DFT Mixing for Dummies

Laurence Marks, October 25, 2022

The purpose of these notes is to provide a basic introduction to what is called "Mixing" in the DFT literature, somewhat general although it has some description of parts which are currently in the Wien2k code. The intent is not to provide a fully rigorous description, rather something which is hopefully a bit readable, albeit without all of the mathematics. Alas, you can't escape math, sorry!

These notes go through the basics in terms of how to think about the density as well as other variables, and what the effect of running through an iteration cycle. How the steps are combined to produce the next step, as well as important issues such as scaling, predicting the Greed and also trust regions are sketched. The inclusion of atom displacements is also described in terms of how this is done.

The last sections are intended to give some insight into how the process of mixing works, drawing analogies to water running downhill, from canyons to lazy rivers. The final part deal with, very briefly, some of the scientific myths of mixing as well as some technical details for coders of mixers, as well as Wien2k. The final section has specifics for the Wien2k code.

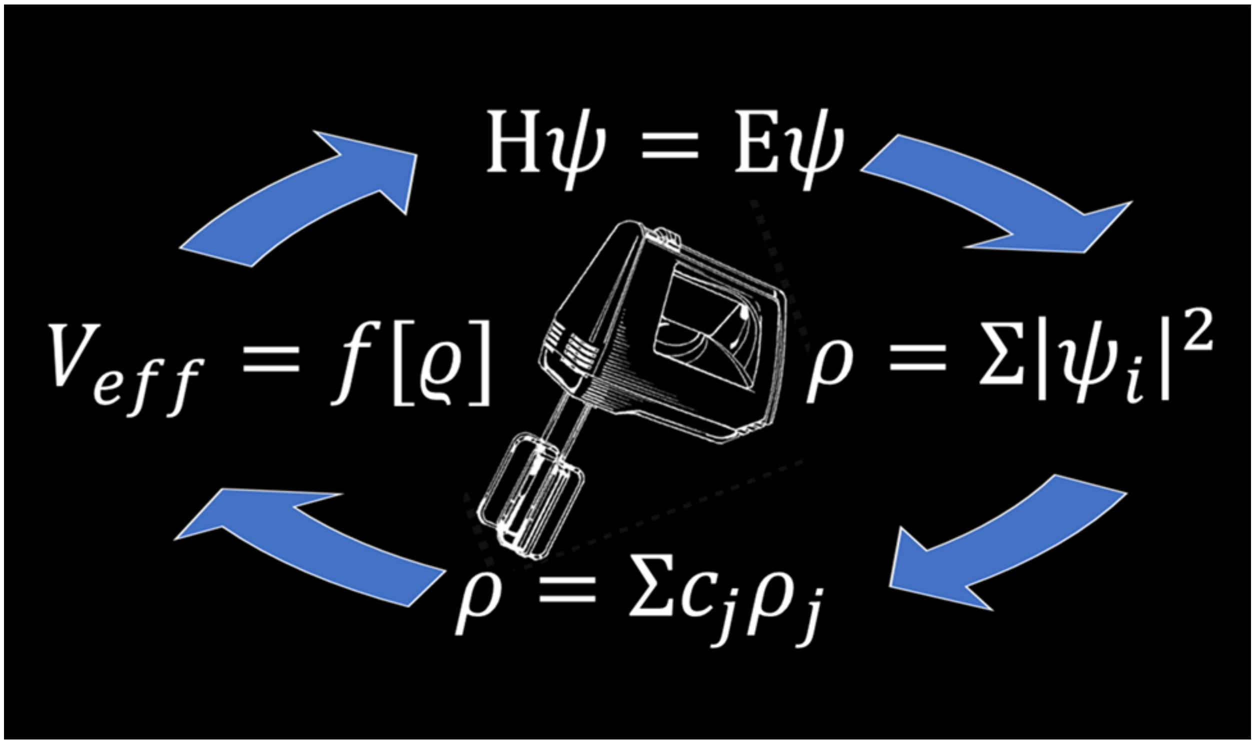
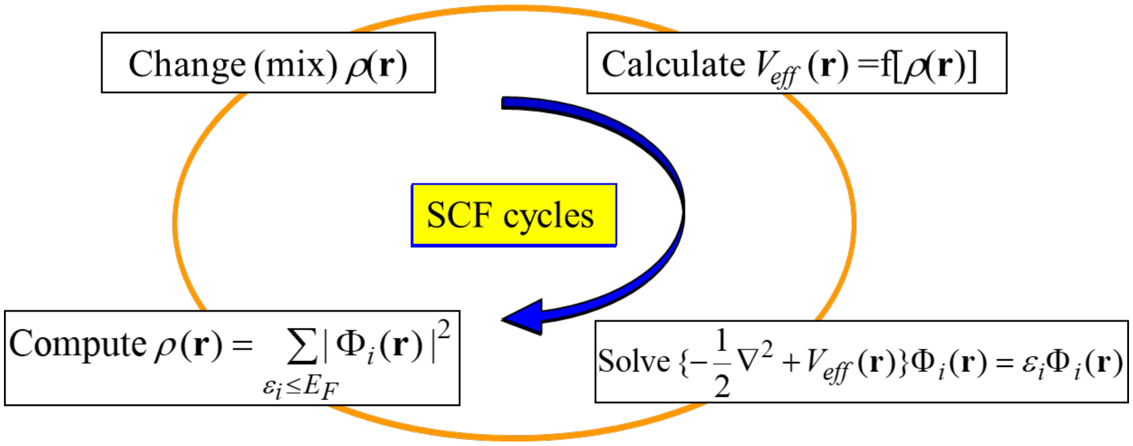


Table of Contents

1. What is Mixing?

The majority of density functional (DFT) codes use a self-consistent approach to determining the electron density, as well as any other terms involved such as the kinetic-energy density, orbital potentials or exact exchange components. The basic idea is illustrated below, and involves a cycle where, for a given density ρ(r), in four steps one:

1) Calculates the effective potential V_{eff} (and other terms if needed)
2) Solves for the orbitals using this potential
3) Compute what the density and other relevant terms would be with these orbitals by filling them up to the Fermi energy
4) Change (mix) the densities and others to produce a new value, then go back to 1) and repeat until the process converges.


Change (mix) $\rho(\mathbf{r})$
Calculate $V_{eff}(\mathbf{r}) = f[\rho(\mathbf{r})]$
SCF cycles
Compute $\rho(\mathbf{r}) = \sum_{\varepsilon_i \le E_F} |\Phi_i(\mathbf{r})|^2$
Solve $\{-\frac{1}{2}\nabla^2 + V_{eff}(\mathbf{r})\}\Phi_i(\mathbf{r}) = \varepsilon_i \Phi_i(\mathbf{r})$

The density is "self-consistent" when the density one uses to calculate the potential is identical to the density that comes out from step 3). The process of changing the densities is called mixing in the DFT literature, probably because most methods involve combining or "mixing" the densities from previous iterations in some sense. In the older mathematics literature this would be called solving for a set of non-linear equations; in other areas of mathematics it is called solving a fixed-point problem.

The term "fixed-point" here deserves a little expansion, because it is a useful way of thinking about these problems. Rather than as some solid matter, we think about how the density changes as a dynamic variable – a moving object such as a piece of chocolate cake moving with time under the influence of Schrödinger's equation (OK, Dirac's chocolate equation if you feel picky). A fixed-point would be when the cake stays in the same position; a bad problem would be when the cake gets eaten.

To handle the various terms that will be encountered in these notes it is useful to introduce a few different terms – more will come later. The first is to give a number for the iteration, i.e. to refer to densities that we use in Step 1) as ρ_k for the "k" iteration, dropping the "(r)" for brevity. The second is to call the densities that come out from 3) as $F(\rho_k)$, where the "F" would be called a mapping that converts from the old densities to new values.

For illustrative purposes only, the simplest approach is to use what is called the Pratt Method. In such a case the next density is given by

$$\rho_{k+1} = (1-\alpha)\rho_k + \alpha F(\rho_k) \tag{1}$$

The parameter "α" is often called the "mixing parameter", but a better approach is to call it the "mixing Greed". This is a little subtle, but is quite fundamental to understanding how the whole process of mixing works. The term "Greed algorithm" has an accepted definition[1]:

"A Greedy algorithm always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution."

As an illustration, suppose you want a total of 41 cents from a combination of 25, 10 and 4 cent coins. The Greedy approach is to reduce this as much as possible with 25 cents, so we take one leaving 16 cents more needed. Next, we reduce this as much as possible with a 10 cent coin – leaving 6. Then we can only use a single 4 cent; oops, we can only make 39 cents!

In this case the Greedy algorithm failed – the right choice is of course a 25 cent and four 4 cent coins, no 10 cents. Using the 10 cent coin was too Greedy. There are cases where being as unGreedy as possible is bad. For instance, suppose that in the previous case we add 0.001 cent coins. The least Greedy approach is to use just these 0.001 cent coins, 45,000 of them, which would be far too many to carry.

For DFT problems a large mixing Greed (e.g. a value of 1) indicates that we think that the density that comes out from Step 3), the mapping $F(\rho_k)$, is a better approximation than what we started with. While there are some materials such as bulk MgO where this may be true, in general it is not. Slightly more complicated, it is not guaranteed that using the Pratt step in fact improves the density, and it may not be the best direction to change the density. While if one is very close to

[1] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C., *Introduction to Algorithms*. MIT Press: Boston, US, 2009; p 1291

a converged density it may be, if you are a long way away then nothing is certain. On the other side, using a mixing Greed of 0.001 would, at least for the Pratt method, take a vast number of iterations to converge (if it does) so is also bad.

The question this raises is how Greedy should one be, and whether one should use the Pratt step or something else. This is at the heart of mixing; good algorithms do this well, and are simple forms of artificial intelligence codes.

2. What Steps can one use?

The next question one has to think about is whether the only choice is the mapping $F(\rho_k)$ and the current step, or whether one can use more than this. The answer is that one has, at least in principle, all the prior densities and mappings. The very old ones are generally from a density which is quite different from the current one, so are less reliable for the purposes of guiding what the next density should be. Hence it is standard practice to limit those used to some reasonable number such as the last 8-16. Like many things in mixing and for algorithms in general, there is often no hard proof of what one should use, just experience based upon tests.

However, we should remember that no algorithm is perfect, so "bad steps" can always occur. A slightly bad step is no problem, and often the algorithms do very well despite these. Unfortunately, a really bad step can lead to misleading information. One can think about these as equivalent to falling into a pit: the landscape at the bottom of the pit does not provide much useful information, particularly if there are stakes present! One has to include ways to throw away very bad steps, and also to try and avoid taking them – which often happens if the algorithm is being too Greedy. If the step was awful, it can still be used to estimate a reduction (for that given step) that can be used in the next cycle – and throw away the data from the bottom of the pit. If the step was bad but not awful (a shallow pit, only half a meter) it makes sense to keep it, but go back to where we were before and recalculate to avoid the depression.

3. How to combine the steps?

There are two ways to consider combining the steps, one via a weighted least-squares approach, the other via a Taylor series. For very simple problems they are identical, and this is also true for many more complex and difficult problems. Where they differ is in how readily they yield understanding of their limitations, and what can (and does) go wrong. For the later the Taylor series approach is much better, so it is the one used here.

First, a few more math terms and equations, so we can express things better later. First, we will define the residue

$$R_k = F(\rho_k) - \rho_k \tag{2}$$

Our aim is to have the residue be zero, and its value both as a vector and absolute is a measure of how bad the density at iteration k is. Looking back to the Pratt step, that can also be written as

$$\rho_{k+1} = \rho_k + \alpha R_k \tag{3}$$

In a sense the residue is like a force term (the negative of a gradient), so it is somewhat like what is called the “steepest descent” method in the literature on minimizing values such as the energy. The steepest decent method only walks straight downhill. As you might guess, the steepest descent method is about the worst approach possible.

Continuing to define values, we next observe that all that really matters is the change in both the residue and the density as the algorithm progresses; absolute values are less important. We therefore introduce two vectors for these which are relative to the current values (k):

$$s_i = \rho_i - \rho_k \ ; \ y_i = R_i - R_k \tag{4}$$

(As a minor technical aside, in the mathematics literature slightly different forms are used which are not so useful for DFT problems.)

What we now do is expand the residue as a Taylor series, that is:

$$R = R_k + B_k \, s_{k+1} + s_{k+1} \, C_k \, s_{k+1} + \cdots \tag{5}$$

Here B_k is called a Jacobian, really a matrix, which relates how the residue will change with position, C_k is the next higher-order term and in principle this series goes on forever. What we do is cross our fingers and ignore C_k and other similar non-linear terms, which is fine so long as our next change s_{k+1} is not too large. (Of course, “too large” is not a well-defined term, and can vary with the problem.) We should not forget this – it comes up again later.

We now target having a zero residue, which requires that we solve for our next step p_{k+1} using the inverse Jacobian H_k (inverse of B_k) in the form:

$$p_{k+1} = H_k \, R_k \tag{6}$$

This is the “Predicted Step”, that is what our value of H_k indicates we should use. It is not everything, there are parts of the residue which when multiplying by H_k give nothing. We will come back to this “Unpredicted Step” later, but first we need to better define H_k.

4. How to construct the Jacobian and its inverse?

The next step is a bit subtle, and is one place where the concept of Greed comes in again. What we do is use the prior values of s_k and y_k for this. The matrix H_k has as many rows and columns as there are variables in the DFT density, and we only have at most 8-16 values of past steps, so we cannot fully define it, we have to use some reasonable approximate method. In addition, the full matrix would almost certainly require far too much computer memory, so we need some smaller form so as not to destroy our computer.

First, we say that our target matrix has to explain our previous steps, which is equivalent to what is called the Secant equation, specifically for the estimate we have in the k'th cycle H_k:

$$s_k = H_k\, y_k \tag{7}$$

We do not want to do this for just a single value, instead for all the relevant prior histories. Defining new matrices to hold these of $S_k = (s_k, s_{k-1}, \dots)$ and $Y_k = (y, y_{k-1}, \dots)$, we can generalize to use

$$S_k = H_k\, Y_k \tag{8}$$

Just as equation (7) is the secant equation, (8) is a multisecant equation and methods using this form are called multisecant methods.

Next, one approach is to use the smallest possible value of H_k that satisfies the multisecant equation. The matrix that does this is a variant of a famous type of inverse, the Penrose-Moore. The form of this is given by

$$H_k = S_k\,(Y_k^T Y_k + \delta I)^{-1}\, Y_k^T \tag{9}$$

with the symbol "T" being used to represent a transpose of the rows and columns. The form here includes a regularization term δI which is needed as the matrix $Y_k^T Y_k$ might have some zeros so could blow up when we take the inverse. The regularization is equivalent to a Wiener filter, and also acts to reduce the effects of noise in the numerical calculations which can always happen – computers are not infallible, neither are coders.

This is the smallest value of H_k, so makes the least assumptions and as such is the least Greedy. It will produce the smallest possible step for the current residue, consistent with the previous history.

It is not the only choice. An alternative approach is to work instead with the inverse, i.e. solve for

$$Y_k = B_k\, S_k \tag{10}$$

which in a similar fashion, without the regularization leads to

$$B_k = Y_k\ (S_k^T S_k)^{-1}\ S_k^T\ and\ H_k = S_k\ (S_k^T Y_k)^{-1}\ S_k^T \tag{11}$$

It is a little more complicated to regularize the inverse for H_k, but it can be done using a singular-values decomposition, which I will not discuss here. (In practice this has to be done, otherwise you may get chocolate on your face.) This produces the largest step consistent with the previous history, and as such is the Greediest. As a consequence, it is harder to use and has failed for many in DFT codes, although this is really not a fundamental problem and may be because when people tried they did not safeguard it. (This algorithm is in Wien2k – it works, but is not the best.)

While the above are two limits, they are not the only ones. In fact, any matrix T_k will satisfy the multisecants equation if one has

$$H_k = S_k\ (T_k^T Y_k)^{-1}\ T_k^T \text{ and } B_k = Y_k\ (T_k^T S_k)^{-1}\ T_k^T \tag{12}$$

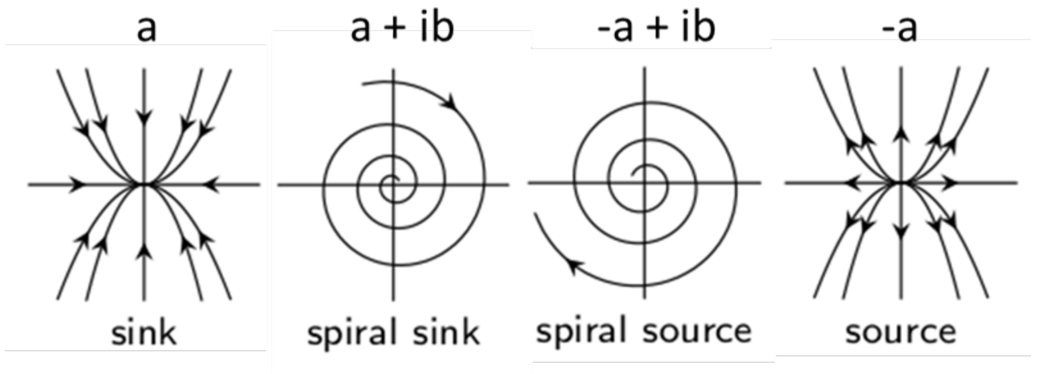
A reasonable way is to use some linear combination, i.e.

$$T_k = (1-\lambda)S_k + \lambda Y_k \tag{13}$$

The least Greedy method is with $\lambda = 1$, the most Greed is $\lambda = 0$, and this combination allows one to move between the two. This method is called MSR1 in the Wien2k code, where for coding and historical reasons $T_k = \alpha S_k + Y_k$ is used to make the scaling easier.

The least Greedy approach is called MSEC in the Wien2k code, and DIIS or Pulay mixing in many other codes (although see later, as there is a scaling issue as well). The Greediest is called MSGB in the Wien2k code, which stands for "Multisecant Good Broyden". This is an interesting example of divergence of communities. In his original work Broyden could not get the least Greedy approach to work, so it was called "Bad Broyden" whereas the one he could became called "Good Broyden". (For rigor, he did not use a multisecants approach.) The majority of the mathematics literature has focused on Good Broyden methods. For DFT problems the Bad Broyden approach dominates the literature. One exception is in Wien2k where the hybrid method MSR1 is used, and it in fact out performs both the alternatives.

From the coin problem discussed before, sometimes Greed is good but also being not Greedy enough can be bad. To date nobody has been able to fully explain under what conditions the two cases of good and bad Broyden are appropriate. At least partially one can understand why MSR1 can be better if you think about the set of residues which, when multiplied by H_k (i.e. the product $H_k R_k$) give non-zero values. With the bad-Broyden method it is only residues which are a linear combination of the prior residues Y_k; with good-Broyden it is the residues which are linear combinations of the prior step S_k. In contrast, the hybrid method is the sum of these two. I will note that this is only a reasonable explanation. In order to choose the value of λ, the largest value such that T_k does not have negative eigenvalues is used, searching up from the value $\lambda = 0$. This is a fundamental feature of the stability of linear systems, albeit there is a change in sign from what is commonly used. The type of behavior one has for each eigenvalue is, as illustrated below in terms of the real and imaginary parts of the eigenvalue:

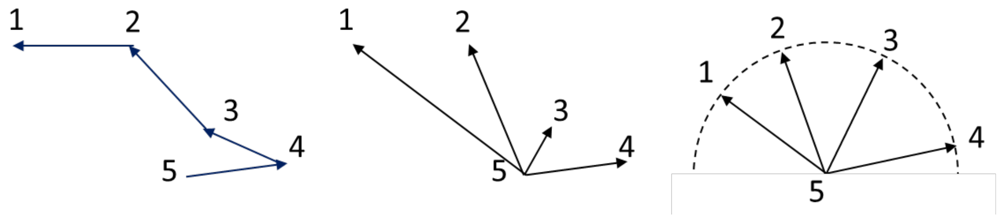


1) If the eigenvalue is positive with no imaginary part, it is a good, converging value or **sink**.
2) If the eigenvalue is positive with an imaginary part, it will spiral in to the solution, a **spiral sink**; the large the imaginary part the more it will spiral
3) If the eigenvalue is negative with an imaginary part, it will spiral away from the solution, a **spiral source**.
4) If the eigenvalue is negative, it moves away – it is divergent or just a **source**.

We know that at the solution the eigenvalues of the Jacobian and its inverse must be positive, as the fixed-point solution is also a minimum of the energy. The choice of λ is equivalent to choosing the Greediest approach that will also have the right type of properties.

5. To Scale or Not to Scale, that is the question

In the last section I talked about generating H_k (and/or B_k) from the prior steps, but there is one additional refinement to ponder – how do we think about the prior steps. In the original approaches where this type of method was used mainly for optimization, the approach used was equivalent to treating them as directions, that is along the direction given by s_k the residue changed by y_k – the left below with "5" the current position and 1-4 older ones. If we just treat them this way then it does not matter how large they are for different steps.



However, when we combine them to give us S_k and Y_k it does matter. The reason is that the larger ones will tend to dominate, for instance with raw values as shown above in the middle. If we scale them all to equal size then we are treating them more as directions as on the right.

Technically this is equivalent to thinking about the set of histories as defining a type of numerical derivative called a Simplex Gradient. In tests it is better to scale them such that the change in residue is the same. In tests the best scaling is to scale the diagonal of $T_k = (1-\lambda)S_k + \lambda Y_k$ to unity, which requires coupling the positivity constraint above with the scaling.

6. Starting to pull everything together: Predicted and Unpredicted

We now have most of the pieces, it is time to start to pull them together. Back towards the start I mentioned that our previous steps give us information from which we can predict part of the next step, to recap this will be (ignoring regularization):

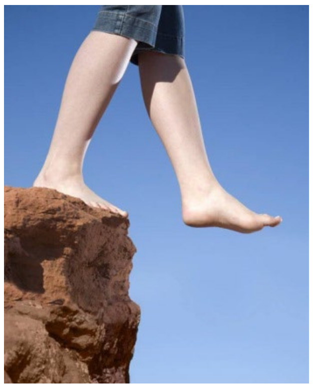
$$p_k = H_k R_k = S_k (T_k^T Y_k)^{-1} T_k^T R_k \tag{14}$$

This only includes part of the residual – there will still be some left, which is called the Unpredicted step and is given by

$$u_k = R_k - Y_k (T_k^T Y_k)^{-1} T_k^T R_k \tag{15}$$

One has the Y_k on the left-hand side since this has to remove the prior residuals from our current residue, since we have already accounted for it. Our next density will therefore be

$$\rho_{k+1} = \rho_k + \alpha_k u_k + \beta_k p_k \tag{16}$$



The mixing Greed (aka mixing factor) α_k has at long last reappeared, similar to what we had before but now it only applies to the that part of the residual about which we know nothing – the unpredicted part. This Unpredicted Greed now determines how aggressively we are going to step into uncharted territory – hopefully not off a cliff.

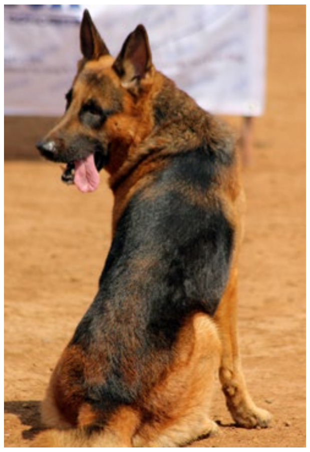
Another term has also appeared, β_k, which multiplies the predicted part – the Predicted Greed. You might think that this is wrong, since we know this part we should just use it but that is being too Greedy – remember that we ignored the higher-order terms in the Taylor series. If our step is small enough we don't have to worry about these non-linear terms. However, in general they can be there and we need to somehow reduce how much of the predicted we use.

How much of the unpredicted and predicted – that's the next topic.

7. How Much Predicted and Unpredicted Step?

The next part is deceptively simple, but fortunately for us the main ideas have been known in a different field, optimization, so we can use them. We start with some guess at both of the two scalings α_1 and β_1 for our first step. Typically the mixing Greed should be small, and the predictive Greed taken as unity. If this gives us a good reduction in the residue then we tentatively increase the mixing Greed; if it is bad we decrease it a little, perhaps more we would reduce the predictive

Greed. This type of approach was used in earlier versions of the Wien2k mixer and has been incorporated into a few other DFT codes that follow the approach in these notes.



There is another method which is better in practice; after all, just increasing or decreasing does not exploit all the information that we have available. The idea is to look at what we did in the last cycle, and then use this to estimate what the best values of the two Greed parameters should have been – we look backwards.

Being specific, for the Unpredicted Greed from our last iteration u_k, we work out the best step that we should have taken. This is equivalent to finding an estimate α^{Est} that minimizes the last unpredicted step component for our *current* Jacobian B_k

$$|u_{k-1} - \alpha^{Est} B_k u_{k-1}|^2 \qquad (17)$$

Which we can solve as

$$\alpha^{Est} = |u_{k-1}|^2 / |u_{k-1} B_k u_{k-1}| \qquad (18)$$

In a similar way we look at what would have been the best Predicted Greed to use, which means that we solve for a scaling of the Predicted step of the last iteration p_{k-1} against the part of the residue that it corresponded to:

$$|(R_{k-1} - u_{k-1}) - \beta^{Est} B_k p_{k-1}|^2 \qquad (19)$$

Which gives us

$$\beta^{Est} = |R_{k-1} - u_{k-1}|^2 / |(R_{k-1} - u_{k-1}) B_k p_{k-1}| + \delta \qquad (20)$$

where $\delta \sim 0.1$ helps convergence. To avoid massive changes, these estimates are combined with what were present in the previous cycle; in practice a simple exponential moving average (EMA) works, i.e. using

$$\beta_k = (\beta^{Est} + \beta_{k-1})/2 \text{ and } \alpha_k = (\alpha^{Est} + \alpha_{k-1})/2 \qquad (21)$$

plus some reasonable bounds to ensure that massive changes don't occur, for instance increase by more than 5/3 or decrease by 3/5, or the values get too large or small.

We can combine now these values, again with the last step to estimate what probably should have been the step that was used in the last cycle, that is

$$Step^{Est} = \alpha_k u_{k-1} + \beta_k u_{k-1} \qquad (22)$$

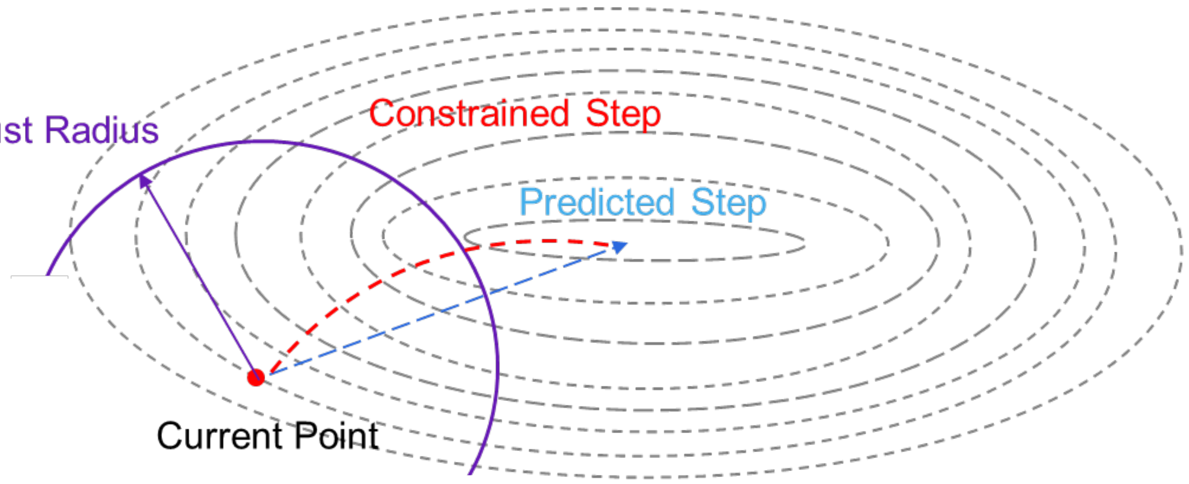
We can now use this to estimate how large a total step we should allow, and also take parts of this estimate if we want to, for instance, limit how large a change in the atomic positions we allow. For these we need a Trust region.

8. Controlling the Step: Trust Radii

Even with the best of choices for H_k and B_k, we are still neglecting the higher-order terms in the Taylor expansion. If we go to equation (4), we will be fine so long as the step we take is small enough, more strictly if

$$B_k \gg C_k s_{k+1} \qquad (23)$$

Unfortunately the higher-order C_k will change both with the type of problem one has as well as whether one is close to or far from the solution. What one therefore has to do is limit the step such that it is small enough that equation (23) is obeyed, even when we do not know how larger C_k is! The way we do this is by setting up a constrained problem where we limit the size of the step, and then find the best value to use as illustrated below with dashed contours for the residue indicated. Shown in the Figure below is the **Trust Radius**, which is the maximum that we are going to allow. The **Predicted Step** is larger than this, so we reduce it following the **Constrained Step** path, which gives us the best reduction for a given magnitude.



To achieve this, we minimize the Lagrangian problem to find the step *f* with a Lagrange multiplier ξ and a trust radius $\mathfrak{R}$

$$\mathcal{L} = |R_k - B_k f|^2 - \xi\{|f|^2 - \mathfrak{R}^2\} \qquad (24)$$

where we limit the step to being a combination of the prior steps. There are many ways to do this; in practice the code starts from the full step, then increases the Lagrange multiplier until the constraint are satisfied.

Similar to how the two Greeds were estimated, we use the estimate from above average with the latest value for the trust radius $\mathfrak{R}$. Exactly the same procedure is used to control the movement of atoms and any other variable if needed. (The less constraints the better, and they are rarely needed for well-constructed problems.)

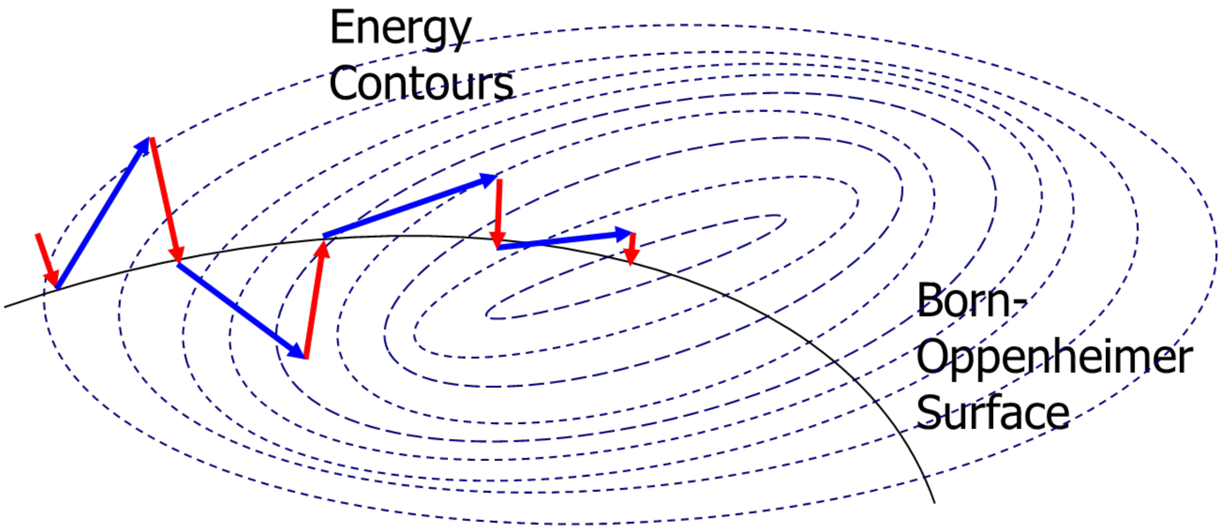
9. Optimizing Positions: First the Old Way

Everything up to now is what is in the Wien2k code, and is similar to what is in other codes (although scaling, control of the two Greeds and trust regions are not used in many to most other codes). One subtle difference is that in Wien2k one does not just mix the densities (and other variables), in fact one can mix atomic positions. This is now optimization of the atomic positions in the unit cell.

The old method, which is common in many codes and is available using the "PORT" option, is to use a double loop:

1) Iterate the density until it is self-consistent, keeping the atomic positions fixed
2) Change the atomic positions, then go back to 1) and keep going until the forces are low enough.

This approach is illustrated below. At any given point we first converge the density along the **red** arrows, moving onto the Born-Oppenheimer Surface which is defined as the surface where the density is converged. Then a **blue** step is taken to improve the atomic positions, although it makes the density unconverged. Then one iterates, **blue** from some other program and **red** in the mixer.



The way that steps are taken is not that different from how the Jacobian was constructed earlier. What is used are the gradients (forces) on the atoms, and the second derivative matrix is generated in a similar way from these. The most common and powerful method by a long way is the Broyden-Fletcher-Goldfarb-Shanno or BFGS method. One important difference is the second derivative approximation has a number of special properties (positive definite) which can be exploited with some trapping; more details can be found in numerous textbooks.

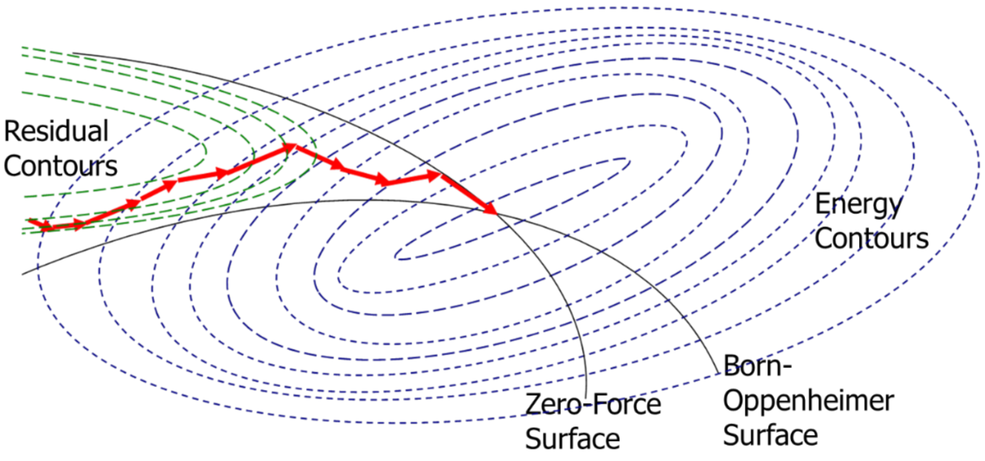
One inefficiency about this approach is that the Jacobian is created new for each different density convergence, and we are throwing away information. Can we retain this information and do better?

10. Optimizing Positions: The New Way in MSR1a

The key to the approach used in Wien2k is to go all the way back to how the residue was defined in equation (2), and now redefine it to include the gradients (negative of the forces) G:

$$R_k = (F(\rho_k) - \rho_k, -G) \tag{25}$$

The simple Pulay step of equation (3) now has a steepest decent term for the atomic positions. We now just use this in *everything,* that is all the earlier equations. The algorithm is illustrated below.



The iterations now proceed down some contours of the residue that was extended in equation (25). In addition to the Born-Oppenheimer surface from before, there is another called the "Zero-Force Surface" which is where the forces are apparently zero, *but the density is not converged.* In fact, we only had reliable forces when the density is converged, so it is better to call these "pseudo-forces".

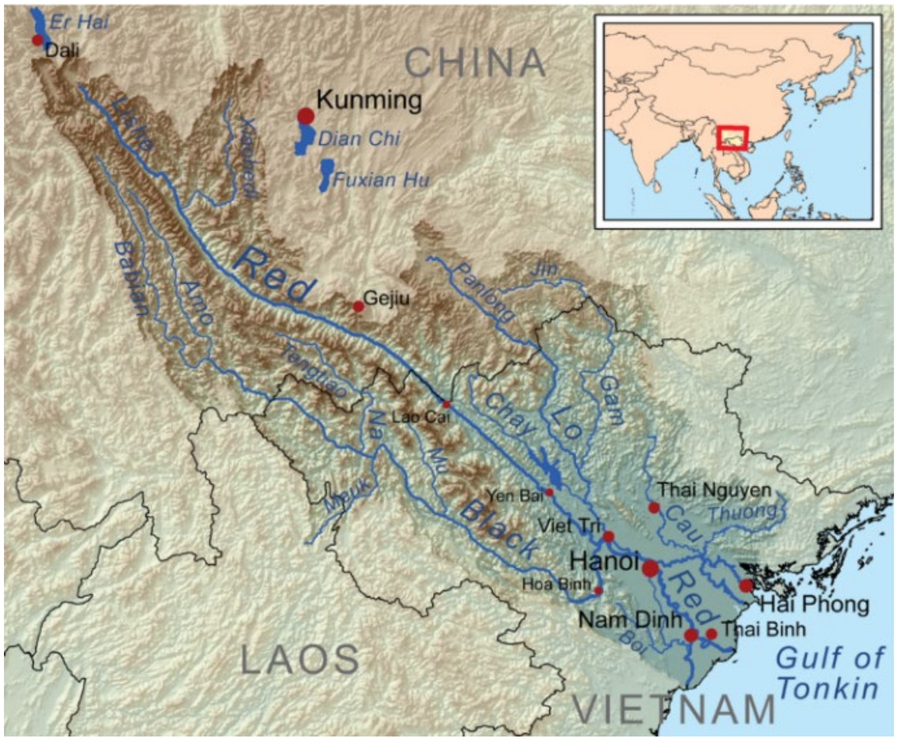
That we do not have true forces does not matter; indeed, the energy that we have when the density is not converged is itself not the real one. All that matters is that at the solution the forces and the density are converged and accurate, and this is a true minimum (of everything).

This works, but one caveat. There is coupling between atomic positions and the electron density, and this can be large in some systems such as metals; in insulators it is typically quite small. As a consequence, this approach where both the atom positions and densities are simultaneously converged is trickier, which is why the various approaches in the previous sections to better control the two Greed parameters as well as the trust region control were developed.

11. Canyons, Lazy Rivers and the Sea

One cannot predict in some cases how the mixing will behave, it heavily depends upon the problem, both what the structure and electronic states are as well as how well it is constructed. Problems which are well-posed often converge better than ones which are badly posed. Silly problems, for instance DFT simulations of cold fusion with a hydrogen atom 0.1 Angstroms from a palladium atom behave…the way they should for something that dumb! While this a rather obvious example, there are many which are not good, which will similarly be very hard to converge – for instance nickel oxide without spin. (It might converge, but ….)

A good way to think about mixing is in terms of water running downhill from the mountains to the sea. Sometimes it is running fast over rapids; sometimes squeezing through tight canyons or ambling rather lazily across a flat valley, the latter being called soft modes. These are when it takes a large change in the density or atomic positions to reduce the residue. One common example is for just the atomic positions, where the hard modes will be optimized first, then the soft modes. It turns out that mixing Greed is needed for the soft modes, more towards the aggressive "Good Broyden"



One example case is the Red River in China and Vietnam, one of the straightest in the world [2]. If we start from Dali (top left), and think about trying to end up in the Gulf of Tonkin then it will not be hard because the downstream direction only varies a little along the length. You can compare this to the Jin/Gam river which winds around a bit, but is not pathological.

[2] By Kmusser - Own work, Elevation data from SRTM, drainage basin from GTOPO [1], all other features from Vector Map., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=12063353

THE GRAND CANYON NATIONAL PARK
ARIZONA
KAIBAB NATIONAL FOREST
KAIBAB PLATEAU
COCONINO PLATEAU
COCONINO TUSAYAN NATIONAL FOREST
NAVAJO INDIAN RESERVATION
PAINTED DESERT
HAVASUPAI INDIAN RESERVATION
MAY 12 1919

Missouri
Illinois
Ozark Plateau
Cape Girardeau
Ohio R.
Mississippi R.
Kentuc
Crowleys Ridge
Mississippi Embayment
Boston Mountains
Tennessee
Arkansas R.
Memphis
Little Rock
Ouachita Mountains
Arkansas
Mississippi River
Mississippi
Jackson
Vicksburg
Red River
Natchez
Alexandria
Louisiana
Baton Rouge
New Orleans
Gulf of Mexico
Elevation
Feet
Meters
(km)
(mi)

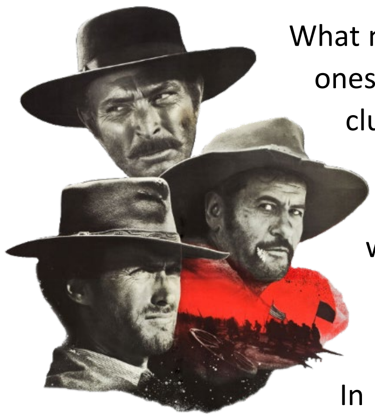
Contrasting is the Grand Canyon in the USA. Where one has very steep sides – the canyon walls which rise about 1,800 meters from the river. Clearly in such a case if too large a step is made one runs into the walls – disaster! Instead the algorithm has to carefully step along, if needed retracing its steps a little when it hits the walls. Since we cannot predict how the river and the canyon walls will bend (the mixer does not have eyes), sometimes it will get things wrong.

A final example is the Mississippi River[3], which meanders very lazily as it approaches the Gulf of Mexico. If one were to follow the river exactly it would be much further than the distance a bird (or electron) flies. A well-constructed algorithm will recognize that the meanders are not important, and skip over them – converging much faster. This is not always the case, and sometimes algorithms will follow the path of the river, eventually reaching the sea…eventually!

[3] By Kbh3rd - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10209178)

12. The Hard, the Soft and the Ugly Modes

Understanding how the algorithms converge has some more complex aspects, in additions to the rivers discussed above. The behavior is ***not*** just dependent upon the number of variables; if it was then DFT problems with thousands to millions of variables would take longer than the life of a graduate student (a viable time unit) to converge. In fact, the convergence depends upon the eigenvalues and eigenvectors of the Jacobian. In a more physical sense we can think of these as modes – comparable, for instance, to soft modes of phonons or coupled electron-phonon waves,



What matters is the number of eigenvalue clusters of the Jacobian, that is ones which are very similar to each other, and also the width of each cluster. As a simple example, consider the case where the contours of equal residue all lie on a sphere. When we move in any direction we change, equally, all the values. If instead we had an ellipse then we will change more along the short direction of the ellipse than the long – it will take longer to converge. Now extend this to multiple dimensions, and allow the ellipse to bend around….

In general, what one has are "hard" eigenvalues and eigenvectors, those where a small change in the density reduces the residue significantly; "soft" ones where a large density change is needed and "ugly" ones which are just noise and do nothing useful.

In general, the algorithms will first converge the hard eigenvalues or eigenvalue clusters, then move down to the soft eigenvalues. Often one of the trickiest parts is converging the soft eigenvalues where there can be very large changes in variables (any and all) for not so much change in the residue.

WYOMING
Green River
Black's Fork
Flaming Gorge Reservoir
Yampa
White
Green
NEVADA
UTAH
Grand Junction
Colorado
Gunnison
Moab
COLORADO
White
Lake Powell
San Juan
St. George
Kanab
Farmington
Virgin
Page
Las Vegas
Henderson
Colorado
Grand Canyon
Tuba City
Chinle
Lake Mead
Little Colorado
Bullhead City
Flagstaff
NEW MEXICO
CALIFORNIA
Prescott
Lake Havasu City
Verde
St. Johns
Salt
Scottsdale
Phoenix
Blythe
Lake Roosevelt
Gila
Mesa
Chandler
ARIZONA
Yuma
San Luis Rio Colorado
Tucson
BAJA CALIFORNIA
SONORA
About 900 miles (1,448 km) across

Very often how the algorithm is behaving can drastically change. One of the most common is when the density far from the solution is metallic, but as the solution is approached it becomes that of an insulator with a good gap. Similar things can happen many times. For instance, if we track the Colorado river from its source in the Rocky Mountains, it starts rushing downhill, then lazily goes through Lake Powell, avoids the walls in the Grand Canyon, again lazily traverses Lake Mead then finally ambles down to end up in the Gulf of California – multiple changes!

13. How can I speed up the convergence?

Improve the model you are using.

Often not having enough k-points, a bad choice of spin state and/or functional for the particular problem and a bad choice of technical parameters can make the mixing behave badly. In an ideal world the mixer will turn around, nudge (hit or slap?) the user and tell her that she is using bad parameters. In Wien2k sometimes the muffin-tin radii are too small, the potential has not been expanded far enough when it is calculated or the eigenvalues are calculated (lapw0 & lapw1). Too often the structure is just silly. For instance, a calculation of the ionic compounds $Mg^{6+}O_3^{2-}$ is not going to behave well, the ion Mg^{6+} is nonsense. One of the most common catastrophes are surfaces, typically of oxides since these have many applications and calculations are easier than experiments, especially bad calculations. Very often people assume that one can simply cut the bulk and get something reasonable – the surfaces will shudder.

If you are lucky, you might be able to adjust things slightly to improve the convergence, there can be times where an expert user knows more about how the mixing will work than the code. This is most important at the start, and if you know that a problem can be badly behaved it may be useful to employ smaller values for the Unpredicted Greed for the first step, and also the initial trust radius for the total step – see the end for the parameters in Wien2k to do this. While there are ways to adjust the mixer later, this is probably not a good idea. Even the author of the code rarely can do better than the algorithm.

14. Don't Panic: Restarts and More Trusts

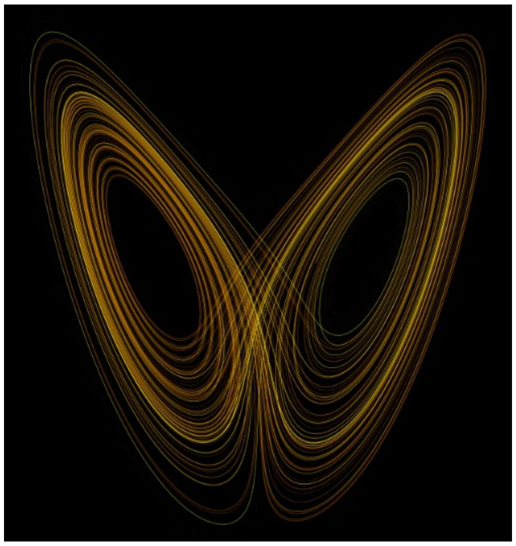
One well established method that sometimes helps which has been rediscovered multiple times for fixed-point as well as optimization is restarting – destroy all the prior history. The reason this can help is when the history is from a different type of problem – for instance the Colorado River history from the Grand Canyon will not be much use when crossing Lake Mead! Similar to this one could change the mode or add a Pratt step or two – again, breaking away from bad history. Unfortunately, there is no recipe that always works, the mixer does not have eyes and cannot tell when it has gone from mountain to lake….

One approach that can help in Wien2k is to increase the number of Trust Region controls. The default is only to control the total step size. One can add a Trust Radius for the total atomic step using ATLIM in case.inm. Other Trust Radii on the changes in the plane waves, density inside the spheres and density inside any one muffin tin can be added by specifying STIFF. Even more can be added by using STIFFER which borrows from older ideas, and does not allow any of the various parameters to increase unless the total Residue is decreasing.

Improve the Model…..the best approach.

15. Will the iterations always converge to the right answer?

The lowest possible energy, for the density by itself as well as when the atoms are included, is a point when density and other variables are all self-consistent (assuming that a fully consistent model is being used) – a fixed-point. However, the opposite is not true; it is possible to have fixed-points which are ***not*** the lowest possible energies. This can occur when one has 4f electrons, but there are other cases; for instance, it is possible with 3d electrons. Many of these are where the spin state of the self-consistent solution is different from that with the lowest possible energy.



Beyond that, nothing in fact guarantees that the iterations will converge! You will probably never run across a case like this, but they do exist. The case I know of is one where the density being used is some distance from the fixed-point, and has the wrong spin. In such a case you can end up having what appears to be something called a strange attractor, where the density can spiral around in a strange way in multiple dimensions, as shown on the left.[4] Behavior like this is known in Chaos Theory, and sometimes the behavior of the density seems to be quite chaotic, more chaotic than the user. (Really a piece of chocolate cake as mentioned much earlier, trying to escape being eaten by a black hole.)

Sometimes it seems to take forever, or the calculation may seem to be oscillating and the user does not know if the chocolate cake is slowly melting, getting burnt or reaching perfection. There are two main possibilities:

a) It is not really oscillating. For instance, it could be that the spin is slowly changing – this can seem to take forever. It might also be that the electronic structure has changed, for instance from metallic to insulating, and the Jacobian is slowly evolving. When the atoms are also varying there are times when rather drastic changes are slowly taking place.
b) It is slowly going through a canyon or following the path of a meandering river. In the literature we would talk about these as being "Tunnels". This can also seem to take forever. Patience is a virtue, eventually you will see the light.

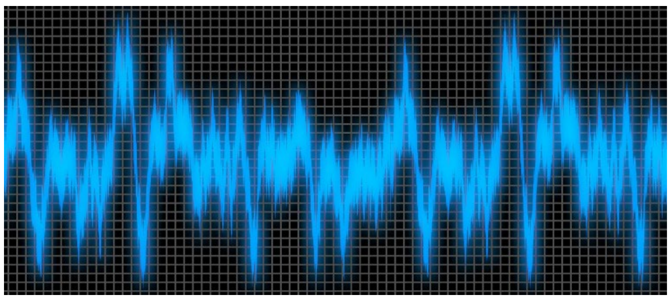
[4] This work by Alan Richmond ('Mandrian') is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

16. Dispelling some common misconceptions

Similar to many areas of science, myths develop which often are not accurate, but at one time were workarounds to fix problems. This is very much the case with mixing where there is a vast, unwritten literature, that is word of mouth rumors. Much of this is wrong.

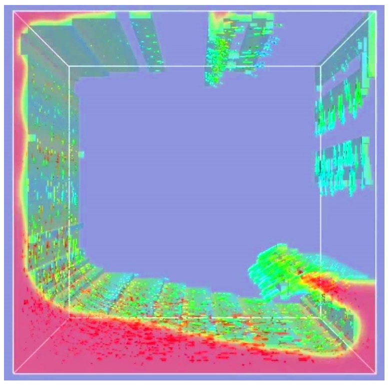
One of the most common is that one should reduce the mixing Greed if the problem is not converging. As discussed earlier, we are approximating by only including the first term in a Taylor series. If we use small steps we might avoid effects from the higher-order terms, but this does not have to be the case. Instead we have to limit the steps so that we remain in the region where the higher-order terms are small enough.

The opposite also is not right – for good problems increasing the mixing Greed does not have to be good. There is an optimum, and the process outlined earlier attempts to find this.



Another common one is that you should do some smearing over states. Again, this can work but it misses the real issue. If you have rapid variations in which states are occupied with changes in the potential, then there can be telegraph noise, that is rapid up/down changes as which states are occupied changes. Indeed, no computer program is perfect and there is almost always some numerical noise; computers (like people) are fallible.

The Taylor series expansion implicitly assumes that the problem is smooth, not randomly jumping around; smearing can smooth over the rough edges (we all have rough edges). The trust region controls can somewhat handle this, although unfortunately most DFT codes do not have these and assume that a naive use of a truncated Taylor series is enough.



A final common misconception goes under the name "sloshing", where electron density can run from one part of the cell to another. Some papers even call this a "feature" of the codes. If you think about the Grand Canyon from earlier, if you go too far you will hit the walls, and may then bounce back; if you go too far you might even bounce out of the Grand Canyon and end up in the desert in Arizona. Chocolate cakes do not do well in deserts. This is really another example of too much Greed – please learn to control yourself.

17. Start the Music

"All things truly wicked start from innocence"

Ernest Hemingway

Mixers, just like optimization codes and machine learning codes can be quite smart – once you give them a chance to learn from their mistakes. How to start a code on the path to learning is always tricky, and it is never going to be possible to code in an ideal answer without more information – which is of course a chicken and egg problem (which came first).

Almost always what one does is start cautiously, then allow the program to get more aggressive as information is accumulated. Being too cautious will slow the code down, the question is how much risk is tolerable? This will depend upon the type of problem being investigated – an experienced user may be much more risk-tolerant and know what to do when chaos occurs.

The Wien2k code makes a decent attempt to guess good parameters for the unpredicted step, but it can get it wrong. In cases where the residue is not a good descent direction, this can be bad. In addition, the code will attempt to partially correct the pseudo-charge in the first iteration, which can also be very bad in such cases. Then one has to start with a very small unpredicted step, e.g. 0.001 (echo .001 > .pratt) and avoid the correction in the first iteration (touch .NoPseudo).

" Success is not final, failure is not fatal: it is the courage to continue that counts"

Winston Churchill

18. Some technical details

For the sake of completeness, a few more technical details are included here, more of interest to coders in many cases.

18.1 Scaling of Variables

The most obvious approach to take is to use the variables that are most convenient for the DFT calculations, mainly the calculation of the potential, directly in the mixer. Beware, this is not the best choice! The "units" that matter in the earlier sections are dot products of the variables. These have to obey some rules:

a) If we repeat the unit cell, for instance go to a 2x2 supercell, the relative contribution of different components has to remain the same, since nothing has really changed.
b) For an isolated atom in a big cell where the outermost parts contain nothing (not even chocolate), add yet more outside vacuum cannot change anything.
c) If we use symmetry to reduce the number of atoms, the different components should have the same relative contributions if we change the symmetry.

This means that the densities have to be scaled such that they take account of degeneracy and also the volume over which each one occupies, such that the dot products scale as the density squared.

18.2 Absolute versus Relative

A relevant ambiguity is whether the trust radii and also the overall step Greed should be relative or absolute values. Because of how they are generated by looking at the previous step, the trust radii automatically behave as absolute numbers, with their radii reducing as the algorithm moves towards the fixed-point. The Greed for the Predicted step is a scaling factor, so it seems natural to use it as relative. (Other literature does this, but we might all be wrong in some cases.) For the Unpredicted step, and also the total step it seems best to average over the relative and absolute values.

18.3 Sequential versus Parallel

The method for constructing the matrices earlier used the current point as the origin, and referenced everything to it. In the optimization literature a different approach is used, where the analysis uses a sequential approach. In addition, instead of creating the Jacobian from all the steps, a sequence of updates are used to build it. The idea of this is that the oldest steps are less relevant than the current ones. In optimization there are ways to ensure that every step is going downhill, so we know that the more recent ones are better. However, this is less clear for fixed-point problems – think about the meandering river case.

18.4 Different Algorithms

There are a plethora of algorithms. With exact arithmetic (which is never possible), and for problems where the first-order Taylor series expansion is completely valid they are all the same. Many papers exist in the literature which claim that one is "better" than the others, but this almost certainly depends more upon what scaling has been used (or abused?) and other, un-reported facts as well as the type of problems picked as tests. In addition, almost no other codes in the DFT literature appear to try and control the Greed from an ab-initio approach, similar to what is described here.

The table below describes the various algorithms based upon what the T matrix is in the general form (repeating it for clarity):

$$H_k = S_k\,(T_k^T Y_k)^{-1}\,T_k^T \text{ and } B_k = Y_k\,(T_k^T S_k)^{-1}\,T_k^T \qquad (26)$$

At least with Wien2k where all of these have been tested with ***just*** the density and orbital potential parameters, the MSR1 approach is better than the others ***in general***. If the atoms are also "mixed", the MSR1 is significantly better than others. Is this always true beyond DFT – unclear.

Name	T Matrix	Centering	Form	Rescaling	Notes
Good Broyden	S	Sequential	Overwriting	None	Rare
Bad Broyden	Y	Sequential	Overwriting	None	Rare
DIIS	Y	Current	Matrix	None	Common
MSEC	Y	Current	Matrix	Diagonal	Obsolete
MSGB	S	Current	Matrix	Diagonal	Noisy
MSR1	Y+αS	Current	Matrix	Diagonal	Optimal?
HYB1	Y+αS	Current	Matrix	None	Good
HYB2	Y+αS	Sequential	Matrix	Diagonal	Good

18.5 Regularization

There are many inverses, some of which are well behaved, but others may not be. To avoid the inverses blowing up it is important to regularize. This damps or eliminates the small eigenvalues, which after inversion can have anomalously large effects. Remembering that there are always numerical issues, some of these small eigenvalues may be just noise.

For any matrix, the approach is to use a regularized Penrose-Moore form, which is the standard form:

$$Inv(W) = (W^T W + \delta I)^{-1} W^T \qquad (26)$$

The value of δ is hard to be accurate about, or derive. A rational way is to scale it to the maximum eigenvalue of the matrix W. A slightly more robust method of doing the inversion is to use Singular Valued Decomposition, then apply an approximate inversion to the singular values, that is

$$\gamma_i = \gamma_i / (\gamma_i^2 + \lambda^2) \qquad (27)$$

18.6 Dual versus Inverse

In the algorithms, a tricky question is how to construct inverses, since the matrices were built with only a limited number of histories so the inverses are not unique. (Non-unique matrices are a computational pain!) One way around this is to ignore the regularization, and use

$$B_k = Y_k \ \mathrm{Reg}(S_k^T S_k)^{-1} \ S_k^T \, and \ H_k = S_k \ \mathrm{Reg}(S_k^T Y_k)^{-1} \ S_k^T \qquad (26)$$

Without the regularization these are true inverses, with it they are approximate. We have to regularize to avoid the chocolate cake exploding, so it seems to be more appropriate to call these "Duals".

18.7 Backtracking

The exactly procedure for backtracking can only be reasonable; this is not something which is amenable to rigorous mathematics. The choice in Wien2k is:

a) If the step goes up by more than 2.0, then perform a quadratic fit using the last point and the current one to estimate the best size of the step that should have been taken. Then move to it, but do not store the density and other information for the step.
b) If the step goes up by more than 1.5, keep the information but go back to the last position and recalculate everything from the Jacobian to the Greeds.

An additional feature to stop oscillations is to ensure that in the ***next*** step the trust regions and Greeds do not increase (they can decrease). This ensures that we have two reasonable values.

18.8 Safety

In an ideal world all that is needed is to control the total step size, assuming that we can correctly scale changes to the atomic positions and their forces to the density, and both to other terms such as orbital potentials. Unfortunately the best one can do is something close, which appears to be using atomic units for everything. The algorithm can still go nuts if one does not include reasonable safety traps:

a) That in any given iteration, none of the mixing parameters changes by an extravagant amount. Limiting them to the range 3/5 to 5/3 seems reasonable.
b) Some hard limits, for instance not allowing atoms to move more than 0.1 au (0.05 Angstroms) in any step.
c) Reducing the trust region if there are obvious physical problems, for instance large changes in the potential or ghostbands.
d) Using additional trusts, for instance on how much the density around each atom can change. Similar to the main mixing Greed these are dynamically controlled by the code using predictions.

18.9 Pseudocharge

In Wien2k and similar codes, the interstitial density is described by Fourier components, and the density inside the muffin tins around the atoms by radial functions involving spherical harmonics. There is an issue with this that can have an impact on the mixing. The Fourier components are not limited to just the region outside the muffin tins, the go everywhere; the parts inside the muffin tins are cancelled out when various parts are calculated, for instance the potentials.

However, the mixer does not know that the density inside the muffin tins is not used. This density is called the pseudocharge, and plays a role in how the mixing behaves. (You cannot simply throw it away, as with a finite basis set this would lead to Gibbs oscillations – no free lunch.) When this charge is very large it can dominate the mixing, as reducing this part can be more important (for the total residue) than anything else. When the pseudocharge is converged, overall convergence is typically fast, but sometimes it takes many iterations for this to happen. A partial compensation is included in the first iteration, or if ".Pseudo" is present to help remove it, but it is not perfect.

18.10 Bookkeeping

Is a pain, but one of the nastiest parts of codes such as these. Indeed, there are more lines in the code to handle this than anything else! These range from simple ones such as recording the last positions and residues, to ones such as the mixing Greeds, trust radii and convergence testing metrics. Many of these are not so useful for the general user, but some are needed in the code, particularly for averaging to avoid anomalous jumps.

18.11 Coding Philosophy: Hands Off

When cake mixes came out in 1929 they sold, but by the 1950's their popularity had decreased. It took Ernest Dichter to recognize that the mixes were too easy; the cooks did not feel emotionally invested enough if all it took was to add water. The answer – have them also add eggs. (They also tasted better that way.)

A design fundamental of the mixer in Wien2k is that it should have as few parameters as possible. Currently the only ones are how rapidly to allow increases and decreases, the initial estimates for the different Greeds and hard limits. Everything is adjusted dynamically.

Similar to cooks, there is always a tendency for users to adjust the iterations – what else to do while it is running (except make chocolate cake)? There are some controls, but few are documented, quite deliberately. Indeed, if the user reduces the Mixing Greed in the main program this ***does not do what they think it does!*** All it really does is increase the number of trust region controls, and also reduce the speed at which parameters can improve. One has to give users something to play with, and ensure minimal harm.

User Guide Details: MIXER (adding and mixing of charge densities)

In **mixer** the electron densities of core, semi-core, and valence states are added to yield the total new (output) density (in some calculations only one or two types will exist). Proper normalization of the densities is checked and enforced. Other terms such as orbital potentials and density matrices are also mixed. As it is well known, simply taking the new densities leads to instabilities in the iterative SCF process. Therefore it is necessary to stabilize the SCF cycle. Several mixing schemes are implemented, but we mention only:

1. Straight mixing as originally proposed by Pratt (52) with a mixing greed Q

$$\rho_{new}(r) = (1-Q)\rho_{old}(r) + Q\rho_{output}(r)$$

2. A Multi-Secant mixing scheme contributed by L. Marks (see Marks and Luke 2008), in which all the expansion coefficients of the density from several preceding iterations (usually 8-12) are utilized to calculate an optimal direction in each iteration. This version is by far superior to the other schemes making them quite obsolete. It is robust and stable (works nicely also for magnetic systems with 3d or 4f states at EF) and usually converges at least 30 % faster than the old BROYD scheme.
3. Two new variants on the Multi-Secant method including a rank-one update (see Marks 2013, 2021) which appear to be faster and equally robust

At the outset of a new calculation (for any changed computational parameter such as k-mesh, matrix size, lattice constant etc.), any existing **case.broydX** files should be deleted (since the iterative history which they contain refers to a ``different`` incompatible calculation).

If the file **case.clmsum_old** cannot be found by **mixer**, a ``PRATT-mixing`` with a mixing greed of 1.0 is done.

*Note: a **case.clmval** file must always be present, since the LM values and the K-vectors are read from this file.*

The total energy and the atomic forces are computed in mixer by reading the **case.scf** file and adding the various contributions computed in preceding steps of the last iteration. Therefore **case.scf** must not contain a certain ``iteration-number'' more than once and the number of iterations in the scf file must not be greater than 999.

For LDA+U calculations **case.dmatup/dn** and for hybrid-DFT (switch -eece) **case.vorbup/dn** files will be included in the mixing procedure. With the new mode MSR1a (or MSECa) atomic positions will also be mixed (effectively optimized).

1 Execution

The program `mixer` is executed by invoking the command:

mixer mixer.def or **x mixer [-eece]**

A spin-polarized case will be detected automatically by `x` due to the presence of a case.clmvalup file. For an example see fcc Ni (sec. 10.2) in the *WIEN2k* package.

2 Dimensioning parameters

The following parameters are collected in file `param.inc`, :

NCOM	number of LM terms in density
NRAD	number of radial mesh points
NSYM	order of point group
Traptouch	minimum acceptable distance between atoms in full optimization model

3 Input

Below a sample input (written automatically by `lstart`) is provided for TiO_2 (rutile), one of the test cases provided with the *WIEN2k* package.

```
------------------ top of file: case.inm --------------------
MSR1  0.d0  YES  (PRATT/MSEC1 background charge (+1 for additional e), NORM
0.2              MIXING GREED
1.0 1.0          Not used, retained for compatibility only
999 8            nbroyd nuse
------------------- bottom of file ------------------------
```

Interpretive comments on this file are as follows:

line 1:

(A5,*)
switch, bgch, norm

switch	MSR1	Recommended: A Rank-One Multisecant that is slightly faster than MSEC3 in most cases. For MSR1a see later.
	MSEC3	Multi-Secant scheme (Marks and Luke 2008). This is similar to DIIS, with trust regions added.
	MSEC4	Similar to MSEC3 (above), but mixes the higher LM values inside spheres by an adaptive PRATT scheme. This leads to a significant reduction of program size and file size (`case.broyd*`) for unit cells with many atoms and low symmetry (factor 10-50) with only slightly worse mixing performance.
	MSR2	A variant of MSR1 which only mixes the L=0 values inside the spheres, similar to MSEC4
	PRATT	Pratt's scheme with a fixed greed
	PRAT0	Pratt's scheme with a greed restrained by previous improvement, similar to MSEC3

bgch	Background charge for charged cells (+1 for additional electron, -1 for core hole, if not neutralized by additional valence electron)

norm	YES	Charge densities are normalized to sum of Z
	NO	Charge densities are not normalized

line 2:

free format

Greed	Mixing Greed Q. Essential for PRAT0 and PRATT, less important for Multisecant methods. Q is automatically controlled by the program. Decreasing it turns on some more controls to reduce the aggressiveness. One should rarely reduce this below 0.05.

line 3 (optional):

(free format)
f_pw, f_clm

f_pw	Not used, retained for compatibility only.
f_clm	Not used, retained for compatibility only.

line 4 (optional):

(free format)
nbroyd, nuse

nbroyd	Not used, retained for compatibility only.
nuse	For all the Multisecant Methods: Only nuse prior steps are used (this value has some influence on the optimal convergence. Usually 6-10 seems reasonable and 8 is recommended). For MSR1a of large cells sometimes 16 is better.

Line 5 or more (optional), additional switches:

For Hard Problems:

STIFF	**Recommended** if you run into severe convergence problems, this will probably solve them. However, improving your model (e.g. RKMAX, RMT, k-points) is more likely to help
STIFFER	A version of **STIFF** that does not increase Trust Radii unless there is an improvement
ATLIM	Turns on the atom movement limit in the default mode with only step size control
TRAD X	Reduces the maximum atom movement to X au from the default 0.1. For hard problems **TRAD** 2E-2 may be useful
BLIM X	Reduces the upper step bound **X**, for instance to **1.0**

Other, only for experts:

FAST	May be faster for well-conditioned problems
PRATT	Uses **PRATT** mode with the step size control of **MSR1**
LAMBDA X	Fixes the regularization value to **X**, for instance 1D-4
VERBOSE	Outputs additional information that may not be useful for most people
LESS	Reduces the amount of information **VERBOSE** produces
RESTART X	Restarts from a converged density with an initial **GREED** of **X**
ELIN	Mixing of global linearization energy, can be useful for d or f electrons
DIIS	The DIIS (PULAY, ANDERSON) method, with are unscaled MSEC3. Not recommended.
MSGB	Multisecant Good Broyden, not recommended.
MSEC3	The same as MSEC3 on the first line. (DIIS/PULAR/ANDERSON/MSGB also)

Control files – for expert fine control, use with care

In all cases below values are changed for the next iteration only. The file is then deleted.

.msec	The existence of the file .msec forces the next step to use an unpredicted greed which is read from the file. The code deletes the file if it is present. It can be useful in the very first iteration, for instance to use a value of 0.005 since the initial density can be quite bad. Also, on some cases the algorithm can get trapped with small greed terms (e.g. 0.001), and this allows it to be reset higher.
.Beta	The existence of this file forces the next step to use the predicted greed in the file.
.pratt	The existence of the file .pratt forces the next step to be a Pratt step, without a restart. If the file contains a number, this is the mixing greed used. The code deletes the file if it is present. It can be useful to have a small value such as 0.005 or even 0.0025 for the very first iteration.
.BLim	Changes the trust region to value in the file for the next iteration – do grep :TRUST case.scf
.Climit	Changes the charge trust region of single atoms to the value in the file for the next iteration – do grep :TRUST case.scf. This only works if STIFF is set.
.CTOlimit	Increases the charge trust region of all atoms to the value in the file for the next iteration – do grep :TRUST case.scf. This only works if STIFF is set.
.PWlimit	Changes the plane wave trust region to the value in the file for the next iteration – do grep :TRUST case.scf. This only works if STIFF is set.
.ATlimit	Changes the charge trust region for atoms movement to the value in the file for the next iteration – do grep :TRUST case.scf. This only works if ATLIM or STIFF is set.
.Pseudo	Do a pseudo-charge correction in the next iteration. File is then deleted.
.NoPseudo	Do not do pseudo-charge correction in first iteration. File is then deleted.
.push	Increases all the trust radii and Greeds by 1.5 in the next iteration. File is deleted.
.pull	Reduces all the trust radii and Greeds by 1.5 in the next iteration. Files is deleted
.fast	Presence uses faster (less safe) controls for the next step only.
.restart	Restarts from the prior density, preserving the trust region radii and exploiting prior information about the GREED.
.minstop	Switches from atoms+density to just density convergence.
.minstart	Switches from just density to atoms+density convergence.
.forcedmat	Forces the density matrices to be those of a full Pratt step
.forceorb	Forces the orbital potential to be that of a full Pratt step

Output from the Mixer in Wien2k

A number of informational values are output in the file case.scfm. A smaller set is output for a non spin-polarized case than for a polarized case, and additional information is output when the atom positions are simultaneously being mixed. Some of these are only output if VERB is included in case.inm; some are skipped if LESS is included. Below is a brief list with a few notes on each. The most important ones are indicated in bold, some of which are output by the program Check_lapw. Below is a guide, which is not complete.

:CINT	Core integrals for each atom, should be integers or very close.
:RTO	Density at nucleus, used for Mössbauer Isomer shifts.
:HFF	Hyperfine Fields for each atom, used for Mössbauer Hyperfine fields and NMR Knight shifts (Contact term).
:NT	New densities, i.e. $F(\rho_k)$ both in the interstitial (:NTO) and for each atom.
:OT	The same for the old density ρ_k.
:CT	The same for the mixed density ρ_{k+1}.
:NPC	Pseudo charge in $F(\rho_k)$. All three of these values should be similar.
:OPC	Pseudo-charge for the old density ρ_k.
:CPC	Pseudo-charge for the mixed density ρ_{k+1}.
:DT	RMS difference for ***just*** the L=0 density in each sphere. The other L values can be large, and are not included here (they are in FULLRMS).
:DIS	A weighted sum of the :DT values above, used as an approximate convergence criterion.
:PW Check	The total number of plane waves checked if STIFF is in case.inm.
:FULLRMS	L2 Metric overall atoms, the residue used in the code. More accurate than :DIS.
:PLANE	Properly scaled plane wave density and residue, a guide that could be used.
:CHARG	Properly scaled density inside the muffin tins and residue, a guide.

Step History Values for the last steps used of key parameters:

Dmix	Mixing Greed before any Trust Radius.
Dmixt	Mixing Greed after any Trust Radius applied, so often small than Dmix.
Red	How much reduction there was in the step in terms of FULLRMS.
Pred	The Predicted Reduction, if the Predicted Step was 100% correct.
Step	The relative Trust region size for the step.
Lambda	The value of λ for $T_k = \lambda S_k + Y_k$.
MagAbs	The absolute Trust Region size, that is the Step multiplies by the movement.
Beta	The Predicted Greed.

Orbital rescaling	Scaling of the orbital potential terms if they are mixed, including as a weight how many electrons there are in each state.
Compression scl	How much each history in T_k has been multiplied after scaling the diagonal of $Y_k^T Y_k$.
Decomposed Diagonal	Values for the diagonal values for different parts of the residue, useful to check scaling.
:PRED	The Predicted Trusts for the current step. These are combined with the history.
Eigenvalues	Lists the eigenvalues of $S_k^T S_k, S_k^T Y_k, Y_k^T Y_k and\ T_k^T T_k$. Ideally these should all be positive, and negative or complex values in $S_k^T Y_k$ are indicative of complex problems.
Singular Values	These are what is used to regularize $T_k^T T_k$. The Projection is the multiplication by the current residue. For complex problems the Projection is larger for the smaller singular values, those with least confidence.
:DLIM, BLIM	With VERB, these indicate different stages of creating the Greed terms, see the code.
:RANK	The rank of both $Y_k^T Y_k and\ T_k^T T_k$. Ideally the rank should be similar to the number of values, indicating that all values of the history are contributing, for instance better than 80%. Often it is small, which indicates that there are equivalent history directions, and the mixing is complex.
:TRUST	Values for the Trust Radii, not all of which are used (depend upon the flags).
Past History Usage	How much of each prior value was used.
:DIRM	Information on the memory, the reduction of the last step, what was predicted and what is predicted for the next step. Ideally the reduction should be similar to the prediction.
:DIRT	The size of the total step, that of a Pratt step and the angle between them. Ideally this angle should be small, but any value 0-70 degrees is fine. Large angles around 90 degrees indicate a complex problem. Values larger than this indicate that the algorithm has reversed direction relative to the Pratt step – which can happen and is not a reason to panic.
:DIRD	The same for the orbital terms.
:DIRA	The same for movement of the atoms.
:DIRP	The same for the plane waves.
:DIRQ	The same for the density inside the muffin tins.
:APOS	Movement for each atom, and estimate of how far it still has to go.
:FRMS	RMS of forces & L1 sum, also the RMS & L1 of the movement and the largest.

:MIX	The mixing used, with regularization, the Mixing Greed, what type of Trust Region Step was used and how large it was relative to a Pratt step. The term “Newton” means that no Trust Region was applied; LMStep means it was. The Trust region that limited the step is indicated. LMBack means that it went back to the last step and recalculated; Back means that it did a quadratic approximation for a bad last step.
:MMT	total spin magnetic moment in unit cell
:MMI	The difference between the integrated up and dn spin densities inside the muffin tins.
:NEC	Total densities, NEC01 is from $F(\rho_k)$, NEC02 from ρ_k and NEC03 from ρ_{k+1}.
:PW Change	Some diagnostics for how some of the plane wave components change, both those which have small (hkl) and large.
:ENE	Approximation for the energy using a Harris (aka Harris-Foulkes) functional.
:FOR, :FCA, :FGL	Forces in mRyd/au.
:STRESS_GPa	Stress tensor in GPa. “partial” indicates incomplete tensor (for convergence check), “total” is the complete tensor.