molsturm:
Modular electronic structure theory framework

A tale inspired by Coulomb-Sturmians

Michael F. Herbst, James E. Avery, Guido Kanschat, Andreas Dreuw

Oberwolfach workshop
Mathematical Methods for Quantum Chemistry

23rd March 2018

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
IWR Interdisciplinary Center for Scientific Computing

Contents

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

Contents

An ideal basis

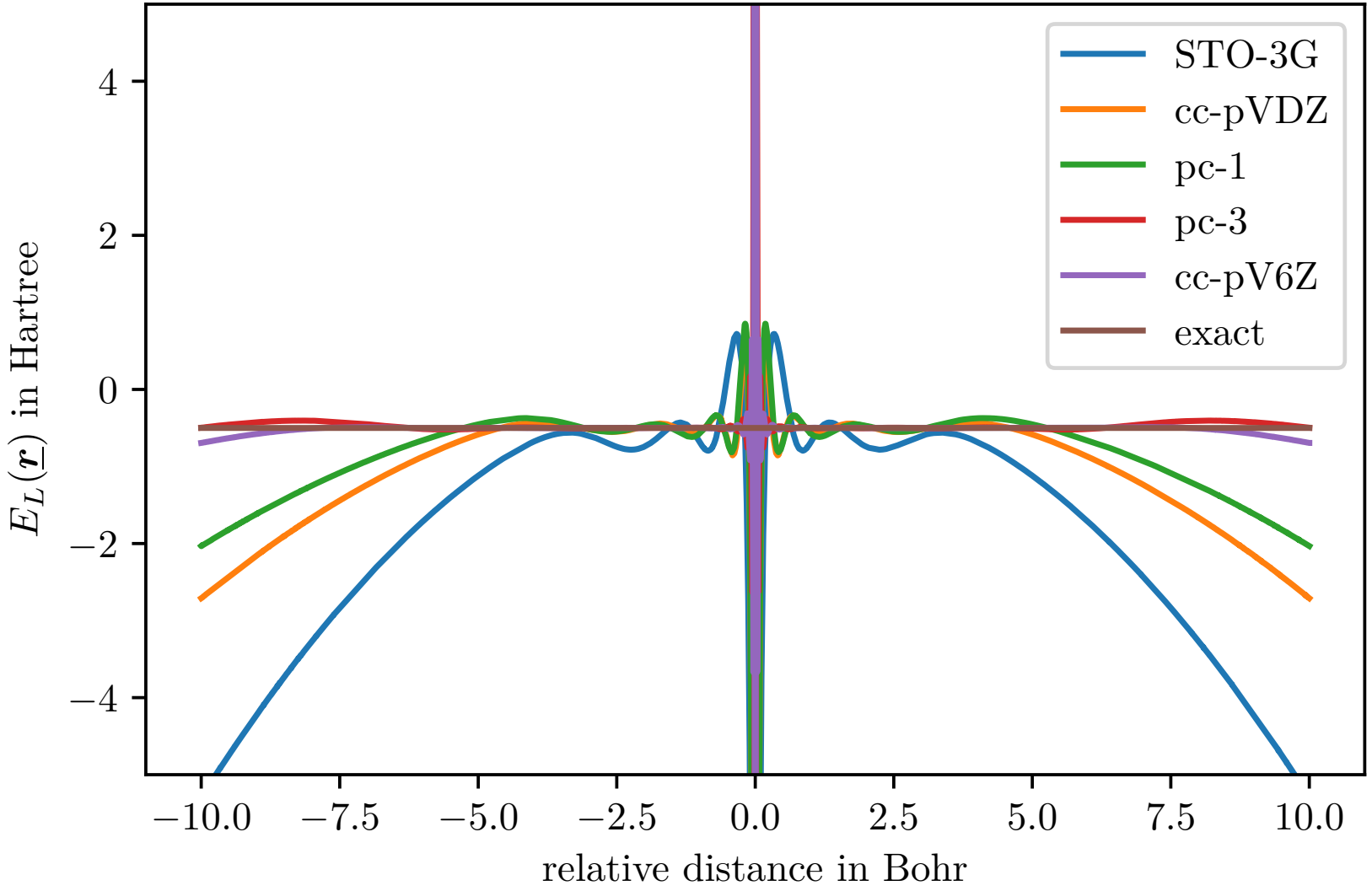
- Represents physical system well
- Results reliable
  - Error margin known
  - Systematic improvement possible
- Prior knowledge
  - Little required
  - If available: Can be incorporated
- Integrals and eigenproblem are feasible

$\Rightarrow$ In reality need a good compromise

Basis function types

- Gaussian-type orbitals
- Geminals
- Slater-type orbitals
- Wavelets
- Finite elements
- Plane wave
- Augmented plane waves
- ...

Discretising hydrogen



Coulomb-Sturmians

- Iso-energetic solutions φ_{nlm} to hydrogen-like equation[1]

$$\left(-\frac{1}{2}\Delta - \beta_n \frac{Z}{r}\right)\varphi_{nlm}(\underline{\boldsymbol{r}}) = E\varphi_{nlm}(\underline{\boldsymbol{r}})$$

- Scaling factor β_n chosen to uniform energy:

$$\beta_n = \frac{kn}{Z} \quad \Rightarrow \quad E = -\frac{k^2}{2}$$

- φ_{nlm} look like hydrogenic orbitals with $\frac{Z}{n}$ replaced by k
- Radial part R_{nl} satisfies

$$\left(-\frac{1}{2r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{l(l+1)}{2r^2} - \frac{nk_{\text{exp}}}{r} - E\right)R_{nl}(r) = 0.$$

$\Rightarrow$ Sturm-Liouville equation[2]

[1] Shull and Löwdin 1959
[2] Rotenberg 1962, Rotenberg 1970

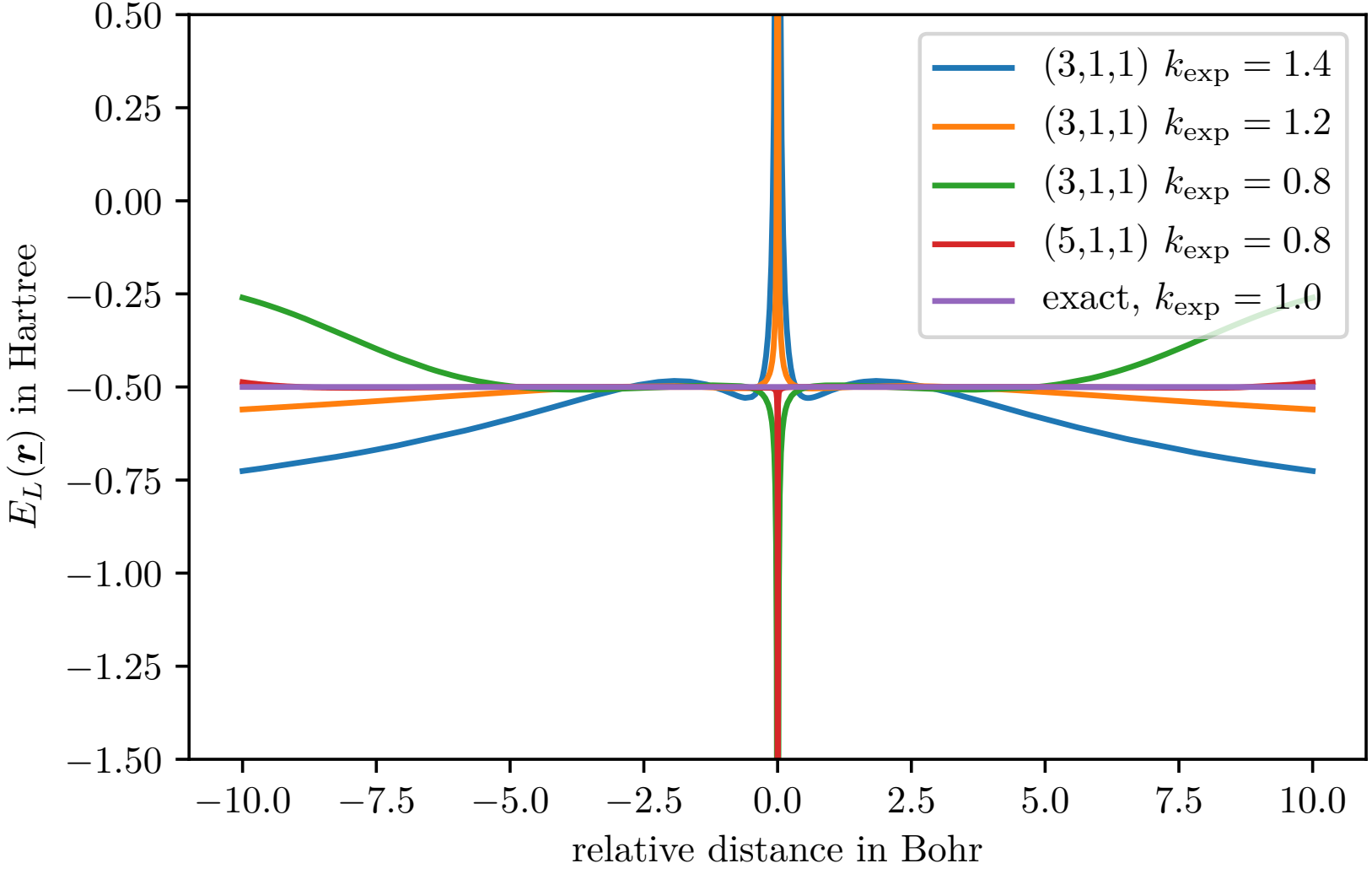
Coulomb-Sturmians properties

- Only atoms: Different Sturmians for molecules required
- Complete basis for $H^1(\mathbb{R}^3)$[1]
- Correctly represent nuclear cusp
- Proper exponential decay for large r
- All functions have the same k_{exp}

- One-to-one onto hyperspherical harmonics
- One-electron integrals sparse and analytic
- Two-electron integrals sparse tensor contraction

[1] Klahn and Bingel 1977

Discretising hydrogen



Fock matrix structures

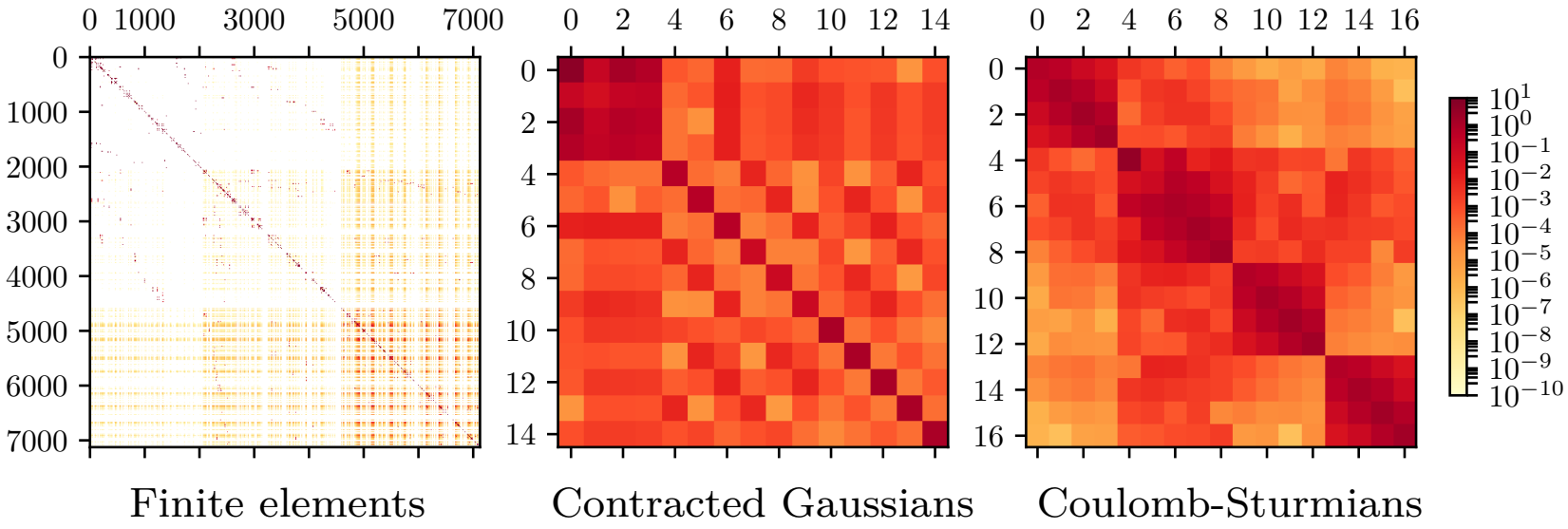
- Main concern: Hartree-Fock
- After discretisation:

$$\mathbf{F}[\mathbf{C}]\,\mathbf{C} = \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)\,\mathbf{S}\,\mathbf{C}$$

Fock matrix structures

- Main concern: Hartree-Fock
- After discretisation:

$$\mathbf{F}[\mathbf{C}]\,\mathbf{C} = \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)\,\mathbf{S}\,\mathbf{C}$$



Contents

SCF scheme

Start with guess $\mathbf{F}^{(0)}$ and iterate for $n = 1, 2, 3, \ldots$

- Diagonalise

$$\tilde{\mathbf{F}}^{(n-1)}\mathbf{C}_F^{(n)}\mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_N) = \mathbf{S}\mathbf{C}_F^{(n)}\mathbf{E}^{(n)}$$

  under the condition

$$\left(\mathbf{C}_F^{(n)}\right)^\dagger \mathbf{S}\mathbf{C}_F^{(n)} = \mathbf{I}_N.$$

- Construct the occupied matrix $\mathbf{C}^{(n)}$ from the full matrix $\mathbf{C}_F^{(n)}$ by the Aufbau principle.
- Build the Fock matrix $\mathbf{F}\left[\mathbf{C}^{(n)}\right]$ and check for convergence.
- Build a trial Fock matrix $\tilde{\mathbf{F}}^{(n)}$ somehow using $\mathbf{F}^{(n)}$ and all insight into the problem gathered so far.

Contraction-based SCF scheme

- Iterative solvers only need matrix-vector products

⇒ Contraction-based or matrix-free[1] algorithm:

  - Never build Fock matrix in storage
  - Use matrix-vector contraction expressions

- Avoid full density matrix
  - Employ SCF iterating orbital coefficients $\mathbf{C}$

[1] Kronbichler and Kormann 2012

Characteristics of contraction-based algorithms

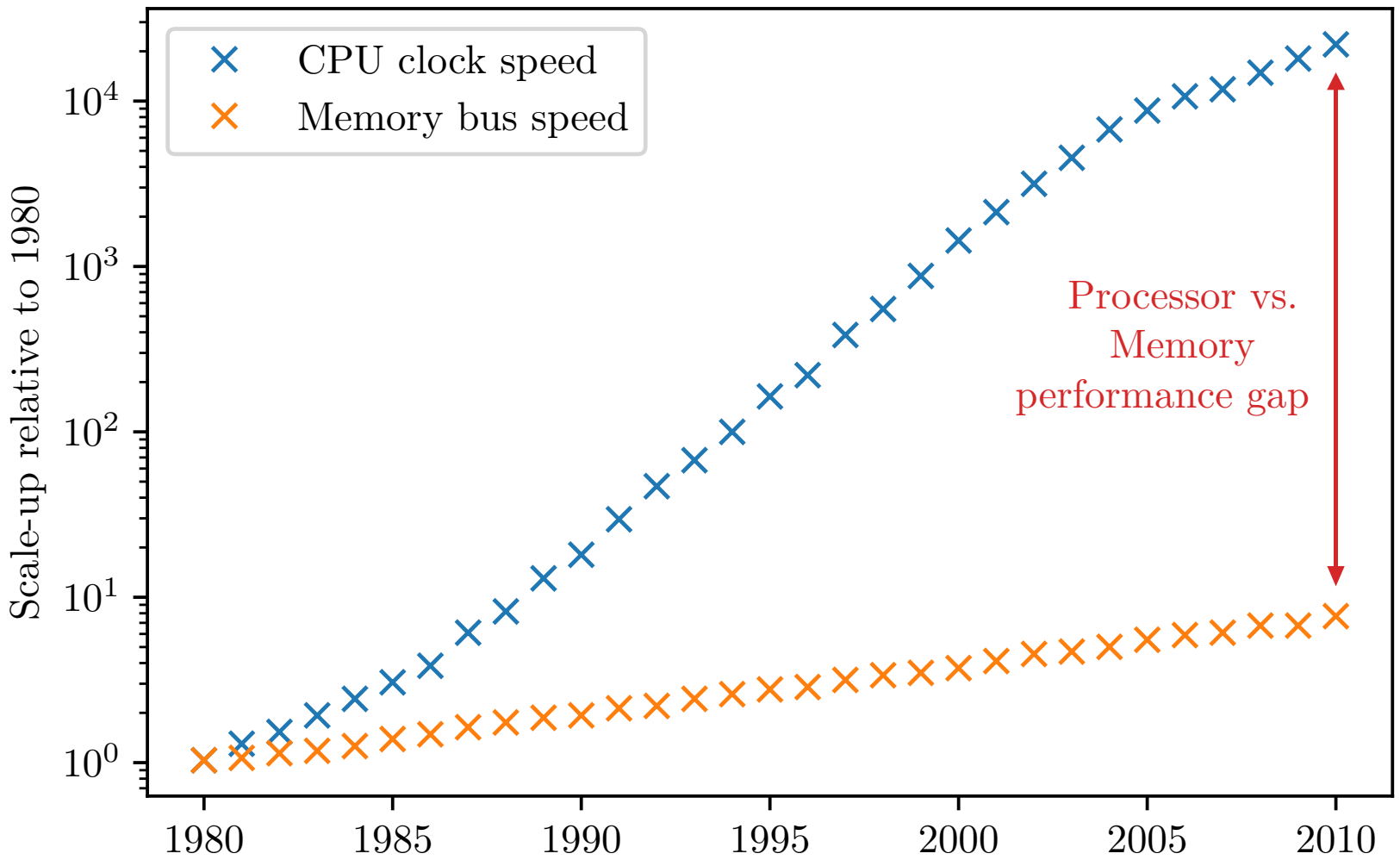
Advantages

- Scaling (storage and time) reduced
- Parallelisation easier
  - ⇒ Less data management
  - ⇒ Easier modularisation
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than contraction-functions
- More computations
  - ⇒ Need efficient contraction schemes for the contraction
  - ⇒ Algorithms more complex

Characteristics of contraction-based algorithms



Data from `https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast`

Characteristics of contraction-based algorithms

Storage layer	Latency /ns	FLOPs
L1 cache	0.5	13
L2 cache	7	180
Main memory	100	2600
SSD read	$1.5 \cdot 10^4$	$4 \cdot 10^5$
HDD read	$1 \cdot 10^7$	$3 \cdot 10^8$

Data from
`https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html`
FLOPs for a Sandy Bridge 3.2GHz CPU with perfect pipelining

Characteristics of contraction-based algorithms

Advantages

- Scaling (storage and time) reduced
- Parallelisation easier
  - ⇒ Less data management
  - ⇒ Easier modularisation
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than contraction-functions
- More computations
  - ⇒ Need efficient contraction schemes for the contraction
  - ⇒ Algorithms more complex

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Generalisation of matrices
    - Elements may be expressions
    - May represent non-linear operators
    - State

  $\Rightarrow$ Obtaining elements expensive
  - Evaluation of internal expression: Delayed until contraction
  - For convenience: Offer matrix-like interface

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally

$$\boxed{\mathbf{D}} = \boxed{\mathbf{A}} + \boxed{\mathbf{B}}$$

- On application:

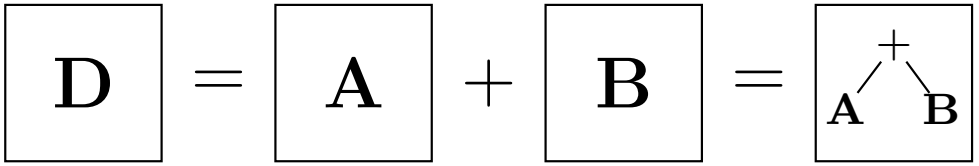
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally



- On application:

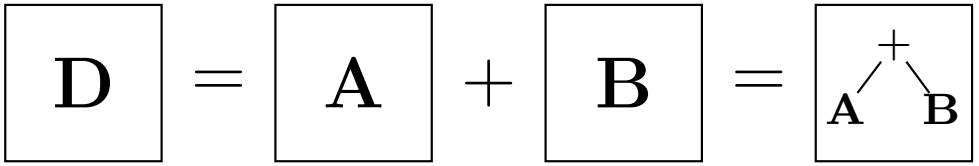
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally



- On application:

$$\mathbf{D}\underline{\boldsymbol{x}} = (\mathbf{A}\underline{\boldsymbol{x}}) + (\mathbf{B}\underline{\boldsymbol{x}})$$

Observations

- Lazy matrices allow layered responsibility for computation, e.g. $(\mathbf{A} + \mathbf{B})\underline{\boldsymbol{x}}$
  - $\mathbf{A}\underline{\boldsymbol{x}}$ and $\mathbf{B}\underline{\boldsymbol{x}}$ decided by implementation of $\mathbf{A}$ and $\mathbf{B}$
  - $(\mathbf{A}\underline{\boldsymbol{x}}) + (\mathbf{B}\underline{\boldsymbol{x}})$ done in linear algebra backend

$\Rightarrow$ Proper modularisation between

- Higher-level algorithms
- Lazy matrix implementations
- LA backends

Lazy matrices for Hartree-Fock codes

- Contraction-based algorithms

⇒ Lower memory footprint, scaling improvements

- Abstraction between integrals and SCF algorithms

⇒ Plug and play integral libraries

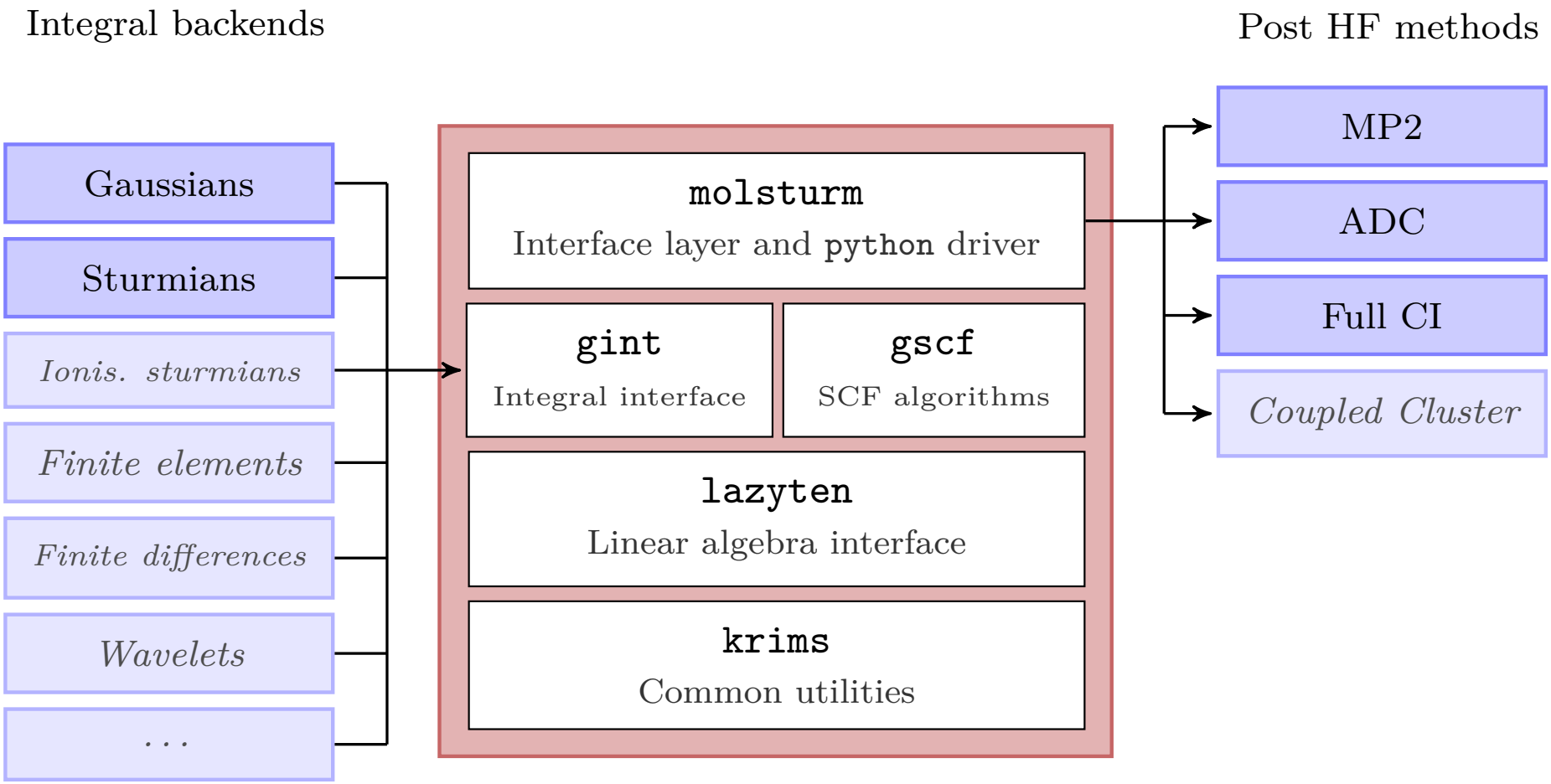
⇒ Swap LA backends

⇒ Basis-type independent SCF

- Integral back end stays in control of contraction

⇒ Decides order of data production and consumption

molsturm structure



- *Italic*: planned

molsturm design

- Lightweight connection between integrals and Post-HF
- Flexiblity as primary goal
- Behaviour controlled via `python`
  - Keywords to change basis type or solver
  - All computed data available in `numpy` format
  - No input file, just a `python` script
- `python` utilities
  - Import / export results
  - Post-HF calculations
- Integration with `python` ecosystem

molsturm interface: CCD residual (parts)

$$
\begin{aligned}
r_{ij}^{ab} = &-\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{af}\, t_{ij}^{eb} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{bf}\, t_{ij}^{ea} - \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{in}^{ef}\, t_{mj}^{ab} \\
&+\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jn}^{ef}\, t_{mi}^{ab} + \frac{1}{4}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{ab}\, t_{ij}^{ef} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{im}^{ae}\, t_{jn}^{bf} \\
&-\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jm}^{ae}\, t_{in}^{bf} - \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{im}^{be}\, t_{jn}^{af} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jm}^{be}\, t_{in}^{af}
\end{aligned}
$$

```
eri_phys = state.eri.transpose((0, 2, 1, 3))
eri = eri_phys - eri_phys.transpose((1, 0, 2, 3))
res = \
    - 0.5  * einsum("mnef,manf,iejb->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,mbnf,ieja->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,ienf,majb->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,jenf,maib->iajb", eri.block("oovv"), t2, t2) \
    + 0.25 * einsum("mnef,manb,iejf->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,iame,jbnf->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,jame,ibnf->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,ibme,janf->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,jbme,ianf->iajb", eri.block("oovv"), t2, t2)
```

Contents

Outlook

- Exploring further basis functions
  - Molecular Sturmians
  - Ionising Sturmians
  - k_{exp}-free Hartree-Fock
- `pammap`
  - Hierachical interface
  - Data transfer between high and low level
  - Easy parameter adjustment from `python`, `julia`, ...
- Fuzzing of integral back ends
- Lazy matrix expression optimisation

Acknowledgements

- Dr. James Avery
- Prof. Andreas Dreuw
- Prof. Guido Kanschat
- HGS Mathcomp

Questions?

- EMail: `michael.herbst@iwr.uni-heidelberg.de`
- `https://michael-herbst.com`
- Projects: `https://lazyten.org` and `https://molsturm.org`