CHAPTER

18 One-Group Diffusion Equation

OUTLINE

There is no real direction here, neither lines of power nor cooperation.

–*Thomas Pynchon*, Gravity's Rainbow

CHAPTER POINTS

- Boundary value problems specify the value of the solution to the differential equations at multiple points.
- We develop a method for solving the diffusion equation for neutrons in slab, spherical, and cylindrical geometries.
- Time-dependent problems can be written as the solution to a succession of modified steady state problems.

In this lecture we will solve the one-dimensional (1-D), one-group neutron diffusion equation. This is an example of a boundary-value problem. A boundary value problem is a differential

DOI: 10.1016/B978-0-12-812253-2.00020-0

equation or a system of differential equations that specifies the value of the solution or its derivatives at more than one point.

In the previous chapter we dealt with initial value problems where the conditions for the solution were prescribed at a single point that we called $t = 0$. In boundary value problems we specify the solution at multiple points as we shall see. This means we cannot start at one point and move the solution in a particular direction. Rather, we must solve a system of equations to find the solution that satisfies the boundary values.

The general neutron diffusion equation for the scalar flux of neutrons, $\phi(r,t)$, is given by

$$\frac{1}{v}\frac{\partial \phi}{\partial t} - \nabla \cdot D(r)\nabla\phi(r,t) + \Sigma_{\mathrm{a}}(r)\phi(r,t) = \nu\Sigma_{\mathrm{f}}(r)\phi(r,t) + Q(r,t).$$

Our notation is standard: v is the neutron speed, $D(r)$ is the diffusion coefficient, Σ_a is the macroscopic absorption cross-section, Σ_f is the macroscopic fission cross section, ν is the number of neutrons per fission, and $Q(r,t)$ is a prescribed source. The boundary conditions we will consider for this equation are generic conditions,

$$\mathcal{A}(r)\phi(r,t) + \mathcal{B}(r)\frac{d\phi}{dr} = \mathcal{C}(r) \qquad \text{for } r \in \partial V.$$

The initial condition is

$$\phi(r,0) = \phi^0(r).$$

We can eliminate the time variable from this equation by integrating over a time step from $t^n = n\Delta t$ to $t^{n+1} = (n+1)\Delta t$:

$$\frac{1}{v}\left(\phi^{n+1}(r) - \phi^n(r)\right) - \int_{t^n}^{t^{n+1}} dt\ [\nabla \cdot D(r)\nabla\phi(r,t) + \Sigma_{\mathrm{a}}(r)\phi(r,t)] = \int_{t^n}^{t^{n+1}} dt\ [\nu\Sigma_{\mathrm{f}}(r)\phi(r,t) + Q(r,t)], \tag{18.1}$$

where $\phi^n(r) = \phi(r, n\Delta t)$.

Using the backward Euler approach to the integrals (that is the right-hand rectangle rule) we get

$$\frac{1}{v\Delta t}\left(\phi^{n+1}(r) - \phi^n(r)\right) - \nabla \cdot D(r)\nabla\phi^{n+1}(r) + \Sigma_{\mathrm{a}}(r)\phi^{n+1}(r) = \nu\Sigma_{\mathrm{f}}(r)\phi^{n+1}(r) + Q^{n+1}(r).$$

The next step is to define a new absorption cross-section and source as

$$\Sigma_{\mathrm{a}}^{*} = \Sigma_{\mathrm{a}}(r) + \frac{1}{v\Delta t}, \qquad Q^{n+1,*}(r) = Q^{n+1}(r) + \frac{1}{v\Delta t}\phi^n(r).$$

With these definitions we get the following equation

$$-\nabla \cdot D(r)\nabla\phi^{n+1}(r) + \Sigma_{\mathrm{a}}^{*}(r)\phi^{n+1}(r) = \nu\Sigma_{\mathrm{f}}(r)\phi^{n+1}(r) + Q^{n+1,*}(r).$$

This is a steady-state diffusion equation for the scalar flux at time $n+1$. Therefore, we can focus our efforts on solving steady-state problems, knowing that to solve time-dependent equations, we just have to redefine the source and absorption cross-sections appropriately.

18.1 DISCRETIZING THE STEADY-STATE DIFFUSION EQUATION

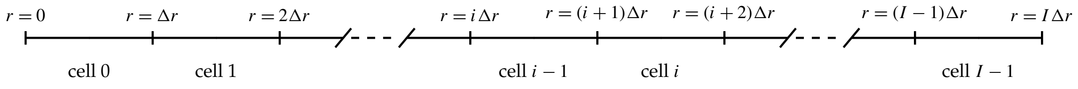
We begin with the steady-state diffusion equation with a reflecting boundary condition at $r=0$ and a general boundary condition at $r=R$:

$$-\nabla \cdot D(r)\nabla\phi(r) + \Sigma_\mathrm{a}(r)\phi(r) = \nu\Sigma_\mathrm{f}(r)\phi(r) + Q(r),$$

$$\left.\frac{d\phi}{dr}\right|_{r=0} = 0,$$

$$\mathcal{A}(R)\phi(R) + \mathcal{B}(R)\left.\frac{d\phi}{dr}\right|_{r=R} = \mathcal{C}(R).$$

We seek to solve this numerically on a grid of spatial cells



In our notation there will be I cells and $I+1$ edges. The cell centers are given by

$$r_i = i\Delta r + \frac{\Delta r}{2}, \qquad i = 0, 1, \ldots, I-1,$$

and the left and right edges are given by the formulas

$$r_{i-1/2} = i\Delta r, \qquad r_{i+1/2} = (i+1)\Delta r, \qquad i = 0, 1, \ldots, I-1.$$

We can construct a Python function that creates the cell edges and cell centers given R and I.

```
In [1]: def create_grid(R,I):
            """Create the cell edges and centers for a
            domain of size R and I cells
            Args:
                R: size of domain
                I: number of cells

            Returns:
                Delta_r: the width of each cell
                centers: the cell centers of the grid
                edges: the cell edges of the grid
```

```
        """
        Delta_r = float(R)/I
        centers = np.arange(I)*Delta_r + 0.5*Delta_r
        edges = np.arange(I+1)*Delta_r
        return Delta_r, centers, edges
```

Our problem will be set up such that inside each cell the material properties are constant so that inside cell i,

$$D(r) = D_i, \quad \Sigma_\mathrm{a}(r) = \Sigma_{\mathrm{a},i}, \quad \Sigma_\mathrm{f}(r) = \Sigma_{\mathrm{f},i}, \quad Q(r) = Q_i, \qquad r \in (r_{i-1/2}, r_{i+1/2}).$$

18.1.1 The Diffusion Operator in Different Geometries

The definition of $\nabla \cdot D(r)\nabla$ in several geometries is given below:

$$\nabla \cdot D(r)\nabla = \begin{cases} \frac{d}{dr} D(r) \frac{d}{dr} & \text{1-D Slab} \\ \frac{1}{r^2}\frac{d}{dr} r^2 D(r)\frac{d}{dr} & \text{1-D Sphere} \\ \frac{1}{r}\frac{d}{dr} r D(r) \frac{d}{dr} & \text{1-D Cylinder} \end{cases}.$$

Also, the differential volume element in each geometry is

$$dV = \begin{cases} dr & \text{1-D Slab} \\ 4\pi r^2 dr & \text{1-D Sphere} \\ 2\pi r dr & \text{1-D Cylinder} \end{cases}.$$

We define ϕ_i as the average value of the scalar flux in cell i. This quantity is given by

$$\phi_i = \frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \phi(r),$$

where V_i is the cell volume

$$V_i = \begin{cases} \Delta r & \text{1-D Slab} \\ \frac{4}{3}\pi(r_{i+1/2}^3 - r_{i-1/2}^3) & \text{1-D Sphere} \\ \pi(r_{i+1/2}^2 - r_{i-1/2}^2) & \text{1-D Cylinder} \end{cases}.$$

To develop the discrete equations we will integrate the diffusion equation term by term. We first integrate the absorption term in the diffusion equation over cell i and divide by V_i,

$$\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \Sigma_\mathrm{a}(r)\phi(r) = \frac{\Sigma_{\mathrm{a},i}}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \phi(r) = \Sigma_{\mathrm{a},i}\phi_i.$$

The fission term is handled in a similar manner

$$\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \nu\Sigma_\mathrm{f}(r)\phi(r) = \nu\Sigma_{\mathrm{f},i}\phi_i.$$

The source term is similarly straightforward because $Q(r)$ is constant in cell i,

$$\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, Q(r) = Q_i.$$

The diffusion term is a bit trickier,

$$-\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \nabla\cdot D(r)\nabla\phi(r) = \begin{cases} -\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dr\, \frac{d}{dr} D(r)\phi(r)\frac{d}{dr} & \text{1-D Slab} \\ -\frac{4\pi}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dr\, \frac{d}{dr} r^2 D(r)\frac{d}{dr}\phi(r) & \text{1-D Sphere} \\ -\frac{2\pi}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dr\, \frac{d}{dr} r D(r)\frac{d}{dr}\phi(r) & \text{1-D Cylinder} \end{cases}.$$

To simplify these we can use the fundamental theorem of calculus to get

$$-\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \nabla\cdot D(r)\nabla\phi(r) = \begin{cases} -\frac{1}{V_i}\left[D_{i+1/2}\frac{d}{dr}\phi(r_{i+1/2}) - D_{i-1/2}\frac{d}{dr}\phi(r_{i-1/2})\right] \\ \text{1-D Slab} \\ -\frac{4\pi}{V_i}\left[D_{i+1/2}r_{i+1/2}^2\frac{d}{dr}\phi(r_{i+1/2}) - D_{i-1/2}r_{i-1/2}^2\frac{d}{dr}\phi(r_{i-1/2})\right] \\ \text{1-D Sphere} \\ -\frac{2\pi}{V_i}\left[D_{i+1/2}r_{i+1/2}\frac{d}{dr}\phi(r_{i+1/2}) - D_{i-1/2}r_{i-1/2}\frac{d}{dr}\phi(r_{i-1/2})\right] \\ \text{1-D Cylinder} \end{cases}. \tag{18.2}$$

Defining the surface area, $S_{i\pm1/2}$ at the cell edge for each geometry will allow us to simplify Eq. (18.2) further:

$$S_{i\pm1/2} = \begin{cases} 1 & \text{1-D Slab} \\ 4\pi r_{i\pm1/2}^2 & \text{1-D Sphere} \\ 2\pi r_{i\pm1/2} & \text{1-D Cylinder} \end{cases}.$$

Using this definition in Eq. (18.2) we get

$$-\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\, \nabla\cdot D(r)\nabla\phi(r) = -\frac{1}{V_i}\left[D_{i+1/2}S_{i+1/2}\frac{d}{dr}\phi(r_{i+1/2}) - D_{i-1/2}S_{i-1/2}\frac{d}{dr}\phi(r_{i-1/2})\right].$$

The final piece of the derivation is to write the value of the derivative at the cell edge using the central difference formula. To do this we will have to interpret the cell average scalar flux as the value of the scalar flux at the cell center:

$$\phi_i \approx \phi(r_i).$$

With this definition we can write

$$\frac{d}{dr}\phi(r_{i+1/2}) = \frac{\phi_{i+1}-\phi_i}{\Delta r} + O(\Delta r^2).$$

Therefore, the fully discrete diffusion term is

$$-\frac{1}{V_i}\int_{r_{i-1/2}}^{r_{i+1/2}} dV\,\nabla^2\phi(r) = -\frac{1}{V_i}\left[D_{i+1/2}S_{i+1/2}\frac{\phi_{i+1}-\phi_i}{\Delta r} - D_{i-1/2}S_{i-1/2}\frac{\phi_i-\phi_{i-1}}{\Delta r}\right].$$

Putting all of this together gives us the diffusion equation integrated over a cell volume:

$$-\frac{1}{V_i}\left[D_{i+1/2}S_{i+1/2}\frac{\phi_{i+1}-\phi_i}{\Delta r} - D_{i-1/2}S_{i-1/2}\frac{\phi_i-\phi_{i-1}}{\Delta r}\right] + \left(\Sigma_{\mathrm{a},i} - \nu\Sigma_{\mathrm{f},i}\right)\phi_i = Q_i. \qquad (18.3)$$

18.1.2 Interface Diffusion Coefficient

We need to define what we mean by $D_{i+1/2}$ and $D_{i-1/2}$ because the diffusion coefficient is only constant inside a cell. For these terms will define a diffusion coefficient so that the neutron current is continuous at a cell face. In particular we define $\phi_{i+1/2}$ so that

$$2D_{i+1}\frac{\phi_{i+1}-\phi_{i+1/2}}{\Delta r} = D_{i+1/2}\frac{\phi_{i+1}-\phi_i}{\Delta r},$$

and

$$2D_i\frac{\phi_{i+1/2}-\phi_i}{\Delta r} = D_{i+1/2}\frac{\phi_{i+1}-\phi_i}{\Delta r}.$$

These two equations state that we want to define $D_{i+1/2}$ and $\phi_{i+1/2}$ so that the current is the same if we calculate it from the left or the right. We also get a similar system defining $D_{i-1/2}$. Solving these equations for $D_{i+1/2}$ and $D_{i-1/2}$ we get

$$D_{i\pm1/2} = \frac{2D_iD_{i\pm1}}{D_i + D_{i\pm1}}. \qquad (18.4)$$

This definition for the diffusion coefficient at the interface takes the harmonic mean of the diffusion coefficient on each side and assures that the neutron current density is continuous at the interface.

18.1.3 Boundary Conditions

We need to enforce our general boundary condition at $r = R$, that is

$$\mathcal{A}(R)\phi(R) + \mathcal{B}(R)\left.\frac{d\phi}{dr}\right|_{r=R} = \mathcal{C}(R).$$

To see how this is going to come into the equation, we examine at the equation for $i = I-1$,

$$-\frac{1}{V_I}\left[D_{I-1/2}S_{I-1/2}\frac{\phi_I-\phi_{I-1}}{\Delta r} - D_{I-3/2}S_{I-3/2}\frac{\phi_{I-1}-\phi_{I-2}}{\Delta r}\right] + \left(\Sigma_{\mathrm{a},I-1} - \nu\Sigma_{\mathrm{f},I-1}\right)\phi_I = Q_{I-1}.$$

Notice that this equation has ϕ_I, which is the undefined value of the scalar flux outside the domain. We need to create a value for this parameter that is consistent with the boundary condition. To do this we will make the following approximations:

$$\phi(R) \approx \frac{1}{2}(\phi_{I-1} + \phi_I),$$

and

$$\left.\frac{d\phi}{dr}\right|_{r=R} \approx \frac{\phi_I - \phi_{I-1}}{\Delta r}.$$

Using these in our boundary condition, we get the final equation that we need

$$\left(\frac{\mathcal{A}}{2} - \frac{\mathcal{B}}{\Delta r}\right)\phi_{I-1} + \left(\frac{\mathcal{A}}{2} + \frac{\mathcal{B}}{\Delta r}\right)\phi_I = \mathcal{C}.$$

Here we have dropped the spatial dependence of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ in the boundary condition.

18.1.3.1 Types of Boundary Conditions on the Outer Surface

Using the general boundary condition, we can enforce a variety of boundary conditions [22]. A Dirichlet boundary condition of the form

$$\phi(R) = c,$$

can be enforced by setting $\mathcal{A} = 1$, $\mathcal{B} = 0$, and $\mathcal{C} = c$.

An albedo boundary condition is used for the case where some fraction of the neutrons that leave the system are reflected back. If we call this fraction that is reflected α, the albedo boundary condition can be written as

$$\frac{(1-\alpha)}{4(1+\alpha)}\phi(R) + \frac{D}{2}\left.\frac{d\phi}{dr}\right|_{r=R} = 0,$$

where the diffusion coefficient is evaluated at $r = R$. This boundary condition is obtained by setting $\mathcal{C} = 0$, and

$$\mathcal{A} = \frac{(1-\alpha)}{4(1+\alpha)}, \qquad \mathcal{B} = \frac{D}{2}.$$

The reflecting boundary is a special case of the albedo condition with $\alpha = 1$. For this boundary condition we set $\mathcal{A} = \mathcal{C} = 0$ and $\mathcal{B} = 1$.

The final boundary condition that we consider is the partial current boundary condition, also called a Marshak boundary condition. These allow us to specify the amount of neutrons that enter the system from the edge, rather than just specifying the scalar flux on the boundary as in the Dirichlet condition. At $R = r$ the rate at which neutrons enter the domain per unit area, (i.e., the partial current), is given by

$$\text{incoming partial current} = \frac{1}{4}\phi(R) + \frac{D}{2}\left.\frac{d\phi}{dr}\right|_{r=R}.$$

Therefore, if we wish to set the incoming partial current into the system at a particular value, J_{in}, we set our boundary constants to be

$$\mathcal{A} = \frac{1}{4}, \qquad \mathcal{B} = \frac{D}{2}, \qquad \mathcal{C} = J_{\text{in}}.$$

A vacuum boundary condition can be obtained by setting $J_{\text{in}} = 0$.

18.1.3.2 Reflecting Boundary Condition at r = 0

At the $r = 0$ boundary we require a reflecting boundary in the curvilinear geometries (spherical and cylindrical), and we want to specify the same for the slab. A reflecting boundary has

$$\frac{d}{dr}\phi(0) = 0.$$

It turns out that this is automatically enforced in the curvilinear geometries because $S_{-1/2} = 0$, and the derivative at the inner edge of cell 0 is effectively zero. To enforce this in slab geometry we just need to force $S_{-1/2} = 0$ to make the equation for $i = 0$

$$-\frac{D_{1/2}S_{1/2}}{V_0}\frac{\phi_1 - \phi_0}{\Delta r} + \left(\Sigma_{\text{a},0} - \nu\Sigma_{\text{f},0}\right)\phi_0 = Q_0.$$

18.2 PYTHON CODE FOR THE DIFFUSION EQUATION

We have now completely specified the discrete equations for our diffusion problem. This section will detail how to build the matrices and vectors.

There are $I + 1$ equations in our system:

$$-\frac{1}{V_i}\left[D_{i+1/2}S_{i+1/2}\frac{\phi_{i+1} - \phi_i}{\Delta r} - D_{i-1/2}S_{i-1/2}\frac{\phi_i - \phi_{i-1}}{\Delta r}\right] + \left(\Sigma_{\text{a},i} - \nu\Sigma_{\text{f},i}\right)\phi_i = Q_i,$$
$$i = 0, \ldots, I-1,$$

and

$$\left(\frac{\mathcal{A}}{2} - \frac{\mathcal{B}}{\Delta r}\right)\phi_{I-1} + \left(\frac{\mathcal{A}}{2} + \frac{\mathcal{B}}{\Delta r}\right)\phi_I = \mathcal{C}.$$

This is a system of equations to solve, to do this we first define our solution vector and righthand side

$$\boldsymbol{\phi} = \begin{pmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_I \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} Q_0 \\ Q_1 \\ \vdots \\ \mathcal{C} \end{pmatrix}.$$

Our system will be written as

$$\mathbf{A}\boldsymbol{\phi} = \mathbf{b}.$$

To define **A**, we will factor our equations to be

$$-\frac{D_{i+1/2}S_{i+1/2}}{\Delta r V_i}\phi_{i+1} + \left[\frac{1}{\Delta r V_i}\left(D_{i+1/2}S_{i+1/2} + D_{i-1/2}S_{i-1/2}\right) + \Sigma_{\mathrm{a},i} - \nu\Sigma_{\mathrm{f},i}\right]\phi_i$$
$$- \frac{D_{i-1/2}S_{i-1/2}}{\Delta r V_i}\phi_{i-1} = Q_i, \qquad i = 0, \ldots, I-1,$$

and

$$\left(\frac{\mathcal{A}}{2} - \frac{\mathcal{B}}{\Delta r}\right)\phi_{I-1} + \left(\frac{\mathcal{A}}{2} + \frac{\mathcal{B}}{\Delta r}\right)\phi_I = \mathcal{C}.$$

From these equations we get that the element of **A** in row i and column j is

$$A_{ij} = \begin{cases} \left[\frac{1}{\Delta r V_i}\left(D_{i+1/2}S_{i+1/2} + D_{i-1/2}S_{i-1/2}\right) + \Sigma_{\mathrm{a},i} - \nu\Sigma_{\mathrm{f},i}\right] & i = j \text{ and } i = 0, 1, \ldots I-1 \\ -\frac{1}{\Delta r V_i} D_{i+1/2}S_{i+1/2} & i+1 = j \text{ and } i = 0, 1, \ldots I-2 \\ -\frac{1}{\Delta r V_i} D_{i-1/2}S_{i-1/2} & i-1 = j \text{ and } i = 1, 2, \ldots I-1 \\ \left(\frac{\mathcal{A}}{2} - \frac{\mathcal{B}}{\Delta r}\right) & j = I-1 \text{ and } i = I \\ \left(\frac{\mathcal{A}}{2} + \frac{\mathcal{B}}{\Delta r}\right) & j = I \text{ and } i = I \\ 0 & \text{otherwise} \end{cases}.$$

We will now set up a function that

- builds the matrix **A**,
- builds the vector **b**,
- uses Gauss elimination to solve the system for the scalar fluxes $\boldsymbol{\phi}$.

The code will call the grid function that we defined before. Additionally, our function will take as arguments the name of a function that defines each of the material properties.

```
In [2]: def DiffusionSolver(R,I,D,Sig_a,nuSig_f, Q,BC, geometry):
            """Solve the neutron diffusion equation in a 1-D geometry
            using cell-averaged unknowns
            Args:
                R: size of domain
                I: number of cells
                D: name of function that returns diffusion coefficient for a given r
                Sig_a: name of function that returns Sigma_a for a given r
                nuSig_f: name of function that returns nu Sigma_f for a given r
                Q: name of function that returns Q for a given r
                BC: Boundary Condition at r=R in form [A,B,C]
                geometry: shape of problem 0 for slab
                        1 for cylindrical
                        2 for spherical
```

```
    Returns:
        centers: the cell centers of the grid
        phi:  cell-average value of the scalar flux

    """
    #create the grid
    Delta_r, centers, edges = create_grid(R,I)
    A = np.zeros((I+1,I+1))
    b = np.zeros(I+1)
    #define surface areas and volumes
    assert( (geometry==0) or (geometry == 1) or (geometry == 2))
    if (geometry == 0):
        #in slab it's 1 everywhere except at the left edge
        S = 0.0*edges+1
        S[0] = 0.0 #this will enforce reflecting BC
        #in slab its dr
        V = 0.0*centers + Delta_r
    elif (geometry == 1):
        #in cylinder it is 2 pi r
        S = 2.0*np.pi*edges
        #in cylinder its pi (r^2 - r^2)
        V = np.pi*( edges[1:(I+1)]**2
                   - edges[0:I]**2 )
    elif (geometry == 2):
        #in sphere it is 4 pi r^2
        S = 4.0*np.pi*edges**2
        #in sphere its 4/3 pi (r^3 - r^3)
        V = 4.0/3.0*np.pi*( edges[1:(I+1)]**3
                   - edges[0:I]**3 )

    #Set up BC at R
    A[I,I] = (BC[0]*0.5 + BC[1]/Delta_r)
    A[I,I-1] = (BC[0]*0.5 - BC[1]/Delta_r)
    b[I] = BC[2]
    r = centers[0]
    DPlus = 0
    #fill in rest of matrix
    for i in range(I):
        r = centers[i]
        DMinus = DPlus
        DPlus = 2*(D(r)*D(r+Delta_r))/(D(r)+D(r+Delta_r))
        A[i,i] = (1.0/(Delta_r * V[i])*DPlus*S[i+1] +
                  Sig_a(r) - nuSig_f(r))
        if (i>0):
            A[i,i-1] = -1.0*DMinus/(Delta_r * V[i])*S[i]
            A[i,i] += 1.0/(Delta_r * V[i])*(DMinus*S[i])
        A[i,i+1] = -DPlus/(Delta_r * V[i])*S[i+1]
        b[i] = Q(r)

    #solve system
    phi = GaussElimPivotSolve(A,b)
    #remove last element of phi because it is outside the domain
    phi = phi[0:I]
    return centers, phi
```

18.3 A TEST PROBLEM FOR EACH GEOMETRY

Before we proceed, recall that the steady state diffusion equation will only have a solution for subcritical problems with a source. If the system is critical or supercritical, there is no finite steady state solution. We will discuss in future lectures how to solve these problems.

In an infinite, homogeneous medium all the spatial derivatives go to zero and the diffusion equation reads

$$\phi(r) = \frac{Q}{\Sigma_\mathrm{a} - \nu\Sigma_\mathrm{f}}.$$

Therefore, we expect that if we make our problem have a reflecting boundary condition it should reproduce this infinite medium solution.

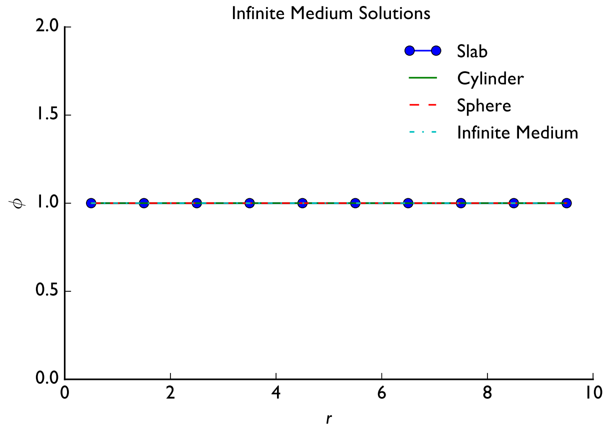
To try this out we define functions for each of D, Σ_a, $\nu\Sigma_\mathrm{f}$, and Q:

```
In [3]: #in this case all three are constant
        def D(r):
            return 0.04;
        def Sigma_a(r):
            return 1;
        def nuSigma_f(r):
            return 0;
        def Q(r):
            return 1
        print("For this problem the diffusion length is", np.sqrt(D(1)/Sigma_a(1)))
        inf_med = Q(1)/(Sigma_a(1) - nuSigma_f(1))
        print("The infinite medium solution is",inf_med)
```

```
For this problem the diffusion length is 0.2
The infinite medium solution is 1.0
```

To compute the solution we call our `DiffusionSolver` function. In this case we set $R = 10$ and $I = 10$; this makes $\Delta r = 1$. We set the boundary condition parameter to be `[0,1,0]` to make it correspond to the reflecting boundary condition discussed above.

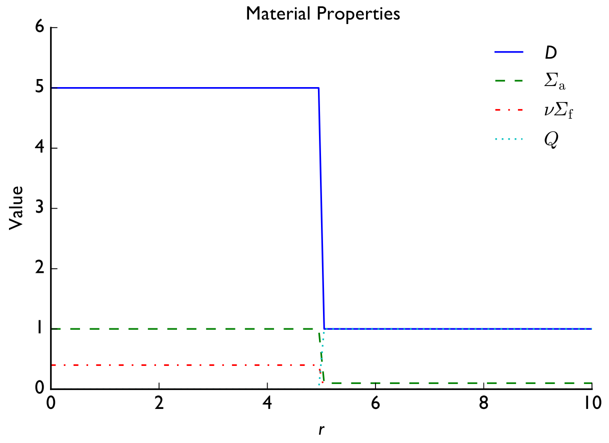
```
In [4]: R = 10
        I = 10
        #Solve Diffusion Problem in Slab geometry
        x, phi_slab = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[0,1,0],0)
        #Solve Diffusion Problem in cylindrical geometry
        rc, phi_cyl = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[0,1,0],1)
        #Solve Diffusion Problem in spherical geometry
        rs, phi_sphere = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[0,1,0],2)
```



These results demonstrate that our method for solving the diffusion equation can solve the simplest possible problem. This is an important test, however, because if our method cannot solve this problem correctly, it is not likely to be able to solve more difficult problems.

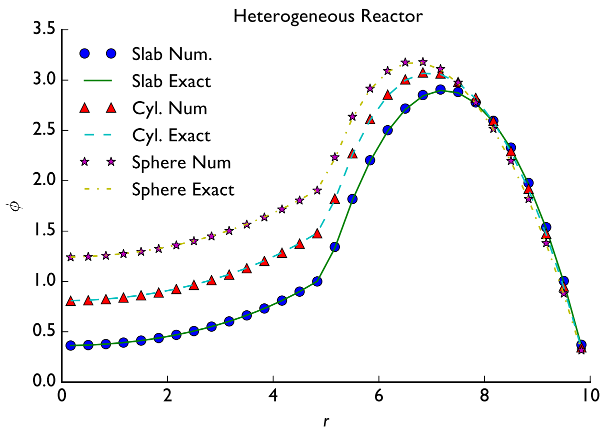
Next, we test our implementation on a heterogeneous problem where the material properties are discontinuous at $r = 5$ with fuel from 0 to 5 and moderator from 5 to 10. If we are solving for the thermal flux, there will be a source in the moderator.

```
In [5]: def D(r):
            value = 5.0*(r<=5) + 1.0*(r>5)
            return value;
        def Sigma_a(r):
            value = 1.0*(r<=5) + 0.1*(r>5)
            return value;
        def nuSigma_f(r):
            value = 0.4*(r<=5) + 0.0*(r>5)
            return value;
        def Q(r):
            value = 0*(r<=5) + 1.0*(r>5)
            return value
```



Now, we use the `DiffusionSolver` function to solve this problem with a zero-Dirichlet boundary conditions at $r = R$ and a reflecting boundary at $r = 0$:

```
In [6]: R = 10
        I = 30
        #Solve Diffusion Problem in Slab geometry
        x, phi_slab = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[1,0,0],0)
        #Solve Diffusion Problem in cylindrical geometry
        rc, phi_cyl = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[1,0,0],1)
        #Solve Diffusion Problem in cylindrical geometry
        rs, phi_sphere = DiffusionSolver(R, I,D, Sigma_a, nuSigma_f, Q,[1,0,0],2)
```



In this figure the exact solutions are also shown. These can be found by solving the diffusion equation in each region and joining the solutions by making the scalar flux and neutron current continuous at $r = 5$ [10]. For example, the slab solution is

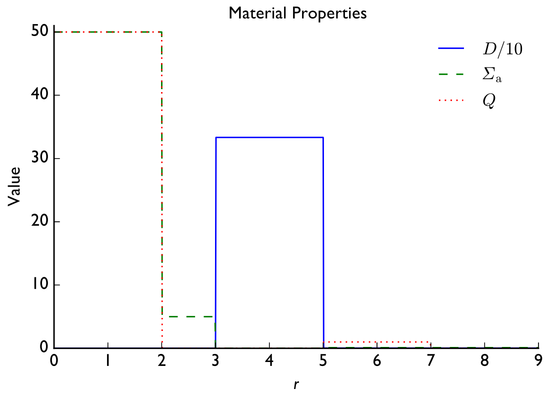
$$\phi(r) = \begin{cases} 0.181299e^{-0.34641r}\left(e^{0.69282r}+1\right) & r \leq 5 \\ 34.5868\sinh(0.316228r) - 35.3082\cosh(0.316228r) + 10 & r \geq 5 \end{cases}.$$

The numerical solutions seem to agree with the exact solutions, and indeed the difference between the two decreases as I gets larger.

18.3.1 Reed's Problem

Reed's problem is a common test problem for numerical methods for solving neutron diffusion and transport problems. It is a heterogeneous reactor problem that has several regions. The material properties for our geometry (reflecting at $r = 0$ and vacuum at $r = R$) is defined below.

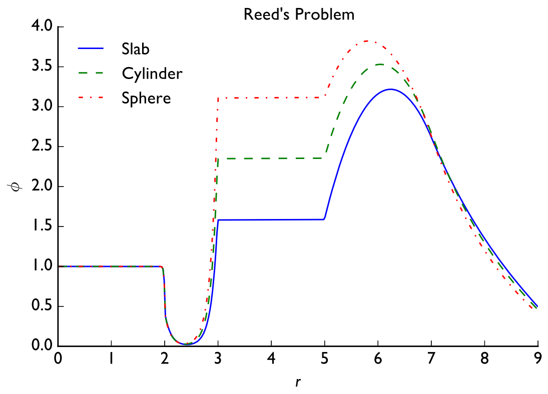
```
In [7]: #in this case all three are constant
        def D(r):
            value = (1.0/3.0*(r>5) +
                     1.0/3.0/0.001 *((r<=5) * (r>3)) +
                     1.0/3.0/5.0 *((r<=3) * (r>2)) +
                 1.0/3.0/50.0 * (r<=2))
        return value;
    def Sigma_a(r):
        value = 0+(0.1*(r>5) +
                 5.0 * ((r<3) * (r>2))+
                 50.0 * (r<=2))
        return value;
    def nuSigma_f(r):
        return 0*r;
    def Q(r):
        value = 0 + 1.0*((r<7) * (r>5)) + 50.0*(r<=2)
        return value;
```



This problem is set up so that there is a

- strong absorber with a strong source from $r = 0$ to 2,
- strong absorber without a source from $r = 2$ to 3,
- void from $r = 3$ to 4,
- scatterer with source from $r = 5$ to 7, and
- scatterer without source from $r = 7$ to 9.

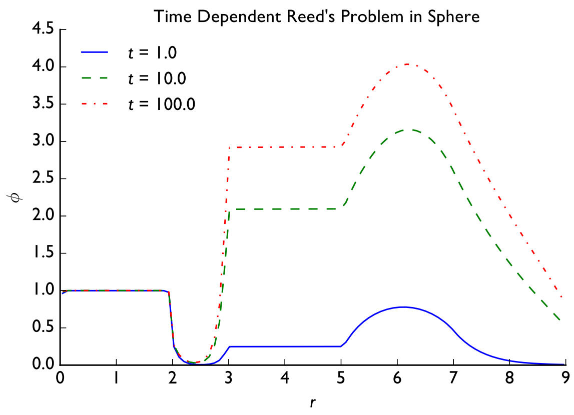
We approximate the void by having a very large diffusion coefficient and set $\Sigma_a = 0$. The solution to this problem using our `DiffusionSolver` function is given below.



Notice that the scalar flux for the sphere is highest, followed by the cylinder, then the slab. This is due to the fact that the leakage from the sphere is the smallest because it has the smallest ratio of surface area to volume. The solutions we obtain are consistent with previous solutions to Reed's problem: the solution is flat in the void region, peaks in the scattering region with source, and goes to a constant in the strong source and absorber region.

We can also solve Reed's problem in time dependent mode. In this case we will set the initial condition to have zero scalar flux everywhere, and solve a series of steady-state problems as we indicated at the beginning of the chapter. In the code below we solve this time dependent problem in spherical geometry with $\Delta t = 0.5$ and $v = 1$. We then plot the solution at $t = 1, 10, 100$.

```
In [8]: dt = 0.5
        tfinal = 100
        v = 1
        steps = np.linspace(dt,tfinal,tfinal/dt)
        R = 9
        I = 100
        dx = R/I
        phi_old = np.zeros(I)
        #define sigma_a star function
        Sigma_a_star = lambda r: Sigma_a(r) + 1/(v*dt)
        for step in steps:
           #construct Q star function, needs to convert r to the cell index
            Q_star = lambda r: Q(r) + phi_old[int((r-0.5*dx)/dx-1)]/(v*dt)
            #Solve Diffusion Problem in cylindrical geometry
            rs, phi_sphere = DiffusionSolver(R, I,D, Sigma_a_star,
                                             nuSigma_f, Q_star,
                                             [0.25,0.5*D(R),0],2)
            #update old solution
            phi_old = phi_sphere.copy()
            if (math.fabs(step-1) < dt):
                plt.plot(rs,phi_sphere,label="t = " + str(step))
            elif (math.fabs(step-10) < dt):
                plt.plot(rs,phi_sphere,'-',label="t = " + str(step))
            elif (math.fabs(step-100) < dt):
                plt.plot(rs,phi_sphere,'-.',label="t = " + str(step))
```



In this problem, steady state is reached relatively quickly near the center of the sphere, but the scattering region takes on the order of 100 seconds to reach steady state. I ran this problem with a coarser spatial mesh because we are solving 1000 steady state problems (instead of just one) to do the time dependent simulation.

CODA

In this chapter we have solved our first boundary value problem, and it is an important one for nuclear engineering: the neutron diffusion equation with a source. To do this we used a finite difference approximation to the second-derivative. Moreover, we have shown how to solve the diffusion equation in multiple geometries and in steady-state and time dependent modes.

The source-driven problems we solved here are important and can address many different applications: from shielding analyses to reactor accident scenarios. Nevertheless, there is a more important type of calculation we can perform: k-eigenvalue calculations to determine the criticality of a nuclear system. We will demonstrate how to do this in the next chapter.

PROBLEMS

Programming Projects

1. Code Testing

In this exercise you will test the implementation of the slab geometry diffusion equation solver. The first step is to develop two analytic solutions to the slab geometry diffusion equation with constant material properties.

QUADRATIC SOLUTION

The first problem we solve has $\Sigma_a = \nu\Sigma_f = 0$ with a zero Dirichlet boundary condition at $r = 1$. The specific equation you need to solve is

$$-D\frac{d^2\phi}{dr^2} = Q,$$

with boundary conditions

$$\left.\frac{d\phi}{dr}\right|_{r=0} = 0, \qquad \phi(1) = 0.$$

Solve this problem for $\phi(r)$.

HYPERBOLIC COSINE SOLUTION

The second problem we solve has $\Sigma_a - \nu\Sigma_f = 1$ with a zero boundary condition at $r = 1$. The specific equation you need to solve is

$$-D\frac{d^2\phi}{dr^2} + \phi(r) = Q,$$

with boundary conditions

$$\left.\frac{d\phi}{dr}\right|_{r=0} = 0, \qquad \phi(1) = 0.$$

Solve this problem for $\phi(r)$.

VERIFYING THE IMPLEMENTATION

Using the analytic solutions you calculated, compare numerical solutions to the exact answer using several values of Δr. Demonstrate that the method is working correctly by

- Stating what the expected behavior of the error should be as Δr changes on each of the two problems.
- Demonstrating that the observed behavior of the error as Δr changes is indeed what you see.

2. Time-Dependent, Super-Critical Excursion

A burst reactor is constructed by building a sphere of plutonium-239 that has a cylindrical hole inside which a slug of plutonium can be inserted. You will model this reactor as a solid sphere with a radius of 7 cm when the slug is in the reactor and as a hollow shell with the same outer radius and an inner radius of 2 cm when the slug is not present. For the plutonium use the following cross-sections from the report *Reactor Physics Constants*, ANL-5800:

Quantity	Value
σ_f [b]	1.85
σ_a [b]	2.11
σ_{tr} [b]	6.8
ν	2.98

For the density of plutonium use 19.74 g/cm^3; the diffusion coefficient is $D = 1/3\Sigma_{tr}$. In the hollow area of the shell, use $\Sigma_a = \nu\Sigma_f = 0$ and $D = 100$ cm. The average neutron speed in this problem is $v = 10^4$ cm/s. Recall that the macroscopic cross-section for reaction i is $\Sigma_i = N\sigma_i$, where N is the number density of nuclei.

An experiment is performed where there is initially 1000 neutrons uniformly distributed in the system at time $t = 0$ when the solid sphere is assembled. The sphere is solid until $t = 0.1$ s at which time you can assume the reactor is a hollow shell. Use a vacuum boundary condition on the edge of the sphere.

Your task is to compute the total number of neutrons that leak out of the sphere from time $t = 0$ to $t = 1$ s as well as the peak fission rate density in the reactor during the experiment.

Finally, determine the maximum value of

$$\frac{d}{dt}\ln\phi(r) \approx \frac{\ln\phi^{n+1}(r) - \ln\phi^{n}(r)}{\Delta t}.$$

Note: The leakage rate from the sphere per unit surface area is

$$-D\frac{d\phi}{dr}\bigg|_{r=7},$$

and the fission rate density at any point in the system is $\Sigma_{\mathrm{f}}(r)\,\phi(r)$.

3. Shielding Problem

You are tasked with shielding a spherical source of fast neutrons of radius 2.3 cm where the source emits 10^6 neutrons per second per cubic centimeter. You are constructing the shield out of iron and for fast neutrons the cross-sections for iron are (again from ANL-5800)

Quantity	**Value**
σ_{a} [b]	0.006
D [cm]	0.1234567

The fission cross-section for iron is 0. Assume the source has the absorption cross-section and diffusion coefficient of pure plutonium-239; ignore fission inside the plutonium (see the previous problem for the cross-sections). Set the boundary condition outside of the shield to have a zero incoming partial current.

How thick does the shield need to be so that no more than 10^4 neutrons leak out per second? The density of iron is 7.874 g/cm^3. *Hint: This might be a good problem for a nonlinear solver: you want to know when the leakage rate out of the reactor equals* 10^4 *as a function of the thickness of the shield.*