NSEG 5994 – Monte Carlo Methods for Particle Transport

Homework – 8

Nagendra Krishnamurthy

16th November, 2011

The source codes for the two problems are provided at the end. Only the parts of the code that are modified for the calculation of scalar flux are provided.

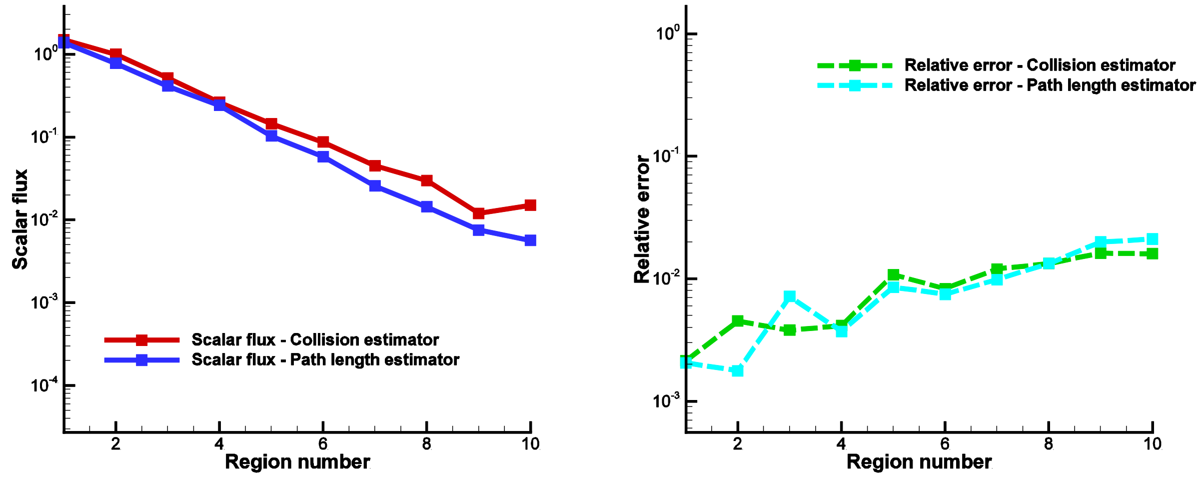
Problem 1:

The following tables provide results obtained from the analog code. The two cases – 10 and 50 regions used for the calculations are provided. For the second case, only the figure is provided. The following conclusions can be made from the results:

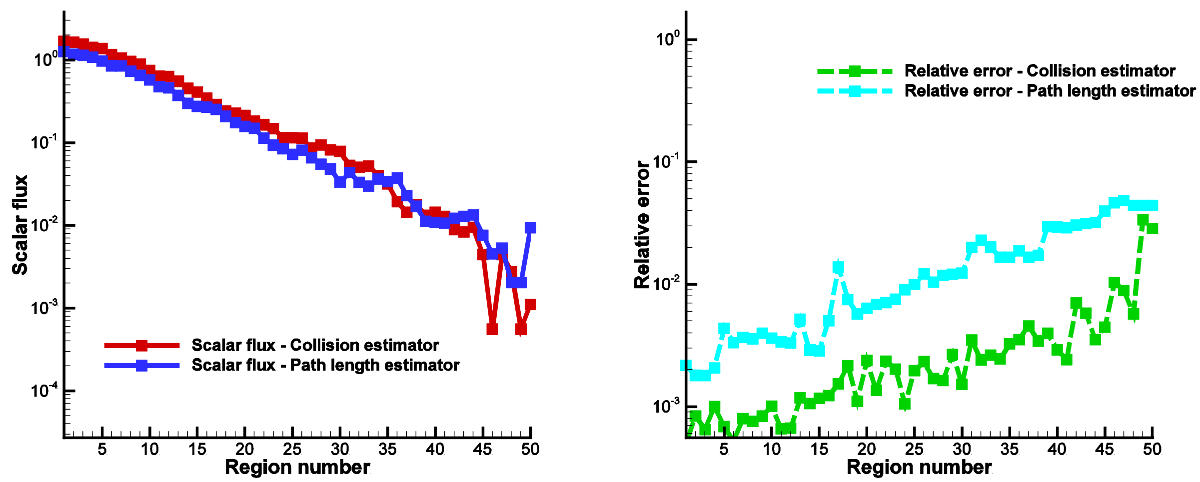
1. It is seen that as the distance from the source is increased, the scalar flux goes down.
2. The relative error on the other hand shows an increasing trend. This is probably due to the lower number of samples used for the scalar flux computation as the distance from the source increases.
3. Both the types of estimators provide similar estimates of scalar flux in most of the regions.

Number of regions – 10:

Region number	Collision estimator		Path length estimator	
	Scalar flux	Relative error	Scalar flux	Relative error
1	1.50E+00	2.15E-03	1.38E+00	2.06E-03
2	1.00E+00	4.51E-03	7.78E-01	1.77E-03
3	5.19E-01	3.81E-03	4.14E-01	7.18E-03
4	2.63E-01	4.15E-03	2.42E-01	3.71E-03
5	1.45E-01	1.08E-02	1.03E-01	8.53E-03
6	8.70E-02	8.29E-03	5.79E-02	7.47E-03
7	4.50E-02	1.20E-02	2.57E-02	9.84E-03
8	3.00E-02	1.33E-02	1.44E-02	1.33E-02
9	1.20E-02	1.62E-02	7.55E-03	2.00E-02
10	1.50E-02	1.60E-02	5.65E-03	2.11E-02



Number of regions – 50:



Problem 2:

Results are provided similar to the analog case.

The following conclusions can be made regarding the results obtained for geometric splitting case in comparison to the analog code results provided in the previous problem.

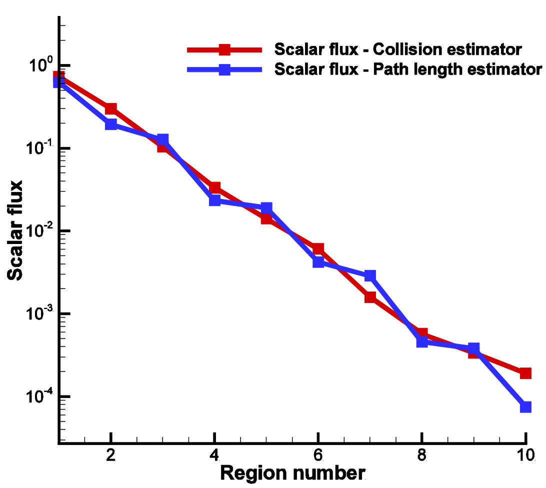
1. The scalar flux values obtained show a decreasing trend as the region under consideration is farther away from the source when 10 regions are considered. This is similar to the analog case. Though the trend appears the same, the values are to be noted here. The scalar flux itself

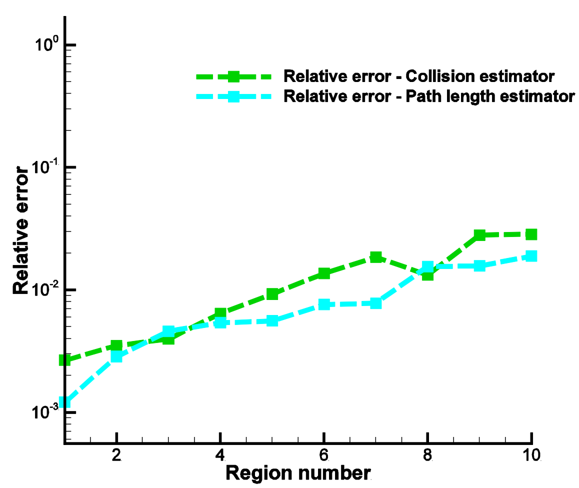
towards to the end of the detector has much lower values (almost 1-2 orders) than the predictions of the analog method.

2. The relative errors obtained however, are slightly higher compared to the analog case throughout the domain. This can be related to the variance reduction technique that is applied. Since higher importance was provided for regions closer to the detector, more samples from those regions have been obtained in the calculation resulting in lesser number of samples in regions closer to the source. Hence, the higher errors in regions closer to the source.

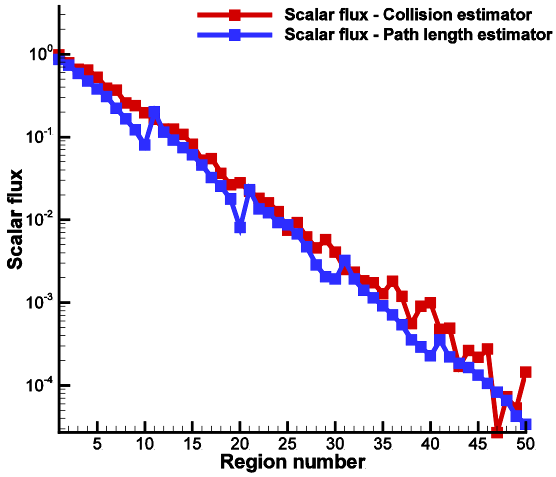
Number of regions – 10:

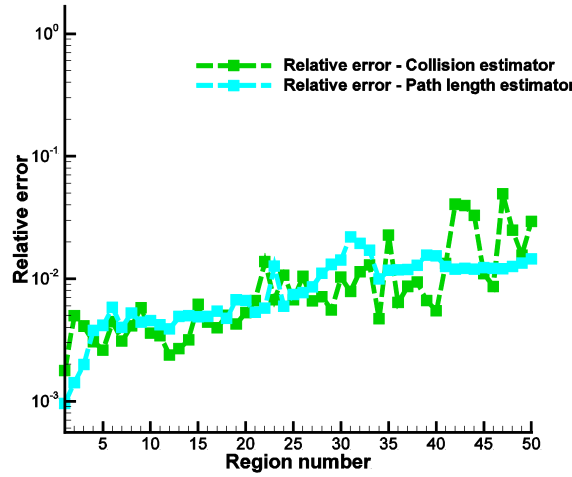
Region number	Collision estimator		Path length estimator	
	Scalar flux	Relative error	Scalar flux	Relative error
1	7.28E-01	2.67E-03	6.23E-01	1.21E-03
2	2.99E-01	3.51E-03	1.94E-01	2.86E-03
3	1.04E-01	3.97E-03	1.27E-01	4.60E-03
4	3.33E-02	6.42E-03	2.34E-02	5.39E-03
5	1.41E-02	9.24E-03	1.90E-02	5.59E-03
6	6.07E-03	1.36E-02	4.22E-03	7.59E-03
7	1.59E-03	1.85E-02	2.88E-03	7.77E-03
8	5.74E-04	1.32E-02	4.58E-04	1.55E-02
9	3.39E-04	2.80E-02	3.82E-04	1.57E-02
10	1.92E-04	2.84E-02	7.52E-05	1.89E-02





Number of regions – 50:





Source code:

Problem 1 code:

```
MODULE allData

  TYPE shield
    REAL :: sigmaTotal, sigmaAbs, xMin, xMax, absRatio, thickness
  END TYPE shield

  TYPE (shield), ALLOCATABLE :: shld(:)

  INTEGER :: nParticles, nShields, parStatus, nParAbs, nParRef,      &
      nParLeak, currentRegion, collStatus, freeFlightFlag,           &
      nTallyingRegions
  REAL :: xInit, scatterAngleInit(3), xLocation, scatterAngle
  REAL :: probAbs(3), probRef(3), probLeak(3),                       &
      relativeErrorLeak(3), relativeErrorRef(3),                     &
      relativeErrorAbs(3), FOMLeak(3), FOMRef(3), FOMAbs(3),         &
      varianceLeak(3), varianceRef(3), varianceAbs(3),               &
      totalThickness
  REAL, ALLOCATABLE :: p(:), fc(:), scalarFluxColl(:),               &
      scalarFluxPL(:), regionWidth(:), regionxMin(:),                &
      regionxMax(:), scalarFluxCollSq(:), scalarFluxPLSq(:),         &
      scalarFluxCollSum(:), &
      scalarFluxPLSum(:), relErrScalFlxColl(:), relErrScalFlxPL(:)

END MODULE allData


PROGRAM multiRegionShield

  USE allData

  IMPLICIT NONE

  INTEGER :: iParticle, iShield, unitProbFile, iScatterAngle, i,     &
      n
  REAL :: pi, maxRelativeError, timeIn, timeOut, totalTime
  CHARACTER(80) :: nameProbFile

  ! Initialization
  xInit = 0.0
  pi = DACOS(-1.0)
```

```
  CALL RANDOM_SEED()

  nShields = 1
  ALLOCATE(shld(nShields))

  nTallyingRegions = 10
  ALLOCATE(p(nTallyingRegions), fc(nTallyingRegions),                &
      scalarFluxColl(nTallyingRegions),                              &
      scalarFluxPL(nTallyingRegions),regionxMin(nTallyingRegions),   &
      regionxMax(nTallyingRegions), regionWidth(nTallyingRegions),   &
      scalarFluxCollSq(nTallyingRegions),                            &
      scalarFluxPLSq(nTallyingRegions),                              &
      scalarFluxCollSum(nTallyingRegions),                           &
      scalarFluxPLSum(nTallyingRegions),                             &
      relErrScalFlxColl(nTallyingRegions),                           &
      relErrScalFlxPL(nTallyingRegions))

  shld(1)%thickness = 1.0
  shld(1)%xMin = xInit
  shld(1)%xMax = shld(1)%xMin + shld(1)%thickness
  DO i = 1, nTallyingRegions
    regionxMin(i) = shld(1)%xMin + (i-1)*1.0/nTallyingRegions
    regionxMax(i) = shld(1)%xMin + i*1.0/nTallyingRegions
    regionWidth(i) = regionxMax(i) - regionxMin(i)
  END DO
  totalThickness = 0.0
  DO n = 1, nShields
    totalThickness = totalThickness + shld(n)%thickness
  END DO

  shld(1)%sigmaTotal = 10.0
! shld(1)%sigmaAbs = 8.0 ! for ratio Es/Et = 0.2
  shld(1)%sigmaAbs = 2.0 ! for ratio Es/Et = 0.8
  shld(1)%absRatio = shld(1)%sigmaAbs/shld(1)%sigmaTotal

  scatterAngleInit(1) = DCOS(0.0)
  scatterAngleInit(2) = DCOS(pi/6.0)
  scatterAngleInit(3) = DCOS(pi/3.0)
  PRINT*, scatterAngleInit(:)

  nParticles = 10000000

  ! Loop over the three scattering angles
  DO iScatterAngle = 1, 1

    nParLeak = 0
    nParAbs = 0
    nParRef = 0
    maxRelativeError = 100.0

    iParticle = 0

    CALL CPU_TIME(timeIn)

    DO WHILE ((iParticle .LT. nParticles) .AND.                    &
        (maxRelativeError .GT. 0.1))
!       (maxRelativeError .GT. 0.0))

      iParticle = iParticle + 1
      IF (MOD(iParticle,1000000) .EQ. 0)                           &
          PRINT*, 'particle number:', iParticle

      ! Initial x and mu values
      xLocation = xInit
      scatterAngle = scatterAngleInit(iScatterAngle)
      parStatus = 0
      currentRegion = 1
      collStatus = 0

      DO WHILE (parStatus .EQ. 0)
```

```
        CALL freeFlightPL

        IF (parStatus .EQ. 0 .AND. collStatus .EQ. 1)                &
            CALL getInteraction

        IF (parStatus .EQ. 0 .AND. collStatus .EQ. 1)                &
            CALL getScatterAngle

      END DO

      DO i = 1, nTallyingRegions
        scalarFluxColl(i) = fc(i)/iParticle/regionWidth(i)
        scalarFluxCollSq(i) = scalarFluxCollSq(i)                    &
            + scalarFluxColl(i)**2.0
        scalarFluxCollSum(i) = scalarFluxCollSum(i)                  &
            + scalarFluxColl(i)
        scalarFluxPL(i) = p(i)/iParticle/regionWidth(i)
        scalarFluxPLSq(i) = scalarFluxPLSq(i) + scalarFluxPL(i)**2.0
        scalarFluxPLSum(i) = scalarFluxPLSum(i) + scalarFluxPL(i)

        IF (MOD(iParticle,1000) .EQ. 0) THEN
        relErrScalFlxColl(i) = DSQRT((scalarFluxCollSq(i)            &
            /(scalarFluxCollSum(i)**2.0)) - 1.0/iParticle)
        relErrScalFlxPL(i) = DSQRT((scalarFluxPLSq(i)                &
            /(scalarFluxPLSum(i)**2.0)) - 1.0/iParticle)
        ! Following for collision estimator relative error
        IF (MINVAL(scalarFluxColl) .NE. 0.0) THEN
        maxRelativeError = MAXVAL(relErrScalFlxColl)
        END IF
        ! Following for path-length estimator relative error
!       IF (MINVAL(scalarFluxPL) .NE. 0.0) THEN
!       maxRelativeError = MAXVAL(relErrScalFlxPL)
!       END IF
        END IF

      END DO

    END DO

    CALL CPU_TIME(timeOut)
    totalTime = timeOut - timeIn
    totalTime = totalTime/60.0 ! Conversion to minutes

  END DO

  unitProbFile = 102
  nameProbFile = 'problem_1b.dat'

  OPEN (UNIT = unitProbFile, FILE = nameProbFile,                    &
      POSITION = 'append', FORM = 'formatted', ACTION = 'write')
  DO i = 1, nTallyingRegions
    WRITE(unitProbFile,502) i, scalarFluxColl(i),                    &
        relErrScalFlxColl(i),                                        &
        1.0/relErrScalFlxColl(i)**2./totalTime,                      &
        scalarFluxPL(i), relErrScalFlxPL(i),                         &
        1.0/relErrScalFlxPL(i)**2./totalTime
  END DO
  CLOSE(unitProbFile)

501 FORMAT (1(i1.1, 1X), 12(e12.5, 1X))
502 FORMAT (1(i2.2, 1X), 6(e12.5, 1X))

END PROGRAM multiRegionShield

SUBROUTINE freeFlightPL

  USE allData

  IMPLICIT NONE

  INTEGER :: i, iOld, iNew
```

```
  REAL :: randNum, pathLength, regionPathLength, oldxLocation

  CALL RANDOM_NUMBER(randNum)

  pathLength = -LOG(randNum)/shld(currentRegion)%sigmaTotal
  oldxLocation = xLocation
  xLocation = xLocation + pathLength*scatterAngle

  IF ((xLocation .GT. shld(currentRegion)%xMax)) THEN
    xLocation = shld(currentRegion)%xMax
    currentRegion = currentRegion + 1
    IF (currentRegion .GT. nShields) THEN
      parStatus = 1
      nParLeak = nParLeak + 1
    END IF
  ELSE IF (xLocation .LT. shld(currentRegion)%xMin) THEN
    xLocation = shld(currentRegion)%xMin
    currentRegion = currentRegion - 1
    IF (currentRegion .LT. 1) THEN
      parStatus = 2
      nParRef = nParRef + 1
    END IF
  ELSE
    collStatus = 1
! Collision estimator
    i = INT((xLocation-shld(1)%xMin)                    &
        *nTallyingRegions/totalThickness) + 1
    fc(i) = fc(i) + 1.0/shld(1)%sigmaTotal
  END IF

  iOld = INT((oldxLocation-shld(1)%xMin)                &
        *nTallyingRegions/totalThickness) + 1
  iNew = INT((xLocation-shld(1)%xMin)                   &
        *nTallyingRegions/totalThickness) + 1
  IF (xLocation .EQ. 1.0) THEN
!   print*, iNew, xLocation, iOld
    iNew = nTallyingRegions
  END IF

  DO i = iOld, iNew
    IF (xLocation .GT. regionxMax(i)) THEN
      regionPathLength = (regionxMax(i)-oldxLocation)/scatterAngle
      p(i) = p(i) + 1.0*regionPathLength
      oldxLocation = regionxMax(i)
    ELSE
      regionPathLength = (xLocation-oldxLocation)/scatterAngle
      p(i) = p(i) + 1.0*regionPathLength
    END IF

  END DO
! Path-length estimator

END SUBROUTINE freeFlightPL
```

Problem 2 code:

```
MODULE allData

  TYPE shield
    REAL :: sigmaTotal, sigmaAbs, xMin, xMax, absRatio, thickness,  &
        importance
  END TYPE shield

  TYPE (shield), ALLOCATABLE :: shld(:,:)

  TYPE particle
    INTEGER :: region, subRegion
    REAL :: scatAngle, xLoc, weight
```

```
    TYPE (particle), POINTER :: next
  END TYPE particle

  TYPE (particle), POINTER :: parList, parListTail

  INTEGER :: nParticles, nShields, parStatus, nParAbs, nParRef,     &
      nParLeak, currentRegion, collStatus, freeFlightFlag,          &
      nSubRegions, currentSubRegion, oldRegion, oldSubRegion,       &
      nTallyingRegions
  REAL :: xInit, scatterAngleInit(3), xLocation, scatterAngle,      &
      parRef, parLeak, parAbs, parWeight, weightCutoff, parKilled,  &
      parAdded, impRatio
  REAL :: probAbs(3), probRef(3), probLeak(3),                      &
      relativeErrorLeak(3), relativeErrorRef(3),                    &
      relativeErrorAbs(3), FOMLeak(3), FOMRef(3), FOMAbs(3),        &
      varianceLeak(3), varianceRef(3), varianceAbs(3),              &
      totalThickness
  REAL, ALLOCATABLE :: p(:), fc(:), scalarFluxColl(:),              &
      scalarFluxPL(:), regionWidth(:), regionxMin(:),               &
      regionxMax(:), scalarFluxCollSq(:), scalarFluxPLSq(:),        &
      scalarFluxCollSum(:),                                         &
      scalarFluxPLSum(:), relErrScalFlxColl(:), relErrScalFlxPL(:)

END MODULE allData


PROGRAM multiRegionShield

  USE allData

  IMPLICIT NONE

  INTEGER :: iParticle, iShield, unitProbFile, iScatterAngle,       &
      rrd, flag, numParLeft, iSubRegion, i, n
  REAL :: pi, probLeakSq, probRefSq, probAbsSq, timeIn, timeOut,    &
      totalTime, maxRelativeError, iParLeak, iParRef, iParAbs
  CHARACTER(80) :: nameProbFile

  TYPE (particle), POINTER :: parCurrent, parTemp

  INTERFACE

    SUBROUTINE insertPar(parCurrent,wt)

    USE allData
    TYPE (particle), POINTER, INTENT(IN) :: parCurrent
    REAL, INTENT (IN) :: wt

    END SUBROUTINE insertPar

  END INTERFACE

  ! Initialization
  xInit = 0.0
  pi = DACOS(-1.0)
  relativeErrorLeak = 100.0
  relativeErrorAbs = 100.0
  relativeErrorRef = 100.0

  CALL RANDOM_SEED()

  nShields = 1
  nSubRegions = 5
  impRatio = 3.4
  ALLOCATE(shld(nShields,nSubRegions))

  nTallyingRegions = 50
  ALLOCATE(p(nTallyingRegions), fc(nTallyingRegions),               &
      scalarFluxColl(nTallyingRegions),                             &
      scalarFluxPL(nTallyingRegions),regionxMin(nTallyingRegions),  &
      regionxMax(nTallyingRegions), regionWidth(nTallyingRegions),  &
```

```
      scalarFluxCollSq(nTallyingRegions),                             &
      scalarFluxPLSq(nTallyingRegions),                               &
      scalarFluxCollSum(nTallyingRegions),                            &
      scalarFluxPLSum(nTallyingRegions),                              &
      relErrScalFlxColl(nTallyingRegions),                            &
      relErrScalFlxPL(nTallyingRegions))

  ALLOCATE(parList)
  NULLIFY(parList%next)

  DO iSubRegion = 1, nSubRegions
    shld(1,iSubRegion)%thickness = 1.0/nSubRegions
    shld(1,iSubRegion)%xMin = xInit + (iSubRegion-1)*1.0/nSubRegions
    shld(1,iSubRegion)%xMax = xInit + iSubRegion*1.0/nSubRegions

    shld(1,iSubRegion)%sigmaTotal = 10.0
    shld(1,iSubRegion)%sigmaAbs = 8.0 ! for ratio Es/Et = 0.2
!   shld(1,iSubRegion)%sigmaAbs = 2.0 ! for ratio Es/Et = 0.8
    shld(1,iSubRegion)%absRatio =                                     &
        shld(1,iSubRegion)%sigmaAbs/shld(1,iSubRegion)%sigmaTotal

    shld(1,iSubRegion)%importance = impRatio**(iSubRegion-1.0)

  END DO
  DO i = 1, nTallyingRegions
    regionxMin(i) = shld(1,1)%xMin + (i-1)*1.0/nTallyingRegions
    regionxMax(i) = shld(1,1)%xMin + i*1.0/nTallyingRegions
    regionWidth(i) = regionxMax(i) - regionxMin(i)
  END DO
  totalThickness = 0.0
  DO n = 1, nShields
    DO iSubRegion = 1, nSubRegions
      totalThickness = totalThickness + shld(n,iSubRegion)%thickness
    END DO
  END DO

  scatterAngleInit(1) = DCOS(0.0)
  scatterAngleInit(2) = DCOS(pi/6.0)
  scatterAngleInit(3) = DCOS(pi/3.0)

  rrd = 5
  weightCutoff = 1.0E-7
  nParticles = 20000000

  ! Loop over the three scattering angles
  DO iScatterAngle = 1, 3

    nParLeak = 0
    nParAbs = 0
    nParRef = 0
    parLeak = 0.0
    parRef = 0.0
    parAbs = 0.0
    parKilled = 0.0
    parAdded = 0.0
    probLeakSq = 0.0; probRefSq = 0.0; probAbsSq = 0.0
    maxRelativeError = 100.0

    iParticle = 0

    CALL CPU_TIME(timeIn)

    DO WHILE ((iParticle .LT. nParticles) .AND.                       &
        (maxRelativeError .GT. 0.1))

      iParticle = iParticle + 1
      IF (MOD(iParticle,100000) .EQ. 0) THEN
        PRINT*, 'particle number:', iParticle
      END IF

      xLocation = xInit
```

```
      scatterAngle = scatterAngleInit(iScatterAngle)
      currentRegion = 1
      currentSubRegion = 1
      oldRegion = 1
      oldSubRegion = 1
      parWeight = 1.0
      iParLeak = 0.0; iParRef = 0.0; iParAbs = 0.0
      parListTail => parList
      parCurrent => parListTail
      CALL insertPar(parCurrent,parWeight)
      parListTail => parCurrent%next
      parCurrent => parCurrent%next

      DO WHILE (ASSOCIATED(parCurrent))
        parStatus = 0
        currentRegion = parCurrent%region
        currentSubRegion = parCurrent%SubRegion
        oldRegion = currentRegion
        oldSubRegion = currentSubRegion
        collStatus = 0
        parWeight = parCurrent%weight
        xLocation = parCurrent%xLoc
        scatterAngle = parCurrent%scatAngle
        DO WHILE (parStatus .EQ. 0)

          CALL freeFlightPL

          IF (parStatus .EQ. 0 .AND. collStatus .EQ. 1) THEN
            CALL getInteraction
          END IF

          IF (parStatus .EQ. 0 .AND. collStatus .EQ. 1) THEN
            CALL getScatterAngle
          END IF

        END DO
        parCurrent => parCurrent%next
      END DO

      DO i = 1, nTallyingRegions
        scalarFluxColl(i) = fc(i)/iParticle/regionWidth(i)
        scalarFluxCollSq(i) = scalarFluxCollSq(i)                   &
            + scalarFluxColl(i)**2.0
        scalarFluxCollSum(i) = scalarFluxCollSum(i)                 &
            + scalarFluxColl(i)
        scalarFluxPL(i) = p(i)/iParticle/regionWidth(i)
        scalarFluxPLSq(i) = scalarFluxPLSq(i) + scalarFluxPL(i)**2.0
        scalarFluxPLSum(i) = scalarFluxPLSum(i) + scalarFluxPL(i)
        IF (MOD(iParticle,1000) .EQ. 0) THEN

        relErrScalFlxColl(i) = DSQRT((scalarFluxCollSq(i)           &
            /(scalarFluxCollSum(i)**2.0)) - 1.0/iParticle)
        relErrScalFlxPL(i) = DSQRT((scalarFluxPLSq(i)               &
            /(scalarFluxPLSum(i)**2.0)) - 1.0/iParticle)

        ! Following for collision estimator relative error
!       IF (MINVAL(scalarFluxColl) .NE. 0.0) THEN
!       maxRelativeError = MAXVAL(relErrScalFlxColl)
!       END IF
        ! Following for path-length estimator relative error
        IF (MINVAL(scalarFluxPL) .NE. 0.0) THEN
        maxRelativeError = MAXVAL(relErrScalFlxPL)
        END IF
        END IF

      END DO

      CALL delParList

    END DO
```

```
    CALL CPU_TIME(timeOut)
    totalTime = timeOut - timeIn
    totalTime = totalTime/60.0 ! Conversion to minutes

  END DO

  unitProbFile = 102
  nameProbFile = 'problem_2b.dat'

  OPEN (UNIT = unitProbFile, FILE = nameProbFile,                   &
      POSITION = 'append', FORM = 'formatted', ACTION = 'write')
  DO i = 1, nTallyingRegions
    WRITE(unitProbFile,502) i, scalarFluxColl(i),                   &
        relErrScalFlxColl(i), scalarFluxPL(i), relErrScalFlxPL(i)
  END DO
  CLOSE(unitProbFile)

501 FORMAT (1(i1.1, 1X), 12(e12.5, 1X))
502 FORMAT (1(i2.2, 1X), 4(e12.5, 1X))

END PROGRAM multiRegionShield

SUBROUTINE freeFlightPL

  USE allData

  IMPLICIT NONE

  INTEGER :: i, iOld, iNew
  REAL :: randNum, pathLength, regionPathLength, oldxLocation

  CALL RANDOM_NUMBER(randNum)

  pathLength = -LOG(randNum)                                        &
      /shld(currentRegion,currentSubRegion)%sigmaTotal
  oldxLocation = xLocation
  xLocation = xLocation + pathLength*scatterAngle

  IF ((xLocation .GT. shld(currentRegion,currentSubRegion)%xMax))   &
      THEN
    xLocation = shld(currentRegion,currentSubRegion)%xMax
    oldSubRegion = currentSubRegion
    currentSubRegion = currentSubRegion + 1
    IF (currentSubRegion .GT. nSubRegions) THEN
      currentSubRegion = 1
      oldRegion = currentRegion
      currentRegion = currentRegion + 1
      IF (currentRegion .GT. nShields) THEN
        parStatus = 1
        nParLeak = nParLeak + 1
        parLeak = parLeak + parWeight
      END IF
    END IF
    IF (parStatus .EQ. 0) CALL geometricSplit
  ELSE IF (xLocation .LT.                                           &
      shld(currentRegion,currentSubRegion)%xMin) THEN
    xLocation = shld(currentRegion,currentSubRegion)%xMin
    oldSubRegion = currentSubRegion
    currentSubRegion = currentSubRegion - 1
    IF (currentSubRegion .LT. 1) THEN
      currentSubRegion = nSubRegions
      oldRegion = currentRegion
      currentRegion = currentRegion - 1
      IF (currentRegion .LT. 1) THEN
        parStatus = 2
        nParRef = nParRef + 1
        parRef = parRef + parWeight
      END IF
    END IF
    IF (parStatus .EQ. 0) CALL geometricSplit
  ELSE
```

```
    collStatus = 1
! Collision estimator
    i = INT((xLocation-shld(1,1)%xMin)                        &
        *nTallyingRegions/totalThickness) + 1
    fc(i) = fc(i) + parWeight/shld(1,1)%sigmaTotal

!   print*, 'collision'
  END IF

  iOld = INT((oldxLocation-shld(1,1)%xMin)                             &
        *nTallyingRegions/totalThickness) + 1
  iNew = INT((xLocation-shld(1,1)%xMin)                                &
        *nTallyingRegions/totalThickness) + 1
  IF (xLocation .EQ. 1.0) THEN
    print*, iNew, xLocation, iOld
    iNew = 10
  END IF

  DO i = iOld, iNew
    IF (xLocation .GT. regionxMax(i)) THEN
      regionPathLength = (regionxMax(i)-oldxLocation)/scatterAngle
      p(i) = p(i) + parWeight*regionPathLength
      oldxLocation = regionxMax(i)
    ELSE
      regionPathLength = (xLocation-oldxLocation)/scatterAngle
      p(i) = p(i) + parWeight*regionPathLength
    END IF

  END DO
! Path-length estimator

END SUBROUTINE freeFlightPL
```