Report Week 3

Matthijs Neutelings, Dirren van Vlijmen, Olivier Brahma

February 10, 2022

1 Exercise 1

Firstly, before the start of the real report we will answer exercise 1 of the exercises for this week here.

1.1 1a

If we demand x to be real valued and $||x||^2 = 1$ we find x to be 1 dimensional. Since:

$$||x||^2 = \sum_i x_i^2 = 1 \tag{1}$$

Which can only be true if x has only one value equal to ± 1. Since this would lead to a 1-1 matrix of weights, and we demand the diagonal of the weights to be zero. We have that w is zero. Which immediately leads to the energy being zero. As a consequence the computational complexity to find the solution is zero, or constant.

1.2 1b

To find the minimum of the energy in this case we will just have to look at the form of the energy:

$$E(x) = -\frac{1}{2}x^\intercal wx = -\frac{1}{2}\sum_{i,j} w_{ij}x_i x_j \tag{2}$$

Where we demand that $w_{ij} > 0$ for all i and j. The most natural way to get the lowest energy is then to have every term within this sum to be positive, due to the negative sign in front. Given that we know that $w_{ij} > 0$ we get that we need to have that $x_i x_j > 0$ for all i and j. Which can be created with two possibilities. Either $x_i = x_j = 1$ for all i and j, or $x_i = x_j = -1$ for all i and j. And thus it is shown that the in the case of all strictly positive weights, the minimal energy is found for x = ± (1,1,...,1,1).

Where it is noted that the strict positivity is needed instead of the possibility of certain weights being zero. Since the latter would lead to the possibility that certain components of x could differ from the above pattern, due to the fact that they would not contribute to the energy level anyway. And thus creating multiple solutions with the same, and thus still, global minimum.

2 Methods of implementation

Firstly it should be noted that in order to run both of the methods (iterative and simulated annealing) for all relevant results we transformed the given "make data" Matlab file into a python file that has the same output. This file can also be found within the GitHub repository [1]. Further whenever the energy is mentioned the energy as laid out in 2 is meant.

A more important note regarding the implementations has to do with the optimization side of things. Both methods will work with the same configuration, and will thus both have to work with the calculation of the energy difference given by a spin flip on a certain site of x. The formula presented within the slides for this difference is the following:

$$\Delta E_{x,F_i x} = 2x_i \sum_{i \neq j} w_{ij}x_j \tag{3}$$

As can be seen this would require, within the implementation, a dot product between two vectors of size 500. That, for both methods, would have to be repeated multiple times. And thus this part of the implementation will require the most focus for the optimization of the to implement methods.

The implementation used make use of something called that we dubbed the energy and difference array. The energy array is an array with the same shape as the weight array, the components follow the following definition:

$$E_{ij} = w_{ij} x_i x_j \tag{4}$$

The difference array has the same shape as x. And is defined by the following relation:

$$D_i = 2 \cdot \sum_j E_{ij} \tag{5}$$

These definitions make it that the the following relations hold:

$$E(x) = -0.5 \cdot \sum_{i,j} E_{ij} = -0.25 \cdot D_i \tag{6}$$

$$\Delta E(x \to F_i x) = D_i \tag{7}$$

. Thus, practically, reducing the computation of the difference in energy to a constant look-up. And the calculation of the full energy to a sum of 500 components. But of course after performing the flip both arrays will need to be updated, which will reduce. This is the reason the energy array is used, since the update of the difference array depends on the energy array. After a flip of the i^{th} spin, the following updates are performed:

$$D_{i,new} = -1 * D_{i,old} \tag{8}$$

$$D_{new} = D_{old} - 4 \cdot E_{:,i,old} \tag{9}$$

$$E_{i,:,new} = -1 * E_{i,:,old} \tag{10}$$

$$E_{:,i,new} = -1 * E_{:,i,old} \tag{11}$$

Here $E_{:,i,new}$ denotes the i^{th} column of the energy array, and $E_{i,:,old}$ the i^{th} row of the energy array. Further it should be noted that the update of the energy array is performed before the update of the difference array within the used implementation. Therefore the sign of the second update for the difference array with eq 11 has a differing sign. An important note regarding these updates is that for these updates to work it is always assumed that $E_{ii} = 0$ for all i. This is trivially true due to $w_{ii} = 0$ for all i being a demand for the weight array. Thus whenever within the below method explanations, a calculation of the energy or a calculation of the difference in energy is mentioned the above explanation of true implementation should be taken into mind. Further, again for both implementations, outside for performing the first calculation of energy, the actual state through calculation is not kept in memory.

2.1 Iterative Method Implementation

The implementation starts with a randomly determined initial state of x. For this initial state the energy is calculated. Of note is that due to the restarts of this method a multiple of this random initial states, and thus creation of the energy and difference array. The latter makes these restarts slower than needed. Therefore the random states used are random rolls of the initial random state. Thus still creating new random states, only less initializing is needed. Then, through a predefined random list of sites, a spin flip is performed on the current site from the random list. The energy difference through this spin flip is then calculated for each of these flips. If this difference is lower then zero. The flipped state is accepted as the new state and the energy and difference array are updated accordingly. Otherwise no update happens and the next random spin flip is tried.

As requested within the assignment this procedure is performed K times. From these K restarts the lowest found energy level is returned. Further $N_{\text{runs}} = 20$ runs are performed of the whole procedure including the K restarts. Due to the latter choice, error bars emerge concerning the minimum energy found by the procedure.

The "convergence" of the implementation is not based on a criterion. There are just L random spin flips performed, note that this does not mean that L states are used. It might be that the spin flip will not result in a negative energy difference and thus will not be performed. The value of L is chosen basically in retrospect. For the two frustrated problems the found values for the w500 problem are compared, per K, with the values presented within the assignment. L = 17500 was found. For the ferromagnetic problem the L was varied and plots of the energy level were made to determine after how many iterations the energy level did no longer change. L = 5000 was found This method of convergence was chosen for two reasons. Firstly the desired values were there for the w500 problem, making it possible for the frustrated problems. And the ferromagnetic problem should theoretically be faster, thus the method should make it possible to choose the hyperparameter[1]. Secondly, it was felt that for the iterative method this was the most natural manner of "convergence". Since it also suggested in this manner, in for example the slides. An alternative could be to for example stop the algorithm after a certain number of rejections of the proposed state, but this would also require the hyperparameter that sets this amount and thus not making it more beneficial. And thus not deleting

[1] To see one of the plots use to determine the hyper parameter, see figure 1

our choice. Another alternative would be to go over the whole neighbourhood and determine that no more beneficial updates are possible. One could also set an threshold for the amount of beneficial updates. But this would require either an extra loop over the difference array, which would lead to higher run times. Thus with practicality and optimizing in mind it was chosen to go with L proposed spin flips.

2.2 Simulated Annealing

The implementation for the Simulated Annealing method is mostly equal to the implementation that is laid out within the slides. Naturally, with the implementation within the slides, the practical version of the two is meant. This means that the implementation starts with a determination of the initial β value. The implementation uses a very slightly changed version of the one proposed within the slides. Within our implementation all possible flips, and the corresponding energy differences are checked. The initial β then becomes:

$$\beta_0 = \frac{1}{\max(\Delta E)} = \frac{1}{\max(D)} \tag{12}$$

Then a random initial state is created, as before the energy is calculated for this initial state. Then trough a predefined list of sites, a proposed state is generated through a spin flip on that site. The proposed state goes through a Metropolis Hastings (MH) acception/rejection step. Where the a value is calculated by for a spin flip of the i^{th} spin:

$$a = e^{-\beta \Delta E_{x,F_i x}} = e^{-\beta D_i} \tag{13}$$

If the proposed state is accepted through MH the updates from eq 2 are performed. Otherwise these arrays remain unchanged. Of note is that for this algorithm all energy values need to be known over the full chain of MH steps, irrespective if the state has changed. Thus an array that keeps track of the energy levels is used, This is needed since after L amount of proposed states, the standard deviation of these energies is checked. If it is zero the process is stopped and the energy of the last state is presented as a possible solution. Otherwise the value of β is changed via a determined schedule. Two options are implemented. Firstly the schedule proposed by Aarts and Koorts (AK), which follows:

$$\beta_{k+1} = \beta_k + \frac{\Delta\beta}{\sqrt{\mathbb{V}_k E}} \tag{14}$$

Where $\mathbb{V}_k E$ is the standard deviation produced by the MH algorithm within the k^{th} iteration and $\Delta\beta$ is a hyperparameter of the algorithm. The other, exponential schedule follows:

$$\beta_{k+1} = f \cdot \beta_k \tag{15}$$

Where f is a hyperparameter of the algorithm. The procedure is then repeated until the standard deviation produced by the MH procedure is zero. Basically denoting that the algorithm cannot find more energy minimising states. A last note regarding the implementation is that, in contrast to the iterative method, no restarts are used. The $N_{\mathrm{runs}} = 20$ repeats of the whole algorithm are still used, and as such the error bars concerning the minimum energy are still present.
For this algorithm the method of convergence is very strictly defined within the suggestion of practical implementation within the slides. And therefore this method of convergence is also used. Possible alternatives would have been to use a threshold for the variation, instead of the full on zero value. And then instead of the last energy value encountered, return the lowest energy value encountered. Another way would be to limit the value that β can rise to. A natural combination with this would be to use bigger steps for updating β. In this way it would more resemble a grid search for a sort of optimal β value for a normal MH chain. In which the minimal energy found would just be found by running at the found β value and checking for the lowest value. Ultimately the suggested methods of convergence could lead to faster running times. But given the quite specific mentioning, and the higher accuracy of, the used method. It was felt to be the better choice.

3 Results & Findings

Within this section the results and the discussion or possible findings regarding this results will be presented. Firstly the results of the iterative method will be compared and discussed relatively to each other. Secondly the results of the Simulated Annealing method will be discussed. And lastly the results of the two methods will be compared to each other. Of course this will mainly be done with regards to the W500 weight array that was given within the assignment. Of note is that all these results, and possibly more, can be found within the GitHub repository [1].

3.1 Iterative Method

Below the results will be showed with regard to the iterative method. Firstly the tables that use the same format as the table presented within the assignment are shown.

K	CPU (sec)	E
20	0.2	-6269 +- 50
100	1.2	-6333 +- 38
200	2.5	-6354 +- 35
500	6.3	-6375 +- 35
1000	12.6	-6404 +- 38
2000	25.2	-6413 +- 29
4000	50.3	-6425 +- 25

Table 1: The results for the W500 problem. The first column denotes the amount of restarts used within the algorithm. The second column the minimal time an iteration takes in seconds. And the last column denotes the mean minimal energy found by the iterative method over 20 runs of the algorithm, together with the standard deviation concerning these 20 runs.

For the used implementation of the algorithm the lowest value found on any run was -6476. The are a few things of note regarding these results and the comparison with the results presented within the assignment of Bert Kappen. Firstly there seems to be a factor of about 2.5 faster CPU times. Of course there could be several reasons for this. A difference within CPU could be possible as the CPU used for the results within the assignment is not mentioned. Further a difference within platform could lead to differences. Also the possibility of difference in memory quickness, especially given the used implementation of this report, could lead to different CPU times. Overall it is felt that the difference is possibly too big for these reasons to be the full cause. But the performance measured for the implementation of the report is optimizes a lot with respect to the first implementation that was thought of. About a factor of 10 in CPU time was acquired, from the first to this implementation.
Another note would be that the quality of the found results is quite similar, especially the means found, thus it is assumed to be okay to really compare CPU differences as is.
A note should be made regarding a slight difference in found standard deviations. The implementation of the report seems to have a bit more stable and lower standard deviations. The difference is small enough that it might be caused by the choice to note fully re-sample the initial state, and thus the energy and difference array as well, but perform a roll on it. This is also noticeable in the difference in ultimately found lowest energy level overall. Our "pseudo-random" initial state possibly makes it that not the full search space is explored.
Overall this also leads to a slightly different conclusion concerning which K is needed for reproducible results. The report would suggest that this would be around K = 2000, given the clearly lowering standard deviation, but the assignment seems to point at K = 4000, with the same argumentation. Naturally the conclusion would be that it will be somewhere between the two, more likely a bit closer to the latter. The reasoning behind using the standard deviation for this, is that when this goes to a more low point more runs retrieve the same or similar results. In other words the result becomes more reproducible.

K	CPU (sec)	E
20	0.3	-7952 +- 80
100	1.3	-8040 +- 46
200	2.5	-8068 +- 30
500	6.4	-8106 +- 56
1000	12.8	-8121 +- 46
2000	25.7	-8155 +- 31
4000	51.9	-8172 +- 26

Table 2: The results for the frustrated problem created by the translated "make data" program. The first column denotes the amount of restarts used within the algorithm. The second column the minimal time an iteration takes in seconds. And the last column denotes the mean minimal energy found by the iterative method over 20 runs of the algorithm, together with the standard deviation concerning these 20 runs.

K	CPU (sec)	E
20	0.1	-62430 +- 0
100	0.5	-62430 +- 0
200	1	-62430 +- 0
500	2.4	-62430 +- 0
1000	4.9	-62430 +- 0
2000	9.7	-62430 +- 0
4000	19.4	-62430 +- 0

Table 3: The results for the ferromagnetic problem created by the translated "make data" program. The first column denotes the amount of restarts used within the algorithm. The second column the minimal time an iteration takes in seconds. And the last column denotes the mean minimal energy found by the iterative method over 20 runs of the algorithm, together with the standard deviation concerning these 20 runs.

Regarding these tables there are also some notes to make. Firstly, table 2 has very similar running times and pattern within standard deviation as table 1. The running times is not surprising, given the same shape of the weight array as well as the same amount of iterations used. But the fact that the standard deviation follows the same pattern shows that again at around 2000 to 4000 restarts the results found start to become closer to reproducible. The next note should go to table 3. We here clearly see the workings of a ferromagnetic problem compared to a frustrated one. The minimum of the energy is clearly uniquely low, since no standard deviation is found, and equal to 0.5 times the non-zero amount of weights. As indicated by the prove given for exercises 1b. Due to the workings of the "make data" program the amount of weights that are non zero are (close to) half the amount of weights. Further this minimum is easily enough reached that a low amount of restarts is required to at least hit it once. This is of no surprise, since the form of the solution can be reached if all of the spins that have the same orientation as the first spin flipped are flipped. Since such a flip will always bring the state closer to the wanted solution. Namely one with all orientations the same. Thus having 5000 randomly proposed spin flips, should be enough to reach all of them. Especially if it is tried multiple times.

A last note regarding the results of the iterative method is made with regards to figures 2 and 3 within the Appendix. These figures seem to show that the chosen value of L might be far too high, since the energy level for both of them does not really update anymore after $\pm$ 10000 iterations. But it was found that the results did still differ when changing L to a lower value. This points at the fact that some restarts do indeed use the full 17500 iterations in order to reach their lowest value, and thus lowering the L value would result in a lesser quality of the ultimate results.

3.2 Simulated Annealing

Within this section the results of the Simulated Annealing will be discussed. For this algorithm only the w500 problem has been tested. Firstly the tables using the same form as the table within the assignment are presented.

$\Delta\beta$	L	CPU (sec)	E
0.1	500	0.3	-6429 +- 71.0
0.01	500	3.2	-6558 +- 35.0
0.001	500	30.5	-6582 +- 34.0
0.001	1000	76.9	-6589 +- 29.0

Table 4: Table of quality of the solution versus the CPU time with Aarts and Koorts updating schedule. First column denotes the value of $\Delta\beta$ used. Second column denotes the length of the chains used. Third column denotes the minimal CPU time per iteration of the whole algorithm in seconds. The last column denotes the mean energy level found over 20 runs, and the standard deviation of those 20 runs.

f	L	CPU (sec)	E
1.01	500	0.9	-6509 +- 57.0
1.001	500	8.3	-6558 +- 34.0
1.0002	500	42.5	-6582 +- 28.0
1.0002	1000	90.4	-6581 +- 32.0

Table 5: Table of quality of the solution versus the CPU time with exponential updating schedule. First column denotes the value of f used. Second column denotes the length of the chains used. Third column denotes the minimal CPU time per iteration of the whole algorithm in seconds. The last column denotes the mean energy level found over 20 runs, and the standard deviation of those 20 runs.

The lowest reached value for either of these two algorithms over any run is -6624, the same value as presented within the assignments. The first thing to note is that the found quality of the results are very similar to that of the results presented within the assignment. This includes the standard deviations found. And is true for both the

updating schedules. Compared to the results within the assignment the difference in CPU time is most clear. The difference factor seems to not really be constant over the different values of hyperparameters, but overall seems to be bigger than the factor of difference that was found for the iterative method. The most logical reasoning for this would be that the implementation by Bert Kappen used a form of optimization that was not discovered by us. It could also be (partially) due to not copying the mechanism of (pseudo) random states as used within the iterative method. This was done since this method does not make use of restarts, and thus the possible optimization made by it would not be big enough to warrant the possible side effect of less exploration of the full search space. As explained within the previous subsection. Of course the mentioned arguments for slower run times within the previous subsection also hold for these results. As well as the fact that the optimization of the run time performed for this method already had very significant effects when compared to the naive first implementation.
When comparing the two updating schedules against each other the found quality of the results seems very similar. The only difference seems to be that if one decides to use fewer total chains, and thus the first hyper parameter values the exponential schedule seems to be more reliable. This is seen within the standard deviation being lower, as well as the mean energy level already being more close to the values found by more extreme values of the hyperparameter. This might be due to the fact that the exponential schedule naturally uses more chains, since the value updates with a set step, where the AK schedule updates with steps determined by the previous chain. But only in this case would the exponential schedule be the better choice of schedule. This is due to the longer running times for the exponential schedule. The difference in running times is not surprising since, as can be seen within figures 4 and 5, the amount of iterations is higher for the exponential schedule. The biggest reason for this seems to be at the later iterations. The exponential schedule goes on with the same idea of steps the whole method. Where the AK schedule starts to take much bigger steps at the latter stages due to the lowering standard deviation of the energies. Basically saying that the found direction seems to be the correct direction for the solution and thus speed up into that direction by limiting the amount of possible steps in energy adverse directions with a higher β. This behaviour is also seen with the plots on the left side of the aforementioned figures, where the mean energy for the AK schedule seems to go quite directly to its goal, the mean evolution for the exponential schedule more resembles a s-shape.

3.3 Iterative vs Simulated Annealing

For this section only the results with regard to the w500 problem are taken into consideration. When comparing the two algorithms it is quite clear that the Simulated Annealing (SA) method presents much better qualities of the eventual solution, even if using less extreme values for the hyperparameter. Thus the possible downside of slower running times for the SA method is countered, since if the running time was of critical importance the hyperparameter could be set at these less extreme values. The fact that the SA method returns better results most likely has to do with the partial evasion of local minimums by this algorithm due to the MH step that is implemented. This step makes it possible to go against the flow of the energy, which enables the possibility of jumping over an energy barrier that blocked going to lower energy levels. This possibility is higher at the beginning of exploration, due to the lower β at that point. Resulting in the state being able to explore a very big part of the search space. Whereas the iterative method does not have such a step. And will always go to the local minimum associated with the initial state. This is partially countered by using restarts, starting from new states. But due to the nature of frustrated problems, with a lot of different local minimums, the actual global minimum might never be found. The only upside of the iterative method compared to the SA method seems to be the fact that less hyper parameters have to be used. Making the implementation more straight forward and less sensitive to possibly subjective choices for these hyper parameters.

4 Summary

This report explained the implementations used for the Simulated Annealing (SA) and iterative minimization methods for the energy as depicted within eq 2. It was found that the implementation of the iterative method was somewhat slower than the results presented within the assignment, with possible, partial reasons being the hardware. But good optimization had already taken place. Further it was found that ferromagnetic problems are much easier for this method, due to nature of these problems resulting in a clear global minimum. The amount of needed restarts for reproducible results was between 2000 and 4000. For the SA method similar reasoning concerning the CPU times stands. Overall the results found that the best schedule to use for good results would be the AK schedule, if running times was to be very crucial one might opt for the exponential schedule due to its better results for the less extreme hyper parameters. If the choice were between the two methods, SA is the clear winner most likely due to its possibility of evading local minimums with its MH stepping procedure. Even considering the fewer hyper parameters needed for the iterative method.

5 Appendix: Figures used within the report

All plots for the different hyperparameter values can be found within the GitHub repository [1]

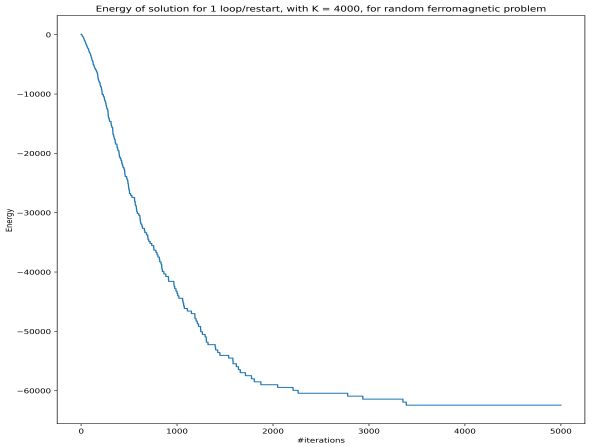

Figure 1: Evolution of the energy level found for the last restart of 4000 of the ferromagnetic problem

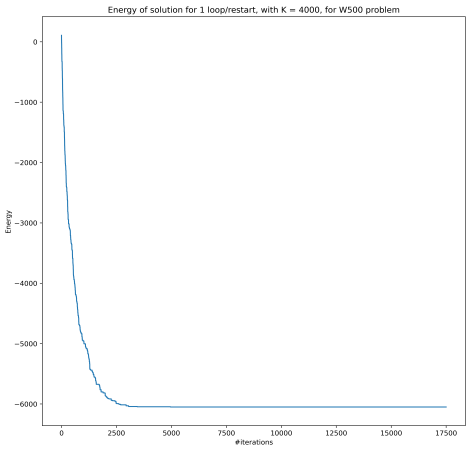

Figure 2: Evolution of the energy level found for the last restart of 4000 of the W500 problem

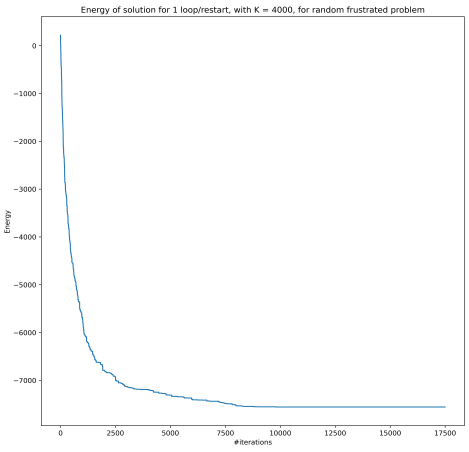

Figure 3: Evolution of the energy level found for the last restart of 4000 of the random frustrated problem

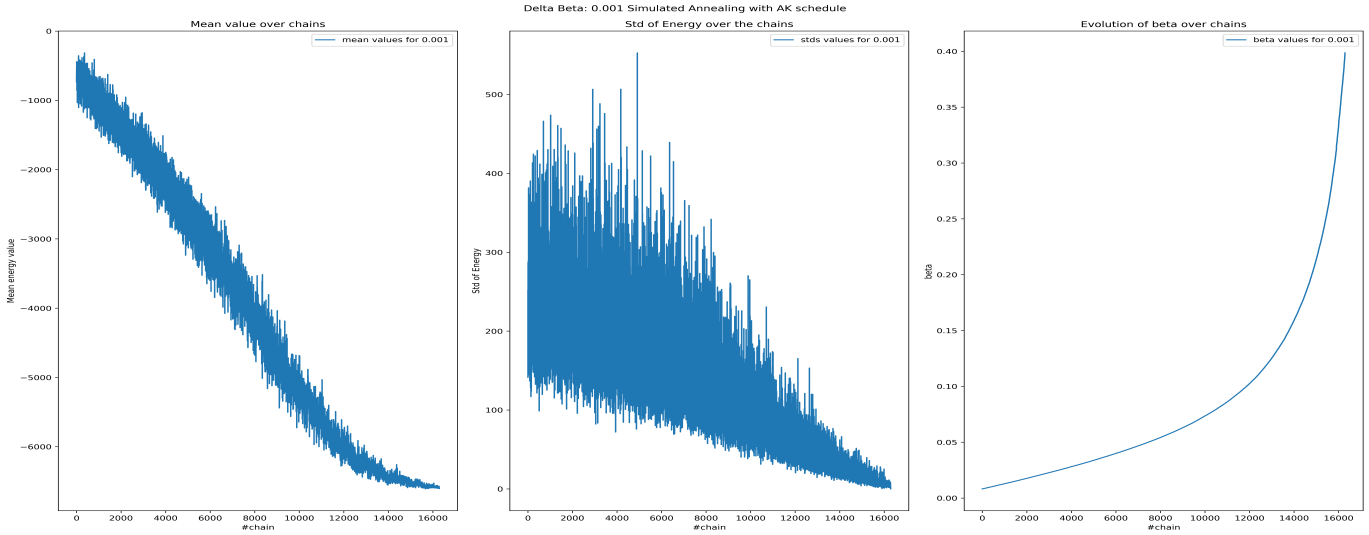

Figure 4: $\Delta\beta = 0.001$; L = 1000 Left: The mean value of the energy found per chain used. Middle: The standard deviation of the energies found per chain. Right: The evolution of β over the chains used.

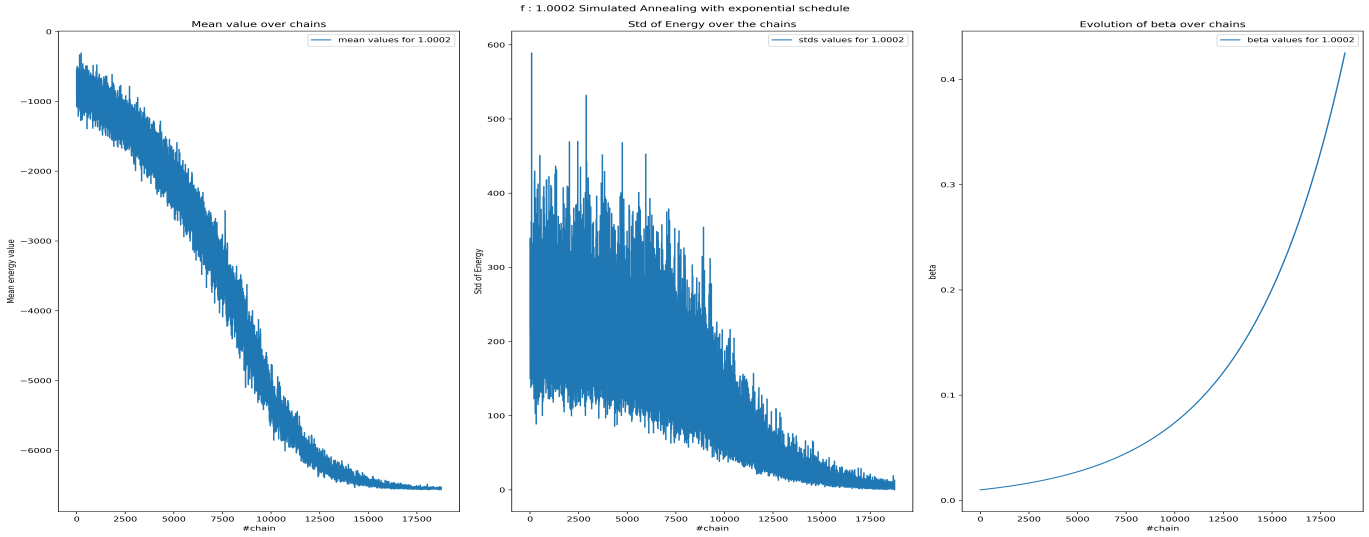

Figure 5: f = 1.0002; L = 1000 Left: The mean value of the energy found per chain used. Middle: The standard deviation of the energies found per chain. Right: The evolution of β over the chains used.

References

[1]M. Neutelings, D. van Vlijmen, and O. Brahma, *Advanced-machine-learning: week 3*, https://github.com/Dirrenvv99/Advanced-Machine-Learning/tree/main/Week%203.