Supplemental material for "Witnessing entanglement in quantum magnets using neutron scattering"

A. Scheie,[1] Pontus Laurell,[2,3] A. M. Samarakoon,[1] B. Lake,[4,5] S. E. Nagler,[1,6] G. Granroth,[1] S. Okamoto,[7,6] G. Alvarez,[2,3] and D. A. Tennant[1,6,8]

[1]*Neutron Scattering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[2]*Center for Nanophase Materials Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[3]*Computational Science and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[4]*Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Hahn-Meitner Platz 1, D-14109 Berlin, Germany*
[5]*Institut für Festkörperphysik, Technische Universität Berlin, Hardenbergstraße 36, D-10623 Berlin, Germany*
[6]*Quantum Science Center, Oak Ridge National Laboratory, Tennessee 37831, USA*
[7]*Materials Science and Technology Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA.*
[8]*Shull Wollan Center - A Joint Institute for Neutron Sciences, Oak Ridge National Laboratory, TN 37831. USA*

(Dated: February 17, 2021)

I. SUPPLEMENTAL: REPRODUCING THE DMRG RESULTS.

Here we provide detailed instructions on how to reproduce the DMRG results used in the main text. The results reported in this work were obtained with DMRG++ versions 5.67, 5.69, 5.70 and PsimagLite versions 2.67, 2.69, 2.70.

The DMRG++ computer program [1] can be obtained with:

```
git clone https://github.com/g1257/dmrgpp.git
```

Dependencies include the BOOST and HDF5 libraries, and PsimagLite. The latter can be obtained with:

```
git clone https://github.com/g1257/PsimagLite.git
```

To compile:

```
cd PsimagLite/lib; perl configure.pl; make
cd ../../dmrgpp/src; perl configure.pl; make
```

To simplify commands below we also run

```
export PATH="<PATH-TO-DMRG++>/src:$PATH"
export SCRIPTS="<PATH-TO-DMRG++>/scripts"
```

The documentation can be found at `https://g1257.github.io/dmrgPlusPlus/manual.html` or can be obtained by doing `cd dmrgpp/doc; make manual.pdf`.

A. Obtaining zero-temperature spectra

The $T = 0$ results can be reproduced as follows. First DMRG++ is run with an input file, `dmrg -f inputGS.ain`, to obtain the ground state, where `inputGS.ain` has the form

```
##Ainur1.0
TotalNumberOfSites=100;
NumberOfTerms=2;

### 1/2(S^+S^- + S^-S^+) part
gt0:DegreesOfFreedom=1;
gt0:GeometryKind="chain";
gt0:GeometryOptions="ConstantValues";
gt0:dir0:Connectors=[1.0];

### S^zS^z part
gt1:DegreesOfFreedom=1;
```

```
gt1:GeometryKind="chain";
gt1:GeometryOptions="ConstantValues";
gt1:dir0:Connectors=[1.0];

Model="Heisenberg";
integer HeisenbergTwiceS=5;

SolverOptions="twositedmrg,calcAndPrintEntropies";
InfiniteLoopKeptStates=1000;
FiniteLoops=[
[ 49, 1000, 8],
[-98, 1000, 8],
[ 49, 1000, 8],
[ 49, 1000, 2],
[-98, 1000, 3]];

# Keep a maximum of 1000 states, but allow
# truncation with tolerance and minimum states
string TruncationTolerance="1e-10,100";

# Tolerance for Lanczos
real LanczosEps=1e-10;
int LanczosSteps=600;

Threads=4;
integer TargetSzPlusConst=250;
```

Here we showed the input for $S = 5/2$. The parameter `TargetSzPlusConst` should equal $S^z + S \cdot N$, where S^z is the targeted S^z sector and N is the system size. The line may be left out for the $S = 1/2$ case.

The next step is to calculate dynamics, using the saved ground state as an input. It is convenient to do the dynamics run in a subdirectory `Szz`, so `cp inputGS.ain Szz/inputSzz.ado` and add/modify the following lines in `inputSzz.ado`,

```
SolverOptions="twositedmrg,restart,minimizeDisk,CorrectionVectorTargeting";

### The finite loops pick up where gs run ended! I.e. the edge.
FiniteLoops=[
[98, 1000, 2],
[-98, 1000, 2]];

# RestartFilename is the name of the GS .hd5 file (extension is not needed)
string RestartFilename="../inputGS";

# The weight of the g.s. in the density matrix
GsWeight=0.1;

# Legacy thing, set to 0
real CorrectionA=0;

# Fermion spectra has sign changes in denominator. For boson operators (as in here) set it to 0
integer DynamicDmrgType=0;

# The site(s) where to apply the operator below. Here it is the center site.
TSPSites=[50];

# The delay in loop units before applying the operator. Set to 0 for all restarts to avoid delays.
TSPLoops=[0];
```

```
# If more than one operator is to be applied, how they should be combined.
# Irrelevant if only one operator is applied, as is the case here.
TSPProductOrSum="sum";

# How the operator to be applied will be specified
TSPOperator="expression";

# The operator expression
OperatorExpression="sz";

# Apply operator to ground state
string TSPApplyTo="|X0>";

# How is the freq. given in the denominator (Matsubara is the other option)
string CorrectionVectorFreqType="Real";

# This is a dollarized input, so the omega will change from input to input.
real CorrectionVectorOmega=$omega;

# The broadening for the spectrum in omega + i*eta
real CorrectionVectorEta=0.10;

# The algorithm
string CorrectionVectorAlgorithm="Krylov";

#The labels below are ONLY read by manyOmegas.pl script

# How many inputs files to create
#OmegaTotal=601

# Which one is the first omega value
#OmegaBegin=0.0

# Which is the "step" in omega
#OmegaStep=0.025

# Because the script will also be creating the batches, indicate what to measure in the batches
#Observable=sz
```

Then all individual inputs (one per ω in the correction vector approach) can be generated and submitted using the `manyOmegas.pl` script:

```
perl -I ${SCRIPTS} ${SCRIPTS}/manyOmegas.pl inputSzz.ado BatchTemplate <test/submit>.
```

It is recommended to run with `test` first to verify correctness, before running with `submit`. Depending on the machine and scheduler, the BatchTemplate can be e.g. a PBS script. The key is that it contains a line

```
dmrg -f $$input "<X0|$$obs|P1>,<X0|$$obs|P2>,<X0|$$obs|P3>" -p 10
```

which allows `manyOmegas.pl` to fill in the appropriate input for each generated job batch. After all outputs have been generated,

```
perl -I ${SCRIPTS} ${SCRIPTS}/procOmegas.pl -f inputSzz.ado -p
perl ${SCRIPTS}/pgfplot.pl
```

can be used to process and plot the results.

B. Obtaining finite-temperature spectra

First note that the $T > 0$ calculation discussed in the main text is very time consuming. It proceeds in three steps: First the $T = \infty$ state is found using a fictitious "entangler" Hamiltonian H_{E} acting in an enlarged Hilbert space [2–4]. Second, the physical system is cooled through evolving in imaginary time with the physical Hamiltonian H acting only on physical sites, i.e. we evolve with $H \otimes I$, where I is the identity operator in the ancilla space. Third, dynamics is calculated using the operator $H \otimes I + I \otimes (-H)$.

In the first step, we use a conventional (grand canonical) entangler, such that the enlarged system (physical and ancilla sites) can be described as a spin ladder, with physical sites on one leg (with even sites $0, 2, 4, \dots$) and ancilla sites on the other (with odd sites $1, 3, 5, \dots$). The entangler Hamiltonian is chosen such that its ground state corresponds to the $T = \infty$ state of the physical system. We find the ground state of H_E by running `dmrg -f Entangler.ain`, where `Entangler.ain` is given as

```
##Ainur1.0
TotalNumberOfSites=100;
NumberOfTerms=2;

gt0:DegreesOfFreedom=1;
gt0:GeometryKind="ladder";
gt0:LadderLeg=2;
gt0:GeometryOptions="ConstantValues";
gt0:dir0:Connectors=[0.0];
gt0:dir1:Connectors=[-10.0];
integer gt0:IsPeriodicX=0;

gt1:DegreesOfFreedom=1;
gt1:GeometryKind="chain";
gt1:GeometryOptions="ConstantValues";
gt1:dir0:Connectors=[0];
integer gt1:IsPeriodicX=0;

Model="Heisenberg";
integer HeisenbergTwiceS=1;

SolverOptions="twositedmrg,MatrixVectorOnTheFly";
InfiniteLoopKeptStates=1000;
FiniteLoops=[
[ 49, 1000, 0],
[-98, 1000, 0],
[ 98, 1000, 0],
[-98, 1000, 0]];

# Keep a maximum of 1000 states, but allow truncation with tolerance and minimum states as below
string TruncationTolerance="1e-8,100";

# Tolerance for Lanczos
real LanczosEps=1e-8;
int LanczosSteps=250;
```

Next the imaginary time evolution is initiated with `dmrg -f Evolution1.ain`, where `Evolution1.ain` adds and modifies some line compared to `Entangler.ain`. For brevity we only reproduce the modified lines below.

```
gt0:GeometryOptions="none";
gt0:dir0:Connectors=[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, ...];
gt0:dir1:Connectors=[0.0, 0.0, 0.0, 0.0, ...];

gt1:GeometryKind="ladder";
gt1:LadderLeg=2;
```

```
gt1:GeometryOptions="none";
gt1:dir0:Connectors=[1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, ...];
gt1:dir1:Connectors=[0.0, 0.0, 0.0, 0.0, ...];

string PrintHamiltonianAverage="s==c";
string RecoverySave="@M=100,@keep,1==1";
SolverOptions="twositedmrg,restart,TargetingAncilla";
FiniteLoops=[
[ 98, 1000, 2],
[-98, 1000, 2]];
RepeatFiniteLoopsTimes=21;

string RestartFilename="Entangler";

TSPTau=0.1;
TSPTimeSteps=5;
TSPAdvanceEach=98;
TSPAlgorithm="Krylov";
TSPSites=[50];
TSPLoops=[0];
TSPProductOrSum="sum";
GsWeight=0.1;

TSPOperator="expression";
OperatorExpression="identity";
```

Above we have abbreviated the lines describing the connectors using .... The `gt?:dir1:Connectors` describe couplings across rungs, and should all be zero since we only act on physical sites. There are $N/2 = 50$ such rungs, and a value for each needs to be listed in the array in the input. The `gt?:dir0:Connectors` describe couplings along legs, starting at site j corresponding to the jth position in the array (indexed from zero). To only couple physical sites every other leg coupling is set to zero. For open boundary conditions there are $N - 2$ such bonds that need to be included in the array in the input.

As further described in the input, we use Krylov imaginary time evolution with a time step $\tau = 0.1$. The time evolution is done with an evolution operator $\exp\left[-\frac{\beta H \tau}{2}\right]$, so τ is given in units of $\beta' = \beta/2$ for $J = 1$. The imaginary time β' can be obtained with `h5dump -d /Def/FinalPsi/TimeSerializer/Time <hd5>`, where `<hd5>` should be replaced with the name of the `hd5` file of interest. To arrive at imaginary times corresponding to experimental temperatures we do additional restarts from appropriate `hd5` files output by the main temperature evolution loop, while tuning the τ value. Explicitly, the targeted β' value can be found as $\beta'(J, T) = T/(2J)$, where J and T are given in Kelvin. Finally, the arguments to `RecoverySave` mean that we keep a maximum of 100 `hd5` outputs, and output one in every loop (when the condition 1 == 1 holds).

Finally, the dynamics calculation proceeds similarly to the $T = 0$ case, but with number of sites and precision as in the preceding $T > 0$ step. We do, however, need to additionally add/modify the following lines

```
string GeometrySubKind="GrandCanonical";

gt0:dir0:Connectors=[1.0, -1.0, 1.0, -1.0, 1.0, -1.0, 1.0, -1.0, ...];
gt0:dir1:Connectors=[0.0, 0.0, 0.0, 0.0, ...];

gt1:dir0:Connectors=[1.0, -1.0, 1.0, -1.0, 1.0, -1.0, 1.0, -1.0, ...];
gt1:dir1:Connectors=[0.0, 0.0, 0.0, 0.0, ...];

string RestartFilename="../Evolution_150K";

SolverOptions="CorrectionVectorTargeting,restart,twositedmrg,minimizeDisk,fixLegacyBugs";

integer RestartSourceTvForPsi=0;
vector RestartMappingTvs=[-1, -1, -1, -1];
integer RestartMapStages=0;
integer TridiagSteps=400;
```

```
real TridiagEps=1e-9;
```

The restart filename should be chosen to match the `hd5` file of interest. Note here that we calculate dynamics with $H\otimes I + I\otimes(-H)$, where $H\otimes I$ acts only on physical sites and $I\otimes(-H)$ acts only on ancilla sites. All rung couplings are zero. As before, we have abbreviated the arrays of coupling constants.

[1] G. Alvarez, The density matrix renormalization group for strongly correlated electron systems: A generic implementation, Comp. Phys. Comms. **180**, 1572 (2009).

[2] A. E. Feiguin and S. R. White, Finite-temperature density matrix renormalization using an enlarged Hilbert space, Phys. Rev. B **72**, 220401 (2005).

[3] A. E. Feiguin and G. A. Fiete, Spectral properties of a spin-incoherent Luttinger liquid, Phys. Rev. B **81**, 075108 (2010).

[4] A. Nocera and G. Alvarez, Symmetry-conserving purification of quantum states within the density matrix renormalization group, Phys. Rev. B **93**, 045137 (2016).