NTNU
Norwegian University of
Science and Technology

Magnetic waves and phase transitions

Exam TFY4235 - Computational Physics

Henrik Friis

May, 2022

Disclaimer

During the exam, I have discussed the task and possible solutions with all the other MTNANO students, Aurora Teien, Eivind Heggelund, Erlend Johansen, Eskil Vik, Henrik Tidemann Kaarbø, Ludvig Lous and Viljar Johan Femoen. Our solution will probably have the same approach, but all code is written individually. During this exam, I have learned a lot about spin and magnetization which I did not know beforehand, so please bear over with my limited theoretical background within this field.

Regarding Assignment 2, I have not discussed it with anyone. Since I wrote the report from Assignment 2 before the 6-page limit and the scientific report-format was abandoned, I have confined myself to around 6 pages, thus not leaving very much room for explaining the code. The source code should, however, be very well documented and readable, should you need it.

1 Introduction

The code is written in Python 3.9.7, using Numpy [1] for array manipulation, Matplotlib [2] for plotting and visualizations, Numba [3] for JIT-compilation and Scipy [6] for curve-fitting. By making sure that as much code as possible uses arrays and is compiled using Numba, the execution time for the code will be manageable, even with many for-loops, which Numba fortunately excel at. All the general equations used are in the file `equations.py`, the functions related to the solution of the ODE is in `solve_ode.py` and the file `plotting_params.py` simply sets the parameters for plotting. Apart from that, all code necessary for task a) is in the file `task_a.py` etc. This was to easily separate the tasks and keep control of initial conditions in an exam situation. The "task_x"-files contains a lot of code, mostly for plotting, but all major functionality is in the `equations.py` or `solve_ode.py` files.

2 General implementation

I will firstly cover the most general equations, which are used many times in all the tasks. Equations 2, 3 and 5 from the exam set, [5], are implemented in the file `equations.py`. Equation 2, the LLG, is implemented exactly as in the exam set, except that the term $\mathbf{S}_j \times \mathbf{F}_j$ is stored for reuse in the end of the function for efficiency. Equation 3 is implemented as the function `F_eff()` and is deduced from taking the functional derivative of Equation 3.

$$\mathbf{F}_j^{eff} = -\frac{1}{\mu_s}\frac{\delta H}{\delta S_j} = \frac{1}{\mu_s}(\sum_{k_n} J_{jk}\mathbf{S}_k + 2d_z(\mathbf{S}_j \cdot \hat{e}_z)\cdot \hat{e}_z + \mu_s \cdot \mathbf{B}_j), \tag{1}$$

where all terms are described in the exam set, except k_n, by which I mean the summation over all the nearest neighbours of a spin $\mathbf{S}_j$. Equation 5 or 6 from the exam is implemented as the function `F_th()`. The function takes a thermal constant that is calculated once per simulation for the given initial parameters and a 3D vector from a Gaussian distribution with mean of 0 and unit standard deviation. This random vector is drawn from a larger matrix containing all the necessary vectors for a time step, in order to have the same random thermal field for each Heun iteration.

2.1 Random numbers

I have used two approaches to generate random vectors for the initial conditions in later tasks and for $\mathbf{\Gamma}(\mathbf{t})$. The first, which I consider the most correct, is calculating a uniform distribution of directions on the unit sphere. This distribution is made by inserting random numbers into spherical coordinates, which will "automatically" normalize the vector. A naive approach to this could create distributions that are heavier at the top and bottom of the sphere, but by accounting for the change of the sphere area with angles, it will be uniform. My approach follows this guide [4]. This approach will serve as initial conditions for f) and the assertion of the correct implementation of f) in 2D in task g). In order to serve as a Gaussian distribution, it is simply to multiply the direction with a gaussian distributed number with 0 mean and unit variance from `np.random.normal()`, corresponding to ρ in spherical coordinates. This approach is called `make_3D_gaussian()` and is found in `equations.py`.

Table 1: Shows units and typical values for constants

	Value	**Units of J**
J	1 meV	1J
d_z	0.1 meV	0.1J
$\mu_s B_0$	0.1 meV	0.1J
α	0.01	-
γ	$0.176\,\mathrm{T}^{-1}\,\mathrm{ps}^{-1}$	-
Δ_t	0.001 ps	-
T	0 K	-
μ_s	$0.057\,88\,\mathrm{meV\,T}^{-1}$	-
B_0	1.72 T	-
k_B	$0.086\,17\,\mathrm{meV\,T}^{-1}$	-

The second approach is to simply return a vector filled with 3 numbers from `np.random.normal()`. This approach will return vectors spanning a cube, thus resulting in a stronger thermal field and a lower T_c, as will be discussed and shown.

2.2 Units

Consistent units are used throughout the exam. The constants are initialized in each task separately. In general, the units I use are tabulated in table 1. Note that for every task, the value for the constant are defined in accordance with the task and do not necessarily equal the tabulated values.

2.3 The data structure

The datastructures used to store data is Numpy's `ndarray`. This can hold multidimensional data. My structure has 4 dimensions with the first being the timestep, the second the number of atoms in the x-direction, the third is the number of atoms in the y-direction and the last is the spin components of the atom, which is an array with 3 elements. This makes it easy to access the spin-components at all locations in the lattice at all timesteps. If non-periodic boundary conditions, non-PBC, are used, the lattice is "padded" with "atoms" with all zero spin-components. This is useful when finding the spin of the neighbouring atoms and make sure that the boundary atoms are not affected by the "opposite" lattice atoms, since vecors of zero will not contribute in the equations. When using PBC, the data is not padded and the atoms can effect each other using modulo to find the neighbours on the "opposite" side.

2.4 The Heun scheme

With the basic functionality having been covered in `equations.py`, except a few minor helper functions, it is time to implement the Heun scheme. This is done in the file `solve_ode.py`. Again I have two approaches, where one is correct and one is a little wrong. The difference between them is that the wrong approach do not take into account that the effective field also must be predicted when $\mathbf{S}_j$ is predicted. There are, however, no big qualitative difference between the results using the different approaches, but the magnetic waves propagate faster when the correct predicted field is being used. This is as expected and is supportive of the correct implementation that I have used throughout the exam. If the examiner wish to run the code, you must be aware that I have on several occasions observed the correct Heun scheme, `evolve_spin()`, terminating for no good reason using non-PBC. Should this happen, the solution is to run it again or add "_old" to the end of the function name to run with the incorrect Heun scheme. In the following I will explain the code for the correct Heun scheme, found in the beginning of `equations.py`. I will explain the one for non-periodic boundary conditions, but there is no real difference between the one with non-PBC and PBC apart from the data structure and some modulo's.

When adapting the Heun scheme, equations 7a and 7b in the exam set to the problem, I take four major steps:

1. Allocate memory to store random vectors for the timestep and store the effective field, the predicted spins and the predicted field.
2. Calculate the predicted spins using Eq. 7a
3. Calculate the predicted effective field using the predicted spins
4. Calculate the spin for the next timestep using the current spin and effective field and the predicted spin and effective field using Eq. 7b

Step 1 is performed for every timestep, while step 2, 3 and 4 are performed for every atom in the lattice at every time step. Allocating random vectors for every time step ensures that the thermal field stays the same for every timestep. As is made clear in the exam, the spins are kept normalized using `normalize()` for both the predicted spin and the end spin. The random vectors are also spatially and temporarily uncorrelated.

3 Solving the tasks

3.1 a)

This task is solved using `evolve_spins()` from `solve_ode.py` and is exactly what is described in 2.4. It is found in `task_a.py`. Non-PBC is assumed, thus the data is padded, before the tilted state is initialized. There is no damping and only the symmetry-breaking term B_0 is included. The result seen in figure 1 shows that the spin precesses around the z-axis at constant velocity. Since the magnetization is pointing up, it should spin anti-clockwise according to the right-hand rule, which it in fact does, with the x-component leading the y-component.

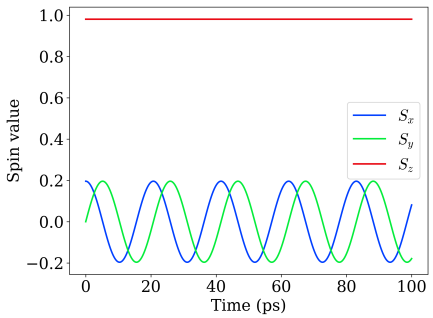

Figure 1: Shows that the spin is precessing according to the right hand rule.

3.2 b)

The initial parameters are the same as for a), with the exception that $\alpha = 0.05$. I then generated the spin evolution. Since the spin was originally tilted slightly in the x-direction, I chose to fit the S_x-data to a cosine modulated by an exponential term. I used the initial magnitude in the x-direction as the amplitude to make it easier for `curve_fit()` from `scipy.optimize` to fit ω and τ to the function. In order for `curve_fit()` to get even better results, I found $\lambda = 1/\tau$, as this proved to be an easier task, and converted it afterwards. The function I fitted the data to was `f()` in `task_b.py`. For these parameters, I found that $\tau = 66.72\,\text{ps}$ and $\omega = 0.3\,\text{rad}\,\text{ps}^{-1}$. This ω would correspond to a period of roughly 20 ps, which can also be seen in figure 3. The error when inserted in equation 8 from the exam set was 1%. In figure 3 one can see that the simulated S_x overlaps completely with the fitted S_x. Furthermore, one can see that two exponentials using the calculated τ and ω envelops the function. The S_y component would follow the same trend, with a slight shift in phase. As S_x and S_y decrease, S_z increases. In figure 2, a visualization of how the spin precesses with and without damping is provided. One can see that the damped spin spirals inward according to the right-hand rule until it points up, upon which the spin is no longer excited. When it is undamped it will keep precessing forever with the parameters defined in a).

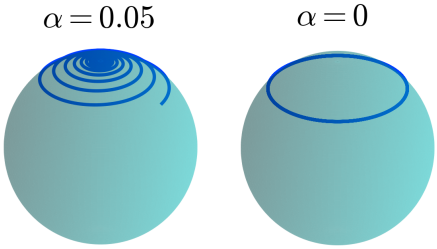

Figure 2: Shows how the spin precesses according to the right-hand rule.

3.3 c)

For this task the parameters are the same as in b), except that $J = 1$ and I have kept the anisotropy term instead of the magnetic field. The simulation was performed for 20 spins on a line. As in a and b, the spins were padded, but this time they were all initialized in the positive z-direction before the leftmost one was slightly excited by tilting it in the x-direction. The time evolution of the magnetic wave is shown in figure 4. There will be many more such figures in the tasks to come. The way to read them is by reading from top to bottom. At the top we have the initial states, while at the bottom we have the final states.

Firstly, in figure 4, one can see that the time development exhibits a dispersive nature. This is due to the Hamiltonian of the system, which incorporates that all spins influence their neighbours. This can be seen as the "blurred lines". A "main path" for the wave is visible as a Z, reflecting back from $\mathbf{S}_{20}$. Note that the colorbars have different scale, so the spins are only slightly excited at the edge. When animating the magnetic wave, it became apparent that the spins start out of

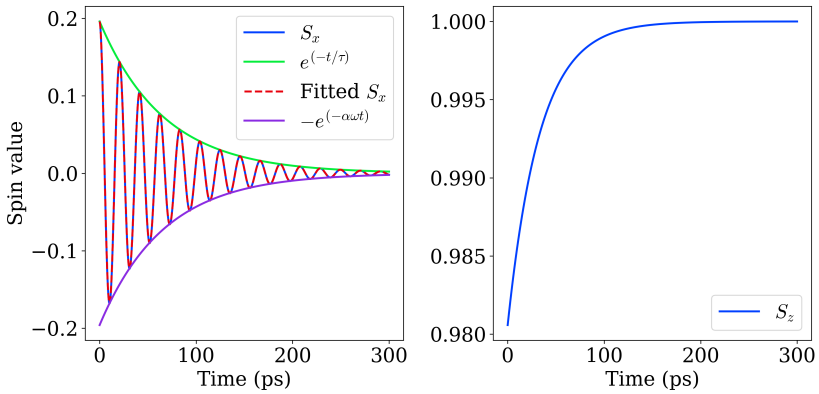

Figure 3: The simulated S_x overlaps completely with the fitted S_x. Furthermore, one can see that two exponentials using the calculated τ and ω envelops the function. The S_y component would follow the same trend, with a slight shift in phase. As S_x and S_y decrease, S_z increases.

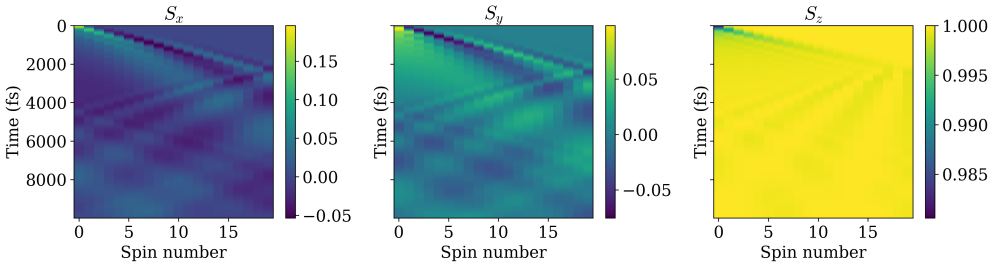

Figure 4: A visualization of $\mathbf{S}_j$(t) (in femtoseconds) with $\alpha = 0.05$. One can see the magnetic wave propagating through the array on slightly exciting the last spin. It then reflects before it dies out.

phase before quickly settling into approximately the same phase and very low frequency in the region 2-4 ps in figure 5. Blue corresponds to the S_x of S_1, green to the seconds spin etc. After this time, the wave has reflected back.

By looking at very small timesteps, one can see that S_1 starts excited in the x-direction while all others only have a non-zero component in the z-direction. It will immediately seek to align with its neighbours by decreasing its x-component. This can be explained mathematically. Using equation 1, there must be an effective field in the positive z-direction from both the first and second term. When taking the cross product in the first term of the LLG, there must then be an increase in the y-component (due to $-\gamma$ in the LLG) and necessarily a decrease in the x-component in the next step, which can be observed in figure 4. The damped term will not contribute to $\mathbf{S}_1$ for the first timestep. $\mathbf{S}_2$ will experience an effective field in the positive z- and positive x-direction and will turn in the negative y-direction. Later, once $\mathbf{S}_1$ has a significant y-component, it will turn in the positive x-direction. After some time, $\mathbf{S}_2$ will have a positive x-component and negative y-component, which necessarily means that $\mathbf{S}_3$ will have to decrease its x- and increase its y-component, which is the reason why $\mathbf{S}_3$ is decreasing while $\mathbf{S}_2$ is increasing in figure 5. Interestingly, this exact same behaviour is seen in reverse for the slightly reflected wave, but this is the topic for task d.

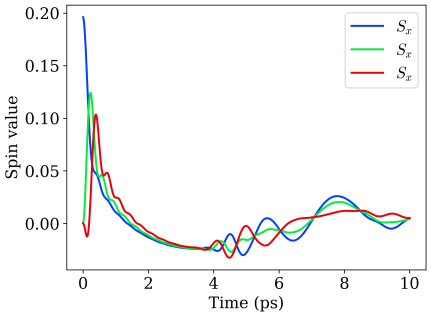

Figure 5: Shows $\mathbf{S}_x(t)$ develops. By comparing the graphs to the quiver simulation, one can see that the spins precess 90 deg out of phase, initially.

A perhaps better visualization is shown in figure 6, only showing the first 4 ps. Here one can see

that initially, the first and second spin start 180° out of phase with each other, while the third is 90° out of phase with both.

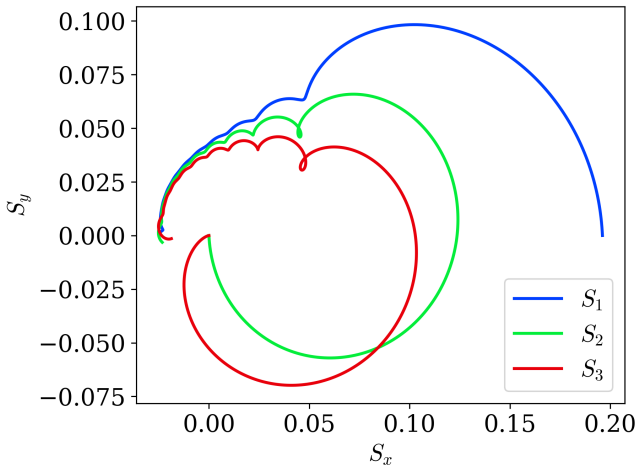

Figure 6: A polar plot of the trajectories for the first three spins from 0-4 ps.

3.4 d)

Task d is done with the same parameters as c, except that $\alpha = 0$. Since the system is closed, I will expect that the last spin $\mathbf{S}_n$ will keep influencing its neighbour causing the magnetic wave to reflect off the edge and travel back (as already seen). This effect can be observed in figure 7. While slight reflection could also be observed in figure 4, this time it persists, since it is undamped. The $S_x(t)$-curve will look the same as for figure 5, until the wave reflects back, as seen in figure 8. One can see that after the reflection, the spins have the same behaviour as they had initially, only know $\mathbf{S}_3$ is the "leading" spin 180° out of phase with its nearest neighbour and 90° out of phase with the leftmost spin, $\mathbf{S}_1$.

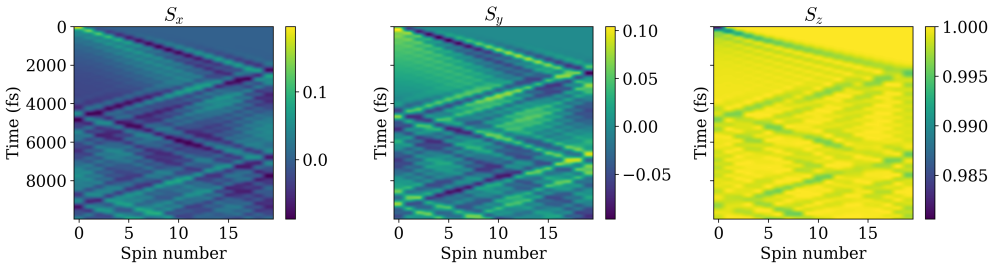

Figure 7: A visualization of $\mathbf{S}_j$(t) (in femtoseconds) with $\alpha = 0$. One can see the magnetic wave propagating through the array before it reflects back.

3.5 e)

For this task, the function that makes the time evolution of the spins is slightly changed, as we are know working with PBC. While it could have been made general, for the ease of it, I have made two functions similar to `evolve_spins()` with the only exception that they use PBC, and one of them is made for a linear lattice while the other is made for a square lattice. They are called `evolve_spins_pbc_linear()` and `evolve_spins_pbc_square()`. Unlike the non-PBC case, the input data is no longer padded and in order to avoid an "index out of range" error, I use the modulo operator to make $\mathbf{S}_1$ the neighbour of $\mathbf{S}_N$. When repeating task d, under the assumption of PBC,

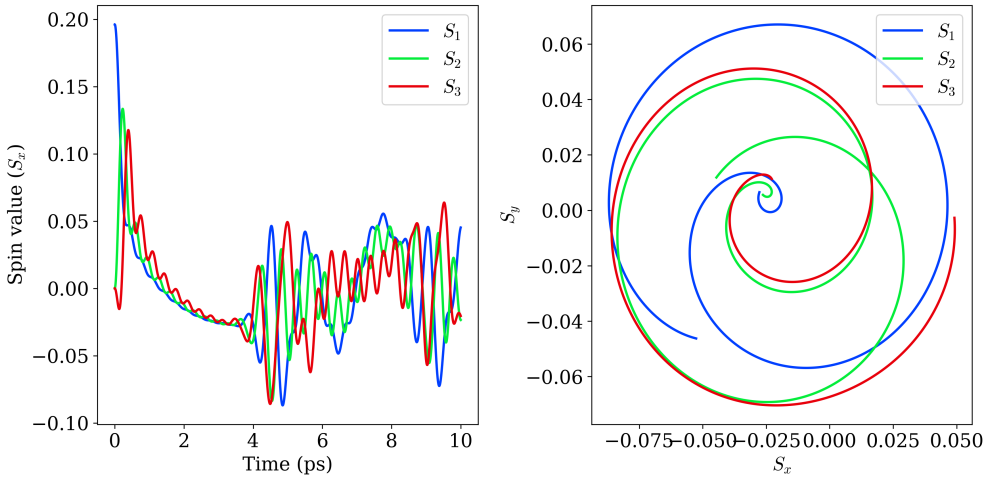

Figure 8: A visualization of the trajectories of the three first spins of the chain in the undamped case. The polar plot is from 3.5-5 ps, with all spins starting at the center.

I observe standing waves, as shown in the figure 9. In figure 10, one can observe several things. Firstly the neighbours of the initial excited spin (red) completely overlap, which is as expected for a standing wave. Next, they seem to usually have the same frequency, although it can be a little hard to observe. If one, however, looks at the peaks, one will always find a peak for S_{xN} and S_{x1} close to where there is a valley for S_{x0}. The same relation holds true for all dimensions, and show that the neighbouring spins have almost the same frequency as its central neighbour (but not always the same), but are shifted by a phase of 180 deg and experience different effects from the dispersion, which is making the detailed plot messy.

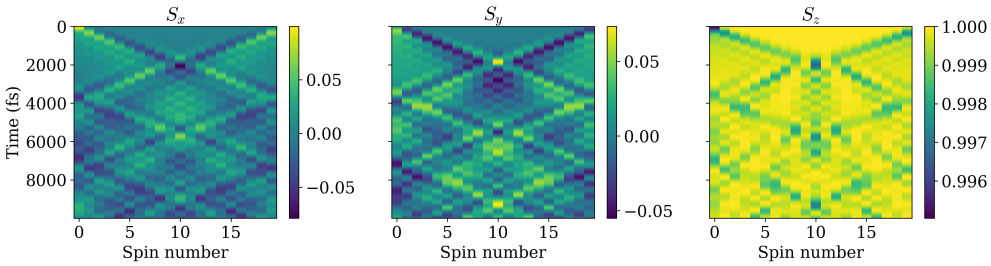

Figure 9: A visualization of $\mathbf{S}_j(\mathrm{t})$ (in femtoseconds). One can see standing waves being formed when using PBC.

3.6 f)

The initialization of all spins in random directions have already been covered in section 2.1, so I will not repeat it here. Apart from that, the problem is much the same as in e, but I will have to increase the number of steps for it to consistently reach the ground state, except for the antiferromagnetic case which always quickly finds it ground state. Figure 11 shows the formation of different magnetization domains. As is seen, it still changes somewhat with time and it might collapse to one single domain later, which I have also observed earlier, so it can perhaps not be called the ground state. But it is interesting and will only happen when the magnetic field is off, since the positive z-direction is not favored. I have also observed all negative magnetization for these parameters.

Figure 12 shows the formation of a single positive magnetization domain. This will always happen sooner or later, since aligning in the positive z-direction is favored when $B_0 > 0$. It has reached

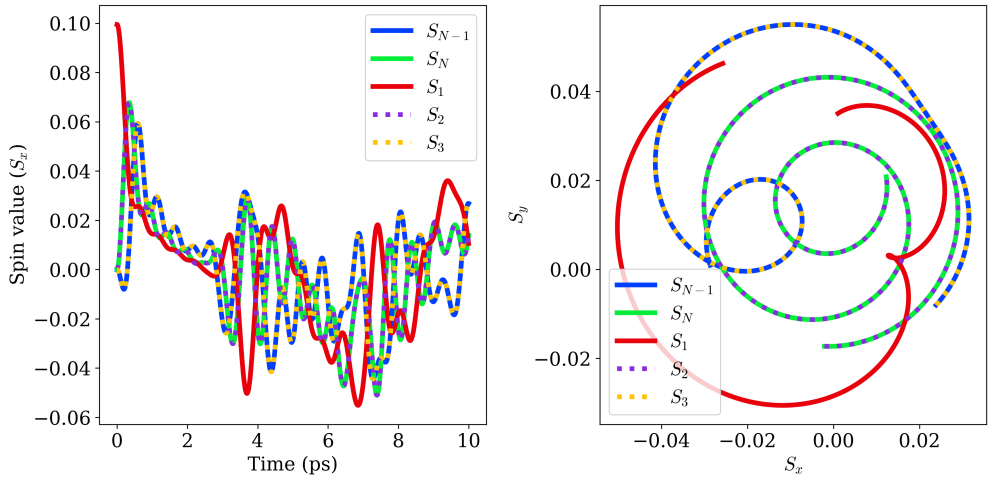

Figure 10: Shows the time evolution for the initial excited state (red) and its neighbours using PBC and no damping. The polar plot is from 3.5-5 ps.

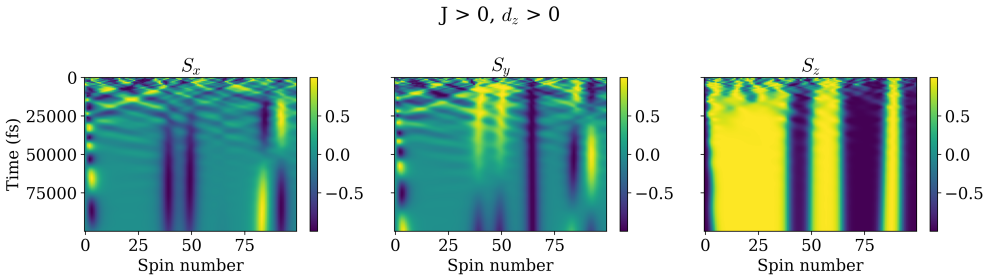

Figure 11: Shows the formation of different magnetization domains. As is seen, it still changes somewhat in time and it might collapse to one single domain later.

its ground state (at least more or less) after 100 ps.

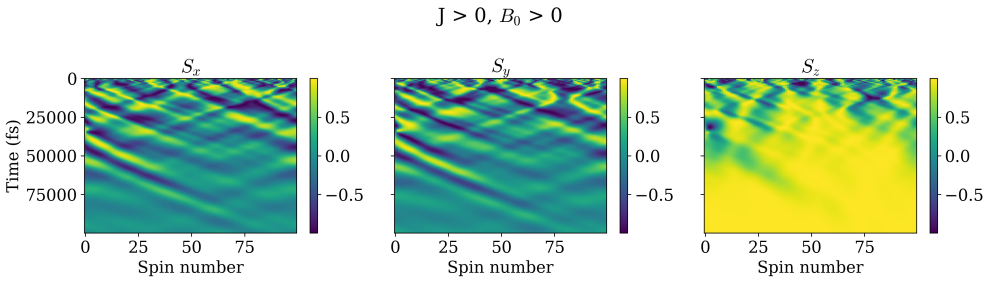

Figure 12: Shows the formation of a single positive magnetization domain.

Figures 13 and 14 both shows the antiferromagnetic case. Figure 13 is not interpolated in the `plt.imshow()` function, so it is very visible that the spins alternates between the positive and the negative z-direction. Figure 14 shows how domains in the xy-plane might also be formed in the antiferromagnetic case if the number of atoms on the linear lattice is large enough. One reason for this might be colliding antiferromagnetic domains, who cannot agree on the ordering, loosely speaking, but I do not know if this is physically plausible. If there was rather a B-field in the positive z-directions, the spins would seek to align in that direction thus also forming alternating spin up spin down in the xy-plane with high probability, rather than in the y-direction.

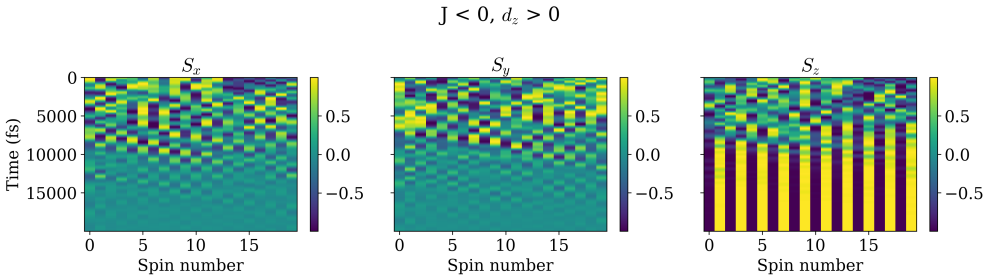

Figure 13: Shows the formation of an antiferromagnetic structure.

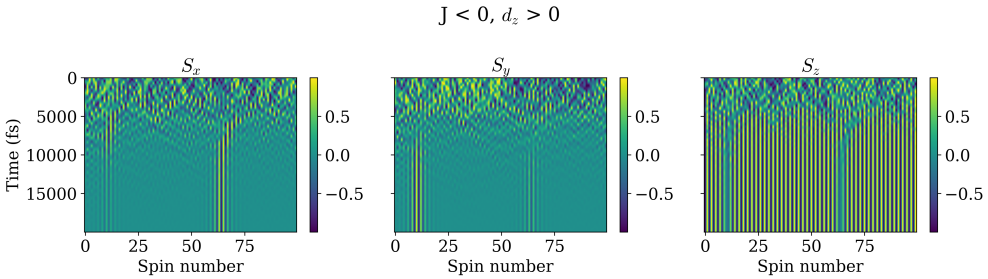

Figure 14: Shows the formation of a single positive magnetization domain.

3.7 g)

We know move to the 2D-lattice with finite temperature and stronger damping. Figure 15 is a sanity check for the implementation in 2D, showing that one still has formation of antiferromagnetic domains and that the PBC works fine. Note that the colorbars have different scale, so there is very little magnetization in the xy-plane and it would probably disappear if I ran it longer. This was also for T = 0. Also note that this shows only the final timestep, and not the time evolution as previously displayed.

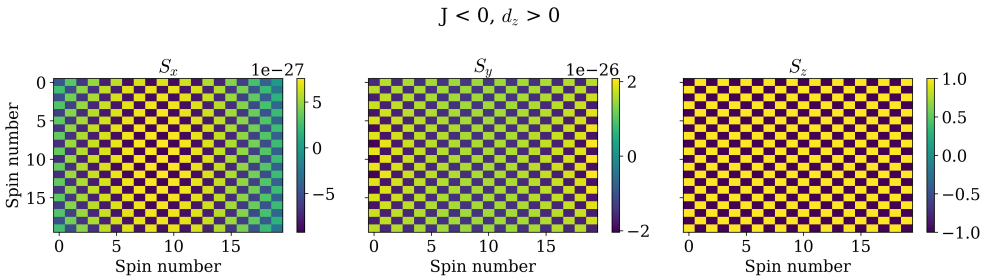

Figure 15: Shows formation of a antiferromagnetic domain in 2D for 20x20 spins. The colorbars have different scale.

Equation 9a and 9b from the exam set is found in the file `equations.py` and is implemented as is. From now on, only the magnetic field is kept as a symmetry-breaking term and all spins are initialized in the positive z-direction. Figure 16 shows the time-dependent magnetization $M(T, t)$ and the temporal average $M(T)$. The time-dependent magnetization fluctuates around 0.98. In figure 16, the temporal average could have been found by averaging the data from a few picoseconds and onwards, but in general, it is safer to let it run from i.e. 20 picoseconds and for a sufficiently long time.

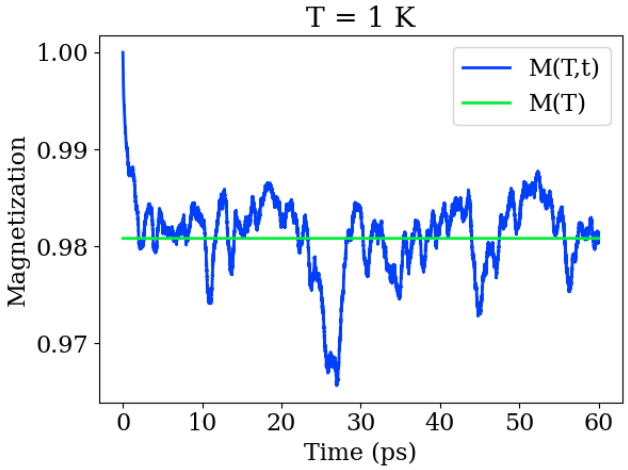

Figure 16: Shows how the time-dependent magnetization fluctuates around the temporal average, $M(T)$ for 1 K.

3.8 h/i)

The last two tasks of the exam is merged into a single in this report. The center plot in figure 17 shows the phase diagram obtained for $B = B_0$. This shows that the magnetization drops to zero around 60 K and the critical temperature, T_c, is definitively found no higher than around 70 K. All three phase diagrams are found using a 30x30 spin lattice, $\alpha = 0.5$, $J = 1$meV and $B_0 = 1.72$T. T_c separates the ordered ferromagnetic phase at $T < T_c$ from the disordered paramagnetic phase at $T > T_c$. When the spins are oriented in random directions, one will expect that the average magnetization is 0, while on the other hand, when the spins are oriented up, the magnetisation is 1, as for the ferromagnetic case. This means that at elevated temperature, the effect of the introduced random thermal field will be the dominant contribution to the effective field. When one decreases the applied magnetic field, one would expect a weaker tendency to remain pointing in the positive z-direction. This is observed for $B = 0.3B_0$ in figure 17. Here, the magnetization drops considerably faster than at $B = B_0$. This is as expected, since with a weaker magnetic field, T_c can be lower, since $F^{th} \propto \sqrt{T}$. For this magnetic field, $T_c \approx 40$K. When $B = 2B_0$, one can see a corresponding increase in T_c. It is hard to judge by the plot, but one can estimate T_c to be around 90K.

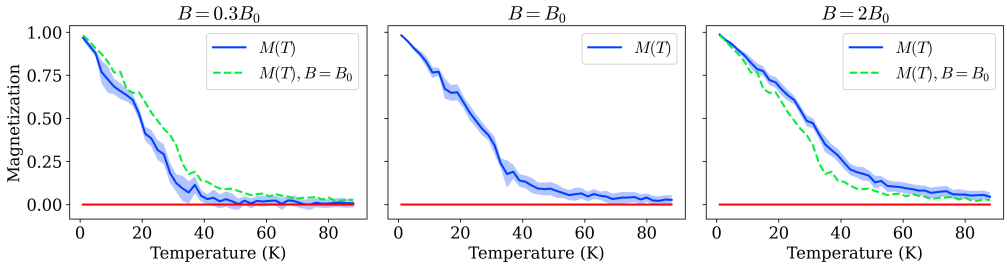

Figure 17: Shows phase diagrams for different magnetic fields. The uncertainty is included as the shaded blue area.

Figure 18 shows how the phase diagrams would look using the alternate random vector generator `make_3D_gaussian_alt()` as previously described. Here one can see that the increased magnitude of the thermal field results in considerably lower T_c's. By comparing the two figures, one can see that the one using the spherical gaussian random noise have T_c's consistently 3 times larger than the one constructing a random vector from three gaussian numbers. This makes sense, since it is the thermal field that determines the critical temperature for a given B_0 and `make_3D_gaussian_alt()`

should be 3 times larger than `make_3D_gaussian()` since it contains three gaussian versus one gaussian in 3D.

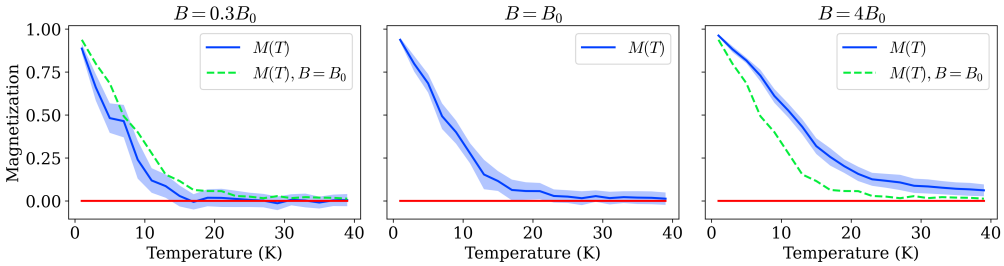

Figure 18: Shows phase diagrams for different magnetic fields using the alternative random vector generator. The uncertainty is included as the shaded blue area.

4 Final remarks

There are definitely room for improvement with regards to code efficiency and better visualizations. The code is partly written to be easy to read and develop. If I wanted to make in more efficient, I could reduce the number of redundant calculations. Another alternative could be to flatten the 2D lattice in the Heun scheme and rather use a bit more complicated indexing. I could then have for-loops of 900 spins instead of 30x30. Then my code could probably be faster using `prange` from numba, which makes for-loops run in parallel. Code efficiency have, however, not been a big problem. I found it hard to visualize the phases and frequencies of the spins among all the noise that I assume to be dispersion and I would have liked to have better visualizations in order to better understand and explain what was going on.

Bibliography

[1] Charles R. Harris et al. 'Array programming with NumPy'. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Siu Kwan Lam, Antoine Pitrou and Stanley Seibert. 'Numba: A llvm-based python jit compiler'. In: (2015), pp. 1–6.

[4] Cory Simon. 'Generating uniformly distributed numbers on a sphere'. In: (2015). URL: http://corysimon.github.io/articles/uniformdistn-on-sphere/ (visited on 29th Apr. 2022).

[5] Ingve Simonsen. 'Exam in TFY4235/FY8904 Computational Physics'. In: (2022).

[6] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.