

Jupyter for Accelerator Physics

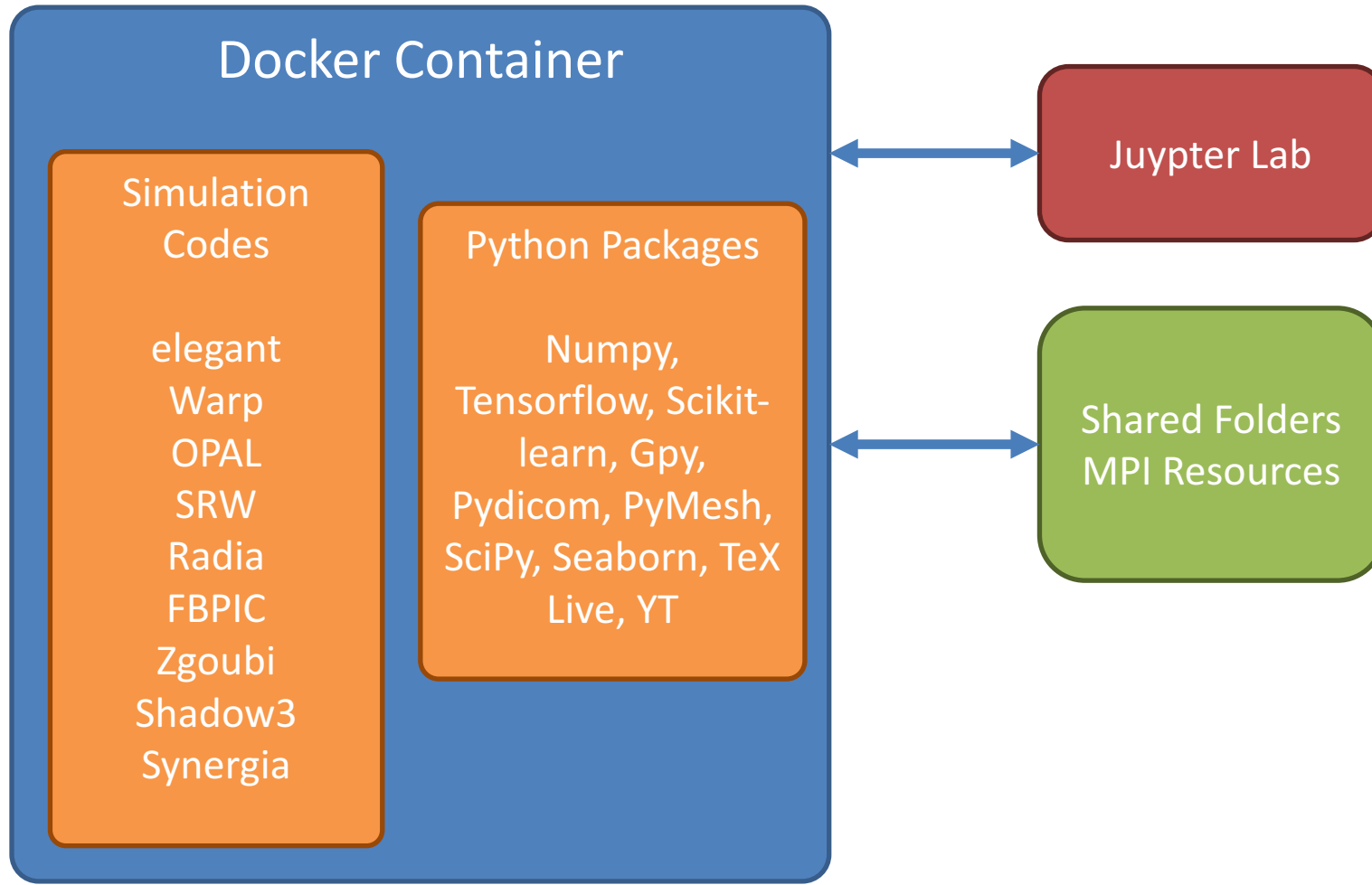
Jonathan Edelen, Robert Nagler, Paul Moeller, David Bruhwiler, Nathan Cook, Chris Hall, and Stephen Webb



Jupyter Tutorial @ ICALEPCS 2019

5 October 2019 – Brooklyn, NY

Jupyter/Hub @ RadiaSoft



RadiaSoft Jupyter/Hub Environment

- *17 staff users and 60 public users (in last 4 weeks)*
- *8TB used*
- *Pools: 3 public nodes, 5 internal nodes*
- *MPI: 12 nodes (pool nodes for workshops)*
- *Nginx proxy*
- *Dev, Alpha, Beta, Prod configurations*

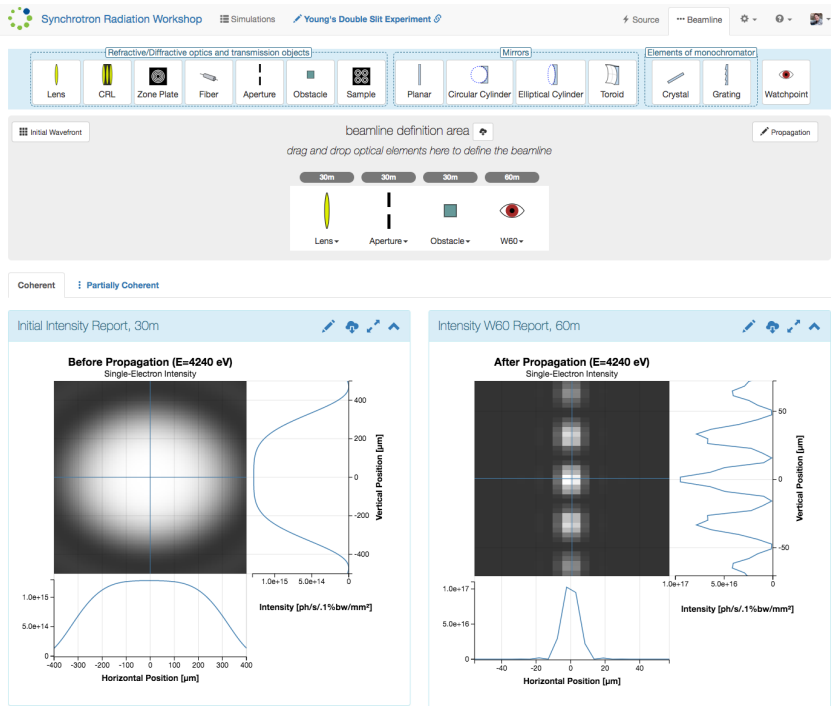
User Customizations

- `github.com/radiasoftware/jupyter.radiasoftware.org`
 - Executes `radia-run.sh` inside container before Jupyter starts
 - Copies template notebooks and other files
 - Used for patches in between releases
 - Runs `git config user.name` and `credential.helper`
 - If user has `jupyter.radiasoftware.org` repo, it runs after global repo
- `~/jupyter/bashrc` runs after container's `bashrc`
- `~/jupyter/bin` in path lets users persist commands

Jupyter/Hub Use Case: Teaching and Workshop Tutorials

- *Fermilab scientist learned Synergia via example notebooks running on jupyter.radiasoft.org*
- *UCLA undergrad learned FBPIC via example notebook in order to complete work study*
- *Grad student at UCLA learned Warp through example RadiaSoft notebooks*
- *Jan 2018 session of US Particle Accelerator School used jupyter.radiasoft.org to teach Synergia to 20 students*
- *ICFA ML Workshop in CH used jupyter.radiasoft.org to teach ML for accelerator physics to 60 participants*

Jupyter/Hub Use Case: Comparing Multiple Simulation Codes



README.md

SHADOW 3.0 SOURCE DISTRIBUTION

Contents:

1. What is SHADOW
2. Download
3. Source files
4. Building SHADOW
5. Other info
6. Contact

1 What is SHADOW

SHADOW is an open source ray tracing code for modeling optical systems.

Targeted to synchrotron radiation beamlines, it has unique features for designing X-ray optical systems.

For more info, please read this paper (open access):

SHADOW3: a new version of the synchrotron X-ray optics modelling package M. Sanchez del Rio, N. Canestrari, F. Jiang and F. Cerrina Journal of Synchrotron Radiation Volume 18, Part 5 (September 2011) <http://dx.doi.org/10.1107/S0909049511026306>

If you are (or want to be) a SHADOW user, it is recommended that you use a user interface. We strongly recommend OASYS (<https://www.elettra.eu/oasys.html>).

Jupyter/Hub Use Case: In-situ Analysis

“This is my most common working arrangement, as I am consistently running simulations in one panel while running analysis in a notebook in another.”

The screenshot displays a JupyterLab environment with two main panels. On the left, a file browser shows a directory structure for a beam propagation simulation. The central terminal window shows the execution of a Python script that sets up a particle-in-cell simulation. The output includes simulation parameters such as particle count, plasma frequency, and beam characteristics. On the right, a notebook cell contains Python code for data analysis and visualization. The code defines initial data, sets up a plot, and displays a scatter plot of particle positions (z vs r) and a heatmap of the longitudinal electric field (Ez) in GV/m. The plot shows four distinct beam spots in different colors (blue, green, red, black) and a corresponding electric field distribution.

```
CTH-PhaseII mkdir BPS
CTH-PhaseII cp BeamPropagate4/bunch4-propagate.py BPS/
CTH-PhaseII cd BPS/
BPS$ ls
bunch4-propagate.py
BPS$ rmpci -m 8 python bunch4-propagate.py -p 2 1 4
# Warp
# Origin date: Fri, 1 Feb 2019 10:07:46 -0700
# Local date: Fri, 1 Feb 2019 10:07:46 -0700
# Commit hash: edef438
# /home/vagrant/.pyenv/versions/py2.7.11b/python2.7/site-packages/warp/warp.py
# /home/vagrant/.pyenv/versions/py2.7.11b/python2.7/site-packages/warp/warpParallel.py
# Wed Jun 5 21:14:24 2019
# 8 processors
# import warp time 0.731453895569 seconds
# For more help, type warphelp()
('1 fftw fast', False)
Plot file name bunch4-propagate.000.cgm
10000 particles per bunch, with a charge of 5e-09 per bunch
zi=0.005-0.0025
CTR limit dt: 1.55459106571e-14
New dt: 1.32140240586e-14
*** particle simulation package W3D generating
--- Resecting lattice array sizes
--- Allocating space for particles
--- Loading particles
--- Setting charge density
--- Done
--- Allocating Win Moments
--- Allocating Z Moments
--- Allocating Lab Moments
Atomic number of ion = 5.4858E-04
Charge state of ion = -1.0000E+00
Initial X, Y emittances = 0.0000E+00, 0.0000E+00 m-rad
Initial X, Y envelope radii = 0.0000E+00, 0.0000E+00 m
Initial X, Y envelope angles = 0.0000E+00, 0.0000E+00 rad
Input beam current = 0.0000E+00 amps
Current density = 0.0000E+00 amps/cm^2
Charge density = 0.0000E+00 Coul/m^3
Number density = -0.0000E+00
Plasma frequency = 0.0000E+00 1/s
times dt = 0.0000E+00
times quad period = 0.0000E+00
Plasma period = 6.2832E+36 s
X-, Y-thermal Velocities = 0.0000E+00, 0.0000E+00 m/s
times dt = 0.0000E+00, 0.0000E+00 m
times dt/dx, dt/dy (X, Y) = 0.0000E+00, 0.0000E+00 m
X-, Y-Debye Wavelengths = 0.0000E+00, 0.0000E+00 m
over dx, dy (X and Y) = 0.0000E+00, 0.0000E+00 m
Longitudinal thermal velocity (rms) = 0.0000E+00 m/s
times dt = 0.0000E+00
times dt/dz = 0.0000E+00
Longitudinal Debye wavelength = 0.0000E+00 m
over dz = 0.0000E+00
Beam velocity = 0.0000E+00 m/s
over c = 0.0000E+00
Kinetic energy = 0.0000E+00 eV
Weight of simulation particles = 3.1208E+10
Number of simulation particles = 0
Number of real particles = 0.0000E+00
Total mass = 0.0000E+00 kg
Total charge = -0.0000E+00 Coul
Generalized permeance = 0.0000E+00
Characteristic current = -1.7043E+04 amps
```

```
#Define initial data
index = ts.iterations[-1] #ts.iterations[-1] #initial iteration to consider
ind_num = np.where(np.asarray(ts.iterations) == index)[0][0]
iter_time = ts[ind_num]
zfield, zmeta = ts.get_field(field='E', coord='z', iteration=index)
#zpart, xpart = ts.get_particle(var_list=['z', 'x'], species='beam', iteration=index)
beam1_z, beam1_x, beam1_y = ts.get_particle(var_list=['z', 'x', 'y'], species='beam1', iteration=index)
beam2_z, beam2_x, beam2_y = ts.get_particle(var_list=['z', 'x', 'y'], species='beam2', iteration=index)
beam3_z, beam3_x, beam3_y = ts.get_particle(var_list=['z', 'x', 'y'], species='beam3', iteration=index)
beam4_z, beam4_x, beam4_y = ts.get_particle(var_list=['z', 'x', 'y'], species='beam4', iteration=index)

#Setup figures and axes
fafig, ax = plt.subplots(figsize=(12,6))

#set initial labels
ax.set_xlim(zmeta.inshow_extent[2]*z_scale)
ax.set_ylim(zmeta.inshow_extent[-2]*z_scale)

ax.set_xlabel("z (mm)")
ax.set_ylabel("r (mm)")
#ax.set_title("Longitudinal Electric Field, $E_z$ - 2D R-Z {}".format(BEAM_NAME))

ax.hlines([-0.25,0.25],zmeta.inshow_extent[0]*z_scale,zmeta.inshow_extent[1]*z_scale,linestyle='dashed')

#set initial plots
splt1 = ax.scatter(beam1_z[::10]/z_scale,beam1_x[::10]/z_scale, s=2, c='k')
splt2 = ax.scatter(beam2_z[::10]/z_scale,beam2_x[::10]/z_scale, s=2, c='r')
splt3 = ax.scatter(beam3_z[::10]/z_scale,beam3_x[::10]/z_scale, s=2, c='g')
splt4 = ax.scatter(beam4_z[::10]/z_scale,beam4_x[::10]/z_scale, s=2, c='b')
eplt = ax.imshow(zfield/field_scale,cmap='viridis',extent=zmeta.inshow_extent*z_scale,vmin=-1.5,vmax=1.5,aspect='auto')

#set initial color bar
cbar = fafig.colorbar(eplt)
cbar.ax.set_xlabel("GV/m")
cbar.ax.xaxis.set_label_position('top')

#fafig.savefig('Ez_cubic.png')
```

Jupyter/Hub Use Case: In-situ Analysis

The screenshot displays a Jupyter Notebook environment with three main components:

- File Explorer (Left):** Shows a directory structure under `/rscn / opal_test /` with files like `dtl_test.h5`, `dtl_test.in`, `dtl_test.lbal`, `dtl_test.stat`, `Energy(MeV).txt`, `envelop_q.txt`, `envelop_z.txt`, `envelope.txt`, `lattice_generation_clo...`, `lattice_generation.ipynb`, and various `map_*.T7` files.
- Terminal (Middle):** Shows the execution of a script named `dtl_test.in`. The output includes parameters for beam size, momenta, correlation, and bunch characteristics. Key parameters include:
 - `rms beam size`: (0.41054, 1.37002, 0.30812) [mm]
 - `rms momenta`: (4.69921e-04, 1.67645e-04, 1.06583e-04) [beta gamma]
 - `rms correlation`: (1.58766e-01, 5.53877e-01, 9.09353e-01)
 - `NP`: 10000
 - `Qtot`: 0.300 [fC], `Qi`: 0.000 [fC]
 - `Ekin`: 2.265 [MeV], `dEkin`: 3.193 [keV]
 - `rmax`: (1.57322, 5.40757, 1.14657) [mm]
 - `rmin`: (-1.74459, -5.11689, -0.40065) [mm]
 - `rms beam size`: (0.45454, 1.39839, 0.31141) [mm]
 - `rms momenta`: (4.69916e-04, 1.67597e-04, 4.90492e-05) [beta gamma]
 - `rms correlation`: (4.52537e-01, 5.78136e-01, -3.67184e-03)
 - `hr`: (133.24399, 133.91376, 0.00001) [um]
 - `dh`: 1.00000e-10 [%]
 - `t`: 86.000 [ns], `dT`: 10.000 [ps]
 - `apous`: 1.403 [m]
- Code Cell (Right):** Contains Python code for plotting. It uses `plt.plot` to create two plots:
 - The first plot shows `opal_stat.energy` vs `opal_stat.position` (labeled 'opal') and `tw_energy` vs `tw_position` (labeled 'tracewin').
 - The second plot shows `opal_stat.rms_x` vs `opal_stat.rms_y` (labeled 'opal') and `env` vs `env` (labeled 'tracewin').
- Figure (Bottom Right):** A line plot showing `energy [MeV]` on the y-axis (ranging from 0.8 to 2.2) versus `position [m]` on the x-axis (ranging from 0.0 to 1.4). Two data series are plotted: 'opal' (blue line) and 'tracewin' (orange line). Both series show a linear increase in energy with position, with the 'opal' series exhibiting a step-like pattern.

Jupyter/Hub Use Case: Machine Learning for Accelerators

Sirepo Machine Learning Tutorial: Predicting the e- beam Longitudinal Phase Space from Elegant Simulations of the LCLS

NAPAC'19 Sirepo Users Workshop (September 3, 2019)

Developed by Auralee Edelen (SLAC) and Christopher Hall (RadiaSoft)

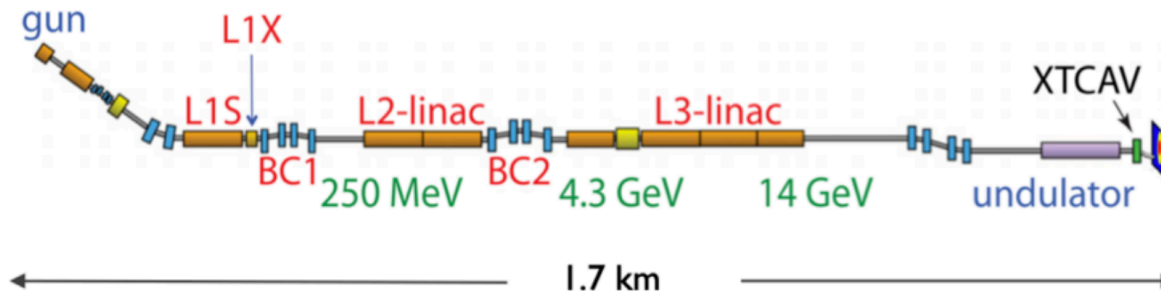
Elegant simulation of LCLS adapted from work by Yuantao Ding (SLAC)

Introduction

The aim of this tutorial is to show one example of how machine learning can be used to predict beam parameter output in an accelerator system.

Although we give some very basic tips along the way, this is not meant to be a pedagogical introduction to machine learning (for that and associate hands-on notebooks with toy accelerator problems see the ICFA ML tutorials: <https://indico.psi.ch/event/6698/sessions/3632/#20190226>).

Here, we run an elegant simulation of the LCLS with phase and amplitude jitter in L1 and produce the resultant e- beam phase space at the end of BC1 or BC2.



Jupyter/Hub Use Case: Machine Learning for Accelerators

Create Training Data - Jitter in Linac Phases and Amplitudes

(or skip if you want to just load the pre-made output file)

The training data simulates the impact of jitter in the linac phases and amplitudes. The data is generated by imposing random errors, with a Gaussian distribution, on phase and amplitude in the L1 linac section. The first run is a fiducializing step that uses the exact linac settings.

- `data_points` : Number of times to run.
- `phase_error` : The width of the Gaussian distribution of error in the L1 phase settings. This is an absolute error in units of Degrees.
- `amplitude_error` : The width of the Gaussian distribution of error in the L1 amplitude (Voltage) setting. This is a fractional error based on the default Voltage setting.

You can choose some range of setting errors and number of points to sample, but use the nominal values first.

More data is not always better -- we need to avoid oversampling the space, which would make our training set look identical to our validation and testing sets.

```
] : make_data = True

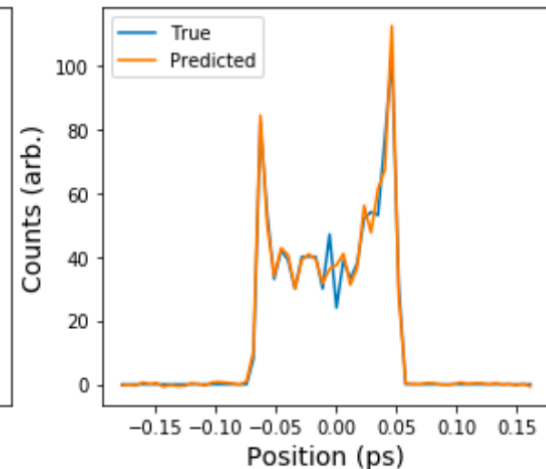
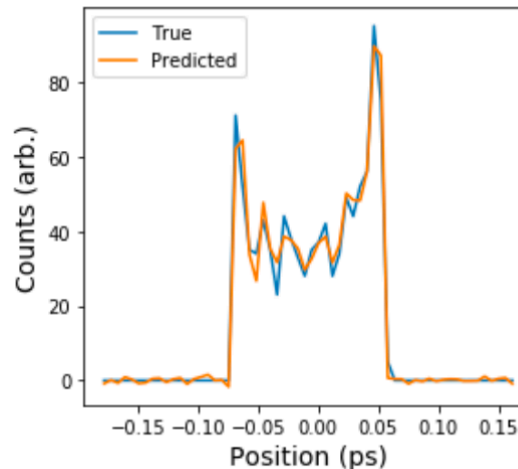
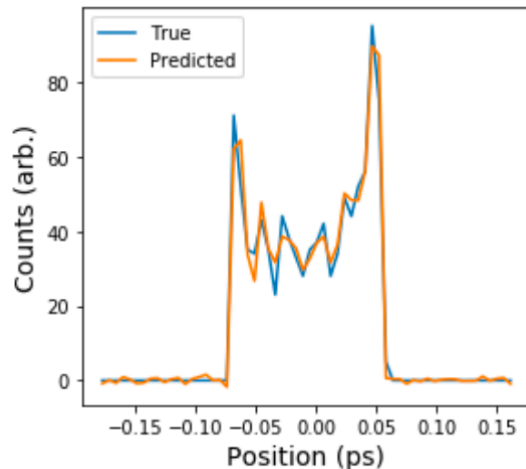
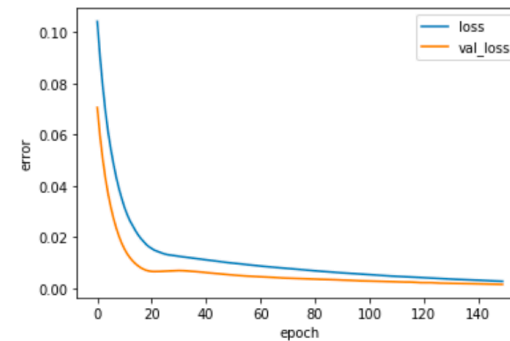
if make_data == True:
    # Scan settings
    data_points = 1000           # nominal: 1000
    phase_error_setting = 0.03  # nominal: 0.03
    amplitude_error_setting = 0.02 # nominal: 0.02
    lattice_name = 'BC2LINE'    # nominal: BC2LINE, may be BC1LINE or BC2LINE

    # Run scan in elegant
    t1=time.time()
    error_scan(max_iter=data_points,
               phase_error=phase_error_setting,
               amplitude_error=amplitude_error_setting,
               lattice=lattice_name)
    print(str('simulations took ' + str(np.round(((time.time()-t1)/60),2))+ ' mins'))
```

Jupyter/Hub Use Case: Machine Learning for Accelerators

```
[1]: #import required packages
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.constants import physical_constants
from SDDS import readSDDS
m_e = physical_constants["electron mass energy equivalent in MeV"][0]
from utilities3 import error_scan, lps_plot
from utilities import read_settings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, MinMaxScaler
import keras
from keras.models import Sequential
from keras.layers import Dense, GaussianNoise
from mlhelpers4 import PlotLosses, plot_a_bunch_of_beams, make_dataset
plot_losses=PlotLosses()
import time
```

```
# fit model
hist = model.fit(x = x_train, y = y_train,
                validation_data = (x_val, y_val),
                batch_size = 100,
                shuffle = 'true',
                epochs = 150,
                verbose = 'false',
                callbacks=[plot_losses])
```



Jupyter/Hub Use Case: Simulating Dielectric Wakefield Accelerators

Slab Dielectric Wakefield Accelerator Demonstration

Nathan Cook & Stephen Webb
RadiaSoft Sirepo User Workshop
swebb@radiasoft.net

This notebook documents the analysis of the GPT phase space provided by Gwanghui Ha. Afterwards, the notebook will be expanded to set up a Warp simulation using that phase space to describe each drive bunch.

This version removes the dielectric and looks at the bunch evolution from the center of the channel. Unique species labels for each bunch are included (e.g. beam1/2/3/4).

This version takes the X-Z geometry with original bunch spacing and changes the deposition order to third order.

Jupyter/Hub Use Case: Simulating Dielectric Wakefield Accelerators

```
%matplotlib inline

# Basic imports
import sys
del sys.argv[1:] # Necessary to run 'from warp import *' in IPython notebook without conflict.

# Import warp-specific packages
from warp import *
from warp.init_tools import *
from opmd_viewer import OpenPMDTimeSeries

# Import rswarp packages
import rswarp
from rswarp.utilities.file_utils import cleanupPrevious
from rswarp.utilities.file_utils import readparticles
from rswarp.utilities.file_utils import loadparticlefiles

# Import plotting and analysis packages
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

# Constants imports
from scipy.constants import e, m_e, c, k, elementary_charge
from scipy.constants import c as clight
kb_eV = 8.6173324e-5 # Boltzmann constant in eV/K
kb_J = k # Boltzmann constant in J/K
m = m_e # electron mass
qe = elementary_charge # electron charge

# Warp
# Origin date: Fri, 1 Feb 2019 10:07:46 -0700
# Local date: Fri, 1 Feb 2019 10:07:46 -0700
# Commit hash: edef438
# /home/vagrant/.pyenv/versions/py2/lib/python2.7/site-packages/warp/warp.py
# /home/vagrant/.pyenv/versions/py2/lib/python2.7/site-packages/warp/warpC.so
# Fri Oct 4 19:18:33 2019
# import warp time 0.634296178818 seconds
# For more help, type warphelp()
('l_fftw_fort', False)
```

Jupyter/Hub Use Case: Simulating Dielectric Wakefield Accelerators

iteration 775

Field type

Field: B E J rho

Coord: x y z

Particle quantities

beam

x y z ux uy uz w

x y z ux uy uz w None

Plotting options

Always refresh Refresh now!

Particle selection

Plotting options

Always refresh Refresh now!

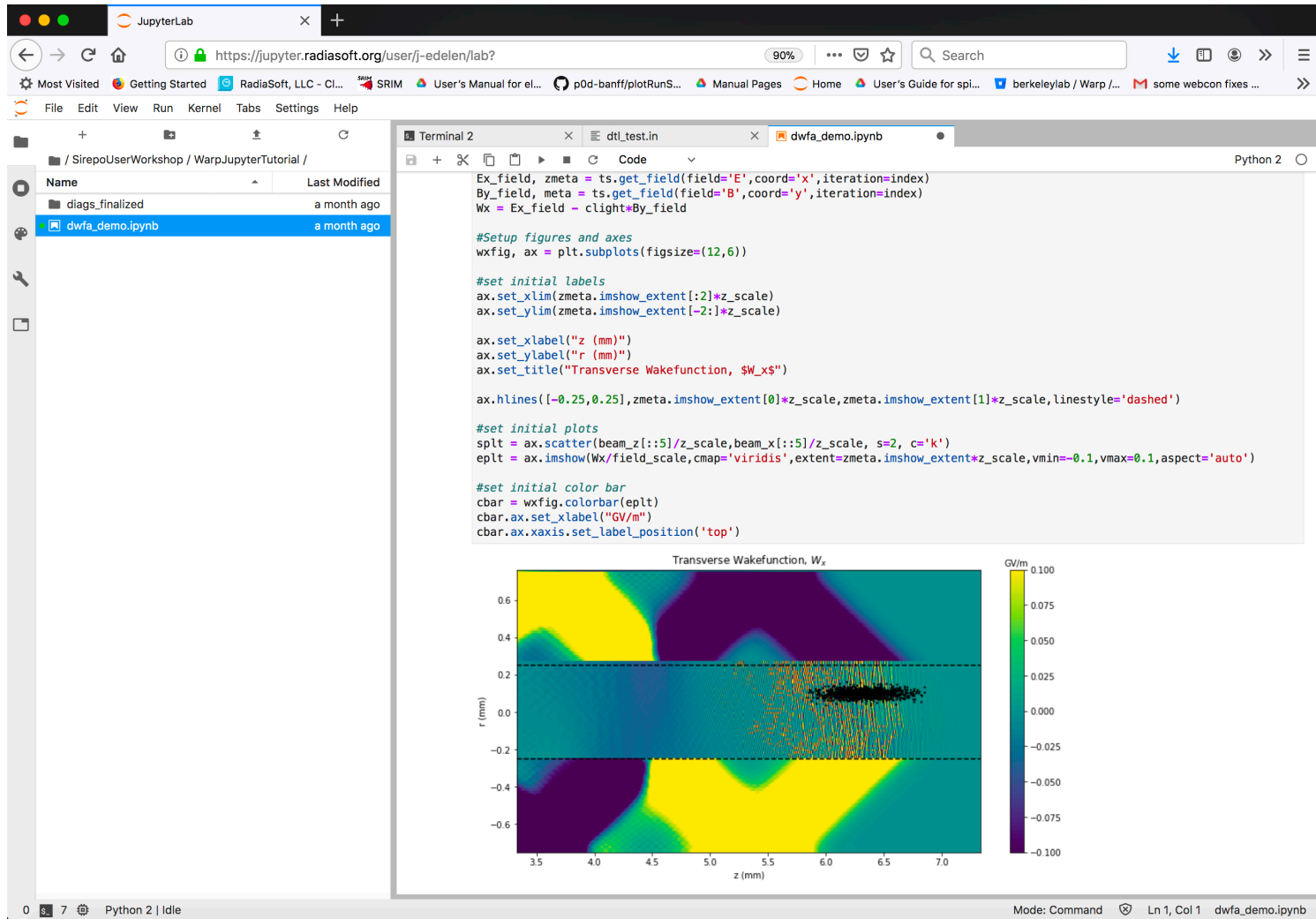
Figure 0

By at 8900.1 fs (iteration 800)

x (μm)

z (μm)

Jupyter/Hub Use Case: Simulating Dielectric Wakefield Accelerators



Takeaways

- *Wish list*
 - *Storage limits (quotas)*
 - *User/group file sharing*
 - *Real-time collaboration/debugging (like CoCalc)*
 - *Better user notifications (server restarts, long operations, no more servers)*
 - *Hub admin page spawner-specific output*
- *Highlights*
 - *Users love Jupyter*
 - *Users want all codes pre-installed*
 - *Users will consume all available resources*
 - *Jupyter/Hub is easily customizable*