CVM Documentation

Charles Yang

Last updated: August 25, 2020

1 Source Code:

The source code can be pulled from the github repo:
https://github.com/daedalus1235/CVM.git
It is written in python 3.8, and requires the SciPy, NumPy, and MatPlotLib libraries to be installed.

2 Units

Throughout the documentation and code, the value of Boltzmann's constant is taken to be 1.

Part I

Numerical Optimization

3 System

The system being examined is the 2D square lattice Ising Model. The two species are consequentally the (+) species and the (−) species.

4 Variables

The coordination number is taken to be $Z = 4$.

The most important variables are the composition variable, x_α, for species α, the bond variable, $y_{\alpha\beta}$, for the bond with endpoints α and β, and the square variable $z_{\alpha\beta\gamma\delta}$ for the square with vertices α, β, γ, and δ. The sum over any two adjacent indices (α and δ are considered adjacent) of the $z_{\alpha\beta\gamma\delta}$ variable gives the appropriate y variable for the non-summed indices, and the sum over any single index of the $y_{\alpha\beta}$ variable gives the appropriate x variable for the non-summed index. These variables are represented differently in each of the programs, depending on the required depth of the variable.

In the point approximation, the composition variables are represented $x_+ = x$ and $x_- = 1 - x$.

In the bond approximation, the composition variables are represented $x_+ = x$ and $x_- = 1 - x$, with the bond variables given $y_{++} = x - y$, $y_{+-} = y_{-+} = y$, and $y_{--} = 1 - x - y$.

In the square approximation, the base variable is given to be the $z_{\alpha\beta\gamma\delta}$ 4D matrix. However, due to symmetries, the matrix is flattened into a 1D array with an associated multiplicity array, and the $y_{\alpha\beta}$ and x_α are derived with corresponding multiplicity arrays as needed. The flattened base variable is given in the order:

$$z = [\{++++\}, \quad \{+++-\}, \quad \{++--\}, \quad \{+-+-\}, \quad \{+---\}, \quad \{----\}]$$

with associated multiplicity

$$c = [1, 4, 4, 2, 4, 1]$$

Similarly, the y variable is flattened as

$$y = [\{++\}, \quad \{+-\}, \quad \{--\}]$$

with multiplicity

$$b = [1, 2, 1]$$

and x

$$x = [\{+\}, \quad \{-\}]$$

with multiplicity

$$a = [1, 1]$$

The helper functions `Y(z)`and `X(z)`derive the y and x arrays from the z array.

The multiplicities are used to calculate the entropies; for example, the square entropy S_z is calculated as

$$S_z = \sum_i c_i z_i \ln(z_i)$$

Similar expressions are used for the a/x pair and the b/y pair of arrays. Note that S_z is *not* the total entropy of the square approximation; it is the information entropy of the z variable.

5 Free Energy Expression

The Hamiltonian is taken to be the Ising Hamiltonian, given:

$$H = -J \cdot \frac{Z}{2} \cdot (y_{+-} + y_{-+} - y_{++} - y_{--}) - h \cdot x_{+} - x_{-}$$

For the point model, which does not have bond variables, this expression is taken to be

$$-J \cdot \frac{Z}{2} \cdot (x_{+}x_{-} + x_{-}x_{+} - x_{+}x_{+} - x_{-}x_{-}) - h \cdot x_{+} - x_{-}$$

Normally, the field variable h is taken to be zero.

The entropy is given by the appropriate CVM approximation. A function xlx is defined to be $xlx(\alpha) = \alpha \ln \alpha$ to simplify notation. For the point approximation, the entropy is given:

$$S_{point} = S_x = -\sum_{\alpha} \mathrm{xlx}(x_\alpha)$$

For the bond approximation, the entropy is

$$S_{bond} = 2S_y - 3S_x = -2\sum_{\alpha\beta} \mathrm{xlx}(y_{\alpha\beta}) + 3\sum_{\alpha} \mathrm{xlx}(x_\alpha)$$

and for the square approximation, the entropy is

$$S_{square} = S_x - 2S_y + S_z = -\sum_{i} a_i \,\mathrm{xlx}(x_i) + 2\sum_{j} b_j \,\mathrm{xlx}(y_j) - \sum_{k} c_k \,\mathrm{xlx}(z_k)$$

Finally, the free energy is given as

$$F = H - TS$$

.

6 Minimization

The scipy.optimize libraries are used to find the minimum free energies and corresponding optimal compositions. The `trust-constr`minimization algorithm is used to allow for the use of composition constraints. These constraints include the total probability constraint $\sum_\alpha x_\alpha = 1$, and the constraint that physical solutions have positive variable values.

The optimal composition is computed for a range of temperatures. The first guess (required for the optimization method) is taken to be pure species $(+)$, and subsequent guesses are the result of the previous temperature's minimization.

7 Running Code

The python scripts are located in the `/CVM/binary/2D/`directory, and are labelled `point.py`, `bond.py`, and `square.py`. The main method for each script is the `min()`function. The `min()`function takes four variables: interaction strength `J`, reduced external field `hpj`, number of partitions of the temperature range `samp`, and the reduced temperature range `Trang=[Tmin,Tmax]`. The reduced variables `hpj` and `Trang` are values per strength `J`; for example, the field strength is given $h = hpj \cdot |J|$. The default values are

Calling the function `min(J=-2,hpj=0,samp=50,Trang[0,5])`, for example, computes the minimum free energy for an interaction strength `J=-2`, no external field, and at 51 samples along the temperature range $[0, 10]$[1]. The discrepancy between the variable `samp`and the number of samples taken was meant so that nice numbers could be chosen for `samp`and result in nice sample temperatures: `samp`=5 for `Trang`=[0,5] would then result in the samples [0, 1, 2, 3, 4, 5].

The output of the program is a set of 4 graphs, all with respect to the reduced temperature. The first graph is the free energy, the second is the composition (x,y) or (x), the third is the energy $E = F - T \cdot \frac{\mathrm{d}F}{\mathrm{d}T}$, and the fourth is the heat capacity $C = T \cdot \frac{\mathrm{d}^2F}{\mathrm{d}T^2}$. Also output is a `.csv`file containing the reduced temperature T in the first column and the optimal composition (x) in the second column. Finally, the slope of the free energy and extrapolated intercept are calculated from the top 25% of the temperature range and output to console.

[1]For the square model, only 49 samples are taken; this is because calculating at $T = 0$ is problematic.

Part II

Natural Iteration Method

8 Variables

8.1 Composition and Bond variables

The variables for 2D bond model are defined as follows. The most important variable is the y_{ij} matrix of bond fractions. It is defined such that the first index represents the composition of the species on the primary sublattice (denoted e, for even), and the second on the secondary lattice (denoted o, for odd). Because the compositions of the sublattices are, in general, not the same, y_{ij} forms an asymmetric matrix. Additionally, the sum over any index should result in the composition vector for the other index:

$$x_i^{(e)} = \sum_j y_{ij}^{(eo)} \qquad x_j^{(o)} = \sum_i y_{ij}^{(eo)}$$

where the superscripts on the y variable are to clarify the sublattice. Further, the total composition vector is given:

$$X_i = \sum_i x_i^{(e)} + x_i^{(o)}$$

and is constrained by

$$1 = \sum_i X_i \tag{1}$$

Define a function $Xe : y \mapsto x^{(e)}$. This function is linear. Further, it easily follows that the corresponding function $Xo : y \mapsto x^{(o)}$ is simply $Xo(y) = Xe(y^\intercal)$. This implies that the variable $y^\intercal$ is simply the bond distribution with respect to the odd sublattice. Further, the normality constraint 1, combined with the linearity of the Xe function shows that

$$1 = \sum_{ij} y_{ij} + y_{ij}^\intercal$$

This implies that a total bond matrix can be written, much in the same way as the total composition vector, as

$$Y = y + y^\intercal = y^{(eo)} + y^{(oe)}$$

Such a matrix is symmetric, which is expected of the overall bond distribution. This matrix also has the desired properties of a cluster variable:

$$X_i = \sum_j Y_{ij} = \sum_j Y_{ji}$$

$$1 = \sum_{ij} Y_{ij}$$

8.2 Free Energy Expression

The variable Z is used to denote coordination number.

8.2.1 Hamiltonian

The Hamiltonian is denoted

$$H = \frac{Z}{2}\sum_{ij} E_{ij}Y_{ij}$$

This can be rewritten as:

$$H = \frac{Z}{2}\sum_{ij}(E_{ij} + E_{ji})y_{ij} = -Z\sum_{ij} E_{ij}y_{ij}$$

where the factor of one-half is effectively absorbed into the total bond matrix to become the even bond matrix. This is calculated as the expression

$$H = Z \cdot \mathrm{dot}(E, y)$$

8.2.2 Entropy

The entropy in the bond approximation is given:

$$(1 - Z)S_x + \left(\frac{Z}{2}\right) S_y$$

where S_x and S_y are being used to denote the entropy of the subscript variables:

$$S_\alpha = -\sum_i \mathrm{xlx}(\alpha_i)$$

These entropies are given:

$$\begin{aligned}
S_x &= -\frac{1}{2}\sum_i \mathrm{xlx}(2x_i^{(e)}) + \mathrm{xlx}(2x_i^{(o)}) \\
S_y &= -\frac{1}{2}\sum_{ij} \mathrm{xlx}(2y_{ij}) + \mathrm{xlx}(2y_{ij}^{\intercal}) \\
&= -\frac{1}{2}\sum_{ij} \mathrm{xlx}(2y_{ij}) + \mathrm{xlx}(2y_{ji}) \\
&= -\sum_{ij} \mathrm{xlx}(2y_{ij})
\end{aligned}$$

where the prefactor of one-half is to serve as an average between the two sublattices, and the inner factor of two is to normalize the probabilities.

Thus, the entropy becomes:

$$S = \frac{Z-1}{2}\sum_{is} \mathrm{xlx}(2x_i^{(s)}) - \frac{Z}{2}\sum_{ij} \mathrm{xlx}(2y_{ij})$$

8.2.3 Free energy

The free energy is in the form

$$F = H - TS$$

The grand potential,

$$\Phi = F - \sum_i \mu_i X_i$$

is not used, as the chemical potential terms can be written as the linear constraint $g_k = X_i(y)$ with Lagrange multiplier $\ln(\lambda_k) = \mu_i$.

9 Minimization

9.1 Composition Iteration

The iteration is derived by using Lagrange multipliers (changing to the Karush-Kuhn-Tucker Conditions might be useful in enforcing positive values for the variables) with the linear constraints $0 = g_k(y) = a_k - \sum_{ij} c_{ij}^{(k)} y_{ij}$.

$$\begin{aligned}
\mathcal{L} &= F - \sum_k \ln(\lambda_k) g_k \\
&= H - TS - \sum_k \ln(\lambda_k) g_k
\end{aligned}$$

$$\begin{aligned}
\mathcal{L} &= \sum_{ij} Z E_{ij} y_{ij} + \frac{T}{2}\left[(1-Z)\sum_{is} \mathrm{xlx}(2x_i^{(s)}) + Z\sum_{ij}\mathrm{xlx}(2y_{ij})\right] - \sum_k \ln(\lambda_k)\sum_{ij} c_{ij}^{(k)} y_{ij} \\
\nabla\mathcal{L}_{ij} &= Z E_{ij} + \frac{T}{2}\left[(1-Z)\cdot 2\cdot(\ln(2x_i^{(e)}) + \ln(2x_j^{(o)}) + 2) + Z\cdot 2\cdot(\ln(2y_{ij})+1)\right] - \sum_k \ln(\lambda_k) c_{ij}^{(k)} \\
0 &= \frac{Z E_{ij}}{T} + \left[-(Z-1)(\ln(x_i^{(e)} x_j^{(o)}) + 2\ln(2) + 2) + Z(\ln(y_{ij}) + \ln(2) + 1)\right] - \sum_k \frac{\ln(\lambda_k) c_{ij}^{(k)}}{T} \\
\ln(y_{ij}) &= -\frac{E_{ij}}{T} + \frac{Z-1}{Z}\ln(x_i^{(e)} x_j^{(o)}) + \sum_k \frac{\ln(\lambda_k) c_{ij}^{(k)}}{ZT} \qquad \text{(2a)} \\
\hat{y}_{ij} &= e^{-E_{ij}/T}\cdot\left[x_i^{(e)} x_j^{(o)}\right]^{\frac{Z-1}{Z}}\cdot\prod_k \lambda_k^{c_{ij}^{(k)}/ZT} \qquad \text{(2b)}
\end{aligned}$$

Or, in terms of the python helper functions,

$$\hat{y}_{ij} = e^{-E[i][j]/T}\cdot[Xe(y)[i]\cdot Xo(y)[j]]^{\frac{Z-1}{Z}}\cdot\prod_k \lambda_k^{c^{(k)}[i][j]/ZT} \qquad \text{(2c)}$$

After the additive constant in line 5 is absorbed into the normalization constraint, the form of equation 2c is in the same form as Kikuchi's Natural Iteration Method (NIM). When the only constraint is the normalization of probabilities, only the first two factors (the Boltzmann and the composition factors) need be computed; the multiplicative constant can be determined by direct normalization.

The y variable is then iterated until the difference between the updated value and the previous value is less than `Y_PRECISION` , or it has been iterated `MAX_ITER` times, whichever comes first. The iteration, with the exception of the application of lagrange multipliers, is written to be as general as possible, so should be easily extensible to quaternary or even n-ary compositions, with appropriately dimensioned `Eband` `guess`. Recall the chemical potentials are encoded as Lagrange-multiplier-esque objects. These Lagrange multiplier constraints are calculated by "dotting" (dot product of the corresponding flattened arrays) the coefficient matrix with the bond distribution matrix y. These coefficient matrices are currently hard-coded as global variables, but can easily be modified to be passed as a parameter in an $n \times 2$ matrix of chemical potential/coefficient matrix pairs.

9.2 Multipliers

At each step of the iteration, the Lagrange multipliers need to be updated. One way is to make a lambda function of the free energy.

$$\begin{aligned} \tilde{y}_{ij} &\equiv e^{-E_{ij}/T} \cdot [Xe_i(y) \cdot Xo_j(y)]^{\frac{Z-1}{Z}} \\ \Pi(\overrightarrow{\lambda}) &\equiv \prod_k \lambda_k^{c_{ij}^{(k)}/ZT} \\ \hat{y}_{ij} &= \tilde{y}_{ij} \cdot \Pi(\overrightarrow{\lambda}) \\ F(\overrightarrow{\lambda}) &\equiv F(\tilde{y} \cdot \Pi(\overrightarrow{\lambda})) \end{aligned} \tag{3}$$

This function can then be minimized using prepackaged optimization techniques. However, at low temperatures, it is difficult to find a minimum, as the free energy surface is likely to be everywhere almost-flat.

Instead, the multipliers (more specifically, the chemical potentials) are taken to be fixed, and the y value is updated using equation 2c, then normalized to a total sum of 0.5. The multipliers are actually given on a logarithmic scale, with mX corresponding to a chemical potential of $\mu_X = \ln(mX)$

10 Phase Diagram

The point of the phase transition is taken to be when the difference between the A composition of the two sublattices is less than `xA_TOL`. The transition temperature is obtained for a given chemical potential μ_A and fixed μ_B, μ_C via a recursive binary search. The search range is chosen such that the lower end has $\Delta x_A >$ `xA_TOL` and the upper end $\Delta x_A \leq$ `xA_TOL` . The midpoint temperature is then taken to be the new upper (lower) bound, depending on the value of Δx_A at that point, updating the temperature range. This is repeated until the width of the temperature range is less than `T_PRECISION`. The transition temperature is then obtained for a range of $\mu_A = \ln(mA)$ to produce a graph.

11 Running Code

The relevant python script is located at `/CVM/ternary/2D/bondIt.py`.

11.1 Temperature Dependence

The temperature dependence of the composition, free energy, energy, and heat capacity can be determined similarly to the Ising model programs via the `min()` function. The parameters that are passed are the matrix of bond energies `Eb`, the temperature range `Trang=[Tmin,Tmax]`, the starting guess for the iteration `guess`, and the array of chemical potentials, `m`. The graphs that are printed are the sublattice compositions, the overall composition, the free energy, the energy, and the heat capacity, all with respect to temperature.

11.2 Phase Diagram

The phase diagram for a fixed (μ_B, μ_C) pair can be generated via the `phase()`function. The parameters that are passed are the bond energy matrix `Eb`, the chemical potential pair `mBC=[mB,mC]`, the temperature range to search `Trang`, the μ_A range to search `mrang=[mA_min,mA_max]`, the number of samples of μ_A to be taken `mnum`, and the starting guess for the iteration `guess`. The result of this program is a plot of the transition temperature T_c with respect to the chemical potential $\mu_A = \ln(mA)$. This graph is also output as `Tc_v_mA_mBXX_mCYY.csv` where XX (YY) represents the fixed $\mu_B = \ln(X.X)$ $(\mu_C = \ln(Y.Y))$. If only one digit is listed, the value is given $\mu_B = \ln(0.X)$