evaluation

October 18, 2018

```
In [1]: from __future__ import division, print_function

        import numpy as np
        import h5py

        import os

        import glob

        import matplotlib.pyplot as plt
        %matplotlib inline

        import seaborn as sns
        sns.set_context('talk', font_scale=1.2)
        sns.set_style('darkgrid', {
            'axes.edgecolor': 'black',
            'axes.linewidth': 2})

        from scipy.io import loadmat
        from scipy.optimize import curve_fit
        from scipy.signal import savgol_filter, find_peaks_cwt

        def make_paper_style():
            plt.grid(False)
            plt.gca().tick_params('both', which='major', size=6, direction='out')
        #     plt.gca().tick_params('both', which='minor', size=4, direction='out')
        #     plt.minorticks_on()
            plt.gca().set_axis_bgcolor('white')
```

```
/user/cpsop/adrian/anaconda2/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Con
  from ._conv import register_converters as _register_converters
```

```
In [2]: import pandas
```

```
In [3]: import sys
        sys.path.append('/afs/cern.ch/work/o/oeftiger/private/git/')
        sys.path.append('/user/cpsop/adrian/python_pkgs/PySUSSIX/')
```

In [4]:
```
import PySussix
```

In [5]:
```
from scipy.signal import hilbert
```

In [6]:
```
from glob import glob
```

In [9]:
```
from scipy.constants import e, m_p, c, epsilon_0
```

1 helper functions

In [12]:
```
def extract_intensity(myDataStruct, ctime=185, window_radius_ms=3):
    t_offset = myDataStruct.PR_BCT_ST.Samples.value.firstSampleTime
    lo, hi = (ctime - window_radius_ms - t_offset,
              ctime + window_radius_ms - t_offset)
    return 1e10 * np.mean(myDataStruct.PR_BCT_ST.Samples.value.samples[lo:hi+1])
```

In [13]:
```
# for transverse emittances
def gauss(x, ampl, mu, sigma):
    '''Args: evaluation position, amplitude, mean, sigma'''
    return ampl * np.exp(-(x - mu)**2 / (2. * sigma**2))

def fit_gauss(position, profile,
        init_guess=[1., 0., 3e-3], # amplitude, mean [m], sigma [m]
        core_sigma=1.8, halo_sigma=4, halo_fit=False):
    baseline = np.median(profile)

    (fit_ampl_pre, fit_mu_pre, fit_sigma_pre), var_matrix_pre = \
        curve_fit(gauss, position, profile - baseline, p0=init_guess)

    core_ids = np.where(np.abs(position - fit_mu_pre) < core_sigma * fit_sigma_pre)[0]
    halo_ids = np.where((np.abs(position - fit_mu_pre) < halo_sigma * fit_sigma_pre) &
                        (np.abs(position - fit_mu_pre) >= core_sigma * fit_sigma_pre))[


    lbound, ubound = halo_ids[[0,-1]]
    pos_uptohalo = np.array(position[lbound:ubound+1])
    prof_uptohalo = np.array(profile[lbound:ubound+1]) - baseline
    prof_uptohalo /= np.trapz(prof_uptohalo, pos_uptohalo)

    std = np.sqrt(np.trapz(pos_uptohalo**2 * prof_uptohalo, pos_uptohalo))

    (fit_ampl_core, fit_mu_core, fit_sigma_core), var_matrix_core = \
        curve_fit(gauss, position[core_ids], profile[core_ids] - baseline,
                  p0=(fit_ampl_pre, fit_mu_pre, fit_sigma_pre))

    fit_sigma_relerr = np.abs(fit_sigma_core - fit_sigma_pre) / fit_sigma_core
    if fit_sigma_relerr >= 0.1:
        print ('*** WARNING: significant relative change of sigma between '
```

```
                'the pre-fit of the whole wire scan, sigma_pre={pre},'
                'and the subsequent core fit, sigma_core={core}'.format(
                    pre=fit_sigma_pre, core=fit_sigma_core))

    fit_quantities = {
        'baseline': baseline,
        'std': std,
        'fit_ampl_pre': fit_ampl_pre,
        'fit_mu_pre': fit_mu_pre,
        'fit_sigma_pre': fit_sigma_pre,
        'var_matrix_pre': var_matrix_pre,
        'core_ids': core_ids, 'halo_ids': halo_ids,
        'fit_ampl_core': fit_ampl_core,
        'fit_mu_core': fit_mu_core,
        'fit_sigma_core': fit_sigma_core,
        'var_matrix_core': var_matrix_core,
        'fit_sigma_relerr': fit_sigma_relerr,
        'core_sigma': core_sigma,
    }

    if halo_fit:
        (fit_ampl_halo, fit_mu_halo, fit_sigma_halo), var_matrix_halo = \
            curve_fit(gauss, position[halo_ids], profile[halo_ids] - baseline,
                      p0=(fit_ampl_pre, fit_mu_pre, fit_sigma_pre))
        fit_quantities.update({
                'fit_ampl_halo': fit_ampl_halo,
                'fit_mu_halo': fit_mu_halo,
                'fit_sigma_halo': fit_sigma_halo,
                'var_matrix_halo': var_matrix_halo,
                'halo_sigma': halo_sigma,
            })

    return fit_quantities

def make_fiterror_function(x, disp_profile, x_profile):
    '''Arguments:
        x: 1D array with regularly spaced positions
        disp_profile: 1D array with dispersive profile D_x * delta along x
        x_profile: 1D array with measured horizontal profile from wire scanner
    '''
    def fiterror_function(sigma):
        '''Arguments:
            sigma: standard deviation of assumed Gaussian distribution
        '''
        betatron_profile = np.exp(-x**2 / (2*sigma**2))
        betatron_profile /= np.trapz(betatron_profile, x)
        dx = x[1] - x[0]
```

```
        x_profile_estimate = np.convolve(betatron_profile, disp_profile, 'same') * dx
        return x_profile - x_profile_estimate
    return fiterror_function

def get_v_emit_geo(data, data_idx, ctime_idx=0, beta_y=11.83):
    '''beta_y for PS 85.V is 11.83m'''
    pos = 1e-6 * data['v_position'][ctime_idx][data_idx]
    prof = data['v_profile'][ctime_idx][data_idx]
    fit_quantities = fit_gauss(pos, prof)
    return fit_quantities['fit_sigma_core']**2 / beta_y, fit_quantities

def get_h_emit_geo(data, data_idx, ctime_idx=0,
                   beta_x=12.62, disp_x=2.3,
                   n_interp_points=1000,
                   savgol_window=7, savgol_order=1):
    '''beta_x for PS 54.H is 12.62m, the disp_x is 2.3m, correspondingly.'''
    pos = 1e-6 * data['h_position'][ctime_idx][data_idx]
    if ctime_idx == 1:
        # wire scanner is going backwards
        pos *= -1
    prof = data['h_profile'][ctime_idx][data_idx]
    disp_pos = disp_x * data['dp_position'][ctime_idx][data_idx]
    disp_prof = data['dp_profile'][ctime_idx][data_idx] / disp_x

    inner_window = (max(np.amin(disp_pos), np.amin(pos)),
                    min(np.amax(disp_pos), np.amax(pos)))

    fit_quantities = fit_gauss(pos, prof)

    interp_pos = np.linspace(*inner_window, num=n_interp_points)
    interp_prof = np.interp(
        interp_pos, pos - fit_quantities['fit_mu_core'],
        prof - fit_quantities['baseline'])
    interp_fprof = savgol_filter(
        interp_prof, savgol_window, savgol_order)
    interp_disp_prof = np.interp(interp_pos, disp_pos, disp_prof)

    interp_fprof /= np.trapz(interp_fprof, interp_pos)
    interp_disp_prof /= np.trapz(interp_disp_prof, interp_pos)

    fiterror_func = make_fiterror_function(
        x=interp_pos, disp_profile=interp_disp_prof,
        x_profile=interp_fprof
    )

    sigma_betatron = scipy.optimize.leastsq(
        fiterror_func, fit_quantities['fit_sigma_core'])[0][0]
    return sigma_betatron**2 / beta_x, fit_quantities
```

```python
import statsmodels.api as sm
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from scipy.stats import norm

def fit_with_ci(xdata, ydata, ytobefit, conf_level_sigma=1,
                verbose_results=False, add_constant=False):
    '''Calculate fit and confidence intervals for an ordinary linear
    regression model.
    Arguments:
        xdata: independent variable data
        ydata: dependent variable data
        ytobefit: tuple with entries for each functional dependency,
            e.g. (xdata, xdata**2, np.sin(xdata))
        conf_level_sigma: number of normal sigma for confidence level
            e.g. 1 corresponds to 68.3%, 2 corresponds to 95.4% c.l.
        verbose_results: prints regression results
        add_constant: adds a variable constant to the fit model
    Return 5-tuple of sorted xdata, fitted ydata and respective
    lower and upper confidence level values, and finally the
    statsmodels model fit results object.
    '''
    perm = np.argsort(xdata)
    # separate tuple entries for each functional dependency:
    X = np.column_stack(ytobefit)
    if add_constant:
        X = sm.add_constant(X)
    model = sm.OLS(ydata, X)
    results = model.fit()
    if verbose_results:
        print (results.summary())

    alpha = 1 - (norm.cdf(conf_level_sigma) - norm.cdf(-conf_level_sigma))
    # Calculate confidence interval lines
    prstd, iv_l, iv_u = wls_prediction_std(results, alpha=alpha)
    return (xdata[perm], results.fittedvalues[perm],
            iv_l[perm], iv_u[perm], results)
```

```
/user/cpsop/adrian/anaconda2/lib/python2.7/site-packages/statsmodels/compat/pandas.py:56: Future
  from pandas.core import datetools
```

2 Overview single case

- longitudinal phase space: tomo
- vertical emittance: 85.V wire scanner
- intensity: BCT

- dipolar tune: BPM
- quadrupolar tune: BQL72_Q

```
In [28]: files = sorted(glob('./2018.10.1*.mat'))
```

```
In [73]: fb = loadmat('./2018.10.18.13.59.55.085.mat', squeeze_me=True,
                                                         struct_as_record=False)['myDataStruct']
```

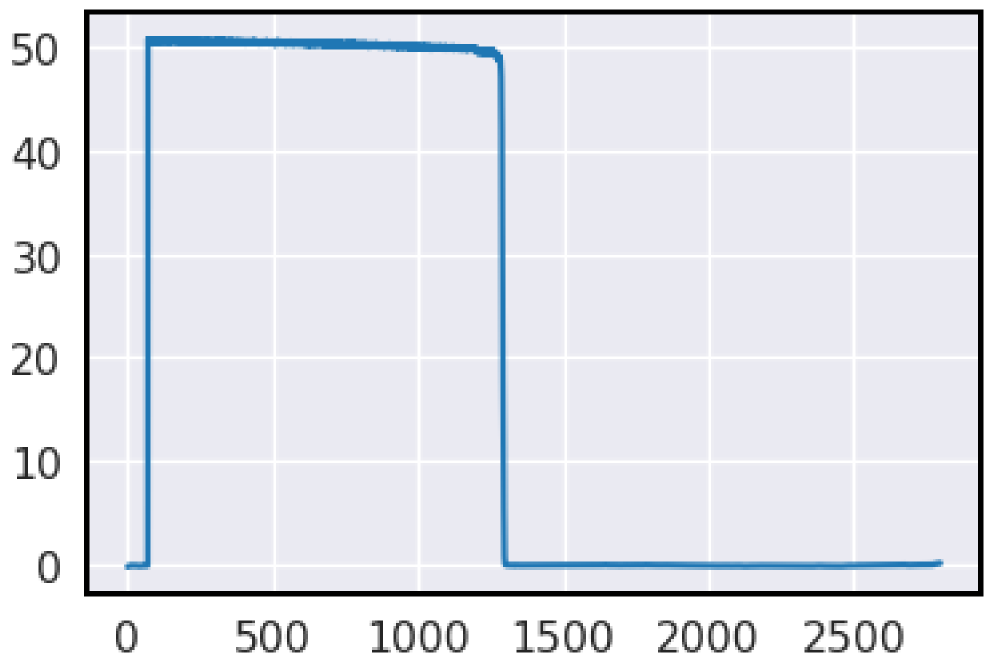
2.1 BCT

```
In [74]: Nin = extract_intensity(fb, ctime=173)
         Nout = extract_intensity(fb, ctime=1300)
         loss = (Nin - Nout) / Nin
         loss
```

```
Out[74]: 0.018481063202694648
```

```
In [75]: plt.plot(fb.PR_BCT_ST.Samples.value.samples)
```

```
Out[75]: [<matplotlib.lines.Line2D at 0x7f14f0331890>]
```



2.2 Dipolar tunes

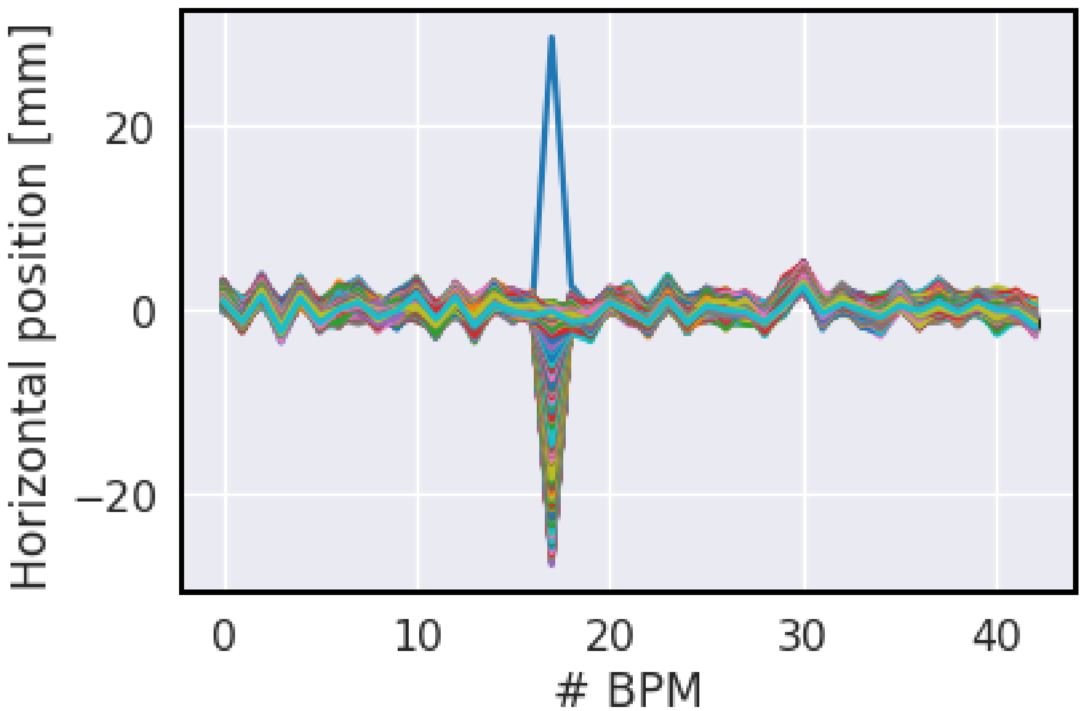
```
In [76]: Qx = fb.PR_BQS72.SamplerAcquisition.value.estimatedTuneH[0]
         Qy = fb.PR_BQS72.SamplerAcquisition.value.estimatedTuneV[0]
         Qx, Qy
```

Out[76]: (nan, 0.08294918770055877)

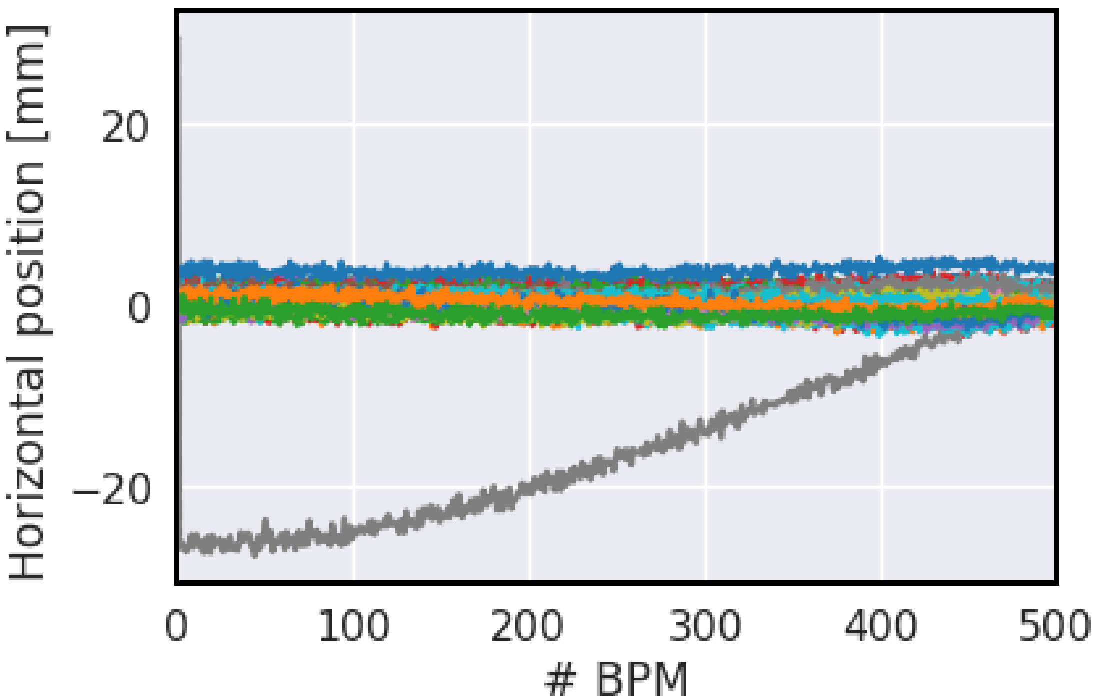
In [77]: fb.PR_BPM.AcquisitionTrajectoryBBB.value.acqState

Out[77]: array([], dtype=float64)

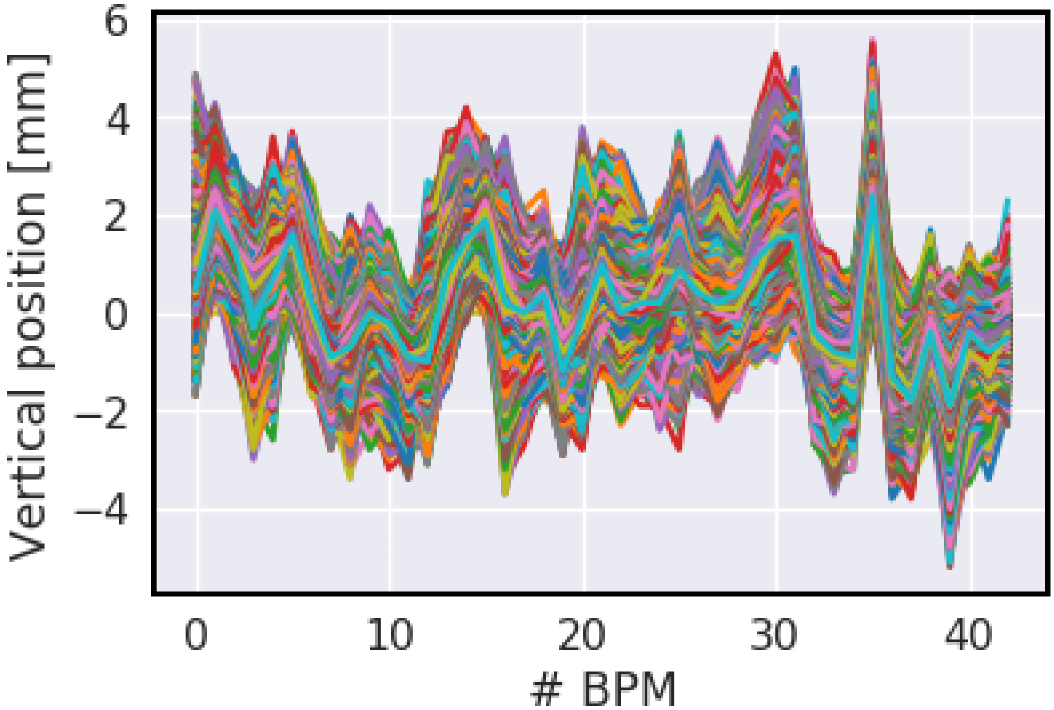
In [78]:
```
plt.plot(fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[:43] *
         10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent * 1e3);
plt.ylabel('Horizontal position [mm]')
plt.xlabel('# BPM');
```



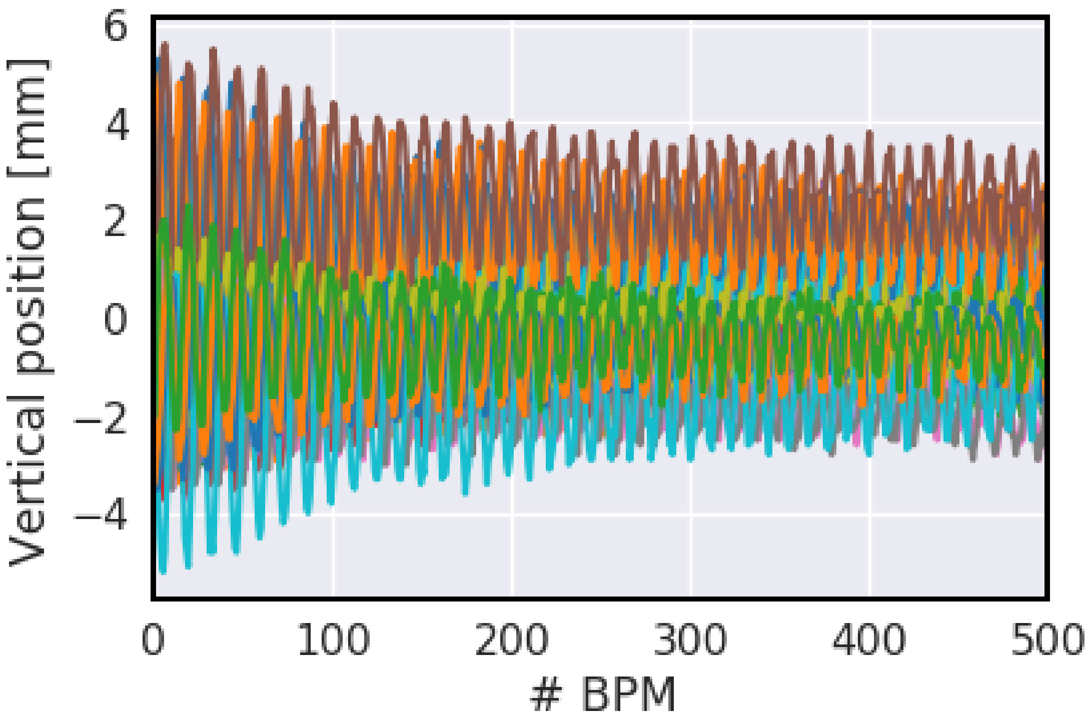
In [119]:
```
plt.plot(fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[:43].T *
         10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent * 1e3);
plt.ylabel('Horizontal position [mm]')
plt.xlim(0, 500)
plt.xlabel('# BPM');
```




```
In [79]: plt.plot(fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[43:] *
                  10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent * 1e3);
         plt.ylabel('Vertical position [mm]')
         plt.xlabel('# BPM');
```



In [123]:
```
plt.plot((fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[43:].T *
          10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent * 1e3));
plt.ylabel('Vertical position [mm]')
plt.xlim(0, 500)
plt.xlabel('# BPM');
```



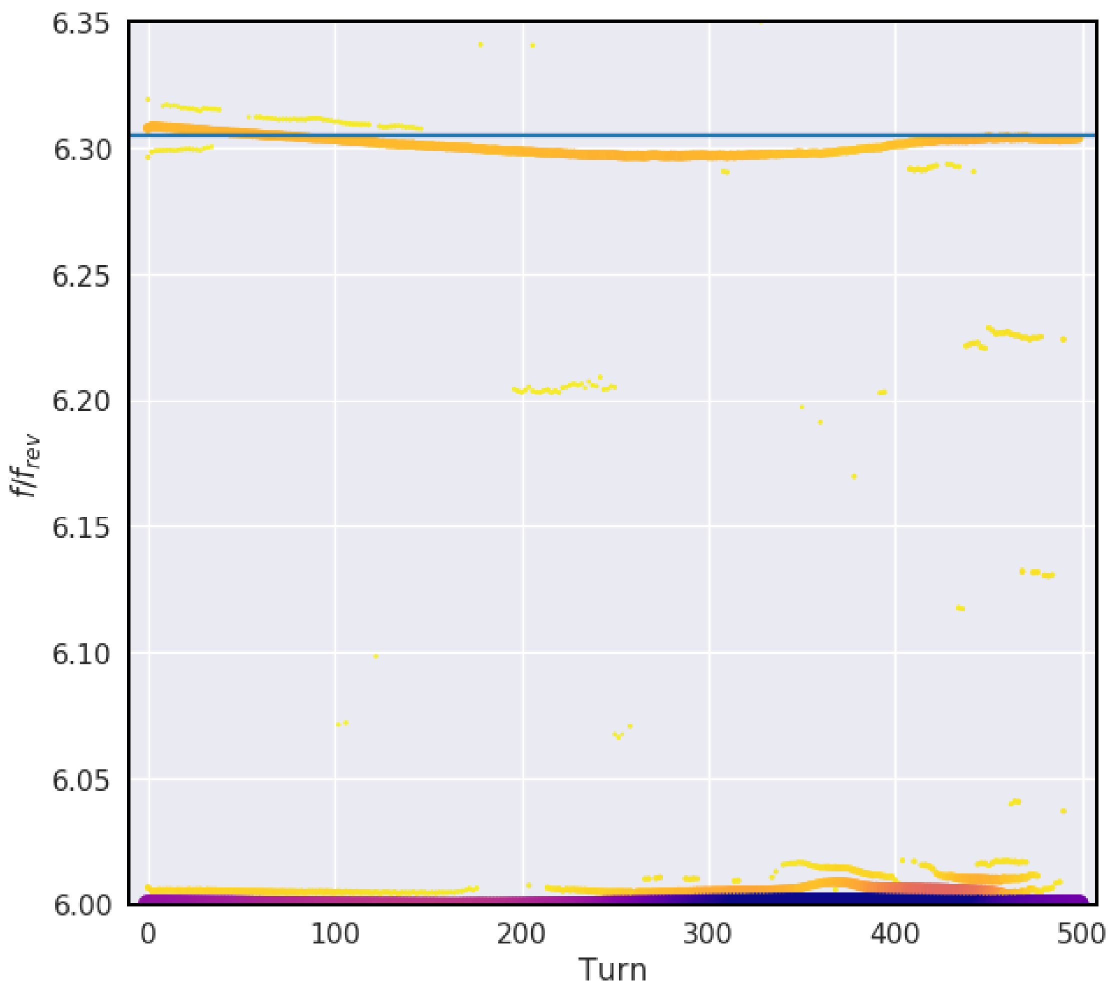

```
In [124]: bpm_turns = len(fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[0])
```

2.2.1 horizontal

```
In [158]: bpm_x = np.empty(43 * bpm_turns, dtype=np.float64)

          unit = 10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent

          for i in xrange(43):
              bpm_x[i::43] = fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[i] * unit

In [159]: bpm_pos = bpm_x.copy()
          bpm_pos_Q = hilbert(bpm_pos)

In [160]: n_fft = 128 * 43
          n_lines = 600

          # assert n_fft < 600

          plot_every = 2 * 43 # every second turn!
          turn_start = np.arange(0, 500 * 43, plot_every)

          amp_x = np.zeros((len(turn_start), n_lines))
          omega_x = np.zeros((len(turn_start), n_lines))
```

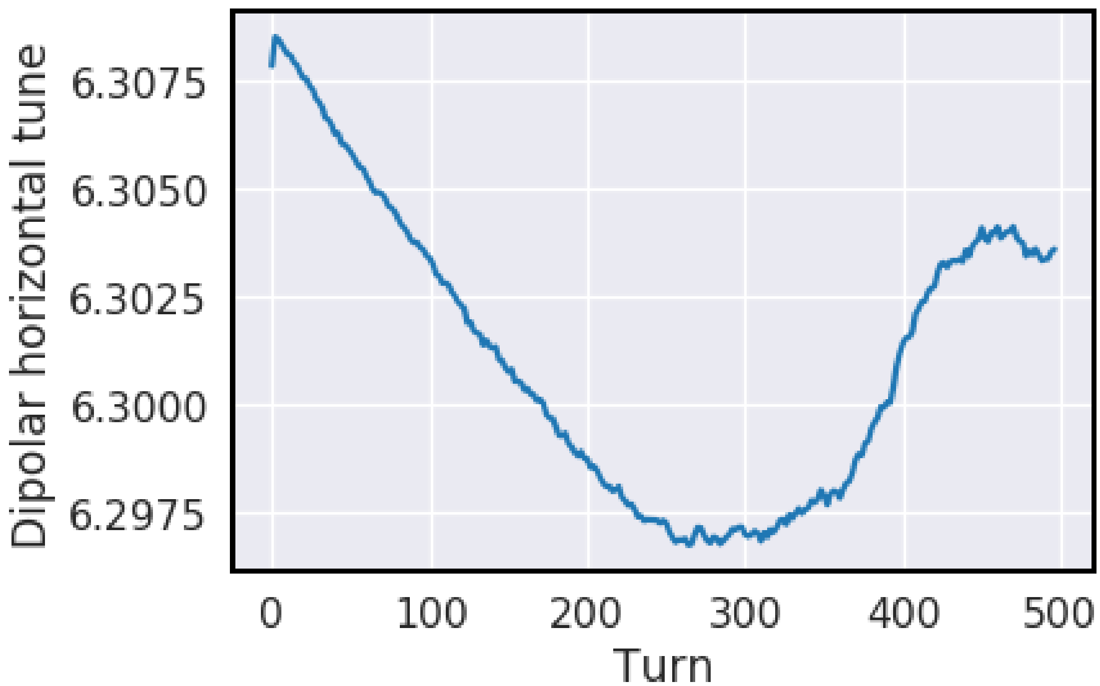
```
for j, start in enumerate(turn_start):
    end = start + n_fft

    x = bpm_pos[start:end]
    xp = np.zeros_like(x) #bpm_pos_Q[start:end]

    SX = PySussix.Sussix()

    SX.sussix_inp(nt1=1, nt2=len(x), idam=2, ir=0,
                  tunex=(6 + Qx) / 43., tuney=(6 + Qy) / 43.)
    SX.sussix(x, xp, x, xp, x, xp)

    ox = np.abs(SX.ox)[:n_lines] * 43
    ax = SX.ax[:n_lines]
    omega_x[j,:], amp_x[j,:] = ox[::-1], ax[::-1]

# re_x = np.array([o[-1] for o in omega_x])

XX, YY = np.meshgrid(turn_start / 43, omega_x[0])

amp_x = np.array([a/np.amax(a) for a in amp_x])
amp_x = np.log(amp_x+1)
```

In [162]:
```
fig, axes = plt.subplots(1, figsize=(9, 8), tight_layout=True)

sc = plt.scatter(XX.T, omega_x, s=256*amp_x,
                 c=amp_x, cmap=plt.cm.plasma_r, lw=.1)

plt.ylim(6, 6.35)
plt.xlim(-10, turn_start[-1] / 43 + 10)

plt.axhline(6.305)

plt.xlabel('Turn')
plt.ylabel(r'$f/f_{rev}$');
```




```
In [204]: q_mask_lower = 6.05
          q_mask_higher = 6.35
          # take all indices of tunes in chosen tune range between q_mask_lower and q_mask_highe
          t, ai = np.where((q_mask_lower < omega_x) & (omega_x < q_mask_higher))
          # find intersections where indices jump from one turn to the next (t variable)
          mask = np.where(np.diff(t))[0]
          # the entry index in amp_x.flatten() corresponding to the dominant tune within chosen
          ti = ai[mask] + t[mask] * omega_x.shape[1]
          # the actual dominant tune within chosen tune range
          Qx_value = (omega_x).flatten()[ti] # value of tune at
          Qx_turn = turn_start[:-1] / 43. # corresponding turn
```

```
In [215]: plt.plot(Qx_turn, Qx_value)
          plt.xlabel('Turn')
          plt.ylabel('Dipolar horizontal tune');
```



2.2.2 vertical

```
In [205]: bpm_turns = len(fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[43])
```

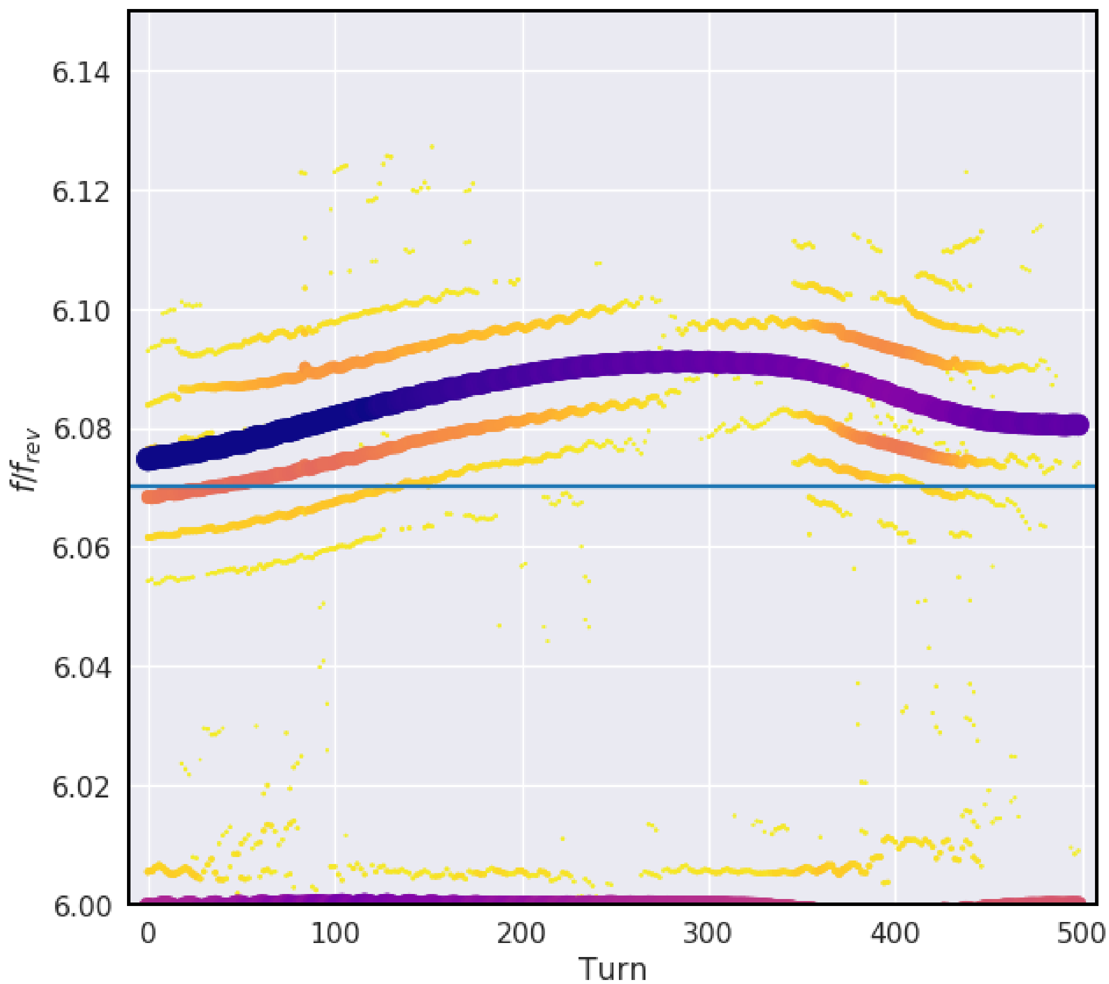
```
In [206]: bpm_y = np.empty(43 * bpm_turns, dtype=np.float64)

          unit = 10**fb.PR_BPM.AcquisitionTrajectoryBBB.value.position_unitExponent

          # for i in xrange(43):
          #     bpm_y[i::43] = fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[43+i] * unit
          bpm_y = (fb.PR_BPM.AcquisitionTrajectoryBBB.value.position[43:].T).flatten() * unit
```

```
In [207]: bpm_pos = bpm_y.copy()
          # bpm_pos_Q = hilbert(bpm_pos)
```

```
In [208]: n_fft = 128 * 43
          n_lines = 600

          # assert n_fft < 600

          plot_every = 2 * 43 # every second turn!
          turn_start = np.arange(0, 500 * 43, plot_every)

          amp_x = np.zeros((len(turn_start), n_lines))
          omega_x = np.zeros((len(turn_start), n_lines))
```

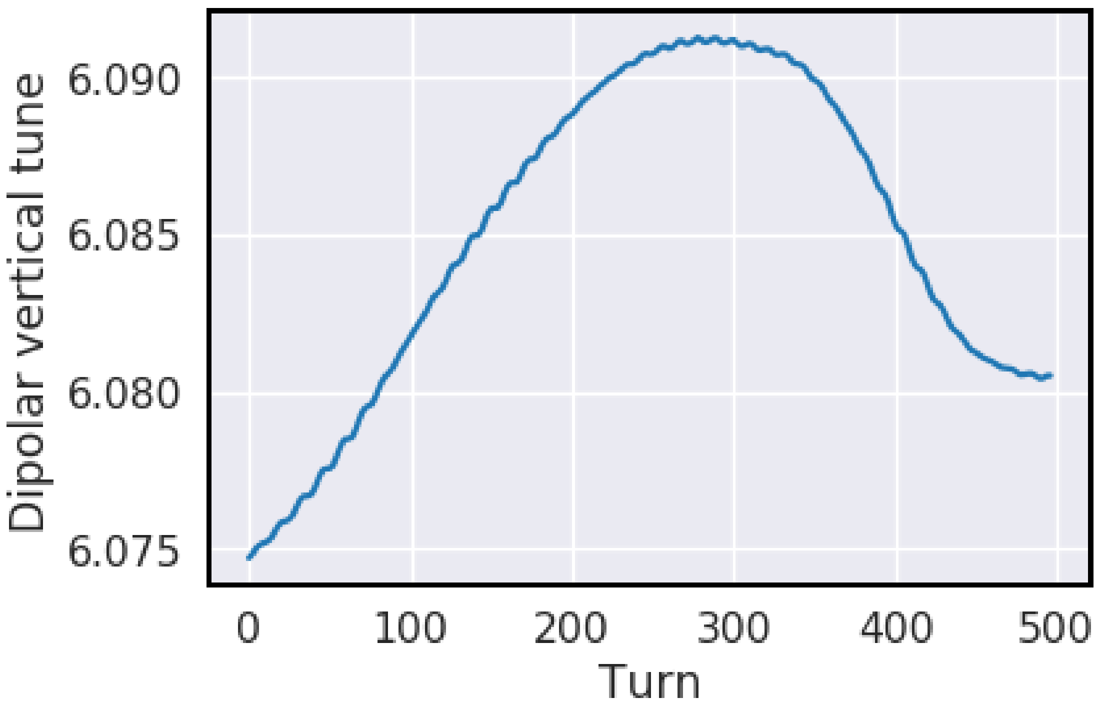
```python
for j, start in enumerate(turn_start):
    end = start + n_fft

    x = bpm_pos[start:end]
    xp = np.zeros_like(x) #bpm_pos_Q[start:end]

    SX = PySussix.Sussix()

    SX.sussix_inp(nt1=1, nt2=len(x), idam=2, ir=0,
                  tunex=(6 + Qx) / 43., tuney=(6 + Qy) / 43.)
    SX.sussix(x, xp, x, xp, x, xp)

    ox = np.abs(SX.ox)[:n_lines] * 43
    ax = SX.ax[:n_lines]
    omega_x[j,:], amp_x[j,:] = ox[::-1], ax[::-1]

# re_x = np.array([o[-1] for o in omega_x])

XX, YY = np.meshgrid(turn_start / 43, omega_x[0])

amp_x = np.array([a/np.amax(a) for a in amp_x])
amp_x = np.log(amp_x+1)
```

In [209]:
```python
fig, axes = plt.subplots(1, figsize=(9, 8), tight_layout=True)

sc = plt.scatter(XX.T, omega_x, s=256*amp_x,
                 c=amp_x, cmap=plt.cm.plasma_r, lw=.1)

plt.ylim(6, 6.15)
plt.xlim(-10, turn_start[-1] / 43 + 10)

plt.axhline(6.07)

plt.xlabel('Turn')
plt.ylabel(r'$f/f_{rev}$');
```



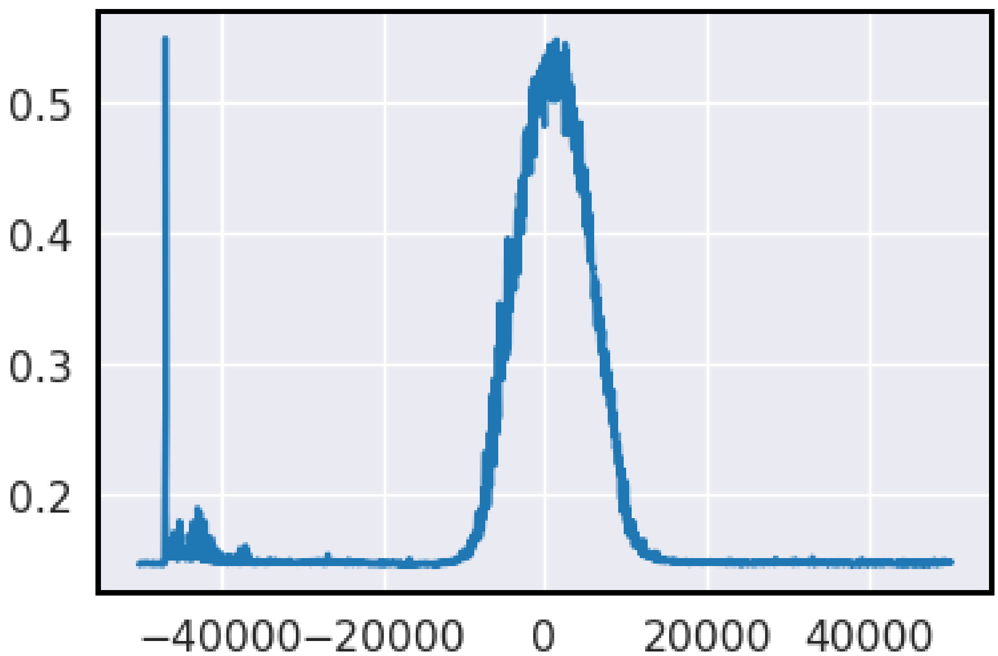

```
In [210]: q_mask_lower = 6.02
          q_mask_higher = 6.15
          # take all indices of tunes in chosen tune range between q_mask_lower and q_mask_highe
          t, ai = np.where((q_mask_lower < omega_x) & (omega_x < q_mask_higher))
          # find intersections where indices jump from one turn to the next (t variable)
          mask = np.where(np.diff(t))[0]
          # the entry index in amp_x.flatten() corresponding to the dominant tune within chosen
          ti = ai[mask] + t[mask] * omega_x.shape[1]
          # the actual dominant tune within chosen tune range
          Qy_value = (omega_x).flatten()[ti] # value of tune at
          Qy_turn = turn_start[:-1] / 43. # corresponding turn
```

```
In [218]: plt.plot(Qy_turn, Qy_value)
          plt.xlabel('Turn')
          plt.ylabel('Dipolar vertical tune');
```



2.2.3 wire scanner

```
In [90]: plt.plot(
             fb.PR_BWS_85_V_ROT.Acquisition.value.projPositionSet1,
             fb.PR_BWS_85_V_ROT.Acquisition.value.projDataSet1
         )

Out[90]: [<matplotlib.lines.Line2D at 0x7f14ee134710>]
```




```
In [91]: Ekin = 1.4e9 * e
         gamma = 1 + Ekin / (m_p * c**2)
         beta = np.sqrt(1 - gamma**-2)
         rp = e**2 / (4 * np.pi * epsilon_0 * m_p * c**2)
         C = 100 * 2 * np.pi
         betagamma = beta * gamma
```

```
In [92]: pos = 1e-6 * fb.PR_BWS_85_V_ROT.Acquisition.value.projPositionSet1
         prof = fb.PR_BWS_85_V_ROT.Acquisition.value.projDataSet1
         fit_quantities = fit_gauss(pos, prof)
         print (
             fit_quantities['fit_sigma_core']**2 / 11.83 * betagamma,
             fit_quantities['fit_sigma_pre']**2 / 11.83 * betagamma
         )
```

```
4.346273021426859e-06 3.962417936748932e-06
```

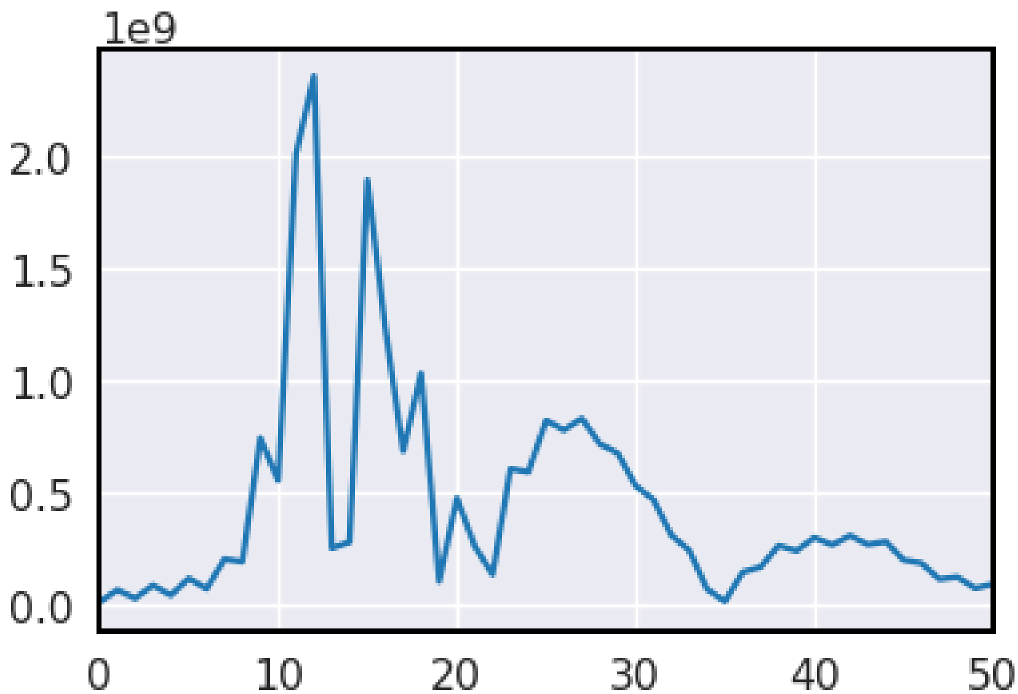
2.3 QPU

```
In [219]: qpu_data = fb.PR_BQL72_Q.Acquisition.value.rawDataH
          qpu_data_H = hilbert(qpu_data)
          qpu_data_Q = qpu_data_H.imag
```

```
In [223]: plt.plot(np.sqrt(qpu_data**2 + qpu_data_Q**2))
          plt.xlim(0, 50)
```

```
/nfs/cs-ccr-nfs6/vol28/u1/cpsop/adrian/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.
  """Entry point for launching an IPython kernel.
```

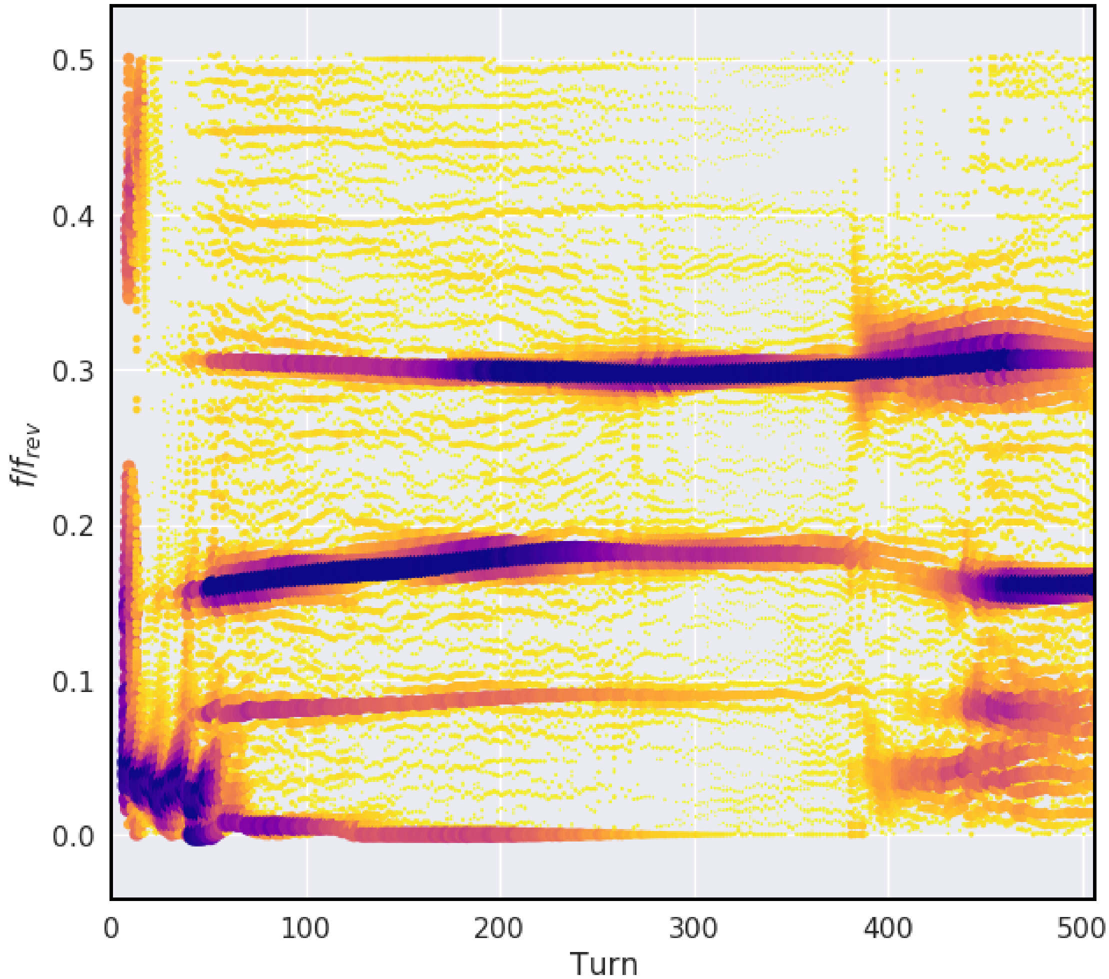
Out[223]: (0, 50)



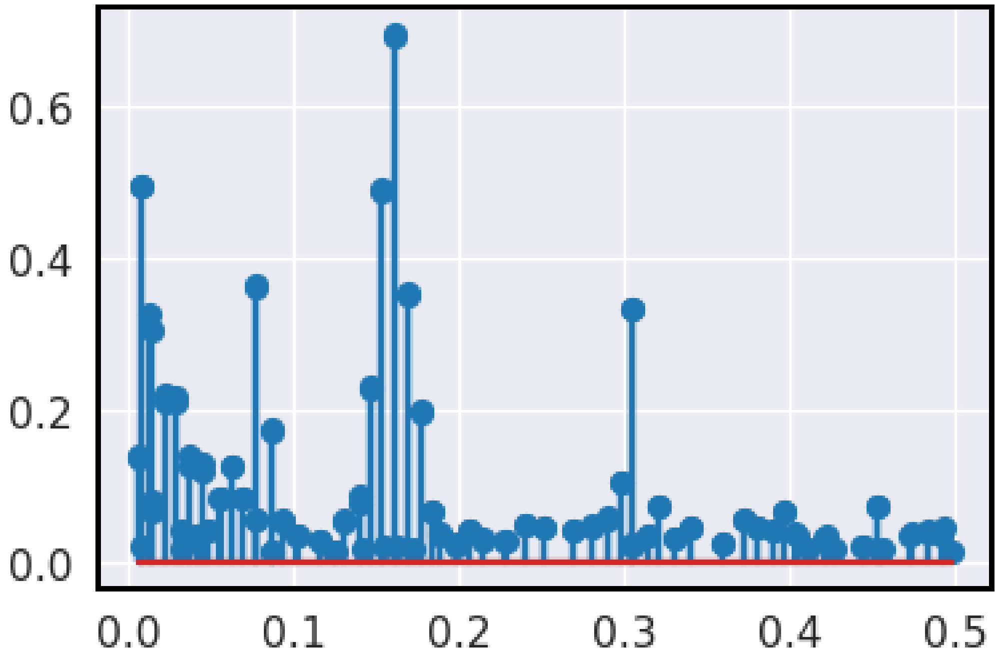
In [225]:
```
n_fft = 128#2048
n_lines = 128

assert n_fft < 600

turn_beam_inj = np.where(np.sqrt(qpu_data**2 + qpu_data_Q**2) > 3e8)[0][0]

# turn_start = np.arange(0, len(qpu_data) - n_fft, n_fft//8)
turn_start = np.arange(turn_beam_inj, turn_beam_inj + 500, 2)

amp_x = np.zeros((len(turn_start), n_lines))
omega_x = np.zeros((len(turn_start), n_lines))

for j, start in enumerate(turn_start):
    end = start + n_fft

    x = qpu_data[start:end]
    xp = np.zeros_like(x) #qpu_data_Q[start:end]
```

```
    SX = PySussix.Sussix()

    SX.sussix_inp(nt1=1, nt2=len(x), idam=2, ir=0,
                  tunex=Qx, tuney=Qy)
    SX.sussix(x, xp, x, xp, x, xp)

    ox = np.abs(SX.ox)[:n_lines]
    ax = SX.ax[:n_lines]
    omega_x[j,:], amp_x[j,:] = ox[::-1], ax[::-1]

XX, YY = np.meshgrid(turn_start, omega_x[0])

amp_x = np.array([a/np.amax(a) for a in amp_x])
amp_x = np.log(amp_x+1)
```

In [230]:
```
fig, axes = plt.subplots(1, figsize=(9, 8), tight_layout=True)

sc = plt.scatter(XX.T, omega_x, s=256*amp_x,
                 c=amp_x, cmap=plt.cm.plasma_r, lw=.1)

# plt.ylim(-3.5, 2)
plt.xlim(-n_fft, len(qpu_data))

plt.xlim(0, turn_start[-1]) #turn_beam_inj

plt.xlabel('Turn')
plt.ylabel(r'$f/f_{rev}$');
```



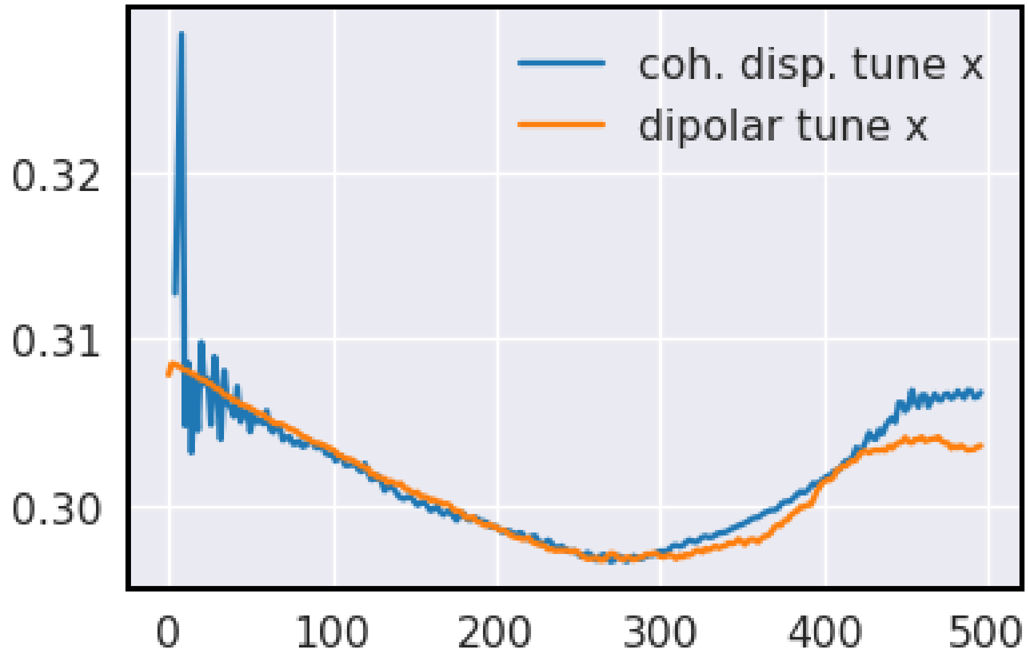

```
In [235]: plt.stem(omega_x[28], amp_x[28])

Out[235]: <Container object of 3 artists>
```



In [269]:
```
q_mask_lower = 0.25
q_mask_higher = 0.33
# take all indices of tunes in chosen tune range between q_mask_lower and q_mask_highe
t, ai = np.where((q_mask_lower < omega_x) & (omega_x < q_mask_higher))
# find intersections where indices jump from one turn to the next (t variable)
mask = np.where(np.diff(t) > 0)[0]
# the entry index in amp_x.flatten() corresponding to the dominant tune within chosen
ti = ai[mask] + t[mask] * omega_x.shape[1]
# the actual dominant tune within chosen tune range
Qxdisp_value = (omega_x).flatten()[ti] # value of tune at
Qxdisp_turn = t[mask] * np.diff(turn_start)[0] # corresponding turn
```

In [270]:
```
plt.plot(Qxdisp_turn, Qxdisp_value, label='coh. disp. tune x')
plt.plot(Qx_turn, Qx_value - 6, label='dipolar tune x')
plt.legend()
```

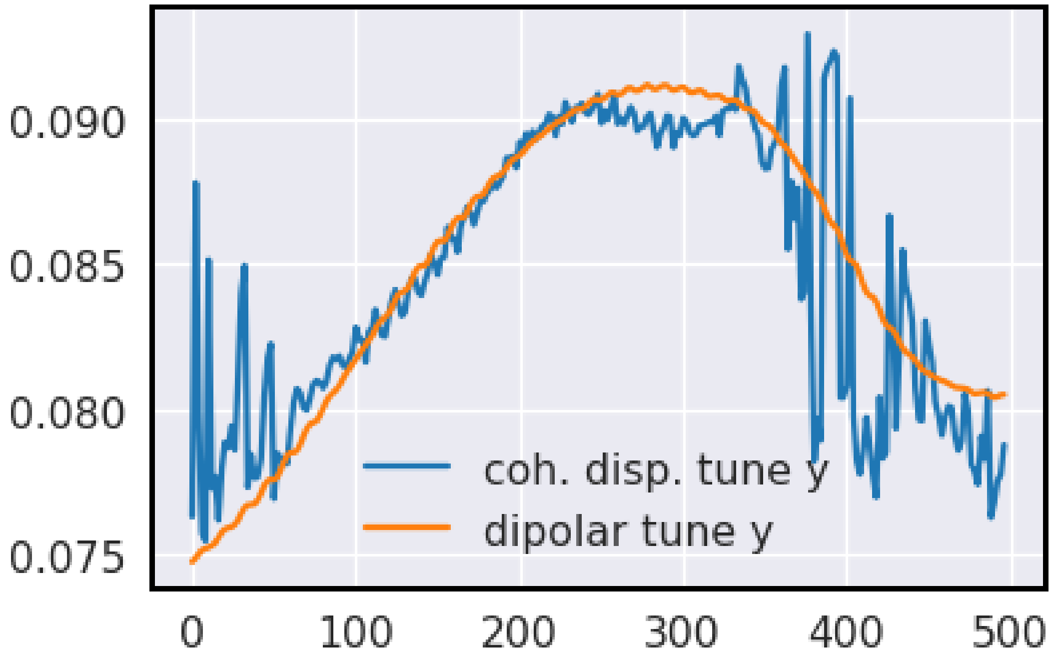
Out[270]:
```
<matplotlib.legend.Legend at 0x7f14f0d75490>
```




```
In [273]: q_mask_lower = 0.075
          q_mask_higher = 0.093
          # take all indices of tunes in chosen tune range between q_mask_lower and q_mask_highe
          t, ai = np.where((q_mask_lower < omega_x) & (omega_x < q_mask_higher))
          # find intersections where indices jump from one turn to the next (t variable)
          mask = np.where(np.diff(t))[0]
          # the entry index in amp_x.flatten() corresponding to the dominant tune within chosen
          ti = ai[mask] + t[mask] * omega_x.shape[1]
          # the actual dominant tune within chosen tune range
          Qydisp_value = (omega_x).flatten()[ti] # value of tune at
          Qydisp_turn = t[mask] * np.diff(turn_start)[0] # corresponding turn
```

```
In [274]: plt.plot(Qydisp_turn, Qydisp_value, label='coh. disp. tune y')
          plt.plot(Qy_turn, Qy_value - 6, label='dipolar tune y')
          plt.legend()
```

```
Out[274]: <matplotlib.legend.Legend at 0x7f14ed1c8b10>
```



3 loop

```
In [44]: class Measurement(object):
             '''A measurement of a single shot with all related evaluated attributes.'''
             Ekin = 1.4e9 * e
             gamma = 1 + Ekin / (m_p * c**2)
             beta = np.sqrt(1 - gamma**-2)
             rp = e**2 / (4 * np.pi * epsilon_0 * m_p * c**2)
             C = 100 * 2 * np.pi
             betagamma = beta * gamma

             beta_y_85V = 11.83

             def __init__(self, filename):
                 self.filename = filename
                 myDataStruct = loadmat(filename, squeeze_me=True,
                                        struct_as_record=False)['myDataStruct']
                 self.loss = self.get_loss(myDataStruct)
                 self.Qx = self.get_dip_tune_x(myDataStruct)
                 self.Qy = self.get_dip_tune_y(myDataStruct)
                 self.epsn_y_gauss, self.epsn_y_core = self.get_vert_emit(myDataStruct)
                 self.ws_time = self.get_ws_acq_delay(myDataStruct)

             def get_loss(self, myDataStruct):
```

```
            Nin = extract_intensity(myDataStruct, ctime=173)
            Nout = extract_intensity(myDataStruct, ctime=1250)
            return (Nin - Nout) / Nin

        def get_dip_tune_x(self, myDataStruct):
            return myDataStruct.PR_BQS72.SamplerAcquisition.value.estimatedTuneH[0]

        def get_dip_tune_y(self, myDataStruct):
            return myDataStruct.PR_BQS72.SamplerAcquisition.value.estimatedTuneV[0]

        def get_vert_emit(self, myDataStruct):
            pos = 1e-6 * myDataStruct.PR_BWS_85_V_ROT.Acquisition.value.projPositionSet1
            prof = myDataStruct.PR_BWS_85_V_ROT.Acquisition.value.projDataSet1
            fit_quantities = fit_gauss(pos, prof)
            return (
                fit_quantities['fit_sigma_pre']**2 / self.beta_y_85V * self.betagamma,
                fit_quantities['fit_sigma_core']**2 / self.beta_y_85V * self.betagamma
            )

        def get_ws_acq_delay(self, myDataStruct):
            return myDataStruct.PR_BWS_85_V_ROT.Acquisition.value.acqDelay
```

```
In [47]: data = []
         for f in files:
             try:
                 data.append(vars(Measurement(f)))
             except:
                 pass
```

```
In [48]: df = pandas.DataFrame(data=data)#[vars(Measurement(f)) for f in files])
```

```
In [49]: df
```

Out[49]:

	Qx	Qy	epsn_y_core	epsn_y_gauss	filename \
0	NaN	0.035325	0.000002	0.000002	./2018.10.18.13.19.35.883.mat
1	NaN	0.037992	0.000002	0.000002	./2018.10.18.13.20.43.095.mat
2	NaN	0.035270	0.000003	0.000003	./2018.10.18.13.21.50.295.mat
3	NaN	0.035274	0.000002	0.000002	./2018.10.18.13.22.57.490.mat
4	NaN	0.035430	0.000003	0.000002	./2018.10.18.13.24.04.694.mat
5	NaN	0.036335	0.000003	0.000003	./2018.10.18.13.25.11.893.mat
6	NaN	0.036177	0.000003	0.000003	./2018.10.18.13.26.19.091.mat
7	NaN	0.036072	0.000003	0.000003	./2018.10.18.13.27.26.289.mat
8	NaN	0.039954	0.000004	0.000004	./2018.10.18.13.28.33.494.mat
9	NaN	0.040321	0.000004	0.000003	./2018.10.18.13.29.40.689.mat
10	NaN	0.040018	0.000004	0.000004	./2018.10.18.13.30.47.892.mat
11	NaN	0.044170	0.000004	0.000004	./2018.10.18.13.31.55.087.mat
12	NaN	0.043206	0.000004	0.000004	./2018.10.18.13.33.02.288.mat
13	NaN	0.043985	0.000004	0.000004	./2018.10.18.13.34.09.485.mat
14	NaN	0.048799	0.000004	0.000004	./2018.10.18.13.35.16.685.mat

15	NaN	0.048852	0.000004	0.000004	./2018.10.18.13.36.23.893.mat
16	NaN	0.047792	0.000004	0.000004	./2018.10.18.13.37.31.092.mat
17	NaN	0.052677	0.000004	0.000004	./2018.10.18.13.38.38.295.mat
18	NaN	0.053084	0.000004	0.000004	./2018.10.18.13.39.45.486.mat
19	NaN	0.053478	0.000004	0.000004	./2018.10.18.13.40.52.685.mat
20	NaN	0.057577	0.000004	0.000004	./2018.10.18.13.41.59.886.mat
21	NaN	0.057722	0.000004	0.000004	./2018.10.18.13.43.07.091.mat
22	NaN	0.057888	0.000004	0.000004	./2018.10.18.13.44.14.290.mat
23	NaN	0.062940	0.000004	0.000004	./2018.10.18.13.45.21.484.mat
24	NaN	0.063015	0.000004	0.000004	./2018.10.18.13.46.28.690.mat
25	NaN	0.062416	0.000004	0.000004	./2018.10.18.13.47.35.896.mat
26	NaN	0.067814	0.000004	0.000004	./2018.10.18.13.48.43.182.mat
27	NaN	0.068465	0.000004	0.000004	./2018.10.18.13.49.50.283.mat
28	NaN	0.068836	0.000004	0.000004	./2018.10.18.13.50.57.494.mat
29	NaN	0.072626	0.000004	0.000004	./2018.10.18.13.52.04.685.mat
30	NaN	0.072938	0.000004	0.000004	./2018.10.18.13.53.11.887.mat
31	NaN	0.072448	0.000004	0.000004	./2018.10.18.13.54.19.088.mat
32	NaN	0.077919	0.000004	0.000004	./2018.10.18.13.55.26.290.mat
33	NaN	0.077837	0.000004	0.000004	./2018.10.18.13.56.33.486.mat
34	NaN	0.077799	0.000004	0.000004	./2018.10.18.13.57.40.685.mat
35	NaN	0.082917	0.000004	0.000004	./2018.10.18.13.58.47.883.mat
36	NaN	0.082949	0.000004	0.000004	./2018.10.18.13.59.55.085.mat
37	NaN	0.083408	0.000004	0.000004	./2018.10.18.14.01.02.285.mat
38	NaN	0.087998	0.000005	0.000004	./2018.10.18.14.02.09.483.mat
39	NaN	0.087696	0.000004	0.000004	./2018.10.18.14.03.16.687.mat
40	NaN	0.088040	0.000004	0.000004	./2018.10.18.14.04.23.884.mat
41	NaN	0.093277	0.000005	0.000004	./2018.10.18.14.05.31.090.mat
42	NaN	0.092948	0.000004	0.000004	./2018.10.18.14.06.38.284.mat
43	NaN	0.092919	0.000004	0.000004	./2018.10.18.14.07.45.482.mat
44	NaN	0.098602	0.000004	0.000004	./2018.10.18.14.08.55.455.mat
45	NaN	0.099975	0.000005	0.000004	./2018.10.18.14.10.02.640.mat
46	NaN	0.097601	0.000004	0.000004	./2018.10.18.14.11.09.865.mat
47	NaN	0.105274	0.000005	0.000004	./2018.10.18.14.12.17.039.mat
48	NaN	0.105466	0.000005	0.000004	./2018.10.18.14.13.24.247.mat
49	NaN	0.105310	0.000005	0.000004	./2018.10.18.14.14.31.468.mat
50	NaN	0.109847	0.000005	0.000005	./2018.10.18.14.15.38.649.mat

	loss	ws_time
0	0.881442	175
1	0.863987	185
2	0.005628	172
3	0.027132	175
4	0.011775	185
5	0.007478	172
6	0.005284	175
7	0.061491	185
8	0.006450	172
9	0.007253	175

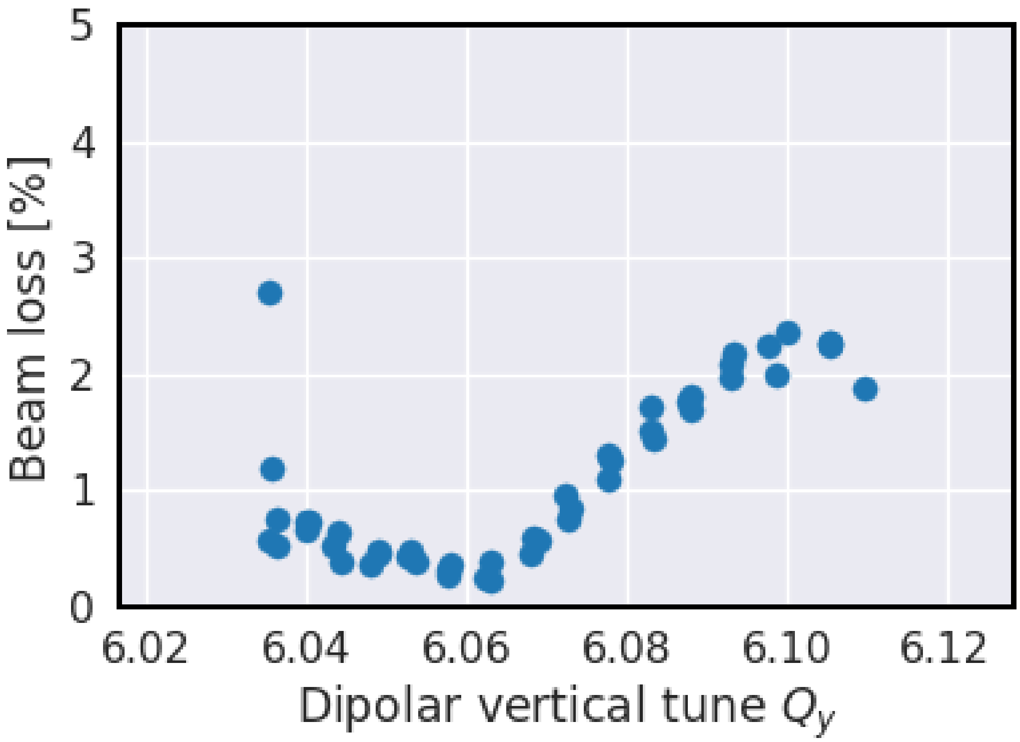
```
10  0.007288      185
11  0.003818      172
12  0.005143      175
13  0.006411      185
14  0.004515      172
15  0.004793      175
16  0.003583      185
17  0.004231      172
18  0.004654      175
19  0.003908      185
20  0.003024      172
21  0.002617      175
22  0.003573      185
23  0.003824      172
24  0.002148      175
25  0.002492      185
26  0.004525      172
27  0.005882      175
28  0.005565      185
29  0.007393      172
30  0.008297      175
31  0.009627      185
32  0.012545      172
33  0.013019      175
34  0.010972      185
35  0.014985      172
36  0.017059      175
37  0.014388      185
38  0.018016      172
39  0.017555      175
40  0.016976      185
41  0.021668      172
42  0.020711      175
43  0.019687      185
44  0.019826      172
45  0.023673      175
46  0.022394      185
47  0.022761      172
48  0.022623      175
49  0.022435      185
50  0.018797      172
```

In [50]:
```
# plt.scatter(6 + df['Qy'], 6 + df['Qx'])
# plt.xlabel('Dipolar vertical tune $Q_y$')
# plt.ylabel('Dipolar horizontal tune $Q_x$');
# plt.ylim(6.2, 6.22)
```

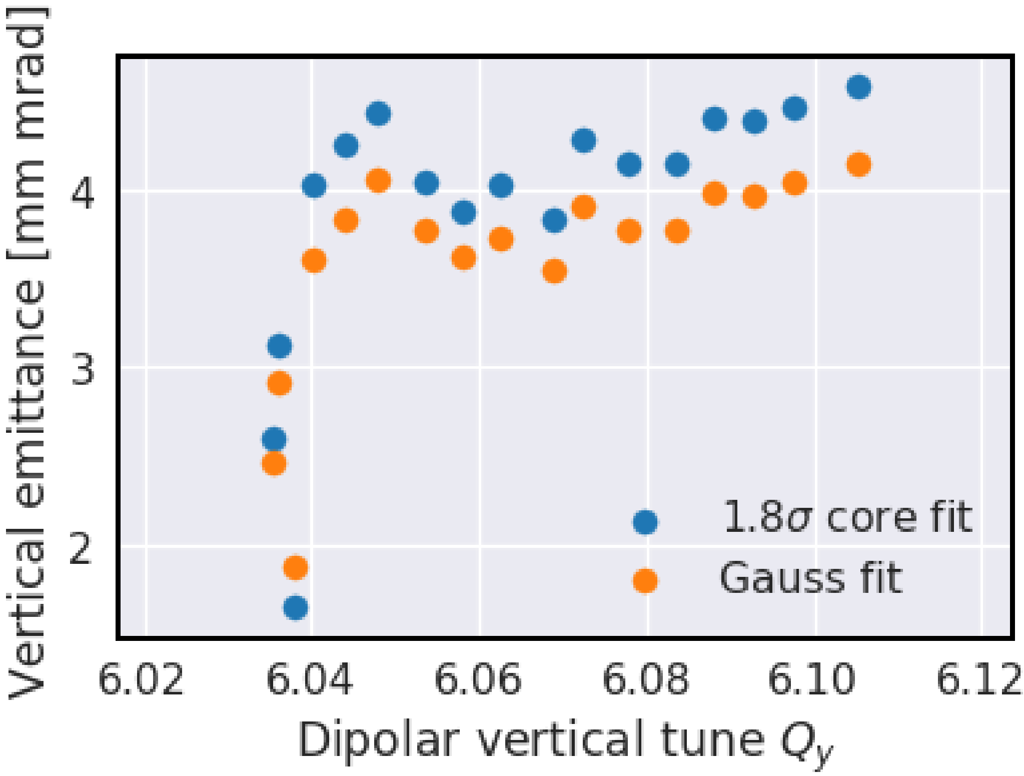
In [53]:
```
plt.scatter(6 + df['Qy'], 100 * df['loss'])
plt.xlabel('Dipolar vertical tune $Q_y$')
```

```
        plt.ylabel('Beam loss [%]');
        plt.ylim(0, 5)
```

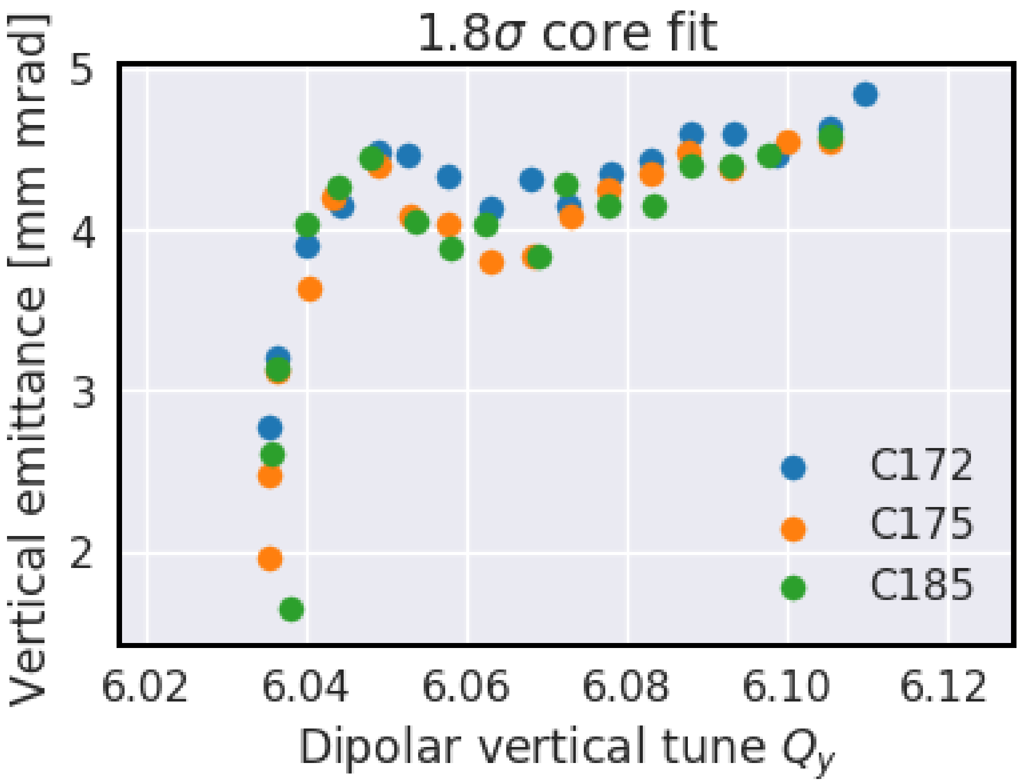
Out[53]: (0, 5)



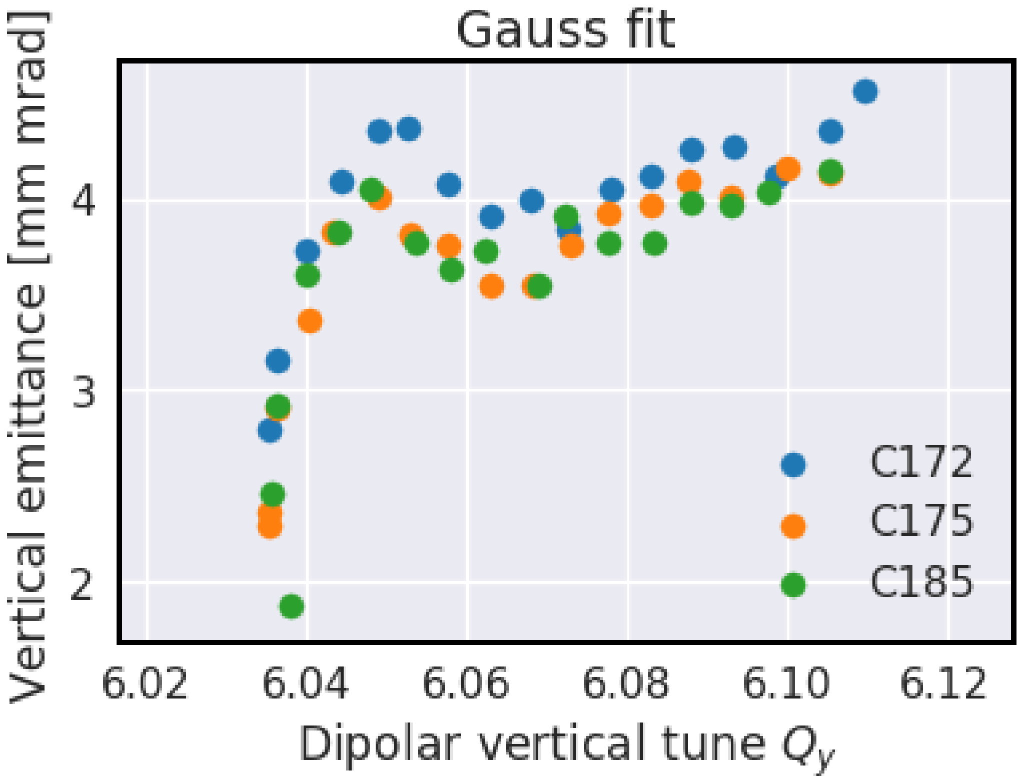

```
In [54]: mask = df['ws_time'] == 185

         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_core'][mask], label=r'$1.8\sigma$ core
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_gauss'][mask], label='Gauss fit')
         plt.legend()
         plt.xlabel('Dipolar vertical tune $Q_y$')
         plt.ylabel('Vertical emittance [mm mrad]');
```



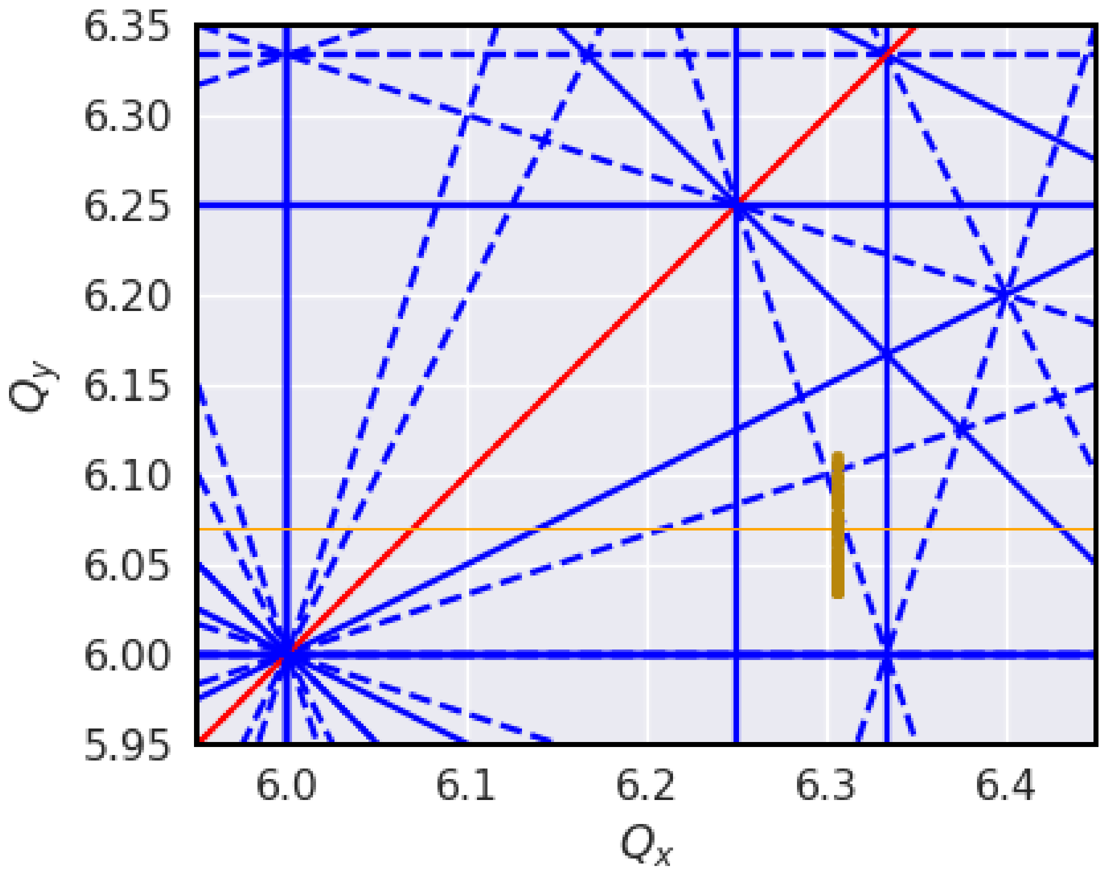

```
In [55]: plt.title(r'$1.8\sigma$ core fit')
         mask = df['ws_time'] == 172
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_core'][mask], label='C172')
         mask = df['ws_time'] == 175
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_core'][mask], label='C175')
         mask = df['ws_time'] == 185
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_core'][mask], label='C185')
         plt.legend()
         plt.xlabel('Dipolar vertical tune $Q_y$')
         plt.ylabel('Vertical emittance [mm mrad]');
```




```
In [56]: plt.title(r'Gauss fit')
         mask = df['ws_time'] == 172
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_gauss'][mask], label='C172')
         mask = df['ws_time'] == 175
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_gauss'][mask], label='C175')
         mask = df['ws_time'] == 185
         plt.scatter(6 + df['Qy'][mask], 1e6 * df['epsn_y_gauss'][mask], label='C185')
         plt.legend()
         plt.xlabel('Dipolar vertical tune $Q_y$')
         plt.ylabel('Vertical emittance [mm mrad]');
```




```
In [57]: from tune_diagram import ResonanceLines

In [61]: fig = plt.figure(figsize=(8,5))
         fig = plt.figure(figsize=(8,5))
         resonances = ResonanceLines((5.95, 6.45), (5.95, 6.35),
                                     range(1, 4+1), 50)
         resonances.plot_resonance(fig)

         plt.gca().set_aspect('equal')

         plt.scatter(6.305 * np.ones_like(df['Qy']), #6 + df['Qx'],
                     6 + df['Qy'], zorder=10, color='darkgoldenrod', marker='.')

         plt.axhline(6.07, color='orange', lw=1, ls='solid')

Out[61]: <matplotlib.lines.Line2D at 0x7f14f1b40510>

<matplotlib.figure.Figure at 0x7f14ef385f50>
```



4 loop for QPU evaluations

In [62]:
```
def plot_qpu_spectrum(filename, n_fft=128, until=500, plot_every=2):
    assert n_fft < 600

    fb = loadmat(filename, squeeze_me=True,
                 struct_as_record=False)['myDataStruct']

    qpu_data = fb.PR_BQL72_Q.Acquisition.value.rawDataH
    qpu_data_H = hilbert(qpu_data)
    qpu_data_Q = qpu_data_H.imag

    turn_beam_inj = np.where(np.sqrt(qpu_data**2 + qpu_data_Q**2) > 3e8)[0][0]

    turn_start = np.arange(turn_beam_inj, turn_beam_inj + until, plot_every)

    amp_x = np.zeros((len(turn_start), n_fft))
    omega_x = np.zeros((len(turn_start), n_fft))
```

```
# not needed
beta_x = 1
beta_y = 1

fig, axes = plt.subplots(1, figsize=(9, 8), tight_layout=True)

for j, start in enumerate(turn_start):
    end = start + n_fft

    x = qpu_data[start:end]
    xp = qpu_data_Q[start:end]

    SX = PySussix.Sussix()

    SX.sussix_inp(nt1=1, nt2=len(x), idam=2, ir=0,
                  tunex=Qx, tuney=Qy)
    SX.sussix(x, xp, x, xp, x, xp)

    ox = np.abs(SX.ox)[:n_fft]
    ax = SX.ax[:n_fft]
    omega_x[j,:], amp_x[j,:] = ox[::-1], ax[::-1]

XX, YY = np.meshgrid(turn_start, omega_x[0])

amp_x = np.array([a/np.amax(a) for a in amp_x])
amp_x = np.log(amp_x+1)

sc = plt.scatter(XX.T, omega_x, s=256*amp_x,
                       c=amp_x, cmap=plt.cm.plasma_r, lw=.1)

# plt.ylim(-3.5, 2)
plt.xlim(-n_fft, len(qpu_data))

plt.xlim(turn_beam_inj, turn_beam_inj + len(turn_start))

plt.xlabel('Turn')
plt.ylabel(r'$f/f_{rev}$');
plt.savefig(filename[:-4] + '_qpuspec.png', bbox_inches='tight', dpi=200)
plt.close()
```

```
In [63]: for f in files:
             try:
                 plot_qpu_spectrum(f)
             except Exception as e:
                 print ('{s} threw an {s}'.format(f, e.message))
```

```
/nfs/cs-ccr-nfs6/vol28/u1/cpsop/adrian/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.
  # This is added back by InteractiveShellApp.init_path()
```

```
/nfs/cs-ccr-nfs6/vol28/u1/cpsop/adrian/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.
  # This is added back by InteractiveShellApp.init_path()
```

```
In [ ]:
```