Infinite slab

August 21, 2017

1 Infinite slab

```
In [2]: %matplotlib inline
        from mayavi import mlab
        import openmc
        import numpy as np
        from IPython.display import Image
        import matplotlib.pyplot as plt
        import matplotlib.cm as cm
        import glob
        from math import pi
        from math import floor
```

We will use OpenMC for all the simulations. First we have to define the materials that we will use.

```
In [30]: o16 = openmc.Element("O")
         h1 = openmc.Element("H")
         zr = openmc.Nuclide("Zr96")  # I changed the cross sections file so that Zr96 actually

         ideal_xs = openmc.Material(material_id = 2, name='Ideal_xs')
         ideal_xs.set_density('g/cm3', 6.55)
         ideal_xs.add_nuclide(zr, 7.2758e-3)

         #absorber_1v = openmc.Material(material_id = 3, name = "1/v absorber")
         #absorber_1v.set_density('g/cm3', 6.55)
         #absorber_1v.add_nuclide(zr, 7.2758e-3)

         #water = openmc.Material( material_id = 1,name = "water")
         #water.add_element(o16, 1.0)
         #water.add_element(h1, 2.0)
         #water.set_density("g/cm3", 1.0)
         #water.add_s_alpha_beta('c_H_in_H2O')

         materials = openmc.Materials([ideal_xs])
         materials.export_to_xml()
```

In [31]:
```
# Set the depth parameter!!

depth = 10
width = 100
#reflector = False

#Transmission vs. reflective boundaries vs. periodic?? --> What do i want?

#fission_source_z = openmc.ZPlane(z0 = -depth/100, boundary_type ="reflective")
min_x = openmc.XPlane(x0 = -width, boundary_type = "reflective")
min_y = openmc.YPlane(y0 = -width, boundary_type = "reflective")
min_z = openmc.ZPlane(z0 = 0, boundary_type = "reflective")
max_x = openmc.XPlane(x0 = width, boundary_type = "reflective")
max_y = openmc.YPlane(y0 = width, boundary_type = "reflective")
max_z = openmc.ZPlane(z0 = +depth, boundary_type = "reflective")

#Infinite slab!!
```

In [32]:
```
water_cell = openmc.Cell(name = "Water cell")
water_cell.fill = ideal_xs
water_cell.region = -max_z & +min_z & - max_y & +min_y & -max_x & +min_x

universe = openmc.Universe(cells =  [water_cell])
#universe.plot(width = (10.0,10.0), basis = "xy")

root_cell = openmc.Cell(name='root cell')
root_cell.fill = universe
root_cell.region = +min_x & -max_x & +min_y & -max_y & +min_z & -max_z

root_universe = openmc.Universe(universe_id=0, name='root universe')
root_universe.add_cell(root_cell)

geometry = openmc.Geometry(root_universe)
geometry.export_to_xml()
```

Now the simulation settings

In [33]:
```
# OpenMC simulation parameters
batches = 2
inactive = 1
particles = 100
particle_track = (1,1,1)
```

In [34]:
```
settings_file = openmc.Settings()
settings_file.batches = batches
settings_file.inactive = inactive
settings_file.particles = particles
settings_file.run_mode = "fixed source"
```

```
settings_file.track = particle_track
settings_file.verbosity = 10

settings_file.cross_sections = "/Users/juankostelec/Applications/OpenMC/scripts/nndc_hd
```

The cross_sections command can be ignored/removed, if we have defined the OPENMC_CROSS_SECTIONS environment variable to the complete path of the cross_sections.xml file in the OpenMC installation directory.

Creating a source:

```
In [35]: source_type = "box"
         source_depth_size = 0.1 # This determines the depth of the initial source region
         bounds = [-width,-width,0,width,width,source_depth_size]
         point_source_loc = (width/2.0, width/2.0,source_depth_size)

         energy_type = "monoenergetic"
         strength = 2.5e6 # In eV
         only_fissionable = False # Whether to consider source sites only if they are in a fissi

         if source_type == "box":
             dist = openmc.stats.Box(bounds[:3], bounds[3:], only_fissionable=only_fissionable)
         elif source_type == "point":
             dist = openmc.stats.Point(xyz=point_source_loc)
         else:
             raise SyntaxError("Incorrect value {0} for source_type".format(source_type))

         if energy_type == "monoenergetic":
             energy = openmc.stats.Discrete([strength],[1])
         elif energy_type == "fission":
             pass
         else:
             raise SyntaxError("Incorrect value {0} for energy_type".format(energy_type))

         settings_file.source = openmc.source.Source(space=dist, energy = energy)

In [36]: settings_file.export_to_xml()
```

1.0.1 Tallies

We are interested in measuring the spetctrum at different depths.

```
In [37]: tallies_file = openmc.Tallies()

         mesh_resolution = 50
         axial_flux_mesh_resolution = 50
         #We specify how many bins does the mesh have (the value has to be sensible, it depends


         # Creating a mesh
```

```
mesh = openmc.Mesh(mesh_id = 1, name = "mesh_1")
mesh.dimension = [mesh_resolution, mesh_resolution]
mesh.lower_left = [-width, -width] # Giving just 2 coordinates assumes it to be a x-y m
mesh.upper_right = [width, width]
mesh_filter = openmc.MeshFilter(mesh, filter_id = 1)

# Reaction rate tally
reaction_rate = openmc.Tally(name = "mesh", tally_id = 1)
reaction_rate.filters = [mesh_filter]
#reaction_rate.scores = ["total", "scatter", "fission", "flux", "flux-Y5"]
reaction_rate.scores += ["total", "flux-Y1", "flux"]
tallies_file.append(reaction_rate)

# Creating tally for total flux along the axis!
total_flux = openmc.Tally(name = "Total flux through depth")
mesh_axial = openmc.Mesh(mesh_id = 2, name = "mesh_2")
mesh_axial.dimension = [1,1,axial_flux_mesh_resolution]
mesh_axial.lower_left = [-width, -width,0]
mesh_axial.upper_right = [width, width,depth]
mesh_axial_filter = openmc.MeshFilter(mesh_axial, filter_id = 2)

total_flux.filters = [mesh_axial_filter]
total_flux.scores = ["flux"]
tallies_file.append(total_flux)


#Neutron spectrum
#Creating energy bins
delta = depth/10
energy_bins = np.logspace(-3,7, num = 100)
energy_filter = openmc.EnergyFilter(bins = energy_bins, filter_id =3)

neutron_spectrum = openmc.Tally(tally_id = 3, name ="neutron spectrum")
neutron_spectrum.filters+=[energy_filter]
neutron_spectrum.scores+=["scatter", "fission", "flux"]
tallies_file.append(neutron_spectrum)


# Tally for angular flux!!

ang_flux_mesh_dimension = [10,10,10]

mesh3 = openmc.Mesh(mesh_id = 3, name = "mesh_3")
mesh3.dimension = [10,10,10]
mesh3.lower_left = [-width, -width,0] # depth-delta instead of 0
mesh3.upper_right = [width, width,depth]
mesh3_filter = openmc.MeshFilter(mesh3, filter_id = 4)
```

```
azimuthal_num_bins = 10
polar_num_bins = 10
polar_bins = np.linspace(0,pi, num = polar_num_bins)
azimuthal_bins = np.linspace(-pi, pi, num = azimuthal_num_bins)  #Azimuthal_num_bins an
                                                                 # I will need this valu
#azimuthal_bins = [-pi,pi/2,pi]
polar_filter = openmc.PolarFilter(bins = polar_bins, filter_id = 5)
azimuthal_filter = openmc.AzimuthalFilter(bins = azimuthal_bins, filter_id = 6)

angular_flux = openmc.Tally(name = "angular flux")
angular_flux.filters += [polar_filter, azimuthal_filter, mesh3_filter]
angular_flux.scores += ["flux"]

tallies_file.append(angular_flux)
```

Let's create a flux mesh tally at each slice of the slab:

In [38]:
```
num_slices = 20
meshes = [None for i in range(num_slices)]
tallies = [None for i in range(num_slices)]

energy_bins = np.logspace(-3,7, num = 100)
energy_filter = openmc.EnergyFilter(bins = energy_bins, filter_id =3)

for i in range(num_slices):
    meshes[i] = openmc.Mesh(name = "Slice_{0}".format(i))
    meshes[i].dimension = [1,1,1]
    meshes[i].lower_left = [-width, -width, i/float(num_slices) * depth]
    meshes[i].upper_right = [width,width, (i+1)/float(num_slices)*depth]

    temp = None
    temp = openmc.Tally(name = "neutron_spectrum_slice_{0}".format(i))
    temp.filters = [openmc.MeshFilter(meshes[i]), energy_filter]
    temp.scores +=["flux", "fission", "scatter","events"]
    tallies_file.append(temp)

tallies_file.export_to_xml()
```

In [39]:
```
!export OPENMC_CROSS_SECTIONS=/Users/juankostelec/Applications/OpenMC/scripts/nndc_hdf5
#openmc.run()
```

Now let's try to plot some of the acquired data

In [12]:
```
# Load the statepoint file
sp = openmc.StatePoint('statepoint.2.h5')
```

In [13]:
```
tally1 = sp.get_tally(name = "Total flux through depth")
tally1.get_pandas_dataframe()
flux_axial = tally1.get_slice(scores = ["flux"])
```