DMRG simulation of bosonic quantum gases confined in 1D lattice

Quantum Information and Computing

L. Borella,
L. Rinaldi.
April 13, 2022

Università
degli Studi
di Padova

Overview

Overview

If we consider a quantum system made of N subsystems of dimension d, we must define d^N parameters to represent its wave function. Instead, the Hamiltonian describing the physics of the system will need $d^N \times d^N$ parameters.

$$|\phi >= \sum_{i=1}^{d} c_i |\alpha_i > \tag{1}$$

$$|\psi >= \sum_{\alpha_1 \cdots \alpha_d} C_{\alpha_1 \cdots \alpha_d} (|\alpha_1 > \otimes \cdots \otimes |\alpha_d >) \tag{2}$$

This dependence on the number of subsystems strongly limits our capability of studying and simulating exactly the behaviour of the complete system taken into consideration.

Tensor Network Methods come to help when dealing with these kind of problems.

By means of approximating complex tensors and applying certain algorithmic procedures that limit the growth of parameters, Tensor Network Methods allow us to achieve satisfying results both in terms of physical meaning and computational efficiency.

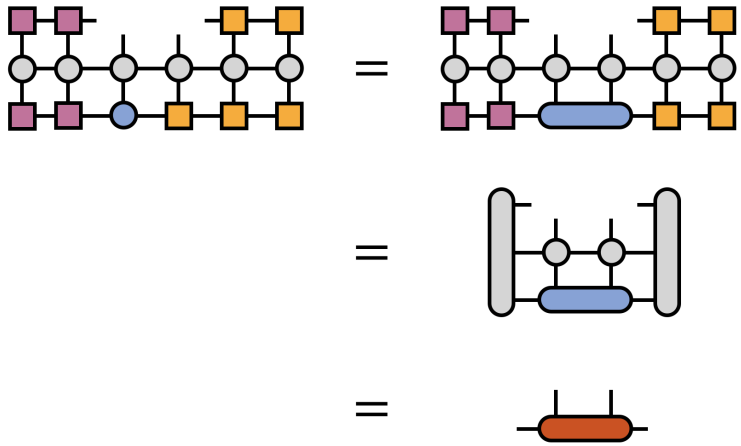


Figure: Main steps of the DMRG algorithm.

Overview

Matrix Product State

The matrix product state (MPS) tensor network is a factorization of a tensor with N indices into a chain-like product of three-index tensors. We will use it in the following as a representation of the state of the quantum system we are studying.

$$T^{s_1 s_2 s_3 \cdots s_N} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} B^{s_2}_{\alpha_1 \alpha_2} C^{s_3}_{\alpha_2 \alpha_3} \cdots D^{s_N}_{\alpha_{N-1}}$$

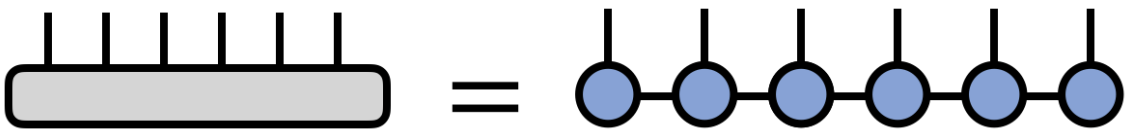
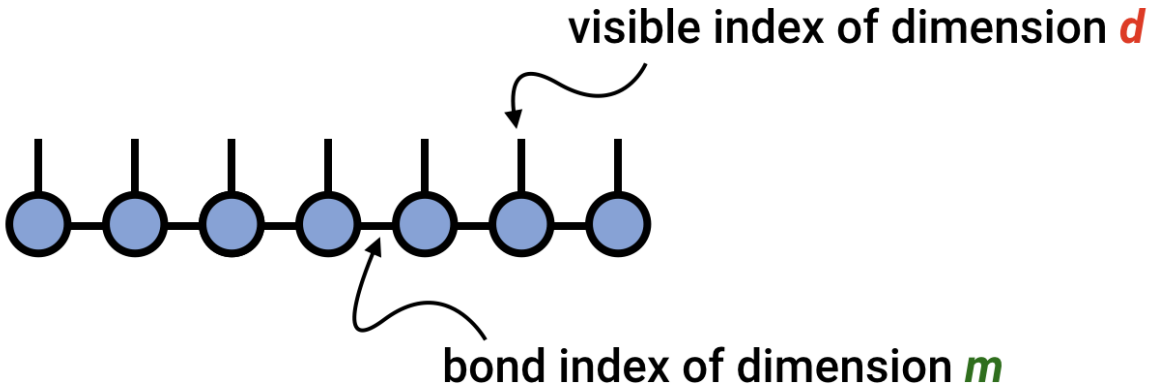


Figure: MPS:traditional and tensor diagram notation.

MPS: Number of Parameters

The dimension of the bond index m is responsible for the expressivity of the MPS: the higher the dimension of the contracted indices, the closer the MPS will be to the original tensor.



While a tensor with N indices of dimension d generally needs d^N parameters, the MPS form allows us only to specify Ndm^2, moving from an exponential to a linear scaling of the parameters with the number of indices.

The aim of the DMRG algorithm

The DMRG algorithm aims to find the ground state of a quantum system by means of contraction and optimization of tensor networks. The MPS represents the eigenstate while the Hamiltonian is represented as tensor with N covariant and N controvarian indices.

$$H|\psi_0>=E_0|\psi_0>$$

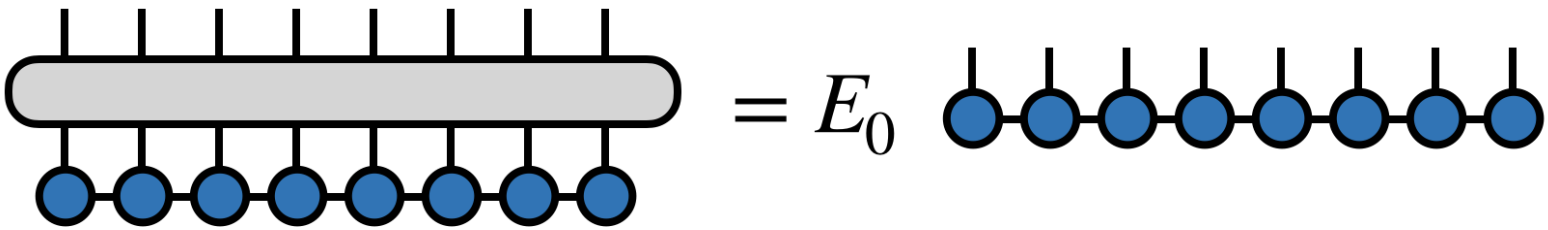
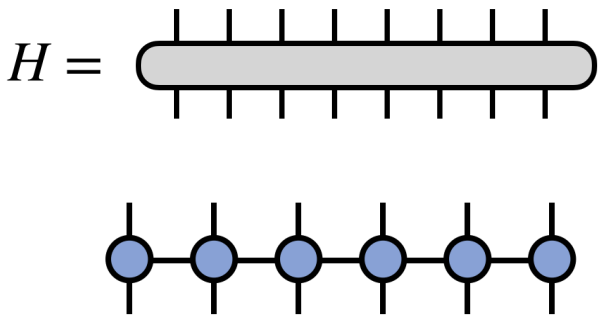
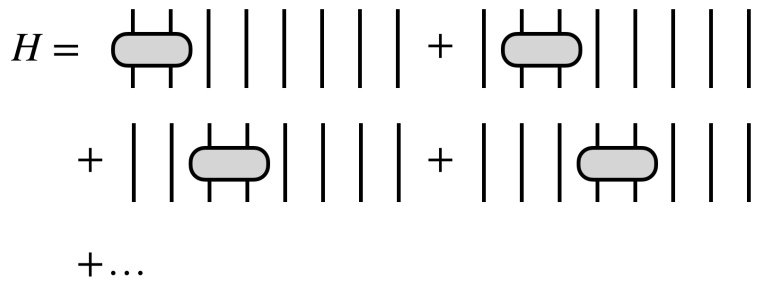


Figure: Eigenvalue equation in tensor diagram notation.

Matrix Product Operator

For the algorithm to be efficient, the Hamiltonian must be expressed as a Matrix Product Operator, which is the approximation of an operator tensor analogous to the MPS case. The MPO factorization is especially applicable in the cases where the Hamiltonian is made of sums of local terms.





DMRG: Steps of the algorithm

1. Select sites i and $i+1$ and move the MPS in orthogonal form.
2. Create left and right environments tensors around sites $i, i+1$.
3. Optimize the bond by contracting the effective hamiltonian with the two-site tensor theta.
4. Restore the MPS form of the result and update the state accordingly.
5. Perform sweeps left and right.
6. If necessary repeat the previous passages to improve convergence.

When looking for the ground state, the DMRG algorithm optimizes bond per bond instead of working with the full network. It is necessary to initially select two contiguous sites and move the rest of the MPS into right/left canonical forms by means of gauge transformations. In this way we can interpret the right/left canonical tensors as a change of basis from the physical indices i to the bond index α.

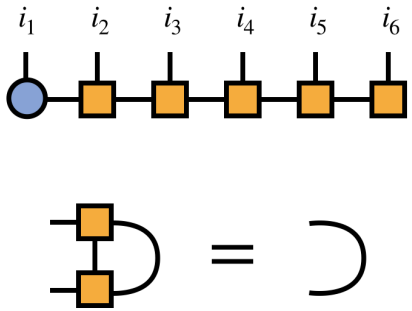


Figure: Right canonical MPS and Gauge transformation.

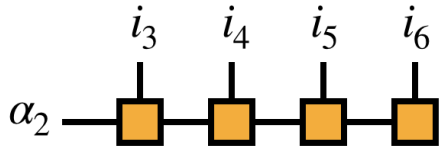
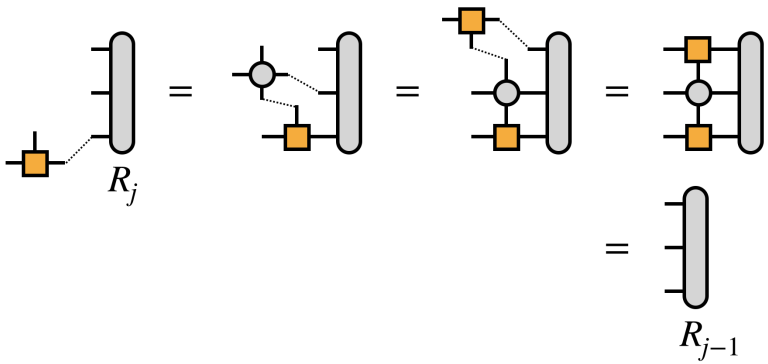
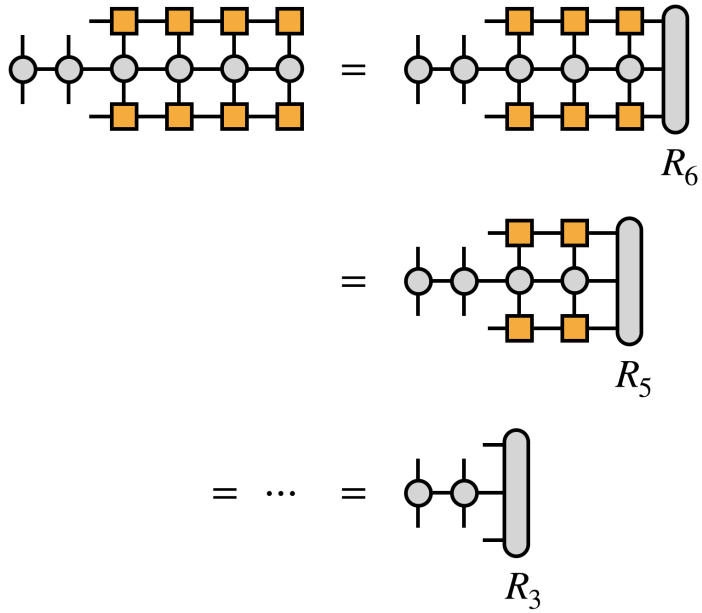


Figure: Change of basis.

2) Left and Right Environments

Once the sites have been selected, we need to contract the Hamiltonian with the orthogonal parts of the MPS and we need to store the resulting tensors, which we will call Left and Right environments.

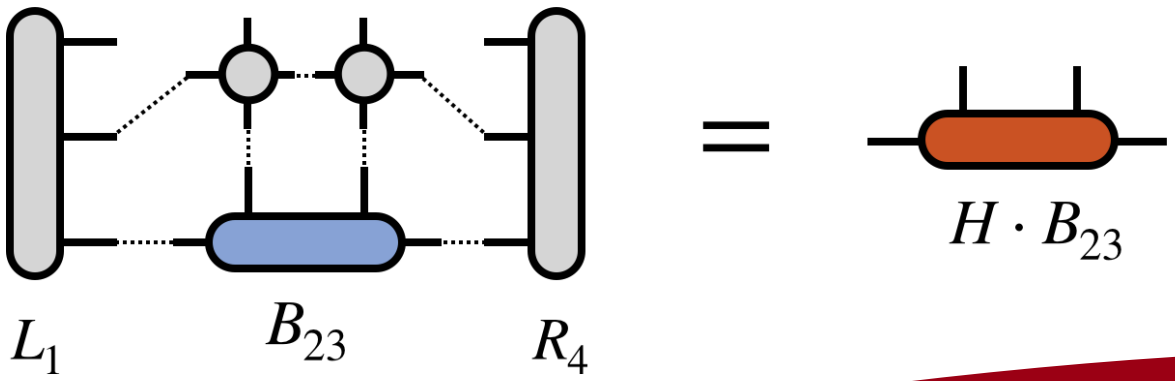




It is important to keep an eye on the order of the contractions performed on the tensor network elements, since it might weaken the efficiency of the algorithm.

3) Bond Optimization

The central passage of the algorithm consists in the actual research for the ground state. In order to do this we need to define the B tensor as the contraction of the i and $i+1$ sites of the MPS and the effective Hamiltonian as the contraction of the left and right environments with the i and $i+1$ MPO terms. The bond optimization can be performed with a range of different procedures, but they all consist into applying the effective Hamiltonian to the bond tensor in order to look for the lowest eigenstate. For our specific case we decided to exploit the Lanczos algorithm to perform this passage.



4) Restoration of MPS

The result of the optimization needs to be modified further. A Singular Value Decomposition is performed in order to split the tensor back into MPS form. This procedure of course consists in an approximation of the original tensor, since a truncation of the bond dimension needs to be done in order to limit the growth of the number of parameters.

The diagonal term of the singular values is eventually reabsorbed into one tensor and the final result is used to update the starting MPS state.

5) Sweeps

The previous steps only concentrate on a single bond. In order to eventually converge to the correct complete ground state of the quantum system we need to repeat the above passages for each bond along the tensor train.

This procedure is called Sweep and is generally performed by considering all the bonds along the chain, starting from the first MPS terms on the left proceeding towards the right and eventually moving back to the left when the last element is reached.

The sweep can be iterated multiple times in order to improve the overall convergence of the algorithm.

Overview

Model

```
236 # MPO MODEL
237 class myModel(CouplingMPOModel):
238
239     def init_sites(self, model_params):
240         n_max = model_params.get('n_max', 0)
241         filling = model_params.get('filling', 0)
242         conserve = model_params.get('conserve', 'N')
243         site = BosonSite(Nmax=n_max, conserve=conserve, filling=filling)
244         return site
245
246     def init_terms(self, model_params):
247         rc = model_params.get('rc', 0)
248         t = model_params.get('t', 1.)
249         V = model_params.get('V', 0.)
250         ryd = model_params.get('ryd', False)
251         for u1, u2, dx in self.lat.pairs['nearest_neighbors']:
252             self.add_coupling(-t, u1, 'Bd', u2, 'B', dx, plus_hc=True)
253         if ryd:
254             for dx in range(1,2*(rc+1)):
255                 self.add_coupling(V/(1+(dx/rc)**6), 0, 'N', 0, 'N', dx)
256         else:
257             for dx in range(1,rc+1):
258                 self.add_coupling(V, 0, 'N', 0, 'N', dx)
259
```

- Definition of sites as Tenpy `BosonSite` objects with fixed `filling`, `conserve` and `n_max`.
- Definition of interaction terms with fixed `rc`, `t` and `V`, with the addition of a `ryd` flag to exploit Rydberg potential.

Effective Hamiltonian

- Initialization of LP, RP, W_0 and W_1.
- *matvec* method called by Lanczos containing instructions for the contraction of H_{eff} and *theta*.

```
168 #EFFECTIVE HAMILTONIAN
169 class TwoSiteH(NpcLinearOperator):
170
171     length = 2
172     acts_on = ['vL', 'p0', 'p1', 'vR']
173
174     def __init__(self, eng, i0):
175         self.LP = eng.LPs[i0]
176         self.RP = eng.RPs[i0 + 1]
177         self.W0 = eng.H_mpo.get_W(i0).replace_labels(['p', 'p*'], ['p0', 'p0*'])
178         # 'wL', 'wR', 'p0', 'p0*'
179         self.W1 = eng.H_mpo.get_W(i0 + 1).replace_labels(['p', 'p*'], ['p1', 'p1*'])
180         # 'wL', 'wR', 'p1', 'p1*'
181         self.dtype = eng.H_mpo.dtype
182
183     def matvec(self, theta):
184         labels = theta.get_leg_labels()
185         theta = npc.tensordot(self.LP, theta, axes=['vR', 'vL'])
186         theta = npc.tensordot(self.W0, theta, axes=[['wL', 'p0*'], ['wR', 'p0']])
187         theta = npc.tensordot(theta, self.W1, axes=[['wR', 'p1'], ['wL', 'p1*']])
188         theta = npc.tensordot(theta, self.RP, axes=[['wR', 'vR'], ['wL', 'vL']])
189         theta.ireplace_labels(['vR*', 'vL*'], ['vL', 'vR'])
190         theta.itranspose(labels)
191         return theta
```

Engine: Initialization

Definition of a `DMRGEngine_Boson` class with different methods to implement the DMRG algorithm.

```
41      def __init__(self, psi, model, chi_max, eps):
42
43          self.H_mpo = model.H_MPO
44          self.psi = psi
45          self.LPs = [None] * psi.L
46          self.RPs = [None] * psi.L
47          self.chi_max = chi_max
48          self.eps = eps
49
50          # initialize left and right environment
51          D = self.H_mpo.dim[0]
52          chi = psi._B[0].shape[0]
53
54          LP = np.zeros([chi, D, chi], dtype=float)  # vR wR vR*
55          RP = np.zeros([chi, D, chi], dtype=float)  # vL* wL vL
56          LP[:, 0, :] = np.eye(chi)
57          RP[:, D - 1, :] = np.eye(chi)
58          self.LPs[0] = npc.Array.from_ndarray(data_flat=LP,
59                                               legcharges=[self.psi._B[0].conj().get_leg('vL*'),
60                                               self.H_mpo.get_W(0).conj().get_leg('wL*'),
61                                               self.psi._B[0].get_leg('vL')],
62                                               labels=['vR', 'wR', 'vR*'])
63          self.RPs[-1] = npc.Array.from_ndarray(data_flat=RP,
64                                               legcharges=[self.psi._B[-1].get_leg('vR'),
65                                               self.H_mpo.get_W(-1).conj().get_leg('wR*'),
66                                               self.psi._B[-1].conj().get_leg('vR*')],
67                                               labels=['vL*', 'wL', 'vL'])
68
69          # initialize necessary RPs
70          for i in range(psi.L - 1, 1, -1):
71              self.update_RP(i)
72
```

Engine: methods

Implemented methods: `sweep`, `update_bond`, `update_RP`, `update_LP` and `run`.

```
136     def update_RP(self, i):
137         """Calculate RP right of site `i-1` from RP right of site `i`."""
138         j = (i - 1) % self.psi.L
139         RP = self.RPs[i]  # vL* wL vL
140         B = self.psi.get_B(i, form='B')  # vL p vR
141         Bc = B.conj()  # vL* p* vR*
142         W = self.H_mpo.get_W(i) # wL wR p p*
143         RP = npc.tensordot(B, RP, axes=('vR', 'vL'))  # vL p [vR], [vL] wL vL*
144         RP = npc.tensordot(RP, W, axes=(('p', 'wL'), ('p*', 'wR')))  # vL [p] [wL] vL*, wL [wR] p [p*]
145         RP = npc.tensordot(RP, Bc, axes=(('vL*', 'p'), ('vR*', 'p*')))  # vL [vL*] wL [p], vL* [p*] [vR*]
146         self.RPs[j] = RP  # vL wL vL*
147
148     def update_LP(self, i):
149         """Calculate LP left of site `i+1` from LP left of site `i`."""
150
151         j = (i + 1) % self.psi.L
152         LP = self.LPs[i]  # vR wR vR*
153         A = self.psi.get_B(i, form='A')  # vL p vR
154         Ac = A.conj()  # vL* p* vR*
155         W = self.H_mpo.get_W(i)  # wL wR i i*
156         LP = npc.tensordot(LP, A, axes=('vR', 'vL'))  # [vR] wR vR*, [vL] i vR
157         LP = npc.tensordot(W, LP, axes=(('wL', 'p*'), ('wR', 'p')))  # [wL] wR p [p*], vR [wR] [p] vR*
158         LP = npc.tensordot(Ac, LP, axes=(('vL*', 'p*'), ('vR*', 'p')))  # [vL*] [p*] vR*, vR [p] [vR*] wR
159         self.LPs[j] = LP  # vR* wR vR (== vL wL* vL* on site i+1)
160
```

Engine: methods


```python
    def update_bond(self, i):

        j = (i + 1) % self.psi.L

        self.Heff = TwoSiteH(self, i)# Build Heff
        th = self.psi.get_theta(i, n=2)# Get theta

        # Diagonalize & find ground state
        lanczos_params = {
            'cutoff':0.001,
            'E_tol': 1.e-10,
            'N_cache':10,
            'N_min':10,
            'reortho':True
        }
        E, th, N = lanczos(self.Heff, th, lanczos_params)

        th = th.combine_legs([['vL', 'p0'], ['p1', 'vR']], qconj=[+1, -1]) # map to 2D
        old_A = self.psi.get_B(i, form='A')
        U, S, VH, err, renormalize = svd_theta(th,{'chi_max' : self.chi_max},
                                               qtotal_LR=[old_A.qtotal, None],
                                               inner_labels=['vR','vL'])

        # Manipulate and put back into MPS
        U.ireplace_label('(vL.p0)', '(vL.p)')
        VH.ireplace_label('(p1.vR)', '(p.vR)')
        A = U.split_legs(['(vL.p)'])
        B = VH.split_legs(['(p.vR)'])
        self.psi.set_B(i, A, form='A')
        self.psi.set_B(i+1, B, form='B')
        self.psi.set_SR(i, S)

        # Update Environment
        self.update_LP(i)
        self.update_RP(j)
        return E
```

Engine: methods


```
82      def sweep(self, desc_prog):
83
84          EL = ER = 0
85          # sweep from left to right
86          for i in tqdm(range(self.psi.L - 2), leave=False, desc=str(desc_prog)+'# right sweep'):
87              EL += self.update_bond(i)
88          # sweep from right to left
89          for i in tqdm(range(self.psi.L - 2, 0, -1), leave=False, desc=str(desc_prog)+'# left sweep'):
90              ER += self.update_bond(i)
91          return((EL+ER)/(2*(self.psi.L - 2)))
92
```

- Calling `update_bond` in the right order for a complete sweep.
- Perform sweep until energy minimization converges.

```
155     def run(self, params):
156         self.max_sweep = params.get('max_sweep', 5)
157         self.eps = params.get('eps', 1e-3)
158         self.V = params.get('V', 1e-6)
159         sweep_counter = 0
160         last_last_E = 2e10
161         last_E = 1e10
162         e_counter = 0
163         while True:
164             sweep_counter += 1
165             if (sweep_counter > self.max_sweep): break
166             if (e_counter > 2): break
167             last_last_E = last_E
168             last_E = self.sweep(sweep_counter)
169             self.Es.append(last_E)
170             if (abs(last_E - last_last_E) <= self.eps*self.V):
171                 e_counter +=1
172
```

Overview

Abstract

Mattioli et Al. investigated the zero-temperature phase of bosonic and fermionic gases confined to one dimension and interacting via a class of finite-range soft-shoulder potentials. They demonstrated the stabilization of critical quantum liquids with qualitatively new features with respect to the Tomonaga-Luttinger liquid paradigm. These features result from frustration and cluster formation in the corresponding classical ground-state.

Hard-core Boson Hamiltonian

The relevant microscopic Hamiltonian for hard-core bosons in 1D geometry reads

$$H = -t \sum_i (b_i^\dagger b_{i+1} + h.c.) + V \sum_i \sum_{l=1}^{r_c} n_i n_{i+l}$$

The soft shoulder potential can be engeneered in clouds of cold Rydberg atoms, where both the strenght V and the range r_c can be properly tuned.

$$\rho = \begin{cases} \bar{n} & \text{if } \bar{n} \leq 1/2 \\ 1 - \bar{n} & \text{if } \bar{n} > 1/2 \end{cases}$$

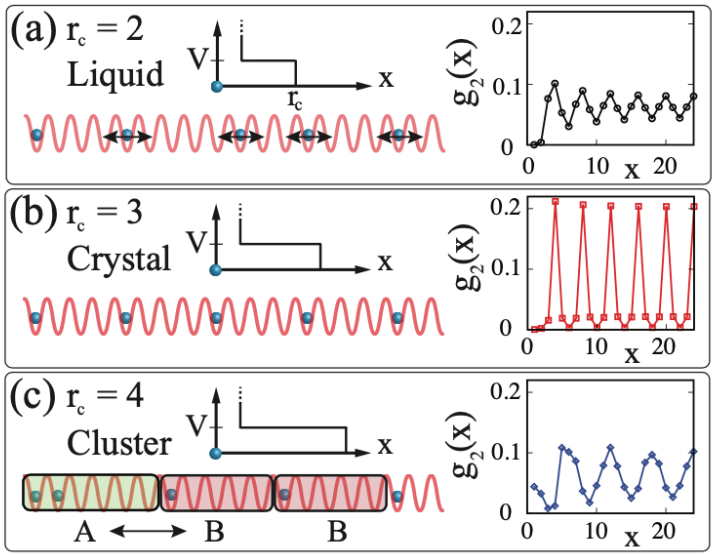
where $\bar{n}$ is the particle density.

Liquid, Crystal and Cluster

The competition of r_c with the critical length $r^* = 1/\rho - 1$ leads to frustration for $r_c > r^*$. This effects result in the formation of highly degenerate cluster-type ground-states in the classical limit ($t = 0$), as well as in novel phases in the quantum regime ($t > 0$). Note that all correlation functions are particle-hole symmetric, such that the phases for $\bar{n}$ and $1 - \bar{n}$ are identical.

The figure shows sketches of the possible ground-states for a system with $\bar{n} = 1/4$ ($r^* = 3$) and $t = 0$. These phases are:

- **liquid** ($r_c < r^*$): the inter-particle distances are larger then r_c;
- **crystal** ($r_c = r^*$): the phase has a particle every $r_c + 1$ sites;
- **cluster** ($r_c > r^*$): this phase is highly degenerates, particles and holes are grouped into blocks of type A and B consisting of two and one particles, respectively, followed by a number of r_c empty sites.



Liquid, Crystal and Cluster

On the insets of the previous figure the authors reported the numerical computations of density-density correlations $g_2(l-j)$ for $V/t = 6$ in the corresponding quantum phases.

$$g_2(l-j) = \langle n_l n_j \rangle - \bar{n}^2$$

where j, l are the indexes of the system's sites.

For $r_c = 4$ in the quantum regime, the arrangements of the particles in clusters, results in a cluster-type liquid with correlation functions different from a standard TL liquid

Observables

The authors characterize the phase diagram focussing on:

- **Static structure factor**: $S(k) = \sum_{l,j} e^{ik(l-j)} g_2(l-j)/L$.
- **Excitation spectrum**: $\Omega(k) = E_k/S(k)$,
  which is an upper bound of the complete spectrum.
- **Momentum distribution**: $n(k) = \sum_{l,k} e^{ik(l-j)} B(l-j)/L$.

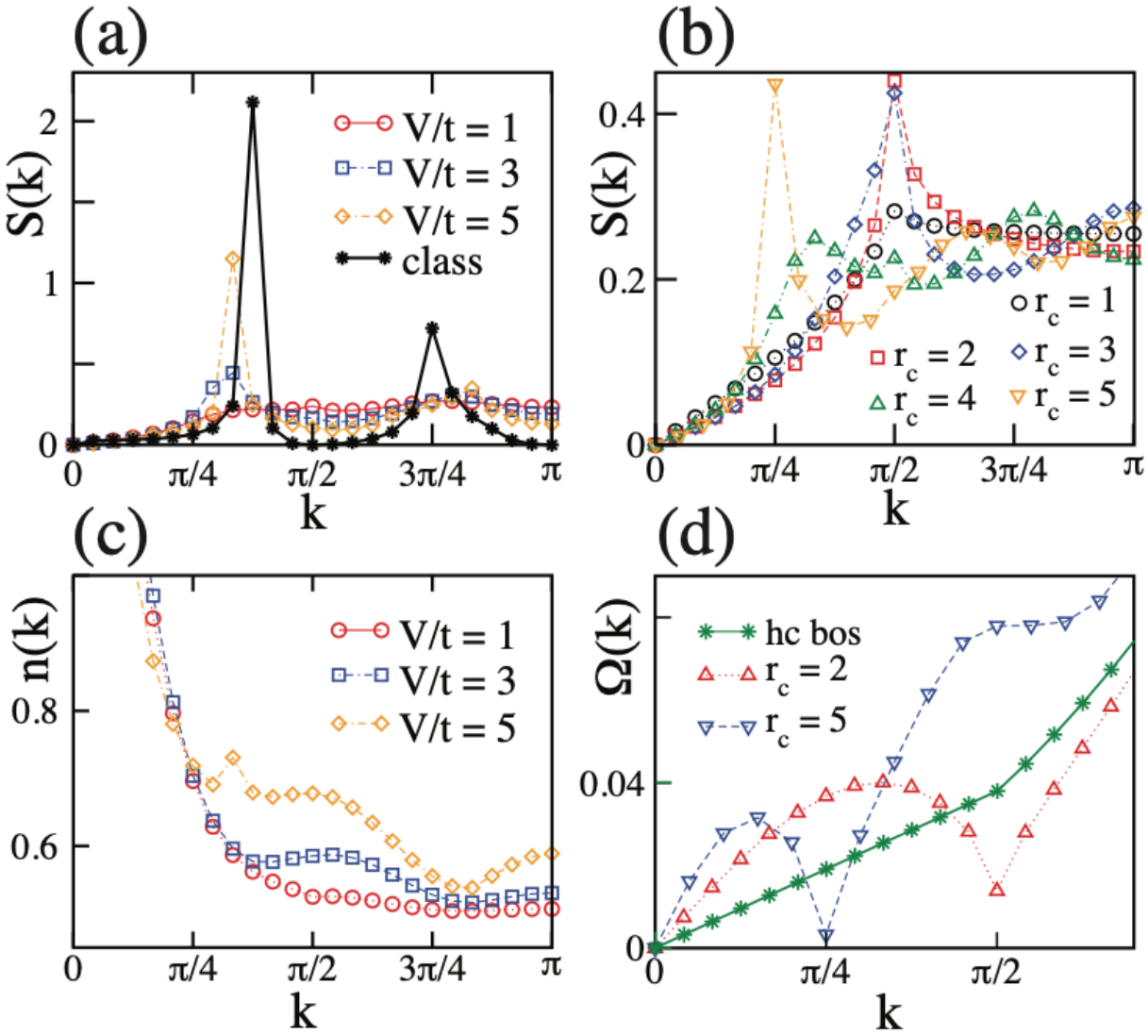
Where

$$E_k = \frac{t}{L}\left[1 - \cos\left(\frac{2\pi k}{L}\right)\right] \langle 0| \sum_i (b_i^\dagger b_{i+1} + h.c.)|0\rangle$$

and

$$B(l-j) = \langle b_l^\dagger b_j \rangle.$$

Results


(a)
S(k)
V/t = 1
V/t = 3
V/t = 5
class
0
1
2
0
π/4
π/2
3π/4
π
k
(b)
S(k)
0.4
0.2
0
r_c = 1
r_c = 2
r_c = 3
r_c = 4
r_c = 5
0
π/4
π/2
3π/4
π
k
(c)
n(k)
0.8
0.6
V/t = 1
V/t = 3
V/t = 5
0
π/4
π/2
3π/4
π
k
(d)
Ω(k)
0.04
0
hc bos
r_c = 2
r_c = 5
0
π/4
π/2
k

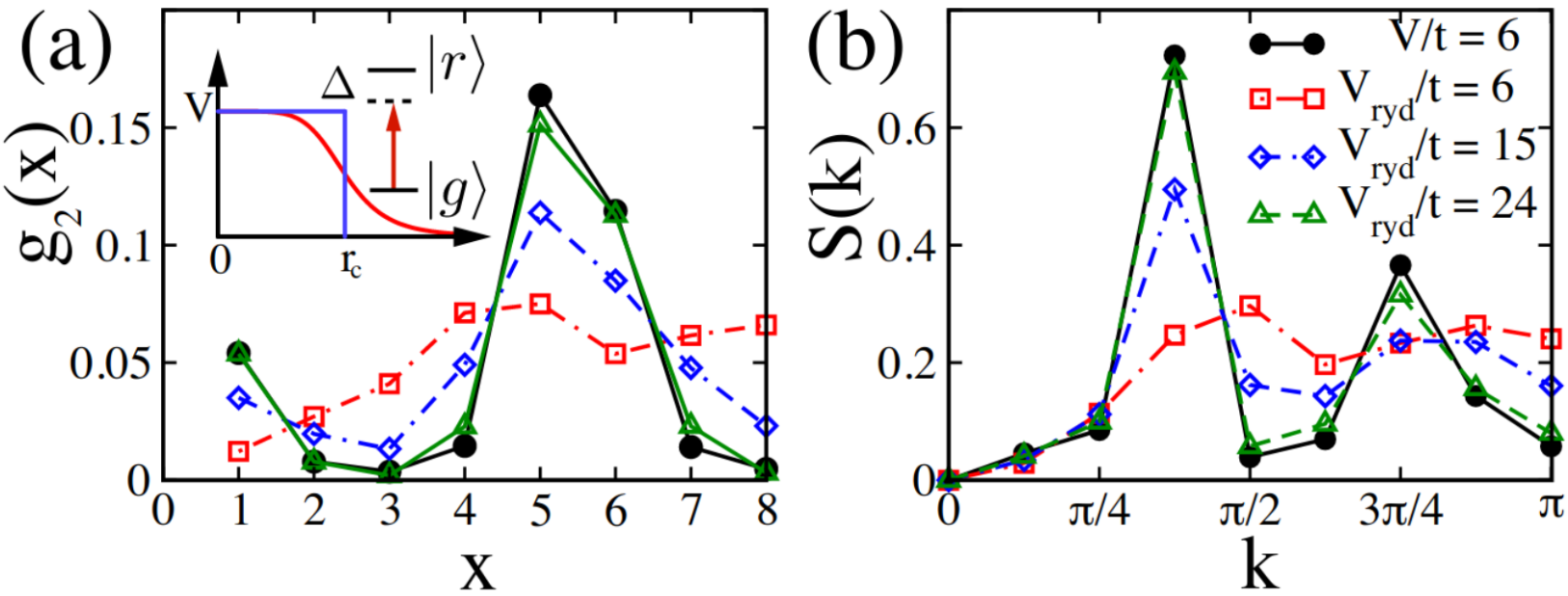
Results

- Differences between TL and cL liquids are evident in $S(k)$: in particular, in the TL phase, $S(k)$ exhibits a peak at $k_c = \pi/2$, commensurate with particle density, while in the cL phase, the exact position depending on r_c.
- In (a) and (b) the authors demonstrated the crossover from TL to cL liquid increasing the strength of interaction V/t at fixed $r_c > r^*$ ad as a function of r_c at fixed V/t, respectively.
- From (c) it can be seen that the cL phase leads to the emergence of new peaks in the momentum distribution.
- The excitation spectrum depicted in (d) shows that the finite-range interactions within the regime $r_c < r^*$ induce a roton excitation at $k_c = \pi/2$, commensurate with density excitation, as expected. For $r_c > r^*$, however, $\Omega(k)$ develops a new roton-type excitation at $k_c < \pi/2$, reflecting the structure of the classical cluster ground-state.

Results

Similar results can be obtained for a Born-Oppenheimer potential: $V_{ryd}(i,j) = V_{ryd}/\{1 + [(i-j)a/r_c]^6\}$, realizable with cold Rydberg atoms.

The results of $g_2(x)$ and $S(k)$ for the two kind of potentials converge for $V_{ryd}/V \gtrsim 2$.



Overview

Remarks

We here reproduced the results of the paper, all the calculations are done with our DMRG algorithm. In all simulations we considered a filling $\bar{n} = 3/4$, unless otherwise stated.

In order to avoid boundary effects, which can be significant due to the finite range interactions, we imposed periodic boundary conditions. We refer to the classical case as the one with $V >> t$.

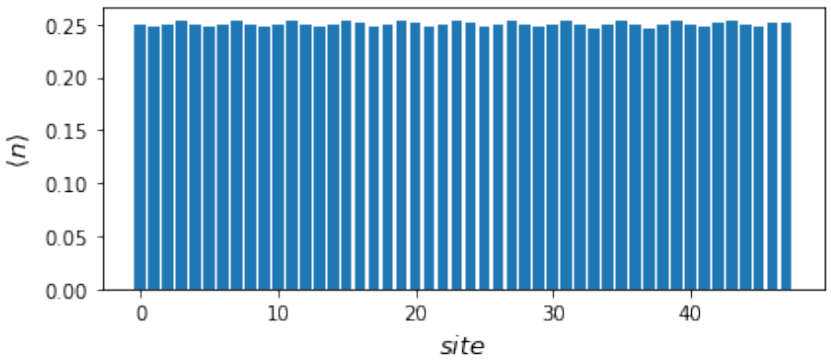
While in the regime $r_c \leq r^*$ all lengths $L = 4l, l \in \mathbb{N}$ are appropriate for finite-size scaling, this is not true anymore in the region where effects due to the cluster structure could emerge. For $r_c = 4$ this impose $L = 16, 32, 46, 64$, while for $r_c = 5$, $L = 20, 40, 60$.

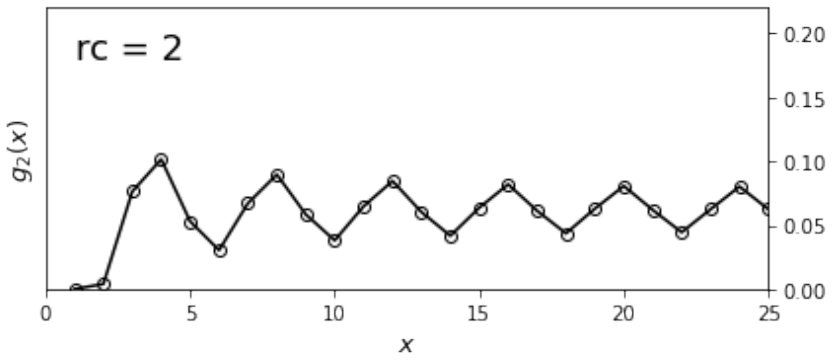
Ground state and g_2

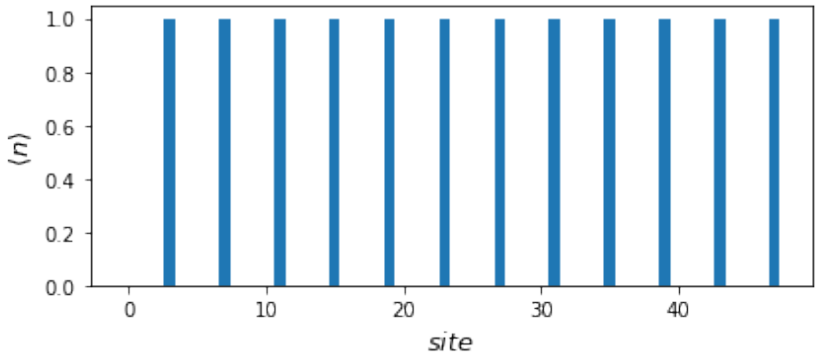
Imposing $V/t = 100$ we calculated the expectation value of the number operator on the ground state, on order to have the occupation of the sites for a system with $L = 48$ and $\bar{n} = 1/4$.

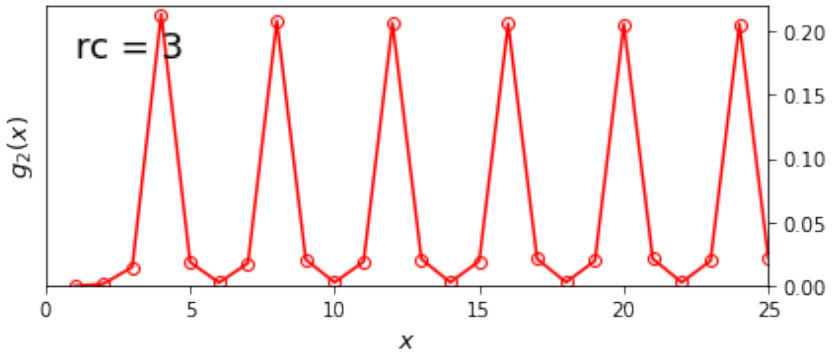
For the same system we computed the density-density correlations $g_2(l - j)$ for $V/t = 6$.

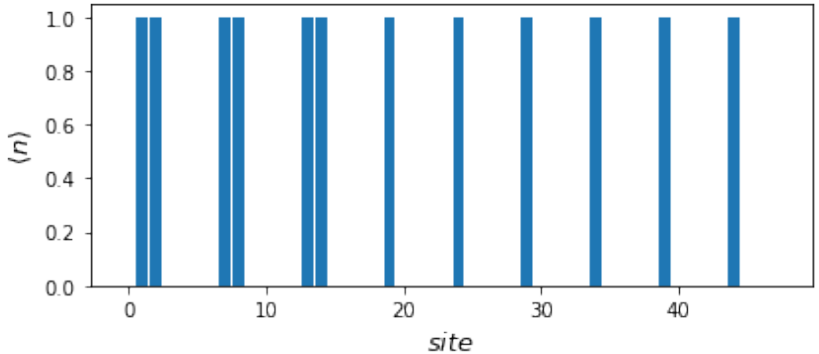
Ground state and g_2

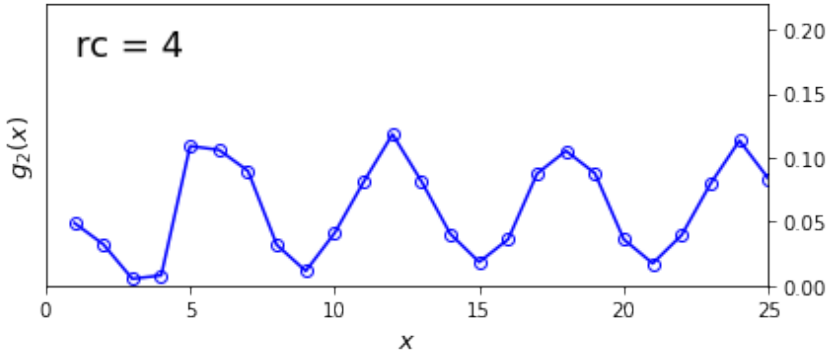






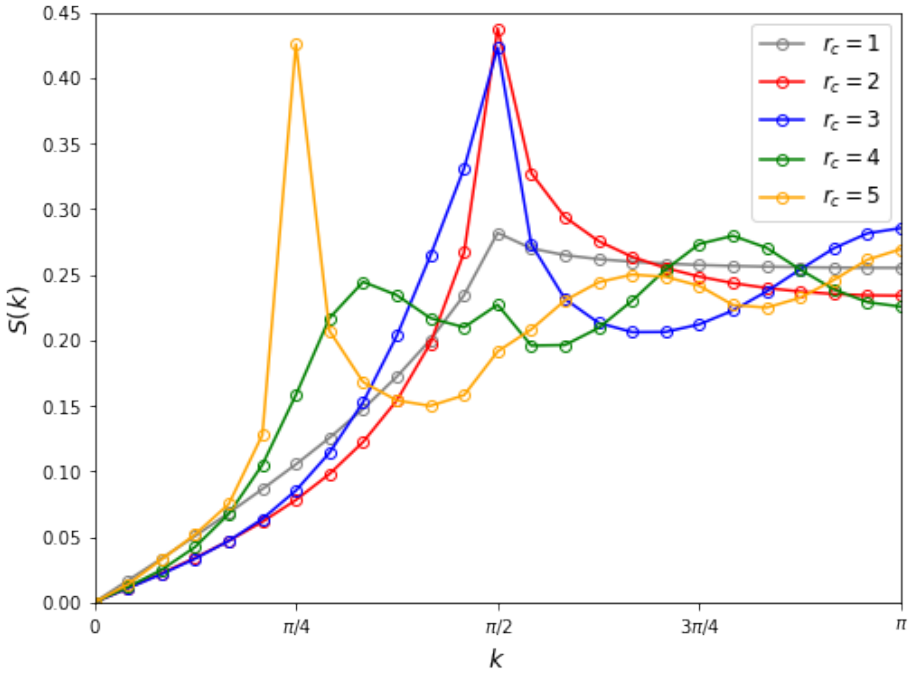




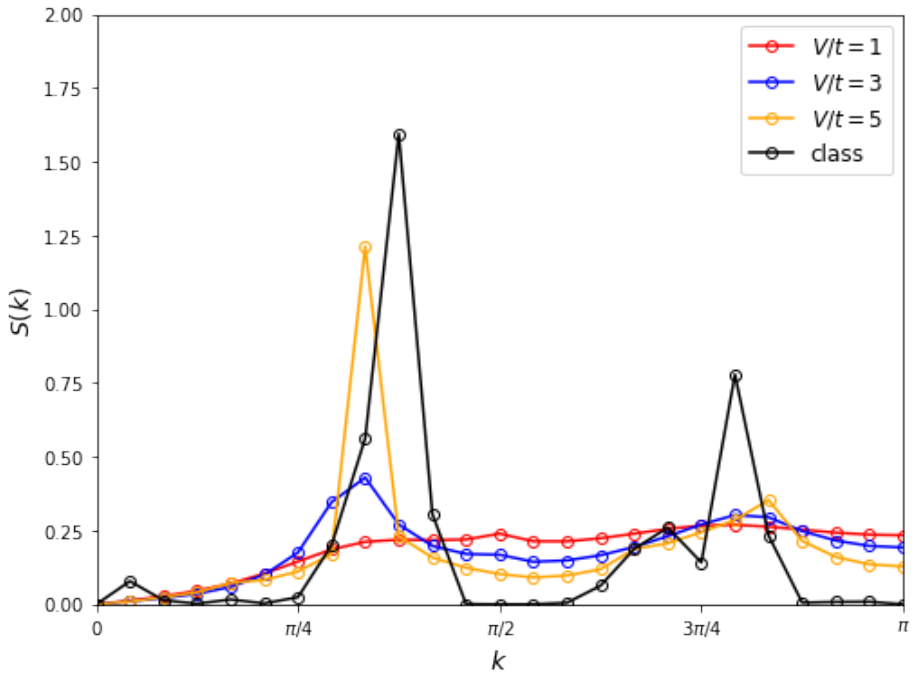


Structure factor $S(k)$, $V/t = 1.5$

We computed the structure factor while increasing the interaction length r_c, at fixed strength of interaction V/t. It can be clearly seen that in the TL phase, for $r_c \leq r*$, the function is peaked at $k_c = \pi/2$, while in the cluster-Luttinger phase the exact position of the peak depends on r_c.

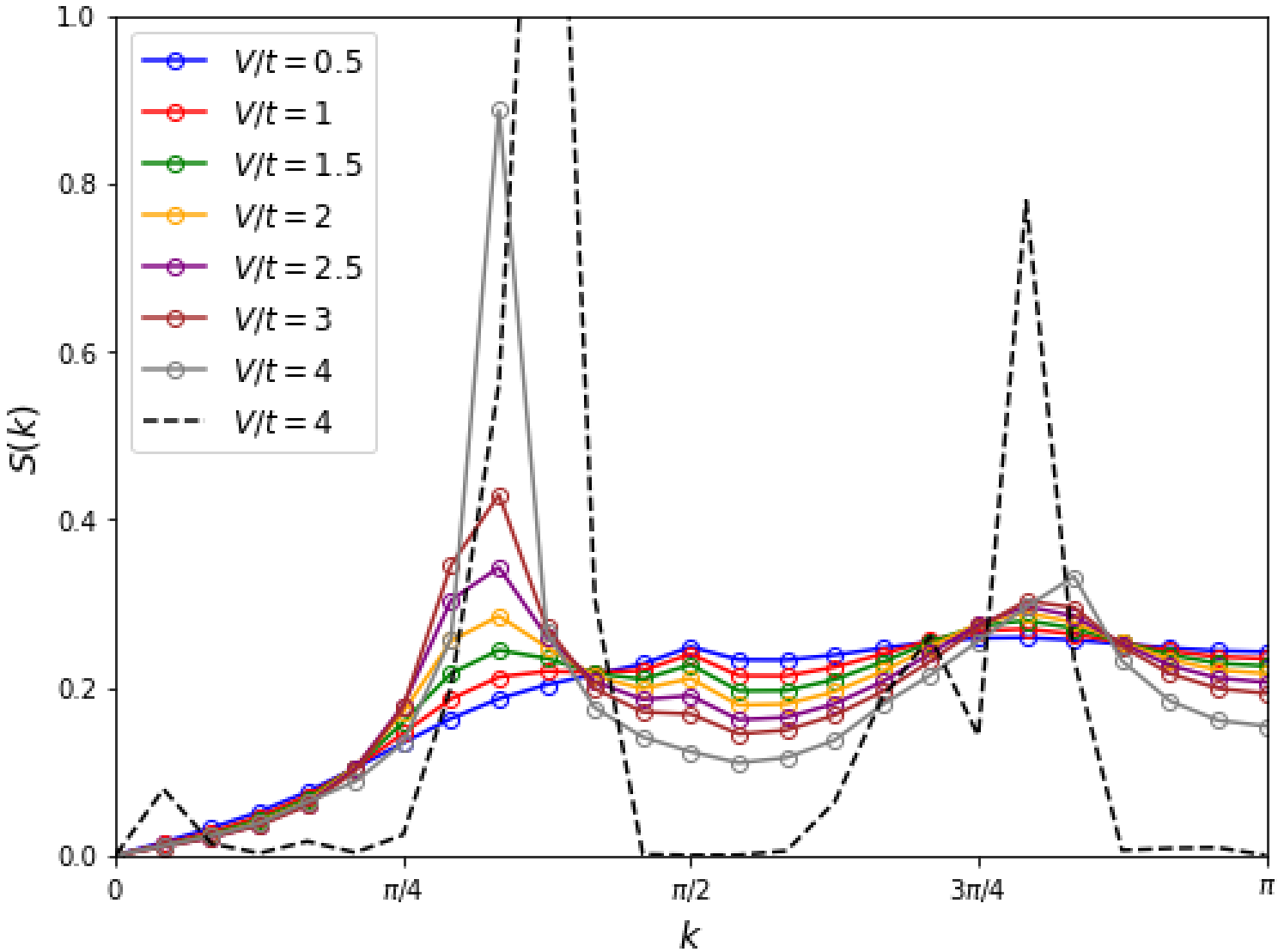


Structure factor $S(k)$, $r_c = 4$



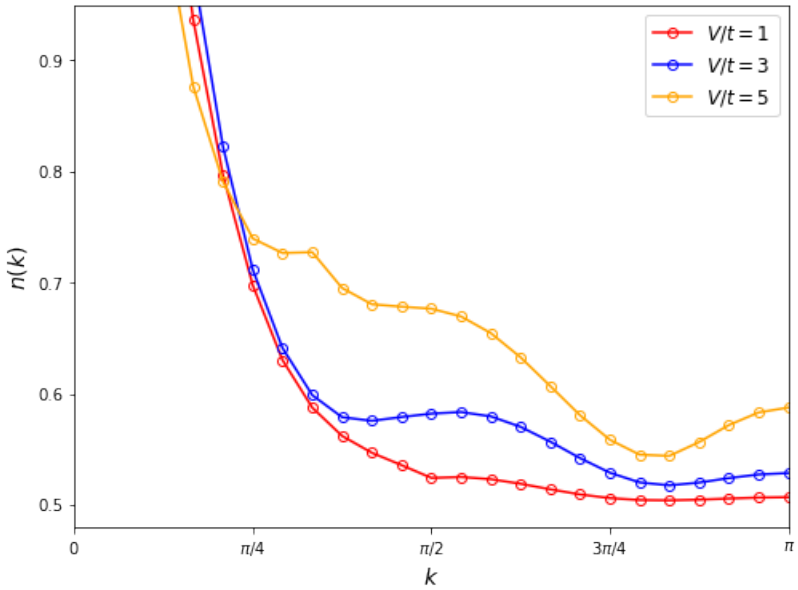
Here we increased the strength of interaction, at fixed $r_c > r^*$. It can be seen that for sufficiently high V/t the structure factor is peaked on $k_c < \pi/2$. The transition from the Luttinger liquid to the cluster-Luttinger liquid for $r_c = 4$ can be better appreciate in the following plot.

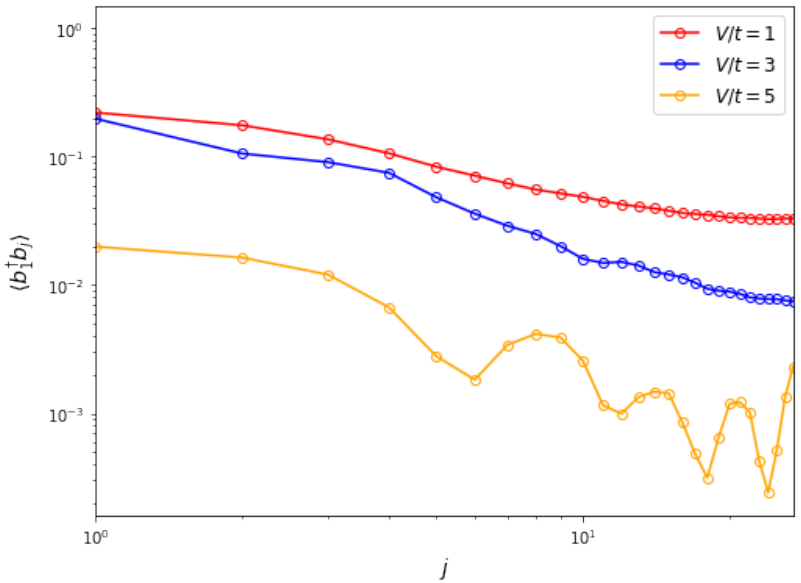
Structure factor $S(k)$, $r_c = 4$



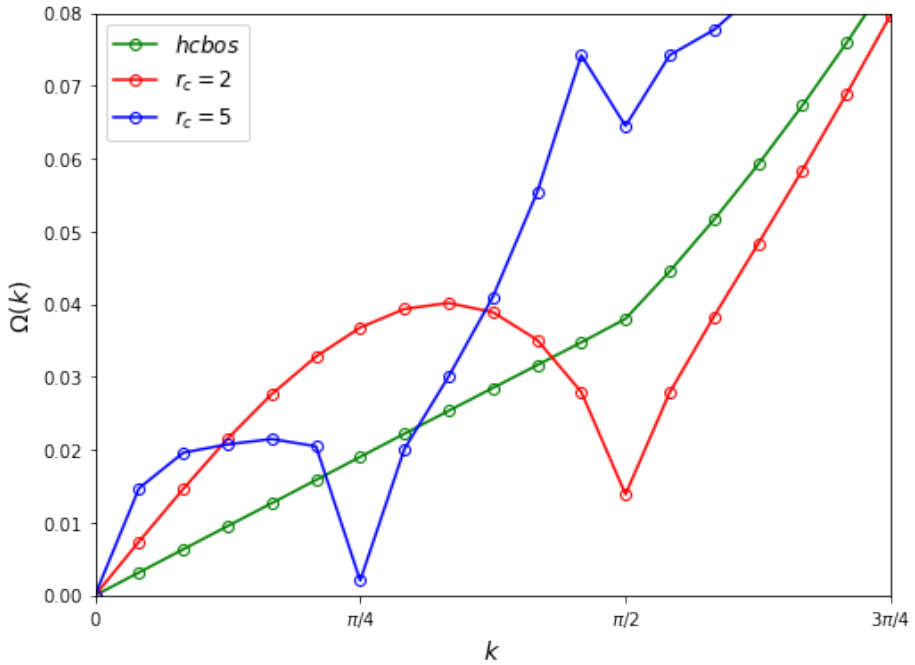
Momentum distribution $n(k)$, $r_c = 4$

The anomalous peak in the momentum distribuion $n(k)$ at non-zero k-vector for $r_c = 4 > r^*$, is associated to large oscillations in the corresponding one-body density matrix $B(l - j)$.
This asymptotic behaviour emerges for sufficiently large interactions ($V/t = 5$).





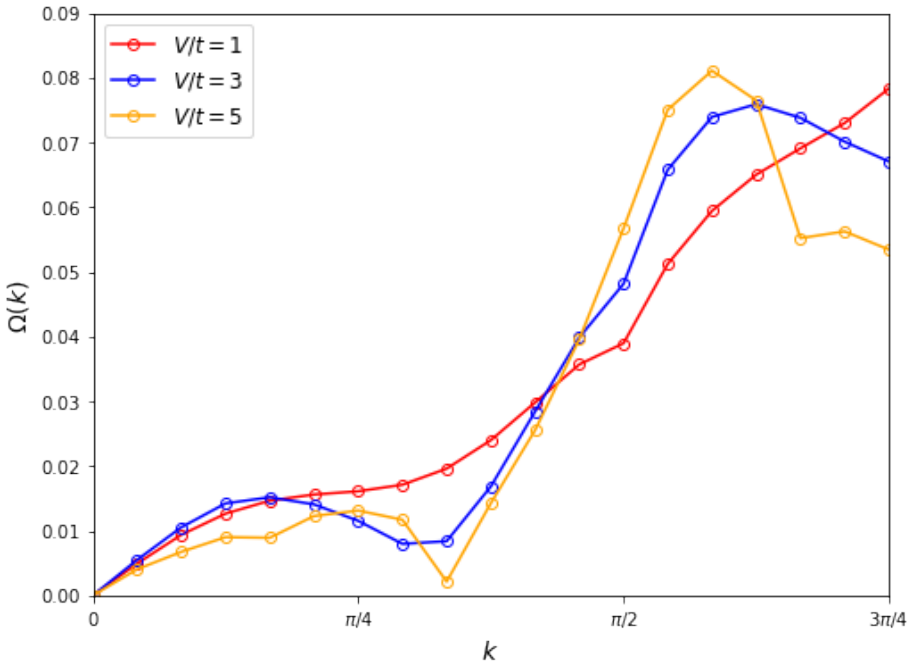
Excitation spectrum $\Omega(k)$, $V/t = 3$



We verified that for $r_c > r^*$, the excitation spectrum develop a new roton-type excitation at $k_c < \pi/2$.

Excitation spectrum $\Omega(k)$, $r_c = 4$

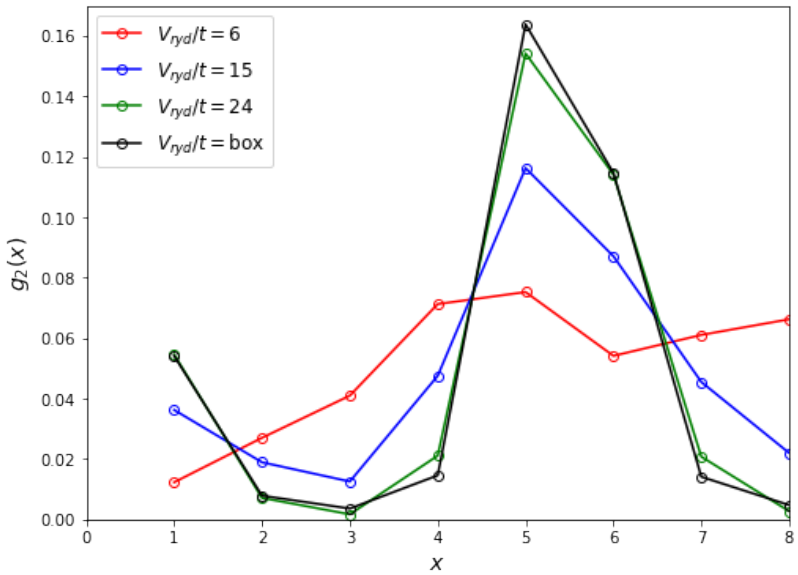
Here we show the formation and the evolution of the roton instability for increasing interaction strength V/t.
It is worth to notice that the roton critical momentum k_c is identical to both the static structure factor and momentum distribution peaks.

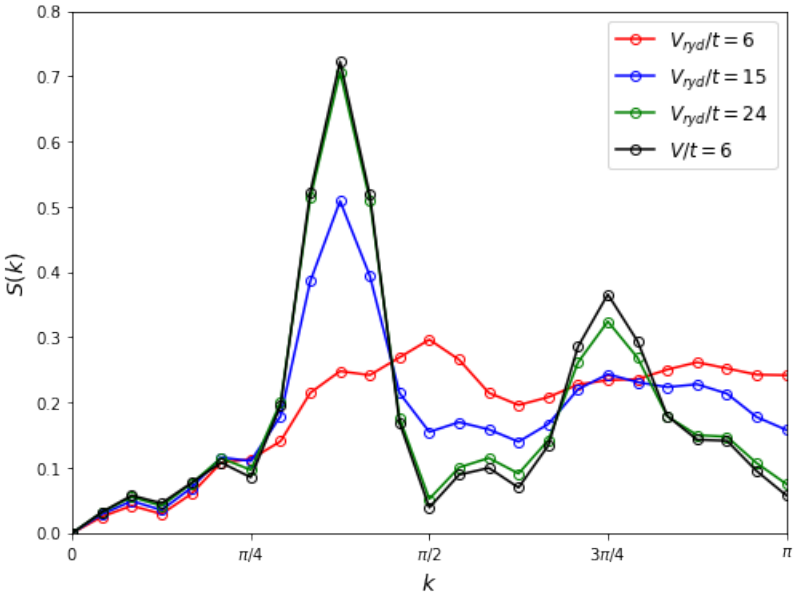


Rydberg potential

Here we applied the DMRG algorithm to a system with $L = 16$ and $r_c = 4$, modelling the interaction by the means of the Born-Oppenheimer potential.

It can be appreciated the convergence of the observables for the different interaction models (given $V_{ryd}/V \gtrsim 2$).





Thanks for your attention!

References I

1. Figures taken from: https://tensornetwork.org
2. M. Mattioli et al., "Cluster Luttinger Liquids of Rydberg-dressed Atoms in Optical Lattices " arXiv:1304.3012
3. TenPy: https://tenpy.readthedocs.io/en/latest/