BDSIM Documentation

Release 0.64

L. Nevay, S. Boogert, L. Deacon, H. Garcia-Morales, J. Snuverink

March 04, 2015

CONTENTS

Contents:

CHAPTER ONE

INTRODUCTION

1.1 Purpose of BDSIM

1.2 General Simulation Steps

CHAPTER TWO

INSTALLATION

2.1 Downloading BDSIM

2.2 Compiling - SL6

CHAPTER

THREE

GETTING STARTED

CHAPTER

FOUR

LATTICE ELEMENTS

CHAPTER

FIVE

LATTICE SEQUENCE

CHAPTER

SIX

SAMPLERS - OUTPUT

CHAPTER

SEVEN

PHYSICS LISTS

CHAPTER

EIGHT

OPTIONS

CHAPTER

NINE

BEAM PARAMETERS

CHAPTER

TEN

DISTRIBUTIONS

- gauss
- gaussTwiss
- reference

Aperture Model	# of parameters	aper1	aper2	aper3	aper4
circular	1	radius	NA	NA	NA
rectangular	2	x half width	y half width	NA	NA
elliptical	2	x semi-axis	y semi-axis	NA	NA
lhcscreen-simple	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
lhcscreen	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
rectellipse	4	x half width of rectangle	y half width of rectangle	x semi-axis of ellipse	y semi-axis of ellipse
racetrack	3	horizontal offset of circle	vertical offset of circle	radius of circular part	NA
octagon	4	x half width	y half width	angle 1 [rad]	angle 2 [rad]

after reading this, go back to the top in *Distributions*

You can also have a look in another document at *Output Analysis*

These parameters can be set with the *option* command as the default parameters and also on a per element basis, that overrides the defaults for that specific element. Up to four parameters can be used to specify the aperture shape (*aper1*, *aper2*, *aper3*, *aper4*). These are used differently for each aperture model and match the MADX aperture definitions. The required parameters and their meaning are given in the following table.

Currently, only circular and rectangular are implemented. More models will be completed shortly.

The outer volume is represented (with the exception of the *drift* element) by a cylinder with inner radius equal to the beampipe outer radius and with outer radius given by default by the global *boxSize* option, which can usually be overridden with the *outR* option.

In Geant4 it is possible to drive different *regions* each with their own production cuts and user limits. In BDSIM three different regions exist, each with their own user defined production cuts (see *Physics*). These are the default region, the precision region and the approximation region. Beamline elements can be set to the precision region by setting the attribute *precisionRegion* equal to 1. For example:

CHAPTER

ELEVEN

MODEL PREPARATION

11.1 Manual Preparation

11.2 MADX Conversion

11.3 MAD8 Conversion

11.4 Python Builder

CHAPTER

TWELVE

OUTPUT ANALYSIS

12.1 ROOT Output

12.2 ASCII Output

pybdsim - python tools for bdsim

dependencies: package - minimum version required numpy - 1.7.1 matplotlib - 1.3.0

Modules: Builder - create generic accelerators for bdsim Convert - convert other formats into gmad Data - read the bdsim output formats Gmad - create bdsim input files - lattices & options Options - methods to generate bdsim options Plot - some nice plots for data

Classes: Analysis - encapsulates functions & plots for a single file Beam - a beam options dictionary with methods Builder

Build generic machines for bdsim. You can create a lattice using one of the predefined simple lattices or by adding many pieces together of your own design. Finally, output the gmad files required.

Classes: Element - beam line element that always has name,type and length Machine - a list of elements

`pybdsim.Builder.`**`CreateDipoleFodoRing`**(*filename*, *ncells=60*, *circumference=200.0*, *samplers='first'*)

Create a ring composed of fodo cells with 2 dipoles per fodo cell.

filename ncells - number of fodo+dipole cells to create circumference - circumference of machine in metres samplers - 'first','last' or 'all'

Hard coded to produce the following cell fractions: 50% dipoles 20% quadrupoles 30% beam pipe / drift

`pybdsim.Builder.`**`CreateDipoleRing`**(*filename*, *ncells=60*, *circumference=100.0*, *dfraction=0.1*, *samplers='first'*)

Create a ring composed solely of dipoles filename ncells - number of cells, each containing 1 dipole and a drift circumference - in metres dfraction - the fraction of dipoles in each cell (0.0<dfraction<1.0) samplers - 'first', 'last' or 'all'

`pybdsim.Builder.`**`CreateFodoLine`**(*filename*, *ncells=10*, *driftlength=4.0*, *magnetlength=1.0*, *samplers='all'*, ***kwargs*)

Create a FODO lattice with ncells.

ncells - number of fodo cells driftlength - length of drift segment in between magnets magnetlength - length of quadrupoles samplers - 'all','first' or 'last' **kwargs - kwargs to supply to quadrupole constructor

class `pybdsim.Builder.`**`Element`**(*name*, *category*, ***kwargs*)

Element - a beam element class - inherits dict

Element(name,type,**kwargs)

A beam line element must ALWAYs have a name, and type. The keyword arguments are specific to the type and are up to the user to specify.

Numbers are converted to a python Decimal type to provide higher accuracy in the representation of numbers - 15 decimal places are used.

`pybdsim.Builder.`**`GenerateSamplersFromBDSIMSurvey`**(*surveyfile*, *outputfilename*, *excludesamplers=True*)

Create a gmad file with samplers for all the elements in a beamline as described by the survey outline from bdsim

bdsim –file=mylattice.gmad –outline=survey.dat –outline_type=survey

excludesamplers - bool - exclude any existing samplers

`pybdsim.Builder.`**`SuggestFodoK`**(*magnetlength*, *driftlength*)

returns k1 (float) value for matching into next quad in a FODO cell. f = 1/(k1 * magnetlength) = driftlength -> solve for k1

Note the convention in pybdsim.Builder is that the quadrupoles in the fodo cell are split in two. So this is in fact half the integrated k you need. This matches with the other functions in Builder.

`pybdsim.Builder.`**`WriteLattice`**(*machine(machine)*, *filename(string)*, *verbose(bool)*)

Write a lattice to disk. This writes several files to make the machine, namely:

filename_components.gmad - component files (max 10k per file) filename_sequence.gmad - lattice definition filename_samplers.gmad - sampler definitions (max 10k per file) filename_options.gmad - options (TO BE IMPLEMENTED) filename.gmad - suitable main file with all sub

files in correct order

these are prefixed with the specified filename / path

class `pybdsim.Data.`**`BDSAsciiData`**(**args*, ***kwargs*)

`Filter`(*booleanarray*)

Filter the data with a booleanarray. Where true, will return that event in the data.

Return type is BDSAsciiData

`IndexFromNearestS`(*S*)

IndexFromNearestS(S)

return the index of the beamline element clostest to S

Only works if “SStart” column exists in data

`MatchValue`(*parametername*, *matchvalue*, *tolerance*)

This is used to filter the instance of the class based on matching a parameter withing a certain tolerance.

a = pybdsim.Data.Load(“myfile.txt”) MatchValue(“S”,0.3,0.0004)

this will match the “S” variable in instance “a” to the value of 0.3 within +- 0.0004.

You can therefore used to match any parameter.

Return type is BDSAsciiData

CHAPTER

THIRTEEN

APPENDIX 1 - TRACKING ROUTINES

13.1 Quadrupole

CHAPTER

FOURTEEN

APPENDIX 2 - GEOMETRY INPUT

14.1 Mokka

CHAPTER

FIFTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX