NE583 Test 2

J.R. Powers-Luhn

November 20, 2018

1 Problem 1

1.1 ^{238}U

Calculate total cross section

$$\sigma_t = 5\,\mathrm{b} + \begin{cases} 0, & \text{if } E < 6.52\,\mathrm{eV} \text{ or } E > 6.62\,\mathrm{eV} \\ 160000 \times (E - 6.52\,\mathrm{eV}), & \text{if } 6.52\,\mathrm{eV} < E < 6.57\,\mathrm{eV} \\ 160000 \times (6.62\,\mathrm{eV} - E), & \text{if } 6.57\,\mathrm{eV} < E < 6.62\,\mathrm{eV} \end{cases}$$

Calculate flux scaling

$$f = \int_6^7 \frac{\mathrm{d}E}{E\sigma_t} = 0.0277995$$

Calculate group cross section

$$\int_6^7 \mathrm{d}E\, \frac{\sigma_t \psi(E)}{fE} = \int_6^7 \mathrm{d}E\, \frac{\sigma_t}{fE^2\sigma_t} = \frac{1}{f}\left(\frac{1}{6} - \frac{1}{7}\right) = 0.8565\,\mathrm{b}$$

1.2 H

Calculate flux scaling

$$f = \int_6^7 \frac{\mathrm{d}E}{E\sigma_t} = \frac{1}{20}\ln\frac{7}{6} = 0.007708$$

Calculate group cross section

$$\frac{1}{f}\int_6^7 \mathrm{d}E\, \frac{\sigma_t}{E^2\sigma_t} = 3.089\,\mathrm{b}$$

1.3 50/50 Mix of U and H

$$\sigma_t = 0.5\sigma_t^H + 0.5\sigma_t^U$$

Plug this in to the flux equation and integrate to get a scaling factor of $f = 0.011136$. Then integrate (numerically) to get $\sigma_t = 2.1380\,\mathrm{b}$.

Problem 1

November 20, 2018

```
In [1]: import numpy as np
        from scipy.integrate import trapz
        import matplotlib.pyplot as plt
```

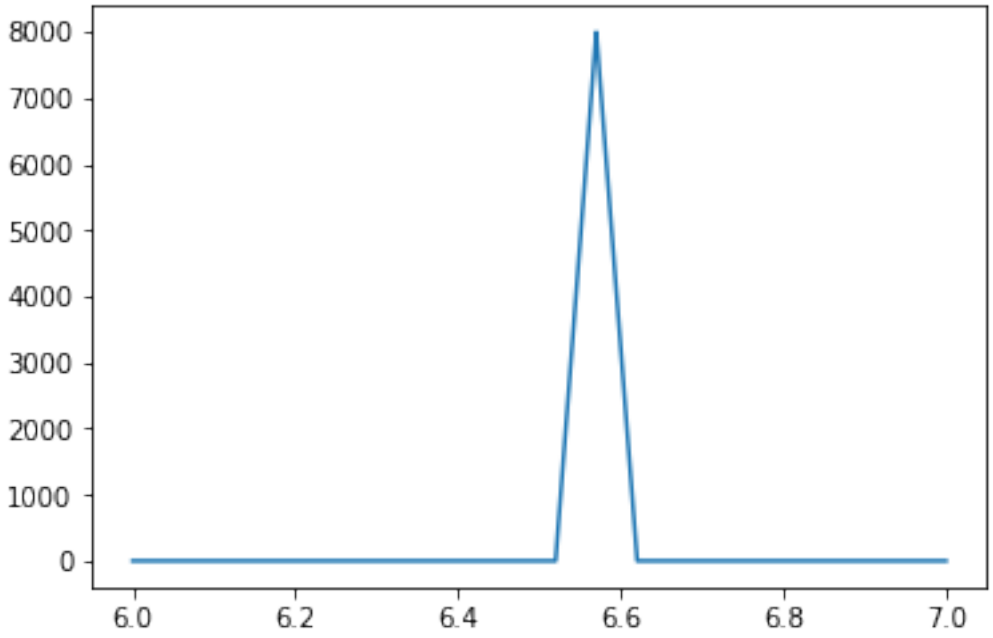
```
In [2]: e = np.linspace(6, 7, 10000)
```

1 Pure Uranium

```
In [3]: @np.vectorize
        def absorption_uranium(energy):
            if energy < 6.52:
                return 0.0
            elif energy < 6.57:
                return 160000 * (energy - 6.52)
            elif energy < 6.62:
                return 160000 * (6.62 - energy)
            else:
                return 0.0
```

```
In [4]: plt.plot(e, absorption_uranium(e))
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x1514b57b38>]
```




```
In [5]: def total_uranium(energy):
            scattering = 5
            return absorption_uranium(energy) + scattering
```

scale = $\int_6^7 \frac{dE}{E\sigma_t(E)}$ performed numerically:

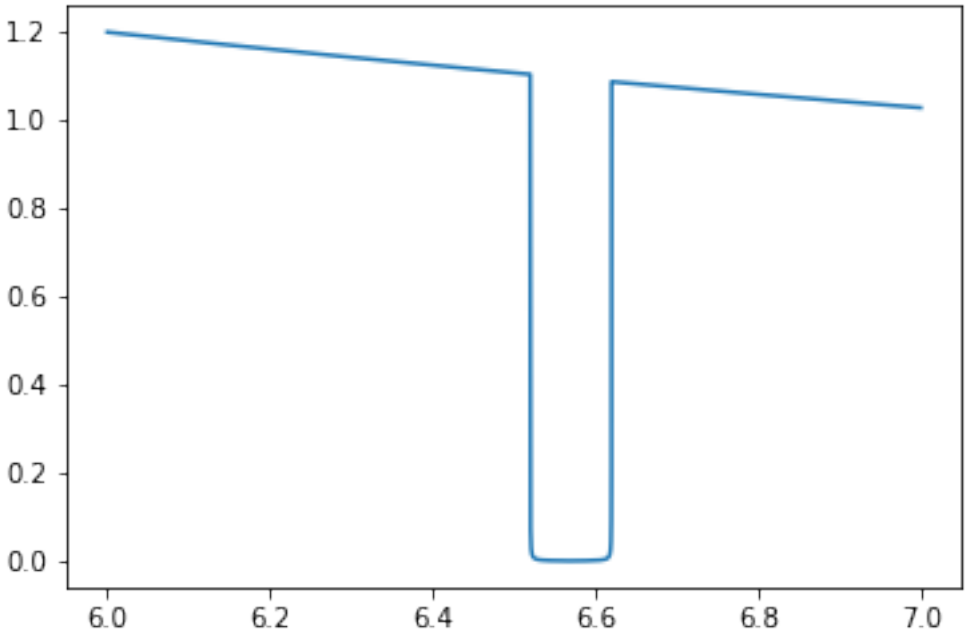
```
In [6]: f = lambda e: 1 / e / total_uranium(e)
        trapz(f(e), e)

Out[6]: 0.02779948909719237

In [7]: def flux_uranium(energy):
            scale = 0.02779948909719237
            return (1 / (energy * total_uranium(energy))) / scale

In [8]: plt.plot(e, flux_uranium(e))

Out[8]: [<matplotlib.lines.Line2D at 0x15172ce400>]
```




```
In [9]: def integrand_uranium(energy):
            return total_uranium(energy) * flux_uranium(energy) / energy

In [10]: trapz(integrand_uranium(e), e)

Out[10]: 0.8564734311892805
```

2 Hydrogen

```
In [11]: def total_hydrogen(energy):
             return 20
```

$$\text{scale} = \frac{1}{\sigma_t^H}(\ln(7) - \ln(6)) = \frac{20}{0.15415} = 0.007708$$

```
In [12]: def flux_hydrogen(energy):
             scale = 0.007707533991670044
             return (1 / energy / total_hydrogen(energy)) / scale

In [13]: def integrand_hydrogen(energy):
             return total_hydrogen(energy) * flux_hydrogen(energy) / energy

In [14]: trapz(integrand_hydrogen(e), e)

Out[14]: 3.089123426262357
```

3 Mixed Hydrogen and Uranium (50/50)

```
In [15]: def total(energy):
             return 0.5 * total_hydrogen(energy) + 0.5 * total_uranium(energy)

In [16]: def flux(energy):
             scale = 0.011136277967125305
             return 1 / total(energy) / energy / scale

In [17]: def integrand_mixed(energy):
             return total(energy) * flux(energy) / energy

In [18]: trapz(integrand_mixed(e), e)

Out[18]: 2.138014503828651

In [ ]:
```

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

$$\alpha = \frac{(A-1)^2}{(A+1)^2} = 0$$

2.3 Scattering Cross section

$$\begin{aligned}\sigma_s^{gg} &= \int_6^7 \mathrm{d}E' \int_{E'}^7 \mathrm{d}E\, \frac{\sigma}{(1-\alpha)E}\frac{\psi(E)}{f} \\ &= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \int_{E'}^7 \frac{\mathrm{d}E}{E^2} \\ &= \frac{\sigma}{f}\int_6^7 \mathrm{d}E' \left(\frac{-1}{7} - \frac{-1}{E'}\right) \\ &= \frac{\sigma}{f}\left(\int_6^7 \frac{\mathrm{d}E'}{E'} - \int_6^7 \frac{\mathrm{d}E'}{7}\right) \\ &= \frac{20\,\mathrm{b}}{0.154151}\left(\ln\frac{7}{6} - \frac{1}{7}\right) \\ &= 1.465\,26\,\mathrm{b}\end{aligned}$$

Problem 3

November 20, 2018

1 Problem 3

```
In [1]: from scipy.integrate import trapz
        from scipy.special import legendre
        from scipy.optimize.zeros import newton
        from numpy.polynomial.legendre import leggauss
        import numpy as np
        import matplotlib.pyplot as plt
```

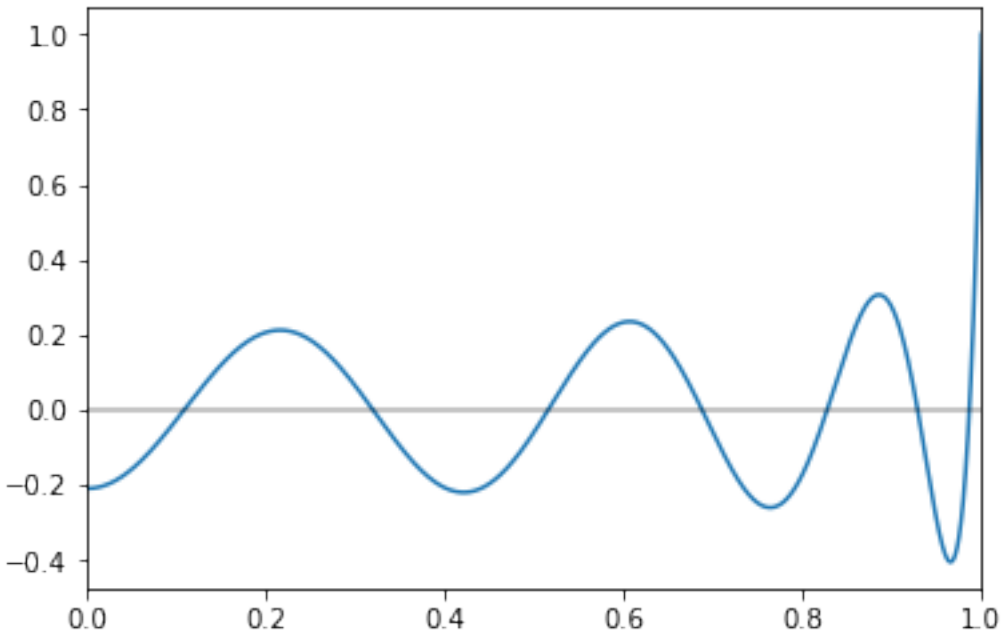
Plot the 14th Legendre polynomial to eyeball the starting guesses for the zeros

```
In [2]: x = np.linspace(-1, 1, 10000)
```

```
In [3]: l = legendre(14)
```

```
In [4]: plt.plot(x, l(x))
        plt.xlim(0, 1)
        plt.axhline(y=0, alpha=0.3, color='black')
```

```
Out[4]: <matplotlib.lines.Line2D at 0x151fc5e630>
```



This has seven positive and seven negative zeros

```
In [5]: guesses = [
            0.1,
            0.33,
            0.52,
            0.7,
            0.8,
            0.9,
            1.0
        ]
```

`newton` uses the Newton-Raphson method to find zeros of a function

```
In [6]: zeros = np.array([newton(l, g) for g in guesses])
        print(zeros)
```

```
[0.10805495 0.31911237 0.51524864 0.6872929  0.82720132 0.92843488
 0.98628381]
```

Now I can construct the matrix of integrals for x^n
Calculate the numerical integral of x^n for even n's

```
In [7]: integrals = np.array([0.5*trapz(x**n, x) for n in range(15)[::2]])
        print(integrals)
```

```
[1.         0.33333334 0.20000001 0.14285716 0.11111114 0.09090912
 0.07692312 0.06666671]
```

Create a matrix where each column j and row i is μ_j^{2i}

```
In [8]: functions = np.array([zeros**n for n in range(15)[::2]])
```

```
In [9]: np.set_printoptions(precision=1)
        print(functions)
```

```
[[1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00 1.0e+00]
 [1.2e-02 1.0e-01 2.7e-01 4.7e-01 6.8e-01 8.6e-01 9.7e-01]
 [1.4e-04 1.0e-02 7.0e-02 2.2e-01 4.7e-01 7.4e-01 9.5e-01]
 [1.6e-06 1.1e-03 1.9e-02 1.1e-01 3.2e-01 6.4e-01 9.2e-01]
 [1.9e-08 1.1e-04 5.0e-03 5.0e-02 2.2e-01 5.5e-01 9.0e-01]
 [2.2e-10 1.1e-05 1.3e-03 2.4e-02 1.5e-01 4.8e-01 8.7e-01]
 [2.5e-12 1.1e-06 3.5e-04 1.1e-02 1.0e-01 4.1e-01 8.5e-01]
 [3.0e-14 1.1e-07 9.3e-05 5.2e-03 7.0e-02 3.5e-01 8.2e-01]]
```

```
In [10]: np.set_printoptions(precision=8)
```

Get the official values to compare with

```
In [11]: mus, wts = leggauss(14)
```

Compare the official μ values (stored in the variable `mus`) to my calculated values (stored in `zeros`)

```
In [12]: mus[7:]

Out[12]: array([0.10805495, 0.31911237, 0.51524864, 0.6872929 , 0.82720132,
                0.92843488, 0.98628381])

In [13]: zeros

Out[13]: array([0.10805495, 0.31911237, 0.51524864, 0.6872929 , 0.82720132,
                0.92843488, 0.98628381])
```

Compare the official weights (stored in `wts`) to my calculated values (stored in `weights`)

```
In [14]: wts[7:]

Out[14]: array([0.21526385, 0.20519846, 0.1855384 , 0.15720317, 0.12151857,
                0.08015809, 0.03511946])

In [15]: weights = np.linalg.inv(functions.T @ functions) @ functions.T @ integrals
         weights

Out[15]: array([0.2152639 , 0.20519833, 0.18553859, 0.15720293, 0.12151886,
                0.08015776, 0.03511963])
```

Calculate the fractional error between my calculated weights and the official ones

```
In [16]: np.abs(weights - wts[7:]) / wts[7:]

Out[16]: array([2.26432997e-07, 6.31964038e-07, 1.02687197e-06, 1.53383609e-06,
                2.34609716e-06, 4.08863853e-06, 4.96849261e-06])
```

Pretty close! Within ~10^{-4}%

```
In [ ]:
```

Problem 4

November 20, 2018

```
In [1]: import numpy as np
        import numpy.linalg as la
```

Caculate μ_n

```
In [2]: mu1 = .2182179
        quadrature = 8
```

```
In [3]: @np.vectorize
        def mu_n(n, mu1=mu1, quad=quadrature):
            c = 2 * (1 - 3 * mu1 ** 2) / (quad - 2)
            mu_n_squared = mu1**2 + (n-1) * c
            return mu_n_squared ** 0.5
```

```
In [4]: inds = np.arange(1, 5)
```

```
In [5]: mus = mu_n(inds)
        print(mus)
```

```
[0.2182179  0.57735027 0.78679579 0.95118973]
```

```
In [6]: integrals = np.array([1 / (2 * (l + 1)) for l in [0, 2, 4, 6, 8]])
```

```
In [7]: integrals
```

```
Out[7]: array([0.5       , 0.16666667, 0.1       , 0.07142857, 0.05555556])
```

```
In [8]: ells = np.arange(2, 13, 2)
        ells
```

```
Out[8]: array([ 2,  4,  6,  8, 10, 12])
```

```
In [9]: def row(l, mus):
            m1 = mus[0]
            m2 = mus[1]
            m3 = mus[2]
            m4 = mus[3]
            r = [
                2*m1**l + m4**l,
```

```
            2*m1**l + 2*m2**l + 2*m3**l,
            m2**l,
        ]
        return r
```

```
In [10]: mat = np.array([row(n, mus) for n in [0, 2, 4, 6, 8]])
```

```
In [11]: mat
```

```
Out[11]: array([[3.        , 6.        , 1.        ],
                [1.        , 2.        , 0.33333333],
                [0.82312924, 0.99319727, 0.11111111],
                [0.7408487 , 0.54875282, 0.03703704],
                [0.67010657, 0.3184167 , 0.01234568]])
```

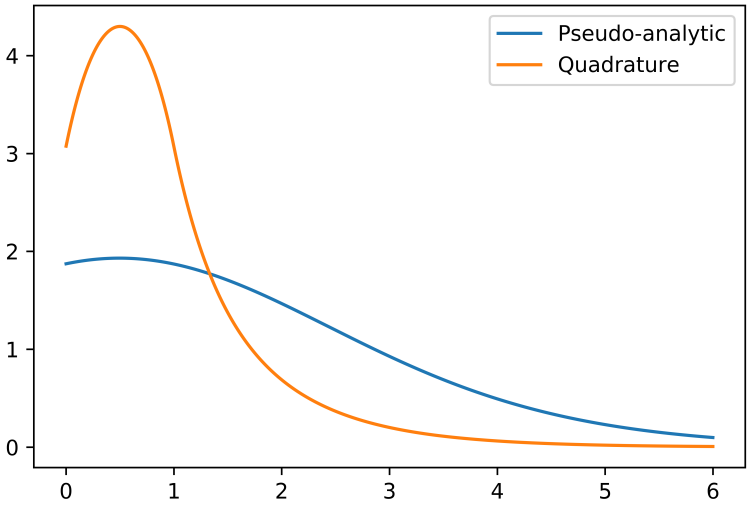
```
In [12]: 2*la.inv(mat.T @ mat) @ mat.T @ integrals
```

```
Out[12]: array([0.12098766, 0.09074074, 0.09259259])
```

These match the values in the book, table 4-1

```
In [ ]:
```

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;
    }
}
```