NUCL 697 Homework 1: Two-dimensional interpolation

Alex Hagen

February 25th, 2016

Use composite rules to find probability of particles to have energy for given temeprature ($1000\,\mathrm{K}$) in the range from $0.1E_{avg}$ to $2E_{avg}$

$$\frac{N\left(E\right)}{N} = \frac{2\pi}{\left(\pi kT\right)^{3/2}} E^{1/2} \exp\left(-\frac{E}{kT}\right)$$

with $E_{avg} = \frac{3}{2}kT$
Compare and plot results for different number of subintervals using midpoint, trapezoidal and Simpson's methods - as a function of intervals number.

Motivation

Often, analytical interpolation is not a viable method for solution of a problem. For the problem proposed, using the Maxwell-Boltzmann distribution, analytical integration is tedious and difficult. For this reason, numerical integration has been devised. Not only does numerical integration allow us to integrate functions that we need the explicit interval of, but because it is function agnostic, we can also use similar methods to solve ordinary differential equations.

Method

Numerical integration is closely related to Riemann Sums, and the three methods described in this assignment illustrate this. The three methods described are the midpoint, trapezoidal, and Simpson's method, corresponding to subinterval shapes whose top boundaries have polynomial degree of 0, 1, and 2, respectively. This is illustrated in Figure 1.

The actual code is rather simple. The code first determines the amount of subintervals that will be created from `stdin`. Then, it creates a one dimensional mesh with constant element size and the requested number of subintervals. Then, it moves forward and performs midpoint rule integration by finding the function value at the

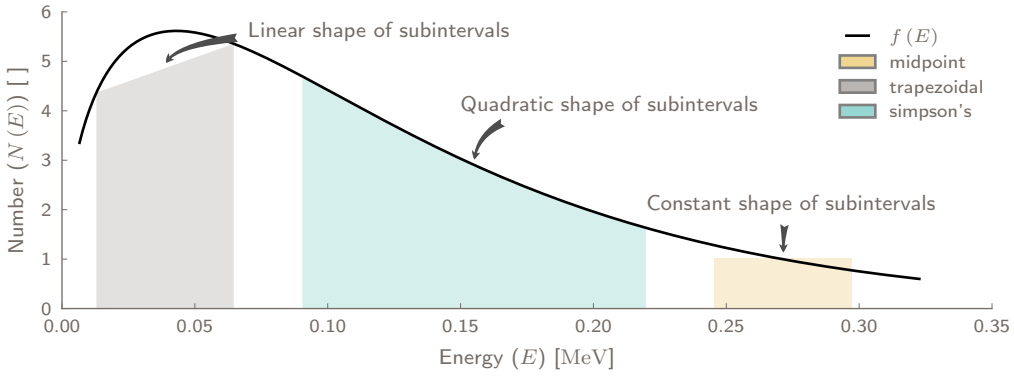

Figure 1: Visual depiction of method for integrating subintervals with the Midpoint, Trapezoidal, and Simpson's method

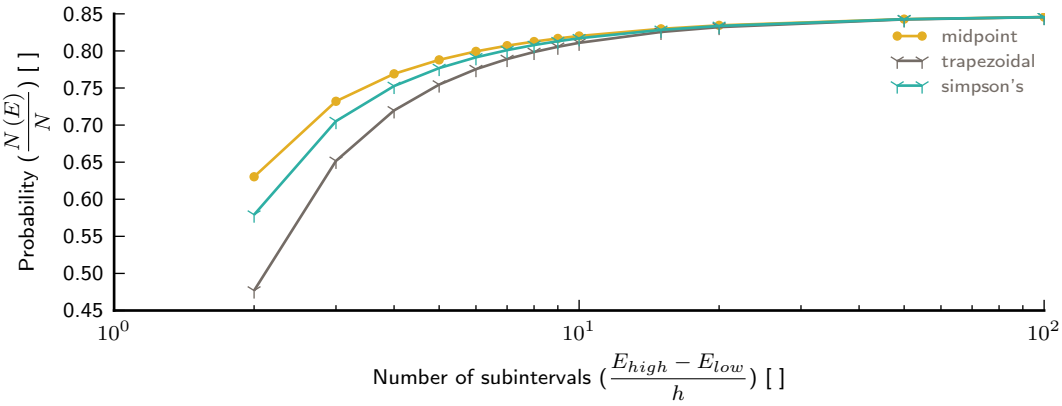

Figure 2: The convergence of the three methods with different

midpoint of each subdomain, and then multiplying that by the horizontal width and summing. After that, it performs trapezoidal rule by averaging the end points of each subdomain, and then multiplying by the horizontal width and summing. Finally, it performs Simpson's rule by the equation

$$\int_a^b f(x)\,dx \approx I_s = \sum_{i=2}^{n} \frac{f(x_{i-1}) + 4f\left(\frac{x_i + x_{i-1}}{2}\right) + f(x_i)}{6}(x_i - x_{i-1})$$

Finally, the results of all three methods are printed to `stdout`.

Results

The convergence of the different methods is shown in Figure 2. Not surprisingly, all of the methods take tens of subintervals to converge, but surprisingly, the Midpoint method is the fastest to converge. It would be assumed that Simpson's method would be closer, because of the higher order shape of it, but in this case that is not true. The trapezoidal method performs least well, likely because of the highly nonlinear behavior of the function hurting the performance of the arithmatic averaging of the end points. In general, with little computational effort (~ 50 subintervals), even this complex nonlinear function can be integrated numerically using any of these three methods.

Code

maxwellian.f90:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                    maxwellian
!                  A two dimensional lagrange interpolation calculator
!                                     Alex Hagen
!
!     Input
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
program maxwellian
! use a variables module which provides global constants and also working
! precision (set to double)
use variables
use functions

! define everything else explicitly
implicit none
! allocate floats for the the high and low energy states and temperature
real(wp)                                :: T, E_avg, E_high, E_low
! allocate a float for the integral
real(wp)                                :: integral
! allocatable arrays for the energy to integrate over
real(wp), allocatable, dimension(:)     :: E
! the number of subintervals and a counter
integer(4)                              :: num, i
! a throwaway text variables
character(80)                           :: text

T = 1000.0_wp
E_avg = 3.0_wp * k * T / 2.0_wp
E_low = 0.1_wp * E_avg
E_high = 2.0_wp * E_avg

! read in the number of desired subintervals
call getarg(1, text)
read(text,'(I10)') num

! allocate our energy array
allocate(E(num))
! put the values into the energy array
E(1) = E_low
do i=2,num
    E(i) = E(i-1) + (E_high - E_low) / num
end do

! do the midpoint rule and write this to a file for later plotting
integral = 0.0_wp
do i=2,num
    integral = integral + &
        N((E(i) + E(i-1))/2.0_wp, T) * (E(i) - E(i-1))
end do
write(*,'(E12.5)') integral

! do the trapezoidal rule and write this to a file for later plotting
```

```
integral = 0.0_wp
do i=2,num
    integral = integral + &
        (N(E(i),T) + N(E(i-1),T))/2.0_wp * (E(i) - E(i-1))
end do
write(*,'(E12.5)') integral

! do the simpson rule and write this to a file for later plotting
integral = 0.0_wp
do i=2,num
    integral = integral + &
        (&
        N(E(i),T) + 4.0_wp*N((E(i)+E(i - 1))/2.0_wp, T) + &
        N(E(i - 1),T)&
        )/6.0_wp * (E(i) - E(i-1))
end do
write(*,'(E12.5)') integral
end program maxwellian
```

functions.f90:

```
module functions
use variables
implicit none

contains

    function N(E, T)
        real(wp)             ::  E, T, N
        N = 2.0_wp * pi * dsqrt(E) * &
        dexp(- E / (k * T)) / dsqrt((pi * k * T)**3.0_wp)
    end function

end module functions
```