Ay 190 Assignment 3

Integration

January 22, 2013

1 Integration via Newton-Cotes Formulae

1.1 Integration of $\int_0^\pi \sin x dx$

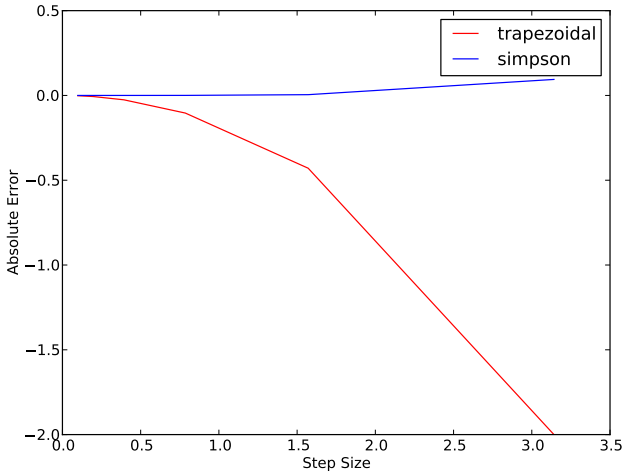

Figure 1: Convergence of $\int_0^\pi \sin x dx = 2$ using the trapezoidal and Simpson's rules with decreasing step size. The absolute error decreases as h^2 under the trapezoidal rule and h^4 under Simpson's rule, as expected.

1.2 Integration of $\int_0^\pi x \sin x dx$

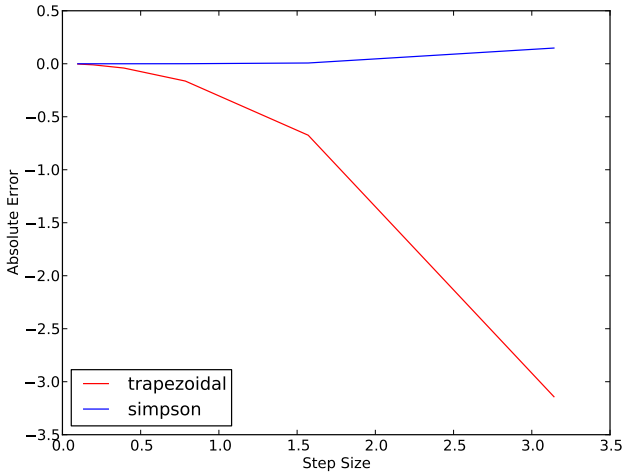

Figure 2: Convergence of $\int_0^\pi x \sin x dx = \pi$ using the trapezoidal and Simpson's rules with decreasing step size. The absolute error decreases as h^2 under the trapezoidal rule and h^4 under Simpson's rule, as expected.

2 Gaussian Quadrature

2.1 Number density of electrons, $kT = 20$ MeV

The number density of electrons in a high temperature environment is given by

$$n_{e^\pm} = \frac{2}{(2\pi\hbar)^3}\int \frac{d^3p}{e^{\beta cp}+1} = \frac{8\pi(k_BT)^3}{(2\pi\hbar c)^3}\int_0^\infty \frac{x^2dx}{e^x+1},$$

where $x = \beta pc$. The rightmost integral was integrated using n-point Gauss-Laguerre integrators, which produced the value 1.80309 to give a number density of 1.902 $\times 10^4 1$ g^{-1}.

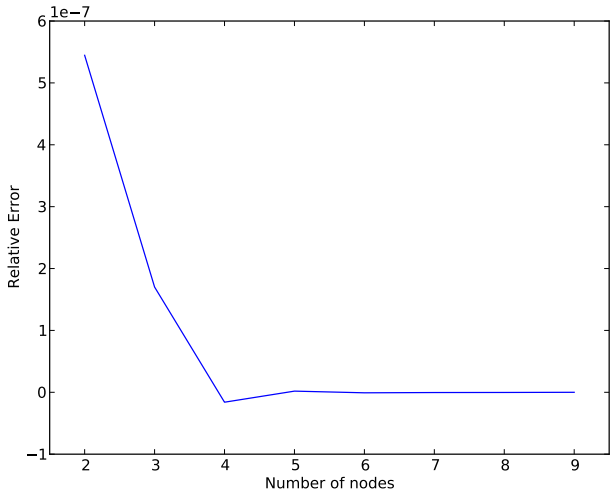

Figure 3: Convergence of the number density of electrons of energy 20 MeV when integrating the rightmost integral in the above equation using n-point Gauss-Laguerre quadrature methods.

2.2 Number density of electron, accounting for spectral distribution

Additionally integrating over kT from 0 to 150 MeV via a Gauss-Legendre integrator produced a number density of $1.01 \times 10^4 3$ g^{-1}, which agreed to within a few parts in $10^1 0$.

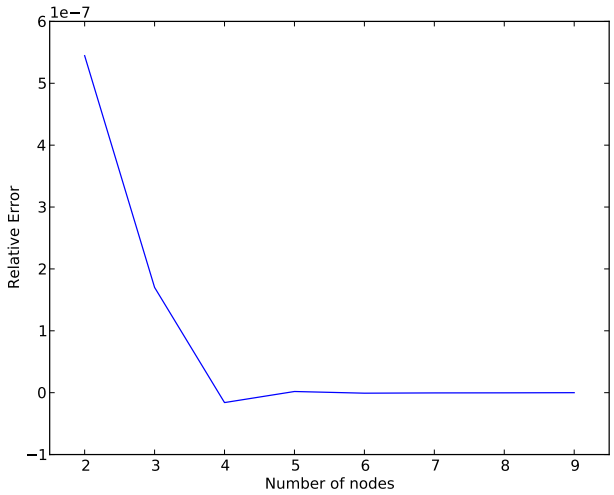

Figure 4: Convergence of the number density of electrons when including the spectral distribution of electrons. The number density was integrated over the spectral distribution using an n-point Gauss-Legendre integrator to integrate over electron energies. To integrate over momenta, a 9-point Gauss-Laguerre integrator was used.

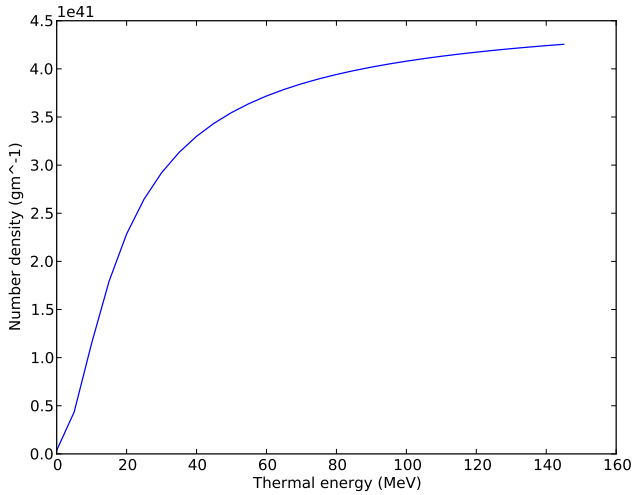

Figure 5: The dependence of the electron number density on electron energy. The total number density was integrated over this curve.

Appendices

The Python modules and scripts used in this assignment include
- A The Integrators: `integrate.py` (see pages)
- B Script for the assignment: `set3.py` (see pages)

A The Integrators: integrate.py

```
# integrate.py
# created 10/5/11 by stacy kim
#
# A suite of quadrature routines, including
#   Newton-Cotes methods (piecewise polynomial interpolation):
#     midpoint rule
#     trapezoidal rule
#     Simpson's rule
#   Gaussian quadrature
#     Gauss-Legendre W(x) = 1
#     Gauss-Laguerre W(x) = x^c * exp(-x)  # here c=0
# All routines integrate the function f over the interval [a,b] over N equally
# spaced sub-intervals.
#
# modified 01/18-20/13
#   made more concise
#   implemented gaussian quadrature routines
#   added new routine integrate_to_accuracy()


import math
import numpy as np
import scipy.special as sp



# NEWTON-COTES METHODS ------------------------------------------------------------

def midpoint(f, a, b, N):
    """Integrates the given function via the midpoint rule."""
    h=float(b-a)/N
    return h*sum([f(x) for x in np.arange(a+h/2,b-h/2,h)])


def trapezoid(f, a, b, N):
    """Integrates the given function via the trapezoidal rule."""
    h=float(b-a)/N
    return h*(sum([f(x) for x in np.arange(a,b,h)-0.5*(f(a)+f(b))]))


def simpson(f, a, b, N):
    """Approximates the integral of a given function via Simpson's formula."""
    h=float(b-a)/N
```

```
    return h/6*sum([f(x)+4*f(x+h/2)+f(x) for x in np.arange(a,b,h)])



# GAUSSIAN QUADRATURE METHODS ----------------------------------------------------

def gauss_legendre(f, a, b, n):
    [roots,weights] = sp.p_roots(n,0)
    return (b-a)/2.*sum([weights[i]*f((b-a)/2.*roots[i]+(a+b)/2.) for i in range(n)])


def gauss_laguerre(f, a, b, n):
    if a != 0 and b != float('int'):
        raise ValueError('Gauss-Laguerre Quadrature integrates over [a,0).')

    w = lambda x: math.exp(x)  # inverse weighting function
    [roots,weights] = sp.l_roots(n,0)
    return sum([weights[i]*w(roots[i])*f(roots[i]) for i in range(n)])



# INTEGRATE TO ACCURACY ----------------------------------------------------------

N=2 # static variable
def integrate_to_accuracy(method, f, a, b, accuracy):
    """Integrates the given function by the given quadrature routine to the given
    accuracy.  Returns the approximate value of the integral and the number of
    sub-intervals or nodes used to obtain the given accuracy."""
    I2=method(f,a,b,N)
    error=9e99

    while (error>accuracy):
        N*=2
        I1=I2
        I2=method(f,a,b,N)
        error=abs((I1-I2)/I1)

    #print "converged to accuracy",accuracy,"with",N,"subintervals"
    return [I2,N]
```

B Script for the assignment: set3.py

```
# set3.py
# created 1/18/13 by stacy kim
import sys
import math
import numpy as np
import matplotlib.pyplot as plt
from integrate import *


# EXERCISE 1 ---------------------------------------------------------------
```

```
# Integration via Newton-Cotes Formulae

# for the function sin(x)
trap=np.array([[math.pi/N, trapezoid(math.sin,0,math.pi,N)] for N in 2**np.arange(0,6)])
simp=np.array([[math.pi/N, simpson  (math.sin,0,math.pi,N)] for N in 2**np.arange(0,6)])

t_relerr=[abs(trap[i,1]-trap[i+1,1]) for i in range(len(trap)-1)]
s_relerr=[abs(simp[i,1]-simp[i+1,1]) for i in range(len(simp)-1)]
print 'sin(x)\nstep size         trapezoidal       simpson'
for i in range(len(trap)-2):
    print '{0:<17} {1:<17} {2:<17}'.format(trap[i,0],t_relerr[i]/t_relerr[i+1],
                                           s_relerr[i]/s_relerr[i+1])

plt.plot(trap[:,0],trap[:,1]-2.,'r',simp[:,0],simp[:,1]-2.,'b')
plt.legend(['trapezoidal','simpson'],loc='best')
plt.xlabel('Step Size')
plt.ylabel('Absolute Error')
plt.savefig('int_sin_convg.pdf')
plt.show()


# for the function x*sin(x)
xsin = lambda x: x*math.sin(x)
trap=np.array([[math.pi/N, trapezoid(xsin,0,math.pi,N)] for N in 2**np.arange(0,6)])
simp=np.array([[math.pi/N, simpson  (xsin,0,math.pi,N)] for N in 2**np.arange(0,6)])

t_relerr=[abs(trap[i,1]-trap[i+1,1]) for i in range(len(trap)-1)]
s_relerr=[abs(simp[i,1]-simp[i+1,1]) for i in range(len(simp)-1)]
print '\nx*sin(x)\nstep size         trapezoidal       simpson'
for i in range(len(trap)-2):
    print '{0:<17} {1:<17} {2:<17}'.format(trap[i,0],t_relerr[i]/t_relerr[i+1],
                                           s_relerr[i]/s_relerr[i+1])

plt.plot(trap[:,0],trap[:,1]-math.pi,'r',simp[:,0],simp[:,1]-math.pi,'b')
plt.legend(['trapezoidal','simpson'],loc='best')
plt.xlabel('Step Size')
plt.ylabel('Absolute Error')
plt.savefig('int_xsin_convg.pdf')
plt.show()


# EXERCISE 2 ------------------------------------------------------------------
# Gaussian Quadrature

EV  = 1.602177e-19 # in J
MEV = 1e6*EV       # in J
HBAR= 1.054571e-34 # in J s
C   = 2.99792e8    # in m/s

coeff = lambda kT:  math.pi*(kT/(math.pi*HBAR*C))**3
f = lambda x: x*x/(math.exp(x)+1)

# Gauss-Laguerre quadrature to compute electron number density
nodes=range(2,10)
```

```
integ=np.array([gauss_laguerre(f,0,float('inf'),n) for n in nodes])
ne   =coeff(20*MEV)*integ

print 'ne =',ne[0],ne[-1], ne[-2]/ne[-1]

relerr=[float(ne[i])/ne[-1]-1 for i in range(len(ne))]
plt.plot(nodes,relerr)
plt.xlim([1.5,nodes[-1]+0.5])
plt.xlabel('Number of nodes')
plt.ylabel('Relative Error')
plt.savefig('ne_convg.pdf')
plt.show()

# Gauss-Legendre quadrature to compute electron number density over E spectrum
ne_nodes=[]
E_range=np.arange(0,150,5.)
const=math.pi*(MEV/(math.pi*HBAR*C))**3
for n in nodes:
    integ_tot=0
    integ=[]
    for E in E_range:
        def f2(E):
            f1 = lambda x: x*x/(math.exp(x/E)+1) # where E in units of MEV
            return gauss_laguerre(f1,0,float('inf'),10)

        integ.append(gauss_legendre(f2,E,E+5,n))
        integ_tot+=integ[-1]

    if n==nodes[-1]:
        plt.plot(E_range,const*np.array(integ))
        plt.xlabel('Thermal energy (MeV)')
        plt.ylabel('Number density (gm^-1)')
        plt.savefig('ne_over_e_spectrum.pdf')
        plt.show()

    ne_nodes.append(integ_tot*const)

relerr=[float(ne_nodes[i])/ne_nodes[-1]-1 for i in range(len(ne))]
plt.plot(nodes, relerr)
plt.xlim([1.5,nodes[-1]+0.5])
plt.xlabel('Number of nodes')
plt.ylabel('Relative Error')
plt.savefig('ne_spectrum_convg.pdf')
plt.show()

print ne_nodes[-1], ne_nodes[-2]/ne_nodes[-1]
```