NE583 Test 2

J.R. Powers-Luhn

November 20, 2018

1 Problem 1

2 Problem 2

2.1 Normalization factor

$$\begin{aligned}\int_6^7 \psi(E)\,\mathrm{d}E &= \int_6^7 \frac{1}{E}\,\mathrm{d}E \\ &= \log 7 - \log 6 \\ f &= 0.154151\end{aligned}$$

2.2 Alpha

$$\alpha = \frac{(A-1)^2}{(A+1)^2} = 1$$

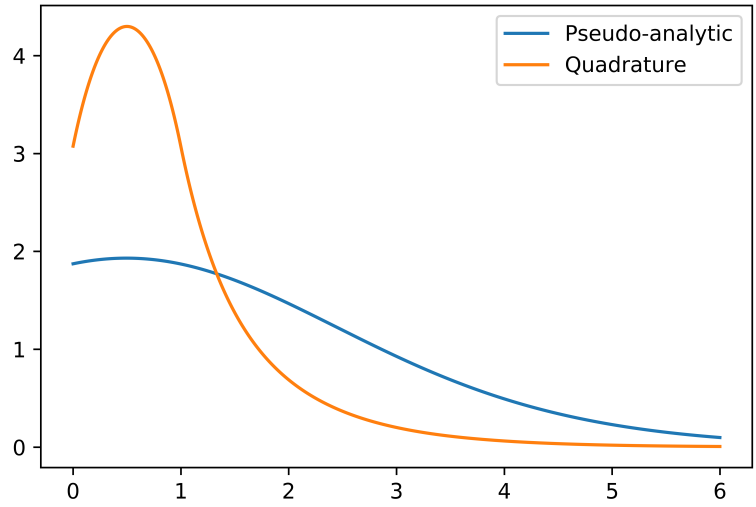
2.3 Scattering Cross section

$$\begin{aligned}\sigma_s^{gg} &= \int_6^7 \mathrm{d}E' \int_6^7 \mathrm{d}E\, \frac{\sigma}{(1-\alpha)E}\frac{\psi(E)}{f} \\ &= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \int_6^7 \frac{\mathrm{d}E}{E^2} \\ &= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \left(\frac{-1}{7} - \frac{-1}{6}\right) \\ &= \frac{\sigma}{f}\left(\frac{1}{6} - \frac{1}{7}\right)(7-6) \\ &= \frac{20\,\mathrm{b}}{0.154151}\left(\frac{1}{6} - \frac{1}{7}\right)(7-6) \\ &= 3.089\,\mathrm{b}\end{aligned}$$

3 Problem 3

4 Problem 4

5 Problem 5



The orange line in the graph above was calculated using the code in the file `test2no5.java` (reproduced below). It is compared with the pseudo-analytic curve (blue line) which was generated by calculating the distance from every point in the source region to every point in the top row (in units of mean free paths) and assuming e^{-r} absorption and $\frac{1}{r^2}$ spatial falloff.

The quadrature solution shows a much more peaked distribution as a result of ray effects.

```
class test02
{

  public static void main(String[] args)
  {
      double totxs = 1.;
      double width = 6. / totxs;
      double height = 6. / totxs;

      int nang = 4;
      int nx = 6000;
      int ny = 6000;
```

```
double[] mu = new double[nang];
double[] eta = new double[nang];
double wt = 0.3333333;

mu[0] = 0.3500212;
mu[1] = 0.8688903;
mu[2] = -mu[0];
mu[3] = -mu[1];
eta[0] = mu[0];
eta[1] = mu[1];
eta[2] = mu[2];
eta[3] = mu[3];

double dx = width / nx;
double dy = height / ny;

double[] tottop = new double[nx];
// Initialize the top row
for (int ix=0; ix<nx; ix++) tottop[ix] = 0.0;

// Set the source matrix
double source[][] = new double[ny][nx];
for (int iy=0; iy<ny; iy++)
{
  for (int ix=0; ix<nx; ix++)
  {
    source[iy][ix] = 0.;
    if (ix*dx < 1. / totxs) source[iy][ix] = 36. / (nx * ny);
  }
}

for (int ieta=0; ieta<nang; ieta++)
{
  double e = eta[ieta];

  for (int imu=0; imu<nang; imu++)
  {
    double m = mu[imu];

    double[] bottom = new double[nx];
    // Initialize the bottom row
    for (int ix=0; ix<nx; ix++) bottom[ix] = 0.;

    double[] average = new double[nx];
    // Initialize the average row
```

```
    for (int ix=0; ix<nx; ix++) average[ix] = 0.;

    for (int iy0=0; iy0<ny; iy0++)
    {
        int iy = iy0;
        // Top to bottom for negative eta
        if (eta[ieta]<0.) iy = ny - 1 - iy0;

        double left = 0.0;
        double right = 0.0;

        double[] top = new double[nx];
        // Initialize the top row
        for (int ix=0; ix<nx; ix++) top[ix] = 0;

        for (int ix0=0; ix0<nx; ix0++)
        {
          int ix = ix0;
          // Right to left for negative mu
          if (mu[imu]<0.0) ix = nx - 1 - ix0;

          average[ix] = favg(m, e, dx, dy, totxs, source[iy][ix],
                             left, bottom[ix]);
          right = fnext(average[ix], left);
          left = right;
        }

        // Set the top values / bottom values for the next row
        for (int ix=0; ix<nx; ix++)
        {
          bottom[ix] = fnext(average[ix], bottom[ix]);
        }
    }
    for (int ix=0; ix<nx; ix++)
    {
        if (eta[ieta]>0.) {
          tottop[ix] += wt * e * wt * Math.abs(m) * bottom[ix];
        }

    }

  }
}

for (int ix=0; ix<nx; ix++)
{
```

```
            System.out.print(tottop[ix] + "\n");
        }
    }
    static double favg(double mu, double eta, double dx, double dy,
                       double totxs, double s, double left, double bottom)
    {
        double num, den;
        den = totxs + 2. * Math.abs(mu) / dx + 2 * Math.abs(eta) / dy;
        num = 2 * Math.abs(mu) / dx * left +
              2 * Math.abs(eta) / dy * bottom + s;
        return num / den;
    }
    static double fnext(double avg, double last)
    {
        return 2.0 * avg - last;
    }
}
```