Cornell University

Department of Physics

Phys 480/680, Astro 690 March 8, 2000

Computational Physics, Spring 2000

Homework Assignment # 6

(Due Thursday, March 16 at 8:40 am in Lab.)

Agenda and readings for the week of February 1:

All readings from *Numerical Recipes: The Art of Scientific Computing*, 2nd edition (in Fortran *or* C).

- Lec9, 03/16: Solution of many non-linear equations in many unknowns (Broyden approaches) **Sections "Intoduction," "Procedure" (you can skip the relativistic part), "Results," and "The LDA approximation," from http://math.nist.gov/DFTdata/atomdata/narrative.html**
- Lab5, 03/09: LDA solutions for atoms

Goal: This homework develops software for complete density functional calculations of the electronic structure and total energy of atoms.

Contents

1 Exchange-correlation energy **2**
- 1.1 Review of background . . . 2
- 1.2 "excPZ" and "excpPZ" . . . 3
  - 1.2.1 Debugging . . . 3
- 1.3 "getVxc" and "getDepsxc" . . . 3
  - 1.3.1 Debugging . . . 4
- 1.4 Vosko, Wilk, and Nusair form for exchange-correlation . . . 4
  - 1.4.1 Debugging . . . 4

2 Putting it all together **4**
- 2.1 "test_atoms.c" . . . 4
  - 2.1.1 Debugging . . . 5
- 2.2 "test_atomsC.c": Carbon . . . 6
- 2.3 "test_atomsX.c": an atom of your very own . . . 6

A "test_excpPZ.c" **7**

B "test_getxc.c" **8**

C VMN forms for exchange correlation **10**

1 Exchange-correlation energy

1.1 Review of background

Within the local density approximation to density functional theory, the total energy of an electronic system is

$$E_{tot} = T + E_{el-nuc} + E_{Hartree} + E_{xc},$$

with

$$\begin{aligned} T &\equiv \sum_i f_i \int \psi_i^*(\vec{x}) \left(-\frac{1}{2}\nabla^2 \psi_i(\vec{x}) \right) \, dV \\ E_{el-nuc} &\equiv \int V_{nuc}(\vec{x}) n(\vec{x}) \, dV \\ E_{Hartree} &\equiv \frac{1}{2} \int \phi(\vec{x}) n(\vec{x}) \, dV \\ E_{xc} &\equiv \int f_{xc}(n(\vec{x})) \, dV, \end{aligned}$$

where

$$n(\vec{x}) \equiv \sum_i f_i |\psi_i(\vec{x})|^2$$

and

$$\nabla^2 \phi(\vec{x}) = -4\pi n(\vec{x}).$$

Minimizing the above total energy is equivalent to solving the Schrödinger equation

$$-\frac{1}{2}\nabla^2 \psi_i(\vec{x}) + V(\vec{x})\psi_i(\vec{x}) = E_i \psi_i(\vec{x}),$$

with

$$V(\vec{x}) \equiv V_{nuc}(\vec{x}) + \phi(\vec{x}) + V_{xc}(\vec{x}),$$

where

$$V_{xc}(\vec{x}) \equiv f'_{xc}(n(\vec{x})).$$

In terms of the electron energies E_i above, an alternative form for the total energy is

$$E_{tot} = \sum_i f_i E_i - E_{Hatree} + \int [f_{xc}(n(\vec{x})) - f'_{xc}(n(\vec{x}))n(\vec{x})] \, dV.$$

This first problem of this homework develops the software to compute the exchange-correlation terms in the above equations. The literature always gives $f_{xc}(n)$ in the form $f_{xc}(n) = \epsilon_{xc}(n)n$. In terms of ϵ_{xc}, the quantities we shall need are

$$\begin{aligned} V_{xc}(\vec{x}) &\equiv f'_{xc}(n(\vec{x})) \\ &= \epsilon'_{xc}(n(\vec{x}))n(\vec{x}) + \epsilon_{xc}(n(\vec{x})) \end{aligned} \tag{1}$$

and the exchange-correlation correction energy,

$$\begin{aligned} \Delta E_{xc} &\equiv \int [f_{xc}(n(\vec{x})) - f'_{xc}(n(\vec{x}))n(\vec{x})] \, dV \\ &\equiv \int \Delta\epsilon_{xc}(n(\vec{x}))n(\vec{x}) \, dV, \end{aligned}$$

where

$$\begin{aligned} \Delta\epsilon_{xc}(n(\vec{x})) &\equiv \frac{1}{n(\vec{x})} [f_{xc}(n(\vec{x})) - f'_{xc}(n(\vec{x}))n(\vec{x})] \\ &= \epsilon_{xc}(n(\vec{x})) - f'_{xc}(n(\vec{x})) \\ &= \epsilon_{xc}(n(\vec{x})) - (\epsilon'_{xc}(n(\vec{x}))n(\vec{x}) + \epsilon_{xc}(n(\vec{x}))) \\ &= -\epsilon'_{xc}(n(\vec{x}))n(\vec{x}) \end{aligned} \tag{2}$$

α	$\frac{3}{4}\left(\frac{3}{2\pi}\right)^{2/3}$
a	0.0311
b	-0.0480
c	0.0020
d	-0.0116
A	-0.1423
B	1.0529
C	0.3334

Table 1: Table of coefficients for Perdew-Zunger form (3) of the local exchange correlation functional.

1.2 "excPZ" and "excpPZ"

`double excPZ(double rs)`

`double excpPZ(double rs)`

Return values:

- exc: value of Perdew-Zunger form for $\epsilon_{xc}(r_s)$
- excp: value of Perdew-Zunger form for $\epsilon'_{xc}(r_s) \equiv d\epsilon_{xc}/dr_s$

Input:

- rs: value of $r_s \equiv \left(\frac{3}{4\pi}\right)^{1/3} n^{-1/3}$

As described in lecture, the literature specifies the exchange-correlation energy *per electron* $\epsilon_{xc}\left(r_s(n)\right)$ as a function of the average distance between electrons $r_s(n) \equiv \left(\frac{3}{4\pi}\right)^{1/3} n^{-1/3}$. Perdew and Zunger (*Phys. Rev.* **B 23**, 5048 (1981).) gave one of the earlier, simplest forms for ϵ_{xc},

$$\epsilon_{xc} = -\frac{\alpha}{r_s} + \begin{cases} a\ln r_s + b + cr_s \ln r_s + dr_s & r_s < 1 \\ \frac{A}{1+B\sqrt{r_s}+Cr_s} & r_s \geq 1 \end{cases}, \tag{3}$$

where Table 1 gives the values of the constant coefficients.

Write routines of the two above two prototypes which compute $\epsilon_{xc}(r_s)$ and $\epsilon'_{xc}(r_s) \equiv d\epsilon_x c/dr_s$ for the Perdew-Zunger form.

1.2.1 Debugging

Even a tiny analytic error in taking the derivative $\epsilon'_{xc}(r_s)$ will result in completely incorrect results for your calculations. (There is no "partial" credit for analytic errors when doing numerical calculations!)

To make certain that you have taken the derivative correctly *and coded it properly*, run your routines exc() and excp() with the main program in Appendix A. When your derivative is right, your results should approach the value one with a very regular digit-shifting pattern!

1.3 "getVxc" and "getDepsxc"

`getVxc(double Vxc[], double Rho[], double r[], double dr[], int N)`

`getDepsxc(double Depsxc[], double Rho[], double r[], double dr[], int N)`

Output:

- Vxc[k]: $V_{xc}(r[k])$ for all grid points r[k]

- Depsxc[k]: $\Delta\epsilon_{xc}(n(r[k])) \equiv -\epsilon'_{xc}(n(\vec{x}))n(\vec{x})$ for all grid points r[k]

Input:

- Rho[k]: $4\pi r^2 n(r[k])$ for all grid points r[k]
- r, dr, N: standard grid information

Write routines of the above prototypes which (a) loop over all grid points k, (b) convert our variable Rho[] to the true value of the density, (c) evaluate rs, (d) call upon excPZ and excpPZ, and (e) evaluate V_{xc} and $\Delta\epsilon_{xc}$ according to Eqs. (1,2). There are two special cases to consider to avoid NaNs corrupting your data. At the origin (k=0), n=Rho[0]/r[0] gives NaN, and, at infinity, (k=N), Rho=0. Thus, if either k=0 *or* Rho<SMALL, your routine should set Vxc=0 and Depsxc=0. (Recall that we #define SMALL 1e-300.)

Hint: In Eqs. (1,2) $\epsilon'_{xc}(n)$ refers to $d\epsilon_{xc}/dn \equiv (d\epsilon_{xc}/dr_s)(dr_s/dn)$, whereas "excpPZ" returns $\epsilon'_{xc}(r_s) \equiv d\epsilon_{xc}/dr_s$. You therefore must multiply the return value of "excpPZ" by $(dr_s/dn) = (-r_s/(3n))$ to evaluate $\epsilon'_{xc}(n)$ in Eqs. (1,2).

1.3.1 Debugging

Run your routines with the main program in Appendix B. This program works with hydrogen. It first makes phi and Vxc for a zero charge density (testing that you have handled Rho[]=0 properly) and creates -1/r+phi+Vxc, which *should* be just -1/r because Rho=0. It then solves for the 1s state. From the density of the 1s state, it then computes Vxc and Depsxc, and finally $\int V_{xc} n\, dV$ and $\Delta E_{xc} = \int \Delta\epsilon_{xc} n\, dV$. You should find that your results converge to -0.3316096642 and 0.0772489584, respectively, for these quantities.

1.4 Vosko, Wilk, and Nusair form for exchange-correlation

The US National Institute of Standards and Technology (NIST) maintains a web site with density functional calculations of all atoms in the periodic table. So that we may compare our results directly with theirs, we must use exactly the same form for exchange-correlation. NIST employs the form suggested by Vosko, Wilk, and Nusair (VWN). Appendix C provides the routines exc() and excp() for this form. You should use these in your code, replacing all calls to excPZ() and excpPZ() with calls to exc(), and excp() respectively.

1.4.1 Debugging

Running "test_getxc" with the VWN form, your results should now approach the slightly different values of -0.3315027563 and 0.0773497767, respectively.

2 Putting it all together

Finally, you are ready to perform density functional calculations of atoms! For this problem set, we will use one of the simplest (but least efficient) approaches to attain self-consistency, "charge mixing".

2.1 "test_atoms.c"

Write a main program which attains self-consistency with the algorithm,

- Initialize:
  1. Rho_0: Rho[k]=0 for all k
- Iterate:
  1. get phi and Vxc from Rho_n
  2. compute V[k]=-1/r[k]+phi[k]+Vxc[k] for all k
  3. get E's from V

4. get Psi's from E's and V
5. get Rhonew from Psi's
6. Rho_{n+1}[k]=$(1-\alpha)$ Rho_n[k]+ α Rhonew[k] for all k

where α is any value between 0 and 1.

Note that if $\alpha = 1$, Rho_{n+1}=Rhonew, and we are just putting the output charge density back in as the input, and iterating. This process, however, often is unstable because the electron density Rho changes too quickly. Taking $\alpha < 1$ helps calm this instability by changing Rho more slowly. For instance, for $\alpha = 0.25$ (which seems to work in most cases), we are "mixing" 75% of the previous estimate of Rho with 25% of the new estimate.

Finally, to monitor the progress of your program, you should have it print out the total energy at each iteration. A suitable code fragment for computing this is

```
                    .
                    .
/* Compute and output total energy */

/* Get correction to sum of electron energies */
getDepsxc(Depsxc,Rho,r,dr,N);
for (k=0; k<=N; k++)
  integrand[k]=(-0.5*phi[k]+Depsxc[k])*Rho[k];
Etot=simpint(integrand,r,dr,N);

/* Add on the sum of the electron energies times the occupancies */
for (l=0; l<=lmax; l++)
  for (n=0; n<=nmax[l]; n++)
    Etot+=F[l][n]*E[l][n];

printf("Etot: %20.15f\n",Etot);
                    .
                    .
```

Note that, because, your solution is converging (hopefully!) and all the quantities eventually stop changing, it doesn't really matter if you use Rho[] or Rhonew[] or if you compute phi[] from Rho[] or Rhonew[] in the above code fragment.

2.1.1 Debugging

Run your test_atoms.c for hydrogen (which has 1 1s electron) with (a) the VWN form for the exchange-correlation functional, (b) N=4000 intervals, (c) the change of variables $r(x) = 1/(1-x)-1-x-x^2-x^3$, and (d) $\alpha = 0.25$. After about about 50 iterations, you should converge to a value of about -0.445671. Compare your result with the "official" result at

`http://math.nist.gov/DFTdata/atomdata/tables/01H.html`

(Note that we are using the *local density approximation* (LDA) to density functional theory.)

Hint: The following code fragment gives an example of a suitable initialization of our arrays for hydrogen:

```
Z=1.; /* Charge on nucleus */
lmax=0; /* Only s states are filled in H */

nmax=ivector(0,lmax); /* Make space for nmax[] */
nmax[0]=0; /* For s states, we only need the first, 1s, with zero nodes */

/* Find nmaxmax=max{nmax} */
```

```
nmaxmax=0;
for (l=0; l<=lmax; l++)
  if (nmax[l]>nmaxmax) nmaxmax=nmax[l];

F=dmatrix(0,lmax,0,nmaxmax);
F[0][0]=1.; /* There is only one electron in the 1s for H */
```

2.2 "test_atomsC.c": Carbon

Run your code with the same α, number of intervals N, change of variables $r(x)$ and form for exchange and correlation as in Problem 2.1.1, but now compute the total energy of Carbon. A suitable initialization for Carbon, which has the electronic structure $1s^2 2s^2 2p^2$, is

```
/* Specs for C */
Z=6.;
lmax=1;

nmax=ivector(0,lmax);
nmax[0]=1;
nmax[1]=0;

nmaxmax=0;
for (l=0; l<=lmax; l++)
  if (nmax[l]>nmaxmax) nmaxmax=nmax[l];

F=dmatrix(0,lmax,0,nmaxmax);
F[0][0]=2.; /* 1s */
F[0][1]=2.; /* 2s */
F[1][0]=2.; /* 2p */
```

Again, compare your results with the official results

`http://math.nist.gov/DFTdata/atomdata/tables`

2.3 "test_atomsX.c": an atom of your very own

Locate the element on the periodic table whose number (Z value) is equal to the last two digits of your social security number or Cornell ID. (If this makes Z>92, pick your favorite element.) Compute the total energy of your element and compare against the official results.

Hint: You may wish to consult a periodic table to first determine the electronic configuration ($1s^2 2s^2 3p^6 \ldots$) for your element.

A "test_excpPZ.c"

```
#include <math.h>
#include <stdio.h>
double excPZ(double rs);

double excpPZ(double rs);

main()
{
  double rs,drs;

  rs=3.; /* Test rs > 1 "branch" */
  printf("Testing derivative for rs=%f ...\n",rs);
  for (drs=1; drs>1e-6; drs/=10)
    printf("    %20.16f %20.16f\n",drs,
   (excPZ(rs+drs)-excPZ(rs))/drs /* <- Finite difference slope */
   /                             /* Ratio should approach 1! */
   excpPZ(rs)                    /* <- Coded value of derivative */
   );

  printf("\n");

  rs=0.3; /* Test rs < 1 "branch" */
  printf("Testing derivative for rs=%f ...\n",rs);
  for (drs=1; drs>1e-6; drs/=10)
    printf("    %20.16f %20.16f\n",drs,
   (excPZ(rs+drs)-excPZ(rs))/drs /* <- Finite difference slope */
   /                             /* Ratio should approach 1! */
   excpPZ(rs)                    /* <- Coded value of derivative */
   );
}
```

B "test_getxc.c"

```
#include "p480.h"
#define Nmx 40000
main()
{
  /* Change of variables info */
  double *r,*dr;
  int N;

  /* Physics variables */
  double *V,*Rho,*Rhonew,*phi,*Vxc,*Depsxc,*integrand,Z;
  int lmax,*nmax,nmaxmax;
  double **E,***Psi,**F;
  double Etot;

  /* Working variables */
  int n,l,k;
  double x;

  /* Solver iteration varibles */
  int it;

  /* Value of pi */
  const double pi=4.*atan(1.);

  /* Specs for hydrogen */
  Z=1.;
  lmax=0;

  nmax=ivector(0,lmax);
  nmax[0]=0;

  nmaxmax=0;
  for (l=0; l<=lmax; l++)
    if (nmax[l]>nmaxmax) nmaxmax=nmax[l];

  F=dmatrix(0,lmax,0,nmaxmax);
  F[0][0]=1.;

  /* The rest is now general for ANY case */
  E=dmatrix(0,lmax,0,nmaxmax); /* Make space for E's and Psi's */
  Psi=d3tensor(0,lmax,0,nmaxmax,0,Nmx);
  Rho=dvector(0,Nmx);
  Rhonew=dvector(0,Nmx);
  phi=dvector(0,Nmx);
  Vxc=dvector(0,Nmx);
  Depsxc=dvector(0,Nmx);

  integrand=dvector(0,Nmx);

  /* Grid vectors */
  r=dvector(0,Nmx);
```

```
dr=dvector(0,Nmx);
V=dvector(0,Nmx);

for (N=40; N<=Nmx; N*=10) {

  /* Set up grid */
  for (k=0; k<=N; k++) {
    x=((double) k)/((double) N);
    r[k]=1/(1-x)-1-x;
    dr[k]=1/(1-x)/(1-x)-1;
  }
  dr[N]=0.;

  /* Initialize charge density */
  for (k=0; k<=N; k++)
    Rho[k]=0.;

  /* Test section */
  {
    /* Make potential from zero charge (debugs NaNs, etc.) */
    getphi(phi,Rho,r,dr,N);
    getVxc(Vxc,Rho,r,dr,N);
    for (k=0; k<=N; k++)
      V[k]=-Z/r[k]+phi[k]+Vxc[k];
    V[0]=0.;

    /* Get H 1s wave function, and DENSITY */
    getallEs(E,lmax,nmax,Z,V,r,dr,N);
    getallPsis(Psi,E,lmax,nmax,V,r,dr,N);
    getRho(Rhonew,Psi,F,lmax,nmax,N);

    /* Compute Exc energy correction */
    getDepsxc(Depsxc,Rhonew,r,dr,N);
    getVxc(Vxc,Rhonew,r,dr,N);

    for (k=0; k<=N; k++)
      integrand[k]=Vxc[k]*Rhonew[k];
    printf("N=%6d <Vxc>=%15.10f ",N,
   simpint(integrand,r,dr,N)
   );
    for (k=0; k<=N; k++)
      integrand[k]=(Depsxc[k])*Rhonew[k];
    printf("<DExc>=%15.10f\n",
   simpint(integrand,r,dr,N)
   );
  }
}

/* Be a good citizen and clean up... */
free_dvector(r,0,Nmx);
free_dvector(dr,0,Nmx);
free_dvector(V,0,Nmx);
```

```
  free_dmatrix(E,0,lmax,0,nmaxmax);
  free_d3tensor(Psi,0,lmax,0,nmaxmax,0,Nmx);
  free_dvector(Rho,0,Nmx);
  free_dvector(Rhonew,0,Nmx);
  free_dvector(phi,0,Nmx);
  free_dvector(Vxc,0,Nmx);
  free_dvector(Depsxc,0,Nmx);
}
```

C VMN forms for exchange correlation

```
double exc(double rs)
{
  /* constants */
  const double
    pi = 4.*atan(1.),
    X1 = 0.75*pow(3.0/(2.0*pi),2.0/3.0),  /* Exchange energy coeff */
    A  =  0.0310907,
    x0 = -0.10498,
    b  = 3.72744,
    c  = 12.9352,
    Q  = sqrt(4*c-b*b),
    X0 = x0*x0+b*x0+c;

  double x=sqrt(rs),X=x*x+b*x+c;

  return -X1/rs
    + A*(
        +log(x*x/X)+2*b/Q*atan(Q/(2*x+b))
        -(b*x0)/X0*(
    log((x-x0)*(x-x0)/X)+2*(2*x0+b)/Q*atan(Q/(2*x+b))
    )
);
}

double excp(double rs)
{
  /* constants */
  const double
    pi = 4.*atan(1.),
    X1 = 0.75*pow(3.0/(2.0*pi),2.0/3.0),  /* Exchange energy coeff */
    A  =  0.0310907,
    x0 = -0.10498,
    b  = 3.72744,
    c  = 12.9352,
    Q  = sqrt(4*c-b*b),
    X0 = x0*x0+b*x0+c;

  double x=sqrt(rs),X=x*x+b*x+c;

  double dx=0.5/x; /* Chain rule needs dx/drho! */

  return dx*(
```

```
     2*X1/(rs*x)+A*(
    2./x-(2*x+b)/X-4*b/(Q*Q+(2*x+b)*(2*x+b))
    -(b*x0)/X0*(2/(x-x0)-(2*x+b)/X-4*(2*x0+b)/
                                                (Q*Q+(2*x+b)*(2*x+b)) )
    )
     );
}
```