Exact diagonalization of the atomic problem

on the example of the *d* electronic shell

(and using the julia-language environment)

tutorial structure

$\approx$ 30min introduction to julia by hands-on training
(optional: can be skipped in case you are already experienced in it or prefer to stick to another language environment, e.g. MatLab,Python,C/C++,... that is "friendly" to arrays manipulations, e.g. matrix multiplication, eigenvalue problem, etc.)

$\approx$ 120min core part of the ED code: writing from scratch and debugging
– Fock-space vectors and their counting
– Hamiltonian matrix
– dealing with the interaction matrix
– fermionic sign factor

$\gtrsim$ 30min physical analysis of the output (including extensions of the code)
– Tanabe-Sugano diagram
– relevant one-particle operators, observables and representations
– commutation relations

(optional) completely binary-based implementation of the code and corresponding matrix representations of all creation and annihilation operators;
wider class of single-particle operators (those that have nonzero off-diagonal entries), their representations and eigenvalues

julia language: brief info and starting

julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Julia's Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing.

(firstly appeared: 2012)

- free, open source and library-friendly
- good performance, approaching that of statically-compiled languages like C, Fortran
- call Python functions: use the PyCall package
- call C functions directly: no wrappers or special application programming interfaces

Homepage and much-much more information & online help: `http://julialang.org/`

---

starting julia

open terminal, simply type `julia`
you will enter the interactive mode:

```
$ julia
        (logo with the currently-installed version)
julia>
```

exiting: "Ctrl+D" or type `exit()` or `quit()`; running the existing script-file: `julia <filename>`

julia language: hands-on training

Copy the folder to your local drive `cp -r ../cms00/ED_julia/ .`

`cd ED_julia/` ; `ls` ; `head -30 intro.jl` ; `julia`

- It is convenient to have two windows (split view): one with interactive julia-mode, another with the opened `intro.jl` (in any preferred text editor, e.g. nano, emacs, vim, gedit,...)
- Use the interactive mode to repeat all steps written

---

part A: basic operations and variable types

- julia has a large variety of variable types
- variable types are dynamic: can be changed accordingly during operation
- by default, the highest possible (64-bit) representation is used for Int, Float, Complex, ...
- additional specific values for Float: 'NaN'=not-a-number and 'Inf'=∞

part B: arrays and linear algebra

- unlike many other technical computing languages, Julia does not expect programs to be written in a vectorized style for performance. Julia's compiler uses type inference and generates optimized code for scalar array indexing, allowing programs to be written in a style that is convenient and readable, without sacrificing performance, and using less memory at times.

part C: control flow, running scripts and input/output

II. Exact diagonalization: implementation of the main part

atom in a crystal field: Hamiltonian and input parameters

atomic Hamiltonian (no spin-orbit coupling) in terms of creation and annihilation operators for electrons

$$H = \sum_{\sigma} \sum_{kl} h_{kl} c_k^{\dagger} c_l + \sum_{\sigma\sigma'} \sum_{ijkl} u_{ijkl} c_{i\sigma}^{\dagger} c_{j\sigma'}^{\dagger} c_{k\sigma'} c_{l\sigma}$$

where i, j, k, l are orbital indices and σ, σ' are spin indices.
To perform ED, we need to construct a vector space and fill the corresponding Hamiltonian matrix with values.

Input (integer) parameters of the model:

- `Nst` – total number of single-particle states ($N_{orbitals} \times N_{spins}$)
- `Qel` – total charge in the shell that remains conserved, $[\hat{N}, \hat{H}] = 0$

Other (floating-point) parameters:

- for simplicity, we consider that h_{kl} matrix is diagonal in orbital indices and all its diagonal entries can be described by a single parameter Δ, i.e. `Delta`, which can be set and varied directly in the ED script-file
- values for the interaction matrix will be provided in a separate file: `uijkl.dat`, i.e. after loading (on a later stage), we will have an array `u[i,j,k,l]`

vector space; using conservation of the total charge

open a new script-file for editing (e.g. `ed.jl`)

Let us specify from the start and consider the electronic d shell ($\ell = 2$)

- set `Nst=...` accordingly
- use some particular value for the total charge `Qel=...` (integer number between 2 and Nst-1; e.g. firstly consider the case of d^3 as in the lecture)

Now we want to separate the vector states, which belong to the chosen `Qel`-manifold, from all others.

- determine now the total number of different many-body states `Ncomb=...` with fixed `Nst` and `Qel`,
  e.g., the ket-states like $|0000000111\rangle$, $|0000001011\rangle$, ..., $|1110000000\rangle$ in d^3
  (Hint: the command `binomial` can be useful)
- initialize arrays that you are going to work with, for example:
  - `myvec = zeros(Ncomb,Nst)` – all the necessary vectors, the second (column) argument corresponds to the coordinate of a particular vector from the set (can take values 0 or 1)
  - `trvec = zeros(1,Nst)` – vector for check and other temporary purposes
  - `Hmatr = zeros(Ncomb,Ncomb)` – Hamiltonian matrix to diagonalize

setting the vector subspace

Now, it is necessary to track vectors (all possible combinations of '1' and '0' of the length `Nst`) and choose only those that correspond to our `Qel`-manifold

(a) tracking

- use the loop `for i=0:Nall-1`, where '`Nall`' is the total number of all possible states in the (d) shell
  – to determine the number '`Nall`', exponentiation `^` (`2^Nst`)
  or left bit shift `<<` operation (`1<<Nst`) can be useful
- inside the loop use a bit representation for each '`i`'-number
  – the `bin` command with two arguments can be useful: `bin(i,Nst)`
- make an output to the screen to check that you have all the required states there

(b) selecting and setting vectors

- use an auxiliary vector '`trvec`' to make a correspondence to all bits of the '`i`'-number [the command `parse("$(str[k])")`, where 'k' is the kth character of the string 'str', can be useful]
- calculate the charge of each 'trvec' (the command `sum`);
  in case it equals to '`Qel`', set `myvec[j,:]=trvec[:]; j=j+1;` (the vector index 'j' must be set to 1 before the for-loop starts)

✓checkpoint: check visually that you have set all '`Ncomb`' vectors properly

Hamiltonian: free-particle term and interaction matrix

Now, you can fill the `Ncomb`×`Ncomb` Hamiltonian matrix with values

(a) free-particle term

- let us consider *d* shell in cubic CF (common to many materials), therefore, five (initially degenerate) orbital states split into three T_{2g} and two E_g levels, so that all diagonal entries of h_{kl} can be written in terms of only one input parameter `'Delta'`, `0.5*Delta*[-1 -1 -1 1 1]` (for simplicity, firstly try with `Delta=2.0`)
- there is no SO coupling, thus all off-diagonal entries are zero
- the structure must be equal for the spin-up and spin-down entries (the tensor product operation `kron` can be useful)
- since there are only diagonal entries in the 1p-basis, this gives only contributions to diagonal elements of the Hamiltonian matrix `'Hmatr'`
- ✓checkpoint:
  after partially filling `'Hmatr'` with proper values (using matrix multiplication operations), you can check the intermediate result by analysis of energies of many-body states. Do energies for non-interacting limit match your expectations?

(b) interaction matrix (from the external file)

- load the file `'uijkl.dat'` (the command `ur=readdlm("uijkl.dat", '\t')`); columns are: (1) u value; (2-5) indices i,j,k,l ['1'-'5' ('6'-'10') correspond to ↑ (↓) sector there]
- rewrite the loaded values in the form of a 4-dimensional array `u[i,j,k,l]`

Hamiltonian: interaction term

To fill properly the Hamiltonian matrix, we need to:

1) find the vectors that are linked by the interaction matrix 'u[i,j,k,l]'
2) account for the fermionic sign factor

One of straightforward strategies:

- use the outer for-cycle over the loaded lines that determine nonzero 'u[i,j,k,l]'
- use the inner (double) for-cycle over all bra- and ket-vectors in the subspace
- use the conditional statement inside to exclude many of non-contributing states, e.g.

```
if(myvec[bra,i]==1 && myvec[bra,j]==1 && i!=j
&& myvec[ket,k]==1 && myvec[ket,l]==1 && k!=l)
```

- directly count electrons for the sign and change the vector state step by step:

```
chvec=myvec[ket,:]; numf=0
numf+=sum(chvec[1:l-1]); chvec[l]=chvec[l]-1
...
numf+=sum(chvec[1:i-1]); chvec[i]=chvec[i]+1
```

- the last (most nested) if-statement to check that the resulting vector is the bra-state `if(chvec==myvec[bra,:])`; if yes, then add the corresponding term (with a proper sign that is easily retrieved from 'numf') to the 'Hmatr' matrix element

Diagonalization and control

```
En=eigvals(Hmatr)
```

✓checkpoint:

In case of `Qel=3`, how large is degeneracy of the ground state?
(Later, we introduce an automatic calculation of states degeneracies, but it can be already done on this stage.)
Do you have some reasonable explanation for that?
What will happen with degeneracies of the ground and the first excited state, if you increase/decrease the crystal field splitting magnitude in 10 times?

Answer the same questions for `Qel=6`. Do you observe a difference comparing to the d^3 case? Why?

Unless your program provides an unphysical output in limiting cases, to have a better insight in physics, we can perform more analysis: introduce operators and corresponding observables, construct Tanabe-Sugano diagrams for each case, etc.

III. ED: physical analysis of the output

Tanabe-Sugano diagram

- introduce an external for-cycle over the magnitude over the crystal field splitting and obtain the Tanabe-Sugano diagram for d^3 electronic shell (the interactions strengths determined by the file '`uijkl.dat`' are kept fixed).
- note that the interaction term is usually time-consuming, thus consider setting it only once outside the for-loop over the CF magnitude
- the reasonable range for CF: $\Delta \in [0, 4]$
- shift all energies to have the lowest state always at zero
- save your eigenenergies as functions of CF splitting to the external file, `writedlm` command (you should have an array with the 1st column = CF magnitude, 2nd till Ncomb+1 = energies)
- use some external software to construct a plot, e.g. gnuplot,

```
plot for [i=2:Ncomb+1] "ts.dat" using 1:i with lines title ''
```

('PyPlot' or 'Gadfly' julia packages for plotting are not installed on the server)

✓checkpoint:

Do you observe some quantitative differences with the plot shown in the lecture? (in case you have enough time, construct the TS diagram for the d^6 shell)

Energy multiplets, operators and observables

Now, fix the CF splitting (pick up some value from the TS diagram) and analyze low-energy states in more detail

degeneracies

- introduce a small routine calculating degeneracies of **five** lowest multiplets (the command `diff` can be useful), make an output to the screen;
- the degeneracy $D^{(j)}$ can already provide some information on states of the multiplet j, $D^{(j)} = (2L^{(j)} + 1)(2S^{(j)} + 1)$

number operators

- introduce a number operator $\hat{n}$ in the single-particle basis (`ones`, `eye` and `kron` (for other operators below) can be useful)
- express it in the new (many-body) basis similarly to the CF-term
- perform a unitary transformation of $\hat{N}$ to the eigenbasis of the Hamiltonian
- find a matrix representation of $\hat{N}$ in space of all energy eigenstates of **three** lowest multiplets

✓checkpoint: Do you observe any difference in matrix entries for the first and the second multiplet? Why?

- introduce the number operators for T_{2g} and E_g states separately and perform a similar analysis. Do you observe some difference now?

Energy multiplets, operators and observables

spin operator

- introduce the operator S_z in the single-particle basis
- express it in the new (many-body) basis
- perform a unitary transformation of $\hat{S}_z$ to the eigenbasis of the Hamiltonian
- d^3 shell: determine eigenvalues for subpaces corresponding to **two** lowest multiplets
- d^6 shell: check that S_z eigenvalues (together with number operators representations for T_{2g} and E_g) clearly indicate a transition from the high spin to the low spin state at some critical value of the CF splitting (perform a corresponding analysis just for two values $\Delta_{1,2}$ of CF, $\Delta_1 < \Delta_{crit} < \Delta_2$)

commutation relations: verify that (as it was mentioned in the lecture), indeed, the following relations hold for the Hamiltonian H:

- charge conservation, $[N, H] = 0$
- S_z conservation, $[S_z, H] = 0$
- S^2 conservation, $[S^2, H] = 0$

?? What about N_{t2g} and N_{eg} operators? Do they commute with the Hamiltonian? Is there any correspondence with the observed form of matrix representations for these operators?