Matrix product operator representation of polynomial interactions

Michael L Wall

Department of Physics, Colorado School of Mines, Golden, Colorado 80401, USA
and The Johns Hopkins University Applied Physics Laboratory, Laurel, MD, 20723, United States of America

Received 17 January 2020, revised 28 March 2020
Accepted for publication 3 April 2020
Published 1 May 2020

Abstract
We provide an exact construction of particular Hamiltonians on a one-dimensional lattice as matrix product operators, a type of tensor network. Namely, we consider Hamiltonians describing interactions between degrees of freedom at lattice sites whose strength grows with the lattice site separation as a polynomial multiplied by an exponential. We show that the bond dimension is \((k + 3)\) for a polynomial of order \(k\), independent of the system size and the number of particles. Our construction is manifestly translationally invariant, and so may be used in finite- or infinite-size variational matrix product state algorithms. Our results provide new insight into the correlation structure of many-body quantum operators, and may also be practical in simulations of many-body systems whose interactions are exponentially screened at large distances, but may have complex short-distance structure.

Keywords: matrix product operators, tensor network, lattice models

1. Introduction

Starting with the seminal work of Affleck, Kennedy, Lieb, and Tasaki [1], matrix product states (MPSs), also known as finitely correlated states [2], have garnered a great deal of theoretical attention. One of the most appealing features of MPSs is that they provide an exact representation of certain translationally invariant quantum states, and they are unique among state ansätze in their ability to do so for entangled states. In addition to their usefulness as an analytic tool, MPSs are also the underpinning of the density-matrix renormalization group method (DMRG), which has become the de facto standard for strongly correlated systems in one spatial dimension (1D). In particular, DMRG can be expressed as a variational method within the space of MPSs [3]. The generalization of such a variational ansatz from pure states to density operators led to the introduction of the operator-valued generalization of MPSs, matrix product operators (MPOs), by Verstraete et al [4]. Later, McCulloch [5] realized that
significant gains can be had if all operators used in an MPS calculation are represented as MPOs, and put forward a lower triangular ‘canonical form’ for MPOs. As an example, the use of MPOs to represent the Hamiltonian operator leads to amortized linear scaling of DMRG sweeps with the system size using caching methods [6], and exact arithmetic can be used on MPOs to obtain quantities like the energy variance of quantum states. Hence, extending the class of operators with exactly known MPO representations not only improves our knowledge of the correlation structure of many-body objects, but also can lead to practical gains in numerical simulations.

Just as MPSs naturally describe quantum states with exponentially decaying correlations [3], MPOs are most naturally suited to describing interactions which have an exponential decay. Pirvu et al [7] and Crosswhite and Doherty [8] showed how general decaying functions may be approximated with MPOs by fitting the functional decay to a sum of MPOs. As the number of exponentials increases, the interaction is approximated to a larger distance, but for any finite number of exponentials there is a range beyond which the interaction no longer accurately approximates the true function. In parallel with analytical representations, numerical methods exist also for combining MPOs through arithmetic operations, or for reducing the bond dimension of an MPO [9]. Exact, automatic methods also can be devised for constructing an MPO from a finite-state machine, including in the presence of symmetries [10]. In reference [11], Frōwis, Nebendahl, and Dür undertook a classification of Hamiltonians which have an MPO representation whose bond dimension is independent of the system size. One of their examples was a polynomial times an exponential function, which was claimed to have a bond dimension of $O(k)$, with $k$ being the order of the polynomial, independent of the system size. An example MPO was given, but no constructive method nor proof of the methodology was presented for general polynomials. In this paper, we put forth a constructive characterization of the MPO representation of Hamiltonians with polynomial times exponential interactions for all orders $k$, and show that the bond dimension is $(k + 3)$.

The organization is as follows: in section 2 we briefly review the theory of MPOs to set notation and discuss previously known examples. In section 3 we present an MPO ansatz for general positive power-law interactions and prove its validity. Section 4 generalizes the results of the previous section to general polynomial interactions. Finally, in section 5 we conclude and give an outlook. Python code to solve for the coefficients of the MPO ansatz and a table of these coefficients in the special cases of power law interactions for the first six powers are given as appendices.

2. Matrix product operator definitions and examples

Let us consider a lattice of $L$ sites, each of which contains a $d$-dimensional Hilbert space spanned by the states $\{ |i\rangle, i = 1, \ldots, d \}$. A matrix product operator (MPO) acting on the Hilbert space of this lattice is defined as

$$\hat{O} = \sum_{i_1, \ldots, i_L} \text{Tr} \left[ W_{i_1}^{\mu_1, 1} \cdots W_{i_L}^{\mu_L, L} \right] |i_1 \ldots i_L \rangle \langle i_1' \ldots i_L'|, \quad (1)$$

where each of the objects $W_{i}^{\mu, j}$ is a matrix whose linear dimension is bounded by $\chi$, which we call the bond dimension of the matrix product operator, and Tr denotes the matrix trace. The indices of the physical Hilbert space, e.g., $i_j$ are called physical indices, while those involved
in the matrix product and trace will be referred to as bond indices. It is useful to re-write this expression as

\[ \hat{O} = \text{Tr} \left[ \hat{\mathcal{V}}^{[1]} \cdots \hat{\mathcal{V}}^{[L]} \right], \tag{2} \]

where now each of the matrix indices \( \hat{\mathcal{V}}^{[0]} = \sum_{i \neq j} \hat{W}^{i,j} \langle i | j \rangle \) is a matrix of operators acting on the Hilbert spaces spanned by the \( \{|i \rangle \} \) and whose matrix indices are the same as the \( \hat{W}^{i,j} \).

For a translationally invariant system, only a single MPO matrix \( \hat{\mathcal{V}} \) suffices to describe the operator. If we are constructing the representation of this operator on a finite chain of \( L \) sites with open boundary conditions, as is the most common scenario for numerical MPS simulations, we simply take the first MPO matrix \( \hat{\mathcal{V}}^{[1]} \) to be the last row of \( \hat{\mathcal{W}} \), the last MPO matrix \( \hat{\mathcal{V}}^{[L]} \) to be the first column of \( \hat{\mathcal{V}} \), and all other MPO matrices \( \hat{\mathcal{V}}^{[1 < j < L]} \) to be \( \hat{\mathcal{W}} \). In what follows, we will focus on such translationally invariant operators, and hence only describe the single MPO operator \( \hat{\mathcal{V}} \).

Similar to MPS representations of quantum states, MPOs are remarkable in their ability to compactly represent many-body operators. As an example, the MPO matrix \( \hat{\mathcal{V}}_{\text{one-body}} \) describing a one-body operator \( \sum_i \hat{X}_i \) is

\[ \hat{\mathcal{V}}_{\text{one-body}} = \begin{pmatrix} I & 0 \\ \hat{X} & I \end{pmatrix}, \tag{3} \]

the matrix \( \hat{\mathcal{V}}_{\text{two-body}} \) for a two-body operator \( \sum_i \hat{X}_i \hat{Y}_{i+1} \) is

\[ \hat{\mathcal{V}}_{\text{two-body}} = \begin{pmatrix} I & 0 & 0 \\ \hat{Y} & 0 & 0 \\ 0 & \hat{X} & I \end{pmatrix}, \tag{4} \]

and that for an exponentially decaying interaction \( \sum_{i < j} \beta^{j-i} \hat{X}_i \hat{Y}_j \) is

\[ \hat{\mathcal{V}}_{\text{exponential}} = \begin{pmatrix} I & 0 & 0 \\ \hat{Y} & \beta I & 0 \\ 0 & \beta \hat{X} & I \end{pmatrix}. \tag{5} \]

In all these examples, the dimensions of the matrices are indexed by bond indices, while the physical Hilbert space is described by the operator character of, e.g., \( \hat{X} \). More examples can be found in the literature, e.g., reference [11].

3. MPO construction of positive power-law interactions

3.1. Statement of MPO ansatz

Our ansatz for the MPO matrix describing the Hamiltonian of a power-law interaction of the form

\[ \hat{H} = \sum_{i < j} (j-i)^k \hat{X}_i \hat{Y}_j, \quad k \in \mathbb{N}, \tag{6} \]
is

\[
\hat{W} = \begin{pmatrix}
\hat{I} & 0 & 0 & \ldots & 0 & 0 & 0 \\
\sqrt{k+1} \hat{Y} & \hat{I} & 0 & \ldots & 0 & 0 & 0 \\
\sqrt{k+1} a_1 \hat{I} & a_1 \hat{I} & \hat{I} & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\
\sqrt{k+1} a_2 \hat{I} & a_2 \hat{I} & a_1 \hat{I} & \hat{I} & \ldots & 0 & 0 \\
0 & \sqrt{k+1} \hat{X} & \sqrt{k+1} \hat{X} & \ldots & \sqrt{k+1} \hat{X} & \sqrt{k+1} \hat{X} & \hat{I}
\end{pmatrix},
\]

(7)

or, in a more compact notation,

\[
\hat{W} = \begin{pmatrix}
\hat{I} & 0 \\
\sqrt{k+1} \hat{Y} & I_{k+1}^T \\
0 & \hat{X}
\end{pmatrix} \begin{pmatrix}
I_n \\
\mathbb{L}_k(a) \hat{I} \\
\hat{I}
\end{pmatrix}.
\]

(8)

In equation (8), \(I_n\) is a vector of length \(n\) whose elements are all 1 and \(\mathbb{L}_k(a)\) is the \((k + 1) \times (k + 1)\) matrix

\[
\mathbb{L}_k(a) = \begin{pmatrix}
1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
a_1 & 1 & 0 & \ldots & 0 & 0 & 0 \\
a_2 & a_1 & 1 & \ldots & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
a_{k-2} & a_{k-3} & \ddots & \ddots & 1 & 0 & 0 \\
a_{k-1} & a_{k-2} & a_{k-3} & \ddots & a_1 & 1 & 0 \\
a_k & a_{k-1} & a_{k-2} & \ldots & a_2 & a_1 & 1
\end{pmatrix},
\]

(9)

parameterized by the vector \(a = (a_1, a_2, \ldots, a_k)\). Note that the indices of the matrix \(\mathbb{L}_k\) and the vectors \(I_{k+1}\) are bond indices. As in the above, only operators denoted by hats act on the physical Hilbert space. \(L_k(a)\) is a Toeplitz matrix, i.e. a diagonally constant matrix, and is lower triangular. Comparing with the exponential MPO in equation (5), we see that a polynomial multiplied by an exponential interaction of the form

\[
\hat{H} = \sum_{i<j} \beta^{j-i} (j-i) \hat{X}_i \hat{Y}_j,
\]

(10)
is immediately obtained by replacing \(\mathbb{L}_k(a) \rightarrow \beta \mathbb{L}_k(a), \hat{X} \rightarrow \beta \hat{X}\), which does not change the structure or bond dimension of the MPO.

To begin to understand how the ansatz equation (8) generates polynomial interactions, let us consider the Hamiltonian on an increasing number of sites. For \(L = 2\) sites, the MPO matrices
are $\hat{V}^{(1)} = \hat{V}_{k+1}$, and $\hat{V}^{(L)} = \hat{V}_{1}$, as described above. Hence,

$$\hat{H}_{L=2} = \frac{1}{k+1} \left( \hat{I}_{k+1}^T \hat{I}_{k+1} \right) \hat{X}_1 \hat{Y}_2 = (2 - 1)^k \hat{X}_1 \hat{Y}_2. \quad (11)$$

For $L = 3$ sites, $\hat{V}^{(1)}$ and $\hat{V}^{(L)}$ remain the same and we introduce an additional matrix $\hat{V}^{(L-1)} = \hat{V}$ in between them, finding

$$\hat{H}_{L=3} = \frac{1}{k+1} \left( \hat{I}_{k+1}^T \hat{I}_{k+1} \right) \left[ \hat{X}_1 \hat{Y}_2 + \hat{X}_2 \hat{Y}_3 \right] + \frac{1}{k+1} \left( \hat{I}_{k+1}^T \mathbb{L}(a) \hat{I}_{k+1} \right) \hat{X}_1 \hat{Y}_3. \quad (12)$$

This gives the condition that $\hat{I}_{k+1}^T \mathbb{L}_k (a) \hat{I}_{k+1} = (k + 1) 2^{k}$ for our ansatz to faithfully reproduce the power-law interaction at this length. Following this line of reasoning through, the conditions on the vector $a$ such that the MPO matrix equation (7) reproduces the Hamiltonian equation (6) are

$$\hat{I}_{k+1}^T \mathbb{L}_n (a) \hat{I}_{k+1} = (k + 1)(n + 1)^k \quad (13)$$

for $n = 1, \ldots, k$; that is, conditions are placed on the elementwise sums of powers of the $\mathbb{L}_k (a)$ matrix. An inductive proof that this set of equations produces the Hamiltonian on any number of lattice sites is saved for section 3.4. Equation (13) represents a system of $k$ equations in $k$ unknowns. However, the $n$th equation is a degree $n$ polynomial in products of the elements of $a$, and the solution of equation (13) is hence a nontrivial task.

### 3.2. Formulation of the constraint equations

We begin the solution of equation (13) by defining the $n \times n$ shift matrix, $Z_n$, as

$$Z_n = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}. \quad (14)$$

From its definition, $Z_n$ is nilpotent with degree $n$, $Z_n^n = 0$. We can write $\mathbb{L}_k (a)$ in terms of the shift matrix as

$$\mathbb{L}_k (a) = \sum_{j=0}^{k} a_j \mathbb{L}_{k+1}^j, \quad (15)$$

where we have set $a_0 = 1$. Using equation (15), the $n$th power of $\mathbb{L}_k (a)$ also has a power series expansion in $Z_{k+1}$

$$\mathbb{L}_k^n (a) = \left( \sum_{j=0}^{k} a_j \mathbb{L}_{k+1}^j \right)^n = \sum_{j=0}^{k} c_{(n,k)j} \mathbb{L}_{k+1}^j, \quad (16)$$

5
where the coefficients $c_j^{(n,k)}$ are defined recursively as

$$c_0^{(n,k)} = 1, \quad c_m^{(n,k)} = \frac{1}{m} \sum_{j=1}^m [j(n+1)-m] a_{c_{m-j}^{(n,k)}}$$  \hspace{1cm} (17)

Furthermore, $I_n^T \cdot A \cdot I_n = \sum_{j=1}^n A_{ij}$ for any $n \times n$ matrix $A$, and so the constraint equations equation (13) may be stated in terms of the coefficients $c_j^{(n,k)}$ as

$$\xi_{nk} \equiv \sum_{j=1}^k c_j^{(n,k)} (k + 1 - j) = (k + 1) \left[ (n + 1)^k - 1 \right].$$  \hspace{1cm} (18)

Let us now derive a recursion relation between the coefficients $c_j^{(n,k)}$ with different $n$. We do so by equating powers of $Z_k + 1$ in the expansion

$$L_n^{(k)}(a) = \sum_{j=0}^k c_j^{(n,k)} Z_j^{k+1} = L_n^{(k-1)}(a) \left( \sum_{j=0}^k c_j^{(1,k)} Z_j^{k+1} \right),$$  \hspace{1cm} (19)

and find

$$c_j^{(nk)} = c_j^{(n-1,k)} + \sum_{p=1}^j c_p^{(1,k)} c_j^{(n-1,k)}. \hspace{1cm} (20)$$

Repeatedly applying the recursion equation (20) on the right-hand side of equation (20), we find

$$c_j^{(n,k)} = c_j^{(0,k)} + \sum_{q=1}^n \binom{n}{q} \sum_{p_1+p_2+\cdots+p_q=j} c_{p_1}^{(0,k)} \cdots c_{p_q}^{(0,k)} \prod_{i} c_{p_i}^{(1,k)}. \hspace{1cm} (21)$$

Here, the primed summation is defined as

$$\sum_{p_1 \cdots p_q} \equiv \sum_{p_1=1}^{j} \sum_{p_2=1}^{j} \cdots \sum_{p_q=1}^{j}. \hspace{1cm} (22)$$

Noting that $c_j^{(0,k)} = \delta_{j0}$ and $c_j^{(1,k)} = a_j$, we find

$$c_j^{(n,k)} = \sum_{q=1}^n \binom{n}{q} \sum_{p_1+p_2+\cdots+p_q=j} a_{p_1} \cdots a_{p_q}, \hspace{1cm} (23)$$

where $p_i \geq 1$ and $j \geq 1$. Stated in terms of $\xi_{nk}$, we have

$$\xi_{nk} = \sum_{j=1}^k (k + 1 - j) \sum_{q=1}^n \binom{n}{q} \sum_{p_1+p_2+\cdots+p_q=j} a_{p_1} \cdots a_{p_q}. \hspace{1cm} (24)$$

The result equation (24) is a precise restatement of the fact that the $n$th order condition equation (13) is a degree-$n$ polynomial in products of the elements of $\mathbf{a}$. In order to simplify the equations it is convenient to work not directly with the sequence $\xi_{nk}$, but with its binomial transform.
\[
\sum_{j=1}^{m} \binom{m}{j} \eta_{jk} = \xi_{mk}, \quad \eta_{mk} = \sum_{j=1}^{m} (-1)^{j+m} \xi_{jk} \binom{m}{j}.
\] (25)

We take the sums from \(j = 1\) due to the fact that \(\xi_{0k} = \eta_{0k} = 0\). Using equations (25) and (24) together, we have that

\[
\eta_{mk} = \sum_{n=1}^{m} (-1)^{n+m} \binom{m}{n} \sum_{j=1}^{k} (k+1-j) \sum_{q=1}^{n} \binom{n}{q} \sum_{p_1+\cdots+p_q=j} a_{p_1} \cdots a_{p_q}
\] (26)

\[
= \sum_{j=1}^{k} (k+1-j) \sum_{p_1+\cdots+p_m=j} a_{p_1} \cdots a_{p_m}.
\] (27)

Because of the restriction that all indices \(p_i \geq 1\), the condition \(\sum_{i=1}^{m} p_i = j\) can only be satisfied for \(j \geq m\), and so

\[
\eta_{mk} = \sum_{j=m}^{k} (k+1-j) \sum_{p_1+\cdots+p_m=j} a_{p_1} \cdots a_{p_m}.
\] (28)

That is, each term in \(\eta_{mk}\) is a monomial of degree \(m\) in the elements of \(a\). Furthermore, from the sum restriction, \(\eta_{mk}\) involves only the elements \(a_p\) with \(p \leq k - m + 1\). In particular, we have that

\[
\eta_{kk} = a_1^k.
\] (29)

Hence, we may solve for \(a_1, a_2, \text{ etc}\) in ascending order by considering the expressions equation (28) in descending order of \(m\). A numeric value for \(\eta_{mk}\) is obtained by using the far right-hand side of equation (18), and yields

\[
\eta_{mk} = (k+1) \sum_{j=1}^{m} (-1)^{j+m} \binom{m}{j} \left[(j+1)^k - 1\right].
\] (30)

Expanding the power on the right-hand side using the binomial theorem and applying the definition of the Stirling numbers of the second kind

\[
\left\{ \begin{array}{c} p \\ k \end{array} \right\} = \frac{1}{k!} \sum_{j=1}^{k} (-1)^{k-j} \binom{k}{j} p^j,
\] (31)

we find

\[
\eta_{mk} = (k+1) m! \sum_{q=1}^{k} \binom{k}{q} \left\{ \begin{array}{c} q \\ m \end{array} \right\}.
\] (32)

Using the identity [12]

\[
\sum_{p} \binom{n}{p} \left\{ \begin{array}{c} p \\ m \end{array} \right\} = \left\{ \begin{array}{c} n+1 \\ m+1 \end{array} \right\},
\] (33)

we then have

\[
\eta_{mk} = (k+1) m! \left\{ \begin{array}{c} k+1 \\ m+1 \end{array} \right\}.
\] (34)
In particular, for \( m = k \), we have \( \eta_{kk} = (k + 1)! \). Hence, if we choose the positive real root \( a_1 = [(k + 1)!]^1/k \), then the entire vector \( \mathbf{a} \) may be taken to be real. As an example, the equations to be solved for \( k = 4 \) are

\[
\eta_{44} = a_4^4 = 120
\]
\[
\eta_{34} = 2a_3^4 + 3a_1^2a_2 = 300
\]
\[
\eta_{24} = 3a_2^4 + 4a_1a_2 + 2a_1a_3 + a_2^2 = 250
\]
\[
\eta_{14} = 4a_1 + 3a_2 + 2a_3 + a_4 = 75.
\]

### 3.3. Solution of the constraint equations

While \( a_1 \) may be found analytically, the other coefficients must be generated numerically. To derive an efficient numerical procedure, we return to equation (28) and note that

\[
\eta_{mk} = (k + 1 - m) a_{k-1}^m + \sum_{q=1}^{k-m} (k + 1 - m - q) P_{mq}^k
\]

where

\[
P_{mq}^k = \sum_{p_1 + \cdots + p_m = m+q} a_{k-p_1} \cdots a_{k-p_m}.
\]

The sum counts the number of ways to partition the integer \((m + q)\) into \( m \) pieces such that each piece \( p_i \geq 1 \). Equivalently, the problem is the number of integer partitions of \( q \) into at most \( m \) pieces. This is a standard problem in combinatorics, and will not be reviewed here; an implementation is given as part of the python program in appendix B. A particular partition \( p \) may be written as the set \( \{s, m\} \), where \( s \) denotes the distinct integers forming the partition and \( m \) denotes their multiplicities, the number of times each integer appears, such that \( \sum s, m = n \). We will refer to the number of distinct integers in a particular partition \( p \) as the length of the partition, and denote it with \( \ell \). In terms of these quantities, the number of different ways that a particular partition \( p \) may be realized is \( \Omega(p) = (\sum m)! / \prod m_i! \).

The numerator counts the number of ways of arranging the \( p_j = 1 \), and the denominator removes identical rearrangements of the other \( p_j \). With all of the integer partitions and their multiplicities, we can generate the monomials in equation (39) and their weights. Substituting the numerical values of previously solved components of the vector \( \mathbf{a} \), this becomes an equation for a single unknown component \( a_j \), see, e.g. (35)–(38). The numerical right hand side of the equation is obtained from equation (30). Hence, starting from \( a_1 \), the entire vector \( \mathbf{a} \) can be obtained to any desired numerical precision. A python implementation of this procedure is given as appendix B.

### 3.4. Proof of construction

In this section we prove that if the conditions equation (13) hold for \( n = 1, \ldots, k \), then they hold for any \( n \in \mathbb{N} \). This demonstrates that the MPO matrix equation (7) faithfully represents the Hamiltonian equation (6) for a system of any number of sites. The proof is inductive. Let us assume that equation (13) is true for all \( n = 1, \ldots, k \), and venture to prove that equation (13) for \( n = (k + 1) \) follows; that is,

\[
\mathbf{I}_{k+1} \mathbf{T}_{k+1}^k \mathbf{L}^k_{k+1} (\mathbf{a}) \mathbf{I}_{k+1} = (k + 1)(k + 2)^k.
\]

\[
(41)
\]
We begin by noting that the eigenvalues of $L_k(a)$ are its diagonal elements, as is true for any triangular matrix. Hence, $L_k(a)$ satisfies the characteristic polynomial equation

$$(L_k(a) - I)^{k+1} = 0. \quad (42)$$

Using the binomial theorem and rearranging, we find

$$L_k^{k+1}(a) = \sum_{r=0}^{k} \binom{k+1}{r} (-1)^{k-r} L_r(a). \quad (43)$$

Multiplying by $1_{k+1}$ on the right, by $1^T_{k+1}$ on the left, and using the hypotheses $1^T_{k+1} L_n(a) 1_{k+1} = (n+1) [k+1]^n - 1$, $n = 1, \ldots, k$, we find

$$1^T_{k+1} L_k^{k+1}(a) 1_{k+1} = (k+1) \sum_{r=1}^{k} \binom{k+1}{r} (-1)^{k-r}(r+1)^k. \quad (44)$$

By virtue of Worpitzky’s identity,

$$x^k = \sum_{q=0}^{k-1} \binom{k}{q} \binom{k}{q}, \quad (45)$$

with $\binom{k}{q}$ an Eulerian number, we have

$$1^T_{k+1} L_k^{k+1}(a) 1_{k+1} = (k+1) \sum_{q=0}^{k-1} \sum_{r=1}^{k} (-1)^{k-r} \binom{k+1}{r} \binom{k}{q} \binom{k+1}{q} \langle k \rangle q^{r+1} k! \langle q \rangle. \quad (46)$$

To determine the value of the term in brackets, we write

$$\sum_{r=1}^{k+1} (-1)^{k-r} \binom{k+1}{r} \binom{k+1}{q} \binom{k+1}{q} \binom{k}{q} \binom{k+1}{q} \langle k \rangle q^{r+1} k! \langle q \rangle = \sum_{r=1}^{k+1} (-1)^{k-r} \binom{k+1}{r} \binom{k+1}{q} \binom{k+1}{q} \binom{k}{q} \binom{k+1}{q} \langle k \rangle q^{r+1} k! \langle q \rangle, \quad (47)$$

where $(x)_k$ is the falling factorial. Expanding the falling factorial in terms of the Stirling numbers of the first kind as

$$(x)_n = \sum_{p=0}^{n} (-1)^{n-p} \binom{n}{p} x^p, \quad (48)$$

we have

$$\sum_{r=1}^{k+1} (-1)^{k-r} \binom{k+1}{r} \binom{k+1}{q} \binom{k+1}{q} \binom{k}{q} \binom{k+1}{q} \langle k \rangle q^{r+1} k! \langle q \rangle = \frac{1}{k!} \sum_{p=0}^{k} (-1)^p \binom{k}{p} \sum_{\ell=0}^{p} \binom{p}{\ell} q^{p-1} \sum_{j=0}^{k+1} \binom{k+1}{j} (-1)^j. \quad (49)$$
The term in brackets in equation (49) vanishes according to the identity

\[
\sum_{i=0}^{k} (-1)^i \binom{k}{i} t^m = 0, \quad 0 \leq m \leq k, \ m, k \in \mathbb{Z},
\]  

which may be proved inductively. Hence,

\[
\sum_{r=1}^{k} (-1)^{k-r} \binom{k+1}{r} \binom{q+r}{k} = \binom{k+q+1}{k},
\]  

and so equation (46) becomes

\[
\mathbf{I}_{k+1}^{T} L_{k} (\mathbf{a}) \mathbf{L}_{k+1} = (k+1) \sum_{q=1}^{k} \binom{k+q+1}{k} \langle \binom{k}{q} \rangle.
\]  

Using Worpitzky’s identity again, we have

\[
\mathbf{I}_{k+1}^{T} L_{k} (\mathbf{a}) \mathbf{L}_{k+1} = (k+1)(k+2)^{k},
\]  

as was to be shown. This proof may be generalized to any arbitrary integer \( n = k + \ell \), \( \ell \in \mathbb{N} \), by multiplying the characteristic polynomial equation (42) by \((L_{k} (\mathbf{a}) - I)^{\ell-1}\) and following an identical line of reasoning. Hence, once the values of \( \mathbf{a} \) have been set by the conditions of equation (13) for \( n = 1, \ldots, k \), the MPO matrix equation (7) reproduces the Hamiltonian equation (6) on any number of sites.

4. Extension to general polynomial interactions

The proof given in section 3.4 relies only on the form of Worpitzky’s identity

\[
x^k = \sum_{q=0}^{k-1} \binom{q+x}{k} \langle \binom{k}{q} \rangle.
\]  

Hence, the same analysis applies to any function \( P_k (x) \) which can be written as a linear combination of \( \left\{ \binom{q+x}{k} \right\}, q = 0, \ldots, k-1 \). Because these binomial coefficients form a basis for the space of polynomials of degree \( k \) with no constant term, the most general functions which are linear combinations of these binomial coefficients are degree-\( k \) polynomials of the form

\[
P_k (x) = \sum_{i=1}^{k} \alpha_i x^i.
\]  

We now wish to express polynomials of the form equation (55) as

\[
P_k (x) = \sum_{q=0}^{k-1} \binom{q+x}{k} W_{kq}.
\]
We do so by noting that finding the coefficients $W_{kq}$ is equivalent to solving the linear system of equations

$$
\begin{pmatrix}
(1 + 0_k) & (1 + 1_k) & \cdots & (1 + k - 1_k) \\
(2 + 0_k) & (2 + 1_k) & \cdots & (2 + k - 1_k) \\
\vdots & \vdots & \ddots & \vdots \\
(k + 0_k) & (k + 1_k) & \cdots & (k + k-1_k)
\end{pmatrix}
\begin{pmatrix}
W_{k0} \\
W_{k1} \\
\vdots \\
W_{kn-1}
\end{pmatrix}
= 
\begin{pmatrix}
P_k(1) \\
P_k(2) \\
\vdots \\
P_k(k)
\end{pmatrix}.
$$

This linear system is solved for any $P_k(x)$ by inverting the Hankel matrix with elements

$$
H_{ij}^{(k)} = \binom{i + j - 1}{k}.
$$

It can be verified that $[H^{(k)}]^{-1}$ is again a Hankel matrix defined by the elements

$$
[H^{(k)}]_{ij}^{-1} = (-1)^{j+1-(i+j)} \binom{k + 1}{k + 1 - (i + j)}.
$$

For the special case $P_k(x) = x^k$, this construction reproduces the known representation of the Eulerian numbers

$$
\langle n \rangle_m = \sum_{k=0}^{m} (-1)^{j} \binom{n + 1}{k} (m + 1 - k)^n,
$$

and hence Worpitzky’s identity.

The above construction demonstrates that for any degree-$k$ polynomial $P_k(x)$ of the form equation (56) an exact MPO representation with bond dimension $(k + 3)$ may be found. To find the vector of coefficients $a$ which reproduces this polynomial, one uses the machinery of sections 3.2 and 3.3 with equation (30) replaced by

$$
\eta_m = (k + 1) \sum_{j=1}^{m} (-1)^{j+m} \binom{m}{j} [P_k(j+1) - 1].
$$

Also, as noted above, the MPO resulting from this construction is immediately generalized to interactions consisting of polynomials multiplied by an exponential using the replacements $L_k(a) \rightarrow \beta L_k(a)$, $X \rightarrow \beta X$, which does not change the structure or bond dimension of the MPO.

5. Conclusions

To summarize, we have put forwards an exact construction of Hamiltonians consisting of interactions whose strength varies as a degree-$k$ polynomial multiplied by an exponential with site separation as a matrix product operator (MPO) with bond dimension $(k + 3)$, independent of the system size or the number of particles. In addition to a proof that this construction reproduces the desired Hamiltonian on any number of sites, we also described an algorithm to determine the parameters appearing in the MPO ansatz to any desired numerical precision; a python implementation of this algorithm is given as appendix B. In addition to being useful for constructing complex operators for use in variational MPS calculations, our results provide new analytic insight into efficiently constructing quantum states and operators with complex correlations.
Table A.1. Table of values of $a$ for the first six powers.

| $k$ | $a_1$        | $a_2$        | $a_3$        | $a_4$        | $a_5$        | $a_6$        |
|-----|--------------|--------------|--------------|--------------|--------------|--------------|
| 1   | 2.000 000 000 000 0000 |             |              |              |              |              |
| 2   | 4.101 020 514 433 6442 | 2.449 489 742 783 1779 |             |              |              |              |
| 3   | 8.474 830 274 969 9516 | 5.435 836 151 592 7998 | 2.884 499 140 614 8166 |              |              |              |
| 4   | 17.555 589 156 123 46 | 11.719 390 564 662 234 | 6.922 208 678 654 8589 | 3.309 750 916 468 731 |              |              |
| 5   | 36.379 219 139 956 668 | 24.929 714 047 082 978 | 15.628 482 874 669 601 | 8.559 076 943 977 9582 | 3.727 919 273 191 3513 |              |
| 6   | 75.347 296 246 5863 | 52.613 014 703 365 58 | 34.275 565 252 333 72 | 20.257 230 769 906 63 | 10.345 394 101 852 634 | 4.140 680 833 465 2885 |
Appendix A. Table of values of a for the first six powers

In table A.1, we collect the numerical values of the vector $a$ for the first six powers. These values were generated using the code provided in appendix B.

Appendix B. Python code for solving for a

```python
from scipy.misc import comb
from math import factorial, log, exp

def Partitions(n, k):
    # Generate all partitions of an integer n into at most k positive integers.
    # The partitions are returned as a dict mapping the integer n_i to its multiplicity m_i such that
    # $\sum_i n_i m_i = n$.
    if n == 0:
        yield {}
        return
    partition = {n: 1}
    # start with trivial partition of n into 1 n
    my_keys = [n]
    # keys in the partition dict, sorted largest to smallest
    yield partition
    while my_keys != [1]:
        reuse = {}
        if my_keys[-1] == 1:
            # If my last generated partition contains ones, count them and re-use
            reuse = partition[1]
            del my_keys[-1]
            del partition[1]
        # (possibly also) reuse the smallest key of the last partition
        # larger than 1
        smallest_key = my_keys[-1]
        new_val = partition[smallest_key] - 1
        partition[smallest_key] = partition[smallest_key] - 1
        reuse += smallest_key
        if new_val == 0:
            del my_keys[-1], partition[smallest_key]
        # take the part to reuse and see how many (smallest_key - 1) s we can squeeze out of it
        skmo = (smallest_key - 1)
        nis, remain = divmod(reuse, skmo)
        partition[skmo] = nis
        my_keys.append(skmo)
        if remain:
            partition[remain] = 1
            my_keys.append(remain)
        bins = sum(partition.values())
```

if bins $\leq k$:
    yield partition

def GenerateProduct(l, q, k):
    "Generate $P_{\ell q}^{\hat{k}}$ as defined in equation (40)"
    Product=[]
    # First enumerate all partitions of the excess $q$ into $l$ pieces
    PqSet=Partitions(q, l)
    for Pq in PqSet:
        LPq=sum(Pq . values())
        # Make new dict with shifted values, then include
        a = {}
        for f in Pq:
            a[f+1]=Pq[f]
        # If the length of the partition is less than $l$, append some ones
        if (l-LPq)!$=0:
            a[1]=l-LPq
            elem={'weight':MultiplicityFactor(l, Pq)*$ (k+1-l-q),
                  'powers':a}
            Product.append(elem)
    return product

def GenerateEtaLHSs(k):
    "Generate all values of $\eta_{mk}$ as defined in equation (39), $m=1,\ldots,k$, as a list"
    etas=[]
    for m in range(1, k+1):
        firstcase={'weight': (k+1-m), 'powers': {1:m}}
        etaLHS=[]
        etaLHS.append(firstcase)
        for q in range(1, k-m+1):
            etaLHS+=GenerateProduct(m, q, k)
        etas.append(etaLHS)
    return etas

def MultiplicityFactor(l, Pq):
    "Compute the multiplicity factor of a given integer partition Pq for a multinomial of degree 1."
    LPq=sum(Pq.values())
    # Compute log of falling factorial $l_{\{LPq\}}$
    numer = 0.0
    for i in range(LPq):
        numer+=log(1-i)
    denom=0.0
    for term in Pq:
        denom+=log(factorial(Pq[term]))
    fac=exp(numer-denom)
    return fac

def GenerateEtaRHSs(k):
    "Generate all numerical values of $\eta_{mk}$, as defined in equation (30), $m=1,\ldots,k$, as a list"
etas = []
for m in range(1, k+1):
    eta = 0
    for j in range(m+1):
        \( x_{jk} = (k + 1) \times ((j + 1)^{k - 1}) \)
        eta += \((-1)^{(j + m)} \times x_{jk} \times \text{comb}(m, j)\)
    etas.append(eta)
return etas

def GenerateNumericalProduct(term, a):
    'Given a monomial in a and the numerical values of a, return
    the numerical value'
    k = len(a)
    val = term['weight']
    for x in term['powers']:
        val *= a[k-x]**(term['powers'][x])
    return val

def SolveEqns(etaLHS, etaRHS):
    'Solve the eta equations for the vector a'
    k = len(etaLHS)
    # Start at the top and work down
    a = [0]*k
    # Top equation is always \( a_{k-1} = (k + 1)! \)
    tmp = \log(factorial(k + 1))
    a[k-1] = \exp(tmp/(k+1.0))
    for p in range(1, k):
        sought = p + 1
        rhs = etaRHS[k-p-1]
        for term in etaLHS[k-p-1]:
            if sought in term['powers']:
                soughtterm = term
            else:
                rhs -= GenerateNumericalProduct(term, a)
        # Everything except the (single) term containing \( a_{k-p} \) has been moved the the rhs
        # Peel off any other factors of a
        val = soughtterm['weight']
        for x in soughtterm['powers']:
            if x != sought:
                val *= a[k-x]**(soughtterm['powers'][x])
        a[k-sought] = rhs/val
    return a

def PrintEtaEquations(LHS, RHS, k):
    'Print out the contents of the LHS and RHS dicts as a human-readable
equation'
    for p in range(k):
        print 'm =', k - p
        mystr = ''
        for term in LHS[k-p-1]:
            powstr = ''
            for pow in term['powers']:
                powstr += 'a[' + str(pow) + ']' + powterm['powers'][pow] + '*'
            mystr += powstr
        print mystr
mystr += str(term['weight'])+' '*powstr[:-2]+'
'+
mystr = mystr[:-2]+' = '+str(etaRHS[k+p-1])
print mystr

if __name__ == '__main__':
    for k in range(1,7):
        print 'k', k
        etaRHS = GenerateEtaRHSs(k)
        etaLHS = GenerateEtaLHSs(k)
        PrintEtaEquations(etaLHS, etaRHS, k)
        a=Solveeqns(etaLHS, etaRHS)
        print 'a', a
        print 'n

ORCID iDs

Michael L. Wall © https://orcid.org/0000-0002-6223-0800

References

[1] Affleck I, Kennedy T, Lieb E H and Tasaki H 1987 Phys. Rev. Lett. 59 799
[2] Fannes M, Nachtergaele B and Werner R F 1992 Commun. Math. Phys. 144 443
[3] Schollwock U 2011 Ann. Phys. 326 96
[4] Verstraete F, García-Ripoll J J and Cirac J I 2004 Phys. Rev. Lett. 93 207204
[5] McCulloch I P 2007 J. Stat. Mech. P10014
[6] Crosswhite G M and Bacon D 2008 Phys. Rev. A 78 012356
[7] Pirvu B, Murg V, Cirac J I and Verstraete F 2010 New J. Phys. 12 025012
[8] Crosswhite G M, Doherty A C and Vidal G 2008 Phys. Rev. B 78 035116
[9] Hubig C, McCulloch I P and Schollwock U 2017 Phys. Rev. B 95 035129
[10] Paeckel S, Köhler T and Manmana S R 2017 SciPost Phys. 3 035
[11] Fröwis F, Nebendahl V and Dür W 2010 Phys. Rev. A 81 062337
[12] Graham R L, Knuth D E, Patashnik O and Liu S 1989 Comput. Phys. 3 106