Part 1: Code Module Preparation

1. **Write a collision probability routine that generates the collision probability kernel, *p(i,j)* given the cylindrical geometry and cross section data. Include the first flight blackness (γ_i) as another output of the routine. You need to be very careful in order not to waste computation or memory in this routine. Use the number of Gauss points specified in the input. The 3-rd order Bickeley function needed for the collision probability function is attached at the end. The Gauss-Jacobi quadrature is also given.**

Collision Probability Kernel은 아래 식 (1.1)과 같이 정의 된다.

$$P_{ij} = \Sigma_i V_i \delta_{ij} + 2\left[\left(S_{i\,j} + S_{i-1\,j-1}\right) - \left(S_{i\,j-1} + S_{i-1\,j}\right)\right] \tag{1.1}$$

$S_{i\,j}$ 값을 모든 Annual ring에 대해서 계산해두면, P_{ij} 을 결정 할 수 있다. P_{ij} 값을 결정할 때 쓰이는 $S_{i\,j}$ 은 아래 식 (1.2)과 같다.

$$S_{ij}^{k} = \int_0^{R_i} \left\{ Ki_3(\tau_{ij}^{+}(y)) - Ki_3(\tau_{ij}^{-}(y)) \right\} dy \tag{1.2}$$

$S_{i\,j}$ 을 계산할 때에도, P_{ij} 을 계산할 때와 마찬가지로, 아래 식 (1.3)과 같이 S_{ij}^{k} 을 이용하면 된다. P_{ij} 값을 계산할 때와의 차이점은 $S_{i\,j}$ 을 메모리에 저장하지만, S_{ij}^{k} 는 저장하지 않고 $S_{i\,j}$ 가 할당된 변수에 계속해서 누적해 나간다는 것이다.

$$S_{ij} = \sum_{k=1}^{i} S_{ij}^{k}$$
$$where\ S_{ij}^{k} = \int_{R_{k-1}}^{R_k} \left\{ Ki_3(\tau_{ij}^{+}(y)) - Ki_3(\tau_{ij}^{-}(y)) \right\} dy \tag{1.3}$$

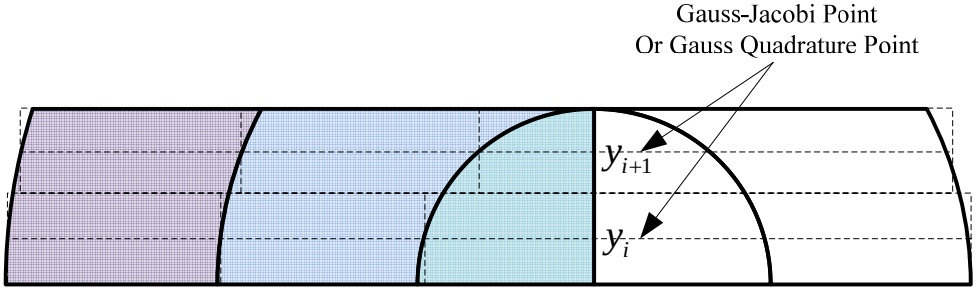


FIG 1 Quadrature set을 이용한 Numerical Integration

$Ki_3(x)$ 을 적분한 형태인 S_{ij}^k 을 Analytic 하게 적분하는 것은 실질적으로 불가능하다. 그러므로 FIG 1과 같이 Gauss Quadrature 나 Gauss-Jacobi Quadrature 를 이용해서 적분한다. Gauss Quadrature 의 경우, 적분 구간이 (-1, 1) 일 때 정의되어 있다. 그러므로 식 (1.3)의 적분 구간을 Changing variable을 통하여 바꾸어야 한다. 새로운 Integration variable을 아래 식 (1.4)와 같이 정의하면 적분은 식 (1.5)와 같이 변환 된다.

$$p = T(y) = 2\frac{y - R_{k-1}}{\Delta_k} - 1 \text{ where } \Delta_k = R_k - R_{k-1} \tag{1.4}$$

$$S_{ij}^k = \int_{R_{k-1}}^{R_k} \underbrace{\left\{ Ki_3(\tau_{ij}^+(y)) - Ki_3(\tau_{ij}^-(y)) \right\}}_{f(y)} dy = \frac{\Delta_k}{2}\int_{-1}^{1} f(T^{-1}(p))dp \tag{1.5}$$

식 (1.5)에 Gauss Quadrature을 적용하면 아래 식 (1.6)과 같이 표현된다.

$$\begin{aligned} S_{ij}^k &= \frac{\Delta_k}{2}\int_{-1}^{1} f(T^{-1}(p))dp \\ &= \frac{\Delta_k}{2}\sum_l \omega_l f(\underbrace{T^{-1}(p_l)}_{y_l}) = \frac{\Delta_k}{2}\sum_l \omega_l f(y_l) \end{aligned} \tag{1.6}$$

Gauss – Jacobi Quadrature 적분은 적분 구간이 (0,1)에 대해서 다음 식 (1.7)와 같다.

$$\int_0^1 xf(x)dx = \sum_i \omega_i f(x_i) \tag{1.7}$$

Where x_i : Gauss-Jacobi Points

식 (1.3)의 S_{ij}^k 의 식 (1.7)의 형태로 변화하기 위해서 아래와 같이 Integration variable을 변환하였다.

$$\begin{aligned} Let\ x = T_1(y) &= \frac{y - R_{k-1}}{\Delta_k} \\ S_{ij}^k = \int_{R_{k-1}}^{R_k} f(y)dy &= \Delta_k \int_0^1 f(T_1^{-1}(x))dx \end{aligned} \tag{1.8}$$

$$\begin{aligned} &\text{Define } q = T_2(x) = (1-x)^{1/2} \\ S_{ij}^k &= 2\Delta_k \int_0^1 q\, f(\underbrace{T_1^{-1}(T_2^{-1}(q))}_{T^{-1}(q)})dq = 2\Delta_k \int_0^1 q\, f(T^{-1}(q))dq \end{aligned} \tag{1.9}$$

식 (1.9)에 Gauss – Jacobi Quadrature을 적용하면 아래 식 (1.10)와 같은 결과를 얻는다.

$$
\begin{aligned}
S_{ij}^{k} &= 2\Delta_k \int_0^1 q\, f(T^{-1}(q))dq \\
&= 2\Delta_k \sum_l \omega_l f(T^{-1}(q_l)) = 2\Delta_k \sum_l \omega_l f(y_l) \\
&\text{where } y_l = \Delta_k(1-q_l^2) + R_{k-1}
\end{aligned}
\tag{1.10}
$$

S_{ij}^{k} 계산하기 위해서 필요한 τ_{ij}^{-}, τ_{ij}^{+}는 아래 FIG 2와 같이 정의 된다.

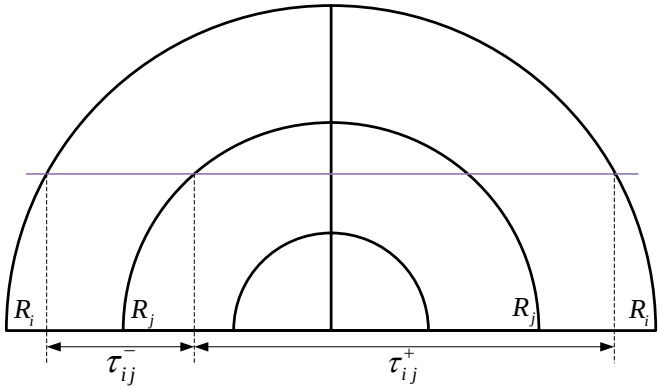


FIG 2 τ_{ij}^{-}, τ_{ij}^{+}의 정의

$\tilde{\tau}_i$을 아래 식 (1.11) 와 같이 정의하면 τ_{ij}^{-}, τ_{ij}^{+}을 아래 식 (1.12)와 같이 표현된다.

$$
\tilde{\tau}_i = \tau_{i\,i-1} \tag{1.11}
$$

$$
\tau_{ij}^{-} = \sum_{k=j+1}^{i} \tilde{\tau}_k\ ,\ \ \tau_{ij}^{+} = \tau_{i,j}^{-} + 2\sum_{k=1}^{j} \tilde{\tau}_k \tag{1.12}
$$

미리 계산해 둔 $\tilde{\tau}_i$ 이용하여, τ_{ij}^{-}, τ_{ij}^{+} 가 필요할 때 계산한다. S_{ij}을 계산하기 위해서는 S_{ij}^{k} 을 연속적으로 계산하게 된다. S_{ij} 계산하기 위한 S_{ij}^{k}을 가장 일반적은 순서는 아래 FIG 3와 같다.

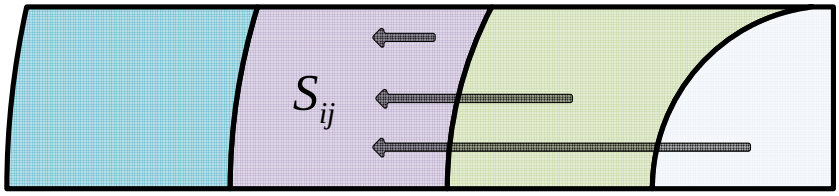


FIG 3 S_{ij} 계산 순서 1

이러면 Algorithm의 경우 반복 loop을 돌아야 하는 횟수가 많아진다. 위와 같이 S_{ij} 한 번에 계산하지 않고 아래와 FIG 4와 같은 순서로 S_{ij}^{k}을 계산하여 해당하는 S_{ij}에 더해나간다. 모든 Loop을 끝나고 나면, S_{ij} 값의 계산이 끝이 난다.

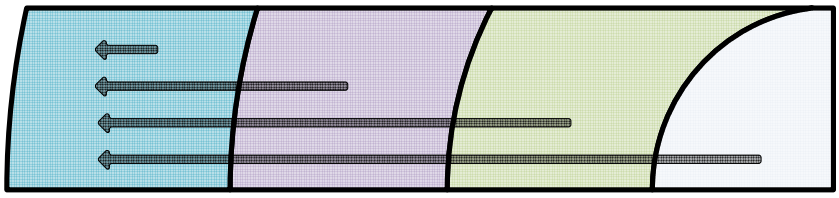


FIG 4 $S_{i\,j}$ 계산 순서 2

식 (1.11)에서 보면, P_{ij} 을 계산할 때 $S_{i\,j}$ 을 계산할 때 마다 2배를 곱해준다. 그래서 S_{ij}^{k} 를 계산할 때 미리 2배를 미리 곱해주었다.

```
      SUBROUTINE set_S
      use global
      implicit none
      include "geom.FI"
      include "prob.FI"
      include "xs.FI"
      include "nIntegration.FI"
      INTEGER :: i,j,k,l
      REAL(NBF) :: Ri,Rj,delR,xi(ngauss),xj(ngauss),pxj(ngauss)
      REAL(NBF) :: ypt(ngauss),tauP(ngauss),tauN(ngauss),tau(ngauss,0:nr)
      REAL(NBF) :: Sjk_i,Ki3
      real(NBF),pointer :: IWT(:),IPT(:)
      ALLOCATE(S(0:nr,0:nr))
#define GQ1
#IFDEF GQ
      IPT => GQPT(1:ngauss,ngauss); IWT => GQWT(1:ngauss,ngauss)
#ELSE
      IPT => GJPT(1:ngauss,ngauss); IWT => GJWT(1:ngauss,ngauss)
#ENDIF
      S=0.0_NBF
      !wt(:) = GQPT
      DO i=1,nr
        Ri= R(i); delR=R(i)-R(i-1)
#IFDEF GQ
        ypt(:) = 0.5_NBF*delR*(IPT+1._NBF)+R(i-1)
#ELSE
        ypt(:) = delR*IPT+R(i-1)
#ENDIF
        xj=0; tau=0
        DO j = i,nr
           pxj=xj
           xj(:) = sqrt(R(j)**2-ypt(:)**2)
           tau(1:,j) = sigt(j)*(xj(:)-pxj(:))
        ENDDO
        continue
        DO j = i, nr
          tauN= - tau(:,j)
          tauP(:)= sum(tau(:,0:j),DIM = 2) + sum(tau(:,0:j-1),DIM = 2)
          DO k = j, nr
            tauN = tauN + tau(:,k)
            tauP(:) = tauP(:) + tau(:,k)
            continue
            Sjk_i = 0._NBF
            DO l = 1, ngauss
```

```
            Sjk_i = Sjk_i + IWT(l)*(Ki3(tauP(l))-ki3(tauN(l)))
          ENDDO
          sjk_i=sjk_i*delR
          S(k,j) = S(k,j) + Sjk_i  !Note S k,j
        ENDDO
      ENDDO
    ENDDO
    DO i=1,nr
      DO j=1,i-1,1
        S(j,i)=S(i,j)
      ENDDO
    ENDDO
    END SUBROUTINE
```

2. Write a LU factorization routine which incorporates partial pivoting so that the resulting L, U factors are for the permuted matrix, namely: $PA = LU$. Note that the permutation matrix is for row exchange which corresponds to changing the order of equations. Then also write the forward and backward substitution module that uses the L and U matrices and the row exchange information vector to solve a linear system given a right hand side vector.

일반적인 LU factorization의 문제점은, Elimination의 과정 중 Pivot의 값이 너무 작을 경우에 Truncation 오차가 증가한다는 것이다(∵Pivot은 역수로 곱해지므로). 이것을 방지하기 위해서 pivoting scheme을 사용하게 된다. 여기서 사용한 scheme은 partial pivoting이다. 만약 i-th column을 elimination 한다면, (i, i)의 element 밑에 있는 element 중에서 가장 작은 것을 pivot 으로 쓰는 것이 partial pivoting scheme 이다. 이 과정에서 row을 서로 바꾸어주어야 한다. 그리고 줄을 바꾼다는 것은 Linear system의 변수의 정렬 순서를 바꾼다는 뜻이므로, 처음과 어떻게 바뀌었는지를 저장하여야 한다. 그것을 저장하는 것이 Permutation Matrix P이다. 하지만 P의 경우 Identity matrix에서 row을 바꾼 것이다. 이 Matrix을 저장하기보다, column vector에 어떻게 row을 바꾸었는지를 저장한다.

```
    subroutine LU_factorize(A,LU,pvec,n)
    use global
    implicit none
    real(NBF),intent(in) :: A(n,n)
    real(NBF),intent(out) :: LU(n,n)
    integer,intent(Out) :: pvec(n)
    integer,intent(in) :: n
    integer :: i,j,k,iipvt(1),ipvt,itp
    real(NBF) :: pvt,rowtp(n),lmnt
    equivalence(ipvt,iipvt(1))
    LU=A
    DO i =1,n
      pvec(i) = i
    ENDDO
    DO i = 1,n-1
      pvt = LU(i,i)
      iipvt = MAXLOC(abs(LU(i,i:n))) + i-1
      pvt = LU(i,ipvt)
```

```
        itp = pvec(i); pvec(i) = ipvt; pvec(ipvt)=itp
        rowtp(:) = LU(:,i);LU(:,i) = LU(:,ipvt)
        LU(:,ipvt) = rowtp(:)
        continue
        pvt=1._NBF/pvt
        DO j = i+1,n
          lmnt = pvt*LU(i,j)
          LU(i,j) = lmnt
          LU(i+1:n,j)=LU(i+1:n,j) - lmnt*(LU(i+1:n,i))
        ENDDO
        continue
      ENDDO
      DO i=1,n
        LU(i,i) = 1._NBF/LU(i,i)
      ENDDO
      continue
      end subroutine
```

LU Factorization의 결과로 Matrix A는 식 (1.13)과 같이 표현 된다.

$$PA = LU$$
$$A = P^{-1}LU \tag{1.13}$$

Ax=b라는 Matrix 를 partial pivot을 한 LU을 통해서 푼다는 것은 아래 식 (1.14)를 푼다는 것과 동일 한 것이다.

$$Ax = \left(P^{-1}LU\right)x = b$$
$$LUx = Pb \tag{1.14}$$

즉 좌변의 b를 LU 과정에서 row을 바꾼 것과 동일하게 바꾼 후, Backward, forward substitution을 해주면 Linear system을 풀게 된다.

```
      subroutine LU_solver(LU,pvec,b,sol,n)
      use global
      implicit none
      REAL(NBF),intent(in) :: LU(n,n),b(n)
      integer,intent(in) :: pvec(n),n
      REAL(NBF),intent(out) :: sol(n)
      REAL(NBF) :: y(n),lmnt
      integer :: i,j
      !Forward Substitution
      sol = 0.0_NBF
      DO i=1,n
        lmnt = 0.0_NBF
        DO j=1,i-1,1
          lmnt = lmnt - LU(j,i)*y(j)
        ENDDO
        y(i) = lmnt + b(pvec(i)) !Row exchange
      ENDDO
      !Backward Substitution
      DO i=n,1,-1
```

```
        lmnt = 0.0_NBF
        DO j= i+1,n,1
          lmnt = lmnt - LU(j,i)*sol(j)
        ENDDO
        sol(i) = (lmnt + y(i))*LU(i,i)
      ENDDO
      end subroutine
```

3. Write a routine that determines the flux factors due to the source and source current, respectively. This will involve the solution of (n+1) linear systems.

Collision probability Method 에서 각 region의 flux는 아래 식 (1.15) 와 같이 표현된다. 식 (1.15)를 잘 보면, flux는 두 가지 요소의 영향을 가지는 것을 알 수 있는데, 하나는 External Current 이며, 다른 하나는 각 Annular region의 source이다.

$$\phi_i = Y_i(\alpha)\, j_{ext} + \sum_{k=1}^{n} X_i^k(\alpha) Q_k \tag{1.15}$$

External Current을 flux에 영향을 나타내는 $Y_i(\alpha)$, source가 주는 영향을 나타내는 $X_i^k(\alpha)$ 은 아래 식 (1.16), (1.17)과 같다.

$$Y_i(\alpha) = Y_i n_\alpha^\Gamma = \frac{Y_i}{1-(1-\Gamma)\alpha}$$
$$\text{where } \Gamma = \sum_{i=1}^{n} \Gamma_i = \sum_{i=1}^{n} \Sigma_{ri} V_i Y_i \tag{1.16}$$

$$X_i^k(\alpha) = X_i^k + \alpha x_k Y_i(\alpha)$$
$$\text{where } x_k = \frac{S_B}{4\Sigma_{rk}} \Gamma_k \tag{1.17}$$

Albedo가 0일 때 $X_i^k(\alpha)$, $Y_i(\alpha)$ 의 값을 나타내는 X_i^k, Y_i 는 아래 식(1.18)와 (1.19)를 통해서 결정할 수 있다.

$$\begin{bmatrix} \Sigma_1 V_1 - P_{11}c_1 & -P_{21}c_1 & \cdots & -P_{n1}c_1 \\ -P_{12}c_1 & \Sigma_2 V_2 - P_{22}c_2 & \cdots & -P_{n2}c_2 \\ \vdots & \vdots & \ddots & \vdots \\ -P_{1n}c_1 & -P_{12}c_2 & \cdots & \Sigma_n V_n - P_{nn}c_n \end{bmatrix} \begin{bmatrix} X_1^k \\ X_2^k \\ \vdots \\ X_n^k \end{bmatrix} = \begin{bmatrix} P_{k1}/\Sigma_1 \\ P_{k2}/\Sigma_2 \\ \vdots \\ P_{kn}/\Sigma_n \end{bmatrix} \tag{1.18}$$

$$\begin{bmatrix} \Sigma_1 V_1 - P_{11}c_1 & -P_{21}c_1 & \cdots & -P_{n1}c_1 \\ -P_{12}c_1 & \Sigma_2 V_2 - P_{22}c_2 & \cdots & -P_{n2}c_2 \\ \vdots & \vdots & \ddots & \vdots \\ -P_{1n}c_1 & -P_{12}c_2 & \cdots & \Sigma_n V_n - P_{nn}c_n \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix} \tag{1.19}$$

X_i^k, Y_i 을 구하는 Routine은 아래와 같다.

```
      subroutine set_XNY
      use global
      implicit none
      include "geom.FI"
```

```
      include "xs.FI"
      include "prob.FI"
      real(NBF) :: LU(nr,nr),rhs(nr),inv_sigt(nr)
      REAL(NBF) :: inv4Sb
      integer :: pvec(nr)
      Integer :: i,j,k

      CALL LU_factorize(A,LU,pvec,nr)
      inv_sigt(:) = 1._NBF/sigt(:)
      ALLOCATE(X(nr,nr),Y(nr))
      DO k = 1, nr
        rhs = P(:,k)*inv_sigt(k)
        CALL LU_solver(LU,pvec,rhs,X(:,k),nr)
      ENDDO
      inv4sB=4._NBF/(2._NBF*pi*r(nr))
      Do i = 1, nr
        rhs(i) = (sigt(i)*vol(i) - sum(P(i,1:nr)))*inv4sb
      ENDDO
      CALL LU_solver(LU,pvec,rhs,Y,nr)
      continue
      deallocate(A)
      END subroutine
```

$X_i^k(\alpha), Y_i(\alpha)$을 계산하는 routine은 다음과 같다.

```
      gamma = 0._NBF
      DO i = 1, nr
        g(i) = sigr(i)*vol(i)*Y(i)
        gamma = gamma + g(i)
      ENDDO
      n_a = 1._NBF/(1._NBF-alb*(1-gamma))
      continue
      Y_a = n_a*Y
      Sb = pi*2._NBF*R(nr)
      x_k= 0.25_NBF*Sb*Vol(1:)*Y
      nsb = sum(q(:)*x_k(:))
      DO k = 1,nr
        X_a(:,k) = X(:,k) + alb*x_k(k)*Y_a
      ENDDO
```

4. Write a routine that determines the flux at each region and currents at the external boundary. Generate the edit shown at the end.

Flux는 위 식 (1.15)를 통해서 구할 수 있다. Incoming Outgoing current는 아래 식 (1.20), (1.21)을 이용하년 결정할 수 있다.

$$J^{-}(\alpha) = \frac{x + (1-\Gamma)J_{ext}}{1-\alpha(1-\Gamma)} \tag{1.20}$$

$$J^{+}(\alpha)=\frac{\alpha x+J_{ext}}{1-\alpha(1-\Gamma)} \tag{1.21}$$

```
      ALLOCATE(phi(nr),src_phi(nr),J_phi(nr))
      DO i = 1,nr
        J_phi(i) = jext*Y_a(i)
        src_phi(i) = sum(q(:)*X_a(i,:))
        phi(i) = J_phi(i) + src_phi(i)
      ENDDO
      Jout = n_a*(nsb+(1-gamma)*Jext)
      Jin = n_a*(alb*nsb+Jext)
      jnet = Jout - Jin
```

Part 2: Performance Examination

1. Solve the two problems attached. The result should be the same as the reference output.

Input set 1

```
Albedo =  0.0000E+00     Jext =  1.0000E+00
    jnet =  1.2097E+00     jout =  2.2097E+00       jin  1.0000E+00
  Region      Radius      SigRem     SigScat      Source    SRC Flux      J Flux  Total Flux     Removal
       1  5.0000E-01  9.0000E-01  1.0000E-01  1.0000E-01  5.2749E-01  3.2944E-01  8.5693E-01  6.0573E-01
       2  6.0000E-01  1.0000E-01  4.0000E-01  0.0000E+00  6.3783E-01  3.7834E-01  1.0162E+00  3.5116E-02
       3  1.0000E+00  1.0000E-01  1.9000E+00  1.0000E+00  7.0847E-01  4.7848E-01  1.1869E+00  2.3865E-01
  Rtot =  8.7949E-01     jnet =  1.2097E+00     Sum =  2.0892E+00     qtot =  2.0892E+00
```

Input set 2

```
Albedo =  1.0000E+00     Jext =  0.0000E+00
    jnet =  0.0000E+00     jout =  1.4208E+00       jin  1.4208E+00
  Region      Radius      SigRem     SigScat      Source    SRC Flux      J Flux  Total Flux     Removal
       1  5.0000E-02  2.2050E+00  2.4500E-01  1.0000E+00  7.9850E-01  0.0000E+00  7.9850E-01  1.3828E-02
       2  1.0000E-01  2.2050E+00  2.4500E-01  1.0000E+00  8.0561E-01  0.0000E+00  8.0561E-01  4.1855E-02
       3  1.5000E-01  2.2050E+00  2.4500E-01  1.0000E+00  8.2024E-01  0.0000E+00  8.2024E-01  7.1025E-02
       4  2.0000E-01  2.2050E+00  2.4500E-01  1.0000E+00  8.4366E-01  0.0000E+00  8.4366E-01  1.0227E-01
       5  2.5000E-01  2.2050E+00  2.4500E-01  1.0000E+00  8.7802E-01  0.0000E+00  8.7802E-01  1.3685E-01
       6  3.0000E-01  2.2050E+00  2.4500E-01  1.0000E+00  9.2729E-01  0.0000E+00  9.2729E-01  1.7665E-01
       7  3.5000E-01  2.2050E+00  2.4500E-01  1.0000E+00  9.9994E-01  0.0000E+00  9.9994E-01  2.2512E-01
       8  4.0000E-01  2.2050E+00  2.4500E-01  1.0000E+00  1.1232E+00  0.0000E+00  1.1232E+00  2.9177E-01
       9  4.8000E-01  4.0000E-01  1.0000E-01  1.0000E+00  1.3387E+00  0.0000E+00  1.3387E+00  1.1843E-01
      10  5.6000E-01  1.7000E-01  1.3300E+00  1.0000E+00  1.4405E+00  0.0000E+00  1.4405E+00  6.4009E-02
      11  6.5000E-01  1.7000E-01  1.3300E+00  1.0000E+00  1.4702E+00  0.0000E+00  1.4702E+00  8.5508E-02
  Rtot =  1.3273E+00     jnet =  0.0000E+00     Sum =  1.3273E+00     qtot =  1.3273E+00
```

2. Try the normal Gauss quadrature instead of the Gauss-Jacobi quadrature and examine the difference in the accuracy of the solution.

Gauss Quadrature와 Gauss-Jacobi Quadrature의 points 수를 늘려가면서 값을 비교해보았다. Reference 값은 6개의 Gauss-Jacobi Quadrature 으로 하였다. 그 결과를 Table 1에 정리하였다. Table 1의 결과를 보면, 동일한 차수의 Gauss-Jacobi의 결과가 오차가 적은 것을 알 수 있다. 그러므로 동일한 적분 point을 사용한다면, Gauss-Jacobi을 사용하는 것이 Gauss quadrature set을 이용하는 것보다 유리하다는 것을 알 수 있다.

Table 1 Gauss Quadrature 와 Gauss Jacobi Quadrature Error 비교

region	Reference[1)]	Gauss-Jacobi Quadrature Rel. Error[%]			Gauss Quadrature Rel. Error [%]		
		1	3	5	1	3	5
1	7.9850E-01	-2.26	-0.003	0.000	-3.862	-0.174	-0.044
2	8.0561E-01	0.14	0.000	0.000	-0.544	-0.017	-0.004
3	8.2024E-01	0.18	0.001	0.000	-0.217	-0.007	-0.001
4	8.4366E-01	0.14	0.000	0.000	-0.114	-0.002	0.000
5	8.7802E-01	0.09	0.000	0.000	-0.076	0.000	0.000

6	9.2729E-01	0.03	0.000	0.000	-0.074	-0.001	0.000
7	9.9994E-01	-0.09	0.000	0.000	-0.106	-0.004	-0.001
8	1.1232E+00	-0.29	0.000	0.000	-0.125	-0.036	-0.009
9	1.3387E+00	0.40	0.000	0.000	0.889	0.067	0.015
10	1.4405E+00	0.22	0.000	0.000	0.535	0.028	0.007
11	1.4702E+00	0.29	0.000	0.000	0.565	0.041	0.014

1) Reference 값은 Gauss Jacobi 6th order scheme 을 이용하여서 계산

3. **Implement a small driver that performs the same calculation 100,000 times. Incorporate random perturbation in the given removal cross section. Your code will be examined on a common platform for comparison with other people's code.**

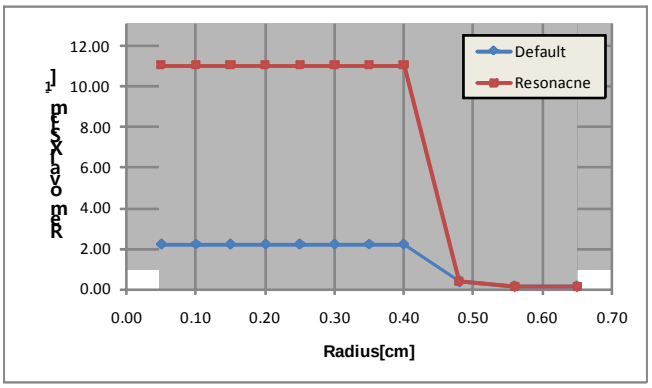
```
      SUBROUTINE PerformanceTest
      use global
      implicit none
      include "xs.FI"
      include "geom.FI"
      integer :: i
      real(NBF) :: t1,t2
      call CPU_time(t1)
      allocate(sigt0(nr))
      sigt0 = sigt
      DO i= 1, 100000
        CALL RMV_change
        CALL set_S
        CALL set_p
        CALL set_A
        CALL set_XNY
        CALL set_flux
      ENDDO
      CALL CPU_time(t2)
      print '(A13,2x,F9.5)', 'Elaspe Time :', t2-t1
      stop
      ENDSUBROUTINE

      SUBROUTINE RMV_change
      use global
      use IFPORT
      implicit none
      include 'geom.FI'
      include 'xs.FI'
      integer :: ir
      real(NBF) :: ranval
      DO ir = 1, nr
        ranval=drand(0)
        sigr(ir) =(sigt0(ir)-sigs(ir))* ranval
        sigt(ir) = sigs(ir)+sigr(ir)
        c(ir) = sigs(ir)/sigt(ir)
      ENDDO
      ENDSUBROUTINE
```

위의 코드를 input 2번에 대해서 수치로 node10번에서 100,000번 수행해 본 결과, 10번 평균 9.87 초의 시간이 소요 되었다.

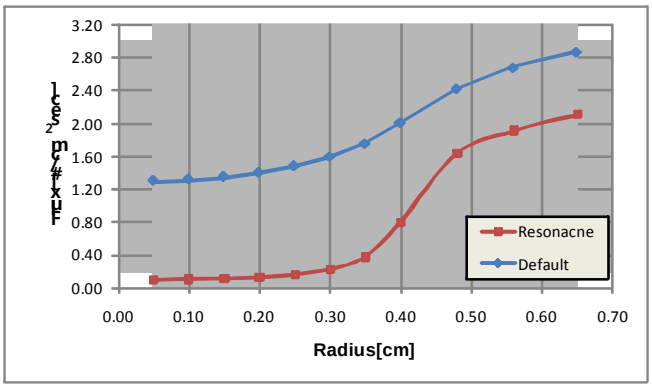
Part 3: Physics Examination

1. Try different absorption cross section to simulate various levels of resonance. Then plot the flux shapes to compare different extents of spatial self shielding. Determine the cross sections on your own according to your own logic.

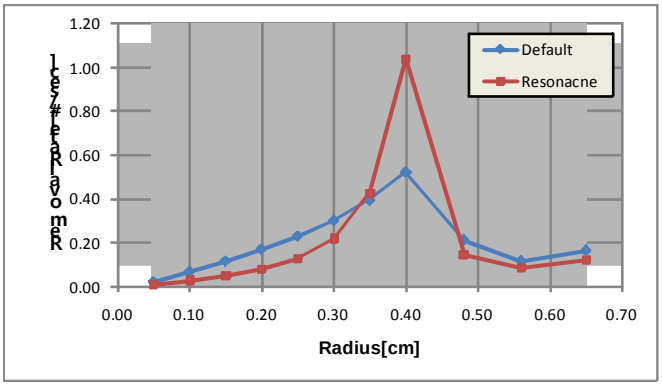
Resonance Self-shielding 효과를 확인하기 위해서, Input 2 번을 수정하였다. 먼저 1~9 annular ring의 removal XS을 5배로 증가시켰다(Fig. (a)). 그리고 External Current을 1.0 #/cm²로 설정하였다. Flux의 shape을 removal XS을 바꾸지 않은 경우와 비교해보면. 전체적인 flux shape가 낮음을 알 수 있다. Annular ring 의 중앙부는 removal XS이 증가 하였으므로, flux의 level이 낮아짐을 알 수 있다. 외각의 flux level이 낮은 이유는 removal XS이 증가한 중앙부에서의 neutron이 collision 을 하지 않고 빠져나갈 확률이 낮기 때문이다. 각 영역별로 removal rate의 분포를 살펴보면, 0.4cm 영역에서 가장 크게 나타나고, 그보다 안쪽 영역에서는 removal XS을 증가시키지 않은 경우보다 작게 나타남을 알 수 있다.



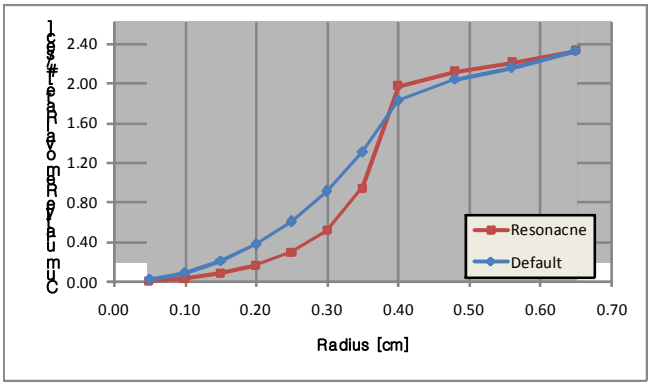
(a) Removal XS



(b) Flux Shape



(c) Removal Rate



(d) Cumulative Removal Rate

FIG 5 Spatial Self-Shielding

2. Write a report that describes the method, the coding, and the analysis result in a self-contained form, meaning that the report can be understood later without resorting to the textbook or the class note.

•Transport Kernel

공간 상에 $\vec{r}$, $\vec{r}\,'$라는 두 지점이 있다고 하자. 먼저 $\vec{r}\,'$지점의 neutron source는 아래 식 (3.1)과 같고, $\vec{r}\,'$지점의 source에 의해 생기는 $\vec{r}\,'$지점의 flux는 아래 식 (3.2)와 같이 표현된다.

$$Q(\vec{r}\,') = \Sigma_s \phi(\vec{r}\,') + Q'(\vec{r}\,') \tag{3.1}$$

$$\phi(\vec{r}\,' \to \vec{r})dV = \frac{e^{-\rho(R)}}{4\pi R^2}\left(\Sigma_s \phi(\vec{r}\,') + Q'(\vec{r}\,')\right)dV' \tag{3.2}$$

이제 공간상에 flux와 source가 균일한 어떤 두 영역 i, j가 있다고 하자.

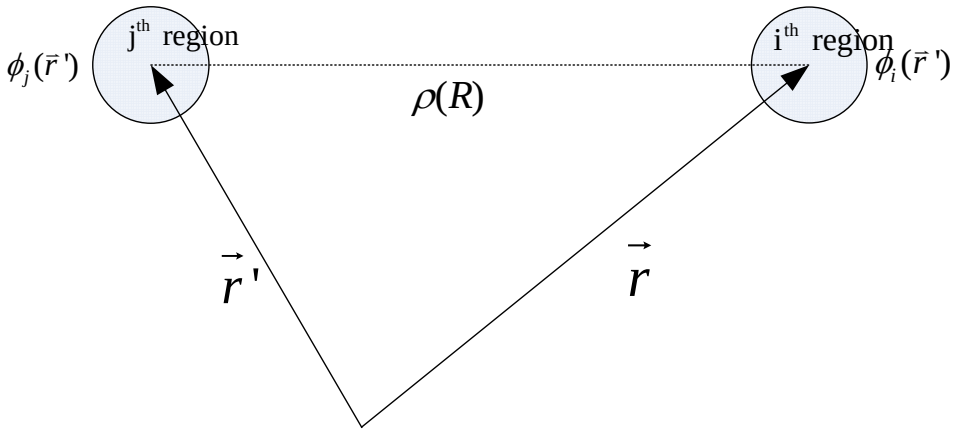


FIG 6 Transport Kernel

그러면 j 영역의 source에 의해서 만들어진 i 영역에 속하는 $\vec{r}$의 flux는 다음과 같다.

$$\phi_{ji}(\vec{r})dV = \int_{V_j} \underbrace{\frac{e^{-\rho(R)}}{4\pi R^2}}_{n(\vec{r}\,' \to \vec{r})} (\Sigma_{sj}\phi_j + Q_j')dV'dV$$

$$\int_{V_j} n(\vec{r}\,' \to \vec{r})(\Sigma_{sj}\phi_j + Q_j')dV'dV \tag{3.3}$$

I의 전체 영역에 대해서 표현하면 아래 식과 같은 결과가 나온다.

$$\phi_{ji} = \frac{1}{V_i}\int_{V_i}\phi_{ji}(\vec{r})dV = \frac{1}{V_i}\int_{V_i}\int_{V_j} n(\vec{r}\,' \to \vec{r})(\Sigma_{sj}\phi_j + Q_j')dV'dV$$
$$= \underbrace{\frac{1}{V_i}\int_{V_i}\int_{V_j} n(\vec{r}\,' \to \vec{r})dV'dV}_{T_{ji}}(\Sigma_{sj}\phi_j + Q_j') \tag{3.4}$$
$$= T_{ji}(\Sigma_{sj}\phi_j + Q_j')$$

Optical Length는 $\rho(R) = \rho(\vec{r}\,', \vec{r}) = \rho(\vec{r}, \vec{r}\,')$ 다음과 같은 관계가 성립한다.

$$\int_{V_i}\int_{V_j}\frac{e^{-\rho(R)}}{4\pi R^2}dV'dV = \int_{V_j}\int_{V_i}\frac{e^{-\rho(R)}}{4\pi R^2}dVdV' \tag{3.5}$$

$$T_{ji}V_i = T_{ij}V_j \quad \text{(Reciprocity Relation)} \tag{3.6}$$

ϕ_i 는 공간을 구성하는 각각의 영역의 source가 i 영역에 주는 영향의 합으로 나타나므로, 다음과 같이 표현된다.

$$\phi_i = \sum_j \phi_{ji} \tag{3.7}$$

Collision Probability

$\vec{r}\,'$ 지점의 unit source 가 $\vec{r}$ 지점에서 처음으로 collision을 할 확률은 아래 식처럼 나타난다.

$$P(\vec{r}\,' \to \vec{r})dV = \Sigma_t n(\vec{r}\,' \to \vec{r})dV \tag{3.8}$$

j 영역의 source가 i 영역에서 처음으로 collision 일으키는 양(*Collision Rate*)은 $\Sigma_t\phi_{ji}V_i$ 이고, 이것을 식 (3.4), (3.8) 을 이용해서 나타내면 아래와 같이 정리된다.

$$
\begin{aligned}
\Sigma_t \phi_{ji} V_i &= \Sigma_t T_{ji} V_j (\Sigma_{s\,j} \phi_j + Q_j) \\
&= \Sigma_t \left(\frac{1}{V_i} \int_{V_i} \int_{V_j} n(\vec{r}\,' \to \vec{r}) dV'dV \right) V_j (\Sigma_{s\,j} \phi_j + Q_j) \\
&= \frac{1}{V_i} \left(\int_{V_i} \int_{V_j} \underbrace{\Sigma_t n(\vec{r}\,' \to \vec{r})}_{P(\vec{r}\,' \to \vec{r})} dV'dV \right) V_j (\Sigma_{s\,j} \phi_j + Q_j) \\
&= \underbrace{\frac{1}{V_i} \left(\int_{V_i} \int_{V_j} P(\vec{r}\,' \to \vec{r}) dV'dV \right)}_{\tilde{P}_{ji}} \underbrace{V_j (\Sigma_{s\,j} \phi_j + Q_j)}_{\text{Total Source at } V_j} \\
&= \tilde{P}_{ji} V_j (\Sigma_{s\,j} \phi_j + Q_j)
\end{aligned}
\tag{3.9}
$$

위 식(3.9) 에서 정의된 $\tilde{P}_{ji}$ 는 영역 j의 isotropic한 unit source가 영역 i 에서 처음으로 충돌할 확률을 의미하고, *Collision Probability* 라고 부른다.

$$
\tilde{P}_{ji} = \frac{1}{V_i} \left(\int_{V_i} \int_{V_j} P(\vec{r}\,' \to \vec{r}) dV'dV \right) \tag{3.10}
$$

식 (3.9)의 collision rate는 아래와 같이 변형된다.

$$
\begin{aligned}
\Sigma_t \phi_{ji} V_i &= \tilde{P}_{ji} V_j (\Sigma_{s\,j} \phi_j + Q_j) \\
&= \Sigma_j \tilde{P}_{ji} V_j \left(\frac{\Sigma_{s\,j}}{\Sigma_j} \phi_j + \frac{Q_j}{\Sigma_j} \right)
\end{aligned}
\tag{3.11}
$$

$$
P_{ji} = \tilde{P}_{ji} V_j \Sigma_j \qquad c_j = \frac{\Sigma_{s\,j}}{\Sigma_j} \tag{3.12}
$$

(3.13)

ㅊ

c_j 는 j 영역에서의 일어나는 총 충돌 중에서 self scattering이 차지하는 비율을 나타낸다. Q_j / Σ_j 은 flux의 단위와 동일하고, 그래서 이 항을 source driven flux 라고 부른다. P_{ji} 을 식 (3.12)와 같이 정의하면, Collision rate는 식 (3.12)처럼 간단하게 표현된다. 이렇게 표현하는 이점은 flux($c_j \phi_j + Q_j / \Sigma_j$)에 P_{ji} 을 곱해줌으로써 Collision rate가 되기 때문이다. 그러한 이유로 P_{ji} 을 *flux to collision factor* 라고 한다.

Cosine Current

어떠한 영역에 균일하고 isotropic 한 flux가 있다면, Angular flux는 식 (3.14)처럼 나타난다.

$$\varphi(\hat{\Omega}) = \frac{\phi}{4\pi} \quad \text{or} \quad \varphi(\mu) = \frac{\phi}{2} \tag{3.14}$$

식 (3.14)와 같은 isotropic하고 균일한 source가 단위면적을 가진 어떠한 surface 위에 있다고 할 때, 이 단위면적인 표면을 통과하는 neutron의 수를 cosine current라고 정의하고, 아래 식 (3.15)와 같이 쓸 수 있다.

$$\varphi_{\perp}(\mu) = \frac{\phi}{2}\mu \tag{3.15}$$

Cosine current 의 분포를 가진 표면에서의 outgoing current는 식 (3.16)과 같다.

$$J_{out} = \int_0^1 \varphi_{\perp}(\mu) d\mu = \frac{1}{4}\phi \tag{3.16}$$

Escape Probability and First Collision Escape Probability

부피가 V인 영역에 uniform한 source가 있다면, 아래 식 (3.17)의 중성자 균형 방정식이 성립한다.

$$\Sigma_{tg}\phi_g = \Sigma_{gg}\phi_g + Q_g'$$
$$\phi_g = \frac{Q_g'}{\Sigma_{rg}} \tag{3.17}$$

단위 시간당 하나의 중성자가 영역에서 생성된다면, Q_g'는 $1/V$ 가 되며 flux는 식 (3.17)에 따라서 식 (3.18)이 된다.

$$\phi_g = \frac{Q_g'}{\Sigma_{rg}V} \tag{3.18}$$

부피가 V이고, 이 영역이 closed surface S에 의해서 둘러 쌓여 있다고 할 때, Escape Probability와 Absorption Blackness는 다음과 같이 정의된다.

Escape Probability(P): 어떤 영역에서 태어나서, 그 영역을 흡수되지 않고 영역을 둘러싸는 표면 S을 지나서 빠져나갈 확률

Absorption Blackness(Γ): Surface S에 균일한 cosine current distribution의 선원이 있다고 할 때, 영역 내에서 흡수될 확률

Q_g'는 $1/V$ 이라면, 영역 내에서 빠져나가는 중성자의 양은 P 이고, surface S을 통과하여서 들어오는 중성자 중에서 영역에서 흡수되는 양은 $J_{in}S\Gamma$ 이다. Steady State 상황에서는 내부에서 생

성되어서 외부로 빠져나가는 양과 외부에서 들어와서 내부에 흡수되는 양이 균형을 이루어야 한다.

$$\frac{1}{4}\phi_s S\Gamma = P \quad (\because J_{in} = \frac{1}{4}\phi_s) \tag{3.19}$$

식 (3.19)에 식 (3.18)을 대입하면 아래와 같다.

$$\Gamma = \frac{4}{\phi S}P = \frac{4V}{S}\Sigma_r P \tag{3.20}$$

Mean Chord Length는 아래 식 (3.21) 과 같이 되며, 식 (3.20)에 대입하면 식 (3.21)을 얻는다.

$$\bar{l} = \frac{4V}{S} \tag{3.21}$$

$$\Gamma = \bar{l}\Sigma_r P \tag{3.22}$$

First collision probability 와 First collision probability는 다음과 같이 정의된다.

First Collision Probability(γ): Surface S에 균일한 cosine current distribution의 선원이 있다고 할 때. 영역 내에서 처음으로 충돌을 할 확률

First Escape Collision Probability(p): 어떤 영역에서 태어나서, 그 영역에서 collision을 하지 않고 영역을 둘러싸는 표면 S을 지나서 빠져나갈 확률

위의 두 확률간에는 아래와 같은 관계가 성립한다.

$$\frac{1}{4}\phi S\gamma = \frac{\Sigma}{\Sigma_r}p \tag{3.23}$$

$$\gamma = \frac{4V}{S}\Sigma p = \bar{l}\Sigma p \tag{3.24}$$

아래 FIG 7과 같이 다수의 영역으로 나누어져 있을 때, 위의 두 확률은 식 (3.24)와 유사한 아래와 같은 관계를 가진다.

$$\gamma_i = \frac{4V_i}{S_i}\Sigma_i p_i \tag{3.25}$$

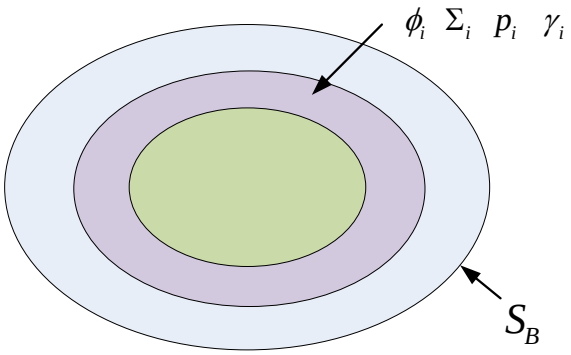


FIG 7 First Collision Probability 와 First Escape Probability

Albedo and Boundary Multiplication

Collision Probability 에서 Boundary condition은 아래 식 (3.26)의 albedo에 의해서 정의 된다.

$$\alpha = \frac{J_{in}}{J_{out}} \tag{3.26}$$

Boundary surface S_B 을 통해서 J_{in} 의 양의 neutron이 영역에 입사했다고 하자. 그러면 이 neutron 중에서 boundary에 다시 도착하는 양은 $(1-\Gamma)J_{in}$ 이다. 이 중에서 $(1-\alpha)(1-\Gamma)J_{in}$ 는 영역을 빠져나갈 것이고, $\alpha(1-\Gamma)J_{in}$ 는 다시 영역으로 재입사 하게 된다. 또 재입사 한 neutron 중에서 $\alpha(1-\Gamma)^2 J_{in}$ 이 또 다시 boundary에 도달할 것이다. 위와 같은 방식으로 아래 Fig 8과 같이 계산해보면, 실질적으로 영역으로 입사하는 neutron의 양은 아래 식 (3.27) 과 같다.

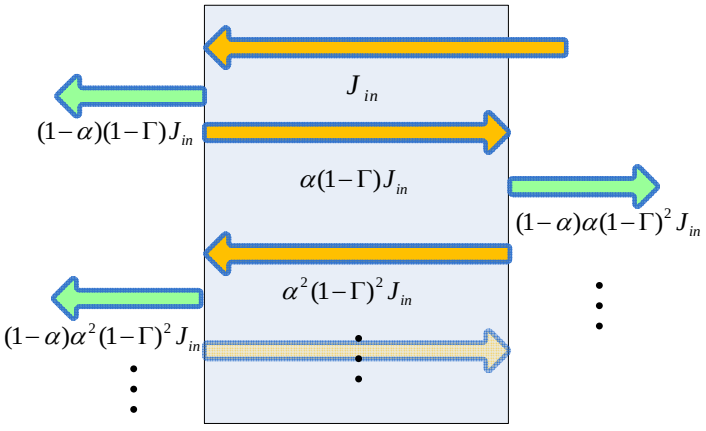


FIG 8 Boundary Multiplication

$$J_{in}(1+\alpha(1-\Gamma)+\alpha^2(1-\Gamma)^2+\cdots) = \underbrace{\frac{1}{1-(1-\Gamma)\alpha}}_{n_\alpha^\Gamma} J_{in} = n_\alpha^\Gamma J_{in} \tag{3.27}$$

Circular Pin-Cell Problem

아래 FIG 9와 같이 정의된 Circular Pin-Cell Problem이 정의 되어 있다고 하자.

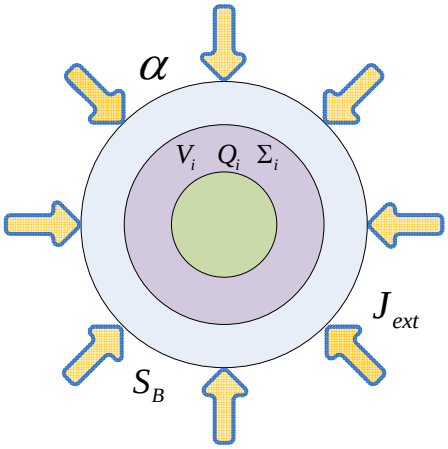


FIG 9 Circular Pin - Cell

영역 i에서의 flux는 크게 Source에 의한 성분과 외부에서 들어오는 neutron에 의한 성분 두 가지로 구성된다. j 영역의 단위 source가 만드는 i 영역의 flux을 $X_i^j(\alpha)$ 라 정의하고, Boundary surface에 하나의 neutron이 입사했을 때 i영역에 생성되는 flux을 $Y_i(\alpha)$ 라고 하자. 그리고 표현을 간단하게 하기 위해서 $\alpha = 0$ 일 때, 각 각의 값을 X_i^j, Y_i 라고 하자. $X_i^j(\alpha)$, $Y_i(\alpha)$ 을 통해서 i 영역의 flux는 아래 식 (3.28)과 같이 표현된다.

$$\phi_i = Y_i(\alpha) j_{ext} + \sum_j X_i^j(\alpha) Q_k \tag{3.28}$$

$X_i^j(\alpha)$, $Y_i(\alpha)$ 을 구하기 이전에, black boundary condition에 대해서 X_i^j, Y_i 을 알고 있다고 가정하자. X_i^j, Y_i 을 이용하여서 $X_i^j(\alpha)$, $Y_i(\alpha)$ 을 적절히 표현 하려고 한다. 먼저 $Y_i(\alpha)$ 와 Y_i 의 관계를 살펴보자. Albedo가 0보다 클 경우는. 경계에서 반사되어서 돌아 오는 neutron이 생긴다. 그러므로 Boundary multiplication에서 살펴보았듯이, 실질적으로 영역 내부로 들어오는 neutron이 증가하는 효과가 생기는 것이다. 그러므로 식 (3.27)의 관계를 이용해서 표현할 수 있다.

$$Y_i(\alpha) = Y_i n_\alpha^{\mathrm{T}} = \frac{Y}{1-(1-\Gamma)\alpha}$$
$$\text{where } \Gamma = \sum_i \Gamma_i \tag{3.29}$$

$X_i^j(\alpha)$ 와 X_i^j 의 관계를 구하기 이전에, j 영역의 unit source에 의해서 태어난 neutron이 흡수되지 않고 처음으로 boundary에 도착하는 수 x_j 을 계산해보자. 이러한 인자를 정의하는 이유는, j

영역에서 태어난 neutron이 boundary에서 반사되어서 i 영역에 flux을 생성하기도 하기 때문이다. x_j 는 escape probability을 이용하면 아래 식과 같이 표현된다.

$$x_j = P_j \underbrace{V_j}_{\substack{\text{Total} \\ \text{source at j}}} = \frac{S_B}{4\Sigma_{rj}}\Gamma_j \tag{3.30}$$

Γ_j 는 unit total incoming current가 있을 때, j 영역에서 remove 되는 양이다. 이것은 unit total incoming current가 있을 때, j 영역에서 생기는 flux에 removal XS을 곱해준 것과 동일하게 된다.

$$\Gamma_j = \Sigma_{rj} Y_j V_j \tag{3.31}$$

식 (3.31)을 식 (3.30)에 대입하면 아래 식 (3.31)와 같다

$$x_j = \frac{S_B}{4} Y_j V_j \tag{3.32}$$

이제, $X_i^j(\alpha)$ 을 X_i^j 와 x_j 을 이용해서 나타내보자. $X_i^j(\alpha)$ 는 j 영역의 source가 경계에 전혀 도달하지 않고 바로 i 영역의 flux에 기여를 하는 것과, 한번 혹은 여려 번의 반사를 거쳐서 i 영역의 flux을 생성하는 두 가지로 나누어 진다. 전자는 단순하게 X_i^j 와 같다. 후자의 경우. 처음으로 boundary에 도달하게 되는 x_j 중에서 αx_j 만 다시 내부로 돌아오게 된다. 이 αx_j 을 external current 로 생각하게 되면, $\alpha x_j Y_i(\alpha)$ 가 i 영역의 flux가 된다.

$$X_i^j(\alpha) = X_i^j + \alpha x_j Y_i(\alpha) \tag{3.33}$$

이제 X_i^j, Y_i 을 구하면, 모든 영역의 flux을 계산할 수 있다. X_i^j, Y_i 는 그것의 정의를 이용해서 계산 할 수 있다. 먼저 X_i^j 을 구하기 위해서 정의에 따라서 $\alpha = 0$ 이고, j 영역에만 unit source가 있다고 하자.

$$Q_i = \delta_{ij} \tag{3.34}$$

그리고 추가적으로 external current도 없다고 하자. 이러한 상황에서 flux는 식 (3.28)에 의해서 아래와 같이 된다.

$$\begin{aligned}\phi_i &= \sum_k X_i^k Q_k = \sum_k X_i^k \delta_{jk} \\ &= X_i^j\end{aligned} \tag{3.35}$$

식 (3.13)과 (3.35)을 이용하면 다음과 같은 식이 나온다

$$\Sigma_i V_i X_i^j = \sum_k P_{ki}(c_k X_k^j + \frac{\delta_{jk}}{\Sigma_k}) = \sum_k P_{ki} c_k X_k^j + \frac{P_{ji}}{\Sigma_j} \tag{3.36}$$

$$\Sigma_i V_i X_i^j - \sum_k P_{ki} c_k X_k^j = \frac{P_{ji}}{\Sigma_j} \quad \text{for } i = 1, \cdots, n \tag{3.37}$$

식 (3.37)을 Matrix로 나타내면 아래 식 (3.38) 과 같고, 이 linear system을 j을 1부터 n까지 n번 풀면 X_i^j 을 결정 할 수 있다.

$$\begin{bmatrix} \Sigma_1 V_1 - P_{11} c_1 & -P_{21} c_1 & \cdots & -P_{n1} c_1 \\ -P_{12} c_1 & \Sigma_2 V_2 - P_{22} c_2 & \cdots & -P_{n2} c_2 \\ \vdots & \vdots & \ddots & \vdots \\ -P_{1n} c_1 & -P_{12} c_2 & \cdots & \Sigma_n V_n - P_{nn} c_n \end{bmatrix} \begin{bmatrix} X_1^j \\ X_2^j \\ \vdots \\ X_n^j \end{bmatrix} = \begin{bmatrix} P_{j1} / \Sigma_1 \\ P_{j2} / \Sigma_2 \\ \vdots \\ P_{jn} / \Sigma_n \end{bmatrix} \tag{3.38}$$

이제 Y_i 을 결정하기 위해서, 모든 영역에서 source가 0이고 black boundary condition에 오직 unit incoming current만 존재한다고 하자. 그러면 각 영역의 flux는 Y_i가 된다. i 영역에서 일어나는 total collision은 다른 영역에서 scattering 된 flux(scattering source)가 i 영역에 flux에 기여하는 양과 external current가 i 영역에서 처음으로 collision을 일으키는 성분 γ_i 으로 구성된다. 이것을 수식으로 표현하면 아래 식 (3.33)이 된다.

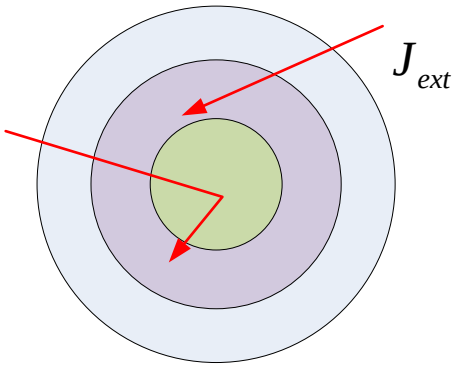


FIG 10 Y_i에 기여 하는 두 성분

$$\Sigma_i V_i Y_i = \sum_j P_{ji} c_j Y_j + \gamma_i \tag{3.39}$$

$$\Sigma_i V_i Y_i - \sum_j P_{ji} c_j Y_j = \gamma_i \quad \text{for } i = 1, \cdots, n \tag{3.40}$$

위 식 (3.40)을 Matrix의 형태로 나타내면 아래 식 (3.41)과 같고, 이 linear system을 풀면 모든 Y_i 가 결정된다.

$$\begin{bmatrix} \Sigma_1 V_1 - P_{11} c_1 & -P_{21} c_1 & \cdots & -P_{n1} c_1 \\ -P_{12} c_1 & \Sigma_2 V_2 - P_{22} c_2 & \cdots & -P_{n2} c_2 \\ \vdots & \vdots & \ddots & \vdots \\ -P_{1n} c_1 & -P_{12} c_2 & \cdots & \Sigma_n V_n - P_{nn} c_n \end{bmatrix} \begin{bmatrix} X_1^k \\ X_2^k \\ \vdots \\ X_n^k \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix} \tag{3.41}$$

하지면 위 linear system을 풀려면 γ_i 을 알아야 한다. γ_i 을 식 (3.25)에 정의 되어 있는데, 이 식에서 아직 결정되지 않은 값은 First Escape Collision Probability(p_j)이다. p_j 값이 작을 경우 아래 식 (3.42)처럼 근사할 수 있다.

$$p_i = 1 - \sum_j \tilde{P}_{ij} \tag{3.42}$$

식 (3.25)와 (3.42)을 결합하면 아래 식 (3.43)와 같이 γ_i 을 표현된다.

$$\begin{aligned} \gamma_i &= \frac{4}{S_B} \left(\Sigma_i \gamma V_i - \sum_j \Sigma_i V_i \tilde{P}_{ij} \right) \\ &= \frac{4}{S_B} \left(\Sigma_i \gamma V_i - \sum_j P_{ij} \right) \end{aligned} \tag{3.43}$$

동일한 linear system의 좌변을 가지고, 우변을 바꾸어 가면서 n+1번 풀게 되면, X_i^j , Y_i 을 결정할 수 있다. Current을 결정하기 위해서 처음으로 boundary에 도달하는 neutron의 양이 필요하다. 이 양은 x_j 을 이용해서 아래 식 (3.44)와 같이 표현된다.

$$x = \sum_j Q_j x_j \tag{3.44}$$

식 (3.44)와 Albedo을 고려해서, boundary을 통해서 빠져나가고 들어오는 current의 양을 결정하면 아래와 같다.

$$J^+(\alpha) = \frac{x + (1-\Gamma) J^{ext}}{1 - \alpha(1-\Gamma)} \tag{3.45}$$

$$J^{-}(\alpha) = \frac{\alpha x + J^{ext}}{1 - \alpha(1 - \Gamma)} \tag{3.46}$$