Application of established heuristic solvers to physics problems

Michael Negus
Diamond Light Source Ltd.
m.negus@warwick.ac.uk

September 16, 2016

Abstract

A range of non-linear heuristic solvers were tested for their effectiveness at solving a two-dimensional system of interacting spins. Then the most effective solver was chosen to simulate more complex, three dimensional systems of spins; including models of $FeBO_3$ and magnetic skyrmions.

1 Introduction

Optimisation problems that are computationally difficult to solve can be addressed using highly efficient heuristic solvers. Solvers are pieces of software which use one or many algorithms to find the solution to a given optimisation problem. Being heuristic means instead of finding the exact answer, they compute a good approximation, and as a result tend to be much faster.

Heuristic solvers find applications in many business operations, such as providing efficient scheduling [1]. Because of this, there are many solvers available which are highly efficient at solving different types of problems. These solvers are not restricted to business applications; they are designed to solve all types of optimisation problems.

Solvers are used in optimisation problems. An optimisation problem is defined using an algebraic modelling language, and then this problem definition is passed into a solver. The advantage of working like this is the problem is completely separated from the solution. Not only does this make code simpler and easy to read, it allows for multiple algorithms to be tested for solving the one problem without having to change any code.

Many problems in physics can be expressed using optimisation. Any problem where a function needs to be minimised classes as optimisation; so an example would be finding the path that minimises the action in Lagrangian mechanics. Hence, these solvers should be of use in physics as well as business. The purpose of this project was to investigate the feasibility of using heuristic solvers in physics problems.

To do this, one physics problem was chosen to solve; a lattice of interacting spins. A range of solvers were then chosen to solve this problem, and their speed and accuracy were compared. The problem size was varied so the effectiveness of the solvers when given different number of variables could be determined. If a solver was able to complete the problem in a reasonable time and with a high degree of accuracy, then it was deemed fit for purpose at solving this particular physics problem.

2 Physics problem

The problem chosen was a two-dimensional lattice of interacting spins. Physically, this represents single layer of material, where you have a Bravais-lattice structure of atoms/molecules. Each of these lattice points has a spin angular momentum vector, which is shortened to simply being called "the spin". How these spins are aligned determines the magnetic properties of the lattice. For example, when all the spins are parallel, the material is said to be ferromagnetic.

This system can be described using the Classical Heisenberg model [2] in two dimensions. The definition used here follows:

For a two dimensional lattice with no external magnetic field, and a set of two dimensional spins

of unit length:

$$\mathbf{S}_i \in \mathbb{R}^2, |\mathbf{S}_i| = 1$$

where each spin is placed on a lattice node.
Then the model is defined through the following Hamiltonian:

$$\mathcal{H} = -\sum_{i,j} J_{i,j} \mathbf{S}_i \cdot \mathbf{S}_j \tag{1}$$

with $J_{i,j}$ being a parameter the strength of the coupling between spins $\mathbf{S}_i$ and $\mathbf{S}_j$, and is a property of the material.

The spins will align in such a way to minimise equation (1). This will depend entirely on $J_{i,j}$. For example, if:

$$J_{i,j} = \begin{cases} -1, & \text{if } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

then each spin will align to be anti-parallel to its neighbours, due to the fact $\mathbf{S}_i$ anti-parallel to $\mathbf{S}_j \implies \mathbf{S}_i \cdot \mathbf{S}_j = -1$, meaning minimised scalar product terms, resulting in $\mathcal{H}$ being minimised. This is anti-ferromagnetic behaviour.

3 Optimisation problem

Once the problem had been defined, solvers appropriate for the problem had to be found. Each solver is designed to solve a certain type of mathematical optimisation problem, so the Classical Heisenberg model has to be expressed as one.

In essence, mathematical optimisation is where some function $f : A \to \mathbb{R}$ from a set A to the real numbers, needs to be minimised. By that, this means finding an element $x_0 \in A$ such that $f(x_0) \leq f(x), \forall x \in A$. There also may be additional bounds and constraints on the x variables that have to be satisfied.

With the Classical Heisenberg model, it is clear that the variables are the spins, as they need to be altered to find an optimum solution. Each spin is a 2D vector, meaning they have two Cartesian components. However, expressing them in polar coordinates, and exploiting the fact all of the spins are unit length, means only one variable is needed per spin: the polar angle.

Defining the Classical Heisenberg model as an optimisation problem:

For a two dimensional lattice, with a set of spins at each lattice node, the variable for each spin is the polar angle:

$$\theta_i \in [0, 2\pi)$$

If there are N spins in total on the lattice, then the Hamiltonian takes the N-dimensional vector $(\theta_1, \ldots, \theta_N)$ as input. Hence, the function:

$$\mathcal{H}(\theta_1, \ldots, \theta_N) = -\sum_{i,j} J_{i,j} \cos(\theta_i - \theta_j) \tag{3}$$

needs to be minimised.

Equation (3) is referred to as the objective function. Optimisation problems that have non-linear terms in the objective function are referred to as non-linear problems. Hence, non-linear solvers were required for this problem.

4 Solver packages

Usually, algebraic modelling languages are used to interface with solvers. These languages are used to define the optimisation problem, then this data is passed into the solver. Some of these are entire languages unto themselves, such as AMPL [3] or GAMS [4]. However, some are based on existing languages, for example Pyomo [5] [6] [7] is a Python-based modelling language.

Pyomo was chosen as the modelling language to use, partly due to the fact it is open-source, and also as it is based in Python. Python is one of the most popular programming languages in the world [8], especially in the science community; so using Pyomo means there's a smaller amount of new syntax to learn for a someone already familiar with Python.

Pyomo is written to be compatible with most of the commercially built solvers available, specifically any compatible with AMPL [3], AIMMS [9] and GAMS [4], which are three of the most popular algebraic modelling languages. Because of this, there should have been a wide range of solvers available. Indeed, there were many solvers, but the majority of the non-linear solvers had to be paid for. The scope of this project meant solvers could not be bought, so only open-source solvers were used.

However, one open source solver compatible for Pyomo was found, which was Ipopt [10]. Ipopt stands for **I**nterior **p**oint **opt**imiser, which is a solver designed to use interior point methods to solve large scale non-linear optimisation problems. For more details on how the algorithm works, see [11].

An issue was that one solver was not enough, as multiple solvers needed to be compared. Two more packages were found that allowed access to solvers for non-linear problems, although they themselves were not algebraic modelling languages, like Pyomo.

First was SciPy [12]. This is a popular Python package and is used by many scientists for its scientific computing features. Specifically, it has various methods for optimisation [13]. It includes a range of different algorithms for finding local and global minima of a given function, which will be used here to solve the spin problem.

Secondly, a Python package called pyOpt [14] [15]. This package is for formulating and solving non-linear optimisation problems. It has 21 built in "optimisers" [16] which can be used to solve a given non-linear optimisation problem.

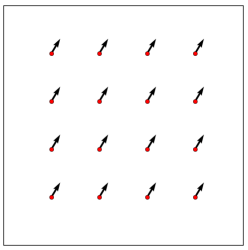
Although terminology differs, all of the methods used by these packages can be referred to as solvers, and will be from here on. Between the three packages, Pyomo, SciPy and pyOpt, there were enough non-linear solvers to have an adequate comparison.

5 Code structure

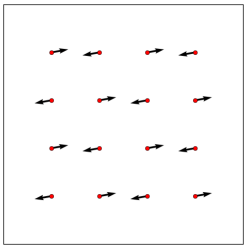
Once the packages were found, the Classical Heisenberg model was formulated in code. As all three packages are Python based, most of the code was repeatable. However conventions and syntax from each package meant the way of defining the optimisation problem varied.

There are two main steps to defining the optimisation problem in code:

- **Variables:** These were defined to be the polar angles of the spins on a grid. The grid dimensions (the number of rows and columns on the grid) were user input. The variables were indexed by row coordinate and column coordinate, using standard matrix notation. The variables were floats in the range $[0, 2\pi)$. Their initial starting direction was chosen by the user.



(a) Ferromagnet



(b) Anti-ferromagnet

Figure 1: Plots of 2D lattice of spins

- **Objective function:** This was the Hamiltonian given in equation (3). It was a Python function which takes the grid of spin variables as an input. The value of $J_{i,j}$ was user input and specified outside of the function.

Once the variables and objective function had been defined, they were passed into the solver by the package. The solver (if correct) returned the minimum value of the objective function and an array with the values of the spin variables at that minimum.

Code was also written to plot the lattice of spins, using the Python package matplotlib [17]. Examples of the plots produced are shown in figure 1.

6 Solver testing

6.1 Test problem

Once the code had been written, the solvers were then ready to be tested. The solvers were given the

Classical Heisenberg model to solve, with:

$$J_{i,j} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

which is a ferromagnetic case. This means a correct solution is where all of the spins on the lattice are parallel to each other.

6.2 Solver selection

A preliminary test was done to see which solvers would be able to attempt solving the problem, as some threw exceptions and could not begin to solve any spin problem. Between the three packages, 15 were chosen that successfully ran when given the problem. These solvers were:
Pyomo: Ipopt
SciPy: Nelder-Mead, Powell, CG, BFGS, L-BFGS-B, TNC, SLSQP, COBYLA, Differential Evolution
pyOpt: ALHSO, CONMIN, PSQP, SDPEN, SOLVOPT

6.3 Correct results

Next, a quantifiable way of determining whether or not a result was "correct" needed to be established. Qualitatively, an answer is correct if all of the spins are parallel. Thus the standard deviation of the spin vectors is expected to be zero. As the spins are represented by the polar angles, the circular standard deviation of the spins was needed. This is similar regular standard deviation, but takes into account that 0 and 2π radians are in fact the same direction. Through analysis of results, it was found if the circular standard deviation of the spins was ≤ 0.1, then the result was deemed correct.

6.4 Tests

The solvers were to tested against this problem, with grid size from 2×2 up to 25×25 spins. This was done in order to establish the scaling of the solvers with different problem sizes.

For each size, the solvers were tested with ten different initial conditions of the spins. The initial conditions involved randomising the direction of each spin.

The time each solver took to run was recorded for each initial condition, as well as whether or not the answer it gave was correct. This was both so the solve time could be averaged and so that it could be established how many times out of ten the solvers gave a correct answer.

7 Testing results

7.1 Individual results

Follows are the results for each individual solver. The results are plotted against the side length of the grid, n (i.e. $n \times n$ grid). On the graphs, the blue line is the time the solver took to solve the problem in seconds, averaged over the ten initial conditions. The error bars indicate the maximum and minimum time the solver took. The red line is the number of times the solver got the correct answer out of the ten attempts.

Note how some of the graphs do not go up to $n = 25$, this is because these solvers were reached their maximum number of iterations and timed out before reaching the larger problems. For example, Differential Evolution would not produce answers for the 14×14 grid, as the number of variables was too large.

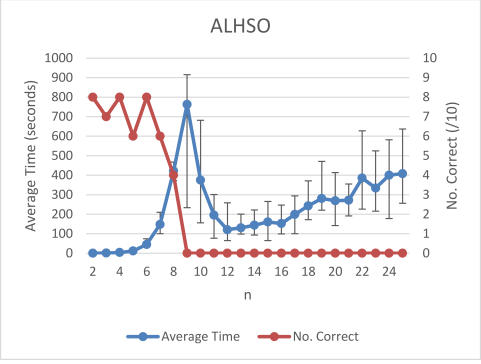


Figure 2: ALHSO

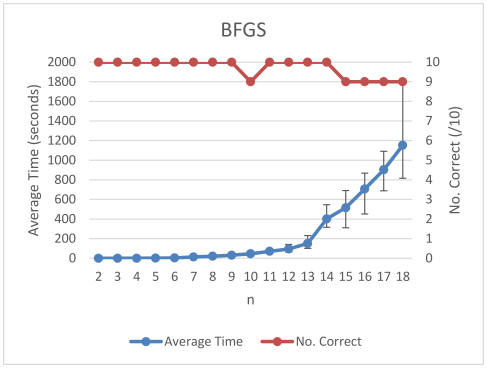


Figure 3: BFGS

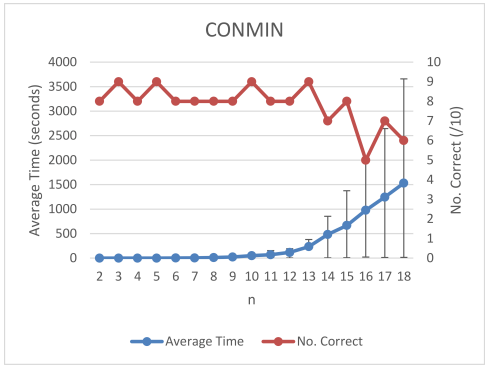


Figure 6: CONMIN

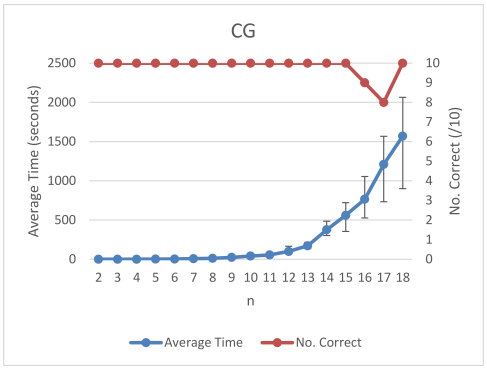


Figure 4: CG

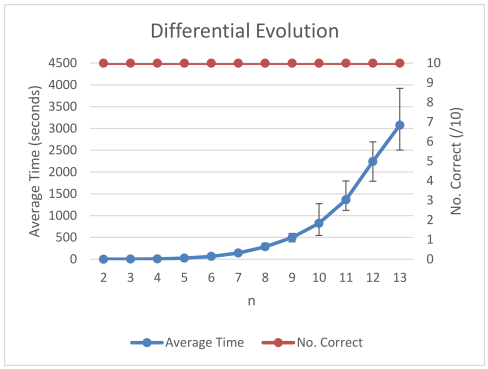


Figure 7: Differential Evolution

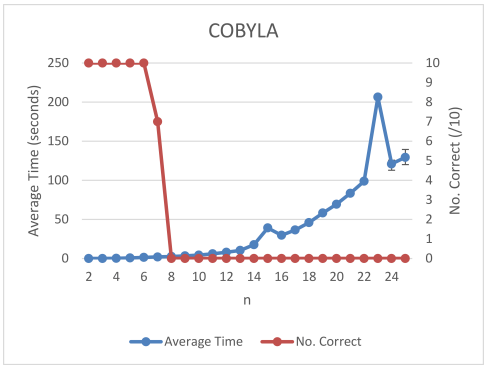


Figure 5: COBYLA

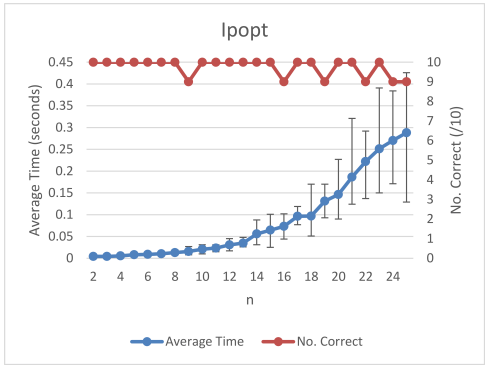


Figure 8: Ipopt

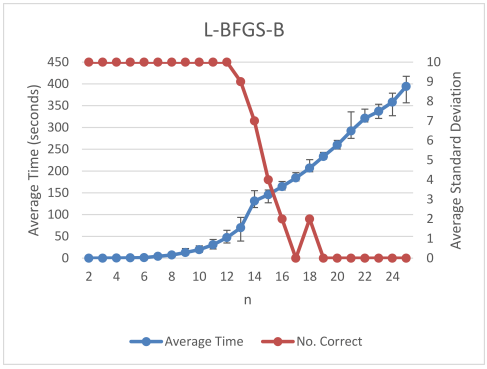


Figure 9: L-BFGS-B

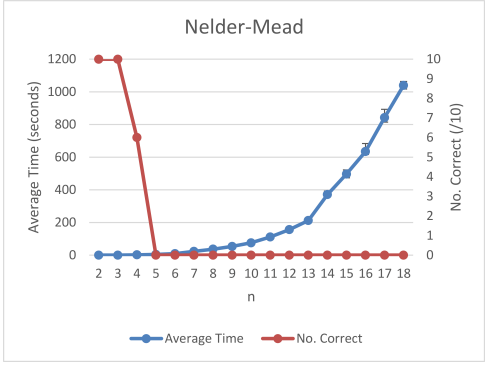


Figure 10: Nelder-Mead

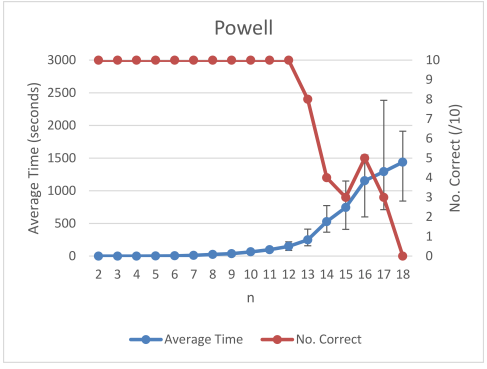


Figure 11: Powell

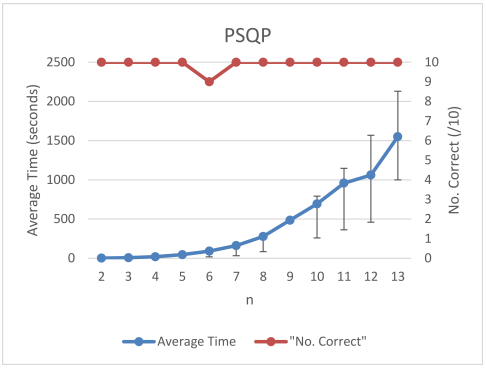


Figure 12: PSQP

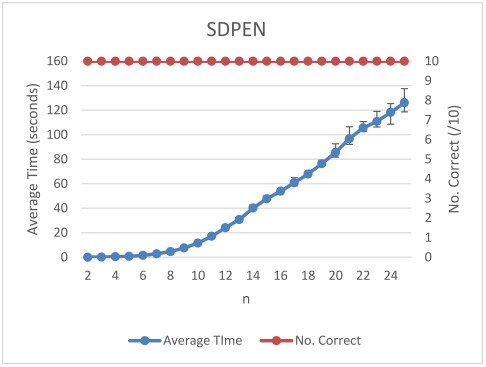


Figure 13: SDPEN

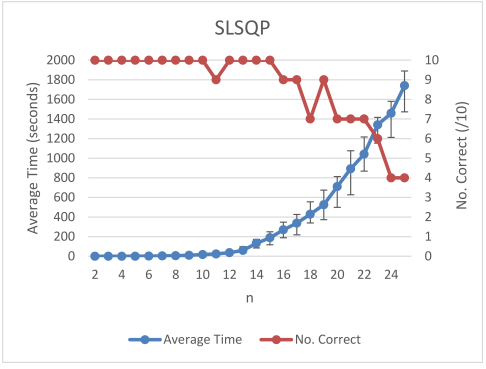


Figure 14: SLSQP

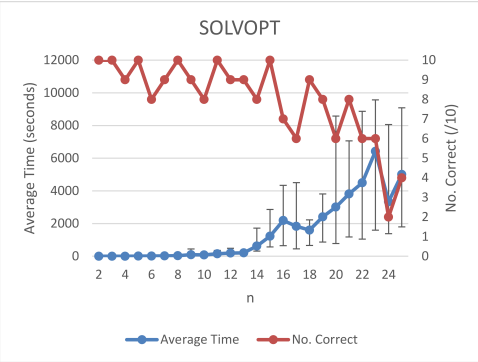


Figure 15: SOLVOPT

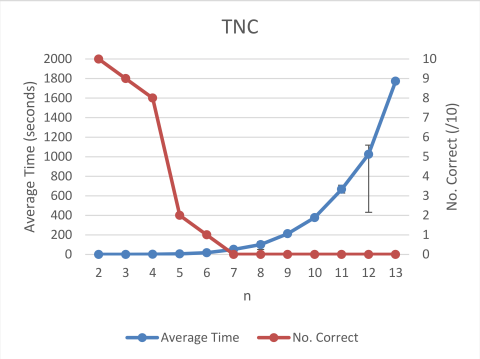


Figure 16: TNC

7.2 Collected results

A graph of all of the solver times is show on figure 17. Note the logarithmic scale on the Average time axis.

8 Testing conclusions

Here the results of the solve times were interpreted and conclusions on the effectiveness of each solver given.

8.1 Accuracy

A few of the solvers were generally unreliable at any problem size. This means the number of correct solutions they gave was erratic, in the sense there was no problem size where they were consistently correct. Solvers that fell under this category are ALHSO, CONMIN, SOLVOPT and TNC.

Many of the solvers started off with consistently correct solutions, but this would break down as the problem size became larger. For example, L-BFGS-B was getting 10/10 correct solutions up until $n = 12$, but then this broke down until it was getting 0/10 correct solutions from n=19 onwards. Other solvers that exhibited similar behaviour were COBYLA, Nelder-Mead, Powell and SLSQP. It can be concluded that these solvers are reliable for smaller problem sizes, but break down as they become larger.

The other solvers were quite consistent at giving correct answers. However it should be noted that some of these solvers did not manage to solve as many problems as others. For example, although Differential Evolution would get 10/10 correct solutions on every problem it solved, it only managed to get to a problem size of $n = 13$. In contrast, Powell got to $n = 12$ and then began to get incorrect solutions after that, so it is possible that Differential Evolution would have began giving incorrect answers if it had carried on. Ipopt and SDPEN were the only solvers which managed to make it to $n = 25$ while still getting consistently good answers (at worse Ipopt would get 9/10 correct solutions).

So in purely reliability terms, SDPEN was the best solver as it managed to solve all 25 problems without getting any incorrect solutions.

8.2 Solve time

Although producing correct solutions is the most important factor, if two solvers are both producing correct answers, then comparing the time they take is the next way to compare. From looking at figure 17, it is immediately clear that the fasted solver was Ipopt. At $n = 25$, its average solve time was around 0.3 seconds. Compare this to the next best solver that gave a good number of correct answers at $n = 25$, SDPEN, which took around 130 seconds on average; making Ipopt was around 400 times faster than the next best solver.

Other solvers were taking a few hundred seconds at this problem size, like SDPEN and SLSQP. The other solvers that did not make it to this problem size were usually having solve times of a few thousand seconds.

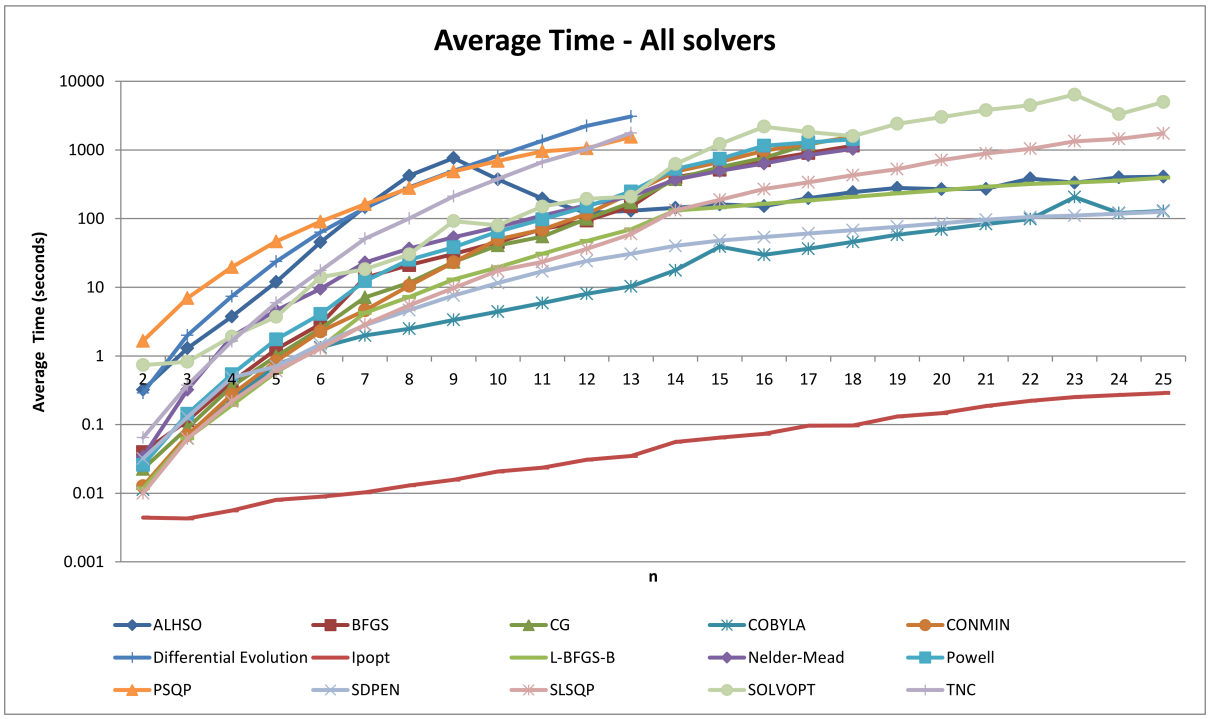


Figure 17: All solver times

9 Extensions

Once the capability of the solvers on a two-dimensional lattice was established, a larger, more complicated problem was written in code to see to what extent the solvers can be used.

9.1 New physics problem

A three-dimensional lattice of three-dimensional spin vectors was chosen to be the problem. This can still be modelled using the Classical Heisenberg model, but now the spin vectors are three-dimensional.

As well as the spin coupling terms, three more interactions were added to the model:

- **Magnetic field:** An external magnetic field can be applied at each lattice point. This magnetic field is expressed by a magnetic field vector.
- **Single-ion-anisotropy:** This is a term that indicates which direction the spin prefers to point in. It can either specify that a spin wants to be in a certain plane, or aligned to a specific direction (either parallel or anti-parallel to it).
- **Dzyaloshinskii-Moriya interaction:** This is a term that involves the coupling of the spins with another one, except it involves a vector product term instead of scalar product, as well as a vector called the Dzyaloshinskii-Moriya vector.

So the definition of this three-dimensional problem follows [18]:
For a three-dimensional lattice, and a set of three-dimensional spins of unit length:

$$\mathbf{S}_i \in \mathbb{R}^3, |\mathbf{S}_i| = 1$$

where each spin is placed on a lattice mode.
Then the model is defined through the following Hamiltonian:

$$\mathcal{H} = -\sum_{i,j} J_{i,j}\mathbf{S}_i \cdot \mathbf{S}_j - \sum_i \mathbf{h}_i \cdot \mathbf{S}_i - \sum_i K_i(\hat{\mathbf{k}}_i \cdot \mathbf{S}_i)^2 - \sum_{i,j} \mathbf{D}_{i,j} \cdot (\mathbf{S}_i \times \mathbf{S}_j) \tag{5}$$

The first term is the same as equation (1), which represents the spin coupling.
The second term is the magnetic field interaction, where $\mathbf{h}_i$ is the magnetic field vector at spin $\mathbf{S}_i$.
The third term is the single-ion-anisotropy. K_i is the strength of the anisotropy, and $\hat{\mathbf{k}}_i$ is the unit direct which the anisotropy lies. If $K_i > 0$, then $\mathbf{S}_i$ will want to be parallel or anti-parallel to $\hat{\mathbf{k}}_i$. If $K_i < 0$, then $\mathbf{S}_i$ will want to be in a plane perpendicular to the vector $\hat{\mathbf{k}}_i$.
Finally, the fourth term is the Dzyaloshinskii-Moriya interaction. The $\mathbf{D}_{i,j}$ is the Dzyaloshinskii-Moriya vector from spin $\mathbf{S}_i$ to spin $\mathbf{S}_j$.

With all of these interactions accounted for in the model, many different magnetic materials can be simulated, rather than just simple ferromagnetic/anti-ferromagnetic behaviour.

9.2 Code structure

The structure of the code was in essence similar to that of the two-dimensional lattice. Extra complications came from the fact the spins were now three-dimensional, so were parameterised in spherical coordinates by the polar angle and the azimuth angle. Also the extra interaction terms had to be taken into account.

Only the Ipopt solver was tested using this model. This was due to the fact it was by far the fastest, and time constraints on the project meant the model was only created in the Pyomo package, so other solvers could not be tested.

The three-dimensional plotting was still done using matplotlib.

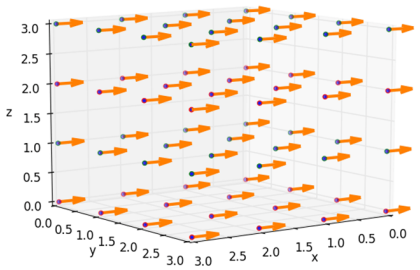
9.3 Example models

A set of example models were created to test the extent of how good the Ipopt solver was at solving the three-dimensional problem.

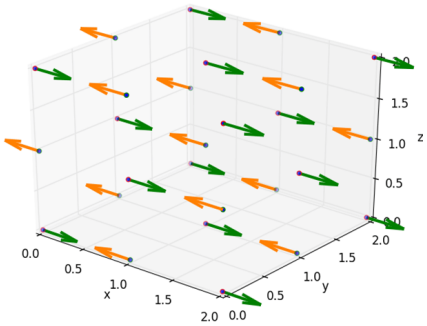
9.3.1 Ferromagnet and anti-ferromagnet

The simplest test was to just have the spin-coupling interactions, with none of the others. This then was the exact same problem as the solvers were tested on before, except in three-dimensions. The plots are show in figure 18.

The parameters in equation (5) are as follows: For the ferromagnet, $J_{i,j}$ is the same as in equation (4), and for the anti-ferromagnet, $J_{i,j}$ is the same as in equation (2). $\mathbf{h}_i$, K_i and $\mathbf{D}_{i,j}$ are all zero for both, as none of these interactions are happening in the model.



(a) Ferromagnet



(b) Anti-ferromagnet

Figure 18: 3D plots of a ferromagnet and an anti-ferromagnet

9.3.2 Weak ferromagnet: $FeBO_3$

The next material modelled was $FeBO_3$, which is described as being a "weak ferromagnet". The iron atoms of the material form ferromagnetic layers, that is all of the spins of the iron atoms on each layer are parallel. For an iron atom in $FeBO_3$, its closest neighbours are in the plane above and below it. There are three above, and three below, which is seen in Figure 1 of [20]. The "neighbours" of a spin will refer to these six other spins. Each spin has an anti-ferromagnetic exchange with the closest spins from the layers above and below it, meaning the spins on each layer want to be anti-parallel to the spins of the layers above and below it. However, there is also a Dzyaloshinskii-Moriya interaction

between each spin and its closest neighbours. This creates a slight "canting" of the spins, meaning the layers of spins are not exactly anti-parallel to each other. A strong anisotropic term restricts the spins to lie in planes parallel to the x-y plane.

This material was studied in [19] and [20]. The parameters used here to model $FeBO_3$ were taken directly from these papers. In them is also a schematic plot of how the spins on the iron atoms should look like, which can be compared to the plots from the code below.

The parameters in equation (5) for this model of $FeBO_3$ are as follows:

$$J_{i,j} = \begin{cases} -10.3, & \text{if } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{D}_{i,j} = \begin{cases} (0,0,-0.5), & \text{if } i \text{ and } j \text{ are neighbours} \\ (0,0,0), & \text{otherwise} \end{cases}$$

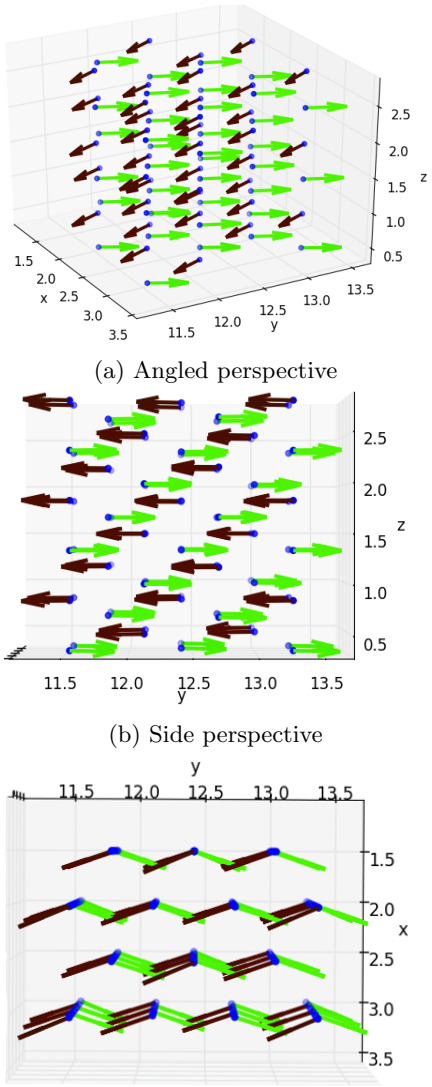
$$K_i = -100$$

$$\hat{\mathbf{k}}_i = (0,0,1)$$

$$\mathbf{h}_i = (100,0,0)$$

The plot is shown in figure 19 and shows a small selection of spins taken from the middle of a plot of size $20 \times 20 \times 20$.

The first image shows the plot from an angled view, which gives an idea of the three dimensions of the spins. You can see how the spins on each layer are parallel to each other, and are parallel to the spins two layers above and below it, as expected.
The second image shows the spins from the side-on. This shows how the spins in each layer are almost anti-parallel to the spins in the layers directly above and below it.
The third image is the view from above, and clearly shows how the spins in alternate layers aren't exactly anti-parallel, and instead are slightly canted towards each other, as expected.



(a) Angled perspective

(b) Side perspective

(c) Above perspective

Figure 19: Plot of $FeBO_3$ from three different perspectives

9.3.3 Magnetic skyrmions

Magnetic skyrmions are quasiparticles which can occur on the surface of magnetic materials. They can be described as "vortices" of spins as they look similar to vortices in fluid systems. They are currently an active research area due to their potential use in data storage.

They appear due to the Dzyaloshinskii-Moriya interaction and an external magnetic field. A single hexagonal lattice layer was modeled, with the values for the spin coupling, Dzyaloshinskii-Moriya vectors and magnetic field inspired by [18]. The neighbours of a spin are the six spins on the hexagon surrounding it.

The parameters in equation (5) for this model of

are as follows:

$$J_{i,j} = \begin{cases} 1.0, & \text{if } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{D}_{i,j} = \begin{cases} (0,0,1) \times \hat{\mathbf{n}}_{ij}, & \text{if } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise} \end{cases}$$

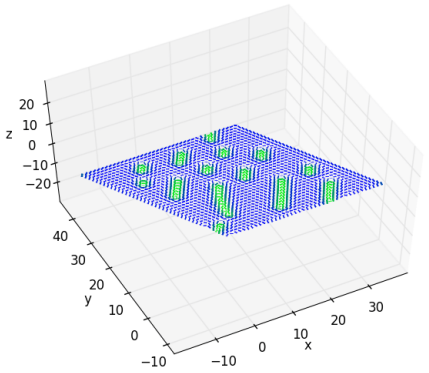
$$K_i = 0.5$$

$$\hat{\mathbf{k}}_i = (0,0,1)$$

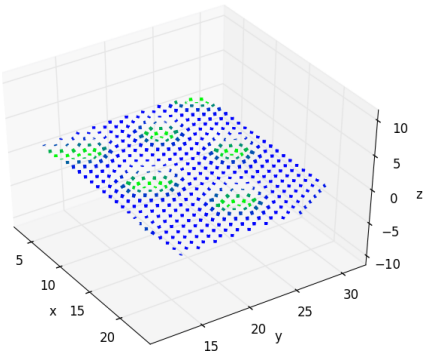
$$\mathbf{h}_i = (0,0,1.1)$$

where $\hat{\mathbf{n}}_{ij}$ is the unit vector from spin i to spin j.

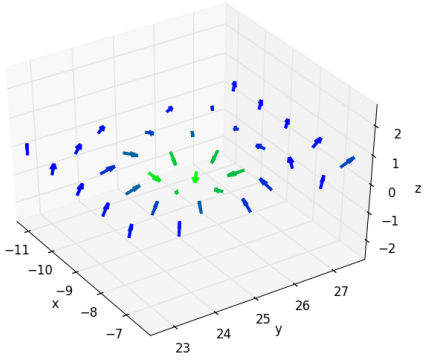
The plot is shown in figure 20 and shows a layer of spins of size 40×40. The spins are coloured such that the more positive their z-component is, the more blue they are, and more negative z-component makes them more green.



(a) Whole lattice



(b) Four skyrmions



(c) Single skyrmion

Figure 20: Plot of a 40×40 lattice containing skyrmions

9.4 Limitations

Despite the success in simulating the materials, there were some limitations that the code had.

- **Multiple solutions:** The solver would sometimes find difficulty converging to a correct answer when there were many different correct solutions. For example, a simple ferromagnetic case has very many valid solutions. In cases like this, the solver would often give an incorrect solution. Whereas for more complex problems, where there may only be one or two correct solutions, like where an anisotropy forces the spins to be in one of two directions, the solver was very likely to give the correct solution. An explanation for this may be due to the fact Ipopt specialises in finding local minima. In the complicated cases with many different interactions, there will be few local minima as there are only a few correct solutions. However, on very open ended problems such as a simple ferromagnet, the solver is more likely to get stuck in a local minima which is not a correct solution. So for this case, a solver that finds global minima may be better suited.

- **Initial conditions:** In some cases, the initial conditions of the spins would determine whether the solver converged to a correct answer. As in the previous point, this would mainly happen with problems that had a large number of possible solutions. Again trouble would come in the simple ferromagnetic case. If the initial conditions of the spins was such they would all point in a random direction, then the solver would be more likely to give

an incorrect solution, especially at larger grid sizes. This may be attributed to the large amount of disorder there is, so for the solver to deal with the huge number of variables to converge into one of many different energy solutions becomes difficult. A similar point happens in the opposite case, if the initial starting configuration of the spins is that they are all parallel, then the solver will often not manage to give a correct answer to the anti-ferromagnetic case, where each spin is anti-parallel to its closest neighbours.

- **Grid size:** A fundamental difference of the code implementation to actual problems is the size of the grid. The solver struggles with grids a lot larger that $20 \times 20 \times 20$, where as many materials experimented on are much larger than this. This issue becomes apparent in the skyrmion model. Usually skyrmions appear in a periodic configuration, as seen in [18]. However, the model only produces them in seemingly random points on the grid. This may be attributed to the fact that the grid of spins is reasonably small, so for periodic configurations of skyrmions to appear, much larger grids are needed. It may be that a solver than is an even more powerful solver is needed to solve larger grid sizes.

9.5 Extension conclusion

It has been demonstrated that the Ipopt solver is capable of modelling more complex physical problems than just a two-dimensional lattice of spins. In terms of solve time, the solver was still considerably fast, the longest it took was around five minutes to solve a $20 \times 20 \times 20$ grid size for $FeBO_3$, this is 16,000 variables. In contrast, the other solvers often took much longer than that to solve the 25×25 grid of 2D spins, which is 625 variables.

10 Further work

Following are some good starting points into future work relating to this project:

- **Extend the 3D model:** There are still many more physics features which could be included in this model.
  Firstly, the current model assumes all of the spins are unit length, which is not always the case, so this could be amended to make some more materials. This will increase the problem size, as the spins would have to be parameterised using three variables, rather than the current two.
  The current model is also a classical, however at the atomic scales, quantum effects often cannot be neglected, so creating the model but using quantum mechanics may give some more experimentally accurate results.
  Entropy of the system would also be interesting to include. This becomes a factor when considering the cases where there are many correct solutions to a problem, such as the spins having no preferred direction in a ferromagnet, as long as they are parallel.

- **Testing the 14 other solvers using the 3D model:** The only reason this wasn't done was due to time constraints, as the code for the other packages is slightly different to Pyomo. A similar test could be done, for example a ferromagnet with varying problem size. It may be that the problems tested were too small for some of the solvers, and they may actually be better at solving larger problems.

- **Test more non-linear solvers compatible with Pyomo:** This may involve paying for one of the commercial solvers, but it would be useful to see whether or not Ipopt's efficiency was due to the algorithm itself being efficient, or whether it is a feature of Pyomo.

- **Investigate other physics problems:** The aim of the project was to see how good these solvers were at dealing with physics problems, so the more physics problems that are tried out, the better of an idea there will be. It might be that some of the solvers that were "bad" at this problem would be a lot better at other physics problems.

11 Links to the code

If you wish to look through the code then: `https://github.com/MNegus/SpinModel2D`

contains the two-dimensional models and: https://github.com/MNegus/SpinModel3D contains the three-dimensional code.

Both links contain the documentation for the code.

References

[1] *AMPL FOR BUSINESS.* http://ampl.com/products/ampl/ampl-for-business/

[2] G. S. Joyce. *Classical Heisenberg Model* (1967). Physical Review, Volume 155, Number 2.

[3] *AMPL.* http://ampl.com/products/ampl/

[4] *GAMS.* https://www.gams.com/

[5] *Pyomo website.* https://www.pyomo.org/

[6] W. E. Hart, C. Laird, J. -P. Watson, D. L. Woodruff. *Pyomo Optimization Modeling in Python.* Vol. 67. Springer, 2012.

[7] W. E. Hart, J. -P. Watson, D. L. Woodruff. *Pyomo: modeling and solving mathematical programs in Python.* Mathematical Programming Computation 3, no. 3 (2011): 219-260.

[8] *The 2016 Top Programming Languages.* http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

[9] *AIMMS.* http://aimms.com/

[10] *Ipopt.* https://projects.coin-or.org/Ipopt

[11] A. Wätcher. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering* (2002). http://researcher.watson.ibm.com/researcher/files/us-andreasw/thesis.pdf

[12] *SciPy.* https://www.scipy.org/

[13] *SciPy optimise.* http://docs.scipy.org/doc/scipy/reference/optimize.html

[14] *pyOpt website.* http://www.pyopt.org/

[15] R. E. Perez, P. W. Jansen, J. R. R. A. Martins. *pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization.* Structures and Multidisciplinary Optimization, 45(1):101-118. 2012.

[16] *pyOpt optimisers.* http://www.pyopt.org/reference/optimizers.html

[17] *matplotlib.* http://matplotlib.org/

[18] A. Roldán-Molina, A. S. Nunez, J. Fernández-Rossier. *Topological spin waves in the atomic-scale magnetic skyrmion crystal* (2015). Page 3. https://arxiv.org/abs/1511.08244

[19] V. E. Dmitrienko, E. N. Ovchinnikova, S. P. Collins, G. Nisbet and G. Beutier. *An X-ray study of the Dzyaloshinskii-Moriya interaction in the weak ferromagnet* $FeBO_3$ (2013). Journal of Physics: Conference Series 519.

[20] V. E. Dmitrienko, E. N. Ovchinnikova, S. P. Collins, G. Nisbet, G. Beutier, Y. O. Kvashnin, V. V. Mazurenko, A. I. Lichtenstein and M. I. Katsnelson. *Measuring the DzyaloshinskiiMoriya interaction in a weak ferromagnet* (2014). Nature Physics.