molsturm: Towards quantum-chemical method development for arbitrary basis functions

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

Molecular Electronic Structure 2018, Metz

28th August 2018

Basis function types

- Gaussian-type orbitals
- Geminals

- Slater-type orbitals
- Sturmian-type orbitals

- ...

- Plane waves
- Augmented plane waves

- Wavelets
- Finite elements

- ...

Basis function types

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

$\Rightarrow$ But Gaussians work fine, right?

Beyond Gaussian-type orbitals

- Well, what about . . .
  - Representing the core region?
  - Representing the exponential decay?
  - Error estimates?
  - Black box modelling?
  - Distributed memory parallelisation?

$\Rightarrow$ Playing field to try other basis function types

Testing alternative basis function types

- Obstacle: 1 Program $\simeq$ 1 basis function type

$\Rightarrow$ Basis type often burned into existing codes

$\Rightarrow$ A new program for each basis type just to try it?

- Structure of SCF problem:

$$\mathbf{FC} = \mathbf{SC}\ \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)$$

$\Rightarrow$ Independent of basis choice

$\Rightarrow$ It should be sufficient to swap the integral backends!

Testing alternative basis function types

- Obstacle: 1 Program $\simeq$ 1 basis function type

$\Rightarrow$ Basis type often burned into existing codes

$\Rightarrow$ A new program for each basis type just to try it?

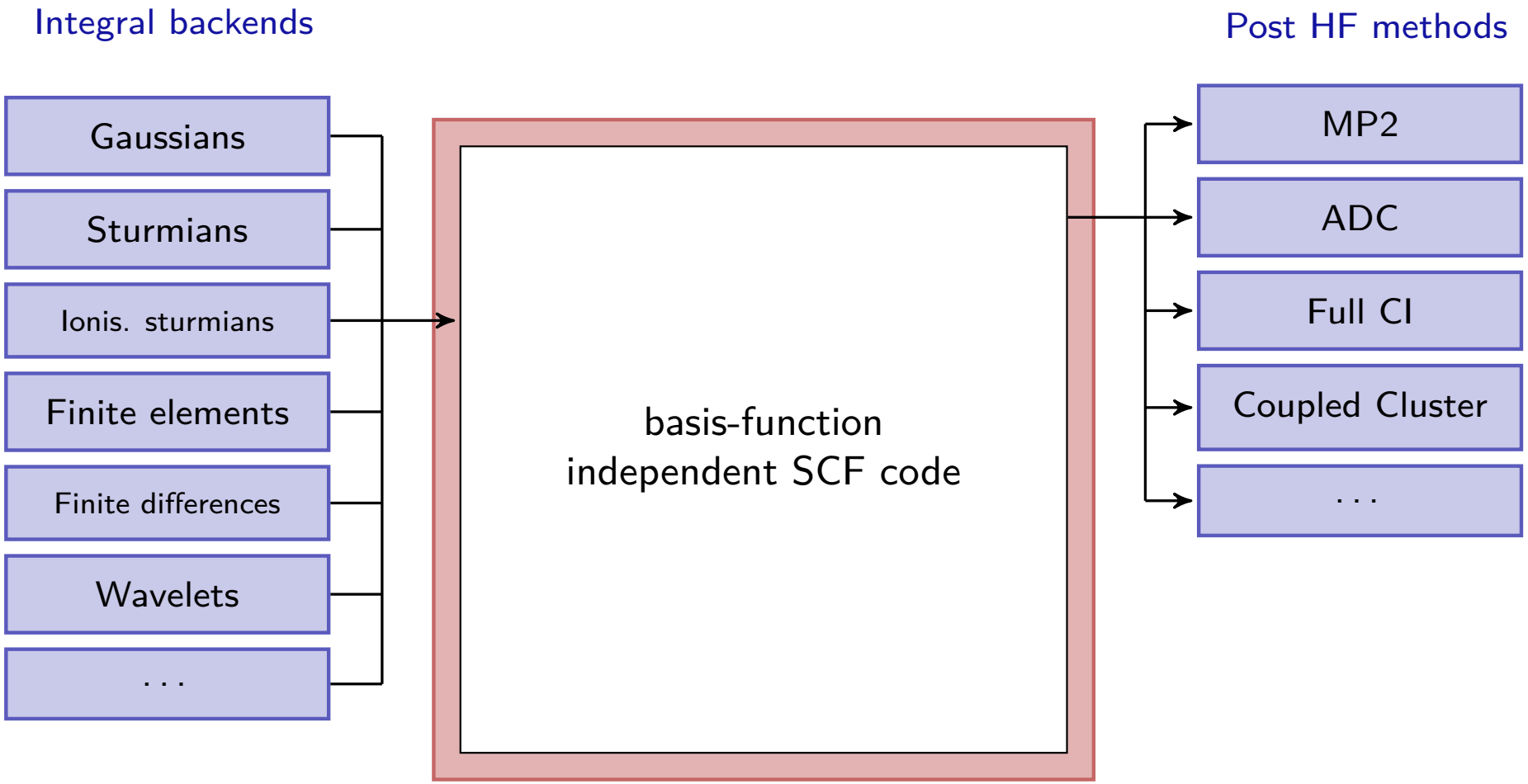
- Structure of SCF problem:

$$\mathbf{FC} = \mathbf{SC}\ \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)$$

$\Rightarrow$ Independent of basis choice

$\Rightarrow$ It should be sufficient to swap the integral backends!

Aims of molsturm



Achievements of molsturm

- Basis-function independent design
  - Plug and play new discretisations
  - Basis-type agnostic SCF procedure
- Easy-to-use interfaces
  - Integrate with existing code (e.g. Post-HF)
  - Avoid reinventing the wheel
  - Rapid prototyping, testing and analysis

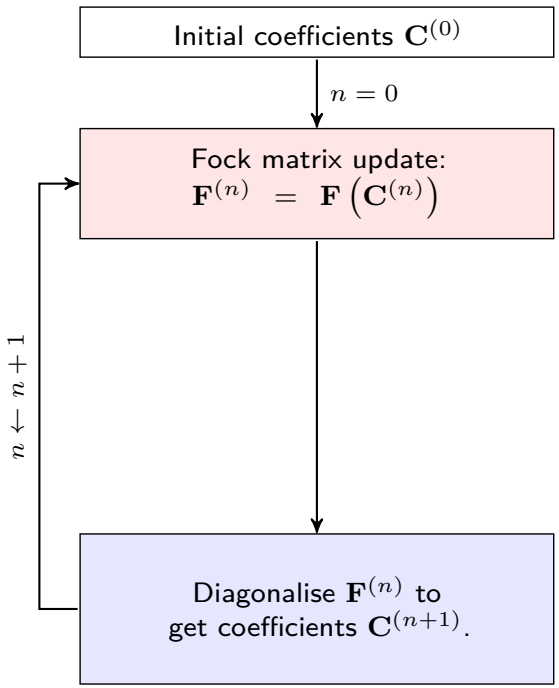
$\Rightarrow$ Explore methods across basis function types[1,2]

---

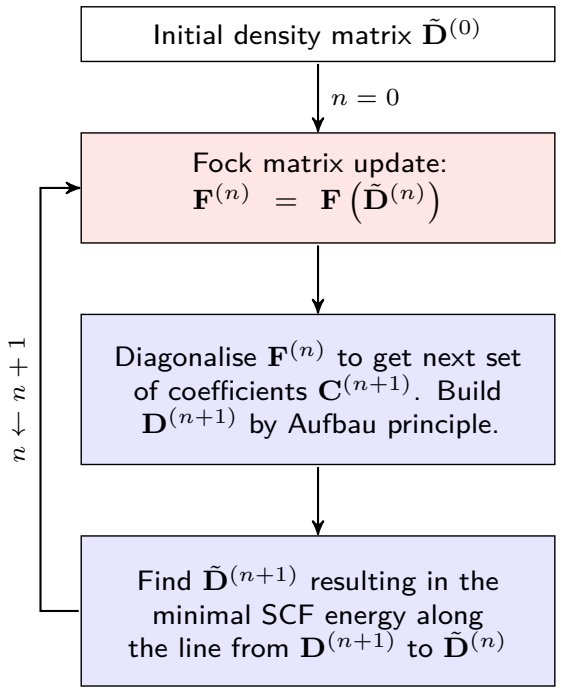
[1] M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018)

[2] M. F. Herbst. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg (2018)

Two-step structure of SCF algorithms



Roothaan scheme

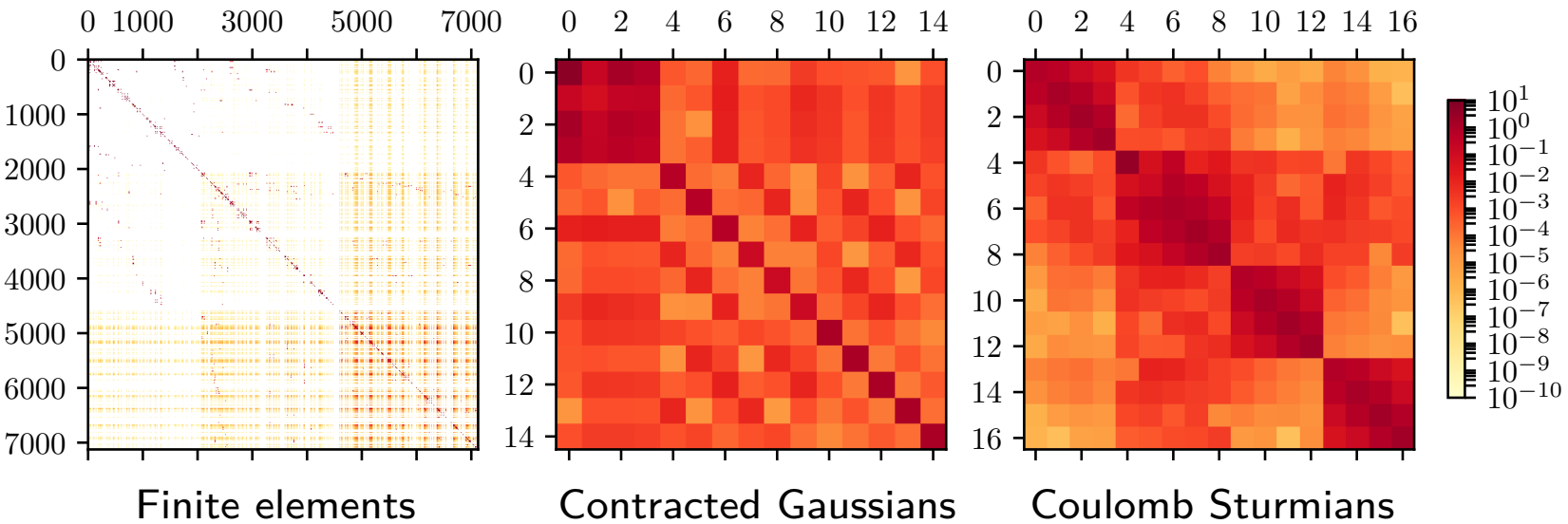


Optimal damping algorithm

- Fock update
- Coefficient update
  (density matrix update)

$\Rightarrow$ Need to be basis-type independent

Challenge: Deviating Fock matrix structures



- Required numerical procedures differ
- Details should be hidden from SCF
- Focus on HF, but our approach extends to DFT

Solution: Contraction-based methods

- Contraction-based methods
  - Avoid storing matrices
  - Employ iterative, subspace-based algorithms
  - Contraction expressions (e.g. matrix-vector products)
  - Common in Post-HF: *Working equations*

$\Rightarrow$ SCF code only needs Fock contraction

$\Rightarrow$ Hide discretisation details inside

$\Rightarrow$ Flexible to exploit discretisation-specific properties

Lazy matrices

- Contraction expressions dressed as a matrix
- Build and pass Fock expression tree to SCF
- Lazy evaluation:

$$\mathbf{F} = \mathbf{h} + \mathbf{J} - \mathbf{K}$$

$$\vdots$$

Iterative solver

$$\underline{y} = \mathbf{F}\underline{x}$$

$$\boxed{\underline{y}} = \boxed{\mathbf{F}}\,\boxed{\underline{x}} = \boxed{\begin{array}{ccc} & & -\mathbf{K} \\ & + & \\ \mathbf{h} & & \mathbf{J} \end{array}}\,\boxed{\underline{x}} = (\mathbf{h} + \mathbf{J} - \mathbf{K})\,\underline{x}$$

- Integral back end provides matrix terms

Contraction-based, two-step SCF

Fock expression	Lazy matrix, sum of integral terms
Coefficient update	Iterative solvers
Fock update	Replace coefficients in expression tree

Achieve basis-function independence:

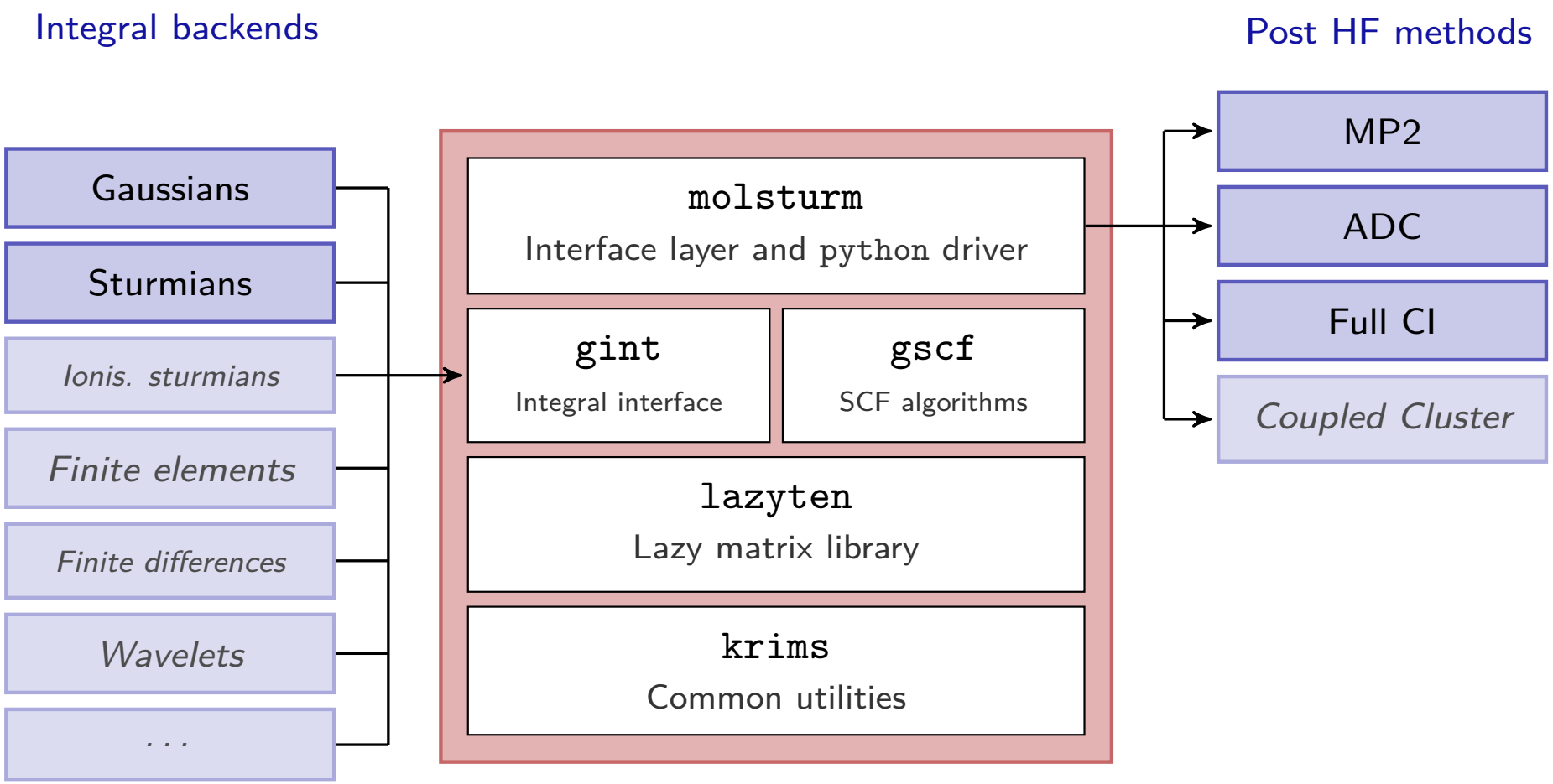
- Lazy matrices: Abstraction between integrals and SCF
- Integral back end: Controls evaluation of contractions

$\Rightarrow$ Decides integral data production and consumption

$\Rightarrow$ Transparent to SCF

$\Rightarrow$ May exploit discretisation-specific properties

molsturm structure



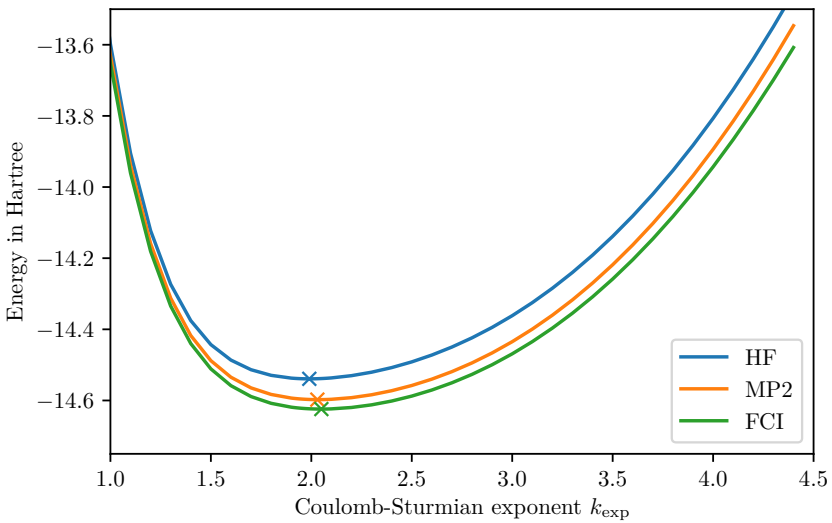
molsturm interface: CCD residual (parts)

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

```
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)
```

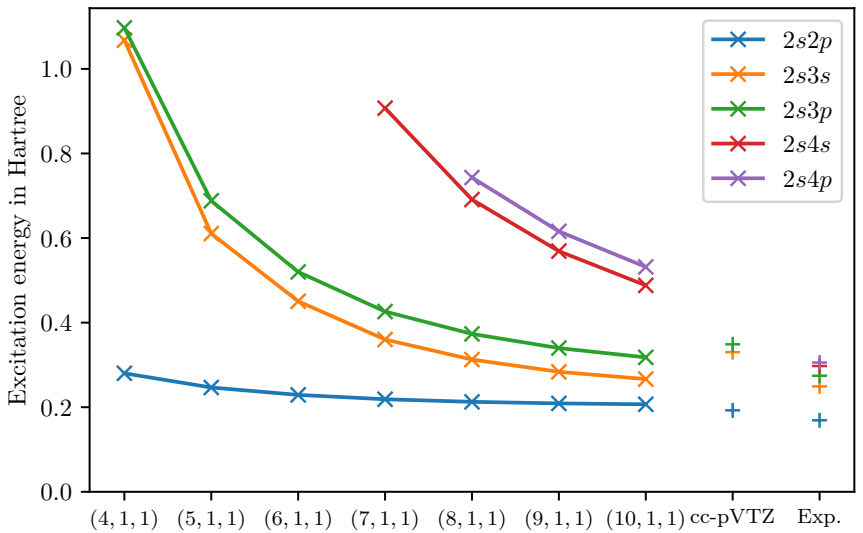
molsturm interface: Linked codes

Coulomb-Sturmian based MP2 and FCI



- FCI from `pyscf`[1]
- Coulomb Sturmians from `sturmint`[2]

Coulomb-Sturmian and Gaussian based ADC(2)



- ADC(2) from `adcman`[3]
- Gaussians from `libint`[4] or `libcint`[5]

[1] Q. Sun et al. WIREs Comput Mol Sci, **8**, e1340 (2017).

[2] J. E. Avery and M. F. Herbst. `https://molsturm.org/sturmint` (2018)

[3] M. Wormit et al. Mol. Phys., **112**, 774 (2014).

[4] E. Valeyev et al. *evaleev/libint: 2.3.1* (2017).

[5] Q. Sun. J. Comput. Chem., **36**, 1664 (2015)

molsturm[1] in essence

- Contraction-based, basis-function independent SCF
- Mediator: Integral libraries $\leftrightarrow$ Post-HF
- Modular and light-weight structure

$\Rightarrow$ Make use of what exists (Post-HF or python)

- Plug-and-play new basis-function types
- Allow apples-to-apples comparison
- Rapid prototyping of new methods

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

Acknowledgements


James Avery


Andreas Dreuw


Adrian Dempwolff


Guido Kanschat


Maximilian Scheurer

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

IWR
Interdisciplinary Center for Scientific Computing

HGS MathComp

Questions?

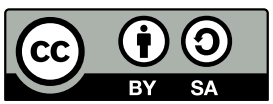
Paper: M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018)

Thesis: `https://michael-herbst.com/phd-thesis.html`

Code: `https://molsturm.org`

Email: `michael.herbst@iwr.uni-heidelberg.de`

Blog: `https://michael-herbst.com/blog`



Lazy matrix evaluation

- Actual expression in source code

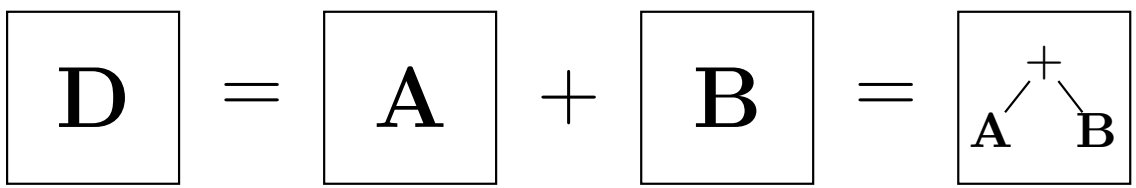
$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation



Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation

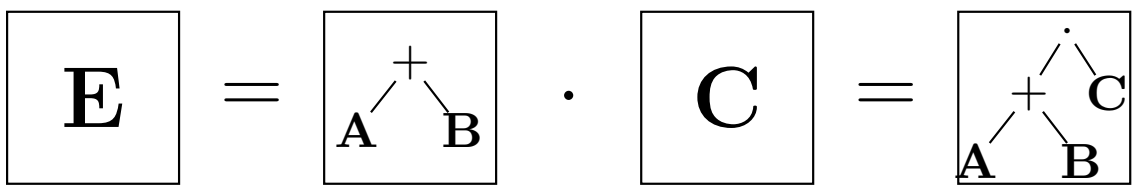
$$\boxed{\mathbf{E}} = \boxed{\mathbf{D}} \cdot \boxed{\mathbf{C}}$$

Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation



Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation

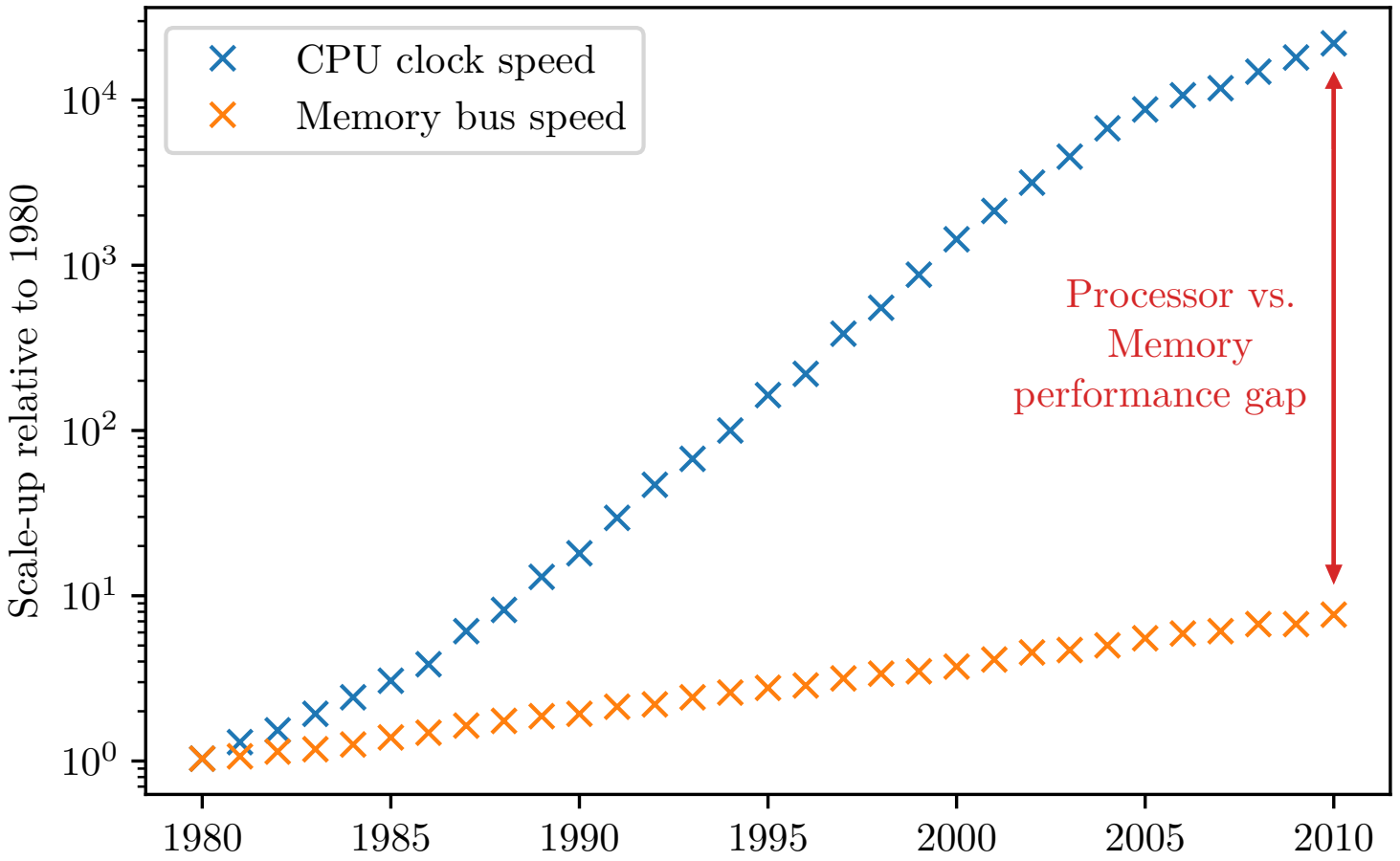
$$\boxed{\underline{\boldsymbol{y}}} = \boxed{\mathbf{E}}\,\boxed{\underline{\boldsymbol{x}}} = \boxed{\begin{array}{ccc} & + & \\ \mathbf{A} & & \mathbf{C} \\ & \mathbf{B} & \end{array}}\,\boxed{\underline{\boldsymbol{x}}} = (\mathbf{A} + \mathbf{B})\,\mathbf{C}\,\underline{\boldsymbol{x}}$$

Typical latency numbers

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

Growth of memory and CPU speeds over time



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

lazyten: Lazy matrix library