UNIVERSIDAD DE LOS ANDES, DEPARTAMENTO DE FÍSICA
FÍSICA ATÓMICA

Valores esperados

Juan Barbosa, 201325901
Marzo 23, 2017

$$\boxed{-\frac{\hbar^2}{2m}\nabla^2\Psi(x)+V(x)\Psi(x)=E\Psi(x)}$$

Para el átomo de helio la ecuación anterior se escribe usando coordenedas esféricas, el método de Hartree consiste en relajar el potencial que actua sobre un electrón. Dado que ambos electrones comparten tres números cuánticos y que la ecuación de Schrödinger no incluye el spín, se consideran indistinguibles. El método consiste en considerar un único electrón a la vez, junto con el potencial producido por los otros dos. Usando consideraciones geométricas, en las cuales a distancias grandes el electrón observará el potencial de una carga puntual con carga $Z-1$ para distancias cortas el efecto será análogo a un monopolo.

$$\begin{aligned} V(r) &= -\frac{Zke^2}{r} \qquad r \ll 1 \\ V(r) &= -\frac{ke^2}{r} \qquad r \gg 1 \end{aligned} \tag{1}$$

Haciendo un cambio de variable $r = \frac{a_0}{z}u$ y la energía propia del sistema:

$$\begin{aligned} \frac{V(u)}{E_0} &= -\frac{2}{u} \qquad u \ll 1 \\ \frac{V(u)}{E_0} &= -\frac{2}{Zu} \qquad u \gg 1 \end{aligned} \tag{2}$$

Este potencial se usa únicamente para la primera iteración, por lo cual no es del todo relevante cual es la forma exacta del mismo para puntos intermédios, dado que con cada iteración el sistema es relajado hasta la estabilidad.

$$\frac{V(u)}{E_0} = -\frac{2}{Zu}\left(1+(Z-1)e^{-u}\right) \tag{3}$$

Usando la probabilidad radial $P = 4\pi R^2 r^2 / \int P dr$ se determina la densidad de carga del electrón.

$$\frac{Q}{e}(u) = -(Z-1)\int_0^u P(w)dw \tag{4}$$

El potencial se obtiene integrando sobre el campo

$$V(u) = -\left(V_0 + \frac{Z}{u} + \int_0^u \frac{2}{Z} \frac{Q/e}{u^2} \right) \tag{5}$$

donde V_0 se ajusta de tal forma que el potencial converja para distancias grandes y Z/u corresponde con el potencial del núcleo.

La implementación se realiza en C, y una última parte en Python para graficar. Se usan 10000 puntos entre $U = 0$ y $U = 30$. La ecuación diferencial a resolver es:

$$\frac{d^2R}{du^2} + \frac{2}{u}\frac{dR}{du} + \left(\epsilon - V(u) - \frac{l(l+1)}{u^2} \right) R = 0 \tag{6}$$

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int N = 1000, Z = 2, l=0;
double dx, *V, *U;

double *linspace(double min, double max, int N);
void initial_potential();
double integrate(double *function, double dx, int N);
double *cal_probability(double *function);
void solver(double *function, double *derivative, double epsilon, int l);
double seaker(double *function, double *derivative);
double *cal_charge(double *probability);
void cal_potential(double *charge);

int main(int argc, char **argv)
{
    int i, j;
    double der, epsilon;
    double *R = malloc(N*sizeof(double));
    double *R_prime = malloc(N*sizeof(double));
    double *P, *charge;
    char name[100];
    V = malloc(N*sizeof(double));
        U = linspace(0.01, 30, N);
    initial_potential();

    R[0] = 1;
    R_prime[0] = -0.99;

    FILE *energies = fopen("energies.dat", "w");

    for(i=0; i<50; i++)
    {
        sprintf(name, "%d_data.dat", i+1);
        FILE *output = fopen(name, "w");
        epsilon = seaker(R, R_prime);
        P = cal_probability(R);
        charge = cal_charge(P);
        cal_potential(charge);
        printf("%d_&_%f_\\\\_\n", i+1, epsilon);
        for(j=0; j<N; j++)
        {
            fprintf(output, "%f_%f_%f_%f\n", U[j], R[j], P[j], charge[j], V[j]);
        }
        fclose(output);
        free(charge);
        free(P);
        fprintf(energies, "%f\n", epsilon);
    }

    free(V);
    free(U);
    free(R);
    free(R_prime);
    fclose(energies);
        return 0;
}

void initial_potential()
{
    int i;
    double coeff, exponent;
    for(i = 0; i<N; i++)
    {
        coeff = -2/(Z*U[i]);
        exponent = exp(-U[i]);
```

```
        V[i] = coeff*(1+(Z-1)*exponent);
    }
}

double *cal_charge(double *probability)
{
    int i;
    double *charge = malloc(N*sizeof(double));
    for(i=0; i<N; i++)
    {
        charge[i] = -(Z-1)*integrate(probability, dx, i+1);
    }
    return charge;
}

void cal_potential(double *charge)
{
    int i;
    double *delta = malloc(N*sizeof(double));
    double V0;
    for(i=0; i<N; i++)
    {
        delta[i] = -(2/Z)*(charge[i]/(U[i]*U[i]));
    }
    for(i=0; i<N; i++)
    {
        V[i] = -(integrate(delta, dx, i+1) + Z/U[i]);
    }
    V0 = 1.0/30 + V[N-1];
    for(i=0; i<N; i++)
    {
        V[i] += -V0;
    }
    free(delta);
}

double *cal_probability(double *function)
{
    int i;
    double *P = malloc(N*sizeof(double)), norm;
    for(i=0; i<N; i++)
    {
        P[i] = pow(U[i]*function[i], 2);
    }

    norm = integrate(P, dx, N);
    for(i=0; i<N; i++)
    {
        P[i] *= 1/norm;
    }
    return P;
}

double integrate(double *function, double dx, int N)
{
    int i;
    double integral = 0;
    for(i=0; i<N; i++)
    {
        integral += function[i];
    }
    return integral*dx;
}

double seaker(double *function, double *derivative)
{
    double energy, last, de, current, low_bound;

    energy = -0.5;
    de = 0.01;
    last = 0;
    current = 1;
    low_bound = 0;
    while((fabs(current) >= 1E-3) && (de > 1E-8) && (energy < 0))
    {
        energy += de;
        solver(function, derivative, energy, 1);
        current = function[N-1];
        if(current*last < 0)
        {
            low_bound = energy;
            energy += -de;
            de *= 0.1;
        }
        if(current > low_bound);
        {
            de *= 1.1;
        }
        last = current;
    }
    return energy;
}
```

```
double *linspace(double min, double max, int N)
{
    int i;
    dx = (max-min)/(N-1);
    double *x = malloc(N*sizeof(double));

    x[0] = min;
    for(i=0; i<(N-1); i++)
    {
        x[i+1] = x[i] + dx;
    }
    return x;
}

void solver(double *function, double *derivative, double epsilon, int l)
{
    int L = l*(l+1), i = 0;
    double der;

    for(i=0; i<(N-1); i++)
    {
        der = -2*derivative[i]/U[i] - function[i]*(epsilon - V[i] - L/(U[i]*U[i]));
        derivative[i+1] = derivative[i] + der*dx;
        function[i+1] = function[i] + derivative[i+1]*dx;
    }
}
```

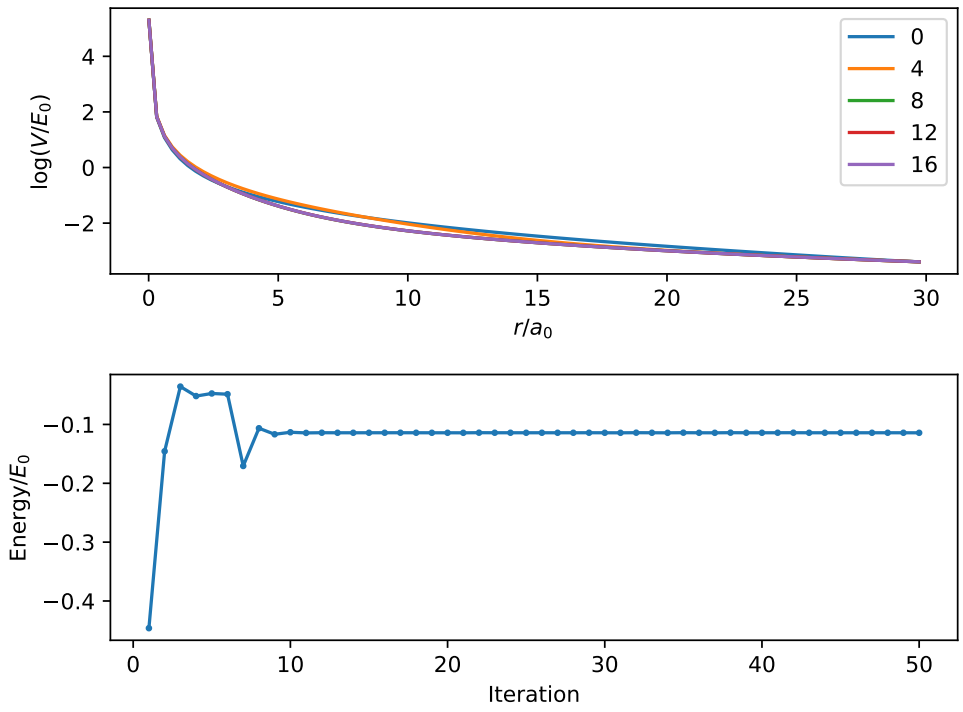

Figura 1: Evolución para el potencial con 10 iteraciones.

La Figura 1 muestra el comportamiento del potencial en para las distintas iteraciones. Se observa que el potencial inicial es considerablemente parecido al real. Las energías se muestran en la parte inferior, de donde se observa que se estabilizan para $\epsilon = -0{,}116$:

La función radial resultante para cada electrón se muestra en la siguiente figura. Además se grafican la probabilidad radial, la densidad de carga y el comportamiento del potencial a distancias cortas y largas.

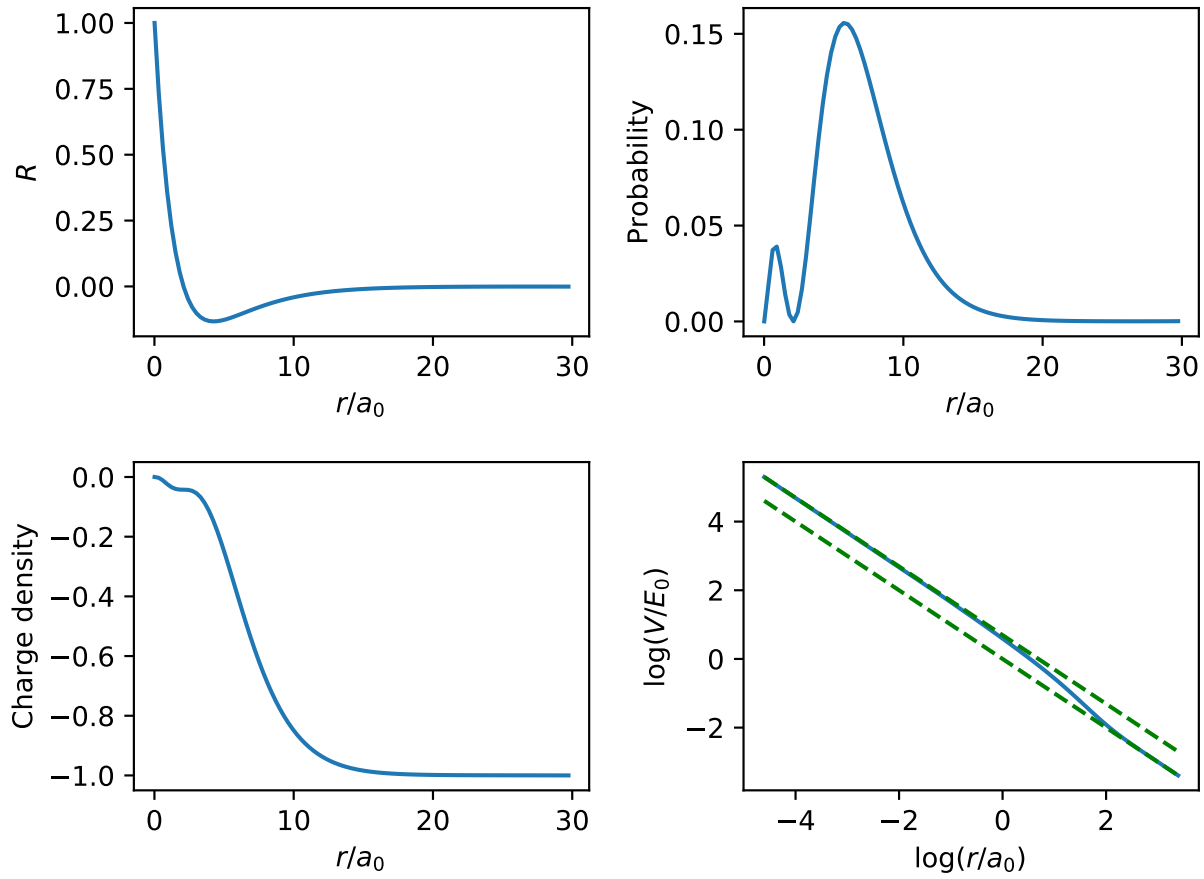

Figura 2: Distintas propiedades de la función encontrada luego de 10 iteraciones.