Problem Set 1 Solutions

22.211 Reactor Physics I

Due: 22 February 2012

Bryan Herman

Problem 1. Generate plots of scatter neutron energy vs. generation for C-12.

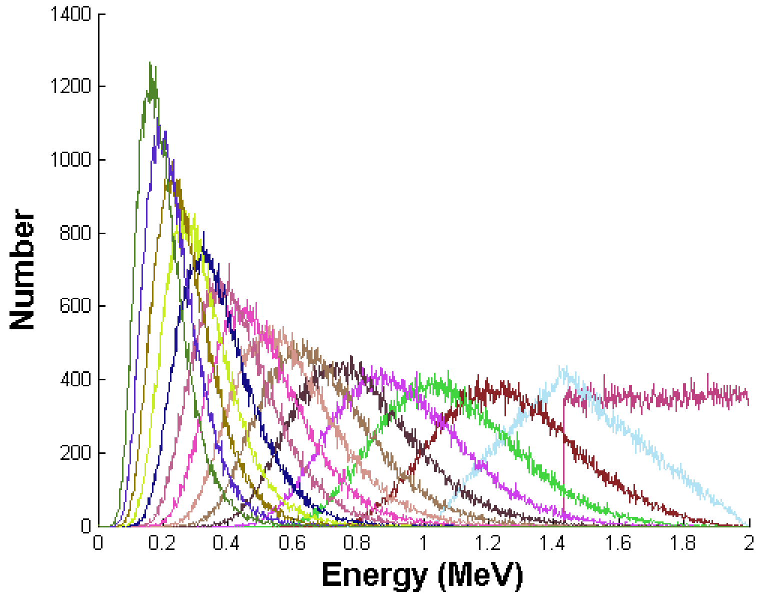

Figure 1: C-12 Scattered Energy by Generation - 15 generations 100,000 neutrons

Problem 2. Generate plots of scatter neutron energy vs. generation for H-1.

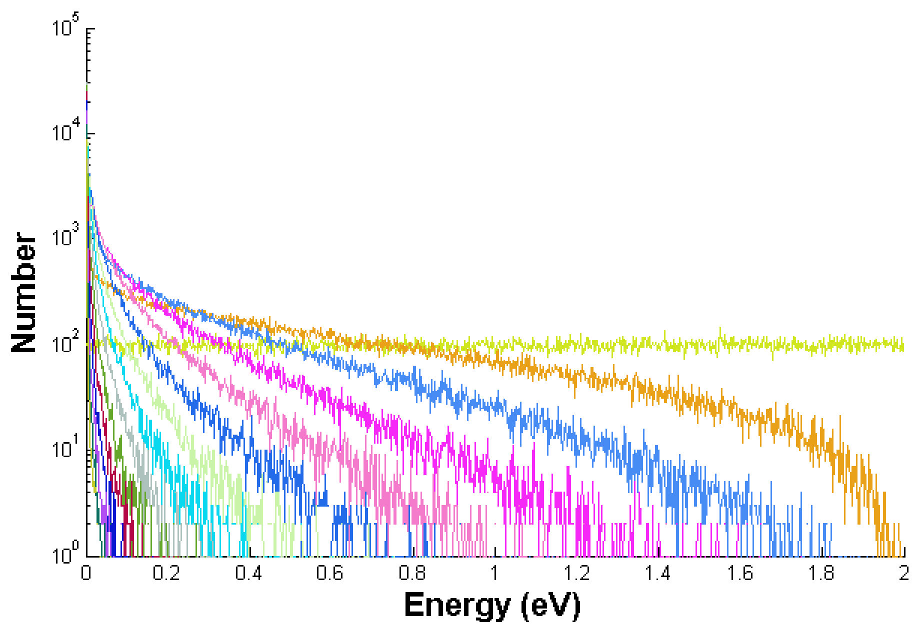

Figure 2: H-1 Scattered Energy by Generation - 15 generations 100,000 neutrons

Problem 3. Generate plots of scatter neutrons flux vs. energy for C-12.

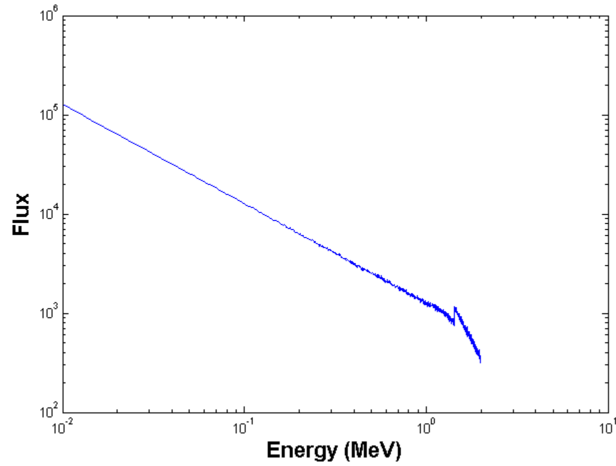

Figure 3: C-12 Flux Spectrum (energy bins) 100,000 neutrons

Problem 4. Generate plots of scatter neutrons flux vs. lethargy for C-12.

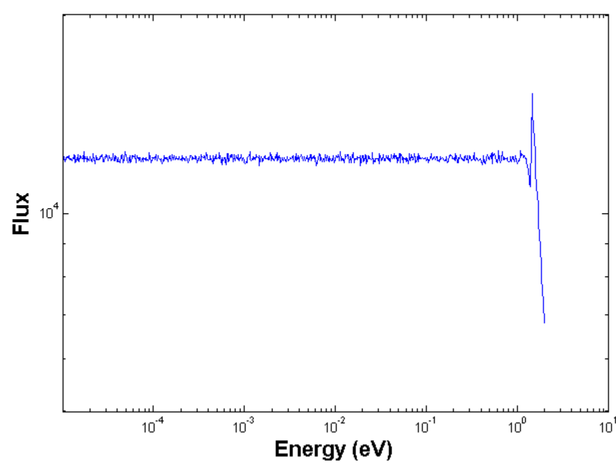

Figure 4: C-12 Flux Spectrum (lethargy bins) 100,000 neutrons

Problem 5. Explain the "Placzek Transients" in flux vs. lethargy for C-12 slowing down.

The Placzek transient is a discontinuity of the flux, and derivatives of the flux, at integer multiples of αE. The transient results from the fact that after the first generation, every neutron must have $E \geq \alpha E$. Neutrons which have an energy $E \geq \alpha E$ can only consist of neutrons in the 2nd, 3rd, or latter generations, and so forth for each multiple of αE. Thus, the discontinuity at αE arises from the fact that to the left and right of each multiple of αE our neutron flux is accumulated from a monotonically decreasing number of neutron generations.

Problem 6. Generate cdf plot for fission neutron emission energy.

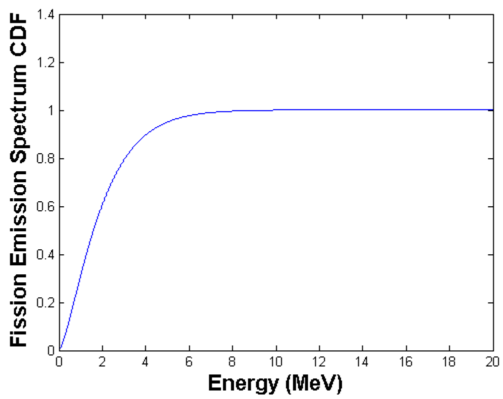

Figure 5: Cumulative Distribution Function for Fission Emission Spectrum

Problem 7. Generate neutron emission plot from 10,000 random fissions.

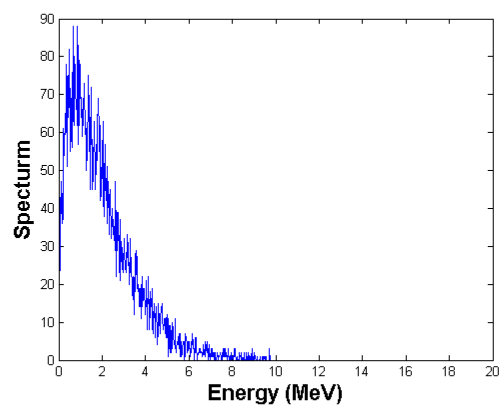

Figure 6: Fission Emission Spectrum - 10,000 neutrons

The flux spectrum for a constant cross section from a fission source is shown below. Notice the slight bump at 1 MeV. This is seen since it is the most probable energy of a neutron emitted from fission. If you don't see this slight bump, you may not be tallying the flux spectrum after you sampled the fission source energy, BUT before you sampled a new energy from scattering.

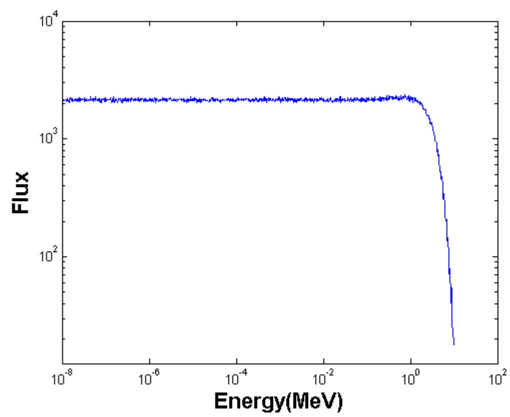

Figure 7: H-1 (constant xs) Flux Spectrum (lethargy bins)

After incorporating the true Hydrogen-1 cross section from ENDF, the shape of the curve changes as shown below.

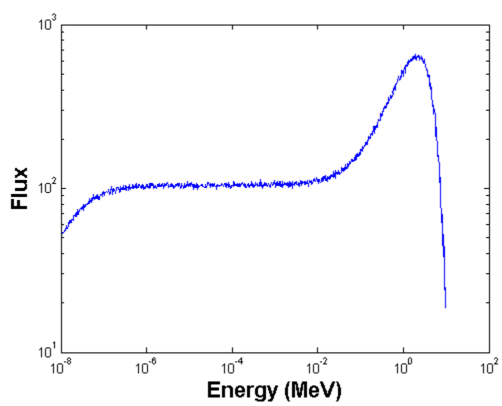

Figure 8: H-1 (ENDF xs) Flux Spectrum (lethargy bins)

A Main Codes

A.1 Slowing Down by Generation

```
% Bryan Herman
% Slowing Down Code
% HW1 22.211
%
%% Slowing down part (by generation)
% set input information
n_histories = 100000;
n_generations = 15;
Eo = 2.0;
A = 1;
seed = 5;
n_bins = 1000;
source = 'const';

% initalize objects
neut(1:n_histories) = particle_class();
tal = tally_class(n_bins,Eo);
pdf = pdf_class(seed);
mat = material_class(A);

% set materials (may put this in class later)
mat = mat.load_isotope('H_1');

% begin loop
for i = 1:n_generations

    % display information
    fprintf('\nExecuting Generation %d\n',i);
    fprintf('=======================\n\n');

    for j = 1:n_histories

        % sample source energy if fission spectrum is on
        if i == 1
            pdf = pdf.sample_source_energy(source);
            neut(j) = neut(j).set_energy(pdf.E);
        end

        % sample new energy
        pdf = pdf.sample_collision_energy(neut(j).E,mat.alpha);

        % set particle to that new energy
        neut(j) = neut(j).set_energy(pdf.E);

        % bank sampled energy (only seed 1 isotope for now)
        tal = tal.bank_tally(neut(j).E,mat.isotopes{1});

        if mod(j,1000) == 0
```

```
            fprintf('Histories: %d ...\n',j);
        end
    end

    % plot that generation
    tal.plot_tally(1);

    % reset tallies
    tal = tal.reset;

end

% plot flux
tal.plot_flux(2);
```

A.2 Fission Emission Spectrum

A.3 Slowing Down by Particle

```
% Bryan Herman
% Slowing Down Code
% HW1 22.211
%
%% Slowing down part (by particle)

% set input information
n_histories = 100000;
Eo = 20.0;
A = 1;
seed = 5;
n_bins = 1000;
source = 'fission';

% initalize objects
neut(1:n_histories) = particle_class();
tal = tally_class(n_bins,Eo);
pdf = pdf_class(seed);
mat = material_class(A);

% set materials (may put this in class later)
mat = mat.load_isotope('H_1');

% begin loop around histories
for j = 1:n_histories

    % sample source energy
    pdf = pdf.sample_source_energy(source);
    neut(j) = neut(j).set_energy(pdf.E);

    while neut(j).alive == 1
```

```
        % bank sampled energy (only seed 1 isotope for now)
        tal = tal.bank_tally(neut(j).E,mat.isotopes{1});

        % sample new energy
        pdf = pdf.sample_collision_energy(neut(j).E,mat.alpha);

        % set particle to that new energy
        neut(j) = neut(j).set_energy(pdf.E);

        if(neut(j).E < 1e-8)

            % kill particle
            neut(j) = neut(j).kill;

            % reset tallies and bank flux
            tal = tal.reset;

        end

    end

    % display calculation progress
    if mod(j,1000) == 0
        fprintf('Histories: %d ...\n',j);
    end

end

% plot flux
tal.plot_flux(1);
```

B Class Files

B.1 Particle Class

```
classdef particle_class
    %PARTICLE_CLASS defines a particle type for slowing down
    %   this class defines a particle along with its methods that will
    %   be used in the Monte Carlo slowing down code

    properties (SetAccess = private, GetAccess = public)
        E
        alive
    end

    methods

        % Constructor
        function obj=particle_class()
            obj.alive = 1;
```

```
        end

        % kill particle
        function obj=kill(obj)
            obj.alive = 0;
        end

        % sets particle energy
        function obj = set_energy(obj,E)
            obj.E = E;
        end

    end

end
```

B.2 PDF Class

```
classdef pdf_class

    % PDF_CLASS contains information about the distribution function
    %   contains all of the random number sampling and pdfs needed by MC

    properties (SetAccess = private, GetAccess = public)

        rng    % random number generator object
        E      % new sampled energy
        egrid  % energy grid for chi pdf
        chicdf % cdf for chi

    end

    methods

        % constructor
        function obj = pdf_class(seed)

            % initialize random number generator with seed
            obj.rng = RandStream('mcg16807','Seed',seed);

            % intialize watt fission spectrum cdf
            obj.egrid = linspace(0,20,10000);

            % create cdf
            obj = obj.watt_fission_cdf();

        end

        % sample energy
        function obj = sample_collision_energy(obj,E,alpha)
```

```
            obj.E = E - E*(1-alpha)*obj.rng.rand;

        end

        % sample source energy
        function obj = sample_source_energy(obj,opt)

            if strcmp(opt,'const')

                % assume fixed source at 2.0 MeV
                obj.E = 2.0;

            elseif strcmp(opt,'fission');

                % sample random number
                rn1 = obj.rng.rand;

                % find bin location
                idx = find(obj.chicdf.*(obj.chicdf >= rn1),1,'first')-1;
                obj.E = obj.egrid(idx);

            else
                error('FATAL==>Source cant be sampled.')
            end

        end

    end

    methods (Access = private)

        % pdf for watt fission spectrum
        function chi = watt_fission(obj,E)

            chi = 0.453*exp(-1.036*E).*sinh(sqrt(2.29*E));

        end

        % construct cdf
        function obj = watt_fission_cdf(obj)

            % allocate
            obj.chicdf = zeros(length(obj.egrid),1);

            % create cdf
            for i = 2:length(obj.egrid)

                % numerically integrate
                obj.chicdf(i) = trapz(obj.egrid(1:i),obj.watt_fission(obj.egrid(1:i)));

            end

        end
```

```
    end

end
```

B.3 Material Class

```
classdef material_class
    %MATERIAL_CLASS Summary of this class goes here
    %   Detailed explanation goes here

    properties (SetAccess = private, GetAccess = public)
        A
        alpha
        n_isotopes
        isotopes
        iso_list
    end

    methods

        % constructor
        function obj = material_class(A)

            % set vars
            obj.A = A;
            obj.alpha = ((A-1)/(A+1))^2;
            obj.n_isotopes = 0;

        end

        % import xsdata file
        function obj = load_isotope(obj,name)

            % increment number of isotopes
            obj.n_isotopes = obj.n_isotopes + 1;

            % call constructor of xsdata
            obj.isotopes{obj.n_isotopes} = cross_section_class(name);

            % append to list
            obj.iso_list{obj.n_isotopes} = name;

        end

    end

end
```

B.4 Cross Section Class

```
classdef cross_section_class
    %CROSS_SECTION keeps track of ENDF xs data sets

    properties
        egrid
        xs
        name
    end

    methods

        % constructor
        function obj = cross_section_class(name)

            % get name
            obj.name = name;

            % load xs data file expecting E and xs
            disp(strcat('Loading xsdata file: ',name,'.m'));
            load(name);

            % set energy grid and xs
            obj.egrid = E;
            obj.xs = xs;

        end
    end

end
```

B.5 Tally Class

```
classdef tally_class
    %TALLY_CLASS defines the object for tallying MC quantites
    %   defines the object for tallying MC quantites

    properties(SetAccess = private,GetAccess = public)

        nbins   % number of energy grid bins
        egrid   % energy grid
        lgrid   % lethargy grid
        counts  % count
        lcounts % letharg bin counts
        eave    % array of average energy in egrid bins
        leave   % arrage of average energy in lethargy bins
        de      % energy spacing
        le      % lethargy spacing
        flux    % flux accumulation
        lflux   % flux in lethargy bins
```

```
    end

    methods

        % constructor
        function obj = tally_class(nbins,Eo)

            % initalize values
            obj.nbins = nbins;
            obj.egrid = linspace(0,Eo,nbins+1);
            obj.eave = linspace((Eo/(nbins*2)),Eo-(Eo/(nbins*2)),nbins);
            obj.counts = zeros(1,nbins);
            obj.flux = zeros(1,nbins);
            obj.de = obj.egrid(2) - obj.egrid(1);

            obj.lgrid = logspace(-8,log10(Eo),nbins+1);
            obj.leave = 0.5*(obj.lgrid(1:nbins) + obj.lgrid(2:nbins+1));
            obj.lcounts = zeros(1,nbins);
            obj.lflux = zeros(1,nbins);
            obj.le = log(max(obj.lgrid)/obj.lgrid(1)) - log(max(obj.lgrid)/obj.lgrid(2));

        end

        % bank tally
        function obj = bank_tally(obj,E,iso)

            % get micro cross section
            xs = interp1(iso.egrid,iso.xs,E,'linear','extrap');

            % find out bin index
            idx = find(obj.egrid.*(obj.egrid >= E),1,'first')-1;

            if idx > length(obj.egrid) || idx == 0
                error('FATAL ==> index out of bounds');
            end
            % bank a count
            obj.counts(idx) = obj.counts(idx) + 1/xs;

            % find out lethargy bin index
            idx = find(obj.lgrid.*(obj.lgrid >= E),1,'first')-1;

            % bank a count (disregard below the cutoff energy)
            if (idx > 0)
                obj.lcounts(idx) = obj.lcounts(idx) + 1/xs;
            end

        end

        % plot tally
        function plot_tally(obj,n)

            % get random rgb code
            r = rand(1);
            g = rand(1);
```

```
            b = rand(1);

            figure(n);
            hold on;
            plot(obj.eave,obj.counts,'Color',[r,g,b]);

        end

        % plot flux
        function plot_flux(obj,n)

            figure(n)
            loglog(obj.eave,obj.flux);

            figure(n+1)
            loglog(obj.leave,obj.lflux);

        end

        % clear tally
        function obj = reset(obj)

            % bank in flux first
            obj.flux = obj.flux + obj.counts;
            obj.lflux = obj.lflux + obj.lcounts;

            % now reset
            obj.counts(:) = 0;
            obj.lcounts(:) = 0;

        end

    end

    methods(Access = private)

        % determine the index on the energy grid
        function idx = get_energy_idx(obj,E)

            idx = ceil(E/obj.de);

        end

        % determine the index on the lethargy grid
        function idx = get_lethargy_idx(obj,E)

            % this is complex since we are putting on a log energy grid
            l = log(max(obj.lgrid)/E);
            idx = length(obj.lgrid) - floor(l/obj.le) - 1;
            if idx <= 0
                idx = 1;
            end

        end
```

```
    end

end
```