state them in PolynomB and PolynomA, and increase accordingly the MaxOrder field of the element. However care has to be taken since different conventions might be used for the magnetic field decomposition. The field decomposition in AT is:

$$B = B\rho * \Sigma_{n=1}^{\infty}((b_n + i * a_n)(z)^(n-1))$$

where $b_n$ and $a_n$ are PolynomB and PolynomA while for the magnet design group of ESRF is:

$$B = B_N(\rho_0) * \Sigma_{n=1}^{\infty}((B_{N,n} + i * A_{N,n})(\frac{z}{\rho_0})^{n-1})$$

where $\rho_0$ is the reference radius and $B_{N,n}$, $A_{N,n}$ the multipole components.

The function *AssignFieldErr* does the conversion and sets the multipole errors, scaling with the main field.

```
% get indexes
indq=find(atgetcells(ring,'Class','Quadrupole'));

% set multipole errors
bn=[0 0 0 0 0 1.10934 0 0 0 -5.18658]*1e-4;

an=[
    0
    0
    4.803458371
    1.910276957
    1.055734675
    0.588073151
    0.312742308
    0.175288289
    0.101114708
    0.064747269]'*1e-4;

[rerr,PolB,PolA]=AssignFieldErr(ring,indq,2,7*1e-3,bn,an);
```

**Multipole errors in correctors**

In the case of correctors magnets, the multipole errors must be updated every time the correctors change strength.

## 1.10 Retrieving and displaying the errors values in AT

# 2 Corrections

This section describes several possible correction tools. All the functions are based on the pseudo-inversion using SVD (available in matlab) of an adequate Response Matrix (RM). All functions rely on orbit, trajectory and dispersion computations in 6D. Also BPM errors are always included in the simulations. The functions used for this purpose are:

- *findtrajectory6Err* : linepass starting at inCOD + BPM errors.
- *findorbit6Err* : orbit in 6D
- *finddispersion6Err* : moves RF frequency to observe orbit variation off energy

The functions implemented for correction (detailed below) are:

- *atfirstturntrajectory* : correct first turn trajectory
- *atsetRFCavity* : correct RF frequency (uses RFCavityPass.c PassMethod)
- *atfittune,fittunedelta2fam* : correct tunes
- *atmatchchromdelta* : correct chromaticity
- *atcorrectorbit* : correct COD using steerers
- *atcorrectdispersion* : correct dispersion using quadrupoles
- *atdispersionfreesteering* : correct COD and dispersion using steerers
- *atRDTdispersioncorrection* : correct RDT and dispersion using quadrupoles

Also fitting functions can be implemented, but are often very dependent on the fit choices and on the available computing clusters. So they are not included here.

## 2.1 Response matrices

All response matrices are computed by a common function that outputs the structure collecting the Responses that is used by all correction functions. The function *getresponsematrices* centralizes the computation of all the matrices. A vector of integers is used to specify wich matrices to be computed and all functions have implemented a default RM computation calling *getresponsematrices*. Below an example of call to *getresponsematrices*.

```
ModelRM...
       =getresponsematrices(...
       ring,...    % lattice to compute RM
       indBPM,...   % Beam position Monitors indexes in ring
       indHorCor,... % Horizontal steerers indexes in ring
       indVerCor,... % Vertical steerers indexes in ring
       indSkewCor,... % Skew quadrupole corrector indexes in ring
       indQuadCor,... % Normal quadrupole corrector indexes in ring
       indSextCor,... % Sextupole corrector indexes in ring
       [0 0 0 0 0 0]',... % guess intial coordinated
       [1 2 3]);   % compute horizontal, vertical and dpp orbit response
```

The structure *ModelRM* contains all the computed RM in different fields. All RM are normalized by the kick used for the computation and computed varying the relevant magnets on "2 sides". All oirbits are computed using 6D functions including BPM errors.

| index | response matrix computed | name of substructure |
|---|---|---|
| 1 | $\frac{\Delta orbit(6D)}{\Delta hsteerer}$ | ModelRM.OrbHCor |
| 2 | $\frac{\Delta orbit(6D)}{\Delta vsteerer}$ | ModelRM.OrbVCor |
| 3 | $\frac{\Delta orbit(6D)}{\Delta dpp}$ | ModelRM.OrbHDpp |
| | $\frac{\Delta orbit(6D)}{\Delta dpp}$ | ModelRM.OrbVDpp |
| 4 | $\frac{\Delta trajectory(6D)}{\Delta hsteerer}$ | ModelRM.TrajHCor |
| 5 | $\frac{\Delta trajectory(6D)}{\Delta vsteerer}$ | ModelRM.TrajVCor |
| 6 | $\frac{\Delta trajectory(6D)}{\Delta dpp}$ | ModelRM.TrajHDpp |
| | $\frac{\Delta trajectory(6D)}{\Delta dpp}$ | ModelRM.TrajVDpp |
| 7 | $\frac{\Delta dispersion(6D)}{\Delta hsteerer}$ | ModelRM.DispHCor |
| 8 | $\frac{\Delta dispersion(6D)}{\Delta vsteerer}$ | ModelRM.DispVCor |
| 9 | $\frac{\Delta dispersion(6D)}{\Delta dpp}$ | ModelRM.DispHDpp |
| 9 | $\frac{\Delta dispersion(6D)}{\Delta dpp}$ | ModelRM.DispVDpp |
| 10 | $\frac{\Delta dispersion(6D)}{\Delta normquad}$ | ModelRM.DispQCor |
| 11 | $\frac{\Delta dispersion(6D)}{\Delta skewquad}$ | ModelRM.DispSCor |
| 12 | $\frac{\Delta tune}{\Delta normquad}$ | ModelRM.TuneQCor |

## 2.2 Tune

Tune is set using *atfittune*. In case of large tune deviations affecting also the integer part, *atmatchtunedelta* matches the total phase advance of the lattice to be corrected.

```
% atfittune
NEWRING = ATFITTUNE(RING,NEWTUNES,QUADFAMILY1,QUADFAMILY2);
```

## 2.3 Chromaticity

Since in a lattice with errors all sextupoles might have different strengths, the chromaticity correction is computed adding a common variation to 2 families of sextupoles. This is implemented using *atmatch* in the function *atmatchchromdelta*. Below an example.

```
% sextupole indexes
indS=find(atgetcells(r0,'Class','Sextupole'))';
pbsxt=atgetfieldvalues(r0,indS,'PolynomB',{1,3});
indSF=indS(pbsxt>0);
indSD=indS(pbsxt<0);

% fit chromaticity adding a common variation to 2 sext failies
rerr=atmatchchromdelta(rerr,chrom,{indSF,indSD});
```

## 2.4 RF cavity

The functions *atsetRFCavity*is used to set the RF cavity frequency and time lag to the optimal values in presence of radiation and eventual lattice errors. Due to the available integration methods in AT (CavityPass and RFCavityPass), at the moment it is not possible to consider the true lattice elongation: in the cavity pass methods in AT the sum of the Length fields in all elements is considered as the length of the lattice.

The function *atRFcorrection* is used to set a zero average energy deviation in the lattice, by minimization of the COD using an RF frequency shift ( cancel dispersive contribution in the orbit). The time lag is the set according to the 6th coordinate in tracking at the cavities.

```
% RF cavity parameters
rfv=9.0e6; % Volts
harm=992;
radon= 1; % 1 = radiation on
DeltaHz=0; % no errors, frequency has no modification due to COD
          % this DeltaHz can be also used to see dispersive orbits.

% set RF cavity freqeuncy and time lag
[rerr]=atsetRFCavity(rerr,rfv,radon,harm,DeltaHz);

% correct RF frequency for <dpp>=0
[   rcor,...           %output corrected lattice
    inCODcor,...   % intial coordinate guess after correction
    fc....         % corrected RF frequency
    ]=atRFcorrection(...
    rerr,...       % lattice to be corrected
    indBPM,...     % BPM indexes for orbit computation
    inCOD,...      % initial orbit guess
    [1 1 1 1 1],... % number of eigenvectors for reiteration
    1,...              % scaling factor for correction
    ModelRM,...    % response matrix, if [] will be computed
    zeros(2,length(indBPM)),... % reference orbit for correction
    true);         % verbosity flag
```

The correction of RF frequency for minimal energy deviation is not optimized for dynamic apertures, while the frequency obtained using *atsetRFCavity* is always the best possible frequency.

## 2.5 Orbit

Orbit correction is performed using *atcorrectorbit*. The functions implements several features for correction:

- the average steerers strenghts 0,
- iteration of the correction varying the number of eigenvectors
- correction of the frequency (as in *atRFcorrection*).
- possibility to limit the steerers strengths

Below an example of usage of the orbit correction function.

```
[rcor,...          % corrected lattice
   inCOD,....      % initial orbit guess after correction
   hs,...          % total horizontal steerers strenghts
   vs....          % total vertical steerers strengths
   ]=atcorrectorbit(....
   rerr,...        % lattice to be corrected
   indBPM,...      % BPM indexes
   indHCor,...     % horizontal steerers indexes
   indVCor,...     % vertical steerers indexes
   inCOD,...       % input 6D closed orbit guess
   [...            % several correction iterations
   [10 20];...     % with different number of eigenvectors
   [30 40];...     % for horizontal and vertical plane
   [50 60];...     % <-- iter 3, use 50 eig hor., 60 eig ver.
   [70 70];...     % <-- iter 4, use 70 eig hor., 70 eig ver.
   [80 80];...     % <-- iter 5, use 80 eig hor., 80 eig ver.
   [97 96];...
   [97 96]...
   ],...
   [true true],... % [do dpp correction, keep average of correctors
       zero]
   1.0,...         % scale factor for correction
   ModelRM,...     % response matrix, if [], compute it
   zeros(2,length(indBPM)),... % reference orbit to correct to
   [0.5e-3 0.5e-3],... % sterrer strengths limits
   true);          % verbosity flag
```

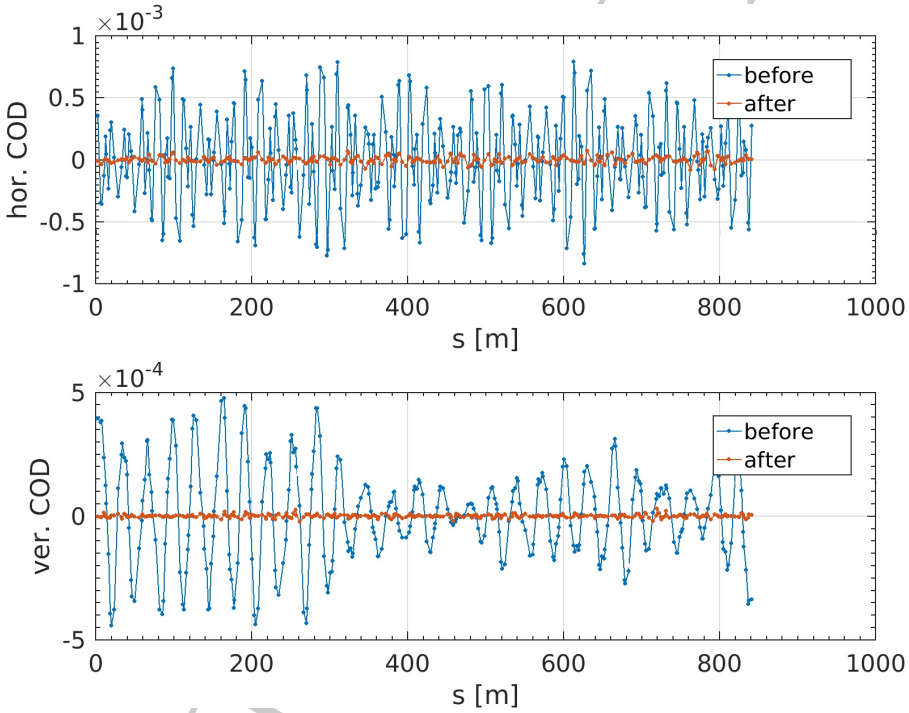



Figure 13: Orbit correction using *atcorrectorbit*

### 2.5.1 Closed orbit bumps

By giving a different orbit reference to *atcorrectorbit*, it is possible to obtain closed orbit bumps. Below the code to obtain this result and a figure representing the result (figure **??**)

```
refx=zeros(size(indBPM));
refy=zeros(size(indBPM));

refx([7 8])=1e-4; % horizontal bump in position
refy([7 8])=[1e-4 -1e-4]; % vertical bump in angle

% correct restrictig response matrices
selbpm=[2 3   7 8  12 13]; % 6 and 9 missing, inside bump
selcor=[2 3         4  5]; % correctors

[rbump,...          % lattice with COD bump
inCOD...
]=atcorrectorbit(...
    ring,...
    indBPM(selbpm),...
    indHCor(selcor),...
    indVCor(selcor),...
    inCOD,...
    repmat([4 4],3,1),... % iterate 3 times with 4 eigenvectors (all)
    [false false],...
    1.0,...
    [],... % compute RM for subset of bpm and correctors
    [refx(selbpm);...
     refy(selbpm)],...
    [],...
    true);
```

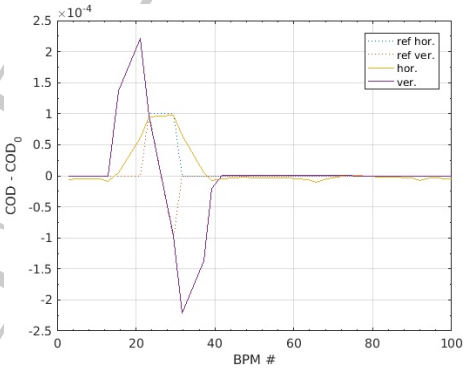



Figure 14: Orbit bump in hor. plane and angle bump in ver. plane using *atcorrectorbit*

## 2.6 Dispersion

Dispersion correction is performed using *atcorrectdispersion*. The functions implements several features for correction:

- the average correctors strenghts 0,
- iteration of the correction varying the number of eigenvectors
- correction of the frequency (dpp/quadrupoles response).
- possibility to limit the correctors strengths

Below an example of usage of the orbit correction function.

```
% initial closed orbit guess and dispersion
inCOD=[0 0 0 0 0 0]';
[l,~,~]=atlinopt(ring,0,indBPM);
refdispersion=zeros(2,length(indBPM));
refdispersion(1,:)=arrayfun(@(a)a.Dispersion(1),l);
refdispersion(2,:)=arrayfun(@(a)a.Dispersion(3),l);


[rcor,...
inCOD,...
hs,vs...            % total quadrupole strengths after correction
]=atcorrectdispersion(...
    rerr,...        % lattice with errors to correct
    indBPM,...   % BPM indexes
    indQCor,...     % quadrupole correctors indexes
    indSCor,...     % skew quad. correctors indexes
    inCOD,...       % initial COD guess 6D
    [floor(linspace(20,250,7));... % eigenvectors at each iteration
     floor(linspace(20,250,7))]',...
    [false true],... % [correct DPP, keep average correctors zero]
    1.0,...         % scale factor for correction
    ModelRM,...     % model RM, if [], compute it
    refdispersion,... % model dispersion
    [],...          % correctors limits, default= no limits
    true);          % verbosity
```

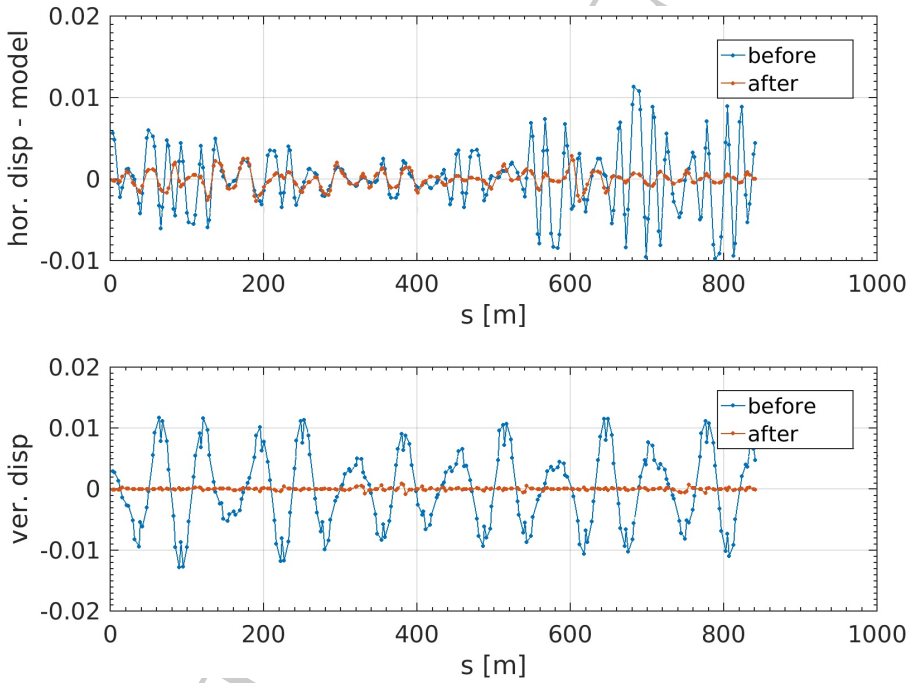



Figure 15: Dispersion correction using *atcorrectdispersion*

## 2.7 Dispersion free steering

The function *atdispersiofreesteering* uses the steerers to simultaneously correct orbit and dispersion (dispersion free steering [3]. The function implementation is very similar to the one of orbit and dispersion correction, but adds a weight parameter to tue the relative correction of orbit and dispersion.

```
% same input as atcorrectorbit, apart for indicated
[rcor,inCOD,hs,vs]=atdispersionfreesteering(...
    rerr,...
    indBPM,...
    indHCor,...
    indVCor,...
    inCOD,...
    [floor(linspace(20,96,7));... % eigenvectors at each iteration
     floor(linspace(20,97,7))]',...
    [true false],...
    1.0,...
    0.9,...  % <-- dispersion weigth
    ModelRM,...
    zeros(2,length(indBPM)),...
    refdispersion,... % <-- dispersion reference
    [],...
    true);
```

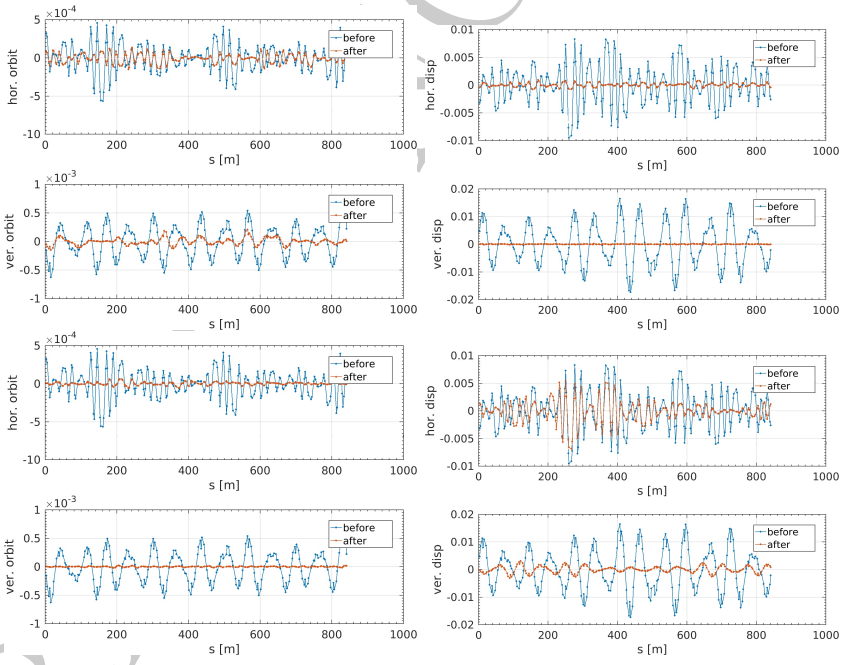



Figure 16: Dispersion free steering using *atdispersiofreesteering*. Top plots, include the dispersion in the correction, bottom ones correct only orbit (weigth = 0.0)

## 2.8 Correction of RDT

The correction of optics may be performed inverting a response to beta functions or phase advance. Unfortunately these parameters do not depend linearly on the normal and skew quadrupoles strengths. In [4] we find a solution to this problem: the correction of normal and skew quadrupole resonance driving terms (RDT). This kind of correction is perfectly suited for the correction of a fitted lattice model that includes normal and skew quadrupoles errors. The functions retrieves the normal and skew quadrupole components in the lattice and computes the RDT values. Then the real and imaginary part of those terms are corrected in a system with dispersion and tunes. This correction is implemented in the function *atRDTdispersioncorrection*. Three spin-off functions are also present:

- *atRDTdispersioncorrection*: normal and skew quadrupole RDT, dispersion and tune correction
- *atQuadRDTdispersioncorrection*: normal quadrupole RDT, dispersion and tune correction
- *atSkewRDTdispersioncorrection*: skew quadrupole RDT, dispersion and tune correction
- *atRDTdispersionmeasuredcorrection*: normal and skew quadrupole RDT, measured dispersion and tune correction

An example of the application of the correction is shown below and in figure 17 and 18.

```
[rcor,...           % corrected lattice
inCOD,...
hs,...              % total quadrupole strengths
vs...               % total skew quadrupole strengths
]=atRDTdispersioncorrection(...
    rerr,...        % lattice with errors, or better, fitted errors model
        from RM measurements
    r0,...          % reference lattice to compute RDT and dispersion
    indBPM,...
    indQCor,...
    indSCor,...
    inCOD,...
    [...            % number of eigenvector at each iteration for
    [15 30];...     % quad and skew quad correctors
    [30 60];...
    ],...
    [true],...   % average correctors to zero
    1.0,...         % scale factor
    [0.8 0.1 0.8],...    % weigths for hor. dispersion, tune, ver.
        dispersion
    ModelRM);       % Response matrices, if[], compute them.
```

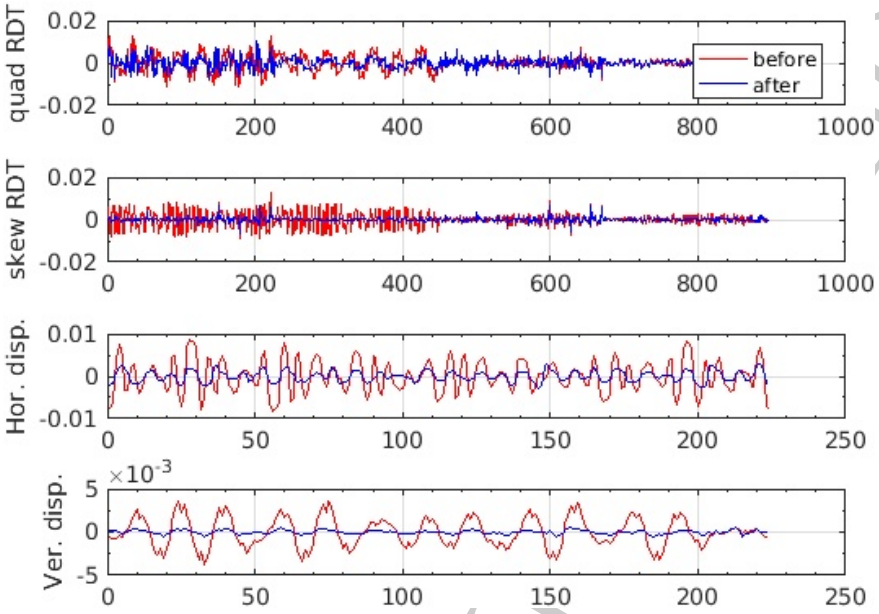



Figure 17: RDT and Dispersion correction using *atRDTdispersiocorrection*. Dispersion weigth = 0.8

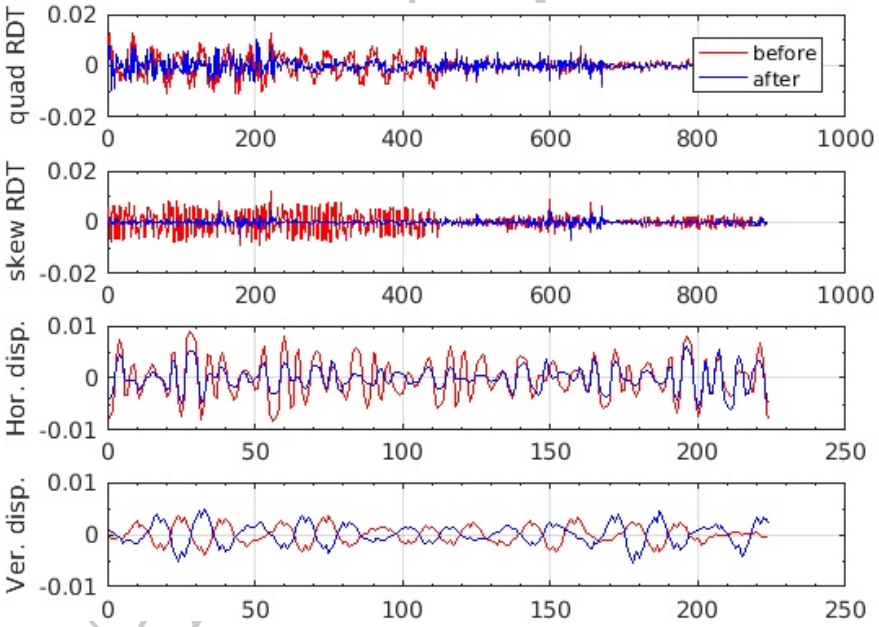



Figure 18: RDT and Dispersion correction using *atRDTdispersiocorrection*. Dispersion weigth = 0.0

## 2.9 Coupling free steering

**description**

**example**

```
% matlab code here
```

## 2.10 Open trajectory correction

If the errors set in the lattice are too large, it is possible that no COD is found by AT. This problem is often found in real commisioning, when the first turn is not at all granted. The function *atfirstturntrajectory* finds a first turns COD by using the available trajectory at BPMs.

The algorithm follows these steps:

- look for all BPMs with signal below a given threshold and correct the trajectory using a response computed from the model without errors.
- if all BPM see a signal, close the trajectory using the last 2 correctors in the lattice to match the reading of the first 2 BPMs.
- if stack at less then all the BPMs, then increase the threshold up to +1mm
- if still stack, look for an optimal injection point.
- if still failing to get to the end of the lattice, compute trajectory response on lattice with errors.

The number of correctors used can be limited to a

```
inCOD=[0 0 0 0 0 0]';

[rcor,...                   % lattice with COD
inCOD...                  % intial orbit guess
]=atfirstturntrajectory(...
    rerr,...              % lattice without COD
    inCOD,...             % initial COD guess
    indBPM,...          % bpm indexes
    indHCor,...           % hor. steerers indexes
    indVCor,...           % ver. steerers indexes
    0.5e-2,...          % bpm reading limit [m]
    30,...                % corrector to use beofre last BPM with signal
    [false true],... % [dpp correction, average correctors to zero]
    ModelRM,...           % RM, if[], compute
    zeros(2,length(indBPM)),... % reference orbit
    [],...                % steerers strengths limits
      true);                 % verbosity
```

## 2.11 Commissioning like correction sequence

The above correction are often requested in a sequence. The function *CorrectionChain* allows to perform a full correction sequence calling the above functions from a single entry point.

```
neigenvectors=[...
    200,... % n eig orbit H
    200,... % n eig orbit V
```

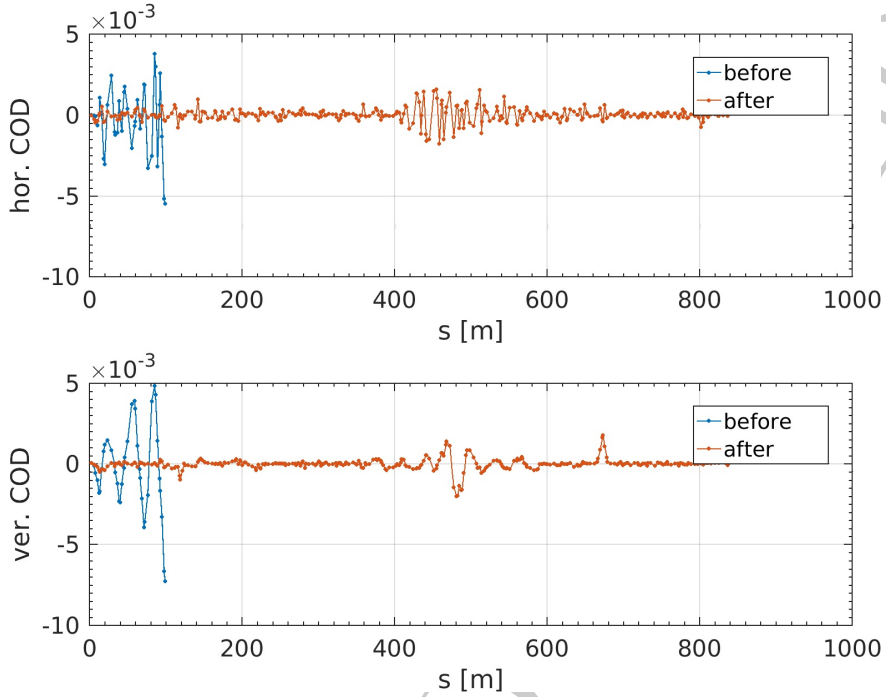



Figure 19: Trajectory correction using *atfirstturntrajectory*.

```
    200,... % skew quadrupole
    250,... % normal quadrupole
    350,... % fit normal quadrupole
    100,... % fit dipole
    350,... % fit skew quadrupole
    ]; % number of eigenvectors

cororder=[0 1 2 3 7 1 2 3 7 1 2 3 -1];
%  '(-1 ): RF cavity frequency and time lag tuning '...
%  '( 0 ): open trajectory (finds closed orbit) '...
%  '( 1 ): orbit '...
%  '( 2 ): tune '...
%  '( 3 ): chromaticity '...
%  '( 4 ): dispersion '...
%  '( 5 ): dispersion free steering '...
%  '( 6 ): rdt + dispersion correction '...
%  '( 7 ): fit errors model and correct model quad RDT + dispersion (6)'

speclab='test';
verbose=true;

[...
    rcor,...            % corrected lattice
     ch,...               % final H cor values
    cv,...              % final V cor values
    cq,...              % final Quad cor values
    cs...               % final Skew Quad cor values
```

```
]=CorrectionChain(...
rerr,...          %1  initial lattice with errors
r0,...            %2  model lattice
indBPM,...        %3  bpm index
indHCor,...       %4  h steerers index
indVCor,...       %5  v steerers index
indSkewQuadCor,... %6 skew quad index
indQuadCor,...    %7  quadrupole correctors index
Neig,...          %8  number of eigen vectors [NeigorbitH,
    NeigorbitV, NeigQuadrdt, Neigdispv, Neigdisph,neig rdt corr,
    SkewQuadRDT]
corrorder,...     %9  correction order 1: orbit, 2: tune, 3:
    skewquad disp v 4: quad disp h 5: quad RDT 6: skew RDT
ModelRM,...        %10 response matrices
speclab,...        %11 response matrices
verbose)           %12 verbose (false): if true print out all
    relevat quantities after each step in corrorder
```

## References

[1] Terebillo

[2] Portman

[3] raimondi

[4] Franchi