# NUEN 629
# Numerical Methods in Reactor Analysis
# Homework 4 & 5 & Project

Due on:
Thursday, November 19, 2015
&
Thursday, December 3, 2015

*Dr. McClarren*

**Paul Mendoza**

# Contents

# Homework 4 Problem Statement

Solve the following problem and submit a detailed report, including a justification of why a reader should believe your results and a description of your methods and iteration strategies.

1. (150 points + 50 points extra credit) In class we discussed the diamond-difference spatial discretization. Another discretization is the step discretization (this has several other names from other disciplines). It writes the discrete ordinates equations with isotropic scattering as, for $\mu_n > 0$ to

$$\mu_n \frac{\psi_{i,n} - \psi_{i-1,n}}{h_x} + \Sigma_t \psi_{i,n} = \frac{\Sigma_s}{2}\phi_i + \frac{Q}{2} \tag{1}$$

   and for $\mu_n < 0$

$$\mu_n \frac{\psi_{i+1,n} - \psi_{i,n}}{h_x} + \Sigma_t \psi_{i,n} = \frac{\Sigma_s}{2}\phi_i + \frac{Q}{2} \tag{2}$$

   The codes provided in class should be modified to implement this discretization.

(a) (50 Points) Your task (should you choose to accept it) is to solve a problem with uniform source of $Q = 0.01$, $\Sigma_t = \Sigma_s = 100$ for a slab in vacuum of width 10 using step and diamond difference discretizations. Use, 10, 50, and 100 zones ($h_x = 1, 0.02, 0.01$) and your expert choice of angular quadratures. Discuss your results and how the two methods compare at each number of zones.

(b) (10 points) Discuss why there is a different form of the discretization for the different signs of $\mu$.

(c) (40 points) Plot the error after each iteration using a 0 initial guess for the step discretization with source iteration and GMRES.

(d) (50 points) Solve Reed's problem (see finite difference diffusion codes). Present convergence plots for the solution in space and angle to a "refined" solution in space and angle.

(e) (50 points extra credit) Solve a time dependant problem for a slab surrounded by vacuum with $\Sigma_t = \Sigma_s = 1$ and initial condition given by $\boldsymbol{\psi(0) = 1/h_x}$ (original problem statement said $\phi(0) = 1/h_x$ and I'm not sure how to solve that). Plot the solution at $t = 1$ $s$, using step and diamond difference. The particles have a speed of 1 cm/s. Which discretization is better with a small time step? What do you see with a small number of ordinates compared to a really large number (100s)?

# Homework 4 Problem Background

Due to the complicated nature of this course, I provided this background for the lay person (me), so that they might have some grounding for the solution and hopefully believe the results. It should be noted that most of this background information is copied from various points in Dr. McClarren's notes, and is in no way original. Anything intelligent in the following is due to this fact and for any errors, I blame myself.

Beginning with the weighty neutron transport equation.

$$\left( \frac{1}{v} \frac{\delta}{\delta t} + \hat{\Omega} \cdot \nabla + \Sigma_t \right) \psi = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ K(\hat{\Omega}' \cdot \hat{\Omega}, v' \to v) \Sigma_s \psi + \frac{1}{4\pi} \chi \int_0^\infty dE' \ \bar{\nu} \Sigma_f \phi + q$$

Where $K(\hat{\Omega}' \cdot \hat{\Omega}, v' \to v)$ represents the probability of scattering from one angle and energy to another given a scattering event occured and $\Sigma_s$ is the macroscopic scattering cross section. The dependencies for the variables are shown below.

$$\Sigma_t(\vec{x}, v, t)$$
$$\psi(\vec{x}, \hat{\Omega}, v, t)$$
$$\Sigma_s(\vec{x}, v, t)$$
$$\chi(\vec{x}, v)$$
$$\Sigma_f(\vec{x}, v, t)$$
$$\phi(\vec{x}, v, t)$$
$$q(\vec{x}, \hat{\Omega}, v, t)$$

There are 7 free variables (three spatial $[\vec{x}]$, two angular $[\hat{\Omega}]$, one energy $[v]$ and one time $[t]$) in this equation. In the steady state $\left( \frac{\delta \psi}{\delta t} = 0, \text{ i.e. no time dependence} \right)$, non fissioning ($\Sigma_f = 0$) case the transport equation reduces to,

$$\left( \hat{\Omega} \cdot \nabla + \Sigma_t \right) \psi = \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \ K(\hat{\Omega}' \cdot \hat{\Omega}, v' \to v) \Sigma_s \psi + q.$$

In order to reduce this to a single energy the following definitions are helpful (remembering all time dependence is gone).

$$\psi(\vec{x}, \hat{\Omega}) = \int_0^\infty dE \ \psi(\vec{x}, \hat{\Omega}, v(E))$$

$$\Sigma_t(\vec{x}) = \frac{\int_0^\infty dE \ \Sigma_t(\vec{x}, v(E)) \psi(\vec{x}, \hat{\Omega}, v(E))}{\psi(\vec{x}, \hat{\Omega})}$$

$$K(\hat{\Omega}' \cdot \hat{\Omega}, v' \to v) = K(\hat{\Omega}' \cdot \hat{\Omega}) K(v' \to v)$$

$$\Sigma_s(\vec{x}) = \frac{\int_0^\infty dE \int_0^\infty dE' \ \Sigma_s(\vec{x}, v(E)) K(v' \to v) \psi(\vec{x}, \hat{\Omega}, v(E))}{\psi(\vec{x}, \hat{\Omega})}$$

$$q(\vec{x}, \hat{\Omega}) = \int_0^\infty dE \ q(\vec{x}, \hat{\Omega}, v(E))$$

Using these definitions, integrating the transport equation over all energy, and assuming cross sections and sources do not vary in space or angle, our transport equation reduces again to,

$$\left( \hat{\Omega} \cdot \nabla + \Sigma_t \right) \psi(\vec{x}, \hat{\Omega}) = \int_{4\pi} d\hat{\Omega}' \ K(\hat{\Omega}' \cdot \hat{\Omega}) \Sigma_s \psi(\vec{x}, \hat{\Omega}') + q.$$

---

Where the double differential was assumed to be separable in angle and energy. The final simplification for our problem will be in space. If we assume that our geometry is infinite in $y$ $\left(\frac{\delta}{\delta y} = 0\right)$ and $x$ $\left(\frac{\delta}{\delta x} = 0\right)$. This also means that $\psi$ depends only on $z$ and $mu$, and if we recall that

$$\hat{\Omega} = (\sqrt{1 - \mu^2}cos(\rho), \sqrt{1 - \mu^2}sin(\rho), \mu),$$

and

$$\nabla = \left(\frac{\delta}{\delta x}, \frac{\delta}{\delta y}, \frac{\delta}{\delta x}\right)$$

also assuming that

$$K(\hat{\Omega}' \cdot \hat{\Omega}) = \frac{1}{4\pi} \text{ Isotropic Scattering}$$

then our transport equation, and the equation I think we are trying to solve for this homework is.

$$\left(\mu\frac{\delta}{\delta z} + \Sigma_t\right) \psi(z, \mu) = \Sigma_s \frac{2\pi}{4\pi} \int_{-1}^{1} d\mu' \ \psi(z, \mu') + q.$$

Checking units,

$$\left(\mu\frac{\delta}{\delta z} + \Sigma_t\right) \left[\frac{1}{cm}\right] \psi(z, \mu) \left[\frac{n \cdot cm}{str \cdot cm^3 \cdot s}\right] = \Sigma_s \frac{1}{2} \left[\frac{1}{cm \cdot rad}\right] \int_{-1}^{1} d\mu' \ \psi(z, \mu') \left[\frac{n \cdot cm}{rad \cdot cm^3 \cdot s}\right] + q \left[\frac{n}{str \cdot cm^3 \cdot s}\right].$$

$\Sigma_s$ was moved outside the integral because it has no angular dependence integration over the azimuthal angle occured because $\psi(z, \hat{\Omega})$ is assumed to be uniform and not depend on that angle.

Using Gauss-Legendre Quadrature for the integration term

$$\phi = \int_{-1}^{1} d\mu' \psi(z, \mu') = \sum_{i=1}^{n} w_i \psi(z, \mu_i')$$

where

$$w_i = \frac{2}{(1 - \mu_i^2)[P_n'(\mu_i)]^2}$$

$P_n'$ is the differential of the legendre polynomial $n$, and $\mu_i'$ are the roots of $P_n$. The weights of even $n$'s of the legendre polynomials should sum to 2, the value of $\int_{-1}^{1} d\mu$, which they do.

---

Putting this all together with time dependence:

$$\left(\frac{1}{v}\frac{\delta}{\delta t} + \mu\frac{\delta}{\delta z} + \Sigma_t\right) \psi_n(z) = \Sigma_s \frac{1}{2} \sum_{n'=1}^{N} w_{n'}\psi_{n'}(z) + q$$

Where $n$ and $n'$ denote the direction being solved for and $N$ is the total number of angles being solved for. Also unites of $w$ are rad.

---

**Diamond difference discretization**

$$\frac{1}{v}\frac{\psi_{n,i}^{\ell+1,j+1} - \psi_{n,i}^{L,j}}{\Delta t} + \mu_n \frac{\psi_{n,i+1/2}^{\ell+1,j+1} - \psi_{n,i-1/2}^{\ell+1,j+1}}{hz} + \Sigma_t\psi_{n,i}^{\ell+1,j+1} = \Sigma_s\frac{1}{2}\sum_{n'=1}^{N} w_i\psi_{n',i}^{\ell,j+1} + q.$$

Where $n$ is for angle, $i$ is the midplane of a spacial discretization, $\ell$ is the iteration index for spacial convergence, $j$ is for a time step and

$$\psi_{n,i}^{\ell+1,j+1} = \frac{1}{2}(\psi_{n,i+1/2}^{\ell+1,j+1} + \psi_{n,i-1/2}^{\ell+1,j+1})$$

---

Homework 4 Problem Background continued on next page...

Writing this in terms of a steady state

$$\mu_n \frac{\psi_{n,i+1/2}^{\ell+1,j+1} - \psi_{n,i-1/2}^{\ell+1,j+1}}{hz} + \Sigma_t^* \psi_{n,i}^{\ell+1,j+1} = \Sigma_s \frac{1}{2} \sum_{n'=1}^{N} w_i \psi_{n',i}^{\ell,j+1} + q^*.$$

where

$$\Sigma_t^* = \Sigma_t + \frac{1}{v\Delta t}$$

$$q^* = q + \frac{\psi_{n,i}^{L,j}}{v\Delta t}$$

The above equation has L for the iteration index to indicate that its value was iteratively determined in the previous time step.

**Step discretization**

Writing this in terms of a steady state for $\mu > 0$

$$\mu_n \frac{\psi_{n,i}^{\ell+1,j+1} - \psi_{n,i-1}^{\ell+1,j+1}}{hz} + \Sigma_t^* \psi_{n,i}^{\ell+1,j+1} = \Sigma_s \frac{1}{2} \sum_{n'=1}^{N} w_i \psi_{n',i}^{\ell,j+1} + q^*.$$

and for $\mu < 0$

$$\mu_n \frac{\psi_{n,i+1}^{\ell+1,j+1} - \psi_{n,i}^{\ell+1,j+1}}{hz} + \Sigma_t^* \psi_{n,i}^{\ell+1,j+1} = \Sigma_s \frac{1}{2} \sum_{n'=1}^{N} w_i \psi_{n',i}^{\ell,j+1} + q^*.$$

where

$$\Sigma_t^* = \Sigma_t + \frac{1}{v\Delta t}$$

$$q^* = q + \frac{\psi_{n,i}^{L,j}}{v\Delta t}$$

**GMRES**

The generalized minimium residual (GMRES) method is an iterative method for solving linear systems of equations. The method approximates the solution by the vector in a Krylov subspace with a minimum residual (see wikipedia or Dr. McClarren's notes, I'm not really sure how this method works, but python has a solver for it).

The system $A\vec{\phi} = b$ is solved with GMRES, where for our situation,

$$A = \left( I - \sum_{n'=1}^{N} L^{-1} \Sigma_s \frac{1}{2} \right)$$

where $L^{-1}$ is a sweep solve for our system and acts as an operator (I think), and

$$b = \sum_{n'=1}^{N} L^{-1} q^*$$

**Reeds Problem**

Reeds problem is a similiar system as above, except the source and scattering and total cross sections are variable in $z$, and the width of $z$ is 16.

# Homework 4 Problem Solution

The code for this problem will be at the end of this section. The answers are below.

(a) (50 Points) Your task is to solve a problem with uniform source of $Q = 0.01$, $\Sigma_t = \Sigma_s = 100$ for a slab in vacuum of width 10 using step and diamond difference discretizations. Use, 10, 50, and 100 zones ($h_x = 1, 0.02, 0.01$) and your expert choice of angular quadratures. Discuss your results and how the two methods compare at each number of zones.

**The angular quadrature used was the Gauss-Legendre Quadrature because of the integration range. Its form was shown in the background section. The plot below was produced with the GMRES method, but the source iteration scheme produced the same results.**
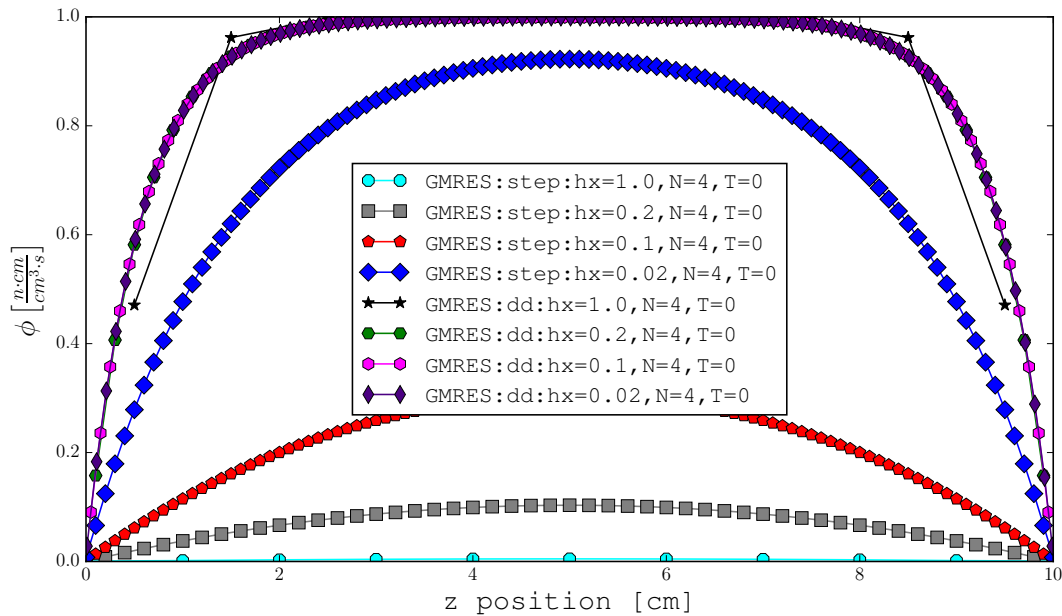


**Both of the iterative solutions converged with max iterations of 100,000 and a slight modification on cross section ($\Sigma_t = \Sigma_t \cdot 1.0001$) to help the system converge. As the number of zones increased for the step solution, the flux magnitude kept increasing to match with the diamond difference and maintained a cosine(ish) shape. As the number of zones increased with the diamond difference, the shape started to converge towards the cosine, but maintained the proper magnitude.**

**Something else I would like to point out in the solution is that the step solution always had one more point plotted than the diamond difference. The reason for this is due to how each solution was solved. This is easier highlighted (for me) with an example, which is shown in the case where the number of zones is 10.**

For the Diamond difference, the average locations (remember they were averaged), $\psi_{n,i}^{L,j+1}$, being solved for were,

$$z = [0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]$$

The points for $\psi_{n,i+1/2}^{L,j+1}$ and $\psi_{n,i-1/2}^{L,j+1}$ were at the points,

$$z = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

When sweeping to the right, $\psi_n(z = 0)$ was set to zero, because the incoming flux is zero, and all points were solved for up to where $z = 10$, and $\psi_{n,i}$ values were determined with averaging. This same thing occured when sweeping to the left (except here $\psi_n(z = 10)$ was set to zero). This would yield 10 values at the points $[0.5, 1.5, ..., 9.5]$.

For the step discretization scheme, the locations (non averaged), $\psi_{n,i}^{L,j+1}$, being solved for were,

$$x = \begin{cases} [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] & \mu > 0 \\ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], & \mu < 0 \end{cases}$$

When combining these two lists for $\phi$, this was considered, and hence the step discretization scheme had one extra point (both lists have 10 points, but the location 10 is unqiue in the first list, and 0 in the second).

(b) (10 points) Discuss why there is a different form of the discretization for the different signs of $\mu$.

**The different forms are needed in the step discretization because in both the diamond and step approaches to the solution a value is needed from a previous zone. Our vacuum boundary condition states that the incoming neutrons are zero, which at the left side of the boundary, determines the angular flux moving to the right, and at the right side of the boundary, the angular flux moving to the left (these values are 0).**

(c) (40 points) Plot the error after each iteration using a 0 initial guess for the step discretization with source iteration and GMRES.
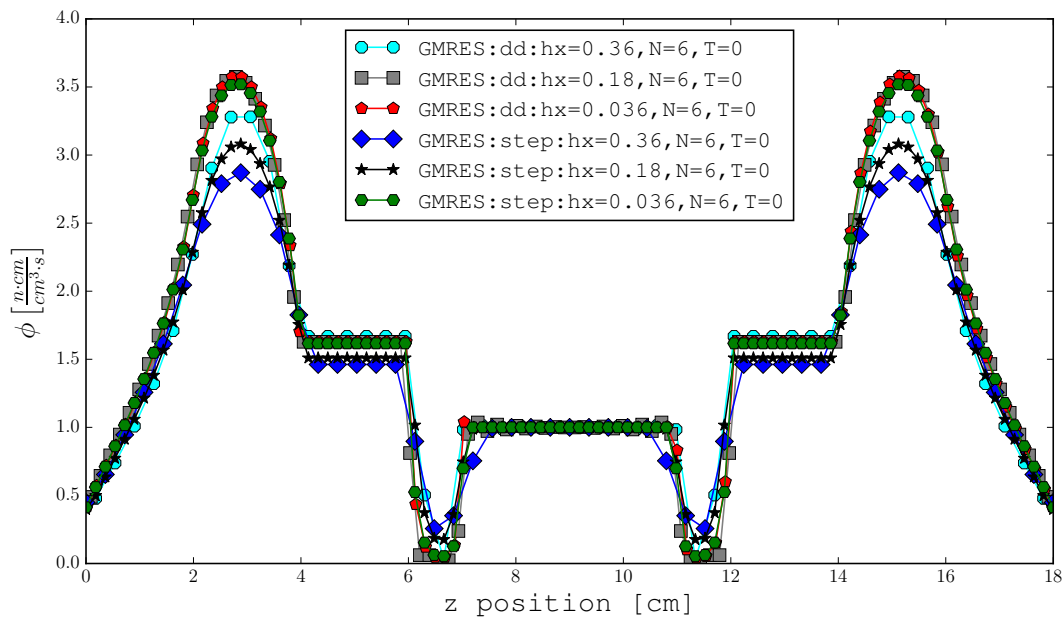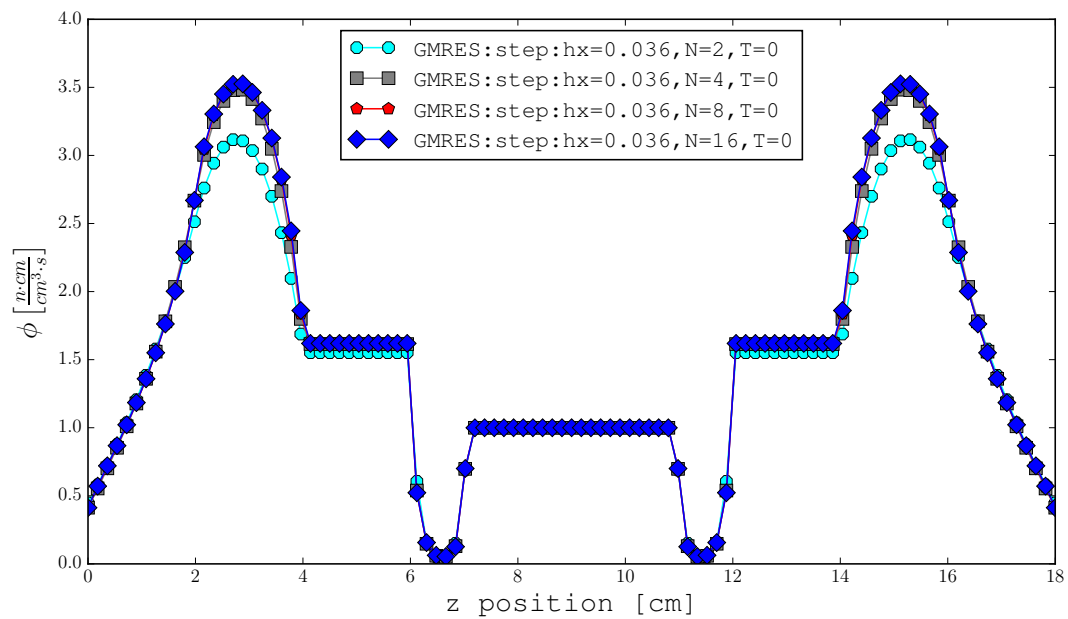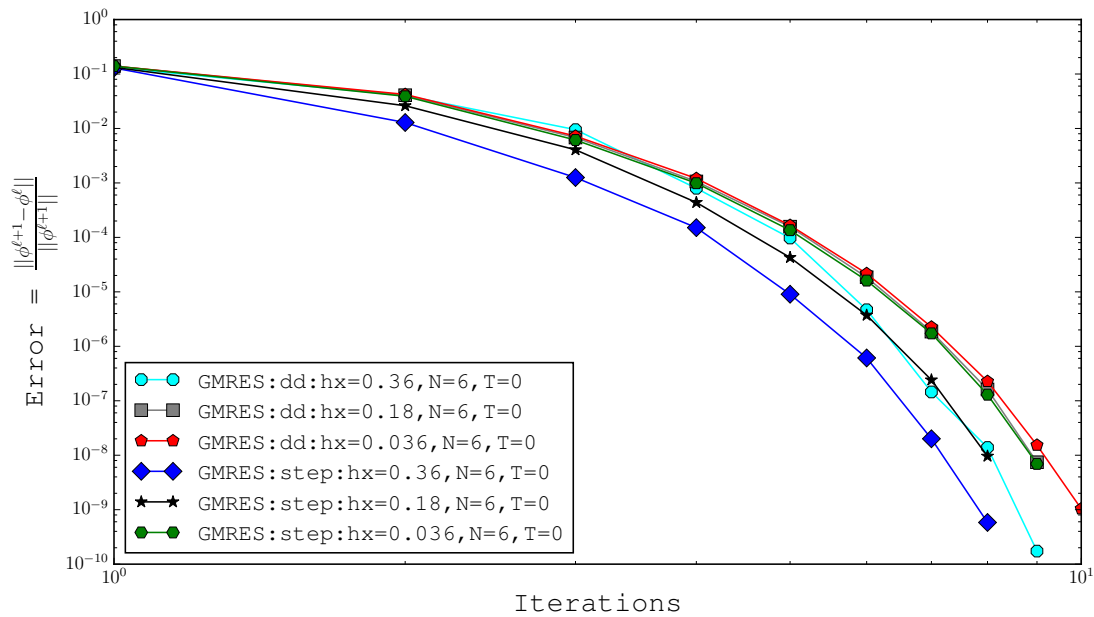
**Error will be determined with the following:**

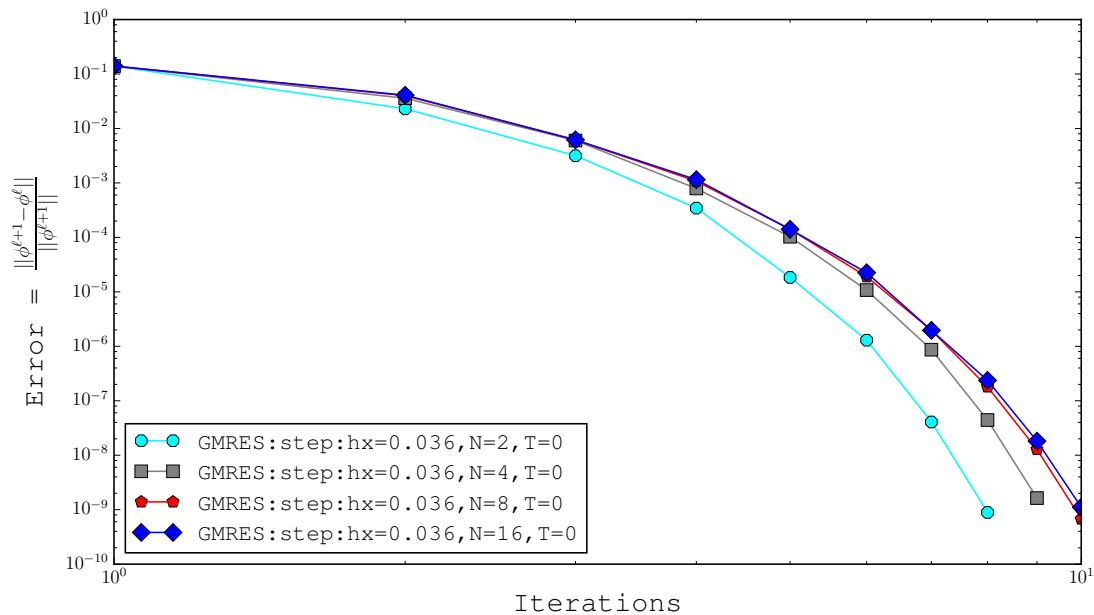$$\text{Error} = \frac{||\phi^{\ell+1} - \phi^{\ell}||}{||\phi^{\ell+1}||}$$
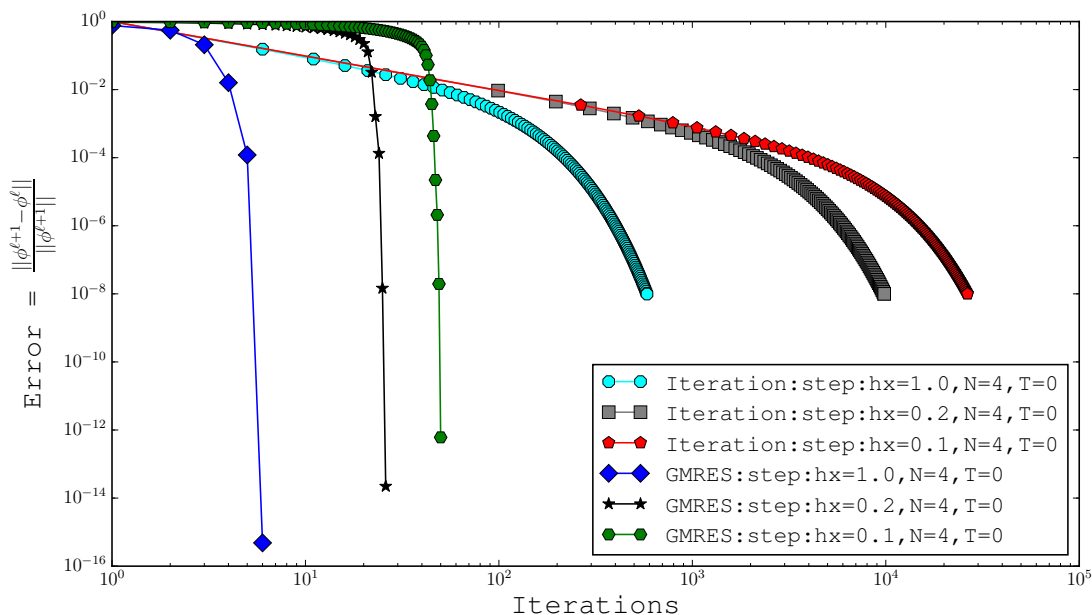
(d) (50 points) Solve Reed's problem (see finite difference diffusion codes). Present convergence plots for the solution in space and angle to a "refined" solution in space and angle.

Plots are below, reduced the number of points so that figures wouldn't take so long to load.

The solution converges with more spatial slices. Increasing the number of angular slices helps upto when N=4, but beyond that it doesn't do much.

(e) (50 points extra credit) Solve a time dependant problem for a slab surrounded by vacuum with $\Sigma_t = \Sigma_s = 1$ and initial condition given by $\boldsymbol{\psi(0) = 1/h_x}$ (original problem statement said $\phi(0) = 1/h_x$ and I'm not sure how to solve that). Plot the solution at $t = 1$ $s$, using step and diamond difference. The particles have a speed of 1 cm/s. Which discretization is better with a small time step? What do you see with a small number of ordinates compared to a really large number (100s)?

---
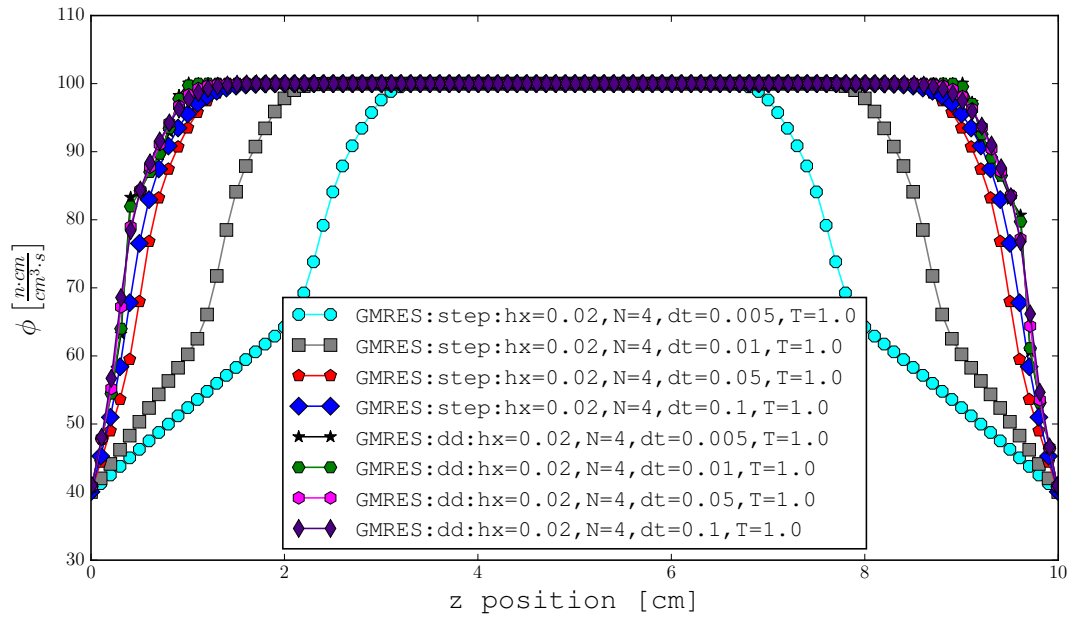
Figure 1: Q=0.01
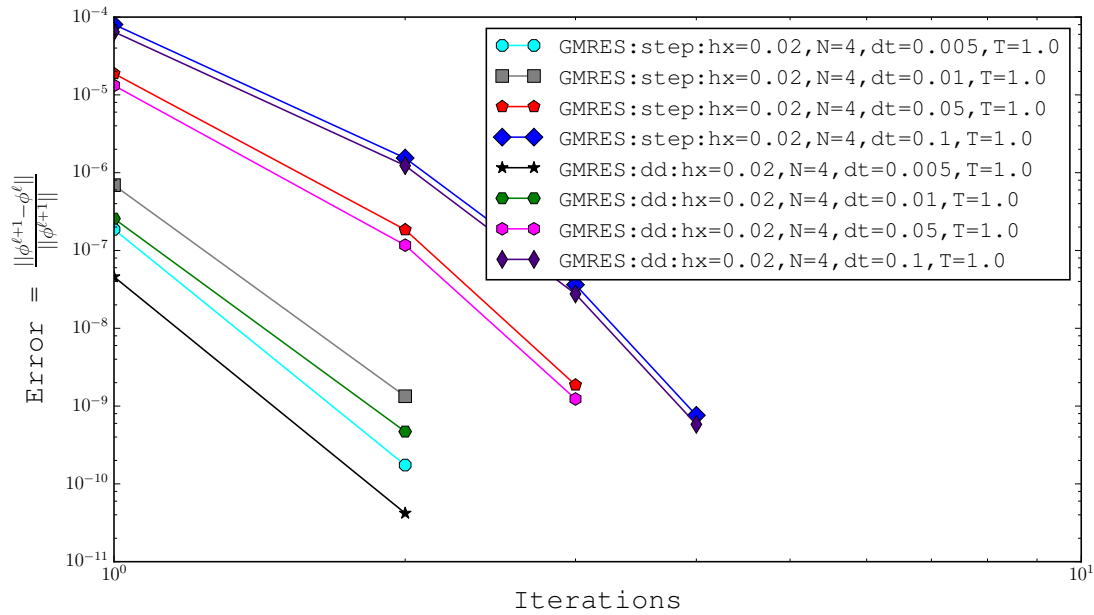


Figure 2: Q=0.01

Based on the above graphs, I am not sure which solution does better with a smaller step size. It depends on what the answer should be. I think the step solution, as the step size increases, look like

they have some nonphysical bends in the solution. This could be due to lots of things, but maybe its because of the smaller step size, which makes me think the diamond difference method is better with smaller step sizes.
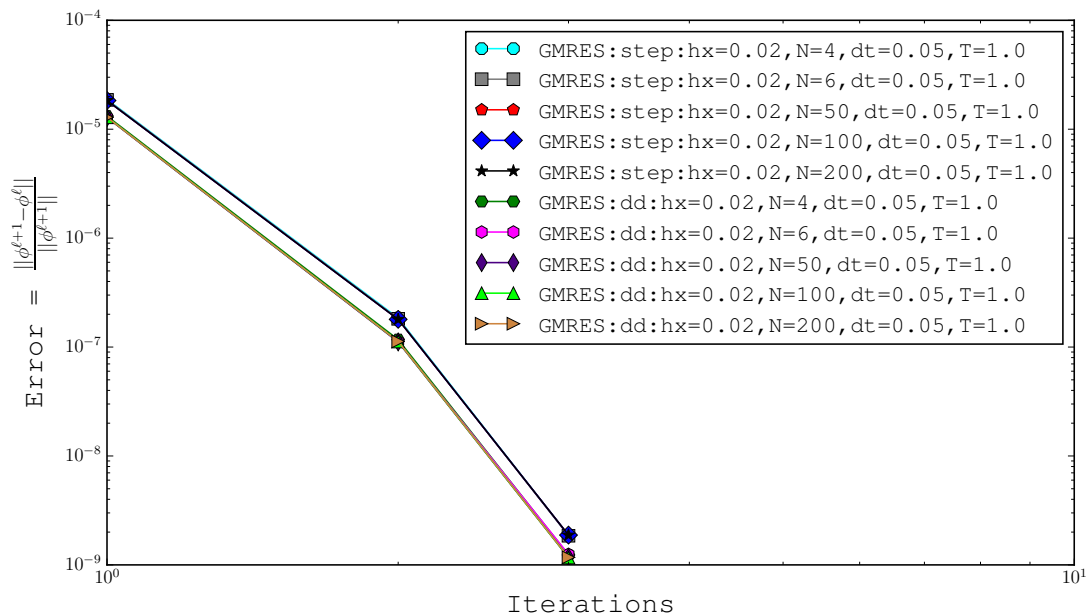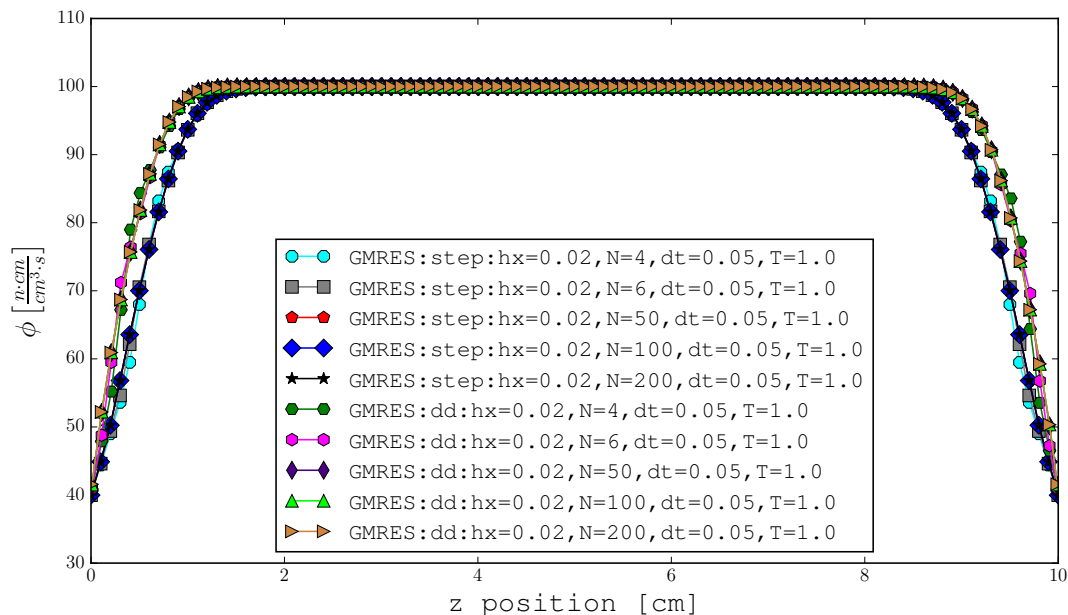




As the number of ordinances increase in the problem there isn't much change in the solution, which is expected because with the Quadrature rule we used, the integral can usually be expressed within

around 6 terms. There was an increase in computational time though.

# Homework 4 Code

Listing 1: **Main Code For Parts a,b and c**

```python
#!/usr/bin/env python3


###############################################################
##################### Import packages #######################
###############################################################


import time
start_time = time.time()
import Functions as f


###############################################################
######################### Inputs #############################
###############################################################


# Constants
Q = 0.01
Sigma_t = 100;Sigma_s=100
# Add adsorption to help converge
if Sigma_t==Sigma_s:
    Sigma_t=Sigma_t*1.0001

# Geometry
L = 10.                      # Width of slab
slices=[10,50,100,500]       # Number of cuts in slab (looped)
N = 4                        # Number of angle slices
BCs = f.np.zeros(N)          # Zero incoming flux

#Time
T=0                          # total Time (A plot made at T)
dt=1                         # Time steps width
v=1                          # Velocity

MAXITS=100000                # Max iterations for source iter
loud=False                   # Echo every Iteration?

#Method
Methods=['GMRES:step',        # 'Iteration' or 'GMRES'
        'GMRES:dd']          # Methods to solve with?
                             # 'step' or 'dd'

tol=1e-8

PlotError=False              # Do we plot the error?

NumOfPoints=100              # Max Number of points for plots


###############################################################
#################### Initialize Figures #####################
###############################################################

```

```
      Check=0
      fig=f.plt.figure(figsize=f.FigureSize)    # Plot all Methods
      ax=fig.add_subplot(111)
      if PlotError:
55        erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
          erax=erfig.add_subplot(111)                 # at T=0


      ################################################################
60    #################### Calculations ##############################
      ################################################################

      for Scheme in Methods:

65        Method=Scheme.split(':')[1]
          ########################################################
          #################### Set Up ############################
          ########################################################

70        for II in slices:
              if Method == 'step': #Step Dude needs one extra
                  I=II+1
              elif Method == 'dd':
                  I=II

75
              #Width, ang lists for materials
              hx = L/II
              q = f.np.ones(I)*Q
              Sig_t_discr = f.np.ones(I)*Sigma_t
80            Sig_s_discr = f.np.ones(I)*Sigma_s

              #Initialize psi (for time steps)
              if T==0:
                  psi=f.np.zeros((N,I))
85                Time=[0]
              else:
                  psi=f.np.ones((N,I))*(1/hx)
                  Time=f.Timevector(T,dt)

90            label_tmp=Scheme+":hx="+str(hx)+",N="+str(N)+",T="
              ########################################################
              #################### Determine phi #####################
              ########################################################

95            for t in Time: #Loop over time

                  label=label_tmp+str(t)

                  #Determine phi (new psi is determined for time steps)
100               x,phi,it,er,psi=f.solver(I,hx,q,Sig_t_discr,
                  Sig_s_discr,N,psi,v,dt,Time,BCs,Scheme,tol,MAXITS,loud)

                  ######################################################
```

```
               ################ Plot Information ###################
105            ####################################################
               fig
               ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
               if t==0 and PlotError:
                   erfig
110                erax,erfig=f.plotE(it,er,erax,label,erfig,
                                       Check,NumOfPoints)
               Check=Check+1


     #########################################################
115  ################### Legend/Save  ########################
     #########################################################

     fig
     f.Legend(ax)
120  #f.plt.savefig('Plots/FluxPlot.pdf')
     if PlotError:
         erfig
         f.Legend(erax)
         #f.plt.savefig('Plots/ErrorPlot.pdf')
125      f.plt.savefig('Plots/ErrorPlotTime.pdf')
         #f.plt.clf()
         f.plt.close()
     fig
     f.plt.savefig('Plots/FluxPlotTime.pdf')
130  #f.plt.show()


     #Why is tmp_psi in the GMRES going negative?


     ###################### Time To execute #################

135
     print("--- %s seconds ---" % (time.time() - start_time))
```

Listing 2: **Main Code For Part d**

```
     #!/usr/bin/env python3


     ################################################################
     ##################### Import packages #########################
5    ################################################################

     import time
     start_time = time.time()
     import Functions as f

10
     ################################################################
     ####################### Inputs #################################
     ################################################################

15   # Geometry
     L = 18.                         # Width of slab
     slices=[500]            # Number of cuts in slab (looped)
     NN = [2,4,8,16]                 # Number of angle slices
```

```python
20  #Time
    T=0                                 # total Time (A plot made at T)
    dt=1                                # Time steps width
    v=1                                 # Velocity

25  MAXITS=1000000                      # Max iterations for source iter
    loud=False                          # Echo every Iteration?

    #Method
    Methods=['GMRES:step']#,            # 'Iteration' or 'GMRES'
30         # 'GMRES:step']        # Methods to solve with?
                                    # 'step' or 'dd'

    tol=1e-8

35  PlotError=True                # Do we plot the error?

    NumOfPoints=100               # Max Number of points for plots

    ###############################################################
40  ################## Initialize Figures #######################
    ###############################################################

    Check=0
    fig=f.plt.figure(figsize=f.FigureSize)    # Plot all Methods
45  ax=fig.add_subplot(111)
    if PlotError:
        erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
        erax=erfig.add_subplot(111)                   # at T=0

50
    ###############################################################
    ##################### Calculations ##########################
    ###############################################################

55  for Scheme in Methods:

        Method=Scheme.split(':')[1]
        ########################################################
        #################### Set Up ##########################
60      ########################################################

        for II in slices:
            if Method == 'step': #Step Dude needs one extra
                I=II+1
65          elif Method == 'dd':
                I=II

            #Width, ang lists for materials
            hx = L/II
70          q = f.np.zeros(I)
            Sig_t_discr = f.np.zeros(I)
```

```
              Sig_s_discr = f.np.zeros(I)

              if Method == 'step':
75                x = f.np.linspace(0,(I-1)*hx,I)
              elif Method == 'dd':
                  x = f.np.linspace(hx/2,I*hx-hx/2,I)

              for i in range(0,len(x)):
80                q[i]=f.QReed(x[i])
                  Sig_t_discr[i]=f.Sigma_tReed(x[i])
                  Sig_s_discr[i]=Sig_t_discr[i]-f.Sigma_aReed(x[i])


85            for N in NN:
                  BCs = f.np.zeros(N)              # Zero incoming flux
                  #Initialize psi (for time steps)
                  if T==0:
                      psi=f.np.zeros((N,I))
90                    Time=[0]
                  else:
                      psi=f.np.ones((N,I))*(1/hx)
                      Time=f.Timevector(T,dt)

95                label_tmp=Scheme+":hx="+str(hx)+",N="+str(N)+",T="
                  #########################################################
                  #################### Determine phi #####################
                  #########################################################

100               for t in Time: #Loop over time

                      label=label_tmp+str(t)

                      #Determine phi (new psi is determined for time steps)
105                   x,phi,it,er,psi=f.solver(I,hx,q,Sig_t_discr,
                      Sig_s_discr,N,psi,v,dt,Time,BCs,Scheme,tol,MAXITS,loud)

                      #####################################################
                      ############### Plot Information ###################
110                   #####################################################
                      fig
                      ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
                      if t==0 and PlotError:
                          erfig
115                       erax,erfig=f.plotE(it,er,erax,label,erfig,
                                             Check,NumOfPoints)
                      Check=Check+1

    #########################################################
120 ################## Legend/Save  #######################
    #########################################################

    fig
    f.Legend(ax)
```

```python
125  #f.plt.savefig('Plots/FluxPlot.pdf')
     if PlotError:
         erfig
         f.Legend(erax)
         #f.plt.savefig('Plots/ErrorPlot.pdf')
130      f.plt.savefig('Plots/ErrorPlotReedVaryN.pdf')
         #f.plt.clf()
         f.plt.close()
     fig
     f.plt.savefig('Plots/FluxPlotReedVaryN.pdf')
135  #f.plt.show()


     #################### Time To execute ################


     print("--- %s seconds ---" % (time.time() - start_time))
```

Listing 3: **Main Code For Part e**

```python
     #!/usr/bin/env python3


     ##############################################################
     #################### Import packages #######################
5    ##############################################################


     import time
     start_time = time.time()
     import Functions as f

10

     ##############################################################
     ####################### Inputs #############################
     ##############################################################


15   # Geometry
     L = 10                         # Width of slab
     # Constants
     Q = 0.01
     Sigma_t = 1;Sigma_s=1
20   # Add adsorption to help converge
     if Sigma_t==Sigma_s:
         Sigma_t=Sigma_t*1.0001

     slices=[500]           # Number of cuts in slab (looped)
25   NN = [4,6,50,100,200]           # Number of angle slices

     #Time
     T=1                           # total Time (A plot made at T)
     dtt=[0.05]                    # Time steps width
30   v=1                           # Velocity

     MAXITS=1000000                # Max iterations for source iter
     loud=False                    # Echo every Iteration?

35   #Method
     Methods=['GMRES:step',        # 'Iteration' or 'GMRES'
```

```
                'GMRES:dd']        # Methods to solve with?
                                   # 'step' or 'dd'

40  tol=1e-8
    Ttol=1e-3


    PlotError=True              # Do we plot the error?

45  NumOfPoints=100                # Max Number of points for plots


    ##############################################################
    #################### Initialize Figures ######################
    ##############################################################
50
    Check=0
    fig=f.plt.figure(figsize=f.FigureSize)   # Plot all Methods
    ax=fig.add_subplot(111)
    if PlotError:
55      erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
        erax=erfig.add_subplot(111)                # at T=0



    ##############################################################
60  #################### Calculations ###########################
    ##############################################################

    for Scheme in Methods:

65      Method=Scheme.split(':')[1]
        ######################################################
        #################### Set Up ##########################
        ######################################################

70      for II in slices:
            if Method == 'step': #Step Dude needs one extra
                I=II+1
            elif Method == 'dd':
                I=II

75
            #Width, ang lists for materials
            hx = L/II
            q = f.np.ones(I)*Q
            Sig_t_discr = f.np.ones(I)*Sigma_t
80          Sig_s_discr = f.np.ones(I)*Sigma_s


            for N in NN:
                BCs = f.np.zeros(N)              # Zero incoming flux
85              for dt in dtt:
                    #Initialize psi (for time steps)
                    if T==0:
                        psi=f.np.zeros((N,I))
                        Time=[0]
```

```
90                 else:
                       psi=f.np.ones((N,I))*(1/hx)
                       Time=f.Timevector(T,dt)

                   label_tmp=Scheme+":hx="+str(hx)+",N="+str(N)+",dt="+\
95                         str(dt)+",T="
                   ###########################################################
                   ################### Determine phi #####################
                   ###########################################################

100                for t in Time: #Loop over time

                       label=label_tmp+str(round(t,3))

                       #Determine phi (new psi is determined for time steps)
105                    x,phi,it,er,psi=f.solver(I,hx,q,Sig_t_discr,
                       Sig_s_discr,N,psi,v,dt,Time,BCs,Scheme,tol,MAXITS,loud)

                       #####################################################
                       ############### Plot Information #################
110                    #####################################################
                       PlotQuestion=abs(t-dt)<Ttol\
                       or abs(t-0.5)<Ttol or abs(t-1)<Ttol
                       PlotQuestion=abs(t-1)<Ttol
                       if PlotQuestion:
115                        fig
                           ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
                           if PlotError:
                               erfig
                               erax,erfig=f.plotE(it,er,erax,label,erfig,
120                                            Check,NumOfPoints)
                           Check=Check+1

    ###########################################################
    ################### Legend/Save   ########################
125 ###########################################################

    fig
    f.Legend(ax)
    #f.plt.savefig('Plots/FluxPlot.pdf')
130 if PlotError:
        erfig
        f.Legend(erax)
        #f.plt.savefig('Plots/ErrorPlot.pdf')
        f.plt.savefig('Plots/ErrorPlotTimeVaryN.pdf')
135     #f.plt.clf()
        f.plt.close()
    fig
    f.plt.savefig('Plots/FluxPlotTimeVaryN.pdf')
    #f.plt.show()
140
    ###################### Time To execute #################
```

```python
    print("--- %s seconds ---" % (time.time() - start_time))
```

Listing 4: **Functions holder**

```python
#!/usr/bin/env python3


################################################################
##################### Import packages #########################
################################################################


import sys
import numpy as np
import scipy.sparse.linalg as spla

import scipy.special as sps
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "monospace"
import matplotlib
matplotlib.rc('text',usetex=True)
matplotlib.rcParams['text.latex.preamble']=[r"\usepackage{amsmath}"]
import random as rn
import matplotlib.mlab as mlab
import copy
import os


#############################################################
######################### Variables #########################
#############################################################


# Basic information
FigureSize = (11, 6)                 # Dimensions of the figure
TypeOfFamily='monospace'             # This sets the type of font for text
font = {'family' : TypeOfFamily}  # This sets the type of font for text
LegendFontSize = 12
Lfont = {'family' : TypeOfFamily}  # This sets up legend font
Lfont['size']=LegendFontSize

Title = ''
TitleFontSize = 22
TitleFontWeight = "bold"  # "bold" or "normal"

#Xlabel='E (eV)'    # X label
XFontSize=18             # X label font size
XFontWeight="normal"   # "bold" or "normal"
XScale="linear"        # 'linear' or 'log'
XScaleE='log'       # Same but for error plot

YFontSize=18                     # Y label font size
YFontWeight="normal"             # "bold" or "normal"
YScale="linear"                  # 'linear' or 'log'
YScaleE='log'

Check=0
```

```
    Colors=["aqua","gray","red","blue","black",
                    "green","magenta","indigo","lime","peru","steelblue",
                    "darkorange","salmon","yellow","lime","black"]
55
    # If you want to highlight a specific item
    # set its alpha value =1 and all others to 0.4
    # You can also change the MarkSize (or just use the highlight option below)
    Alpha_Value=[1  ,1  ,1  ,1  ,1  ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
60  MarkSize=   [8  ,8  ,8  ,8  ,8  ,8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]

    Linewidth=[1  ,1  ,1  ,1  ,1  ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

    # Can change all these to "." or "" for nothing "x" isn't that good
65  MarkerType=["8","s","p","D","*","H","h","d","^",">"]

    # LineStyles=["solid","dashed","dash_dot","dotted","."]
    LineStyles=["solid"]

70  SquishGraph = 0.75
    BBOXX = 1.24
    BBOXY = 0.5          # Set legend on right side of graph

    NumberOfLegendColumns=1
75
    Xlabel='z position [cm]'
    Ylabel="$\phi\left[\\frac{n\cdot cm}{cm^3\cdot s}\\right]$"

    XlabelE='Iterations'
80  YlabelE="Error = $\\frac{||\phi^{\ell+1}-\phi^\ell||}{||\phi^{\ell+1}||}$"

    ##############################################################
    ######################## Functions ##########################
    ##############################################################
85
    def Sigma_tReed(r):
        value = 0 + ((1.0*(r>=14) + 1.0*(r<=4)) +
                     5.0 *((np.abs(r-11.5)<0.5) or (np.abs(r-6.5)<0.5)) +
                     50.0 * (np.abs(r-9)<=2) )
90      return value;
    def Sigma_aReed(r):
        value = 0 + (0.1*(r>=14) + 0.1*(r<=4) +
                     5.0 *((np.abs(r-11.5)<0.5) or (np.abs(r-6.5)<0.5)) +
                     50.0 * (np.abs(r-9)<=2) )
95      return value;
    def QReed(r):
        value = 0 + 1.0*((r<16) * (r>14))+ 1.0*((r>2) * (r<4)) + 50.0*(np.abs(r-9)<=2)
        return value;

100 def Timevector(T,dt):
        Time=[dt]
        while Time[-1]<T:
            Time.append(Time[-1]+dt)
```

```python
        return(Time)
105
    def diamond_sweep1D(I,hx,q,sigma_t,mu,boundary):
        """Compute a transport diamond difference sweep for a given
        Inputs:
          I:               number of zones
110       hx:              size of each zone
          q:               source array
          sigma_t:         array of total cross-sections
          mu:              direction to sweep
          boundary:        value of angular flux on the boundary
115     Outputs:
          psi:             value of angular flux in each zone
        """
        assert(np.abs(mu) > 1e-10)
        psi = np.zeros(I)
120     ihx = 1./hx
        if (mu > 0):
          psi_left = boundary
          for i in range(I):
            psi_right = (q[i] + (mu*ihx-0.5*sigma_t[i])*psi_left)\
125                     /(0.5*sigma_t[i] + mu*ihx)
            psi[i] = 0.5*(psi_right + psi_left)
            psi_left = psi_right
        else:
          psi_right = boundary
130       for i in reversed(range(I)):
            psi_left = (q[i]+ (-mu*ihx-0.5*sigma_t[i])*psi_right)\
                      /(0.5*sigma_t[i] - mu*ihx)
            psi[i] = 0.5*(psi_right + psi_left)
            psi_right = psi_left
135     return psi

    def step_sweep1D(I,hx,q,sigma_t,mu,boundary):
        """Compute a transport step sweep for a given
        Inputs:
140       I:               number of zones
          hx:              size of each zone
          q:               source array
          sigma_t:         array of total cross-sections
          mu:              direction to sweep
145       boundary:        value of angular flux on the boundary
        Outputs:
          psi:             value of angular flux in each zone
        """
        assert(np.abs(mu) > 1e-10)
150     psi = np.zeros(I)
        ihx = 1./hx
        if (mu > 0):
          psi_left = boundary
          psi[0] = 0
155       for i in range(1,I):
            psi_right = (q[i] + mu*ihx*psi_left)/(mu*ihx + sigma_t[i])
```

```
            psi[i] = 0.5*(psi_right + psi_left)
            psi_left = psi_right
        else:
160         psi_right = boundary
            psi[-1] = 0
            for i in reversed(range(0,I-1)):
                psi_left = (q[i] - mu*ihx*psi_right)/(sigma_t[i] - mu*ihx)
                psi[i] = 0.5*(psi_right + psi_left)
165             psi_right = psi_left
        return psi


    def source_iteration(I,hx,q,sigma_t,sigma_s,N,psiprevioustime,
170                        v,dt,Time,BCs,sweep_type,
                           tolerance = 1.0e-8,maxits = 100, LOUD=False ):
        """Perform source iteration for single-group steady state problem
        Inputs:
            I:               number of zones
175         hx:              size of each zone
            q:               source array
            sigma_t:         array of total cross-sections
            sigma_s:         array of scattering cross-sections
            N:               number of angles
180         BCs:             Boundary conditions for each angle
            sweep_type:      type of 1D sweep to perform solution
            tolerance:       the relative convergence tolerance for the iterations
            maxits:          the maximum number of iterations
            LOUD:            boolean to print out iteration stats
185     Outputs:
            x:               value of center of each zone
            phi:             value of scalar flux in each zone
        """
        iterations = []
190     Errors = []
        phi = np.zeros(I)
        phi_old = phi.copy()
        converged = False
        MU, W = np.polynomial.legendre.leggauss(N)
195     iteration = 1
        tmp_psi=psiprevioustime.copy()
        if len(Time)==1:
            sigma_ts=sigma_t
        else:
200         sigma_ts=sigma_t+1/(v*dt)

        while not(converged):
            phi = np.zeros(I)
            #sweep over each direction
205         for n in range(N):
                #qs=(q*W[n])/2+(phi_old*sigma_s)/2+psiprevioustime[n,:]/(v*dt)
                qs=(q)/2+(phi_old*sigma_s)/2+psiprevioustime[n,:]/(v*dt)
                if sweep_type == 'dd':
                    tmp_psi[n,:] = diamond_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
```

```python
210          elif sweep_type == 'step':
                 tmp_psi[n,:] = step_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
             else:
                 sys.exit("Sweep method specified not defined in SnMethods")
             phi = phi+tmp_psi[n,:]*W[n]
215      #check convergence
         change = np.linalg.norm(phi-phi_old)/np.linalg.norm(phi)
         iterations.append(iteration)
         Errors.append(change)
         #iterations.append(iteration)
220      #Errors.append(change)
         converged = (change < tolerance) or (iteration > maxits)
         if (LOUD>0) or (converged and LOUD<0):
           print("Iteration",iteration,": Relative Change =",change)
         if (iteration > maxits):
225        print("Warning: Source Iteration did not converge : "+\
                   sweep_type+", I : "+str(I)+", Diff : %.2e" % change)
         #Prepare for next iteration
         iteration += 1
         phi_old = phi.copy()
230    if sweep_type == 'step':
           x = np.linspace(0,(I-1)*hx,I)
       elif sweep_type == 'dd':
           x = np.linspace(hx/2,I*hx-hx/2,I)
       return x, phi, iterations, Errors, tmp_psi
235


   def gmres_solve(I,hx,q,sigma_t,sigma_s,N,psiprevioustime,
                   v,dt,Time,BCs, sweep_type,
                   tolerance = 1.0e-8,maxits = 100, LOUD=False,
240                restart = 20 ):
       """Solve, via GMRES, a single-group steady state problem
       Inputs:
       I:              number of zones
       hx:             size of each zone
245    q:              source array
       sigma_t:        array of total cross-sections
       sigma_s:        array of scattering cross-sections
       N:              number of angles
       BCs:            Boundary conditions for each angle
250    sweep_type:     type of 1D sweep to perform solution
       tolerance:      the relative convergence tolerance for the iterations
       maxits:         the maximum number of iterations
       LOUD:           boolean to print out iteration stats
       Outputs:
255    x:              value of center of each zone
       phi:            value of scalar flux in each zone
       """
       iterations = []
       Errors = []
260
       #compute RHS side
       RHS = np.zeros(I)
```

```python
      MU, W = np.polynomial.legendre.leggauss(N)
265   tmp_psi=psiprevioustime.copy()
      if len(Time)==1:
          sigma_ts=sigma_t
      else:
          sigma_ts=sigma_t+1/(v*dt)
270
      for n in range(N):
        qs=q/2+psiprevioustime[n,:]/(v*dt)
        if sweep_type == 'dd':
          tmp_psi[n,:] = diamond_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
275     elif sweep_type == 'step':
          tmp_psi[n,:] = step_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
        #tmp_psi = sweep1D(I,hx,q,sigma_t,MU[n],BCs[n])
        RHS += tmp_psi[n,:]*W[n]

280   #define linear operator for gmres
      def linop(phi):
        tmp = phi*0
        #sweep over each direction
        for n in range(N):
285       if sweep_type == 'dd':
            tmp_psi[n,:] = diamond_sweep1D(I,hx,(phi*sigma_s)/2,
                                    sigma_ts,MU[n],BCs[n])
          elif sweep_type == 'step':
            tmp_psi[n,:] = step_sweep1D(I,hx,(phi*sigma_s)/2,
290                                   sigma_ts,MU[n],BCs[n])
          tmp += tmp_psi[n,:]*W[n]
        return phi-tmp
      A = spla.LinearOperator((I,I), matvec = linop, dtype='d')

295
      #define a little function to call when the iteration is called
      iteration = np.zeros(1)
      def callback(rk, iteration=iteration):
        iteration += 1
300     if (LOUD>0):
          print("Iteration",iteration[0],"norm of residual",np.linalg.norm(rk))
        iterations.append(iteration[0])
        Errors.append(np.linalg.norm(rk))

305   #Do the GMRES Solve
      phi,info = spla.gmres(A,RHS,x0=RHS,tol=tolerance,
                            restart=int(restart),callback=callback)

      #Print important information
310   if (LOUD):
        print("Finished in",iteration[0],"iterations.")
      if (info >0):
        print("Warning, convergence not achieved :"+str(sweep_type)+" "+str(hx))
      if sweep_type == 'step':
315       x = np.linspace(0,(I-1)*hx,I)
```

```python
      elif sweep_type == 'dd':
          x = np.linspace(hx/2,I*hx-hx/2,I)

      #Calculate Psi for time iterations
320   phi2 = np.zeros(I)
      #sweep over each direction
      for n in range(N):
          #qs=(q*W[n])/2+(phi_old*sigma_s)/2+psiprevioustime[n,:]/(v*dt)
          qs=(q)/2+(phi*sigma_s)/2+psiprevioustime[n,:]/(v*dt)
325       if sweep_type == 'dd':
              tmp_psi[n,:] = diamond_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
          elif sweep_type == 'step':
              tmp_psi[n,:] = step_sweep1D(I,hx,qs,sigma_ts,MU[n],BCs[n])
          else:
330           sys.exit("Sweep method specified not defined in SnMethods")
          phi2 = phi2+tmp_psi[n,:]*W[n]

      return x, phi, iterations, Errors,tmp_psi

335 def solver(I,hx,q,Sig_t,Sig_s,N,psi,v,dt,Time,BCs,Scheme,tol,MAXITS,loud):
      Method=Scheme.split(':')[1]
      if "Iteration" in Scheme:
          x, phi, iterations, errors, psi =source_iteration(I,
              hx,q,Sig_t,Sig_s,N,psi,v,dt,Time,BCs,
340           Method,tolerance=tol,maxits=MAXITS,LOUD=loud)
      elif "GMRES" in Scheme:
          x, phi, iterations, errors, psi =gmres_solve(I,
              hx,q,Sig_t,Sig_s,N,psi,v,dt,Time,BCs,
              Method,tolerance=tol,maxits=MAXITS,LOUD=loud,restart=MAXITS)
345   else:
          print("Improper sweep selected")
          quit()
      return x, phi, iterations, errors,psi


350 ################################################################
      ################# Plotting Function #########################
      ################################################################

      def reduceList(List,N):
355   List2=[List[0]]
      Div=int(len(List)/N)
      for i in range(1,len(List)-1):
          if i % Div == 0:
              List2.append(List[i])
360   List2.append(List[-1])
      return(List2)

      def loop_values(list1,index):
          """
365       This function will loop through values in list even if
          outside range (in the positive sense not negative)
          """
          while True:
```

```python
                try:
370                     list1[index]
                        break
                except IndexError:
                        index=index-len(list1)
        return(list1[index])
375
    def plot(x,y,ax,label,fig,Check,NumOfPoints):
        if len(x)>300:
            x=reduceList(x,NumOfPoints)
            y=reduceList(y,NumOfPoints)
380     #Plot X and Y
        ax.plot(x,y,
                    linestyle=loop_values(LineStyles,Check),
                    marker=loop_values(MarkerType,Check),
                    color=loop_values(Colors,Check),
385                 markersize=loop_values(MarkSize,Check),
                    alpha=loop_values(Alpha_Value,Check),
                    label=label)


        #Log or linear scale?
390     ax.set_xscale(XScale)
        ax.set_yscale(YScale)
        #Set Title
        fig.suptitle(Title,fontsize=TitleFontSize,
                    fontweight=TitleFontWeight,fontdict=font,
395                                                 ha='center')
        #Set X and y labels
        ax.set_xlabel(Xlabel,
                        fontsize=XFontSize,fontweight=XFontWeight,
                        fontdict=font)
400     ax.set_ylabel(Ylabel,
                        fontsize=YFontSize,
                        fontweight=YFontWeight,
                        fontdict=font)
        return(ax,fig)
405

    def plotE(x,y,erax,label,erfig,Check,NumOfPoints):
        if len(x)>300:
            x=reduceList(x,NumOfPoints)
410         y=reduceList(y,NumOfPoints)
        #Plot X and Y
        erax.plot(x,y,
                    linestyle=loop_values(LineStyles,Check),
                    marker=loop_values(MarkerType,Check),
415                 color=loop_values(Colors,Check),
                    markersize=loop_values(MarkSize,Check),
                    alpha=loop_values(Alpha_Value,Check),
                    label=label)


420     #Log or linear scale?
        erax.set_xscale(XScaleE)
```

```
          erax.set_yscale(YScaleE)
          #Set Title
          erfig.suptitle(Title,fontsize=TitleFontSize,
425                  fontweight=TitleFontWeight,fontdict=font,
                                                    ha='center')
          #Set X and y labels
          erax.set_xlabel(XlabelE,
                      fontsize=XFontSize,fontweight=XFontWeight,
430                  fontdict=font)
          erax.set_ylabel(YlabelE,
                      fontsize=YFontSize,
                      fontweight=YFontWeight,
                      fontdict=font)
435      return(erax,erfig)

     def Legend(ax):
          handles,labels=ax.get_legend_handles_labels()
          ax.legend(handles,labels,loc='best',
440              fontsize=LegendFontSize,prop=font)
          return(ax)

     # def Legend(ax):
     #          handles,labels=ax.get_legend_handles_labels()
445  #          box=ax.get_position()
     #          ax.set_position([box.x0, box.y0, box.width*SquishGraph,
     #                          box.height])
     #          ax.legend(handles,labels,loc='center',
     #                      bbox_to_anchor=(BBOXX,BBOXY),
450  #                      fontsize=LegendFontSize,prop=font,
     #                      ncol=NumberOfLegendColumns)
     #          return(ax)
```

# Homework 5 Problem Statement

Solve the following problem and submit a detailed report, including a justification of why a reader should believe your results.

**Clean Fusion Energy**

(100 points) Consider a thermonuclear fusion reactor producing neutrons of energy 14.1 and 2.45 MeV. The reactor is surrounded by FLiBe (a 2:1 mixture of LiF and $BeF_2$) to convert the neutron energy into heat. All the constituents in the FLiBe have their natural abundances. Using data from JANIS, and assuming the total neutron flux is $10^{14}$ n/cm$^2$·s. Perform the following analyses.

(a) (25 points) Write out the depletion (or in thise case activation) chains that will occur in the system.

(b) (50 points) Over a two-year cycle compute the inventory of nuclides in the system using two methods discussed in class. What is the maximum concentration of tritium?

(c) (25 points) After discharging the FLiBe blanket, how long will it take until the material is less radioactive than Brazil Nuts ? (444 Bq/kg)

# Homework 5 Background

Please note, that most of this background is copied directly from Dr. McClarren's notes, but are reproduced here.

The production of an isotope is dictated by production and loss

$$\frac{dn_i}{dt} = -\lambda_i^{eff} n_i + \sum_{j=1}^{N} b_{j \to i}^{eff} \lambda_j^{eff} n_j$$

Where,

$$\lambda_i^{eff} = \lambda_i + \phi \sum_{j=1}^{N} \sigma_{i \to j}$$

and

$$b_{j \to i}^{eff} = \frac{b_{j \to i} \lambda_j + \sigma_{j \to i} \phi}{\lambda_j^{eff}}$$

For a system of isotopes, this can be reduced to:

$$\frac{d\vec{n}}{dt} = \boldsymbol{A}\vec{n}(t)$$

Where $\boldsymbol{A}$ is a matrix whose diagonal elements are $[-\lambda_1^{eff}, -\lambda_2^{eff}, ..., -\lambda_N^{eff}]$, all off diagonal elements are $b_{j \to i}^{eff} \lambda_j^{eff}$ (i for the diagonal, and j is for the off diagonal position) and $\vec{n}(t) = [n_1, n_2, ..., n_N]$.

The solution to this system is obvious (it wasn't to me at first - but that's because I'm a newb)

$$\vec{n} = e^{\boldsymbol{A}t} \vec{n}_0$$

Determing $e^{\boldsymbol{A}t} \vec{n}_0$ will be done 3 different ways,

**Matrix Exponential**

Analytic Solution, unstable with large N.

$$\vec{n}(t) = e^{\boldsymbol{A}t} \vec{n}_0 \approx \left[ \sum_{m=0}^{\infty} \frac{1}{m!} \boldsymbol{A}^m t^m \right] \vec{n}_0$$

**Backward Euler**

Unstable for large $\Delta t$, but can take time steps.

$$\frac{d\vec{n}}{dt} \approx \frac{\vec{n}(\Delta t) - \vec{n}_0}{\Delta t} \approx \boldsymbol{A}\vec{n}(\Delta t)$$
$$\vec{n}(\Delta t) \approx (\boldsymbol{I} - \boldsymbol{A}\Delta t)^{-1} \vec{n}_0$$

**Rational Approximation**

$$\vec{n}(t) = e^{\boldsymbol{A}t} \vec{n}_0 \approx -2\Re \sum_{k=1}^{N/2} c_k (z_k \boldsymbol{I} - \boldsymbol{A}t)^{-1} \vec{n}_0$$

    

The $\Re$ symbol means taking the real part of the solution. Further,

$$c_k = \frac{i}{N}e^{z_k}w_k$$

where $z_k$ and $w_k$ are both scalers defined as

$$z_k = \phi(\theta_k)$$
$$w_k = \phi'(\theta_k)$$

with

$$\phi(\theta) = N[0.1309 - 0.1194\theta^2 + 0.2500i\theta]$$
$$\text{or}$$
$$\phi(\theta) = 2.246N\left[1 - sin(1.1721 - 0.3443i\theta]\right.$$
$$\text{or}$$
$$\phi(\theta) = N[0.5071\theta cot(0.6407\theta) - 0.6122 + 0.2645i\theta]$$
$$\text{or}$$
$$\phi(\theta) = \text{Best Possible}$$

and

$$\theta_k = \pm\frac{\pi}{N}(1 + 2k) \quad \text{k from 0 to N-1}$$

Where $N$ doesn't have to go much higher than 10 to have low errors (for the best Rational Approximation). Also both plus and minus terms were written here, but the first equation in this solution method only uses the positive terms. This is because using the negative $\beta$'s yields the same real part as the positive $\beta$'s, with opposite complex parts (the complex cancels). Thats why the $\sum$ only goes to $N/2$ and the solution is multiplied by 2.

The solution utilized the assumption that half the neutron flux was 14 MeV and the other half was 2.45 MeV. The $A$ matrix was built in terms of days and atoms per kg of initial fuel.

$$1 \text{ Kg FLiBe} \cdot \frac{6.022E23}{98.9 \text{ g}} = 6.09E24 \text{ atoms of FLiBe}$$

This number was the starting condition for $^9$Be. Twice this number (with natural abundancy considerations) for $^6$Li and $^7$Li, and 4 times this number for $^{19}$F.

# Homework 5 Solution

(a) (25 points) Write out the depletion (or in thise case activation) chains that will occur in the system.

The chains were written out in a form so that the diagram would be uncluttered, and not all possible reactions are displayed.

**$^{10}_{5}$B** cell:
$(n,\alpha)^7_3Li$　　Stable - 19.9%
2.45 MeV: 0.281
14.1 MeV: 0.0445

**$^{8}_{4}$Be** cell:
$\alpha \to {}^4_2He$
$^8_4$Be
6E-17 s

**$^{9}_{4}$Be** cell:
$(n,2n)^8_4Be$　　Stable - 100%
$(n,t)^7_3Li$
$(n,\alpha)^6_2He$
$(n,\gamma)^{10}_4Be$
$(n,p)^9_3Li$
$^9_4$Be
2.45 MeV: 0.021, 0, 0.083
14.1 MeV: 0.48, 0.021, 0.01

**$^{10}_{4}$Be** cell:
$\beta^{-1} \to {}^{10}_5B$
$^{10}_4$Be
1.4E6 y

**$^{6}_{3}$Li** cell:
$(n,t)^4_2He$　　Stable - 7.59%
$(n,2n\,\alpha)^1_1H$
$(n,p)^6_2He$
$(n,\gamma)^7_3Li$
$^6_3$Li
2.45 MeV: 0.21,0,0
14.1 MeV: 0.03,0.08,0.01

**$^{7}_{3}$Li** cell:
$(n,2n)^6_3Li$　　Stable - 92.41%
$(n,2n\,\alpha)^2_1H$
$(n,d)^6_2He$
$(n,3n\,\alpha)^1_1H$
$(n,\gamma)^8_3Li$
$^7_3$Li
2.45 MeV: 0,0,0
14.1 MeV: 0.03,0.02,0.01

**$^{8}_{3}$Li** cell:
$\beta^{-1} \to {}^8_4Be$
$^8_3$Li
0.8399 s

**$^{9}_{3}$Li** cell:
$\beta^{-1} \to {}^9_4Be$ (49.2)
$\beta^{-1}n \to {}^8_4Be$ (50.8)
$^9_3$Li
0.178 s

**$^{3}_{2}$He** cell:
$(n,p)^3_1He$　　Stable - 0.0001%
$(n,d)^2_1H$
$(n,\gamma)^4_2He$
$^3_2$He
2.45 MeV: 0.71,0,9E-5
14.1 MeV: 0.12,0.08,3E-5

**$^{4}_{2}$He** cell:
Stable - 99.9999%
$^4_2$He

**$^{6}_{2}$He** cell:
$\beta^{-1} \to {}^6_3Li$
$^6_2$He
0.8 s

**$^{2}_{1}$H** cell:
$(n,2n)^1_1H$　　Stable - 0.0115%
$(n,\gamma)^3_1H$
$^2_1$H
2.45 MeV: 0,8E-6
14.1 MeV: 0.17,9E-6

**$^{3}_{1}$H** cell:
$(n,2n)^2_1H$　　$\beta^{-1} \to {}^3_2He$
$^3_1$H
12.32 y
2.45 MeV: 0
14.1 MeV: 0.05

Stable - 90.48%

$^{20}_{10}$**Ne**

$EC \rightarrow\,^{18}_{8}O$

| $(n, n\,\alpha)\,^{15}_{7}N$ | Stable - 100% |
| --- | --- |

$(n, n\,p)\,^{18}_{8}O$
$(n, 2n)\,^{18}_{9}F$
$(n, \alpha)\,^{16}_{7}N$
$(n, d)\,^{18}_{8}O$
$(n, p)\,^{19}_{8}O$
$(n, t)\,^{17}_{8}O$
$(n, \gamma)\,^{20}_{9}F$

$\beta^{-1} \rightarrow\,^{20}_{10}Ne$

$^{18}_{9}$**F**

$^{19}_{9}$**F**

$^{20}_{9}$**F**

**110 min**

**11.1 s**

**2.45 MeV: 0,0,0**
**14.1 MeV: 0.4,0.06,0.04**

$(n, \alpha)\,^{13}_{6}C$    Stable - 99.757%
$(n, p)\,^{16}_{7}N$
$(n, d)\,^{15}_{7}N$
$(n, \gamma)\,^{17}_{8}O$

$(n, \alpha)\,^{14}_{6}C$    Stable - 0.038%
$(n, 2n)\,^{16}_{8}O$
$(n, n\,\alpha)\,^{13}_{6}C$
$(n, p)\,^{17}_{7}N$
$(n, d)\,^{16}_{7}N$
$(n, \gamma)\,^{18}_{8}O$

Stable - 0.21%

$\beta^{-1} \rightarrow\,^{19}_{9}F$

$^{16}_{8}$**O**

$^{17}_{8}$**O**

$^{18}_{8}$**O**

$^{19}_{8}$**O**

**26.9 s**

**2.45 MeV: 0,0,0**
**14.1 MeV: 0.14,0.04,0.02**

**2.45 MeV: 0.12,0,0**
**14.1 MeV: 0.3,0.1,0.04**

$(n, 2n)\,^{14}_{7}N$    Stable - 0.364%
$(n, \alpha)\,^{12}_{5}B$
$(n, n\,p)\,^{14}_{6}C$
$(n, t)\,^{13}_{6}C$
$(n, p)\,^{15}_{6}C$
$(n, d)\,^{14}_{6}C$

$EC \rightarrow\,^{16}_{8}O$

$^{15}_{7}$**N**

$^{16}_{7}$**N**

**7.13 s**

**2.45 MeV: 0,0,0**
**14.1 MeV: 0.11,0.07,0.04**

(b) (50 points) Over a two-year cycle compute the inventory of nuclides in the system using two methods discussed in class. What is the maximum concentration of tritium?

The matrix exponential method did not converge. The Backwards Euler and the Rational approximation solutions did converge. Their answers are basically the same (all plots will have both solutions, but their answers are on top of one another). It should be noted that I changed the solver from in the notes from a GMRES solver to a normal matrix inverter for the rational approach to speed up the algorithm and give better results.
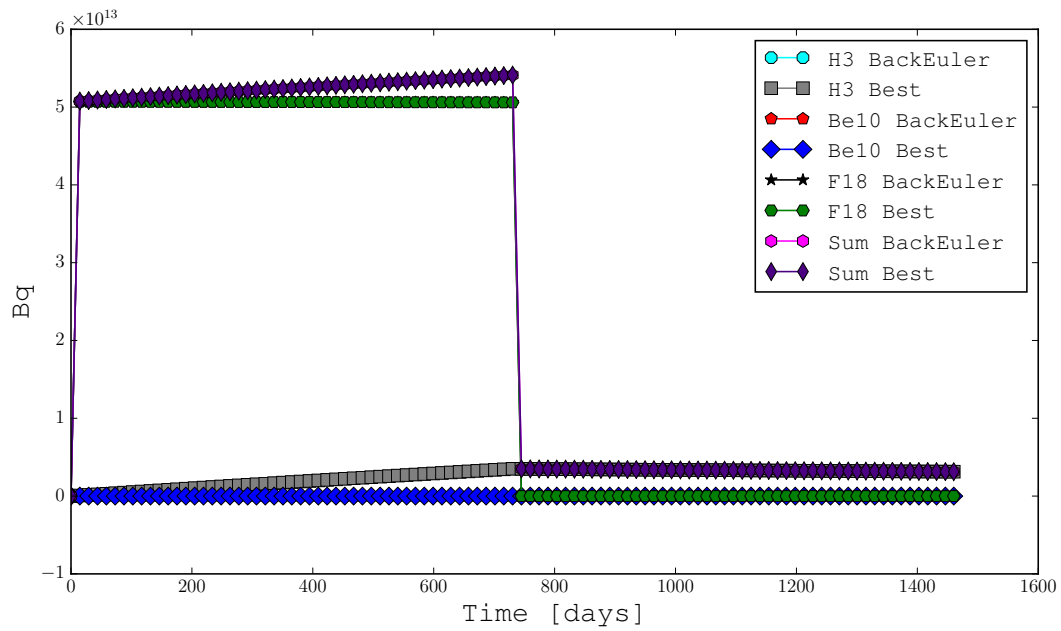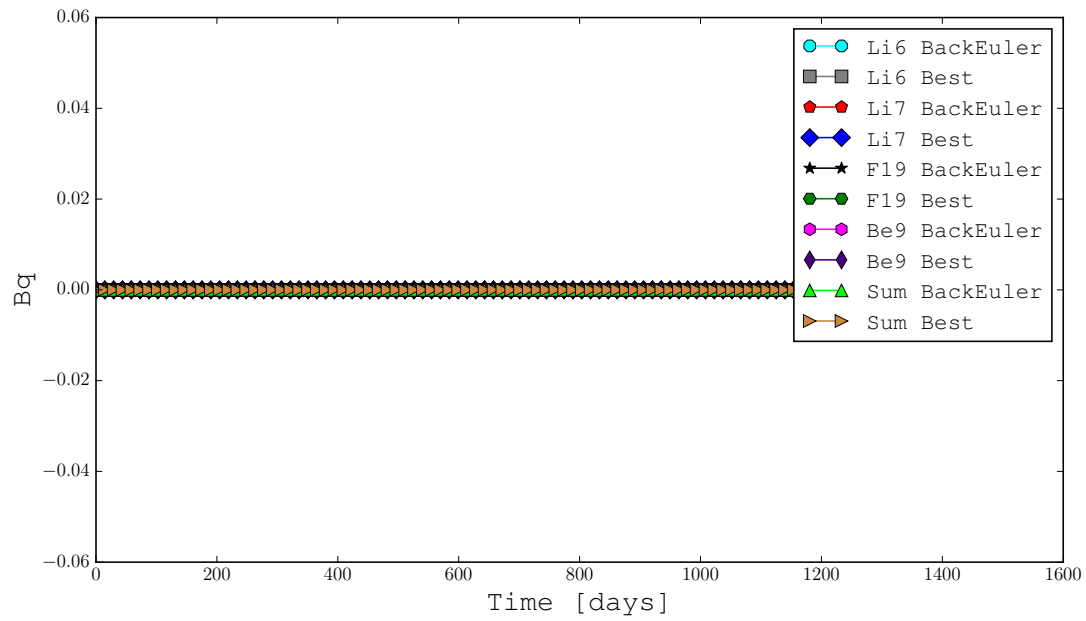
Plots were split up into two groups, with initial nuclides, and product nuclides which would have appreciable activity after a couple of days. Below plots are shown in grams.

The rational approximation used the "Best" method with 10 quadrature points,
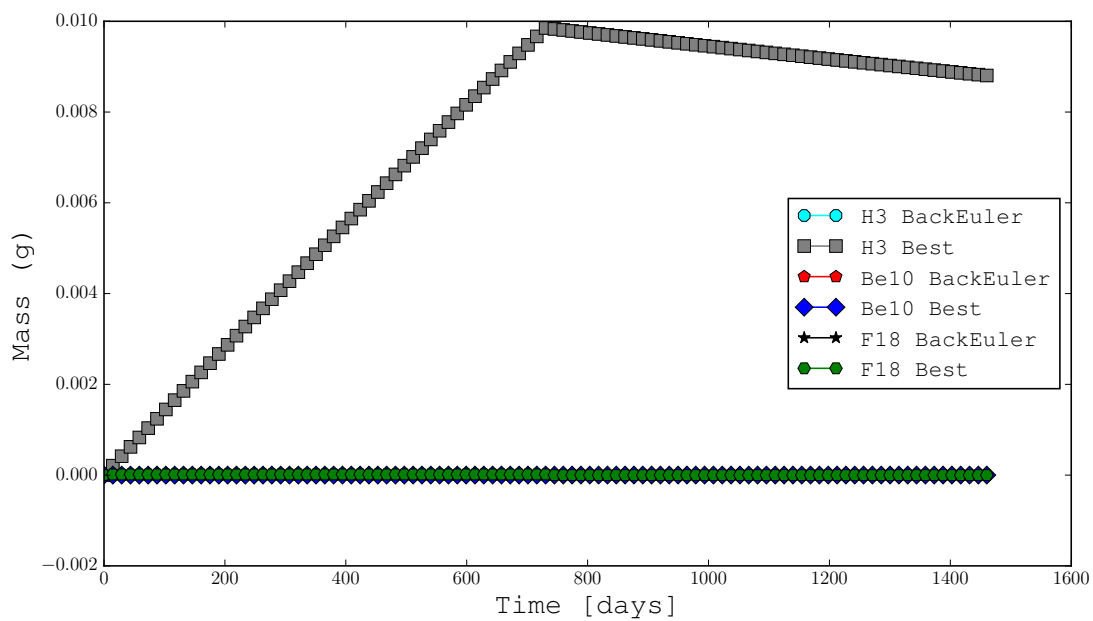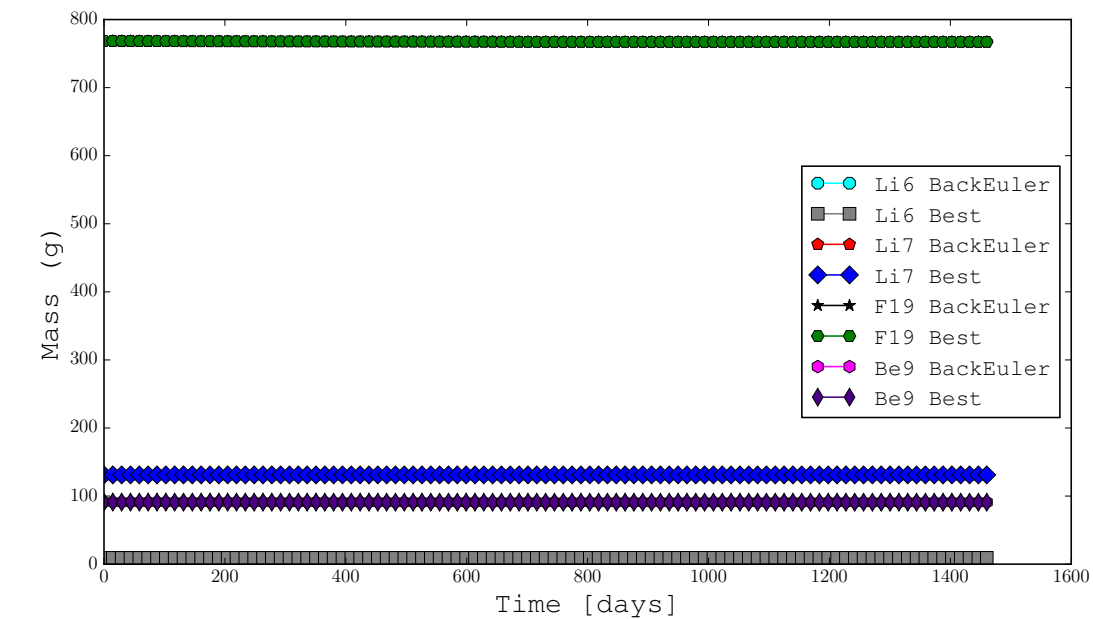
(c) (25 points) After discharging the FLiBe blanket, how long will it take until the material is less radioactive than Brazil Nuts? (444 Bq/kg)

The same solutions as above are depicted below, but in terms of activity.

# Project