**03/07/2019 (not all output pictures are presented)**

**Source(Git repository):** [**https://github.com/radiasoft/rssynergia/blob/master/examples/iota\_examples/rssynergia\_example\_notebook.ipynb**](https://github.com/radiasoft/rssynergia/blob/master/examples/iota_examples/rssynergia_example_notebook.ipynb)**)**

**File:** [rssynergia](https://github.com/radiasoft/rssynergia)/[examples](https://github.com/radiasoft/rssynergia/tree/master/examples)/[iota\_examples](https://github.com/radiasoft/rssynergia/tree/master/examples/iota_examples)/**rssynergia\_example\_notebook.ipynb**

[@bruhwilerbruhwiler](https://github.com/bruhwiler) [fixed call to read\_bunch again](https://github.com/radiasoft/rssynergia/commit/74a8c900d419ef4050e556505bc79ffd2b9940a5) 23 hours ago

**An example notebook showcasing Synergia simulations with IOTA**

This notebook showcases some of the basic functionality provided by the IPython notebook environment for running simulations with Synergia. We make use of basic Synergia functions as well as our own scripts. This notebook, the associated scripts, and a host of other resources can be found at RadiaSoft's [rssynergia](https://github.com/radiasoft/rssynergia) github repository: <https://github.com/radiasoft/rssynergia>.

*NOTE: Lattice files may be found in the corresponding* [*ioptics*](https://github.com/radiasoft/ioptics) *repository.*

**Basic Simulation Parameters**

* IOTA v6.6 lattice with 1 integrable optics section
* Sextupole strengths set to zero
* Nonlinear element adapted from MADX script - comprised of 20 discrete multipoles
* Full CHEF propagation - symplectic single particle tracking including nonlinear effects
* Each element in the lattice is sliced into 5 segments for computing dynamics.

**Basic Beam Parameters**

* 2.5 MeV proton beam
* No space charge or collective effects considered
* Beam is matched to a generalized KV distribution with normalized emittance mm-mrad

**Specific setup notes**

1. Be sure to have the rssynergia repo in your path
2. Adjust the lattice\_repo in code block 4 to match the location of your lattice files.
3. If you do not have the watermark package, then its import may be commented out as its non-essential to running the notebook.

**Imports and Setup**

**IPython magics**

In [10]:

%matplotlib inline

%load\_ext autoreload

%autoreload 2

The autoreload extension is already loaded. To reload it, use:

%reload\_ext autoreload

**Basic Python imports**

In [11]:

import sys, os

import numpy as np

import matplotlib.pyplot as plt

import matplotlib as mpl

import scipy

import tables

from mpi4py import MPI

**Synergia specific imports**

In [12]:

import rssynergia

from rssynergia.base\_diagnostics import read\_bunch

from rssynergia.base\_diagnostics import workflow

from rssynergia.base\_diagnostics import lfplot

from rssynergia.base\_diagnostics import latticework

from rssynergia.base\_diagnostics import basic\_calcs

from rssynergia.base\_diagnostics import pltbunch

from rssynergia.base\_diagnostics import elliptic\_sp

from rssynergia.base\_diagnostics import options

from rssynergia.base\_diagnostics import diagplot

from rssynergia.elliptic import elliptic\_beam6d

import synergia

import synergia\_workflow

**Lattice imports from MADX files and visualization**

Synergia supports lattice imports from MADX files, and can replicate a variety of magnetic elements, including the unique nonlinear elliptic magnets designed for the IOTA lattice.

By default, Synergia uses CHEF propagation for all elements. However, this can lead to some poor dynamics at larger emittances. For clarity, we use 1st order map propagation for all elements outside of the nonlinear insert.

In [14]:

#dictionary of lattices

lattices = {}

#================== Load the lattice =======================

#Assumes radiasoft/ioptics and radiasoft/rssynergia share a common directory

lattices = {}

lattice\_repo = '../../../ioptics/ioptics/lattices/Iota6-6/'

lattices['t1\_1IO\_66'] = lattice\_repo + "lattice\_1IO\_center.madx" #centered t1 6.6 1IO lattice

lattices['t3\_1IO\_66'] = lattice\_repo + "lattice\_1IO\_nll\_center.madx" #centered t3 6.6 1IO lattice

name = 't3\_1IO\_66'

lattice = synergia.lattice.MadX\_reader().get\_lattice("iota", lattices[name])

for elem in lattice.get\_elements():

if elem.get\_type() == 'nllens':

elem.set\_string\_attribute("extractor\_type", "chef\_propagate")

else:

elem.set\_string\_attribute("extractor\_type", "chef\_map")

nsteps\_per\_element = 5

nsteps = len(lattice.get\_elements())\*nsteps\_per\_element

order = 1

outputdir = 'example\_run'

opts = workflow.make\_opts(name, order, outputdir, nsteps, nsteps\_per\_element)

opts.macro\_particles=10000

#opts.emitx = 1.0e-5

workflow.make\_path(outputdir)

stepper = synergia.simulation.Independent\_stepper\_elements(lattice, opts.map\_order, opts.steps\_per\_element)

lattice\_simulator = stepper.get\_lattice\_simulator()

#construct bare lattice for comparison

bare\_lattice = synergia.lattice.MadX\_reader().get\_lattice("iota", lattices['t1\_1IO\_66'])

bare\_stepper = synergia.simulation.Independent\_stepper\_elements(bare\_lattice, opts.map\_order, opts.steps\_per\_element)

bare\_lattice\_simulator = bare\_stepper.get\_lattice\_simulator()

**Plot the bare lattice functions**

The Tier 1 (bare) IOTA lattice is completely symmetric, with equal tunes in x and y. The drift regions designed to accomodate the nonlinear element feature zero dispersion and a fixed "tune" advance of .

In [15]:

opts.lattice\_name = 'Bare IOTA 6-6 Lattice with 1IO'

opts.ID = None

opts.path = None

opts.turns = opts.turns

opts.variance = 0.5

opts.lattice\_simulator = bare\_lattice\_simulator

opts.relpath = opts.output\_dir

opts.lf\_fns = ['beta\_x','beta\_y','D\_x']

opts.lattice = bare\_lattice

opts.save = False

opts.scale = 2

lfplot.plot\_sliced\_lattice\_functions(opts)

**Plot the lattice functions with the NL element included**

Inclusion of the nonlinear element breaks the symmetry of the lattice, adjusting the lattice tunes but maintaining individual symmetry in x and y through the nonlinear element, along with the zero dispersion requirement.

In [16]:

opts.lattice\_name = 'IOTA 6-6 Lattice with 1IO and NL Element'

opts.ID = None

opts.path = None

opts.turns = opts.turns

opts.variance = 0.5

opts.lattice\_simulator = lattice\_simulator

opts.relpath = opts.output\_dir

opts.lf\_fns = ['beta\_x','beta\_y','D\_x']

opts.lattice = lattice

opts.save = False

opts.scale = 2

lfplot.plot\_sliced\_lattice\_functions(opts)

**Generate a matched bunch for the IOTA lattice**

Synergia includes functions for generating matched bunches with Gaussian or K-V distributions for a given linear lattice.

**Gaussian Bunch matched to the bare lattice**

In [17]:

#get refence particle to find beta, gamma values

ref = lattice.get\_reference\_particle()

beta = ref.get\_beta()

gamma = ref.get\_gamma()

#We want the normalized emittance in x to be 0.3 mm-mrad

opts.norm\_emittance = 0.3\*1.e-6

opts.emitx = basic\_calcs.calc\_geometric\_emittance(opts.norm\_emittance, beta, gamma)

opts.emity = opts.emitx

#Construct a matched bunch

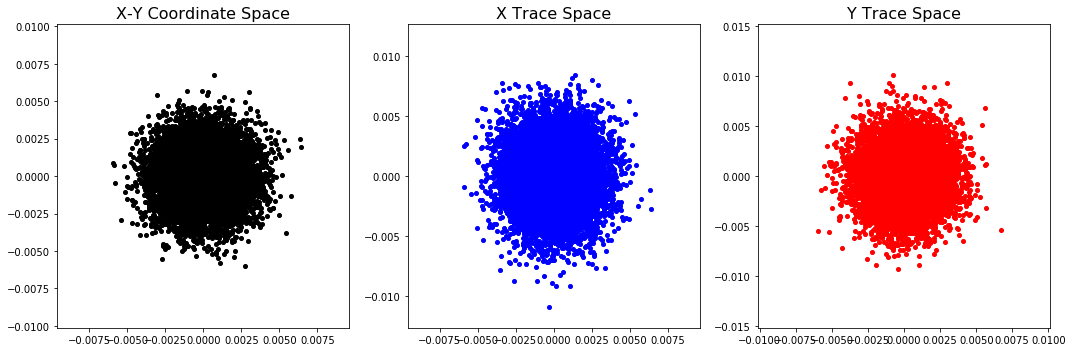
myBunch = synergia.optics.generate\_matched\_bunch\_transverse(

bare\_lattice\_simulator, opts.emitx, opts.emity, opts.stdz,

opts.dpop, opts.real\_particles,

opts.macro\_particles, opts.seed)

pltbunch.plot\_bunch(myBunch)

**Generalized KV distribution matched to the IOTA lattice at the center of the NL element**

For the nonlinear IOTA ring, a more complex matching is required. Stephen Webb has devised a matching procedure which cosntructs a generalized K-V distribution, with a fixed value of the 1st invariant, . To produce the same normalized emittance of mm-mrad for the injected proton beam, we construct a bunch with fixed mm-mrad.

In [18]:

tval = 0.4 #elliptic strength parameter

cval = 0.01 #aperture parameter

opts.t = tval

opts.c = cval

opts.new\_tune = 0.3

opts.lnll = 1.8

opts.nseg = 20

vals = basic\_calcs.get\_base\_nll(opts.lnll, opts.new\_tune, opts.t, opts.c)

#specify vals for center of the section

opts.betae = vals[3]

opts.alphae = 0 #fixed 0 alpha for center

opts.beta0 = vals[3]

opts.emits = [9.74e-6]

opts.lattice = lattice

In [20]:

particles = elliptic\_beam6d.toyellipticalbeam6D(opts)

#construct a toyheader for quick calculation of bunch properties

toyheader = {}

toyheader['s\_val'] = 0.

for index in range(len(opts.emits)):

bunch = particles[index]

initialH,initialI = elliptic\_sp.calc\_bunch\_H(bunch,opts)

bunch\_mean = np.mean(initialH)

bunch\_std = np.std(initialH)

bunch\_var = (bunch\_std/bunch\_mean)\*100

print "Constructed bunch with {} macroparticles, having mean H: {} and std: {}%".format(opts.macro\_particles, bunch\_mean,bunch\_var)

#now add longitudinal momentum variation

#For random samples with mean = 0, sigma = sigma, use sigma\*np.random.randn(...)

bunch[:,5] = opts.dpop\*np.random.randn(1,len(bunch))

#bunch[:,5] = np.zeros(len(bunch)) #0 dpop

opts.num\_total\_particles = opts.macro\_particles\*len(opts.emits)

opts.tracked\_particles = opts.num\_total\_particles

particles\_file = '{}/myBunch.txt'.format(opts.output\_dir)

np.savetxt(particles\_file,bunch) #write the bunch to a text file

bucket\_length = beta\*lattice.get\_length()/4 #RF harmonic number is 4

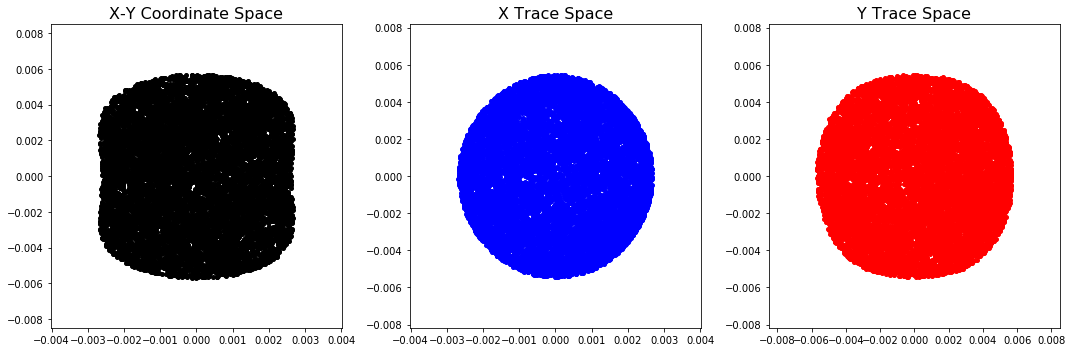
comm = synergia.utils.Commxx(True) #define a communicator

myBunch = read\_bunch.read\_bunch(particles\_file, ref, opts.real\_particles, comm, bucket\_length)

pltbunch.plot\_bunch(myBunch)

Constructed bunch with 10000 macroparticles, having mean H: 1.19923435041e-05 and std: 33.035518509%

Read 10000 particles

**Perform a basic simulation**

We will run our matched beam through the nonlinear lattice for 100 turns, outputing individual particle coordinates (Diagnostics\_particles) each turn and basic RMS bunch properties (Diagnostics\_basic) each step (slice) of the simulation.

In [21]:

#Construct the bunch simulator

bunch\_simulator = synergia.simulation.Bunch\_simulator(myBunch)

#basic diagnostics - PER STEP

basicdiag = synergia.bunch.Diagnostics\_basic("basic.h5", opts.output\_dir)

bunch\_simulator.add\_per\_step(basicdiag)

print "Saving basic diagnostics each step"

#include full diagnostics

fulldiag = synergia.bunch.Diagnostics\_full2("full.h5", opts.output\_dir)

bunch\_simulator.add\_per\_turn(fulldiag)

print "Saving full2 diagnostics each turn"

#particle diagnostics - PER TURN

opts.turnsPerDiag = 1

particlediag = synergia.bunch.Diagnostics\_particles("particles.h5",0,0,opts.output\_dir)

bunch\_simulator.add\_per\_turn(particlediag, opts.turnsPerDiag)

print "Saving turn-by-turn particle data every {} turns".format(opts.turnsPerDiag)

opts.turns = 100

opts.checkpointperiod = 10

opts.maxturns = opts.turns+1

myrank = comm.get\_rank()

print "setting up propagator for rank {}".format(myrank)

propagator = synergia.simulation.Propagator(stepper)

propagator.set\_checkpoint\_period(opts.checkpointperiod)

print "starting simulation for rank {}".format(myrank)

propagator.propagate(bunch\_simulator,opts.turns, opts.maxturns,opts.verbosity)

#clean up files

workflow.cleanup(opts.output\_dir)

Saving basic diagnostics each step

Saving full2 diagnostics each turn

Saving turn-by-turn particle data every 1 turns

setting up propagator for rank 0

starting simulation for rank 0

**Basic Diagnostics**

RMS beam properties are easily reconstructed from the basic diagnostics. Below, RMS bunch size in x and y, respectively, are overlayed for turn 1 and for turn 100.

In [22]:

opts.inputfile = opts.output\_dir + '/basic.h5'

opts.plots = ['x\_std', 'y\_std']

plotVals = diagplot.getPlotVals(opts.inputfile, opts.plots)

#define specific value arrays

xmaster = plotVals['s']

xstd = plotVals['x\_std']

ystd = plotVals['y\_std']

#we want to plot turn 0 and turn 100 on the same plot, so we should separate these specifically

interval = len(stepper.get\_steps())

xstd\_0 = xstd[:interval]

xstd\_2 = xstd[1\*interval:2\*interval]

xstd\_100 = xstd[-1\*interval:]

ystd\_0 = ystd[:interval]

ystd\_100 = ystd[-1\*interval:]

#We can use the same s value for each plot

sval\_0 = xmaster[:interval]

fig = plt.figure(figsize=(8,6))

ax = plt.gca()

ax.plot(sval\_0,xstd\_0\*1.e3,'b-',alpha=0.7, label='Turn 1') #plot the 1st turn

ax.plot(sval\_0,xstd\_100\*1.e3,'g-',alpha=0.7, label='Turn 100') #plot the 1st turn

axtitle = "Beam envelope evolution - $\sigma\_x$ over 100 turns"

ax.set\_title(axtitle, y = 1.02, fontsize = 18)

ax.set\_xlabel("s [m]",fontsize=14)

ax.set\_ylabel("rms beam size $\sigma\_x$ [mm]",fontsize=14)

ax.tick\_params(axis='x', labelsize=14)

ax.tick\_params(axis='y', labelsize=14)

ax.set\_xlim([0,lattice.get\_length()])

ax.legend()

sv\_title = '{}/RMS\_x\_1\_100.pdf'.format(opts.output\_dir)

fig.tight\_layout()

fig.savefig(sv\_title,bbox\_inches='tight')

**Particle Diagnostics**

Individual particles may be tracked, or alternatively their coordinates at specified positions in the ring may be dumped. Below, Poincare surfaces of sections are displayed for a handful of particles, illustration their motion in normalized coordinates.

In [23]:

opts.lost = None

opts.plots = ['x','px']

opts.relpath = opts.output\_dir

#opts.num = 100

opts.hcoord = opts.plots[0]

opts.vcoord = opts.plots[1]

opts.lattice\_name = name

opts.scale = 2

opts.lattice = lattice

opts.save = True

opts.num = 10

opts.plot\_lost = False

elliptic\_sp.toy\_plot\_Poincare(opts)

In [24]:

opts.lost = None

opts.plots = ['y','py']

opts.relpath = opts.output\_dir

#opts.num = 100

opts.hcoord = opts.plots[0]

opts.vcoord = opts.plots[1]

opts.lattice\_name = name

opts.scale = 2

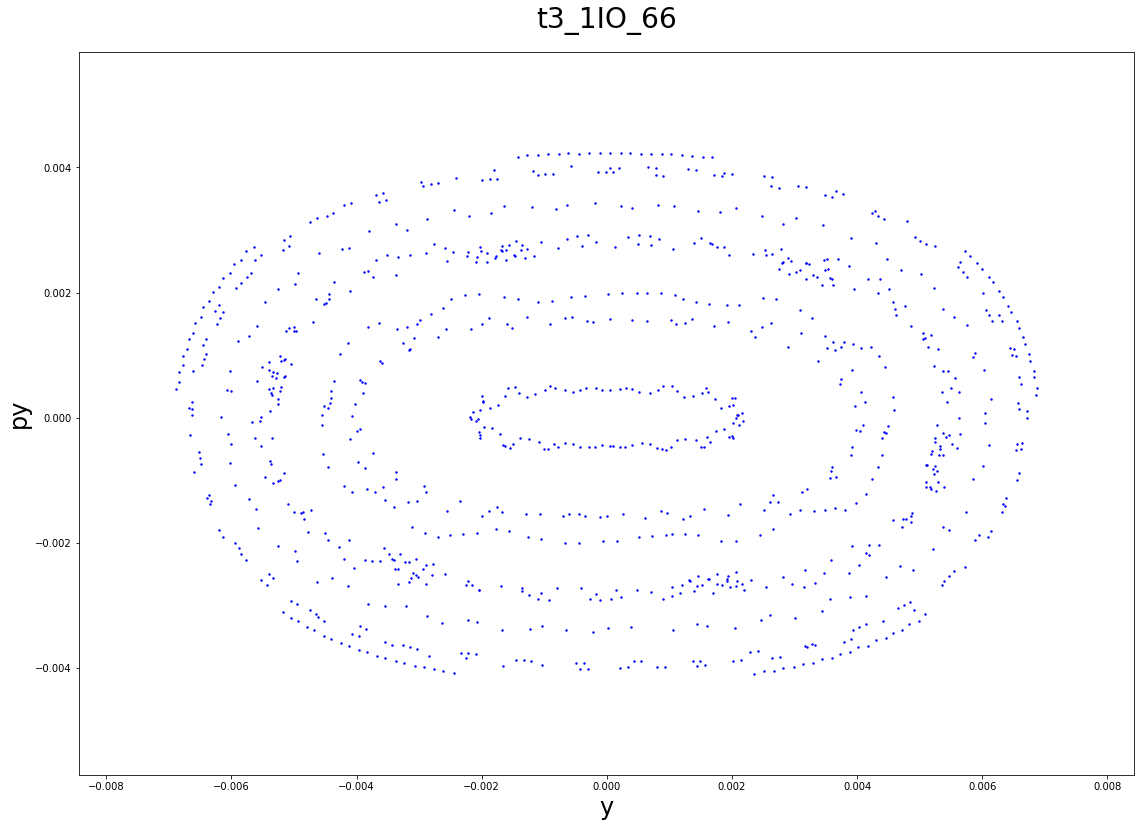
opts.lattice = lattice

opts.save = True

opts.num = 10

opts.plot\_lost = False

elliptic\_sp.toy\_plot\_Poincare(opts)

In [22]:

opts.lost = None

opts.plots = ['x','y']

opts.relpath = opts.output\_dir

#opts.num = 100

opts.hcoord = opts.plots[0]

opts.vcoord = opts.plots[1]

opts.lattice\_name = name

opts.scale = 2

opts.lattice = lattice

opts.save = True

opts.num = 50

opts.plot\_lost = False

elliptic\_sp.toy\_plot\_Poincare(opts)

**Bunch evolution**

We can similarly plot the bunch at a given turn to look for instances of nonlinear effects driving particles to large amplitude.

In [23]:

from matplotlib import gridspec

#Load particles

opts.relpath = opts.output\_dir

turnnums = ['00','100']

filelist = ["particles\_00{}.h5".format(val) for val in turnnums]

filelist[-1] = "particles\_0100.h5" #force this one

path = os.path.join(os.getcwd(),opts.relpath)

pathlist = [os.path.join(path,fn) for fn in filelist] #construct path list for files

twiss = elliptic\_sp.get\_toy\_twiss(opts)

log = [] #log is an array for storing output text

part\_array = []

for index,outfile in enumerate(pathlist):

header,particle\_vals = elliptic\_sp.get\_particles(outfile)

if index == 0:

part\_array = particle\_vals

else:

part\_array = np.vstack([part\_array,particle\_vals])

parts\_0 = part\_array[:10000]

parts\_100 = part\_array[10000:20000]

#Construct coordinates

header= {}

header['s\_val'] = 0.

norm\_coords0 = elliptic\_sp.normalized\_coordinates(header, parts\_0, twiss)

x\_c0 = norm\_coords0[:,0]

px\_c0 = norm\_coords0[:,1]

y\_c0 = norm\_coords0[:,2]

py\_c0 = norm\_coords0[:,3]

header= {}

header['s\_val'] = 0.

norm\_coords100 = elliptic\_sp.normalized\_coordinates(header, parts\_100, twiss)

x\_c100 = norm\_coords100[:,0]

px\_c100 = norm\_coords100[:,1]

y\_c100 = norm\_coords100[:,2]

py\_c100 = norm\_coords100[:,3]

#Make the plot

fig = plt.figure(figsize=(12,6))

gs = gridspec.GridSpec(1, 2, width\_ratios=[1,1])

ax0 = plt.subplot(gs[0])

ax0.scatter(1000\*x\_c0,1000\*y\_c0, s=4, c='k')

ax0.set\_title('Proton beam at turn 0', y = 1.02, fontsize = 18)

ax0.set\_xlabel("x [mm]",fontsize=14)

ax0.set\_ylabel("y [mm]",fontsize=14)

ax0.tick\_params(axis='x', labelsize=14)

ax0.tick\_params(axis='y', labelsize=14)

ax0.set\_xlim([-10,10])

ax0.set\_ylim([-10,10])

ax1 = plt.subplot(gs[1])

ax1.scatter(1000\*x\_c100,1000\*y\_c100, s=4, c='k')

ax1.set\_title('Proton beam at turn 100', y = 1.02, fontsize = 18)

ax1.set\_xlabel("x [mm]",fontsize=14)

ax1.set\_ylabel("y [mm]",fontsize=14)

ax1.tick\_params(axis='x', labelsize=14)

ax1.tick\_params(axis='y', labelsize=14)

ax1.set\_xlim([-10,10])

ax1.set\_ylim([-10,10])

sv\_title = '{}/beam\_0\_100.pdf'.format(opts.output\_dir)

fig.tight\_layout()

fig.savefig(sv\_title,bbox\_inches='tight')

fig.show()

/home/vagrant/.pyenv/versions/2.7.10/lib/python2.7/site-packages/matplotlib/figure.py:397: UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the figure

"matplotlib is currently using a non-GUI backend, "