Modern interdisciplinary software development in electronic-structure theory

Michael F. Herbst*[1,2], A. Levitt[1], J. E. Avery[3], M. Scheurer[4], E. Cancès[1], A. Dreuw[4]

[1]Universté Paris-Est, CERMICS (ENPC), Marne-la-Vallée, France; Inria Paris, Paris, France
[2]Sorbonne Universtés, ISCD (UPMC), Paris, France
[3]Niels Bohr Institute, University of Copenhagen, Denmark
[4]Interdisciplinary Center for Scientific Computing, Heidelberg University, Germany

*Email: michael.herbst@enpc.fr — Website: https://michael-herbst.com

Accuracy versus efficiency

- Demand for ever more accurate and reliable electronic structure simulations
- But: Solving the electronic Schrödinger equation is inherently complex

⇒ Need approximate models *and* approximate numerics

- Requires a good balance of **accuracy versus efficiency** in
  - Approximate model
  - Discretisation / basis functions
  - Algorithmic approach to numerics

⇒ No unique *best* answer (depends on system, simulation target, ...)
⇒ Ideal balance continuously shifted:
  - Modern **hardware developments** (GPGPU, FPGA, ...)
  - Above four aspects *not* independent

Interdisciplinary code development

- Electronic structure theory: **Interdisciplinary research** field
- Different researchers have **deviating demands**:
  - Mathematics: Tests only on provable toy problems
  - Method development: Understandable and extensible code base
  - Modelling: Scale-up to real-world applications
- Computational architectures become increasingly inhomogeneous
  ⇒ Re-use of existing code?

⇒ Need **flexible** and **modular** code bases
⇒ **Modern languages** (python, Julia, ...):
  - Reduced coding effort
  - Performance can be comparable to pure C++, FORTRAN, ...

molsturm: Towards method development for arbitrary basis functions [1]

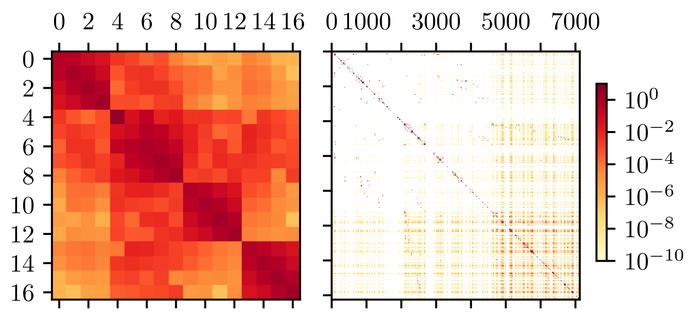

Figure 1. Fock matrices originating from a beryllium calculation employing Coulomb Sturmians (left) or finite elements (right).

- Aim: Screen over basis function types
- Rapid development of new discretisations
- **Abstract** self-consistent field (SCF) implementation
- Challenge: Varying **Fock matrix structures** **Fig. 1**

- Solution: **Contraction-based SCF** scheme
- Basis-function-independent SCF
- Plug-and-play integral libraries in modular framework

⇒ One Code, **multiple basis** functions

- Applications: Convergence of Coulomb-Sturmians [2]
- Python interface connected to PySCF, adcc, ...
- Get code: https://molsturm.org

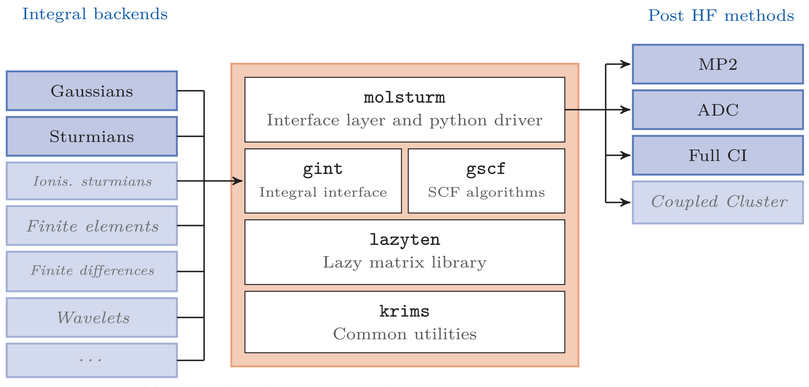

Figure 2. Structure of the molsturm framework

DFTK.jl: The density-functional tool kit [3]

- Plane-wave discretised density-functional theory (DFT)
- Established method, but still lack of understanding:
  - Existence / uniqueness / optimality of solutions
  - **Convergence behaviour** and proof

  ⇒ Are there general lessons to be learnt?
- Joint project: Mathematicians and chemists

- Rigorous mathematical analysis only on **toy problems**
- Ideally proof of concepts should be scaled to **HPC level**

⇒ Need code accessible to mathematicians and scientists

- Project started: January 2019
- DFTK ongoing work:
  - Analysis of SCF algorithms
  - Mixed and elevated precision
  - Mixed basis / grid methods
- https://github.com/mfherbst/DFTK.jl

```julia
function davidson(A, SS; prec=I, tol=1e-8)
  m = size(X0, 2)
  for i in 1:100
    Ass = A * SS
    rvals, rvecs = eigen(SS' * Ass, 1:m)
    Ax = Ass * rvecs

    R = Ax - SS * rvecs * Diagonal(rvals)
    if norm(R) < tol
      return rvals, SS * rvecs
    end

    SS = Array(qr(hcat(SS, prec * R)).Q)
  end
end
```

Figure 3. Julia implementation of a simple Davidson diagonalisation algorithm. No implicit reference to the matrix type is made: This code will work with e.g. sparse, dense or distributed arrays of any data type *at once*.

- DFTK is built on top of julia ecosystem
- Modern HPC in high-level syntax
- Just-in-time compiled scripting language
  - **Performance**: Comparable to C
  - Exploit hardware-specific optimisations
- Key concept: **Multiple dispatch**
  - Write code once
  - Use in multiple contexts, e.g. **Fig. 3**
  - E.g. precision, computational backend
- Language support to call Python / C / C++ / R
- Upcoming Julia features:
  - Adjoint-mode automatic differentiation
  - GPU backends
  - Large-scale data analysis

adcc: Seamlessly connect your program to ADC [4]

```python
# pyscf
#
mol = pyscf.gto.M(atom=open("h2o.xyz").read(),
                  basis='cc-pvtz')
scfresult = pyscf.scf.RHF(mol)
scfresult.conv_tol = 1e-13
scfresult.kernel()

# adcc
#
# ADC(3) calculation on top of scfresult
states = adcc.adc3(scfresult, n_singlets=10)

# Print resulting states and plot spectrum
print(states.describe())
states.plot_spectrum(label="water cc-pVTZ")
plt.legend(); plt.show()
```

Figure 4. Python script performing an ADC(3) calculation on top of an scfresult, which has been obtained from a third-party code (in this case PySCF)

- ADC(n): algebraic diagrammatic construction approach for the polarisation propagator (levels $n \leq 3$ supported)
- Reliable Post-HF **excited-states** method

- adcc Python layer:
  - Calculation workflow
  - SCF interface
  - **Solver algorithms** (e.g. Davidson)
- adcc C++ layer: Tensor computations [5]

⇒ Performance comparable to pure C++ code
⇒ Combine adcc with **any third-party SCF**:
  - E.g. PySCF, Psi4, molsturm, ...

⇒ Easily develop new numerical schemes for ADC

- Will be part of the Gator spectroscopy program [6]

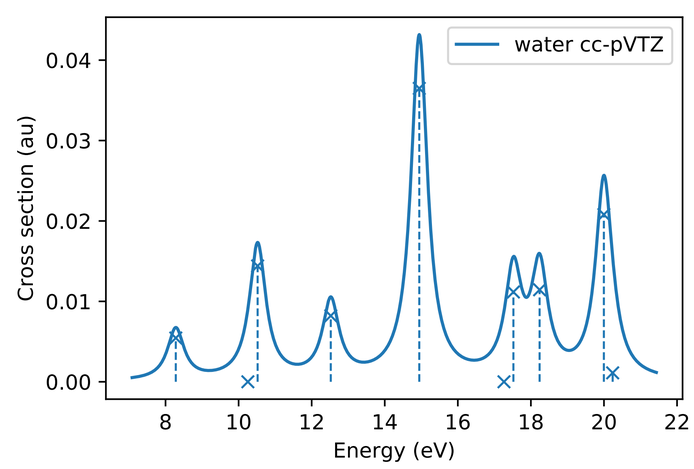

Figure 5. Spectrum obtained with the script of **Fig. 4** for a cc-pVTZ water Hartree-Fock reference in scfresult.

Acknowledgements

Discussions with Brian Vinter, Mads Kristensen (Copenhagen) and Thomas Fransson, Guido Kanschat, Adrian Dempwolff (Heidelberg) are gratefully acknowledged.

References

[1] M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018).

[2] M. F. Herbst, J. E. Avery and A. Dreuw. Phys. Rev. A, **99**, 012512 (2019).

[3] M. F. Herbst and A. Levitt. *The density-functional tool kit.* https://github.com/mfherbst/DFTK.jl/.

[4] M. F. Herbst, M. Scheurer, *adcc: Seamlessly connect your program to ADC. To be released.*

[5] E. Epifanovsky, M. Wormit, T. Kuś, *et. al.*, J. Comput. Chem., **34**, 2293 (2013).

[6] D. R. Rehn, Z. Rinkevicius, M. F. Herbst, *et. al.*, *Gator: a Python-driven program for spectroscopy simulations using correlated wave functions. Submitted to WIREs Comput. Mol. Sci.*