Supplementary Material for:
Electron addition spectral functions of low-density polaron liquids

Alberto Nocera[1, 2] and Mona Berciu[1, 2, 3]

[1]*Stewart Blusson Quantum Matter Institute, University of British Columbia, Vancouver, British Columbia, V6T 1Z4 Canada*
[2]*Department of Physics Astronomy, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z1*
[3]*Leibniz Institute for Solid State and Materials Research (IFW) Dresden, Helmholtzstrasse 20, 01069 Dresden, Germany*

(Dated: March 31, 2023)

Computational details to reproduce the DMRG results

Here we provide instructions on how to reproduce the DMRG results used in the main text. The results reported in this work were obtained with DMRG++ versions 6.05 and PsimagLite versions 3.04. The DMRG++ computer program [1] can be obtained with:

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

The main dependencies of the code are BOOST and HDF5 libraries. To compile the program:

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

The DMRG++ documentation can be found at `https://g1257.github.io/dmrgPlusPlus/manual.html` or can be obtained by doing `cd dmrgpp/doc; make manual.pdf`. In the description of the DMRG++ inputs below, we follow very closely the description in the supplemental material of Ref. [2], where similar calculations were performed.

The spectral function results for the 1D Holstein model for $L = 80$ sites, $N = 8$ electrons (thus $x = 0.1$), $\Omega = t$ and $\lambda = 0.25$ can be reproduced as follows. We first run `./dmrg -f inputGS.ain -p 12` to obtain the ground state wave-function and ground state energy with 12 digit precision using the `-p 12` option. The `inputGS.ain` has the form (this is provided at [3])

```
##Ainur1.0
TotalNumberOfSites=160;
NumberOfTerms=3;

### \sum_{<i,j>} -t * (c^\dag_i c_j + h.c.)
gt0:DegreesOfFreedom=1;
gt0:GeometryKind="ladder";
gt0:GeometryOptions="ConstantValues";
gt0:dir0:Connectors=[1.0];
gt0:dir1:Connectors=[0.0];
gt0:LadderLeg=2;

### bosonic hopping \sum_{<i,j>} -t_B * (b^\dag_i b_j + h.c.)
gt1:DegreesOfFreedom=1;
gt1:GeometryKind="ladder";
gt1:GeometryOptions="ConstantValues";
gt1:dir0:Connectors=[0.0];
gt1:dir1:Connectors=[0.0];
gt1:LadderLeg=2;

### electron-phonon interaction \sum_i g*c^\dag_i c_i * (b^\dag_i+b_i) g=sqrt(2*Omega*lambda)
gt2:DegreesOfFreedom=1;
gt2:GeometryKind="ladder";
gt2:GeometryOptions="ConstantValues";
gt2:dir0:Connectors=[0.0];
gt2:dir1:Connectors=[0.707];
```

```
gt2:LadderLeg=2;

Model="HolsteinSpinlessThin";
SolverOptions="twositedmrg,CorrectionTargeting,vectorwithoffsets,useComplex";
InfiniteLoopKeptStates=24;
FiniteLoops=[[79, 1000, 0],
[-158, 1000, 0],
[158, 1000, 0]];
# Keep a maximum of 1000 states, but allow SVD truncation with
# tolerance 1e-12 and minimum states equal to 24
TruncationTolerance="1e-12,24";
# Symmetry sector for ground state N_e = 8
TargetElectronsTotal=8;
# Associated with CorrectionTargeting: noise strength added to
# reduced density matrix to avoid the algorith gets stuck;
CorrectionA=0.01;
OutputFile="dataGS_L80_N8_nph8";
```

The next step is to calculate dynamics for the $A(\mathbf{k},\omega)$ spectral function using the saved ground state as an input. It is convenient to do the dynamics run in a subdirectory `Aqw`, so create this directory first, and then add/modify the following lines in `inputAqw.ado` (this input is provided at [3])

```
# The finite loops now start from the final loop of the gs calculation.
# Total number of finite loops equal to N+2, here N=6
FiniteLoops=[
[-158, 1000, 2],[158, 1000, 2],
[-158, 1000, 2],[158, 1000, 2],
[-158, 1000, 2],[158, 1000, 2],
[-158, 1000, 2],[158, 1000, 2]];

# Keep a maximum of 1000 states, but allow SVD truncation with
# tolerance 1e-6 and minimum states equal to 24
TruncationTolerance="1e-6,24";

# The exponent in the root-N CV method
CVnForFraction=6;

# Tolerance for Tridiagonal Decomposition of the effective Hamiltonian
TridiagonalEps=1e-12;

# Solver options should appear on one line, here we have two lines because of formatting purposes
SolverOptions="useComplex,twositedmrg,vectorwithoffsets,
               TargetingCVEvolution,restart,fixLegacyBugs,minimizeDisk";

# RestartFilename is the name of the GS .hd5 file (extension is not needed)
RestartFilename="../dataGS_L80_N8_nph8";

# The weight of the g.s. in the density matrix
GsWeight=0.1;
# Legacy, set to 0
CorrectionA=0;
# Fermion spectra has sign changes in denominator.
# For boson operators (as in here) set it to 0
DynamicDmrgType=0;
# The site(s) where to apply the operator below. Here it is the center site.
TSPSites=[79];
# The delay in loop units before applying the operator. Set to 0
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";
# Sets the number of sweeps to 1 before advancing in "time"=1/N
TSPAdvanceEach=78;
# How the operator to be applied will be specified
string TSPOp0:TSPOperator=expression;
# The operator expression to apply the c^\dagger operator on the center site
string TSPOp0:OperatorExpression="c?0'";
# How is the freq. given in the denominator (Matsubara is the other option)
CorrectionVectorFreqType="Real";
# This is a dollarized input, so the
# omega will change from input to input.
CorrectionVectorOmega=$omega;
# The broadening for the spectrum in omega + i*eta
CorrectionVectorEta=0.05;
# The algorithm
CorrectionVectorAlgorithm="Krylov";
#The labels below are ONLY read by manyOmegas.pl script
# How many inputs files to create
#OmegaTotal=40
# Which one is the first omega value
#OmegaBegin=-4.0
# Which is the "step" in omega
#OmegaStep=0.1
# Because the script will also be creating the batches,
# indicate what to measure in the batches
#Observable=c?0
```

Notice that the main change with respect of a standard CV method input is given by the option `TargetingCVEvolution` in the SolverOptions instead of `CorrectionVectorTargeting`, and the addition of the line `CVnForFraction=6;` We note also that the number of finite loops must be at least equal to the number equal to the exponent in the root-N CV method. As in the standard CV approach, all individual inputs (one per ω in the correction vector approach) can be generated and submitted using the `manyOmegas.pl` script which can be found in the `dmrgpp/src/script` folder (but also provided at [3]):

```
perl manyOmegas.pl inputAqw.ado batchTemplate.pbs <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 or SLURM script. The key is that it contains a line `./dmrg -f $$input "<gs|$$obs|P1>" -p 12` which allows `manyOmegas.pl` to fill in the appropriate input for each generated job batch. After all outputs have been generated,

```
perl myprocAkw.pl inputAqw.ado
```

can be used to process (this script is also provided at [3]) and generate a data file `outSpectrum.c?0.gnuplot` ready to be plotted (using the Gnuplot software, for example).

[1] G. Alvarez, Computer Physics Communications **180**, 1572 (2009).

[2] A. Nocera and G. Alvarez, Phys. Rev. B **106**, 205106 (2022), URL https://link.aps.org/doi/10.1103/PhysRevB.106.205106.

[3] DMRG inputs, submission scripts, and data postprocessing scripts are available at, URL 10.5281/zenodo.7790340.