Monte Carlo Project Part 2

22.901 Introduction to Computer Programming for Nuclear Engineers

January 30 - February 2, 2012

Accesing pdfs

- link in the pdf static library, already in Makefile
- function: get_particle_pos - send it slab length, it returns x location
- function: get_particle_mu - it returns mu (-1 to 1)
- function: get_scatter_mu - it returns mu(-1 to 1)
- function: get_collision_distance - send it the total xs and it returns the free-flight distance
- function: get_collision_type - send it, respectfully, the absorption, scattering and total xs and it returns the collision id
- ID=1 is Absorption and ID=2 is Scattering

Module `execute` - Subroutine `run_problem`

- call `initialize_rng`
- begin loop over the number of histories
    - call `particle_init`
    - call `reset_tallies`
    - get the slab id for the neutron, call `get_slab_id`
    - begin loop *while* the neutron is alive
        - call `transport`
        - call `interaction` *if* the neutron is still alive
- call `bank_tallies`

Module `execute` - Subroutine `transport` I

- declare 4 variables:
  - real, `s` - free flight distance
  - real, `newx` - temporary new x location
  - real, `neig` - x location of nearest neighbor in direction
  - logical, `resample` - resample distance
- set resample to true
- begin a loop while resample is true
  - access `get_collision_distance` function - see pdfs slide, set to free flight dist.
  - based on free-flight and neutron mu compute new x location
  - get nearest neighbor x in mu direction
  - check for a surface crossing
  - check if the neutron leaked, can only happen if slab_id is 1 or n_slabs

Module execute - Subroutine transport II

- kill neutron
- set resample to false
- record track length to tally (only free-flight to surface)
- move neutron to neighbor surface
- change the slab number
- if the neutron did not cross a surface, it collided
- record full free-flight to track tally
- move neutron to new xlocation
- set resample to false

Module execute - Subroutine transport III

In summary:
Find a new flight distance. Project it on the x axis with its angle of travel. Determine the nearest neighbor x location in the direction mu. If you cross a surface, then check if the neutron leaked out of a system. Note: you can only leak if the neutron is originating in slab 1 or the final slab. This can be checked with the slab_id. If you do leak, kill the particle and do not resample the distance. If you cross a surface even if you leak, move the neutron to the that surface. Compute the free-flight distance to that surface using the change in x and mu. Append that to the temporary tally.
If you don't cross a surface then you have a collision. Do not resample, tally the free-flight distance and exit the transport routine.

Module execute - Subroutine interaction

- make a local integer variable called `id`
- add a 1/totalxs to the collision temp variable
- get the reaction type by accessing the function `get_collision_type` - see pdfs slide
- if the id is 1 then absorption happened - kill particle
- if the id is not 1 then scattering and sample a new angle
  - to get a new angle access `get_scatter_mu` - see pdfs slide

Module execute - Subroutine get_slab_id

- make a local integer variable called id
- add a 1/totalxs to the collision temp variable
- get the reaction type by accessing the function get_collision_type - see pdfs slide
- if the id is 1 then absorption happened - kill particle
- if the id is not 1 then scattering and sample a new angle
  - to get a new angle access get_scatter_mu - see pdfs slide

Module `execute` - Subroutine `reset_tallies`

- loop around slabs
- call `tally_reset` and send it a tally object for that slab

Module execute - Subroutine bank_tallies

- loop around slabs
- call bank_tally and send it a tally object for that slab

Module `execute` - Subroutine `print_tallies`

- loop around slabs
- print out tally information by slab region
- e.g. Slab 1 Coll: mean+/-var Track: mean+/-var

Module `global` - Subroutine `allocate_problem`

- no arguements just allocated the tally object for the number of slabs

Module `global` - Subroutine `free_memory`

- no arguements just deallocate the tally object

Module particle - Subroutine particle_init

- arguments are the particle object and length of slab
- call pdf function get_particle_pos
- call pdf function get_particle_mu
- make neutron alive

Flux Volume Tallies

- For each tally we have two varbles [s(c)1 and s(c)2] - used for mean and variance calculation
- We also have temp. variable track and coll
- During a neutrons life, events will be accumulated in track and coll
- When life over we "bank" them into s(c)1 and s(c)2
- Collision estimator (c)

$$\phi_c = \frac{1}{NX} \sum \frac{1}{\Sigma_t}$$

- Track-length estimator (s)

$$\phi_c = \frac{1}{NX} \sum d$$

d is the free flight distance and I also use s for this as well.

Module `tally` - Subroutine `tally_reset`

- set the track and coll variables in the tally instance to zero

Module `tally` - Subroutine `bank_tally`

- Do the following math for the tally instance

$$c1 = c1 + coll$$

$$c2 = c2 + coll^2$$

$$s1 = s1 + track$$

$$s2 = s2 + track^2$$

Module tally - Subroutine perform_statistics

- pass in the tally instance, number of neutrons and volume
- compute mean for each tally estimator

$$mean = \frac{s(c)1}{dx * nhist}$$

$$var = \frac{1}{nhist - 1}\left[\frac{s(c)2}{nhist} - \left(\frac{s(c)1}{nhist}\right)^2\right]$$