Ph 121c
Assignment 1
Lorenzo Van Munoz
April 21, 2021

Contents

1 Dense ED

1.1 Discussion

The Transverse Field Ising Model (TFIM) for a spin chain of length L with parameter h (and zero indexed) is given by

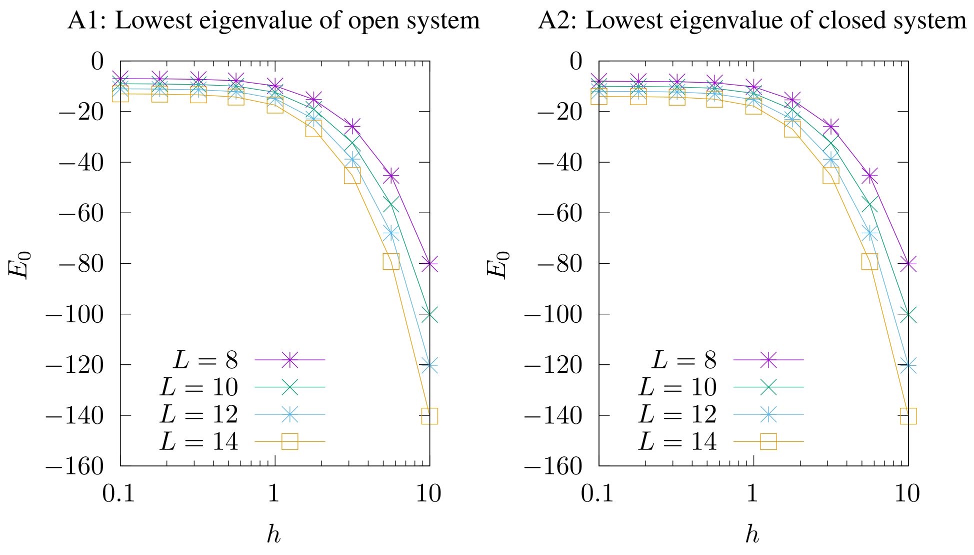
$$\hat{H} = -\sum_{j=0}^{L-2} \hat{\sigma}_j^z \hat{\sigma}_{j+1}^z - \left(\hat{\sigma}_{L-1}^z \hat{\sigma}_0^z\right) - h \sum_{j=0}^{L-1} \hat{\sigma}_j^x. \tag{1}$$

I wrote two algorithms for initializing the dense Hamiltonian corresponding to the TFIM: a direct construction by taking tensor products (implied in $\hat{H}$ above) and a direct construction for any vector by representing the Pauli operators as bitwise operations on a binary representation of the computational basis.

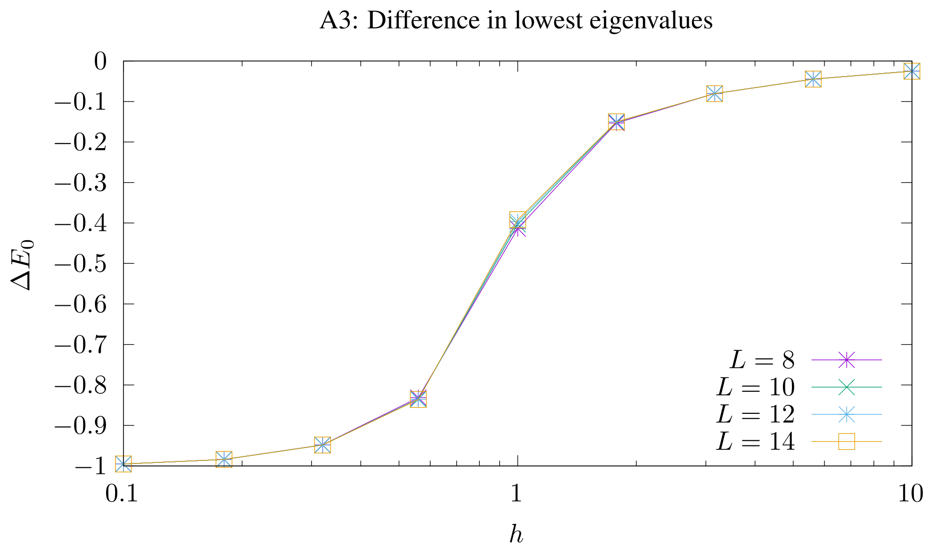
Since the latter is less prone to memory allocation errors (unlike the former, it does not need to repeatedly sum large matrices and does not use a recursive implementation) I was more comfortable using it.

1.2 Results

After generating the dense Hamiltonian, I diagonalized it with the Fortran LAPACK95 routine `syev` to obtain the spectrum of the Hamiltonian. For this problem, we are just interested in the ground state energies as a function of the parameters L and h.



As a function of L, it appears that in figures A1 and A2, larger values of L have lower ground state energies that get increasingly lower as h increases. This is because in the large h regime, the system favors aligning with the transverse applied field, and since those terms in the Hamiltonian act on each site, the more site there are, the lower energy can be achieved. In addition, the logarithmic scale on the horizontal axis shows a shoulder around $h \approx 1$, with the ground state energy approximately constant at $h < 1$ and rapidly decreasing for $h > 1$. And I don't know why they look like low-pass filters – maybe that is just me.



This figure shows how much lower in energy the ground state of the closed system is compared to that of the open system. Notably, the energy difference has a sigmoid shape as a function of h, with the inflection point at about the critical value of $h = 1$. Thus, the different boundary conditions of the system have a substantial difference on the ground state energy in the $h < 1$ phase, but this becomes negligible for $h >> 1$ due to the interaction with the transverse field. Also, the energy gap is nearly identical as a function of system size, due to the fact the main difference is due to the additional term in the Hamiltonian with closed boundaries.

2 Sparse ED

2.1 Discussion

Using intel MKL, I would have two routines for solving sparse eigenvalue problems for real matrices: `?feast_scsrev` for finding parts of a spectra, and `mkl_sparse_?_ev` for finding the extremal eigenvalues. The former routine requires a 3-array CSR format, and the latter requires the MKL Sparse BLAS CSR format, which is a 4-array format. I will probably implement a function to do that later in the course, because for this set I wrote my own Lanczos routine for finding extremal eigenvalues, and it works beautifully with impressive speed. Therefore, I circumvented the use sparse array formats because the algorithm only needs a function that can multiply a vector by the Hamiltonian. So I probably shouldn't call this section "Sparse ED".

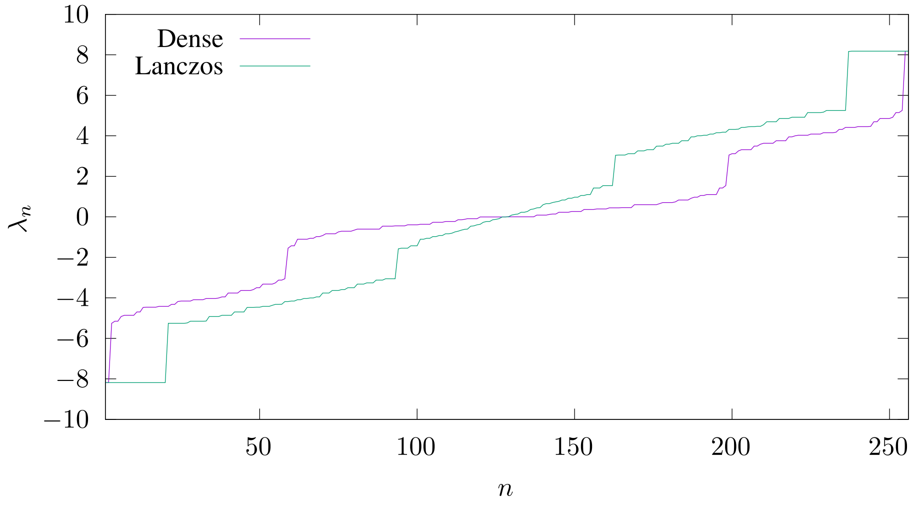
I implemented the Lanczos routine as described on Wikipedia. I am comfortably able to push my Lanczos implementation to sizes of $L = 24$ for $m = 32$ eigenvalues, but I won't risk more since each run already takes a minute at that level. In some cases I knew I hit a computational limit when I tried to run the program and experienced a segmentation fault error (some of these can be avoided by increasing the stack limit). In other cases the compiler simply failed to compile due to allocation overflows.

Returning to the question of implementing sparse format at a later date, I will be faced with two very different implementations that mirror two very different implementations I made for the dense case. If I go to tensor product route, I will have to find a way to tensor product sparse matrices from scratch, in whatever format is needed. If I go the bitwise route, I will have to find out how to put together the sparse matrix one row at a time In both cases, I will have to decide whether I consider using only a symmetric representation or a full one. In both cases, I will have to find a way to figure out the size of the sparse matrix, ideally without ever creating the dense matrix, which presents a problem with allocating the arrays for the sparse matrix because I don't know their size apriori (hopefully without having to create a new array every time I need to add new elements). Perhaps a linked list would work well to build the matrix, and then convert the linked list into an array. The only thing I know is that the problem becomes increasingly sparse in larger dimensions.

2.2 Results

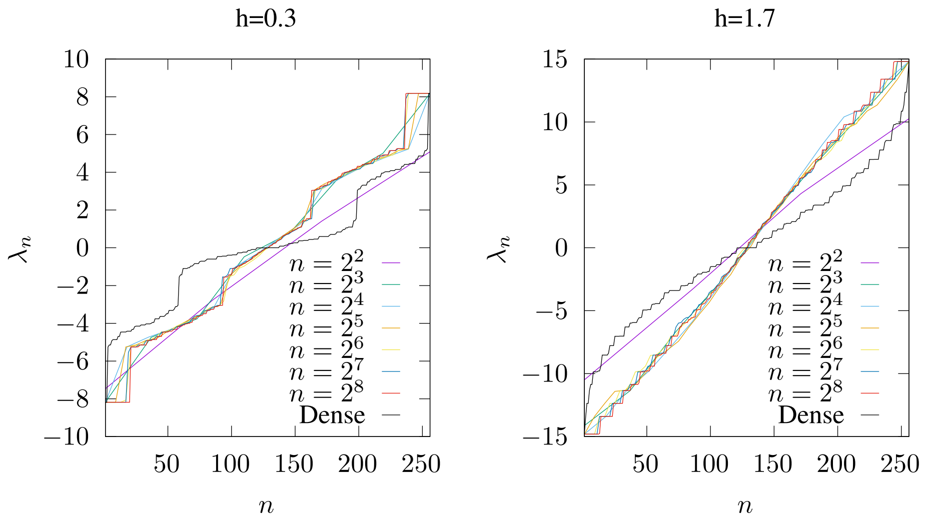
The plot next page opens a can of worms: the Lanczos algorithm has a lot of behaviors going on. A few I can mention are that it get the largest and smallest eigenvalues (and some in the middle, where the lines intersect) accurately. Generally, the Lanczos spectrum is much flatter than the dense spectrum: it looks linear whereas the dense spectrum has curves. Also, the Lanczos spectrum has regions where it plateaus – in particular and the ends of the spectrum. I suspect that in these regions, the algorithm has over-converged: too many eigenvalues were attracted to the extremes (this sounds like a queue for some fun music).

B1: Comparison of numerical spectra for open b.c., $L = 8$, $h = 0.3$



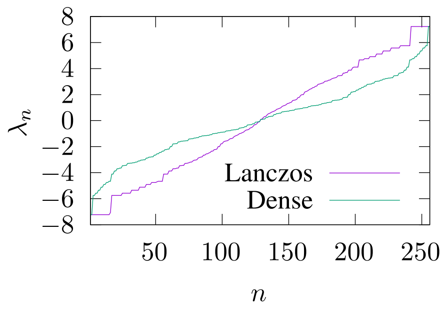
I wanted to study the convergence of the Lanczos algorithm as it relates to the overconvergence issue noticed above. Perhaps by asking the algorithm to find fewer eigenvalues, it doesn't overconverge as much. Figure B2 shows how the Lanczos spectrum evolves as a function of the requested number of eigenvalues (with rescaled x axis). To me it seems like 32 eigenvalues captures all the details the Lanczos spectrum seems capable of, but with less overconvergence.

B2: Lanczos convergence: closed b.c., $L = 8$

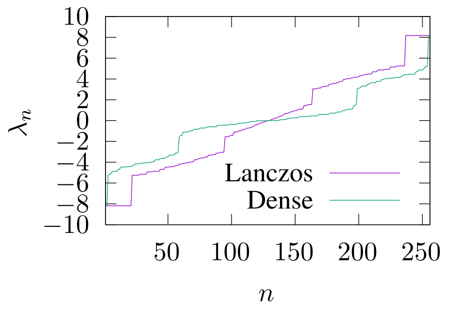


B3: Comparison of numerical spectra

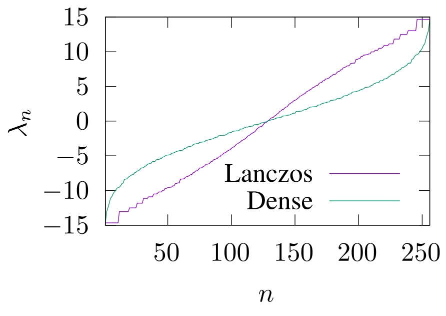
open b.c., L=8, h=0.3



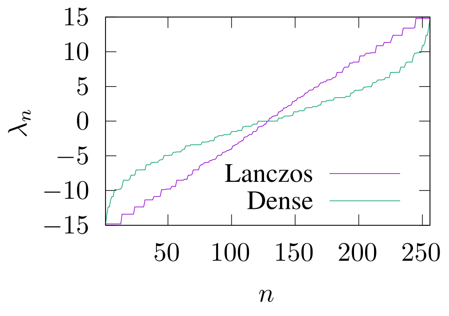
closed b.c., L=8, h=0.3



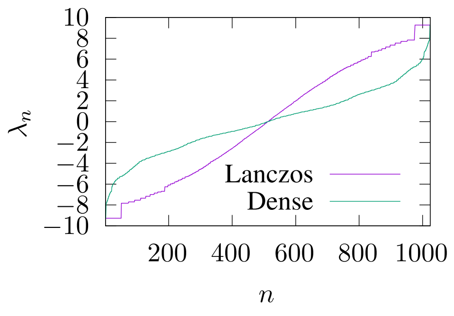
open b.c., L=8, h=1.7



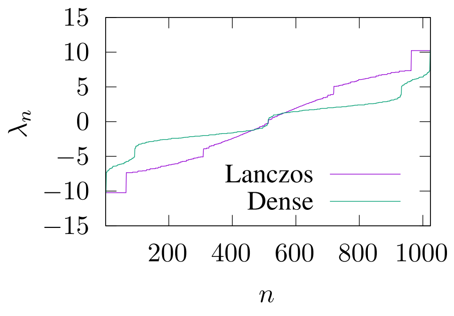
closed b.c., L=8, h=1.7



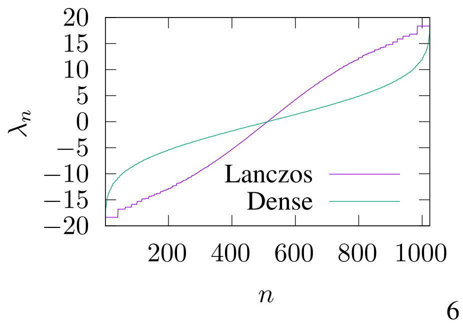
open b.c., L=10, h=0.3



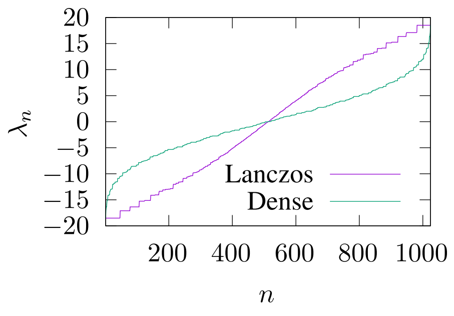
closed b.c., L=10, h=0.3



open b.c., L=10, h=1.7



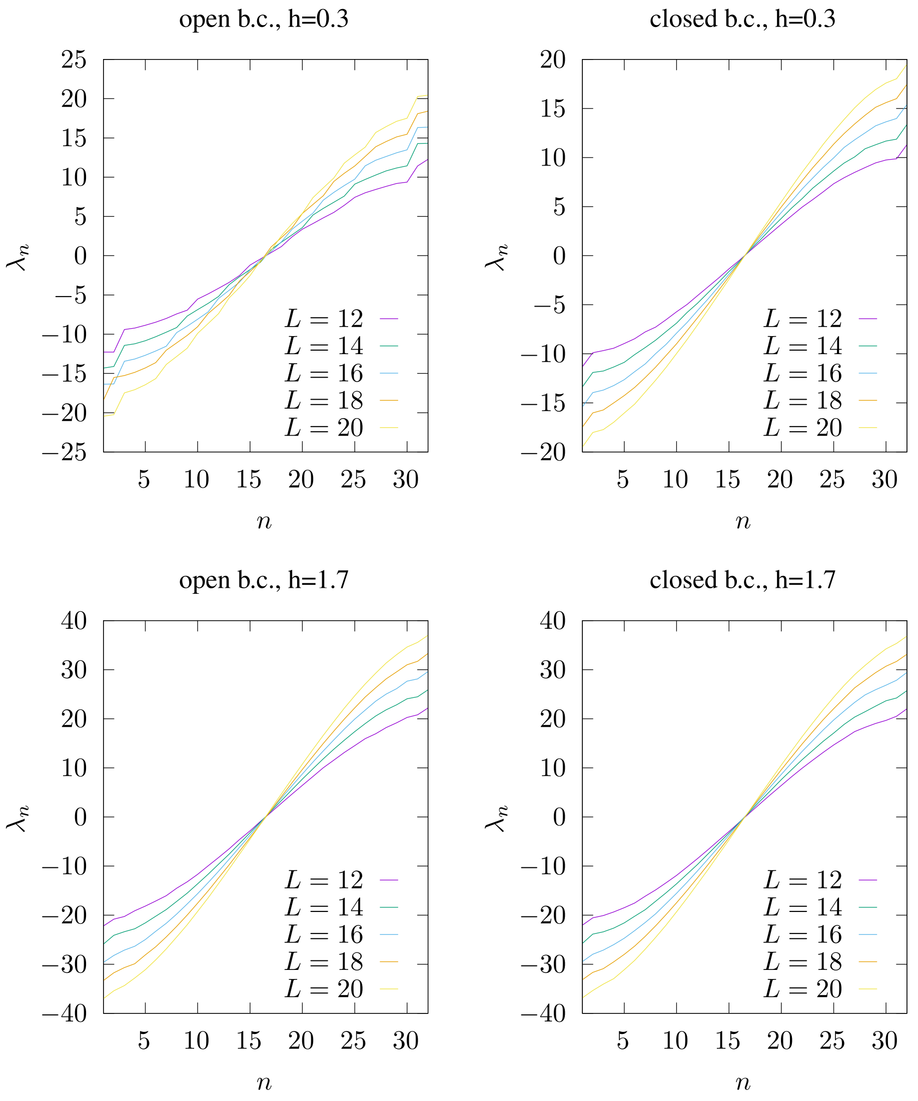
closed b.c., L=10, h=1.7



On the previous page I plotted much more data at different values of h, L, and for each boundary condition. In each case, I asked the Lanczos algorithm for the entire spectrum. All I can add about those plots is that we still see the accuracy of the Lanczos method at the extremal states, but due to overconvergence problems when computing all the eigenvalues we get erroneous spectra.

When I pushed the Lanzos implementation to larger values of L, I intentionally told the solver to compute fewer eigenvalues, only 32. This leads to the following distorted spectra, which nonetheless will obtain accurate ground states.

B4: Lanczos spectra with 32 eigenvalues



3 Convergence with system size

3.1 Discussion

In this section, we study how the energy of the ground state behaves as we increase the system length. This means running a lot of simulations.

3.2 Results

A range of simulations for the ground state energy were conducted with the Lanczos solver and only obtaining 32 eigenvalues. These simulations went to the largest system size I could simulate in a minute, $L = 24$.

C1: Convergence of ground state energy per site

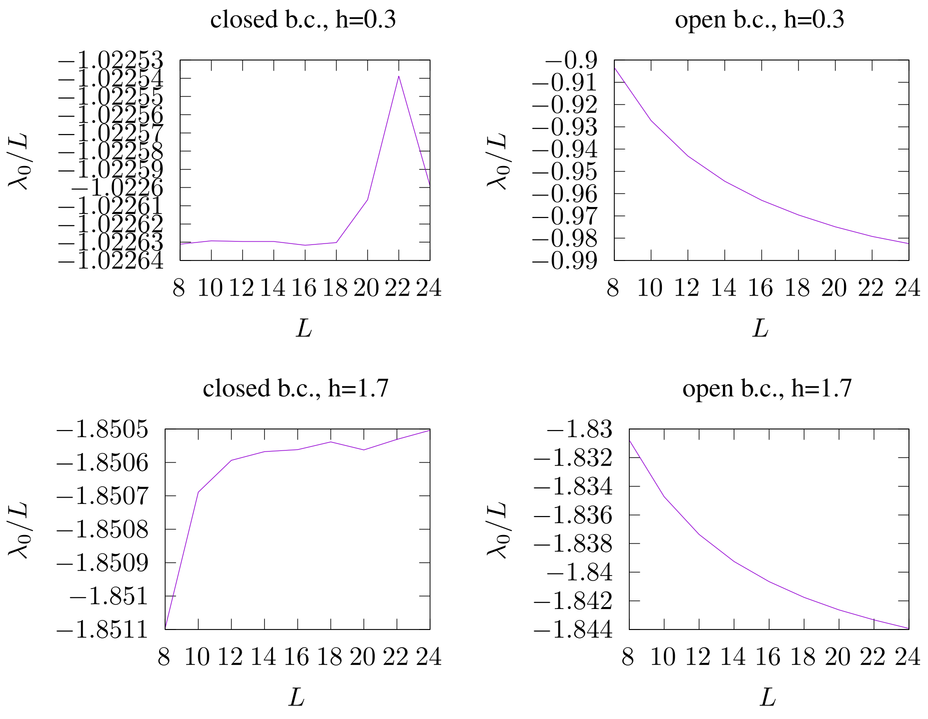
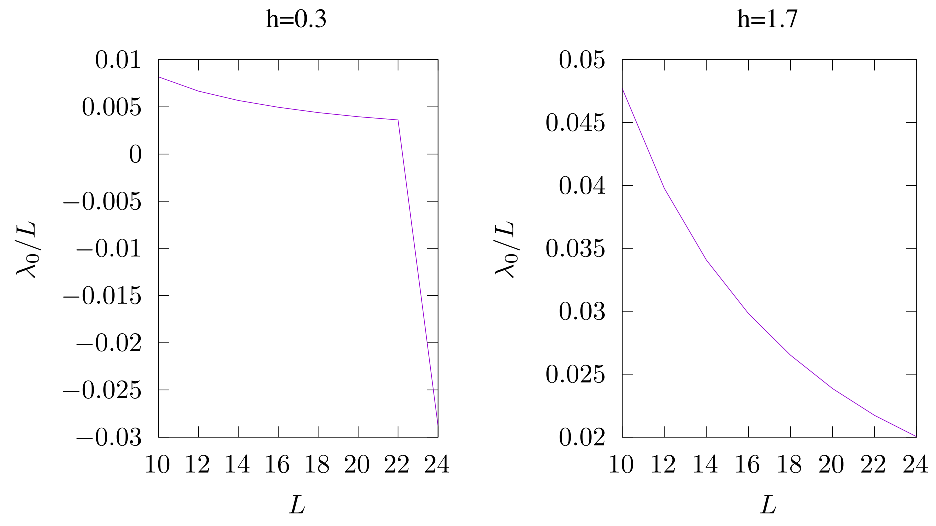


Figure C1 shows the ground state energy per site, λ_0/L as a function of h, L and the boundary conditions. Larger values of h result in lower values of λ_0/L. Open boundary conditions allow for substantially more variation in λ_0/L, though it appears to be decaying at least algebraically. Closed boundary conditions vary in λ_0 about one order of magnitude less than the open boundary conditions, when you compare the decimal digits which appear to vary in each plot. The top left panel shows an unusally large spike at $L = 22$, which may be due to numerical errors at large L while keeping the number of eigenvalues fixed. Despite the spike in the closed systems, their λ_0/L

varies much less than in the open systems, which makes closed systems a better choice to study to avoid finite-size effects.

In figure C2 is plotted the quantity $(\lambda_0(L) - \lambda_0(L-2))/2$ for only systems with open boundary conditions. This represents an additional energy acquired per site at the center of the chain at that size. In general, it is a quantity that decreases with the system size, though there does seem to be a spike that at $L = 24$ that might be due to numerical errors. The decreasing trend indicates the diminishing effect of the open boundaries as the size of the bulk system increases.

C2: Additional ground state energy per center site, open b.c.



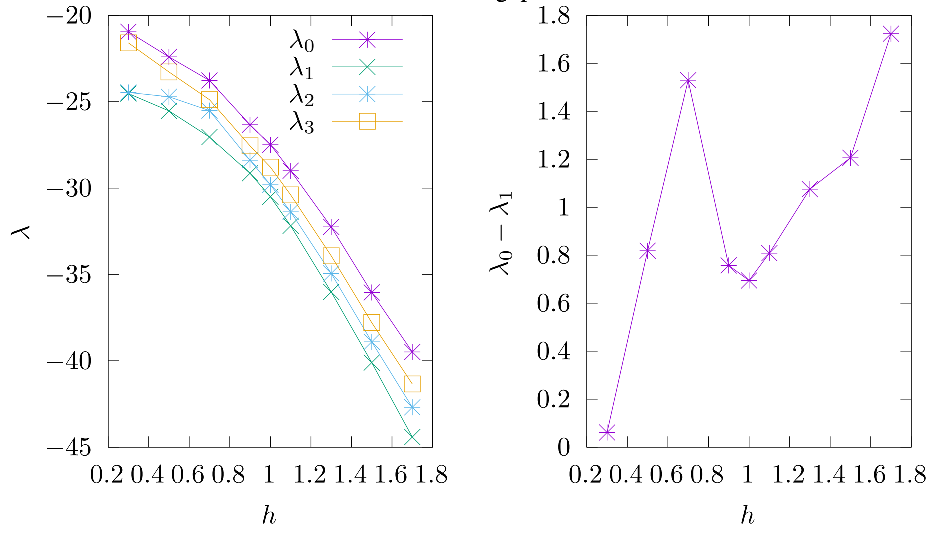
4 Finding the quantum phase transition

4.1 Discussion

Here we are studying the quantum phase transition at the largest system size in terms of the effect on the excitation gap from the ground state.

4.2 Results

D1: Ground state excitation gap, $L = 24$, closed b.c.



D2: Fidelity, dh = 0.01, $L = 24$, closed b.c.

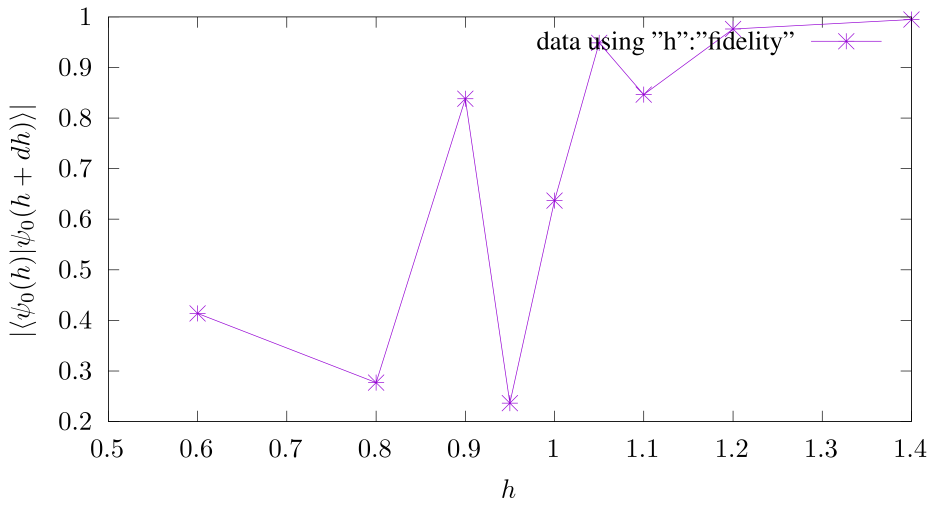


Figure D1, left panel, shows the first four eigenvalues from each Lanczos diagonalization. The leftmost value of h resulted in overconvergence, which is why the gap from the ground state to the first excited state is 0 on the right panel for $h = 0.3$. On the right panel, it appears as though there is a local minimum in the ground state gap at $h = 1$. I am not sure why the gap shrinks for small h, but I would assume that has to do with errors from my implementation fo the Lanczos method. I would like to estimate the exponent ν describing $\lambda_0 - \lambda_1 \propto |h-1|^{\nu}$, however I'm afraid my data just isn't good enough for that. I would probably need a more accurate solver and to scan many more points near the critical value.

In figure D2, I plotted the fidelity of the ground state I obtained with my Lanczos implementation, using a step size of $dh = 0.01$. There appears to be a deep dip at about $h = 0.95$, but there are also dips for smaller h. My Lanczos routine is probably making mistakes in this $h < 1$ region, where we know that the spectrum is steeper based on the dense diagonalization spectra in Figure B3.

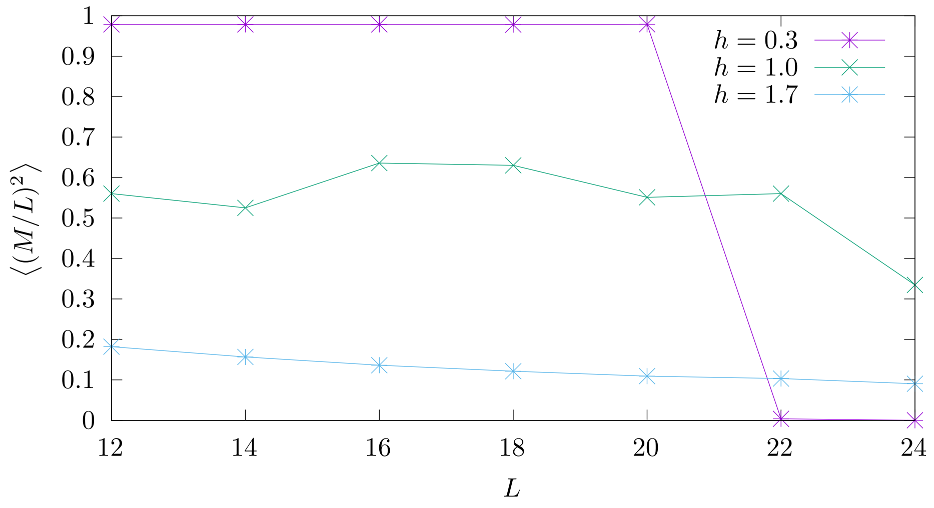
5 Magnetic ordering

5.1 Discussion

In this section we look at the long-range order of spins within the Ising chain in terms of the correlations of the spins orientations.

5.2 Results

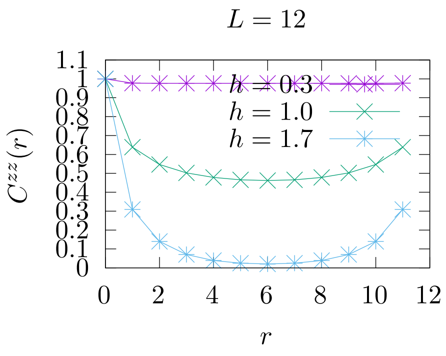
E1: Long-range magnetic order, $L = 24$, closed b.c.

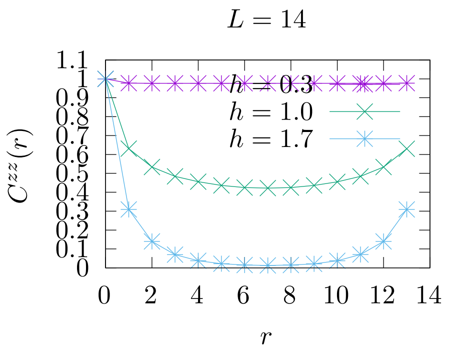


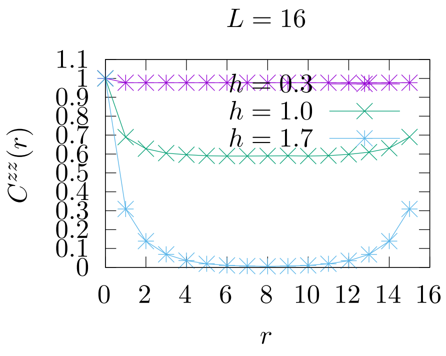
In the plot above is shown the variance of the magnetization per site. Again, my solver seems to have failed at $h = 0.3$ and large L. Ignoring this, for small $h < 1$, it is a constant with respect to L. This is the ferromagnetic phase when the magnet is saturated. For $h = 1$, the magnetization variance wavers or decays slightly. For $h > 1$, the variance of the magnetization per site, decreases slowly, likely due to the alignment of the spins with the transverse field.

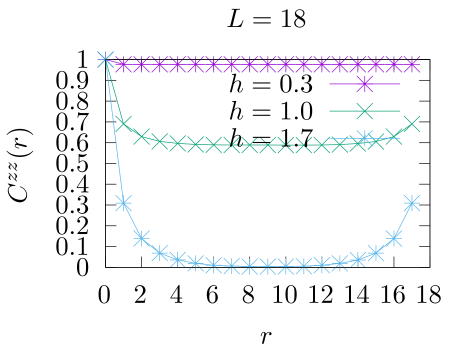
In figure E2, the parwise correlator, $C^{zz}(r) = \langle \sigma_1^z \sigma_{1+r}^z \rangle$ is plotted for various chain lengths and characteristic values of h. For $h < 1$ (except for large L where my implementation falls short), the correlation is nearly 1 across the chain. For $h = 1$, the correlation drops along the length of the chain (to lower values at midpoints of longer chains), but never zero. For $h > 1$, the correlation drops fairly quickly to zero away from the first site on the chain. In summary, for r at the midpoint, the correlation is about 1 for $h < 1$, decreasing with L at the critical point, and 0 for $h > 1$.

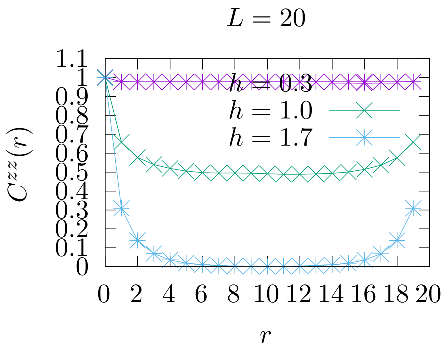
E2: Pairwise correlation, closed b.c.

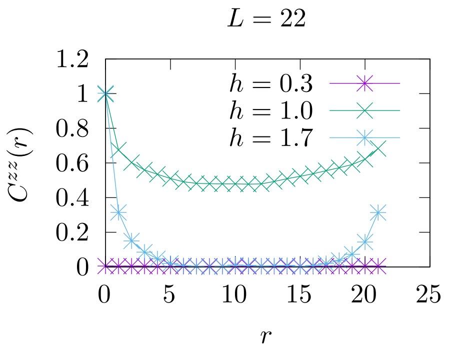


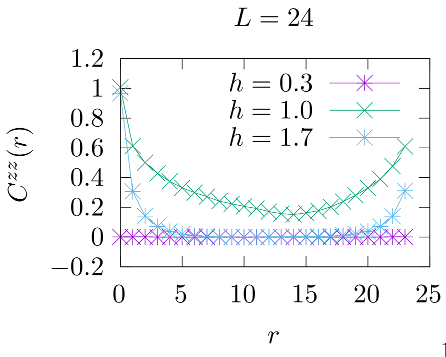












6 Using Ising symmetry

6.1 Discussion

To transfer from the z basis, $\{\left|\uparrow\right\rangle, \left|\downarrow\right\rangle\}$, to the x basis, $\{\left|+\right\rangle, \left|-\right\rangle\}$, use this operator

$$\hat{T} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2}$$

Notably, T is its own inverse. Rotating the Hamiltonian does the following:

$$\begin{aligned}
\hat{H}_z \left|\psi\right\rangle_z \rightarrow & \left(\prod_j \hat{T}_j\right) \hat{H}_z \left|\psi\right\rangle_z & (3)\\
&= \left(\prod_j \hat{T}_j\right) \hat{H}_z \left(\prod_j \hat{T}_j\right)\left(\prod_j \hat{T}_j\right) \left|\psi\right\rangle_z & (4)\\
&= \hat{H}_x \left|\psi\right\rangle_x . & (5)
\end{aligned}$$

The form of H_x is as follows:

$$\begin{aligned}
\hat{H}_x &= \left(\prod_j \hat{T}_j\right) \hat{H}_z \left(\prod_j \hat{T}_j\right) & (6)\\
&= \left(\prod_j \hat{T}_j\right)\left(-\sum_{j=0}^{L-2} \hat{\sigma}_j^z \hat{\sigma}_{j+1}^z - \hat{\sigma}_{L-1}^z \hat{\sigma}_0^z - h\sum_{j=0}^{L-1} \hat{\sigma}_j^x\right)\left(\prod_j \hat{T}_j\right) & (7)\\
&= -\sum_{j=0}^{L-2} \hat{T}_j \hat{\sigma}_j^z \hat{T}_j \hat{T}_{j+1} \hat{\sigma}_{j+1}^z \hat{T}_{j+1} - \hat{T}_{L-1} \hat{\sigma}_{L-1}^z \hat{T}_{L-1} \hat{T}_0 \hat{\sigma}_0^z \hat{T}_0 - h\sum_{j=0}^{L-1} \hat{T}_j \hat{\sigma}_j^x \hat{T}_j & (8)\\
&= -\sum_{j=0}^{L-2} \hat{\sigma}_j^x \hat{\sigma}_{j+1}^x - \hat{\sigma}_{L-1}^x \hat{\sigma}_0^x - h\sum_{j=0}^{L-1} \hat{\sigma}_j^z . & (9)
\end{aligned}$$

This effectively interchanges the role of the sign-flip and spin-flip operators. (Sorry if the notation is confusing: the Hamiltonian is in the x basis but the Pauli matrices still have the same matrix elements as in the z basis)

With this form of the Hamiltonian, we are one permutation away from expressing the disjoint symmetry sectors as a Hamiltonian in block diagonal form, with each block acting on its own parity sector. Recall the parity operator

$$\hat{U}_z = \prod_j \hat{\sigma}_j^x \tag{10}$$

which in the x basis is diagonal:

$$\hat{U}_x = \prod_j \hat{\sigma}_j^z . \tag{11}$$

Notice how if we zero-index the basis states in the x basis by integers, the parity operator is the same as the parity of the bit string of the integer if we take the bit parity of zero to correspond to + eigenstates and the bit parity of one to correspond to - eigenstates. So in order to separate the x basis, we just need an efficient way to flip bit strings that acts as a permutation of $\hat{H}_x$ to transform it into block diagonal form.

Fix L and let $\left|+_j\right\rangle, \left|-_j\right\rangle$ denote the j-th basis element of the x basis in the + and - parity sectors, respectively (Note there are 2^{L-1} states in each sector). Let's zero index these states and find a bit operator map that sends the index j to the kth element of the computational x basis, whose Hamiltonian, $\hat{H}_x$, we know. Note we also need the inverse permutation in order to correctly reassign the action of terms of the Hamiltonian.

One way to do this is to calculate the diagonal entries of $\hat{U}_x$, and to store an array which in the j-th index has the value k. The reverse map is obtained by placing j in the k-th entry of the array which was the diagonal of $\hat{U}_x$ The diagonal of $\hat{U}_x$ can be constructed recursively as follows: let b_j be a bit string of length 2^j, starting with $b_1 = (0, 1)$, and obtain b_{j+1} by concatentation: $b_{j+1} = \left(b_j, -b_j\right)$, where a minus sign denotes a bitwise flip operator.

Note that the kth entry of b_j is the bit string parity of k (zero indexed). The first few bit strings are:

$$b_1 = (0, 1) \tag{12}$$

$$b_2 = (0, 1, 1, 0) \tag{13}$$

$$b_3 = (0, 1, 1, 0, 1, 0, 0, 1) \tag{14}$$

which translates to the following diagonal entries of $\hat{U}_x$:

$$\left(\hat{U}_x\right)_{ii}^{L=1} = (1, -1) \tag{15}$$

$$\left(\hat{U}_x\right)_{ii}^{L=2} = (1, -1, -1, 1) \tag{16}$$

$$\left(\hat{U}_x\right)_{ii}^{L=3} = (1, -1, -1, 1, -1, 1, 1, -1). \tag{17}$$

Thus these indices provide all the bookkeeping necessary to take the symmetry sectors into a block diagonal form.

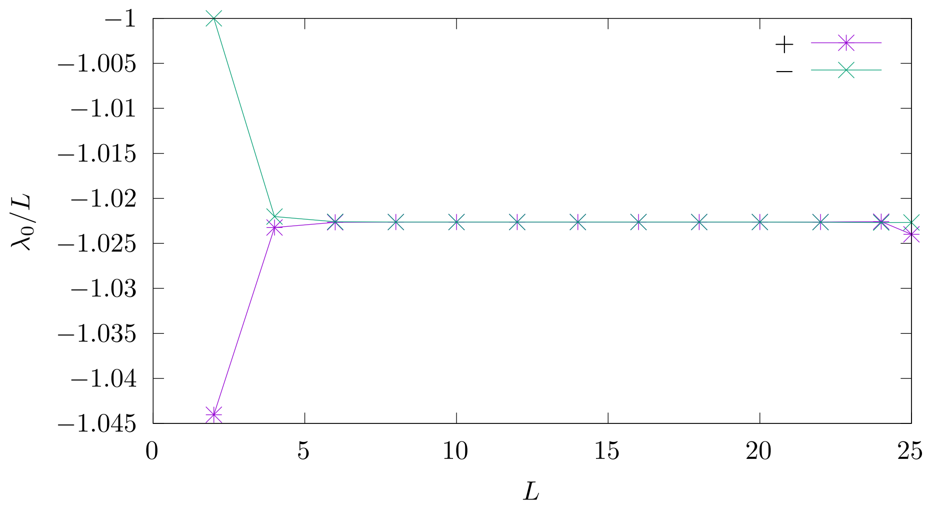
If you study this experimentally, perhaps using the `parity_diag` function I wrote in the `tfim_dense` module, one finds that for states in $\mathcal{H}_+$ symmetry sector, the efficient bit operator that does the bookkeeping from the j-th + parity state to the k-th x basis state is $k = 2*j + \epsilon(j)$ where ϵ is the bit string parity operator ($\epsilon(j) = 0$ if j has an even number of 1's and $\epsilon(j) = 1$ if j has an odd number of 1's). Likewise, for the j-th state in the - sector, it corresponds to $k = 2*j + (\epsilon^{-1}(j)$ (where $\epsilon^{-1}(j) = \epsilon(j+1)$). The inverse operation in both cases is $j = (k - (k\%2))/2$. In Fortran:

```fortran
k = (2*j + poppar(j))           ! from + sector to full basis
k = (2*j + (1 .xor. poppar(j))  ! from - sector to full basis
j = (k - mod(k, 2)) / 2         ! Inverse in both sectors
```

6.2 Results

The first thing to look at is how the values of the ground state energy differ in the different sectors.

F1: Ground state sector energy gap, closed b.c., $h = 0.3$

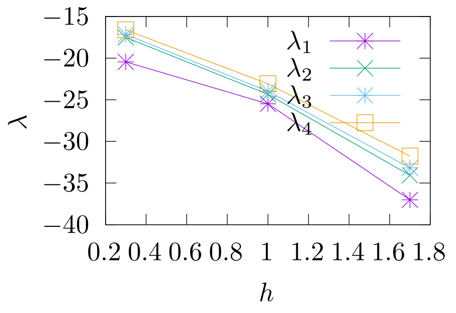


Well indeed, the energy splitting in the ground state, which in the plot is normalized by the system size, appears to decay exponentially with the system size, or at least very quickly. It does also seem like there is some numerical error at the largest system size of $L = 25$.

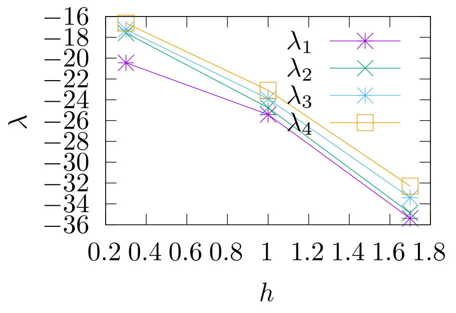
On the following page, figure F2, I used the symmetry sector solver to compute the lowest eigenvalues of larger systems, solving for 32 eigenvalues in each sector. They behave like before, some of the solutions $L \geq 24$ showing overconvergence at the lowest eigenvalue, and for smaller L, there appears to be a dip in the ground state excitation energy at the critical point, because the lines for λ_0 and λ_1 get closer together at $h = 1$. In addition, there appears to be little difference across the two sectors, but it did make it possible to solve for a larger system. Because I told my Lanczos routine to solve for the same number of eigenvalues anyway, I saw none of the promised 4-fold speedup, when combining runtimes from both sectors, because my Lanczos algorithm as implemented only needs to multiply vectors by the Hamiltonian, and that algorithm, when using the symmetry property, only had to do half as many bit operations (1/2 as long a vector), but then there are two sectors. In total, I calculated twice as many eigenvalues in the same amount of time.

F2: Lowest eigenvalues by symmetry sector, closed b.c.

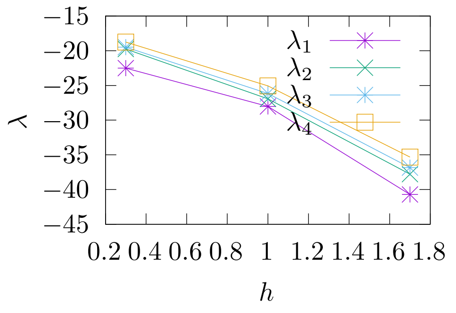
$+$ sector, $L = 20$



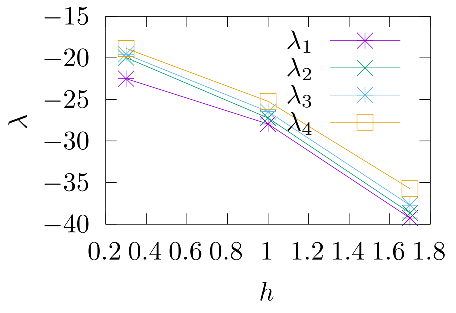
$-$ sector, $L = 20$



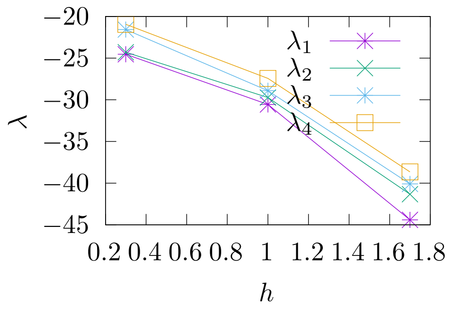
$+$ sector, $L = 22$



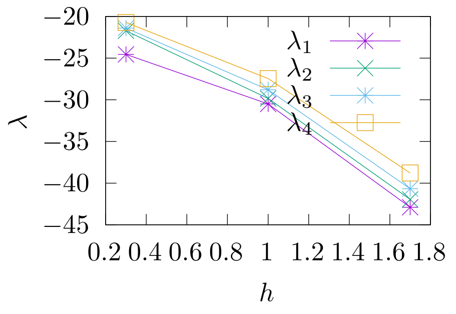
$-$ sector, $L = 22$



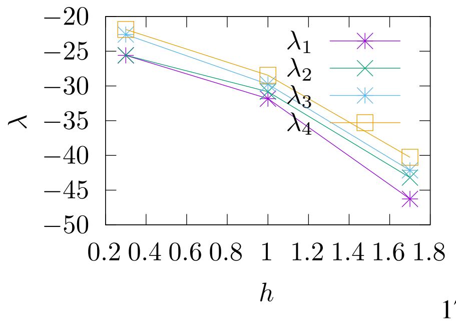
$+$ sector, $L = 24$



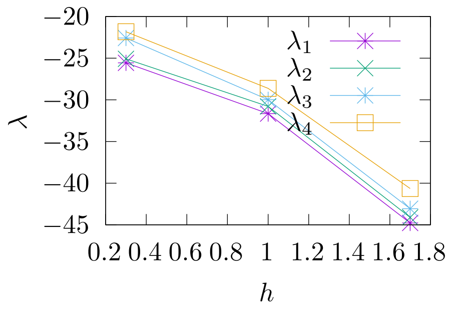
$-$ sector, $L = 24$



$+$ sector, $L = 25$



$-$ sector, $L = 25$



7 Conclusion

This has been an absurd but instructive experience. Absurd because there was probably no concrete benefit for making myself do it this way. Instructive because I learned how to use new software.

Learning Fortran was going to hurt, but I didn't expect it to hurt for so long. I feel like I mainly struggled with how to organize my code into parts that worked well with each other. I'll say that Fortran discourages convenience and complexity but as soon as you know what you want to do, implementation isn't much harder than in other programming languages.

I also spent lots of time reading the intel and MKL documentation. It is so dense that you can't diagonalize the manual to get a wide spectrum of knowledge from it in any short amount of time. It is an excellent resource, in addition to the examples that come installed with the oneAPI packages. Now I know that I could work with it much better in the future since I'll be less ignorant about where to find help and details.

Gnuplot was also a bear to pick up. There are a lot of details you need to get right with any plotting program, and since there's no native support for data frames, things get tricky. I could probably make my programs more elegant by incorporating `grep` to filter out all the lines I want in a data file, but the ease will never be as simple as with using a program like R whose data frames are super straightforward to plot. Or using python, pandas + matplotlib allow for similar operations.

If I could change other things about this assignment, it would be going through the exercise of making sparse matrix formats (as paralyzing as all the choices that need to be made are) and possibly changing the ”Discussion”, ”Results” format into ”Approach”, ”Results”, and ”Discussion” to talk more about the results themselves.

8 Appendix: Code

All of my source code is available by viewing or cloning this git repository. The following sections are a minimal reference about my computer and the project.

8.1 Computing environment

I am using the Intel oneAPI base toolkit (with MKL) and HPC toolkit (with Fortran), which are freely available here. Here is information about my computer and software:

```
$ uname -a
Linux kaos 5.10.0-5-amd64 #1 SMP Debian 5.10.26-1 (2021-03-27) x86_64 GNU/Linux
$ dpkg-query --show intel*kit gnuplot* | grep [0-9]$
gnuplot-data    5.4.1+dfsg1-1
gnuplot-nox     5.4.1+dfsg1-1
intel-basekit   2021.2.0-2883
intel-hpckit    2021.21.0-2997
```

This pdf document was generated by latexmk using TEX Live 2020.

8.2 Building this project

Please refer to the README's in the repository on how to compile the code and generate the data presented in this document.