BNL-221455-2021-TECH

NSLSII-ASD-TN-351

NSLS-II APHLA Linear Coupling Correction Tools

Y. Li

May 2021

Photon Sciences

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Basic Energy Sciences (BES) (SC-22)

DISCLAIMER

NSLS-II TECHNICAL NOTE BROOKHAVEN NATIONAL LABORATORY	NUMBER **NSLSII-ASD-TN-351**
AUTHORS: Yongjun Li	DATE 05/20/2021
NSLS-II APHLA Linear Coupling Correction Tools	

NSLS-II APHLA Linear Coupling Correction Tools

Yongjun Li

1. Introduction

This technote explains the NSLS-II linear coupling correction tools.

2. Principle of linear coupling characterization and correction

The linear coupling is extracted from the aligned turn-by-turn data with the BPMs. As explain the next page, two consecutive turns data between two neighboring BPMs is used to construct the 4x4 transportation matrix, then off-diagonal elements are minimized by skew quadrupoles.

Characterization and control of linear coupling using turn-by-turn beam position monitor data

I. LINEAR COUPLING

Consider a 2-dimensional (4-dimensinal phase space) coupled linear periodic dynamical system, such as a charged particle traveling in a storage ring. Particle coordinates in the phase space are denoted by $\vec{v}(s) = (x, p_x, y, p_y)^T$, the 4 dimensional vector with the positions and momenta at location s. Here $\mathbf{x}^T$ means the transpose of vector or matrix $\mathbf{x}$. The one-turn-map $\mathbf{R}$ transforms the vector $\vec{v}^{(n)}$ at turn n to $\vec{v}^{(n+1)}$ at turn $n+1$

$$\vec{v}(s)^{(n+1)} = \mathbf{R}(s)\vec{v}(s)^{(n)}. \tag{1}$$

$\mathbf{R}(s)$ is a 4×4 matrix observed at the location of s and can be written as.

$$\mathbf{R}(s) = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}. \tag{2}$$

Here $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ are 2×2 matrices. In the absence of damping, $\mathbf{R}$ satisfies the symplecticity condition

$$\mathbf{R}^T\mathbf{S}\mathbf{R} = \mathbf{S}, \tag{3}$$

in which, $\mathbf{S}$ is the symplectic matrix

$$\mathbf{S} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{4}$$

Eq. (3) constrains the number of independent elements of $\mathbf{R}$ to be 10.

In Lie algebra language, $\mathbf{R}$ can be interpreted as a quadratic Lie generator [1, 2]

$$f_2 = \sum_{\substack{k+l+m+n=2 \\ k,l,m,n \geq 0}} C_{klmn} x^k p_x^l y^m p_y^n. \tag{5}$$

Thus, the transformation of $\vec{v}$ through the matrix $\mathbf{R}$ is equivalent to a exponential Lie map transformation,

$$\vec{v}^{(n+1)} = R(s)\vec{v}^{(n)} \;\leftrightarrow\; v_i^{(n+1)} = e^{:f_2:} v_i|_{\vec{v}=\vec{v}^{(n)}}. \tag{6}$$

Here, v_i is the i^{th} component of $\vec{v}$. There are also 10 independent quadratic terms in f_2, which correspond the 10 independent elements in $\mathbf{R}$.

The coefficients of monomial terms in f_2 are determined by solving a symmetric, positive definite matrix $\mathbf{F}$ from

$$e^{\mathbf{SF}} = \mathbf{R}. \tag{7}$$

And the Lie generator f_2 reads as

$$\begin{aligned} f_2 &= -\tfrac{1}{2}\vec{v}^T\mathbf{F}\vec{v} \\ &= f_2^{(0)} + f_2^{(c)} \\ &= C_{2000}x^2 + C_{1100}xp_x + C_{0200}p_x^2 + \\ &\quad C_{0020}y^2 + C_{0011}yp_y + C_{0002}p_y^2 + \\ &\quad C_{1010}xy + C_{1001}xp_y + C_{0110}p_xy + C_{0101}p_xp_y. \end{aligned} \tag{8}$$

Here, $f_2^{(0)}$ is the uncoupled generator. $f_2^{(c)}$ is the linear coupling generator, which includes four plane-crossing terms, xy, p_xy, xp_y and p_xp_y. The coefficients of $f_2^{(c)}$ terms actually characterize the linear coupling between two planes. Our algorithm is to extract the coefficients of $f_2^{(c)}$ from TbT data, then to minimize them with non-dispersive skew quadrupoles directly.

Based on previous analyses accomplished by others, some parameterizations can be derived from the coupled Lie generator f_2. Here we briefly discuss how these parameterizations are related to $f_2^{(c)}$. In Sect.**??**, we will compute these parameters using TbT data before and after correction to demonstrate that our algorithm is equivalent to these previous analyses.

First, the fully-coupled matrix $\mathbf{R}$ and the Lie generator f_2 can be converted through Eq.(6) directly. The coupling strength can be observed from two non-zero off-diagonal blocks $\mathbf{B}$ and $\mathbf{C}$. Quantitatively Edwards and Teng normalized $\mathbf{R}$ to a block-diagonal normal mode format [3]. The coupling can be characterized by a 2×2 symplectic matrix $\mathbf{D}$ and a phase ϕ.

Mais and Ripken [4] proposed another parameterization with four generalized eigenvectors of $\mathbf{R}$. In this case two modes I and II, and four β-functions can be derived to describe the frequency and the envelope functions of betatron oscillation. In each plane, the betatron oscillation is composed of two linear independent modes

$$\begin{aligned} u = &\sqrt{J_{u,I}\beta_{u,I}}\cos(\mu_{u,I} + \psi_{u,I}) + \\ &\sqrt{J_{u,II}\beta_{u,II}}\cos(\mu_{u,II} + \psi_{u,II}), \end{aligned} \tag{9}$$

where $u = x, y$, $\beta_{I,II}$ and $\mu_{I,II}$ are the generalized Courant-Snyder betatron envelope functions and phase advances for two modes I and II. $J_{I,II}$ and $\psi_{I,II}$ are constants determined by initial conditions. They will degenerate to the standard Courant-Snyder parameterization when coupling vanishes.

The fully coupled Lie generator f_2 can be separated into two parts as Eq. (8), uncoupled part $f_2^{(0)}$, and coupled part $f_2^{(c)}$. First, the uncoupled part $f_2^{(0)}$ can parameterized with Courant-Snyder normalization as

$$\begin{aligned} f_2^{(0)} = &\frac{\mu_x}{2}(\gamma_x x^2 + 2\alpha_x xp_x + \beta_x p_x^2) + \\ &\frac{\mu_y}{2}(\gamma_y y^2 + 2\alpha_y yp_y + \beta_y p_y^2) = \frac{\mu_x}{2}A_x + \frac{\mu_x}{2}A_y. \end{aligned} \tag{10}$$

Here, $\mu_{x,y}$ is the betatron phase advance per turn, $\alpha_{x,y}, \beta_{x,y}$ and $\gamma_{x,y}$ are the Twiss parameters at s, and $A_{x,y}$ are the action variables. Then four coupled terms,

$$f_2^{(c)} = C_{1010}xy + C_{1001}xp_y + C_{0110}p_xy + C_{0101}p_xp_y \quad (11)$$

can be expressed as a summation of the resonance basis of $f_2^{(0)}$

$$f_2^{(c)} = \sum_{a+b=1,c+d=1} h_{abcd}|abcd\rangle. \quad (12)$$

Where

$$|abcd\rangle = (\sqrt{A_x}e^{i\phi_x})^a(\sqrt{A_x}e^{-i\phi_x})^b(\sqrt{A_y}e^{i\phi_y})^c(\sqrt{A_y}e^{-i\phi_y})^d \quad (13)$$

are the resonance basis of non-coupled $f_2^{(0)}$ in Eq. (10). $A_{x,y}$ and $\phi_{x,y}$ in Eq. (10) and (13) are the action-angle canonical variables. Thus, two pairs of complex conjugates coefficients characterize the coupling

$$\begin{aligned} h_{1010} &= h^*_{0101}, \\ h_{1001} &= h^*_{0110}, \end{aligned} \quad (14)$$

in which h_{1010} is referred as linear sum resonance driving term (RDT), because it can drive a sum resonance when the system tune is close to the resonance line $\nu_x + \nu_y = n$. h_{1001} is defined as difference RDT for the same reason. Ref. [5] proves that the RDTs can be merged with Edwards-Teng parameterization [3].

Assuming the system is decoupled at a certain observation point s, in the matrix language, two off-diagonal blocks $\mathbf{B} = \mathbf{C} = 0$ in $\mathbf{R}$; in Lie algebra language, four plane-crossing terms disappear from f_2 of Eq. (5); and $\beta_{x,II}$ and $\beta_{y,I}$ in Mais-Ripken's parameterization also degenerate to zeros, so are RDTs in (14). In other words, the off-diagonal matrices $\mathbf{B}$ and $\mathbf{C}$ in $\mathbf{R}$, the crossing terms in f_2, two RDTs, and two coupling β-functions characterize a common physics quantity - linear coupling observed at the location of this specific position s. The goal of linear coupling control is to minimize these non-zero terms.

It is worthwhile to point out that, even though a system is decoupled at one observation point, it is not necessarily decoupled at another one. This is clear since $f_2^{(c)}$ is s-dependent. For ring-based light sources, it is crucial to control the coupling at insertion device locations.

II. EXPERIMENTAL CHARACTERIZATION OF LINEAR COUPLING

In this section, we discuss how to characterize the coupling with BPMs' TbT readings experimentally, some other techniques can be found in [6, 7]. In order to obtain synchronized and consecutive TbT data, the beam needs to be excited by pulse magnets, then all BPM readings must be timed with the pulse magnets triggering event within one revolutionary period. From the TbT data array, we first choose two neighboring BPMs, P_i and P_{i+1}, with only a few magnets in-between, and assume the linear transforming matrix $\mathbf{M}_{i,i+1}$ between these two BPMs is known. By ignoring damping and de-coherence, the two BPMs' readings (after subtracting the closed orbit) at the n^{th} turn are related by $\mathbf{M}_{i,i+1}$

$$\begin{pmatrix} x^{(n)} \\ p_x^{(n)} \\ y^{(n)} \\ p_y^{(n)} \end{pmatrix}_{i+1} = \mathbf{M}_{i,i+1} \begin{pmatrix} x^{(n)} \\ p_x^{(n)} \\ y^{(n)} \\ p_y^{(n)} \end{pmatrix}_{i}. \quad (15)$$

With Eq. (15), $p_x^{(n)}, p_y^{(n)}$ at both BPMs are determined. Therefore we obtain beam coordinates in phase space at the locations of the two BPMs for multiple turns. Then the one-turn-map at the location of the i^{th} BPM is the least-squares solution of the linear equations for multiple turns.

$$\begin{pmatrix} x^{(n)} & \dots & x^{(2)} \\ p_x^{(n)} & \dots & p_x^{(2)} \\ y^{(n)} & \dots & y^{(2)} \\ p_y^{(n)} & \dots & p_y^{(2)} \end{pmatrix}_{i} = \mathbf{R}_i \begin{pmatrix} x^{(n-1)} & \dots & x^{(1)} \\ p_x^{(n-1)} & \dots & p_x^{(1)} \\ y^{(n-1)} & \dots & y^{(1)} \\ p_y^{(n-1)} & \dots & p_y^{(1)} \end{pmatrix}_{i}. \quad (16)$$

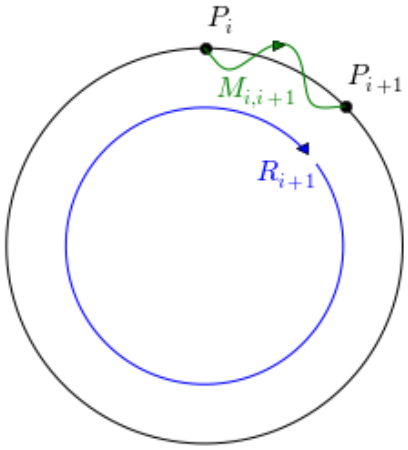

FIG. 1. Using two neighboring BPMs reading and the transport matrix $M_{i,i+1}$ to construct beam coordinates in phase space, then fit out the one-turn-map $\mathbf{R}$ at each observation BPM.

In principle, two consecutive turns data can uniquely define a one-turn map. But due to various errors from BPMs readings, magnet power suppliers jittering, and etc., we have to fit multiple consecutive turns data with the least square method to filter those random errors out. Usually more than 500 turns data are used to solve Eq. (16).

As mentioned before, the couplings seen at different BPMs locations could be different. For a ring-based light source unless for special purpose, e.g. increasing beam volume for longer Touschek life time, it is preferable to have no coupling all over the storage ring, especially at the source points where insertion devices are located. Thus we need to fit out the one-turn-maps at multiple BPM pairs by applying Eq. (15) and (16) repeatedly.

There existes another method to extract the N-turn map to avoid computing $p_{x,y}$ in hadron rings [8]. In the NSLS-II storage ring, a strong amplitude variation due to radiation damping, or decoherence is visible (see Fig. 2) from TbT data. In order to mitigate this effect, we choose two neighboring BPMs to reconstruct momenta $p_{x,y}$, then two consecutive turns data to extract one-turn maps.

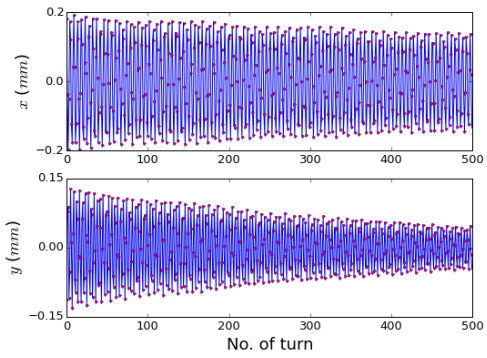

FIG. 2. A set of typical TbT data observed by a BPM on the NSLS-II ring. A strong amplitude variation, especially in the vertical plane, is visible. With damping wigglers gaps closed the damping rate will be further enhanced.

Once $\mathbf{R}$ is obtained, the coefficients of coupling terms C_{klmn}, and then RDTs h_{abcd} can be calculated with Eqs. (7), (8) and (12) respectively. The two coupling β-functions can be calculated with the approach explained in [9].

One thing needs to be emphasized here is, the direct measured $\mathbf{R}$ with Eq. (16) is not always exactly symplectic due to various measurement errors. A symplectic matrix $\mathbf{R_s}$ can be obtained in the following way. First, $\mathbf{R}$ can be approximated to a Lie generator f_2 using Eq. (7). Then we can act f_2 on each canonical variable to get one row of a symplectic matrix $\mathbf{R_s}$ as explained in ref. [2],

$$\begin{aligned} x_1 &= e^{:f_2:}x|_{x=x_0,y=y_0,p_x=p_{x0},p_y=p_{y0}} \\ &= R_{s,11}x_0 + R_{s,12}p_{x0} + R_{s,13}y_0 + R_{s,14}p_{y0}. \end{aligned} \quad (17)$$

Here, only the first row is listed, other three rows can be obtained in the same way. An alternative way of symplectifying $\mathbf{R}$ is given in ref. [8].

Now we discuss the control of various measurement errors. First, BPM's imperfections can affect the calculation of $f_2^{(c)}$ and therefore C_{klmn}. In order to mitigate these affects, for each BPM, four parameters fitted by the LOCO [10–13] method give the full linear transformation between the raw TbT readings $(\bar{x}, \bar{y})$ and the realistic beam trajectory (x, y):

$$\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} = \begin{pmatrix} G_x & C_x \\ C_y & G_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \quad (18)$$

where, $G_{x,y}$ are the gain calibrations, and $C_{x,y}$ are the coupling calibrations due to the roll and the associated construction errors. The four parameters vary for each BPM, as shown in Fig. 3. Calibrated data are obtained by implementing the inverse transformation of Eq. (18) on raw data.

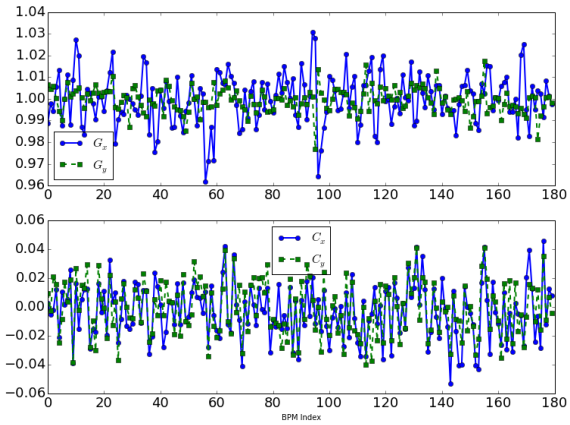

FIG. 3. Gain (upper) and coupling (lower) calibration coefficients for 180 BPMs

BPM resolution is measured as $1\mu m$ at 10mA stored beam current [14]. During the coupling characterization, we usually excite beam with an amplitude less than $\pm 0.5mm$. This resolution can satisfy the requirement of precise characterization of linear coupling.

III. CORRECTION ALGORITHM

Based on the designed lattice model, four plane-crossing terms' dependence on non-dispersive skew quadrupoles are calculated with

$$N_{klmn,i,j} = \frac{\partial C_{klmn,i}}{\partial K_j}, \quad (19)$$

where $C_{klmn,i}$ are the coefficients of Eq. (11) observed at the location of i^{th} BPMs, K_j is the j^{th} skew quadrupole normalized focusing strength, $N_{klmn,i,j}$ is the i^{th} BPM's dependence on the j^{th} skew quadrupole. Once these four coefficients in Eq. (11) at each observation location are measured, the needed skew quadrupole corrections to minimize them are obtained by iteratively solving the following linear equations

$$\Delta\vec{\mathbf{C}}_{klmn} = \mathbf{N}_{klmn}\Delta\vec{\mathbf{K}}. \quad (20)$$

Since there are four goal functions per observation location, we vertically stack their response matrices $\mathbf{N}_{klmn}$ with different weights to minimize them simultaneously.

[1] A. Dragt, AIP Conf.Proc. **87**, 147 (1982).
[2] A. Chao, SLAC-PUB-9574 (2002).
[3] D. Edwards and L. Teng, IEEE Trans.Nucl.Sci. **20**, 885 (1973).
[4] H. Mais and G. Ripken, DESY M-82/05 (1982).
[5] R. Calaga, R. Tomas, and A. Franchi, Phys.Rev.ST Accel.Beams **8**, 034001 (2005).
[6] D. Sagan and D. Rubin, Phys.Rev.ST Accel.Beams **2**, 074001 (1999).
[7] X. Huang, J. Sebek, and D. Martin, Phys.Rev.ST Accel.Beams **13**, 114002 (2010).
[8] W. Fischer, Phys. Rev. ST Accel. Beams **6**, 062801 (2003).
[9] F. Willeke and G. Ripken, AIP Conf.Proc. **184**, 758 (1989).
[10] J. Safranek, Nucl.Instrum.Meth. **A388**, 27 (1997).
[11] J. Safranek, ICFA Beam Dyn.Newslett. **44**, 43 (2007).
[12] L. Yang, S. Lee, X. Huang, and B. Podobedov, Conf.Proc. **C070625**, 804 (2007).
[13] X. Yang, private communication (2015).
[14] W. Cheng, private communication (2015).

3. Software

There is a python package “linOptPy3” provides the needed library functions.

A python jupyter notebook provides a standard template to iteratively correct the optics.

4. Usage

Linear Coupling Correction Version 5.0 By YONGJUN LI 2015-11-04

step 1: initilize (don't modify it)

In [ ]:

```
import matplotlib
import matplotlib.pylab as plt
import linopt_dev as linopt
import pylatt as latt
import h5py,copy
import numpy as np
import aphla as ap
import time

# --- test on the bare lattice
bare,dw08,dw18,dw28,dwall,op12 = 1,0,0,0,0,0

if bare:
    import lattice.nsls2sr_bare_20141015 as nsls2
    fid = h5py.File('nsls2sr_bare_rm_20141015.h5','r')
if dw08:
    import lattice.nsls2sr_dw08_20141205 as nsls2
if dw18:
    import lattice.nsls2sr_dw18_20141205 as nsls2
if dw28:
    import lattice.nsls2sr_dw28_20141119 as nsls2
if dwall:
    import lattice.nsls2sr_dw_20141119 as nsls2
    fid = h5py.File('nsls2sr_bare_rm_20141015.h5','r')
if op12:
    import lattice.nsls2sr_20161123 as nsls2
    fid = h5py.File('nsls2sr_20161123.h5','r')


ap.machines.load('nsls2','SR')

bpmIndex = nsls2.ring.getIndex('moni',exclude='u')
# --- transforming matrix between BPMs
tm = []
for i in range(len(bpmIndex)):
    tm.append(nsls2.ring.getTransMat(bpmIndex[i-1],bpmIndex[i]))

# ---  calculate f2 vs sqh response matrix
# ---  here we only use the first BPMs in 30 straights
#reload(nsls2)
bpmIndex = nsls2.ring.getIndex('moni',exclude='u',exitport=0)
```

```
bpmIndex = bpmIndex[::6]+bpmIndex[5::6]
bpmIndex.sort()
sqhs = nsls2.ring.getElements('skew','sqh')[::2]
c1010rm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['c1010rm']))
c0101rm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['c0101rm']))
c1001rm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['c1001rm']))
c0110rm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['c0110rm']))
h1010rrm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['h1010rrm']))
h1010irm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['h1010irm']))
h1001rrm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['h1001rrm']))
h1001irm = copy.deepcopy(np.array(fid['LinearCouplingResponseMatrix']['h1001irm']))
```

step 2: back-up current skew settings (optinal, just in case you need to recove the old settings)

In [ ]:

```
# --- backup old settings
skqs = ap.getElements('sqh*')
ks1 = []
for skq in skqs:
    ks1.append(skq.get('b1'))
```

step 3: ping beam to get TbT data

1. Two pingers MUST be on and aligned with the filled bunches
2. BPMs MUST in TbT mode and synchronized with pingers

In [ ]:

```
# --- measure TbT data n times
# --- BPM must be in on-demand TbT mode
n = 3
nturns = 2000
X,Y = [],[]

ap.nsls2.resetSrBpms(wfmsel=1, evcode=35)

ap.caput("SR:C21-PS{Pinger:H}V-Sp",0.18)
ap.caput("SR:C21-PS{Pinger:V}V-Sp",0.25)
time.sleep(5)
ap.caput("SR:C21-PS{Pinger}Ping-Cmd",1)
time.sleep(2)
for ii in range(n):
    ap.caput("SR:C21-PS{Pinger}Ping-Cmd",1)
    time.sleep(5)
    (name, x, y, Isum, ts, offset) = ap.nsls2.getSrBpmData(
            trig=1,count=nturns,output=False)
    X.append(x)
    Y.append(y)
    sys.stdout.write('\r --- %04i out of %04i done ---'\
                     %(ii+1,n))
    sys.stdout.flush()

# --- save data to file with given name and comment
```

step 4: Calibrate TbT data (optional, but recommended)

In [ ]:

```
# --- apply BPM calibration
nxy = len(X)
for i in xrange(nxy):
    X[i],Y[i] = linopt.caliBpmGainRoll('nsls2sr_bpm_calibration.h5',X[i],Y[i])
```

step 5: calculate linear coupling, including FFT

In [ ]:

```
# --- carry out fft for one BPM to get tune
ibpm = 1
figsize=(15,3)
n0,n1 = 25,25+1024
x,y = X[0],Y[0]
plt.figure(figsize=figsize)
plt.subplot(211)
plt.plot(x[ibpm,n0:n1]/1e6)
plt.subplot(212)
plt.plot(y[ibpm,n0:n1]/1e6)
plt.show()

nux,ampx,septx1,septx2 = linopt.getTuneTbT(x[ibpm,n0:n1]/1e6,rng=[0.,0.5],ith=1,
    hann=1,verbose=1,semilog=0,figsize=figsize)
nuy,ampy,septy1,septy2 = linopt.getTuneTbT(y[ibpm,n0:n1]/1e6,rng=[0.,0.5],ith=1,
    hann=1,verbose=1,semilog=0,figsize=figsize)

# --- 1st
(c1010_1,c1010err_1),(c0101_1,c0101err_1), \
(c1001_1,c1001err_1),(c0110_1,c0110err_1), \
(h1010r_1,h1010rerr_1),(h1010i_1,h1010ierr_1), \
(h1001r_1,h1001rerr_1),(h1001i_1,h1001ierr_1), \
(rawR_1,symR_1) = \
  linopt.getCoupling(X,Y,tm,cali=0,index=range(0,180,6),prev=1,n0=25,n1=125,verbose=
```

step 6: choose correction method (H or C) and apply correction

In [ ]:

```
# method = 'H', correct h1010 and h1001, real and image parts of driving terms
# method = 'C', correct C1010, C1001, C0110 and C1001 in Hamiltonian

method = 'C'
if method == 'H':
    # --- specify wights on h1010r,h1010i,h1001r,h1001i
    w1010r,w1010i,w1001r,w1001i = 1,1,5,5
    # --- cobmine weighted cofficients for correction
    ct = np.hstack((w1010r*h1010r_1,w1010i*h1010i_1,
                    w1001r*h1001r_1,w1001i*h1001i_1))
    # --- combine weighted response matrix
    rm = np.vstack((w1010r*h1010rrm,w1010i*h1010irm,
                    w1001r*h1001rrm,w1001i*h1001irm))
    plt.figure()
    plt.pcolor(rm)
    plt.colorbar()
    plt.text(15, 30,r'$\Re h_{1010}$',rotation=90,fontsize=15)
    plt.text(15, 90,r'$\Im h_{1010}$',rotation=90,fontsize=15)
    plt.text(15,150,r'$\Re h_{1001}$',rotation=90,fontsize=15)
```

```
    plt.text(15,210,r'$\lm h_{1001}$',rotation=90,fontsize=15)
    plt.axis([0,16.5,0,240])
    plt.xlabel('Index of skew quadrupoles')
    plt.ylabel(r'Index of $H_{klmn}$')
    plt.savefig('respmat.png')
    plt.savefig('respmat.eps')
    plt.show()

    plt.figure(figsize=(15,4))
    plt.plot(range(len(ct)),ct,'o-',linewidth=2)
    plt.show()

elif method == 'C':
    # --- specify wights on c1010,c0101,c1001,c0110
    w1010,w0101,w1001,w0110 = 5,0.2,1,1
    # --- cobmine weighted cofficients for correction
    ct = np.hstack((w1010*c1010_1,w0101*c0101_1,
                    w1001*c1001_1,w0110*c0110_1))
    # --- combine weighted response matrix
    rm = np.vstack((w1010*c1010rm,w0101*c0101rm,
                    w1001*c1001rm,w0110*c0110rm))
    plt.figure()
    plt.pcolor(rm)
    plt.colorbar()
    plt.text(15, 30,r'$C_{1010}$',rotation=90,fontsize=15)
    plt.text(15, 90,r'$C_{0101}$',rotation=90,fontsize=15)
    plt.text(15,150,r'$C_{1001}$',rotation=90,fontsize=15)
    plt.text(15,210,r'$C_{0110}$',rotation=90,fontsize=15)
    plt.axis([0,16.5,0,240])
    plt.xlabel('Index of skew quadrupoles')
    plt.ylabel(r'Index of $C_{klmn}$')
    plt.savefig('respmat.png')
    plt.savefig('respmat.eps')
    plt.show()

    plt.figure(figsize=(15,4))
    plt.plot(range(len(ct)),ct,'o-',linewidth=2)
    plt.show()

# --- calculate needed dK1 for skews
u,s,v = np.linalg.svd(rm)
dk = np.linalg.lstsq(rm,ct,rcond=0.001)[0]
plt.figure(figsize=(15,3))
plt.subplot(131)
plt.bar(range(len(ct)),ct)
plt.xlabel('index of c1010,c0101,c1001,c0110')
plt.ylabel('coefficiets value')
plt.subplot(132)
plt.plot(s/s[0],'-o')
plt.xlabel('index of sigular value')
plt.ylabel('sigular value')
plt.subplot(133)
plt.bar(range(len(dk)),dk)
plt.xlabel('index of SQH')
```

```
plt.ylabel(r'$\Delta K_1 (m^{-2})$')
plt.show()
```

step 6: apply correction

In [ ]:

```
# --- caution: implementation
sqhs =  ap.getElements('sqh*')
b1 = []
for isqh,sqh in enumerate(sqhs):
    b1.append(-dk[isqh]*sqh.length)
linopt.rampMagnet(sqhs,b1,'b1',rampStep=10)
```

step 7: iteration to repeat from step 3 to step 6 (optional)

In [ ]:

```

```

Linear Coupling Correction Version 5.0 By YONGJUN LI 2015-11-04

step 1: initilize (don't modify it)

In [ ]:

```

```

step 2: back-up current skew settings (optinal, just in case you need to recove the old settings)

In [ ]:

```

```