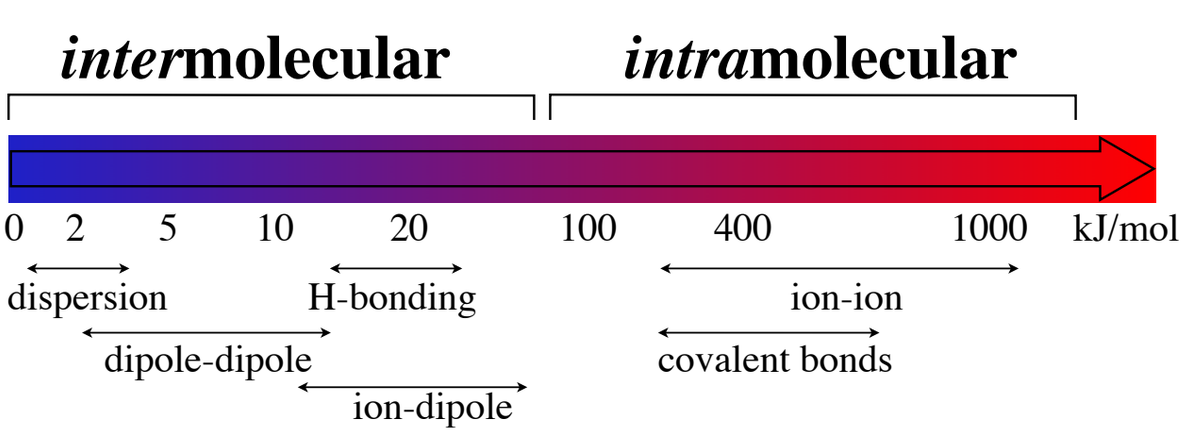
Libmbd: A general-purpose package for scalable many-body dispersion calculations

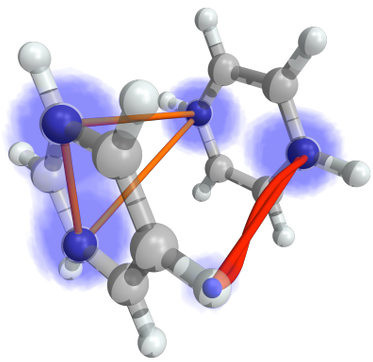
Jan Hermann

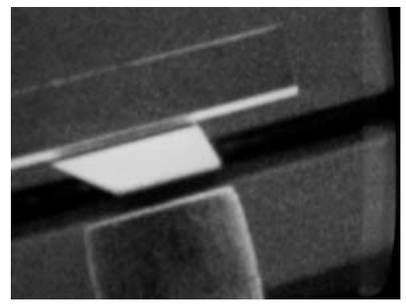
FU Berlin, Department of Mathematics
https://jan.hermann.name

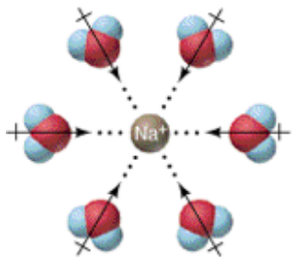
Electronic Structure Software Development, CECAM HQ, Lausanne, 10 Oct 2022

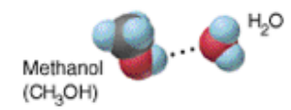
Van der Waals (dispersion) interactions







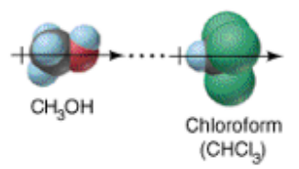


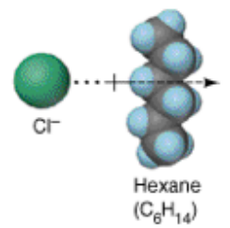


H-bond

- Chemistry breaks and forms *intra*molecular bonds
- *Inter*molecular "bonds" break and form continually all the time
- Liquids, molecular crystals, nanostructured materials, surface science, soft matter
- Van der Waals forces—always attractive, weak on atomic scale, importance grows with scale

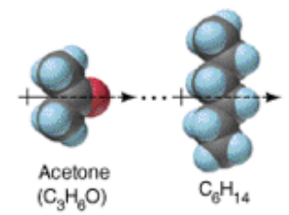
ion–dipole



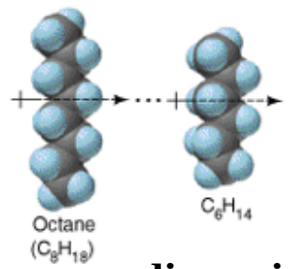


dipole–dipole

ion–induced dipole



dipole–induced dipole

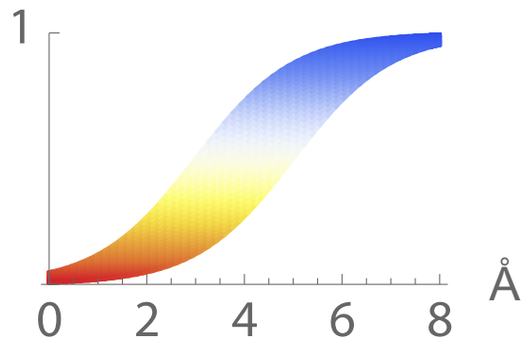


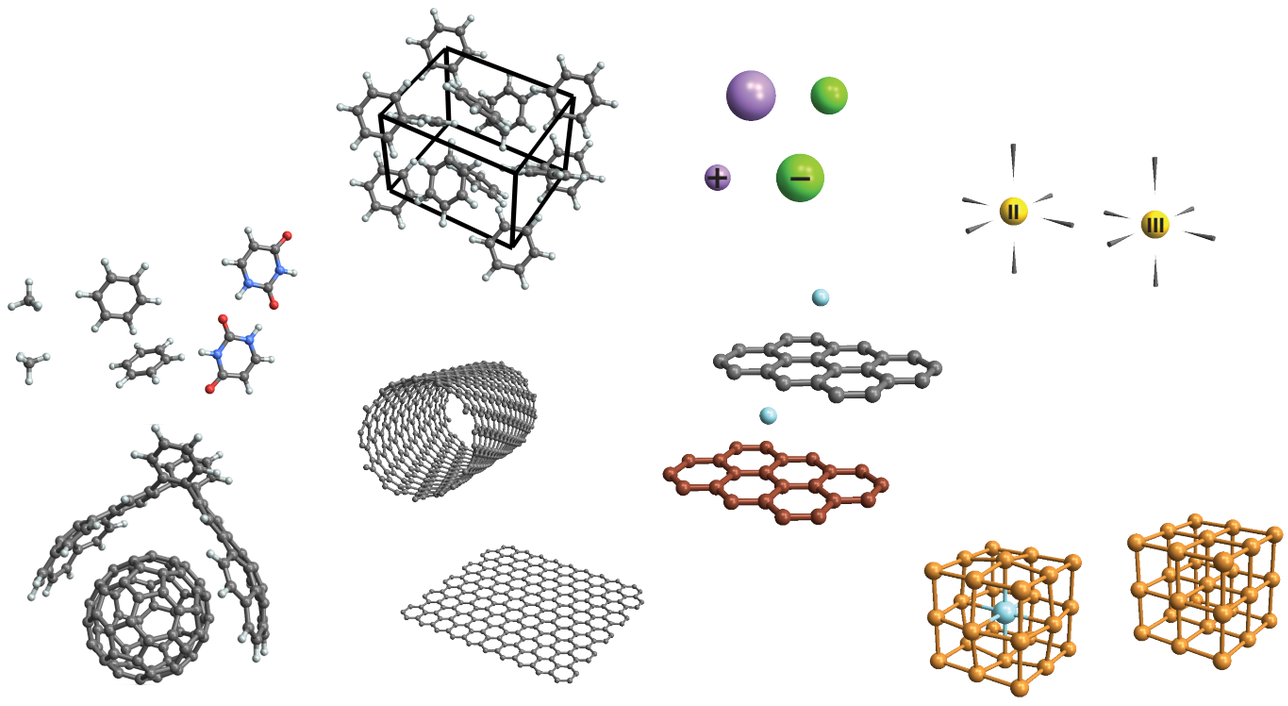
dispersion

Modeling van der Waals interactions with DFT

- Van der Waals interactions—long-range part of electron correlation energy
- No electronic-structure method simultaneously general, accurate, and efficient for vdW interactions
- All semilocal/hybrid functionals in DFT are short-ranged (DFTB, ML)

$$v(R) = (1 - f(R))v(R) + f(R)v(R)$$



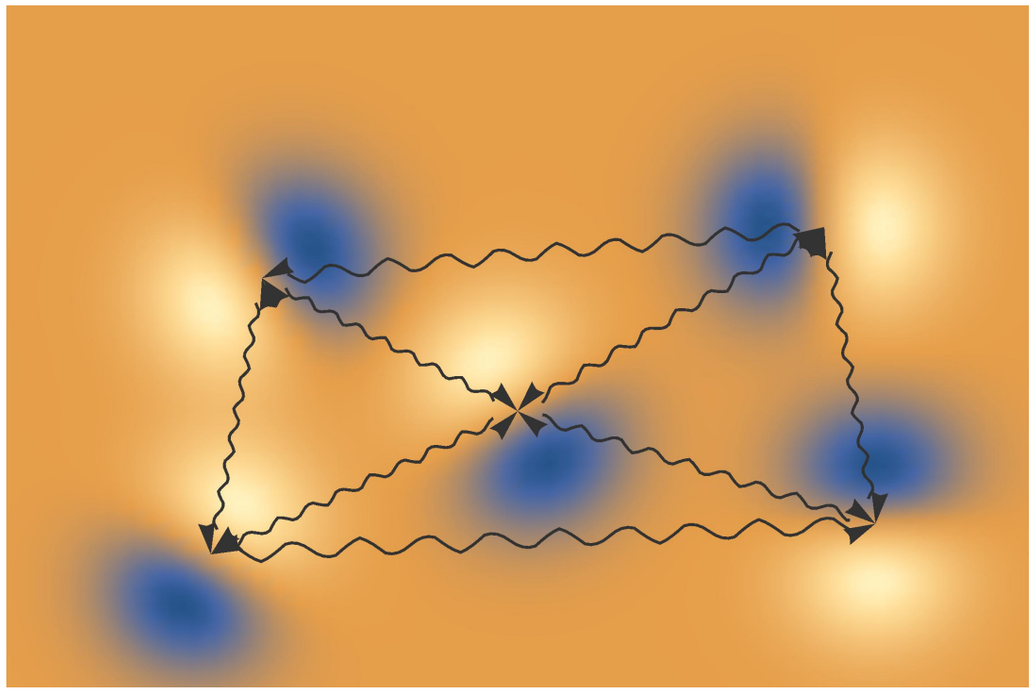


SCAN M06 XDM
D3 LDA TS
vdW-DF
B3LYP
VV10 PBE MBD
PBE0

Many-body dispersion (MBD)

$$\hat{H}^{\mathrm{MBD}} = \sum_i \frac{1}{2}\nabla_i^2 + \sum_i \frac{1}{2}\omega_i^2 \hat{\mathbf{r}}_i^{\,2} + \frac{1}{2}\sum_{i\neq j} \omega_i \omega_j \sqrt{\alpha_{0,i}\alpha_{0,j}}\, \hat{\mathbf{r}}_i \cdot \mathbf{T}_{ij}^{\mathrm{lr}} \hat{\mathbf{r}}_j$$

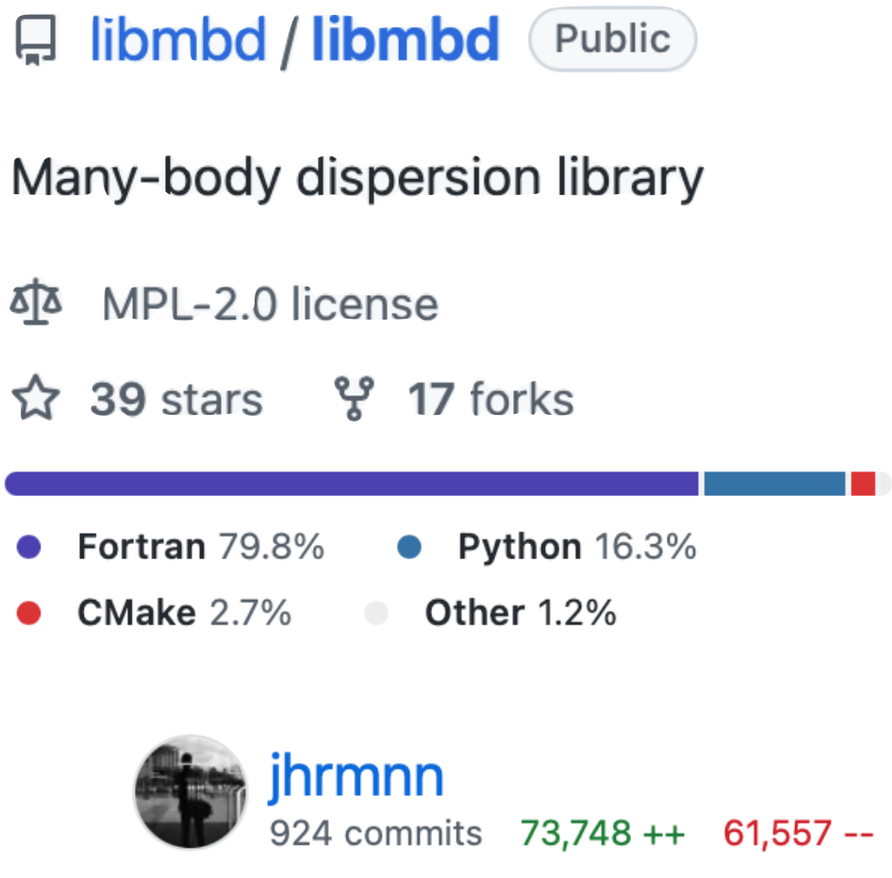
- Coarse-grains electronic structure to oscillators for efficiency, full many-body treatment for accuracy
- Integrated with DFT, DFTB, ML force fields
- Continuously improved and extended

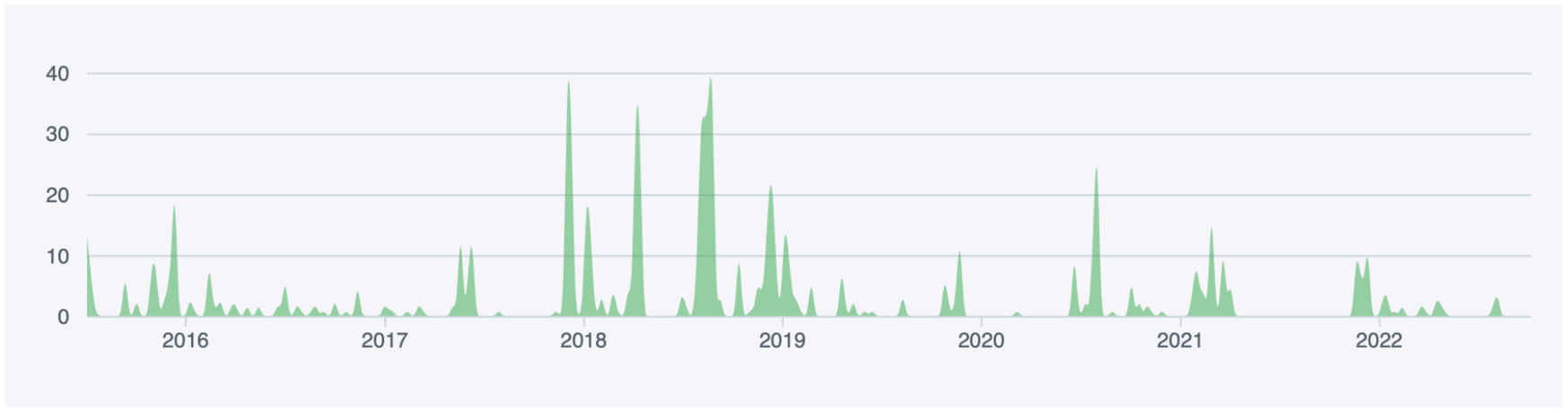


Tkatchenko et al., *Phys. Rev. Lett.* **108**, 236402 (2012)

Libmbd

- Started as a reimplementation of MBD in FHI-aims to ease development
- Two requirements:
  1. general modular framework for quick method development
  2. fast enough for production calculations
- Embedded in FHI-aims, DFTB+, Quantum Espresso, Q-Chem





Many-body dispersion "API"

$$\hat{H}^{\mathrm{MBD}} = \sum_i \frac{1}{2}\nabla_i^2 + \sum_i \frac{1}{2}\omega_i^2 \hat{\mathbf{r}}_i^{\,2} + \frac{1}{2}\sum_{i\neq j} \omega_i \omega_j \sqrt{\alpha_{0,i}\alpha_{0,j}}\, \hat{\mathbf{r}}_i \cdot \mathbf{T}_{ij}^{\mathrm{lr}} \hat{\mathbf{r}}_j$$

- Molecular geometry
- ω_i, $\alpha_{0,i}$: oscillator response properties
- $\mathbf{T}_{ij}^{\mathrm{lr}}$: long-range dipole interaction
- Both functional (different MBD *methods*) and numerical parametrization

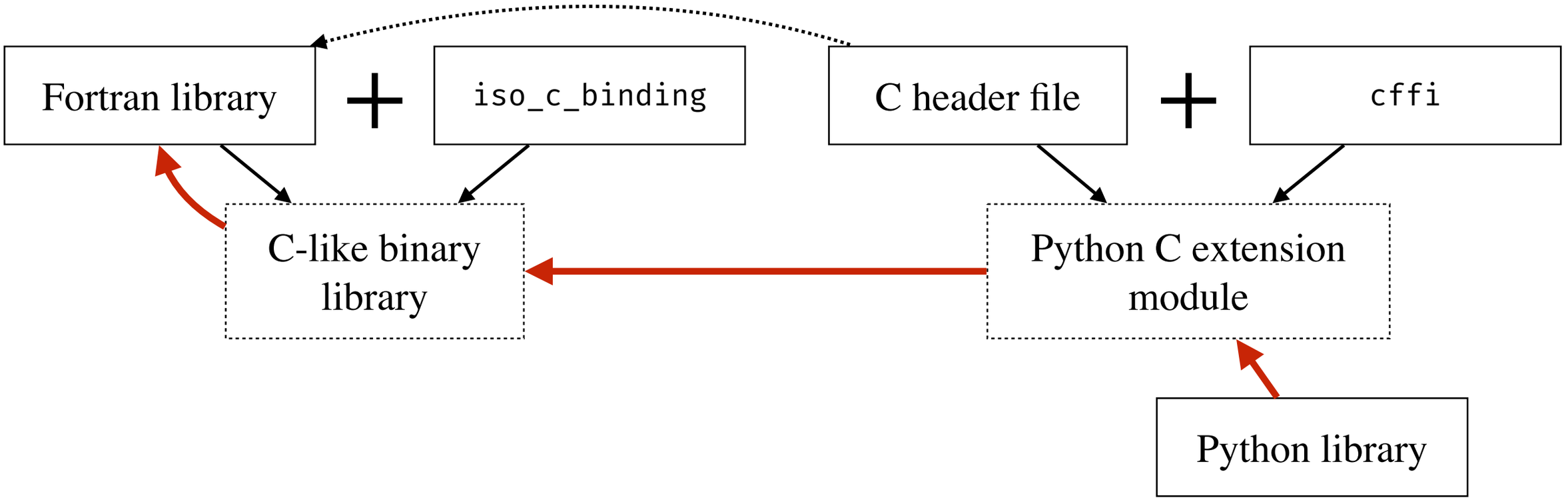
```
use mbd, only: mbd_input_t, mbd_calc_t

type(mbd_input_t) :: inp
type(mbd_calc_t) :: calc
real(8) :: energy, gradients(3, 2)
integer :: code
character(200) :: origin, msg

inp%atom_types = ['Ar', 'Ar']
inp%coords = reshape([0d0, 0d0, 0d0, 0d0, 0d0, 7.5d0], [3, 2])
inp%method = 'mbd-rsscs'
inp%xc = 'pbe'
call calc%init(inp)
call calc%get_exception(code, origin, msg)
if (code > 0) then
    print *, msg
    stop 1
end if
call calc%update_vdw_params_from_ratios([0.98d0, 0.98d0])
call calc%evaluate_vdw_method(energy)
call calc%get_gradients(gradients)
call calc%destroy()
```

Libmbd and Pymbd

- Two modes:
  1. Fixed method, fast routine execution on variable systems—Fortran
  2. Fixed system, flexible experimental execution with variable methods—Python



Compiling and installing Python/Fortran code

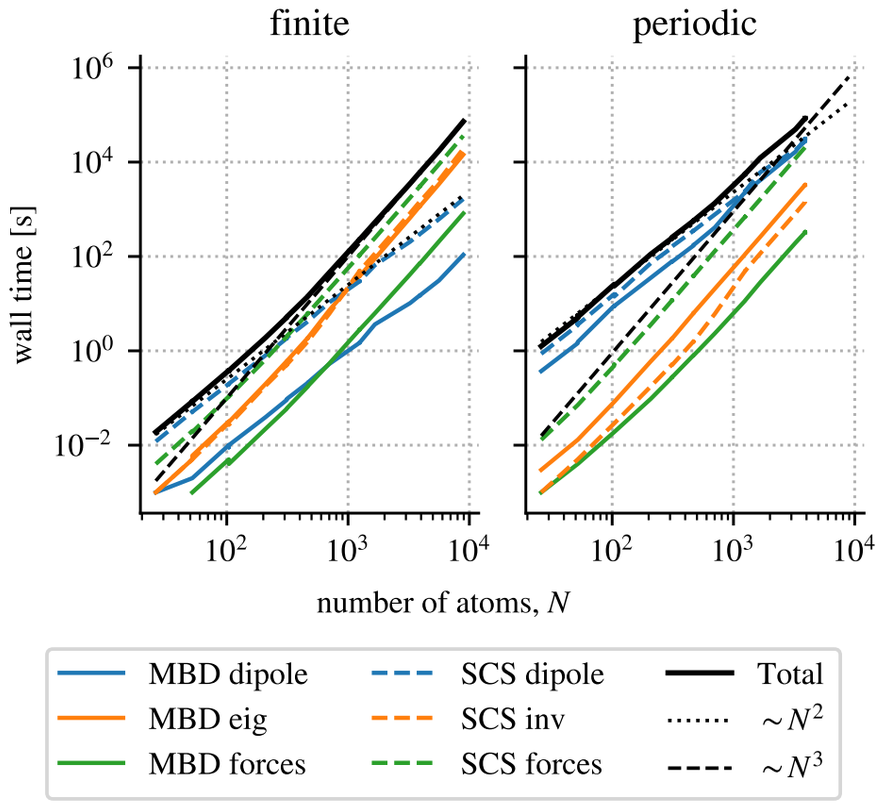
- Cmake for Fortran
- Setuptools/CFFI for Python
- Distribution with Conda-forge and PyPI
- Also available in ESL Bundle

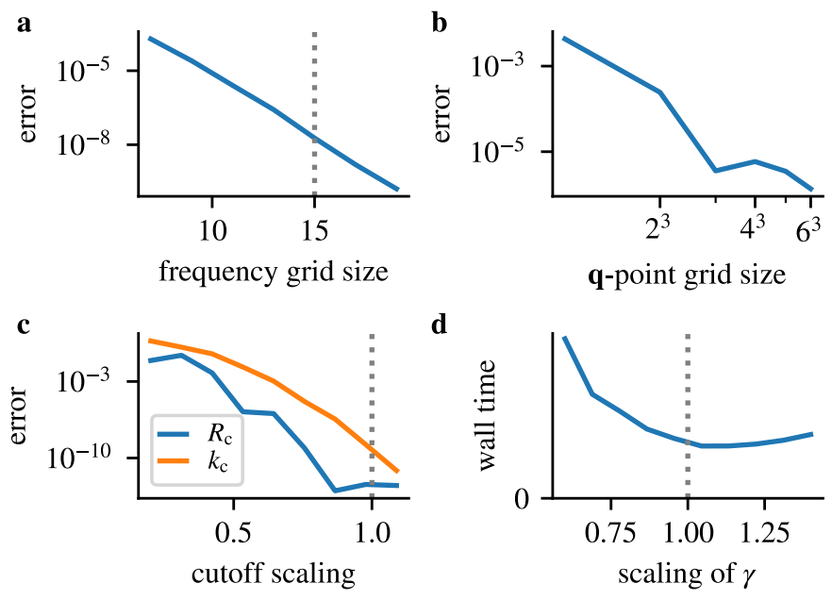
```
conda create -p ./env -c conda-forge python cmake gfortran_linux-64 pytest openblas numpy scipy
conda activate -p ./env
cmake -B build --install-prefix ./env
cmake --build ./build
ctest --test-dir ./build --output-on-failure
cmake --install ./build
pip install -e .
pytest -v
```

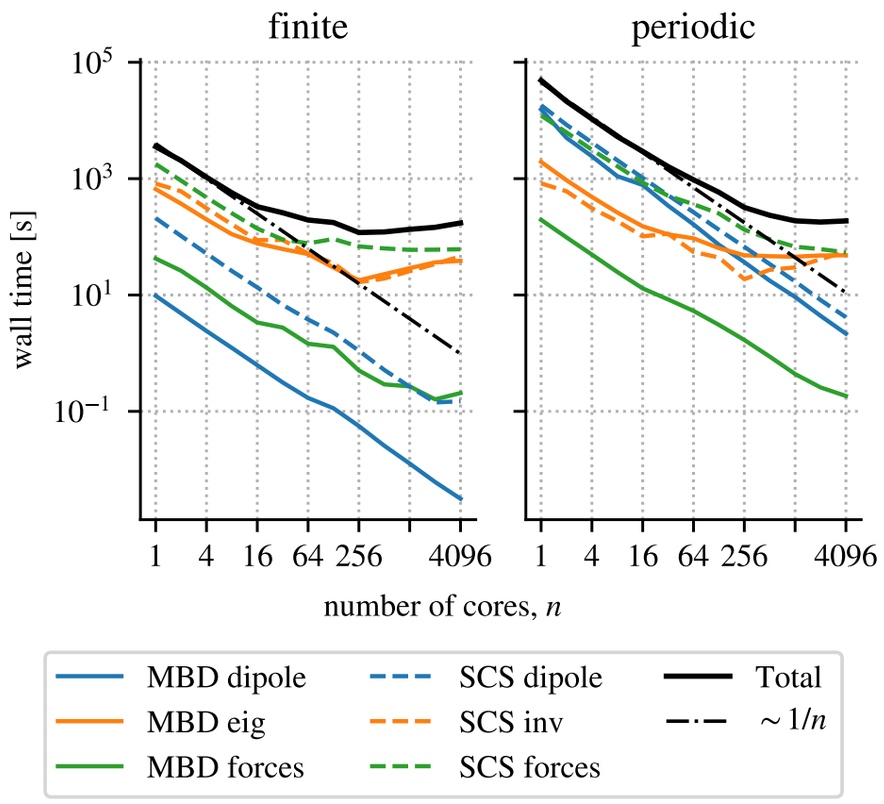
```
conda install -c conda-forge libmbd
pip install pymbd
```

Libmbd features

- MPI/Scalapack/ELSI parallelization
- Finite and periodic systems
- Analytical gradients
- Converged default parameters
- MBD properties beyond energies and forces

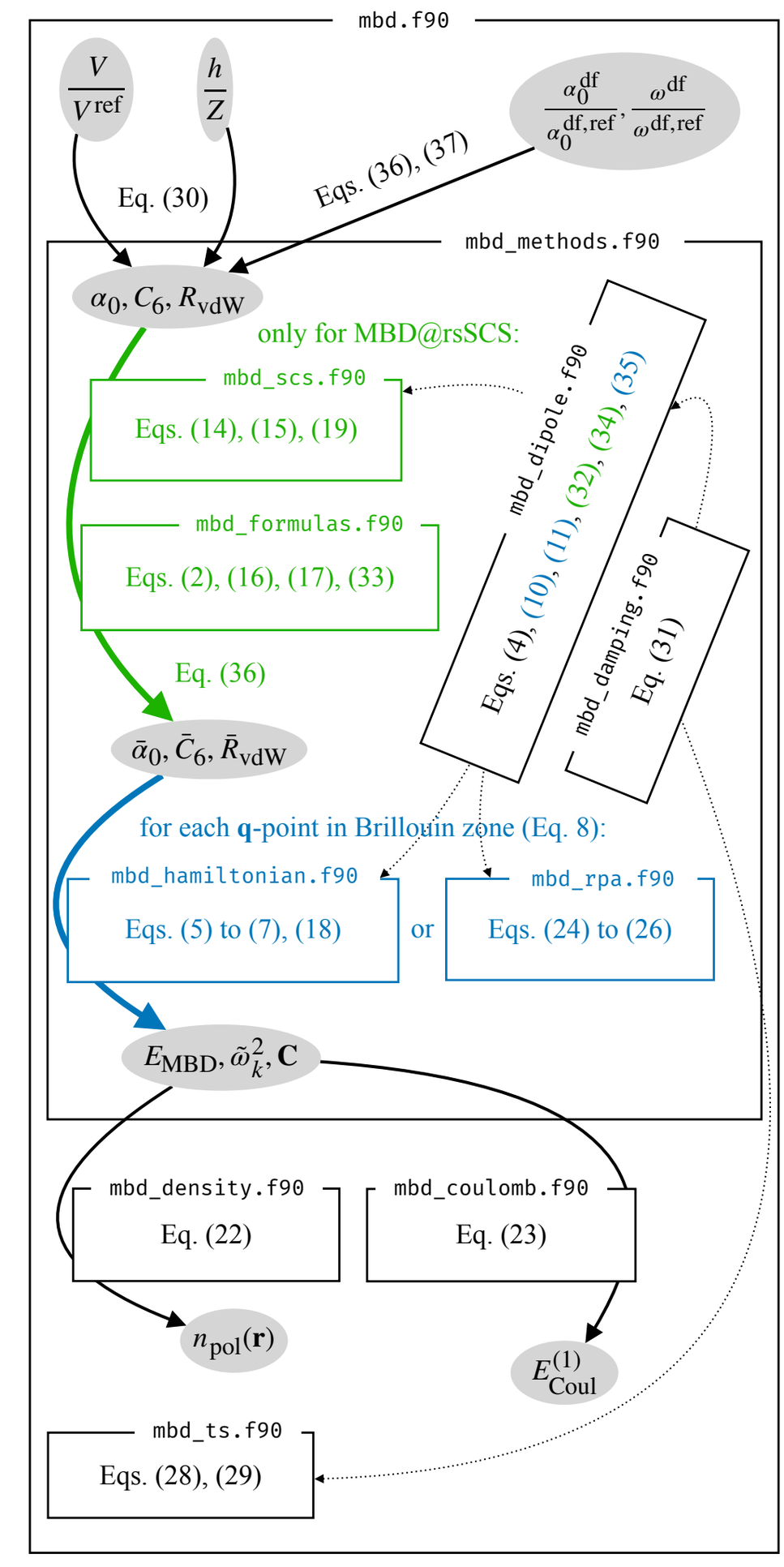






Code structure

- Basic principle—as close correspondence between physics/ math and code as possible
- Each “physics function” can be defined through equations
- Separating physics code and implementation “details”



Documentation

- Library — targeting developers
- All physics code documented locally with equations
- Automatic generation with FORD

```f90
function omega_qho(C6, alpha, domega, grad) result(omega)
    !! $$
    !! \omega=\frac{4C_6}{3\alpha_{0}^2},\qquad
    !! \partial\omega=\omega\left(
    !! \frac{\partial C_6}{C_6}-\frac{2\partial\alpha_0}{\alpha_0}
    !! \right)
    !! $$
    real(dp), intent(in) :: C6(:)
    real(dp), intent(in) :: alpha(:)
    type(grad_t), intent(out), optional :: domega
    type(grad_request_t), intent(in), optional :: grad
    real(dp) :: omega(size(C6))

    omega = 4d0 / 3 * C6 / alpha**2
    if (.not. present(grad)) return
    if (grad%dC6) domega%dC6 = omega / C6
    if (grad%dalpha) domega%dalpha = -2 * omega / alpha
end function
```

public function omega_qho(C6, alpha, domega, grad) result(omega)

$$\omega = \frac{4C_6}{3\alpha_0^2}, \qquad \partial\omega = \omega\left(\frac{\partial C_6}{C_6} - \frac{2\partial\alpha_0}{\alpha_0}\right)$$

Forward gradient accumulation

- Each physics function returns an output value and (when requested) its gradient w.r.t. function inputs
- Chain-rule application separate from output value evaluation

```
function omega_qho(C6, alpha, domega, grad) result(omega)
    real(dp), intent(in) :: C6(:)
    real(dp), intent(in) :: alpha(:)
    type(grad_t), intent(out), optional :: domega
    type(grad_request_t), intent(in), optional :: grad
    real(dp) :: omega(size(C6))

    omega = 4d0 / 3 * C6 / alpha**2
    if (.not. present(grad)) return
    if (grad%dC6) domega%dC6 = omega / C6
    if (grad%dalpha) domega%dalpha = -2 * omega / alpha
end function
```

Matrix operations and parallelization

- `matrix_re_t` and `matrix_cplx_t` types enable parallelization- and real/complex-agnostic physics code

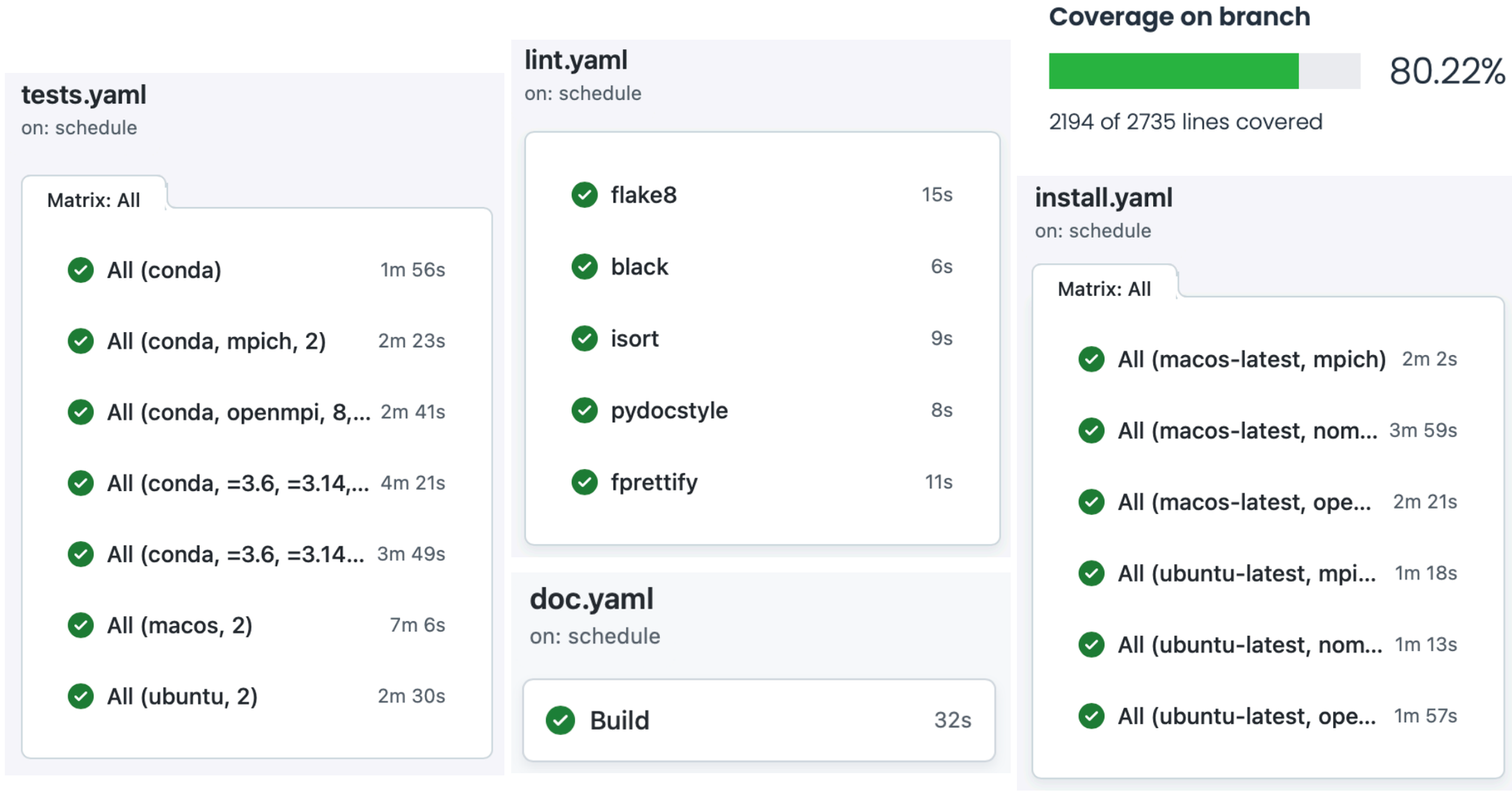
```
subroutine matrix_re_eigh(A, eigs, exc, src, vals_only)
    [...]

#ifdef WITH_SCALAPACK
    if (A%idx%parallel) then
#ifdef WITH_ELSI
        if (present(clock)) call clock%clock(18)
        call elsi_eigh(A%val, A%blacs, eigs, exc, src%val, vals_only)
        if (present(clock)) call clock%clock(-18)
#else
        call peigh(A%val, A%blacs, eigs, exc, src%val, vals_only)
#endif
        return
    end if
#endif
    call eigh(A%val, eigs, exc, src%val, vals_only)
end subroutine
```

```
call c_lambda12i_c%copy_from(modes)
call c_lambda12i_c%mult_cols_3n(eigs**(-1d0 / 4))
c_lambda12i_c = c_lambda12i_c%mmul(c_lambda12i_c, transB='C')
```

Tests and continuous integration

- Github Actions
- Tests, code style, documentation, coverage, installation



Unit testing in Fortran with CTest

- Single Fortran executable parametrized by a test name
- Tests are collected by a Python script, executed by CTest
- Regression tests via Pymbd and Pytest

```
call get_command_argument(1, test_name)
n_failed = 0
call exec_test(test_name)
if (n_failed /= 0) stop 1

contains

subroutine exec_test(test_name)
    character(len=*), intent(in) :: test_name

    select case (test_name)
    case ('T_bare_deriv'); call test_T_bare_deriv()
    case ('T_GG_deriv_expl'); call test_T_GG_deriv_expl()
```

```
execute_process(
    COMMAND ${CMAKE_CURRENT_SOURCE_DIR}/collect-mbd-tests.py
    OUTPUT_VARIABLE TESTS
    OUTPUT_STRIP_TRAILING_WHITESPACE
)
foreach(TEST ${TESTS})
    add_test(NAME ... COMMAND ...)
endforeach()
```

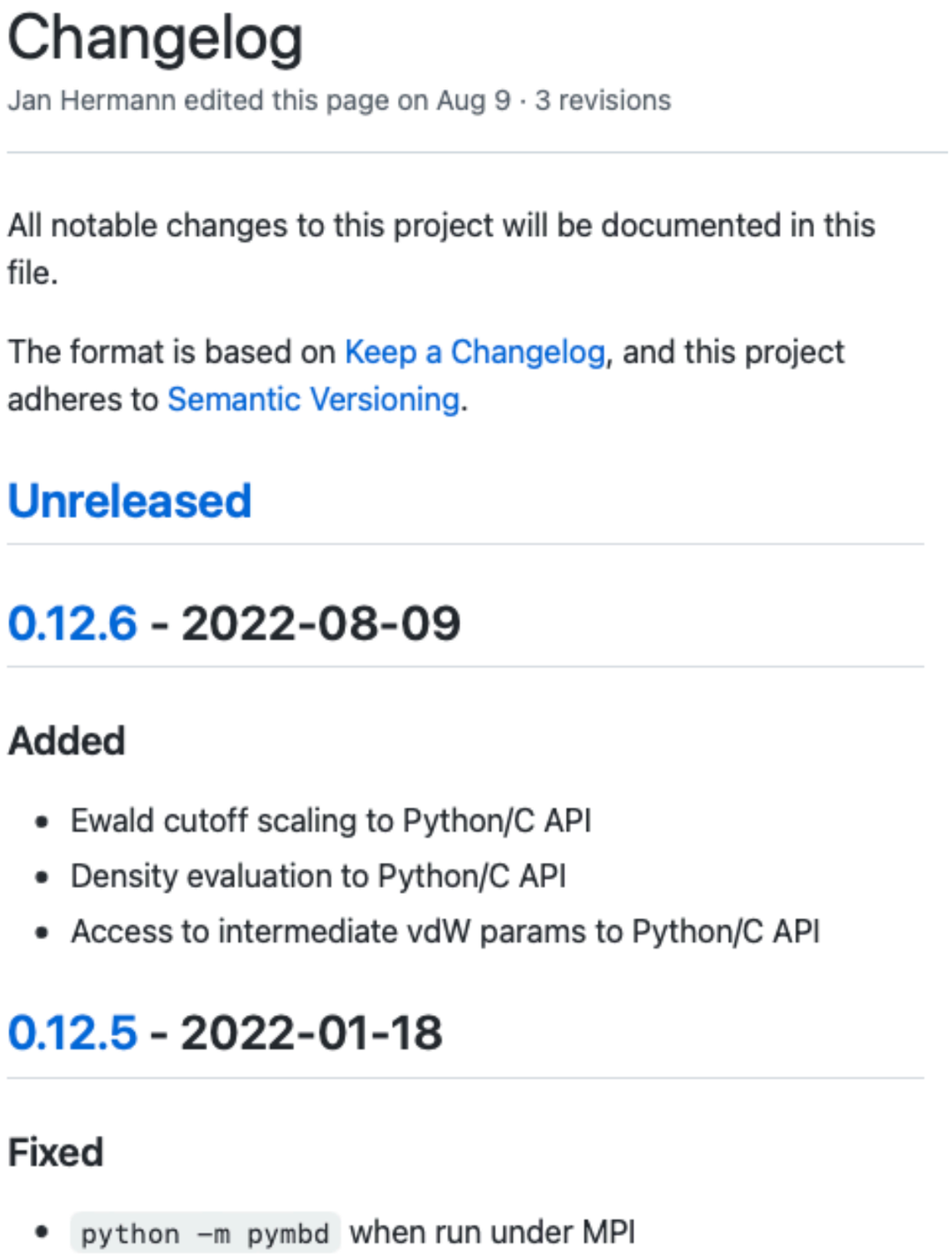
```
ctest --test-dir /home/runner/work/libmbd/libmbd/build --output-on-failure
      Start  1: grad/T_bare_deriv
 1/42 Test  #1: grad/T_bare_deriv .....................   Passed    0.00 sec
      Start  2: grad/T_GG_deriv_expl
 2/42 Test  #2: grad/T_GG_deriv_expl ..................   Passed    0.00 sec
[...]
      Start 42: api/ts_gradients
42/42 Test #42: api/ts_gradients .....................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 42

Total Test time (real) =   1.37 sec
```

Changelog

- Structured changelog based on Keep a Changelog
- Added, changed, removed, fixed
- Automatically generated from release tags



Changelog

Jan Hermann edited this page on Aug 9 · 3 revisions

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

0.12.6 - 2022-08-09

Added

- Ewald cutoff scaling to Python/C API
- Density evaluation to Python/C API
- Access to intermediate vdW params to Python/C API

0.12.5 - 2022-01-18

Fixed

- `python -m pymbd` when run under MPI

Summary

- Libmbd is a Fortran/Python software package for many-body dispersion calculations aimed at both performant scalable calculations and easy development
- Alignment between code and physics
- Lean development—modern techniques with simplest possible tools, no “hacks”
- “How easily could I hand over maintenance to someone else?”