Jupyter for Accelerator Physics

Robert Nagler   Paul Moeller   David Bruhwiler

Chris Hall   Nathan Cook

rsl.link/jcw19

radiasoft

Jupyter for Science User Facilities and High Performance Computing 2019

11 June 2019 – Berkeley

This work is supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Award #DE-SC0011340.

U.S. DEPARTMENT OF ENERGY

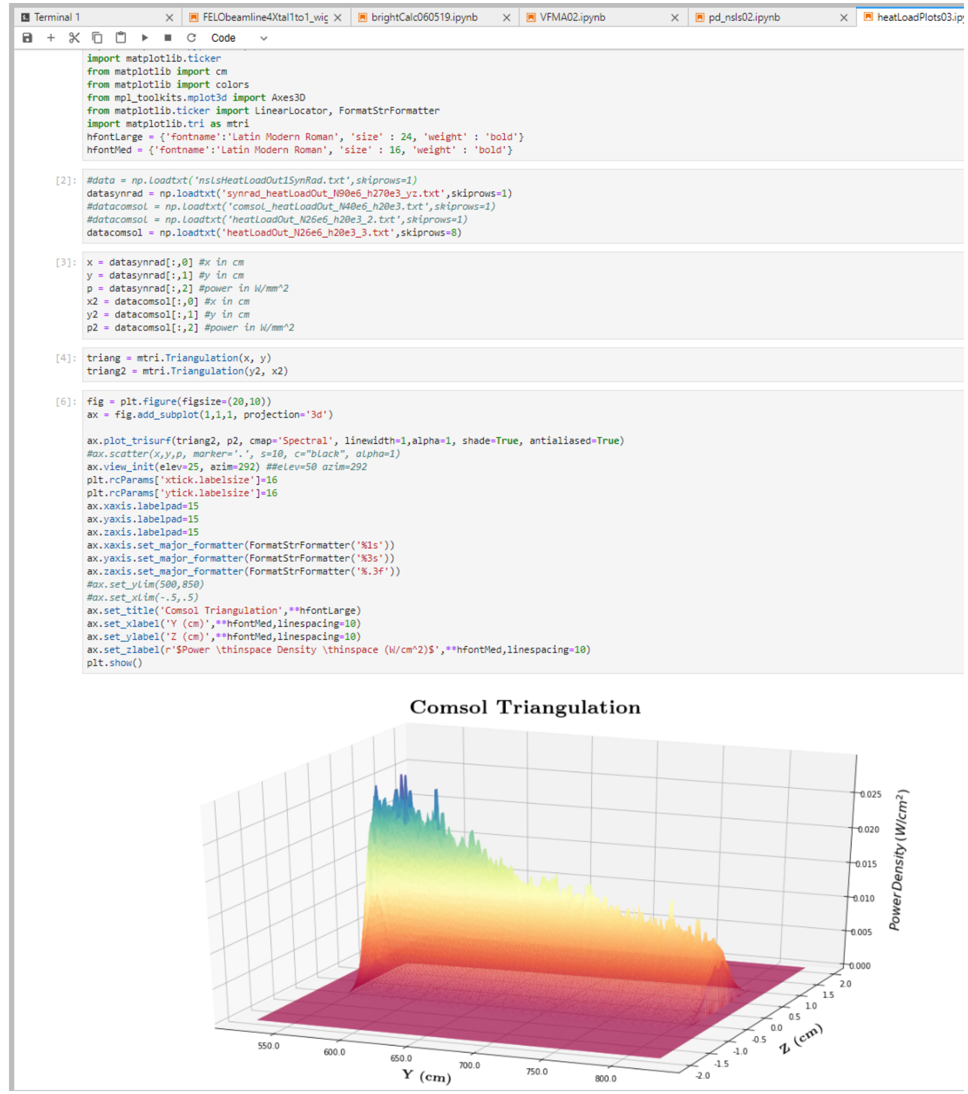
Office of Science

Overview

- *Why RadiaSoft uses Jupyter/Hub*
- *RadiaSoft implementation*
- *Wish list*

Use Case: Comparing Two Codes

- *SynRad and COMSOL*
- *Heat Load*
- *Note: Living Code*

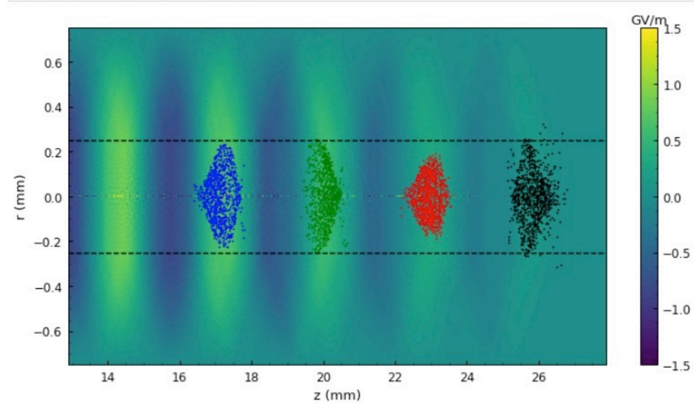


Use Case: In Situ Analysis

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

```
# Wed Jun  5 21:14:24 2019
# 8 processors
# import warp time 0.731453895569 seconds
# For more help, type warphelp()
('l_fftw_fort', False)
Plot file name bunch4-propagate.000.cgm
10000 particles per bunch, with a charge of 5e-09 per bunch
Z:-0.005--0.0025
CFL limit dt: 1.55459106571e-14
New dt: 1.32140240586e-14
 ***  particle simulation package W3D generating
 ---  Resetting 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/m**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
 X-, Y-Debye Wavelengths  =  0.0000E+00,   0.0000E+00 m
    over dx, dy (X and Y) =  0.0000E+00,   0.0000E+00
 Longitudinal thermal velocity (rms) =  0.0000E+00 m/s
    times dt                   =  0.0000E+00 m
    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 perveance =  0.0000E+00
 Characteristic current = -1.7045E+04 amps
```

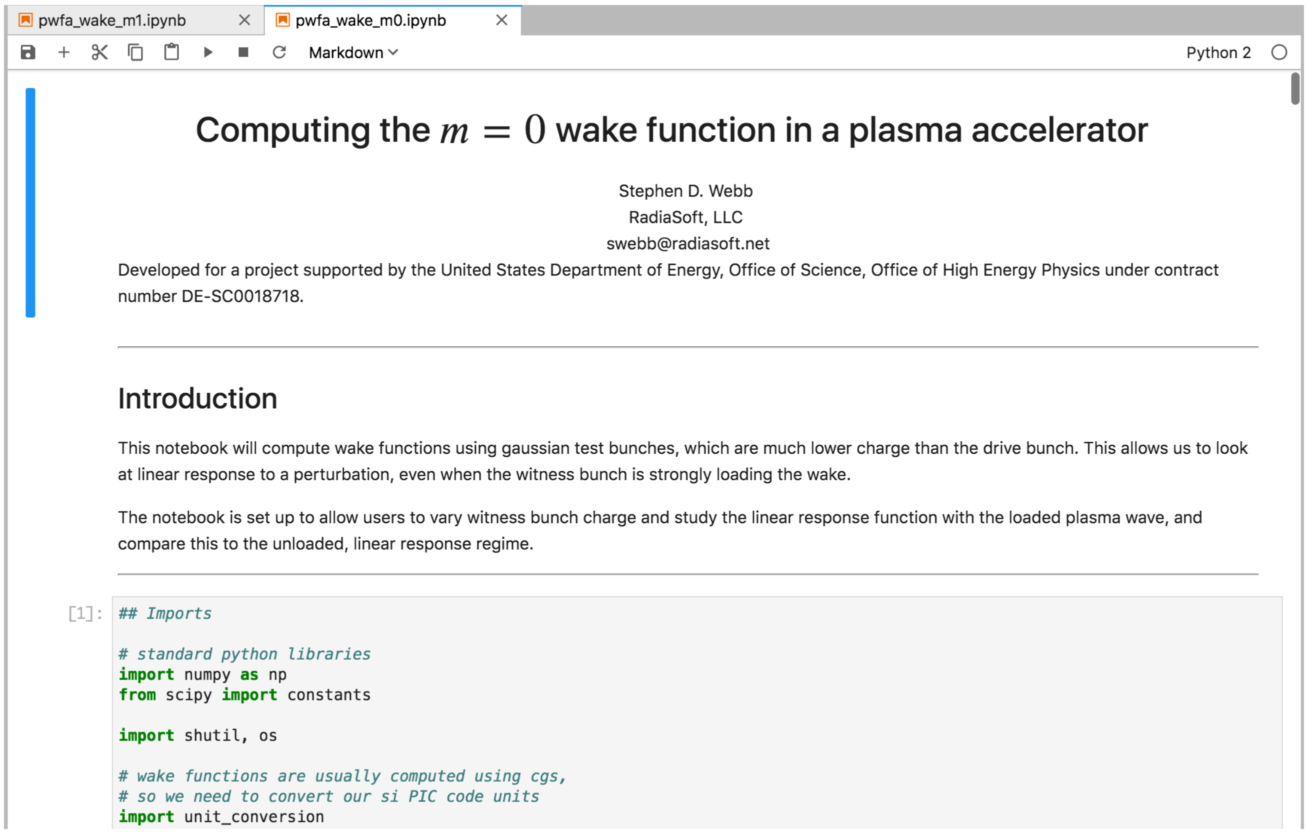
```python
#setup figures and axes
fafig, ax = plt.subplots(figsize=(12,6))

#set initial labels
ax.set_xlim(zmeta.imshow_extent[:2]*z_scale)
ax.set_ylim(zmeta.imshow_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.imshow_extent[0]*z_scale,zmeta.imshow_extent[1]*z_scale,lin

#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.imshow_extent*z_scale,vm

#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')
```




```python
def updatePlot(frame):
```

Use Case: Documentation

"Allow user to vary witness bunch charge and study the linear response function with the loaded plasma wave, and compare this to the unloaded, linear response regime."



pwfa_wake_m1.ipynb | pwfa_wake_m0.ipynb

Markdown | Python 2

Computing the $m = 0$ wake function in a plasma accelerator

Stephen D. Webb
RadiaSoft, LLC
swebb@radiasoft.net

Developed for a project supported by the United States Department of Energy, Office of Science, Office of High Energy Physics under contract number DE-SC0018718.

Introduction

This notebook will compute wake functions using gaussian test bunches, which are much lower charge than the drive bunch. This allows us to look at linear response to a perturbation, even when the witness bunch is strongly loading the wake.

The notebook is set up to allow users to vary witness bunch charge and study the linear response function with the loaded plasma wave, and compare this to the unloaded, linear response regime.

[1]:
```
## Imports

# standard python libraries
import numpy as np
from scipy import constants

import shutil, os

# wake functions are usually computed using cgs,
# so we need to convert our si PIC code units
import unit_conversion
```

Use Case: Teaching

- *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 under James Rosenzweig*
- *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*

Why RadiaSoft Uses Jupyter/Hub

- *In general, Jupyter*
  - *makes it easy to edit and to test models, easier than an IDE like PyCharm (in situ analysis is easier)*
  - *allows us to run and to develop HPC jobs simultaneously*
  - *allows us to develop Python/Fortran/C/C++ code or Python notebooks seamlessly (workflow modularization)*
- *RadiaSoft Jupyter environment*
  - *has all the codes and tools we need to run jobs immediately*
  - *enables technology transfer (teaching, customer deliverables)*
  - *provides easy access to enough cores to run jobs effectively in real-time*

RadiaSoft Jupyter/Hub Environment

- *14 staff users and 46 public users (in last 2 months)*
- *7TB used*
- *Pools: 1 public node, 4 internal nodes*
- *MPI: 13 nodes (pool nodes for workshops)*
- *Nginx proxy*
- *Dev, Alpha, Beta, Prod configurations*

RadiaSoft Jupyter Docker Image

- *JupyterHub compatible to support:*
  - *Accelerator Phyics: elegant, EPICS, FBPIC, JSPEC, OPAL, Radia, Shadow3, SRW, Synergia, Warp, Zgoubi*
  - *Machine Learning: GPy, Keras, scikit-learn, Tensorflow*
  - *Visualization: Pydicom, PyMesh, SciPy, Seaborn, TeX Live, YT*
  - *Integrated Python Environment (pyenv py2:py3)*
- *Takes a long time to build and to pull (10GB)*
- *Supports both Docker and Virtualbox/Vagrant*
- *Curl installer to download and start in single user mode*
- *Jupyter Lab is default GUI*
- *GitHub for authentication*

MPI Jobs

- *Started by user from Jupyter just like* `mpiexec`
- `~/jupyter` *mounted in MPI containers*
- *Users make requests for allocations (infrequently)*
- *Most users only want one or two nodes*
- *Admins run configuration manager for all hosts*
- *Containers running SSHD on MPI nodes*
- *Per user/node SSH/D config for security*
- *Docker host networking with separate VLAN*
- *MPI MCA network config (avoids MPI confusion)*
- *Wrapper abstracts hosts and SSH config for user*
  - `rsmpi -n 10 <command>`
  - `rsmpi -h 1,2 <command>`

RSDockerSpawner

- *DockerSpawner subclass*
- *Managed server pools*
- *Automatic server reallocation*
- *CPU and memory limits*
- *Static port range (iptables)*
- *Host networking (MPI)*
- *`mkdir` for bind mounts*
- *Multi-CA Docker TLS config*
- *State snapshot log*

```
pools:
  default:
    cpu_limit: 0.5
    hosts: [ v3.radia.run ]
    mem_limit: 1G
    min_activity_hours: 1
    servers_per_host: 4
    users: [ ]
  internal:
    hosts:
      - v2.radia.run
      - v5.radia.run
    servers_per_host: 1
    users:
      - bruhwiler
      - robnagler
port_base: 8888
tls_dir: /srv/jupyterhub/tls
```

User Customizations

- *`github.com/radiasoft/jupyter.radiasoft.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.radiasoft.org repo, it runs after global repo*
- *`~/jupyter/bashrc` runs after container's bashrc*
- *`~/jupyter/bin` in path lets users persist commands*

Sharing

- *`~/jupyter` (NFS) is user's home (shared with MPI nodes)*
- *`~/jupyter/workshop-name` for tutorials*
- *Tried `~/public` but is too open esp. for public server*
- *Users share with each other via GitHub and Email*
- *CPU/memory limits allow host (node) sharing*
- *Single notebook server for real-time workspace sharing*

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*

Takeaways

- *Users love Jupyter*
- *Users want all codes pre-installed*
- *Users will consume all available resources*
- *Jupyter/Hub is easily customizable*

Thank You!

Questions?

[*rsl.link/jcw19*](rsl.link/jcw19)