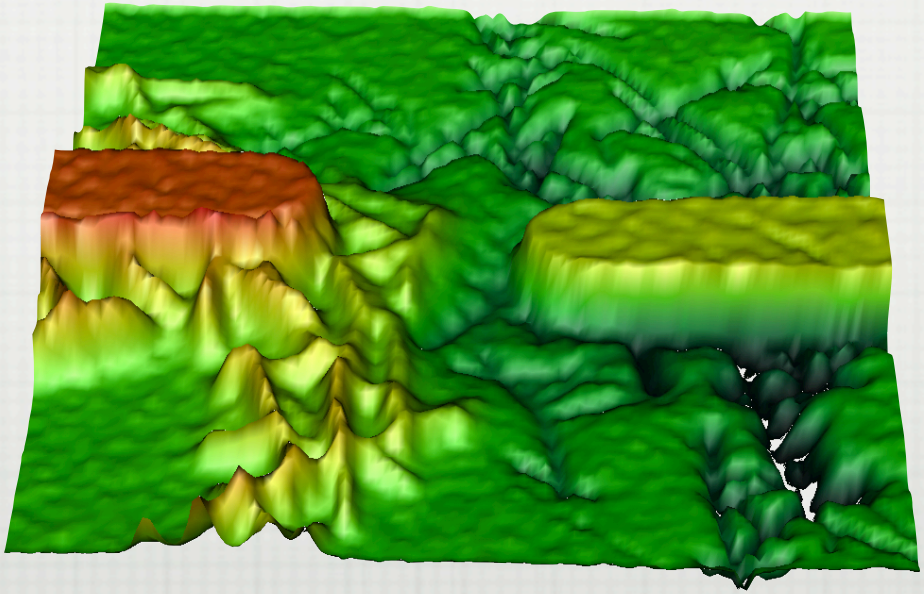
PHY-4810
COMPUTATIONAL PHYSICS

LECTURE 13B: MATRIX ALGEBRA, ADVANCED TOPICS



SUMMARY FROM LAST FRIDAY

$$\mathcal{A}\ x = b$$

- **Gauss-Jordan: compute A^{-1} explicitly (N^3 process)**
- **LU decomposition: $N^3/3$ (does not yield A^{-1})**
- **Taking advantage of form of the matrix (e.g. Cholesky, etc)**

$$\begin{bmatrix} \alpha_{00} & 0 & 0 & 0 \\ \alpha_{10} & \alpha_{11} & 0 & 0 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & 0 \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \cdot \begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ 0 & \beta_{11} & \beta_{12} & \beta_{13} \\ 0 & 0 & \beta_{22} & \beta_{23} \\ 0 & 0 & 0 & \beta_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

ADVANCED TOPICS

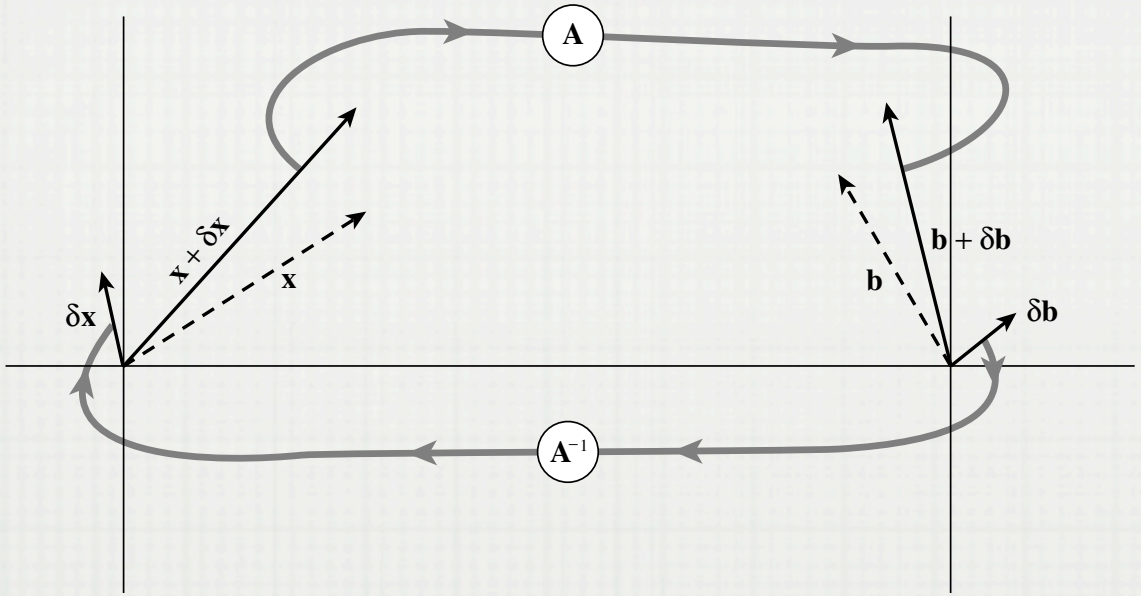
ST I:
ITERATIVE IMPROVEMENTS

- Suppose we know the solution up to some error: $x+\delta x$
- We can inject the solution in the initial problem to compute how far we are from b: $A \cdot (x+\delta x) = b + \delta b$
- In fact we have: $A \cdot \delta x = \delta b$
- We can now solve this equation to estimate δx:

  $A \cdot \delta x = A \cdot (x+\delta x) - b$

- Once we know δx, we can then improve solution x
- *(See method mprove from ludcmp.h header file)*

IN PRACTICE
(SEE LUDCMP.H FOR IMPLEMENTATION)

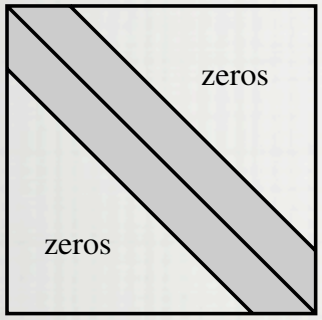


- Each call to improve realize one improvement, which is typically sufficient you can run it a few more times to confirm convergence it is a good idea to use this iterative improvement (only a N^2 cost)

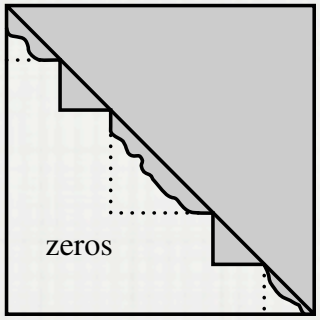
ST2:
SINGULAR VALUE DECOMPOSITION

- **When problems are too close to singularity**
  - **SVD will DIAGNOSE the problem and even provide a SOLUTION to it**
  - **SVD also solve linear LEAST-SQUARE problems (overdetermined problems)**
- **This is an advanced topic that could be covered in depth in a project**

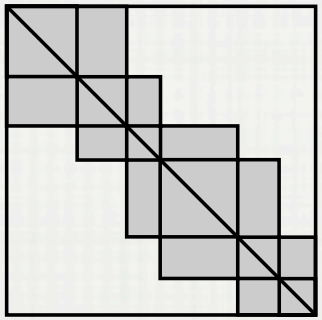
ST2:
SPARSE MATRICES



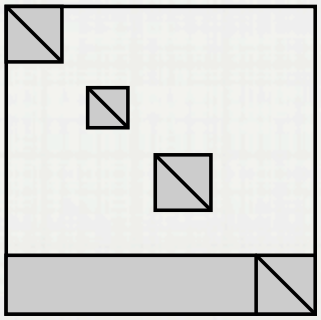
Band-diagonal



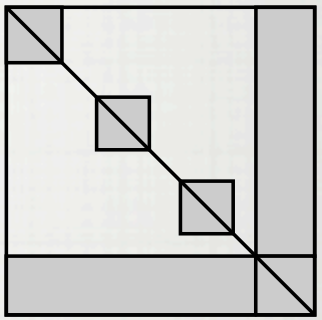
Block triangular



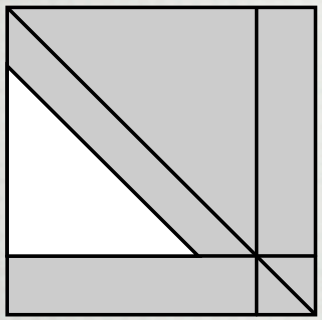
Block tridiagonal



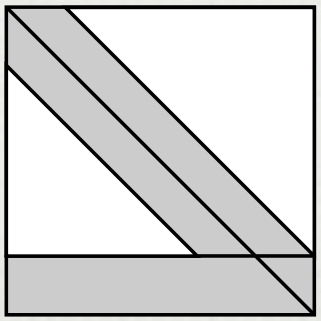
singly bordered block diagonal

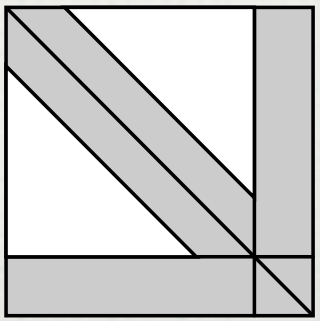


doubly bordered block diagonal

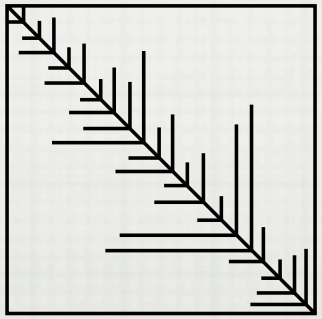


bordered band-triangular





singly and doubly bordered band-diagonal



OTHERS

INDEXED STORAGE: COMPRESSED COLUMN STORAGE

$$\begin{bmatrix} 3.0 & 0.0 & 1.0 & 2.0 & 0.0 \\ 0.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 7.0 & 5.0 & 9.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 6.0 & 5.0 \end{bmatrix}$$

index k	0	1	2	3	4	5	6	7	8
val[k]	3.0	4.0	7.0	1.0	5.0	2.0	9.0	6.0	5.0
row_ind[k]	0	1	2	0	2	0	2	4	4

index i	0	1	2	3	4	5
col_ptr[i]	0	1	3	5	8	9

HARWELL-BOEING FORMAT

- 3 vectors are used
  - val : non zero-values
  - row_ind: row index
  - col_ptr: column index ***
    - col_ptr[0]=0;
    - col_ptr[ncol+1]=nsparse
    - col_ptr[i] =>
      - the first nonzero in column i is in col_ptr[i]; the last is at col_ptr[i+1]-1
      - [[answers the question: *How many non-zero elements have we encountered in previous column?]]*

$$\begin{bmatrix} 3.0 & 0.0 & 1.0 & 2.0 & 0.0 \\ 0.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 7.0 & 5.0 & 9.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 6.0 & 5.0 \end{bmatrix}$$

index k	0	1	2	3	4	5	6	7	8
val[k]	3.0	4.0	7.0	1.0	5.0	2.0	9.0	6.0	5.0
row_ind[k]	0	1	2	0	2	0	2	4	4

index i	0	1	2	3	4	5
col_ptr[i]	0	1	3	5	8	9

$$\begin{bmatrix} 3.0 & 0.0 & 1.0 & 2.0 & 0.0 \\ 0.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 7.0 & 5.0 & 9.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 6.0 & 5.0 \end{bmatrix}$$

index k	0	1	2	3	4	5	6	7	8
val[k]	3.0	4.0	7.0	1.0	5.0	2.0	9.0	6.0	5.0
row_ind[k]	0	1	2	0	2	0	2	4	4

index i	0	1	2	3	4	5
col_ptr[i]	0	1	3	5	8	9

```
/* in this first example I explicitely define the variables
to be used later to build a sparse matrix */
    NRsparseMat A(5,5,9);

//the following should not be needed since they have been
given above
    A.nrows=5;
    A.ncols=5;
    A.nvals=9;

// colum index (see lecture notes)
    A.col_ptr[0]=0;
    A.col_ptr[1]=1;
    A.col_ptr[2]=3;
    A.col_ptr[3]=5;
    A.col_ptr[4]=8;
    A.col_ptr[5]=9;

// rows where sparse elements will be located
    A.row_ind[0]=0;
    A.row_ind[1]=1;
    A.row_ind[2]=2;
    A.row_ind[3]=0;
    A.row_ind[4]=2;
    A.row_ind[5]=0;
    A.row_ind[6]=2;
    A.row_ind[7]=4;
    A.row_ind[8]=3; //this is modified compared to NR book
par.2.7 since the matrix there is singular

//now the values themselves
    A.val[0] = 3.;
    A.val[1] = 4.;
    A.val[2] = 7.;
    A.val[3] = 1.;
    A.val[4] = 5.;
    A.val[5] = 2.;
    A.val[6] = 9.;
    A.val[7] = 6.;
    A.val[8] = 5.;
```

```cpp
#include <iostream>
#include "nr3.h"
#include "sort.h"
#include "sparse.h"

int main (int argc, char * const argv[]) {


    int nsparse=0;
    int i, j;
    double eps=0.00001;

    //example from lecture notes
    int n=5;
    MatDoub Full(n,n,0.0);

    Full[0][0]=3.0;
    Full[0][2]=1.0;
    Full[0][3]=2.0;
    Full[1][1]=4.0;
    Full[2][1]=7.0;
    Full[2][2]=5.0;
    Full[2][3]=9.0;
    Full[4][3]=6.0;
    Full[4][4]=5.0;

    //first we need to find out how many non-zero elements we have
    // we store that information in "nsparse"
    for (i= 0; i<n; i++){
          for (j = 0; j<n; j++){
               if(fabs((Full[i][j]))>eps) {
                    nsparse++;
               }
          }
    }
    //now we can create a sparse object
    NRsparseMat Sparse(n,n,nsparse);

    nsparse=0;
    for (j = 0; j<n; j++){
          if(j>0) {
               Sparse.col_ptr[j+1]=Sparse.col_ptr[j];
          }
          else
          {
               Sparse.col_ptr[0]=0;
          }
          for (i = 0; i<n; i++){
               if(fabs(Full[i][j])>eps) {
                    Sparse.row_ind[nsparse]=i;
                    Sparse.val[nsparse]=Full[i][j];
                    Sparse.col_ptr[j+1]++;
                    nsparse++;
               }
          }
    }

          COUT << "NSPARSE=" << NSPARSE << ENDL;
          COUT << "I= \T";
          FOR(I=0;I<NSPARSE;I++){
                COUT << I << " \T" ;
          }
          COUT << ENDL;
          COUT << "VAL= \T";

          FOR(I=0;I<NSPARSE;I++){
                COUT << SPARSE.VAL[I] << " \T";
          }
          COUT << ENDL;
          COUT << "ROW= \T";

          FOR(I=0;I<NSPARSE;I++){
                COUT << SPARSE.ROW_IND[I] << " \T";
          }
          COUT << ENDL;
          COUT << ENDL;
          COUT << "I= \T";
          FOR(I=0;I<N+1;I++){
                COUT << I << " \T";
          }
          COUT << ENDL;
          COUT << "COL= \T";

          FOR(I=0;I<N+1;I++){
                COUT << SPARSE.COL_PTR[I] << " \T";
          }
          COUT << ENDL;


          RETURN 0;
    }
```

```
vinces-MacBook-Pro:Debug vm2$ ./Resistors
Nsparse=9
i=      0       1       2       3       4       5       6       7       8
val=    3       4       7       1       5       2       9       6       5
row=    0       1       2       0       2       0       2       4       4

i=      0       1       2       3       4       5
col=    0       1       3       5       8       9
vinces-MacBook-Pro:Debug vm2$
```

IN PRACTICE: SPARSE.H

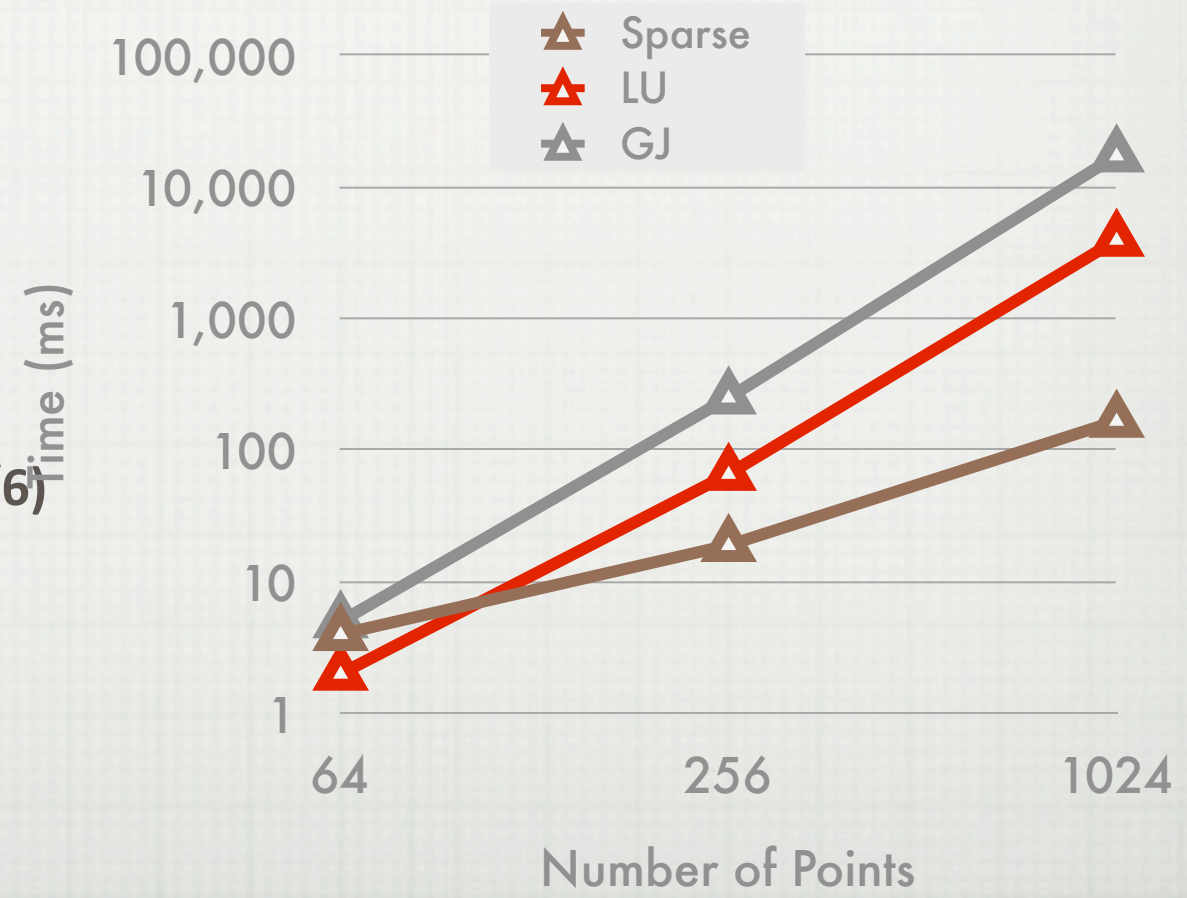
```
struct NRsparseMat
Sparse matrix data structure for compressed column storage.
{
    Int nrows;                                  Number of rows.
    Int ncols;                                  Number of columns.
    Int nvals;                                  Maximum number of nonzeros.
    VecInt col_ptr;                             Pointers to start of columns.  Length is ncols+1.
    VecInt row_ind;                             Row indices of nonzeros.
    VecDoub val;                                Array of nonzero values.

    NRsparseMat();                              Default constructor.
    NRsparseMat(Int m,Int n,Int nnvals);        Constructor.  Initializes vector to zero.
    VecDoub ax(const VecDoub &x) const;         Multiply A by a vector x[0..ncols-1].
    VecDoub atx(const VecDoub &x) const;        Multiply A^T by a vector x[0..nrows-1].
    NRsparseMat transpose() const;              Form A^T.
};
```

SPARSE: EXAMPLE (SEE SPARSE.CPP)

```
#include "sparse.h"


#include "linbcg.h"
#include "asolve.h"

NRsparseMat A(5,5,9);

//here we define A

NRsparseLinbcg C(A);

VecDoub x(5,1.0);
VecDoub b(5,0.0);

//here we define x and b

Int itol=1;
Doub tol=1.0E-72;;
Int itmax=10000;
Int iter;
Doub err;

C.solve(b,x,itol,tol,itmax,iter,err);
```

SCALING

- Gaussj -> N^3
- LU -> $N^3/3$
- (Cholesky -> $N^3/6$)
- Sparse -> N^2



ST4:
IS MATRIX INVERSION AN N^3 PROCESS?

- **How many multiplications do you need for the following problem?**

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \cdot \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix}$$

- **EIGHT, right?**

$$c_{00} = a_{00} \times b_{00} + a_{01} \times b_{10}$$
$$c_{01} = a_{00} \times b_{01} + a_{01} \times b_{11}$$
$$c_{10} = a_{10} \times b_{00} + a_{11} \times b_{10}$$
$$c_{11} = a_{10} \times b_{01} + a_{11} \times b_{11}$$

COULD YOU DO IT IN 7 MULTIPLICATIONS?

$$
\begin{aligned}
Q_0 &\equiv (a_{00} + a_{11}) \times (b_{00} + b_{11}) \\
Q_1 &\equiv (a_{10} + a_{11}) \times b_{00} \\
Q_2 &\equiv a_{00} \times (b_{01} - b_{11}) \\
Q_3 &\equiv a_{11} \times (-b_{00} + b_{10}) \\
Q_4 &\equiv (a_{00} + a_{01}) \times b_{11} \\
Q_5 &\equiv (-a_{00} + a_{10}) \times (b_{00} + b_{01}) \\
Q_6 &\equiv (a_{01} - a_{11}) \times (b_{10} + b_{11})
\end{aligned}
$$

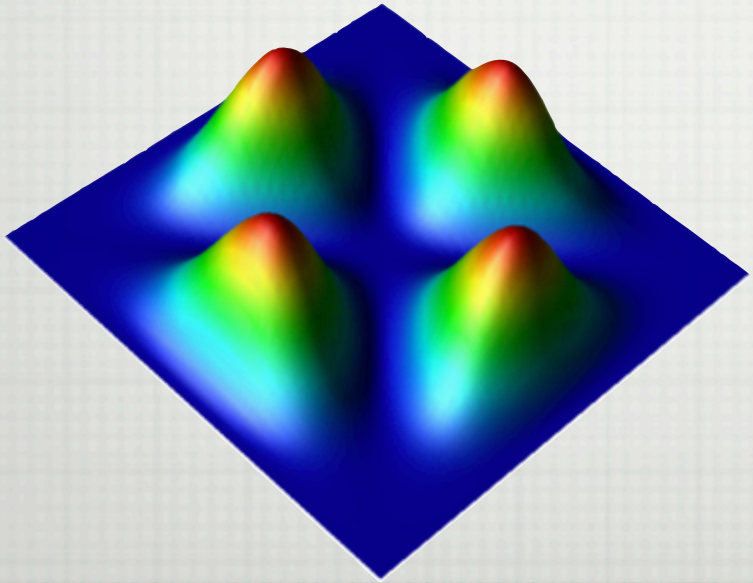
$$
\begin{aligned}
c_{00} &= Q_0 + Q_3 - Q_4 + Q_6 \\
c_{10} &= Q_1 + Q_3 \\
c_{01} &= Q_2 + Q_4 \\
c_{11} &= Q_0 + Q_2 - Q_1 + Q_5
\end{aligned}
$$

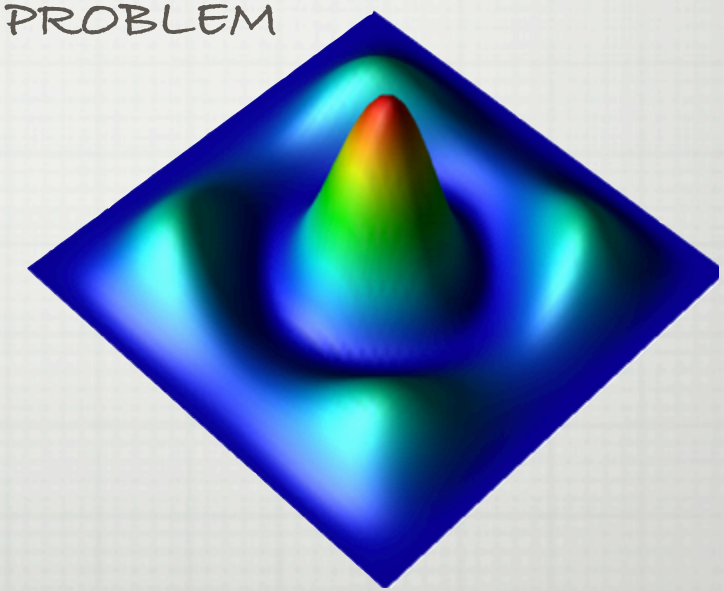
- **What's the use of this? There is one fewer multiplications than in full equation, but many more additions and subtractions. It is not clear that anything has been gained.**
- **But notice that in the a's and b's are never commuted. Therefore these relationships are valid when the a's and b's are themselves matrices.**

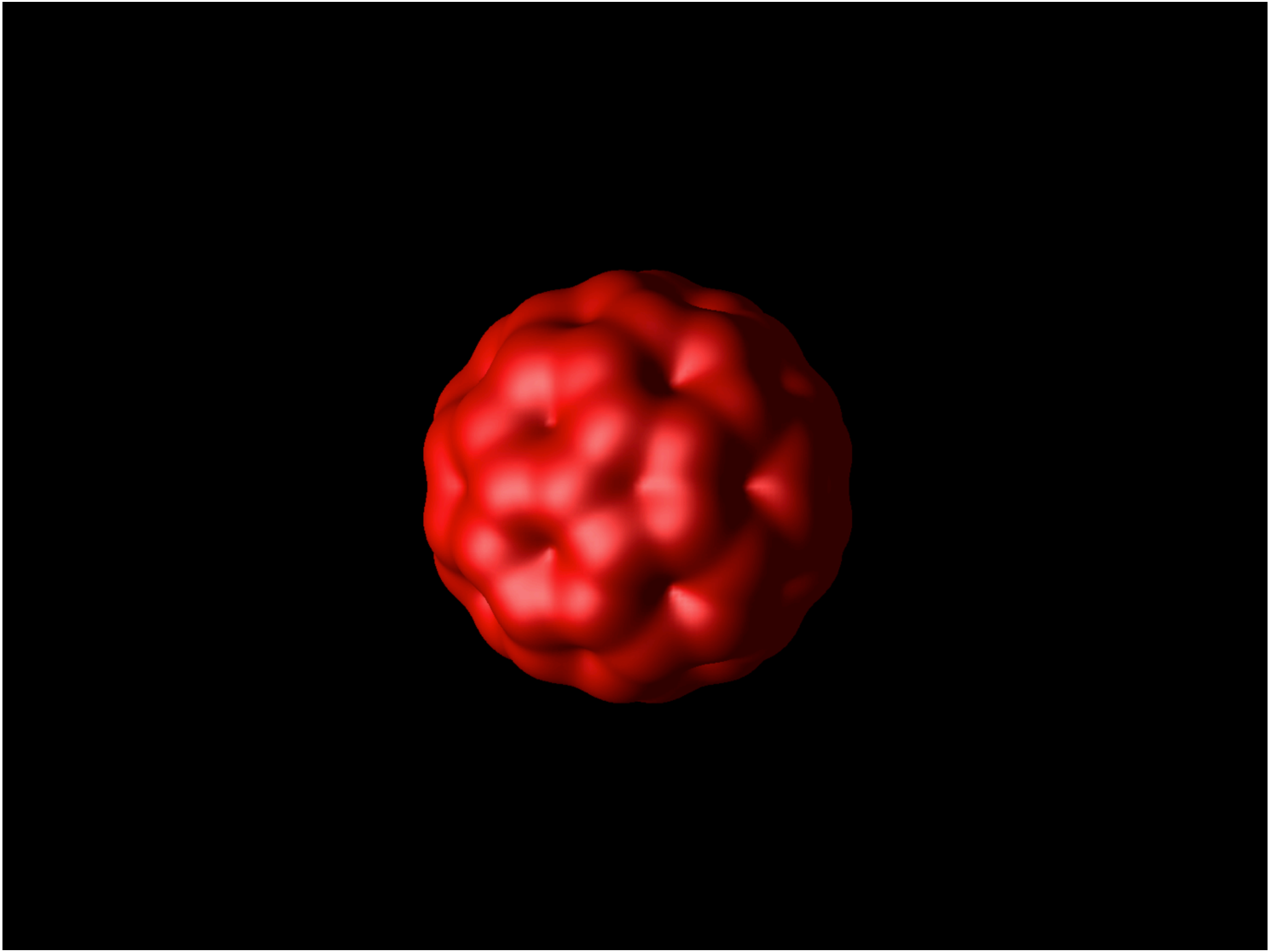
- The problem of multiplying two very large matrices (of order N = 2^m for some integer m) can now be broken down recursively by partitioning the matrices into quarters, sixteenths, etc.
- And note the key point: The savings is not just a factor "7/8"; it is that factor at each hierarchical level of the recursion. In total it reduces the process of matrix multiplication to order $N^{\log_2(7)}$ instead of N^3 . (that's $N^{2.8}$)
- for n=100: 411,822 instead of 1,000,000 (41%)
- for n=10000: 169,044,093,164 instead of 1,000,000,000,000 (17%)
- What about the additions? For large N , it turns out that there are six times as many additions as multiplications. But, if N is very large, this constant factor is no match for the change in the exponent from N^3 to $N^{\log_2 7}$.
- This type of algorithm starts to show up in high level *blas* routines!

PHY-4810
COMPUTATIONAL PHYSICS

LECTURE 14: EIGENVALUE PROBLEM







EIGENVALUE, EIGENVECTORS?

- An eigenvalue and eigenvector of a square matrix A are a scalar λ and a nonzero vector x so that

$$Ax = \lambda x$$

- The eigenvalue-eigenvector equation for a square matrix can be written

$$(A - \lambda I)x = 0, \;\; x \neq 0.$$

- The characteristic equation or characteristic polynomial of A.

$$\det(A - \lambda I) = 0.$$

IN PRACTICE

- **Numerical recipes has a collection of methods to compute the eigenvalues/ eigenvectors (chapter 11)**
- **Here we will employ the methods developed for symmetric matrices**

```
struct Symmeig {
```
eigen_sym.h

Computes all eigenvalues and eigenvectors of a real symmetric matrix by reduction to tridiagonal form followed by QL iteration.

EXAMPLE: SCHROEDINGER EQUATION

- **Time-independent S.E.:**

$$-\frac{\hbar^2}{2m}\nabla^2\psi(x,y,z)+V(x,y,z)\psi(x,y,z)=E\psi(x,y,z)$$

- **Or, equivalently:**

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

- **Finally:**

$$(\frac{\hbar^2}{2m}=1) \qquad (-\nabla^2+V)\psi=E\psi$$

SOLUTION: DISCRETIZATION OF ∇^2

1D:

$$\nabla^2 f = \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2}$$

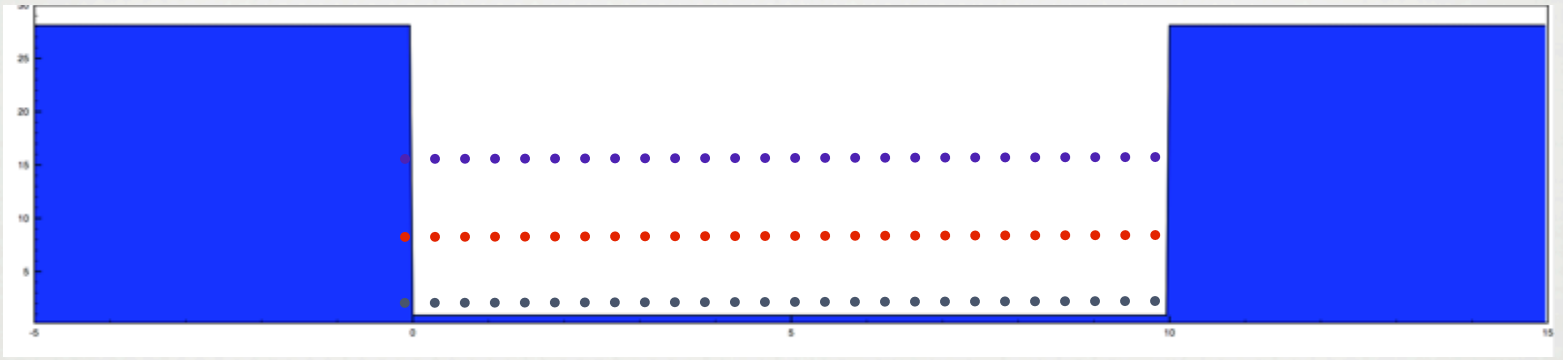
$$(-\nabla^2 + V)f = \frac{-f_{i-1} + (2 + h^2 V_i)f_i - f_{i+1}}{h^2}$$

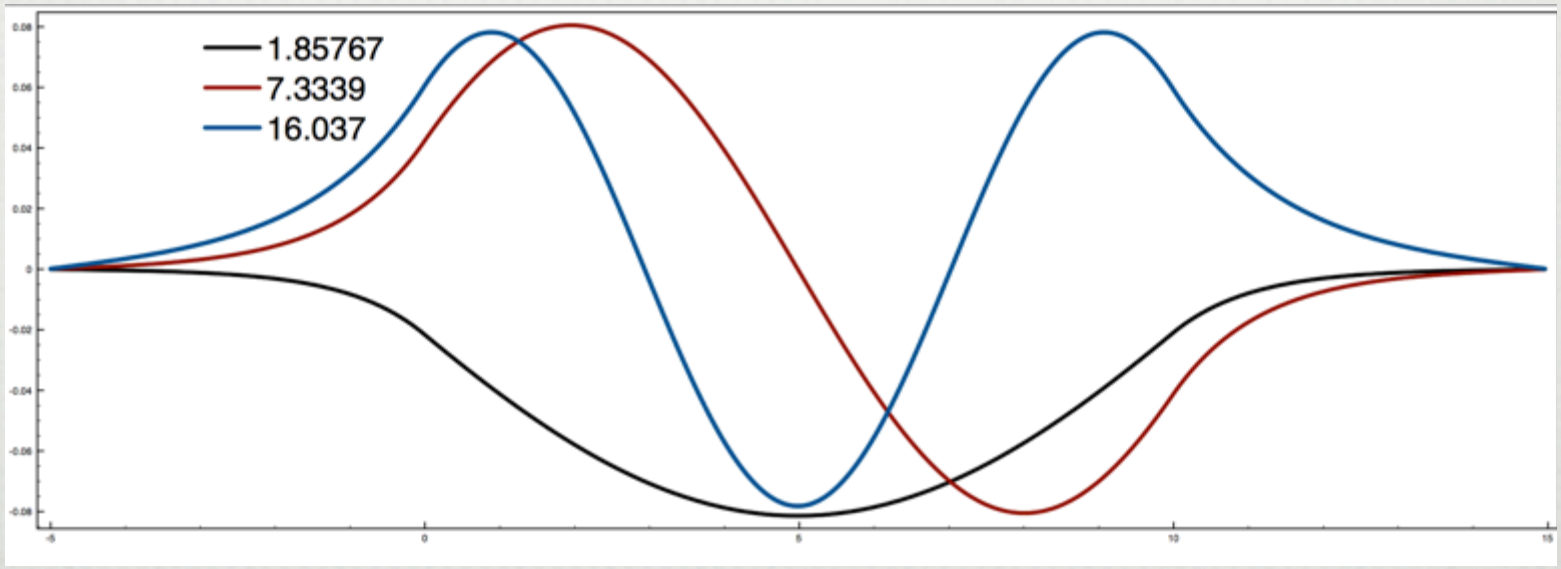
2D:

$$\nabla^2 f = \frac{f_{i-1,j} + f_{i+1,j} - 4f_{ij} + f_{i,j-1} + f_{i,j-1}}{h^2}$$

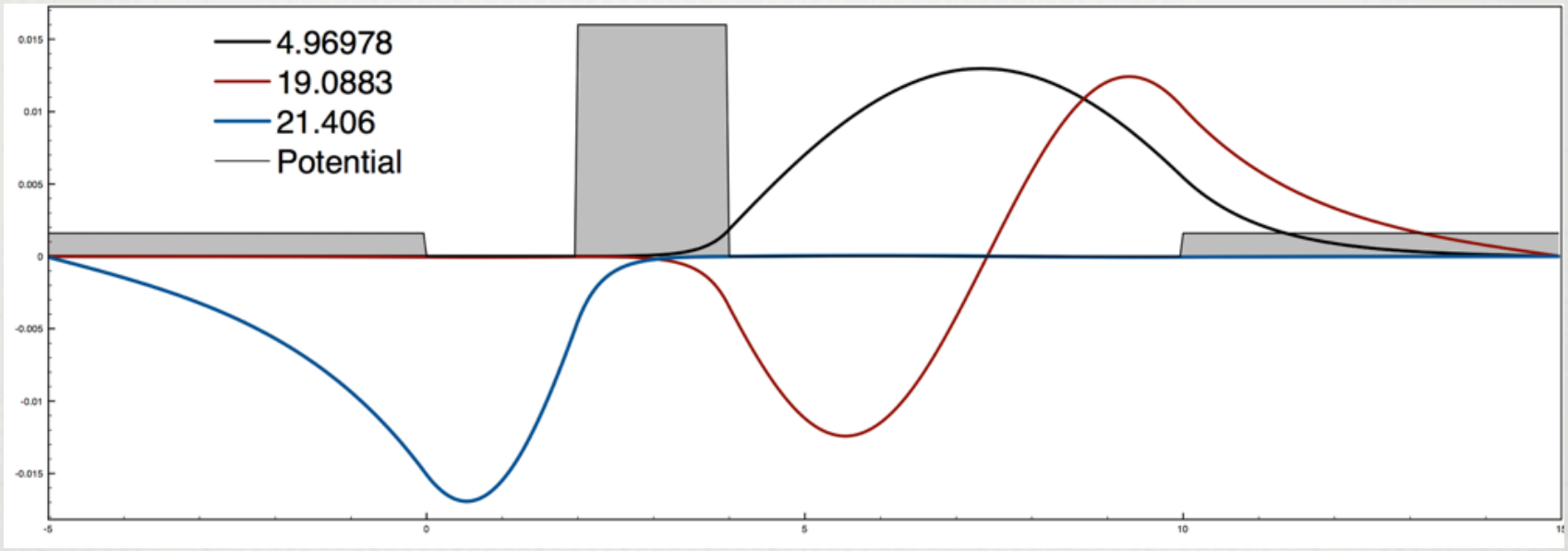
$$((-\nabla^2 + V)f)_{ij} = \frac{-f_{i-1,j} - f_{i+1,j} + (4 + h^2 V_{ij})f_{ij} - f_{i,j-1} - f_{i,j-1}}{h^2}$$

PARTICLE IN A 1D WELL (BOUND STATES)





PARTICLE IN A MULTIPLE-WELL



Amplitude

Position

HOW?

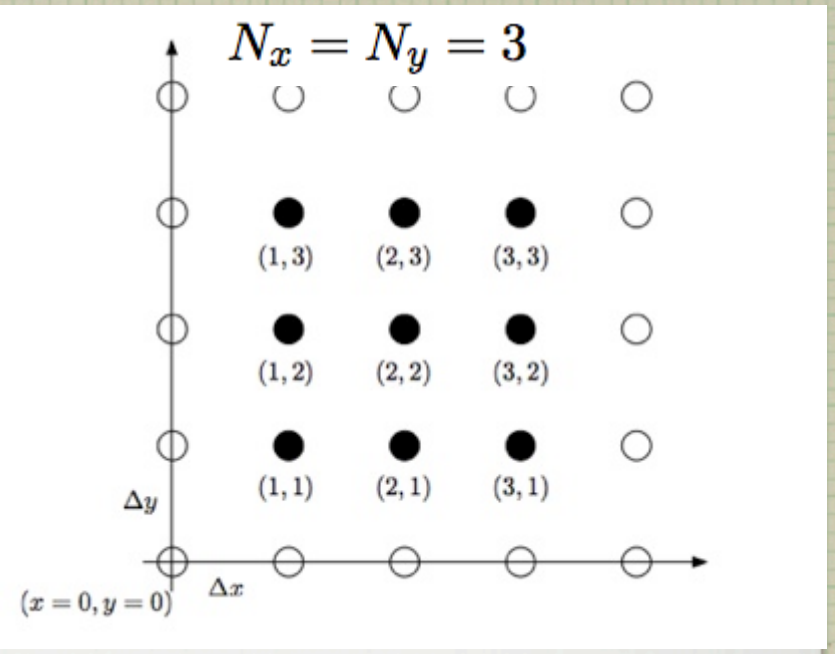
```
//define Hamiltonian
for(int i=0;i<n;i++){
    Ham[i][i]=2.0+Pot[i];
    if(i<n-1) Ham[i][i+1]=-1.;
    if(i>0) Ham[i][i-1]=-1.;
}

Symmeig Schrodinger(Ham);

for(int i=n-10;i<n;i++){
    cout << "i= " << i << " Eigenvalue (eV)=" << hartree*Schrodinger.d[i]/d2 << endl;
}
cout << endl;

ofstream outfile;
outfile.open("wavefunction.dat");
for(int i=0;i<n;i++){
    outfile << (i-n1)*delta << " ";
    for(int j=0;j<10;j++){
       outfile<< Schrodinger.z[i][n-j-1] << " " ;
                    }
    outfile << endl;
}
outfile.close();
```

DISCRETIZED S.E. IN 2D

$$\left[(-\nabla^2 + V)f\right]_{ij} = \frac{-f_{i-1,j} - f_{i+1,j} + (4 + h^2 V_{ij}) f_{ij} - f_{i,j-1} - f_{i,j-1}}{h^2}$$

REMEMBER LAST FRIDAY!

ONE WAY TO BUILD THE MATRIX



$$\partial^2 x \equiv \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}$$

$$\partial^2 y \equiv \begin{pmatrix} -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -2 \end{pmatrix}$$

$$\frac{\partial^2 f(x_n, y_m)}{\partial x^2} \sim \frac{f(x_n - \Delta x, y_m) - 2\, f(x_n, y_m) + f(x_n + \Delta x, y_m)}{\Delta x^2}$$

PUTTING IT ALL TOGETHER (∇^2)

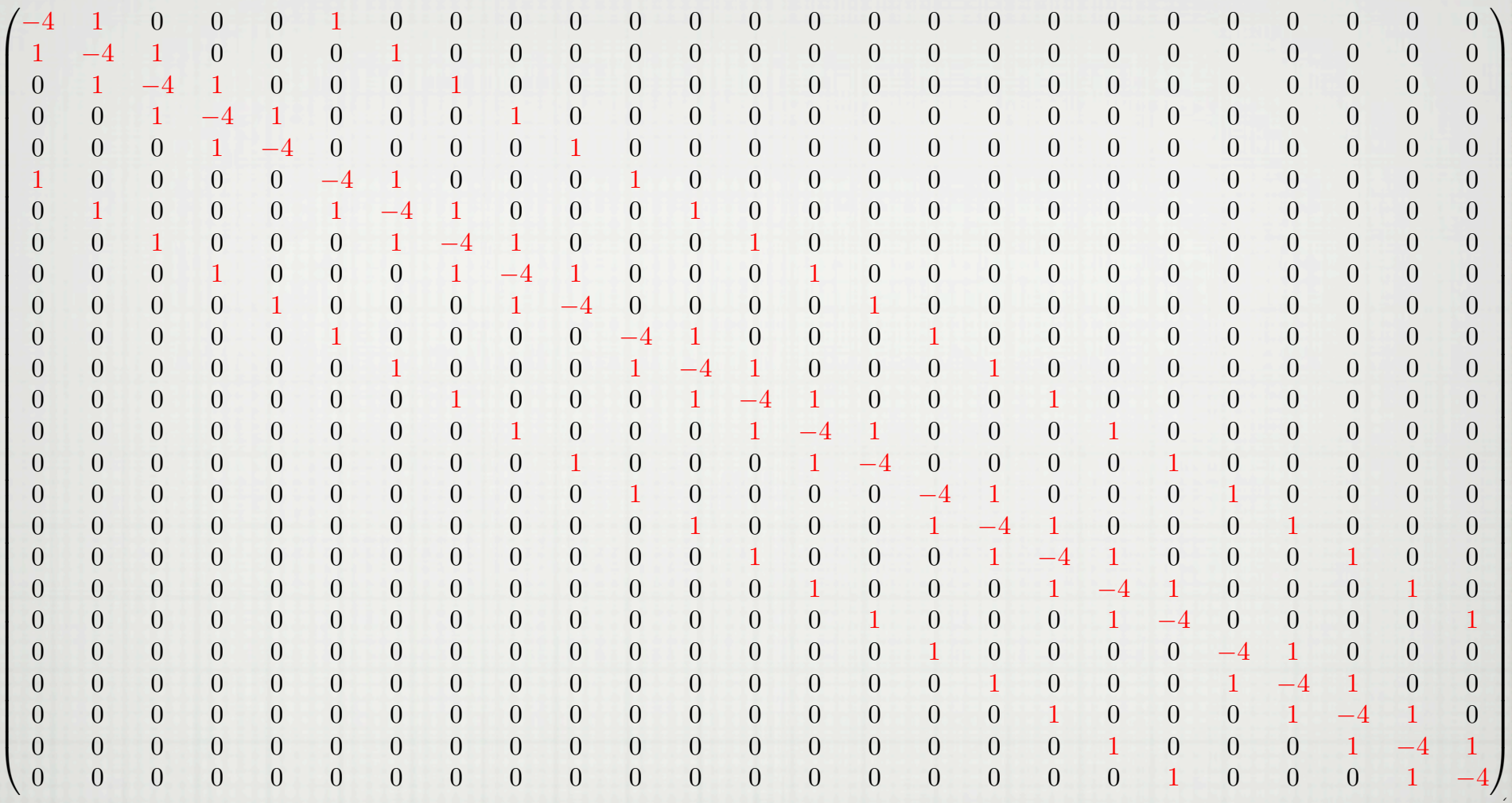
$$\nabla^2 \equiv \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

$$\nabla^2 \equiv \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

$$\begin{pmatrix} -4 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & -4 \end{pmatrix}$$

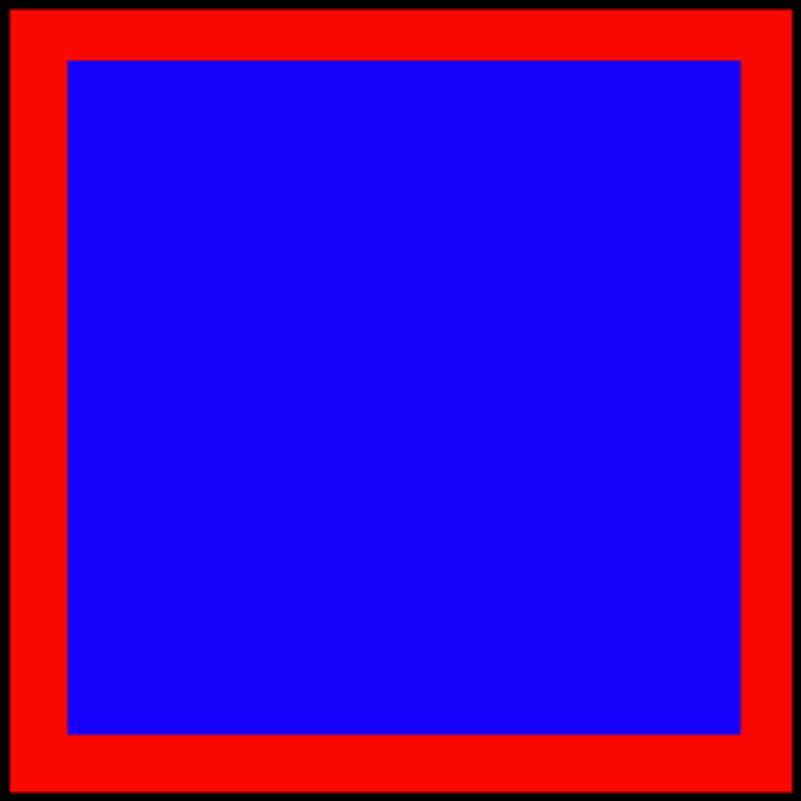
n_y x (n_x x n_x) blocks along the diagonal

blocks are connected by a diagonal of “1”



500 x 500

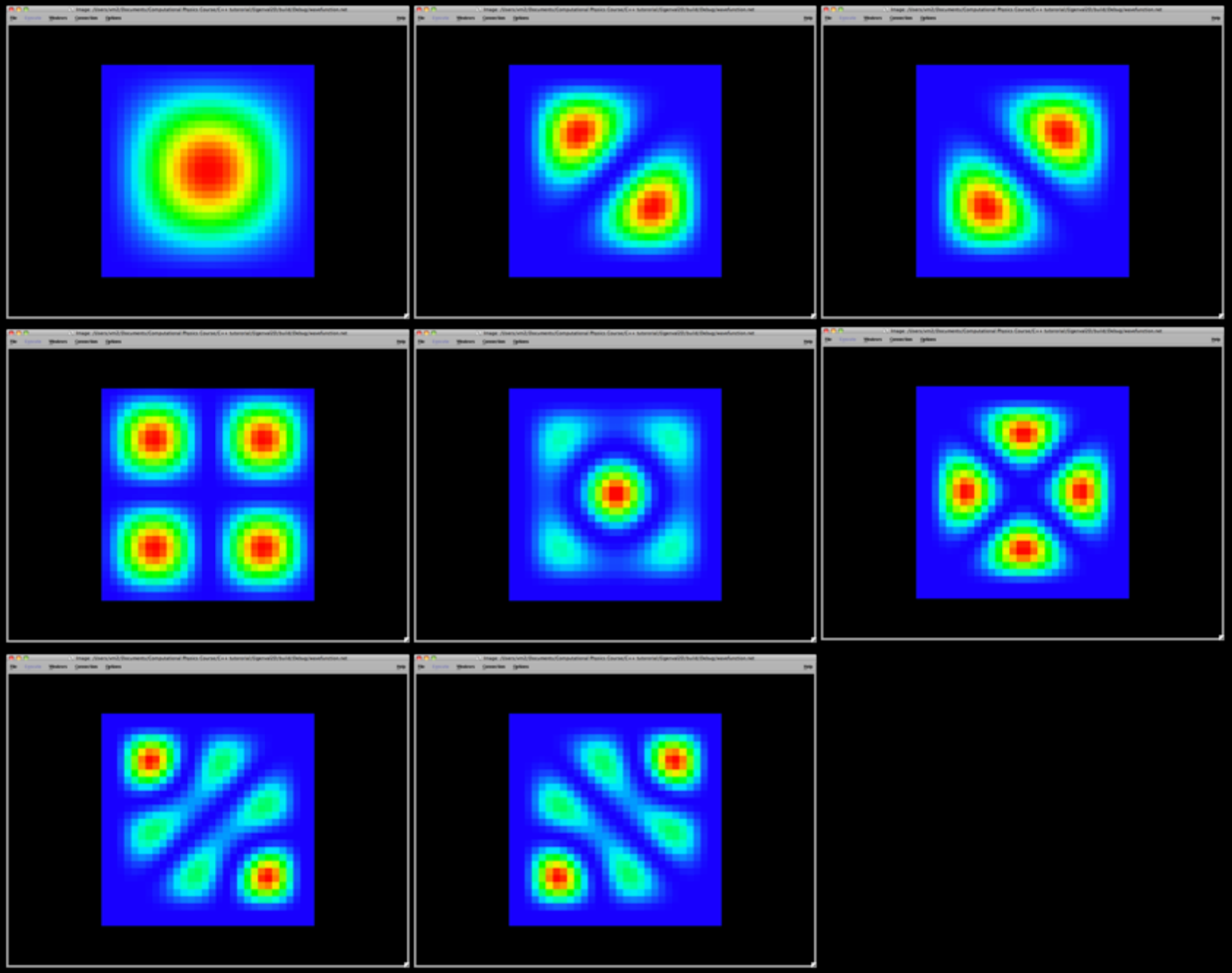
Square well



V=0

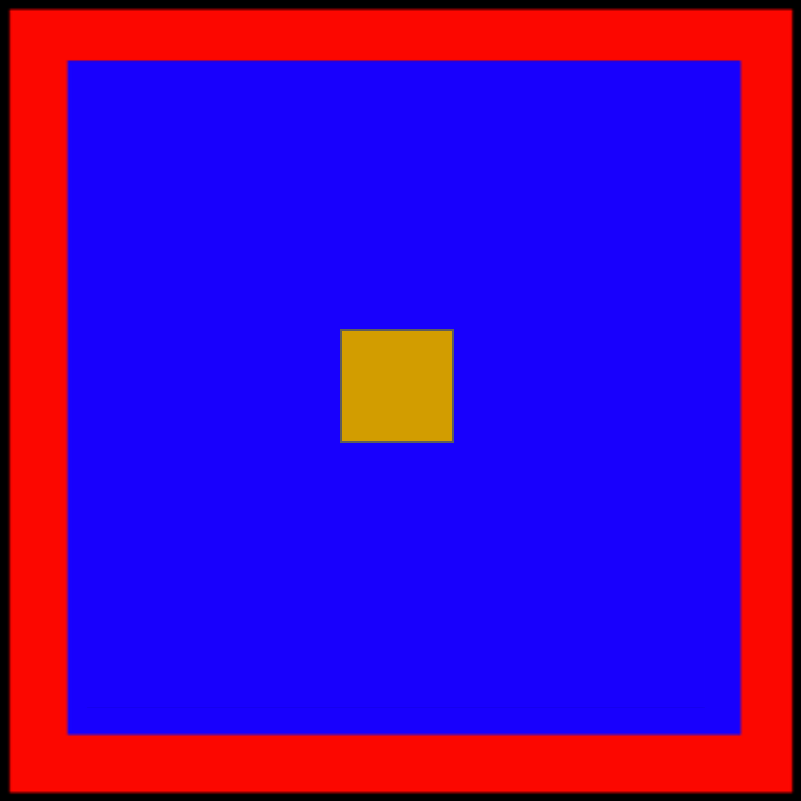
V=100

Square well: wavefunctions



E_i
0.0270
0.0674
0.0674
0.1078
0.1341
0.1341
0.1745
0.1745

Square well with a central pole

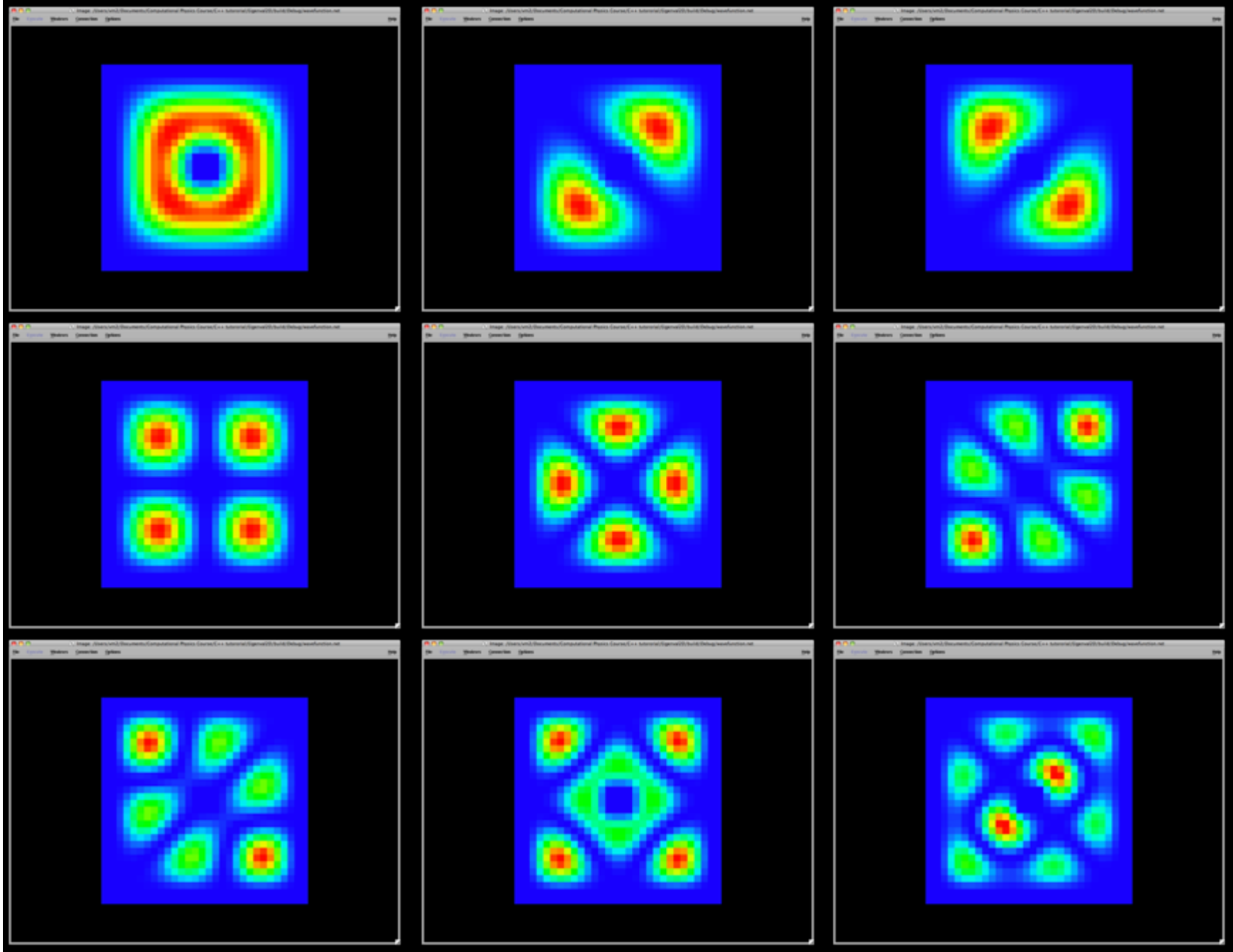


V=0

V=100

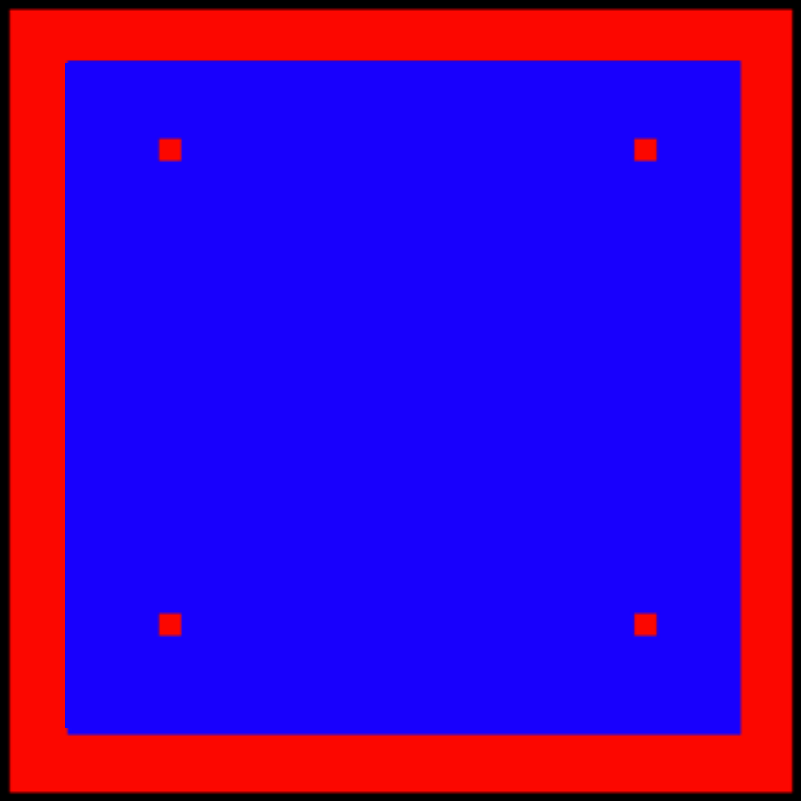
V=10000

Square well with a pole: wavefunctions



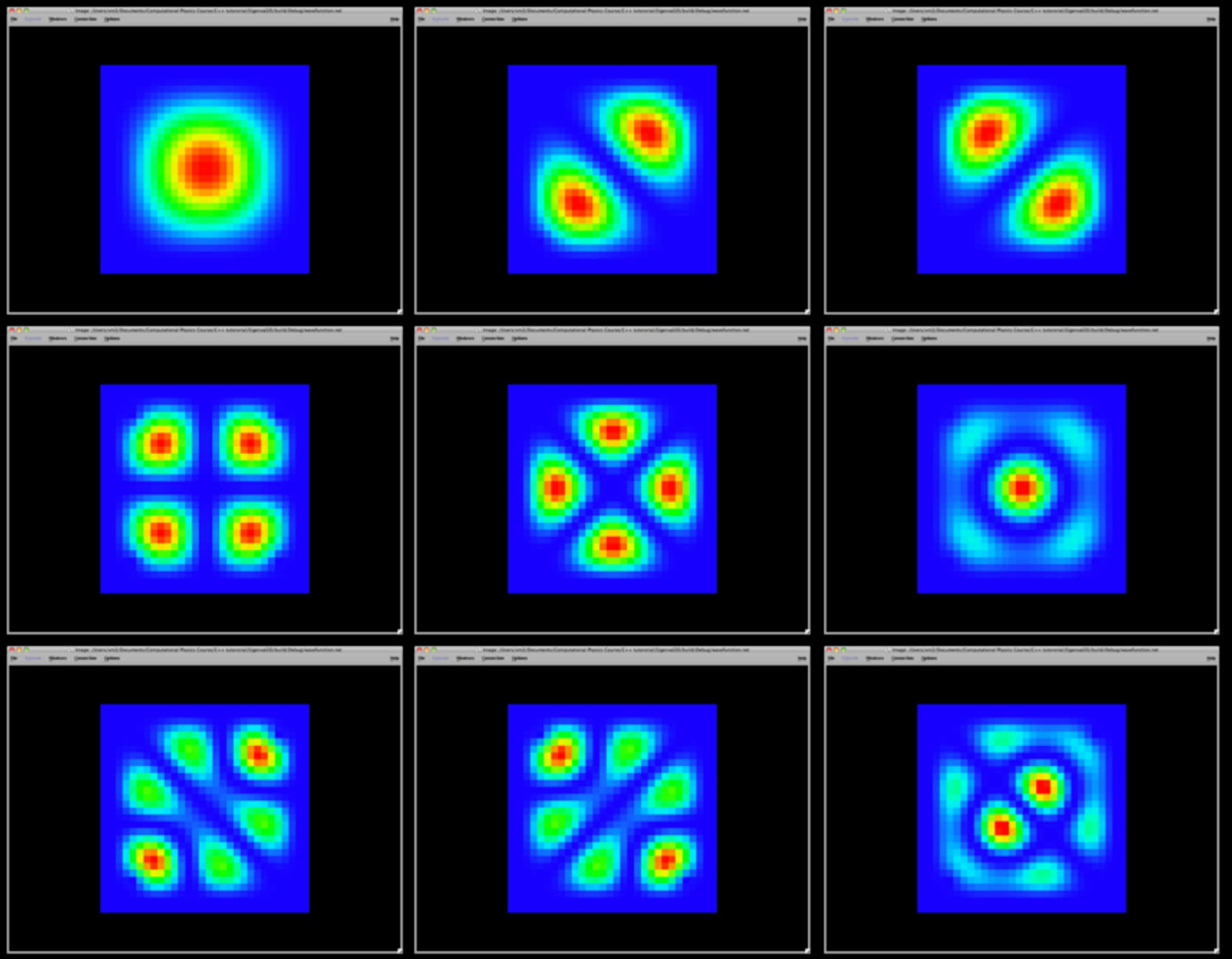
E_l
0.0548
0.0733
0.0733
0.108
0.134
0.178
0.178
0.196
0.246

Square well with perturbations

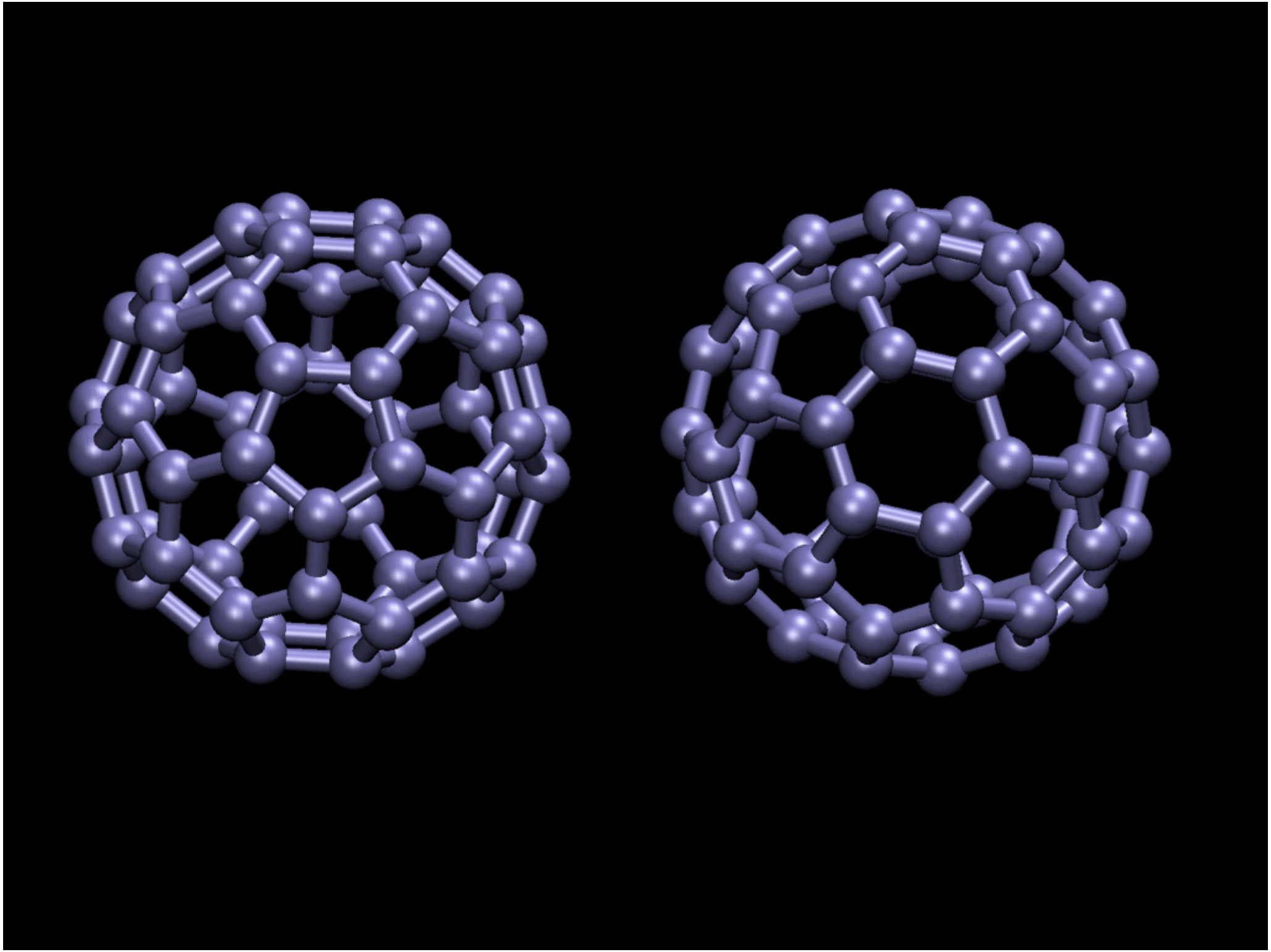


$V=0$

$V=100$



How Schroedinger Equation is solved in quantum chemistry and computational nanoscience!



INTRODUCTION TO LCAO METHOD

$$H|\alpha >= E_\alpha|\alpha >$$

$$|\alpha\rangle = \sum_a c_a|\phi_a\rangle$$

We use atomic orbitals as basis: we suppose that the molecule can be approximated by addition of properties of the materials

$$\sum_a |\phi_a\rangle\langle\phi_a| = 1$$

$$< \phi_a|\phi'_a >= \delta_{aa'}$$

Properties of completeness and orthogonality

$$|\alpha\rangle = \sum_b |\phi_b\rangle\langle\phi_b|\alpha\rangle \qquad c_a = \langle\phi_a|\alpha\rangle$$

$$H|\alpha>=E_\alpha|\alpha> \qquad |\alpha\rangle=\sum_a c_a|\phi_a\rangle$$

$$\sum_a c_a H|\phi_a>=E_\alpha\sum_a c_a|\phi_a>$$

$$<\phi_{a'}|\sum_a c_a H|\phi_a>=E_\alpha\sum_a c_a<\phi_{a'}|\phi_a>$$

$$\sum_a c_a<\phi_{a'}|H|\phi_a>=E_\alpha\sum_a c_a<\phi_{a'}|\phi_a>$$

$$\sum_a c_a H_{aa'}=E_\alpha\sum_a c_a\delta_{aa'}$$

$$\sum_a H_{a'a}c_a=E_\alpha c'_a \qquad \mathcal{H}C=EC$$

HUCKEL THEORY (TIGHT-BINDING)

- **All interactions are limited to neighboring atoms, and are identical**

$$H_{aa'} =< \phi_a|H|\phi'_a >= \begin{cases} 0 \\ \gamma \end{cases}$$

- **The overlap between atoms is zero**

$$< \phi_a|\phi'_a >= \delta_{aa'}$$

IN PRACTICE

```
for(int i=0;i<60;i++){
     for(int j=0;j<60;j++){
        distance=sqrt(pow(r[i][0]-r[j][0],2)+pow(r[i][1]-r[j]
[1],2)+pow(r[i][2]-r[j][2],2));
        if(distance<rcut && distance > 0.1) {
           H[i][j]=gamma0;
        }
     }
  }

  Symmeig Schrodinger(H);
  for(int i=1;i<nat;i++){
     cout << nat-i << " " << Schrodinger.d[i] << endl;
  }
```

PERIODIC TABLE OF THE ELEMENTS

http://www.ktf-split.hr/periodni/en/

PERIOD	GROUP 1 IA	2 IIA	3 IIIB	4 IVB	5 VB	6 VIB	7 VIIB	8	9 VIIIB	10	11 IB	12 IIB	13 IIIA	14 IVA	15 VA	16 VIA	17 VIIA	18 VIIIA
1	1 1.0079 **H** HYDROGEN																	2 4.0026 **He** HELIUM
2	3 6.941 **Li** LITHIUM	4 9.0122 **Be** BERYLLIUM											5 10.811 **B** BORON	6 12.011 **C** CARBON	7 14.007 **N** NITROGEN	8 15.999 **O** OXYGEN	9 18.998 **F** FLUORINE	10 20.180 **Ne** NEON
3	11 22.990 **Na** SODIUM	12 24.305 **Mg** MAGNESIUM											13 26.982 **Al** ALUMINIUM	14 28.086 **Si** SILICON	15 30.974 **P** PHOSPHORUS	16 32.065 **S** SULPHUR	17 35.453 **Cl** CHLORINE	18 39.948 **Ar** ARGON
4	19 39.098 **K** POTASSIUM	20 40.078 **Ca** CALCIUM	21 44.956 **Sc** SCANDIUM	22 47.867 **Ti** TITANIUM	23 50.942 **V** VANADIUM	24 51.996 **Cr** CHROMIUM	25 54.938 **Mn** MANGANESE	26 55.845 **Fe** IRON	27 58.933 **Co** COBALT	28 58.693 **Ni** NICKEL	29 63.546 **Cu** COPPER	30 65.39 **Zn** ZINC	31 69.723 **Ga** GALLIUM	32 72.64 **Ge** GERMANIUM	33 74.922 **As** ARSENIC	34 78.96 **Se** SELENIUM	35 79.904 **Br** BROMINE	36 83.80 **Kr** KRYPTON
5	37 85.468 **Rb** RUBIDIUM	38 87.62 **Sr** STRONTIUM	39 88.906 **Y** YTTRIUM	40 91.224 **Zr** ZIRCONIUM	41 92.906 **Nb** NIOBIUM	42 95.94 **Mo** MOLYBDENUM	43 (98) **Tc** TECHNETIUM	44 101.07 **Ru** RUTHENIUM	45 102.91 **Rh** RHODIUM	46 106.42 **Pd** PALLADIUM	47 107.87 **Ag** SILVER	48 112.41 **Cd** CADMIUM	49 114.82 **In** INDIUM	50 118.71 **Sn** TIN	51 121.76 **Sb** ANTIMONY	52 127.60 **Te** TELLURIUM	53 126.90 **I** IODINE	54 131.29 **Xe** XENON
6	55 132.91 **Cs** CAESIUM	56 137.33 **Ba** BARIUM	57-71 **La-Lu** Lanthanide	72 178.49 **Hf** HAFNIUM	73 180.95 **Ta** TANTALUM	74 183.84 **W** TUNGSTEN	75 186.21 **Re** RHENIUM	76 190.23 **Os** OSMIUM	77 192.22 **Ir** IRIDIUM	78 195.08 **Pt** PLATINUM	79 196.97 **Au** GOLD	80 200.59 **Hg** MERCURY	81 204.38 **Tl** THALLIUM	82 207.2 **Pb** LEAD	83 208.98 **Bi** BISMUTH	84 (209) **Po** POLONIUM	85 (210) **At** ASTATINE	86 (222) **Rn** RADON
7	87 (223) **Fr** FRANCIUM	88 (226) **Ra** RADIUM	89-103 **Ac-Lr** Actinide	104 (261) **Rf** RUTHERFORDIUM	105 (262) **Db** DUBNIUM	106 (266) **Sg** SEABORGIUM	107 (264) **Bh** BOHRIUM	108 (277) **Hs** HASSIUM	109 (268) **Mt** MEITNERIUM	110 (281) **Uun** UNUNNILIUM	111 (272) **Uuu** UNUNUNIUM	112 (285) **Uub** UNUNBIUM		114 (289) **Uuq** UNUNQUADIUM				

RELATIVE ATOMIC MASS (1)
GROUP IUPAC
GROUP CAS
ATOMIC NUMBER
SYMBOL
ELEMENT NAME

Example: 5 10.811 **B** BORON

Metal | Semimetal | Nonmetal
1 Alkali metal
2 Alkaline earth metal
Transition metals
Lanthanide
Actinide
16 Chalcogens element
17 Halogens element
18 Noble gas

STANDARD STATE (25 °C; 101 kPa)
Ne - gas
Ga - liquid
Fe - solid
Tc - synthetic

LANTHANIDE

57 138.91 **La** LANTHANUM	58 140.12 **Ce** CERIUM	59 140.91 **Pr** PRASEODYMIUM	60 144.24 **Nd** NEODYMIUM	61 (145) **Pm** PROMETHIUM	62 150.36 **Sm** SAMARIUM	63 151.96 **Eu** EUROPIUM	64 157.25 **Gd** GADOLINIUM	65 158.93 **Tb** TERBIUM	66 162.50 **Dy** DYSPROSIUM	67 164.93 **Ho** HOLMIUM	68 167.26 **Er** ERBIUM	69 168.93 **Tm** THULIUM	70 173.04 **Yb** YTTERBIUM	71 174.97 **Lu** LUTETIUM

ACTINIDE

89 (227) **Ac** ACTINIUM	90 232.04 **Th** THORIUM	91 231.04 **Pa** PROTACTINIUM	92 238.03 **U** URANIUM	93 (237) **Np** NEPTUNIUM	94 (244) **Pu** PLUTONIUM	95 (243) **Am** AMERICIUM	96 (247) **Cm** CURIUM	97 (247) **Bk** BERKELIUM	98 (251) **Cf** CALIFORNIUM	99 (252) **Es** EINSTEINIUM	100 (257) **Fm** FERMIUM	101 (258) **Md** MENDELEVIUM	102 (259) **No** NOBELIUM	103 (262) **Lr** LAWRENCIUM

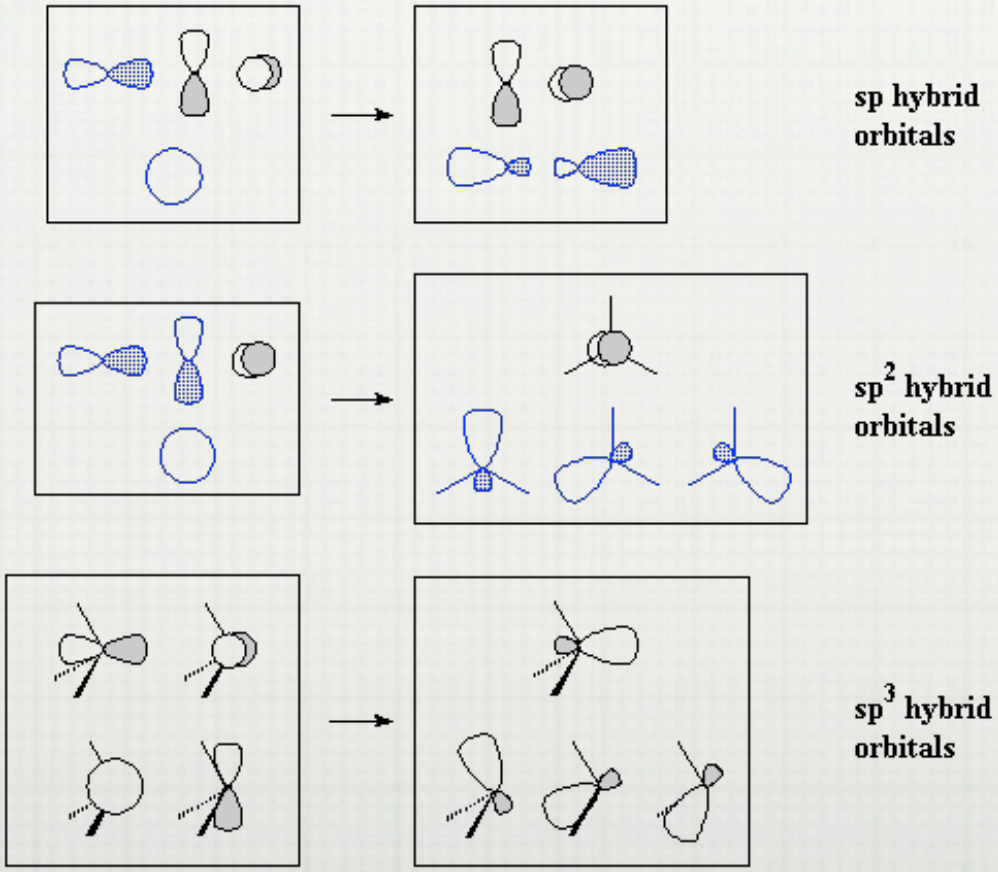
(1) Pure Appl. Chem., 73, No. 4, 667-683 (2001)

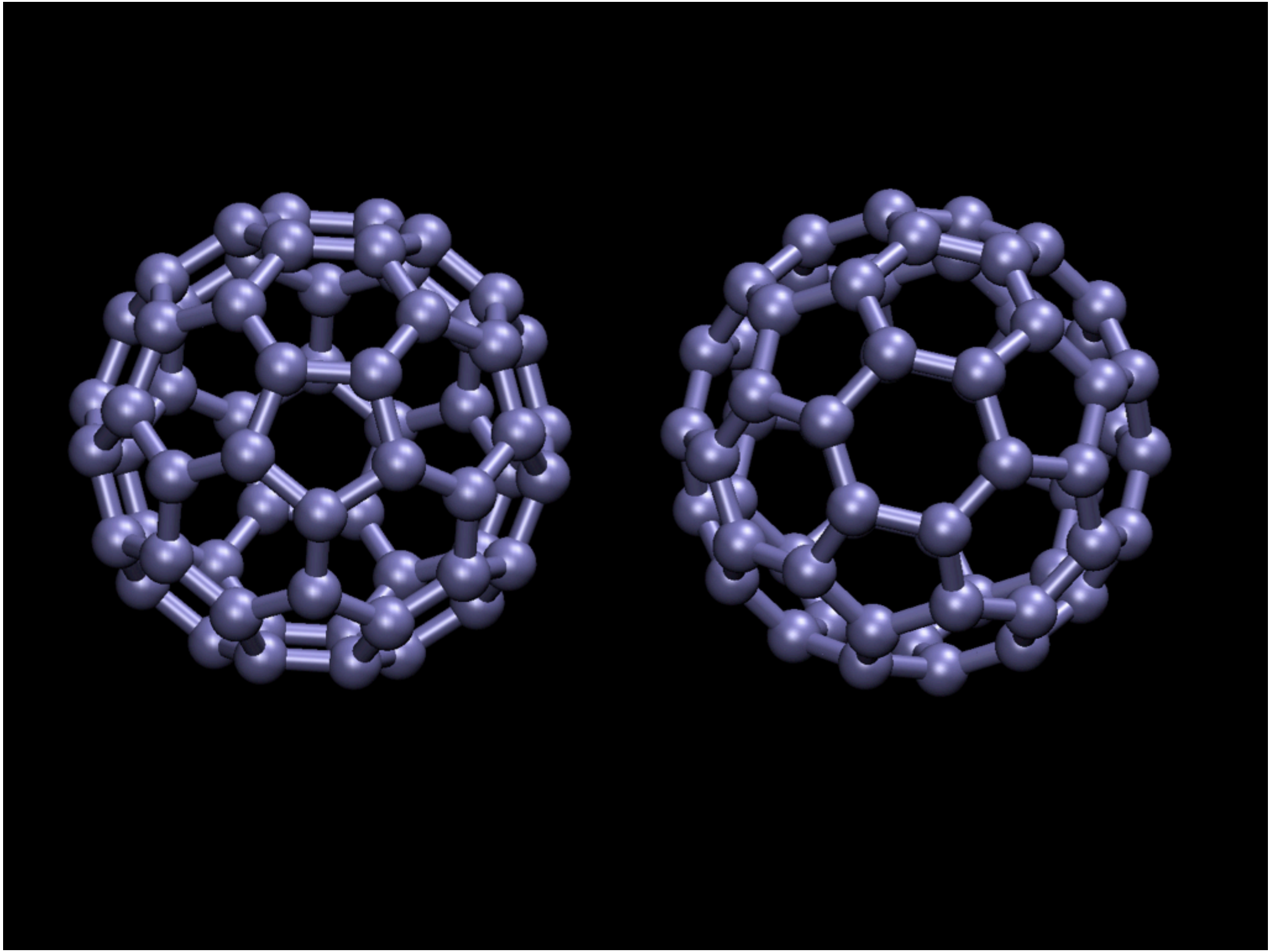
Relative atomic mass is shown with five significant figures. For elements have no stable nuclides, the value enclosed in brackets indicates the mass number of the longest-lived isotope of the element.

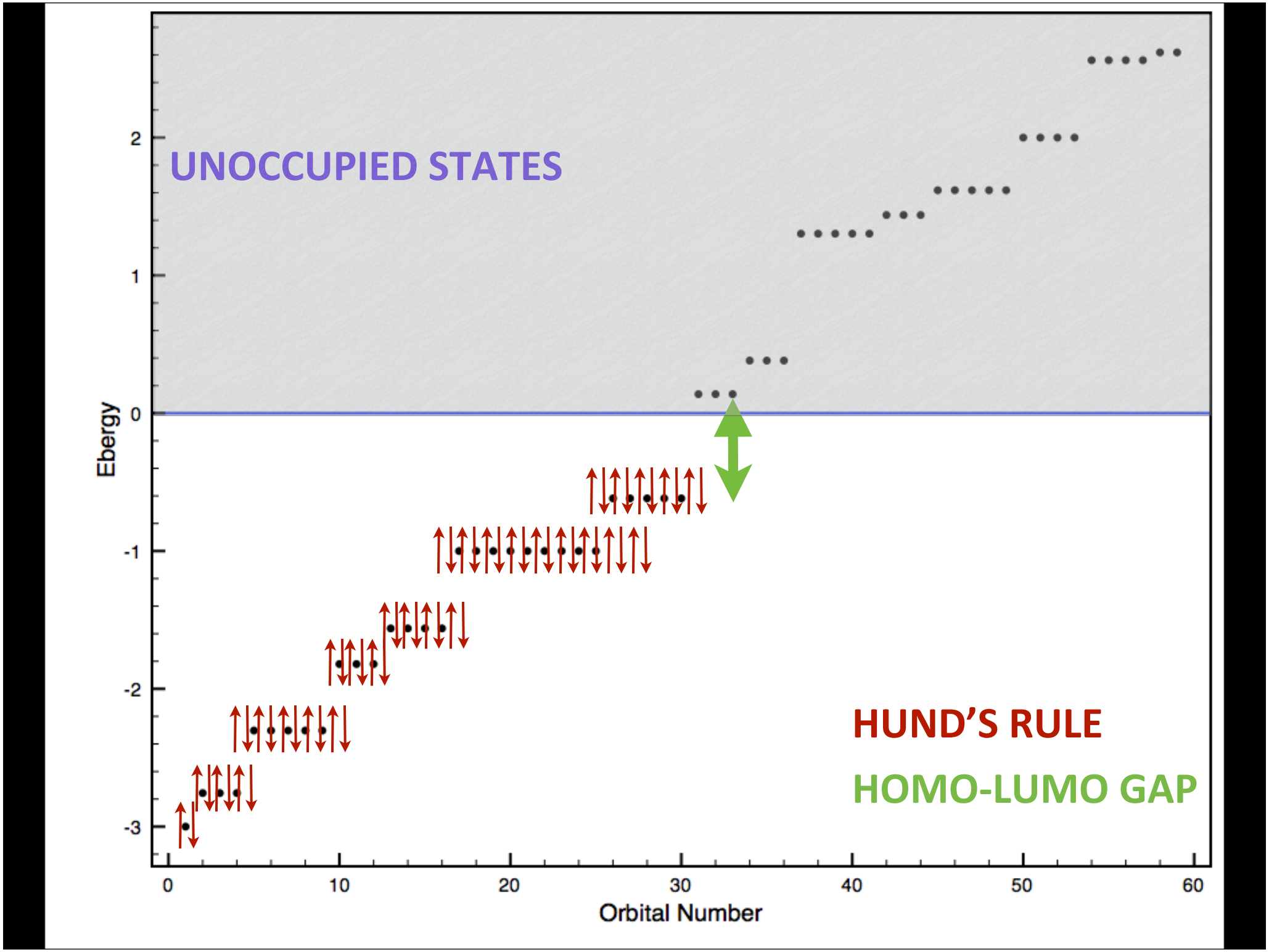
However three such elements (Th, Pa, and U) do have a characteristic terrestrial isotopic composition, and for these an atomic weight is tabulated.

Editor: Aditya Vardhan (adivar@nettlinx.com)

CARBON SP2


sp hybrid orbitals
sp2 hybrid orbitals
sp3 hybrid orbitals

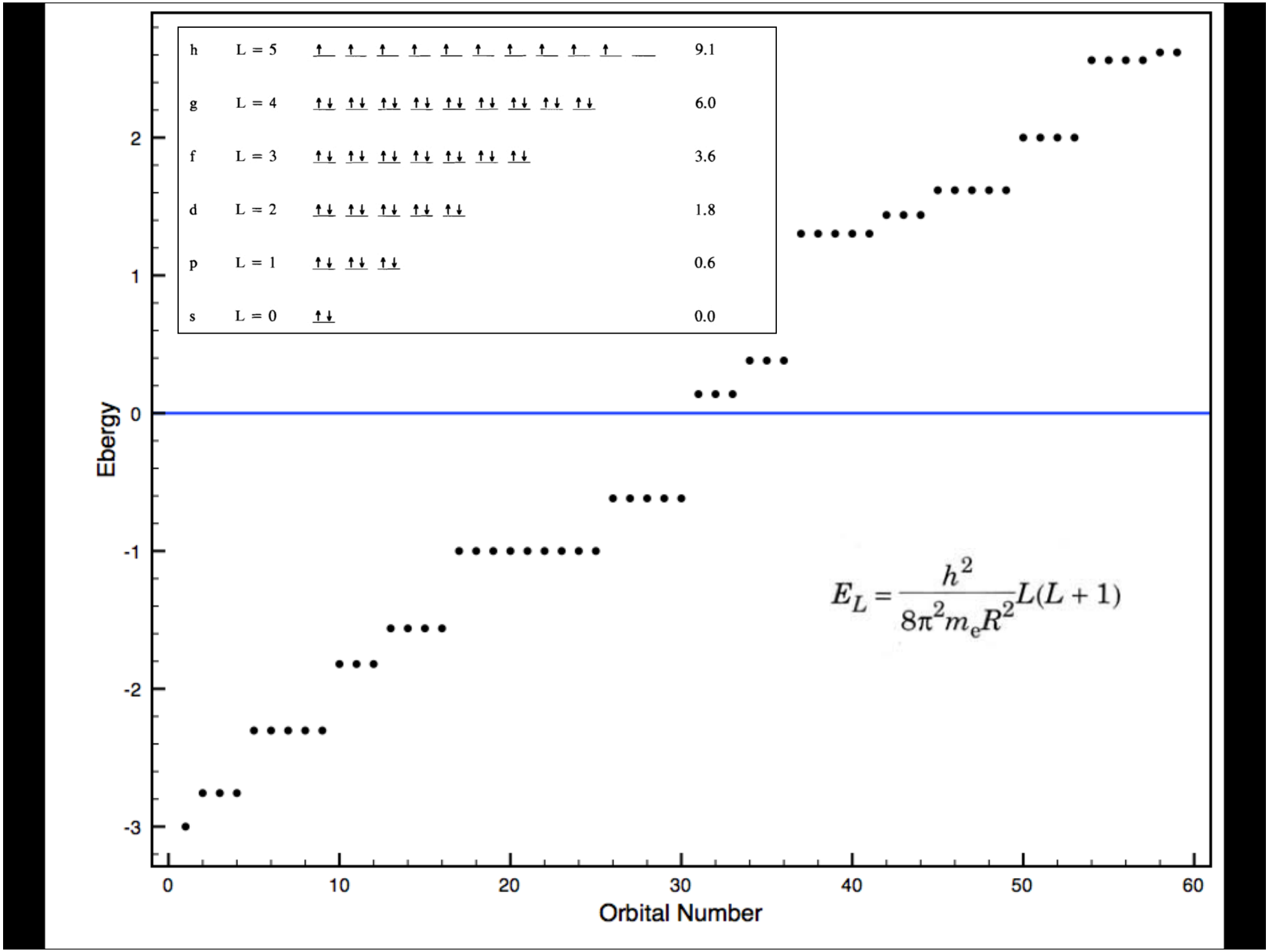



UNOCCUPIED STATES
HUND'S RULE
HOMO-LUMO GAP
Ebergy
2
1
0
-1
-2
-3
0
10
20
30
40
50
60
Orbital Number

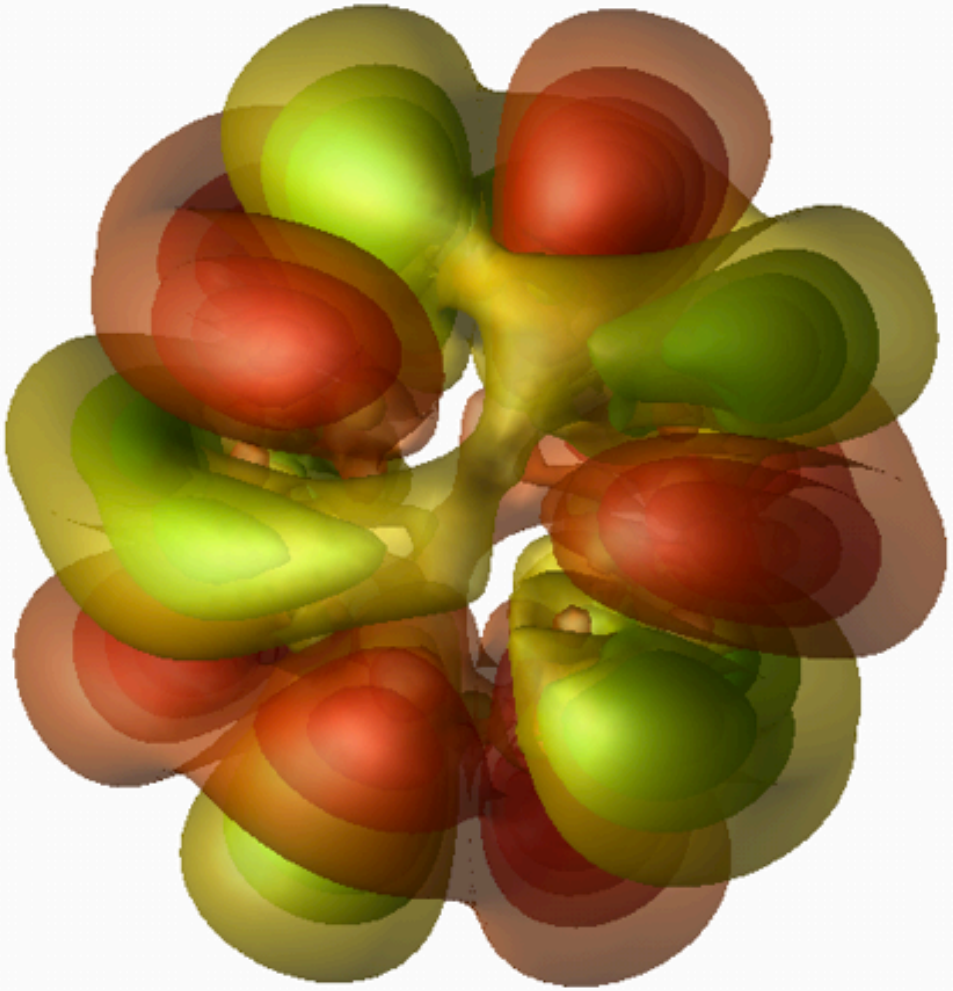
SPHERICAL SHELL MODEL

h	L = 5	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	9.1
g	L = 4	↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓	6.0
f	L = 3	↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓ ↑↓	3.6
d	L = 2	↑↓ ↑↓ ↑↓ ↑↓ ↑↓	1.8
p	L = 1	↑↓ ↑↓ ↑↓	0.6
s	L = 0	↑↓	0.0

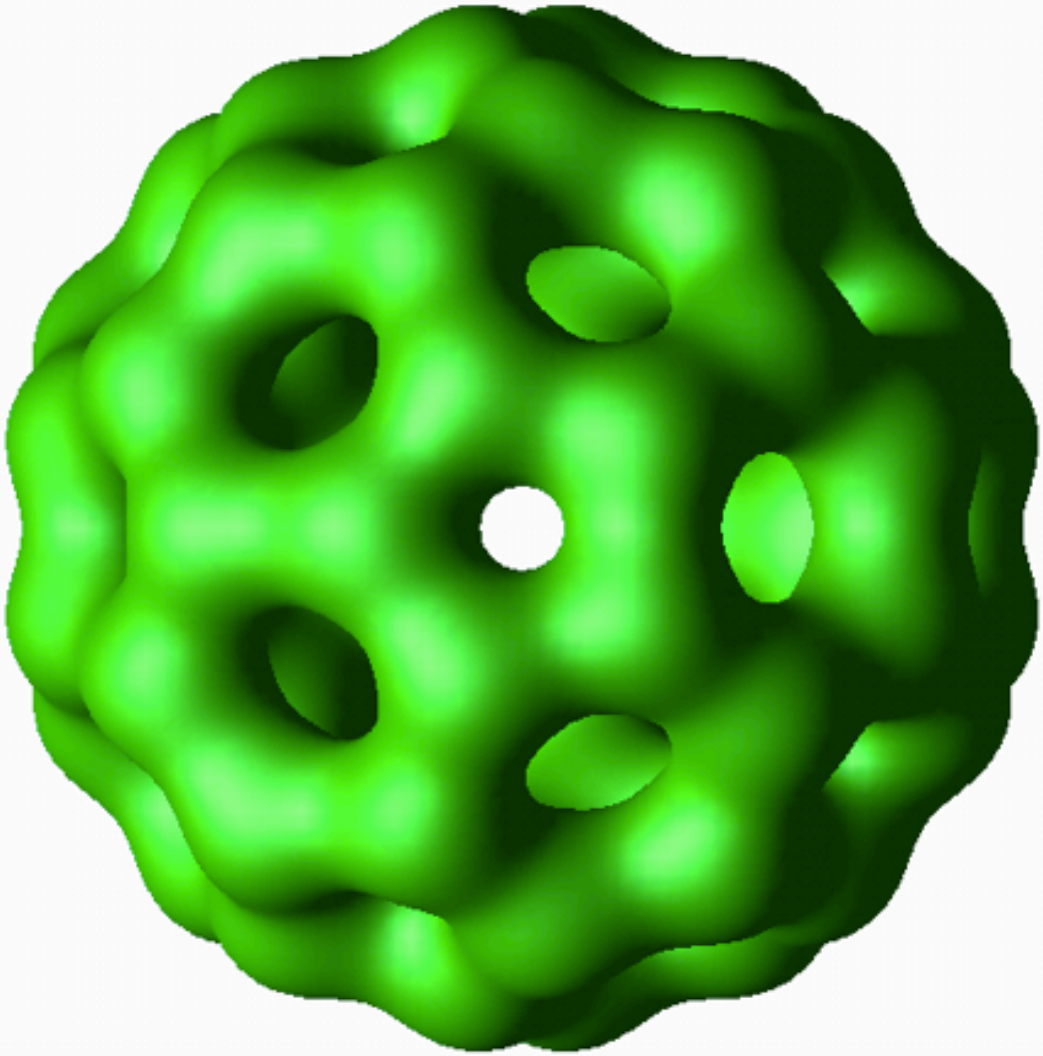
$$E_L = \frac{h^2}{8\pi^2 m_e R^2} L(L+1)$$

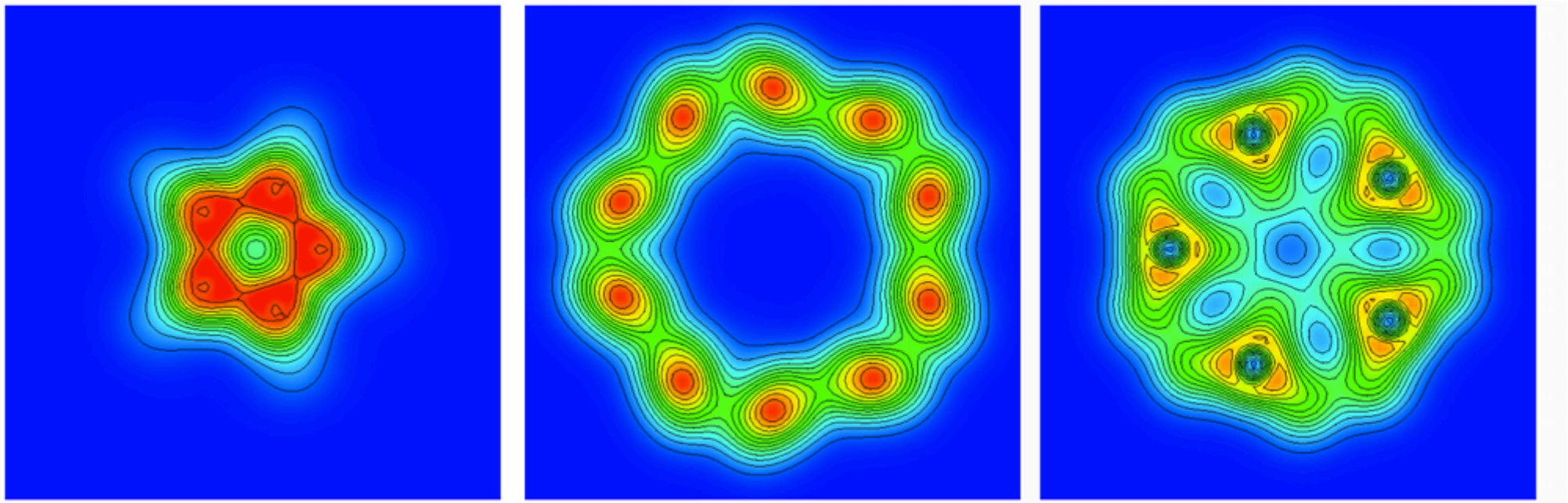

h L = 5 9.1
g L = 4 6.0
f L = 3 3.6
d L = 2 1.8
p L = 1 0.6
s L = 0 0.0
$E_L = \frac{h^2}{8\pi^2 m_e R^2} L(L+1)$
Ebergy
Orbital Number
2
1
0
-1
-2
-3
0
10
20
30
40
50
60

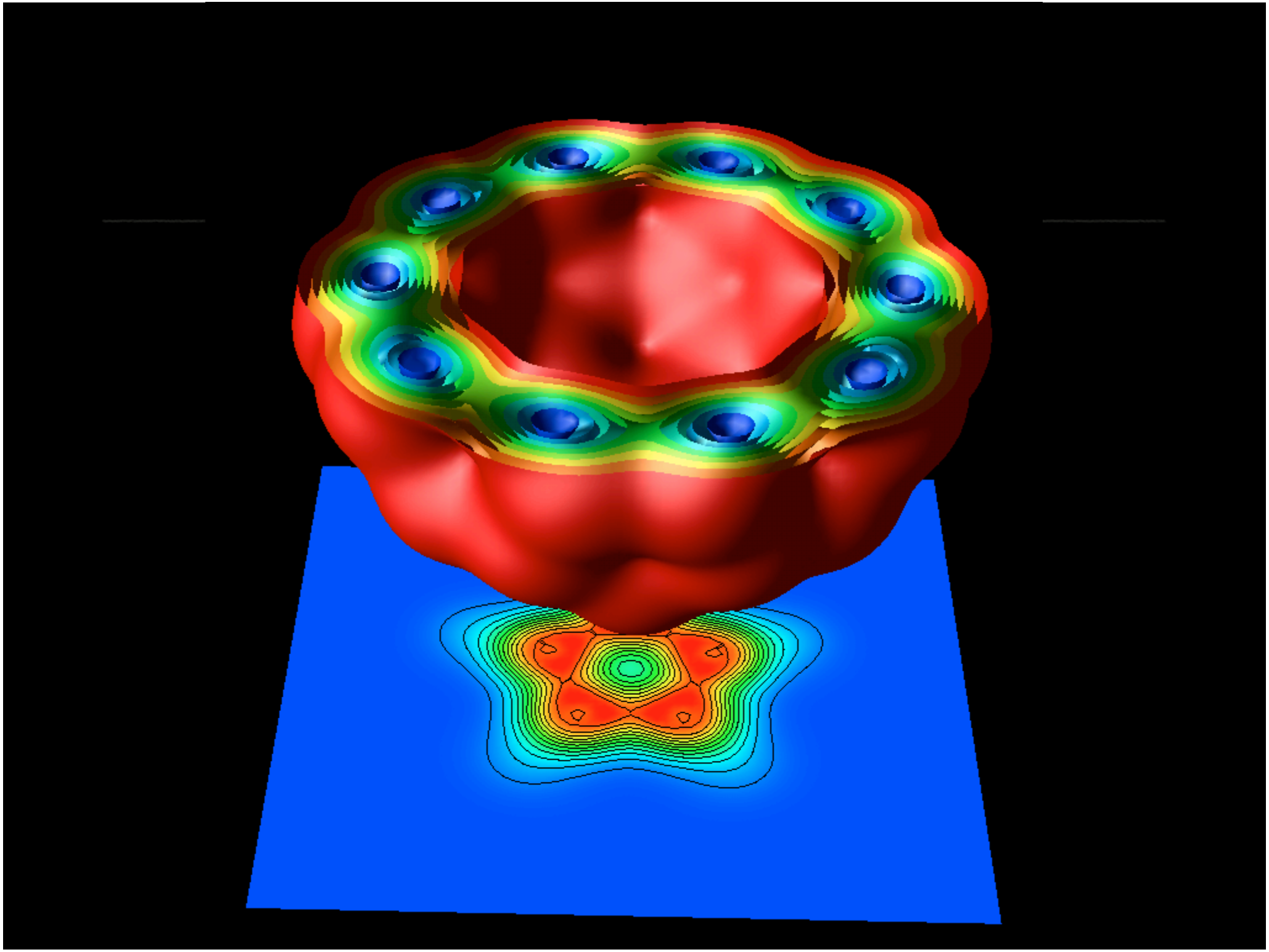
MOLECULAR ORBITAL

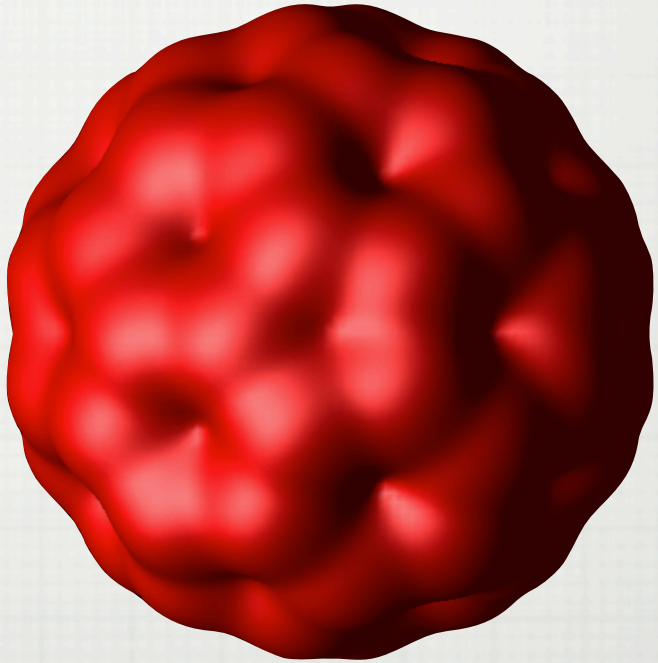


DENSITY OF STATES










HAVE A GOOD DAY