

Improvements in the error calculation of the action of a kicked beam

Alexander Sherman

Keywords: optics

Summary

This report details a new calculation for the action performed in the optics measurement and correction software. The action of a kicked beam is used to calculate the dynamic aperture and detuning with amplitude. The current method of calculation has a large uncertainty due to the use of all BPMs (including those near interaction points and ones which are malfunctioning) and the model beta function. Instead, only good BPMs are kept and the measured beta function from phase is used, and significant decreases are seen in the relative uncertainty of the action.

Contents

1	Calculation of the action	2
1.1	Theory and calculation	2
1.2	New code	2
1.3	Results of new calculation	3
2	Appendix: Python Code	5

1 Calculation of the action

1.1 Theory and calculation

At a point s along the ideal orbit, the amplitude of oscillation for a particle can be written as [1]

$$A_{x,y}(s) = \sqrt{2J_{x,y}\beta_{x,y}(s)} \quad (1)$$

where $2J_{x,y}$ is the action and $\beta_{x,y}$ is the β function, in the respective x, y planes. It follows that

$$2J_{x,y}(s) = \frac{(A_{x,y}(s))^2}{\beta_{x,y}(s)} \quad (2)$$

so using half of the peak-to-peak values of the beam at a point over many turns (extracted in 'Drive') for the amplitude, along with its beta function at that point, we have a calculated value for the action. To get the best value for the action of the beam, the average can be taken over all BPMs:

$$\langle 2J_{x,y} \rangle = \frac{\sum_{BPMs} \frac{(\frac{1}{2}PK2PK_{x,y}(s))^2}{\beta_{x,y}(s)}}{\#BPMs} \quad (3)$$

The uncertainty is given by

$$\sigma_{2J_{x,y}} = \sqrt{\langle (2J_{x,y})^2 \rangle - \langle 2J_{x,y} \rangle^2} \quad (4)$$

1.2 New code

Previously, the beta function used in equation (3) was from the MADX model. But there are known beta-beating levels of 5% and higher, so one would expect a larger spread in the action values over all BPMs and therefore a higher uncertainty than what measurement should produce.

To reduce this uncertainty, the beta function measured from the phase data at each BPM is used. The shortfall of this is that some BPMs may be giving bad data due to either being near an interaction point or because they are simply malfunctioning. If they are included in the calculation, the uncertainty in the action would be very large. Therefore, BPMs are rejected from the calculation based on their beta-beating value and relative uncertainty.

In detail, the GetLLM code rejects a BPM in the action calculation for the x or y plane if

$$\frac{|\beta_{phase} - \beta_{model}|}{\beta_{model}} > \text{bbthreshold} \quad (5)$$

or

$$\frac{\beta_{phase}}{\sigma_{\beta_{phase}}} > \text{errthreshold} \quad (6)$$

where 'bbthreshold' and 'errthreshold' are set by the user in the command line with "-k (float)" (or "--bbthreshold=(float)") for the beta beating threshold level, and "-e (float)" (or "--errthreshold=(float)") for the relative error threshold. The default level for each threshold is set to 15%.

Another note should be made on how the action is being calculated in GetLLM. Previously, the square root of the action was being calculated as

$$\langle \sqrt{2J_{x,y}} \rangle = \frac{\sum_{BPMs} \frac{(\frac{1}{2}PK2PK_{x,y}(s))}{\sqrt{\beta_{x,y}(s)}}}{\#BPMs} \quad (7)$$

Beam 2 on June 25, 2012					
Time Taken	Plane	% BPM Rej	Max Rel Diff [10^{-3}]	Rel Diff to Model [10^{-3}]	Phase Rel Uncert/ Model Rel Uncert
01_13_11.078	x	12.3	7.47	5.44	0.722
01_13_11.078	y	14.0	5.92	2.83	0.970
01_54_37.397	x	2.94	4.89	0.736	0.906
01_54_37.397	y	13.4	8.68	0.306	0.952
02_06_06.257	x	1.96	8.27	0.433	0.970
02_06_06.257	y	4.31	4.12	0.574	0.836
01_59_05.209	x	1.76	7.05	0.277	0.945
01_59_05.209	y	4.70	6.45	1.19	0.8522

Table 1: Results of action calculation with phase beta function. ‘% BPM Rej’ is the percentage of BPMs rejected at default thresholds. ‘Max Rel Diff’ is the largest absolute relative difference between the action with default thresholds and the action with other thresholds such that less than 1/3 of BPMs were rejected. ‘Rel Diff to Model’ is the absolute relative difference between the action calculated using the model beta function and the action using beta from phase with default threshold values. Finally, ‘Phase Rel Uncert’ is relative uncertainty in the action at default thresholds using beta from phase, and ‘Model Rel Uncert’ is the analogous quantity for when the model beta function was used.

Then $2J_{x,y}$ was set as the square of this value, $(< \sqrt{2J_{x,y}} >)^2$. This method will in general produce a different value than $< 2J_{x,y} >$, although their values have been seen to be well within the uncertainty of one another. The old method of calculation is currently being output, but both calculations are done in the code (see the appendix for the code).

1.3 Results of new calculation

The new method of calculation implemented as described above and output into ‘getkick-phase.out’ was analyzed for several data sets allowing the beta-beating and error thresholds to vary. Ideally, we want to use as many BPMs from the data as possible for the most accurate calculation of the action, so the threshold levels cannot be set too low.

In table 1, we see a summary of the results when measured data sets were used. First note that the new calculation produces an action with a value well within 1% of what the model produced (the relative uncertainty in the action was generally well above 1%). The ‘Max Rel Diff’ column tells us that after less than 1/3 of the BPMs were rejected in the calculation the value of the action did not change by more than 1% when the threshold levels varied. The main change is in the relative uncertainty in the action which decreases in each case, although sometimes by only a small amount. Indeed, the level of reduction in uncertainty varied with the data sets, and sometimes smaller thresholds are necessary to achieve greater reductions. The choice of 15% for the default error thresholds was to take into account that some data sets rejected too many BPMs at lower thresholds such as 10%. Here we see that all data sets used over 85% of their BPMs in the action calculation, which is what we would like to have in general.

For a more qualitative look at how the thresholds affected the relative uncertainty, Figures 1 and 2 show the relative uncertainty versus the beta-beating threshold and error threshold respectively. In the plots, blue data points denote threshold levels which rejected less than 1/3 of the BPMs, while red points denote threshold levels where more than a third of the

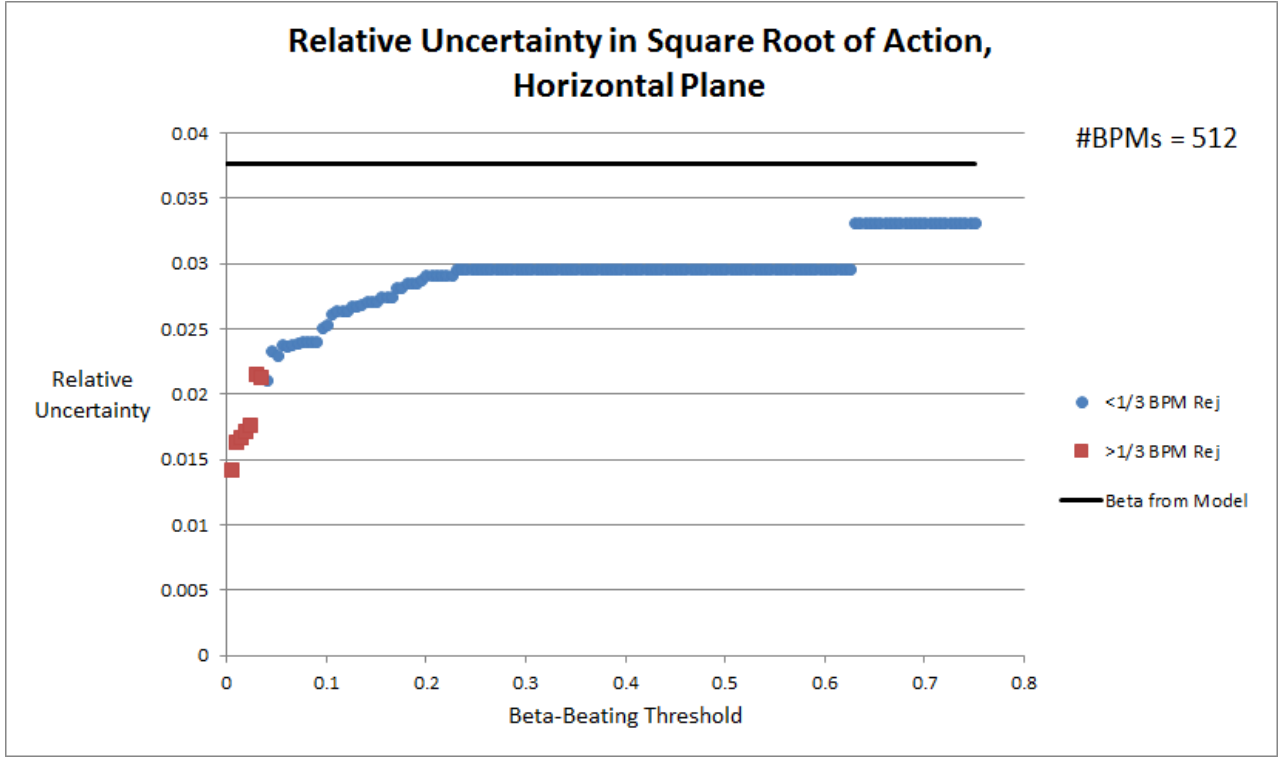


Figure 1: The relative uncertainty in the action as a function of 'bbthreshold', with 'errthreshold' fixed at 15%. The kick experienced here was in the horizontal plane. From the data set Beam2@Turn@2012_06_25@01_13_11_078

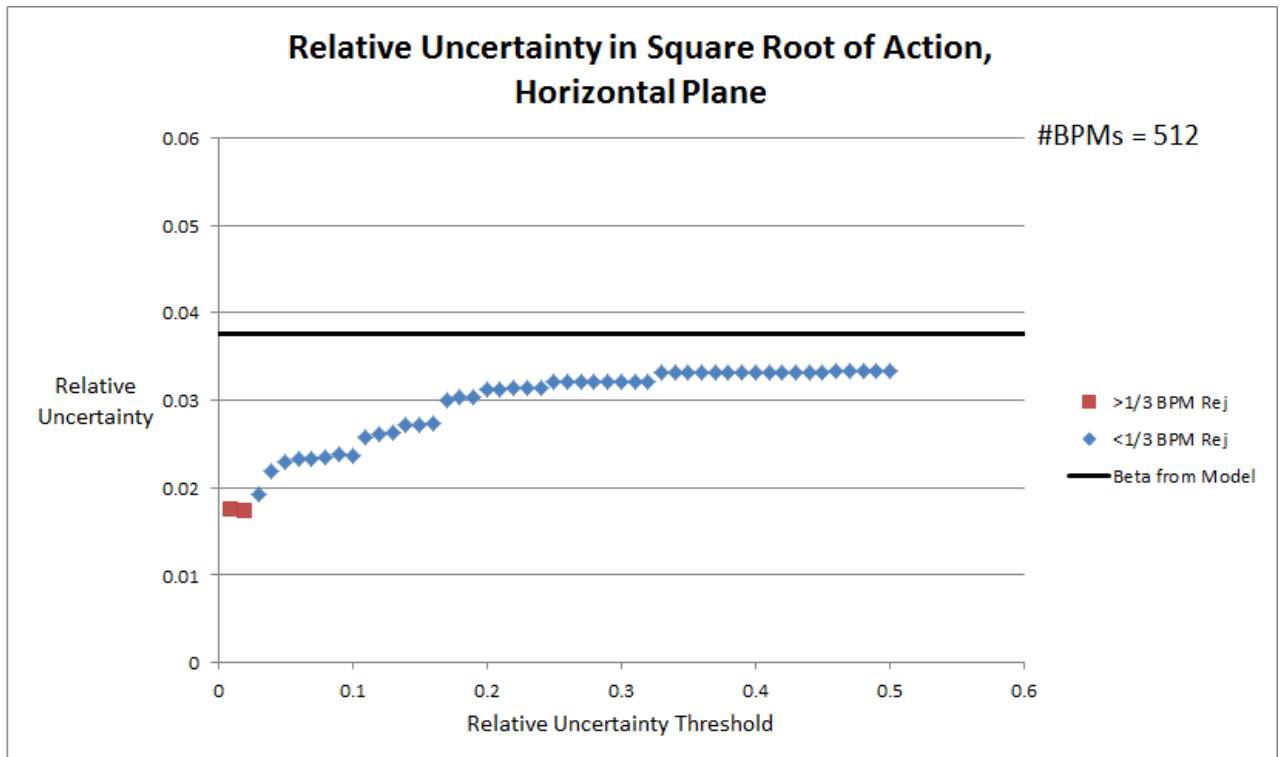


Figure 2: Same data set as in Figure 1 with the relative uncertainty in the action as a function of 'errthreshold', with 'bbthreshold' fixed at 15%.

BPMs were rejected. The LHC has around 530 BPMs, so assuming that at least 500 took data, this '1/3' level would guarantee that at least 333 BPMs were used in the calculation for the action.

In Figure 1, 'errthreshold' was fixed while 'bbthreshold' was allowed to vary. We see that as 'bbthreshold' increases, more BPMs are accepted and the relative uncertainty increases until it reaches a plateau value. This represents a gap in beta-beating levels of the BPMs in which those which are still being rejected are deviating from the model significantly more than a supermajority of the BPMs. This was a common feature seen in these plots.

In Figure 2, 'bbthreshold' was instead fixed while 'errthreshold' varied, and we see very similar results. This shows that both thresholds are removing unwanted BPMs, so using both will provide more stability in the results obtained.

Acknowledgments

I would like to thank Rogelio Tomás, Viktor Maier, and Ewen Maclean for their guidance in this project. An additional thanks goes to Phil Rubin and the IRES at CERN program for giving me the opportunity to do a project at CERN.

References

- [1] Glenn Vanbavinckhove, [2012], "Optics measurements and corrections for colliders and other storage rings" , Ph.D, CERN, Switzerland

2 Appendix: Python Code

The following is the function in the python code which calculates the action given the axis and source of the beta function. This function is in the module "helper.py" which is a submodule of "algorithms" in the GetLLM directory. Irrelevant parts of the code have been removed for better readability.

```
def gen_kick_calc(list_of_files, mad_twiss, beta_d, source, plane, bbthreshold,
errthreshold):
    """
    :Parameters:
        'list_of_files': list
            list of either linx or liny files input, produced by drive
        'mad_twiss': class
            The model file.
        'beta_d': class
            Contains measured beta functions calculated in GetLLM
        'source': string
            From where the beta function was calcuated, either model, phase, or
amp
    returns: list, list, int
        The value and uncertainty for sqrt(2J) and 2J resp., along #BPM rej
    """
    commonbpms = Utilities.bpm.intersect(list_of_files)
    if source == 'model':
        commonbpms = Utilities.bpm.model.intersect(commonbpms, mad_twiss)
    elif source == 'amp':
        if plane == 'H':
```

```

        commonbpms = Utilities.bpm.intersect_with_bpm_list(commonbpms, beta_d
.x_amp.keys())
        elif plane == 'V':
            commonbpms = Utilities.bpm.intersect_with_bpm_list(commonbpms, beta_d
.y_amp.keys())
        elif source == 'phase':
            if plane == 'H':
                commonbpms = Utilities.bpm.intersect_with_bpm_list(commonbpms, beta_d
.x_phase.keys())
            elif plane == 'V':
                commonbpms = Utilities.bpm.intersect_with_bpm_list(commonbpms, beta_d
.y_phase.keys())

meansqrt2j = []
mean2j = []
rejbpmscount = 0

#Two different action calcs follow, one is by finding mean(sqrt(2j))--¿
meansqrt2j and the other is by finding mean(2j)--¿mean2j
for i in range(0, len(commonbpms)):
    bn1 = str.upper(commonbpms[i][1])
    #Gets beta function for use in action calculation - performs checks on
its quality, rejects if above threshold
    if plane == 'H':
        if source == 'model':
            tembeta = mad_twiss.BETX[mad_twiss.indx[bn1]]
        elif source == 'phase':
            if beta_d.x_phase[bn1][0] <= 0:
                rejbpmscount += 1
                continue
            elif abs((beta_d.x_phase[bn1][0] - mad_twiss.BETX[mad_twiss.indx[
bn1]])/mad_twiss.BETX[mad_twiss.indx[bn1]]) > bbthreshold:
                rejbpmscount += 1
                continue
            elif (beta_d.x_phase[bn1][1]/beta_d.x_phase[bn1][0]) >
errthreshold:
                rejbpmscount += 1
                continue
            tembeta = beta_d.x_phase[bn1][0]
        elif plane == 'V':
            if source == 'model':
                tembeta = mad_twiss.BETY[mad_twiss.indx[bn1]]
            elif source == 'phase':
                if beta_d.y_phase[bn1][0] <= 0:
                    rejbpmscount += 1
                    continue
                elif (abs(beta_d.y_phase[bn1][0] - mad_twiss.BETY[mad_twiss.indx[
bn1]])/mad_twiss.BETY[mad_twiss.indx[bn1]]) > bbthreshold:
                    rejbpmscount += 1
                    continue
                elif (beta_d.y_phase[bn1][1]/beta_d.y_phase[bn1][0]) >
errthreshold:
                    rejbpmscount += 1
                    continue
            tembeta = beta_d.y_phase[bn1][0]

meansqrt2j_i = 0.0
mean2j_i = 0.0

```

```

for tw_file in list_of_files:
    meansqrt2j_i += tw_file.PK2PK[tw_file.indx[bn1]]/2.
    mean2j_i = (tw_file.PK2PK[tw_file.indx[bn1]]/2)**2

    meansqrt2j_i = meansqrt2j_i/len(list_of_files)
    meansqrt2j.append(meansqrt2j_i/math.sqrt(tembeta))
    mean2j_i = mean2j_i/len(list_of_files)
    mean2j.append(mean2j_i/tembeta)

if len(commonbpms) == rejbpmcount:
    print "Beta function calculated from", source, "in plane", plane, "is
no good, no kick or action will be calculated from it."
    j_old, mean_2j = [0,0], [0,0]
    return j_old, mean_2j, rejbpmcount

meansqrt2j = np.array(meansqrt2j)
meansqrt2j_ave = np.average(meansqrt2j)
meansqrt2j_std = math.sqrt(np.average(meansqrt2j*meansqrt2j) - meansqrt2j_ave
**2+2.2e-16)
mean2j = np.array(mean2j)
mean2j_ave = np.average(mean2j)
mean2j_std = math.sqrt(np.average(mean2j*mean2j) - mean2j_ave**2+2.2e-16)

meansqrt_2j = [meansqrt2j_ave, meansqrt2j_std]
mean_2j = [mean2j_ave, mean2j_std]

return meansqrt_2j, mean_2j, rejbpmcount

```