HW4 Solutions
Computational Physics
Spring 2004

Gregg Stiesberg

3/2/04

getEdebug-new

```
N=1000
Nstates=5
State with  0 nodes, E= -0.499999999846
State with  1 nodes, E= -0.124999998968
State with  2 nodes, E= -0.055555538269
State with  3 nodes, E= -0.031249824704
State with  4 nodes, E= -0.019998845348
N=10000
Nstates=5
State with  0 nodes, E= -0.500000000000
State with  1 nodes, E= -0.125000000000
State with  2 nodes, E= -0.055555555554
State with  3 nodes, E= -0.031249999983
State with  4 nodes, E= -0.019999999881
```

You should also be able to run with N=100000.

getallEsdebug

```
E_1s=-17.983776531256; E_2s= -4.479375169711; E_2p= -4.496747765121;
E_1s=-17.999998254570; E_2s= -4.499997193776; E_2p= -4.499999537814;
E_1s=-17.999999999825; E_2s= -4.499999999719; E_2p= -4.499999999954;
E_1s=-18.000000000000; E_2s= -4.500000000000; E_2p= -4.500000000000;
```

getPsidebug

In finding kmatch, we must make sure it is an even number, or schint won't work. There was also a subtlety in renormalizing the partial arrays to Psi[kmatch];

depending on how your loops were set up, a temporary variable was necessary to divide into the array elements.

Amazingly after all that array interpolating and massaging, nearly fourth order scaling emerges!

```
    40   0.0295771469347061
   400   0.0000063194037136
  4000   0.0000000008329453
 40000   0.0000000000000865
```

getallPsisdebug

When copying from the "nrutil.c" file to "p480.c", there are two versions of the tensor routines; make sure to copy the ANSI versions (they were first in my file).

To compile at home (Debian Sarge, gcc 3.3.3) I needed to copy the macro lines

```
#define NR_END 1
#define FREE_ARG char*
```

that are needed by free_d3tensor(). I also needed to add the header file

```
#include <stddef.h>
```

to define "size_t" in d3tensor().

Some select results:

```
.
.
1s:       0.155415723094  0.266090096970  0.266090125348
1s:       0.158776801802  0.270932491875  0.270932532556
1s:       0.162198501131  0.275825791034  0.275825820181
1s:       0.165681818182  0.280769599734  0.280769641913
1s:       0.169227768265  0.285763588780  0.285763618703
.
.
.
2s:      13.843066666667  0.057179070925  0.057169942834
2s:      14.560498097826  0.044585205492  0.044551887857
2s:      15.343793181818  0.033726951187  0.033713565823
2s:      16.202362797619  0.024704488389  0.024665821430
2s:      17.147500000000  0.017374802101  0.017356592413
2s:      18.192875328947  0.011708277694  0.011672063443
2s:      19.355197222222  0.007464894446  0.007443110832
.
.
```

```
.
2p:       0.010836635220  0.000023841331  0.000023841312
2p:       0.011273402997  0.000025796124  0.000025796243
2p:       0.011722784810  0.000027887568  0.000027887552
2p:       0.012185019841  0.000030123053  0.000030123187
2p:       0.012660350318  0.000032511486  0.000032511474
2p:       0.013149021565  0.000035060986  0.000035061136
2p:       0.013651282051  0.000037781308  0.000037781302
```

getRhodebug

```
Total charge is:    5.999999999999996
```

getphidebug

The initial condition changes to `y[1]=6.0` for the carbon atom, since there are six electrons now. To handle the general case we could just integrate Rho.

Some care is needed when dividing by r[0] and r[N] to avoid bunk values in your arrays. This has no effect on the calculations thus far, but it's a good habit to keep track of such irregular points.

For hydrogen:

```
   40       0.310701311734 (Hartree Energy)
  400       0.312499799660 (Hartree Energy)
 4000       0.312499999978 (Hartree Energy)
40000       0.312500000000 (Hartree Energy)
```

For carbon:

```
   40      26.132477617762 (Hartree Energy)
  400      26.239183309302 (Hartree Energy)
 4000      26.239197529387 (Hartree Energy)
40000      26.239197530864 (Hartree Energy)
```

We saw variations of ± 2 in the last digit of the final energy.