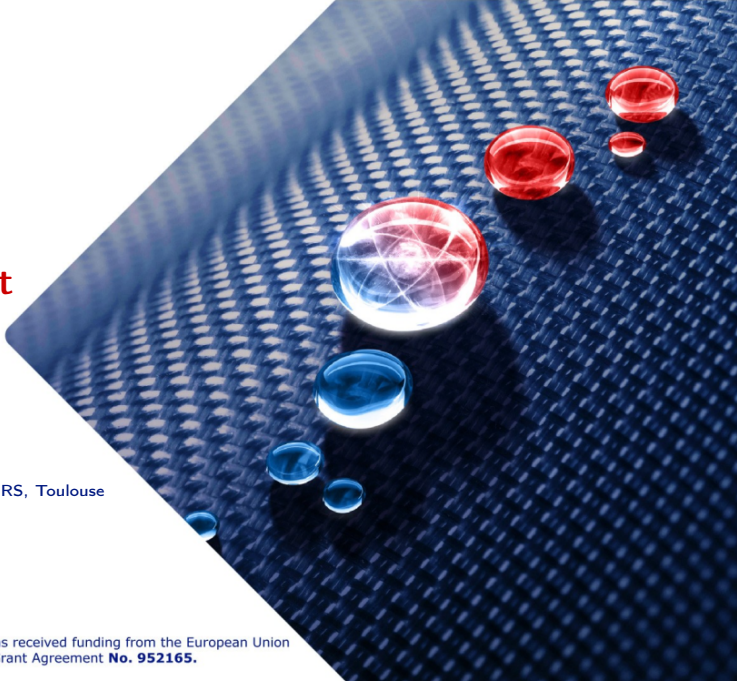
Targeting Real chemical accuracy at the EXascale

The TREXIO file format and library

Anthony Scemama

6/04/2023

Lab. Chimie et Physique Quantiques, FERMI, UPS/CNRS, Toulouse (France)



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

- M. Boggio Pasqua has made hundreds of CAS-SCF calculations with Molpro
- S. Battaglia wants to use these wavefunctions to apply his new variants of CAS-PT2 with Molcas

They asked me

"I wrote a Molden file with Molpro, but I am not sure I will be able to read it in Molcas because of the ordering of the *d* orbitals. Is going to work?"

My answer

I don't know:

1. I remember that in 2008 Molden files produced by Molcas were incorrect ($1/\sqrt{3}$), so I don't know if the molden files produced by Molpro are correct.

My answer

I don't know:

1. I remember that in 2008 Molden files produced by Molcas were incorrect ($1/\sqrt{3}$), so I don't know if the molden files produced by Molpro are correct.
2. I don't know if the molden file produced by most of the codes follow the documentation, and use the "Molden" ordering of *d* orbitals.

My answer

I don't know:

1. I remember that in 2008 Molden files produced by Molcas were incorrect ($1/\sqrt{3}$), so I don't know if the molden files produced by Molpro are correct.
2. I don't know if the molden file produced by most of the codes follow the documentation, and use the "Molden" ordering of *d* orbitals.
3. Be sure to read all the basis set parameters, because if you rely on the name given to the program, the MO coefficients might not be transferable.

My answer

I don't know:

1. I remember that in 2008 Molden files produced by Molcas were incorrect ($1/\sqrt{3}$), so I don't know if the molden files produced by Molpro are correct.
2. I don't know if the molden file produced by most of the codes follow the documentation, and use the "Molden" ordering of *d* orbitals.
3. Be sure to read all the basis set parameters, because if you rely on the name given to the program, the MO coefficients might not be transferable.

This talk

How to make more science with multiple codes with *very* little effort

- A program is a function p : input $\longrightarrow$ output

- A program is a function p : input $\longrightarrow$ output
- If the output of a program p_1 is of the same type as the input of a program p_2, we can define a new program $p_3 = p_2 \circ p_1$:

$$\begin{aligned} p_1 &: t_1 \to t_2 \\ p_2 &: t_2 \to t_3 \\ p_2 \circ p_1 &: t_1 \to t_3 \end{aligned}$$

Douglas McIlroy (1978)

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".

Douglas McIlroy (1978)

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

Douglas McIlroy (1978)

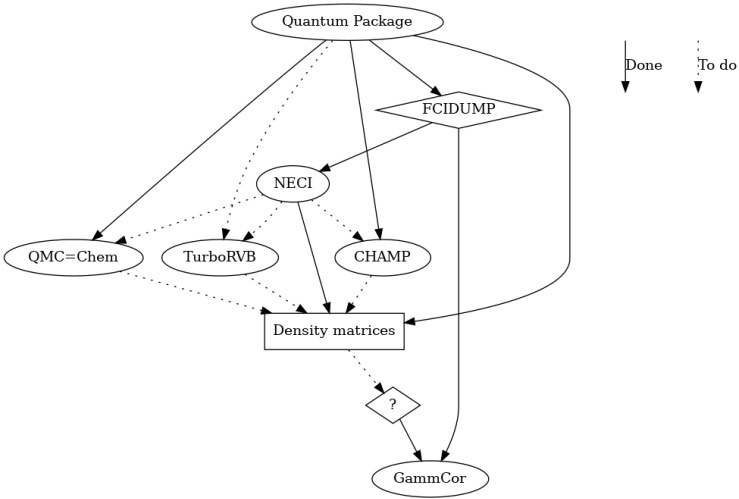
1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

Examples

- `git log -1 | head -1 | cut -d ' ' -f 2`
- 45 years later, the unix pipe is still widely used!
- Monolythic codes are contrary to these principles

- Quantum Package (AS): CIPSI
  - Input: xyz coordinates, AOs, a wave function
  - Output: MOs, CI expansion, 1e/2e Integrals, 1e/2e Density matrices
- NECI (A. Alavi): FCIQMC
  - Input: 1e/2e Integrals in MO basis
  - Output: CI expansion, 1e/2e Density matrices
- QMC=Chem (AS), TurboRVB (S. Sorella), CHAMP (C. Filippi): QMC
  - Input: A wave function
  - Output: A wave function
- GammCor (K. Pernal): SAPT + AC
  - Input: 1e/2e Integrals, 1e/2e Density matrices

Situation two years ago



Question

How can we use composition to build easily new programs?

Question

How can we use composition to build easily new programs?

Answer

Making codes have the same signature with same type for input and output:

$$code : t \longrightarrow t$$

- By modifying the code
- Or by composition: $c_2 \circ \text{code} \circ c_1 : t \longrightarrow \text{input} \longrightarrow \text{output} \longrightarrow t$

- With Unix programs, the type t is a `string`.
- The `string` type is too primitive for Ψ, so we need to define a common file format.

In Unix philosophy, *text files* are recommended:

- portability (architecture independent)
- can be read as a stream
- readable in any language
- no conversion required

If the common format is `text`, the programs can be composed with all unix tools (`grep`, `cut`, `head`, `tail`, text editors, ...)

In Unix philosophy, *text files* are recommended:

- portability (architecture independent)
- can be read as a stream
- readable in any language
- no conversion required

If the common format is `text`, the programs can be composed with all unix tools (`grep`, `cut`, `head`, `tail`, text editors, ...)

Problems with text files

- Large storage size (archiving), but can be compressed
- Expensive conversion from ASCII to binary representation
- $\Longrightarrow$ Poor I/O performance, bad for HPC

TREXIO: Domain-specific I/O Library

- Very permissive license (BSD-3-clause)
- Domain-specific: *Do one thing and do it well* $\longrightarrow$ wave functions
- Portable (C API): usable with any language
- Single front-end, multiple back-ends
  - Text back-end: multiple text files
  - HDF5 back-end: single binary file

Advantages

- Binary files if wanted (performance, small files)
- Text files if wanted (unix tools, git repositories, etc)
- HDF5 binary files are portable (endianness)
- If HDF5 is de-activated at compile time: zero dependency (pure C code)

No external knowledge is needed to compute $\Psi(r_1, ..., r_N)$:

- Ψ is just a mathematical function defined by parameters.
- We define a general form for Ψ, and we want to be able to read/write its parameters.
- All the needed numbers are stored in the file: no external database or integral computation required.

"cc-pVDZ" is not enough information:

Web Link: https://www.basissetexchange.org/basis/cc-pvdz/format/gamess_us/?version=0&elements=1&optimize_general=false
API Link: https://www.basissetexchange.org/api/basis/cc-pvdz/format/gamess_us/?version=0&elements=1&optimize_general=false

Download | Copy to Clipboard

```
!----------------------------------------------------------------------
! Basis Set Exchange
! Version v0.9
! https://www.basissetexchange.org
!----------------------------------------------------------------------
!   Basis set: cc-pVDZ
! Description: cc-pVDZ
!        Role: orbital
!     Version: 0  (Data from the Original Basis Set Exchange)
!----------------------------------------------------------------------


$DATA

HYDROGEN
S   4
1        13.0100000              0.0196850
2         1.9620000              0.1379770
3         0.4446000              0.4781480
4         0.1220000              0.5012400
S   1
1         0.1220000              1.0000000
P   1
1         0.7270000              1.0000000

$END
```

Web Link: https://www.basissetexchange.org/basis/cc-pvdz/format/gamess_us/?version=0&elements=1&optimize_general=true
API Link: https://www.basissetexchange.org/api/basis/cc-pvdz/format/gamess_us/?version=0&elements=1&optimize_general=true

Download | Copy to Clipboard

```
!----------------------------------------------------------------------
! Basis Set Exchange
! Version v0.9
! https://www.basissetexchange.org
!----------------------------------------------------------------------
!   Basis set: cc-pVDZ
! Description: cc-pVDZ
!        Role: orbital
!     Version: 0  (Data from the Original Basis Set Exchange)
!----------------------------------------------------------------------


$DATA

HYDROGEN
S   3
1        13.0100000              0.0196850
2         1.9620000              0.1379770
3         0.4446000              0.4781480
S   1
1         0.1220000              1.0000000
P   1
1         0.7270000              1.0000000

$END
```

Example: Different AO conventions

- Ordering of the AOs:
  - $d_{-2}, d_{-1}, d_0, \ldots$ or $d_0, d_{+1}, d_{-1}, \ldots$?
  - $d_{x^2}, d_{y^2}, d_{z^2}, \ldots$?

$\Longrightarrow$ The order is fixed and given in the documentation.

Example: Different AO conventions

- Ordering of the AOs:
  - $d_{-2}, d_{-1}, d_0, \ldots$ or $d_0, d_{+1}, d_{-1}, \ldots$?
  - $d_{x^2}, d_{y^2}, d_{z^2}, \ldots$?

$\Longrightarrow$ The order is fixed and given in the documentation.

- Are the AOs assumed normalized?
- Should d_{xy} have the same normalization coefficient as d_{z^2}?

$$\iiint (x\,y\,G(x,y,z))^2\,dx\,dy\,dz \neq \iiint \left(z^2\,G(x,y,z)\right)^2\,dx\,dy\,dz$$

$\Longrightarrow$ All the normalization coefficients are stored in the file.

Data stored with TREXIO

- Hierarchical data layout:
  - Groups

`Metadata`	`Electron`	`Nucleus`	`ECP`	`Basis`	`QMC`
`AO`	`MO`	`Determinant`	`State`	`Cell`	`PBC`
`AO_1e_int`	`MO_1e_int`	`RDM`	`AO_2e_int`	`MO_2e_int`	`grid`
`CSF`	`Amplitude`	`Jastrow`			

- Hierarchical data layout:
  - Groups

Metadata	Electron	Nucleus	ECP	Basis	QMC
AO	MO	Determinant	State	Cell	PBC
AO_1e_int	MO_1e_int	RDM	AO_2e_int	MO_2e_int	grid
CSF	Amplitude	Jastrow			

- Inside each group, multiple values

Example: `Nucleus` group

Variable	Type	Dimensions	Description
num	dim		Number of nuclei
charge	float	(nucleus.num)	Charges of the nuclei
coord	float	(3,nucleus.num)	Coordinates of the atoms
label	str	(nucleus.num)	Atom labels
point_group	str		Symmetry point group
repulsion	float		Nuclear repulsion energy

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`
- Safe API: Takes max dimension as arguments for memory safety

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`
- Safe API: Takes max dimension as arguments for memory safety
- Returns an error code for error handling

```c
#define MAX_SIZE 10
double charge_array[MAX_SIZE];
trexio_exit_code rc;

rc = trexio_read_nucleus_charge(trexio_file, charge_array);
rc = trexio_read_safe_nucleus_charge_64(trexio_file, charge_array, MAX_SIZE);
```

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`
- Safe API: Takes max dimension as arguments for memory safety
- Returns an error code for error handling

```c
#define MAX_SIZE 10
double charge_array[MAX_SIZE];
trexio_exit_code rc;

rc = trexio_read_nucleus_charge(trexio_file, charge_array);
rc = trexio_read_safe_nucleus_charge_64(trexio_file, charge_array, MAX_SIZE);
```

- Usable in C, C++, Fortran, Python, Julia, OCaml

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`
- Safe API: Takes max dimension as arguments for memory safety
- Returns an error code for error handling

```c
#define MAX_SIZE 10
double charge_array[MAX_SIZE];
trexio_exit_code rc;

rc = trexio_read_nucleus_charge(trexio_file, charge_array);
rc = trexio_read_safe_nucleus_charge_64(trexio_file, charge_array, MAX_SIZE);
```

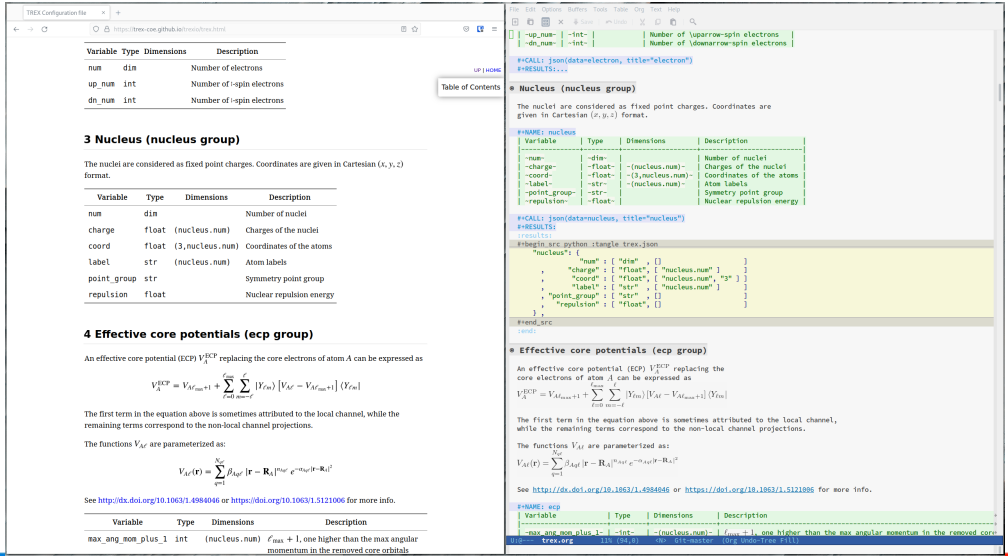
- Usable in C, C++, Fortran, Python, Julia, OCaml
- Auto-generated

- Computable function names (32- or 64-bit variants):
  `trexio_<read|write|has>[_safe]_<group>_<data>[_32|_64]`
  `(trexio_file, data[, size])`
- Safe API: Takes max dimension as arguments for memory safety
- Returns an error code for error handling

```c
#define MAX_SIZE 10
double charge_array[MAX_SIZE];
trexio_exit_code rc;

rc = trexio_read_nucleus_charge(trexio_file, charge_array);
rc = trexio_read_safe_nucleus_charge_64(trexio_file, charge_array, MAX_SIZE);
```

- Usable in C, C++, Fortran, Python, Julia, OCaml
- Auto-generated
- Literate programming with Org-mode: easy to extend



TREX Configuration file

https://trex-coe.github.io/trexio/trex.html

Variable	Type	Dimensions	Description
num	dim		Number of electrons
up_num	int		Number of ↑-spin electrons
dn_num	int		Number of ↓-spin electrons

UP | HOME

Table of Contents

3 Nucleus (nucleus group)

The nuclei are considered as fixed point charges. Coordinates are given in Cartesian (x, y, z) format.

Variable	Type	Dimensions	Description
num	dim		Number of nuclei
charge	float	(nucleus.num)	Charges of the nuclei
coord	float	(3,nucleus.num)	Coordinates of the atoms
label	str	(nucleus.num)	Atom labels
point_group	str		Symmetry point group
repulsion	float		Nuclear repulsion energy

4 Effective core potentials (ecp group)

An effective core potential (ECP) V_A^{ECP} replacing the core electrons of atom A can be expressed as

$$V_A^{\mathrm{ECP}} = V_{A\ell_{\max}+1} + \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} |Y_{\ell m}\rangle \left[V_{A\ell} - V_{A\ell_{\max}+1}\right] \langle Y_{\ell m}|$$

The first term in the equation above is sometimes attributed to the local channel, while the remaining terms correspond to the non-local channel projections.

The functions $V_{A\ell}$ are parameterized as:

$$V_{A\ell}(\mathbf{r}) = \sum_{q=1}^{N_{q\ell}} \beta_{Aq\ell}\, |\mathbf{r} - \mathbf{R}_A|^{n_{Aq\ell}}\, e^{-\alpha_{Aq\ell}|\mathbf{r}-\mathbf{R}_A|^2}$$

See http://dx.doi.org/10.1063/1.4984046 or https://doi.org/10.1063/1.5121006 for more info.

Variable	Type	Dimensions	Description
max_ang_mom_plus_1	int	(nucleus.num)	$\ell_{\max} + 1$, one higher than the max angular momentum in the removed core orbitals

Example (Fortran)

```fortran
use trexio                                ! ISO-C-binding module to be included with your code
double precision          :: charge(3)
integer                   :: n
integer(trexio_t)         :: f            ! File handle
integer(trexio_exit_code) :: rc           ! Return code

!  Write
f = trexio_open('water.h5', 'w', TREXIO_HDF5, rc)    ! rc -> TREXIO_SUCCESS
charge = (/ 8., 1., 1. /)
rc = trexio_write_nucleus_num(f, 3)
rc = trexio_write_nucleus_charge(f, charge)
rc = trexio_close(f)

!  Read
f  = trexio_open('water.h5', 'r', TREXIO_HDF5, rc)    ! rc -> TREXIO_SUCCESS
rc = trexio_read_nucleus_num(f, n)                    ! n = 3
charge(:) = 0.
rc = trexio_read_nucleus_charge(f, charge)            ! charge = (/ 1., 2., 3./)
rc = trexio_close(f)
```

Example (Python)

```python
import trexio

# Write
with trexio.File('water.h5', mode='w', back_end=trexio.TREXIO_HDF5) as f:
    charge = [ 8., 1., 1. ]
    trexio.write_nucleus_num(f,3)
    trexio.write_nucleus_charge(f,charge)

# Read
with trexio.File('water.h5', mode='r', back_end=trexio.TREXIO_HDF5) as f:
    n = trexio.read_nucleus_num(f)              # n = 3
    charge = trexio.read_nucleus_charge(f)      # charge = [1., 2., 3.]
```

```c
#include <trexio.h>
#include <assert.h>

trexio_exit_code rc;

/*  Write */
trexio_t* f = trexio_open("water.h5", 'w', TREXIO_HDF5, &rc);
double charge[] = { 8., 1., 1. };
rc = trexio_write_nucleus_num(f, 3);          assert (rc == TREXIO_SUCCESS);
rc = trexio_write_nucleus_charge(f, charge);  assert (rc == TREXIO_SUCCESS);
rc = trexio_close(f);

/* Read */
charge = { 0., 0., 0. };
f  = trexio_open("water.h5", 'r', TREXIO_HDF5, &rc);
rc = trexio_read_nucleus_num(f, n);           assert (rc == TREXIO_SUCCESS);
rc = trexio_read_nucleus_charge(f, charge);   assert (rc == TREXIO_SUCCESS);
rc = trexio_close(f);
```

`https://github.com/TREX-CoE/trexio_tools`

Repository of tools to manipulate TREXIO files:

- Spherical $\longrightarrow$ Cartesian AO conversion
- Numerical computation of AO overlap matrix compared with the one stored in the file: $\Longrightarrow$ debugging
- Compare numerical computation of MO overlap matrix with identity: $\Longrightarrow$ debugging when no integrals are available
- Converters for GAMESS, Gaussian, PySCF, FCIDUMP, Molden
- ...
- Contributions welcome! :-)

- Checking PySCF $\longrightarrow$ TurboRVB interface:
  - PySCF $\longrightarrow$ TREXIO $\longrightarrow$ QP: check energy
  - TREXIO $\longrightarrow$ TurboRVB: OK

- Checking PySCF $\longrightarrow$ TurboRVB interface:
  - PySCF $\longrightarrow$ TREXIO $\longrightarrow$ QP: check energy
  - TREXIO $\longrightarrow$ TurboRVB: OK
- QP $\longrightarrow$ TREXIO $\longrightarrow$ NECI: Compare CIPSI and FCIQMC estimates of FCI energy

- Checking PySCF $\longrightarrow$ TurboRVB interface:
  - PySCF $\longrightarrow$ TREXIO $\longrightarrow$ QP: check energy
  - TREXIO $\longrightarrow$ TurboRVB: OK
- QP $\longrightarrow$ TREXIO $\longrightarrow$ NECI: Compare CIPSI and FCIQMC estimates of FCI energy
- QP $\longrightarrow$ TREXIO $\longrightarrow$ TurboRVB

- Checking PySCF $\longrightarrow$ TurboRVB interface:
  - PySCF $\longrightarrow$ TREXIO $\longrightarrow$ QP: check energy
  - TREXIO $\longrightarrow$ TurboRVB: OK
- QP $\longrightarrow$ TREXIO $\longrightarrow$ NECI: Compare CIPSI and FCIQMC estimates of FCI energy
- QP $\longrightarrow$ TREXIO $\longrightarrow$ TurboRVB
- QP $\longrightarrow$ TREXIO $\longrightarrow$ GammCor: SAPT with CIPSI density matrices

First applications of TREXIO

- Checking PySCF $\longrightarrow$ TurboRVB interface:
  - PySCF $\longrightarrow$ TREXIO $\longrightarrow$ QP: check energy
  - TREXIO $\longrightarrow$ TurboRVB: OK
- QP $\longrightarrow$ TREXIO $\longrightarrow$ NECI: Compare CIPSI and FCIQMC estimates of FCI energy
- QP $\longrightarrow$ TREXIO $\longrightarrow$ TurboRVB
- QP $\longrightarrow$ TREXIO $\longrightarrow$ GammCor: SAPT with CIPSI density matrices
- Introduction of TREXIO in FHI-AIMS: CIPSI and QMC with numerical atomic basis sets
- ...

Monday (evening) in my office

1. Molden $\longrightarrow$ TREXIO $\longrightarrow$ Quantum Package
2. Compute the energy
3. Confirm that Molden file is OK
4. "You should now modify Molcas to Read/Write TREXIO files"
5. "You should also advise Molpro developers to Read/Write TREXIO files"

Summary

What is TREXIO?

TREXIO is a ~~binary~~ file format designed for storing and exchanging data in quantum chemistry. It was developed to address the need for a standardized, efficient, and interoperable file format for quantum chemistry data, similar to how the JPEG image format is used for storing and sharing images. The name "TREXIO" stands for ~~"Transferable Expanded Computational Chemistry Object,~~" and it was developed by a group of researchers and software developers in the quantum chemistry community.

TREXIO is intended to be a universal format that can be used with any quantum chemistry software package, making it easier to share data and collaborate across different research groups and institutions. The format supports a wide range of data types, including molecular geometries, electronic structures, and ~~thermodynamic properties~~, and it can be used for both small and large-scale quantum chemistry calculations.

TREXIO is still a relatively new file format, and its adoption is still growing in the quantum chemistry community. However, it has the potential to become an important standard for data exchange and collaboration in this field.

Summary

Should I use TREXIO?

Whether or not you should use TREXIO depends on your specific needs and circumstances. Here are some factors to consider when deciding whether to use TREXIO:

1. Compatibility: If you frequently work with different quantum chemistry software packages, using a standardized file format like TREXIO can make it easier to exchange data and collaborate with colleagues.
2. Efficiency: TREXIO is designed to be a highly efficient file format, which can be especially useful for storing and exchanging large amounts of data. If you frequently work with large quantum chemistry datasets, TREXIO may be a good choice for you.
3. Availability: While TREXIO is a growing standard in the quantum chemistry community, it may not be supported by all software packages or tools. Before deciding to use TREXIO, make sure that the software you use supports the format.
4. Complexity: While TREXIO is designed to be easy to use and understand, it may still require some additional effort to learn and incorporate into your workflow. If you are already familiar with another file format and it meets your needs, there may be no need to switch to TREXIO.

Overall, TREXIO can be a useful file format for quantum chemistry data exchange and collaboration, especially if you frequently work with different software packages or large datasets. However, you should weigh the pros and cons of using TREXIO against your specific needs and circumstances before making a decision.

Quotation

"TREXIO will one day become the JPEG of quantum chemistry" (Anonymous)

- TREXIO documentation: `https://trex-coe.github.io/trexio/`
- TREXIO source code: `https://github.com/trex-coe/trexio/`
- ArXiv preprint: `https://arxiv.org/pdf/2302.14793.pdf` to appear in JCP