OpenMC Primer

4/30/2021
ChEn 612
Nick Rollins
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)

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.

[k = \eta \epsilon p f p_{ENL} P_{TNL}]

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

OpenMC Workflow

Job File (`.py` or `.ipynb`)
- Materials Section
- Geometry Section
- Settings Section
- Tally Section

`.xml` Input Files

openmc.run()

Output Files
- statpoint.120.h5
- summary.h5
- openmc_simulation_n0.h5
- tallies.out
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.

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

```python
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 \textit{must} be parallel

3. Rotational periodicity is available on 90° segments from one dimensional plane to another (i.e. XZ <-> YZ)
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

3. Create openmc.Tallies object

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 ###
```
Understanding OpenMC Output

Initializing source particles...

<table>
<thead>
<tr>
<th>Bat./Gen.</th>
<th>k</th>
<th>Average k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>0.0102</td>
<td></td>
</tr>
<tr>
<td>2/1</td>
<td>0.01783</td>
<td></td>
</tr>
<tr>
<td>3/1</td>
<td>0.01684</td>
<td></td>
</tr>
<tr>
<td>4/1</td>
<td>0.01633</td>
<td></td>
</tr>
<tr>
<td>5/1</td>
<td>0.01708</td>
<td></td>
</tr>
<tr>
<td>6/1</td>
<td>0.01604</td>
<td></td>
</tr>
<tr>
<td>7/1</td>
<td>0.01725</td>
<td></td>
</tr>
<tr>
<td>8/1</td>
<td>0.01578</td>
<td></td>
</tr>
<tr>
<td>9/1</td>
<td>0.01630</td>
<td></td>
</tr>
<tr>
<td>10/1</td>
<td>0.01705</td>
<td></td>
</tr>
<tr>
<td>11/1</td>
<td>0.01667</td>
<td></td>
</tr>
<tr>
<td>12/1</td>
<td>0.01665</td>
<td></td>
</tr>
<tr>
<td>13/1</td>
<td>0.01652</td>
<td></td>
</tr>
<tr>
<td>14/1</td>
<td>0.01698</td>
<td></td>
</tr>
<tr>
<td>15/1</td>
<td>0.01657</td>
<td></td>
</tr>
<tr>
<td>16/1</td>
<td>0.01642</td>
<td></td>
</tr>
<tr>
<td>17/1</td>
<td>0.01634</td>
<td></td>
</tr>
<tr>
<td>18/1</td>
<td>0.01660</td>
<td></td>
</tr>
<tr>
<td>19/1</td>
<td>0.01650</td>
<td></td>
</tr>
<tr>
<td>20/1</td>
<td>0.01704</td>
<td></td>
</tr>
<tr>
<td>21/1</td>
<td>0.01644</td>
<td></td>
</tr>
<tr>
<td>22/1</td>
<td>0.01670</td>
<td>0.01657 +/- 0.00013</td>
</tr>
<tr>
<td>23/1</td>
<td>0.01642</td>
<td>0.01652 +/- 0.00009</td>
</tr>
<tr>
<td>24/1</td>
<td>0.01646</td>
<td>0.01649 +/- 0.00007</td>
</tr>
<tr>
<td>25/1</td>
<td>0.01662</td>
<td>0.01652 +/- 0.00006</td>
</tr>
<tr>
<td>26/1</td>
<td>0.01620</td>
<td>0.01646 +/- 0.00007</td>
</tr>
</tbody>
</table>

Creating state point statepoint.120.h5...

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMING STATISTICS</td>
<td></td>
</tr>
<tr>
<td>Total time for initialization</td>
<td>= 2.6695e+00 seconds</td>
</tr>
<tr>
<td>Reading cross sections</td>
<td>= 2.5328e+00 seconds</td>
</tr>
<tr>
<td>Total time in simulation</td>
<td>= 1.9509e+00 seconds</td>
</tr>
<tr>
<td>Time in transport only</td>
<td>= 1.6422e+00 seconds</td>
</tr>
<tr>
<td>Time in inactive batches</td>
<td>= 3.2509e-01 seconds</td>
</tr>
<tr>
<td>Time in active batches</td>
<td>= 1.6252e+00 seconds</td>
</tr>
<tr>
<td>Time synchronizing fission bank</td>
<td>= 7.6256e-02 seconds</td>
</tr>
<tr>
<td>Sampling source sites</td>
<td>= 6.1630e-02 seconds</td>
</tr>
<tr>
<td>SEND/RECV source sites</td>
<td>= 1.4109e-02 seconds</td>
</tr>
<tr>
<td>Time accumulating tallies</td>
<td>= 1.7578e-04 seconds</td>
</tr>
<tr>
<td>Total time for finalization</td>
<td>= 1.1416e-02 seconds</td>
</tr>
<tr>
<td>Total time elapsed</td>
<td>= 4.6510e+00 seconds</td>
</tr>
<tr>
<td>Calculation Rate (Inactive)</td>
<td>= 614074. particles/second</td>
</tr>
<tr>
<td>Calculation Rate (Active)</td>
<td>= 615311. particles/second</td>
</tr>
</tbody>
</table>

RESULTS

| k-effective (Collision)   | 0.01662 +/- 0.00006 |
| k-effective (Track-length) | 0.01653 +/- 0.00004 |
| k-effective (Absorption)   | 0.01664 +/- 0.000021 |
| Combined k-effective      | 0.01655 +/- 0.00004 |
| Leakage Fraction          | 0.99150 +/- 0.00010 |
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
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 ###
```
Next Time: 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?