UNIVERSITY OF OSLO

COMPUTATIONAL PHYSICS

Project 3

UiO : University of Oslo

Authors:

Birgitte Madsen

Magnus Isaksen

Soumya Chalakkal

Autumn 2015

UiO : University of Oslo

Department of Physics
University of Oslo
Sem Sælands vei 24
0371 Oslo, Norway
+47 22 85 64 28
http://www.mn.uio.no/fysikk/english/

Course:

Computational Physics

Project number:

3

Link to GitHub folder:

`https://??`

Hand-in deadline:

Friday, October 23, 2015

Project Members:

Birgitte Madsen
Magnus Isaksen
Soumya Chalakkal

Copies: 1
Page count: ??
Appendices: 0
Completed: ??

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

CHAPTER

2 METHOD

2.1 Nature of the problem

Even though the Schrö dinger equation cannot be solved exactly for the helium atom or more complicated atomic or ionic species due to electron-electron interaction, the ground state energy of the helium atom can be calculated using approximate methods. One method is to assume that the electrons in helium atom occupies scaled hydrogen 1s orbital so that the product of the wave function of the two electrons can be given as

$$\Phi(r_1, r_2) = exp[-\alpha(r1 + r2)] \tag{2.1}$$

in which $\Phi_{1s}(r_i) = exp(-\alpha(r_i))$ is the single particle wave function for a particle at position r_i α is a parameter that corresponds to the charge of helium atom and $r_i = sqrt(x_i^2 + y_i^2 + z_i^2)$ is the cartesian coordinate of particle The ground state correlation energy between these two electrons in the helium atom can be calculated by solving the integral

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha x}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.2}$$

which is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical coulomb interaction. The closed-form solution of Eq. (2.2) is $5\pi^2/16^2$. ??????? Closed form solution is there in the below link but I didnt understand it properly. `https://www.physics.ohio-state.edu/~ntg/6810/readings/hjorth-jensen_notes2013_14.pdf` [1]

2.2 Gauss-Legendre Method for Computing the Integral

To be able to use the Gauss-Legendre method to compute the integral in Eq. (2.2), the limits of the integral must be made finite. Since the wave function

$$e^{-2\alpha x} \tag{2.3}$$

rapidly goes toward zero as x is increased (see Fig. 2.1), the integral can be approximated by the same integral with finite limits.

[1] FiXme Note: write closed form solution ??

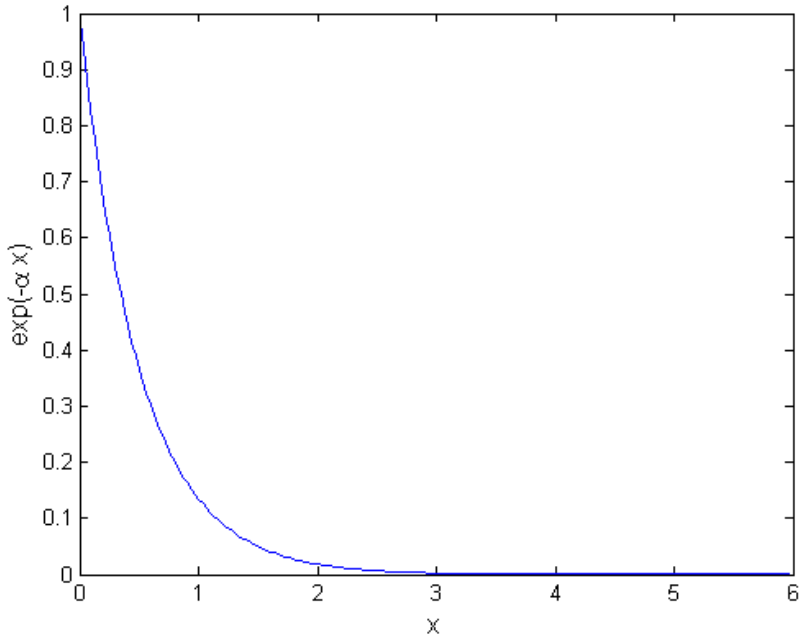

Figure 2.1. Plot of $e^{-2\alpha x}$ with $\alpha = 2$.

[2]

In this project, we [3] have accepted that 10^{-9} is close enough to zero to neglect contributions from the part of the wave function when the wave function gives a value of this order. For $x = 5$ the value of the wave function is $e^{-10\alpha} \approx 2.1 \cdot 10^{-9}$, when $\alpha = 2$. Hence, the considered integral that is to be solved by the Gauss-Legendre method is given by

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-5}^{5} \frac{e^{-2\alpha x}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.4}$$

To be able to use the Gauss-Legendre method, the limits have to be -1 and 1. This is, however, easily obtained by a change in variables using the following quantity

$$\int_a^b f(t)dt = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right) dx \tag{2.5}$$

The first step of the Gauss-Legendre method is then to compute the roots of the n'th Legendre polynomial. The i'th root z_i is approximated by

$$z_i = \cos\left(\frac{\pi(4 \cdot i - 1)}{4n + 2}\right) \tag{2.6}$$

for large n. The n'th [4] Legendre polynomial $L_n(x)$ can then be computed using the recursive relation

$$(j+1)L_{j+1}(z) + jL_{j-1}(x) - (2j+1)zL_j(z) = 0 \tag{2.7}$$

[2] FiXme Note: fix figur
[3] FiXme Note: sorry about the "we"
[4] FiXme Note: right??

with $L_{-1}(x) = 0$ and $L_0(x) = 1$. The code for generation the Legendre polynomial can be found in sec ?? [5] together with a test of the Legendre polynomial generating algorithm.

The derivative of the Legendre polynomial can then be computed as

$$L_n'(z) = \frac{-n \cdot z \cdot L_n(z) + n \cdot L_{n-1}(z)}{1 - z^2} \tag{2.8}$$

Newton's method is then applied to find the best estimation of the roots by considering the expression

$$z_1 = z - \frac{L_n(z)}{L_n'(z)} \tag{2.9}$$

in which z is the first approximation of the root given by Eq. (2.6), and z_1 is a better approximation for the root. The algorithm for generation of the roots and the Legendre polynomials is run until the difference between z_1 and the root approximated by Eq. (2.6) is ultimately zero, which in this project is set to 10^{-6}.

The weight w_i of the i'th root z_i is then obtained by

$$w_i = \frac{2}{(1 - z_i^2)(L_n'(z_i))^2} \tag{2.10}$$

2.3 Generation of Legendre Polynomials

The following lines of code generates the n'th order Legendre polynomial from the recursive relation given in Eq. (2.7) and its roots. Since the roots are symmetric about 0, the for-loop is run from $i = 1$ to $i < m$ for $m = n + 1$ with n being the order of the computed polynomial and hence the length of the array containing the roots and the array containing the respective weights of the roots.

```
//Code for computing the Legendre polynomials to determine the roots of the n'th
    Legendre polynomial
    for (int i = 1; i<m;i++){
        root = cos(pi * (4*i-1) / (4*n + 2 ));
        //approximation of the root of the n'th polynomial
    do{
        // This uses the recursive relation to compute the Legendre polynomials
        double L_plus = 1.0 , L = 0.0, L_minus;
        for (int j = 0; j < n; j++)
        {
        L_minus = L;
        L = L_plus;
        L_plus = (2.0*j +1)*root*L - j*L_minus ;
        L_plus /= j+1;
        }
      //derivative of the Legendre polynomial (L_plus)
        dev_L = (-n*root*L_plus + n*L)/(1-root*root);
        // Newton's method
        z = root;
        root  = z - L_plus/dev_L;
```

[5] FiXme Note: ref to sec

```
    } while(fabs(root - z) > pow(10,-6));
    //compute values of x (array containing the roots) and w (array containing the
        roots)
     * w_temp1 = 2/((1-root*root)*(dev_L*dev_L));
     *(x_temp1++) = root;
     *(x_temp2--) = -root;
     *(w_temp2--) = *(w_temp1++);
}
```

The algorithm starts with an approximation of the i'th root of the n'th order Legendre polynomial as described in Eq. (2.6). After computation of the Legendre polynomial, the approximation is improved by Newton's method. If the difference between the improved approximation and the initial approximation is greater than zero (that is 10^{-6}), the n'th order Legendre polynomial is computed using this new approximation, and with this new Legendre polynomial the previously improved approximation of the root is again improved, and once again, if the difference between the approximated roots is greater than zero, new Legendre polynomials are computed. This procedure is run until the difference between the approximation of the i'th root and the improved approximation of the i'th root is zero. When this is achieved, the root z_i is put into the i'th and $(n-i)$'th entry of the array containing the roots, and the corresponding weights are computed by Eq. (2.10).

2.3.1 Test of the Legendre Polynomial Generation

???? provides the roots and corresponding weights of the n'th Legendre polynomial. [6] For $n = 5$, these are

roots	weights
0	0.568889
$\pm$ 0.538469	0.478629
$\pm$ 0.90618	0.236927

[7] which is exactly what is obtained up to the same accuracy as in Fig. 2.3.1 by running the above lines of code for $n = 5$. (see [8])

[6] FiXme Note: source
[7] FiXme Note: do we want caption??
[8] FiXme Note: ref. to GitHub

CHAPTER

3 CONCLUSION

Conclude.... conclude.... conclude....

APPENDIX

A MATLAB CODE FOR SMT....

This is how, we write MatLab code in the report

```
close all
clear all
clc
%I am a comment

filename = 'Results.xlsx';
sheet = 4;
xlRange = 'B3:C12';

[v,T,vT] = xlsread(filename, sheet, xlRange);
x10=v(:,1);y10=v(:,2);

figure
plot(??)
legend(??)

xlim([??])
ylim([??])

title(??)
xlabel('x')
ylabel('y')
```