Thank you for the introduction!

The Simulation of Quantum-Many body systems is a very challenging problem. A general many-body wavefunction that describes a system with N subsystems can be written as follows. We sum over basis states, which are made up of the local basis states of the subsystems, for example subsystem 1 can be in d\_1 states, and so this sum goes up to d\_1. The amplitude of each basis vector is stored in this complex number Psi. We can now count how many parameters we need to describe this state and it turns out to be d^N, which is of course exponential in system size. Because of this exponential scaling it is very hard to do anything with the wave function, and thus we need approximations. 1min

This brings me to the contents of my talk. I will start by introducing Matrix Product States (MPS), which have great success in tackling this problem in one-dimensional systems.

Next, I will talk about isometric Tensor Product States (isoTPS), which generalize the canonical form of MPS to two and higher dimensions. These isoTPS were introduced in 2019.

In the third and biggest section of my talk I will discuss the work of my thesis, which was implementing an alternative canonical form for isoTPS, which we call YB-isoTPS for reasons I will talk about later. I will discuss how this canonical form is defined and how certain algorithms can be implemented. I will then show benchmarks of this method on the Transverse Field Ising Model. 2min

Before I start with MPS, I want to give a quick general introduction to tensor networks. We define a tensor of rank n to be an n-dimensional array of complex numbers T. It is convenient to introduce a diagrammatic notation. In this notation, we draw tensors as shapes, and indices as lines emerging from these shapes. A few simple examples: A vector has only one index, so we can draw it as just a circle with a singular index sticking out. A matrix has two indices which we call I and j, and last we draw a general tensor of rank n.

I want to also introduce the linear operation called index contraction. An index contraction of two tensors is performed by multiplying the two tensors, summing over certain connected indices. A simple example would be the scalar product of two vectors. Both vectors have a single index alpha, over which we sum here. The result is a scalar with zero indices. In the diagrammatic notation, index contractions are denoted by connecting the bonds that are contracted. Vectors have one index, so we can just connect the two legs of A and B to form the scalar product. 3 min

Similarly, the matrix product of two matrices is an index contraction, and can also be drawn as a tensor diagram.

Ok, with this we can then define tensor networks as a set of tensors that is contracted in a specified order. For example, here we show a simple tensor network of three tensors that are contracted in a triangle shape.

Matrix product states are a powerful Ansatz for one-dimensional quantum states, for example spins on a chain. I write here again the many-body wavefunction. Note that we can interpret the amplitude Psi as a tensor of rank N. The idea of MPS is now to write this large tensor as a contraction of many smaller tensors T of rank 3. If we additionally restrict the bond dimension between these tensors to a maximum value of chi, we have then reduced the number of parameters from d^N to N chi^2 d, which is linear in system size. Of course this is an approximation. However, as it turns out, ground states of local, gapped Hamiltonians have an entanglement structure that follows an area law, which means that when cutting the chain into two parts, the entanglement between the left and right part is smaller than a constant. Remarkably, these states are exactly those that can be represented efficiently by the MPS ansatz. To go from the large tensor Psi to the MPS, one can for example use a series of Singular Value decompositions. 4:40

A powerful feature of MPS is the existence of a canonical form which greatly speeds up many algorithms. To discuss this canonical form, we must first introduce isometries. An isometry is a m x n matrix W with n >= m, which satisfies the so called isometry constraint: The adjoint of W multiplied with W must be equal to the identity, while multiplying the other way gives a projector P. In tensor diagrams we draw this by decorating the legs of the tensor with arrows. As a convention we draw the larger dimension, in this case n, with incoming arrows, and the smaller dimension, in this case m, with outgoing arrows. The isometry condition can then be visualized as follows: Contracting W with its complex conjugate along the larger dimension gives identity, which we visualize with just a line, and contracting the other way gives a projector. An isometric tensor is then a tensor that can be transformed into an isometry by grouping all the legs with incoming and outgoing arrows respectively, as shown here. 6:00

If both dimensions are equal, an isometry becomes a unitary. We draw unitaries with bidirectional arrows, and they give identity when contracting in both directions.

We can now bring an MPS into the canonical form as follows: Starting from a normal MPS, we perform QR-decompositions starting from the left. A QR-decomposition decomposes a tensor into an isometry Q and an upper triangular matrix R. We do the same starting from the right, until we meet at the center. We then call all tensors with arrows pointing to the right A and all tensors with arrows pointing to the left B. The single tensor with only incoming arrows is called the orthogonality center Lambda. Now, the A tensors fulfil a left-isometry condition: They turn into identity when contracting with the complex conjugate from the left. The B tensors similarly fulfil a right-isometry condition. Note that no information was last, we were able to exactly transform the MPS into canonical form. 7 min

This allows us to simplify certain algorithms. As an example i want to show the computation of a local expectation value. Let us assume we want to compute the expectation value of an operator that acts on a site in the center of the chain. We can interpret the operator as a tensor of rank two, and computing the expectation value can then be drawn as a tensor network as follows: We contract the MPS with its complex conjugate, sandwiching the operator in between. Without the canonical form, computing this expectation value actually scales linearly with the system size, because we need to contract the full network. However, since were in the canonical form, we can successively use the isometry condition to turn the left and right boundaries into identities, reducing the expectation value to a contraction of only 3 tensors! 8 min

I want to introduce one more algorithm for MPS, which can be used to perform time evolution. To evolve the state by a time Delta t, we need to apply the time evolution operator U to the state, which is given by the exponential of -i\*\Delta t\*the Hamiltonian of the model we want to look at, where we set hbar to 1. Let us assume that the Hamiltonian can be split into terms acting only on neighboring sites j and j+1. We can then split the Hamiltonian into two terms, one acting only on even and one acting only on odd bonds. We can then approximate this exponential by a so-called Suzuki-Trotter decomposition with an error of Delta t^2. We have now split the exponentials of even and odd terms. Because all operators acting on even bonds commute with each other, this exponential factorizes into these bond operators acting only on neighboring bonds, and the same holds true for the odd terms. 9 min

If we now want to evolve the MPS by one time step, we need to apply these operators to the MPS. We can draw this as a tensor network as follows: Here we have the MPS, and we act first with the bond operators on all even bonds and then on all odd bonds. Now, we need to approximate this network with an updated MPS. This can be done by absorbing the bond operators into the MPS one after the other. A single bond operator can be absorbed by contracting it with the two MPS tensors into a tensor theta, and then splitting this tensor again with a truncated SVD. 9:45

Next, let me introduce isometric Tensor Product States (isoTPS), which generalize the canonical form of MPS to two and higher dimensions. First of all, the generalization of MPS to higher dimensions is known as Projected entangled pair states (PEPS), where we now write the wavefunction of a system on a 2D square lattice as the contraction C of tensors as shown in this tensor network. We call the bond dimension of the legs drawn in black D. The problem with this approach is that it is not possible to transform the network into a canonical form without error because of the presence of closed loops. Because of this, algorithms defined for PEPS are often very expensive and also instable. For example, a full TEBD update scales with the bond dimension D to the power of 10. 10:40

Isometric Tensor Product States are then defined as follows. The tensors are isometrized in a way that all arrows point towards a special row and column which is called the orthogonality hypersurface and is here drawn in red. Along the orthogonality hypersurface, all arrows point to a singular orthogonality center, which we draw in orange. Now, similar to MPS, we call columns left of the orthogonality hypersurface A and columns right of the orthogonality hypersurface B, while we call the orthogonality hypersurface itself Lambda. In contrast to MPS, a PEPS can not be transformed into an isoTPS without error. In fact, the expressional power of isoTPS is lower than PEPS, meaning that not all states that can be represented with PEPS can also be represented with isoTPS. 11:30

Most algorithms defined on isoTPS need to be able to move around the orthogonality center. Moving the orthogonality center along the orthogonality hypersurface is easy, and can be done via simple QR-decompositions.

Moving the orthogonality hypersurface to the left or to the right is a harder problem. Lets say the orthogonality column is at position n and we want to move it to position n+1. This problem can be solved by first splitting the column lambda^n into two columns A^n and Lambda, and then absorbing the column Lambda to the right. As a result we again obtain two columns, but now all arrows point to the right column. This last step of multiplying Lambda with the column B is equivalent to multiplying an MPS with a Matrix Product Operator, and this problem has existing efficient algorithms. For solving the first problem we want to minimize the distance Lambda^n – A^n contracted with Lambda. One idea is to variationally minimize this by sweeping over the two columns and optimizing one tensor at a time. However it is found that the so-called Moses-Move is more efficient while producing a solution that is almost as good as the variational one. 13:00

The Moses-Move can be performed as follows: We start at the bottom of the orthogonality column, and split the orthogonality center into three tensors using a tripartite decomposition. We then absorb the orange tensor into the next upper tensor. We now repeat this procedure, again splitting into three tensors and so on, until the top of the column is reached. The question is now how this tripartite decomposition can be performed. 13:30

A tripartite decomposition can be performed as follows: First, the tensor W is split into two tensors Q and theta via a singular value decomposition, where the bonds are also truncated to D^2. Next it is crucial to note that there is a gauge degree of freedom on the connecting bonds, we can insert a unitary and its adjoint. After absorbing U into theta and Udagger into Q, we then split theta via another SVD into two tensors B and C, finalizing the tripartite decomposition. In the third step, the unitary U can be chosen in such a way that it minimizes the truncation error of the final split in the last step. A good choice is a unitary that reduces the entanglement between the subsystems represented by the upper and lower indices of theta. This unitary is therefore called the disentangling unitary. We will discuss how to find a good disentangling unitary later. 15:00

In a similar way to MPS, one can perform a TEBD update step on isoTPS by first updating the tensors along all columns and then along all rows. Along the orthogonality hypersurface, the TEBD is very similar to MPS. The algorithm is much faster than for PEPS, reducing the computational complexity from D^10 to D^7! 15:20

In my thesis, I implemented an alternative canonical form, which was already thought of when the original isoTPS was developed but was never implemented. We call an isoTPS in this alternative canonical form Yang-Baxter isoTPS, for reasons that will become clear later. The canonical form is defined as follows: First, a PEPS is rotated by 45 degrees as shown here. We then introduce so-called auxiliary tensors, which we draw in red, and which make up the orthogonality hypersurface. The orthogonality hypersurface is placed in between two rows of PEPS tensors. Again, we isometrize such that all arrows point towards the orthogonality hypersurface, and towards a single tensor which is called the orthogonality center. Here, I also draw a unit cell, which now includes two sites.

In analogy to MPS and isoTPS, one can easily evaluate local expectation values. Lets say we want to compute the expectation value of an operator acting only on the two sites i and j. The two tensors representing sites I and j and the three neighboring auxiliary tensors form a sub-network of only incoming arrows. To compute the expectation value, we contract the state with its complex conjugate, sandwiching the operator in between. Now we can use the isometry conditions starting from the boundary, reducing contractions to identity, until we end up with a contraction of only the five marked tensors, there complex conjugates, and the bond operator.

As in the original isoTPS, most algorithms need to be able to move around the orthogonality center. Again, moving the orthogonality center along the orthogonality hypersurface is easy and just requires QR-decompositions. The problem arises again when we want to move the hypersurface to the left or to the right.

This can be done by starting with the orthogonality center at the bottom of the orthogonality hypersurface. Now, two auxiliary tensors are pulled through the physical tensor to the other side. Then we move up the orthogonality center and again pull two tensors through to the other side. The final tensor can then be easily moved by contracting and using a SVD. The main difficulty is the process of pulling the auxiliary tensors through the physical tensor, which we draw again here. We call this a Yang-Baxter move, because of the visual similarity to the Yang-Baxter equation. We call the tensors before this move T, W\_1 and W\_2, and denote the tensors after the move by T^prime, W\_1^prime and W\_2^prime. The error of the YB-move can be written as the norm of Psi – Psi^prime. We can expand this as the square root of the norm squared of psi + the norm squared of psi^prime – 2 times the real part of the overlap. We impose that both psi and psi^prime are normalized to one, and so this simply becomes the square root of 2 – the real part of the overlap.

The problem of finding a good YB move can therefore be written as a constrained optimization problem: The best tensors T^prime, W\_1^prime and W\_2^prime are the ones that maximise the real part of the overlap, under the constraint that T^prime and W\_1^prime are isometries and W\_2^prime is normalized to one, since it will be the new orthogonality center. In my thesis, I looked at two algorithms: A variational optimization with local updates and a tripartite decomposition similar to the one used in the MM.

Let’s start with the first of the two algorithms, the variational optimization. The real part of the overlap can be written as the following tensor network, where we have again used the isometry condition to turn a contraction of the full networks to a contraction of only a few tensors. Around the orthogonality center.