OpenMC Primer

4/30/2021

ChEn 612

Nick Rollins

Discussion Agenda

- Introduction: What is OpenMC?
- Tools and Resources: Teaching a man to fish...
- Basic File Structure: Build your own nuclear reactor
- Homework Tips

Introduction

What is OpenMC?

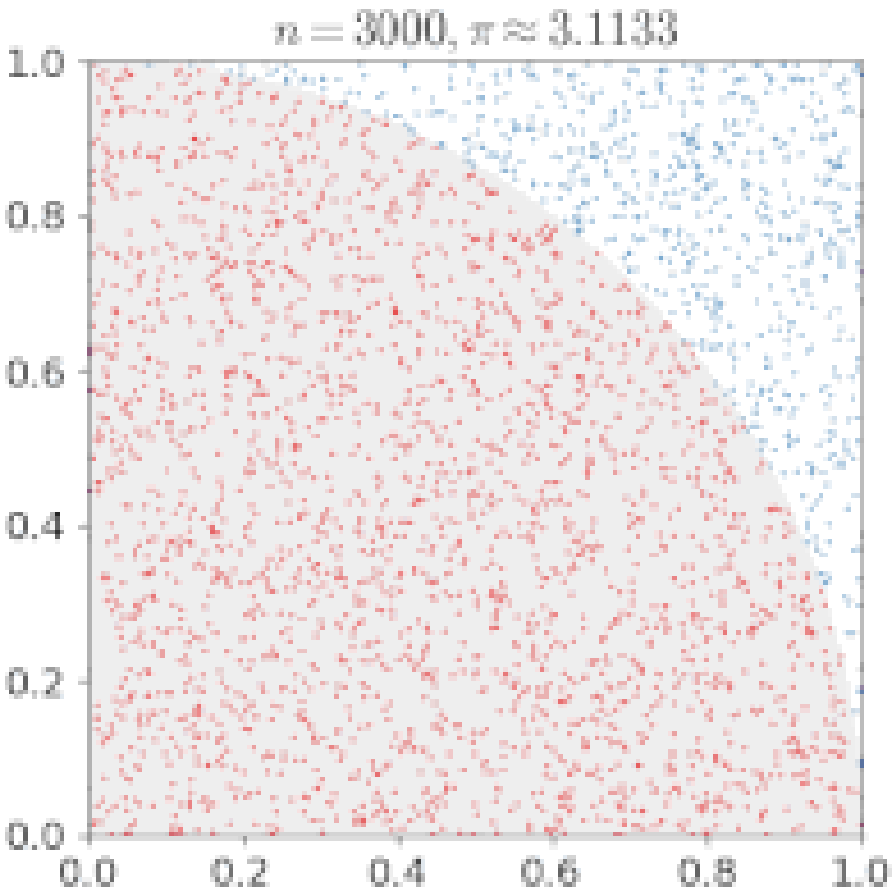
- "OpenMC is an open-source Monte Carlo particle transport code focused on reactor modeling and reactor physics methods research."

1. Used to model neutron, photon, and electron transport/interactions
2. Used to determine critical sizes/configurations of reactor cores
3. Can be used to determine changes in isotopic abundance from fission/absorption (i.e. depletion calculations)

http://web.mit.edu/smharper/www/#:~:text=OpenMC%20is%20an%20open%20source,flexibility%20and%20usability%20in%20mind.

Monte Carlo Method

- "The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches."



Monte Carlo method applied to approximating the value of π.

https://en.wikipedia.org/wiki/Monte_Carlo_method

Why Use OpenMC?

- Model complex reactor geometries
- Powerful post-processing tools
- Used to analyze:

  Criticality

  Shielding

  Radiolysis

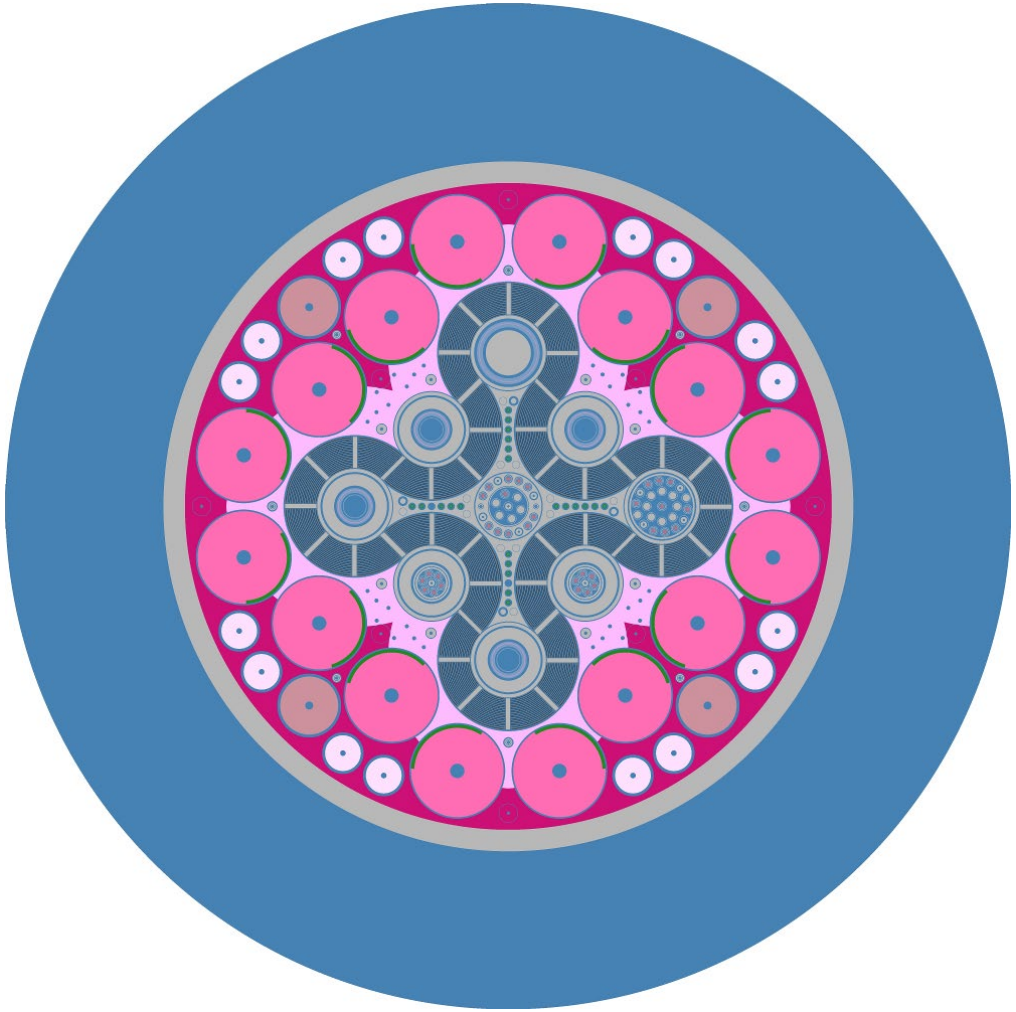
  Breeding

  Feedbacks

  Lifetime

  Etc.

$$\cancel{k = \eta \varepsilon p f p_{FNL} p_{ThNL}}$$

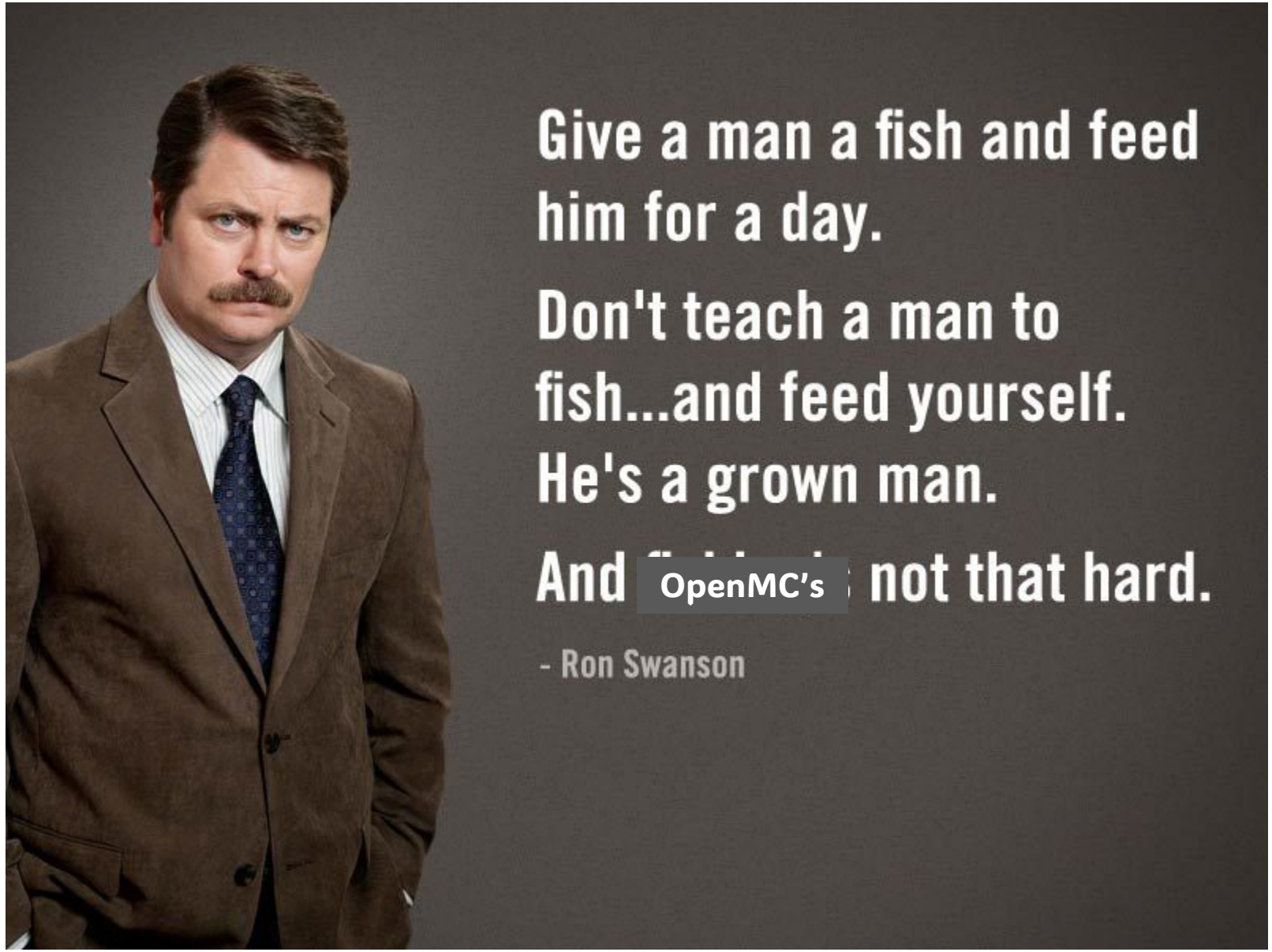


Cross-sectional view of an OpenMC model of the Advanced Test Reactor (ATR)

https://docs.openmc.org/en/stable/usersguide/plots.html

Tools and Resources


peacock
Parks and Recreation


Give a man a fish and feed him for a day.
Don't teach a man to fish...and feed yourself. He's a grown man.
And
not that hard.
- Ron Swanson

OpenMC's

Teaching Yourself OpenMC

- Documentation

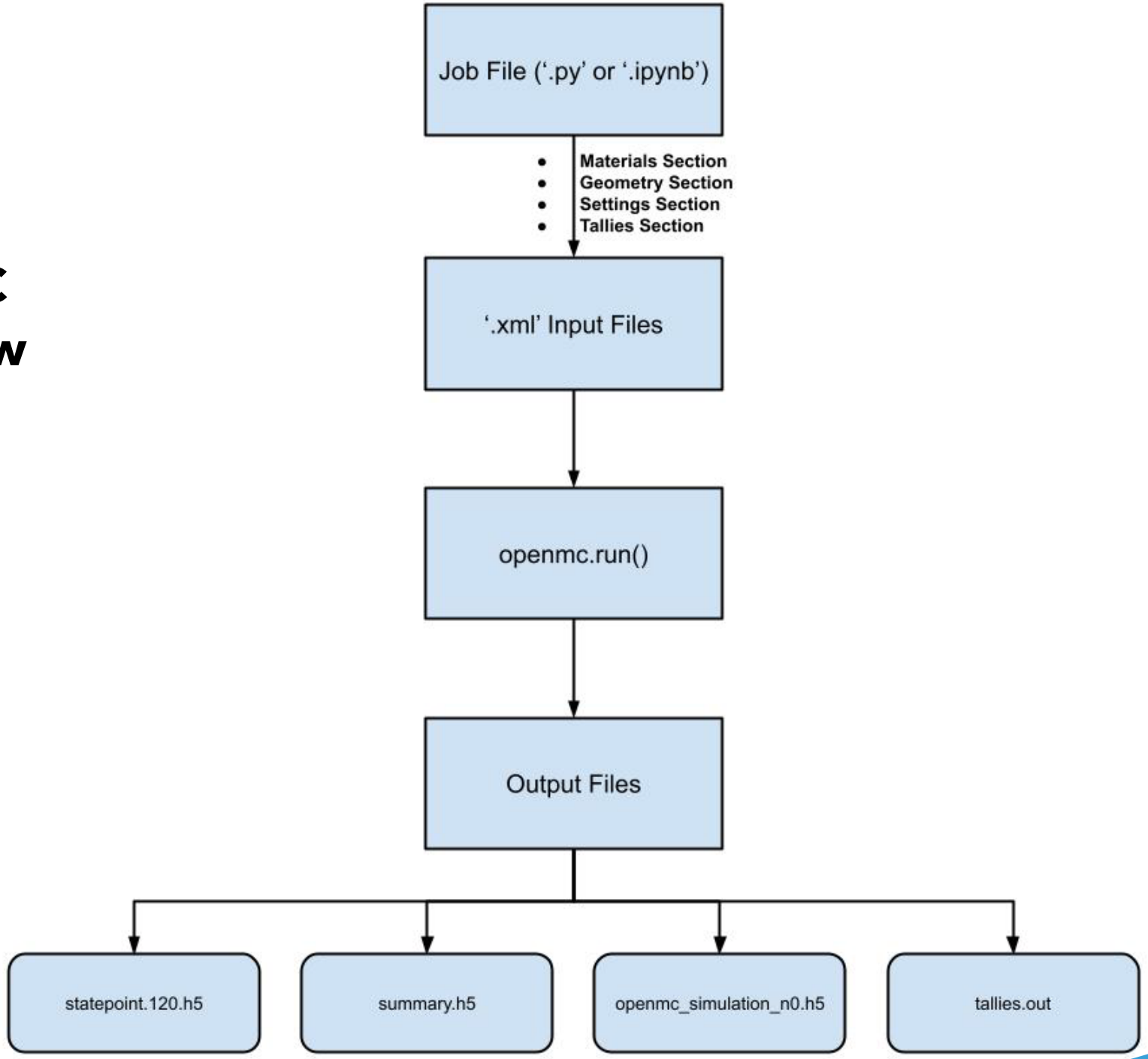
https://docs.openmc.org/en/stable/index.html

- Discourse and User Help

https://openmc.discourse.group/

Basic File Structure and Operation

OpenMC Workflow



Materials

1. Create an openmc.Material object
2. Define composition
    1. ‘ao’ – atom% or mol%, default
    2. ‘wo’ – weight%
3. Set Temperature (in Kelvin)
4. Set density
    1. ‘g/cm3’
    2. ‘kg/m3’

```python
### Begin Materials Section ###

uo2 = openmc.Material(1, "uo2")
uo2.add_nuclide('U235', 0.20)
uo2.add_nuclide('U238', 0.80)
uo2.add_nuclide('O16', 2.0)
uo2.temperature = 900 #K
uo2.set_density('g/cm3', 10.766)


zirconium = openmc.Material(2, "zirconium")
zirconium.add_element('Zr', 1.0, 'wo')
zirconium.temperature = 900 #K
zirconium.set_density('g/cm3', 6.398)


water = openmc.Material(3, "h2o")
water.add_nuclide('H1', 2.0)
water.add_element('O', 1.0)
water.add_s_alpha_beta('c_H_in_H2O')
water.temperature = 635 #K
water.set_density('g/cm3', 0.4907)


mats = openmc.Materials([uo2, zirconium, water])
mats.export_to_xml()

### End Materials Section ###
```

Geometry

1. Create a Surface object (units=cm)
    1. Set boundary (boundary_type='transmission')
        1. 'transmission' (default)
        2. 'vacuum'
        3. 'reflective'
        4. 'white'
        5. 'periodic'
2. Define Half-spaces
3. Define Regions
    1. & - intersection
    2. | - union
    3. ~ - complement
4. Define openmc.Cell object
    1. Specify volumetric Region
    2. Fill with Material
5. Create openmc.Universe object
    1. Fill with Cell objects
6. Create openmc.Geometry object

```python
### Begin Geometry Section ###

sph = openmc.Sphere(r=1.0)

inside_sphere = -sph
outside_sphere = +sph

sphere_in_cell = openmc.Cell(region=inside_sphere, fill=uo2)
sphere_out_cell = openmc.Cell(region=outside_sphere, fill=water)

universe = openmc.Universe(cells=[sphere_in_cell, sphere_out_cell])

geom = openmc.Geometry(universe)
geom.export_to_xml()

### End Geometry Section ###
```

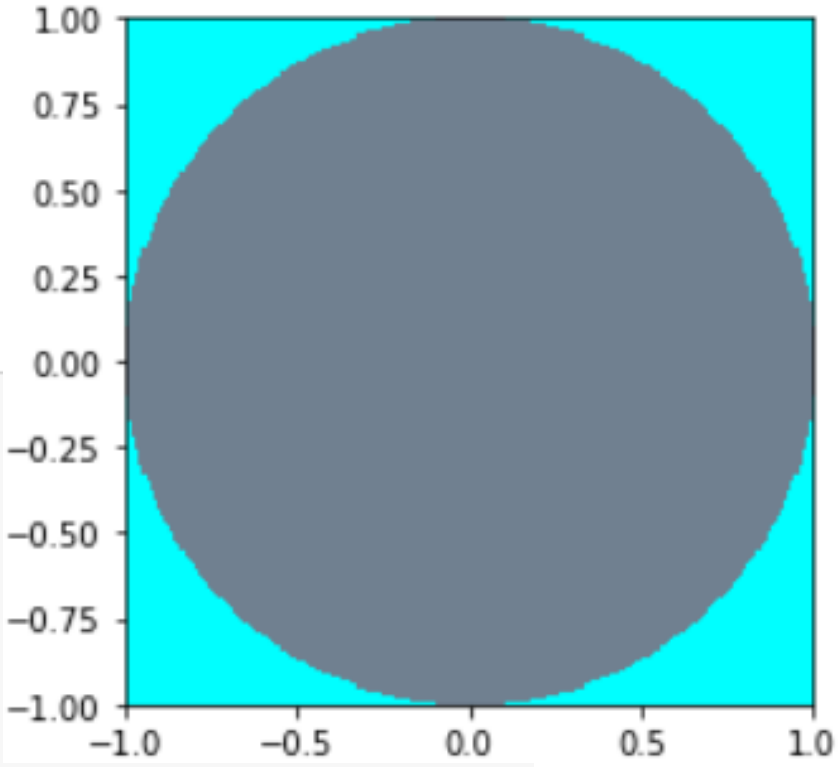
NOTE: Simulation universe *MUST* be finite.

Surfaces

`openmc.Plane`	An arbitrary plane of the form $Ax + By + Cz = D$.
`openmc.XPlane`	A plane perpendicular to the x axis of the form $x - x_0 = 0$
`openmc.YPlane`	A plane perpendicular to the y axis of the form $y - y_0 = 0$
`openmc.ZPlane`	A plane perpendicular to the z axis of the form $z - z_0 = 0$
`openmc.XCylinder`	An infinite cylinder whose length is parallel to the x-axis of the form $(y - y_0)^2 + (z - z_0)^2 = r^2$.
`openmc.YCylinder`	An infinite cylinder whose length is parallel to the y-axis of the form $(x - x_0)^2 + (z - z_0)^2 = r^2$.
`openmc.ZCylinder`	An infinite cylinder whose length is parallel to the z-axis of the form $(x - x_0)^2 + (y - y_0)^2 = r^2$.
`openmc.Sphere`	A sphere of the form $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$.
`openmc.Cone`	A conical surface parallel to the x-, y-, or z-axis.
`openmc.XCone`	A cone parallel to the x-axis of the form $(y - y_0)^2 + (z - z_0)^2 = r^2(x - x_0)^2$.
`openmc.YCone`	A cone parallel to the y-axis of the form $(x - x_0)^2 + (z - z_0)^2 = r^2(y - y_0)^2$.
`openmc.ZCone`	A cone parallel to the x-axis of the form $(x - x_0)^2 + (y - y_0)^2 = r^2(z - z_0)^2$.
`openmc.Quadric`	A surface of the form $Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Jz + K = 0$.

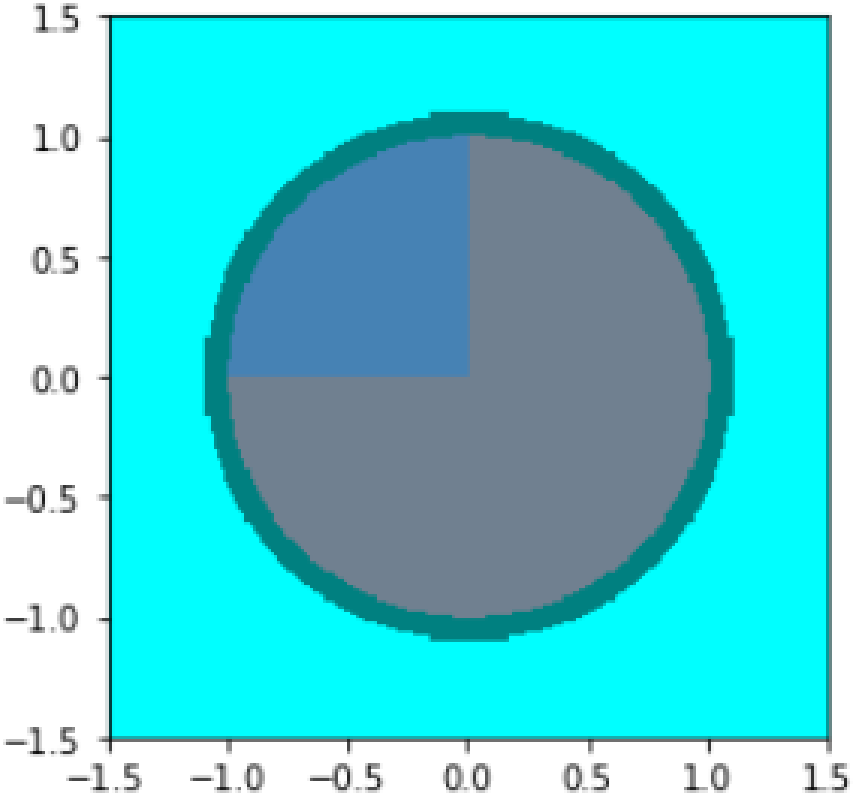
Geometry – Half-spaces

```
sph = openmc.Sphere(r=1.0)

inside_sphere = -sph
outside_sphere = +sph

sphere_in_cell = openmc.Cell(region=inside_sphere, fill=uo2)
sphere_out_cell = openmc.Cell(region=outside_sphere, fill=water)

universe = openmc.Universe(cells=[sphere_in_cell, sphere_out_cell])

univers.plot(basis'xy', origin=(0.0,0.0,0.0), width=(2.0,2.0), pixels=(200,200), color_by='cell')
```



Geometry – Half-spaces cont.

```python
sph_1 = openmc.Sphere(r=1.0)
sph_2 = openmc.Sphere(r=1.1)

xz_plane = openmc.YPlane(y0=0)
yz_plane = openmc.XPlane(x0=0)

NW_sphere = -sph_1 & +xz_plane & -yz_plane
inside_sphere = -sph_1 & ~NW_sphere #OR -sph1 & (-xy_plane | +yz_plane)
shell = +sph_1 & -sph_2
outside_sphere = +sph_2

sphere_NW_cell = openmc.Cell(region=NW_sphere, fill=uo2)
sphere_in_cell = openmc.Cell(region=inside_sphere, fill=uo2)
shell_cell = openmc.Cell(region=shell, fill=zirconium)
sphere_out_cell = openmc.Cell(region=outside_sphere, fill=water)

universe = openmc.Universe(cells=[sphere_NW_cell, sphere_in_cell, shell_cell, sphere_out_cell])

universe.plot(basis='xy', origin=(0.0,0.0,0.0), width=(3.0,3.0), pixels=(200,200), color_by='cell')
```



Periodic Boundary Conditions

1. Only Planes can be periodic
2. Periodic Surfaces *must* be parallel
3. Rotational periodicity is available on 90° segments from one dimensional plane to another (i.e. XZ <-> YZ)

```
boundbox_xlow = openmc.XPlane(x0 = -2.0, boundary_type='periodic')
boundbox_xhigh = openmc.XPlane(x0 = 2.0, boundary_type='periodic')
boundbox_ylow = openmc.YPlane(y0 = -2.0, boundary_type='periodic')
boundbox_yhigh = openmc.YPlane(y0 = 2.0, boundary_type='periodic')
boundbox_zlow = openmc.ZPlane(z0 = -2.0, boundary_type='periodic')
boundbox_zhigh = openmc.ZPlane(z0 = 2.0, boundary_type='periodic')
boundbox_xlow.periodic_surface = boundbox_xhigh
boundbox_ylow.periodic_surface = boundbox_yhigh
boundbox_zlow.periodic_surface = boundbox_zhigh
```

Settings

1. Create openmc.Source object
    1. Spacial source distribution
    2. Source particle
    3. Energy source distribution
2. Create openmc.Settings object
    1. Specify Source
    2. Specify total number of batches
    3. Specify number of inactive batches
    4. Specify number of particles per batch

```python
### Begin Settings Section ###

point = openmc.stats.Point((0, 0, 0))
src = openmc.Source(space=point)

settings = openmc.Settings()
settings.source = src
settings.batches = 120
settings.inactive = 20
settings.particles = 10000
settings.temperature = {'method': 'interpolation'}

settings.export_to_xml()

### End Settings Section ###
```

Tallies

1. Create openmc.Filter objects
    1. SurfaceFilter
    2. CellFilter
    3. EnergyFilter
    4. MaterialFilter
2. Create openmc.Tally object
    1. Specify filters
    2. Specify scores
        1. ‘flux’, ‘absorption’, ‘elastic’, ‘fission’, ‘scatter’, ‘total’, ‘event’, ‘(n,gamma)’, ‘heating-local’, ‘delayed-nu-fission’, ‘prompt-nu-fission’, ‘nu-fission’, etc.
3. Create openmc.Tallies object

```python
### Begin Tallies Section ###

shell_cell_filter = openmc.CellFilter(shell_cell)

shell_tally = openmc.Tally(1, name='Shell Tally')
shell_tally.filters = [shell_cell_filter]
shell_tally.scores = ['absorption']

tallies = openmc.Tallies([shell_tally])

tallies.export_to_xml()

### End Tallies Section ###
```

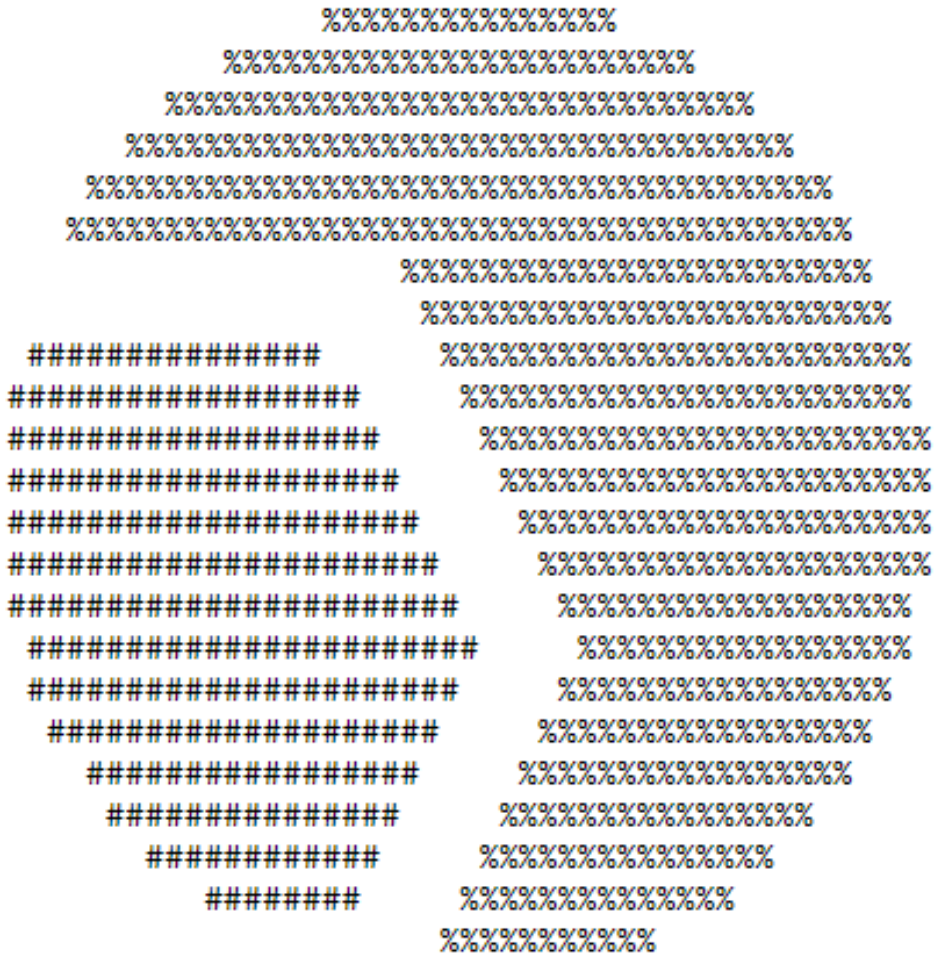
NOTE: ‘tallies.xml’ is the only optional input file.

Tallies - Example

What does this code do?

```python
### Begin Tallies Section ###
from numpy import linspace

mesh = openmc.RectilinearMesh()
mesh.x_grid = linspace(0, 2.5, 11) #cells exist between these points
mesh.y_grid = linspace(0, 2.5, 11)
mesh.z_grid = linspace(0, 2.5, 11)
mesh_filter = openmc.MeshFilter(mesh)

energy_filter = openmc.EnergyFilter([0.0, 1.0, 2.0e6]) #eV

fuel_tally = openmc.Tally(name='Spatial Flux')
fuel_tally.filters = [mesh_filter, energy_filter]
fuel_tally.nuclides = ['U235', 'all']
fuel_tally.scores = ['flux', 'fission']

tallies = openmc.Tallies([fuel_tally])

tallies.export_to_xml()

### End Tallies Section ###
```

```
openmc.run()
```



Understanding OpenMC Output

```
Initializing source particles...

====================>     K EIGENVALUE SIMULATION     <====================

 Bat./Gen.      k            Average k
 =========   ========   ====================
       1/1    0.02102
       2/1    0.01783
       3/1    0.01684
       4/1    0.01633
       5/1    0.01708
       6/1    0.01604
       7/1    0.01725
       8/1    0.01578
       9/1    0.01630
      10/1    0.01705
      11/1    0.01667
      12/1    0.01665
      13/1    0.01652
      14/1    0.01698
      15/1    0.01657
      16/1    0.01642
      17/1    0.01634
      18/1    0.01669
      19/1    0.01650
      20/1    0.01704
      21/1    0.01644
      22/1    0.01670    0.01657 +/- 0.00013
      23/1    0.01642    0.01652 +/- 0.00009
      24/1    0.01640    0.01649 +/- 0.00007
      25/1    0.01662    0.01652 +/- 0.00006
      26/1    0.01620    0.01646 +/- 0.00007
```

```
      114/1    0.01614    0.01653 +/- 0.00004
      115/1    0.01697    0.01653 +/- 0.00004
      116/1    0.01675    0.01654 +/- 0.00004
      117/1    0.01663    0.01654 +/- 0.00004
      118/1    0.01681    0.01654 +/- 0.00004
      119/1    0.01639    0.01654 +/- 0.00004
      120/1    0.01620    0.01653 +/- 0.00004
Creating state point statepoint.120.h5...

=======================>     TIMING STATISTICS     <=======================

Total time for initialization     = 2.6695e+00 seconds
  Reading cross sections          = 2.5328e+00 seconds
Total time in simulation          = 1.9509e+00 seconds
  Time in transport only          = 1.6432e+00 seconds
  Time in inactive batches        = 3.2569e-01 seconds
  Time in active batches          = 1.6252e+00 seconds
  Time synchronizing fission bank = 7.6256e-02 seconds
    Sampling source sites         = 6.1630e-02 seconds
    SEND/RECV source sites        = 1.4109e-02 seconds
  Time accumulating tallies       = 1.7570e-04 seconds
Total time for finalization       = 1.1416e-02 seconds
Total time elapsed                = 4.6510e+00 seconds
Calculation Rate (inactive)       = 614074. particles/second
Calculation Rate (active)         = 615311. particles/second

============================>     RESULTS     <============================

k-effective (Collision)     = 0.01662 +/- 0.00006
k-effective (Track-length)  = 0.01653 +/- 0.00004
k-effective (Absorption)    = 0.01664 +/- 0.00021
Combined k-effective        = 0.01655 +/- 0.00004
Leakage Fraction            = 0.99150 +/- 0.00010
```

Practical Advice & Homework Tips

Copying a file to WSL for use with OpenMC

```
(openmc-env) bclayto4@TABLET-PR939TQS:~$ cd ~/../../mnt/c/Users/brade/Downloads/
```

```
(openmc-env) bclayto4@TABLET-PR939TQS:/mnt/c/Users/brade/Downloads$
```

```
(openmc-env) bclayto4@TABLET-PR939TQS:/mnt/c/Users/brade/Downloads$ cp ChEn_612* ~
```

```
(openmc-env) bclayto4@TABLET-PR939TQS:/mnt/c/Users/brade/Downloads$ cd ~
```

```
(openmc-env) bclayto4@TABLET-PR939TQS:~$
```

```
(openmc-env) bclayto4@TABLET-PR939TQS:~$ ls
 412                                 'MAT HW 10.ipynb'                          TenPlate_Models
 Andrews_Paper_2                     'MAT HW9.ipynb'                            Thermo
 CRE                                  Mambaforge-Linux-x86_64.sh                Untitled.ipynb
 ChEn_612_criticality_example.ipynb   NJOY21                                    Untitled1.ipynb
 Classwork                            Nuc_Mat                                   Untitled2.ipynb
 Files                                PCHEM                                     mambaforge3
 HEX_REACTOR_ATECH                    RADIAL_OUT_PAPER1                         new_test.txt
'HW 311 2.1.ipynb'                    Stats                                     projects
'Heat and MAss'                      'Systems of Equations + fsolve (1).ipynb'  ssh
 Jackson_OUT                         'Systems of Equations + fsolve.ipynb'      test.txt
```

Opening Jupyter Notebook with OpenMC

```
(openmc-env) bclayto4@TABLET-PR939TQS:~$ jupyter-notebook
```

```
[I 08:42:46.472 NotebookApp] Serving notebooks from local directory: /home/bclayto4
[I 08:42:46.473 NotebookApp] Jupyter Notebook 6.4.3 is running at:
[I 08:42:46.473 NotebookApp] http://localhost:8888/?token=b84c27b7ab3efdf694769be5902d15d8f1fa8f87822f8809
[I 08:42:46.474 NotebookApp]  or http://127.0.0.1:8888/?token=b84c27b7ab3efdf694769be5902d15d8f1fa8f87822f8809
[I 08:42:46.474 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 08:42:46.519 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/bclayto4/.local/share/jupyter/runtime/nbserver-191-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=b84c27b7ab3efdf694769be5902d15d8f1fa8f87822f8809
     or http://127.0.0.1:8888/?token=b84c27b7ab3efdf694769be5902d15d8f1fa8f87822f8809
```

Homework

- https://www.et.byu.edu/~mjm82/che612/Winter2024/Homework/homework.html

OpenMC Day 2

Spiritual Thought

Discussion Outline

1) Homework Questions
2) OpenMC Tools
3) Homework Help (If time allows)

OpenMC Tools

OpenMC has many tools at out disposable. For the Design of a Reactor, these tools will be most useful

- Generating Flux and Power Profiles
- Depletion Calculations
- Approximating Feedbacks

Flux and Power Profile Set Up

Flux and Fission Tallies

```
mesh = openmc.RectilinearMesh()
mesh.x_grid = linspace(-80.111, 80.111, 1001) #cells exist between these points
mesh.y_grid = linspace(-80.111, 80.111, 1001)
mesh.z_grid = linspace(0, 2*st + 4*mt + 2*dt, 31)
mesh_filter = openmc.MeshFilter(mesh)

tally = openmc.Tally(name='spatial flux')
tally.filters = [mesh_filter]
tally.scores = ['flux', 'fission','heating-local']
print("><><><><><><><><><><><><><><><><><><><><><><")
print("Tally Refrenced: Flux")
print("><><><><><><><><><><><><><><><><><><><><><><")
tallies = openmc.Tallies([tally])
```

- Why Would knowing the Flux or Power Profile be important?

Heating Local

```
mesh = openmc.RectilinearMesh()
mesh.x_grid = linspace(-80.111, 80.111, 4) #cells exist between these points
mesh.y_grid = linspace(-80.111, 80.111, 4)
mesh.z_grid = linspace(0, 2*st + 4*mt + 2*dt, 2)
mesh_filter = openmc.MeshFilter(mesh)

tally = openmc.Tally(name='spatial flux')
tally.filters = [mesh_filter]
tally.scores = ['heating-local']
print("><><><><><><><><><><><><><><><><><><><><><><")
print("Tally Refrenced: Heating")
print("><><><><><><><><><><><><><><><><><><><><><><")
tallies = openmc.Tallies([tally])
```

- What is the Heating Local Assumption?

Normalizing Tallies

the `heating` score over the entire system. This score provides the heating rate in units of [eV/source], which we'll denote H. Then, calculate the heating rate in J/source as

$$H' = 1.602 \times 10^{-19} \left[\frac{\mathrm{J}}{\mathrm{eV}}\right] \cdot H \left[\frac{\mathrm{eV}}{\mathrm{source}}\right].$$

Dividing the power by the observed heating rate then gives us a normalization factor that can be applied to other tallies:

$$f = \frac{P}{H'} = \frac{[\mathrm{J/s}]}{[\mathrm{J/source}]} = \left[\frac{\mathrm{source}}{\mathrm{s}}\right].$$

Multiplying by the normalization factor and dividing by volume, we can then get the flux in typical units:

$$\phi' = \frac{f\phi}{V} = \frac{[\mathrm{source/s}][\mathrm{particle\text{-}cm/source}]}{[\mathrm{cm}^3]} = \left[\frac{\mathrm{particle}}{\mathrm{cm}^2 \cdot \mathrm{s}}\right]$$

There are several slight variations on this procedure:

```
import h5py
import openmc
import matplotlib.pyplot as plt
import numpy as np
%matplotlib notebook
```

```
# Load the statepoint file
sp_file = 'radial_sanscntrl_HastX_'
sp = openmc.StatePoint( sp_file + 'H.sp.h5')

tally = sp.get_tally(name='spatial flux')
print(tally)

heat = tally.get_slice(scores=['heating-local'])
print(heat)
```

```
H = float(np.mean(heat.mean)) #eV/source
H_prime = 1.60218E-19*H #J/source
num_plates = 90
dem = num_plates/2*4
P = 45.0E6/dem #W; total power of reactor slice
f = P/H_prime #source/s

print(f)
```

Processing the Data

Pull Data

```
sp = openmc.StatePoint( sp_file + 'F.sp.h5')
tally = sp.get_tally(scores=['flux'])
print(tally)

flux = tally.get_slice(scores=['flux'])
fission = tally.get_slice(scores=['fission'])
print(heat)
```

Rearrange pt1

```
#restructure array
numcells_x = 500
numcells_y = 500
numcells_z = 30

heat_array = np.ones((numcells_x, numcells_y, numcells_z))
flux_array = np.ones((numcells_x, numcells_y, numcells_z))
fission_array = np.ones((numcells_x, numcells_y, numcells_z))
i = j = k = 0
for n in range(len(flux.mean)):
    flux_array[i][j][k] = float(flux.mean[n])
    fission_array[i][j][k] = float(fission.mean[n])
    i += 1
    if i >= numcells_x:
        i = 0
        j += 1
    if j >= numcells_y:
        j = 0
        k += 1
    if k >= numcells_z:
        k = 0
print('done')
```

Rearrange pt2

```
#re-organize dimensions (x, y, z) -> (z, y, x)
flux_array_new = np.ones((numcells_z, numcells_y, numcells_x))
fission_array_new = np.ones((numcells_z, numcells_y, numcells_x))
for i in range(len(flux_array)):
    for j in range(len(flux_array[i])):
        for k in range(len(flux_array[i][j])):
            flux_array_new[k][j][i] = flux_array[i][j][k] #converted from particle-cm/source to particle/(cm^2*s)
            fission_array_new[k][j][i] = fission_array[i][j][k]
print('done')
```

Normalize

```
vol_cell = (80.111/numcells_x)*(80.111/numcells_y)*(5.2/numcells_z) #cm^3/cell
P_fission = 3.171E-11 #J/fission_U233

flux_farray = np.ones((numcells_z, numcells_y, numcells_x))
power_farray = np.ones((numcells_z, numcells_y, numcells_x))
for n in range(len(flux_array_new)):
    for m in range(len(flux_array_new[n])):
        for p in range(len(flux_array_new[n][m])):
            flux_farray[n][m][p] = f*flux_array_new[n][m][p]/vol_cell #particle/(cm^2*s)
            power_farray[n][m][p] = f*fission_array_new[n][m][p]*P_fission/vol_cell #J/s/cm^3
print('done')
```

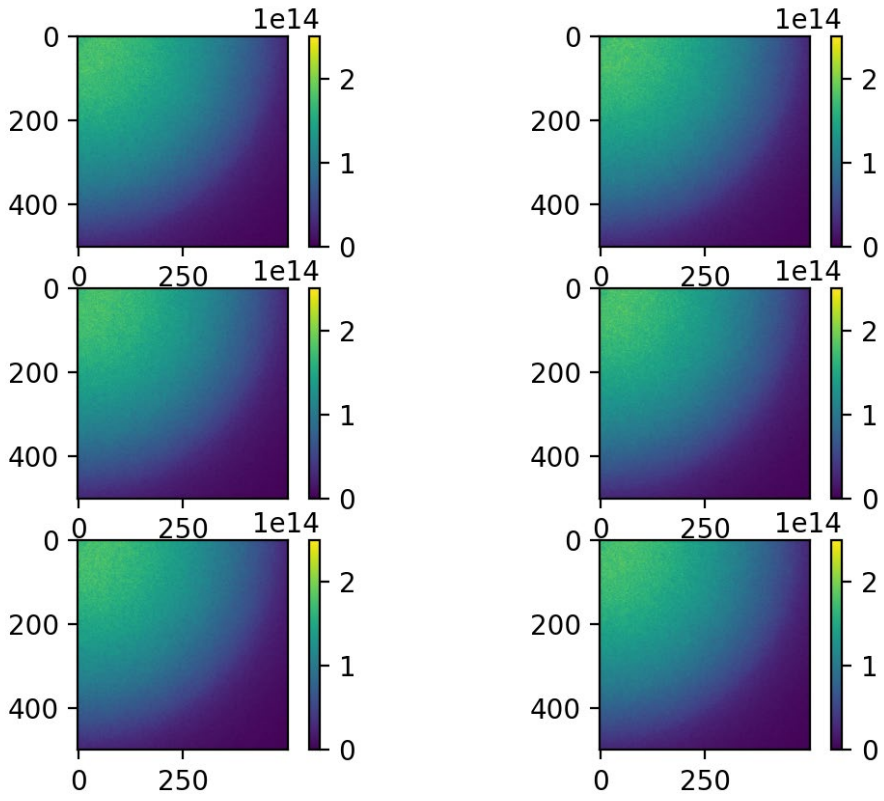
Output Means

```
print(sp_file)
print(np.mean(flux_farray), 'particles/(cm^2*s)')
print(np.mean(fission_array_new), 'fissions/source')
print(np.mean(power_farray), 'W/cm^3')
print('flux/source',np.mean(flux_array_new), 'particle-cm/source')
```

```
radial_sanscntrl_HastX_
81322604347261.19 particles/(cm^2*s)
5.692323567019851e-08 fissions/source
7.412980025246132 W/cm^3
flux/source 1.9801781732634628e-05 particle-cm/source
```

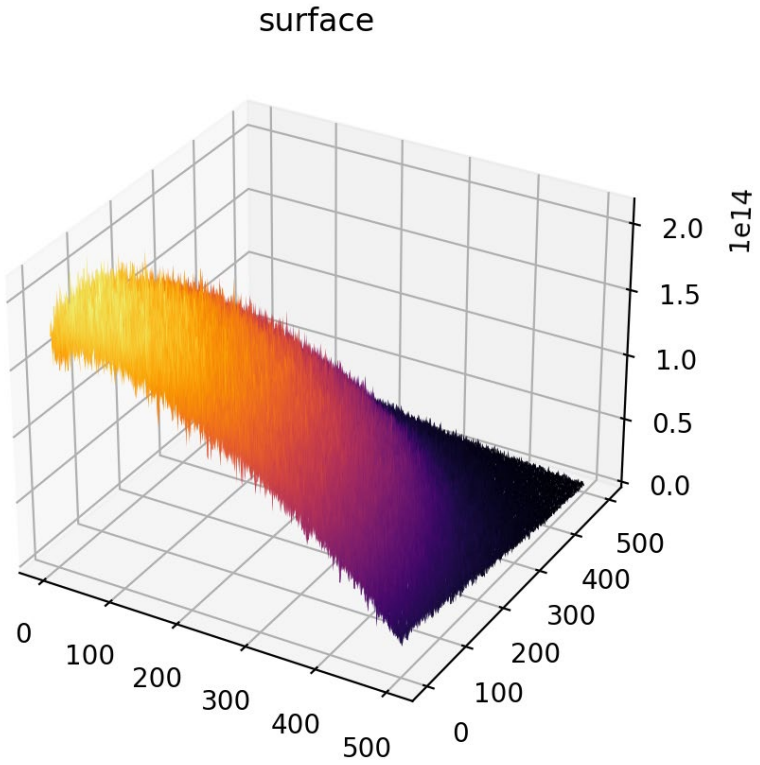
Graphing: Flux

```
#true flux
fig11 = plt.subplot(321)
im11 = fig11.imshow(flux_farray[0],vmin = 0,vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im11)
fig21 = plt.subplot(322)
im21 = fig21.imshow(flux_farray[5],vmin = 0, vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im21)
fig31 = plt.subplot(323)
im31 = fig31.imshow(flux_farray[13],vmin = 0, vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im31)
fig41 = plt.subplot(324)
im41 = fig41.imshow(flux_farray[15],vmin = 0, vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im41)
fig51 = plt.subplot(325)
im51 = fig51.imshow(flux_farray[17],vmin = 0, vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im51)
fig61 = plt.subplot(326)
im61 = fig61.imshow(flux_farray[29],vmin = 0, vmax = 2.5e14) #particle/(cm^2*s)
plt.colorbar(im61)
```



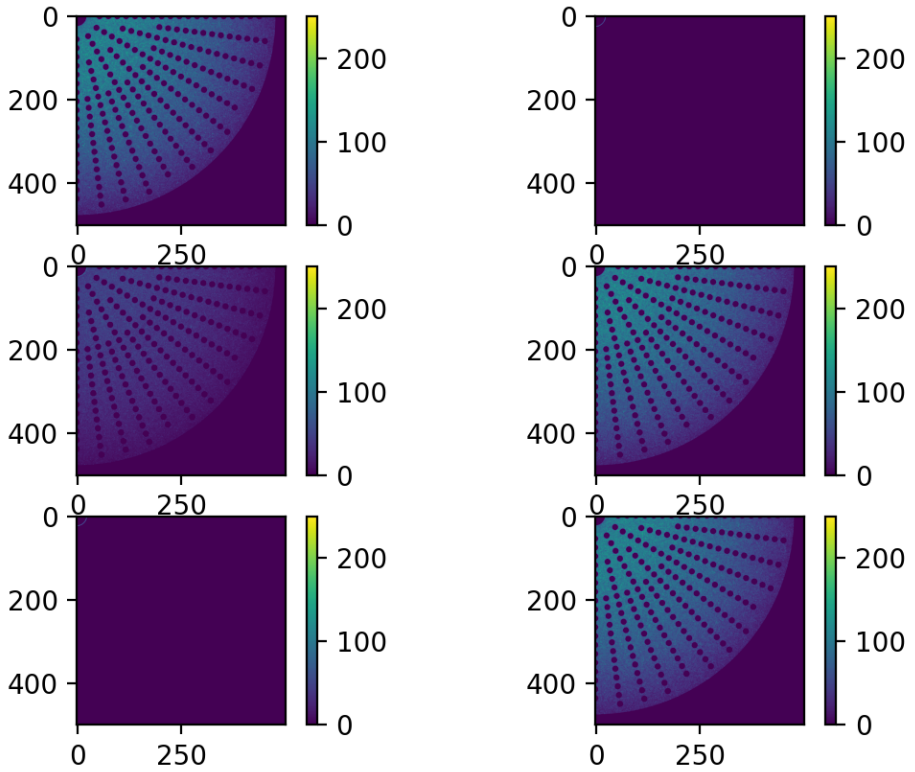
Graphing: Flux

```
fig = plt.figure()
ax = plt.axes(projection='3d')

x = np.linspace(0, 500, 500)
y = np.linspace(0, 500, 500)

X, Y = np.meshgrid(x, y)
Z = flux_farray[15]
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1
                ,
                cmap='inferno', edgecolor='none')
ax.set_title('surface');
```



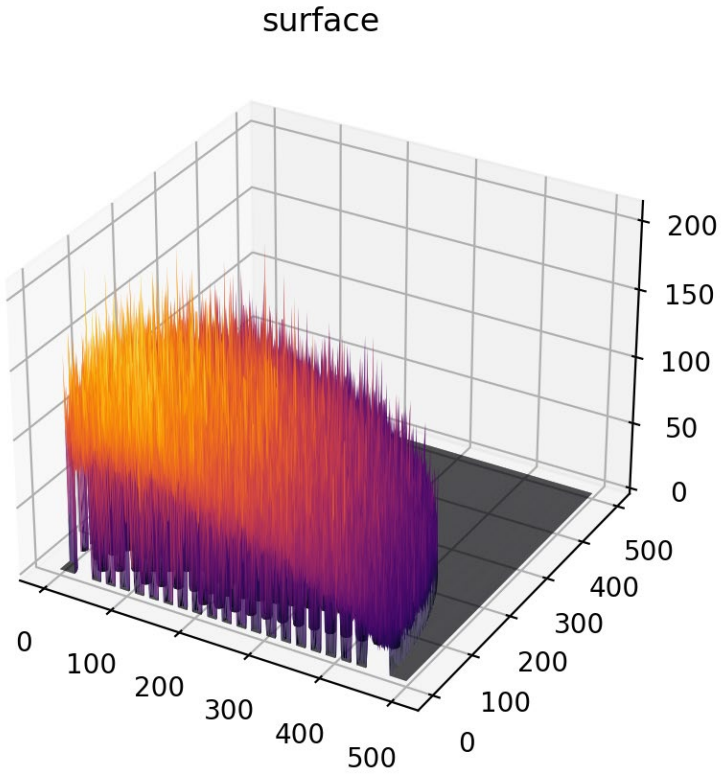
Graphing: Power

```
#power
fig12 = plt.subplot(321)
im12 = fig12.imshow(power_farray[0], vmin=0,vmax=250) #W
plt.colorbar(im12)
fig22 = plt.subplot(322)
im22 = fig22.imshow(power_farray[5],vmin=0,vmax=250) #W
plt.colorbar(im22)
fig32 = plt.subplot(323)
im32 = fig32.imshow(power_farray[13],vmin=0,vmax=250) #W
plt.colorbar(im32)
fig42 = plt.subplot(324)
im42 = fig42.imshow(power_farray[15],vmin=0,vmax=250) #W
plt.colorbar(im42)
fig52 = plt.subplot(325)
im52 = fig52.imshow(power_farray[17],vmin=0,vmax=250) #W
plt.colorbar(im52)
fig62 = plt.subplot(326)
im62 = fig62.imshow(power_farray[29],vmin=0,vmax=250) #W
plt.colorbar(im62)
```



Graphing: Power

```
PP = power_farray[15]
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, PP, rstride=1, cstride=1
                ,
                cmap='inferno', edgecolor='none')
ax.set_title('surface');
```



Depletion/Burnup

1. Set desired materials to “depletable”
2. Define path to depletion chain ‘.xml’
3. Create openmc.deplete.Operator
4. Create integrator
   1. Specify power (W)
   2. Specify step size (s)
   3. Specify number of steps
5. Run OpenMC

How do we find the volume for complex Geometries?

```python
### Begin Depletion Section ###

import openmc.deplete

chainpath = "/home/username/openmc/Cross_Section_Libraries/endfb71_hdf5/chain_endfb71_sfr.xml" #example path

chain = openmc.deplete.Chain.from_xml(chainpath)

operator = openmc.deplete.Operator(geometry, settings, chainpath)

power = 3.0e9 #W; power rating for the reactor

max_step = 2 * operator.heavy_metal / power * 1e3 #days; heavy_metal is mass of fuel atom in kg

time_steps = 4*[max_step * 24 * 60 * 60] #days -> s

integrator = openmc.deplete.PredictorIntegrator(operator, time_steps, power)

integrator.integrate()

### End Depletion Section ###
```

```python
uo2 = openmc.Material(1, "uo2")
uo2.add_nuclide('U235', 0.03)
uo2.add_nuclide('U238', 0.97)
uo2.add_nuclide('O16', 2.0)
uo2.temperature = 1000 #K
uo2.set_density('g/cm3', 10.0)
uo2.volume = 4.14072 #cm^3
uo2.depletable = True
```

Volume Calculations

1. Create openmc.VolumeCalculation object
    1. Specify domains (materials or cells)
    2. Specify number of samples (affects accuracy)
    3. Specify volume bounds

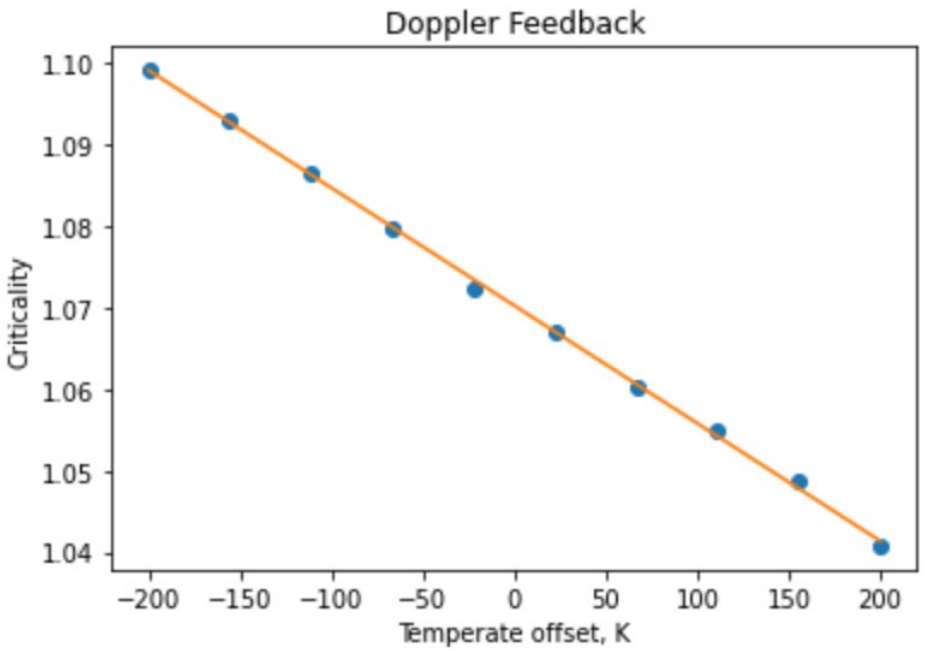
```python
### Begin VolumeCalculation Section ###

vol = openmc.VolumeCalculation(domains=[uo2], \
                               samples=int(1e5), \
                               lower_left=[-3.0, -3.0, -3.0], \
                               upper_right=[3.0, 3.0, 3.0])
settings = openmc.Settings()
settings.volume_calculations = [vol]
settings.temperature = {'method': 'interpolation'}
settings.output = {'tallies': False}
settings.export_to_xml()


openmc.calculate_volumes()

### End VolumeCalculation Section ###
```

Feedback Coefficients

- Generally, any average value for a reactor in units of (worth change/unit change)
- How could I approximate…
  - Fuel Temperature Feedback?
  - Moderator Temperature Feedback?
  - Moderator Void Feedback?
  - Poison Concentration Feedback?

$$\rho = \frac{k_{eff} - 1}{k_{eff}}$$

Feedbacks: Doppler

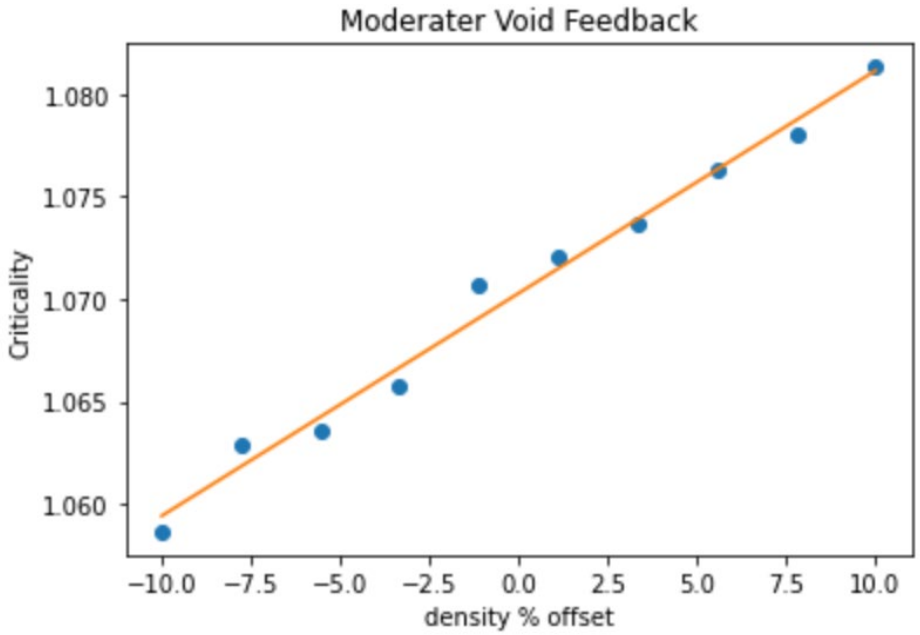



```python
def feedback(Tdev=0,T_pert_fuel=0,T_pert_mod=0,rho_mod=1,rho_cool=1):
    wt_poison = 0 #20% run

    with open('./mgst/temp.py','r') as temp:
        exec(temp.read())

    with open('./mgst/mat.py','r') as mat:
        exec(mat.read())

    with open('./mgst/geom.py','r') as geom:
        exec(geom.read())

    with open('./mgst/settings.py','r') as settings:
        exec(settings.read())

    if TalEV == 0:
        with open('./mgst/flux.py','r') as flux:
            exec(flux.read())
    elif TalEV == 1:
        with open('./mgst/heat.py','r') as heat:
            exec(heat.read())

    #print("true U233 wo%:",xpu)

    ### Begin Export Section ###

    geometry.export_to_xml()

    settings.export_to_xml()

    if TalEV == 0 or TalEV == 1:
        tallies.export_to_xml()
        print("tallly made")
    else:
        print("no tally made")

    materials.export_to_xml()

    ### End Export Section ###


    openmc.run()

nruns = 10
Tlist = np.linspace(-200,200,nruns)
klist = np.zeros(nruns)
erlist = np.zeros(nruns)


for i in range(nruns):
    feedback(0,Tlist[i],0)
    #openmc.run()

    results = h5py.File('./statepoint.120.h5','r')
    #print(results.keys())
    #print(results.get('k_combined')[0]) #0 is keff, 1 is error
    klist[i] = results.get('k_combined')[0]
    erlist[i] = results.get('k_combined')[1]
    results.close()

###loop through temps, loop through openmc runs, store keffs - need to extract keff from .h5


data = np.array([klist,erlist,Tlist])

np.savetxt('Fuel Temp Coefficient.out', data, delimiter=',')
```

Feedbacks: Doppler

```
salt.temperature = T = 1100 + Tdev + T_pert_fuel #K
salt.set_density('kg/m3', density =  trho(T,xu,xp)) #FLiNaK at 956 K
salt.volume = 16328.8
salt.depletable = True
```

Feedbacks: Moderator Void




```python
import poison_def
poison = poison_def.set_poison()
print('the working poison is',poison.name)
Tdev = 0
T_pert_fuel = 0
T_pert_mod = 0
rho_mod = 1
rho_cool = 1

### End Poison Section ###

def feedback(Tdev=0,T_pert_fuel=0,T_pert_mod=0,rho_mod=1,rho_cool=1):
    wt_poison = 0 #20% run

    with open('./mgst/mat.py','r') as mat:
        exec(mat.read())

    with open('./mgst/geom.py','r') as geom:
        exec(geom.read())

    with open('./mgst/settings.py','r') as settings:
        exec(settings.read())

    with open('./mgst/tallies.py','r') as tallies:
        exec(tallies.read())

    openmc.run()


nruns = 10
Tlist = np.linspace(-200,200,nruns)
klist = np.zeros(nruns)
erlist = np.zeros(nruns)
rholist = np.linspace(0.9,1.1,nruns)

for i in range(nruns):
    feedback(rho_mod=rholist[i])
    #openmc.run()

    results = h5py.File('./statepoint.120.h5','r')
    #print(results.keys())
    #print(results.get('k_combined')[0]) #0 is keff, 1 is error
    klist[i] = results.get('k_combined')[0]
    erlist[i] = results.get('k_combined')[1]
    results.close()

###loop through temps, loop through openmc runs, store keffs - need to extract keff from .h5


data = np.array([klist,erlist,Tlist])

np.savetxt('Moderator_void_feedback.out', data, delimiter=',')
```

Feedbacks: Moderator Void

```
moderator = openmc.Material(6, name="Moderator")
moderator.add_element('C', 1.0)
#moderator.add_s_alpha_beta('c_Graphite')     #bound atom cross sections
moderator.temperature = 918 + Tdev + T_pert_mod#K
moderator.set_density('g/cm3', density = rho_mod*(3.51/(1 + 3*(3.4833333e-9*moderator.temperature + 8.901941667e-7)*(moderator.temperature - 298.15)))) #Diamond at Temp
```