

# Optimization Notes

*L. D. Marks, November 2004*

## 1. Introduction

Most of the more complicated structures have free internal structural parameters, which can either be taken from experiment or optimized using the calculated **forces on the nuclei**. An example of a minimization is provided in the **User Guide** for TiO<sub>2</sub>. The shell script **min\_lapw** is provided which, together with the program **mini**, automatically determines the equilibrium position of all individual atoms (obeying the symmetry constraints of the space group in the **case.struct** file).

A typical sequence of commands for an optimization of the internal positions would look like:

- \* Generate struct file
- \* init lapw
- \* run lapw -fc 1 [other options] (this may take some time)
- \* Inspect the scf file whether you have significant forces (usually at least .gt. 5 mRy/bohr), otherwise you are more or less at the optimal positions
- \* min lapw [options] (this may take some time)

Without -NI switch **min\_lapw** performs an initialization first:

- \* generates default **case.inM** (if not present);
- \* removes "histories" (case.broyd\*, case.tmpM) if present;
- \* copies **.min\_hess** to **.minrestart** (if present from previous min).

When **case.scf** is not present, an scf-cycle will be performed first, otherwise the corresponding forces are extracted into **case.finM** and **mini** generates a new **case.struct** with modified atomic positions. The previous step is saved under **case\_1/2/3...**. Then a new scf-cycle is executed and this loop continues until convergence (default: forces below 2mRy/bohr) is reached. The last iteration of each geometry step is appended to **case.scf\_mini**, so that this file contains the complete history of the minimization and can be used to monitor the progress (grep :ENE \*mini; or :FORxxx ...).

Note, **mini** requires an input file **case.inM** which is created automatically and **MUST NOT** be changed while **min\_lapw** is running (except the force tolerance, which terminates the optimization). The PORT minimization method, a reverse-communication trust-region Quasi-Newton method from the Port library seems to be stable, efficient and does not depend too much on the users input. The PORT option also produces a file **.min\_hess**, which contains the (approximate) Hessian matrix (lower-triangle Cholesky factor) If you restart a minimization with different k-points, RMT, RKmax, ... or do a similar calculation (eg. for a different volume, ...) it will be copied to **.minrestart** (unless -nohess is specified), so that you start with a reasonable approximation for the Hessian. When using PORT you may also want to check its progress using

grep -e :LABEL case.outputM

where :LABEL can be any of

- :ENE (should decrease overall, but can go up for single steps),
- :GRAD (should also go down, but could sometimes also go up for some time as long as the energy still decreases),
- :MIN (provides a condensed summary of the progress),
- :WARN (may indicate a problem),
- :DD (provides information about the step sizes and mode used).

Some general explanations:

1) The algorithm takes steps along what it considers are good directions (using some internal logic), provided that these steps are smaller than what is called the trust-region radius. After a good step (e.g. large energy decrease) it expands the trust-region; after a bad one it reduces it. Sometimes it will try too large a step then have to reduce it, so the energy does not always go down. You can see this by using "grep -e :DD" and "grep -e :MIN case.outputM".

2) A grep on :MIN gives a condensed progress output, in which the most significant terms are E (energy in some rescaled units), RELDF (last energy reduction), PRELDF (what the algorithm predicted for the step), RELDX (RMS change in positions in Angstroms) and NPRELDF (predicted change in next cycle). Near the solution RELDF and RELDX should both become small. However, sometimes you can have soft modes in your structure in which case RELDX will take a long time before it becomes small.

3) A warning that the step was reduced due to overlapping spheres if it happens only once (or twice) is not important; the algorithm tested too large a step. However, if it occurs many times it may indicate that the RMT's are too big.

4) A warning "CURVATURE CONDITION FAILED" indicates that you are still some distance from the minimum, and the Hessian is changing a lot. If you see many of these, it may be that the forces and energy are not consistent. For instance, with a too small an RKMAX, there can be an error of 10 mRy/bohr in the forces.

The main control file is case.inM which has the format:

```
----- top of file: case.inM -----  
PORT 2.0          (PORT tolf (a4,f5.2))  
1.0 1.0 1.0 3.0   ( 1..3:DELTA, eta)  
1.0 1.0 1.0 6.0   ( 1..3=0 constraint)  
----- bottom of file -----
```

Interpretive comments on this file are as follows.

**line 1:** format(a4,f5.2)

MINMOD Modus of the calculation, here PORT

TOLF Force tolerance, geometry optimization will stop when all forces are below TOLF.

**line 2:** free format

DELTA(1-3) Precondition parameters which primarily influence the size of the first geometry step. Within the code the x,y,z forces are multiplied by DELTA(1), DELTA(2), DELTA(3) respectively before being used. DELTA(i) = 0 will therefore constrain the corresponding i-th coordinate. The deltas correspond to the global coordinates (the same as the positions in **case.struct** and the forces **:FGL** from **case.scf**).

ETA: Bond-order parameter: should be set approximately to the number of nearest neighbors (or left at 1.). The diagonal elements of the initial Hessian approximation (if **.minrestart** is not present) for each atom are multiplied by ETA.

>>> **line 2:** must be repeated for every atom

## 2. Optimization Method

While there is no need to understand in detail the workings of the port optimization algorithm in order to use it, some understanding is needed in order to exploit it to its fullest potential. For this, a very brief introduction to some elements of optimization.

Almost all optimization codes use what is called a quadratic approximation, namely they expand the energy in a form

$$E^* = E + \mathbf{g}^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H} \mathbf{s} \quad [1]$$

where  $E^*$  is the predicted energy for a step  $\mathbf{s}$  from the current point,  $E$  and  $\mathbf{g}$  are the energy and gradient (negative of the force) calculated at the current point and  $\mathbf{H}$  is the Hessian matrix. Different algorithm use different approaches to the Hessian matrix. The most primitive is steepest descent, which takes  $\mathbf{H}$  as the unitary matrix so will take a step along the direction of the force. Better algorithms such as conjugant gradient methods use some information about the previous step. By far and away the most common method is to exploit the Hessian, either by directly computing it (very CPU expensive for codes such as Wien) or to create an estimate of it that improves as the calculation proceeds. The most successful approach is the Broyden-Fletcher-Goldberg-Shamo (BFGS) update. If we take a step  $\mathbf{s}_k$ , the gradient will change from  $\mathbf{g}_k$  at the previous point to  $\mathbf{g}_{k+1}$  at the new point, and we can write

$$\mathbf{g}_k - \mathbf{g}_{k+1} = \mathbf{y}_k = \mathbf{H}_k \mathbf{s}_k \quad [2]$$

In principle there are many ways to exploit this information. The method used in BFGS is to update the Hessian for the next step via:

$$\begin{aligned} \mathbf{H}_{k+1} &= \mathbf{H}_k + \Delta \mathbf{H}_k \\ \Delta \mathbf{H}_k &= - \left( \begin{array}{cc} \mathbf{y}_k \mathbf{y}_k^T & \mathbf{H}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{H}_k \\ \mathbf{s}_k^T \mathbf{y}_k & \mathbf{s}_k^T \mathbf{H}_k \mathbf{s}_k \end{array} \right) \end{aligned} \quad [3]$$

The procedure is then to solve for

$$\mathbf{s}_{k+1} = -\mathbf{H}_{k+1}^{-1} \mathbf{g}_k \quad [4]$$

move by  $\mathbf{s}_{k+1}$ , recalculate the gradient, update the Hessian  $\mathbf{H}$  and iterate. Often the first estimate for the Hessian is the unitary matrix, although it does not have to be and the better the initial guess is, the faster the algorithm will converge. Often the estimate of the Hessian will change rather a lot during the calculation, and at some locations can be rather bad. The power of the BFGS method is that experience over the last decade has indicated that in most cases it will correct itself rather quickly, and is therefore rather robust.

However, some care is needed. Near the minimum the Hessian must be positive definite. Away from the minimum the true Hessian does not have to be. Unfortunately, if it is not positive definite the solution to equation [4] may be an uphill direction and the update procedure can go badly wrong. Going back to equation 2, we can rephrase this as:

$$\mathbf{s}_k^T \mathbf{y}_k = \mathbf{s}_k^T \mathbf{H} \mathbf{s}_k \quad [5]$$

If  $\mathbf{H}$  is positive definite the right-hand side is positive so  $\mathbf{s}_k^T \mathbf{y}_k > 0$ . This is called the curvature condition. If it is not met (i.e.  $\mathbf{s}_k^T \mathbf{y}_k < 0$  at some stage) precautions have to be taken to prevent the Hessian from going bad; this is flagged in **mini** as a warning for the user during an optimization run. There is also the possibility that  $\mathbf{s}_k^T \mathbf{H} \mathbf{s}_k$  might become small relative to  $\mathbf{s}_k^T \mathbf{y}_k$ , which can lead to ill-conditioning (this is also flagged as a warning in **mini**).

One other point merits mention. Moving by a full step  $\mathbf{s}_{k+1}$  is often not appropriate; it may be too large. One approach would be to search along the direction of  $\mathbf{s}_{k+1}$ , but this can be inefficient since it would involve many calculations along a single direction. An alternative approach is to use what is called a **Trust-Region** method. Here one calculates the best step for a quadratic model with the current approximation for the Hessian with the additional constraint that  $\|\mathbf{s}_{k+1}\| \leq R$  where “R” is the trust region radius. If a very good step is chosen, the current approximation for the Hessian is good so it is safe to increase the radius; if the step is poor (for instance the energy increases) the radius is decreased. One can see how the **Trust-Region** is changing by doing “`grep -e :DD case.outputM`” where this information is provided. Compared to a line search method this approach does not give such a good improvement per direction, but often will be faster in terms of the net improvement per function evaluation (**run\_lapw**). The port routines go one step further and use one of a number of variants of the trust-region approach, switching between them depending upon what it thinks is most likely to give the best results.

### 3. Implimentation within Wien2k

The optimization routines have been implemented in Wien2k using reverse communication, exploiting the existing driver `drmng` within the port library with a couple of minor modifications primarily to obtain output more consistent with Wien2k conventions. When called, the routine will first generate an initial estimate of the Hessian based upon the multiplicity as well as the bond-order information provided by the user, or if the file `.minrestart` is available it will read in an initial estimate from this file. It will then call `drmng` which decides what to do next. The subroutine `drmng` will then process some information and return to the subroutine `haupt` in `mini` seeking either energy or derivative (force) information. The subroutine `haupt` then looks to see if this information is already available (stored from a previous calculation), and if it is will then call `drmng` again with this information. (Then `drmng` will do some more work, and return for more information) If the information requested by `drmng` it is not available, `haupt` and `mini` create a new `case.struct` file and returns control back to the `min_lapw` script to process a new calculation with different atomic positions. The subroutine `haupt` will also print out some information, store the current value of the Cholesky factor of the Hessian approximation and process termination flags if appropriate.

Rather than using the full Hessian itself, the routine works with the Cholesky factorization. This automatically helps ensure that the Hessian remains positive definite, and is simpler in terms of calculating the inverse to obtain the direction along which to move the atoms. At each cycle the Cholesky factorization is written to a file `.min_hess` which can be inspected. As a caveat, the BFGS algorithm calculates an approximation to the Hessian, not the true Hessian so you cannot use it to, for instance, estimate phonon spectra.

### 4. Some suggestions about how to optimize a structure within your lifetime

a) Start with a calculation that is fast, but not necessarily that accurate. You can save a lot of CPU time by using a minimal `RKMAX` and a smaller number of k-points. How small depends a lot on your problem, often an `RKMAX` of 5 is good enough and for larger calculations (e.g. surfaces) only 5-10 unique k-points can be OK. Some approximate numbers for an approximate calculation are:

Elements	H	sp	d	f
<code>RKMAX=</code>	2.5-3.5	4.5-5	5.5-6	6.5-7

b) Don't overdo the force convergence initially, perhaps only use `-fc 5.0` if you are far from a minimum.

c) As you move closer to the minimum, increase your tolerance of the forces and also improve your calculation parameters. When you increase `RKMAX` or other parameters, copy `.min_hess` to `.minrestart` then delete the old `case.tmpM` and `case.finM` files since

there is normally a change in the absolute energy. The Hessian estimate that you have previously calculated is probably better than the default.

d) The closer you want to converge the structure, the more care will be required in terms of the **RKMAX** that you are using, the number of **k-points**, as well as ensuring that you do not have leakage of core electrons out of the atomic spheres.

e) Be careful about using larger Gaussian or Temperature smearing. While these can improve the convergence within a single scf iteration, in some cases they might produce small inconsistencies between the energies and forces (at the 1-2 mRyd/bohr level).

f) **min\_lapw** will copy **.min\_hess** to **.minrestart** before it starts. In case you have troubles with the minimization and suspect that the previous approximate Hessian could be the reason, you may want to delete these two files (usually they will speed up similar minimization runs).

## 5. Minimizing atomic positions and lattice constants/angles simultaneously

If one wants to know the theoretically predicted structure of a material within a given space group, e.g. TiO<sub>2</sub> in space group 136 (P4<sub>2</sub>/mnm), then all free parameters have to be varied. For the TiO<sub>2</sub> example, they are the a and c lattice constants, and the x-value for the O-position. This means that in principle you should find the absolute minimum of the total energy **:ENE** as a function of 3 variables (a, c, O-x).

Unfortunately, WIEN2k does not have a minimizer that can find this directly. The **min\_lapw** script (**mini** program) is an efficient way of finding the energy minimum as a function of the internal degrees of freedom (O-x for TiO<sub>2</sub>). The lattice constants (and angles, if relevant) still have to be scanned in a systematic way, a procedure that takes many steps. For each set of lattice constants and angles, **min\_lapw** has to be executed to find the optimized internal coordinates.

Preparing a series of case.struct files with the lattice constants/angles that one wants to scan, can often be done by the **optimize** program. Running all calculations automatically can be done by making suitable changes to the **optimize.job** script that it produces.

Below are two examples of **optimize.job** scripts, which have been created by “x optimize”, but need to be modified slightly by the user (you may use additional parameters like **-p**, **-in1orig**, **-in1new 5**, different convergence criteria, ... or **runsp\_lapw**).

1) Calculate the total energy for 5 different volumes of a structure with fixed A:B:C ratio, and optimize the internal coordinates for each volume. The 5 different lattice constants are put in 5 previously prepared structure files, of which the headers have to be listed in the **optimize.job** script :

```

#!/bin/csh -f
#
# Loop over different volumes calculated by optimize
foreach i ( \
            case_vol___4.0 \
            case_vol___2.0 \
            case_vol___0.0 \
            case_vol___-2.0 \
            case_vol___-4.0 \
        )
    cp $i.struct case.struct
# Perform an optimization of the internal parameters
min_lapw -I -j "run_lapw -I -fc 1. -renorm -i 40"
set stat = $status
if ($stat) then
    echo "ERROR status in" $i
    exit 1
endif
save_lapw $i
end

```

After successfully running this script, you have to inspect the total energies of all 5 saved \*.scf files, and determine from this information the optimal volume (**eplot\_lapw** or **w2web**). Together with the optimized internal positions for that volume, you have a theoretical prediction for the structure of this material.

2) The same as 1), but now scan 5 different volumes, and for each volume calculate a series of c/a ratios with that volume fixed. Perform the following steps:

```

cp case.struct case_original.struct
x optimize          (generate structures with 5 different c/a ratios; -4,-2,0,2,4)
cp optimize.job optimize_vol0.job
edit optimize_vol0.job      (see below, specify VOL_0.0 as directory for
save_lapw)
optimize_vol0.job          (this will take some time, it produces the first c/a
                             optimization for volume=V0)

```

Now start a loop over the 5 c/a ratios:

```

cp VOL_0.0/case_coa___-4.0.struct case_initial.struct
# (optimize uses case_initial.struct if present)
x optimize          (generate structure files for 4 other volumes with c/a ratio fixed to -4)
cp optimize.job optimize_master.job (do this only the first time!)
edit optimize_master.job (see below, specify COA_-4.0 as dir for save_lapw)
optimize.job        (this will take some time, it produces a volume optimization for a
                     given c/a ratio)

```

go back and loop over all c/a ratios (change the save\_lapw line accordingly).

This is an example of optimize\_vol0.job :

```
#!/bin/csh -f
#
# Loop over different coa's at constant volume=V0
foreach i ( \
            case_coa___4.0 \
            case_coa___2.0 \
            case_coa___0.0 \
            case_coa__ -2.0 \
            case_coa__ -4.0 \
        )
    cp $i.struct case.struct
    min_lapw -I -j "run_lapw -I -fc 1. -renorm -i 40"
    set stat = $status
    if ($stat) then
        echo "ERROR status in" $i
        exit 1
    endif
    save_lapw -d VOL_0.0 $i
end
```

This is an example of **optimize\_master.job**

```
#!/bin/csh -f
#
# Loop over different volumes at constant c/a
foreach i ( \
            case_vol___4.0 \
            case_vol___2.0 \
            case_vol__ -2.0 \
            case_vol__ -4.0 \
        )
    cp $i.struct case.struct
    min_lapw -I -j "run_lapw -I -fc 1. -renorm -i 40"
    set stat = $status
    if ($stat) then
        echo "ERROR status in" $i
        exit 1
    endif
    save_lapw -d COA_-4.0 $i
end
```

At the end you should have directory VOL\_0.0 (with a set of c/a ratios, each of them with optimized internal positions) and five COA\_x.0 directories (each with a set of 4 volumes;

V=0 is in VOL\_0.0, and you probably want to copy (move) them into the appropriate COA directories).

PS: The reason for this slightly complicated setup is saving of time. For a given c/a ratio one performs first an optimization of the internal parameters. The subsequent volume variation uses these optimized struct files because the internal coordinates usually change much less with volume than with c/a (less min steps necessary).

## 6. What can go wrong (and perhaps what to do about it)

The code will *always* find better values of the energy, but how well it will do this depends on how well the user has constructed the base lapw calculation and how accurately the forces are calculated. The most common problem is that the calculation has not adequately converged, in which case the energies and forces can be inconsistent and the optimization will stagnate (mostly for spin-polarized cases). At the current moment the standard method is to just check for consistency of the forces, but sometimes this is not enough and either (or both) the core charges and the plane wave components are not fully converged. A simple fix is to converge the scf-cycle with multiple convergence criteria (requires WIEN2k\_04/11 or later) like

```
min -I -j "runsp_lapw -I -i 80 -cc 0.001 -ec 0.0001 -fc 1.0"
```

Sometimes the optimization will take several good steps, then there may be a warning about the curvature condition and after this for several steps the convergence may be very slow. What is happening is that the Hessian has changed rather a lot, and it is taking the BFGS several cycles to update the Hessian to a more reasonable value. It may be better to stop the optimization and do a clean restart (remove **.minrestart**, **.min\_hess** and **case.tmpM** files), alternatively just be patient.

One issue that can require care is when you see a warning about overlapping spheres. It may be that the algorithm has just been too ambitious and taken too large a step which would bring atoms too close together (it is not that smart). However, it may be that you have RMT's which are too large. In general you want to have the RMT's about 5-10% smaller than you expect the optimum final values to be to allow more freedom for movement of the atoms. If you see the warning only once that is not a problem; if you see it repeatedly you should probably reduce your RMT's using the **clminter** code.

It is worth emphasizing that the symmetry of your structure is something which is *a-priori* information for the calculations. If you choose too low a symmetry the calculation may give you in the end a lower-symmetry structure; if you pick too high the opposite. In certain cases you may want at the end to deliberately reduce the symmetry and rerun a calculation. However, be careful since numerical errors may well give you instabilities and hence prefer an unphysically lower symmetry structure.

## 7. What would be nice to add (wish list)

- 1) The better the initial approximation to the Hessian, the faster the convergence will be. At the moment an “adequate” initial approximation for the diagonal elements is used. There are many ways one can envisage creating a better initial approximation, ranging from a simple spring-model to something more complicated such as pair-wise interactions. These do not have to be exceptionally accurate, but would almost certainly be an improvement.
- 2) The port routine (and many existing optimization routines) work on the assumption that calculating the gradient (forces in Wien2k) is CPU expensive, whereas calculating the function value (energy) is not. As a consequence they tend to only use function values in many cases, for instance when estimating whether a step is acceptable. In fact, calculating the gradient is almost free in Wien2k. While it would not be trivial, elements of the decision making routines in port could possibly be improved to exploit the free gradient information. In addition, it might be useful to exploit the gradient information available from bad steps (energy increasing) to update the Hessian matrix, and thereby improve the overall convergence speed.
- 3) No attempt has been made to tweak some of the internal “tuning” parameters within port (buried in the initialization routines). Someone who has plenty of CPU time available might want to play with these, perhaps find a more appropriate combination for DFT problems.