Lazy matrices in quantum chemistry

Michael F. Herbst
michael.herbst@iwr.uni-heidelberg.de
http://michael-herbst.com

Interdisziplinäres Zentrum für wissenschaftliches Rechnen
Ruprecht-Karls-Universität Heidelberg

15th June 2017

Contents

Contents

Hartree-Fock equations

- Hartree-Fock equations

$$\left(-\frac{1}{2}\Delta + \hat{\mathcal{V}}_{\mathrm{Nuc}} + \hat{\mathcal{V}}_{\mathrm{H}}\left[\{\psi_f\}_{f\in I}\right] + \hat{\mathcal{V}}_x\left[\{\psi_f\}_{f\in I}\right]\right)\psi_f = \varepsilon_f\psi_f$$

with

$-\frac{1}{2}\Delta$	Kinetic energy of electrons
$\hat{\mathcal{V}}_{\mathrm{Nuc}}$	Electron-nuclear interaction
$\hat{\mathcal{V}}_{\mathrm{H}}\left[\{\psi_f\}_{f\in I}\right]$	Hartree potential
$\hat{\mathcal{V}}_x\left[\{\psi_f\}_{f\in I}\right]$	Exchange potential

- Non-linear system of partial differential equations
- Eigenproblem for eigenpairs $\{(\varepsilon_f, \psi_f)\}_{f\in I}$

Discretisation of Hartree-Fock

- Discretise using finite basis $\{\varphi_b\}_{b\in B}$, i.e.

$$\psi_f = \sum_b c_b^{(f)} \varphi_b$$

$\Rightarrow$ Non-linear discretised Eigenproblem

$$\left(\mathbf{T} + \mathbf{V}_{\text{Nuc}} + \mathbf{J}\left[\{c_b^{(f)}\}\right] + \mathbf{K}\left[\{c_b^{(f)}\}\right]\right) \underline{\boldsymbol{c}}^{(f)} = \varepsilon_f \underline{\boldsymbol{c}}^{(f)}$$

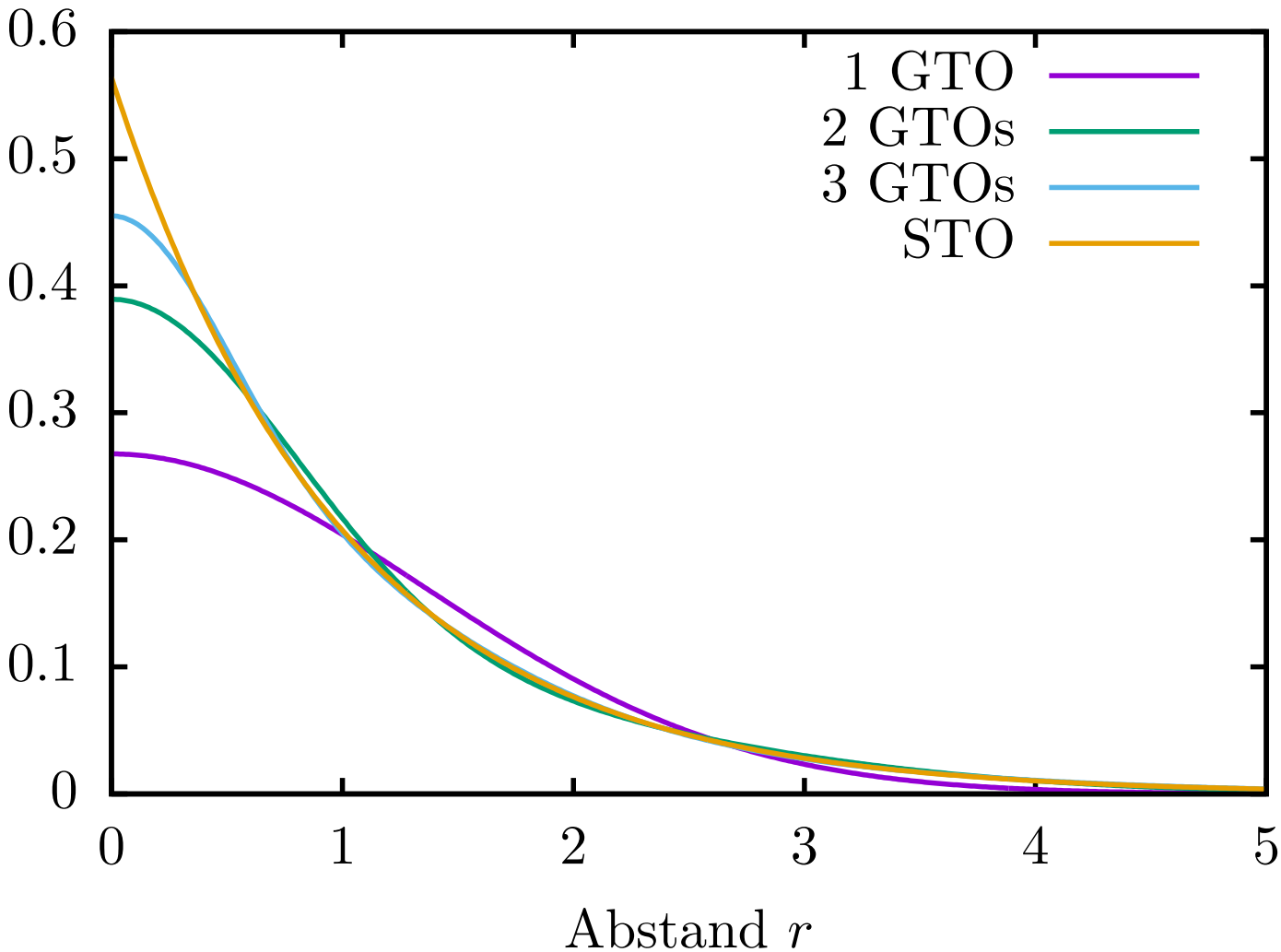
- Variational, but which choices of φ_b are best?

An ideal basis

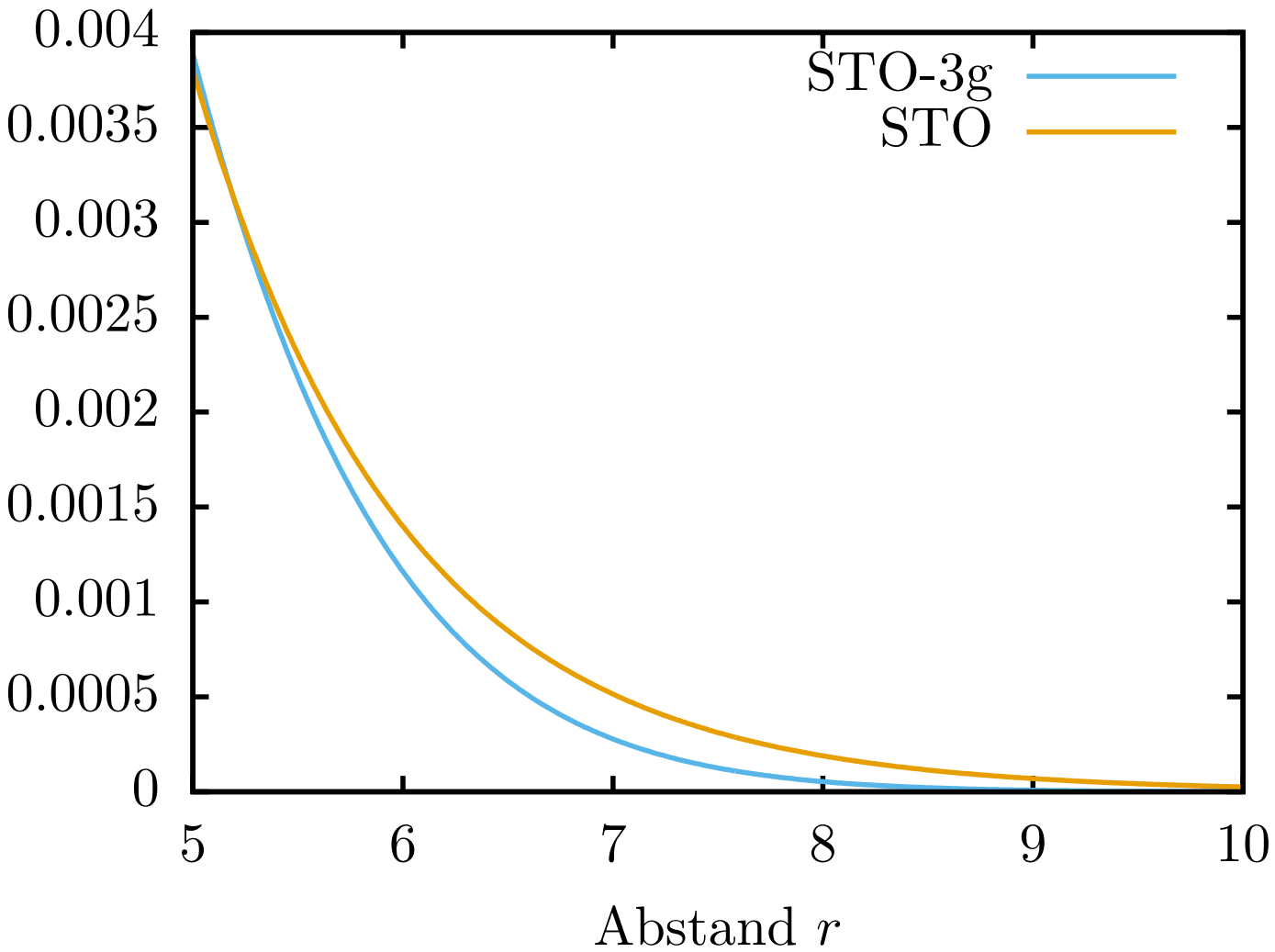
- Represents physical system well (Geometry!)
- Results reliable
  - Error margin known
  - Systematic improvement possible
- Prior knowledge
  - Little required
  - Can still be incorporated
- Integrals and eigenproblem are feasible

$\Rightarrow$ In reality need a good compromise

Gaussian basis sets



Gaussian basis sets



Gaussian basis sets

- Gaussian-product theorem:
  - ERI tensor $\langle \varphi_a \varphi_b | \varphi_c \varphi_d \rangle$ feasible
  - **J** and **K** feasible
- Need contracted GTOs (cGTOs)
- Nuclear cusp and tail still problematic

$\Rightarrow$ Some tasks require special basis sets (e.g. diffuse functions)

- cGTOs can become strongly linearly dependant
- Small to moderate basis sizes
- Basis set families with known convergence properties

Functional form

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

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

- Relationship between scaling factor and 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
- Correctly represent nuclear cusp
- Proper exponential decay for large r

Properties

- Only atoms: Different Sturmians for molecules required
- Complete basis set
- No contractions (yet)
- Basis has free tuning parameter k
- k-free Full-CI exists
- One-electron integrals sparse
- ERI tensor from pre-computed sparse tensor:

$$\langle \phi_a \phi_b | \phi_c \phi_d \rangle = \sum_{\mu\nu} \mathcal{C}^{\mu}_{ab} I_{\mu\nu} \mathcal{C}^{\nu}_{cd}$$

Preliminary results

- Helium HF energies:

22 Sturmians	$-2.861\,679\,995\,612\,\mathrm{E_H}$
CBS value[1]	$-2.861\,679\,995\,615\,\mathrm{E_H}$

Coulomb Sturmians with $n_{\max} = 22$, $l_{\max} = m_{\max} = 0$, $k = 2$

- Beryllium comparison:

	E_{HF}	E_{MP2}
21 Sturmians	$-14.5634\,\mathrm{E_H}$	$-14.6248\,\mathrm{E_H}$
21 cGTOs	$-14.5730\,\mathrm{E_H}$	$-14.6217\,\mathrm{E_H}$

Coulomb Sturmians with $n_{\max} = 6, l_{\max} = m_{\max} = 1$, $k = 2.1$,
cGTOs: cc-pV5Z, but only $l = 0$ and $l = 1$

[1] N. H. Morgon et. al., *Comp. Theor. Chem.*, **1997**, *394*, 95-100

Preliminary results

- Helium HF energies:

22 Sturmians	$-2.861\,679\,995\,612\,\mathrm{E_H}$
CBS value[1]	$-2.861\,679\,995\,615\,\mathrm{E_H}$

Coulomb Sturmians with $n_{\max} = 22$, $l_{\max} = m_{\max} = 0$, $k = 2$

- Beryllium comparison:

	E_{HF}	E_{MP2}
21 Sturmians	$-14.5634\,\mathrm{E_H}$	$-14.6248\,\mathrm{E_H}$
21 cGTOs	$-14.5730\,\mathrm{E_H}$	$-14.6217\,\mathrm{E_H}$

Coulomb Sturmians with $n_{\max} = 6, l_{\max} = m_{\max} = 1$, $k = 2.1$,
cGTOs: cc-pV5Z, but only $l = 0$ and $l = 1$

[1] N. H. Morgon et. al., *Comp. Theor. Chem.*, **1997**, *394*, 95-100

Apply-based scheme

- Iterative solvers only need matrix-vector products
- Matrix-vector product of **J** and **K**:
  Theoretically $\mathcal{O}(N)$ by means of contraction scheme

$$\text{e.g.} \qquad \left(K\tilde{c}^{(f)}\right)_b = \sum_{acdo\mu\nu} c_a^{(o)} \; \mathcal{C}_{ab}^{\mu} I_{\mu\nu} \mathcal{C}_{cd}^{\nu} \; c_c^{(o)} \tilde{c}_d^{(f)}$$

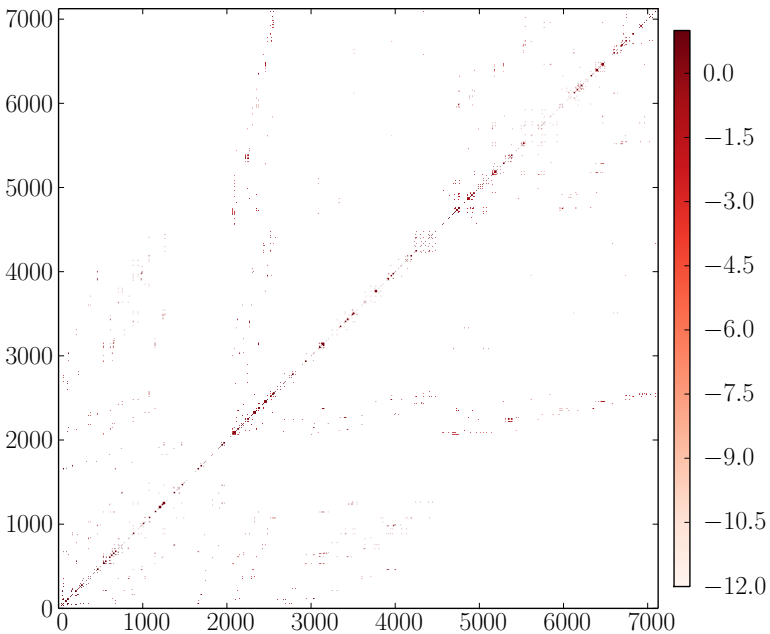
$\Rightarrow$ Apply-based or matrix-free algorithm:

- Never build **K** in storage
- Use expression for **K** to directly `apply` matrix to vectors
- SCF based on orbital coefficients $\underline{\boldsymbol{c}}^{(f)}$

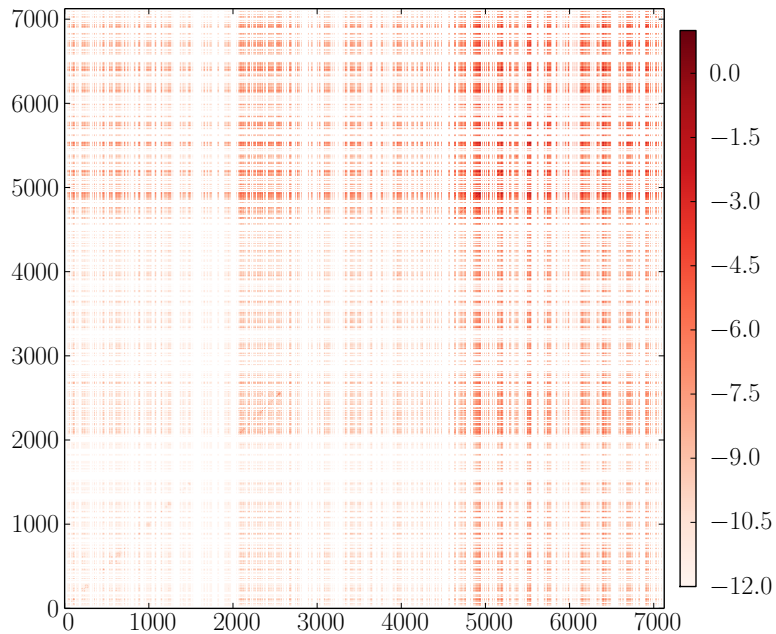
Finite elements

- Very local
- Large number of basis functions ($> 10^6$)
- Building $\mathbf{K}$: $\Omega(N^2)$ time and $\mathcal{O}(N^2)$ storage
- But: Theoretical $\mathcal{O}(N)$ scaling in apply-based scheme
- Typical for "numerical" basis functions
- Compare all basis functions in the same program?

Finite elements



$\mathbf{T} + \mathbf{V}_{\mathrm{Nuc}} + \mathbf{J}$



$\mathbf{K}$

Finite elements

- Very local
- Large number of basis functions ($> 10^6$)
- Building **K**: $\Omega(N^2)$ time and $\mathcal{O}(N^2)$ storage
- But: Theoretical $\mathcal{O}(N)$ scaling in apply-based scheme
- Typical for "numerical" basis functions
- Compare all basis functions in the same program?

Contents

Characteristics of apply-based algorithms

Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `apply`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `apply`
    - Algorithms more complicated

Characteristics of apply-based algorithms

Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `apply`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `apply`
    - Algorithms more complicated

Characteristics of apply-based algorithms

Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `apply`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `apply`
    - Algorithms more complicated

Characteristics of apply-based algorithms

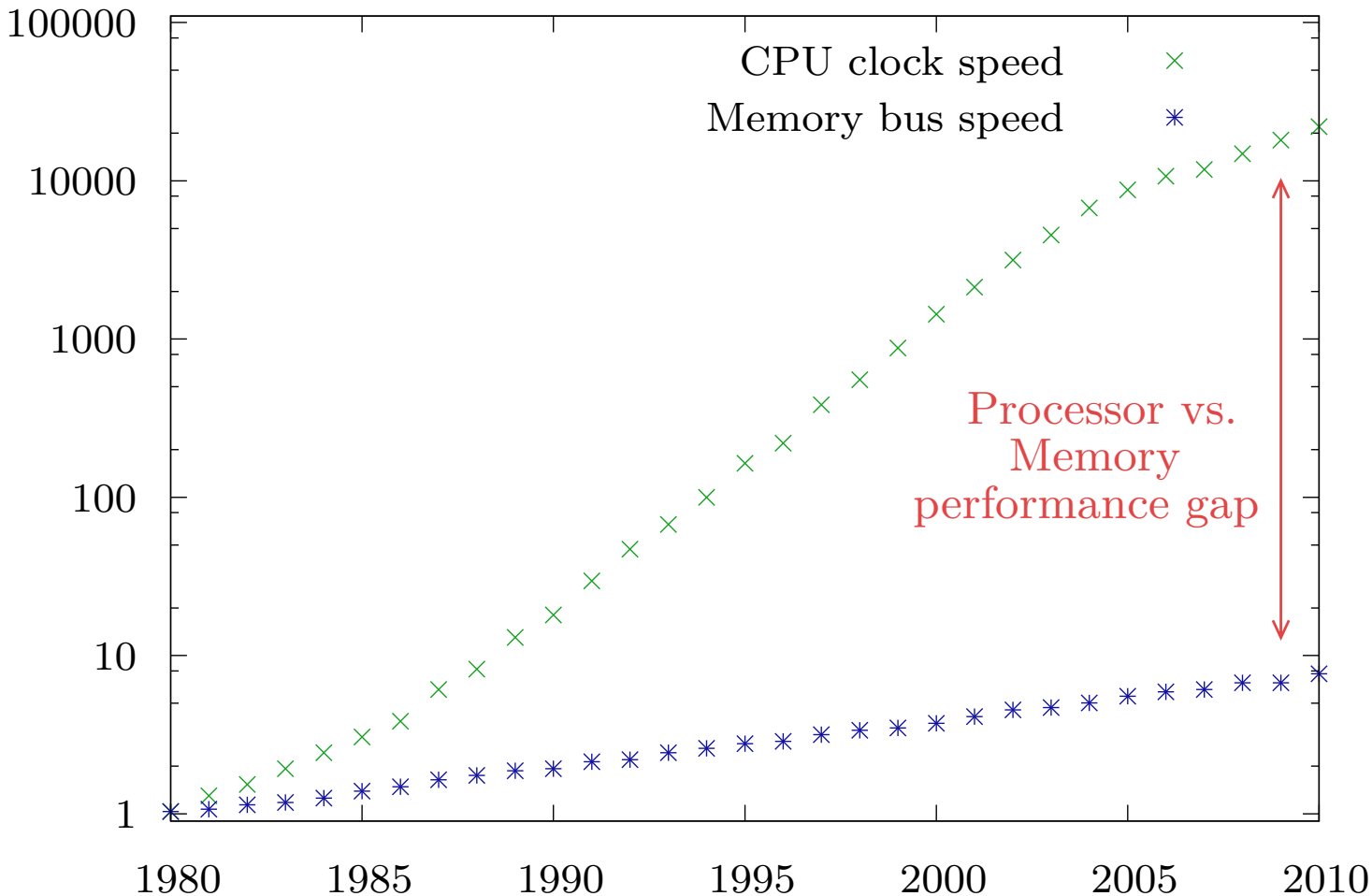
Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `apply`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `apply`
    - Algorithms more complicated

Processor vs. memory performance improvement



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

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Generalisation of matrices
    - State
    - Non-linear
    - Elements may be expressions

  $\Rightarrow$ Obtaining elements expensive
  - Evaluation of internal expression: Delayed until `apply`
  - 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:

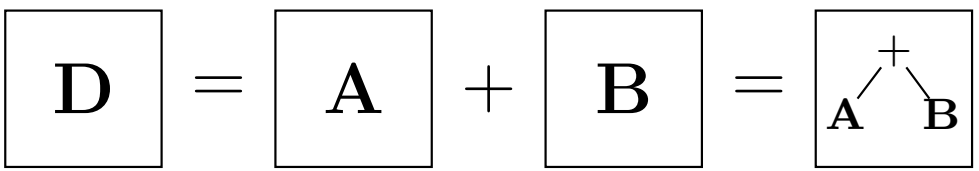
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

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

- Build expression tree internally



- On application:

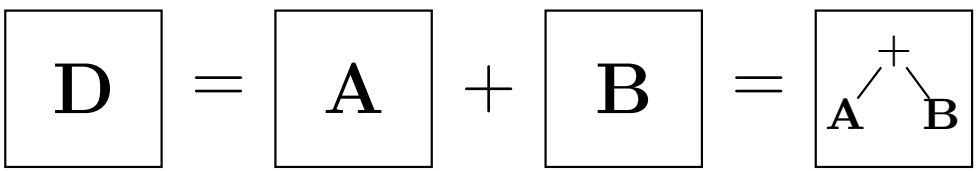
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

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

- Build expression tree internally



- On application:

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

Interface and example code

- linalgwrap[1]: Prototype C++ implementation

```
1  typedef SmallVector<double> vector_type;
2  typedef SmallMatrix<double> matrix_type;
3  auto v = random<vector_type>(100);
4  DiagonalMatrix<matrix_type> diag(v);
5  auto mat = random<matrix_type>(100,100);
6
7  // No computation: Just build expression tree
8  auto sum = diag + mat;
9  auto prod = trans(sum) * diag * sum;
10 auto tree = mat + prod;
11
12 // Evaluate tree on application:
13 SmallVector<double> res = tree * v;
```

[1] https://linalgwrap.org

Notes and observations

- `linalgwrap`: Bookkeeping for `apply`-functions
- Programmer still sees matrices

⇒ Language for writing `apply`-based algorithms

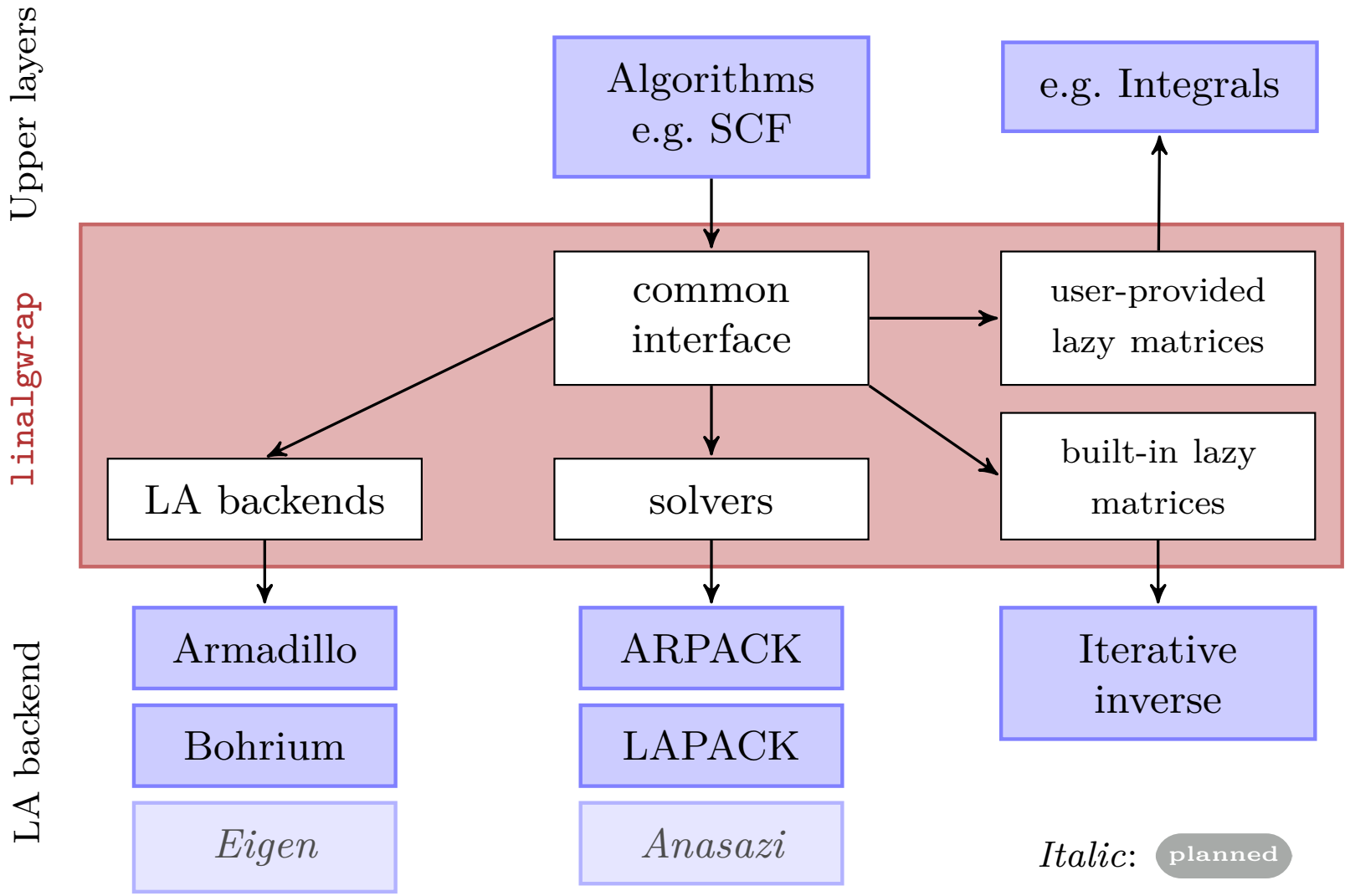
- Lazy matrices allow layered responsibility for computation, e.g. $(\mathbf{A} + \mathbf{B})\underline{\boldsymbol{x}}$
  - $\mathbf{A}\underline{\boldsymbol{x}}$ and $\mathbf{B}\underline{\boldsymbol{x}}$ decided by implementation of $\mathbf{A}$ and $\mathbf{B}$
  - $(\mathbf{A}\underline{\boldsymbol{x}})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

⇒ Proper modularisation between

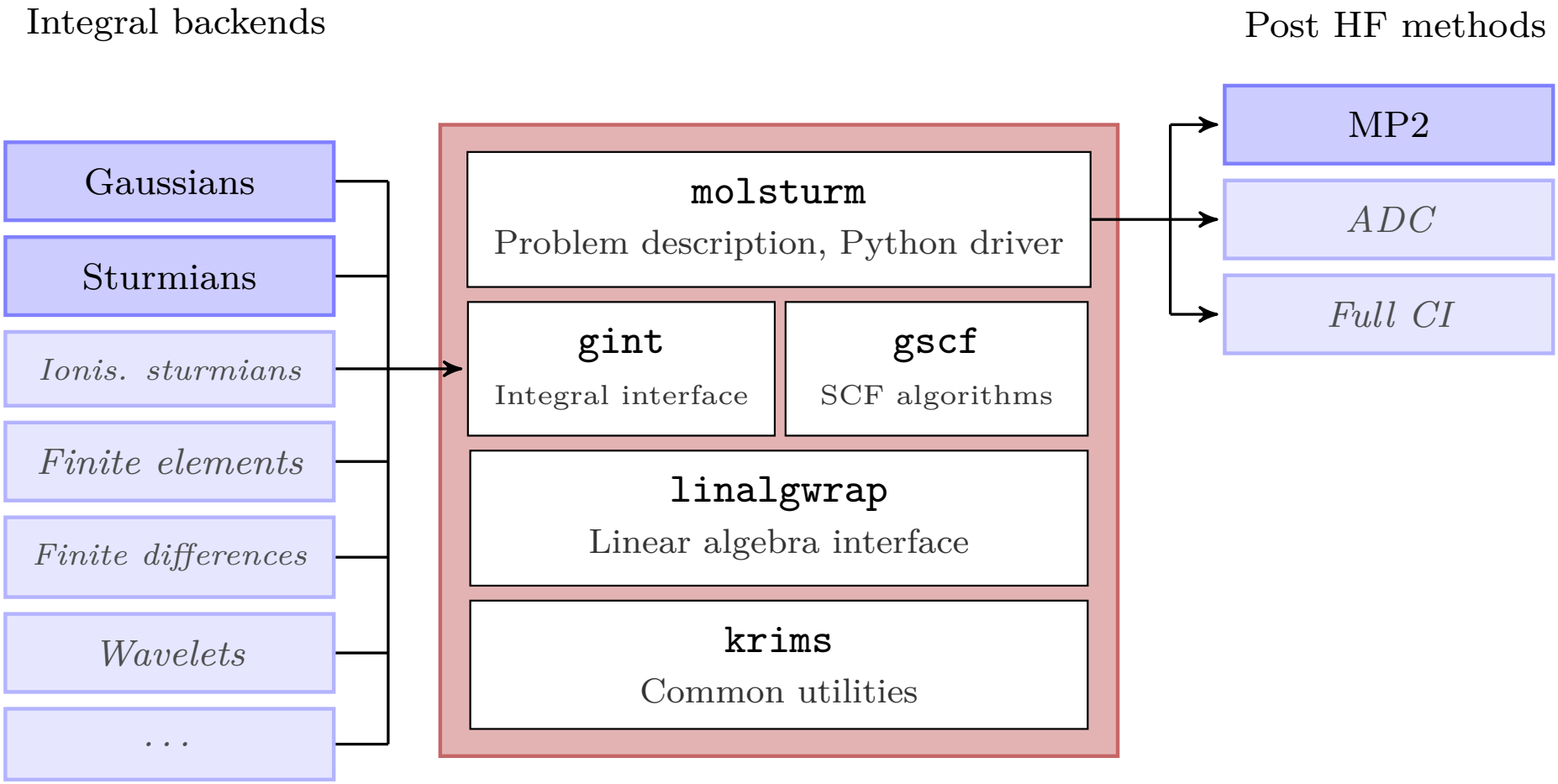
- Higher-level algorithms
- Lazy matrix implementations
- LA backends

linalgwrap

Lazy linear algebra wrapper library



molsturm structure



- *Italic*: planned

molsturm design

- Enables apply-based SCF routines
- Flexiblity first, speed second
- Behaviour controlled via `python`
  - Keywords to change basis type or solver
  - All computed data available for analysis
  - No input file, just a `python` script
- `python` utilities
  - Import / export results
  - Post-HF calculations

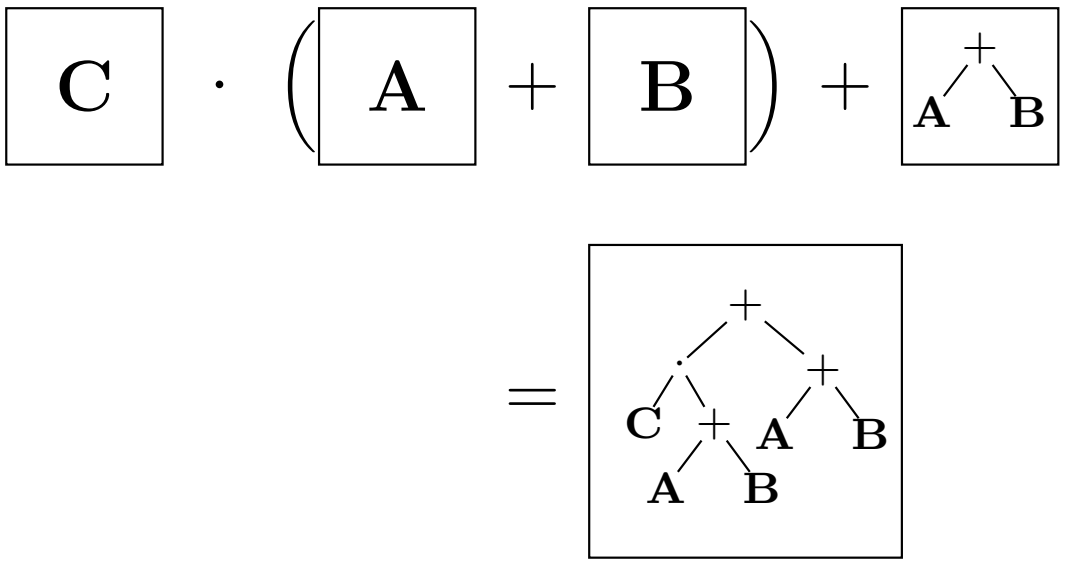
Demo

DEMO

of using molsturm

Contents

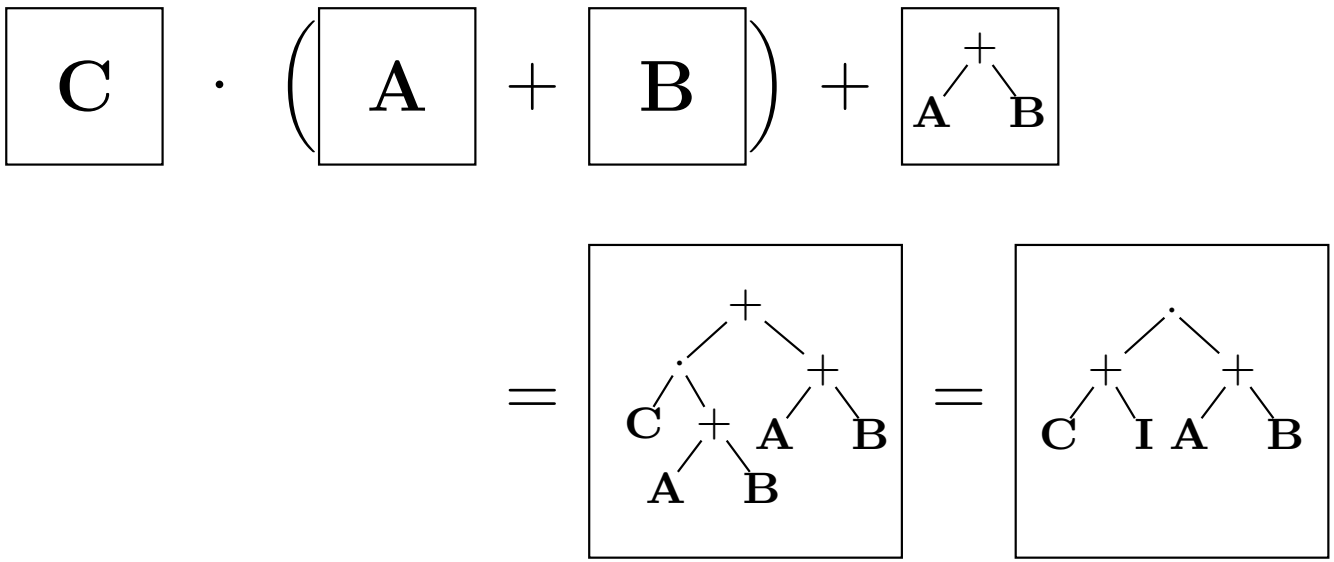
Lazy matrix expression optimisation



- Matrix expression tree ≡ abstract syntax tree

⇒ May be optimised by standard methods

Lazy matrix expression optimisation



- Matrix expression tree ≡ abstract syntax tree

⇒ May be optimised by standard methods

Selection of LA backend

- Right now: LA backend is compiled in
- e.g. Bohrium backend
  - Uses just-in-time (JIT) compilation
  - Very specific for hardware
  - Compilation takes time
- Better: Dynamic selection
  - Load on the expression tree
  - Availablility of backends
  - Hardware specs

Extension to lazy tensors

- `apply` is a special tensor contraction

⇒ Lazy tensors:

- Delay all tensor contractions as long as possible

$$\text{e.g.} \qquad \tilde{k}_{bf} = \sum_{acdo\mu\nu} \mathcal{C}_{ao} \; c^{\mu}_{ab} I_{\mu\nu} c^{\nu}_{cd} \; \mathcal{C}_{co} \mathcal{C}_{df}$$

- Compare possible contraction schemes by complexity
- Execute cheapest evaluation scheme

⇒ Determine optimal contraction sequence automatically

Acknowledgements

- Dr. James Avery
- Prof. Andreas Dreuw and the Dreuw group

- Prof. Guido Kanschat
- HGS Mathcomp

Questions?

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