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 μ.

(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 Σ_s is the macroscopic scattering cross section. The dependencies for the variables are shown below.

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

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).

$$\begin{aligned}\psi(\vec{x},\hat{\Omega})&=\int_0^\infty dE\;\psi(\vec{x},\hat{\Omega},v(E))\\ \Sigma_t(\vec{x})&=\frac{\int_0^\infty dE\;\Sigma_t(\vec{x},v(E))\psi(\vec{x},\hat{\Omega},v(E))}{\psi(\vec{x},\hat{\Omega})}\\ K(\hat{\Omega}'\cdot\hat{\Omega},v'\to v)&=K(\hat{\Omega}'\cdot\hat{\Omega})K(v'\to v)\\ \Sigma_s(\vec{x})&=\frac{\int_0^\infty dE\int_0^\infty dE'\;\Sigma_s(\vec{x},v(E))K(v'\to v)\psi(\vec{x},\hat{\Omega},v(E))}{\psi(\vec{x},\hat{\Omega})}\\ q(\vec{x},\hat{\Omega})&=\int_0^\infty dE\;q(\vec{x},\hat{\Omega},v(E))\end{aligned}$$

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,

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

Homework 4 Problem Background continued on next page. . .

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 ψ 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,

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

Σ_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 μ_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, ℓ is the iteration index for spacial convergence, j is for a time step and

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

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

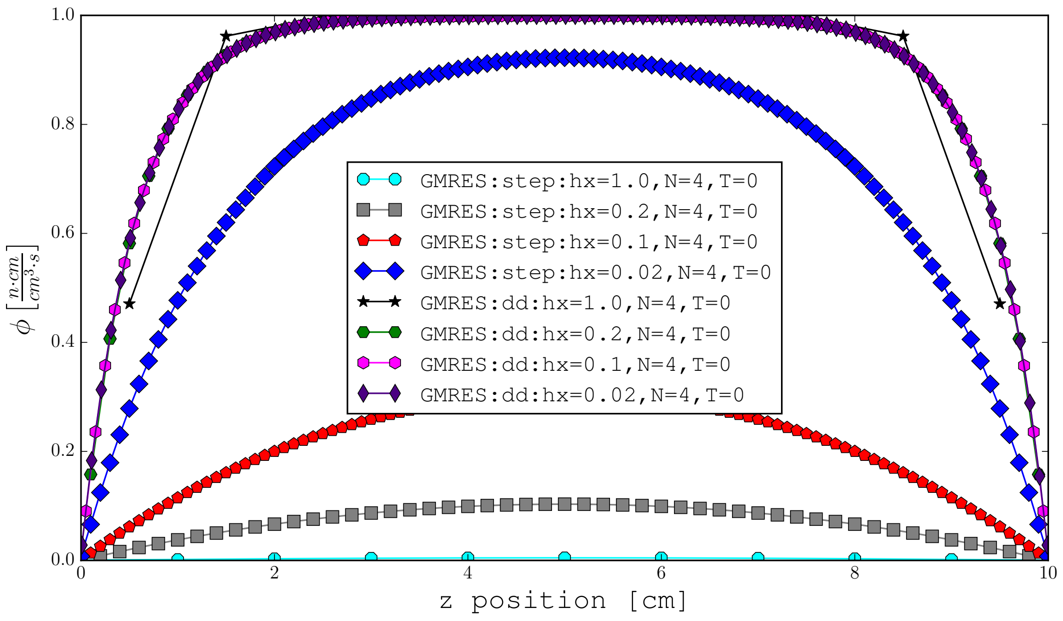
Reeds problem is a similiar system as above, except the source and scattering and total cross sections are variable in z, and the width of z is 16.

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 ($\boldsymbol{\Sigma_t = \Sigma_t \cdot 1.0001}$) 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 ϕ, 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).

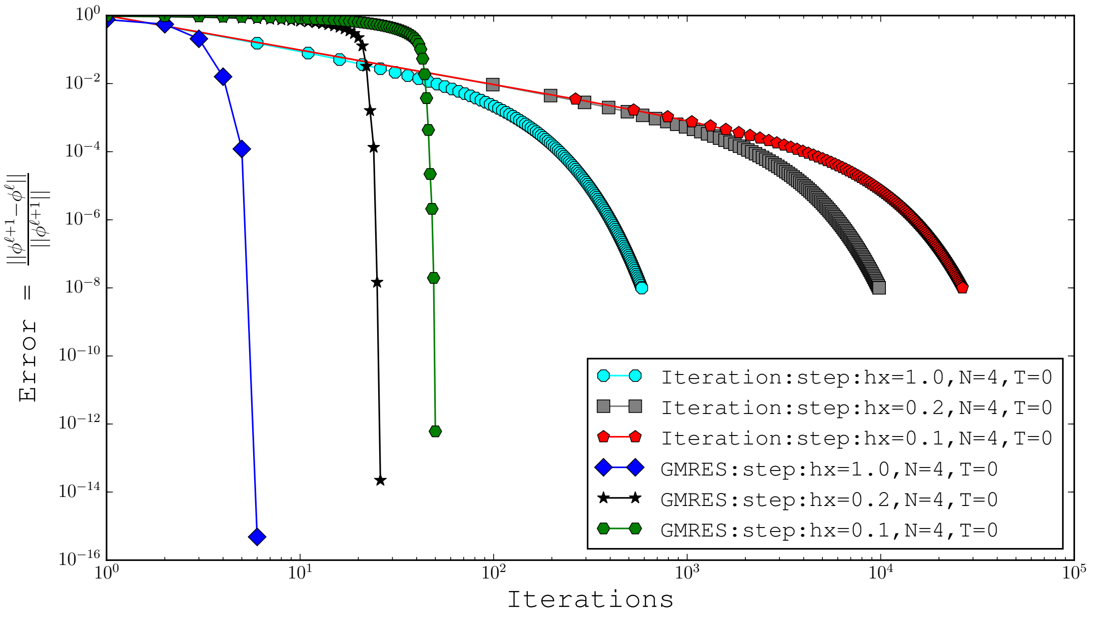
(b) (10 points) Discuss why there is a different form of the discretization for the different signs of μ.

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.

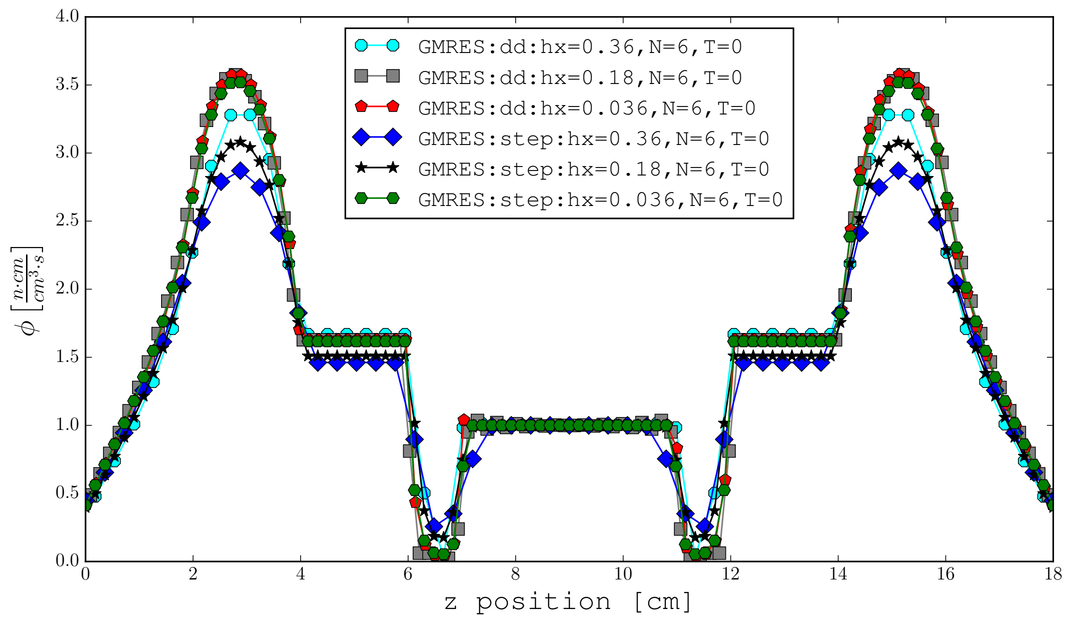
Error will be determined with the following:

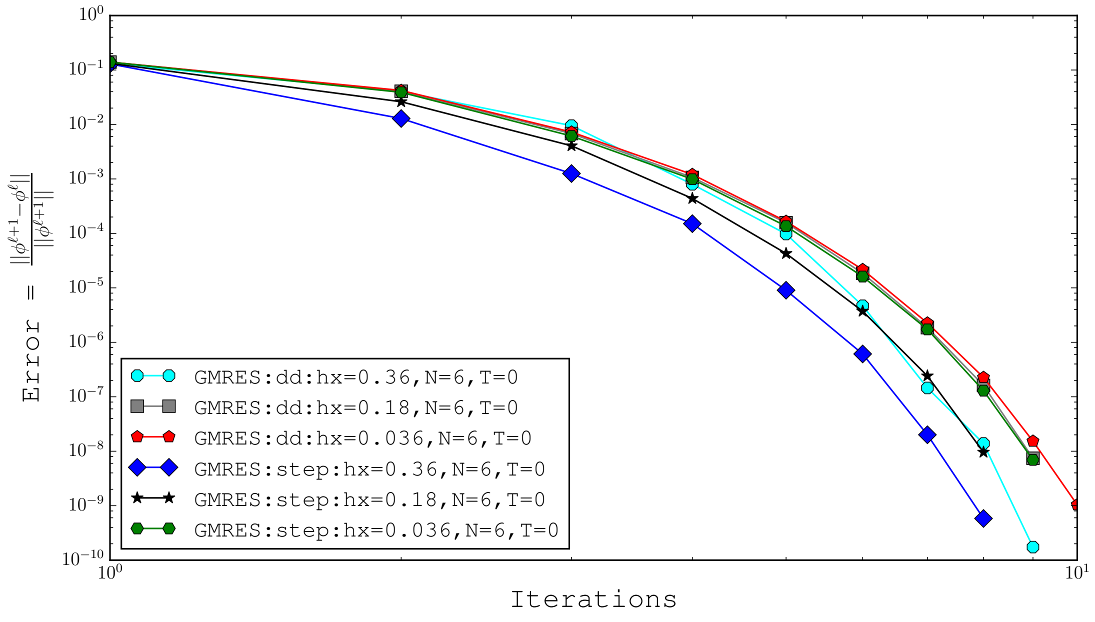
$$\text{Error} = \frac{||\boldsymbol{\phi}^{\ell+1} - \boldsymbol{\phi}^{\ell}||}{||\boldsymbol{\phi}^{\ell+1}||}$$

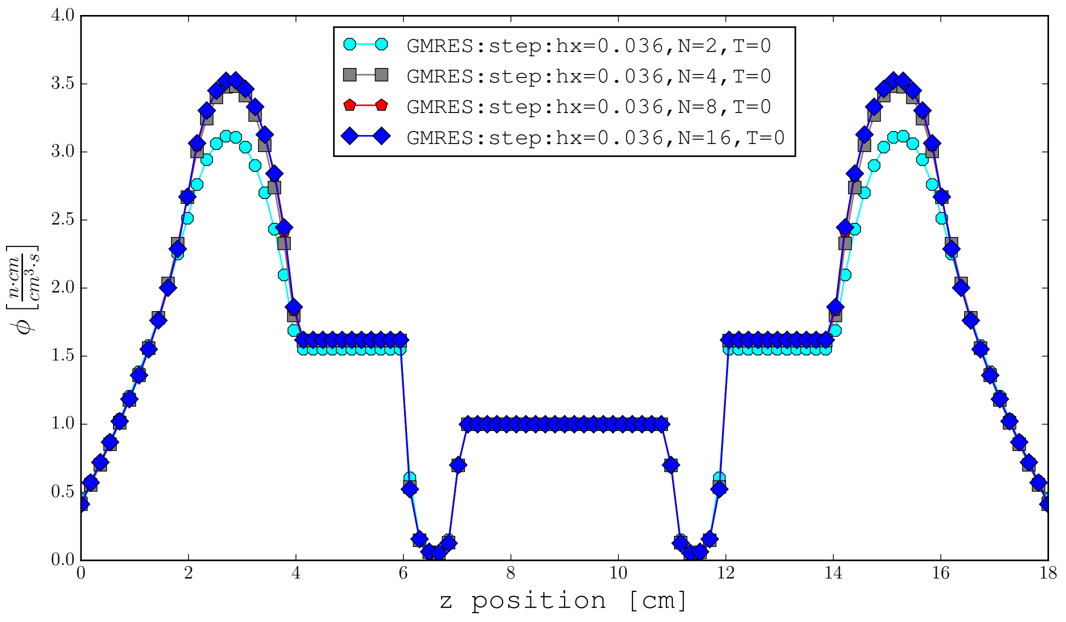


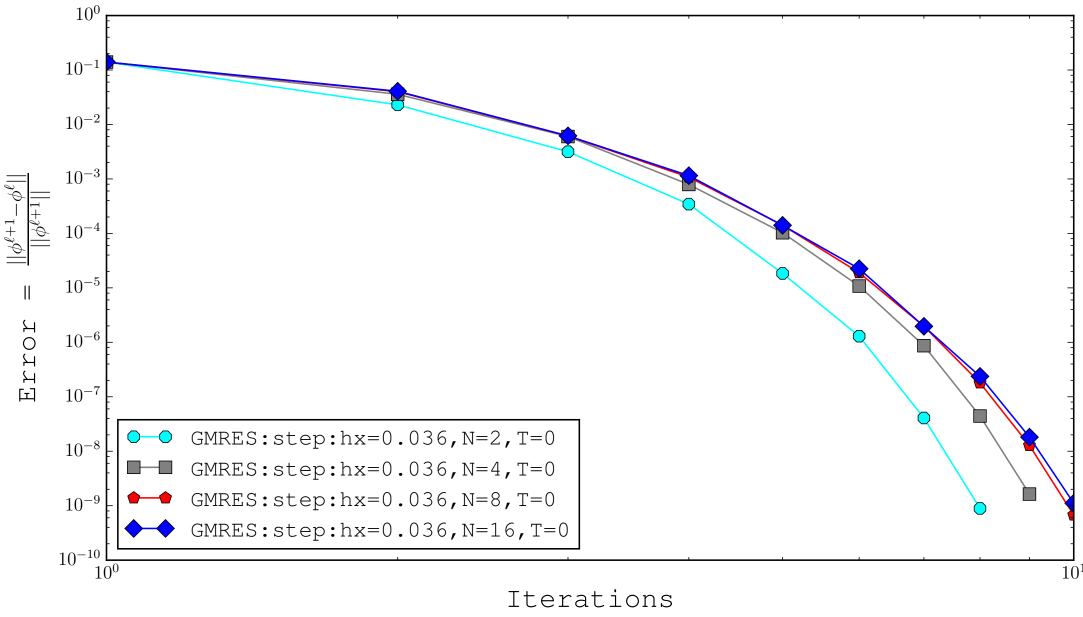
(d) (50 points) Solve Reed's problem (see finite difference diffusion codes). Present convergence plots for the solution in space and angle to a "refined" solution in space and angle.

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




10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{0}
10^{1}
Error = $\frac{||\phi^{\ell+1}-\phi^{\ell}||}{||\phi^{\ell+1}||}$
Iterations
GMRES:dd:hx=0.36,N=6,T=0
GMRES:dd:hx=0.18,N=6,T=0
GMRES:dd:hx=0.036,N=6,T=0
GMRES:step:hx=0.36,N=6,T=0
GMRES:step:hx=0.18,N=6,T=0
GMRES:step:hx=0.036,N=6,T=0


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
2
4
6
8
10
12
14
16
18
$\phi\ \left[\frac{n\cdot cm}{cm^3\cdot s}\right]$
z position [cm]
GMRES:step:hx=0.036,N=2,T=0
GMRES:step:hx=0.036,N=4,T=0
GMRES:step:hx=0.036,N=8,T=0
GMRES:step:hx=0.036,N=16,T=0



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)?

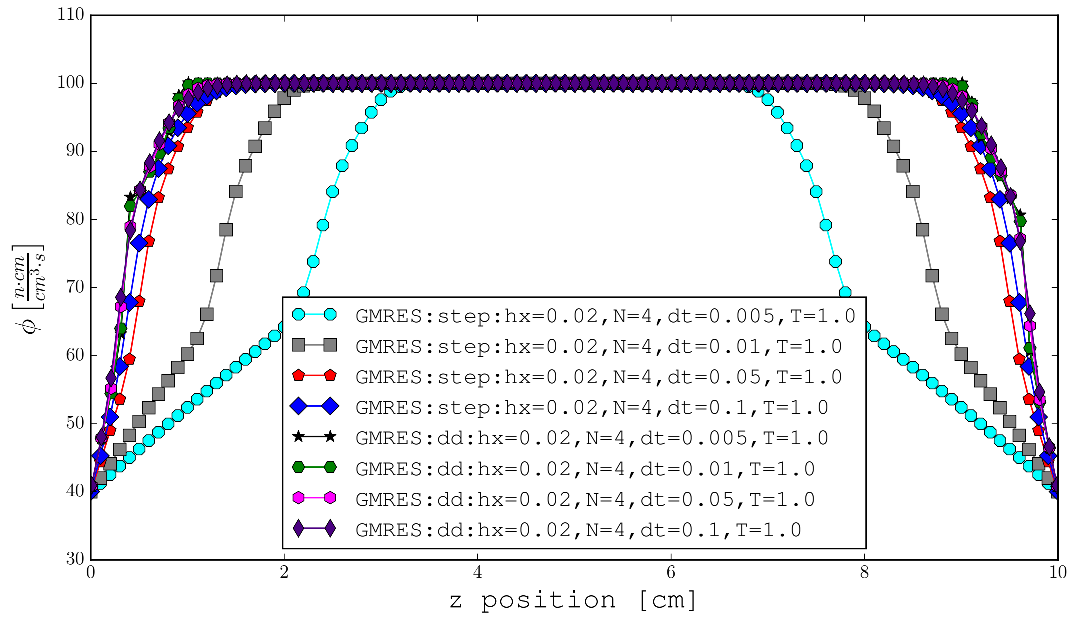


Figure 1: Q=0.01

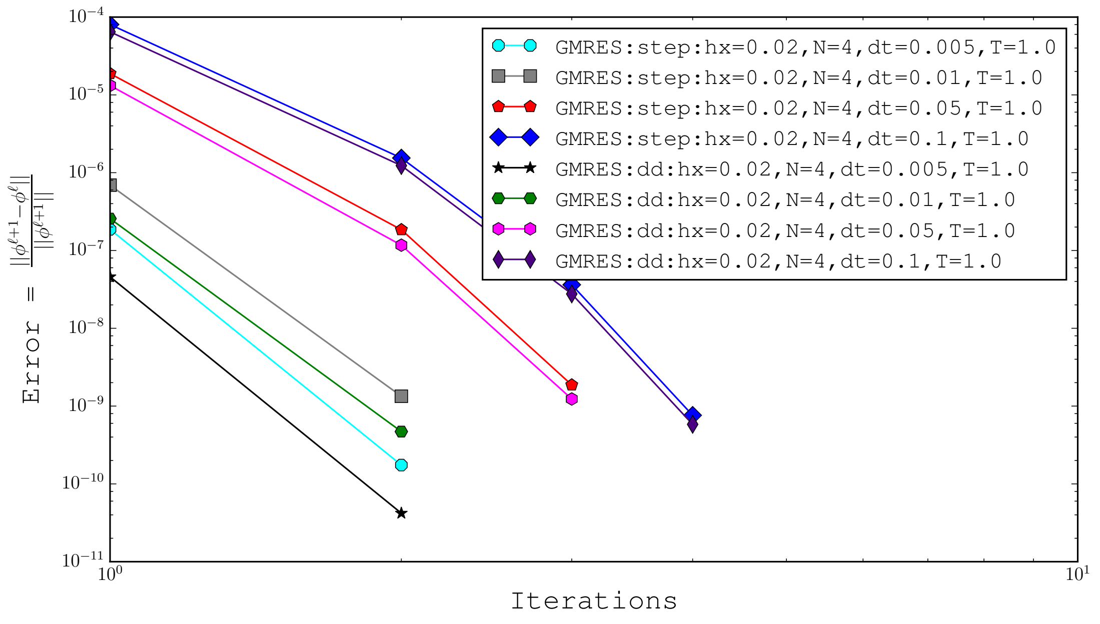
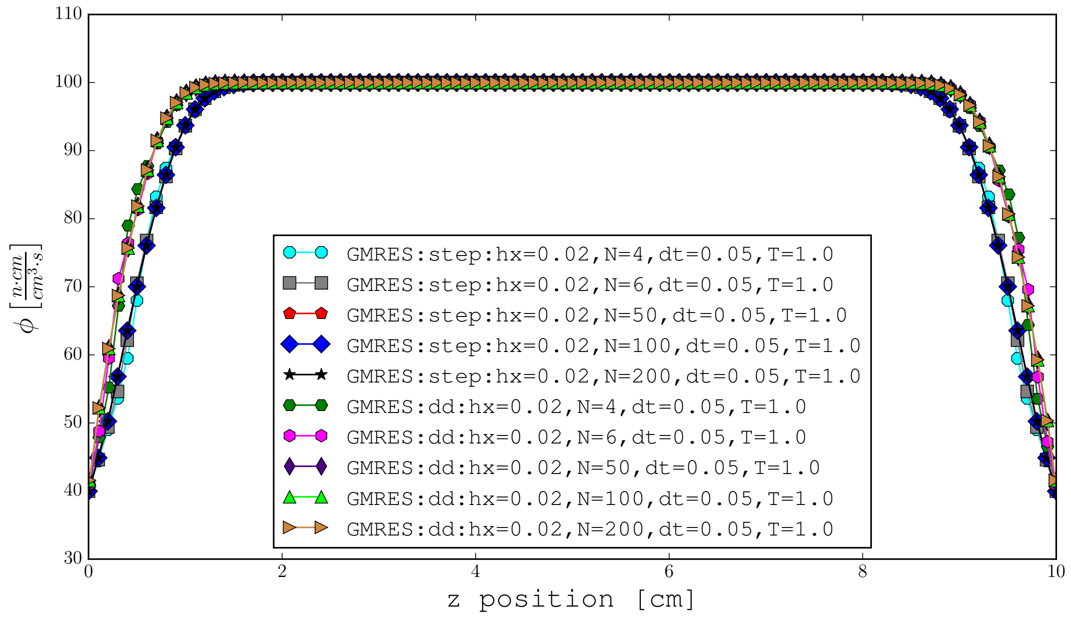
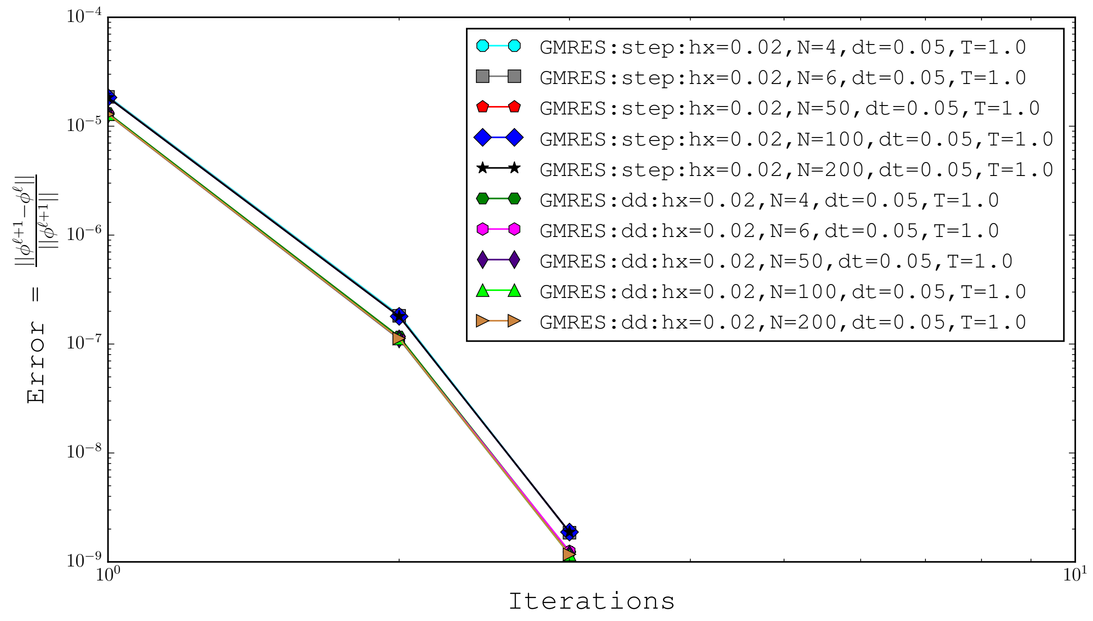


Figure 2: Q=0.01

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**

```
#!/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:
    erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
    erax=erfig.add_subplot(111)              # at T=0


################################################################
#################### Calculations ############################
################################################################

for Scheme in Methods:

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

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

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

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

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

        for t in Time: #Loop over time

            label=label_tmp+str(t)

            #Determine phi (new psi is determined for time steps)
            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 ###################
            #####################################################
            fig
            ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
            if t==0 and PlotError:
                erfig
                erax,erfig=f.plotE(it,er,erax,label,erfig,
                                   Check,NumOfPoints)
            Check=Check+1

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

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

#Why is tmp_psi in the GMRES going negative?

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

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

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

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

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

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

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

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

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

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

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

tol=1e-8

PlotError=True                 # 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:
    erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
    erax=erfig.add_subplot(111)              # at T=0


################################################################
#################### Calculations ############################
################################################################

for Scheme in Methods:

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

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

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

        if Method == 'step':
            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)):
            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])


        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))
                Time=[0]
            else:
                psi=f.np.ones((N,I))*(1/hx)
                Time=f.Timevector(T,dt)

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

            for t in Time: #Loop over time

                label=label_tmp+str(t)

                #Determine phi (new psi is determined for time steps)
                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 ###################
                #####################################################
                fig
                ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
                if t==0 and PlotError:
                    erfig
                    erax,erfig=f.plotE(it,er,erax,label,erfig,
                                       Check,NumOfPoints)
                Check=Check+1

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

fig
f.Legend(ax)
```

Homework 4 Code continued on next page...

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

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

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

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

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

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

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

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

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

slices=[500]            # Number of cuts in slab (looped)
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
v=1                             # Velocity

MAXITS=1000000                  # 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
Ttol=1e-3

PlotError=True               # 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:
    erfig=f.plt.figure(figsize=f.FigureSize) # Err Plot
    erax=erfig.add_subplot(111)              # at T=0


#################################################################
#################### Calculations ###############################
#################################################################

for Scheme in Methods:

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

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

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


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

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

                for t in Time: #Loop over time

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

                    #Determine phi (new psi is determined for time steps)
                    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 ###################
                    ######################################################
                    PlotQuestion=abs(t-dt)<Ttol\
                    or abs(t-0.5)<Ttol or abs(t-1)<Ttol
                    PlotQuestion=abs(t-1)<Ttol
                    if PlotQuestion:
                        fig
                        ax,fig=f.plot(x,phi,ax,label,fig,Check,NumOfPoints)
                        if PlotError:
                            erfig
                            erax,erfig=f.plotE(it,er,erax,label,erfig,
                                                   Check,NumOfPoints)
                        Check=Check+1

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

fig
f.Legend(ax)
#f.plt.savefig('Plots/FluxPlot.pdf')
if PlotError:
    erfig
    f.Legend(erax)
    #f.plt.savefig('Plots/ErrorPlot.pdf')
    f.plt.savefig('Plots/ErrorPlotTimeVaryN.pdf')
    #f.plt.clf()
    f.plt.close()
fig
f.plt.savefig('Plots/FluxPlotTimeVaryN.pdf')
#f.plt.show()

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

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

Listing 4: **Functions holder**

```
#!/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"]

# 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]
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
MarkerType=["8","s","p","D","*","H","h","d","^",">"]

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

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

NumberOfLegendColumns=1

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

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

#################################################################
######################### Functions ###########################
#################################################################

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) )
    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) )
    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;

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

def diamond_sweep1D(I,hx,q,sigma_t,mu,boundary):
  """Compute a transport diamond difference sweep for a given
  Inputs:
    I:                number of zones
    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
  Outputs:
    psi:              value of angular flux in each zone
  """
  assert(np.abs(mu) > 1e-10)
  psi = np.zeros(I)
  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)\
                  /(0.5*sigma_t[i] + mu*ihx)
      psi[i] = 0.5*(psi_right + psi_left)
      psi_left = psi_right
  else:
    psi_right = boundary
    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
  return psi

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

```python
      psi[i] = 0.5*(psi_right + psi_left)
      psi_left = psi_right
  else:
    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)
      psi_right = psi_left
  return psi


def source_iteration(I,hx,q,sigma_t,sigma_s,N,psipreviousstime,
                     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
    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
    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
  Outputs:
    x:                value of center of each zone
    phi:              value of scalar flux in each zone
  """
  iterations = []
  Errors = []
  phi = np.zeros(I)
  phi_old = phi.copy()
  converged = False
  MU, W = np.polynomial.legendre.leggauss(N)
  iteration = 1
  tmp_psi=psipreviousstime.copy()
  if len(Time)==1:
      sigma_ts=sigma_t
  else:
      sigma_ts=sigma_t+1/(v*dt)

  while not(converged):
    phi = np.zeros(I)
    #sweep over each direction
    for n in range(N):
      #qs=(q*W[n])/2+(phi_old*sigma_s)/2+psipreviousstime[n,:]/(v*dt)
      qs=(q)/2+(phi_old*sigma_s)/2+psipreviousstime[n,:]/(v*dt)
      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:
        sys.exit("Sweep method specified not defined in SnMethods")
      phi = phi+tmp_psi[n,:]*W[n]
    #check convergence
    change = np.linalg.norm(phi-phi_old)/np.linalg.norm(phi)
    iterations.append(iteration)
    Errors.append(change)
    #iterations.append(iteration)
    #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):
      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()
  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


def gmres_solve(I,hx,q,sigma_t,sigma_s,N,psipreviousptime,
                v,dt,Time,BCs, sweep_type,
                tolerance = 1.0e-8,maxits = 100, LOUD=False,
                restart = 20 ):
  """Solve, via GMRES, a single-group steady state problem
  Inputs:
    I:               number of zones
    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
    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
  Outputs:
    x:               value of center of each zone
    phi:             value of scalar flux in each zone
  """
  iterations = []
  Errors = []

  #compute RHS side
  RHS = np.zeros(I)
```

```
MU, W = np.polynomial.legendre.leggauss(N)
tmp_psi=psipreviostime.copy()
if len(Time)==1:
    sigma_ts=sigma_t
else:
    sigma_ts=sigma_t+1/(v*dt)

for n in range(N):
  qs=q/2+psipreviostime[n,:]/(v*dt)
  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])
  #tmp_psi = sweep1D(I,hx,q,sigma_t,MU[n],BCs[n])
  RHS += tmp_psi[n,:]*W[n]

#define linear operator for gmres
def linop(phi):
  tmp = phi*0
  #sweep over each direction
  for n in range(N):
    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,
                                    sigma_ts,MU[n],BCs[n])
    tmp += tmp_psi[n,:]*W[n]
  return phi-tmp
A = spla.LinearOperator((I,I), matvec = linop, dtype='d')


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

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

#Print important information
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':
    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
    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)
        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:
            sys.exit("Sweep method specified not defined in SnMethods")
        phi2 = phi2+tmp_psi[n,:]*W[n]

    return x, phi, iterations, Errors,tmp_psi

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,
            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)
    else:
        print("Improper sweep selected")
        quit()
    return x, phi, iterations, errors,psi

#################################################################
################### Plotting Function ##########################
#################################################################

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

def loop_values(list1,index):
    """
    This function will loop through values in list even if
    outside range (in the positive sense not negative)
    """
    while True:
        try:
            list1[index]
            break
        except IndexError:
            index=index-len(list1)
    return(list1[index])

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

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


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

    #Log or linear scale?
    erax.set_xscale(XScaleE)
    erax.set_yscale(YScaleE)
    #Set Title
    erfig.suptitle(Title,fontsize=TitleFontSize,
                 fontweight=TitleFontWeight,fontdict=font,
                                                         ha='center')
    #Set X and y labels
    erax.set_xlabel(XlabelE,
                  fontsize=XFontSize,fontweight=XFontWeight,
                  fontdict=font)
    erax.set_ylabel(YlabelE,
                  fontsize=YFontSize,
                  fontweight=YFontWeight,
                  fontdict=font)
    return(erax,erfig)

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

# def Legend(ax):
#         handles,labels=ax.get_legend_handles_labels()
#         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),
#                   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 Solution

(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)

Project