CHAPTER

16

Gauss Quadrature and Multi-dimensional Integrals

OUTLINE

They're two, they're four, they're six, they're eight
Shunting trucks and hauling freight

–Thomas and Friends "Roll Call"

CHAPTER POINTS

- Gauss quadrature is designed to integrate functions with a small number of points.
- The idea behind Gauss quadrature is to exactly integrate the highest degree polynomial possible given a number of function evaluations.
- Multidimensional integrals can be found by applying 1-D integrals.

16.1 GAUSS QUADRATURE RULES

In the last chapter we saw that we can approximate integrals by fitting the integrand with an interpolating polynomial and then integrating the polynomial exactly. In this chapter

we take a different approach that guarantees the maximum accuracy for a given number of points. These types of methods are called Gauss quadrature rules. We will discuss the rule for finite intervals.

Gauss quadrature rules re-write an integral as the sum of the function evaluated at a given number of points multiplied by a weight function:

$$\int_a^b f(x)\,dx \approx \sum_{\ell=1}^{L} w_\ell f(x_\ell),$$

where the weights w_ℓ and quadrature points x_ℓ are chosen to give the integral certain properties. There are several types of Gauss-quadrature rules, but the type we will cover in detail is known as Gauss–Legendre quadrature. In particular this quadrature rule is for integrals of the form

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{\ell=1}^{L} w_\ell f(x_\ell).$$

The integral does not need to be limited just to the range, $[-1, 1]$, however. If we want to integrate $f(x)$ from $[a, b]$, we define a variable

$$x = \frac{a+b}{2} + \frac{b-a}{2}z, \qquad dx = \frac{b-a}{2}dz,$$

to make the transformation

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z + \frac{a+b}{2}\right)dz.$$

We still have not shown how to pick the weights, w_ℓ, and abscissas, x_ℓ. These are chosen so that the rule is as accurate as possible with L points. It turns out that we can pick the weights and abscissas such that the Gauss–Legendre quadrature formula is exact for polynomials of degree $2L-1$ or less. This should not be a complete surprise because the integral of a $2L-1$ degree polynomial is a degree $2L$ polynomial. Such a polynomial has $2L+1$ coefficients, only $2L$ of these depend on the original polynomial because the constant term is determined by the integration bounds. Therefore, the integral has $2L$ degrees of freedom, the exact number of degrees of freedom we have with our L weights and abscissas.

The weights and abscissas are given for L up to 8 in Table 16.1. Notice that the odd L sets all have $x = 0$ in the set. Also, the sum of the w_ℓ adds up to 2 because the range of integration has a length of 2.

16.1.1 Where Did These Points Come From?

One way to derive the Gauss–Legendre quadrature rules is by looking at the integral of generic monomials of degree 0 up to $2L-1$ and setting each equal to the L point Gauss–

TABLE 16.1 The Abscissae and Weights for Gauss–Legendre Quadrature up to $L = 8$

L	x_ℓ	w_ℓ
1	0	2
2	±0.57735026918962576450914878	1
3	0	0.888888889
	±0.77459666924148337703585310	0.555555556
4	±0.33998104358485626480266575	0.652145155
	±0.86113631159405257522394648	0.347854845
5	0	0.568888889
	±0.53846931010568309103631442	0.47862867
	±0.90617984593866399279762687	0.236926885
6	±0.23861918608319690863050172	0.467913935
	±0.66120938646626451366139959	0.360761573
	±0.93246951420315202781230155	0.171324492
7	0	0.417959184
	±0.40584515137739716690660641	0.381830051
	±0.74153118559939443986386477	0.279705391
	±0.94910791234275852452618968	0.129484966
8	±0.18343464249564980493947614	0.362683783
	±0.52553240991632898581773904	0.313706646
	±0.79666477413626739591553852	0.222381034
	±0.96028985649753623168356087	0.101228536

Legendre quadrature rule:

$$\int_{-1}^{1} dx\, a_0 x^0 = a_0 \sum_{\ell=1}^{L} w_\ell x_\ell^0,$$
$$\int_{-1}^{1} dx\, a_1 x^1 = a_1 \sum_{\ell=1}^{L} w_\ell x_\ell^1,$$

and continuing until

$$\int_{-1}^{1} dx\, a_{2L-1} x^{2L-1} = a_{2L-1} \sum_{\ell=1}^{L} w_\ell x_\ell^{2L-1}.$$

Notice that the a_i constants cancel out of each equation so they do not matter. This system is $2L$ equations with L weights, w_ℓ, and L abscissas, x_ℓ. We could solve these equations to get the weights and abscissas, though this is not how it is done in practice generally—this is accomplished by using the theory of orthogonal polynomials.

BOX 16.1 NUMERICAL PRINCIPLE

Gauss quadrature rules pick the values of the points and weights in the quadrature rule so that the highest degree polynomial can be exactly integrated. An L point quadrature rule has $2L$ degrees of freedom (L weights and L abscissas), and can therefore integrate a polynomial of degree $2L-1$ exactly.

16.1.2 Code for Gauss–Legendre Quadrature

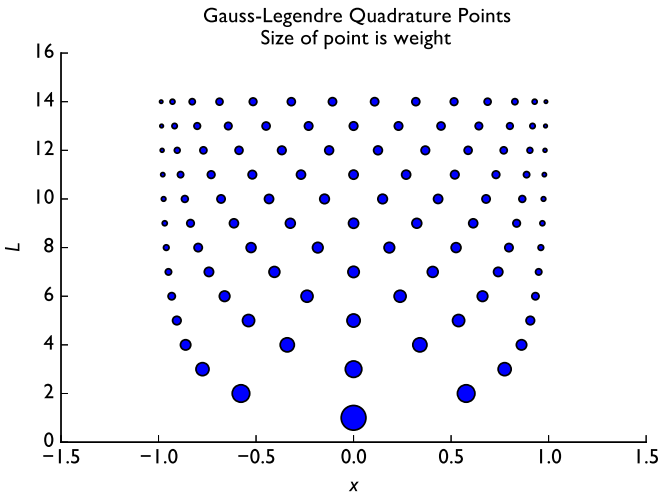
We will now write a function that will compute the integral of a function from $[-1, 1]$ using these quadrature rules. For values of L beyond 2, we will use a NumPy function that generates the points and weights. The NumPy documentation asserts that the rules for $L > 100$ have not been tested and may be inaccurate.

```
In [1]: def GLQuad(f, L=8,dataReturn = False):
            """Compute the Gauss-Legendre Quadrature estimate
            of the integral of f(x) from -1 to 1
            Inputs:
            f:   name of function to integrate
            L:   Order of integration rule (8 or less)

            Returns:
            G-L Quadrature estimate"""
            assert(L>=1)
            if (L==1):
                weights = np.ones(1)*2
                xs = np.array([0])
            elif (L==2):
                weights = np.ones(2)
                xs = np.array([-np.sqrt(1.0/3.0),np.sqrt(1.0/3.0)])
            else: #use numpy's function
                xs, weights = np.polynomial.legendre.leggauss(L)

            quad_estimate = np.sum(weights*f(xs))
            if (dataReturn):
                return quad_estimate, weights, xs
            else:
                return quad_estimate
```

The weights and abscissas are shown in the following figure where the size of a point is proportional to the weight:



As a simple demonstration of the Gauss–Legendre quadrature, let's show that it integrates polynomials of degree $2L-1$ exactly. Consider the integral

$$\frac{L}{2^{2L-1}}\int_{-1}^{1}(x+1)^{2L-1}\,dx = 1.$$

In the following code we show that we integrate this function exactly (to floating point precision) using Gauss–Legendre quadrature:

```
In [2]: L = np.arange(1,12)
        for l in L:
            f = lambda x: (x+1)**(2*l-1)*l/(2**(2*l - 1))
            integral = 1.0
            GLintegral = GLQuad(f,l)
            print("L =", l,"\t Estimate is",GLintegral,
                  "Exact value is",integral,
                  "\nAbs. Relative Error is", np.abs(GLintegral-integral)/integral)
```

```
L = 1    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 0.0
L = 2    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.11022302463e-16
L = 3    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 2.22044604925e-16
L = 4    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 7.77156117238e-16
L = 5    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.4408920985e-16
L = 6    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.99840144433e-15
L = 7    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 3.99680288865e-15
L = 8    Estimate is 1.0 Exact value is 1
```

```
Abs. Relative Error is 2.22044604925e-15
L = 9    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.33226762955e-15
L = 10   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.4408920985e-16
L = 11   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.21884749358e-15
```

As mentioned earlier, we are generally interested in integrals not just over the domain $x \in [-1, 1]$. We'll now make a function that does Gauss–Legendre quadrature over a general range using the formula

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z + \frac{a+b}{2}\right)dz.$$

```
In [3]: def generalGL(f,a,b,L):
            """Compute the Gauss-Legendre Quadrature estimate
            of the integral of f(x) from a to b
            Inputs:
            f:   name of function to integrate
            a:   lower bound of integral
            b:   upper bound of integral
            L:   Order of integration rule
            Returns:
            G-L Quadrature estimate"""
            assert(L>=1)
            #define a re-scaled f
            f_rescaled = lambda  z: f(0.5*(b-a)*z + 0.5*(a+b))
            integral = GLQuad(f_rescaled,L)
            return integral*(b-a)*0.5
```

We can show that this version integrates polynomials of degree $2L - 1$ exactly. Consider the integral

$$\frac{2L}{9^L - 4^L}\int_{-3}^{2} (x+1)^{2L-1}\,dx = 1.$$

```
In [4]: L = np.arange(1,12)
        for l in L:
            f = lambda x: (x+1)**(2*l-1)*(2*l/(9**l-4**l))
            integral = 1.0
            GLintegral = generalGL(f,-3,2,l)
            print("L =", l,"\t Estimate is",GLintegral,
                  "Exact value is",integral,
                  "\nAbs. Relative Error is", np.abs(GLintegral-integral)/integral)

L = 1    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 0.0
L = 2    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.11022302463e-16
L = 3    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.4408920985e-16
```

```
L = 4    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 6.66133814775e-16
L = 5    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.7763568394e-15
L = 6    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 2.44249065418e-15
L = 7    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 6.66133814775e-15
L = 8    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 3.33066907388e-15
L = 9    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 3.10862446895e-15
L = 10   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.4408920985e-16
L = 11   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 6.43929354283e-15
```

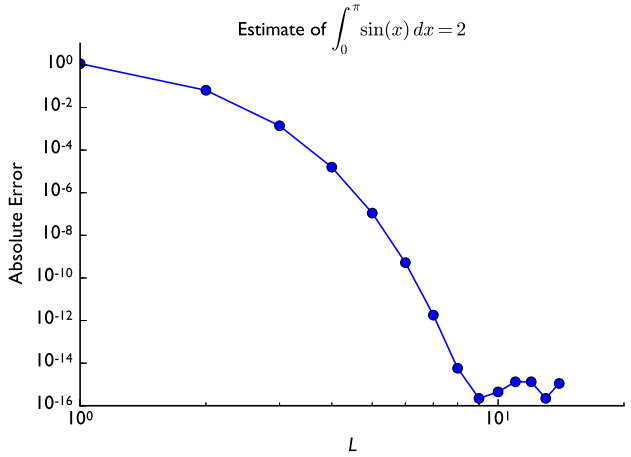
So far we have only used Gauss–Legendre quadrature on polynomials, below we test it on two functions that are not polynomials:

$$\int_0^{\pi} \sin(x)\,dx = 2,$$

and

$$\int_0^{1} 4\sqrt{1-x^2}\,dx = \pi.$$

For the integral of $\sin(x)$ we get exponential convergence:



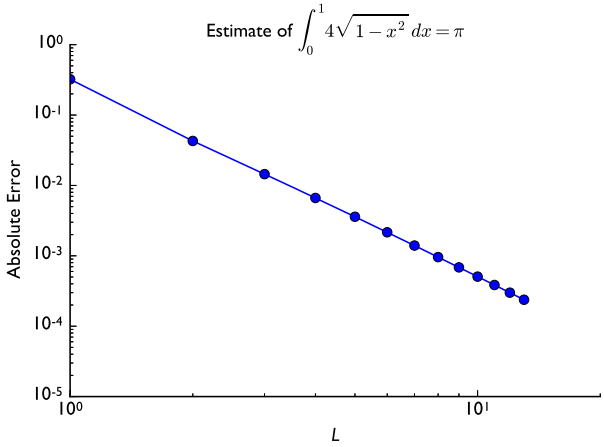
Notice that we get to the smallest error possible by evaluating the integral at only 8 points. This is much better than we saw with trapezoid or Simpson's rules. The approximation converges exponentially to the exact answer. What this means is that it error decreases faster than

a polynomial of the form L^{-n}. Exponential convergence can be seen in the plot of the error because the error goes to zero faster than linearly.

This exponential convergence will only be obtained on smooth solutions without singularities in the function or its derivatives. In the last chapter we discussed the integral

$$\int_0^1 4\sqrt{1-x^2}\,dx = \pi.$$

This integral has a singularity in its derivative at $x = 1$. Gauss–Legendre quadrature will not have exponential convergence on this function.




```
Slope of line from L = 8 to 11 is -2.81212265648
```

Not quite as impressive, but still pretty good considering that we only evaluated the function at 12 points. The difficulty in the obtaining a highly accurate solution is the same as with Newton–Cotes: the function cannot be described well by a polynomial near $x = 1$.

There is a big difference between the exponential convergence we saw in the integral of the sine function and the polynomial convergence in this problem. Even at a high rate of convergence (order 2.8), the error converges slowly in that we are still only accurate to 3 digits at $L = 13$.

16.2 MULTI-DIMENSIONAL INTEGRALS

Up to this point we have only dealt with integrals in one dimension. In all likelihood, the integrals that you will want to evaluate numerically will be multi-dimensional. We spent the effort we did on learning 1-D integrals because these methods are crucial for multi-D quadrature.

If we want to do a multi-dimensional integral, we can do this rather simply by defining multi-dimensional integrals in terms of 1-D integrals. Consider the 2-D integral:

$$\int_a^b dx \int_c^d dy\, f(x,y).$$

If we define an auxiliary function,

$$g(x) = \int_c^d dy\, f(x,y),$$

then the 2-D integral can be expressed as a 1-D integral of $g(x)$:

$$\int_a^b dx \int_c^d dy\, f(x,y) = \int_a^b dx\, g(x).$$

We could do the same thing with a 3-D integral. In this case we define

$$\int_a^b dx \int_c^d dy \int_e^f dz\, f(x,y,z) = \int_a^b dx\, h(x),$$

where

$$g(x,y) = \int_e^f dz\, f(x,y,z),$$

and

$$h(x) = \int_c^d dy\, g(x,y).$$

What this means in practice is that we need to define intermediate functions that involve integrals over one dimension. We define a general 2-D integral where the 1-D integrals use Gauss–Legendre quadrature.

```
In [5]: def GLQuad2D(f,L):
            """Compute the Gauss-Legendre Quadrature estimate
            of the integral of f(x,y) from x = -1 to 1, y = -1 to 1
            Inputs:
            f:   name of function to integrate
            L:   Order of integration rule (8 or less)
            Returns:
            G-L Quadrature estimate"""
            assert(L>=1)
            #get weights and abscissas for GL quad
            temp, weights, xl = GLQuad(lambda x: x,L,dataReturn=True)
            estimate = 0
            for l in range(L):
                f_onevar = lambda y: f(x,y)
                #set x so that when f_onevar is called it knows what x is
                x = xl[l]
```

```
            g = lambda x: GLQuad(f_onevar,L)
            estimate += weights[l] * g(xl[l])
        return estimate
```

This approach to computing 2-D integrals requires L^2 function evaluations because at each of the L quadrature points in y it evaluates the integrand at L points.

The next step is to generalize this to any rectangular domain in a similar fashion as we did in 1-D.

```
In [6]: def generalGL2D(f,a,b,c,d,L):
            """Compute the Gauss-Legendre Quadrature estimate
            of the integral of f(x,y) from x = a to b
            and y = a to b
            Inputs:
            f:   name of function to integrate
            a:   lower bound of integral in x
            b:   upper bound of integral in x
            c:   lower bound of integral in y
            d:   upper bound of integral in y
            L:   Order of integration rule
            Returns:
            G-L Quadrature estimate"""
            assert(L>=1)
            #define a re-scaled f
            f_rescaled = lambda z,zz: f(0.5*(b-a)*z +
                                       0.5*(a+b),0.5*(d-c)*zz + 0.5*(c+d))
            integral = GLQuad2D(f_rescaled,L)
            return integral*(b-a)*0.5*(d-c)*0.5
```

To test this, we use the following integral

$$\frac{2L}{-3(-1)^{2L}+2^{4L+3}-5}\int_{-2}^{3}dx\int_{0}^{3}dy\left[(x+1)^{2L-1}+(y+1)^{2L-1}\right]=1.$$

We expect that this will be integrated exactly by a two-dimensional Gauss–Legendre quadrature rule with L^2 points.

```
In [7]: L = np.arange(1,12)
        for l in L:
            f = lambda x,y: ((x+1)**(2*l-1) + (y+1)**(2*l-1))*(2*l)/(-3*(-1)**(2*l)
                            + 2**(4*l+3)-5)
            integral = 1
            GLintegral = generalGL2D(f,-2,3,0,3,l)
            print("L =", l,"\t Estimate is",GLintegral,
                  "Exact value is",integral,
                  "\nAbs. Relative Error is", np.abs(GLintegral-integral)/integral)

L = 1    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 0.0
L = 2    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 0.0
L = 3    Estimate is 1.0 Exact value is 1
```

```
Abs. Relative Error is 4.4408920985e-16
L = 4    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.4408920985e-16
L = 5    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.33226762955e-15
L = 6    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.88737914186e-15
L = 7    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 4.66293670343e-15
L = 8    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.33226762955e-15
L = 9    Estimate is 1.0 Exact value is 1
Abs. Relative Error is 1.33226762955e-15
L = 10   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 8.881784197e-16
L = 11   Estimate is 1.0 Exact value is 1
Abs. Relative Error is 3.77475828373e-15
```

The method has successfully extended the properties from 1-D to 2-D.

As a further test of this consider a 2-D Cartesian reactor that is a reactor that is finite in x and y and infinite in z (this is nearly the case in power reactors as the reactor is very tall relative to the radius). The scalar flux in this reactor is given by

$$\phi(x, y) = 2 \times 10^9 \left[\cos\left(\frac{\pi x}{100}\right)\cos\left(\frac{\pi y}{100}\right)\right],$$

for $x \in [-50, 50]$ cm and $y \in [-50, 50]$ cm. If $\Sigma_f = 0.1532\ \text{cm}^{-1}$, what is the total fission rate per unit height in this reactor?

The answer is

$$\int_{-50}^{50} dx \int_{-50}^{50} dy\, \Sigma_f \phi(x, y) = 1.241792427 \times 10^{12}.$$

```
In [8]: Sigma_f = 0.1532
        phi = lambda x,y: 2.0e9 *np.cos(np.pi*x*0.01) * np.cos(np.pi*y*0.01)
        FissionRate = generalGL2D(phi,-50,50,-50,50,6)*Sigma_f
        print("Estimated fission rate per unit height is",
              FissionRate,"fissions/cm")

Estimated fission rate per unit height is 1.24179242607e+12 fissions/cm
```

We get 9 digits of accuracy when we evaluate the integrand at only 4^2 points. This is a very efficient way to evaluate an integral.

CODA

We have discussed two basic quadrature techniques: Newton–Cotes and Gauss quadrature, and shown how to generalize them to multi-dimensional integrals. Between numerical integration and differentiation we have the basic tools to solve calculus problems. When we

combine this with the linear algebra and nonlinear solver skills that we learned previously, we will be able to solve systems of ordinary differential equations and partial differential equations.

We start down this path in the next chapter when we discuss the solution of initial value problems for ordinary differential equations.

PROBLEMS

Short Exercises

Using Gauss–Legendre quadrature estimate the following integrals with $L = 2, 4, 6, 8$, and 30.

16.1. $\int_0^{\pi/2} e^{\sin x}\, dx \approx 3.104379017855555098181$

16.2. $\int_0^{2.405} J_0(x)\, dx \approx 1.470300035485$, where $J_0(x)$ is a Bessel function of the first kind given by

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\,\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}.$$

Programming Projects

1. Gauss–Lobatto Quadrature

One sometimes desires a quadrature rule to include the endpoints of the interval. The Gauss–Legendre quadrature rules do not include $x = \pm 1$. Gauss–Lobatto quadrature includes both of these points in the set.

1. Derive the $L = 2$ Gauss–Lobatto quadrature set. There is only one degree of freedom in this quadrature set, the weight, and it needs to integrate linear polynomials exactly. This quadrature rule will have the form

$$\int_{-1}^{1} f(x)\, dx = wf(-1) + wf(1).$$

2. Now derive the $L = 3$ Gauss–Lobatto quadrature set. Now there are two degrees of freedom because the x's must be ± 1 and 0. This rule will integrate cubics exactly and have the form:

$$\int_{-1}^{1} f(x)\, dx = w_1 f(-1) + w_2 f(0) + w_1 f(1).$$

3. Implement this quadrature rule and verify that it integrates the appropriate polynomials exactly.

2. *Gauss–Hermite Quadrature*

Other types of Gauss quadrature are possible. Consider integrals of the form

$$\int_0^\infty f(x)e^{-x}\,dx.$$

These integrals can be handled with Gauss–Hermite quadrature.

1. Derive an $L = 1$ quadrature rule (i.e., determine x_1 and w_1) such that

$$\int_0^\infty f(x)e^{-x}\,dx = w_1 f(x_1),$$

is exact for any linear function $f(x)$.

2. Derive an $L = 2$ quadrature rule such that

$$\int_0^\infty f(x)e^{-x}\,dx = w_1 f(x_1) + w_2 f(x_2),$$

is exact for any cubic (or lower degree) polynomial $f(x)$.

3. Implement these two quadrature rules and verify that they integrate linear and cubic polynomials exactly.

3. *Integration and Root Finding*

Consider a 1-D cylindrical reactor with geometric buckling 0.0203124 cm^{-1} and $D =$ 9.21 cm, $\nu\Sigma_\text{f} = 0.1570\ \text{cm}^{-1}$, and $\Sigma_\text{a} = 0.1532\ \text{cm}^{-1}$. The geometric buckling for a cylinder is given by

$$B_\text{g}^2 = \left(\frac{2.405}{R}\right)^2.$$

1. Find the critical radius of this reactor, that is when $B_\text{g}^2 = B_\text{m}^2$ with

$$B_\text{m}^2 = \frac{\nu\Sigma_\text{f} - \Sigma_\text{a}}{D}.$$

2. Using the numerical integration method of your choice, find the peak scalar flux assuming that power per unit height is 2 MW/cm. Use 200 MeV/fission = 3.204×10^{-11} J.
3. Now assume the reactor has a height of 500 cm and a power of 1000 MW. What is the peak scalar flux? You will need a multi-dimensional integral in this case.