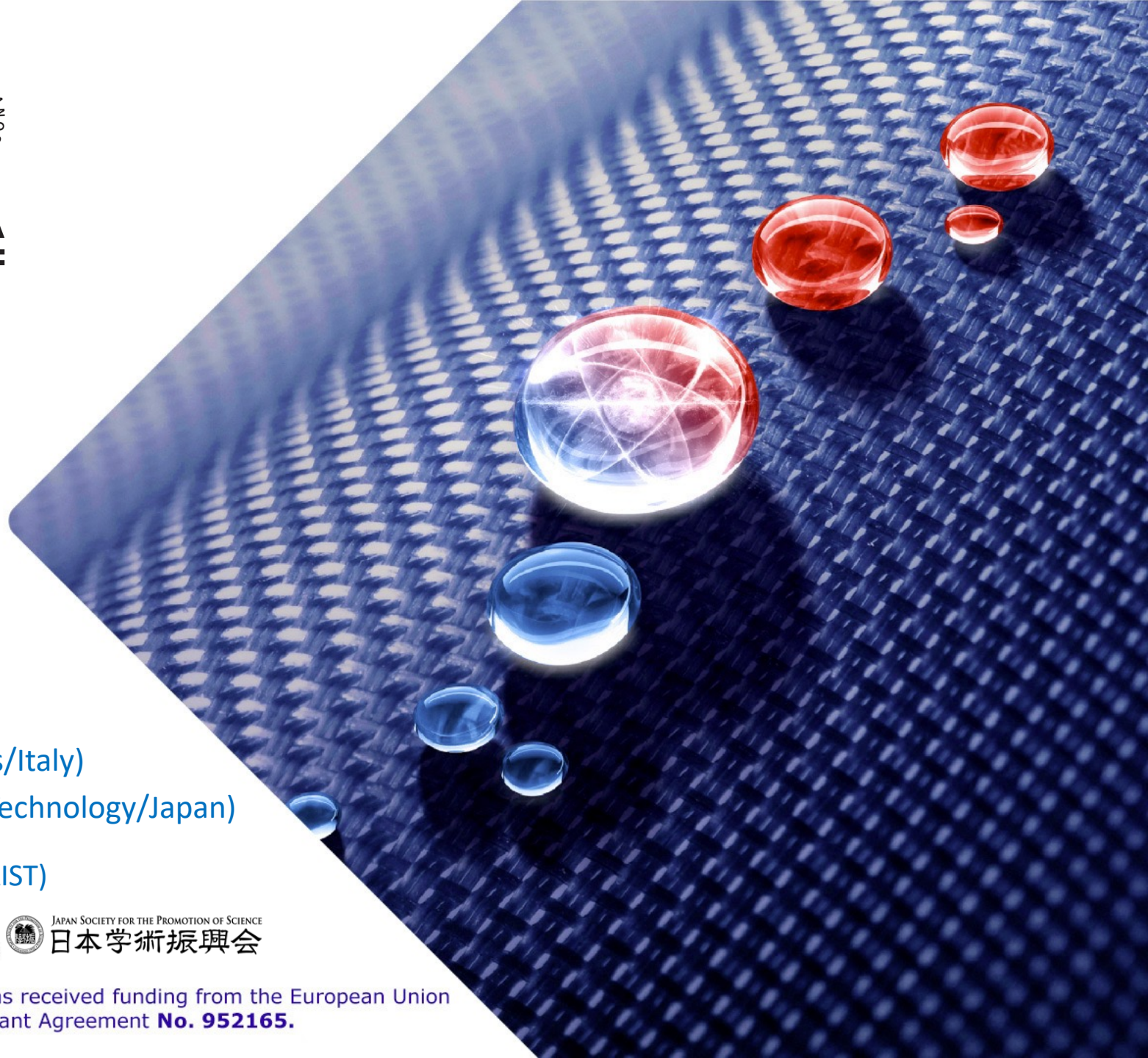
Targeting Real chemical accuracy at the EXascale

SISSA



TurboRVB and Turbo-Genius: Overview and Workflow

Kosuke Nakano

- SISSA (International School for Advanced Studies/Italy)
- JAIST (Japan Advanced Institute of Science and Technology/Japan)

(Prof. Sorella group/SISSA) (Prof. Maezono group/JAIST)

JAIST Japan Advanced Institute of Science and Technology 1990

Japan Society for the Promotion of Science 日本学術振興会

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement **No. 952165.**

Targeting Real chemical accuracy at the EXascale

SISSA

Day 3 and 4:
- Overview
- Hands-on session

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement **No. 952165.**

Targeting Real chemical accuracy at the EXascale

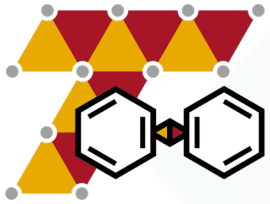
SISSA

Day 3 and 4:

- Overview
- Hands-on session

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement **No. 952165.**

Packages: TurboRVB and Turbo-Genius



TurboRVB

Quantum Monte Carlo Package SISSA

QMC engines (DFT, VMC-optimization, VMC, LRDMC)

K. Nakano, C. Attaccalite, M. Barborini, L. Capriotti, M. Casula, E. Coccia, M. Dagrada, Y. Luo, G. Mazzola, A. Zen, and S. Sorella, ***J. Chem. Phys.*** 152, 204121 (2020)

TurboGenius

User friendly python wrappers.

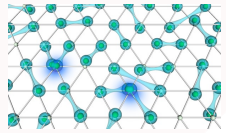
K. Nakano et al., ***in preparation*** (2022)

Wavefunction ansatz

$\Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_N)$ should be anti-symmetric under exchange of electron positions.

Slater determinant: the most straightforward ansatz $\Psi_{VMC} = Det|\phi_1, \phi_2, \cdots, \phi_N|$

- Linear combination of the slater determinants. $\Psi_{VMC} = Det|\phi_1, \phi_2, \cdots, \phi_N| + \cdots$



c.f. P.W. Anderson

- Geminal functions (i.e., considering pairs of electrons.)
- Backflow functions (i.e., increase in the effective mass of electrons.)
- Fermi-net (i.e., anti-symmetric neural network.)

More complex.

The more complex an ansatz is, the better energy we could get. However, the computational cost also increases.

One should increase the number of variational parameters, considering “physics”.

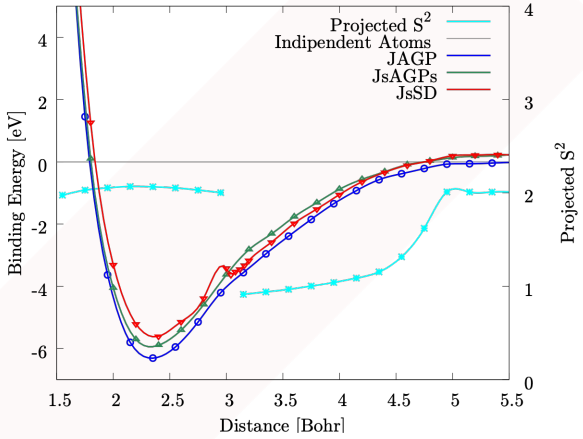
C_2 molecule

Double-bond?? Quadruple-bond??, spin-singlet.

S. Shaik, et al. Nat. Chem. 4 195–200 (2012)

DMC results

Wavefunction	C atom (Ha)	C_2 molecule (Ha)	Binding (eV)
Jastrow Slater	−37.82966(4)	−75.8672(1)	5.656(3)
Jastrow Geminal (Singlet)	−37.8364(1)	−75.8938(2)	6.01(1)
Jastrow Geminal (Singlet + broken sym.)	−37.8364(1)	−75.8935(2)	6.00(1)
Jastrow Geminal (All-pairing, Pfaffian)	−37.8363(1)	−75.9045(2)	6.31(1)
Estimated exact	-37.8450	−75.9045(2)	6.44(2) (Exp.)

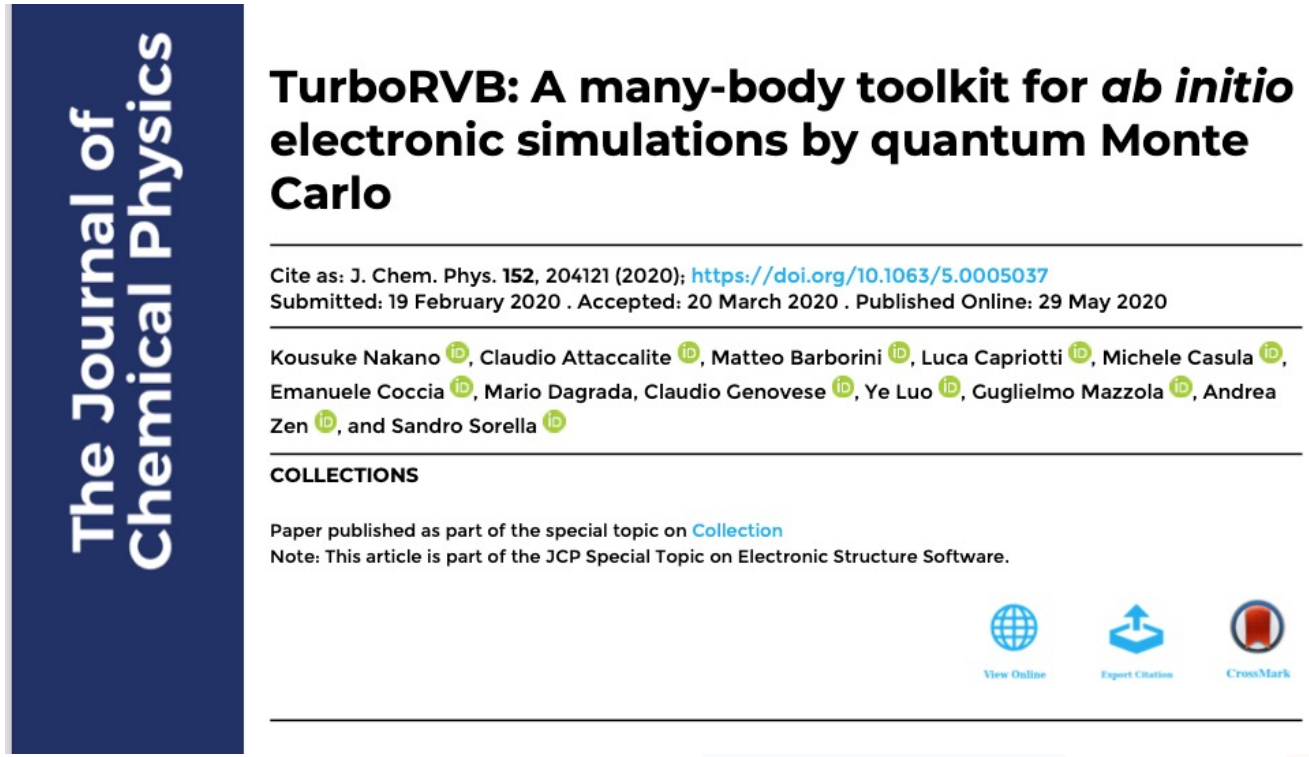


More complex.

C.G, T.S, **K.Nakano**, and S.S. J. Chem. Theory Comput. 16, 6114 (2020)

CCSD(T) with the V5Z basis = 6.24 eV

DMC gives a more accurate result than CCSD(T) for the challenging molecule!



The Journal of Chemical Physics

TurboRVB: A many-body toolkit for *ab initio* electronic simulations by quantum Monte Carlo

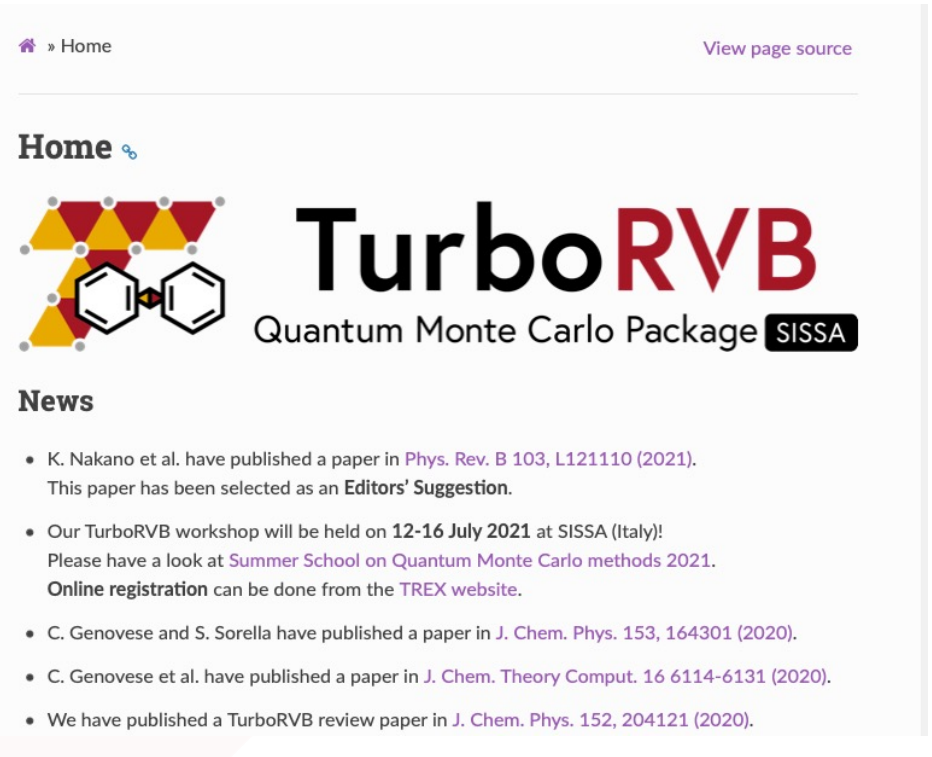
Cite as: J. Chem. Phys. **152**, 204121 (2020); https://doi.org/10.1063/5.0005037
Submitted: 19 February 2020 . Accepted: 20 March 2020 . Published Online: 29 May 2020

Kousuke Nakano, Claudio Attaccalite, Matteo Barborini, Luca Capriotti, Michele Casula, Emanuele Coccia, Mario Dagrada, Claudio Genovese, Ye Luo, Guglielmo Mazzola, Andrea Zen, and Sandro Sorella

COLLECTIONS

Paper published as part of the special topic on Collection
Note: This article is part of the JCP Special Topic on Electronic Structure Software.

View Online Export Citation CrossMark



» Home View page source

Home

TurboRVB
Quantum Monte Carlo Package SISSA

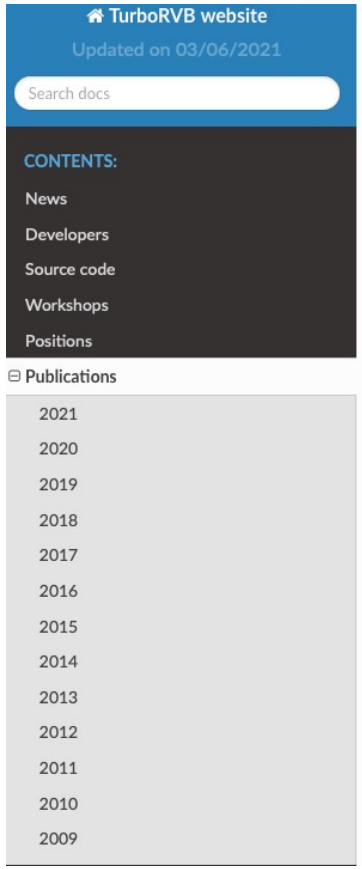
News

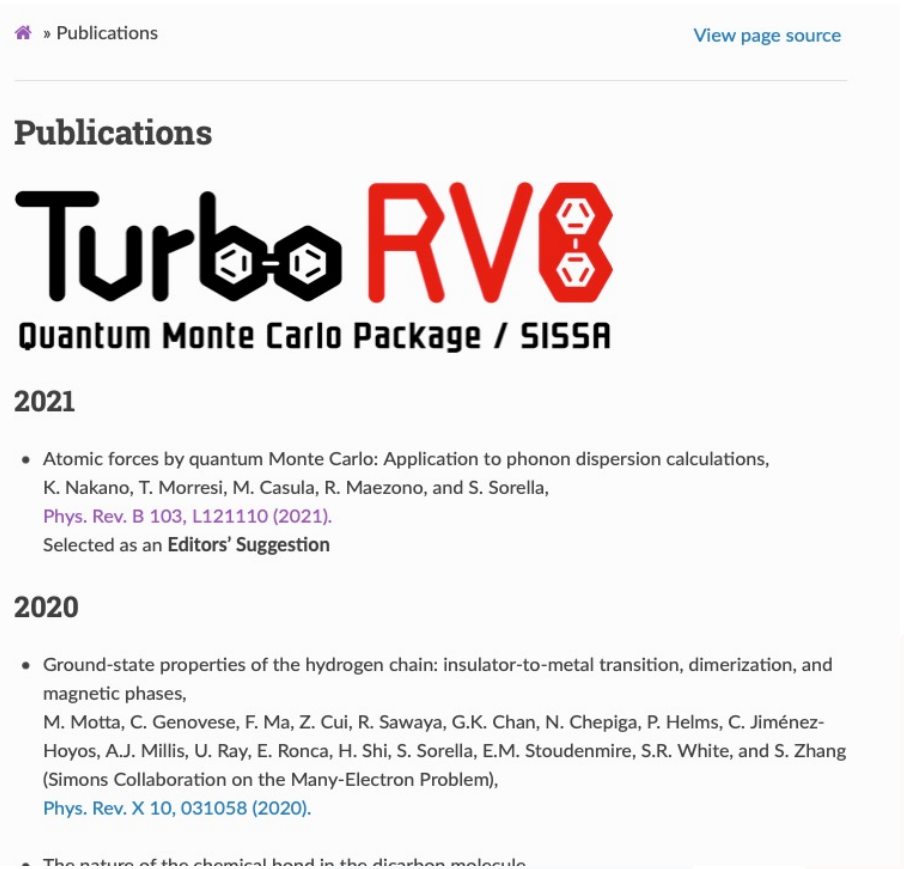
- K. Nakano et al. have published a paper in Phys. Rev. B 103, L121110 (2021).
  This paper has been selected as an **Editors' Suggestion**.
- Our TurboRVB workshop will be held on **12-16 July 2021** at SISSA (Italy)!
  Please have a look at Summer School on Quantum Monte Carlo methods 2021.
  Online registration can be done from the TREX website.
- C. Genovese and S. Sorella have published a paper in J. Chem. Phys. 153, 164301 (2020).
- C. Genovese et al. have published a paper in J. Chem. Theory Comput. 16 6114-6131 (2020).
- We have published a TurboRVB review paper in J. Chem. Phys. 152, 204121 (2020).

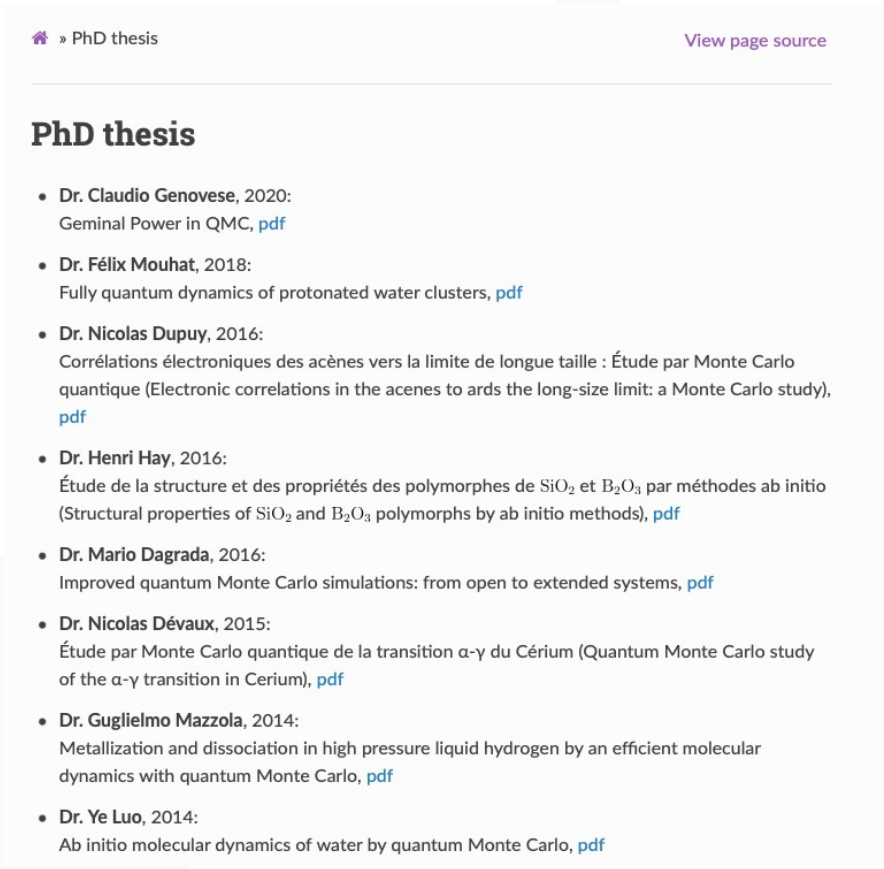
K. Nakano, C. Attaccalite, M. Barborini, L. Capriotti, M. Casula, E. Coccia, M. Dagrada, Y. Luo, G. Mazzola, A. Zen, and S. Sorella, ***J. Chem. Phys.*** 152, 204121 (2020)

What we can do with TurboRVB?

Please visit our website :-) All the papers and Ph.D thesis using TurboRVB are listed here.

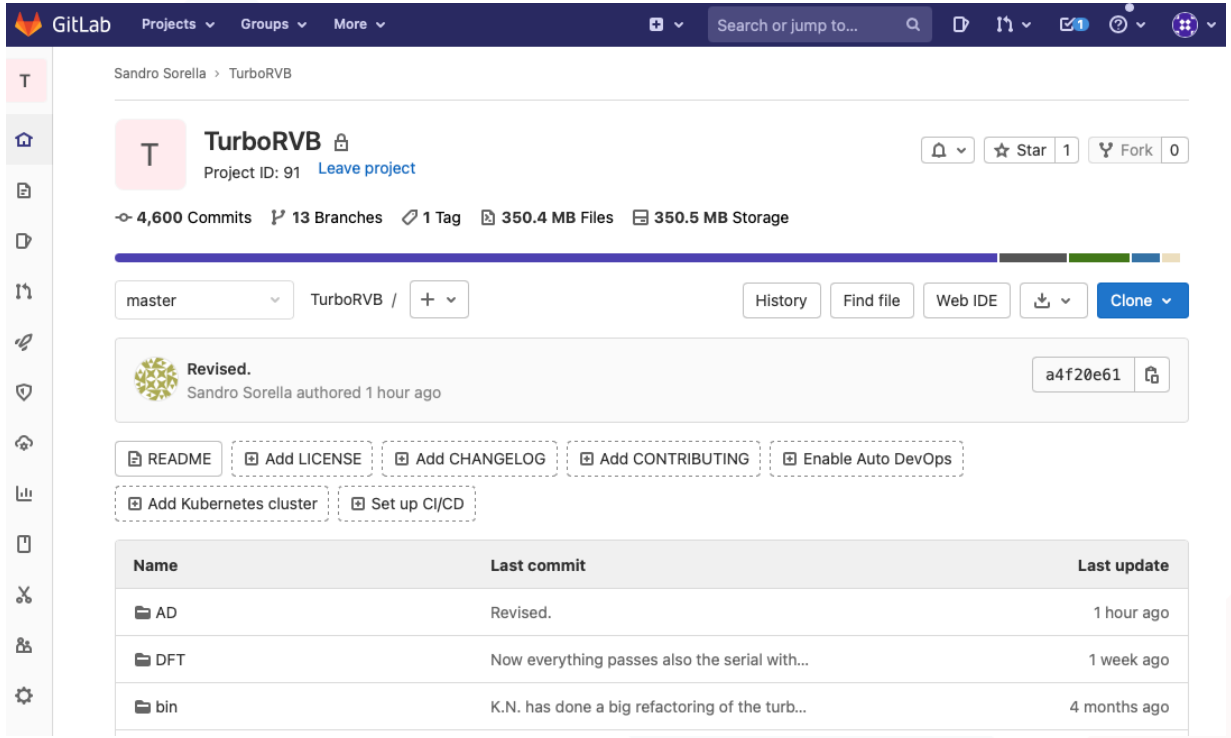






SISSA Gitlab server

Where can we download TurboRVB and Turbo-Genius?



From SISSA-gitlab server.

https://git-scm.sissa.it

A request: to kousuke_1123@icloud.com

For the time being, turborvb and turbo-genius are inhouse codes, so please DO NOT distribute to the public.

Within a year, all the codes will be public under an appropriate license (maybe BSD) :-)

Targeting Real chemical accuracy at the EXascale

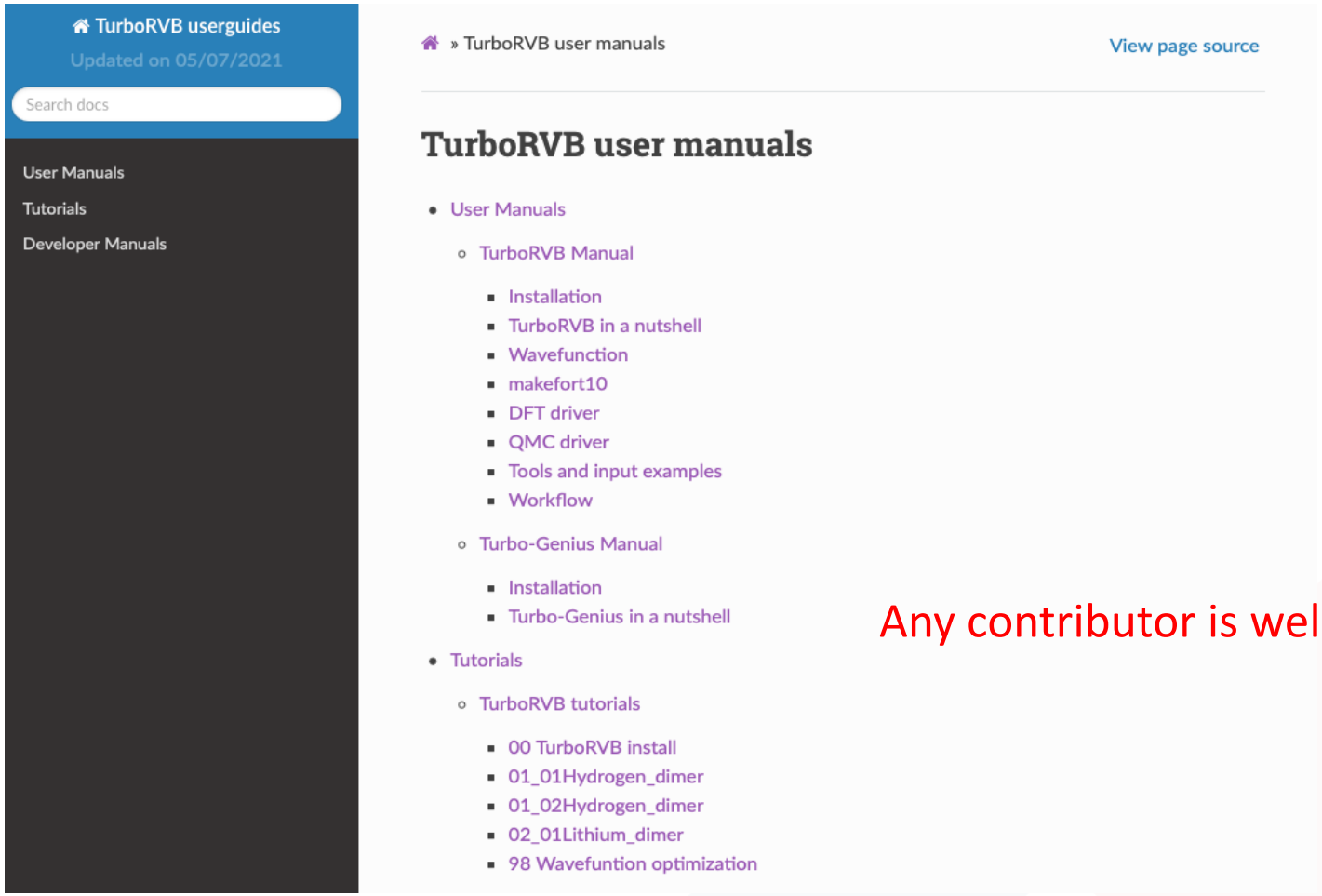
SISSA

Day 3 and 4:

- Overview
- Hands-on session

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement **No. 952165.**

TurboRVB/Turbo-Genius manual and tutorials



From SISSA-gitlab server.

https://git-scm.sissa.it

Any contributor is welcome!!!

They are composed by sphinx. All the tutorial in this school is included here.

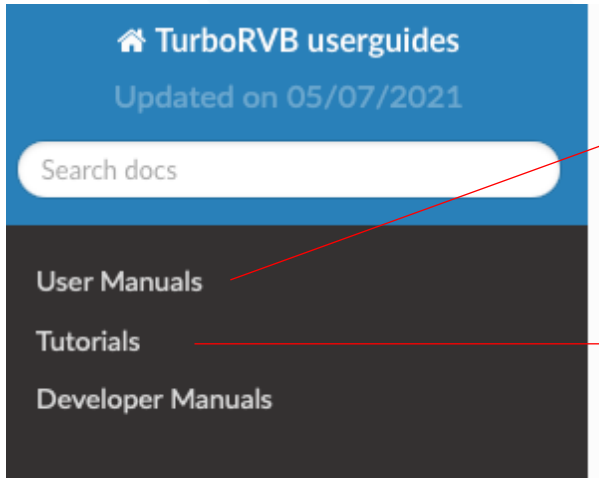
TurboRVB/Turbo-Genius manual and tutorials

If you want to see the userguide, please let us know. We will give you the permission.

git clone git@git-scm.sissa.it:sorella/turborvb_userguides.git

Any contributor is welcome!!!

Open /turborvb_userguides -> build -> html -> index.html



- User Manuals.
- TurboRVB tutorials.
- Turbo-Genius tutorials (for the hands-on session).

on your local comp. (for the TREX summer school).

Compilation (TurboRVB)

We strongly recommend Intel, IBM, and Fujitsu Fortran compilers. (Not gfortran).

1. Legacy make:

<u>You do not have to do this for the hands-on session!!!</u>

Copy a config file: config/make_XXX.inc make.inc

Copy a make.txt file: src/make.txt_standard src/make.txt

Compile TurboRVB: ./makeall (serial) or ./makeall-mpi

If you want to clean it: make cleanall

Fugaku, Hokusai (RIKEN)

Marconi, Marconi100 (CINECA)

SISSA-cluster (SISSA)

Kagayaki (JAIST)

2. Modern CMake:

```
mkdir build
cd build
cmake  -DXXXX = YYYY etc...
make install    # copy generated binaries to ./bin directory.
```

A typical workflow in QMC

(Input) Atomic positions, basis sets, pseudo potentials…

1) Pre-process:

WFT (Gaussian16, PySCF)
or **DFT** (Quanaum Espresso)

The obtained one-particle WFs

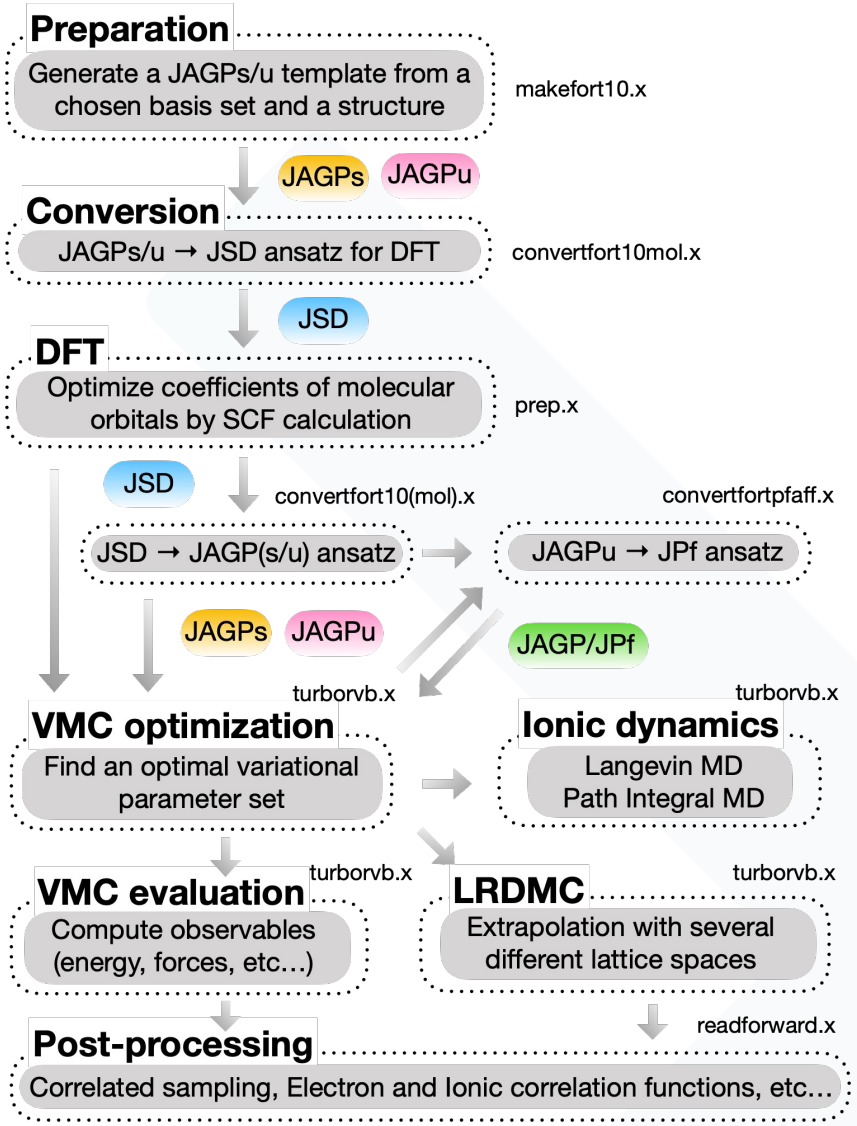
2) Post-process:

QMC (TurboRVB, etc…)

(Output)

Many-body WFs, its energy, forces, etc…

A typical workflow in TurboRVB (TurboGenius)



= Workflow =

1. Prepare a structure and basis set **makefort10**

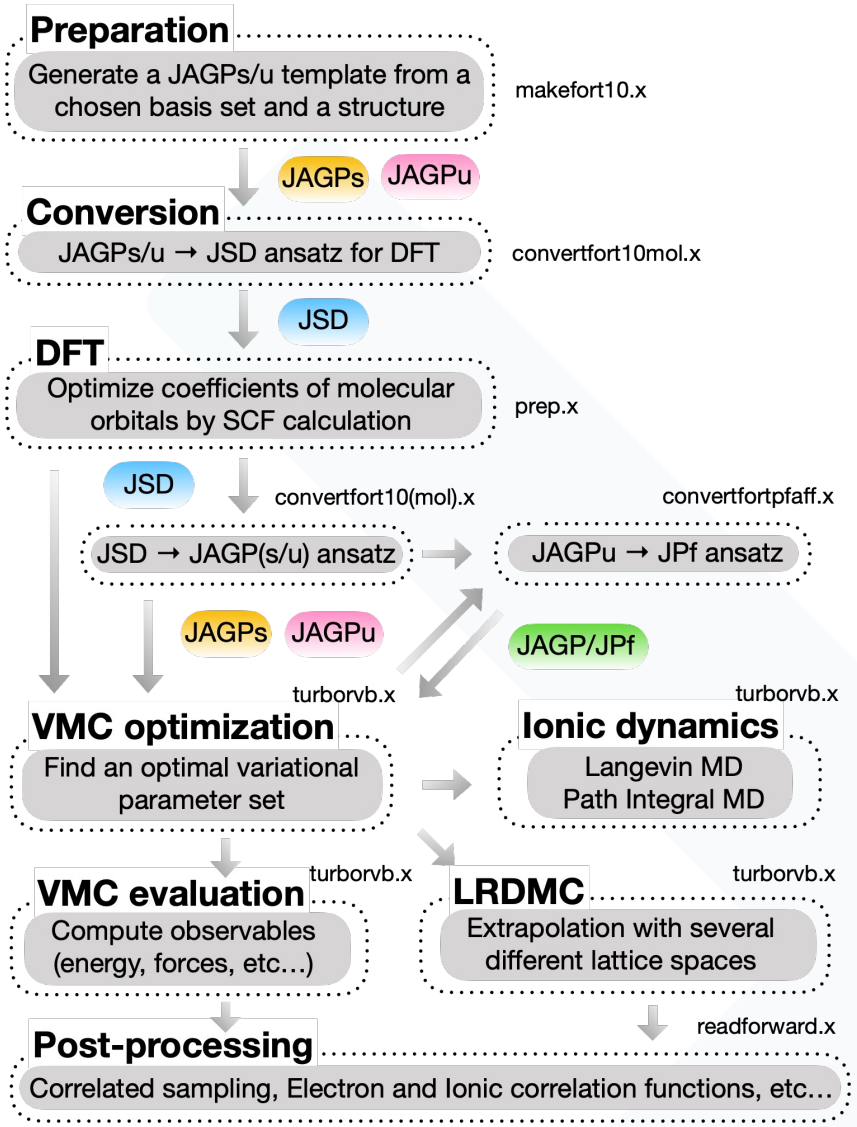
2. DFT Construct a reasonable initial WF! **prep**

3. VMC-opt Optimize the wavefunction **turborvb**

4. VMC Do a VMC run. **turborvb**

5. LRDMC LRDMC with the optimized WF. **turborvb**

A typical workflow in TurboRVB (TurboGenius)



= Workflow =

1. Prepare a structure and basis set **makefort10**

2. DFT Construct a reasonable initial WF! **prep**

3. VMC-opt Optimize the wavefunction **turborvb**

4. VMC Do a VMC run. **turborvb**

5. LRDMC LRDMC with the optimized WF. **turborvb**

Wavefunction (makefort.10.x)

Input: makefort10.input → Binary: makefort10.x → Output:fort10_new

makefort10.x is a tool for generating JAGP WF(fort.10) from makefort10.input.

```
# Ion coordinates
 N1 Z1                x1      y1      z1
 N2 Z2                x2      y2      z2
   ..                ..      ..      ..
 Nn Zn                xn      yn      zn
```

Structural information.

```
#       Parameters atomic wf
        1           4           300
        1    2.0    1.0   3.231   7.54
```

Basis-set information.

```
posunits='crystal'
natoms=2
ntyp=1
complexfort10=.false.
pbcfort10=.true.
!yes_pfaff=.true.
celldm(1)=4.648726266579395
celldm(2)=1.0
celldm(3)=4.065040650406504
celldm(4)=1.5707963267948966
celldm(5)=1.5707963267948966
celldm(6)=2.0943951023931953
yes_tilted=.true.
nxyz(1)=3
nxyz(2)=3
nxyz(3)=1
phase(1)=0.0
phase(2)=0.0
phase(3)=0.0
phasedo(1)=0.0
phasedo(2)=0.0
phasedo(3)=0.0
```

makefort10.input file

makefort10.x

```
# fort.10 of the C2-dimer (the Pfaffian ansatz with the Filippi pseudo potential.)
# Nelup  #Nel  # Ion
        4          -8           2
# Shell Det.   # Shell Jas.
        50          43
# Jas 2body  # Det   #  3 body atomic par.
       -22        1482          42
# Det mat. =/0  # Jas mat. =/0
        120        8370
# Eq. Det atomic par.  # Eq. 3 body atomic. par.
        741          21
# unconstrained iesfree,iessw,ieskinr,I/O flag
        8370         120           6           0
# Ion coordinates
4.00000000000000        6.00000000000000        0.000000000000000E+000
0.000000000000000E+000  -1.14999954166875
4.00000000000000        6.00000000000000        0.000000000000000E+000
0.000000000000000E+000   1.14999954166875
#  Constraints for forces: ion - coordinate
        1           1           1
        1           1           2
        1           1           3
        1           2           1
        1           2           2
        1           2           3
#          Parameters Jastrow two body
       -1  0.342214663461764
...
```

Wavefunction file (fort.10)

Wavefunction file (fort.10)

All the information (except for pseudo potential info.) is included in a single file, named “fort.10”

```
# fort.10 of the C2-dimer (the Pfaffian ansatz with the Filippi pseudo potential.)
# Nelup  #Nel  # Ion
        4          -8          2
# Shell Det.   # Shell Jas.
        50          43
# Jas 2body  # Det   #  3 body atomic par.
        -22        1482          42
# Det mat. =/0  # Jas mat. =/0
        120        8370
# Eq. Det atomic par.  # Eq. 3 body atomic. par.
        741          21
# unconstrained iesfree,iessw,ieskinr,I/O flag
        8370         120           6           0
# Ion coordinates
4.00000000000000        6.00000000000000       0.00000000000000E+000
0.00000000000000E+000  -1.14999954166875
4.00000000000000        6.00000000000000       0.00000000000000E+000
0.00000000000000E+000   1.14999954166875
#  Constraints for forces: ion - coordinate
        1           1           1
        1           1           2
        1           1           3
        1           2           1
        1           2           2
        1           2           3
#          Parameters Jastrow two body
        -1  0.342214663461764
...
```

“fort.10” can be generated by “makefort10.x” (see later).

Header:

```
# Nelup  #Nel  # Ion
          2           4           1
# Shell Det.   # Shell Jas.
          3           3
# Jas 2body  # Det   #  3 body atomic par.
         -8          16           8
# Det mat. =/0  # Jas mat. =/0
          6           6
# Eq. Det atomic par.  # Eq. 3 body atomic. par.
         13           8
# unconstrained iesfree,iessw,ieskinr,I/O flag
          4           4           0           0
```

Nelup: The number of spin up electrons in the system.

Nel: The total number of electrons in the system.

Ion: The number of nuclei in the system.

Jas 2body: Onebody and Twobody Jastrow types

The number of atomic forces.

The total number of determinant variational param.

The total number of Jastrow variational param.

Coordinates:

```
# Ion coordinates
 N1 Z1              x1     y1     z1
 N2 Z2              x2     y2     z2
   ..              ..     ..     ..
 Nn Zn              xn     yn     zn
```

- N: Atomic number

- Z: The number of valence electrons

- xn, yn, zn : atomic positions (Bohr)

Pseudo potential case N != Z

If you want to use a H-pseudo potential, please put N=1.0, Z=1.00001 (dummy).

Basis set in makefort10.input/fort.10

Basis set for the determinant part:

```
#            Parameters atomic wf
Shell_Multiplicity      Number of par.
Ion index                [par (1, NUMBER OF PAR.)]

#            Parameters atomic wf
             1           1          16
             1  0.500000000000000
             3           1          36
             1   1.00000000000000
             1           1          16
             2  0.300000000000000
             1           1          16
             3  0.300000000000000
             1           1          16
             4  0.300000000000000
             1           1          16
             5  0.300000000000000
```

$$\phi(r) \sim \exp(-Zr^2)$$

```
#       Parameters atomic wf
        1           4          300
        1    2.0    1.0  3.231  7.54
```

$$\phi(r) = 3.231 \cdot \exp(-2.0 \cdot r^2) + 7.54 \cdot \exp(-1.0 \cdot r^2)$$

Shell codes:
- 16 -> s orbital
- 36 -> p orbital
- 68 -> d orbital
- 48 -> f orbital
- 51 -> g orbital
- 72 -> h orbital
- 73 -> i orbital

Single Slater-determinant (convertfort10mol.x)

Input: convertfort10mol.input, fort.10_in → Binary: convertfort10mol.x → Output:fort10_new

convertfort10mol.x is a tool for adding molecular orbitals to fort.10_in.

This is used for converting a JAGP WF to a JSD WF. JAGPs → JSD

JSD Slater Determinant

$$f\left(\mathbf{r}_i,\mathbf{r}_j\right)=\sum_{k=1}^{M=N_{\mathrm{el}}/2}\lambda_k\tilde{\Phi}_k\left(\mathbf{r}_i\right)\tilde{\Phi}_k\left(\mathbf{r}_j\right)$$

with $\tilde{\Phi}_k=\sum_{i=1}^{N_{\mathrm{basis}}}c_{i,k}\cdot\phi_i\left(\boldsymbol{r}\right)$

JAGPs

$$f\left(\mathbf{r}_i,\mathbf{r}_j\right)=\sum_{l,m,a,b}A_{\{a,l\},\{b,m\}}\psi_{a,l}\left(\mathbf{r}_i\right)\psi_{b,m}\left(\mathbf{r}_j\right)$$

DFT (prep.x) works only with molecular orbitals!! So, one should convert a WF from the JsAGPs to JSD.

Coefficients of the Determinant part (JAGPs case)

```
#          Nonzero values of detmat
           1       5      9.421753101774391E-002
           1       6      9.421753101774391E-002
           1       7      9.421753101774391E-002
```

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ & A_{22} & \dots & A_{2n} \\ & & \ddots & \vdots \\ & & & A_{nn} \end{pmatrix}$$

$$f\left(\mathbf{r}_i, \mathbf{r}_j\right) = \sum_{l,m,a,b} A_{\{a,l\},\{b,m\}} \psi_{a,l}\left(\mathbf{r}_i\right) \psi_{b,m}\left(\mathbf{r}_j\right)$$

$$g_{ud}(\mathbf{i},\mathbf{j}) \equiv g_s(\mathbf{i},\mathbf{j}) = f_S(\mathbf{r}_i,\mathbf{r}_j)\frac{|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle}{\sqrt{2}},$$

→ JAGPs

$A_{\{a,l\},\{b,m\}}$ is a symmetric matrix!

Molecular orbitals (100000): In fort.10, 1000000 indicates a molecular orbital.

```
       #always 1,   the number of components, 100000
       #index of basis [1,2,....]
       #coefficients for basis [1,2,....]
        1         180     1000000
        1           1           2           3           4           5
        6           7           8           9          10          11
        12          13          14          15          16          17
        18          19          20          21          22          23
        24          25          26          27          28          29
        30          31          32          33          34          35
        36          37          38          39          40          41
        42          43          44          45          46          47
        48          49          50          51          52          53
        54          55          56          57          58          59
        60          61          62          63          64          65
        66          67          68          69          70          71
        72          73          74          75          76          77
        78          79          80          81          82          83
        84          85          86          87          88          89
        90  0.438271164894104      -4.608166217803955E-002
0.189550578594208       7.299757003784180E-002 -0.129178702831268
-0.241831779479980      -7.793867588043213E-002 -0.143670558929443
-0.181271851062775      -0.265352427959442       0.374841809272766
```

$$\Phi_k = \sum_{i=1}^{N_{\text{basis}}} c_{i,k} \cdot \phi_i(\boldsymbol{r})$$

JSD

Molecular orbitals can be added by “convertfort10mol.x”. DFT works only with molecular orbitals.

Onebody and Twobody Jastrows

twobody: 1B and 2B Jastrows: Various Jastrow types are implemented (see the manual.)

Typically:

-6: Open/PBC with pseudo potentials

Only two-body parameter. $1\mathrm{b} = \frac{1}{2a}(1 - e^{-ar})$

i.e., electron-ion cusp conditions are enough.

-15: Open/PBC with all-electrons

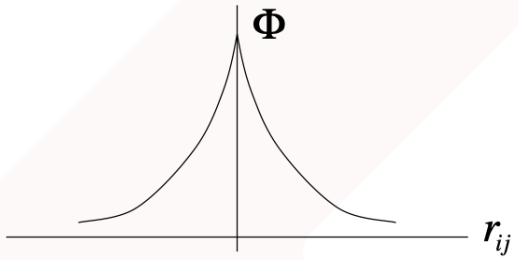
two-body and one-body parameters.

$$2\mathrm{b} = \frac{r}{2(1+br)} \quad 1\mathrm{b} = \frac{1}{2b}(1 - e^{-br})$$

-22: Open/PBC with JAGPu/JPf.

Only one-body or two-body and one-body parameters.

Spin-dependent Jastrow factors:



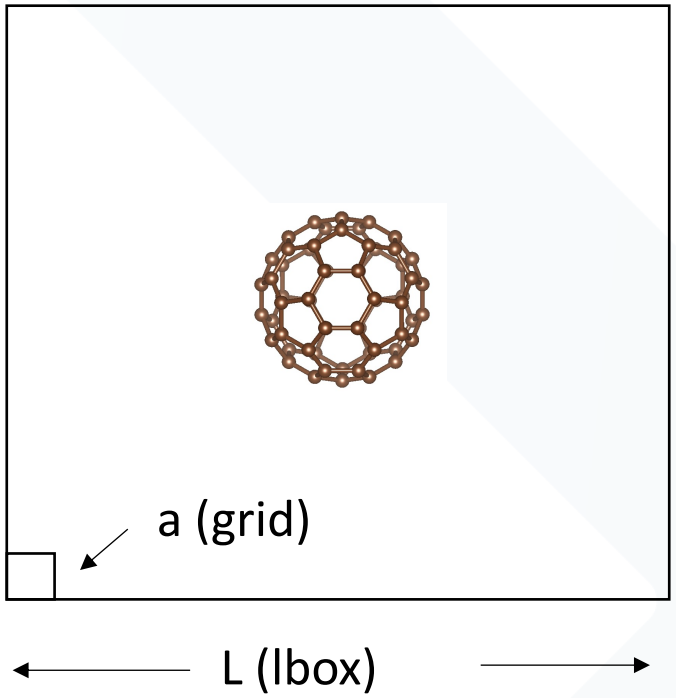
Electron-ion (1b).

Electron-Electron (2b).

To satisfy the cusp conditions.

Input: prep.input, fort.10 → Binary: prep-(serial, mpi).x → Output:fort10_new

Box and mesh sizes are so important for obtaining converged results in practice !!



$$H = \hat{T} + V_{\text{ele-ion}}\left(\vec{r}\right) + V_{\text{ele-ele}}\left(\vec{r}\right) + V_{\text{XC}}\left(\vec{r}\right)$$

For a calculation with PPs, a~0.10 bohr is small enough.

For an all-electron calculation, a < 0.05 bohr is needed.
The double-grid algorithm should also be helpful.

If you have enough memories, we recommend
L ~ 20 Bohr for the safety.

Lz = cell length for a periodic system. Automatically set.

Built-in DFT code (prep.x)

prep.x is a built-in DFT code!!

Why built-in?

$$\tilde{\phi}_j^b(\mathbf{r}-\mathbf{R}_b) = \phi_j^b(\mathbf{r}-\mathbf{R}_b)\,\tilde{J}_1(\mathbf{r})$$

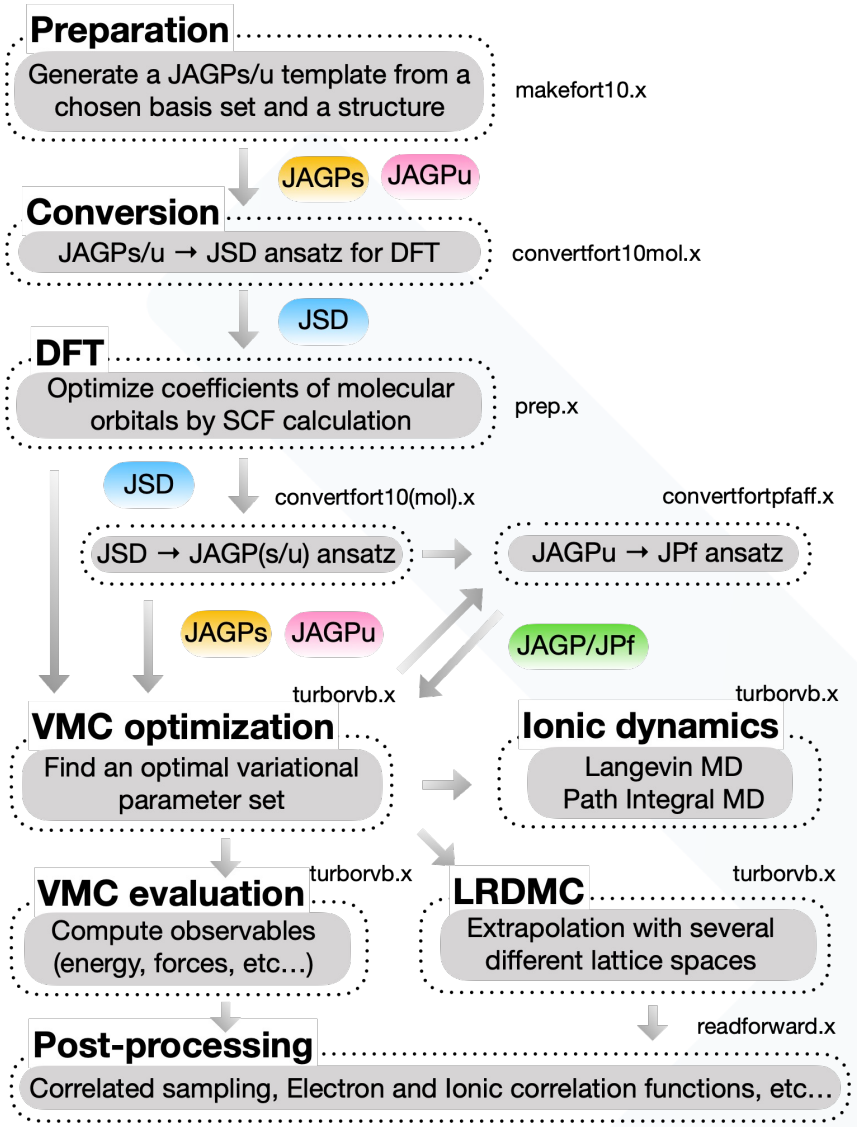
As mentioned before, the modified gaussian orbital is used.

So, we cannot exploit the analytical integration even though we employ the Gaussian primitive orbitals.

The CRYSTAL basis + cusp. for PBC cases.

We are also implementing converters for several QC codes (e.g., Gaussian) via TREX-IO.

A typical workflow in TurboRVB (TurboGenius)



= Workflow =

1. Prepare a structure and basis set — makefort10

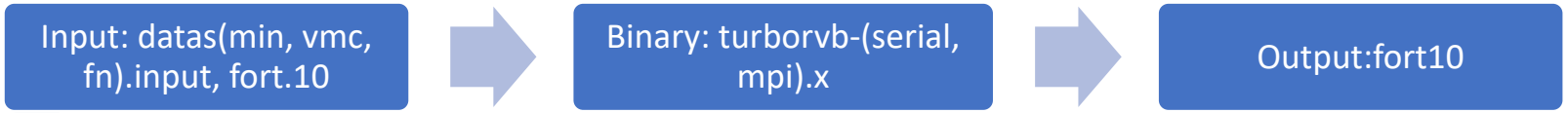
2. DFT — Construct a reasonable initial WF! — prep

3. VMC-opt — Optimize the wavefunction — turborvb

4. VMC — Do a VMC run. — turborvb

5. LRDMC — LRDMC with the optimized WF. — turborvb

Main QMC engine: turborvb-(serial, mpi).x



turborvb.x is the main QMC engine in the turborvb package.

=VMC-opt, VMC, DMC, and LRDMC=

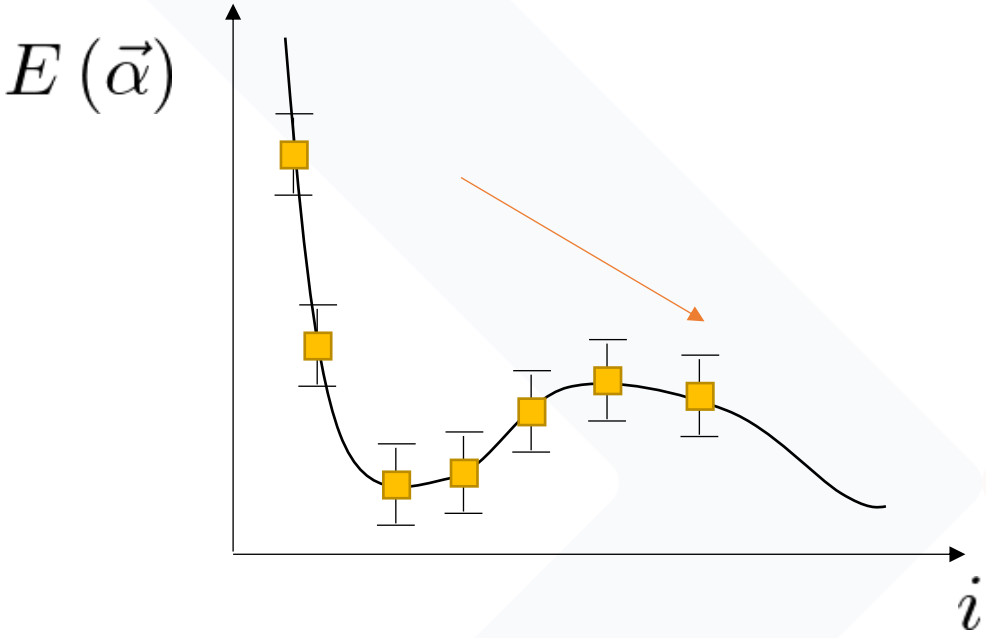
- Single-shot VMC run (itestr4=-2 in the &simulation namelist).
- VMC optimization (itestr4=-4,-5,-8,-9 in the &simulation namelist).
- Single-shot LRDMC run (itestr4=-6 in the &simulation namelist).
- Single-shot DMC run (itestr4=-5 in the &simulation namelist), but not maintained.

What the VMC optimization does?

$$E\left(\vec{\alpha}\right) = \frac{\int d\vec{R} \cdot \Psi^{*}\left(\vec{R}, \vec{\alpha}\right) \cdot \hat{\mathcal{H}}\Psi\left(\vec{R}, \vec{\alpha}\right)}{\int d\vec{R} \cdot \Psi^{*}\left(\vec{R}, \vec{\alpha}\right) \Psi\left(\vec{R}, \vec{\alpha}\right)} \geq E_0$$

The variational principle

This integral is evaluated using the MCMC method.



Variational parameters!

$$\vec{\alpha}_{i+1} \leftarrow \vec{\alpha}_i + \Delta\vec{\alpha}$$

e.g.,

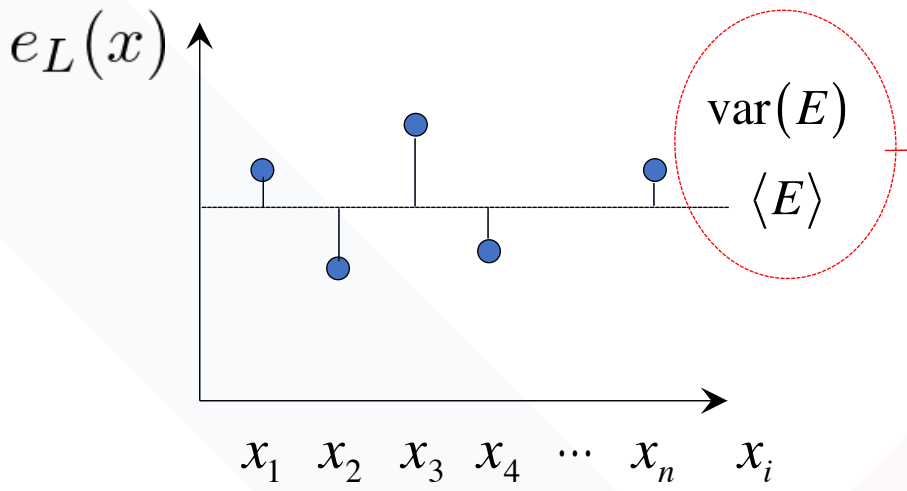
$$f_S\left(\mathbf{r}_i, \mathbf{r}_j\right) = \sum_{l,m,a,b} A_{\{a,l\},\{b,m\}} \psi_{a,l}\left(\mathbf{r}_i\right) \psi_{b,m}\left(\mathbf{r}_j\right).$$

The number of MCMC steps

```
&simulation
     itestr4=-4
     ngen=10000
     iopt=1
     maxtime=10800
/

&pseudo
/

&vmc
     epscut=0.0
/

&optimization
     nweight=100
     nbinr=5
     iboot=0
     tpar=0.35
     parr=-0.001
/

&readio
     !iread=3
/

&parameters
     !iessw=1
     !iesup=1
     !iesm=1
     !ieskin=1
     iesd=1
     iesfree=1
/
```

Relation between ngen and nweight

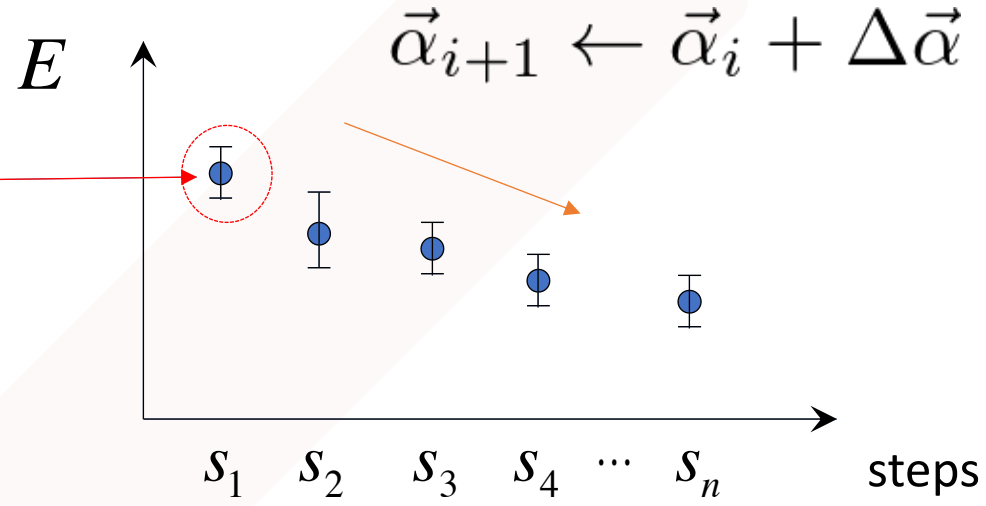
Each iteration



$=$

nweight $\times$ nw

Optimization



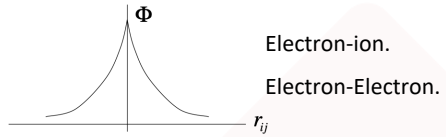
Total optimization steps: $\dfrac{\mathrm{ngen}}{\mathrm{nweight}}$

(default: the num. walker = the num. of MPI process)

The most difficult part in practice

3. VMC　Optimize wavefunctions and VMC run.

A wavefunction reads $\Psi_{AS}(\vec{R}) \times \exp(J(\vec{R}))$ ⟶



Anti-symmetric part.　Jastrow factor.

To satisfy the cusp.

Jastrow factor = $\exp(J(\vec{R}))$　No effect on the nodal surface!!　$\Psi(\vec{R}) \equiv \Psi(\vec{r_1}, \vec{r_2}, \cdots, \vec{r_N}) = 0$

Anti-symmetric part = $\Psi_{AS}(\vec{R})$　Determines the nodal surface.　Its initial guess is taken from a DFT calculation!!

1st Step JSD　　2nd Step JAGPs

Implemented optimization methods

Implemented optimization algorithms

-9, -5): Stochastic reconfiguration (natural gradient method)

S Sorella, et al., ***J. Chem. Phys.*** 127, 014105 (2007).

-4, -8): Linear method with the natural gradient

C.J. Umrigar, et al., ***Phys. Rev. Lett.*** 98, 110201 (2007).

In both cases, the most important parameters in practice are

1. tpar: Acceleration parameter (learning rate.)

e.g., $\alpha_k \rightarrow \alpha_k + \Delta \cdot \left(\boldsymbol{S'}^{-1} \mathbf{f} \right)_k$

tpar = 3.5d-1, and 1.0d-3 for -4 and -9, respectively.

2. parr: Regularization (c.f. LASSO)

e.g., $s'_{i,i} = s_{i.i}(1+\varepsilon)$

Depending on the accuracy your need. parr = ~ 1.0d-3

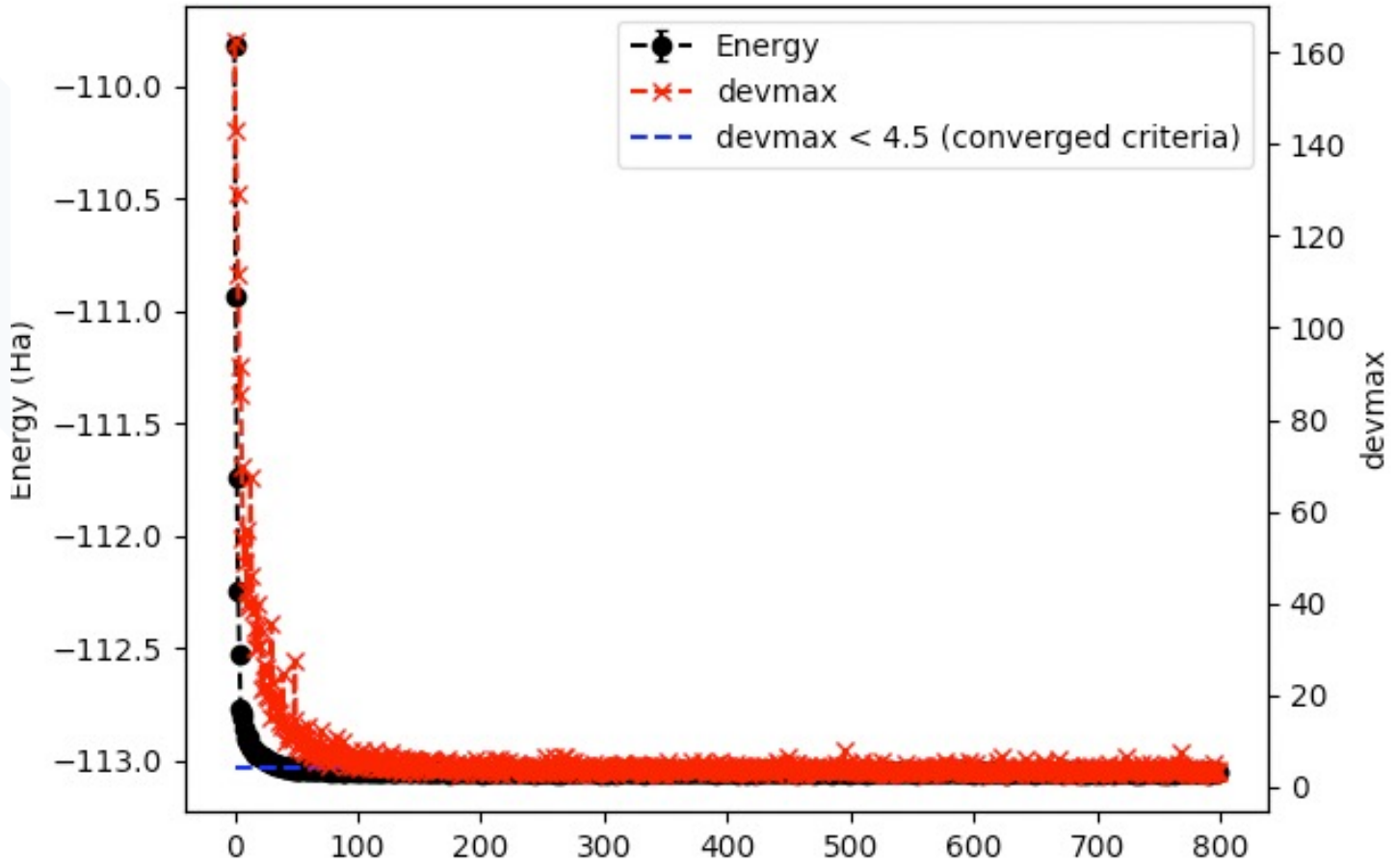
Optimization criteria

At least, ``devmax`` should be smaller than 4.0 after optimization. However, we also have experienced that this simple criteria is sometimes not sufficient to obtain a converged result.

The definition of ``devmax`` is: $devmax \equiv \max_k \left(\left| \frac{f_k}{\sigma_{f_k}} \right| \right)$

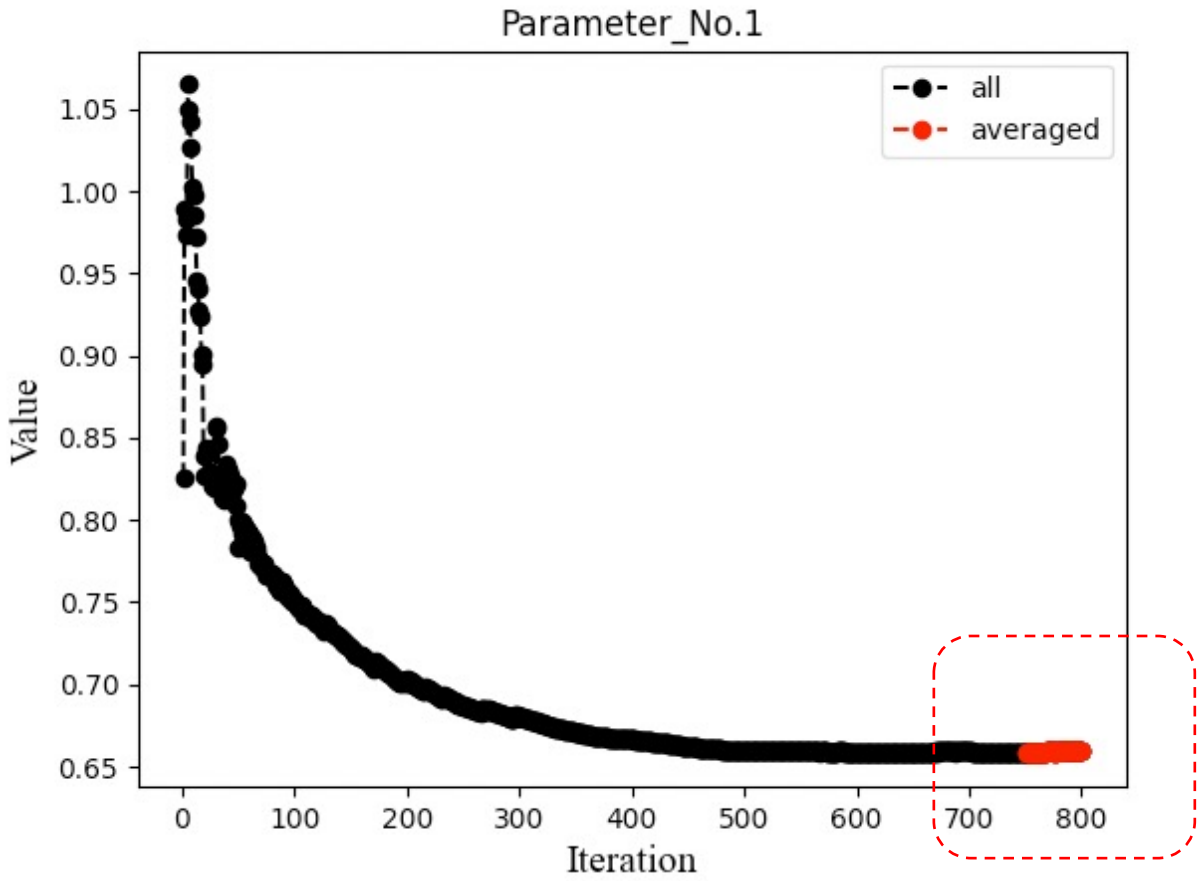
where σ_{f_k} represents the estimated error bar of a general force $f_k = -\frac{\partial E(\alpha)}{\partial \alpha_k} = -\frac{\partial}{\partial \alpha_k} \frac{\langle \Psi_\alpha | \hat{\mathcal{H}} | \Psi_\alpha \rangle}{\langle \Psi_\alpha | \Psi_\alpha \rangle}.$

Plotting Graphs (Turbo-Genius)



%turbo-genius.sh -j vmcopt -post -am interactive_detail

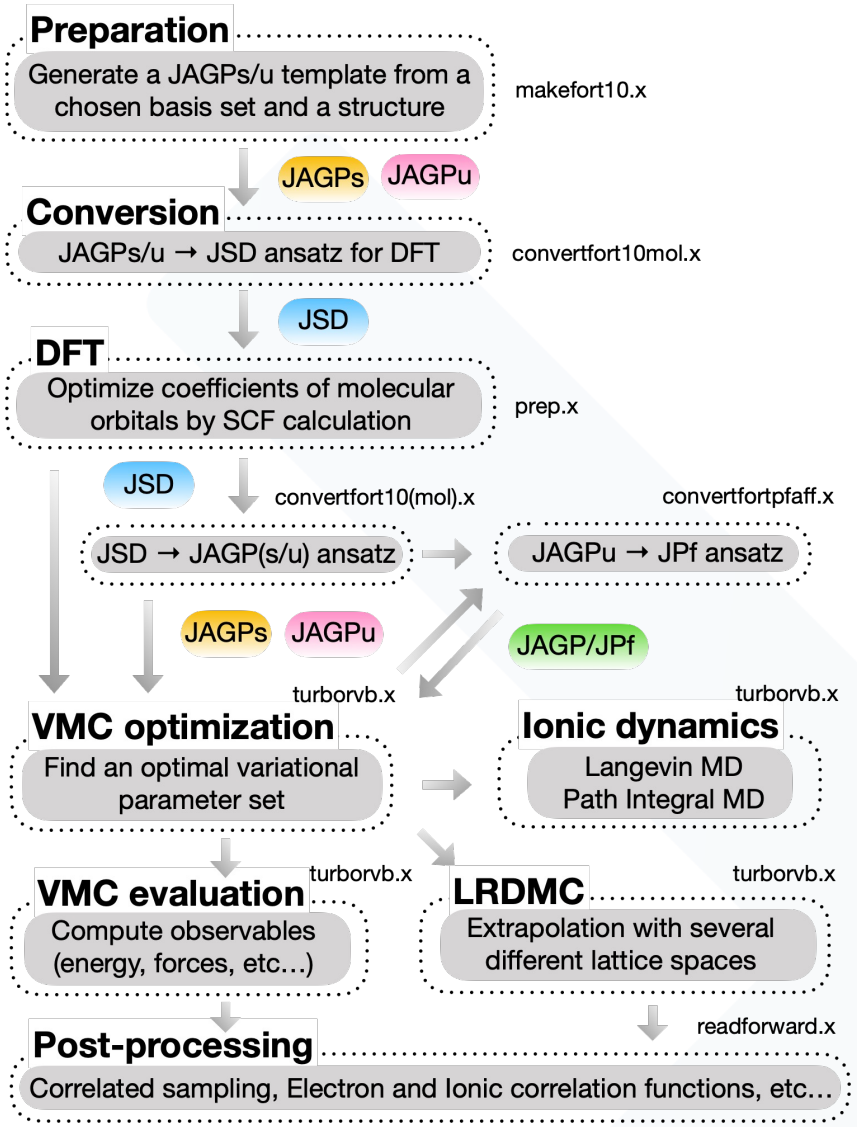
Averaging parameters (Turbo-Genius)



Averaged!

turbo-genius.sh -j vmcopt -post -am interactive_detail

A typical workflow in TurboRVB (TurboGenius)

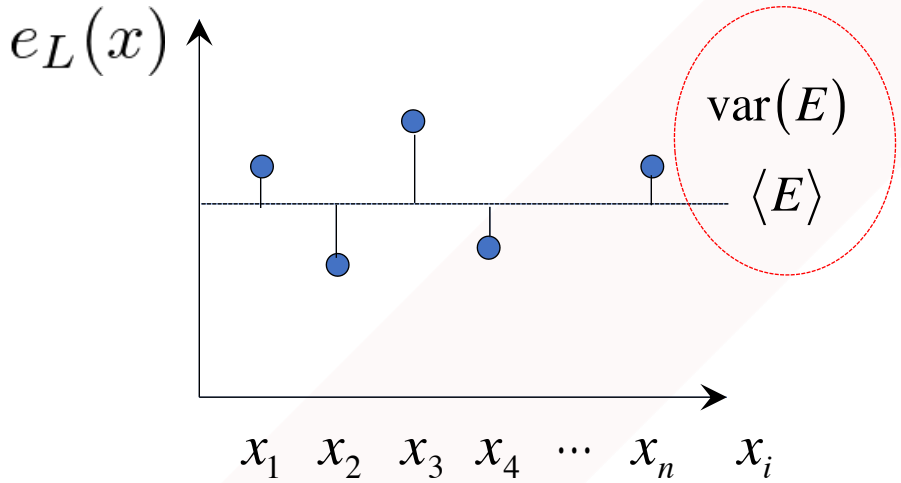


= Workflow =

1. Prepare a structure and basis set — makefort10
2. DFT — Construct a reasonable initial WF! — prep
3. VMC-opt — Optimize the wavefunction — turborvb
4. VMC — Do a VMC run. — **turborvb**
5. LRDMC — LRDMC with the optimized WF. — turborvb

```
&simulation
    itestr4=2
    ngen=10000
    maxtime=10800
    iopt=1
/

&pseudo
/

&vmc
    !epscut=0.0
/

&readio
    !iread=3
/

&parameters
    !ieskin=1
/
```

One-shot VMC



=

ngen × nw

ngen is the total number of Monte Carlo steps.

Default: nw is the number of MPI processes.

All information

```python
from scipy.io import FortranFile
import numpy as np

# check length of fort.12
f = FortranFile('fort.12', 'r')
a = f.read_reals(dtype='float64')
column_length = len(a)
f.close()

# start reading fort.12
head = ("head", "<i")
tail = ("tail", "<i")
dt = np.dtype([head, ("a", "<{}d".format(column_length)), tail])
fd = open('fort.12', "r")
fort12 = np.fromfile(fd, dtype=dt, count=-1)
data_length=len(fort12)
fd.close()
# end reading fort.12

print(fort12)
```

```
# for ngen=10
>>> fort12
array([(40, [  1.        ,   1.        , -11.23924971, -11.23924971, 126.32073395], 40),
       (40, [  1.        ,   1.        , -11.4465321 , -11.4465321 , 131.02309712], 40),
       (40, [  1.        ,   1.        , -11.25058355, -11.25058355, 126.57563015], 40),
       (40, [  1.        ,   1.        , -11.88021352, -11.88021352, 141.13947319], 40),
       (40, [  1.        ,   1.        , -10.89686295, -10.89686295, 118.74162225], 40),
       (40, [  1.        ,   1.        , -11.8906161 , -11.8906161 , 141.38675112], 40),
       (40, [  1.        ,   1.        , -10.50040878, -10.50040878, 110.25858451], 40),
       (40, [  1.        ,   1.        , -10.85804034, -10.85804034, 117.89704005], 40),
       (40, [  1.        ,   1.        , -11.3042634 , -11.3042634 , 127.78637111], 40),
       (40, [  1.        ,   1.        , -10.86745849, -10.86745849, 118.10165397], 40)],
      dtype=[('head', '<i4'), ('a', '<f8', (5,)), ('tail', '<i4')])
```

e(L), etc… -> written in fort.12

forcevmc.sh “bin”, “init”, “pulay”, or
turbo-genius.sh –j vmc -post -reb “bin”, -eq “init”

“bin”: the length of reblocking size

“init”: the length of equilibration steps (init * bin)

“pulay”: the ratio of the pulay term (1 is OK)

pip0.d=energy

```
#cat pip0.d

number of bins read =        1496
Energy =  -1.1379192772188327        1.7589095174214898E-004
Variance square =   1.7369139136828382E-003   2.7618833870090571E-005
Est. energy error bar =   1.7510470092362484E-004   3.9800256121536918E-006
Est. corr. time  =   2.6420266523220208       0.10738159557488412
```

forcevmc.dat=forces

```
Force component 1
Force    =  6.0047638692014 90E-003  4.997922374161991E-005
6.2735656333633 22E-007
Der Eloc =  6.9276758527247 24E-003  4.999242839793062E-005
<OH> =  0.557134685159244       7.437283601136703E-005
<O><H> = -0.557596141151006       7.447559481785158E-005
2*(<OH> - <O><H>) = -9.229119835232336E-004  2.922997214772288E-006
```

Ionic Forces:

```
# Constraints for forces: ion - coordinate
          1      1      3
```

The number of forces, atom index, direction

F1,z for the first atom will be calculated.

```
# Constraints for forces: ion - coordinate
          2      1      1      2      -3
```

F1,x and F2,z will be calculated, assuming, F = F1,x = - F2,z.

The output value (forcevmc.dat) is the sum of two forces, i.e., (F = F1,x - F2,z.)

If you want to calculate forces, please set “ieskin=1” in the &parameter section in your VMC input.

forcevmc.dat=forces

```
Force component 1
Force    =  6.004763869201490E-003  4.997922374161991E-005
6.273565633363322E-007
Der Eloc =  6.927675852724724E-003  4.999242839793062E-005
<OH> =  0.557134685159244       7.437283601136703E-005
<O><H> = -0.557596141151006       7.447559481785158E-005
2*(<OH> - <O><H>) = -9.229119835232336E-004  2.922997214772288E-006
```

Force (total) $$F_\alpha = -\langle \frac{d}{d\mathbf{R}_\alpha} E_L \rangle - 2 \left(\langle E_L \cdot \frac{d}{d\mathbf{R}_\alpha} \log(J^{1/2}\Psi) \rangle - \langle E_L \rangle \cdot \langle \frac{d}{d\mathbf{R}_\alpha} \log(J^{1/2}\Psi) \rangle \right),$$

Der Eloc: (Hellmann-Feynman term)

2*(<OH> - <O><H>) (Pulay term)

where, J is the Jacobian of the warp transformation. S Sorella, L Capriotti, ***J. Chem. Phys.*** 133, 234111 (2010).

```
# Constraints for forces: ion - coordinate
          2      1      1      2      -3
```

The output value (forcevmc.dat) is the sum of two forces, i.e., (F = F1,x - F2,z.)

How to choose basis set for Force calc.? (PBC)

TurboRVB employs the CRYSTAL periodic basis for PBC calculations:

$$\psi^{\mathrm{PBC}}_{l,m,I}(\mathbf{r};\zeta) = \sum_{\mathbf{T}_s} \psi_{l,m,I}(\mathbf{r}+\mathbf{T}_s;\zeta)\, e^{-i\mathbf{k}_s\cdot\mathbf{T}_s}$$

-PBC, pseudo potential:

Unfortunately, provided basis sets for open systems are redundant for periodic cases, so we recommend that one should cut several smaller exponents, typically, smaller than 0.10.

-PBC, all-electron:

The same for all-electron cases. Basis sets provided for open systems such as Basis set exchange [https://www.basissetexchange.org] are usually redundant for a periodic case, so we recommend that one should cut several smaller exponents, typically, smaller than 0.10.

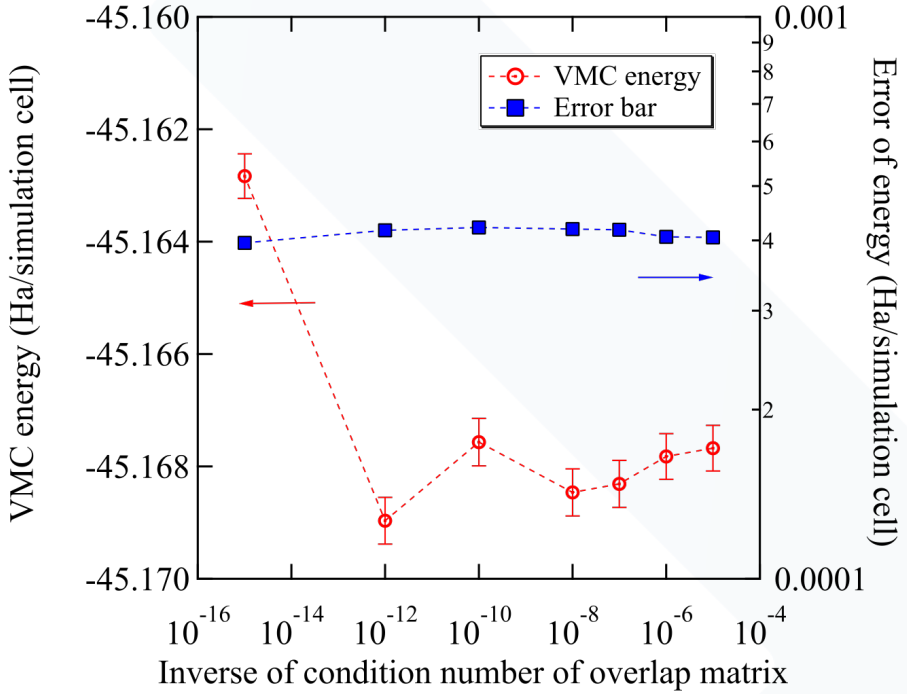
One can also use all-electron basis sets optimized for periodic systems such as ones provided in the CRYSTAL DFT code [https://www.crystal.unito.it/basis-sets.php].

Basis set redundancy for periodic cases

$$\Psi = J \times \Psi_{\mathrm{SD}} \qquad \phi_i^{\mathbf{R}} = \sum_{a,l} c_{i,\{a,l\}} \psi_{\{a,l\}}^{R_a}$$

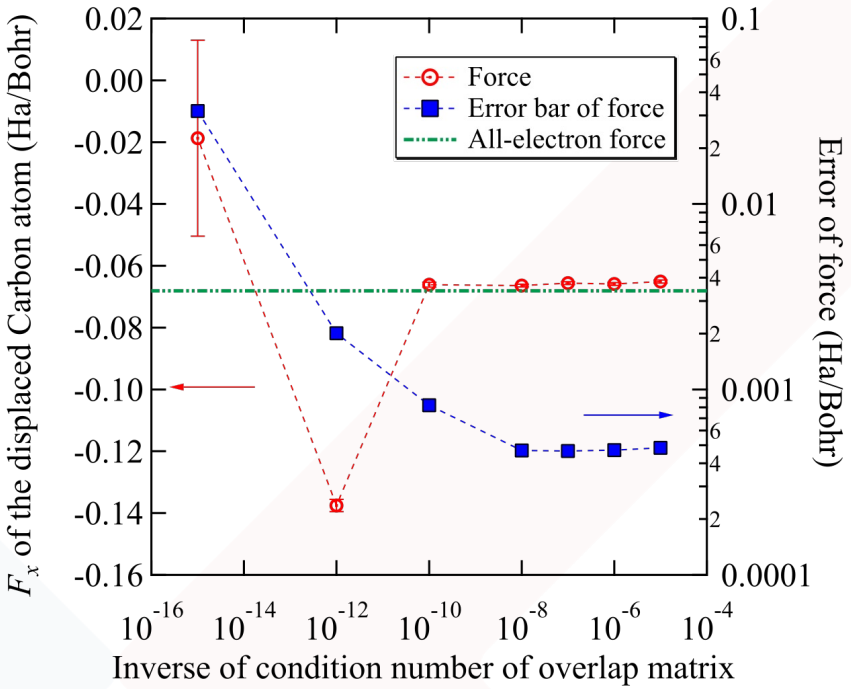
Linear dependency = the condition number of the overlap matrix (S).

- Diamond: Total Energy (E)

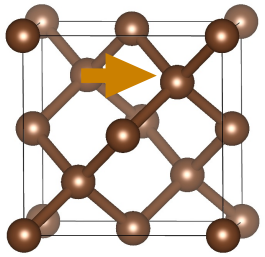


Large <- Lin. Dep. -> Small

- Diamond: Force (F)



Large <- Lin. Dep. -> Small



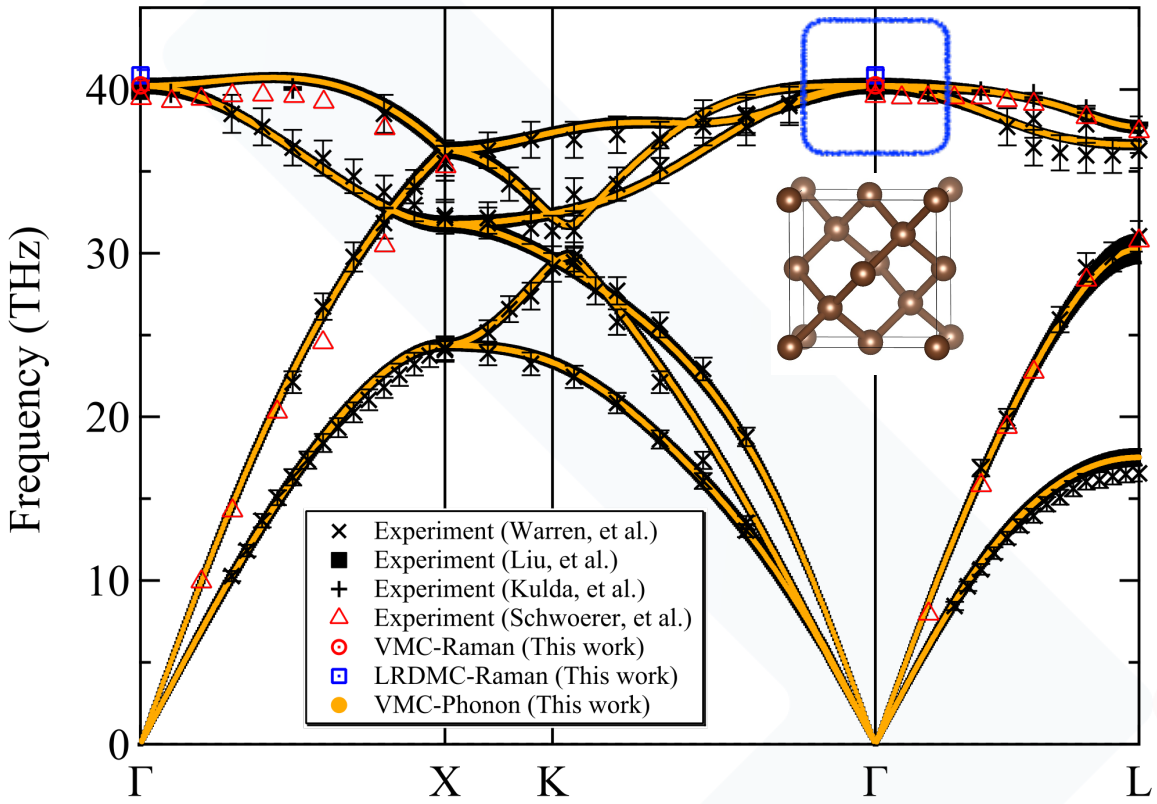
Only one C is displaced

K. Nakano *et al.*, ***Phys. Rev. B*** 103, L121110 (2021)

Recent progress: Phonon dispersion calculations

- Diamond: the conventional 2x2x2 supercell with the experimental lattice parameter
- The frozen phonon method implemented in Phonopy package.

A. Togo and I. Tanaka, *Scr. Mater.* 108, 1 (2015).



* including anharmonic corrections.

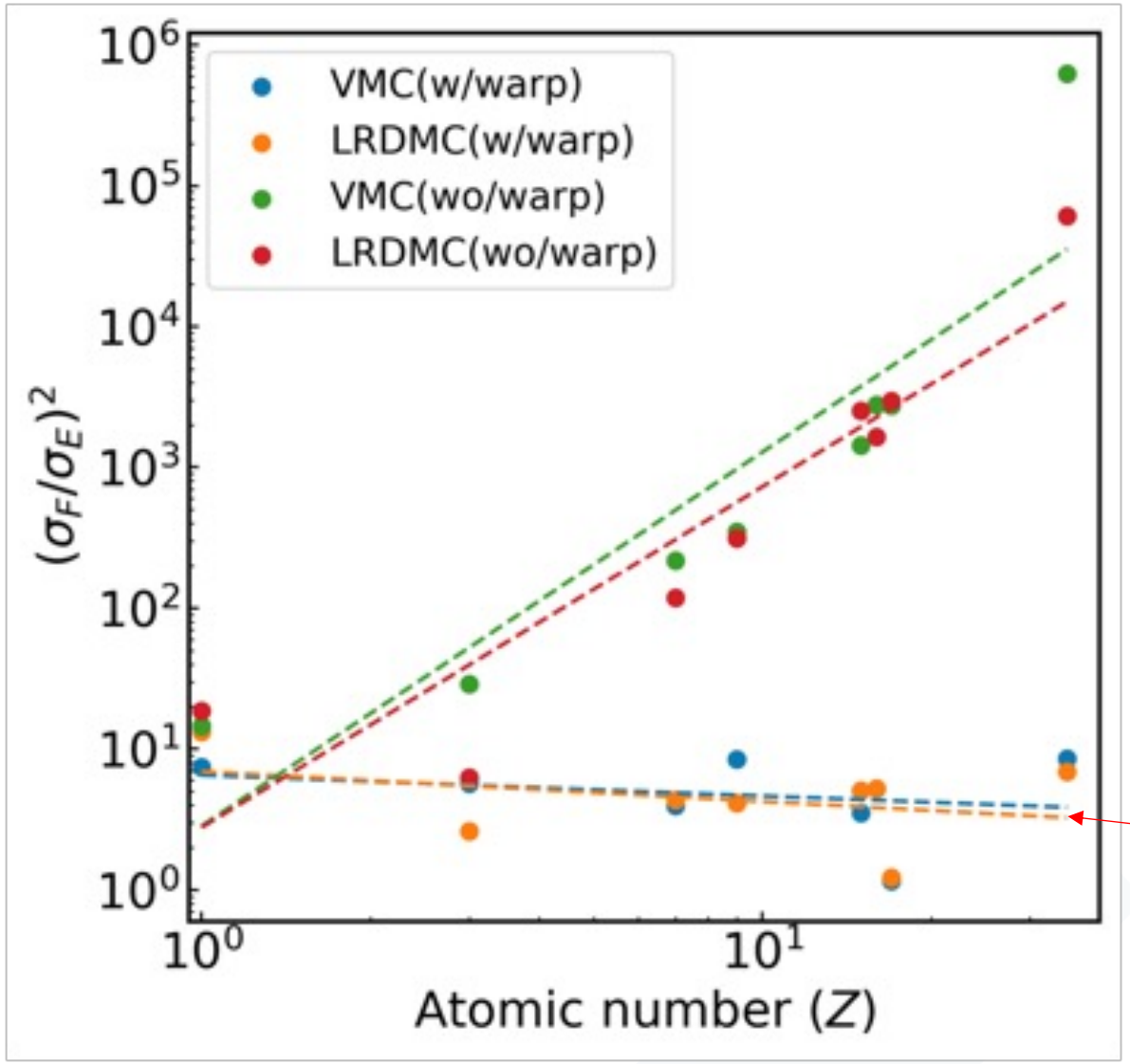
Raman Freq. (optical phonon at Γ)

DFT-LDA	38.55 THz
VMC	40.65(38) THz
Exp.	40.35 THz

** These are harmonic frequencies

K. Nakano *et al.*, ***Phys. Rev. B*** 103, L121110 (2021)

Importance of the Space-warp coordinate transformation



$(\sigma_F/\sigma_E)^2$ scales as $Z^{\sim 2.5}$ **without SWCT**,

consistent with QMCPACK group's paper

J. Tiihonen, et al. *J. Chem. Phys.* 154, 204111 (2021)

QMCPACK group shows that the scaling does not change even **with SWCT**…

No, the ratio is independent of Z !!

K. Nakano *et al.*, ***J. Chem. Phys.*** 156, 034101 (2022)

Energy derivative v.s. Force

$$\frac{dE}{dR_\alpha} = \langle \frac{\partial}{\partial R_\alpha} E_L \rangle + 2(\langle E_L \frac{\partial}{\partial R_\alpha} \log \Psi \rangle - \langle E_L \rangle \langle \frac{\partial}{\partial R_\alpha} \log \Psi \rangle)$$

$$+ \sum_{i=1}^{N_{\mathrm{par}}} \frac{\partial E}{\partial c_i} \frac{\partial c_i}{\partial R_\alpha}$$

Additional terms!!

JSD ? $\phi_i^{\mathbf{R}} = \sum_{a,l} c_{i,\{a,l\}} \psi_{\{a,l\}}^{R_a}$

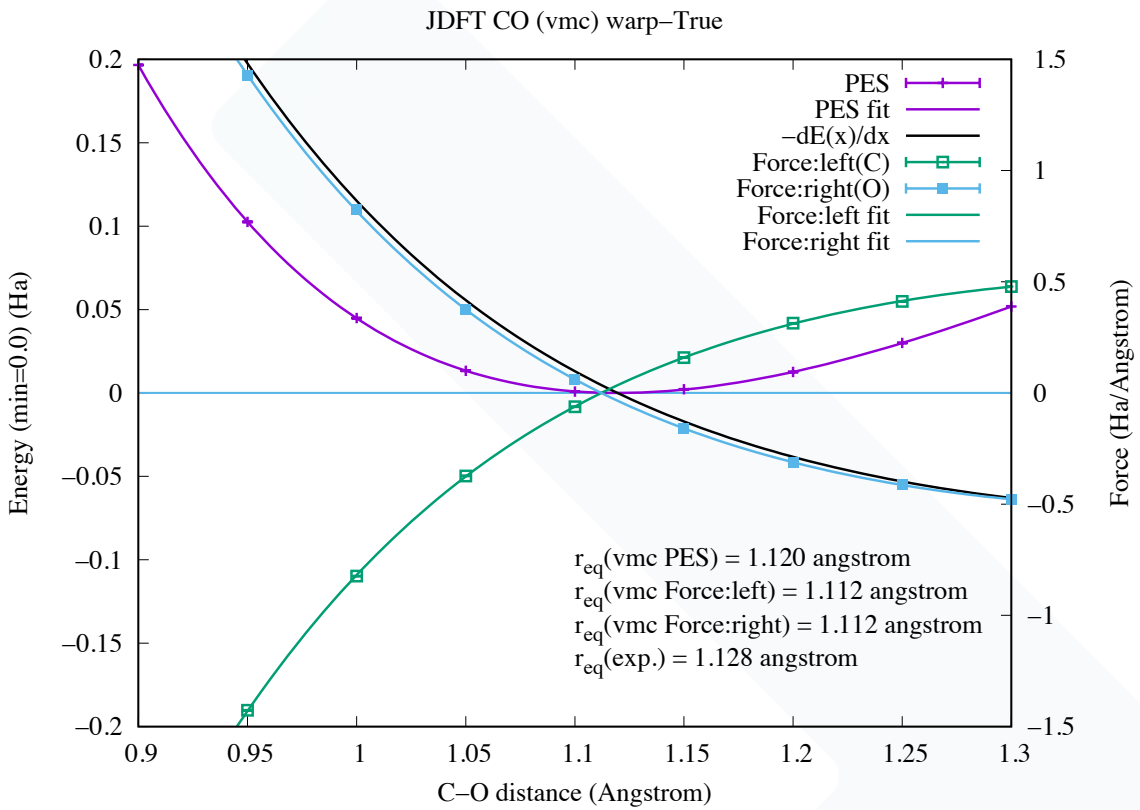
1. The system is already at a variational minimum. $\frac{\partial E}{\partial c_i} = 0$ → JAGPs ○

2. The variational parameters are not allowed to vary with changing the atomic pos. $\frac{\partial c_i}{\partial R_\alpha} = 0$

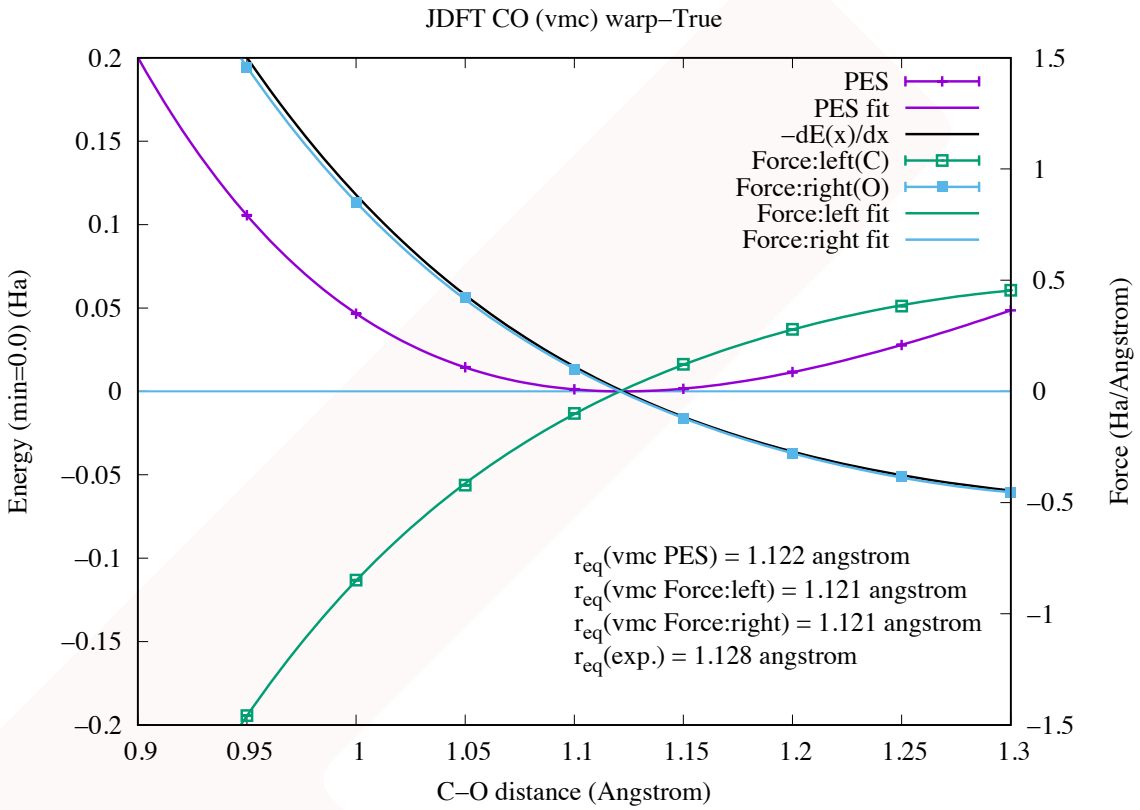
J. Tiihonen et al., ***J. Chem. Phys.*** 154, 204111 (2021)

Carbon monoxide: PES v.s. Force

All-electron calculations, VMC (**JDFT**). Jastrow factors were optimized for each C-O distance.

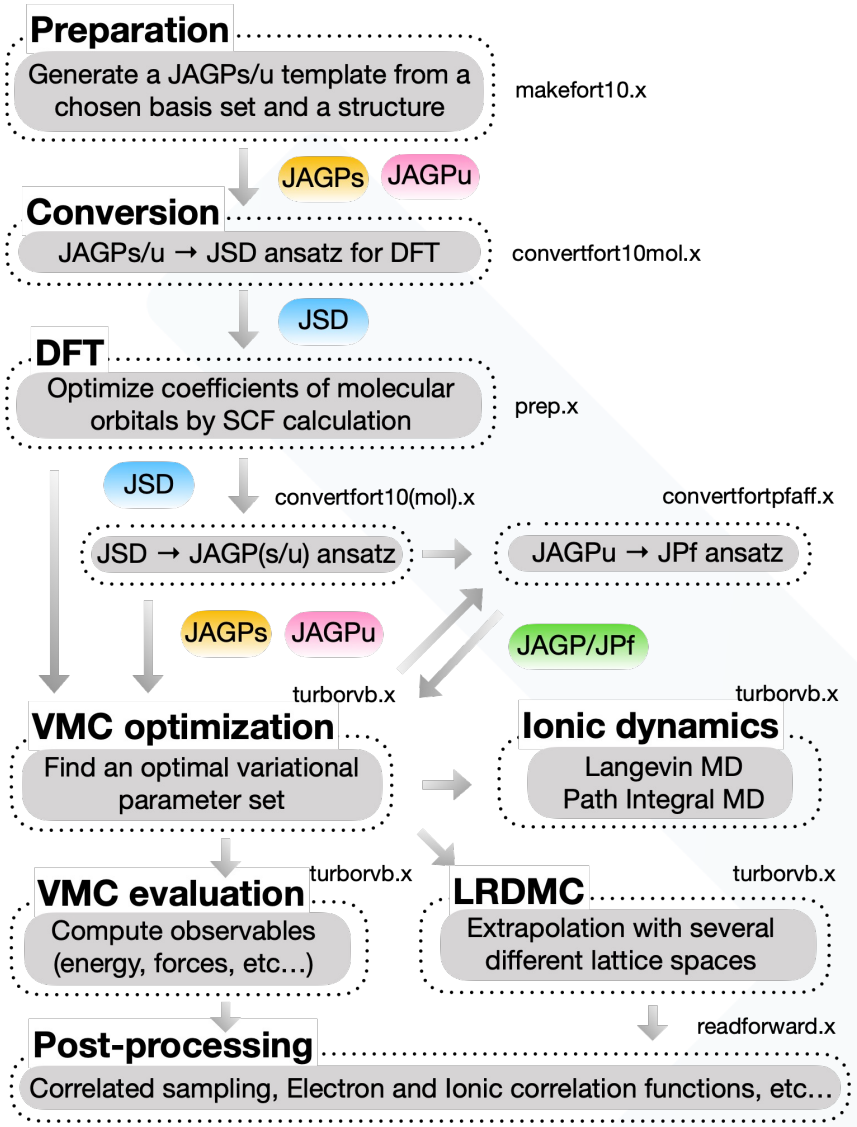


Basis = cc-pVDZ



Basis = cc-pVQZ

A typical workflow in TurboRVB (TurboGenius)



= Workflow =

1. Prepare a structure and basis set — makefort10
2. DFT — Construct a reasonable initial WF! — prep
3. VMC-opt — Optimize the wavefunction — turborvb
4. VMC — Do a VMC run. — turborvb
5. LRDMC — LRDMC with the optimized WF. — **turborvb**

The projection technique to filer out the ground state from a trial wave function (typically, optimized by VMC).

M. Casula et al., ***Phys. Rev. Lett.*** 95, 100201 (2005)

$$|\Upsilon_0\rangle \propto \lim_{M\to\infty} \left(\mathbf{\Lambda} - \hat{\mathcal{H}}\right)^M |\Psi_{\mathrm{T}}\rangle$$

$$= \lim_{M\to\infty} (\lambda - E_0)^M \left[a_0 |\Upsilon_0\rangle + \sum_{n\neq 0} \left(\frac{\lambda - E_n}{\lambda - E_0}\right)^M a_n |\Upsilon_n\rangle \right],$$

Since $\frac{\lambda - E_n}{\lambda - E_0} < 1$ the projection filters out the ground state WF from a given trial WF

In TurboRVB, “etry” is the corresponding parameter. $\lambda = -2 \times \mathrm{etry}$

e.g., one can use a VMC energy for etry.

To apply the GFMC method for continuous systems.

M. Casula et al., ***Phys. Rev. Lett.*** 95, 100201 (2005)

$$\Delta_i f(x_i, y_i, z_i) \approx \Delta_i^a f(x_i, y_i, z_i)$$
$$\equiv \frac{1}{a^2}\{[f(x_i + a) - f(x_i)] + [f(x_i - a) - f(x_i)]\}$$
$$\leftrightarrow y_i \leftrightarrow z_i,$$

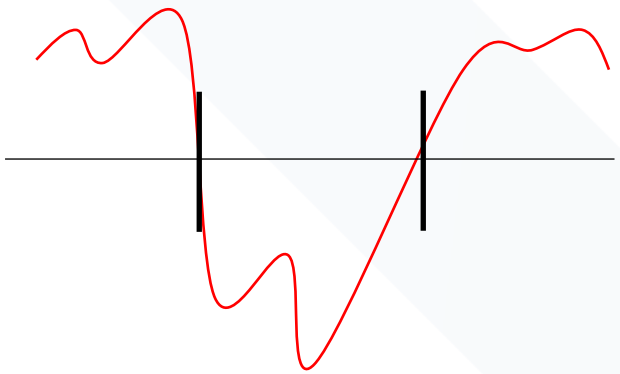
$$V^a(\mathbf{x}) = V(\mathbf{x}) + \frac{1}{2}\left[\frac{\sum_i (\Delta_i^a - \Delta_i)\Psi_G(\mathbf{x})}{\Psi_G(\mathbf{x})}\right].$$

In TurboRVB, "**alat**" is the corresponding parameter. The unit is bohr.

Since the Trotter-Suzuki decomposition is not needed in the LRDMC framework, the "time-step" error does not exist in LRDMC unlike DMC, but this "lattice-space" error exists instead. We need extrapolation for alat. (later)

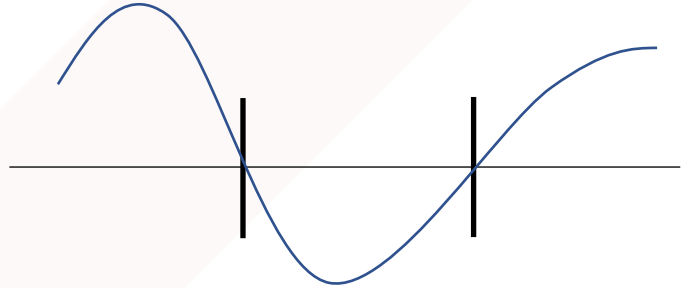
The Green's function cannot be made strictly positive for fermions; therefore, the fixed-node (FN) approximation has to be introduced in order to avoid the sign problem.

Trial WF



Projection.

Fixed-node WF



The nodal surface never changes during the simulation! i.e., Only the amplitude is relaxed.

Input file for Lattice regularized DMC (LRDMC)

```
&simulation
    itestr4=-6
    ngen=24100
    iopt=1
    maxtime=10800
/

&pseudo
/

&dmclrdmc
    tbra=0.1
    etry=-5.50
    Klrdmc=0.0
    alat=-0.40
    !iesrandoma=.true.
    !alat2=0.0
    gamma=0.0
    parcutg=1
/

&readio
    !iread=2
/

&parameters
    !ieskin=1
/
```

M. Casula et al., ***Phys. Rev. Lett.*** 95, 100201 (2005)

Important parameters:

Itestr4 = -6: LRDMC

ngen: The number of iterations (branchings)

tbra: Projection time

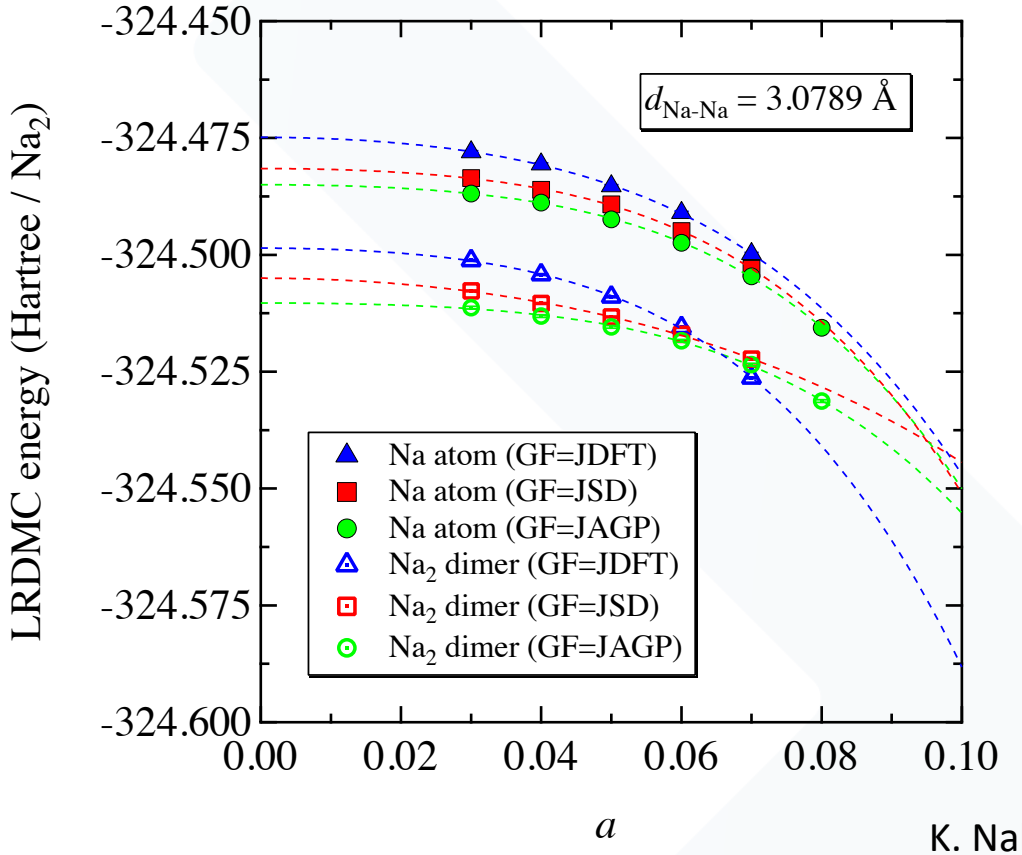
etry: Energy shift for the projection

alat: Coarser grid size (Bohr).

alat2: Denser grid size (Bohr).

$$\Delta_i f(x_i, y_i, z_i) \approx \Delta_i^a f(x_i, y_i, z_i)$$
$$\equiv \frac{1}{a^2}\{[f(x_i + a) - f(x_i)] + [f(x_i - a) - f(x_i)]\}$$

The extrapolation behaves well unlike the standard DMC!!



alat extrapolation with funvsa.x

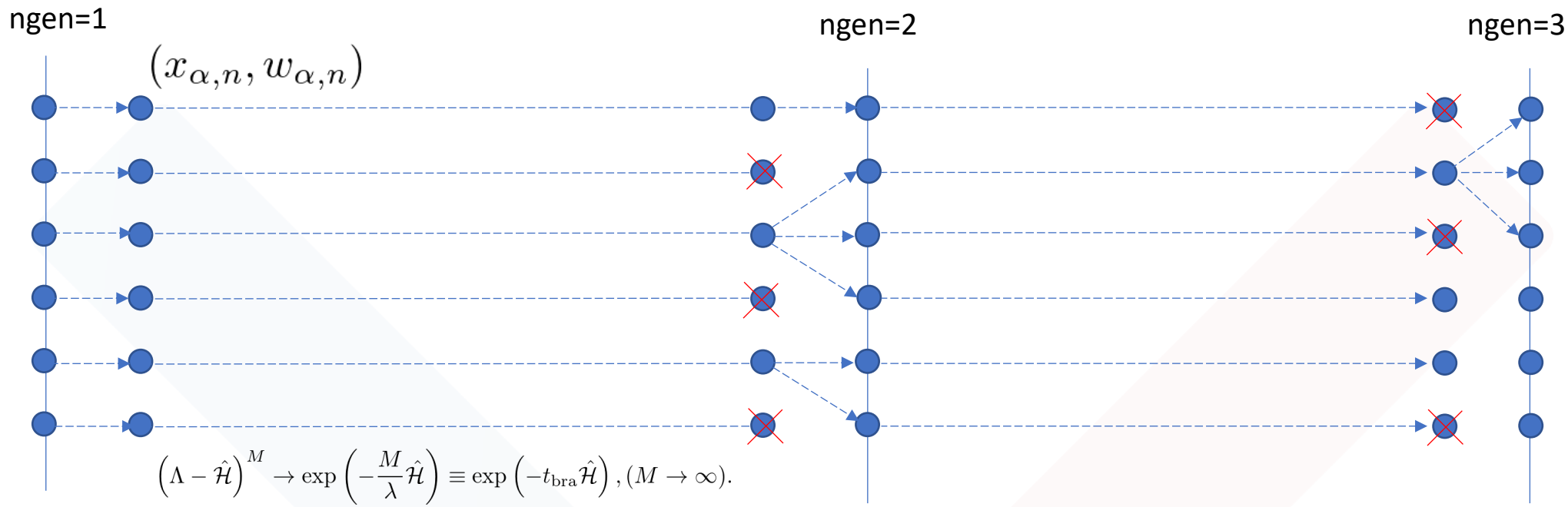
quartic: $E(a) = E(0) + k_1 \cdot a^2 + k_2 \cdot a^4$

quadratic: $E(a) = E(0) + k_1 \cdot a^2$

```
# See. Readme of funvsa.x in detail.
# 2=(up to a^4) number of data 4 1
2  5  4  1
0.10 -11.0850188375511 1.250592379643920E-004
0.20 -11.0854289356563 1.239503202184784E-004
0.30 -11.0855955871707 1.334024389855123E-004
0.40 -11.0860656088368 1.279739901272860E-004
0.50 -11.0868942724581 1.340429878094154E-004
```

K. Nakano et al., ***J. Chem. Theory Comput.*** 15, 4044-4055 (2019)

Branching and related parameters



weights and positions are updated.

Branching!!

1. Set the new weights $w'_{\alpha,n} = \bar{w} \equiv \frac{1}{N_w}\sum_\beta w_{\beta,n}.$
2. Select the new walkers $p_{\alpha,n} = w_{\alpha,n}/\sum_\beta w_{\beta,n},$

- “ngen” is the number of branchings! (ngen).
- The branching is done every τ_{nbra} steps. (tbra).
  - too small tbra -> The weights are not update.
  - too large tbra -> Only few walkers survive.

Check your output! Av. num. of survived walkers/ # walkers in the branching 0.99 > 0.90!

forcefn.sh “bin”, “corr”, “init”, “pulay”, or

turbo-genius.sh –j lrdmc -post -reb “bin”, -eq “init” -col “corr”

pip0_fn.d=energy

```
% cat pip0_fn.d
number of bins read =        1201
Energy =  -11.0854289356563       1.239503202184784E-004
Variance square =  0.126708380716482       1.148750765092961E-003
Est. energy error bar =  1.234807072779590E-004  2.503947626011507E-006
Est. corr. time  =   1.85075908836029       7.596952532743223E-002
Energy (ave) = -11.0854159959592 1.144905833254917E-004
```

forcefn.dat=forces

```
Force component 1
Force    =  6.004763869201490E-003  4.997922374161991E-005
6.273565633363322E-007
Der Eloc =  6.927675852724724E-003  4.999242839793062E-005
<OH> =  0.557134685159244       7.437283601136703E-005
<O><H> = -0.557596141151006       7.447559481785158E-005
2*(<OH> - <O><H>) = -9.229119835232336E-004  2.922997214772288E-006
```

“bin”: the length of reblocking (binning) size

“corr”: correcting factor

“init”: the length of equilibration steps (init * bin)

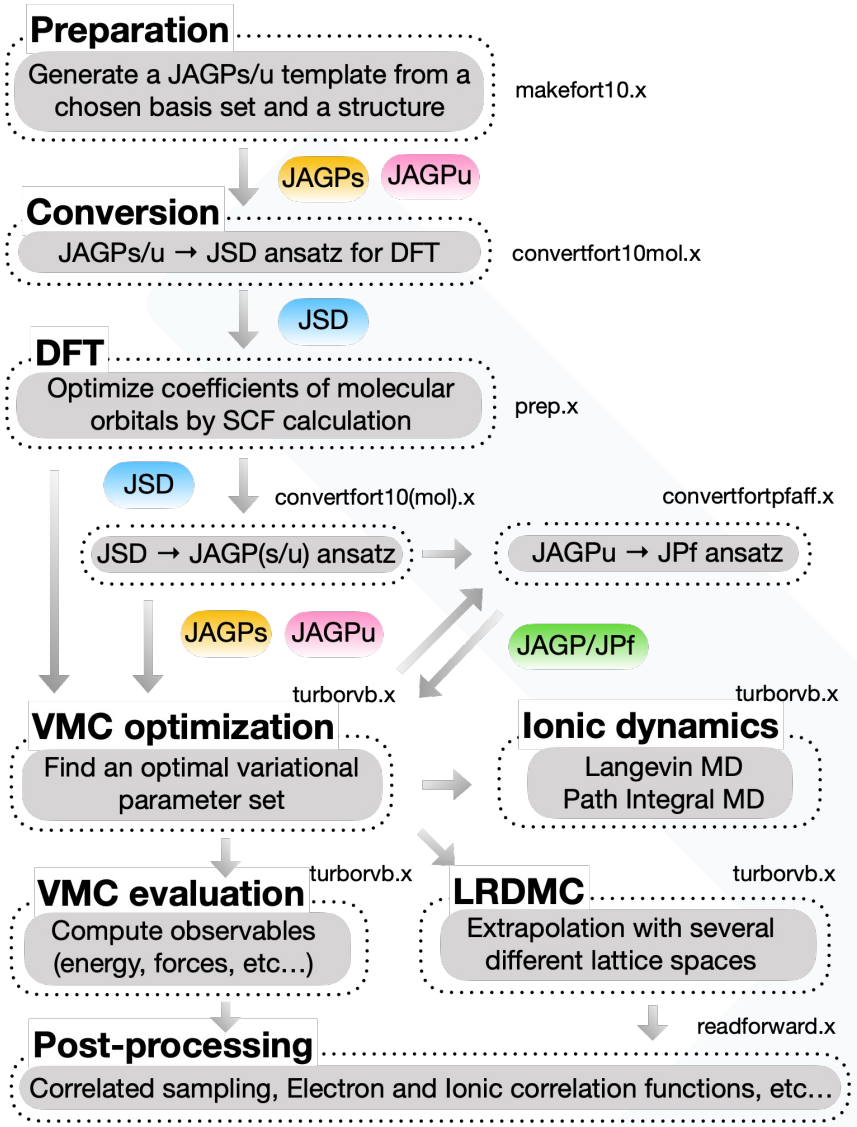
“pulay”: the ratio of the pulay term (1 is OK)

“corr”: correcting factor (p)

The average weights are stored and are set to one for all walkers after each branching.

$$E_0 \approx \frac{\sum_n \mathcal{G}_n^p e_L(x_n)}{\sum_n \mathcal{G}_n^p} \quad \mathcal{G}_n^p = \prod_{j=1}^{p} \bar{w}_{n-j},$$

A typical workflow in TurboRVB (TurboGenius)



= Workflow =

1. Prepare a structure and basis set — makefort10

2. DFT — Construct a reasonable initial WF! — prep

3. VMC-opt — Optimize the wavefunction — **turborvb**

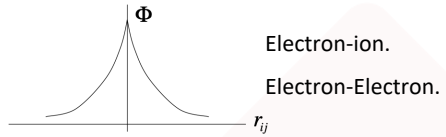
4. VMC — Do a VMC run. — turborvb

5. LRDMC — LRDMC with the optimized WF. — turborvb

The most difficult part in practice

3. VMC Optimize wavefunctions and VMC run.

A wavefunction reads $\Psi_{AS}(\vec{R}) \times \exp(J(\vec{R}))$ ⟶



Anti-symmetric part. Jastrow factor.

To satisfy the cusp.

Jastrow factor = $\exp(J(\vec{R}))$ No effect on the nodal surface!! $\Psi(\vec{R}) \equiv \Psi(\vec{r_1}, \vec{r_2}, \cdots, \vec{r_N}) = 0$

Anti-symmetric part = $\Psi_{AS}(\vec{R})$ Determines the nodal surface. Its initial guess is taken from a DFT calculation!!

1st Step JSD 2nd Step JAGPs

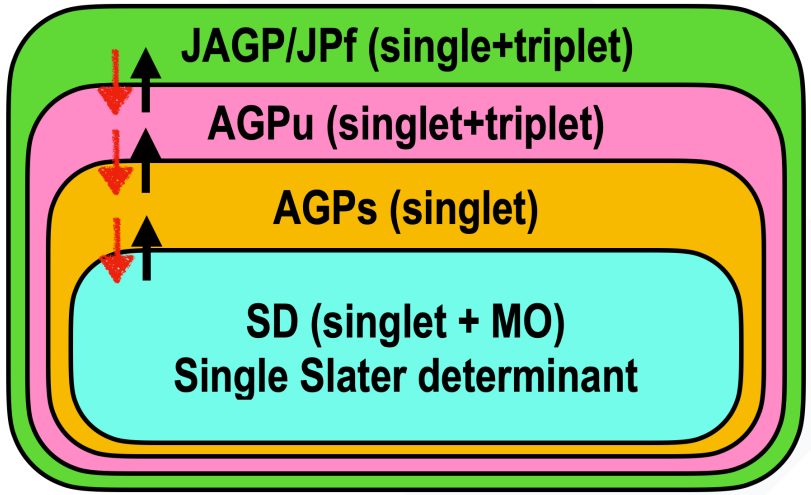
Conversion tools (convertfort10.x, convertpfaff)

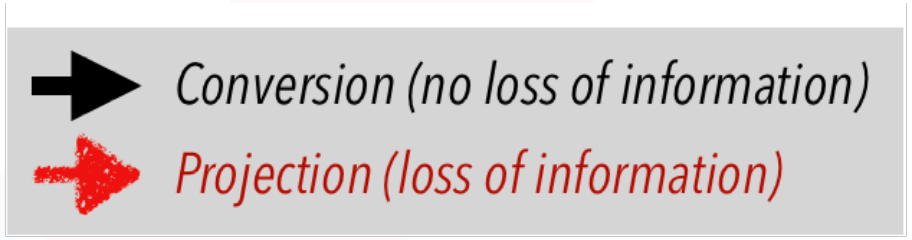
Input: convertfort.input, fort.10_in, fort.10_out → Binary: convertfort10.x → Output:fort10_new

Convergfort10.x is a tool for converting a WF type., e.g., JSD → JAGPs JAGPu

Input: fort.10_in, fort.10_out → Binary: convertfortpfaff.x → Output:fort10_new

Convertpfaff.x is a tool for projecting a WF., e.g., JAGPu → JAGP/JPf





Correlated sampling

readforward.x enables us to calculate the difference in two WF using the correlated sampling.

- The difference in energies
- The Overlap between the two WFs

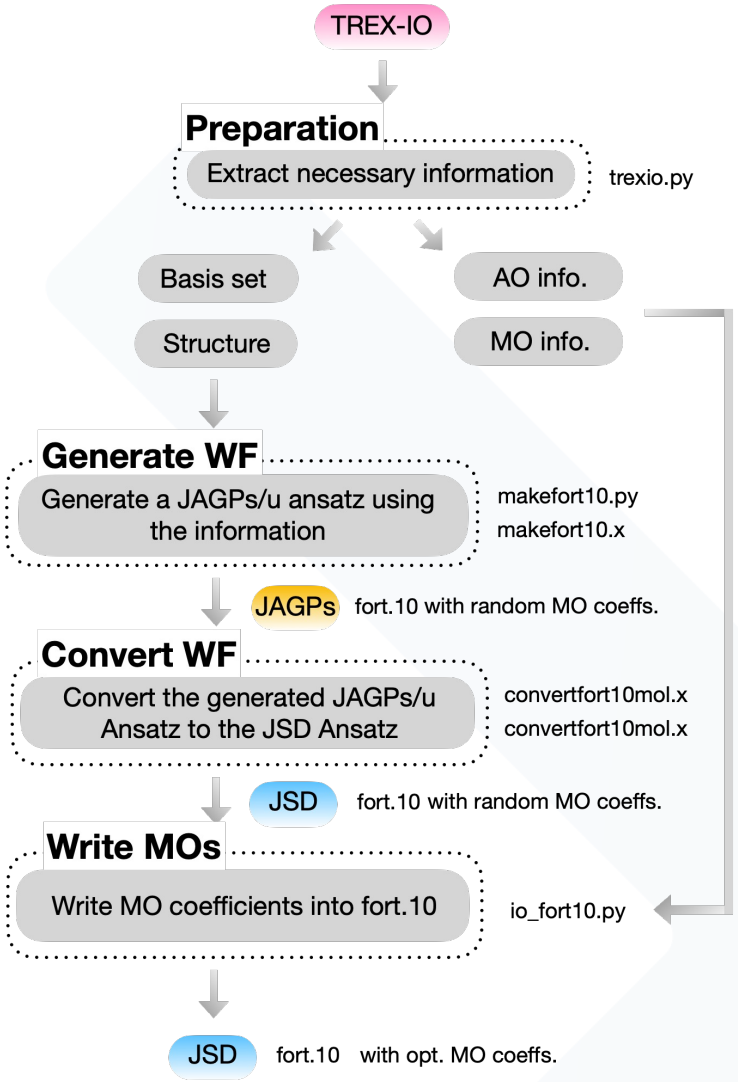
(the maximum is unity)

%cat corrsampling.dat

reference energy: E(fort.10) -0.110045875E+02 0.252368934E-01
reweighted energy: E(fort.10_corr) -0.110045875E+02 0.252368985E-01
reweighted difference: E(fort.10)-E(fort.10_corr) -0.148834687E-07 0.316227766E-07
Overlap square : (fort.10,fort.10_corr) 0.999999998E+00 0.316227766E-07

If the overlap is unity, it means that the conversion has been done without losing any information.

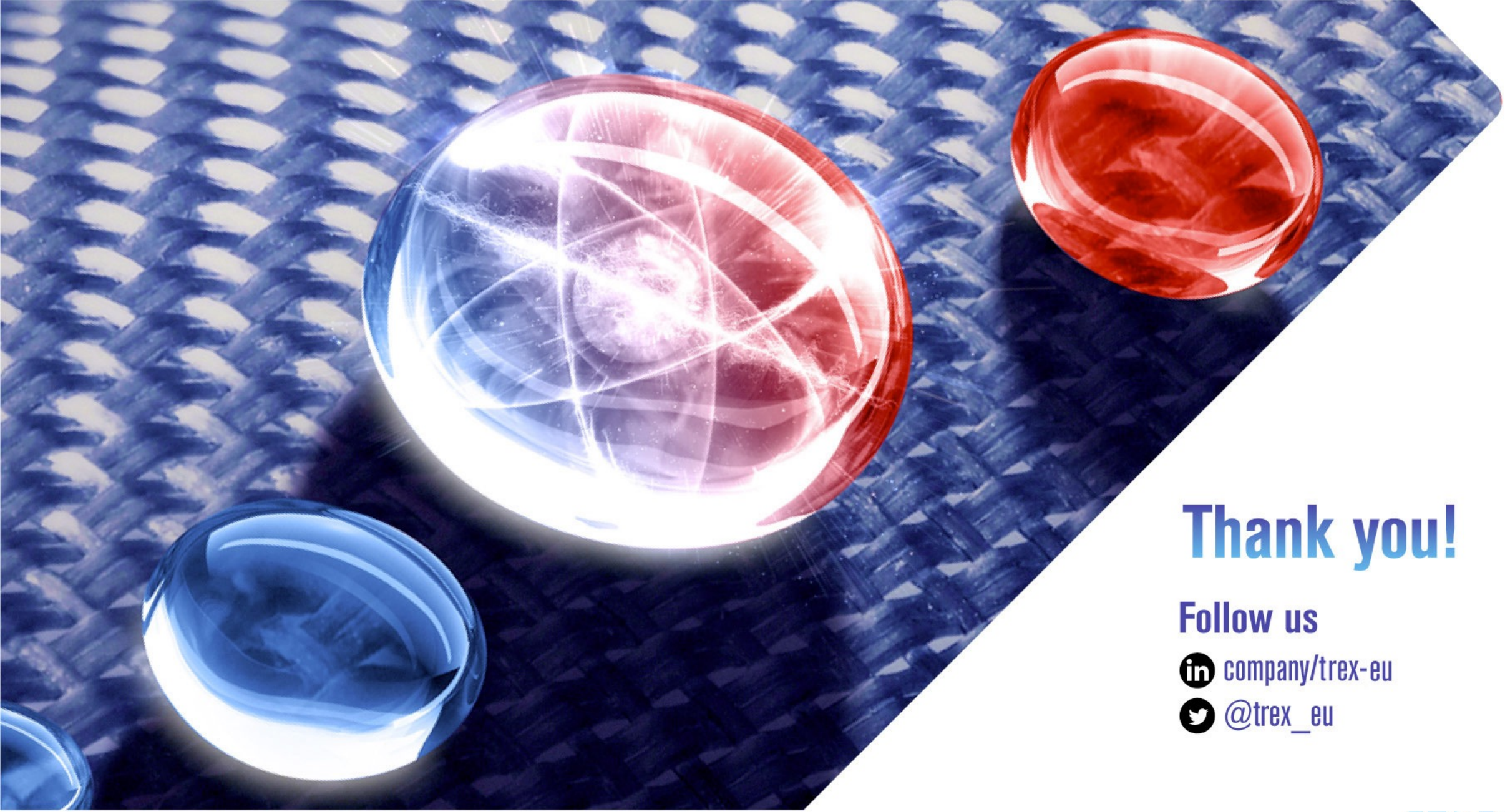
Workflow of the TREXIO-TurboRVB converter



First, the converter generates a TurboRVB WF file using only basis set and structure information stored in a TREX-IO file.

Then, the converter writes the MO information stored in a TREX-IO file into the generated WF file.

This is because of the complication of the TurboRVB WF format.



Thank you!

Follow us

company/trex-eu

@trex_eu

Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement **No. 952165.**

TREX
Targeting Real chemical accuracy at the EXascale