Sep 22, 2025

Selected Topics in Computational Quantum Physics

量子物理计算方法选讲

Shuo Yang（杨硕）

Department of Physics, Tsinghua University

Email: shuoyang@tsinghua.edu.cn

WeChat: condmat-ys

Exercise last time

- 3-site Heisenberg model

$$H = \mathbf{S}_1 \cdot \mathbf{S}_2 + \mathbf{S}_2 \cdot \mathbf{S}_3$$

find out all eigenvalues and eigenstates

- answer:

$$\begin{aligned} H &= \mathbf{S}_1 \cdot \mathbf{S}_2 + \mathbf{S}_2 \cdot \mathbf{S}_3 \\ &= S_1^x \otimes S_2^x \otimes \mathbb{I}_3 + S_1^y \otimes S_2^y \otimes \mathbb{I}_3 + S_1^z \otimes S_2^z \otimes \mathbb{I}_3 \\ &+ \mathbb{I}_1 \otimes S_2^x \otimes S_3^x + \mathbb{I}_1 \otimes S_2^y \otimes S_3^y + \mathbb{I}_1 \otimes S_2^z \otimes S_3^z \end{aligned}$$

$$S^x = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, S^y = \frac{1}{2}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},$$
$$S^z = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

$$H = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$H|\psi\rangle = E|\psi\rangle$$

eigenvalues:

$$E = -1, -1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0$$

Sample code

```
import numpy as np
import numpy.linalg as LA
```

```
def SpinOper(ss):
    spin = (ss-1)/2.0
    dz = np.zeros(ss)
    mp = np.zeros(ss-1)

    for i in range(ss):
        dz[i] = spin-i
    for i in range(ss-1):
        mp[i] = np.sqrt((2*spin-i)*(i+1))

    S0 = np.eye(ss)
    Sp = np.diag(mp,1)
    Sm = np.diag(mp,-1)
    Sx = 0.5*(Sp+Sm)
    Sy = -0.5j*(Sp-Sm)
    Sz = np.diag(dz)

    return S0,Sp,Sm,Sz,Sx,Sy
```

```
S0,Sp,Sm,Sz,Sx,Sy = SpinOper(2)
print(Sp)
print(Sx)
print(Sz)
```

```
[[0. 1.]
 [0. 0.]]
[[0.  0.5]
 [0.5 0. ]]
[[ 0.5  0. ]
 [ 0.  -0.5]]
```

```
H2 = np.kron(Sx,Sx)+np.kron(Sy,Sy)+np.kron(Sz,Sz)
H3 = np.kron(H2,S0)+np.kron(S0,H2)
```

```
S,V = LA.eigh(H3)
print(np.sort(S))
```

```
[-1.00000000e+00 -1.00000000e+00 -5.43456339e-17
-5.43456339e-17
  5.00000000e-01  5.00000000e-01  5.00000000e-01
5.00000000e-01]
```

Exact Diagonalization (ED)

$$H|\Psi\rangle = E|\Psi\rangle$$

- main idea
- matrix representation of Hamiltonian
- implement symmetries
- Lanczos method
- time evolution
- finite temperature
- various applications

Why exact diagonalization?

- complete knowledge of a quantum system can be obtained

  with the eigenstates available, any static or dynamic quantity can be computed

- insights gained from exact diagonalization studies are very useful

  in there own right and as a complement to other calculations

- exact results for small lattices are indispensable for testing the correctness of other methods

  e.g., DMRG, QMC, TN, etc

- exact diagonalization methods provide a concrete path for learning many important aspects of quantum mechanics

  in particular the symmetry properties of many-body states

Hilbert space sizes

- the Hilbert space of a quantum many body system grows exponentially in general

- for N spin 1/2 particles, the complete Hilbert space has dim = 2^N states

  10 spins dim = 1'024

  30 spins dim = 1'073'741'824

  50 spins dim = 1'125'899'906'842'624 ...

- the quantum mechanical wave function is a vector in this Hilbert (vector) space and we would like to know the ground state and a few other low lying eigenstates

- exact diagonalization studies are limited to rather small lattices

  great care therefore has to be taken in drawing conclusions about thermodynamic limit, which may not even be possible

Exact diagonalization: applications

- quantum magnets: nature of novel phases, critical points in 1D, dynamical correlation functions in 1D & 2D

- fermionic models (Hubbard / *t*-*J*): gaps, pairing properties, correlation exponents, etc

- fractional quantum Hall states: energy gaps, overlap with model states, entanglement spectra

- quantum dimer models or other constrained models (anyon chains, ...)

- full Configuration Interaction (CI) in quantum chemistry, nuclear structure

- quantum field theory

Exact diagonalization: present day limits

- spin $S = 1/2$ models

  square lattice: $N = 40$, triangular lattice: $N = 39$, Honeycomb lattice: $N = 42$, kagome lattice: $N = 48$, maximum dimension of basis: 1.5 billion

- fractional quantum hall effect

  different filling fractions ν, up to 16-20 electrons

  maximum dimension of basis: 3.5 billion

- Hubbard models

  square lattice at half filling: $N = 20$, triangular lattice: $N = 21$, honeycomb lattice: $N = 24$, quantum dot structure: $N = 20$

  maximum dimension of basis: 3 billion

- Holstein models

  chain with $N = 14$ + phonon pseudo-sites

  maximum dimension of basis: 30 billion

Structure of an exact diagonalization code

- Hilbert space

  basis representation, lookup techniques

  symmetries

- Hamiltonian matrix

  sparse matrix representation (memory/disk)

  matrix recalculation on the fly (matrix-free)

- linear algebra : eigensolver / time propagation

  LAPACK full diagonalization

  Lanczos type diagonalization (needs only operations)

- observables

  static quantities (correlation functions, correlation density matrices, ...)

  dynamic observables (spectral functions, density of states, …)

  real-time evolution

Basis states

spin-1/2, 4-sites

$|\uparrow\rangle \rightarrow |1\rangle\,, |\downarrow\rangle \rightarrow |0\rangle$

binary to decimal

$\|\downarrow\downarrow\downarrow\downarrow\rangle$	$\|0000\rangle$	$\|0\rangle$
$\|\downarrow\downarrow\downarrow\uparrow\rangle$	$\|0001\rangle$	$\|1\rangle$
$\|\downarrow\downarrow\uparrow\downarrow\rangle$	$\|0010\rangle$	$\|2\rangle$
$\|\downarrow\downarrow\uparrow\uparrow\rangle$	$\|0011\rangle$	$\|3\rangle$
$\|\downarrow\uparrow\downarrow\downarrow\rangle$	$\|0100\rangle$	$\|4\rangle$
$\|\downarrow\uparrow\downarrow\uparrow\rangle$	$\|0101\rangle$	$\|5\rangle$
$\|\downarrow\uparrow\uparrow\downarrow\rangle$	$\|0110\rangle$	$\|6\rangle$
$\|\downarrow\uparrow\uparrow\uparrow\rangle$	$\|0111\rangle$	$\|7\rangle$
$\|\uparrow\downarrow\downarrow\downarrow\rangle$	$\|1000\rangle$	$\|8\rangle$
$\|\uparrow\downarrow\downarrow\uparrow\rangle$	$\|1001\rangle$	$\|9\rangle$
$\|\uparrow\downarrow\uparrow\downarrow\rangle$	$\|1010\rangle$	$\|10\rangle$
$\|\uparrow\downarrow\uparrow\uparrow\rangle$	$\|1011\rangle$	$\|11\rangle$
$\|\uparrow\uparrow\downarrow\downarrow\rangle$	$\|1100\rangle$	$\|12\rangle$
$\|\uparrow\uparrow\downarrow\uparrow\rangle$	$\|1101\rangle$	$\|13\rangle$
$\|\uparrow\uparrow\uparrow\downarrow\rangle$	$\|1110\rangle$	$\|14\rangle$
$\|\uparrow\uparrow\uparrow\uparrow\rangle$	$\|1111\rangle$	$\|15\rangle$

3^{rd} ← → 0^{th}

Bit operations

- bitwise operations

x & y　　x | y　　x ^ y　　~ x

```
AND | 0 1     OR | 0 1     XOR | 0 1    NOT | 0 1
----+-----    ---+----    ----+----    ----+----
 0  | 0 0      0 | 0 1       0 | 0 1        | 1 0
 1  | 0 1      1 | 1 1       1 | 1 0
```

```
    201: 1100 1001
AND  15: 0000 1111
------------------
 IS   9  0000 1001
```

- bit shift

x << y
x >> y

```
x = 1          # 0001
x << 2         # Shift left 2 bits: 0100
# Result: 4
```

- show bit configuration

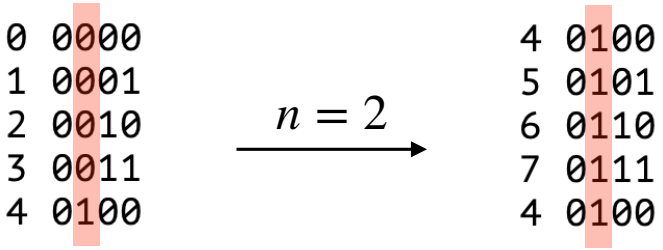
```
>>> np.binary_repr(3)
'11'
>>> np.binary_repr(-3)
'-11'
>>> np.binary_repr(3, width=4)
'0011'
```

```
>>> np.base_repr(5)
'101'
>>> np.base_repr(6, 5)
'11'
>>> np.base_repr(7, base=5, padding=3)
'00012'
```

Bit operations

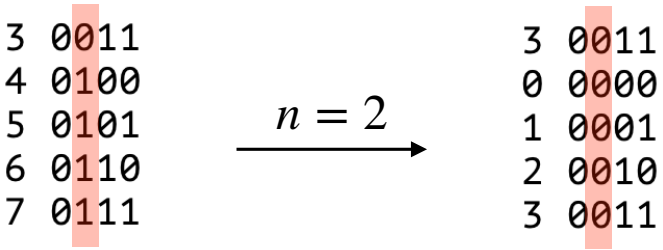
- set n^{th} bit to 1

```
def SetBit(i,n):
    return i|(1<<n)
```



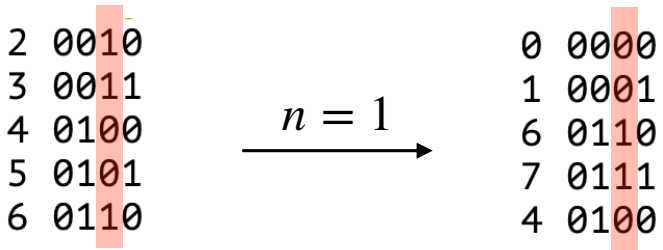
- clear n^{th} bit to 0

```
def ClearBit(i,n):
    return i&~(1<<n)
```



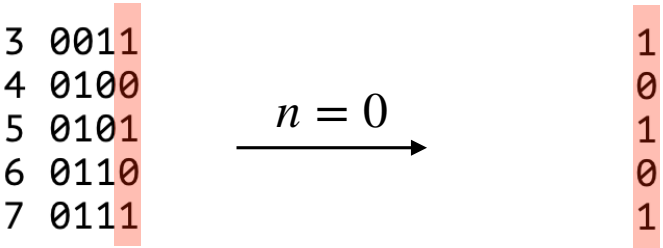
- flip n^{th} bit

```
def FlipBit(i,n):
    return i^(1<<n)
```



- read n^{th} bit

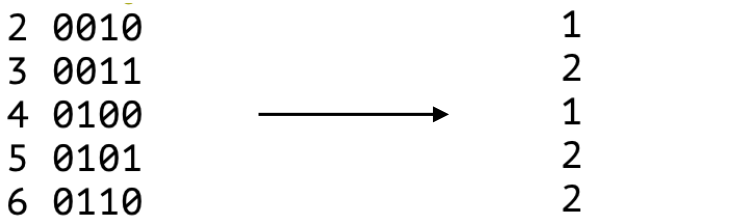
```
def ReadBit(i,n):
    return (i&(1<<n))>>n
```



Bit operations

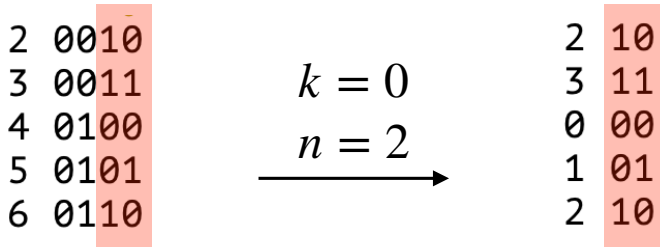
- count how many 1 bits

```
def PopCntBit(i):
    return bin(i).count("1")
```



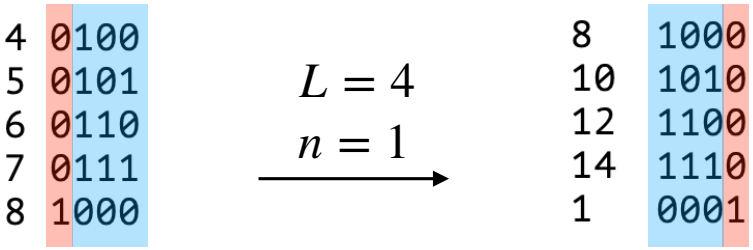
- pick up n bits from k^{th} bit

```
def PickBit(i,k,n):
    return (i&((2**n-1)<<k))>>k
```



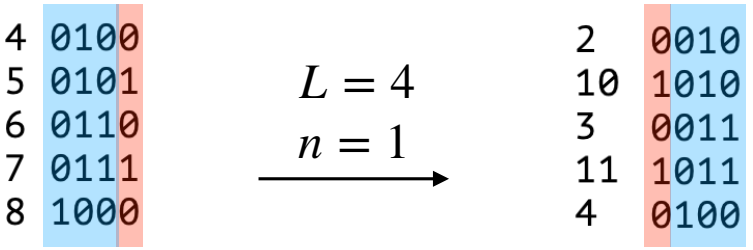
- circular bit shift left

```
def RotLBit(i,L,n):
    return (PickBit(i,0,L-n)<<n)+(i>>(L-n))
```



- circular bit shift right

```
def RotRBit(i,L,n):
    return (PickBit(i,0,n)<<(L-n))+(i>>n)
```



Spin system - Hamiltonian matrix

- spin-1/2 AF Heisenberg model on a ring

$$H = \sum_{i=0}^{N-1} \mathbf{S}_i \cdot \mathbf{S}_{i+1} = \sum_{i=0}^{N-1} \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z \right)$$
$$= \sum_{i=0}^{N-1} \left[\left(S_i^+ S_{i+1}^- + S_i^- S_{i+1}^+ \right)/2 + S_i^z S_{i+1}^z \right]$$

- periodic boundary condition $S_N = S_0$

- diagonal terms $\langle \mathrm{col} | S_i^z S_{i+1}^z | \mathrm{col} \rangle$

$${}_i\langle \uparrow | S_i^z | \uparrow \rangle_i = +0.5, \quad {}_i\langle \downarrow | S_i^z | \uparrow \rangle_i = 0, \quad \Rightarrow \quad {}_i\langle 1 | S_i^z | 1 \rangle_i = +0.5, \quad {}_i\langle 0 | S_i^z | 1 \rangle_i = 0,$$

$${}_i\langle \downarrow | S_i^z | \downarrow \rangle_i = -0.5, \quad {}_i\langle \uparrow | S_i^z | \downarrow \rangle_i = 0, \quad \Rightarrow \quad {}_i\langle 0 | S_i^z | 0 \rangle_i = -0.5, \quad {}_i\langle 1 | S_i^z | 0 \rangle_i = 0,$$

$$\langle \mathrm{col} | S_i^z S_{i+1}^z | \mathrm{col} \rangle = \langle S_i^z \rangle \langle S_{i+1}^z \rangle$$

$$\langle S_i^z \rangle = \mathrm{ReadBit(col, i)} - 0.5$$

Off-diagonal terms

- matrix elements $\left\langle \text{row} \left| \left(S_i^+ S_{i+1}^- + S_i^- S_{i+1}^+ \right)/2 \right| \text{col} \right\rangle$

$$_{i+1}\langle\downarrow|_i\langle\uparrow|S_i^+S_{i+1}^-|\downarrow\rangle_i|\uparrow\rangle_{i+1} = 1, \qquad _{i+1}\langle\downarrow|_i\langle\uparrow|S_i^-S_{i+1}^+|\downarrow\rangle_i|\uparrow\rangle_{i+1} = 0,$$

$$_{i+1}\langle\uparrow|_i\langle\downarrow|S_i^+S_{i+1}^-|\uparrow\rangle_i|\downarrow\rangle_{i+1} = 0, \qquad _{i+1}\langle\uparrow|_i\langle\downarrow|S_i^-S_{i+1}^+|\uparrow\rangle_i|\downarrow\rangle_{i+1} = 1,$$

- to have non-zero contributions

  the i-th spin and the $(i+1)$-th spin must be different

  one need to flip the i-th spin and the $(i+1)$-th spin simultaneously

$$_{i+1}\langle 0|_i\langle 1|\left(S_i^+S_{i+1}^- + S_i^-S_{i+1}^+\right)/2|0\rangle_i|1\rangle_{i+1} = 0.5$$

$$_{i+1}\langle 1|_i\langle 0|\left(S_i^+S_{i+1}^- + S_i^-S_{i+1}^+\right)/2|1\rangle_i|0\rangle_{i+1} = 0.5$$

Off-diagonal terms

- matrix elements $\left\langle \text{row} \left| \left(S_i^+ S_{i+1}^- + S_i^- S_{i+1}^+ \right)/2 \right| \text{col} \right\rangle$

$(\text{col})_{10}$	$(\text{col})_2$	i	$i+1$	$(\text{row})_2$	$(\text{row})_{10}$	value
1	0001	0	1	0010	2	0.5
1	0001	3	0	1000	8	0.5
2	0010	0	1	0001	1	0.5
2	0010	1	2	0100	4	0.5
3	0011	1	2	0101	5	0.5
3	0011	3	0	1010	10	0.5
4	0100	1	2	0010	2	0.5
4	0100	2	3	1000	8	0.5
5	0101	0	1	0110	6	0.5
5	0101	1	2	0011	3	0.5
5	0101	2	3	1001	9	0.5
5	0101	3	0	1100	12	0.5
6	0110	0	1	0101	5	0.5
6	0110	2	3	1010	10	0.5
7	0111	2	3	1011	11	0.5
7	0111	3	0	1110	14	0.5

$(\text{col})_{10}$	$(\text{col})_2$	i	$i+1$	$(\text{row})_2$	$(\text{row})_{10}$	value
8	1000	2	3	0100	4	0.5
8	1000	3	0	0001	1	0.5
9	1001	0	1	1010	10	0.5
9	1001	2	3	0101	5	0.5
10	1010	0	1	1001	9	0.5
10	1010	1	2	1100	12	0.5
10	1010	2	3	0110	6	0.5
10	1010	3	0	0011	3	0.5
11	1011	1	2	1101	13	0.5
11	1011	2	3	0111	7	0.5
12	1100	1	2	1010	10	0.5
12	1100	3	0	0101	5	0.5
13	1101	0	1	1110	14	0.5
13	1101	1	2	1011	11	0.5
14	1110	0	1	1101	13	0.5
14	1110	3	0	0111	7	0.5

- sparse matrix, only keep: row, column, value

Sample code

- AF Heisenberg model on a 1D chain with open boundary condition

```
def GetHopList(Ns):
    HopList = []
    for i in range(Ns-1):
        HopList.append([i,i+1])
    return HopList
```

```
HI = []
HJ = []
HV = []

for i0 in range(Nl):
    for ih in range(len(HopList)):
        Pos0 = HopList[ih][0]
        Pos1 = HopList[ih][1]
```

$i \rightarrow$ Pos0

$i + 1 \rightarrow$ Pos1

col $\rightarrow$ i0

row $\rightarrow$ i1

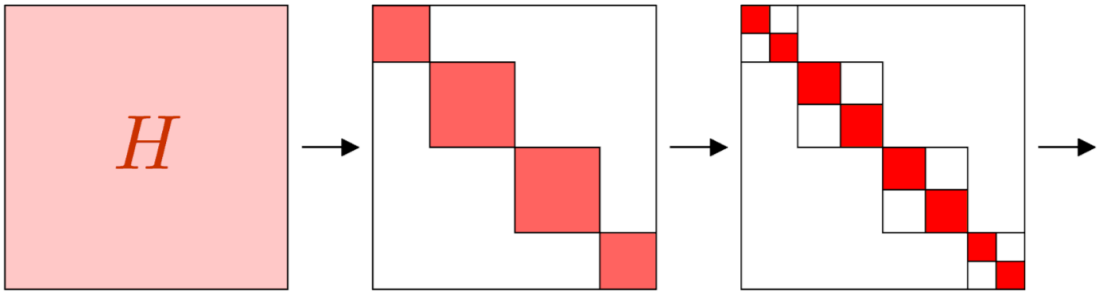
```
        if ReadBit(i0,Pos0) != ReadBit(i0,Pos1):
            i1 = FlipBit(i0,Pos0)
            i1 = FlipBit(i1,Pos1)
            HI.append(i1)
            HJ.append(i0)
            HV.append(0.5)
```

off-diagonal terms

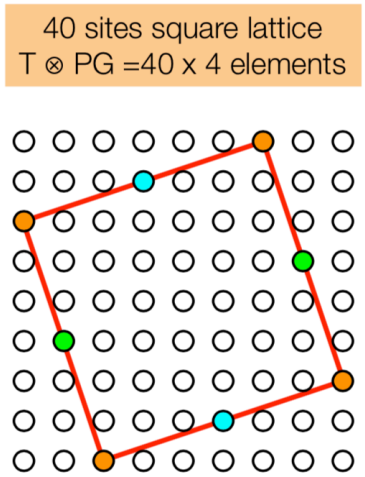
diagonal terms

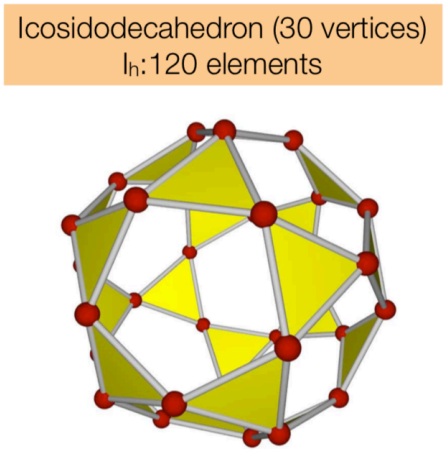
```
        HI.append(i0)
        HJ.append(i0)
        HV.append((ReadBit(i0,Pos0)-0.5)*(ReadBit(i0,Pos1)-0.5))

Hamr = sparse.coo_matrix((HV,(HI,HJ)),shape=(Nl,Nl)).tocsc()
```

Symmetries



- $U(1)$ related symmetries: conservation of particle numbers or total S_z
- translation symmetry, momentum space
- parity symmetry, reflection symmetry
- spin-inversion symmetry
- $SU(2)$ symmetry
- various spatial symmetry





Implement U(1) symmetry

# up spin = 0	$\|0000\rangle = \|0\rangle$	0
# up spin = 1	$\|0001\rangle = \|1\rangle$	0
	$\|0010\rangle = \|2\rangle$	1
	$\|0100\rangle = \|4\rangle$	2
	$\|1000\rangle = \|8\rangle$	3
# up spin = 2	$\|0011\rangle = \|3\rangle$	0
	$\|0101\rangle = \|5\rangle$	1
	$\|0110\rangle = \|6\rangle$	2
	$\|1001\rangle = \|9\rangle$	3
	$\|1010\rangle = \|10\rangle$	4
	$\|1100\rangle = \|12\rangle$	5
# up spin = 3	$\|0111\rangle = \|7\rangle$	0
	$\|1011\rangle = \|11\rangle$	1
	$\|1101\rangle = \|13\rangle$	2
	$\|1110\rangle = \|14\rangle$	3
# up spin = 4	$\|1111\rangle = \|15\rangle$	0

- Dimension of each subspace

$$C_4^0 = 1$$
$$C_4^1 = 4$$
$$C_4^2 = 6$$
$$C_4^3 = 4$$
$$C_4^4 = 1$$

- in general, there are $N+1$ subspaces
- total dimension is

$$\sum_{i=0}^{N} C_N^i = 2^N$$

Basis lookup procedures

- how to save time and memory?

- lookup can be done with two direct memory reads
  this is a time and memory efficient approach

PHYSICAL REVIEW B VOLUME 42, NUMBER 10 1 OCTOBER 1990

Exact diagonalization of quantum-spin models

H. Q. Lin
Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, New Mexico 87545
(Received 12 April 1990)

We have developed a technique to replace hashing in implementing the Lanczös method for exact diagonalization of quantum-spin models that enables us to carry out numerical studies on substantially larger lattices than previously studied. We describe the algorithm in detail and present results for the ground-state energy, the first-excited-state energy, and the spin-spin correlations on various finite lattices for spins $S=\frac{1}{2}$, 1, $\frac{3}{2}$, and 2. Results for an infinite system are obtained by extrapolation. We also discuss the generalization of our method to other models.

Lin tables

- six sites, three up-spins, three-down spins ⇒ three 0 and three 1

configuration A	
111	1
011	1
101	2
110	3
001	1
010	2
100	3
000	1

configuration B	
000	0
001	1
010	4
100	7
011	10
101	13
110	16
111	19

A \ B	000
111	1=1+0

A \ B	001	010	100
011	2=1+1	5=1+4	8=1+7
101	3=2+1	6=2+4	9=2+7
110	4=3+1	7=3+4	10=3+7

A \ B	011	101	110
001	11=1+10	14=1+13	17=1+16
010	12=2+10	15=2+13	18=2+16
100	13=3+10	16=3+13	19=3+16

A \ B	111
000	20=1+19

Lin tables

- six sites, three up-spins, three-down spins ⇒ three 0 and three 1

TABLE II. Spin configurations, their representations I_a and I_b, their base vectors $J_a(I_a)$ and position vectors $J_b(I_b)$ and their positions in the storage table $J(I)$ by using the new coding technique for unsymmetrized basis (see the text) for a spin-$\frac{1}{2}$ system of size six.

Configuration A	I_a	$J_a(I_a)$	Configuration B	I_b	$J_b(I_b)$	$J=J_a+J_b$
1 1 1	7	1	0 0 0	0	0	1
0 1 1	3	1	0 0 1	1	1	2
1 0 1	5	2	0 0 1	1	1	3
1 1 0	6	3	0 0 1	1	1	4
0 1 1	3	1	0 1 0	2	4	5
1 0 1	5	2	0 1 0	2	4	6
1 1 0	6	3	0 1 0	2	4	7
0 1 1	3	1	1 0 0	4	7	8
1 0 1	5	2	1 0 0	4	7	9
1 1 0	6	3	1 0 0	4	7	10
0 0 1	1	1	0 1 1	3	10	11
0 1 0	2	2	0 1 1	3	10	12
1 0 0	4	3	0 1 1	3	10	13
0 0 1	1	1	1 0 1	5	13	14
0 1 0	2	2	1 0 1	5	13	15
1 0 0	4	3	1 0 1	5	13	16
0 0 1	1	1	1 1 0	6	16	17
0 1 0	2	2	1 1 0	6	16	18
1 0 0	4	3	1 1 0	6	16	19
0 0 0	0	1	1 1 1	7	19	20

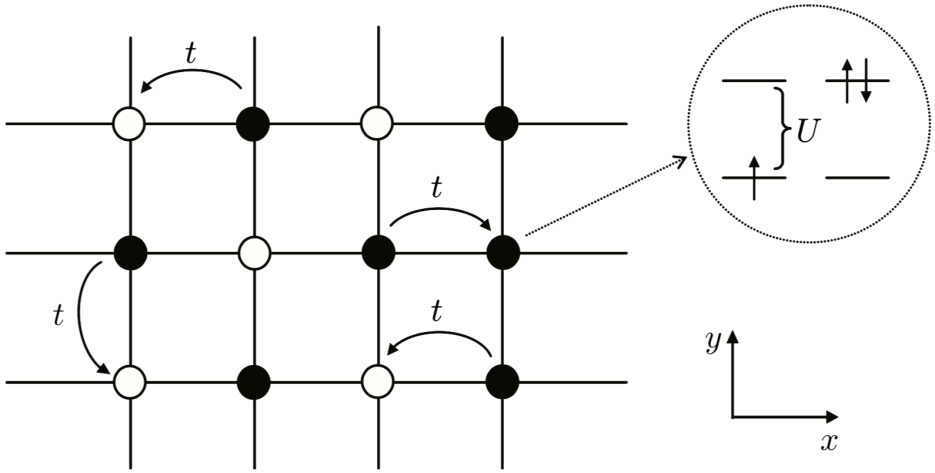
question: how about higher spins? bosons?

Fermionic systems

- Hubbard model

$$H = -t \sum_{\langle i,j\rangle,\sigma} C^{\dagger}_{i,\sigma} C_{j,\sigma} + U \sum_{i} n_{i\uparrow} n_{i\downarrow},$$

$$n_{i,\sigma} = C^{\dagger}_{i,\sigma} C_{i,\sigma}$$



- fermionic operators satisfy anti-commutation relations

$$\{C_{i,\sigma}, C^{\dagger}_{i',\sigma'}\} = \delta_{ii'}\delta_{\sigma\sigma'}, \{C^{\dagger}_{i,\sigma}, C^{\dagger}_{i',\sigma'}\} = \{C_{i,\sigma}, C_{i',\sigma'}\} = 0$$

- we fix the convention, all spin-up fermions are put on the left side of spin-down fermions, e.g. $N = 4$,

$$C^{\dagger}_{0,\uparrow} C^{\dagger}_{2,\uparrow} C^{\dagger}_{3,\uparrow} C^{\dagger}_{1,\downarrow} C^{\dagger}_{3,\downarrow} |0\rangle$$

spin ↓	spin ↑	decimal
1010	1101	$(10101101)_2 = 173$
$3^{\text{rd}} \longleftarrow \longrightarrow 0^{\text{th}}$	$3^{\text{rd}} \longleftarrow \longrightarrow 0^{\text{th}}$	$7^{\text{th}} \longleftarrow \longrightarrow 0^{\text{th}}$

Fermionic systems

- useful operations, N-site system

$$C^{\dagger}_{i,\uparrow}\left[\cdots 0_{i,\uparrow}\cdots|0\rangle\right] = (-1)^{\sum_{k=0}^{i-1} n_{k,\uparrow}}\left[\cdots C^{\dagger}_{i,\uparrow}\cdots|0\rangle\right]$$

$$C_{i,\uparrow}\left[\cdots C^{\dagger}_{i,\uparrow}\cdots|0\rangle\right] = (-1)^{\sum_{k=0}^{i-1} n_{k,\uparrow}}\left[\cdots 0_{i,\uparrow}\cdots|0\rangle\right]$$

$$C^{\dagger}_{i,\downarrow}\left[\cdots 0_{i,\downarrow}\cdots|0\rangle\right] = (-1)^{\sum_{k=0}^{i-1} n_{k,\downarrow}+\sum_{k=0}^{N-1} n_{k,\uparrow}}\left[\cdots C^{\dagger}_{i,\downarrow}\cdots|0\rangle\right]$$

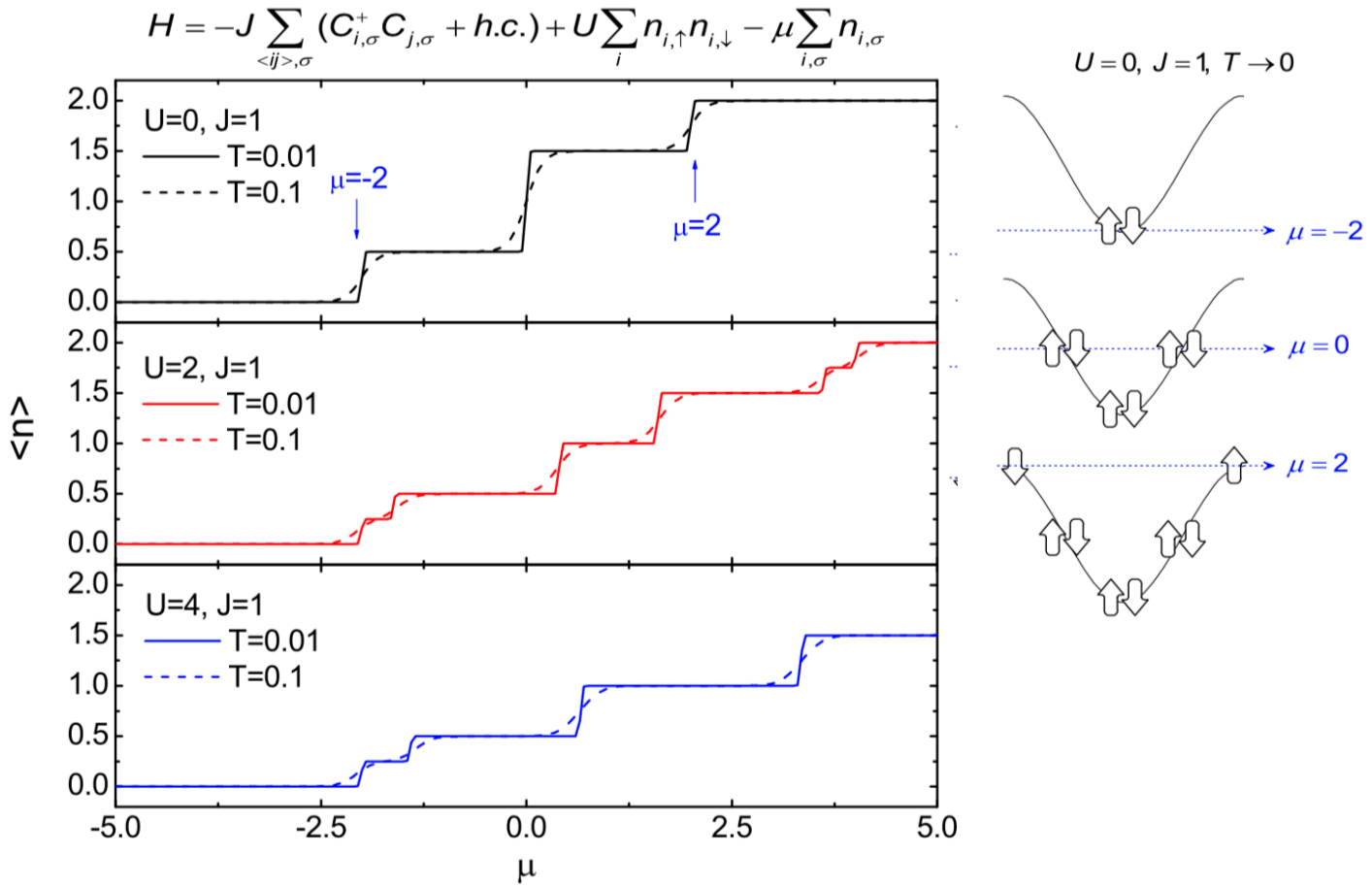
$$C_{i,\downarrow}\left[\cdots C^{\dagger}_{i,\downarrow}\cdots|0\rangle\right] = (-1)^{\sum_{k=0}^{i-1} n_{k,\downarrow}+\sum_{k=0}^{N-1} n_{k,\uparrow}}\left[\cdots 0_{i,\downarrow}\cdots|0\rangle\right]$$

- H may also introduce an additional sign

$$\left[C^{\dagger}_{3\downarrow}C_{0\downarrow}\right]\left[C^{\dagger}_{0\downarrow}C^{\dagger}_{1\downarrow}|0\rangle\right] = C^{\dagger}_{3\downarrow}C^{\dagger}_{1\downarrow}|0\rangle = -C^{\dagger}_{1\downarrow}C^{\dagger}_{3\downarrow}|0\rangle$$

Finite temperature

- calculate expectation value $\langle O \rangle = \frac{Tr\left(e^{-\beta H} O\right)}{Tr\left(e^{-\beta H}\right)} = \frac{\sum_n O_n e^{-\beta E_n}}{\sum_n e^{-\beta E_n}}$
- charge density $\langle n \rangle = \left\langle \frac{1}{N} \sum_{i=1}^{N} \left(n_{i\uparrow} + n_{i\downarrow}\right) \right\rangle = \frac{1}{N} \sum_{i=1}^{N} \left(\langle n_{i\uparrow} \rangle + \langle n_{i\downarrow} \rangle\right)$
- Hubbard model 4-site ring



Magnetization

- total magnetization of a and b sub-lattices

$$M_{a(b)} = \sum_{i \in a(b)} S_i^z = \sum_{i \in a(b)} \left(n_{i\uparrow} - n_{i\downarrow} \right) /2$$

- uniform magnetization and staggered magnetization

$$m_u = \frac{1}{N} \left(\langle M_a \rangle + \langle M_b \rangle \right) = \frac{1}{2N} \sum_{i=1}^{N} \left(\langle n_{i\uparrow} \rangle - \langle n_{i\downarrow} \rangle \right)$$

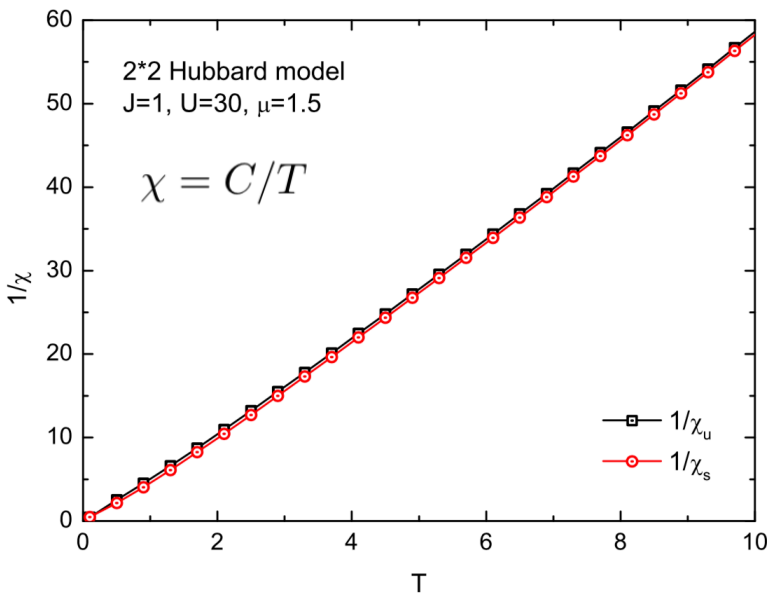
$$m_s = \frac{1}{N} \left(\langle M_a \rangle - \langle M_b \rangle \right) = \frac{1}{2N} \left[\sum_{i \in a} \left(\langle n_{i\uparrow} \rangle - \langle n_{i\downarrow} \rangle \right) - \sum_{i \in b} \left(\langle n_{i\uparrow} \rangle - \langle n_{i\downarrow} \rangle \right) \right]$$

- uniform and staggered susceptibility

$$\chi_u = \left. \frac{dm_u(h)}{dh} \right|_{h=0}, \chi_s = \left. \frac{dm_s(h)}{dh} \right|_{h=0}$$

$$\chi_u = \frac{\beta}{N} \left[\left\langle (M_a + M_b)^2 \right\rangle - \langle M_a + M_b \rangle^2 \right]$$

$$\chi_s = \frac{\beta}{N} \left[\left\langle (M_a - M_b)^2 \right\rangle - \langle M_a - M_b \rangle^2 \right]$$



Python packages

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.15 Manual | NumPy Reference | Routines | Linear algebra (`numpy.linalg`) | index | next | previous

numpy.linalg.eig

numpy.linalg.eig(*a*) [source]

Compute the eigenvalues and right eigenvectors of a square array.

Parameters: a : *(..., M, M) array*

Matrices for which the eigenvalues and right eigenvectors will be computed

Returns: w : *(..., M) array*

The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When *a* is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs

v : *(..., M, M) array*

The normalized (unit "length") eigenvectors, such that the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`.

Previous topic

numpy.linalg.svd

Next topic

numpy.linalg.eigh

Quick search

search

```
>>> w, v = LA.eig(np.array([[1, -1], [1, 1]]))
>>> w; v
array([ 1. + 1.j,  1. - 1.j])
array([[ 0.70710678+0.j        ,  0.70710678+0.j        ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]])
```

complete diagonalization

Python packages

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | SciPy v1.1.0 Reference Guide | Sparse linear algebra (`scipy.sparse.linalg`) | index | modules | next | previous

scipy.sparse.linalg.eigs

scipy.sparse.linalg.**eigs**(*A, k=6, M=None, sigma=None, which='LM', v0=None, ncv=None, maxiter=None, tol=0, return_eigenvectors=True, Minv=None, OPinv=None, OPpart=None*) [source]

Find k eigenvalues and eigenvectors of the square matrix A.

Solves `A * x[i] = w[i] * x[i]`, the standard eigenvalue problem for w[i] eigenvalues with corresponding eigenvectors x[i].

If M is specified, solves `A * x[i] = w[i] * M * x[i]`, the generalized eigenvalue problem for w[i] eigenvalues with corresponding eigenvectors x[i]

Parameters:

A : *ndarray, sparse matrix or LinearOperator*

An array, sparse matrix, or LinearOperator representing the operation `A * x`, where A is a real or complex square matrix.

k : *int, optional*

The number of eigenvalues and eigenvectors desired. *k* must be smaller than N-1. It is not possible to compute all eigenvectors of a matrix.

M : *ndarray, sparse matrix or LinearOperator, optional*

An array, sparse matrix, or LinearOperator representing the operation M*x for the generalized eigenvalue problem

A * x = w * M * x.

M must represent a real, symmetric matrix if A is real, and must represent a complex, hermitian matrix if A is complex. For best results, the data type of M should be the same as that of A. Additionally:

If *sigma* is None, M is positive definite

If sigma is specified, M is positive semi-definite

If sigma is None, eigs requires an operator to compute the solution of the linear equation `M * x = b`. This is done internally via a (sparse) LU decomposition for an explicit matrix M, or via an iterative solver for a general linear operator. Alternatively, the user can supply the matrix or operator Minv, which gives `x = Minv * b = M^-1 * b`.

Previous topic

scipy.sparse.linalg.lsmr

Next topic

scipy.sparse.linalg.eigsh

which : *str, ['LM' | 'SM' | 'LR' | 'SR' | 'LI' | 'SI'], optional*

Which *k* eigenvectors and eigenvalues to find:

'LM' : largest magnitude

'SM' : smallest magnitude

'LR' : largest real part

'SR' : smallest real part

'LI' : largest imaginary part

'SI' : smallest imaginary part

iterative diagonalization

Python packages

SciPy.org Sponsored By ENTHOUGHT

scipy.sparse.linalg.LinearOperator

class **scipy.sparse.linalg.LinearOperator**(*shape*, *matvec*, *rmatvec=None*, *matmat=None*, *dtype=None*) [source]

Common interface for performing matrix vector products

Many iterative methods (e.g. cg, gmres) do not need to know the individual entries of a matrix to solve a linear system A*x=b. Such solvers only require the computation of matrix vector products, A*v where v is a dense vector. This class serves as an abstract interface between iterative solvers and matrix-like objects.

Parameters:

shape *: tuple*
Matrix dimensions (M,N)

matvec *: callable f(v)*
Returns returns A * v.

Other Parameters:

rmatvec *: callable f(v)*
Returns A^H * v, where A^H is the conjugate transpose of A.

matmat *: callable f(V)*
Returns A * V, where V is a dense matrix with dimensions (N,K).

dtype *: dtype*
Data type of the matrix.

Previous topic
scipy.sparse.linalg.ArpackError
Next topic
scipy.sparse.linalg.LinearOperator.__call_

```
>>> from scipy.sparse.linalg import LinearOperator
>>> from scipy import *
>>> def mv(v):
...     return array([ 2*v[0], 3*v[1]])
...
>>> A = LinearOperator( (2,2), matvec=mv )
>>> A
<2x2 LinearOperator with unspecified dtype>
>>> A.matvec( ones(2) )
array([ 2.,  3.])
>>> A * ones(2)
array([ 2.,  3.])
```

Real time evolution

- the time evolution of a quantum system is governed by the time-dependent Schroedinger equation

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle$$

- for a given initial state $|\psi(0)\rangle$

  using exact diagonalization, we obtain E_n and $|\psi_n\rangle$

  the final state is

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle = \sum_n e^{-iE_nt/\hbar}|\psi_n\rangle\langle\psi_n|\psi(0)\rangle = \sum_n C_n e^{-iE_nt/\hbar}|\psi_n\rangle$$

  where $C_n = \langle\psi_n|\psi(0)\rangle$

- using the Lanczos vectors, only a very small Krylov space is needed

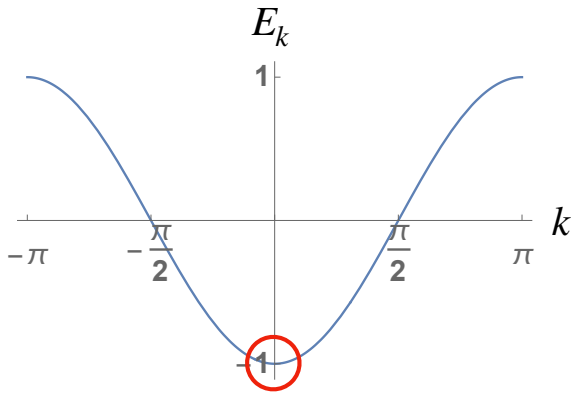
  $n \leq 20$ is sufficient

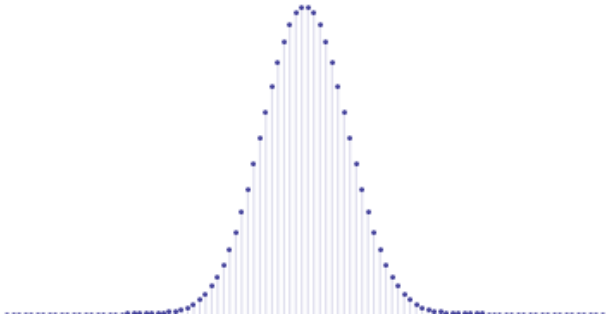
  time evolution can be computed efficiently

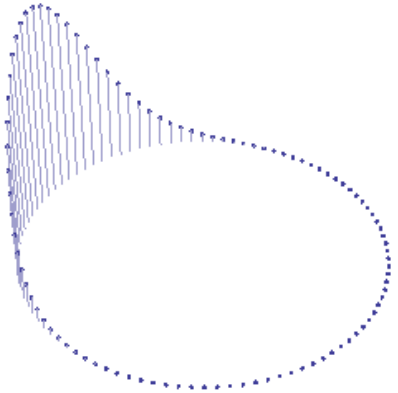
Real time evolution example

$$H = -\sum_j \left(a^\dagger_{j,\sigma} a_{j+1,\sigma} + \mathrm{H.c.} \right), \qquad |\psi(t=0)\rangle = \frac{1}{\sqrt{\Omega}} \sum_j e^{-(\alpha^2/2)(j-N_0)^2} e^{ik_0 j} a^\dagger_{j,\sigma} |0\rangle$$

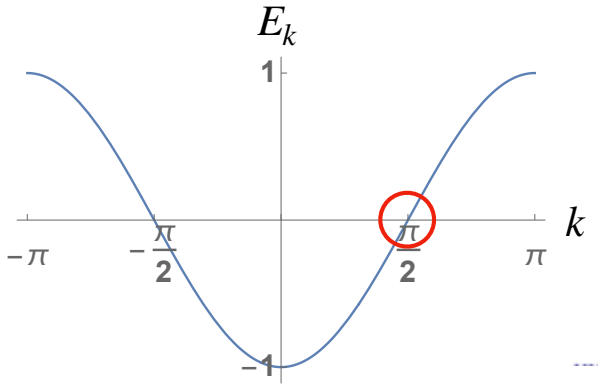
- Gaussian wave packet with momentum $k_0 = 0$

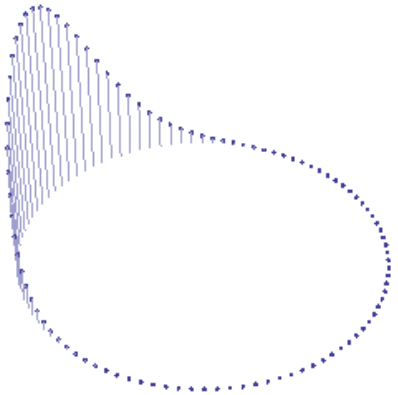






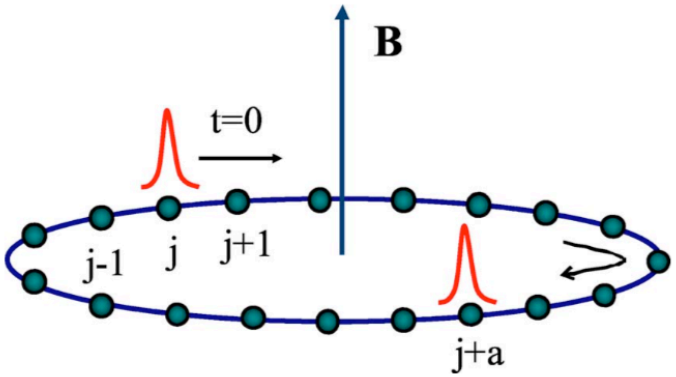
- Gaussian wave packet with momentum $k_0 = \pi/2$





Real time evolution example

- a ring lattice with N sites threaded by a magnetic field



$$H[\phi] = -J\sum_{j,\sigma}\left(e^{i2\pi\phi/N}a^{\dagger}_{j,\sigma}a_{j+1,\sigma} + \mathrm{H.c.}\right)$$

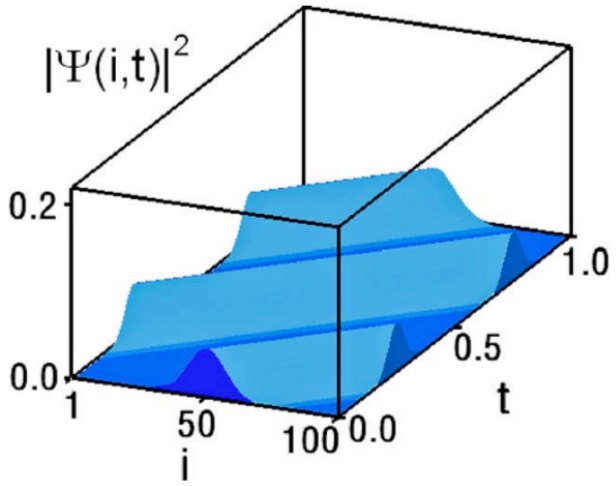


FIG. 2. (Color online) Numerical simulation of the time evolution of a zero-momentum Gaussian wave packet (GWP) with $\alpha = 0.1$ in position-space i of the 100-site ring with $\phi = N/4$. The time t is in the unit of $100/J$.

- time evolution of a Gaussian wave packet with momentum k_0

$$|\psi(t=0)\rangle = \frac{1}{\sqrt{\Omega}}\sum_{j} e^{-(\alpha^2/2)(j-N_0)^2} e^{ik_0 j} a^{\dagger}_{j,\sigma}|0\rangle$$

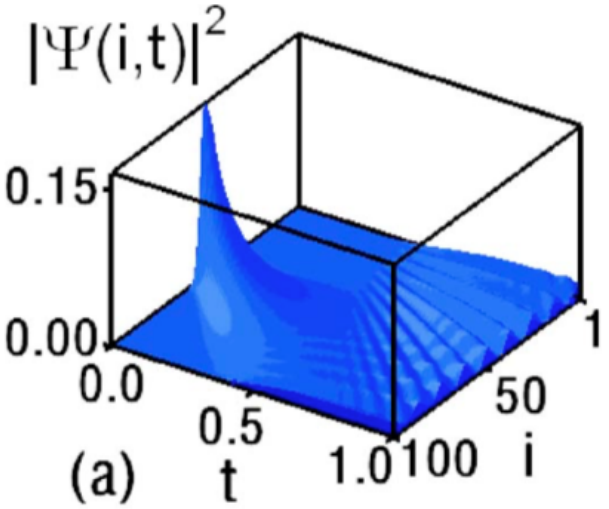
$$\Omega = \sum_{j}\exp[-\alpha^2(j-N_0)^2]$$

- non-spreading wave packet evolution
- solid-state flying qubit

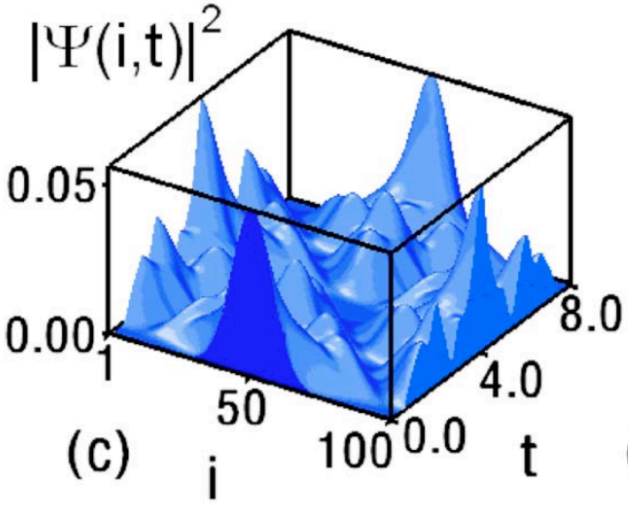
S. Yang, Z. Song, and C. P. Sun, PRA 73, 022317 (2006)

Real time evolution example

- self interference of GWP



- quantum revival of GWP



- open boundary condition

  GWP with momentum $k_0 \sim \pi/2$

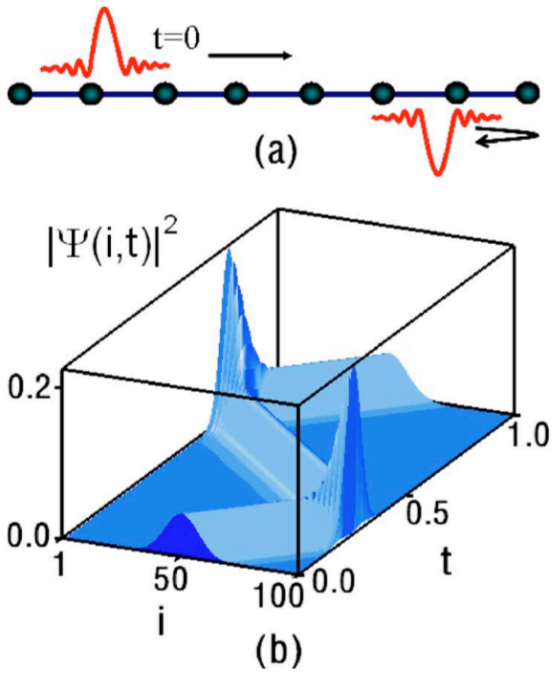


FIG. 5. (Color online) (a) The schematic illustration for the time evolution of a GWP in a chain. The π shift occurs at the boundary. (b) Numerical simulation of the time evolution of a moving GWP with α=0.1 and k_0=2$\pi\phi$/N=π/2 in the 100-site chain. The unit of time t is 100/J.

S. Yang, Z. Song, and C. P. Sun,
PRA 73, 022317 (2006)

Non-Hermitian Hamiltonian

- non-Hermitian Hamiltonian

$$H \neq H^{\dagger}$$

- bi-orthogonal basis

$$H \left| F_n \right\rangle = \omega_n \left| F_n \right\rangle$$

$$\left\langle G_n \right| H = \left\langle G_n \right| \omega_n \qquad \text{or} \qquad H^{\dagger} \left| G_n \right\rangle = \omega_n^{*} \left| G_n \right\rangle$$

- identity operator

$$\mathbb{I} = \sum_n \frac{\left| F_n \right\rangle \left\langle G_n \right|}{\left\langle G_n \middle| F_n \right\rangle}$$

- time-evolution operator

$$e^{-iHt} = \sum_n \frac{\left| F_n \right\rangle e^{-i\omega_n t} \left\langle G_n \right|}{\left\langle G_n \middle| F_n \right\rangle}$$

this in principle solves all the dynamics

Homework rules

- this homework has two tasks
  please submit through http://learn.tsinghua.edu.cn

- please submit source code and a detailed note to explain your code and results

- deadline is Sep. 29, 23:00

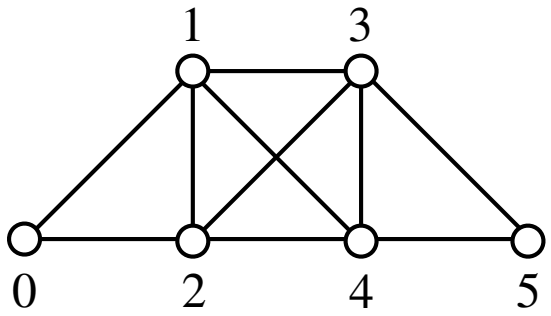
- if you submit after the deadline, you will have less points

- if you are an expert in exact diagonalization, this homework may be waived
  please contact me in person and submit something you have done before

Homework (task 1)

- write a code for diagonalizing the extended Hubbard model on a 6-site ladder

$$H = -t \sum_{\{i,j\},\sigma} (C^{\dagger}_{i,\sigma} C_{j,\sigma} + C^{\dagger}_{j,\sigma} C_{i,\sigma}) + U \sum_{i} n_{i\uparrow} n_{i\downarrow} - V \sum_{\{\{i,j\}\}} (n_{i,\uparrow} + n_{i,\downarrow})(n_{j,\uparrow} + n_{j,\downarrow}) - \mu \sum_{i,\sigma} n_{i,\sigma}$$



$\{i,j\} \in (0,2), (1,3), (2,4), (4,5), (1,2), (3,4)$

$\{\{i,j\}\} \in (0,1), (2,3), (1,4), (3,5)$

- take $t = 1.0$, $U = 8.0$, $V = 0.4$, $\mu = 4.0$

  please make full use of the $U(1)$ symmetry (there are totally 49 subspaces)

(1) show the lowest 6 eigenvalues in the ($N_{\uparrow} = 2$, $N_{\downarrow} = 4$) subspace

(2) show the lowest 20 eigenvalues of the whole system

(3) calculate $\langle n_{i,\sigma} \rangle$ for each site and spin using the true ground state

Sample output (task 1)

- please note the input parameters are different from homework

```
C:\Users\edwar\Desktop>python ExactDiagonalization.py --t 1.0 --U 4.0 --V 0.1 --mu 2.0
==========Task 1: Output===========
(1)_U(1)_sector_up2_down4: [-15.2600026  -14.95060662 -14.70152294 -14.51119205 -14.36481718
 -14.07944009]

(2)_lowest_20_eigenvalues: [-15.70641887 -15.2600026  -15.2600026  -15.2600026  -14.95060662
 -14.95060662 -14.95060662 -14.70152294 -14.70152294 -14.70152294
 -14.64334949 -14.64334949 -14.51119205 -14.51119205 -14.51119205
 -14.51119205 -14.51119205 -14.48657144 -14.47852901 -14.47852901]

(3)_Density_Up_Spin: [0.49920488 0.501906   0.49888912 0.501906   0.49888912 0.49920488]
(3)_Density_Down_Spin: [0.49920488 0.501906   0.49888912 0.501906   0.49888912 0.49920488]
```

Homework (task 2)

- consider the time evolution of a single-particle Gaussian wave packet on a tight-binding chain with a gradient field

$$H = -\sum_{j=1}^{N-1} \left(|j\rangle\langle j+1| + |j+1\rangle\langle j| \right) + F \sum_{j=1}^{N} j\,|j\rangle\langle j|$$

$$\left|\psi(t=0)\right\rangle = \frac{1}{\sqrt{\Omega}} \sum_{j} e^{-(\alpha^2/2)(j-N_0)^2} e^{ik_0 j} |j\rangle$$

- take $N = 101$, $F = 0.1$, $k_0 = \pi/2$, $\alpha = 0.15$, $N_0 = 51$, $t \in [0,100]$

(1) diagonalize the Hamiltonian and show the lowest 10 eigenvalues

(2) show $|\psi(j,t)|^2$ for $t = 42$ and $j = 10,\ 20,\ 30,\ 40,\ 50$

(3) plot $|\psi(j,t)|^2$ as functions of j and t

you will observe an oscillation behavior, which is Bloch oscillation

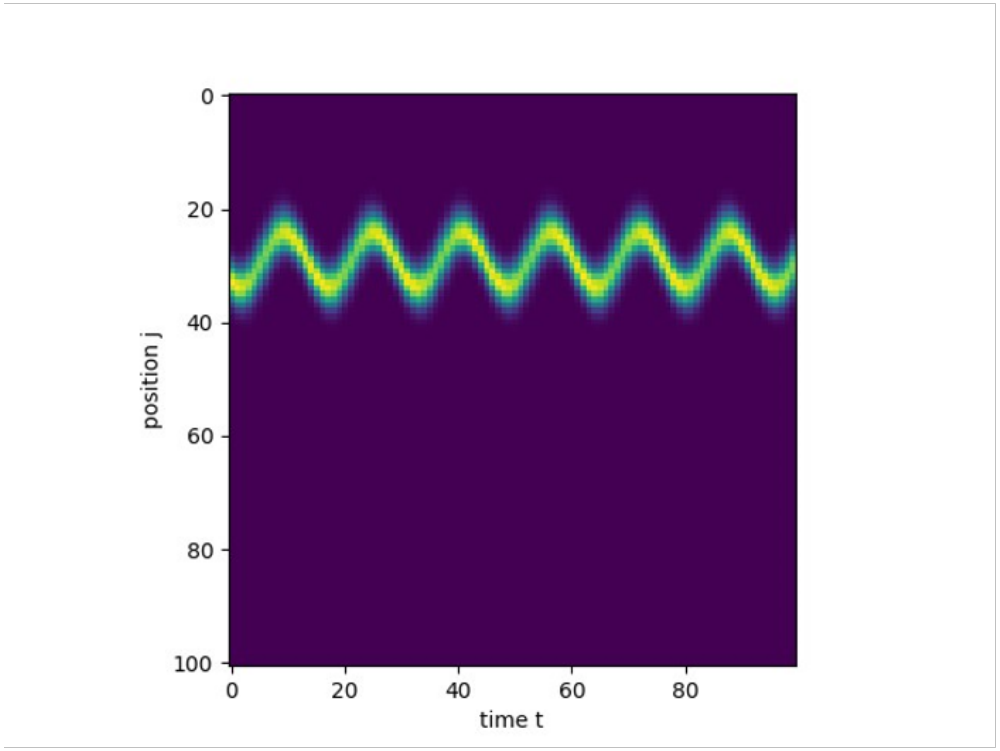
Bloch oscillation describes the oscillation of a particle confined in a periodic potential when a constant force is acting on it

you may also try other parameters for fun

Sample output (task 2)

- please note the input parameters are different from homework

```
C:\Users\edwar\Desktop>python TimeEvolution.py --N 101 --F 0.4 --k0 0.618 --alpha 0.314 --N0 3
3 --tmax 100 --t 42 --j_list 10 20 30 40 50
============Task 2: Output===========
(1)_10_eigenvalues: [-0.75734575  0.13035468  0.82547729  1.41163676  1.92122775  2.37508492
  2.79449719  3.1991657   3.59990942  3.99999256]

(2)_Probability: [1.2302506732777663e-08, 0.022391225439097 99, 0.009280806433433748, 2.1221075
976023325e-12, 1.1242641261441105e-23]

(3)_Plot: figure saved as student_id_homework_number_pic_number.png
```



Grading of this homework

- academic integrity [+50]

  completing assignments independently, creating and expressing your own ideas, DON'T copy answers from others or allow others to copy your answers

- the source code can be executed [+10] and can provide correct results [+10]

  the source code has high readability [+10]

- there is a detailed note file to explain the basic principle, source code and results [+10]

  the note file is well-written and easy to understand [+10]

- task 1: 70% score, task 2: 30% score

  task 1: did not use $U(1)$ symmetry [-20]

- time-dependent score: after DDL: [-1 per day]

- self-motivation score: new ideas or extra results [+10]