Improvement of the MSR1a algorithm in the WIEN2k code

Theo Guerber

Acknowledgement

First, I would like to acknowledge Xavier Rocquefelte for offering me a very rewarding internship and for being my tutor during this one.

I would like to express my very great appreciation Laurence Marks for working with me throughout the internship.

I acknowledge Yvon Lafranche for agreeing to be my reference teacher.

I thank the CTI team of ISCR for hosting me during my internship.

Abstract

Researchers working in theoretical chemistry work not with chemical compounds and glassware but with molecular and solid optimization software. These programs are based on optimization algorithms. The topic for my internship was to try to improve the algorithm named MSR1 of WIEN2K software used for density functional theory. The problem is that due to calculations of eigenvalues and of eigenvectors of matrices of considerable size, an iteration of the algorithm can take from a few minutes to several hours. Thus, the ideal is for the algorithm to converge but in a reduced number of iterations.
So I first did a bibliography and studied the associated mathematical theory that I present in chapter 1. Then, I coded a program in MATLAB language in order to obtain usable data that I present in chapter 2. Finally, I analyzed the data collected in order to find useful clues for the improvement of the algorithm, results that allowed an advance and a future upgrade of the software, all this is presented in chapter 3.

Table of Contents

Acknowledgment **3**

Abstract **3**

1 Internship's context **4**
1.1 ISCR presentation 4
1.1.1 CTI (Chimie Theorique Inorganique) team 4
1.1.2 WIEN2K problematic 4
1.2 Density functional Theory 5
1.2.1 Kohn-Sham equations 5
1.2.2 Fixed-point theory 5
1.3 State of the art 6
1.3.1 Quasi-Newton method 6
1.3.2 Approximation of matrix H_n in MSR1 algorithm 6

2 Modelling and programming **8**
2.1 Modelling of a "WIEN2K mixer" 8
2.1.1 Choice of programming language 8
2.1.2 Recreation of MSR1 algorithm 8
2.1.3 Test functions 11
2.2 Transformation in a "real" program 12
2.2.1 IHM 12
2.2.2 Functions 14
2.2.3 Algorithm 14

3 Result and futurs researchs **15**
3.1 Tests and results 15
3.1.1 Iteration according to α 15
3.1.2 Eigenvalues/Eigenvectors of H_n 16
3.1.3 Study of eigenvalues/eigenvectors of the other matrices of the algorithm 17
3.2 Conclusion of the results and future prospects 17
3.2.1 Contribution of the internship 17
3.2.2 Trust-region control 18

Chapter 1

Internship's context

1.1 ISCR presentation

The Beaulieu campus brings together a large number of infrastructures dedicated to research. I did my internship in the one dedicated to chemistry: Institut des sciences chimiques de Rennes (ISCR). ISCR has 280 permanent staff and several research teams. I worked with the CTI (Chimie Theorique Inorganique) team.

1.1.1 CTI (Chimie Theorique Inorganique) team

The CTI team works differently from other teams. Where the majority of other teams use chemistry equipment, the CTI team uses server clusters. This team uses many softwares often designed by the chemists themselves. These softwares often have the function of solving chemical equations as for example the Schrodinger equation. Most of the time, software calculations are staggering, hence the use of server clusters. The idea behind my involvement with the CTI team was to try to improve the computational time of the software. The software to which I have contributed is called WIEN2K.

1.1.2 WIEN2K problematic

Laurence Marks, the researcher I worked with during my internship, has already made improvements to WIEN2K and has written an article about it. This article was the core of my internship.WIEN2K is linked to two complementary algorithms. Laurence Marks' idea is to perform a linear combination of these two algorithms. So I worked during my internship on this linear combination in order to extract information that could make it optimal. But before I start explaining what I did during my internship, I will first explain the chemical theory related to WIEN2K and the state of the art of the WIEN2K algorithm.

1.2 Density functional Theory

As stated on the website dedicated to the WIEN2k software: WIEN2k allows to perform electronic structure calculations of solids using density functional theory (DFT). I will not detail all the DFT here but focus on what concerns me mainly in the DFT, the Kohn-Sham equations.

1.2.1 Kohn-Sham equations

Kohn-Sham equations are equations of the type Schrodinger equations, that is equations whose solutions are eigenfunctions and eigenvalues.

$$H_{KS}\Phi_k = \varepsilon_k \Phi_k \tag{1.1}$$

Here, H_{KS} is a functional, Φ_k eigenfunctions and ε_k eigenvalues. Due to the nature of the functional, these equations can only be resolved in an iterative approach. Moreover, since chemists are required to solve this equation in the case of solids, the resolution of an iteration can take a long time, this adds a constraint, the algorithm to be optimized must converge but must also converge in few iterations. Mathematically, this can therefore be summed up as a fixed point problem on a vector function.

1.2.2 Fixed-point theory

Let us consider a vector function F, that is to say:

$$F : \mathbb{R}^N \to \mathbb{R}^N \tag{1.2}$$

Finding a fixed point of F is like finding $x_* \in \mathbb{R}^N$ such that :

$$F(x_*) = x_* \tag{1.3}$$

Let's put G, an other vector function, as:

$$G : x \to F(x) - x \tag{1.4}$$

Finding a fixed point of F is like finding a root of G. One method to find an approximate value of x_* is Newton's generalized method at these vector functions. x_* becomes the limit of a series (x_n) whose recurrence is:

$$x_{n+1} = x_n - J_G(x_n)^{-1}.G(x_n) \tag{1.5}$$

where J_G is the jacobian matrix of the function G. That is:

$$J_G(x_n) = B_n \quad B_n^{-1} = H_n \in \mathcal{M}_N(\mathbb{R}) \quad b_{i,j} = \frac{\partial g_i}{\partial x_j} \tag{1.6}$$

In the context of the DFT, the number of unknown, N, of the vector function could reach values in the order of 10^4. In this case, calculating the inverse of a matrix of this size becomes prohibitive at the level of computational costs. An approximation of the matrix is therefore necessary.

1.3 State of the art

The approximation of the matrix and which method to choose to approximate hold a central place in the article by Laurence Marks. Indeed, the linear combination I mentioned earlier is a linear combination of two approximations of this H_n matrix. In the event that the calculation of H is too expensive, the Newton method is therefore applied with a matrix approximated according to the previous ones. Newton's method then becomes a quasi-newton method.

1.3.1 Quasi-Newton method

In a Quasi-Newton's method, we don't calculate the matrix H_n, we approximate it. The idea is a generalization of the secant method for multidimensional problems.

$$\tilde{B}_n.(x_n - x_{n-1}) = G(x_n) - G(x_{n-1}) \tag{1.7}$$

where $\tilde{B}_n$ is the approximation of the Jacobian. There are several methods to approximate this matrix. The best known methods are:

- Davidon-Fletcher-Powell (DFP)
- Broyden-FLetcher-Goldfarb-Shanno (BFGS)
- Symmetric rank one
- "Good" and "Bad" Broyden

It has been shown in previous work[1] that it is the Broyden methods that are most suitable in the context of DFT.

1.3.2 Approximation of matrix H_n in MSR1 algorithm

As mentioned earlier, the algorithm in WIEN2K is a linear combination of the two Broyden algorithms. This algorithm being the core of my internship, we will detail its functioning in order to fully understand the associated problem.

"Good" and "Bad" Broyden

First of all, we introduce new variables :

$$y_{j,n} = G(x_n) - G(x_j) \tag{1.8}$$
$$s_{j,n} = x_n - xj \tag{1.9}$$

In his article, Broyden used two methods[2]. The first, referred to "good" Broyden's method is to find an approximation of B_n in respect of Frobenius norm. So we have[1] :

$$B_{n+1} = B_n + \frac{(y_n - \sigma_n B_n s_n) s_n^T}{\sigma_n ||s_n||^2} \tag{1.10}$$

where $||s|| = \sqrt{s^T s}$ is the Euclidian norm and σ_n is a step size parameter. We need a approximation of B_n^{-1} so (1.10) become :

$$B_{n+1}^{-1} = B_n^{-1} + \frac{(\sigma_n s_n - B_n^{-1} y_n) s_n^T B_n^{-1}}{s_n^T B_n^{-1} y_n} \tag{1.11}$$

The second method, referred to "bad" Broyden's has directly an approximation of $H_n = B_n^{-1}$

$$H_{n+1} = H_n + \frac{(\sigma_n s_n - H_n y_n) y_n^T}{||y_n||^2} \tag{1.12}$$

Multi-secant method

The weakness of the Broyden approximations presented above is that they are based only on the previous step, the idea is therefore to combine them with multi-secant method in order to make an approximation of H_n no longer with only the previous step but with several previous steps. The idea of multi-secant method is precisely to use secant method for multidimensional problem, not only with one previous step but also many other previous steps[3].

The mixing of methods

Now, we combine the multi-secant method with the Broyden's method. First of all, we create matrix of many precedent steps :

$$S_n = [s_{n-k,n}, s_{n-k+1,n}, \cdots, s_{n-1,n}] \tag{1.13}$$

$$Y_n = [y_{n-k,n}, y_{n-k+1,n}, \cdots, y_{n-1,n}] \tag{1.14}$$

After calculations, we obtain the following approximation of H_n

$$H_n = \sigma_n I + (S_n - \sigma_n Y_n)(Y_n^T W)^{-1} W^T \tag{1.15}$$

Whither W is a matrix with same size than Y_n or S_n[4].
Thus L. Marks created a new algorithm called MSR1 algorithm, wherein he used a linear combinaisons of "good" and "bad" Broyden's algorithms :

$$W = Y_n + \alpha S_n \quad \alpha \geqslant 0 \tag{1.16}$$

It must be understood that the range of values proposed by Laurence Marks in his article was purely conjectural. During my internship, I will try to find an optimal value of α for many problems as possible or find goods clues about this value.

Chapter 2

Modelling and programming

2.1 Modelling of a "WIEN2K mixer"

I had the idea to create a fictitious model of the algorithm of L. Marks because I wanted to observe the number of iterations according to the value of α with test functions found in the literature.

2.1.1 Choice of programming language

The choice of the programming language was made naturally, knowing in advance that the algorithm would have to manipulate matrices of consequence sizes, I chose the language created with the aim of manipulating matrices: MATLAB.
Its downside is that it is not opensource like other language but I had a license on my personal computer during the internship and if it had been necessary, it was always possible to transpose the work into a free equivalent like Scilab.

2.1.2 Recreation of MSR1 algorithm

Here I will present the algorithm I created based on the work of L. Marks. In section 1.3.2, I presented the general idea of the MSR1 algorithm, here we will explain the different parts of the algorithm (initialization, regularization and pseudo-inversion) whose purpose is to find a root of the following function:

$$F : \mathbb{R}^N \to \mathbb{R}^N \tag{2.1}$$

Initialization

As I said in section 1.3.1, the algorithms of the quasi-Newton methods use the results of the previous step or steps. But as we see on equations (1.8) and (1.9), as a difference is made, it is necessary to have at least two points.

The test functions I acquired were presented with an initial point x^0. In order not to start in the wrong "direction", the point x^1 is calculated as follows:

$$x^1 = x^0 - p \cdot F(x^0) \tag{2.2}$$

Where p is a arbitrary value analogous to Pratt step[4]. In my program the p value was set to: 0.01.

Approximation of $J_F(x^n)^{-1}$

We need to find a good value of H_n approximation of $J_F(x^n)^{-1}$. In order to best recreate the MSR1 algorithm, I drew on the algorithm presented in the two articles by L. Marks.[4][1].
First of all, like in the section 1.3.2, I create new variables

$$y_{j,n} = F(x^n) - F(x^j) \tag{2.3}$$

$$s_{j,n} = x^n - x^j \tag{2.4}$$

and associated matrix :

$$S_n = [s_{n-k,n}, s_{n-k+1,n}, \cdots, s_{n-1,n}] \tag{2.5}$$

$$Y_n = [y_{n-k,n}, y_{n-k+1,n}, \cdots, y_{n-1,n}] \tag{2.6}$$

Then, I create a regularization matrix Ψ_n[1]. The use of the regularization matrix helps to avoid creating instability related to the accuracy of the calculations. In fact, the closer we get to the root the closer the matrix Y_n gets to the null matrix, the regularization gets closer to a normalization of the matrix Y_n.

$$\Psi_n = \begin{pmatrix} 1/||y_{n-k,n}|| & 0 & \cdots & 0 \\ 0 & 1/||y_{n-k+1,n}|| & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1/||y_{n-1,n}|| \end{pmatrix} \tag{2.7}$$

Finally, I create the following matrices, always based on the work of L. Marks[1] :

$$A_n = \Psi_n Y_n^T (Y_n + \alpha S_n)\Psi_n \tag{2.8}$$

$$H_n = \sigma_n I + (S_n - \sigma_n Y_n)\Psi_n A_n^{-1}\Psi_n (Y_n + \alpha S_n)^T \tag{2.9}$$

Here, σ_n is what L. Marks calls *algorithm greed* in his article[4]. The notion of *algorithm greed* is important because as L. Marks says in his article, a major difference between the "Good" and "Bad" Broyden algorithm is the fact of being a *greedy algorithm* or not. A definition of an *greedy algorithm* given in the article of L. Marks is as follows:

> *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.*

In order to focus on the linear combination of the two algorithms, it was chosen to take a constant σ_n value.

$$\sigma_n = \sigma = 0.01 \tag{2.10}$$

Singular value decomposition

As we see in equations (2.8) and (2.9), matrix A_n must be inverted but that matrix can be singular(not invertible). So we don't calculate the inverse but the pseudo-inverse. Suppose that we need to invert A. First of all, we used a singular-value decomposition.

$$A = USV^T \tag{2.11}$$

S is a singular matrix (diagonal matrix with singular values of A). On MATLAB, I use the command **svd** to find U,V,S. After that, we create the pseudo-inverse of S.

$$s_{ij}^+ = \frac{s_{ij}}{s_{ij}^2 + \beta} \tag{2.12}$$

β is a constant added to the denominator to avoid divisions by zero. Finally, the pseudo-inverse of A, B is created :

$$B = VS^+U^T \tag{2.13}$$

Thus, in my algorithm, the equation to calculate H_n is the following :

$$H_n = \sigma_n I + (S_n - \sigma_n Y_n)\Psi_n B_n \Psi_n (Y_n + \alpha S_n)^T \tag{2.14}$$

with B_n the pseudo-inverse of A_n.

Divergence

In some cases, we have a divergence of the algorithm, $||F(x^n)|| \longrightarrow +\infty$. So, if $||F(x^n)||$ exceeds a certain value, the algorithm is stopped and the number of iteration is fixed to a arbitrary value.

$$||F(x^n)|| > M \quad N_{iter}(\alpha) = N_{iter^{max}} * 1.1 \tag{2.15}$$

Stop criterion

In order not to have an algorithm stuck in an endless loop, I set a stop criterion related to the number of iterations. thus, if the algorithm exceeds a number of iterations set by the user, it stops even if the stop criterion linked to function F is not yet reached.

The function-related shutdown criterion is the one that when achieved first shows the effectiveness of the algorithm. Since the goal of the algorithm is to reach an F root, we set an ε criterion such that we have:

$$||F(x^n)|| < \varepsilon \tag{2.16}$$

the algorithm stops.

2.1.3 Test functions

In order to represent as accurately as possible the calculations occurring in WIEN2K software, it was necessary to use test functions whose size was not fixed in advance and could be chosen by the user. It was also necessary that the functions found give with the algorithm created exploitable results.

After some research, here are the test functions used in the MATLAB program. They come from an article testing different algorithms using quasi-Newton methods[5]. In short, the role of the test functions is to simulate a fixed point problem with a large vector function. Thus the functions shown below are vector functions of which we know the existence of a fixed point. They were also selected by the researchers to test the robustness and speed of convergence of algorithms using quasi-Newton methods.

The extended Rosenbrock function

$$\begin{array}{c} F_{2i-1}(x) = 10(x_{2i} - x_{2i-1}^2) \\ F_{2i}(x) = 1 - x_{2i-1} \end{array} \quad i = 1, \cdots, \frac{m}{2} \tag{2.17}$$

The extended Powell function

$$\begin{array}{l} F_{4i-3}(x) = x_{4i-3} + 10x_{4i-2} \\ F_{4i-2}(x) = \sqrt{5}(x_{4i-1} - x_{4i}) \\ F_{4i-2}(x) = (x_{4i-2} - 2x_{4i-1})^2 \\ F_{4i}(x) = \sqrt{10}(x_{4i-3} - x_{4i})^2 \end{array} \quad i = 1, \cdots, \frac{m}{4} \tag{2.18}$$

The Broyden tridiagonal function

$$\begin{array}{c} F_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1 \\ x_0 = x_{m+1} = 0 \end{array} \quad i = 1, \cdots, m \tag{2.19}$$

The Broyden banded function

$$\begin{aligned} F_i(x) &= x_i(2+5x_i^2)+1-\sum_{j\in J_i} x_j(1+x_j) \\ J_i &= \{j : j\neq i, max(1,i-m_l)\leqslant j \leqslant min(m,i+m_u)\} \qquad i=1,\cdots,m \\ m_l &= 5, m_u = 1 \end{aligned} \tag{2.20}$$

The Brown almost-linear function

$$\begin{aligned} F_i(x) &= x_i + \sum j = 1^m x_j - (n+1) \quad 1\leqslant i < m \\ F_m(x) &= \left(\prod_{j=1}^{m} x_j\right) - 1 \end{aligned} \tag{2.21}$$

2.2 Transformation in a "real" program

Along with the work of running the algorithm and studying the resulting data, I had the will to make it a real program with a minimum of human-machine interface in order to be able to manipulate the algorithm more easily.

Thus, at the end of the course, the program is presented in the form shown in figure 1 in the appendix.

The program consists of three parts:

- Human-machine interface (IHM in french)
- Functions
- Algorithm

2.2.1 IHM

This part of the program aims to simplify the launch of the program, because although it is a very simplified version of a part of WIEN2K, it is nevertheless rather complex. The first dialog box, called "Function Selector" allows the user to choose which test function he will use (cf figure 2). The choice to put the function selection before selecting the values of the parameters internal to the algorithm is that each function does not have the same values for which the results are useful.For this reason, depending on the function chosen by the user, the default parameter values that can be chosen by the user are not the same.

As we see in figure 3, the parameters that can be modified are the parameters that we discussed in the sections above.

α:

The choice of the first parameter is particular because it will not set a value for alpha but create a vector containing all the alpha values between the min value and the max value chosen with the chosen step between each value. For each alpha value, we will run the algorithm and retrieve data specific to that value.

β:

The goal of β is to be able to create a pseudo-inverse as just as possible, this value represents the compromise between getting as close as possible to the mathematical limit while avoiding calculation errors due to zero machine

σ:

As explained in section 2.1.2, the sigma parameter is *algorithm greed*. Although it has been set to 0.01 by default, the user can always change it if desired, however, the fact that it varies with each iteration has not been implemented in the program.

"Pratt step" :

As just above, the "Pratt step" is explained in the section 2.1.2. It can of course also be changed by the user.

Number of variables :

This number refers to N in equation (2.1). In the context of DFT, the order of magnitude of N is 10^4 but in order to limit memory requirements and shorten calculation times, the default value is 200. This value is high enough to allow through a sufficient number of data as will be seen later.

Iteration maximum :

This parameter allows the algorithm to stop if there is no convergence. I had to vary its value regularly during my tests during my internship. This will be detailed in the next chapter but to sum up, by increasing it sufficiently, I was able to reveal cases of very slow convergence (oscillation before convergence).

ε :

This parameter is the stop criterion of the algorithm. In order to ensure that we have a convergence and not just an oscillation or a "stroke of luck", we need to restart a simulation identical to the previous one and diminishing this parameter.

2.2.2 Functions

As can be seen in figure 1, it is in this part that the functions called by the "function selector" dialog box have been coded. We may notice that not all the functions mentioned in section 2.1.3 are present, this will be explained in the next chapter.

2.2.3 Algorithm

It is in this part of the program that the algorithm presented in the section 2.1.2 takes place.

MSR1

This function aims to recover the data chosen by the user, to start a loop on the different alpha values and in this loop, apply the MSR1 algorithm for the different alpha values. There were also coded functions finally to extract useful data. To do all these actions, it uses several functions.

Data : This function will create the X_n, Y_n and S_n matrices using the previous points. The previous maximum step used is 8 because it turned out that it was an optimal value to have conclusive results.

Hcalcul : This function will simply calculate H_n according to X_n,Y_n,S_n and of course α.

eigactive : This function will calculate Sh_n which will be explained in section 3.1.2.

Eigcomp : The purpose of this function is to count the number of H_n's eigenvalue possessing a negative real part and that of which the imaginary part is not zero.

Chapter 3

Result and futurs researchs

3.1 Tests and results

3.1.1 Iteration according to α

As I explained in the previous chapter, I created my algorithm so that it will vary the value of α and performs my pseudo algorithm MSR1 while counting the number of iterations to converge. So I ended up with graphs of the number of iterations based on the α.There was no clear correlation between the different alpha values and the number of iterations needed to converge. Some functions, however, yielded interesting results.

Rosenbrock function case with 50 variables

As can be seen in figure 4, although no correlation can be traced, it is already observed that beyond a certain alpha value, non-convergent divergence and oscillations appear.As I already explained in section 2.1.2, I coded the algorithm in such a way that we can distinguish between cases of explosive divergence and cases of non-convergent oscillation. Thus the points between 10000 and 12000 iterations are false points because the algorithm stopped before having to calculate values too high for the MATLAB computing capacity. This function seemed to be a good candidate to study non-convergent oscillation cases.

Gueri-Mancino function with 200 variables

As can be seen in figure 6, the Gheri-Mancino function has the default, if you can put it that way, of converging for any alpha value. The variation being so small and the convergences so fast that I didn't have enough data to be able to extract information from it. That is why I decided not to use this function for more advanced tests.

Broyden tridiagonal function with 200 variables

In the case of the tridiagonal Broyden function, as can be seen in figure 7, the number of iterations is more sensitive to changes in the α value. As I will explain later, this function being one of the functions that best represents the functions calculated in WIEN2K. As the final idea of the WIEN2K algorithm enhancement project (of which my internship is a part) is to find α values where it converts quickly, this function was useful to look for clues between two rapid convergences but one of which was double the other in number of iteration.

3.1.2 Eigenvalues/Eigenvectors of H_n

After running the algorithm by varying the α value, we get different convergence value depending on the α value. But for specific convergence, for example the shortest and longest convergence over a given interval, the idea will be to look at the eigenvalues and eigenvectors of the different matrices making up the algorithm in order to obtain useful information.

Dispersion of eigenvalues in $\mathbb{C}$

The first information that was extracted from the H matrix's eigenvalues was the number of those eigenvalues that were negative and the number of those eigenvalues that were complex.

After several tests on the functions mentioned above, it turned out that the presence of a negative value in the eigenvalues of H_n prevented the rapid convergence of the algorithm. This result was not discovered immediately because, as can be seen in the figure 9, convergence seems more in relation to the number of complex eigenvalues.

In order to validate the information extracted from the various tests, it must first be ensured that it is reliable. A good example is the Rosenbrock function. As can be seen from the figure 10, we can first think that this function shows a very strong relationship between the number of complex eigenvalues and convergence. If this is due to a particular aspect of function, then causality would lose all its meaning. But Rosenbrock's function, because of its hilly appearance, makes its roots also minimal.

Ratio of complexes eigenvalues

The complex eigenvalues having the appearance of influencing the convergence of the algorithm, it was necessary to succeed in extracting more information from these eigenvalues than only their numbers. In order to see whether complex values influenced convergence, it was necessary to measure the ratio between complex part and real part of the eigenvalues whose eigenvectors most influenced the next step. I have therefore made a sum of the ratio of the eigenvalues weighted by the scalar product between associated eigenvectors and the

value of the function in this iteration. In others words :

$$Sh_n = \sum_{i=1}^{N} |\frac{\Re(\lambda_n^i)}{\Im(\lambda_n^i)}| \cdot < \varepsilon_n^i | F(x_n) > \tag{3.1}$$

where N is the number of eigenvalues, n the n-th iteration, $\Re(\lambda_n^i)$ and $\Im(\lambda_n^i)$ real and imaginary part of the i-th eigenvalues at the n-th iteration, ε_n^i eigenvector associated with the λ_n^i eigenvalues and $< \cdot | \cdot >$ is the usual dot product. The idea was therefore to plot each S_n according to the iterations. A figure is then obtained as figure 11. In blue, $ln(Sh_n)$ with $Sh_n < 1$,in cyan $10 * ln(Sh_n)$ when $Sh_n > 1$. On this figure, a possible correlation appears, showing that there would be convergence when the complex share lost importance over the next steps. Unfortunately, as shown in figure 12, this correlation was not verified as a counter-example was found.

3.1.3 Study of eigenvalues/eigenvectors of the other matrices of the algorithm

The reasons why it is the H_n matrix whose eigenvalues have been studied are on the one hand that it is the approximation of the inverse of the Jacobian and on the other because it has as many values as variables of the function, which makes it possible to have a lot of usable data. As I finish my internship, this line of research could not be exploited but the main idea was to succeed in extracting information from the Y_n and S_n matrices making up the H matrix because obtaining their impact on the convergence of the algorithm would eventually give information about the value that must take α.

3.2 Conclusion of the results and future prospects

The period of the internship being short and the advances in the field of research take time, so I could not make a revolutionary discovery but I allowed a breakthrough which is already a very good thing.

3.2.1 Contribution of the internship

My contribution to the improvement of the algorithm

In the end, it turned out that the eigenvalues of H_n which were negative negatively influence the convergence of the algorithm and that if the eigenvalues have a complex part it will positively influence the convergence. Laurence Marks did some testing on WIEN2K software that confirmed this. Thus, I was able to get at least one useful result that will be implemented in the next version of WIEN2K.

Experience gained

This internship provided me with a lot of enriching and useful experience for my future professional experiences. First of all, I had a vision of the world of research that will be a plus if I am asked to do a thesis in the future. Second, the autonomy I had and the fact that my initiatives were appreciated made me feel ready for the world of work. Finally, working with an American researcher allowed me to improve my English, now working in foreign lands is an option that I am reconsidering.

3.2.2 Trust-region control

The research to improve the Wien2k software is far from complete, there is still much work to be done. The next line of research is to tighten the trust-region control. This echoes a line of research that I did not develop during my internship, research on basins attractors[6][7]. This line of research highlights a problem present in the fixed point theory: starting from an initial point sufficiently close to the solution. But without information on the solution, it is difficult to know whether the chosen point is sufficiently close to the solution. In the case of WIEN2K, the idea is to use chemical postulates to ensure that the initial point is sufficiently close to the solution. But this is not always enough, moreover, it may be that the initial point is on a border between two basins of solution attractors, generating instability and oscillations preventing convergence.

Conclusion

It's hard not to get lost in research. There is still much to discover and making advances is an arduous task. Despite this, knowing that you have contributed to the advancement of research is a very enjoyable experience.

Bibliography

[1] L. D. Marks and D. R. Luke. Robust Mixing for Ab-Initio Quantum Mechanical Calculations. *Physical Review B*, 78(7), August 2008. arXiv: 0801.3098.

[2] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–577, January 1965.

[3] M. Bierlaire and F. Crittin. A generalization of secant methods for solving nonlinear systems of equations. *3rd Swiss Transport Research Conference, Monte-Verita, Ascona (Switzerland)*, 2004. P 2003.04.

[4] L. D. Marks. Fixed-Point Optimization of Atoms and Density in DFT. *Journal of Chemical Theory and Computation*, 9(6):2786–2800, June 2013.

[5] Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom. Testing Unconstrained Optimization Software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[6] Melvin Scott, Beny Neta, and Changbum Chun. Basin attractors for various methods. *Applied Mathematics and Computation*, 218(6):2584–2599, November 2011.

[7] Rajni Sharma and Ashu Bahl. A sixth order transformation method for finding multiple roots of nonlinear equations and basin attractors for various methods. *Applied Mathematics and Computation*, 269:105–117, October 2015.

Appendix

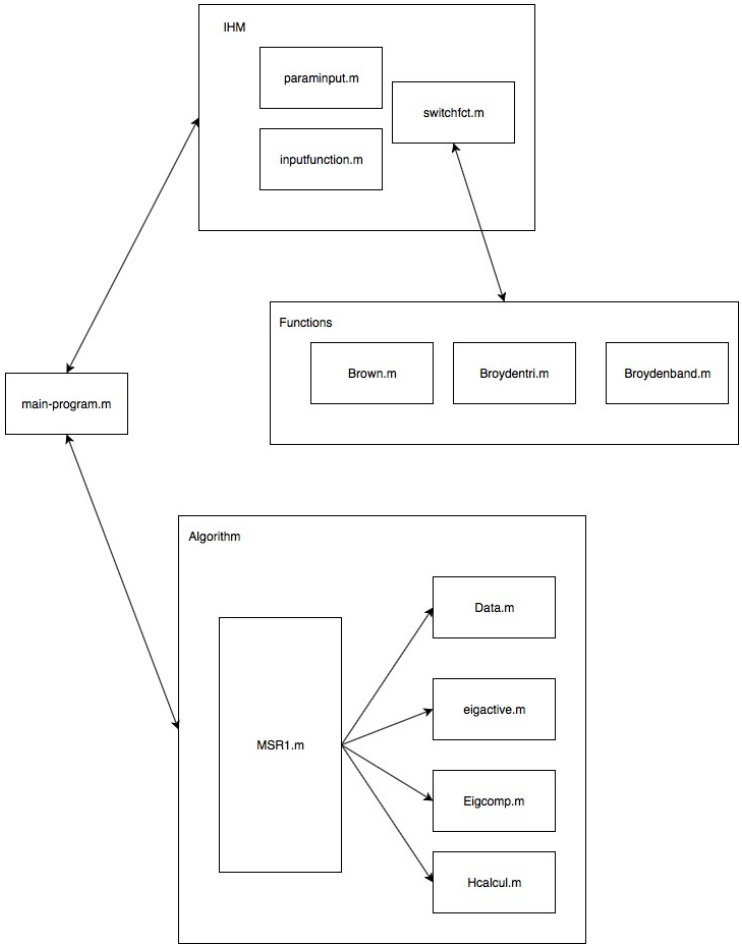

Figure 1: Diagram of MATLAB project

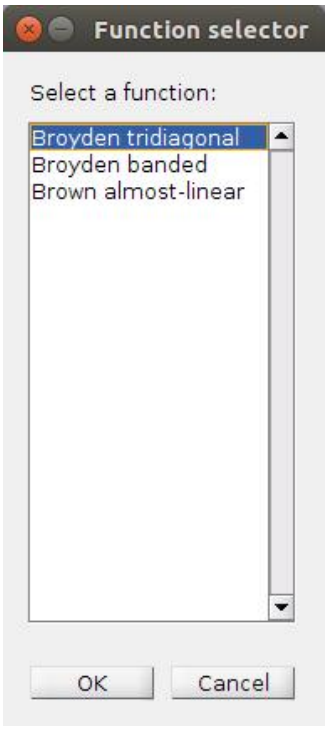

Figure 2: Dialog Box "Function Selector"

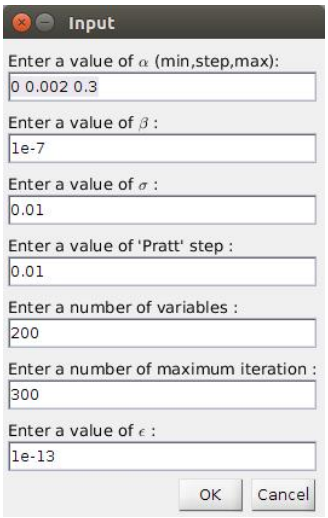

Figure 3: Dialog Box "Input parameters"

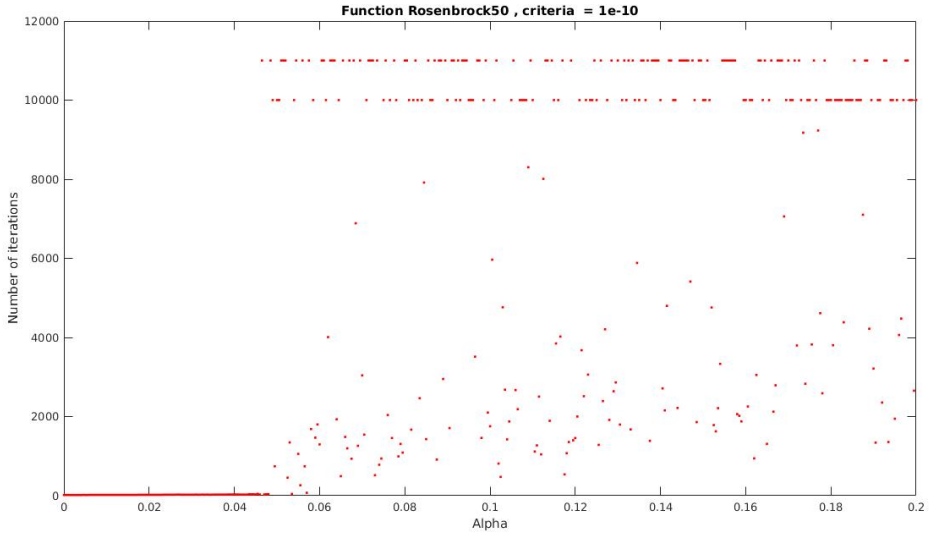

Figure 4: Iterations based on α with $\alpha \in [0, 0.2]$

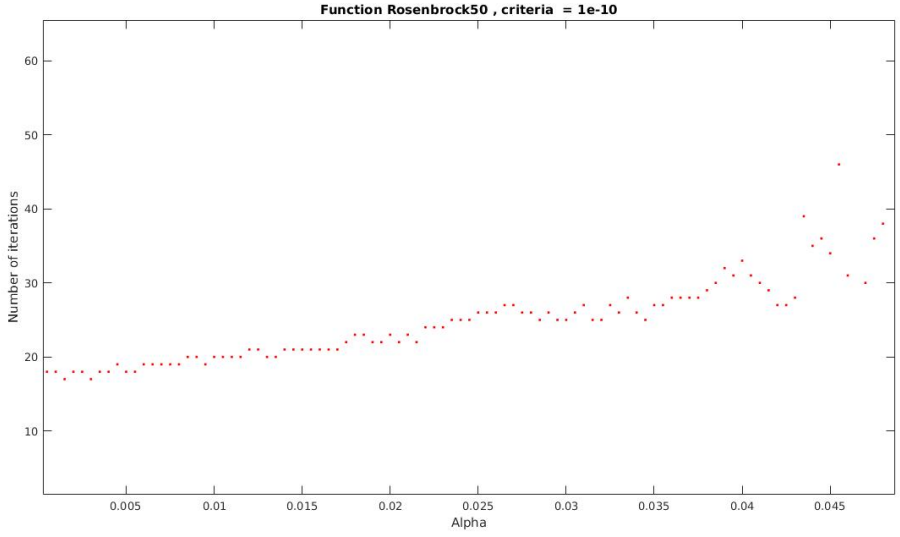

Figure 5: Zoom of figure (4) on $\alpha \in [0, 0.05]$

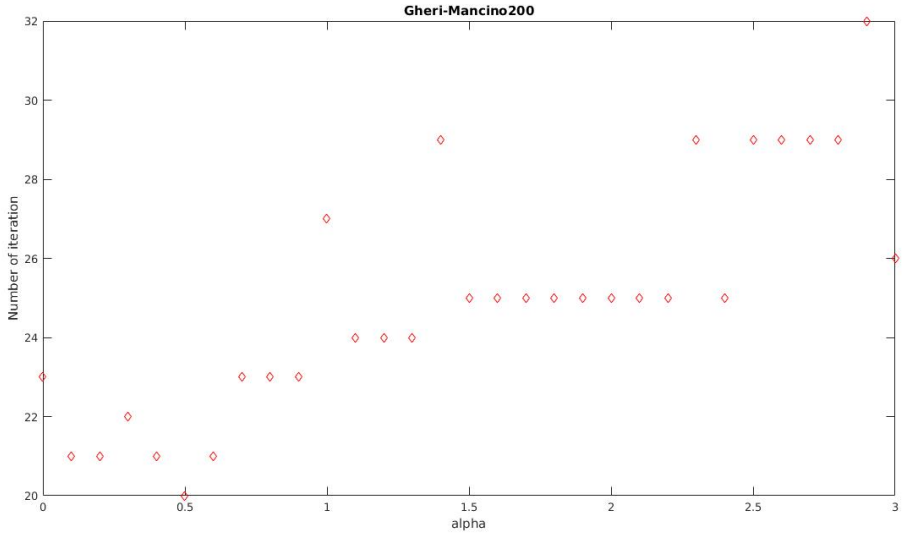

Figure 6: Iterations based on α with $\alpha \in [0, 3]$

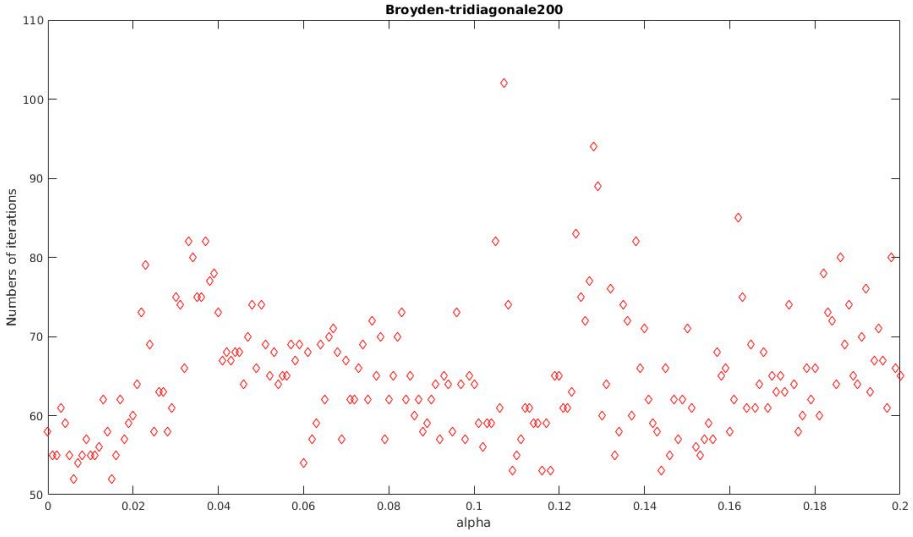

Figure 7: Iterations based on α with $\alpha \in [0, 0.2]$

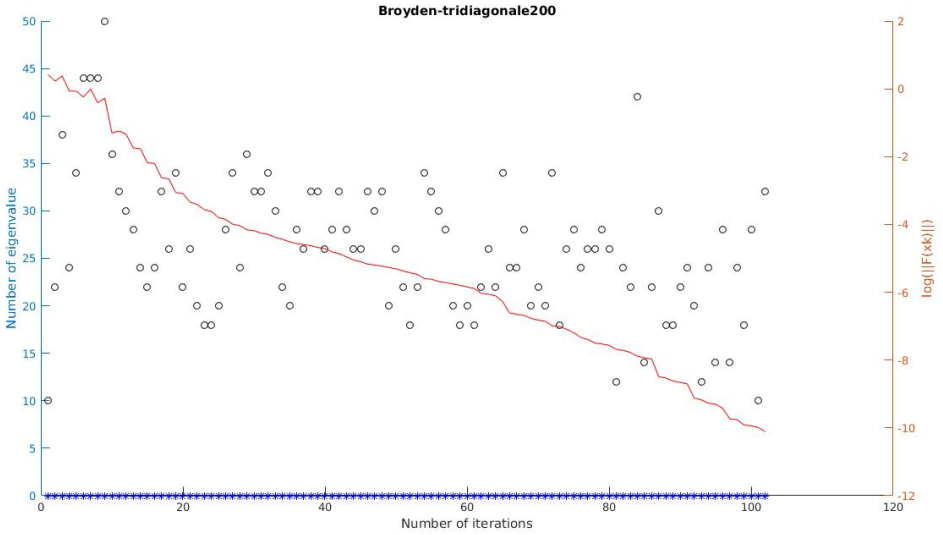

Figure 8: The slowest convergence of MSR1 $\alpha \in [0, 0.2]$

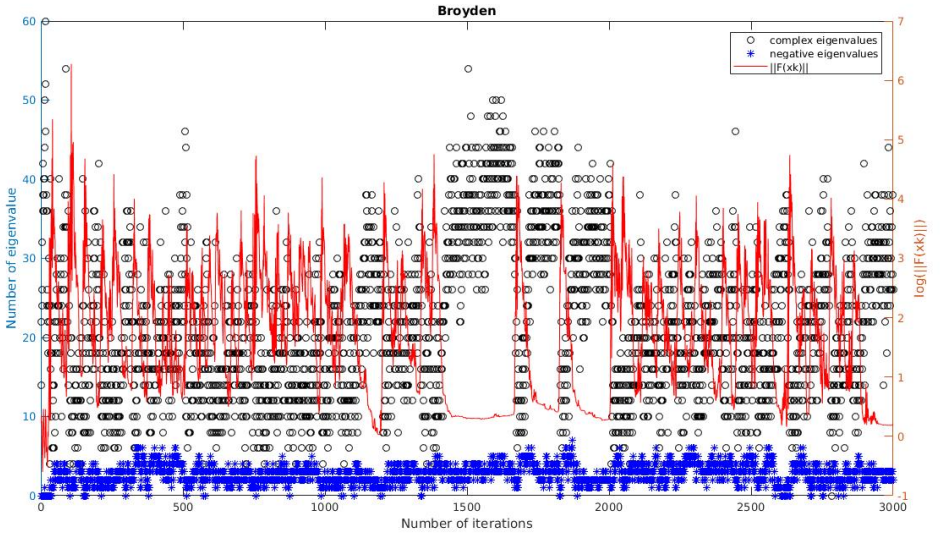

Figure 9: Oscillation of MSR1

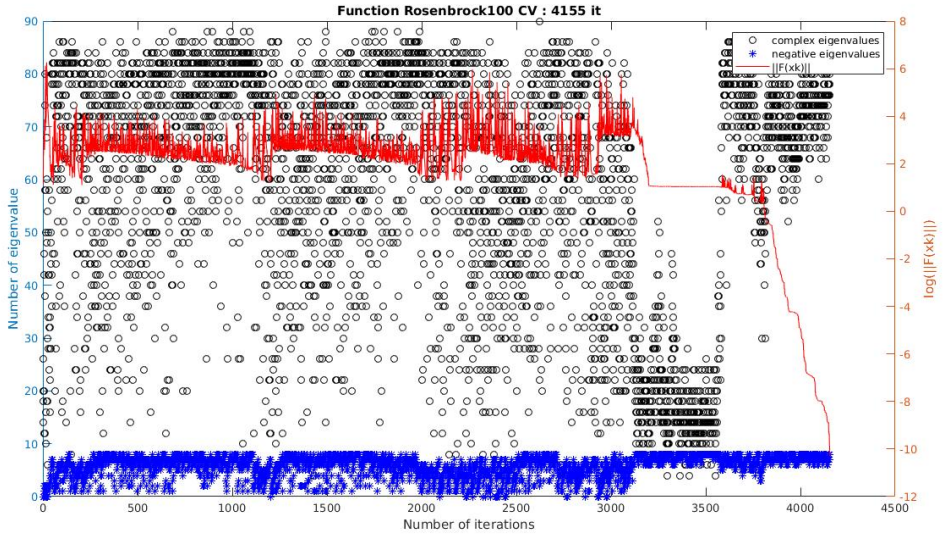

Figure 10: Example of misleading data

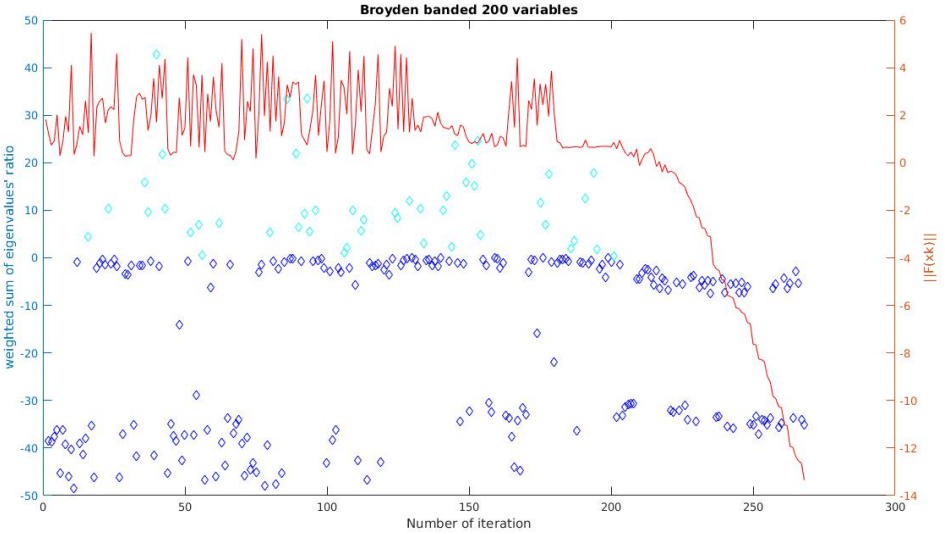

Figure 11: Result showing possible correlation

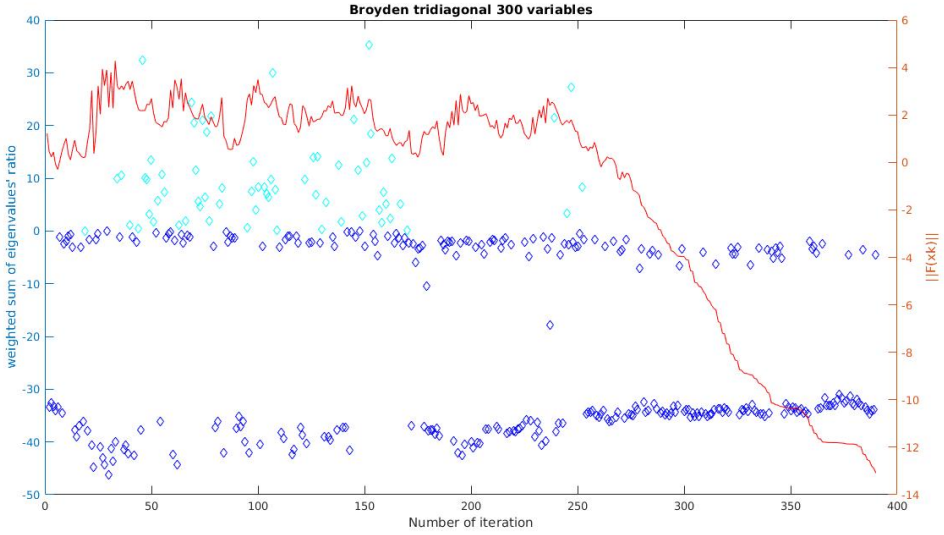

Figure 12: Counter example of a possible correlation